CS 373U: Combinatorial Algorithms, Spring 2004
Homework 0

Due January 28, 2004 at noon

Name:	
Net ID:	Alias:

☐ I understand the Homework Instructions and FAQ.

- Neatly print your full name, your NetID, and an alias of your choice in the boxes above. Grades will be listed on the course web site by alias; for privacy reasons, your alias should not resemble your name or NetID. By providing an alias, you agree to let us list your grades; if you do not provide an alias, your grades will not be listed. ***Never*** *give us your Social Security number!*

- Before you do anything else, read the Homework Instructions and FAQ on the course web page, and then check the box above. This web page gives instructions on how to write and submit homeworks—staple your solutions together in order, start each numbered problem on a new sheet of paper, write your name and NetID one every page, don't turn in source code, analyze and prove everything, use good English and good logic, and so on. See especially the policies regarding the magic phrases "I don't know" and "and so on". If you have *any* questions, post them to the course newsgroup or ask in lecture.

- This homework tests your familiarity with prerequisite material—basic data structures, big-Oh notation, recurrences, discrete probability, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Chapters 1–10 of CLRS should be sufficient review, but you may also want consult your discrete mathematics and data structures textbooks.

- Every homework will have five required problems and one extra-credit problem. Each numbered problem is worth 10 points.

#	1	2	3	4	5	6*	Total
Score							
Grader							

1. Sort the functions in each box from asymptotically smallest to asymptotically largest, indicating ties if there are any. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway, just for practice. Don't merge the lists together.

   To simplify your answers, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$, and write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions $n^2, n, \binom{n}{2}, n^3$ could be sorted either as $n \ll n^2 \equiv \binom{n}{2} \ll n^3$ or as $n \ll \binom{n}{2} \equiv n^2 \ll n^3$.

   (a)

$2^{\sqrt{\lg n}}$	$2^{\lg \sqrt{n}}$	$\sqrt{2^{\lg n}}$	$\sqrt{2}^{\lg n}$	$\lg 2^{\sqrt{n}}$	$\lg \sqrt{2}^{n}$	$\lg \sqrt{2^n}$	$\sqrt{\lg 2^n}$
$\lg n^{\sqrt{2}}$	$\lg \sqrt{n}^2$	$\lg \sqrt{n^2}$	$\sqrt{\lg n^2}$	$\lg^2 \sqrt{n}$	$\lg^{\sqrt{2}} n$	$\sqrt{\lg^2 n}$	$\sqrt{\lg n}^2$

   ⋆(b)

$\lg(\sqrt{n}!)$	$\lg(\sqrt{n!})$	$\sqrt{\lg(n!)}$	$(\lg \sqrt{n})!$	$(\sqrt{\lg n})!$	$\sqrt{(\lg n)!}$

   [Hint: Use Stirling's approximation for factorials: $n! \approx n^{n+1/2}/e^n$]

2. Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Proofs are *not* required; just give us the list of answers. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway, just for practice. Assume reasonable but nontrivial base cases. If your solution requires specific base cases, state them! Extra credit will be awarded for more exact solutions.

   (a) $A(n) = 9A(n/3) + n^2$

   (b) $B(n) = 2B(n/2) + n/\lg n$

   (c) $C(n) = \dfrac{2C(n-1)}{C(n-2)}$ *[Hint: This is easy!]*

   (d) $D(n) = D(n-1) + 1/n$

   (e) $E(n) = E(n/2) + D(n)$

   (f) $F(n) = 2F\big(\lfloor (n+3)/4 \rfloor - \sqrt{5n \lg n} + 6\big) + 7\sqrt{n+8} - \lg^9 \lg \lg n + 10^{\lg^* n} - 11/n^{12}$

   (g) $G(n) = 3G(n-1) - 3G(n-2) + G(n-3)$

   ⋆(h) $H(n) = 4H(n/2) - 4H(n/4) + 1$ *[Hint: Careful!]*

   (i) $I(n) = I(n/3) + I(n/4) + I(n/6) + I(n/8) + I(n/12) + I(n/24) + n$

   ★(j) $J(n) = \sqrt{n} \cdot J(2\sqrt{n}) + n$
   [Hint: First solve the secondary recurrence $j(n) = 1 + j(2\sqrt{n})$.]

3. Scientists have recently discovered a planet, tentatively named "Ygdrasil", which is inhabited by a bizarre species called "nertices" (singular "nertex"). All nertices trace their ancestry back to a particular nertex named Rudy. Rudy is still quite alive, as is every one of his many descendants. Nertices reproduce asexually, like bees; each nertex has exactly one parent (except Rudy). There are three different types of nertices—red, green, and blue. The color of each nertex is correlated exactly with the number and color of its children, as follows:

   - Each red nertex has two children, exactly one of which is green.
   - Each green nertex has exactly one child, which is not green.
   - Blue nertices have no children.

   In each of the following problems, let R, G, and B respectively denote the number of red, green, and blue nertices on Ygdrasil.

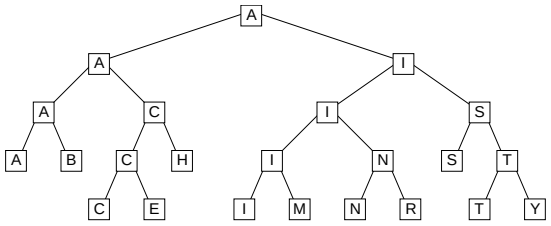
   (a) Prove that $B = R + 1$.

   (b) Prove that either $G = R$ or $G = B$.

   (c) Prove that $G = B$ if and only if Rudy is green.

4. Algorithms and data structures were developed millions of years ago by the Martians, but not quite in the same way as the recent development here on Earth. Intelligent life evolved independently on Mars' two moons, Phobos and Deimos.[1] When the two races finally met on the surface of Mars, after thousands of years of separate philosophical, cultural, religious, and scientific development, their disagreements over the proper structure of binary search trees led to a bloody (or more accurately, ichorous) war, ultimately leading to the destruction of all Martian life.

   A *Phobian* binary search tree is a full binary tree that stores a set X of search keys. The root of the tree stores the *smallest* element in X. If X has more than one element, then the left subtree stores all the elements less than some pivot value p, and the right subtree stores everything else. Both subtrees are *nonempty* Phobian binary search trees. The actual pivot value p is *never* stored in the tree.



A Phobian binary search tree for the set {M, A, R, T, I, N, B, Y, S, E, C, H}.

   (a) Describe and analyze an algorithm FIND(x, T) that returns TRUE if x is stored in the Phobian binary search tree T, and FALSE otherwise.

   (b) A *Deimoid* binary search tree is almost exactly the same as its Phobian counterpart, except that the *largest* element is stored at the root, and both subtrees are Deimoid binary search trees. Describe and analyze an algorithm to transform an n-node Phobian binary search tree into a Deimoid binary search tree in $O(n)$ time, using as little additional space as possible.

[1] Greek for "fear" and "panic", respectively. Doesn't that make you feel better?

5. Penn and Teller agree to play the following game. Penn shuffles a standard deck[2] of playing cards so that every permutation is equally likely. Then Teller draws cards from the deck, one at a time without replacement, until he draws the three of clubs (3♣), at which point the remaining undrawn cards instantly burst into flames.

   The first time Teller draws a card from the deck, he gives it to Penn. From then on, until the game ends, whenever Teller draws a card whose value is smaller than the last card he gave to Penn, he gives the new card to Penn.[3] To make the rules unambiguous, they agree beforehand that $A = 1$, $J = 11$, $Q = 12$, and $K = 13$.

   (a) What is the expected number of cards that Teller draws?

   (b) What is the expected *maximum* value among the cards Teller gives to Penn?

   (c) What is the expected *minimum* value among the cards Teller gives to Penn?

   (d) What is the expected number of cards that Teller gives to Penn?

   Full credit will be given only for *exact* answers (with correct proofs, of course).

*6. **[Extra credit]**[4]

   Lazy binary is a variant of standard binary notation for representing natural numbers where we allow each "bit" to take on one of three values: 0, 1, or 2. Lazy binary notation is defined inductively as follows.

   - The lazy binary representation of zero is 0.
   - Given the lazy binary representation of any non-negative integer n, we can construct the lazy binary representation of $n + 1$ as follows:
     (a) increment the rightmost digit;
     (b) if any digit is equal to 2, replace the rightmost 2 with 0 and increment the digit immediately to its left.

   Here are the first several natural numbers in lazy binary notation:

   0, 1, 10, 11, 20, 101, 110, 111, 120, 201, 210, 1011, 1020, 1101, 1110, 1111, 1120, 1201, 1210, 2011, 2020, 2101, 2110, 10111, 10120, 10201, 10210, 11011, 11020, 11101, 11110, 11111, 11120, 11201, 11210, 12011, 12020, 12101, 12110, 20111, 20120, 20201, 20210, 21011, 21020, 21101, 21110, 101111, 101120, 101201, 101210, 102011, 102020, 102101, 102110, ...

   (a) Prove that in any lazy binary number, between any two 2s there is at least one 0, and between two 0s there is at least one 2.

   (b) Prove that for any natural number N, the sum of the digits of the lazy binary representation of N is exactly $\lfloor \lg(N + 1) \rfloor$.

[2] In a standard deck of 52 cards, each card has a *suit* in the set $\{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\}$ and a *value* in the set $\{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\}$, and every possible suit-value pair appears in the deck exactly once. Actually, to make the game more interesting, Penn and Teller normally use razor-sharp ninja throwing cards.

[3] Specifically, he hurls them from the opposite side of the stage directly into the back of Penn's right hand.

[4] The "I don't know" rule does not apply to extra credit problems. There is no such thing as "partial extra credit".

CS 373U: Combinatorial Algorithms, Spring 2004
Homework 1

Due Monday, February 9, 2004 at noon

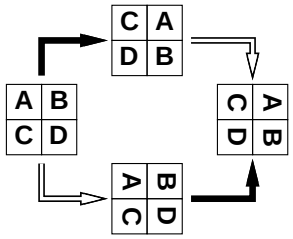
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- For this and all following homeworks, groups of up to three people can turn in a single solution. Please write *all* your names and NetIDs on *every* page you turn in.

#	1	2	3	4	5	6*	Total
Score							
Grader							

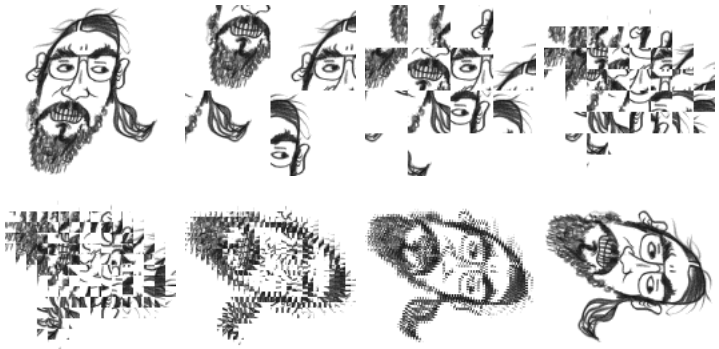
1. Some graphics hardware includes support for an operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

   Suppose we want to rotate an $n \times n$ pixelmap 90° clockwise. One way to do this is to split the pixelmap into four $n/2 \times n/2$ blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we can first recursively rotate the blocks and blit them into place afterwards.



Two algorithms for rotating a pixelmap.
Black arrows indicate blitting the blocks into place.
White arrows indicate recursively rotating the blocks.

   The following sequence of pictures shows the first algorithm (blit then recurse) in action.



   In the following questions, assume n is a power of two.

   (a) Prove that both versions of the algorithm are correct. *[Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]*

   (b) *Exactly* how many blits does the algorithm perform?

   (c) What is the algorithm's running time if each $k \times k$ blit takes $O(k^2)$ time?

   (d) What if each $k \times k$ blit takes only $O(k)$ time?

2. The traditional Devonian/Cornish drinking song "The Barley Mow" has the following pseudolyrics[1], where *container*$[i]$ is the name of a container that holds 2^i ounces of beer.[2]

```
BarleyMow(n):
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  "We'll drink it out of the jolly brown bowl,"
  "Here's a health to the barley-mow!"
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  for i ← 1 to n
      "We'll drink it out of the container[i], boys,"
      "Here's a health to the barley-mow!"
      for j ← i downto 1
          "The container[j],"
      "And the jolly brown bowl!"
      "Here's a health to the barley-mow!"
      "Here's a health to the barley-mow, my brave boys,"
      "Here's a health to the barley-mow!"
```

(a) Suppose each container name *container*$[i]$ is a single word, and you can sing four words a second. How long would it take you to sing BarleyMow(n)? (Give a tight asymptotic bound.)

(b) If you want to sing this song for $n > 20$, you'll have to make up your own container names, and to avoid repetition, these names will get progressively longer as n increases[3]. Suppose *container*$[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing BarleyMow(n)? (Give a tight asymptotic bound.)

(c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and 2^i ounces for each *container*$[i]$. Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang BarleyMow(n)? (Give an *exact* answer, not just an asymptotic bound.)

[1] Pseudolyrics are to lyrics as pseudocode is to code.

[2] One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

[3] "We'll drink it out of the hemisemidemiyottapint, boys!"

3. In each of the problems below, you are given a 'magic box' that can solve one problem quickly, and you are asked to construct an algorithm that uses the magic box to solve a different problem.

   (a) **3-Coloring**: A graph is *3-colorable* if it is possible to color each vertex red, green, or blue, so that for every edge, its two vertices have two different colors. Suppose you have a magic box that can tell you whether a given graph is 3-colorable in constant time. Describe an algorithm that constructs a 3-coloring of a given graph (if one exists) as quickly as possible.

   (b) **3SUM:** The 3SUM problem asks, given a set of integers, whether any three elements sum to zero. Suppose you have a magic box that can solve the 3SUM problem in constant time. Describe an algorithm that actually finds, given a set of integers, three elements that sum to zero (if they exist) as quickly as possible.

   (c) **Traveling Salesman:** A *Hamiltonian cycle* in a graph is a cycle that visits every vertex exactly once. Given a complete graph where every edge has a weight, the *traveling salesman* cycle is the Hamiltonian cycle with minimum total weight; that is, the sum of the weight of the edges is smaller than for any other Hamiltonian cycle. Suppose you have a magic box that can tell you the weight of the traveling salesman cycle of a weighted graph in constant time. Describe an algorithm that actually constructs the traveling salesman cycle of a given weighted graph as quickly as possible.

4. (a) Describe and analyze an algorithm to sort an array $A[1..n]$ by calling a subroutine SqrtSort(k), which sorts the subarray $A[k+1..k+\lceil\sqrt{n}\rceil]$ in place, given an arbitrary integer k between 0 and $n - \lceil\sqrt{n}\rceil$ as input. Your algorithm is *only* allowed to inspect or modify the input array by calling SqrtSort; in particular, your algorithm must not directly compare, move, or copy array elements. How many times does your algorithm call SqrtSort in the worst case?

   (b) Prove that your algorithm from part (a) is optimal up to constant factors. In other words, if $f(n)$ is the number of times your algorithm calls SqrtSort, prove that no algorithm can sort using $o(f(n))$ calls to SqrtSort.

   (c) Now suppose SqrtSort is implemented recursively, by calling your sorting algorithm from part (a). For example, at the second level of recursion, the algorithm is sorting arrays roughly of size $n^{1/4}$. What is the worst-case running time of the resulting sorting algorithm? (To simplify the analysis, assume that the array size n has the form 2^{2^k}, so that repeated square roots are always integers.)

5. In a previous incarnation, you worked as a cashier in the lost Antarctican colony of Nadira, spending the better part of your day giving change to your customers. Because paper is a very rare and valuable resource on Antarctica, cashiers were required by law to use the fewest bills possible whenever they gave change. Thanks to the numerological predilections of one of its founders, the currency of Nadira, called Dream Dollars, was available in the following denominations: \$1, \$4, \$7, \$13, \$28, \$52, \$91, \$365.[4]

   (a) The greedy change algorithm repeatedly takes the largest bill that does not exceed the target amount. For example, to make \$122 using the greedy algorithm, we first take a \$91 bill, then a \$28 bill, and finally three \$1 bills. Give an example where this greedy algorithm uses more Dream Dollar bills than the minimum possible.

   (b) Describe and analyze a recursive algorithm that computes, given an integer k, the minimum number of bills needed to make k Dream Dollars. (Don't worry about making your algorithm fast; just make sure it's correct.)

   (c) Describe a dynamic programming algorithm that computes, given an integer k, the minimum number of bills needed to make k Dream Dollars. (This one needs to be fast.)

⋆6. **[Extra Credit]** A popular puzzle called "Lights Out!", made by Tiger Electronics, has the following description. The game consists of a 5×5 array of lighted buttons. By pushing any button, you toggle (on to off, off to on) that light and its four (or fewer) immediate neighbors. The goal of the game is to have every light off at the same time.

   We generalize this puzzle to a graph problem. We are given an arbitrary graph with a lighted button at every vertex. Pushing the button at a vertex toggles its light and the lights at all of its neighbors in the graph. A *light configuration* is just a description of which lights are on and which are off. We say that a light configuration is *solvable* if it is possible to get from that configuration to the everything-off configuration by pushing buttons. Some (but clearly not all) light configurations are unsolvable.

   (a) Suppose the graph is just a cycle of length n. Give a simple and complete characterization of the solvable light configurations in this case. (What we're really looking for here is a *fast* algorithm to decide whether a given configuration is solvable or not.) *[Hint: For which cycle lengths is* ***every*** *configuration solvable?]*

   ⋆(b) Characterize the set of solvable light configurations when the graph is an arbitrary tree.

   ★(c) A *grid graph* is a graph whose vertices are a regular $h \times w$ grid of integer points, with edges between immediate vertical or horizontal neighbors. Characterize the set of solvable light configurations for an arbitrary grid graph. (For example, the original Lights Out puzzle can be modeled as a 5×5 grid graph.)

[4] For more details on the history and culture of Nadira, including images of the various denominations of Dream Dollars, see http://www.dream-dollars.com.

CS 373U: Combinatorial Algorithms, Spring 2004
Homework 2

Due Friday, February 20, 2004 at noon
(so you have the whole weekend to study for the midterm)

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- Starting with this homework, we are changing the way we want you to submit solutions. For each numbered problem, if you use more than one page, staple all those pages together. **Please do *not* staple your entire homework together.** This will allow us to moreeasily distribute the problems to the graders. Remember to print the name and NetID of every member of your group, as well as the assignment and problem numbers, on every page you submit. You do not need to turn in this cover page.
- Unless specifically stated otherwise, you can use the fact that the following problems are NP-hard to prove that other problems are NP-hard: Circuit-SAT, 3SAT, Vertex Cover, Maximum Clique, Maximum Independent Set, Hamiltonian Path, Hamiltonian Cycle, k-Colorability for any $k \geq 3$, Traveling Salesman Path, Travelling Salesman Cycle, Subset Sum, Partition, 3Partition, Hitting Set, Minimum Steiner Tree, Minesweeper, Tetris, or any other NP-hard problem described in the lecture notes.
- This homework is a little harder than the last one. You might want to start early.

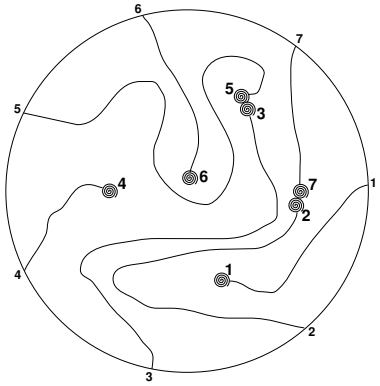
#	1	2	3	4	5	6*	Total
Score							
Grader							

1. In lecture on February 5, Jeff presented the following algorithm to compute the length of the longest increasing subsequence of an n-element array $A[1..n]$ in $O(n^2)$ time.

```
LengthOfLIS(A[1..n]):
  A[n+1] = ∞
  for i ← 1 to n+1
      L[i] ← 1
      for j ← 1 to i−1
          if A[j] < A[i] and 1 + L[j] < L[i]
              L[i] ← 1 + L[j]
  return L[n+1] − 1
```

   Describe another algorithm for this problem that runs in $O(n \log n)$ time. *[Hint: Use a data structure to replace the inner loop with something faster.]*

2. Every year, as part of its annual meeting, the Antarctican Snail Lovers of Union Glacier hold a Round Table Mating Race. A large number of high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to n. The snails wander around the table, each snail leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail (even their own). When any two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if n is even and the race goes on forever.



The end of an Antarctican SLUG race. Snails 1, 4, and 6 never find a mate.
The organizers must pay $M[3,5] + M[2,7]$.

   For every pair of snails, the Antarctican SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array $M[1..n, 1..n]$ posted on the wall behind the Round Table, where $M[i,j] = M[j,i]$ is the reward if snails i and j meet.

   Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the $n \times n$ array M as input.

3. Describe and analyze a *polynomial-time* algorithm to determine whether a boolean formula in conjunctive normal form, with exactly *two* literals in each clause, is satisfiable.

4. This problem asks you to prove that four different variants of the minimum spanning tree problem are NP-hard. In each case, the input is a connected undirected graph G with weighted edges. Each problem considers a certain subset of the possible spanning trees of G, and asks you to compute the spanning tree with minimum total weight in that subset.

   (a) Prove that finding the minimum-weight *depth first search* tree is NP-hard. (To remind yourself what depth first search is, and why it computes a spanning tree, see Jeff's introductory notes on graphs or Chapter 22 in CLRS.)

   (b) Suppose a subset S of the nodes in the input graph are marked. Prove that it is NP-hard to compute the minimum spanning tree whose leaves are all in S. *[Hint: First consider the case $|S| = 2$.]*

   (c) Prove that for any integer $\ell \geq 2$, it is NP-hard to compute the minimum spanning tree with exactly ℓ leaves. *[Hint: First consider the case $\ell = 2$.]*

   (d) Prove that for any integer $d \geq 2$, it is NP-hard to compute the minimum spanning tree with maximum degree d. *[Hint: First consider the case $d = 2$. By now this should start to look familiar.]*

   You're welcome to use reductions among these four problems. For example, even if you can't solve part (d), if you can prove that (d) implies (b), you will get full credit for (b). Just don't argue circularly.

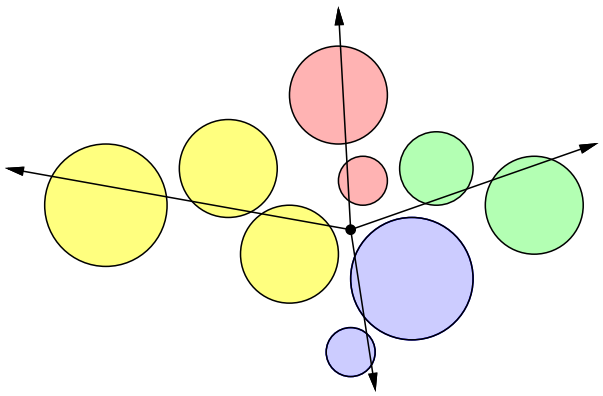
5. Consider a machine with a row of n processors numbered 1 through n. A *job* is some computational task that occupies a contiguous set of processors for some amount of time. Each processor can work on only one job at a time. Each job is represented by a pair $J_i = (n_i, t_i)$, where n_i is the number of processors required and t_i is the amount of processing time required to perform the job. A *schedule* for a set of jobs $\{J_1, \ldots, J_m\}$ assigns each job J_i to some set of n_i contiguous processors for an interval of t_i seconds, so that no processor works on more than one job at any time. The *make-span* of a schedule is the time from the start to the finish of all jobs.

   The *parallel scheduling problem* asks, given a set of jobs as input, to compute a schedule for those jobs with the smallest possible make-span.

   (a) Prove that the parallel scheduling problem is NP-hard.

   (b) Give an algorithm that computes a 3-approximation of the minimum make-span of a set of jobs in $O(m \log m)$ time. That is, if the minimum make-span is M, your algorithm should compute a schedule with make-span at most $3M$. You can assume that n is a power of 2.

*6. **[Extra credit]** Suppose you are standing in a field surrounded by several large balloons. You want to use your brand new Acme Brand Zap-O-Matic™ to pop all the balloons, without moving from your current location. The Zap-O-Matic™ shoots a high-powered laser beam, which pops all the balloons it hits. Since each shot requires enough energy to power a small country for a year, you want to fire as few shots as possible.



Nine balloons popped by 4 shots of the Zap-O-Matic™

The *minimum zap* problem can be stated more formally as follows. Given a set C of n circles in the plane, each specified by its radius and the (x, y) coordinates of its center, compute the minimum number of rays from the origin that intersect every circle in C. Your goal is to find an efficient algorithm for this problem.

(a) Describe and analyze a greedy algorithm whose output is within 1 of optimal. That is, if m is the minimum number of rays required to hit every circle in the input, then your greedy algorithm must output either m or $m + 1$. (Of course, you must prove this fact.)

(b) Describe an algorithm that solves the minimum zap problem in $O(n^2)$ time.

*(c) Describe an algorithm that solves the minimum zap problem in $O(n \log n)$ time.

Assume you have a subroutine INTERSECTS(r, c) that determines, in $O(1)$ time, whether a ray r intersects a circle c. It's not that hard to write this subroutine, but it's not the interesting part of the problem.

CS 373U: Combinatorial Algorithms, Spring 2004
Homework 3

Due Friday, March 12, 2004 at noon

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- For each numbered problem, if you use more than one page, staple all those pages together. **Please do *not* staple your entire homework together.** This will allow us to more easily distribute the problems to the graders. Remember to print the name and NetID of every member of your group, as well as the assignment and problem numbers, on every page you submit. You do not need to turn in this cover page.
- This homework is challenging. You might want to start early.

#	1	2	3	4	5	6*	Total
Score							
Grader							

1. Let S be a set of n points in the plane. A point p in S is called *Pareto-optimal* if no other point in S is both above and to the right of p.

   (a) Describe and analyze a deterministic algorithm that computes the Pareto-optimal points in S in $O(n \log n)$ time.

   (b) Suppose each point in S is chosen independently and uniformly at random from the unit square $[0, 1] \times [0, 1]$. What is the *exact* expected number of Pareto-optimal points in S?

2. Suppose we have an oracle RANDOM(k) that returns an integer chosen independently and uniformly at random from the set $\{1, \ldots, k\}$, where k is the input parameter; RANDOM is our only source of random bits. We wish to write an efficient function RANDOMPERMUTATION(n) that returns a permutation of the integers $\langle 1, \ldots, n \rangle$ chosen uniformly at random.

   (a) Consider the following implementation of RANDOMPERMUTATION.

```
RandomPermutation(n):
  for i = 1 to n
      π[i] ← NULL
  for i = 1 to n
      j ← Random(n)
      while (π[j] != NULL)
          j ← Random(n)
      π[j] ← i
  return π
```

   Prove that this algorithm is correct. Analyze its expected runtime.

   (b) Consider the following partial implementation of RANDOMPERMUTATION.

```
RandomPermutation(n):
  for i = 1 to n
      A[i] ← Random(n)
  π ← SomeFunction(A)
  return π
```

   Prove that if the subroutine SOMEFUNCTION is deterministic, then this algorithm cannot be correct. *[Hint: There is a one-line proof.]*

   (c) Consider a correct implementation of RANDOMPERMUTATION(n) with the following property: whenever it calls RANDOM(k), the argument k is at most m. Prove that this algorithm ***always*** calls RANDOM at least $\Omega(\frac{n \log n}{\log m})$ times.

   (d) Describe and analyze an implementation of RANDOMPERMUTATION that runs in expected worst-case time $O(n)$.

3. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

   - MAKEQUEUE: Return a new priority queue containing the empty set.
   - FINDMIN(Q): Return the smallest element of Q (if any).
   - DELETEMIN(Q): Remove the smallest element in Q (if any).
   - INSERT(Q, x): Insert element x into Q, if it is not already there.
   - DECREASEKEY(Q, x, y): Replace an element $x \in Q$ with a smaller key y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q containing x.
   - DELETE(Q, x): Delete the element $x \in Q$. The input is a pointer directly to the node in Q containing x.
   - MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2; this operation destroys Q_1 and Q_2.

   A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

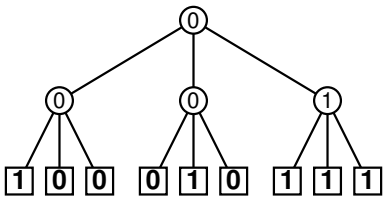
```
MELD(Q1, Q2):
  if Q1 is empty return Q2
  if Q2 is empty return Q1

  if key(Q1) > key(Q2)
      swap Q1 ↔ Q2

  with probability 1/2
      left(Q1) ← MELD(left(Q1), Q2)
  else
      right(Q1) ← MELD(right(Q1), Q2)

  return Q1
```

   (a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (not just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. *[Hint: How long is a random root-to-leaf path in an n-node binary tree if each left/right choice is made with equal probability?]*

   (b) **[Extra credit]** Prove that MELD(Q_1, Q_2) runs in $O(\log n)$ time with high probability.

   (c) Show that each of the other meldable priority queue operations cab be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)

4. A *majority tree* is a complete binary tree with depth n, where every leaf is labeled either 0 or 1. The *value* of a leaf is its label; the *value* of any internal node is the majority of the values of its three children. Consider the problem of computing the value of the root of a majority tree, given the sequence of 3^n leaf labels as input. For example, if $n = 2$ and the leaves are labeled $1, 0, 0, 0, 1, 0, 1, 1, 1,$ the root has value 0.



A majority tree with depth $n = 2$.

   (a) Prove that *any* deterministic algorithm that computes the value of the root of a majority tree *must* examine every leaf. *[Hint: Consider the special case $n = 1$. Recurse.]*

   (b) Describe and analyze a randomized algorithm that computes the value of the root in worst-case expected time $O(c^n)$ for some constant $c < 3$. *[Hint: Consider the special case $n = 1$. Recurse.]*

5. Suppose n lights labeled $0, \ldots, n-1$ are placed clockwise around a circle. Initially, each light is set to the off position. Consider the following random process.

```
LightTheCircle(n):
  k ← 0
  turn on light 0
  while at least one light is off
      with probability 1/2
          k ← (k + 1) mod n
      else
          k ← (k − 1) mod n
      if light k is off, turn it on
```

   Let $p(i, n)$ be the probability that light i is the last to be turned on by LightTheCircle$(n, 0)$. For example, $p(0, 2) = 0$ and $p(1, 2) = 1$. Find an exact closed-form expression for $p(i, n)$ in terms of n and i. Prove your answer is correct.

6. **[Extra Credit]** Let G be a *bipartite* graph on n vertices. Each vertex v has an associated set $C(v)$ of $\lg 2n$ colors with which v is compatible. We wish to find a coloring of the vertices in G so that every vertex v is assigned a color from its set $C(v)$ and no edge has the same color at both ends. Describe and analyze a randomized algorithm that computes such a coloring in expected worst-case time $O(n \log^2 n)$. *[Hint: For any events A and B, $\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$.]*

CS 373U: Combinatorial Algorithms, Spring 2004
Homework 4

Due Friday, April 2, 2004 at noon

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- For each numbered problem, if you use more than one page, staple all those pages together. **Please do *not* staple your entire homework together.** This will allow us to more easily distribute the problems to the graders. Remember to print the name and NetID of every member of your group, as well as the assignment and problem numbers, on every page you submit. You do not need to turn in this cover page.
- As with previous homeworks, we strongly encourage you to begin early.

#	1	2	3	4	5	6*	Total
Score							
Grader							

1. Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

   - After an insertion, if the table is more than 3/4 full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
   - After a deletion, if the table is less than 1/4 full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

   Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method (like CLRS does); there is a much easier solution.

2. Remember the difference between stacks and queues? Good.

   (a) Describe how to implement a queue using two stacks and $O(1)$ additional memory, so that the amortized time for any enqueue or dequeue operation is $O(1)$. The *only* access you have to the stacks is through the standard subroutines PUSH and POP.

   (b) A *quack* is a data structure combining properties of both stacks and queues. It can be viewed as a list of elements written left to right such that three operations are possible:

   - **Push:** add a new item to the left end of the list;
   - **Pop:** remove the item on the left end of the list;
   - **Pull:** remove the item on the right end of the list.

   Implement a quack using *three* stacks and $O(1)$ additional memory, so that the amortized time for any push, pop, or pull operation is $O(1)$. Again, you are *only* allowed to access the stacks through the standard functions PUSH and POP.

3. Some applications of binary search trees attach a *secondary data structure* to each node in the tree, to allow for more complicated searches. Maintaining these secondary structures usually complicates algorithms for keeping the top-level search tree balanced.

   Suppose we have a binary search tree T where every node v stores a secondary structure of size $O(|v|)$, where $|v|$ denotes the number of descendants of v in T. Performing a rotation at a node v in T now requires $O(|v|)$ time, because we have to rebuild one of the secondary structures.

   (a) **[1 pt]** Overall, how much space does this data structure use in the worst case?

   (b) **[1 pt]** How much space does this structure use if the top-level search tree T is balanced?

   (c) **[2 pt]** Suppose T is a splay tree. Prove that the *amortized* cost of a splay (and therefore of a search, insertion, or deletion) is $\Omega(n)$. *[Hint: This is easy!]*

   (d) **[3 pts]** Now suppose T is a scapegoat tree, and that rebuilding the subtree rooted at v requires $\Theta(|v| \log |v|)$ time (because we also have to rebuild all the secondary structures). What is the *amortized* cost of inserting a new element into T?

   (e) **[3 pts]** Finally, suppose T is a treap. What's the worst-case *expected* time for inserting a new element into T?

4. In a *dirty* binary search tree, each node is labeled either *clean* or *dirty*. The lazy deletion scheme used for scapegoat trees requires us to *purge* the search tree, keeping all the clean nodes and deleting all the dirty nodes, as soon as half the nodes become dirty. In addition, the purged tree should be perfectly balanced.

   Describe an algorithm to purge an arbitrary n-node dirty binary search tree in $O(n)$ time, using only $O(\log n)$ additional memory. For 5 points extra credit, reduce the additional memory requirement to $O(1)$ *without repeating an old CS373 homework solution.*[1]

5. This problem considers a variant of the lazy binary notation introduced in the extra credit problem from Homework 0. In a *doubly lazy binary number*, each bit can take one of *four* values: -1, 0, 1, or 2. The only legal representation for zero is 0. To increment, we add 1 to the least significant bit, then carry the rightmost 2 (if any). To decrement, we subtract 1 from the lest significant bit, and then borrow the rightmost -1 (if any).

```
LazyIncrement(B[0..n]):
  B[0] ← B[0] + 1
  for i ← 1 to n − 1
      if B[i] = 2
          B[i] ← 0
          B[i + 1] ← B[i + 1] + 1
          return
```

```
LazyDecrement(B[0..n]):
  B[0] ← B[0] − 1
  for i ← 1 to n − 1
      if B[i] = −1
          B[i] ← 1
          B[i + 1] ← B[i + 1] − 1
          return
```

   For example, here is a doubly lazy binary count from zero up to twenty and then back down to zero. The bits are written with the least significant bit (*i.e.*, $B[0]$) on the right. For succinctness, we write $\bar{1}$ instead of -1 and omit any leading 0's.

$$
\begin{aligned}
&0 \xrightarrow{++} 1 \xrightarrow{++} 10 \xrightarrow{++} 11 \xrightarrow{++} 20 \xrightarrow{++} 101 \xrightarrow{++} 110 \xrightarrow{++} 111 \xrightarrow{++} 120 \xrightarrow{++} 201 \xrightarrow{++} 210 \\
&\xrightarrow{++} 1011 \xrightarrow{++} 1020 \xrightarrow{++} 1101 \xrightarrow{++} 1110 \xrightarrow{++} 1111 \xrightarrow{++} 1120 \xrightarrow{++} 1201 \xrightarrow{++} 1210 \xrightarrow{++} 2011 \xrightarrow{++} 2020 \\
&\xrightarrow{--} 2011 \xrightarrow{--} 2010 \xrightarrow{--} 2001 \xrightarrow{--} 2000 \xrightarrow{--} 20\bar{1}1 \xrightarrow{--} 2\bar{1}10 \xrightarrow{--} 2\bar{1}01 \xrightarrow{--} 1100 \xrightarrow{--} 11\bar{1}1 \xrightarrow{--} 1010 \\
&\xrightarrow{--} 1001 \xrightarrow{--} 1000 \xrightarrow{--} 10\bar{1}1 \xrightarrow{--} 1\bar{1}10 \xrightarrow{--} 1\bar{1}01 \xrightarrow{--} 100 \xrightarrow{--} 1\bar{1}1 \xrightarrow{--} 10 \xrightarrow{--} 1 \xrightarrow{--} 0
\end{aligned}
$$

   Prove that for any intermixed sequence of increments and decrements of a doubly lazy binary number, starting with 0, the amortized time for each operation is $O(1)$. Do *not* assume, as in the example above, that all the increments come before all the decrements.

[1] That was for a slightly different problem anyway.

6. [**Extra credit**] My wife is teaching a class[2] where students work on homeworks in groups of exactly three people, subject to the following rule: *No two students may work together on more than one homework.* At the beginning of the semester, it was easy to find homework groups, but as the course progresses, it is becoming harder and harder to find a legal grouping. Finally, in despair, she decides to ask a computer scientist to write a program to find the groups for her.

   (a) We can formalize this homework-group-assignment problem as follows. The input is a graph, where the vertices are the n students, and two students are joined by an edge if they have not yet worked together. Every node in this graph has the same degree; specifically, if there have been k homeworks so far, each student is connected to exactly $n - 1 - 2k$ other students. The goal is to find $n/3$ disjoint triangles in the graph, or conclude that no such triangles exist. Prove (or disprove!) that this problem is NP-hard.

   (b) Suppose my wife had planned ahead and assigned groups for every homework at the beginning of the semester. How many homeworks can she assign, as a function of n, without violating the no-one-works-together-twice rule? Prove the best upper and lower bounds you can. To prove the upper bound, describe an algorithm that actually assigns the groups for each homework.

[2] Math 302: Non-Euclidean Geometry. Problem 1 from last week's homework assignment: "Invert Mr. Happy."

CS 373U: Combinatorial Algorithms, Spring 2004
Homework 5

Due Wednesday, April 28, 2004 at noon

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- For each numbered problem, if you use more than one page, staple all those pages together. **Please do *not* staple your entire homework together.** This will allow us to more easily distribute the problems to the graders. Remember to print the name and NetID of every member of your group, as well as the assignment and problem numbers, on every page you submit. You do not need to turn in this cover page.
- As with previous homeworks, we strongly encourage you to begin early.
- This will be the last graded homework.

#	1	2	3	4	5	Total
Score						
Grader						

1. (a) Prove that every graph with the same number of vertices and edges has a cycle.

   (b) Prove that every graph with exactly two fewer edges than vertices is disconnected.

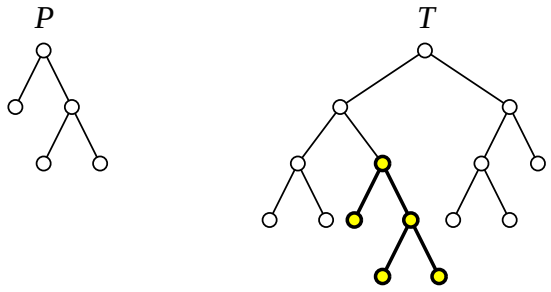
   Both proofs should be entirely self-contained. In particular, they should not use the word "tree" or any properties of trees that you saw in CS 225 or CS 273.

2. A *palindrome* is a string of characters that is exactly the same as its reversal, like `X`, `FOOF`, `RADAR`, or `AMANAPLANACATACANALPANAMA`.

   (a) Describe and analyze an algorithm to compute the longest *prefix* of a given string that is a palindrome. For example, the longest palindrome prefix of `RADARDETECTAR` is `RADAR`, and the longest palindrome prefix of `ALGORITHMSHOMEWORK` is the single letter `A`.

   (b) Describe and analyze an algorithm to compute a longest *subsequence* of a given string that is a palindrome. For example, the longest palindrome subsequnce of `RADARDETECTAR` is `RAETEAR` (or `RADADAR` or `RADRDAR` or `RATETAR` or `RATCTAR`), and the longest palindrome subsequence of `ALGORITHMSHOMEWORK` is `OMOMO` (or `RMHMR` or `OHSHO` or...).

3. Describe and analyze an algorithm that decides, given two binary trees P and T, whether T is a subtree of P. There is no actual *data* stored in the nodes—these are not binary *search* trees or binary *heaps*. You are only trying to match the *shape* of the trees.



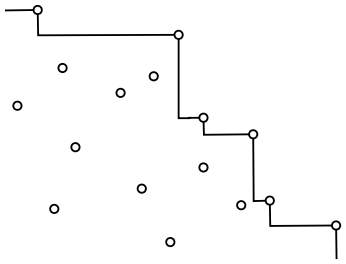
P appears exactly once as a subtree of T.

4. Describe and analyze an algorithm that computes the *second* smallest spanning tree of a given connected, undirected, edge-weighted graph.

5. Show that if the input graph is allowed to have negative edges (but no negative cycles), Dijkstra's algorithm[1] runs in exponential time in the worst case. Specifically, describe how to construct, for every integer n, a weighted directed graph G_n without negative cycles that forces Dijkstra's algorithm to perform $\Omega(2^n)$ relaxation steps. Give your description in the form of an algorithm! *[Hint: Towers of Hanoi.]*

[1]This refers to the version of Dijkstra's algorithm described in Jeff's lecture notes. The version in CLRS is always fast, but sometimes gives incorrect results for graphs with negative edges.

1. Let P be a set of n points in the plane. Recall that a point $p \in P$ is *Pareto-optimal* if no other point is both above and to the right of p. Intuitively, the sorted sequence of Pareto-optimal points describes a *staircase* with all the points in P below and to the left. Your task is to describe some algorithms that compute this staircase.



The staircase of a set of points

   (a) Describe an algorithm to compute the staircase of P in $O(nh)$ time, where h is the number of Pareto-optimal points.

   (b) Describe a divide-and-conquer algorithm to compute the staircase of P in $O(n \log n)$ time. *[Hint: I know of at least two different ways to do this.]*

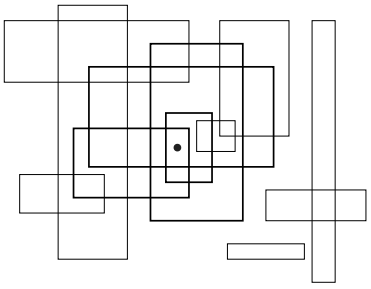
   *(c) Describe an algorithm to compute the staircase of P in $O(n \log h)$ time, where h is the number of Pareto-optimal points. *[Hint: I know of at least two different ways to do this.]*

   (d) Finally, suppose the points in P are already given in sorted order from left to right. Describe an algorithm to compute the staircase of P in $O(n)$ time. *[Hint: I know of at least two different ways to do this.]*

2. Let R be a set of n rectangles in the plane.

   (a) Describe and analyze a plane sweep algorithm to decide, in $O(n \log n)$ time, whether any two rectangles in R intersect.

   *(b) The *depth* of a point is the number of rectangles in R that contain that point. The *maximum depth* of R is the maximum, over all points p in the plane, of the depth of p. Describe a plane sweep algorithm to compute the maximum depth of R in $O(n \log n)$ time.

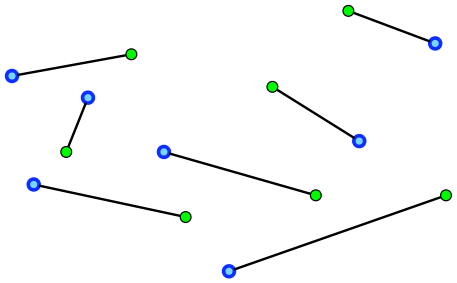


A point with depth 4 in a set of rectangles.

   (c) Describe and analyze a polynomial-time reduction from the maximum depth problem in part (b) to MaxClique: Given a graph G, how large is the largest clique in G?

   (d) MaxClique is NP-hard. So does your reduction imply that P=NP? Why or why not?

3. Let G be a set of n green points, called "Ghosts", and let B be a set of n blue points, called "ghostBusters", so that no three points lie on a common line. Each Ghostbuster has a gun that shoots a stream of particles in a straight line until it hits a ghost. The Ghostbusters want to kill all of the ghosts at once, by having each Ghostbuster shoot a different ghost. It is **very important** that the streams do not cross.



A non-crossing matching between 7 ghosts and 7 Ghostbusters

(a) Prove that the Ghostbusters can succeed. More formally, prove that there is a collection of n non-intersecting line segments, each joining one point in G to one point in B. *[Hint: Think about the set of joining segments with minimum total length.]*

(b) Describe and analyze an algorithm to find a line ℓ that passes through one ghost and one Ghostbuster, so that same number of ghosts as Ghostbusters are above ℓ.

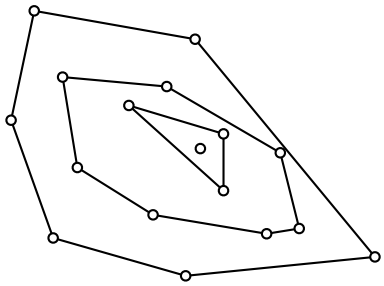
*(c) Describe and analyze an algorithm to find a line ℓ such that exactly half the ghosts and exactly half the Ghostbusters are above ℓ. (Assume n is even.)

(d) Using your algorithm for part (b) or part (c) as a subroutine, describe and analyze an algorithm to find the line segments described in part (a). (Assume n is a power of two if necessary.)

Spengler: *Don't cross the streams.*
Venkman: *Why?*
Spengler: *It would be bad.*
Venkman: *I'm fuzzy on the whole good/bad thing. What do you mean "bad"?*
Spengler: *Try to imagine all life as you know it stopping instantaneously and every molecule in your body exploding at the speed of light.*
Stantz: *Total protonic reversal!*
Venkman: *That's bad. Okay. Alright, important safety tip, thanks Egon.*

— Dr. Egon Spengler (Harold Ramis), Dr. Peter Venkman (Bill Murray), and Dr. Raymond Stanz (Dan Aykroyd), *Ghostbusters*, 1984

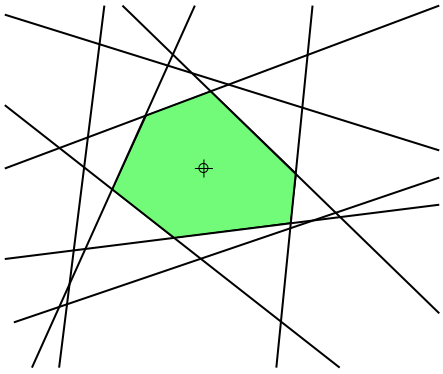
4. The *convex layers* of a point set P consist of a series of nested convex polygons. The convex layers of the empty set are empty. Otherwise, the first layer is just the convex hull of P, and the remaining layers are the convex layers of the points that are not on the convex hull of P.



The convex layers of a set of points.

Describe and analyze an efficient algorithm to compute the convex layers of a given n-point set. For full credit, your algorithm should run in $O(n^2)$ time.

5. Suppose we are given a set of n lines in the plane, where none of the lines passes through the origin $(0, 0)$ and at most two lines intersect at any point. These lines divide the plane into several convex polygonal regions, or *cells*. Describe and analyze an efficient algorithm to compute the cell containing the origin. The output should be a doubly-linked list of the cell's vertices. *[Hint: There are literally dozens of solutions. One solution is to reduce this problem to the convex hull problem. Every other solution looks like a convex hull algorithm.]*



The cell containing the origin in an arrangement of lines.

Write your answers in the separate answer booklet.

1. **Multiple Choice:** Each question below has one of the following answers.

   A: $\Theta(1)$ B: $\Theta(\log n)$ C: $\Theta(n)$ D: $\Theta(n \log n)$ E: $\Theta(n^2)$ X: I don't know.

   For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point. Each X earns you $\frac{1}{4}$ point. **Each incorrect answer *costs* you $\frac{1}{2}$ point.** Your total score will be rounded **down** to an integer. Negative scores will be rounded up to zero.

   (a) What is $\sum_{i=1}^{n} \lg i$?

   (b) What is $\sum_{i=1}^{\lg n} i2^i$?

   (c) How many decimal digits are required write the nth Fibonacci number?

   (d) What is the solution of the recurrence $T(n) = 4T(n/8) + n \log n$?

   (e) What is the solution of the recurrence $T(n) = T(n-3) + \frac{5}{n}$?

   (f) What is the solution of the recurrence $T(n) = 5T\left(\left\lceil \frac{n+13}{3} \right\rceil + \lfloor \sqrt{n} \rfloor\right) + (10n-7)^2 - \frac{\lg^3 n}{\lg \lg n}$?

   (g) How long does it take to construct a Huffman code, given an array of n character frequencies as input?

   (h) How long does it take to sort an array of size n using quicksort?

   (i) Given an unsorted array $A[1..n]$, how long does it take to construct a binary search tree for the elements of A?

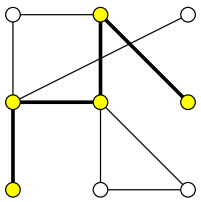
   (j) A train leaves Chicago at 8:00pm and travels south at 75 miles per hour. Another train leaves New Orleans at 1:00pm and travels north at 60 miles per hour. The conductors of both trains are playing a game of chess over the phone. After each player moves, the other player must move before his train has traveled five miles. How many moves do the two players make before their trains pass each other (somewhere near Memphis)?

2. Describe and analyze efficient algorithms to solve the following problems:

   (a) Given a set of n integers, does it contain a pair of elements a, b such that $a + b = 0$?

   (b) Given a set of n integers, does it contain three elements a, b, c such that $a + b = c$?

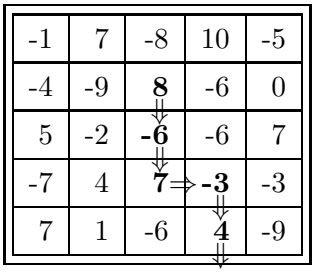
3. A *tonian path* in a graph G is a simple path in G that visits more than half of the vertices of G. (Intuitively, a tonian path is "most of a Hamiltonian path".) Prove that it is NP-hard to determine whether or not a given graph contains a tonian path.



A tonian path in a 9-vertex graph.

4. *Vankin's Mile* is a solitaire game played on an $n \times n$ square grid. The player starts by placing a token on any square of the grid. Then on each turn, the player moves the token either one square to the right or one square down. The game ends when player moves the token off the edge of the board. Each square of the grid has a numerical value, which could be positive, negative, or zero. The player starts with a score of zero; whenever the token lands on a square, the player adds its value to his score. The object of the game is to score as many points as possible.

   For example, given the grid below, the player can score $8 - 6 + 7 - 3 + 4 = 10$ points by placing the initial token on the 8 in the second row, and then moving down, down, right, down, down. (This is *not* the best possible score for these values.)



   Describe and analyze an algorithm to compute the maximum possible score for a game of Vankin's Mile, given the $n \times n$ array of values as input.

5. Suppose you are given two sorted arrays $A[1..m]$ and $B[1..n]$ and an integer k. Describe an algorithm to find the kth smallest element in the union of A and B in $\Theta(\log(m+n))$ time. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1..5] = [2, 5, 8, 17, 19] \qquad k = 6$$

   your algorithm should return 8. You can assume that the arrays contain no duplicates. *[Hint: What can you learn from comparing one element of A to one element of B?]*

Write your answers in the separate answer booklet.

1. **Multiple Choice:** Each question below has one of the following answers.

   A: $\Theta(1)$ B: $\Theta(\log n)$ C: $\Theta(n)$ D: $\Theta(n \log n)$ E: $\Theta(n^2)$ X: I don't know.

   For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point. Each X earns you $\frac{1}{4}$ point. **Each incorrect answer *costs* you $\frac{1}{2}$ point.** Your total score will be rounded **down** to an integer. Negative scores will be rounded up to zero.

   (a) What is $\sum_{i=1}^{n} \frac{n}{i}$?

   (b) What is $\sum_{i=1}^{\lg n} 4^i$?

   (c) How many bits are required to write $n!$ in binary?

   (d) What is the solution of the recurrence $T(n) = 4T(n/2) + n \log n$?

   (e) What is the solution of the recurrence $T(n) = T(n-3) + \frac{5}{n}$?

   (f) What is the solution of the recurrence $T(n) = 9T\left(\lceil \frac{n+13}{3} \rceil\right) + 10n - 7\sqrt{n} - \frac{\lg^3 n}{\lg \lg n}$?

   (g) How long does it search for a value in an n-node binary search tree?

   (h) Given a sorted array $A[1..n]$, how long does it take to construct a binary search tree for the elements of A?

   (i) How long does it take to construct a Huffman code, given an array of n character frequencies as input?

   (j) A train leaves Chicago at 8:00pm and travels south at 75 miles per hour. Another train leaves New Orleans at 1:00pm and travels north at 60 miles per hour. The conductors of both trains are playing a game of chess over the phone. After each player moves, the other player must move before his train has traveled five miles. How many moves do the two players make before their trains pass each other (somewhere near Memphis)?

2. Describe and analyze an algorithm to find the length of the longest substring that appears both forward and backward in an input string $T[1..n]$. The forward and backward substrings must not overlap. Here are several examples:

   - Given the input string ALGORITHM, your algorithm should return 0.
   - Given the input string RECURSION, your algorithm should return 1, for the substring R.
   - Given the input string REDIVIDE, your algorithm should return 3, for the substring EDI. (The forward and backward substrings must not overlap!)
   - Given the input string DYNAMICPROGRAMMINGMANYTIMES, your algorithm should return 4, for the substring YNAM.

   For full credit, your algorithm should run in $O(n^2)$ time.

3. The *median* of a set of size n is its $\lceil n/2 \rceil$th largest element, that is, the element that is as close as possible to the middle of the set in sorted order. It's quite easy to find the median of a set in $O(n \log n)$ time—just sort the set and look in the middle—but you (correctly!) think that you can do better.

   During your lifelong quest for a faster median-finding algorithm, you meet and befriend the Near-Middle Fairy. Given any set X, the Near-Middle Fairy can find an element $m \in X$ that is *near* the middle of X in $O(1)$ time. Specifically, at least a third of the elements of X are smaller than m, and at least a third of the elements of X are larger than m.

   Describe and analyze an algorithm to find the median of a set in $O(n)$ time if you are allowed to ask the Near-Middle Fairy for help. *[Hint: You may need the* PARTITION *subroutine from* QUICKSORT.*]*

4. SUBSETSUM and PARTITION are two closely related NP-hard problems.

   - SUBSETSUM: Given a set X of integers and an integer k, does X have a subset whose elements sum up to k?
   - PARTITION: Given a set X of integers and an integer k, can X be partitioned into two subsets whose sums are equal?

   (a) Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION.

   (b) Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM.

5. Describe and analyze efficient algorithms to solve the following problems:

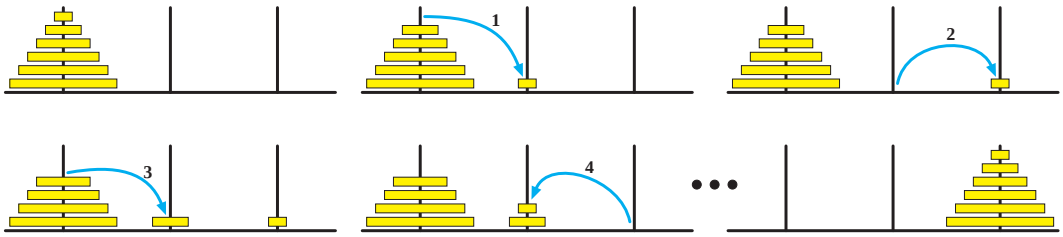
   (a) Given a set of n integers, does it contain a pair of elements a, b such that $a + b = 0$?

   (b) Given a set of n integers, does it contain three elements a, b, c such that $a + b + c = 0$?

Write your answers in the separate answer booklet.

1. In the well-known *Tower of Hanoi* problem, we have three spikes, one of which has a tower of n disks of different sizes, stacked with smaller disks on top of larger ones. In a single step, we are allowed to take the top disk on any spike and move it to the top of another spike. We are never allowed to place a larger disk on top of a smaller one. Our goal is to move all the disks from one spike to another.

   Hmmm.... You've probably known how to solve this problem since CS 125, so make it more interesting, let's add another constraint: The three spikes are arranged in a row, and we are also forbidden to move a disk directly from the left spike to the right spike or vice versa. In other words, we *must* move a disk either to or from the middle spike at *every* step.



The first four steps required to move the disks from the left spike to the right spike.

   (a) **[4 pts]** Describe an algorithm that moves the stack of n disks from the left needle to the right needle in as few steps as possible.

   (b) **[6 pts] Exactly** how many steps does your algorithm take to move all n disks? A correct Θ-bound is worth 3 points. *[Hint: Set up and solve a recurrence.]*

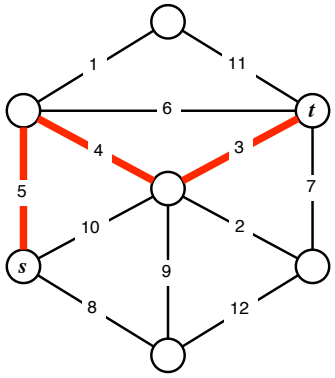
2. Consider a random walk on a path with vertices numbered $1, 2, \ldots, n$ from left to right. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex n. In Midterm 2, you were asked to prove that if we start at vertex 1, the probability that the walk ends by falling off the *left* end of the path is exactly $n/(n+1)$.

   (a) **[6 pts] Prove** that if we start at vertex 1, the expected number of steps before the random walk ends is exactly n. *[Hint: Set up and solve a recurrence. Use the result from Midterm 2.]*

   (b) **[4 pts]** Suppose we start at vertex $n/2$ instead. State a tight Θ-bound on the expected length of the random walk in this case. **No proof is required.** *[Hint: Set up and solve a recurrence. Use part (a), even if you can't prove it.]*

3. **Prove** that any connected acyclic graph with n vertices has exactly $n-1$ edges. Do not use the word "tree" or any well-known properties of trees; your proof should follow entirely from the definitions.

4. Consider a path between two vertices s and t in an undirected weighted graph G. The *bottleneck length* of this path is the maximum weight of any edge in the path. The *bottleneck distance* between s and t is the minimum bottleneck length of any path from s to t. (If there are no paths from u to v, the bottleneck distance between s and t is ∞.)



The bottleneck distance between s and t is 5.

Describe and analyze an algorithm to compute the bottleneck distance between every pair of vertices in an arbitrary undirected weighted graph. Assume that no two edges have the same weight.

5. The 5COLOR asks, given a graph G, whether the vertices of a graph G can be colored with five colors so that no edge has two endpoints with the same color. You already know from class that this problem is NP-complete.

   Now consider the related problem 5COLOR±1: Given a graph G, can we color each vertex with an integer from the set $\{0, 1, 2, 3, 4\}$, so that for every edge, the colors of the two endpoints differ by exactly 1 modulo 5? (For example, a vertex with color 4 can only be adjacent to vertices colored 0 or 3.) We would like to show that 5COLOR±1 is NP-complete as well.

   (a) **[2 pts]** Show that 5COLOR±1 is in NP.

   (b) **[1 pt]** To prove that 5COLOR±1 is NP-hard (and therefore NP-complete), we must describe a polynomial time algorithm for *one* of the following problems. Which one?

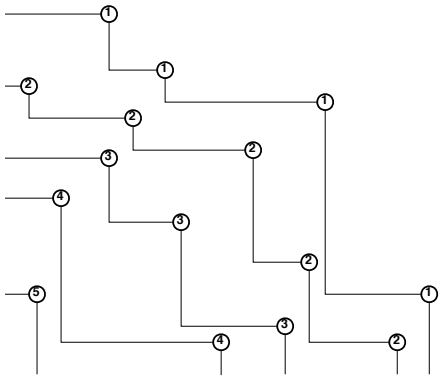
   - Given an arbitrary graph G, compute a graph H such that 5COLOR(G) is true if and only if 5COLOR±1(H) is true.
   - Given an arbitrary graph G, compute a graph H such that 5COLOR±1(G) is true if and only if 5COLOR(H) is true.

   (c) **[1 pt]** Explain briefly why the following argument is not correct.

   > For any graph G, if 5COLOR±1(G) is true, then 5COLOR(G) is true (using the same coloring). Therefore if we could solve 5COLOR±1 quickly, we could also solve 5COLOR quickly. In other words, 5COLOR±1 is at least as hard as 5COLOR. We know that 5COLOR is NP-hard, so 5COLOR±1 must also be NP-hard!

   (d) **[6 pts]** Prove that 5COLOR±1 is NP-hard. *[Hint: Look at some small examples. Replace the edges of G with a simple gadget, so that the resulting graph H has the desired property from part (b).]*

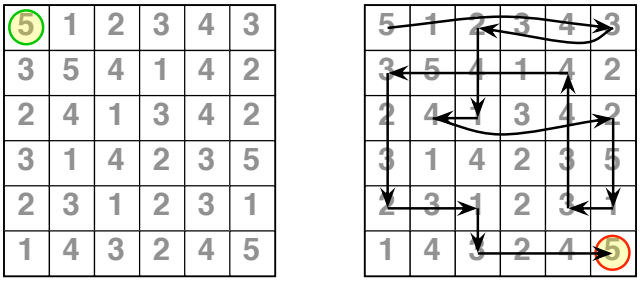
6. Let P be a set of points in the plane. Recall that a point $p \in P$ is *Pareto-optimal* if no other points in P are both above and to the right of p. Intuitively, the sequence of Pareto-optimal points forms a *staircase* with all the other points in P below and to the left. The *staircase layers* of P are defined recursively as follows. The empty set has no staircase layers. Otherwise, the first staircase layer contains all the Pareto-optimal points in P, and the remaining layers are the staircase layers of P minus the first layer.



A set of points with 5 staircase layers

Describe and analyze an algorithm to compute the number of staircase layers of a point set P as quickly as possible. For example, given the points illustrated above, your algorithm would return the number 5.

7. Consider the following puzzle played on an $n \times n$ square grid, where each square is labeled with a positive integer. A token is placed on one of the squares. At each turn, you may move the token left, right, up, or down; the distance you move the token must be equal to the number on the current square. For example, if the token is on a square labeled "3", you are allowed more the token three squares down, three square left, three squares up, or three squares right. You are never allowed to move the token off the board.



A sequence of legal moves from the top left corner to the bottom right corner.

(a) **[4 pts]** Describe and analyze an algorithm to determine, given an $n \times n$ array of labels and two squares s and t, whether there is a sequence of legal moves that takes the token from s to t.

(b) **[6 pts]** Suppose you are only given the $n \times n$ array of labels. Describe how to preprocess these values, so that afterwards, given any two squares s and t, you can determine in $O(1)$ time whether there is a sequence of legal moves from s to t.

Answer four of these seven problems; the lowest three scores will be dropped.

1. Suppose we are given an array $A[1..n]$ with the special property that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. We say that an element $A[x]$ is a *local minimum* if it is less than or equal to both its neighbors, or more formally, if $A[x-1] \geq A[x]$ and $A[x] \leq A[x+1]$. For example, there are five local minima in the following array:

9	7	7	2	**1**	3	7	5	**4**	7	**3**	**3**	4	8	**6**	9

   We can obviously find a local minimum in $O(n)$ time by scanning through the array. Describe and analyze an algorithm that finds a local minimum in $O(\log n)$ time. *[Hint: With the given boundary conditions, the array **must** have at least one local minimum. Why?]*

2. Consider a random walk on a path with vertices numbered $1, 2, \ldots, n$ from left to right. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex n. In Midterm 2, you were asked to prove that if we start at vertex 1, the probability that the walk ends by falling off the *left* end of the path is exactly $n/(n+1)$.

   (a) **[6 pts]** **Prove** that if we start at vertex 1, the expected number of steps before the random walk ends is exactly n. *[Hint: Set up and solve a recurrence. Use the result from Midterm 2.]*

   (b) **[4 pts]** Suppose we start at vertex $n/2$ instead. State **and prove** a tight Θ-bound on the expected length of the random walk in this case. *[Hint: Set up and solve a recurrence. Use part (a), even if you can't prove it.]*

3. **Prove** that any connected acyclic graph with $n \geq 2$ vertices has at least two vertices with degree 1. Do not use the words "tree" of "leaf", or any well-known properties of trees; your proof should follow entirely from the definitions.

4. Consider the following sketch of a "reverse greedy" algorithm. The input is a connected undirected graph G with weighted edges, represented by an adjacency list.

```
ReverseGreedyMST(G):
  sort the edges E of G by weight
  for i ← 1 to |E|
      e ← ith heaviest edge in E
      if G \ e is connected
          remove e from G
```

   (a) **[4 pts]** What is the worst-case running time of this algorithm? (Answering this question will require fleshing out a few details.)

   (b) **[6 pts]** **Prove** that the algorithm transforms G into its minimum spanning tree.

5. SUBSETSUM and PARTITION are two closely related NP-hard problems.

   - SUBSETSUM: Given a set X of integers and an integer k, does X have a subset whose elements sum up to k?
   - PARTITION: Given a set X of integers, can X be partitioned into two subsets whose sums are equal?

   (a) **[2 pts]** Prove that PARTITION and SUBSETSUM are both in NP.

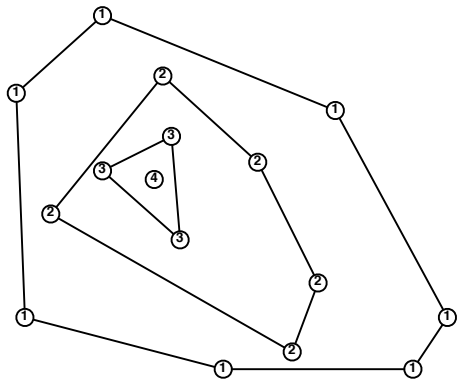
   (b) **[1 pt]** Suppose we knew that SUBSETSUM is NP-hard, and we wanted to prove that PARTITION is NP-hard. Which of the following arguments should we use?

   - Given a set X and an integer k, compute a set Y such that PARTITION(Y) is true if and only if SUBSETSUM(X, k) is true.
   - Given a set X, construct a set Y and an integer k such that PARTITION(X) is true if and only if SUBSETSUM(Y, k) is true.

   (c) **[3 pts]** Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM. (See part (b).)

   (d) **[4 pts]** Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION. (See part (b).)

6. Let P be a set of points in the plane. The *convex layers* of P are defined recursively as follows. If P is empty, it ha no convex layers. Otherwise, the first convex layer is the convex hull of P, and the remaining convex layers are the convex layers of P minus its convex hull.

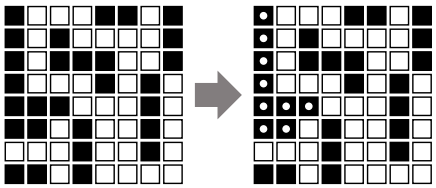


A set of points with 4 convex layers

Describe and analyze an algorithm to compute the number of convex layers of a point set P as quickly as possible. For example, given the points illustrated above, your algorithm would return the number 4.

7. (a) **[4 pts]** Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an $n \times n$ bitmap $B[1..n, 1..n]$.

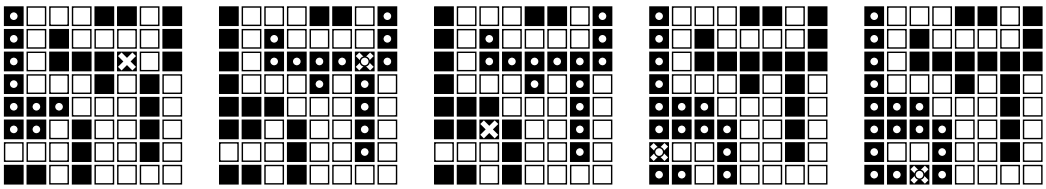
   For example, given the bitmap below as input, your algorithm should return the number 9, because the largest conected black component (marked with white dots on the right) contains nine pixels.



9

   (b) **[4 pts]** Design and analyze an algorithm BLACKEN(i, j) that colors the pixel $B[i, j]$ black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.

   For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the BLACKEN algorithm.



9 14 14 16 17

   (c) **[2 pts]** What is the *worst-case* running time of your BLACKEN algorithm?

Write your answers in the separate answer booklet.

1. A *data stream* is an extremely long sequence of items that you can only read only once, in order. A good example of a data stream is the sequence of packets that pass through a router. Data stream algorithms must process each item in the stream quickly, using very little memory; there is simply too much data to store, and it arrives too quickly for any complex computations. Every data stream algorithm looks roughly like this:

```
DoSomethingInteresting(stream S):
  repeat
      x ← next item in S
      ⟨⟨do something fast with x⟩⟩
  until S ends
  return ⟨⟨something⟩⟩
```

   Describe and analyze an algorithm that chooses one element uniformly at random from a data stream, *without knowing the length of the stream in advance*. Your algorithm should spend $O(1)$ time per stream element and use $O(1)$ space (not counting the stream itself). Assume you have a subroutine RANDOM(n) that returns a random integer between 1 and n, each with equal probability, given any integer n as input.

2. Consider a random walk on a path with vertices numbered $1, 2, \ldots, n$ from left to right. We start at vertex 1. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex n.

   Prove that the probability that the walk ends by falling off the *left* end of the path is exactly $n/(n+1)$. *[Hint: Set up a recurrence and verify that $n/(n+1)$ satisfies it.]*

3. Consider the following algorithms for maintaining a family of disjoint sets. The UNION algorithm uses a heuristic called *union by size*.

```
MakeSet(x):
  parent(x) ← x
  size(x) ← 1
```

```
Find(x):
  while x ≠ parent(x)
      x ← parent(x)
  return x
```

```
Union(x, y):
  x̄ ← Find(x)
  ȳ ← Find(y)
  if size(x̄) < size(ȳ)
      parent(x̄) ← ȳ
      size(x̄) ← size(x̄) + size(ȳ)
  else
      parent(ȳ) ← x̄
      size(ȳ) ← size(x̄) + size(ȳ)
```

   Prove that if we use union by size, FIND(x) runs in $O(\log n)$ time *in the worst case*, where n is the size of the set containing element x.

4. Recall the SUBSETSUM problem: Given a set X of integers and an integer k, does X have a subset whose elements sum to k?

   (a) **[7 pts]** Describe and analyze an algorithm that solves SUBSETSUM in time $O(nk)$.

   (b) **[3 pts]** SUBSETSUM is NP-hard. Does part (a) imply that P=NP? Justify your answer.

5. Suppose we want to maintain a set X of numbers under the following operations:

   - INSERT(x): Add x to the set X.
   - PRINTANDDELETEBETWEEN(a, z): Print every element $x \in X$ such that $a \leq x \leq z$, in order from smallest to largest, and then delete those elements from X.

   For example, PRINTANDDELETEBETWEEN($-\infty, \infty$) prints all the elements of X in sorted order and then deletes everything.

   (a) **[6 pts]** Describe and analyze a data structure that supports these two operations, each in $O(\log n)$ *amortized* time, where n is the maximum number of elements in X.

   (b) **[2 pts]** What is the running time of your INSERT algorithm in the worst case?

   (c) **[2 pts]** What is the running time of your PRINTANDDELETEBETWEEN algorithm in the worst case?

CS 473G: Combinatorial Algorithms, Fall 2005

Homework 0

Due Thursday, September 1, 2005, at the beginning of class (12:30pm CDT)

Name:	
Net ID:	Alias:

☐ I understand the Homework Instructions and FAQ.

- Neatly print your full name, your NetID, and an alias of your choice in the boxes above. Grades will be listed on the course web site by alias. Please write the same alias on every homework and exam! For privacy reasons, your alias should not resemble your name or NetID. By providing an alias, you agree to let us list your grades; if you do not provide an alias, your grades will not be listed. ***Never*** *give us your Social Security number!*

- Read the "Homework Instructions and FAQ" on the course web page, and then check the box above. This page describes what we expect in your homework solutions—start each numbered problem on a new sheet of paper, write your name and NetID on every page, don't turn in source code, analyze and prove everything, use good English and good logic, and so on—as well as policies on grading standards, regrading, and plagiarism. **See especially the course policies regarding the magic phrases "I don't know" and "and so on".** If you have *any* questions, post them to the course newsgroup or ask during lecture.

- Don't forget to submit this cover sheet with the rest of your homework solutions.

- This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, discrete probability, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Chapters 1–10 of CLRS should be sufficient review, but you may also want consult your discrete mathematics and data structures textbooks.

- Every homework will have five required problems. Most homeworks will also include one extra-credit problem and several practice (no-credit) problems. Each numbered problem is worth 10 points.

#	1	2	3	4	5	6*	Total
Score							
Grader							

1. Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway, just for practice. Assume reasonable but nontrivial base cases. **If your solution requires specific base cases, state them!**

   (a) $A(n) = 2A(n/4) + \sqrt{n}$

   (b) $B(n) = \max_{n/3<k<2n/3} \big(B(k) + B(n-k) + n\big)$

   (c) $C(n) = 3C(n/3) + n/\lg n$

   (d) $D(n) = 3D(n-1) - 3D(n-2) + D(n-3)$

   (e) $E(n) = \dfrac{E(n-1)}{3E(n-2)}$ *[Hint: This is easy!]*

   (f) $F(n) = F(n-2) + 2/n$

   (g) $G(n) = 2G\big(\lceil (n+3)/4 \rceil - 5n/\sqrt{\lg n} + 6\lg\lg n\big) + 7\sqrt[8]{n-9} - \lg^{10} n/\lg\lg n + 11^{\lg^* n} - 12$

   *(h) $H(n) = 4H(n/2) - 4H(n/4) + 1$ *[Hint: Careful!]*

   (i) $I(n) = I(n/2) + I(n/4) + I(n/8) + I(n/12) + I(n/24) + n$

   ★(j) $J(n) = 2\sqrt{n} \cdot J(\sqrt{n}) + n$
   [Hint: First solve the secondary recurrence $j(n) = 1 + j(\sqrt{n})$.]

2. Penn and Teller agree to play the following game. Penn shuffles a standard deck[1] of playing cards so that every permutation is equally likely. Then Teller draws cards from the deck, one at a time without replacement, until he draws the three of clubs (3♣), at which point the remaining undrawn cards instantly burst into flames and the game is over.

   The first time Teller draws a card from the deck, he gives it to Penn. From then on, until the game ends, whenever Teller draws a card whose value is smaller than the previous card he gave to Penn, he gives the new card to Penn. To make the rules unambiguous, they agree on the numerical values $A = 1$, $J = 11$, $Q = 12$, and $K = 13$.

   (a) What is the expected number of cards that Teller draws?

   (b) What is the expected *maximum* value among the cards Teller gives to Penn?

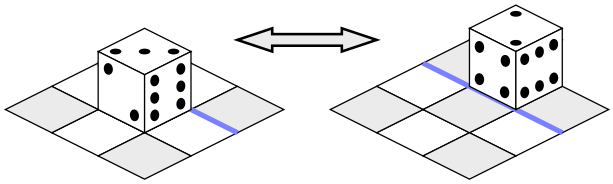
   (c) What is the expected *minimum* value among the cards Teller gives to Penn?

   (d) What is the expected number of cards that Teller gives to Penn?

   Full credit will be given only for *exact* answers (with correct proofs, of course).

[1] In a standard deck of 52 cards, each card has a *suit* in the set $\{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\}$ and a *value* in the set $\{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\}$, and every possible suit-value pair appears in the deck exactly once. Penn and Teller normally use exploding razor-sharp ninja throwing cards for this trick.

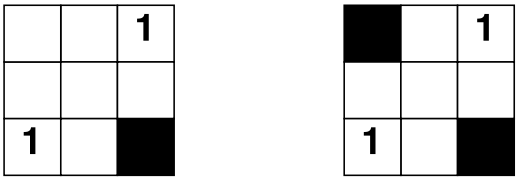
3. A *rolling die maze* is a puzzle involving a standard six-sided die[2] and a grid of squares. You should imagine the grid lying on top of a table; the die always rests on and exactly covers one square. In a single step, you can *roll* the die 90 degrees around one of its bottom edges, moving it to an adjacent square one step north, south, east, or west.



Rolling a die.

Some squares in the grid may be *blocked*; the die can never rest on a blocked square. Other squares may be *labeled* with a number; whenever the die rests on a labeled square, the number of pips on the *top* face of the die must equal the label. Squares that are neither labeled nor marked are *free*. You may not roll the die off the edges of the grid. A rolling die maze is *solvable* if it is possible to place a die on the lower left square and roll it to the upper right square under these constraints.

For example, here are two rolling die mazes. Black squares are blocked. The maze on the left can be solved by placing the die on the lower left square with 1 pip on the top face, and then rolling it north, then north, then east, then east. The maze on the right is not solvable.



Two rolling die mazes. Only the maze on the left is solvable.

(a) Suppose the input is a two-dimensional array $L[1..n][1..n]$, where each entry $L[i][j]$ stores the label of the square in the ith row and jth column, where 0 means the square is free and -1 means the square is blocked. Describe and analyze a polynomial-time algorithm to determine whether the given rolling die maze is solvable.

*(b) Now suppose the maze is specified *implicitly* by a list of labeled and blocked squares. Specifically, suppose the input consists of an integer M, specifying the height and width of the maze, and an array $S[1..n]$, where each entry $S[i]$ is a triple (x, y, L) indicating that square (x, y) has label L. As in the explicit encoding, label -1 indicates that the square is blocked; free squares are not listed in S at all. Describe and analyze an efficient algorithm to determine whether the given rolling die maze is solvable. For full credit, the running time of your algorithm should be polynomial in the input size n.

[Hint: You have some freedom in how to place the initial die. There are rolling die mazes that can only be solved if the initial position is chosen correctly.]

[2] A standard die is a cube, where each side is labeled with a different number of dots, called *pips*, between 1 and 6. The labeling is chosen so that any pair of opposite sides has a total of 7 pips.

4. Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons will always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. (Like most things, revenge is a foreign concept to pigeons.) Surprisingly, the overall pecking order in a set of pigeons can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

   Prove that any set of pigeons can be arranged in a row so that every pigeon pecks the pigeon immediately to its right.

5. Scientists have recently discovered a planet, tentatively named "Ygdrasil", which is inhabited by a bizarre species called "vodes". All vodes trace their ancestry back to a particular vode named Rudy. Rudy is still quite alive, as is every one of his many descendants. Vodes reproduce asexually, like bees; each vode has exactly one parent (except Rudy, who has no parent). There are three different colors of vodes—cyan, magenta, and yellow. The color of each vode is correlated exactly with the number and colors of its children, as follows:

   - Each cyan vode has two children, exactly one of which is yellow.
   - Each yellow vode has exactly one child, which is not yellow.
   - Magenta vodes have no children.

   In each of the following problems, let C, M, and Y respectively denote the number of cyan, magenta, and yellow vodes on Ygdrasil.

   (a) Prove that $M = C + 1$.

   (b) Prove that either $Y = C$ or $Y = M$.

   (c) Prove that $Y = M$ if and only if Rudy is yellow.

   [Hint: Be very careful to ***prove*** *that you have considered* ***all*** *possibilities.]*

⋆6. **[Extra credit]**[3]

Lazy binary is a variant of standard binary notation for representing natural numbers where we allow each "bit" to take on one of three values: 0, 1, or 2. Lazy binary notation is defined inductively as follows.

- The lazy binary representation of zero is 0.
- Given the lazy binary representation of any non-negative integer n, we can construct the lazy binary representation of $n + 1$ as follows:
  (a) increment the rightmost digit;
  (b) if any digit is equal to 2, replace the rightmost 2 with 0 and increment the digit immediately to its left.

Here are the first several natural numbers in lazy binary notation:

0, 1, 10, 11, 20, 101, 110, 111, 120, 201, 210, 1011, 1020, 1101, 1110, 1111, 1120, 1201, 1210, 2011, 2020, 2101, 2110, 10111, 10120, 10201, 10210, 11011, 11020, 11101, 11110, 11111, 11120, 11201, 11210, 12011, 12020, 12101, 12110, 20111, 20120, 20201, 20210, 21011, 21020, 21101, 21110, 101111, 101120, 101201, 101210, 102011, 102020, 102101, 102110, ...

(a) Prove that in any lazy binary number, between any two 2s there is at least one 0, and between two 0s there is at least one 2.

(b) Prove that for any natural number N, the sum of the digits of the lazy binary representation of N is exactly $\lfloor \lg(N + 1) \rfloor$.

[3]The "I don't know" rule does not apply to extra credit problems. There is no such thing as "partial extra credit".

Practice Problems

The remaining problems are for practice only. Please do not submit solutions. On the other hand, feel free to discuss these problems in office hours or on the course newsgroup.

1. Sort the functions in each box from asymptotically smallest to asymptotically largest, indicating ties if there are any. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway, just for practice.

1	$\lg n$	$\lg^2 n$	$\sqrt{n}$	n	n^2	$2^{\sqrt{n}}$	$\sqrt{2}^n$
$2^{\sqrt{\lg n}}$	$2^{\lg\sqrt{n}}$	$\sqrt{2^{\lg n}}$	$\sqrt{2}^{\lg n}$	$\lg 2^{\sqrt{n}}$	$\lg \sqrt{2}^n$	$\lg \sqrt{2^n}$	$\sqrt{\lg 2^n}$
$\lg n^{\sqrt{2}}$	$\lg \sqrt{n}^2$	$\lg \sqrt{n^2}$	$\sqrt{\lg n^2}$	$\lg^2 \sqrt{n}$	$\lg^{\sqrt{2}} n$	$\sqrt{\lg^2 n}$	$\sqrt{\lg n}^2$

   To simplify your answers, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$, and write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions $n^2, n, \binom{n}{2}, n^3$ could be sorted either as $n \ll n^2 \equiv \binom{n}{2} \ll n^3$ or as $n \ll \binom{n}{2} \equiv n^2 \ll n^3$.

2. Recall the standard recursive definition of the Fibonacci numbers: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Prove the following identities for all positive integers n and m.

   (a) F_n is even if and only if n is divisible by 3.

   (b) $\sum_{i=0}^{n} F_i = F_{n+2} - 1$

   (c) $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$

   ★(d) If n is an integer multiple of m, then F_n is an integer multiple of F_m.

3. Penn and Teller have a special deck of fifty-two cards, with no face cards and nothing but clubs—the ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ..., 52 of clubs. (They're big cards.) Penn shuffles the deck until each each of the 52! possible orderings of the cards is equally likely. He then takes cards one at a time from the top of the deck and gives them to Teller, stopping as soon as he gives Teller the three of clubs.

   (a) On average, how many cards does Penn give Teller?

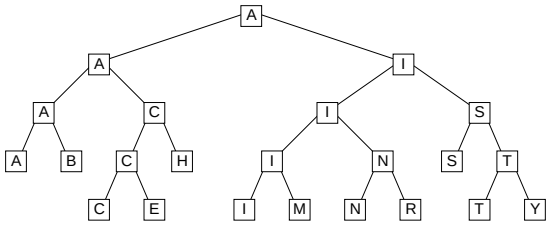
   (b) On average, what is the smallest-numbered card that Penn gives Teller?

   ⋆(c) On average, what is the largest-numbered card that Penn gives Teller?

   Prove that your answers are correct. (If you have to appeal to "intuition" or "common sense", your answers are probably wrong.) *[Hint: Solve for an n-card deck, and then set n to 52.]*

4. Algorithms and data structures were developed millions of years ago by the Martians, but not quite in the same way as the recent development here on Earth. Intelligent life evolved independently on Mars' two moons, Phobos and Deimos.[4] When the two races finally met on the surface of Mars, after thousands of years of separate philosophical, cultural, religious, and scientific development, their disagreements over the proper structure of binary search trees led to a bloody (or more accurately, ichorous) war, ultimately leading to the destruction of all Martian life.

   A *Phobian* binary search tree is a full binary tree that stores a set X of search keys. The root of the tree stores the *smallest* element in X. If X has more than one element, then the left subtree stores all the elements less than some pivot value p, and the right subtree stores everything else. Both subtrees are *nonempty* Phobian binary search trees. The actual pivot value p is *never* stored in the tree.



A Phobian binary search tree for the set {M, A, R, T, I, N, B, Y, S, E, C, H}.

   (a) Describe and analyze an algorithm FIND(x, T) that returns TRUE if x is stored in the Phobian binary search tree T, and FALSE otherwise.

   (b) A *Deimoid* binary search tree is almost exactly the same as its Phobian counterpart, except that the *largest* element is stored at the root, and both subtrees are Deimoid binary search trees. Describe and analyze an algorithm to transform an n-node Phobian binary search tree into a Deimoid binary search tree in $O(n)$ time, using as little additional space as possible.

5. Tatami are rectangular mats used to tile floors in traditional Japanese houses. Exact dimensions of tatami mats vary from one region of Japan to the next, but they are always twice as long in one dimension than in the other. (In Tokyo, the standard size is 180cm×90cm.)

   (a) How many different ways are there to tile a $2 \times n$ rectangular room with 1×2 tatami mats? Set up a recurrence and derive an *exact* closed-form solution. *[Hint: The answer involves a familiar recursive sequence.]*

   (b) According to tradition, tatami mats are always arranged so that four corners never meet. How many different *traditional* ways are there to tile a $3 \times n$ rectangular room with 1×2 tatami mats? Set up a recurrence and derive an *exact* closed-form solution.

   *(c) How many different *traditional* ways are there to tile an $n \times n$ square with 1×2 tatami mats? Prove your answer is correct.

[4] Greek for "fear" and "panic", respectively. Doesn't that make you feel better?

CS 473G: Combinatorial Algorithms, Fall 2005

Homework 1

Due Tuesday, September 13, 2005, by midnight (11:59:59pm CDT)

Name:	
Net ID:	Alias:

Name:	
Net ID:	Alias:

Name:	
Net ID:	Alias:

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your answer to problem 1.

There are two steps required to prove NP-completeness: (1) Prove that the problem is in NP, by describing a polynomial-time verification algorithm. (2) Prove that the problem is NP-hard, by describing a polynomial-time reduction from some other NP-hard problem. Showing that the reduction is correct requires proving an if-and-only-if statement; don't forget to prove both the "if" part and the "only if" part.

Required Problems

1. Some NP-Complete problems

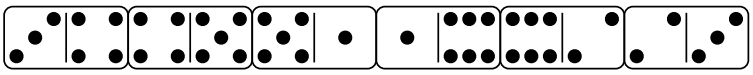
   (a) Show that the problem of deciding whether one graph is a subgraph of another is NP-complete.

   (b) Given a boolean circuit that embeds in the plane so that no 2 wires cross, PLANARCIRCUITSAT is the problem of determining if there is a boolean assignment to the inputs that makes the circuit output true. Prove that PLANARCIRCUITSAT is NP-Complete.

   (c) Given a set S with $3n$ numbers, 3PARTITION is the problem of determining if S can be partitioned into n disjoint subsets, each with 3 elements, so that every subset sums to the same value. Given a set S and a collection of three element subsets of S, X3M (or *exact 3-dimensional matching*) is the problem of determining whether there is a subcollection of n disjoint triples that exactly cover S.

   Describe a polynomial-time reduction from 3PARTITION to X3M.

(d) A *domino* is a 1×2 rectangle divided into two squares, each of which is labeled with an integer.[1] In a *legal arrangement* of dominoes, the dominoes are lined up end-to-end so that the numbers on adjacent ends match.



A legal arrangement of dominos, where every integer between 1 and 6 appears twice

Prove that the following problem is NP-complete: Given an arbitrary collection D of dominoes, is there a legal arrangement of a subset of D in which every integer between 1 and n appears exactly twice?

2. Prove that the following problems are all polynomial-time equivalent, that is, if *any* of these problems can be solved in polynomial time, then *all* of them can.

   - CLIQUE: Given a graph G and an integer k, does there exist a clique of size k in G?
   - FINDCLIQUE: Given a graph G and an integer k, find a clique of size k in G if one exists.
   - MAXCLIQUE: Given a graph G, find the size of the largest clique in the graph.
   - FINDMAXCLIQUE: Given a graph G, find a clique of maximum size in G.

3. Consider the following problem: Given a set of n points in the plane, find a set of line segments connecting the points which form a closed loop and do not intersect each other.

   Describe a linear time reduction from the problem of sorting n numbers to the problem described above.

4. In graph coloring, the vertices of a graph are assigned colors so that no adjacent vertices recieve the same color. We saw in class that determining if a graph is 3-colorable is NP-Complete.

   Suppose you are handed a magic black box that, given a graph as input, tells you *in constant time* whether or not the graph is 3-colorable. Using this black box, give a *polynomial-time* algorithm to 3-color a graph.

5. Suppose that Cook had proved that graph coloring was NP-complete first, instead of CIRCUITSAT. Using only the fact that graph coloring is NP-complete, show that CIRCUITSAT is NP-complete.

[1] These integers are usually represented by pips, exactly like dice. On a standard domino, the number of pips on each side is between 0 and 6; we will allow arbitrary integer labels. A standard set of dominoes has one domino for each possible unordered pair of labels; we do not require that every possible label pair is in our set.

Practice Problems

1. Given an initial configuration consisting of an undirected graph $G = (V, E)$ and a function $p : V \to \mathbb{N}$ indicating an initial number of pebbles on each vertex, PEBBLE-DESTRUCTION asks if there is a sequence of pebbling moves starting with the initial configuration and ending with a single pebble on only one vertex of V. Here, a pebbling move consists of removing two pebbles from a vertex v and adding one pebble to a neighbor of v. Prove that PEBBLE-DESTRUCTION is NP-complete.

2. Consider finding the median of 5 numbers by using only comparisons. What is the *exact* worst case number of comparisons needed to find the median? To prove your answer is correct, you must exhibit both an algorithm that uses that many comparisons and a proof that there is no faster algorithm. Do the same for 6 numbers.

3. PARTITION is the problem of deciding, given a set S of numbers, whether it can be partitioned into two subsets whose sums are equal. (A *partition* of S is a collection of disjoint subsets whose union is S.) SUBSETSUM is the problem of deciding, given a set S of numbers and a target sum t, whether any subset of number in S sum to t.

   (a) Describe a polynomial-time reduction from SUBSETSUM to PARTITION.

   (b) Describe a polynomial-time reduction from PARTITION to SUBSETSUM.

4. Recall from class that the problem of deciding whether a graph can be colored with three colors, so that no edge joins nodes of the same color, is NP-complete.

   (a) Using the gadget in Figure 1(a), prove that deciding whether a *planar* graph can be 3-colored is NP-complete. *[Hint: Show that the gadget can be 3-colored, and then replace any crossings in a planar embedding with the gadget appropriately.]*

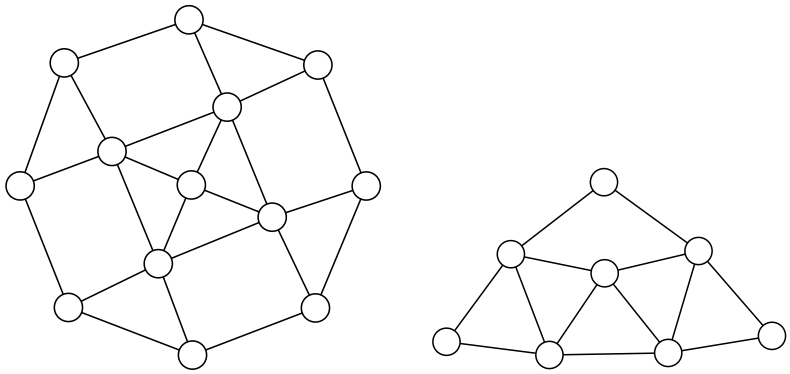


Figure 1. (a) Gadget for planar 3-colorability. (b) Gadget for degree-4 planar 3-colorability.

   (b) Using the previous result and the gadget in figure 1(b), prove that deciding whether a planar graph *with maximum degree 4* can be 3-colored is NP-complete. *[Hint: Show that you can replace any vertex with degree greater than 4 with a collection of gadgets connected in such a way that no degree is greater than four.]*

5. (a) Prove that if G is an undirected bipartite graph with an odd number of vertices, then G is nonhamiltonian. Describe a polynomial-time algorithm to find a hamiltonian cycle in an undirected bipartite graph, or establish that no such cycle exists.

   (b) Describe a polynomial time algorithm to find a hamiltonian *path* in a directed acyclic graph, or establish that no such path exists.

   (c) Why don't these results imply that P=NP?

6. Consider the following pairs of problems:

   (a) MIN SPANNING TREE and MAX SPANNING TREE

   (b) SHORTEST PATH and LONGEST PATH

   (c) TRAVELING SALESMAN PROBLEM and VACATION TOUR PROBLEM (the longest tour is sought).

   (d) MIN CUT and MAX CUT (between s and t)

   (e) EDGE COVER and VERTEX COVER

   (f) TRANSITIVE REDUCTION and MIN EQUIVALENT DIGRAPH

   (all of these seem dual or opposites, except the last, which are just two versions of minimal representation of a graph).

   Which of these pairs are polytime equivalent and which are not? Why?

7. Prove that PRIMALITY (Given n, is n prime?) is in NP $\cap$ co-NP. *[Hint: co-NP is easy—What's a certificate for showing that a number is composite? For NP, consider a certificate involving primitive roots and recursively their primitive roots. Show that this tree of primitive roots can be verified an used to show that n is prime in polynomial time.]*

8. How much wood would a woodchuck chuck if a woodchuck could chuck wood?

CS 473G: Combinatorial Algorithms, Fall 2005
Homework 2

Due Thursday, September 22, 2005, by midnight (11:59:59pm CDT)

Name:	
Net ID:	Alias:

Name:	
Net ID:	Alias:

Name:	
Net ID:	Alias:

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your homework.

Required Problems

1. (a) Suppose Lois has an algorithm to compute the shortest common supersequence of two arrays of integers in $O(n)$ time. Describe an $O(n \log n)$-time algorithm to compute the longest common subsequence of two arrays of integers, using Lois's algorithm as a subroutine.

   (b) Describe an $O(n \log n)$-time algorithm to compute the longest increasing subsequence of an array of integers, using Lois's algorithm as a subroutine.

   (c) Now suppose Lisa has an algorithm that can compute the longest increasing subsequence of an array of integers in $O(n)$ time. Describe an $O(n \log n)$-time algorithm to compute the longest common subsequence of two arrays $A[1..n]$ and $B[1..n]$ of integers, **where $A[i] \neq A[j]$ for all $i \neq j$**, using Lisa's algorithm as a subroutine.[1]

[1] For extra credit, remove the assumption that the elements of A are distinct. This is probably impossible.

2. In a previous incarnation, you worked as a cashier in the lost 19th-century Antarctican colony of Nadira, spending the better part of your day giving change to your customers. Because paper is a very rare and valuable resource on Antarctica, cashiers were required by law to use the fewest bills possible whenever they gave change. Thanks to the numerological predilections of one of its founders, the currency of Nadira, called Dream Dollars, was available in the following denominations: \$1, \$4, \$7, \$13, \$28, \$52, \$91, \$365.[2]

   (a) The greedy change algorithm repeatedly takes the largest bill that does not exceed the target amount. For example, to make \$122 using the greedy algorithm, we first take a \$91 bill, then a \$28 bill, and finally three \$1 bills. Give an example where this greedy algorithm uses more Dream Dollar bills than the minimum possible.

   (b) Describe and analyze an efficient algorithm that computes, given an integer n, the minimum number of bills needed to make n Dream Dollars.

3. Scientists have branched out from the bizarre planet of Yggdrasil to study the vodes which have settled on Ygdrasil's moon, Xryltcon. All vodes on Xryltcon are descended from the first vode to arrive there, named George. Each vode has a color, either cyan, magenta, or yellow, but breeding patterns are *not* the same as on Yggdrasil; every vode, regardless of color, has either two children (with arbitrary colors) or no children.

   George and all his descendants are alive and well, and they are quite excited to meet the scientists who wish to study them. Unsurprisingly, these vodes have had some strange mutations in their isolation on Xryltcon. Each vode has a *weirdness* rating; weirder vodes are more interesting to the visiting scientists. (Some vodes even have negative weirdness ratings; they make other vodes more boring just by standing next to them.)

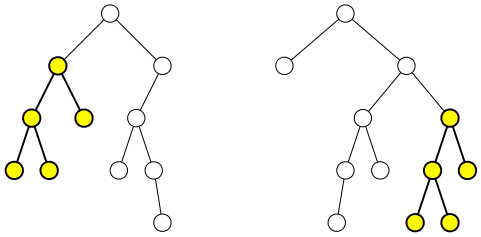
   Also, Xryltconian society is strictly governed by a number of sacred cultural traditions.

   - No cyan vode may be in the same room as its non-cyan children (if it has any).
   - No magenta vode may be in the same room as its parent (if it has one).
   - Each yellow vode must be attended at all times by its grandchildren (if it has any).
   - George must be present at any gathering of more than fifty vodes.

   The scientists have exactly one chance to study a group of vodes in a single room. You are given the family tree of all the vodes on Xryltcon, along with the wierdness value of each vode. Design and analyze an efficient algorithm to decide which vodes the scientists should invite to maximize the sum of the wierdness values of the vodes in the room. Be careful to respect all of the vodes' cultural taboos.

[2] For more details on the history and culture of Nadira, including images of the various denominations of Dream Dollars, see http://www.dream-dollars.com. Really.

4. A *subtree* of a (rooted, ordered) binary tree T consists of a node and all its descendants. Design and analyze an efficient algorithm to compute the *largest common subtree* of two given binary trees T_1 and T_2, that is, the largest subtree of T_1 that is isomorphic to a subtree in T_2. The *contents* of the nodes are irrelevant; we are only interested in matching the underlying combinatorial structure.



Two binary trees, with their largest common subtree emphasized

5. Let $D[1..n]$ be an array of digits, each an integer between 0 and 9. An *digital subsequence* of D is an sequence of positive integers composed in the usual way from disjoint substrings of D. For example, $3, 4, 5, 6, 23, 38, 62, 64, 83, 279$ is an increasing digital subsequence of the first several digits of π:

$$\boxed{3}, 1, \boxed{4}, 1, \boxed{5}, 9, \boxed{6}, \boxed{2,3}, 4, \boxed{3,8}, 4, \boxed{6,2}, \boxed{6,4}, 3, 3, \boxed{8,3}, \boxed{2,7,9}$$

The *length* of a digital subsequence is the number of integers it contains, *not* the number of digits; the previous example has length 10.

Describe and analyze an efficient algorithm to compute the longest increasing digital subsequence of D. *[Hint: Be careful about your computational assumptions. How long does it take to compare two k-digit numbers?]*

*6. *[Extra credit]* The *chromatic number* of a graph G is the minimum number of colors needed to color the nodes of G so that no pair of adjacent nodes have the same color.

   (a) Describe and analyze a *recursive* algorithm to compute the chromatic number of an n-vertex graph in $O(4^n \operatorname{poly}(n))$ time. *[Hint: Catalan numbers play a role here.]*

   (b) Describe and analyze an algorithm to compute the chromatic number of an n-vertex graph in $O(3^n \operatorname{poly}(n))$ time. *[Hint: Use dynamic programming. What is $(1+x)^n$?]*

   (c) Describe and analyze an algorithm to compute the chromatic number of an n-vertex graph in $O((1+3^{1/3})^n \operatorname{poly}(n))$ time. *[Hint: Use (but don't regurgitate) the algorithm in the lecture notes that counts all the maximal independent sets in an n-vertex graph in $O(3^{n/3})$ time.]*

Practice Problems

*1. Describe an algorithm to solve 3SAT in time $O(\phi^n \operatorname{poly}(n))$, where $\phi = (1+\sqrt{5})/2$. *[Hint: Prove that in each recursive call, either you have just eliminated a pure literal, or the formula has a clause with at most two literals.]*

2. Describe and analyze an algorithm to compute the longest increasing subsequence in an n-element array of integers in $O(n \log n)$ time. *[Hint: Modify the $O(n^2)$-time algorithm presented in class.]*

3. The *edit distance* between two strings A and B, denoted $\text{Edit}(A, B)$, is the minimum number of insertions, deletions, or substitutions required to transform A into B (or vice versa). Edit distance is sometimes also called the *Levenshtein distance*.

   Let $\mathbf{A} = \{A_1, A_2, \ldots, A_k\}$ be a set of strings. The *edit radius* of $\mathbf{A}$ is the minimum over all strings X of the maximum edit distance from X to any string A_i:

$$\text{EditRadius}(\mathbf{A}) = \min_{\text{strings } X} \max_{1 \le i \le k} \text{Edit}(X, A_i)$$

   A string X that achieves this minimum is called an *edit center* of $\mathbf{A}$. A set of strings may have several edit centers, but the edit radius is unique.

   Describe an efficient algorithm to compute the edit radius of three given strings.

4. Given 5 sequences of numbers, each of length n, design and analyze an efficent algorithm to compute the longest common subsequence among all 5 sequences.

5. Suppose we want to display a paragraph of text on a computer screen. The text consists of n words, where the ith word is $W[i]$ pixels wide. We want to break the paragraph into several lines, each exactly L pixels long. Depending on which words we put on each line, we will need to insert different amounts of white space between the words. The paragraph should be fully justified, meaning that the first word on each line starts at its leftmost pixel, and *except for the last line*, the last character on each line ends at its rightmost pixel. (Look at the paragraph you are reading right now!) There must be at least one pixel of white space between any two words on the same line. Thus, if a line contains words i through j, then the amount of *extra* white space on that line is $L - j + i - \sum_{k=i}^{j} W[k]$.

   Define the *slop* of a paragraph layout as the sum, over all lines *except the last*, of the *cube* of the extra white space in each line. Describe an efficient algorithm to layout the paragraph with minimum slop, given the list $W[1..n]$ of word widths as input. You can assume that $W[i] < L/2$ for each i, so that each line contains at least two words.

6. A *partition* of a positive integer n is a multiset of positive integers that sum to n. Traditionally, the elements of a partition are written in non-decreasing order, separated by + signs. For example, the integer 7 has exactly twelve partitions:

$$\begin{array}{ccc} 1+1+1+1+1+1+1 & 3+1+1+1+1 & 4+1+1+1 \\ 2+1+1+1+1+1 & 3+2+1+1 & 4+2+1 \\ 2+2+1+1+1 & 3+2+2 & 4+3 \\ 2+2+2+1 & 3+3+1 & 7 \end{array}$$

The *roughness* of a partition $a_1 + a_2 + \cdots + a_k$ is defined as follows:

$$\rho(a_1 + a_2 + \cdots + a_k) = \sum_{i=1}^{k-1} |a_{i+1} - a_i - 1| \; + \; a_k - 1$$

A *smoothest* partition of n is the partition of n with minimum roughness. Intuitively, the smoothest partition is the one closest to a descending arithmetic series $k + \cdots + 3 + 2 + 1$, which is the only partition that has roughness 0. For example, the smoothest partitions of 7 are $4 + 2 + 1$ and $3 + 2 + 1 + 1$:

$$\begin{array}{ccc} \rho(1+1+1+1+1+1+1) = 6 & \rho(3+1+1+1+1) = 4 & \rho(4+1+1+1) = 4 \\ \rho(2+1+1+1+1+1) = 4 & \rho(3+2+1+1) = 1 & \rho(4+2+1) = 1 \\ \rho(2+2+1+1+1) = 3 & \rho(3+2+2) = 2 & \rho(4+3) = 2 \\ \rho(2+2+2+1) = 2 & \rho(3+3+1) = 2 & \rho(7) = 7 \end{array}$$

Describe and analyze an algorithm to compute, given a positive integer n, a smoothest partition of n.

CS 473G: Combinatorial Algorithms, Fall 2005

Homework 3

Due Tuesday, October 18, 2005, at midnight

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your homework.

---

1. Consider the following greedy approximation algorithm to find a vertex cover in a graph:

```
GreedyVertexCover(G):
  C ← ∅
  while G has at least one edge
      v ← vertex in G with maximum degree
      G ← G \ v
      C ← C ∪ v
  return C
```

   In class we proved that the approximation ratio of this algorithm is $O(\log n)$; your task is to prove a matching lower bound. Specifically, prove that for any integer n, there is a graph G with n vertices such that GreedyVertexCover(G) returns a vertex cover that is $\Omega(\log n)$ times larger than optimal.

2. Prove that for *any* constant k and *any* graph coloring algorithm A, there is a graph G such that $\boldsymbol{A(G) > OPT(G) + k}$, where $A(G)$ is the number of colors generated by algorithm A for graph G, and $OPT(G)$ is the optimal number of colors for G.

   [Note: This does not contradict the possibility of a constant ***factor*** *approximation algorithm.]*

3. Let R be a set of rectangles in the plane, with horizontal and vertical edges. A *stabbing set* for R is a set of points S such that every rectangle in R contains at least one point in S. The *rectangle stabbing* problem asks, given a set R of rectangles, for the smallest stabbing set S.

   (a) Prove that the rectangle stabbing problem is NP-hard.

   (b) Describe and analyze an efficient approximation algorithm for the rectangle stabbing problem. Give bounds on the approximation ratio of your algorithm.

4. Consider the following approximation scheme for coloring a graph G.

```
TreeColor(G):
  T ← any spanning tree of G
  Color the tree T with two colors
  c ← 2

  for each edge (u, v) ∈ G \ T
      T ← T ∪ {(u, v)}

      if color(u) = color(v)    ⟨⟨Try recoloring u with an existing color⟩⟩
          for i ← 1 to c
              if no neighbor of u in T has color i
                  color(u) ← i

      if color(u) = color(v)    ⟨⟨Try recoloring v with an existing color⟩⟩
          for i ← 1 to c
              if no neighbor of v in T has color i
                  color(v) ← i

      if color(u) = color(v)    ⟨⟨Give up and use a new color⟩⟩
          c ← c + 1
          color(u) ← c

  return c
```

   (a) Prove that this algorithm correctly colors any bipartite graph.

   (b) Prove an upper bound C on the number of colors used by this algorithm. Give a sample graph and run that requires C colors.

   (c) Does this algorithm approximate the minimum number of colors up to a constant factor? In other words, is there a constant α such that $\textsc{TreeColor}(G) < \alpha \cdot OPT(G)$ for any graph G? Justify your answer.

5. In the *bin packing* problem, we are given a set of n items, each with weight between 0 and 1, and we are asked to load the items into as few bins as possible, such that the total weight in each bin is at most 1. It's not hard to show that this problem is NP-Hard; this question asks you to analyze a few common approximation algorithms. In each case, the input is an array $W[1..n]$ of weights, and the output is the number of bins used.

```
NextFit(W[1..n]):
  b ← 0
  Total[0] ← ∞

  for i ← 1 to n
      if Total[b] + W[i] > 1
          b ← b + 1
          Total[b] ← W[i]
      else
          Total[b] ← Total[b] + W[i]

  return b
```

```
FirstFit(W[1..n]):
  b ← 0

  for i ← 1 to n
      j ← 1; found ← False

      while j ≤ b and found = False
          if Total[j] + W[i] ≤ 1
              Total[j] ← Total[j] + W[i]
              found ← True
          j ← j + 1

      if found = False
          b ← b + 1
          Total[b] = W[i]

  return b
```

(a) Prove that NextFit uses at most twice the optimal number of bins.

(b) Prove that FirstFit uses at most twice the optimal number of bins.

(c) Prove that if the weight array W is initially sorted in decreasing order, then FirstFit uses at most $(4 \cdot OPT + 1)/3$ bins, where OPT is the optimal number of bins. The following facts may be useful (but you need to prove them if your proof uses them):

- In the packing computed by FirstFit, every item with weight more than $1/3$ is placed in one of the first OPT bins.
- FirstFit places at most $OPT - 1$ items outside the first OPT bins.

CS 473G: Combinatorial Algorithms, Fall 2005

Homework 4

Due Thursday, October 27, 2005, at midnight

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

Homeworks may be done in teams of up to three people. Each team turns in just one solution; every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your solution to problem 1.

If you are an I2CS student, print "(I2CS)" next to your name. Teams that include both on-campus and I2CS students can have up to four members. Any team containing both on-campus and I2CS students automatically receives 3 points of extra credit.

For the rest of the semester, unless specifically stated otherwise, you may assume that the function RANDOM(m) returns an integer chosen uniformly at random from the set $\{1, 2, , \ldots, m\}$ in $O(1)$ time. For example, a fair coin flip is obtained by calling RANDOM(2).

1. Consider the following randomized algorithm for choosing the largest bolt. Draw a bolt uniformly at random from the set of n bolts, and draw a nut uniformly at random from the set of n nuts. If the bolt is smaller than the nut, discard the bolt, draw a new bolt uniformly at random from the unchosen bolts, and repeat. Otherwise, discard the nut, draw a new nut uniformly at random from the unchosen nuts, and repeat. Stop either when every nut has been discarded, or every bolt except the one in your hand has been discarded.

   What is the *exact* expected number of nut-bolt tests performed by this algorithm? Prove your answer is correct. *[Hint: What is the expected number of unchosen nuts and bolts when the algorithm terminates?]*

2. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

   - MAKEQUEUE: Return a new priority queue containing the empty set.
   - FINDMIN(Q): Return the smallest element of Q (if any).
   - DELETEMIN(Q): Remove the smallest element in Q (if any).
   - INSERT(Q, x): Insert element x into Q, if it is not already there.
   - DECREASEKEY(Q, x, y): Replace an element $x \in Q$ with a smaller key y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q containing x.
   - DELETE(Q, x): Delete the element $x \in Q$. The input is a pointer directly to the node in Q containing x.
   - MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2; this operation destroys Q_1 and Q_2.

   A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```
MELD(Q1, Q2):
  if Q1 is empty return Q2
  if Q2 is empty return Q1

  if key(Q1) > key(Q2)
      swap Q1 ↔ Q2

  with probability 1/2
      left(Q1) ← MELD(left(Q1), Q2)
  else
      right(Q1) ← MELD(right(Q1), Q2)

  return Q1
```

   (a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (not just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. *[Hint: How long is a random root-to-leaf path in an n-node binary tree if each left/right choice is made with equal probability?]*

   (b) Prove that MELD(Q_1, Q_2) runs in $O(\log n)$ time with high probability.

   (c) Show that each of the other meldable priority queue operations cab be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)

3. Let $M[1..n][1..n]$ be an $n \times n$ matrix in which every row and every column is sorted. Such an array is called *totally monotone*. No two elements of M are equal.

   (a) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, compute the number of elements of M smaller than $M[i][j]$ and larger than $M[i'][j']$.

   (b) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, return an element of M chosen uniformly at random from the elements smaller than $M[i][j]$ and larger than $M[i'][j']$. Assume the requested range is always non-empty.

   (c) Describe and analyze a randomized algorithm to compute the median element of M in $O(n \log n)$ expected time.

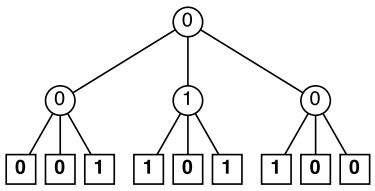
4. Let $X[1..n]$ be an array of n distinct real numbers, and let $N[1..n]$ be an array of indices with the following property: If $X[i]$ is the largest element of X, then $X[N[i]]$ is the smallest element of X; otherwise, $X[N[i]]$ is the smallest element of X that is larger than $X[i]$.

   For example:

i	1	2	3	4	5	6	7	8	9
$X[i]$	83	54	16	31	45	99	78	62	27
$N[i]$	6	8	9	5	2	3	1	7	4

   Describe and analyze a randomized algorithm that determines whether a given number x appears in the array X in $O(\sqrt{n})$ expected time. **Your algorithm may not modify the arrays X and $Next$.**

5. A *majority tree* is a complete ternary tree with depth n, where every leaf is labeled either 0 or 1. The *value* of a leaf is its label; the *value* of any internal node is the majority of the values of its three children. Consider the problem of computing the value of the root of a majority tree, given the sequence of 3^n leaf labels as input. For example, if $n = 2$ and the leaves are labeled $0, 0, 1, 1, 0, 1, 1, 1, 0, 0$, the root has value 0.



A majority tree with depth $n = 2$.

   (a) Prove that *any* deterministic algorithm that computes the value of the root of a majority tree *must* examine every leaf. *[Hint: Consider the special case $n = 1$. Recurse.]*

   (b) Describe and analyze a randomized algorithm that computes the value of the root in worst-case expected time $O(c^n)$ for some constant $c < 3$. *[Hint: Consider the special case $n = 1$. Recurse.]*

*6. *[Extra credit]* In the usual theoretical presentation of treaps, the priorities are random real numbers chosen uniformly from the interval $[0, 1]$, but in practice, computers only have access to random *bits*. This problem asks you to analyze a modification of treaps that takes this limitation into account.

Suppose the priority of a node v is abstractly represented as an infinite sequence $\pi_v[1..\infty]$ of random bits, which is interpreted as the rational number

$$\mathit{priority}(v) = \sum_{i=1}^{\infty} \pi_v[i] \cdot 2^{-i}.$$

However, only a finite number ℓ_v of these bits are actually known at any given time. When a node v is first created, *none* of the priority bits are known: $\ell_v = 0$. We generate (or 'reveal') new random bits only when they are necessary to compare priorities. The following algorithm compares the priorities of any two nodes in $O(1)$ expected time:

```
LargerPriority(v, w):
  for i ← 1 to ∞
      if i > ℓ_v
          ℓ_v ← i;  π_v[i] ← RandomBit
      if i > ℓ_w
          ℓ_w ← i;  π_w[i] ← RandomBit
      if π_v[i] > π_w[i]
          return v
      else if π_v[i] < π_w[i]
          return w
```

Suppose we insert n items one at a time into an initially empty treap. Let $L = \sum_v \ell_v$ denote the total number of random bits generated by calls to LargerPriority during these insertions.

(a) Prove that $E[L] = \Theta(n)$.

(b) Prove that $E[\ell_v] = \Theta(1)$ for any node v. *[Hint: This is equivalent to part (a). Why?]*

(c) Prove that $E[\ell_{\text{root}}] = \Theta(\log n)$. *[Hint: Why doesn't this contradict part (b)?]*

CS 473G: Combinatorial Algorithms, Fall 2005

Homework 5

Due Thursday, November 17, 2005, at midnight

(because you *really* don't want homework due over Thanksgiving break)

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

Homeworks may be done in teams of up to three people. Each team turns in just one solution; every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Attach this sheet (or the equivalent information) to the top of your solution to problem 1.

If you are an I2CS student, print "(I2CS)" next to your name. Teams that include both on-campus and I2CS students can have up to four members. Any team containing both on-campus and I2CS students automatically receives 3 points of extra credit.

Problems labeled ♈ are likely to require techniques from next week's lectures on cuts, flows, and matchings. See also Chapter 7 in Kleinberg and Tardos, or Chapter 26 in CLRS.

♈ 1. Suppose you are asked to construct the minimum spanning tree of a graph G, but you are not completely sure of the edge weights. Specifically, you have a *conjectured* weight $\tilde{w}(e)$ for every edge e in the graph, but you also know that up to k of these conjectured weights are wrong. With the exception of one edge e whose true weight you know exactly, you don't know which edges are wrong, or even how they're wrong; the true weights of those edges could be larger or smaller than the conjectured weights. Given this unreliable information, it is of course impossible to reliably construct the true minimum spanning tree of G, but it is still possible to say something about your special edge.

   Describe and analyze an efficient algorithm to determine whether a specific edge e, whose *actual* weight is known, is *definitely not* in the minimum spanning tree of G under the stated conditions. The input consists of the graph G, the conjectured weight function $\tilde{w} : E(G) \to \mathbb{R}$, the positive integer k, and the edge e.

2. Most classical minimum-spanning-tree algorithms use the notions of 'safe' and 'useless' edges described in the lecture notes, but there is an alternate formulation. Let G be a weighted undirected graph, where the edge weights are distinct. We say that an edge e is *dangerous* if it is the longest edge in some cycle in G, and *useful* if it does not lie in any cycle in G.

   (a) Prove that the minimum spanning tree of G contains every useful edge.

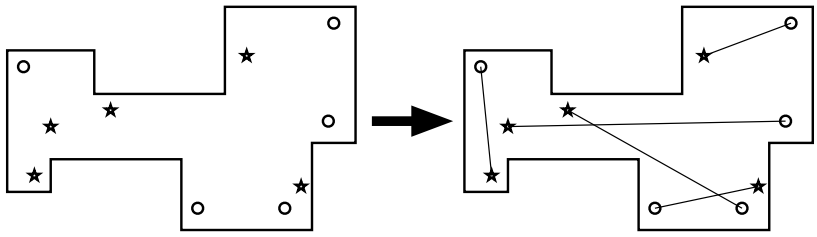
   (b) Prove that the minimum spanning tree of G does not contain any dangerous edge.

   (c) Describe and analyze an efficient implementation of the "anti-Kruskal" MST algorithm: Examine the edges of G in *decreasing* order; if an edge is dangerous, remove it from G. *[Hint: It won't be as fast as the algorithms you saw in class.]*

♈ 3. The UIUC Computer Science department has decided to build a mini-golf course in the basement of the Siebel Center! The playing field is a closed polygon bounded by m horizontal and vertical line segments, meeting at right angles. The course has n *starting points* and n *holes*, in one-to-one correspondence. It is always possible hit the ball along a straight line directly from each starting point to the corresponding hole, without touching the boundary of the playing field. (Players are not allowed to bounce golf balls off the walls; too much glass.) The n starting points and n holes are all at distinct locations.

   Sadly, the architect's computer crashed just as construction was about to begin. Thanks to the herculean efforts of their sysadmins, they were able to recover the *locations* of the starting points and the holes, but all information about which starting points correspond to which holes was lost!

   Describe and analyze an algorithm to compute a one-to-one correspondence between the starting points and the holes that meets the straight-line requirement, or to report that no such correspondence exists. The input consists of the x- and y-coordinates of the m corners of the playing field, the n starting points, and the n holes. Assume you can determine in constant time whether two line segments intersect, given the x- and y-coordinates of their endpoints.



A minigolf course with five starting points ($\star$) and five holes ($\circ$), and a legal correspondence between them.

♈ 4. Let $G = (V, E)$ be a directed graph where the in-degree of each vertex is equal to its out-degree. Prove or disprove the following claim: For any two vertices u and v in G, the number of mutually edge-disjoint paths from u to v is equal to the number of mutually edge-disjoint paths from v to u.

5. You are given a set of n boxes, each specified by its height, width, and depth. The order of the dimensions is unimportant; for example, a $1 \times 2 \times 3$ box is exactly the same as a $3 \times 1 \times 2$ box of a $2 \times 1 \times 3$ box. You can nest box A inside box B if and only if A can be rotated so that it has strictly smaller height, strictly smaller width, and strictly smaller depth than B.

   (a) Design and analyze an efficient algorithm to determine the largest sequence of boxes that can be nested inside one another. *[Hint: Model the nesting relationship as a graph.]*

   (b) Describe and analyze an efficient algorithm to nest all n boxes into as few groups as possible, where each group consists of a nested sequence. You are not allowed to put two boxes side-by-side inside a third box, even if they are small enough to fit.[1] *[Hint: Model the nesting relationship as a* ***different*** *graph.]*

6. *[Extra credit]* Prove that Ford's generic shortest-path algorithm (described in the lecture notes) can take exponential time in the worst case when implemented with a stack instead of a heap (like Dijkstra) or a queue (like Bellman-Ford). Specifically, construct for every positive integer n a weighted directed n-vertex graph G_n, such that the stack-based shortest-path algorithm call RELAX $\Omega(2^n)$ times when G_n is the input graph. *[Hint: Towers of Hanoi.]*

[1] Without this restriction, the problem is NP-hard, even for one-dimensional "boxes".

CS 473G: Combinatorial Algorithms, Fall 2005
Homework 6

Practice only; nothing to turn in.

---

1. A small airline, Ivy Air, flies between three cities: Ithaca (a small town in upstate New York), Newark (an eyesore in beautiful New Jersey), and Boston (a yuppie town in Massachusetts). They offer several flights but, for this problem, let us focus on the Friday afternoon flight that departs from Ithaca, stops in Newark, and continues to Boston. There are three types of passengers:

   (a) Those traveling from Ithaca to Newark (god only knows why).

   (b) Those traveling from Newark to Boston (a very good idea).

   (c) Those traveling from Ithaca to Boston (it depends on who you know).

   The aircraft is a small commuter plane that seats 30 passengers. The airline offers three fare classes:

   (a) Y class: full coach.

   (b) B class: nonrefundable.

   (c) M class: nonrefundable, 3-week advanced purchase.

   Ticket prices, which are largely determined by external influences (i.e., competitors), have been set and advertised as follows:

	Ithaca-Newark	Newark-Boston	Ithaca-Boston
Y	300	160	360
B	220	130	280
M	100	80	140

   Based on past experience, demand forecasters at Ivy Air have determined the following upper bounds on the number of potential customers in each of the 9 possible origin-destination/fare-class combinations:

	Ithaca-Newark	Newark-Boston	Ithaca-Boston
Y	4	8	3
B	8	13	10
M	22	20	18

   The goal is to decide how many tickets from each of the 9 origin/destination/fare-class combinations to sell. The constraints are that the place cannot be overbooked on either the two legs of the flight and that the number of tickets made available cannot exceed the forecasted maximum demand. The objective is to maximize the revenue.

   Formulate this problem as a linear programming problem.

2. (a) Suppose we are given a directed graph $G = (V, E)$, a length function $\ell : E \to \mathbb{R}$, and a source vertex $s \in V$. Write a linear program to compute the shortest-path distance from s to every other vertex in V. *[Hint: Define a variable for each vertex representing its distance from s. What objective function should you use?]*

   (b) In the *minimum-cost multicommodity-flow* problem, we are given a directed graph $G = (V, E)$, in which each edge $u \to v$ has an associated nonnegative *capacity* $c(u \to v) \geq 0$ and an associated *cost* $\alpha(u \to v)$. We are given k different commodities, each specified by a triple $K_i = (s_i, t_i, d_i)$, where s_i is the source node of the commodity, t_i is the target node for the commodity i, and d_i is the *demand*: the desired flow of commodity i from s_i to t_i. A *flow* for commodity i is a non-negative function $f_i : E \to \mathbb{R}_{\geq 0}$ such that the total flow into any vertex other than s_i or t_i is equal to the total flow out of that vertex. The *aggregate flow* $F : E \to \mathbb{R}$ is defined as the sum of these individual flows: $F(u \to v) = \sum_{i=1}^{k} f_i(u \to v)$. The aggregate flow $F(u \to v)$ on any edge must not exceed the capacity $c(u \to v)$. The goal is to find an aggregate flow whose total *cost* $\sum_{u \to v} F(u \to v) \cdot \alpha(u \to v)$ is as small as possible. (Costs may be negative!) Express this problem as a linear program.

3. In class we described the duality transformation only for linear programs in canonical form:

Primal (Π)		**Dual (Π)**
$\max \quad c \cdot x$	$\Longleftrightarrow$	$\min \quad y \cdot b$
$\text{s.t. } Ax \leq b$		$\text{s.t. } yA \geq c$
$x \geq 0$		$y \geq 0$

   Describe precisely how to dualize the following more general linear programming problem:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{d} c_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{d} a_{ij} x_j \leq b_i \quad \text{for each } i = 1 \mathinner{..} p \\
& \sum_{j=1}^{d} a_{ij} x_j = b_i \quad \text{for each } i = p+1 \mathinner{..} p+q \\
& \sum_{j=1}^{d} a_{ij} x_j \geq b_i \quad \text{for each } i = p+q+1 \mathinner{..} n
\end{aligned}$$

   Your dual problem should have one variable for each primal constraint, and the dual of your dual program should be precisely the original linear program.

4. (a) Model the maximum-cardinality bipartite matching problem as a linear programming problem. The input is a bipartite graph $G = (U, V; E)$, where $E \subseteq U \times V$; the output is the largest matching in G. Your linear program should have one variable for every edge.

   (b) Now dualize the linear program from part (a). What do the dual variables represent? What does the objective function represent? What problem is this!?

5. An *integer program* is a linear program with the additional constraint that the variables must take only integer values.

   (a) Prove that deciding whether an integer program has a feasible solution is NP-complete.

   (b) Prove that finding the optimal feasible solution to an integer program is NP-hard.

   [Hint: Almost any NP=hard decision problem can be rephrased as an integer program. Pick your favorite.]

6. Consider the LP formulation of the shortest path problem presented in class:

$$\begin{aligned} \text{maximize} \quad & d_t \\ \text{subject to} \quad & d_s = 0 \\ & d_v - d_u \le \ell_{u\to v} \quad \text{for every edge } u \to v \end{aligned}$$

   Characterize the feasible bases for this linear program in terms of the original weighted graph. What does a simplex pivoting operation represent? What is a locally optimal (*i.e.*, dual feasible) basis? What does a dual pivoting operation represent?

7. Consider the LP formulation of the maximum-flow problem presented in class:

$$\begin{aligned} \text{maximize} \quad & \sum_w f_{s\to w} - \sum_u f_{u\to s} \\ \text{subject to} \quad & \sum_w f_{v\to w} - \sum_u f_{u\to v} = 0 \qquad \text{for every vertex } v \neq s,t \\ & f_{u\to v} \le c_{u\to v} \quad \text{for every edge } u \to v \\ & f_{u\to v} \ge 0 \qquad \text{for every edge } u \to v \end{aligned}$$

   Is the Ford-Fulkerson augmenting path algorithm an instance of the simplex algorithm applied to this linear program? Why or why not?

*8. *Helly's theorem* says that for any collection of convex bodies in $\mathbb{R}^n$, if every $n+1$ of them intersect, then there is a point lying in the intersection of all of them. Prove Helly's theorem for the special case that the convex bodies are halfspaces. *[Hint: Show that if a system of inequalities $Ax \ge b$ does not have a solution, then we can select $n+1$ of the inequalities such that the resulting system does not have a solution. Construct a primal LP from the system by choosing a 0 cost vector.]*

You have 90 minutes to answer four of these questions.

Write your answers in the separate answer booklet.

You may take the question sheet with you when you leave.

1. You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, the cards are dealt face up in a long row. Each card is worth a different number of points. After all the cards are dealt, you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, you can decide which of the two cards to take. The winner of the game is the player that has collected the most points when the game ends.

   Having never taken an algorithms class, Elmo follows the obvious greedy strategy—when it's his turn, Elmo *always* takes the card with the higher point value. Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely *hates* it when grown-ups let him win.)

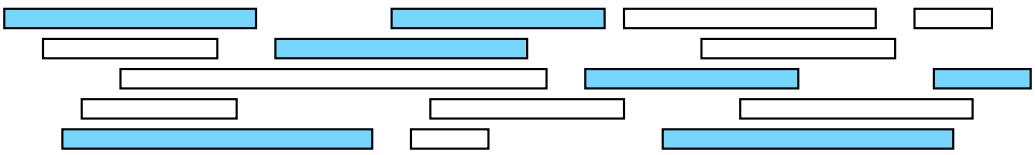
   (a) Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do *not* follow the same greedy strategy as Elmo.

   (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.

2. Suppose you are given a magical black box that can tell you in constant time whether or not a given graph has a Hamiltonian cycle. Using this magic black box as a subroutine, describe and analyze a *polynomial-time* algorithm to actually compute a Hamiltonian cycle in a given graph, if one exists.

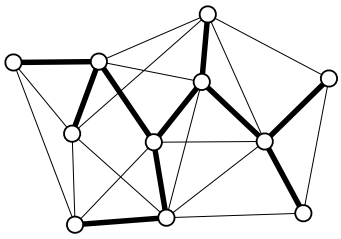
3. Let X be a set of n intervals on the real line. A subset of intervals $Y \subseteq X$ is called a *tiling path* if the intervals in Y cover the intervals in X, that is, any real value that is contained in some interval in X is also contained in some interval in Y. The *size* of a tiling cover is just the number of intervals.

   Describe and analyze an algorithm to compute the smallest tiling path of X as quickly as possible. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X.



A set of intervals. The seven shaded intervals form a tiling path.

4. Prove that the following problem is NP-complete: Given an undirected graph, does it have a spanning tree in which every node has degree at most 3?

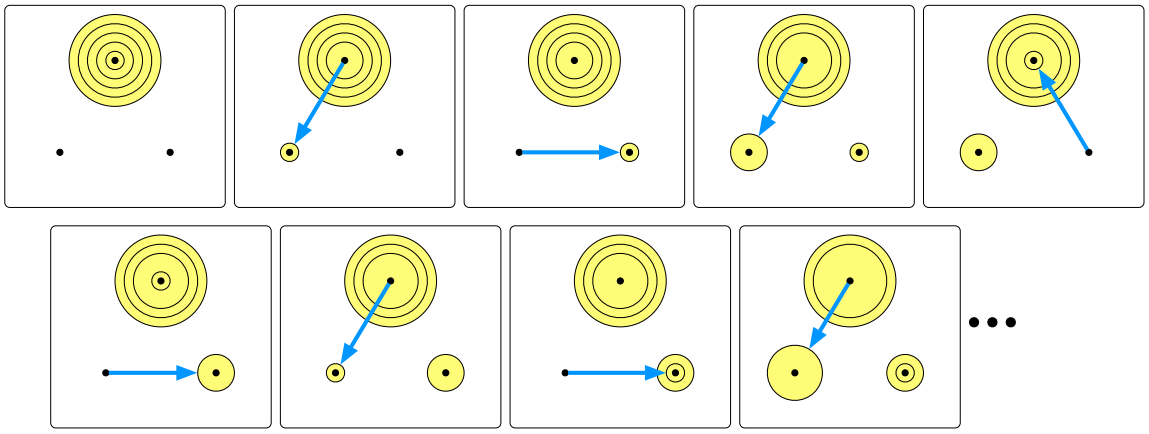


A graph with a spanning tree of maximum degree 3.

5. The *Tower of Hanoi* puzzle, invented by Edouard Lucas in 1883, consists of three pegs and n disks of different sizes. Initially, all n disks are on the same peg, stacked in order by size, with the largest disk on the bottom and the smallest disk on top. In a single move, you can move the topmost disk on any peg to another peg; however, you are never allowed to place a larger disk on top of a smaller one. Your goal is to move all n disks to a different peg.

   (a) Prove that the Tower of Hanoi puzzle can be solved in exactly $2^n - 1$ moves. *[Hint: You've probably seen this before.]*

   (b) Now suppose the pegs are arranged in a circle and you are *only* allowed to move disks *counterclockwise*. How many moves do you need to solve this restricted version of the puzzle? Give a upper bound in the form $O(f(n))$ for some function $f(n)$. Prove your upper bound is correct.



A top view of the first eight moves in a counterclockwise Towers of Hanoi solution

You have 90 minutes to answer four of these questions.

Write your answers in the separate answer booklet.

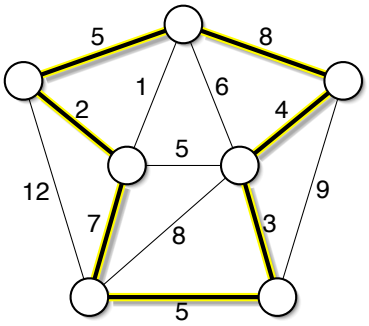
You may take the question sheet with you when you leave.

Chernoff Bounds: If X is the sum of independent indicator variables and $\mu = \mathrm{E}[X]$, then the following inequalities hold for any $\delta > 0$:

$$\Pr[X < (1-\delta)\mu] < \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu} \qquad \Pr[X > (1+\delta)\mu] < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

1. Describe and analyze an algorithm that randomly shuffles an array $X[1..n]$, so that each of the $n!$ possible permutations is equally likely, in $O(n)$ time. (Assume that the subroutine RANDOM(m) returns an integer chosen uniformly at random from the set $\{1, 2, , \ldots, m\}$ in $O(1)$ time.)

2. Let G be an undirected graph with weighted edges. A *heavy Hamiltonian cycle* is a cycle C that passes through each vertex of G exactly once, such that the total weight of the edges in C is at least half of the total weight of all edges in G. Prove that deciding whether a graph has a heavy Hamiltonian cycle is NP-complete.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

3. A sequence of numbers $\langle a_1, a_2, a_3, \ldots a_n \rangle$ is *oscillating* if $a_i < a_{i+1}$ for every *odd* index i and $a_i > a_{i+1}$ for every *even* index i. Describe and analyze an efficient algorithm to compute the longest oscillating subsequence in a sequence of n integers.

4. This problem asks you to how to efficiently modify a maximum flow if one of the edge capacities changes. Specifically, you are given a directed graph $G = (V, E)$ with capacities $c : E \to \mathbb{Z}_+$, and a maximum flow $F : E \to \mathbb{Z}$ from some vertex s to some other vertex t in G. Describe and analyze efficient algorithms for the following operations:

   (a) INCREMENT(e) — Increase the capacity of edge e by 1 and update the maximum flow F.

   (b) DECREMENT(e) — Decrease the capacity of edge e by 1 and update the maximum flow F.

   Both of your algorithms should be significantly faster than recomputing the maximum flow from scratch.

5.

6. Let $G = (V, E)$ be an undirected graph, each of whose vertices is colored either red, green, or blue. An edge in G is *boring* if its endpoints have the same color, and *interesting* if its endpoints have different colors. The *most interesting 3-coloring* is the 3-coloring with the maximum number of interesting edges, or equivalently, with the fewest boring edges.

   (a) Prove that it is NP-hard to compute the most interesting 3-coloring of a graph. *[Hint: There is a one-line proof. Use one of the NP-hard problems described in class.]*

   (b) Let $zzz(G)$ denote the number of boring edges in the most interesting 3-coloring of a graph G. Prove that it is NP-hard to approximate $zzz(G)$ within a factor of $10^{10^{100}}$. *[Hint: There is a one-line proof.]*

   (c) Let $wow(G)$ denote the number of interesting edges in the most interesting 3-coloring of G. Suppose we assign each vertex in G a *random* color from the set $\{\text{red}, \text{green}, \text{blue}\}$. Prove that the expected number of interesting edges is at least $\frac{2}{3}wow(G)$.

7.

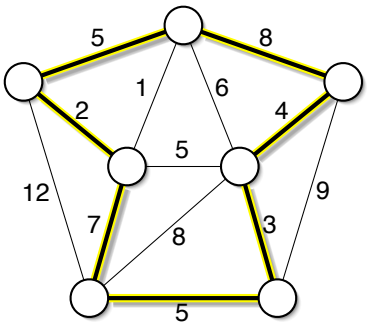
You have 180 minutes to answer six of these questions.

Write your answers in the separate answer booklet.

You may take the question sheet with you when you leave.

1. Describe and analyze an algorithm that randomly shuffles an array $X[1..n]$, so that each of the $n!$ possible permutations is equally likely, in $O(n)$ time. (Assume that the subroutine $\textsc{Random}(m)$ returns an integer chosen uniformly at random from the set $\{1, 2, , \ldots, m\}$ in $O(1)$ time.)

2. Let G be an undirected graph with weighted edges. A *heavy Hamiltonian cycle* is a cycle C that passes through each vertex of G exactly once, such that the total weight of the edges in C is at least half of the total weight of all edges in G. Prove that deciding whether a graph has a heavy Hamiltonian cycle is NP-complete.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

3. Suppose you are given a directed graph $G = (V, E)$ with capacities $c : E \to \mathbb{Z}_+$ *and* a maximum flow $F : E \to \mathbb{Z}$ from some vertex s to some other vertex t in G. Describe and analyze efficient algorithms for the following operations:

   (a) $\textsc{Increment}(e)$ — Increase the capacity of edge e by 1 and update the maximum flow F.

   (b) $\textsc{Decrement}(e)$ — Decrease the capacity of edge e by 1 and update the maximum flow F.

   Both of your algorithms should be significantly faster than recomputing the maximum flow from scratch.

4. Suppose you are given an undirected graph G and two vertices s and t in G. Two paths from s to t are *vertex-disjoint* if the only vertices they have in common are s and t. Describe and analyze an efficient algorithm to compute the maximum number of vertex-disjoint paths between s and t in G. *[Hint: Reduce this to a more familiar problem on a suitable directed graph G'.]*

5. A sequence of numbers $\langle a_1, a_2, a_3, \ldots a_n \rangle$ is *oscillating* if $a_i < a_{i+1}$ for every *odd* index i and $a_i > a_{i+1}$ for every *even* index i. For example, the sequence $\langle 2, 7, 1, 8, 2, 8, 1, 8, 3 \rangle$ is oscillating. Describe and analyze an efficient algorithm to compute the longest oscillating subsequence in a sequence of n integers.

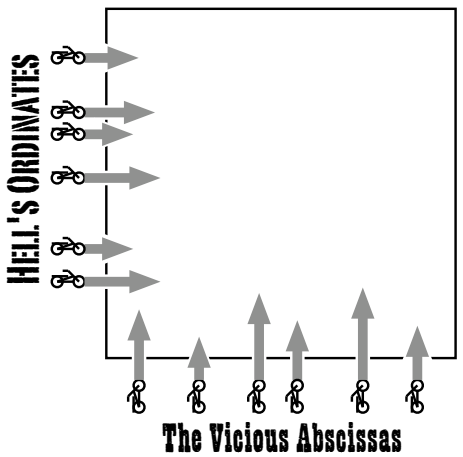
6. Let $G = (V, E)$ be an undirected graph, each of whose vertices is colored either red, green, or blue. An edge in G is *boring* if its endpoints have the same color, and *interesting* if its endpoints have different colors. The *most interesting 3-coloring* is the 3-coloring with the maximum number of interesting edges, or equivalently, with the fewest boring edges. Computing the most interesting 3-coloring is NP-hard, because the standard 3-coloring problem we saw in class is a special case.

   (a) Let $zzz(G)$ denote the number of boring edges in the most interesting 3-coloring of a graph G. Prove that it is NP-hard to approximate $zzz(G)$ within a factor of $10^{10^{100}}$.

   (b) Let $wow(G)$ denote the number of interesting edges in the most interesting 3-coloring of G. Suppose we assign each vertex in G a *random* color from the set $\{\text{red}, \text{green}, \text{blue}\}$. Prove that the expected number of interesting edges is at least $\frac{2}{3} wow(G)$.

7. It's time for the 3rd Quasi-Annual Champaign-Urbana Ice Motorcycle Demolition Derby Race-O-Rama and Spaghetti Bake-Off! The main event is a competition between two teams of n motorcycles in a huge square ice-covered arena. All of the motorcycles have spiked tires so that they can ride on the ice. Each motorcycle drags a long metal chain behind it. Whenever a motorcycle runs over a chain, the chain gets caught in the tire spikes, and the motorcycle crashes. Two motorcycles can also crash by running directly into each other. All the motorcycle start simultaneously. Each motorcycle travels in a straight line at a constant speed until it either crashes or reaches the opposite wall—no turning, no braking, no speeding up, no slowing down. The Vicious Abscissas start at the south wall of the arena and ride directly north (vertically). Hell's Ordinates start at the west wall of the arena and ride directly east (horizontally). If any motorcycle completely crosses the arena, that rider's entire team wins the competition.

   Describe and analyze an efficient algorithm to decide which team will win, given the starting position and speed of each motorcycle.



CS 473U: Undergraduate Algorithms, Fall 2006
Homework 0

Due Friday, September 1, 2006 at noon in 3229 Siebel Center

Name:	
Net ID:	Alias:

☐ I understand the Homework Instructions and FAQ.

- Neatly print your full name, your NetID, and an alias of your choice in the boxes above, and submit this page with your solutions. We will list homework and exam grades on the course web site by alias. For privacy reasons, your alias should not resemble your name, your NetID, your university ID number, or (God forbid) your Social Security number. Please use the same alias for every homework and exam.

  Federal law forbids us from publishing your grades, even anonymously, without your explicit permission. **By providing an alias, you grant us permission to list your grades on the course web site; if you do not provide an alias, your grades will not be listed.**

- Please carefully read the Homework Instructions and FAQ on the course web page, and then check the box above. This page describes what we expect in your homework solutions—start each numbered problem on a new sheet of paper, write your name and NetID on every page, don't turn in source code, analyze and prove everything, use good English and good logic, and so on—as well as policies on grading standards, regrading, and plagiarism. **See especially the policies regarding the magic phrases "I don't know" and "and so on".** If you have *any* questions, post them to the course newsgroup or ask in lecture.

- This homework tests your familiarity with prerequisite material—basic data structures, big-Oh notation, recurrences, discrete probability, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Each numbered problem is worth 10 points; not all subproblems have equal weight.

#	1	2	3	4	5	6*	Total
Score							
Grader							

Please put your answers to problems 1 and 2 on the same page.

1. Sort the functions listed below from asymptotically smallest to asymptotically largest, indicating ties if there are any. **Do not turn in proofs**, but you should probably do them anyway, just for practice.

   To simplify your answers, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$, and write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions $n^2, n, \binom{n}{2}, n^3$ could be sorted either as $n \ll n^2 \equiv \binom{n}{2} \ll n^3$ or as $n \ll \binom{n}{2} \equiv n^2 \ll n^3$.

   $$\begin{array}{cccccccc} \lg n & \ln n & \sqrt{n} & n & n \lg n & n^2 & 2^n & n^{1/n} \\ n^{1+1/\lg n} & \lg^{1000} n & 2^{\sqrt{\lg n}} & (\sqrt{2})^{\lg n} & \lg^{\sqrt{2}} n & n^{\sqrt{2}} & (1+\tfrac{1}{n})^n & n^{1/1000} \\ H_n & H_{\sqrt{n}} & 2^{H_n} & H_{2^n} & F_n & F_{n/2} & \lg F_n & F_{\lg n} \end{array}$$

   In case you've forgotten:

   - $\lg n = \log_2 n \neq \ln n = \log_e n$
   - $\lg^3 n = (\lg n)^3 \neq \lg \lg \lg n$.
   - The harmonic numbers: $H_n = \sum_{i=1}^{n} 1/i \approx \ln n + 0.577215\ldots$
   - The Fibonacci numbers: $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$

2. Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Proofs are *not* required; just give us the list of answers. **Don't turn in proofs**, but you should do them anyway, just for practice. Assume reasonable but nontrivial base cases. **If your solution requires specific base cases, state them.** Extra credit will be awarded for more exact solutions.

   (a) $A(n) = 2A(n/4) + \sqrt{n}$

   (b) $B(n) = 3B(n/3) + n/\lg n$

   (c) $C(n) = \dfrac{2C(n-1)}{C(n-2)}$ *[Hint: This is easy!]*

   (d) $D(n) = D(n-1) + 1/n$

   (e) $E(n) = E(n/2) + D(n)$

   (f) $F(n) = 4F\left(\left\lceil \dfrac{n-8}{2} \right\rceil + \left\lfloor \dfrac{3n}{\log_\pi n} \right\rfloor\right) + 6\binom{n+5}{2} - 42n\lg^7 n + \sqrt{13n-6} + \dfrac{\lg\lg n + 1}{\lg n \lg\lg\lg n}$

   (g) $G(n) = 2G(n-1) - G(n-2) + n$

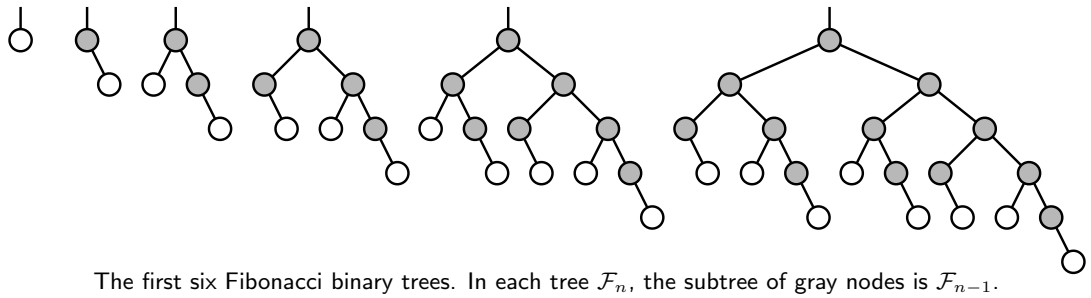
   (h) $H(n) = 2H(n/2) - 2H(n/4) + 2^n$

   (i) $I(n) = I(n/2) + I(n/4) + I(n/6) + I(n/12) + n$

   ★(j) $J(n) = \sqrt{n} \cdot J(2\sqrt{n}) + n$
   [Hint: First solve the secondary recurrence $j(n) = 1 + j(2\sqrt{n})$.]

3. The nth *Fibonacci binary tree* $\mathcal{F}_n$ is defined recursively as follows:

   - $\mathcal{F}_1$ is a single root node with no children.
   - For all $n \geq 2$, $\mathcal{F}_n$ is obtained from $\mathcal{F}_{n-1}$ by adding a right child to every leaf and adding a left child to every node that has only one child.



The first six Fibonacci binary trees. In each tree $\mathcal{F}_n$, the subtree of gray nodes is $\mathcal{F}_{n-1}$.

   (a) Prove that the number of leaves in $\mathcal{F}_n$ is precisely the nth Fibonacci number: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$.

   (b) How many nodes does $\mathcal{F}_n$ have? For full credit, give an *exact*, closed-form answer in terms of Fibonacci numbers, and prove your answer is correct.

   (c) Prove that the left subtree of $\mathcal{F}_n$ is a copy of $\mathcal{F}_{n-2}$.

4. Describe and analyze a data structure that stores set of n records, each with a numerical *key* and a numerical *priority*, such that the following operation can be performed quickly:

   RANGETOP(a, z) : return the highest-priority record whose key is between a and z.

   For example, if the (key, priority) pairs are

   $$(3,1),\ (4,9),\ (9,2),\ (6,3),\ (5,8),\ (7,5),\ (1,4),\ (0,7),$$

   then RANGETOP$(2, 8)$ would return the record with key 4 and priority 9 (the second record in the list).

   You may assume that no two records have equal keys or equal priorities, and that no record has a key equal to a or z. Analyze both the size of your data structure and the running time of your RANGETOP algorithm. For full credit, your data structure must be as small as possible *and* your RANGETOP algorithm must be as fast as possible.

   [Hint: How would you compute the number *of keys between a and z? How would you solve the problem if you knew that a is always $-\infty$?]*

5. Penn and Teller agree to play the following game. Penn shuffles a standard deck[1] of playing cards so that every permutation is equally likely. Then Teller draws cards from the deck, one at a time without replacement, until he draws the three of clubs (3♣), at which point the remaining undrawn cards instantly burst into flames.

   The first time Teller draws a card from the deck, he gives it to Penn. From then on, until the game ends, whenever Teller draws a card whose value is smaller than the last card he gave to Penn, he gives the new card to Penn.[2] To make the rules unambiguous, they agree beforehand that $A = 1$, $J = 11$, $Q = 12$, and $K = 13$.

   (a) What is the expected number of cards that Teller draws?

   (b) What is the expected *maximum* value among the cards Teller gives to Penn?

   (c) What is the expected *minimum* value among the cards Teller gives to Penn?

   (d) What is the expected number of cards that Teller gives to Penn?

   Full credit will be given only for *exact* answers (with correct proofs, of course).

*6. **[Extra credit]**[3]

   Lazy binary is a variant of standard binary notation for representing natural numbers where we allow each "bit" to take on one of three values: 0, 1, or 2. Lazy binary notation is defined inductively as follows.

   - The lazy binary representation of zero is 0.
   - Given the lazy binary representation of any non-negative integer n, we can construct the lazy binary representation of $n + 1$ as follows:
     (a) increment the rightmost digit;
     (b) if any digit is equal to 2, replace the rightmost 2 with 0 and increment the digit immediately to its left.

   Here are the first several natural numbers in lazy binary notation:

   0, 1, 10, 11, 20, 101, 110, 111, 120, 201, 210, 1011, 1020, 1101, 1110, 1111, 1120, 1201, 1210, 2011, 2020, 2101, 2110, 10111, 10120, 10201, 10210, 11011, 11020, 11101, 11110, 11111, 11120, 11201, 11210, 12011, 12020, 12101, 12110, 20111, 20120, 20201, 20210, 21011, 21020, 21101, 21110, 101111, 101120, 101201, 101210, 102011, 102020, 102101, 102110, ...

   (a) Prove that in any lazy binary number, between any two 2s there is at least one 0, and between two 0s there is at least one 2.

   (b) Prove that for any natural number N, the sum of the digits of the lazy binary representation of N is exactly $\lfloor \lg(N + 1) \rfloor$.

[1] In a standard deck of 52 cards, each card has a *suit* in the set $\{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\}$ and a *value* in the set $\{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\}$, and every possible suit-value pair appears in the deck exactly once. Actually, to make the game more interesting, Penn and Teller normally use razor-sharp ninja throwing cards.

[2] Specifically, he hurls them from the opposite side of the stage directly into the back of Penn's right hand.

[3] The "I don't know" rule does not apply to extra credit problems. There is no such thing as "partial extra credit".

CS 473U: Undergraduate Algorithms, Fall 2006
Homework 1

Due Tuesday, September 12, 2006 in 3229 Siebel Center

Starting with this homework, groups of up to three students can submit or present a single joint solution. If your group is submitting a written solution, please remember to **print the names, NetIDs, and aliases of *every* group member on *every* page**. Please remember to submit **separate, individually stapled** solutions to each of the problems.

---

1. Recall from lecture that a *subsequence* of a sequence A consists of a (not necessarily contiguous) collection of elements of A, arranged in the same order as they appear in A. If B is a subsequence of A, then A is a *supersequence* of B.

   (a) Describe and analyze a **simple** recursive algorithm to compute, given two sequences A and B, the length of the *longest common subsequence* of A and B. For example, given the strings ALGORITHM and ALTRUISTIC, your algorithm would return 5, the length of the longest common subsequence ALRIT.

   (b) Describe and analyze a **simple** recursive algorithm to compute, given two sequences A and B, the length of a *shortest common supersequence* of A and B. For example, given the strings ALGORITHM and ALTRUISTIC, your algorithm would return 14, the length of the shortest common supersequence ALGTORUISTHIMC.

   (c) Let $|A|$ denote the length of sequence A. For any two sequences A and B, let $\text{lcs}(A, B)$ denote the length of the longest common subsequence of A and B, and let $\text{scs}(A, B)$ denote the length of the shortest common supersequence of A and B.
   Prove that $\boldsymbol{|A| + |B| = \text{lcs}(A, B) + \text{scs}(A, B)}$ for all sequences A and B. *[Hint: There is a simple non-inductive proof.]*

   In parts (a) and (b), we are *not* looking for the most efficient algorithms, but for algorithms with simple and correct recursive structure.

2. You are a contestant on a game show, and it is your turn to compete in the following game. You are presented with an $m \times n$ grid of boxes, each containing a unique number. It costs \$100 to open a box. Your goal is to find a box whose number is larger than its neighbors in the grid (above, below, left, and right). If you spend less money than your opponents, you win a week-long trip for two to Las Vegas and a year's supply of Rice-A-Roni™, to which you are hopelessly addicted.

   (a) Suppose $m = 1$. Describe an algorithm that finds a number that is bigger than any of its neighbors. How many boxes does your algorithm open in the worst case?

   (b) Suppose $m = n$. Describe an algorithm that finds a number that is bigger than any of its neighbors. How many boxes does your algorithm open in the worst case?

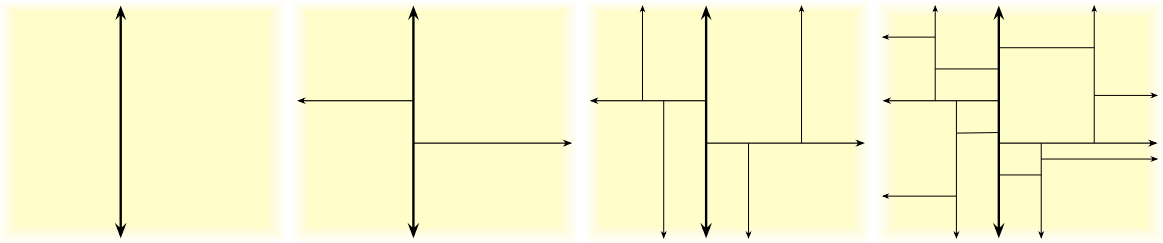
   *(c) **[Extra credit]**[1] Prove that your solution to part (b) is asymptotically optimal.

[1] The "I don't know" rule does not apply to extra credit problems. There is no such thing as "partial extra credit".

3. A kd-tree is a rooted binary tree with three types of nodes: horizontal, vertical, and leaf. Each vertical node has a *left* child and a *right* child; each horizontal node has a *high* child and a *low* child. The non-leaf node types alternate: non-leaf children of vertical nodes are horizontal and vice versa. Each non-leaf node v stores a real number p_v called its *pivot value*. Each node v has an associated *region* $R(v)$, defined recursively as follows:

   - $R(\textit{root})$ is the entire plane.
   - If v is is a horizontal node, the horizontal line $y = p_v$ partitions $R(v)$ into $R(\textit{high}(v))$ and $R(\textit{low}(v))$ in the obvious way.
   - If v is is a vertical node, the vertical line $x = p_v$ partitions $R(v)$ into $R(\textit{left}(v))$ and $R(\textit{right}(v))$ in the obvious way.

   Thus, each region $R(v)$ is an axis-aligned rectangle, possibly with one or more sides at infinity. If v is a leaf, we call $R(v)$ a *leaf box*.



The first four levels of a typical kd-tree.

   Suppose T is a perfectly balanced kd-tree with n leaves (and thus with depth exactly $\lg n$).

   (a) Consider the horizontal line $y = t$, where $t \neq p_v$ for all nodes v in T. *Exactly* how many leaf boxes of T does this line intersect? *[Hint: The parity of the root node matters.]* Prove your answer is correct. A correct $\Theta(\cdot)$ bound is worth significant partial credit.

   (b) Describe and analyze an efficient algorithm to compute, given T and an arbitrary horizontal line ℓ, the number of leaf boxes of T that lie *entirely above* ℓ.

CS 473U: Undergraduate Algorithms, Fall 2006
Homework 2

Due Tuesday, September 19, 2006 in 3229 Siebel Center
Remember to turn in in separate, individually stapled solutions to each of the problems.

---

1. You are given an $m \times n$ matrix M in which each entry is a 0 or 1. A *solid block* is a rectangular subset of M in which each entry is 1. Give a correct efficent algorithm to find a solid block in M with maximum area.

1	1	0	1	1
0	1	1	1	0
1	1	1	1	1
1	1	0	1	1

   An algorithm that runs in $\Theta(n^c)$ time will earn $19 - 3c$ points.

2. You are a bus driver with a soda fountain machine in the back and a bus full of very hyper students, who are drinking more soda as they ride along the highway. Your goal is to drop the students off as quickly as possible. More specifically, every minute that a student is on your bus, he drinks another ounce of soda. Your goal is to drop the students off quickly, so that in total they drink as little soda as possible.

   You know how many students will get off of the bus at each exit. Your bus begins partway along the highway (probably not at either end), and moves at a constant rate. You must drive the bus along the highway- however you may drive forward to one exit then backward to an exit in the other direction, switching as often as you like (you can stop the bus, drop off students, and turn around instantaneously).

   Give an efficient algorithm to drop the students off so that they drink as little soda as possible. The input to the algorithm should be: the bus route (a list of the exits, together with the travel time between successive exits), the number of students you will drop off at each exit, and the current location of your bus (you may assume it is at an exit).

3. Suppose we want to display a paragraph of text on a computer screen. The text consists of n words, where the ith word is p_i pixels wide. We want to break the paragraph into several lines, each exactly P pixels long. Depending on which words we put on each line, we will need to insert different amounts of white space between the words. The paragraph should be fully justified, meaning that the first word on each line starts at its leftmost pixel, and *except for the last line*, the last character on each line ends at its rightmost pixel. There must be at least one pixel of whitespace between any two words on the same line.

   Define the *slop* of a paragraph layout as the sum over all lines, *except the last*, of the cube of the number of extra white-space pixels in each line (not counting the one pixel required between every adjacent pair of words). Specifically, if a line contains words i through j, then the amount of extra white space on that line is $P - j + i - \sum_{k=i}^{j} P_k$. Describe a dynamic programming algorithm to print the paragraph with minimum slop.

CS 473U: Undergraduate Algorithms, Fall 2006
Homework 3

Due Wednesday, October 4, 2006 in 3229 Siebel Center

Remember to turn in separate, individually stapled solutions to each of the problems.

---

1. Consider a perfect tree of height h, where every non-leaf node has 3 children. (Therefore, each of the 3^h leaves is at distance h from the root.) Every leaf has a boolean value associated with it - either 0 or 1. Every internal node gets the boolean value assigned to the majority of its children. Given the values assigned to the leaves, we want to find an algorithm that computes the value (0 or 1) of the root.

   It is not hard to find a (deterministic) algorithm that looks at every leaf and correctly determines the value of the root, but this takes $O(3^h)$ time. Describe and analyze a *randomized* algorithm that, on average, looks at asymptotically fewer leaves. That is, the expected number of leaves your algorithm examines should be $o(3^h)$.

2. We define a *meldable heap* to be a binary tree of elements, each of which has a priority, such that the priority of any node is less than the priority of its parent. (Note that the heap does **not** have to be balanced, and that the element with greatest priority is the root.) We also define the priority of a heap to be the priority of its root.

   The *meld* operation takes as input two (meldable) heaps and returns a single meldable heap H that contains all the elements of both input heaps. We define *meld* as follows:

   - Let H_1 be the input heap with greater priority, and H_2 the input heap with lower priority. (That is, the priority of $root(H_1)$ is greater than the priority of $root(H_2)$.) Let H_L be the left subtree of $root(H_1)$ and H_R be the right subtree of $root(H_1)$.
   - We set $root(H) = root(H_1)$.
   - We now flip a coin that comes up either "Left" or "Right" with equal probability.
     - If it comes up "Left", we set the left subtree of $root(H)$ to be H_L, and the right subtree of $root(H)$ to be $meld(H_R, H_2)$ (defined recursively).
     - If the coin comes up "Right", we set the right subtree of $root(H)$ to be H_R, and the left subtree of $root(H)$ to be $meld(H_L, H_2)$.
   - As a base case, melding any heap H_1 with an empty heap gives H_1.

   (a) Analyze the expected running time of $meld(H_a, H_b)$ if H_a is a (meldable) heap with n elements, and H_b is a (meldable) heap with m elements.

   (b) Describe how to perform each of the following operations using only melds, and give the running time of each.

   - $DeleteMax(H)$, which deletes the element with greatest priority.
   - $Insert(H, x)$, which inserts the element x into the heap H.
   - $Delete(H, x)$, which - given a pointer to element x in heap H - returns the heap with x deleted.

3. Randomized Selection. Given an (unsorted) array of n distinct elements and an integer k, SELECTION is the problem of finding the kth smallest element in the array. One easy solution is to sort the array in increasing order, and then look up the kth entry, but this takes $\Theta(n \log n)$ time. The randomized algorithm below attempts to do better, at least on average.

```
QuickSelect(Array A, n, k)
pivot ← Random(1, n)
S ← {x | x ∈ A, x < A[pivot]}
s ← |S|
L ← {x | x ∈ A, x > A[pivot]}
if (k = s + 1)
   return A[pivot]
else if (k ≤ s)
   return QuickSelect(S, s, k)
else
   return QuickSelect(L, n − (s + 1), k − (s + 1))
```

Here we assume that Random(a, b) returns an integer chosen uniformly at random from a to b (inclusive of a and b). The pivot position is randomly chosen; S is the set of elements smaller than the pivot element, and L the set of elements larger than the pivot. The sets S and L are found by comparing every other element of A to the pivot. We partition the elements into these two 'halves', and recurse on the appropriate half.

(a) Write a recurrence relation for the expected running time of QuickSelect.

(b) Given any two elements $x, y \in A$, what is the probability that x and y will be compared?

(c) Either from part (a) or part (b), find the expected running time of QuickSelect.

4. **[Extra Credit]**: In the previous problem, we found a $\Theta(n)$ algorithm for selecting the kth smallest element, but the constant hidden in the $\Theta(\cdot)$ notation is somewhat large. It is easy to find the *smallest* element using at most n comparisons; we would like to be able to extend this to larger k. Can you find a randomized algorithm that uses $n + \Theta(k \log k \log n)$[1] expected comparisons? (Note that there is no constant multiplying the n.)

Hint: While scanning through a random permutation of n elements, how many times does the smallest element seen so far change? (See HBS 0.) How many times does the kth smallest element so far change?

[1] There is an algorithm that uses $n + \Theta(k \log(n/k)$ comparisons, but this is even harder.

CS 473U: Undergraduate Algorithms, Fall 2006
Homework 4

Due Tuesday, October 10, 2006 in 3229 Siebel Center

Remember to submit **separate, individually stapled** solutions to each of the problems.

---

1. Chicago has many tall buildings, but only some of them have a clear view of Lake Michigan. Suppose we are given an array $A[1..n]$ that stores the height of n buildings on a city block, indexed from west to east. Building i has a good view of Lake Michigan if every building to the east of i is shorter than i. We present an algorithm that computes which buildings have a good view of Lake Michigan. Use the taxation method of amortized analysis to bound the amortized time spent in each iteration of the for loop. What is the total runtime?

```
GOODVIEW(A[1..n]):
  Initialize a stack S
  for i = 1 to n
      while (S not empty and A[i] ≥ A[S.top])
          POP(S)
      PUSH(S, i)
  return S
```

2. Design and analyze a simple data structure that maintains a list of integers and supports the following operations.

   (a) CREATE(): creates and returns a new list L

   (b) PUSH(L, x): appends x to the end of L

   (c) POP(L): deletes the last entry of L and returns it

   (d) LOOKUP(L, k): returns the kth entry of L

   Your solution may use these primitive data structures: arrays, balanced binary search trees, heaps, queues, single or doubly linked lists, and stacks. If your algorithm uses anything fancier, you must give an explicit implementation. Your data structure should support all operations in amortized constant time. In addition, your data structure should support LOOKUP() in worst-case $O(1)$ time. At all times, your data structure should use space which is linear in the number of objects it stores.

3. Consider a computer game in which players must navigate through a field of landmines, which are represented as points in the plane. The computer creates new landmines which the players must avoid. A player may ask the computer how many landmines are contained in any simple polygonal region; it is your job to design an algorithm which answers these questions efficiently.

   You have access to an efficient static data structure which supports the following operations.

- CREATES($\{p_1, p_2, \ldots, p_n\}$): creates a new data structure S containing the points $\{p_1, \ldots, p_n\}$. It has a worst-case running time of $T(n)$. Assume that $T(n)/n \geq T(n-1)/(n-1)$, so that the average processing time of elements does not decrease as n grows.
- DUMPS(S): destroys S and returns the set of points that S stored. It has a worst-case running time of $O(n)$, where n is the number of points in S.
- QUERYS(S, R): returns the number of points in S that are contained in the region R. It has a worst-case running time of $Q(n)$, where n is the number of points stored in S.

Unfortunately, the data structure does not support point insertion, which is required in your application. Using the given static data structure, design and analyze a dynamic data structure that supports the following operations.

(a) CREATED(): creates a new data structure D containing no points. It should have a worst-case constant running time.

(b) INSERTD(D, p): inserts p into D. It should run in amortized $O(\log n) \cdot T(n)/n$ time.

(c) QUERYD(D, R): returns the number of points in D that are contained in the region R. It should have a worst-case running time of $O(\log n) \cdot Q(n)$.

CS 473U: Undergraduate Algorithms, Fall 2006
Homework 5

Due Tuesday, October 24, 2006 in 3229 Siebel Center
Remember to turn in in separate, individually stapled solutions to each of the problems.

---

1. Makefiles:
   In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called 'make' that only recompiles those files that were changed after the most recent compilation, *and* any intermediate files in the compilation that depend on those that were changed. A Makefile is typically composed of a list of source files that must be compiled. Each of these source files is dependent on some of the other files that must be compiled. Thus a source file must be recompiled if a file on which it depends is changed.

   Assuming you have a list of which files have been recently changed, as well as a list for each source file of the files on which it depends, design and analyze an efficient algorithm to recompile only the necessary files. DO NOT worry about the details of parsing a Makefile.

2. Consider a graph G, with n vertices. Show that if any two of the following properties hold for G, then the third property must also hold.

   - G is connected.
   - G is acyclic.
   - G has $n - 1$ edges.

3. The weight of a spanning tree is the sum of the weights on the edges of the tree. Given a graph, G, describe an efficient algorithm (the most efficient one you can) to find the k lightest (with least weight) spanning trees of G.

   Analyze the running time of your algorithm. Be sure to prove your algorithm is correct.

CS 473U: Undergraduate Algorithms, Fall 2006
Homework 6

Due Wednesday, November 8, 2006 in 3229 Siebel Center

Remember to turn in separate, individually stapled solutions to each of the problems.

---

1. Dijkstra's algorithm can be used to determine shortest paths on graphs with some negative edge weights (as long as there are no negative cycles), but the worst-case running time is much worse than the $O(E+V \log V)$ it takes when the edge weights are all positive. Construct an infinite family of graphs - with negative edge weights - for which the asymptotic running time of Dijkstra's algorithm is $\Omega(2^{|V|})$.

2. It's a cold and rainy night, and you have to get home from Siebel Center. Your car has broken down, and it's too windy to walk, which means you have to take a bus. To make matters worse, there is no bus that goes directly from Siebel Center to your apartment, so you have to change buses some number of times on your way home. Since it's cold outside, you want to spend as little time as possible waiting in bus shelters.

   From a computer in Siebel Center, you can access an online copy of the MTD bus schedule, which lists bus routes and the arrival time of every bus at each stop on its route. Describe an algorithm which, given the schedule, finds a way for you to get home that minimizes the time you spend at bus shelters (the amount of time you spend on the bus doesn't matter). Since Siebel Center is warm and the nearest bus stop is right outside, you can assume that you wait inside Siebel until the first bus you want to take arrives outside. Analyze the efficiency of your algorithm and prove that it is correct.

3. The Floyd-Warshall all-pairs shortest path algorithm computes, for each $u, v \in V$, the shortest path from u to v. However, if the graph has negative cycles, the algorithm fails. Describe a modified version of the algorithm (*with the same asymptotic time complexity*) that correctly returns shortest-path distances, even if the graph contains negative cycles. That is, if there is a path from u to some negative cycle, and a path from that cycle to v, the algorithm should output $dist(u, v) = -\infty$. For any other pair u, v, the algorithm should output the length of the shortest directed path from u to v.

CS 473U: Undergraduate Algorithms, Fall 2006
Homework 6

Due at **4 p.m. on** Friday, November 17, 2006 in 3229 Siebel Center

---

Remember to turn in separate, individually stapled solutions to each of the problems.

---

1. Given an undirected graph $G(V, E)$, with three vertices $u, v, w \in V$, you want to know whether there exists a path from u to w via v. (That is, the path from u to w must use v as an intermediate vertex.) Describe an efficient algorithm to solve this problem.

2. *Ad-hoc Networks*, made up of cheap, low-powered wireless devices, are often used on battlefields, in regions that have recently suffered from natural disasters, and in other situtations where people might want to monitor conditions in hard-to-reach areas. The idea is that a large collection of the wireless devices could be dropped into the area from an airplane (for instance), and then they could be configured into an efficiently functioning network.

   Since the devices are cheap and low-powered, they frequently fail, and we would like our networks to be reliable. If a device detects that it is likely to fail, it should transmit the information it has to some other device (called a *backup*) within range of it. The range is limited; we assume that there is a distance d such that two devices can communicate if and only if they are within distance d of each other. To improve reliability, we don't want a device to transmit information to a neighbor that has already failed, and so we require each device v to have at least k backup devices that it could potentially contact, all of which must be within d meters of it. We call this the *backup set* of v. Also, we do not want any device to be in the backup set of too many other devices; if it were, and it failed, a large fraction of our network would be affected.

   The input to our problem is a collection of n devices, and for each pair u, v of devices, the distance between u and v. We are also given the distance d that determines the range of a device, and parameters b and k. Describe an algorithm that determines if, for each device, we can find a backup set of size k, while also requiring that no device appears in the backup set of more than b other devices.

3. **UPDATED:** Given a piece of text T and a pattern P (the 'search string'), an algorithm for the string-matching problem either finds the first occurrence of P in T, or reports that there is none. Modify the Knuth-Morris-Pratt (KMP) algorithm so that it solves the string-matching problem, even if the pattern contains the wildcards '?' and '*'. Here, '?' represents any *single* character of the text, and '*' represents any substring of the text (including the empty substring). For example, the pattern "A?B*?A" matches the text "ABACBCABBCCACBA" starting in position 3 (in three different ways), and position 7 (in two ways). For this input, your algorithm would need to return '3'.

   UPDATE: You may assume that the pattern you are trying to match containst at most 3 blocks of question marks; the usage of '*' wildcards is stll unrestricted. Here, a block refers to a string of consecutive '?'s in the pattern. For example, AAB??ACA???????BB contains 2 blocks of question marks; A?B?C?A?C contains 4 blocks of question marks.

4. In the two-dimensional pattern-matching problem, you are given an $m \times n$ matrix M and a $p \times q$ pattern P. You wish to find all positions (i, j) in M such that the the submatrix of M between rows i and $i + p - 1$ and between columns j and $j + q - 1$ is identical to P. (That is, the $p \times q$ sub-matrix of M below and to the right of position (i, j) should be identical to P.) Describe and analyze an efficient algorithm to solve this problem.[1]

[1] Note that the normal string-matching problem is the special case of the 2-dimensional problem where $m = p = 1$.

CS 473U: Undergraduate Algorithms, Fall 2006
Homework 8

Due Wednesday, December 6, 2006 in 3229 Siebel Center

Remember to submit **separate, individually stapled** solutions to each of the problems.

---

1. Given an array $A[1..n]$ of $n \geq 2$ distinct integers, we wish to find the second largest element using as few comparisons as possible.

   (a) Give an algorithm which finds the second largest element and uses at most $n + \lceil \lg n \rceil - 2$ comparisons in the worst case.

   *(b) Prove that every algorithm which finds the second largest element uses at least $n + \lceil \lg n \rceil - 2$ comparisons in the worst case.

2. Let R be a set of rectangles in the plane. For each point p in the plane, we say that the *rectangle depth* of p is the number of rectangles in R that contain p.

   (a) (Step 1: Algorithm Design) Design and analyze a polynomial-time algorithm which, given R, computes the maximum rectangle depth.

   (b) (Step 2: ???) Describe and analyze a polynomial-time reduction from the maximum rectangle depth problem to the maximum clique problem.

   (c) (Step 3: Profit!) In 2000, the Clay Mathematics Institute described the Millennium Problems: seven challenging open problems which are central to ongoing mathematical research. The Clay Institute established seven prizes, each worth one million dollars, to be awarded to anyone who solves a Millennium problem. One of these problems is the P = NP question. In (a), we developed a polynomial-time algorithm for the maximum rectangle depth problem. In (b), we found a reduction from this problem to an NP-complete problem. We know from class that if we find a polynomial-time algorithm for any NP-complete problem, then we have shown P = NP. Why hasn't Jeff used (a) and (b) to show P = NP and become a millionaire?

3. Let G be a complete graph with integer edge weights. If C is a cycle in G, we say that the *cost* of C is the sum of the weights of edges in C. Given G, the traveling salesman problem (TSP) asks us to compute a Hamiltonian cycle of minimum cost. Given G, the traveling salesman cost problem (TSCP) asks us to compute the cost of a minimum cost Hamiltonian cycle. Given G and an integer k, the traveling salesman decision problem (TSDP) asks us to decide if there is a Hamiltonian cycle in G of cost at most k.

   (a) Describe and analyze a polynomial-time reduction from TSP to TSCP.

   (b) Describe and analyze a polynomial-time reduction from TSCP to TSDP.

   (c) Describe and analyze a polynomial-time reduction from TSDP to TSP.

(d) What can you conclude about the relative computational difficulty of TSP, TSCP, and TSDP?

4. Let G be a graph. A set S of vertices of G is a *dominating set* if every vertex in G is either in S or adjacent to a vertex in S. Show that, given G and an integer k, deciding if G contains a dominating set of size at most k is NP-complete.

1. Probability

   (a) n people have checked their hats with a hat clerk. The clerk is somewhat absent-minded and returns the hats uniformly at random (with no regard for whether each hat is returned to its owner). On average, how many people will get back their own hats?

   (b) Let S be a uniformly random permutation of $\{1, 2, \ldots, n-1, n\}$. As we move from the left to the right of the permutation, let X denote the smallest number seen so far. On average, how many different values will X take?

2. A *tournament* is a directed graph where each pair of distinct vertices u, v has either the edge uv or the edge vu (but not both). A *Hamiltonian path* is a (directed) path that visits each vertex of the (di)graph. Prove that every tournament has a Hamiltonian path.

3. Describe and analyze a data structure that stores a set of n records, each with a numerical *key*, such that the following operation can be performed quickly:

   `Foo`(a): return the sum of the records with keys at least as large as a.

   For example, if the keys are:

   3 4 9 6 5 8 7 1 0

   then `Foo`(2) would return 42, since $3, 4, 5, 6, 7, 8, 9$ are all larger than 2 and $3+4+5+6+7+8+9 = 42$.

   You may assume that no two records have equal keys, and that no record has a key equal to a. Analyze both the size of your data structure and the running time of your `Foo` algorithm. Your data structure must be as small as possible and your `Foo` algorithm must be as fast as possible.

1. The Acme Company is planning a company party. In planning the party, each employee is assigned a *fun value* (a positive real number). The goal of the party planners is to maximize the total fun value (sum of the individual fun values) of the employees invited to the party. However, the planners are not allowed to invite both an employee and his direct boss. Given a tree containing the boss/underling structure of Acme, find the invitation list with the highest allowable fun value.

2. An *inversion* in an array A is a pair i, j such that $i < j$ and $A[i] > A[j]$. (In an n-element array, the number of inversions is between 0 and $\binom{n}{2}$.)

   Find an efficient algorithm to count the number of inversions in an n-element array.

3. A *tromino* is a geometric shape made from three squares joined along complete edges. There are only two possible trominoes: the three component squares may be joined in a line or an L-shape.

   (a) Show that it is possible to cover all but one square of a 64 x 64 checkerboard using L-shape trominoes. (In your covering, each tromino should cover three squares and no square should be covered more than once.)

   (b) Show that you can leave *any* single square uncovered.

   (c) Can you cover all but one square of a 64 x 64 checkerboard using *line* trominoes? If so, which squares can you leave uncovered?

1. **Moving on a Checkerboard**

   Suppose that you are given an $n \times n$ checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:

   1) the square immediately above
   2) the square that is one up and one to the left (but only if the checker is not already in the leftmost column)
   3) the square that is one up and one to the right (but only if the checker is not already in the rightmost column)

   Each time you move from square x to square y, you receive $p(x, y)$ dollars. You are given a list of the values $p(x, y)$ for each pair (x, y) for which a move from x to y is legal. Do not assume that $p(x, y)$ is positive.

   Give an algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. You algorithm is free to pick any square along the bottom edge as a starting point and any square along the top edge as a destination in order to maximize the number of dollars gathered along the way. What is the running time of your algorithm?

2. **Maximizing Profit**

   You are given lists of values $h_1, h_2, \ldots, h_k$ and $l_1, l_2, \ldots, l_k$. For each i you can choose $j_i = h_i$, $j_i = l_i$, or $j_i = 0$; the only catch is that if $j_i = h_i$ then j_{i-1} must be 0 (except for $i = 1$). Your goal is to maximize $\sum_{i=1}^{k} j_i$.

   Give an efficient algorithm that returns the maximum possible value of $\sum_{i=1}^{k} j_i$.

3. **Maximum alternating subsequence**

   An *alternating sequence* is a sequence $a_1, a_2, \ldots$ such that no three consecutive terms of the sequence satisfy $a_i > a_{i+1} > a_{i+2}$ or $a_i < a_{i+1} < a_{i+2}$.

   Given a sequence, efficiently find the longest alternating subsequence it contains. What is the running time of your algorithm?

1. **Championship Showdown**

   What excitement! The Champaign Spinners and the Urbana Dreamweavers have advanced to meet each other in the World Series of Basketweaving! The World Champions will be decided by a best of $2n - 1$ series of head-to-head weaving matches, and the first to win n matches will take home the coveted Golden Basket (for example, a best-of-7 series requires four match wins, but we will keep the generalized case). We know that for any given match there is a constant probability p that Champaign will win, and a subsequent probability $q = 1 - p$ that Urbana will win.

   Let $P(i, j)$ be the probability that Champaign will win the series given that they still need i more victories, whereas Urbana needs j more victories for the championship. $P(0, j) = 1$, $1 \leq j \leq n$, because Champaign needs no more victories to win. $P(i, 0) = 0$, $1 \leq i \leq n$, as Champaign cannot possibly win if Urbana already has. $P(0, 0)$ is meaningless. Champaign wins any particular match with probability p and loses with probability q, so

   $$P(i, j) = p \cdot P(i - 1, j) + q \cdot P(i, j - 1)$$

   for any $i \geq 1$ and $j \geq 1$.

   Create and analyze an $O(n^2)$-time dynamic programming algorithm that takes the parameters n, p, and q and returns the probability that Champaign will win the series (that is, calculate $P(n, n)$).

2. **Making Change**

   Suppose you are a simple shopkeeper living in a country with n different types of coins, with values $1 = c[1] < c[2] < \cdots < c[n]$. (In the U.S., for example, $n = 6$ and the values are $1, 5, 10, 25, 50$, and 100 cents.) Your beloved benevolent dictator, El Generalissimo, has decreed that whenever you give a customer change, you must use the smallest possible number of coins, so as not to wear out the image of El Generalissimo lovingly engraved on each coin by servants of the Royal Treasury.

   Describe and analyze a dynamic programming algorithm to determine, given a target amount A and a sorted array $c[1..n]$ of coin values, the smallest number of coins needed to make A cents in change. You can assume that $c[1] = 1$, so that it is possible to make change for any amount A.

3. **Knapsack**

   You are a thief, who is trying to choose the best collection of treasure (some subset of the n treasures, numbered 1 through n) to steal. The weight of item i is $w_i \in \mathbb{N}$ and the profit is $p_i \in \mathbb{R}$. Let $C \in \mathbb{N}$ be the maximum weight that your knapsack can hold. Your goal is to choose a subset of elements $S \subseteq \{1, 2, \ldots, n\}$ that maximizes your total profit $P(S) = \sum_{i \in S} p_i$, subject to the constraint that the sum of the weights $W(S) = \sum_{i \in S} w_i$ is not more than C.

   Give an algorithm that runs in time $O(Cn)$.

1. **Randomized Edge Cuts**

   We will randomly partition the vertex set of a graph G into two sets S and T. The algorithm is to flip a coin for each vertex and with probability $1/2$, put it in S; otherwise put it in T.

   (a) Show that the expected number of edges with one endpoint in S and the other endpoint in T is exactly half the edges in G.

   (b) Now say the edges have weights. What can you say about the sum of the weights of the edges with one endpoint in S and the other endpoint in T?

2. **Skip Lists**

   A *skip list* is built in layers. The bottom layer is an ordinary sorted linked list. Each higher layer acts as an "express lane" for the lists below, where an element in layer i appears in layer $i+1$ with some fixed probability p.

   ```
   1
   1-----4---6
   1---3-4---6-----9
   1-2-3-4-5-6-7-8-9-10
   ```

   (a) What is the probability a node reaches height h.

   (b) What is the probability any node is above $c \log n$ (for some fixed value of c)? Compute the value explicitly when $p = 1/2$ and $c = 4$.

   (c) To search for an entry x, scan the top layer until you find the last entry y that is less than or equal to x. If $y < x$, drop down one layer and in this new layer (beginning at y) find the last entry that is less than or equal to x. Repeat this process (dropping down a layer, then finding the last entry less than or equal to x) until you either find x or reach the bottom layer and confirm that x is not in the skip list. What is the expected search time?

   (d) Describe an efficient method for insertion. What is the expected insertion time?

3. **Clock Solitaire**

   In a standard deck of 52 cards, put 4 face-down in each of the 12 'hour' positions around a clock, and 4 face-down in a pile in the center. Turn up a card from the center, and look at the number on it. If it's number x, place the card face-up next to the face-down pile for x, and turn up the next card in the face-down pile for x (that is, the face-down pile corresponding to hour x). You win if, for each Ace $\leq x \leq$ Queen, all four cards of value x are turned face-up before all four Kings (the center cards) are turned face-up.

   What is the probability that you win a game of Clock Solitaire?

1. **Simulating Queues with Stacks**

   A *queue* is a first-in-first-out data structure. It supports two operations *push* and *pop*. Push adds a new item to the back of the queue, while pop removes the first item from the front of the queue. A *stack* is a last-in-first-out data structure. It also supports push and pop. As with a queue, push adds a new item to the back of the queue. However, pop removes the last item from the back of the queue (the one most recently added).

   Show how you can simulate a queue by using two stacks. Any sequence of pushes and pops should run in amortized constant time.

2. **Multistacks**

   A *multistack* consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first move all the elements in S_i to stack S_{i+1} to make room. But if S_{i+1} is already full, we first move all its members to S_{i+2}, and so on. To clarify, a user can only push elements onto S_0. All other pushes and pops happen in order to make space to push onto S_0. Moving a single element from one stack to the next takes $O(1)$ time.

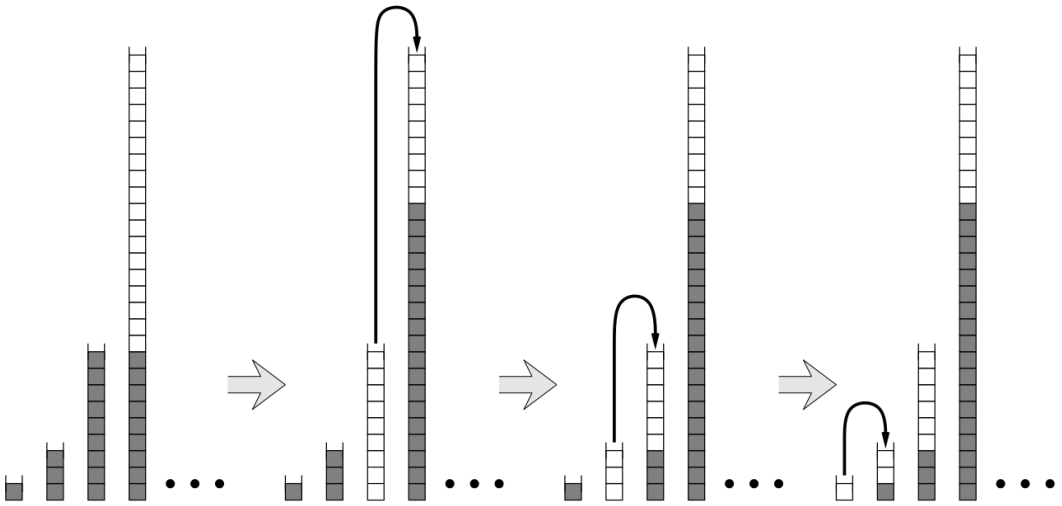


Figure 1. Making room for one new element in a multistack.

   (a) In the worst case, how long does it take to push one more element onto a multistack containing n elements?

   (b) Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack.

3. **Powerhungry function costs**

   A sequence of n operations is performed on a data structure. The ith operation costs i if i is an exact power of 2, and 1 otherwise. Determine the amortized cost of the operation.

1. **Representation of Integers**

   (a) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers–if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, $17 = F_7 + F_4 + F_2$.

   (b) Prove that any integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example $42 = 3^4 - 3^3 - 3^2 - 3^1$, $25 = 3^3 - 3^1 + 3^0$, $17 = 3^3 - 3^2 - 3^0$.

2. **Minimal Dominating Set**

   Suppose you are given a rooted tree T (not necessarily binary). You want to label each node in T with an integer 0 or 1, such that every node either has the label 1 or is adjacent to a node with the label 1 (or both). The *cost* of a labeling is the number of nodes with label 1. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree T.

3. **Names in Boxes**

   The names of 100 prisoners are placed in 100 wooden boxes, one name to a box, and the boxes are lined up on a table in a room. One by one, the prisoners are led into the room; each may look in at most 50 boxes, but must leave the room exactly as he found it and is permitted no further communication with the others.

   The prisoners have a chance to plot their strategy in advance, and they are going to need it, because unless *every single prisoner finds his own name* all will subsequently be executed. Find a strategy for them which has probability of success exceeding 30%. You may assume that the names are distributed in the boxes uniformly at random.

   (a) Calculate the probability of success if each prisoner picks 50 boxes uniformly at random.

   *(b) Consider the following strategy.

   The prisoners number themselves 1 to 100. Prisoner i begins by looking in box i. There he finds the name of prisoner j. If $j \neq i$, he continues by looking in box j. As long as prisoner i has not found his name, he continues by looking in the box corresponding to the last name he found.

   Describe the set of permutations of names in boxes for which this strategy will succeed.

   *(c) Count the number of permutations for which the strategy above succeeds. Use this sum to calculate the probability of success. You may find it useful to do this calculation for general n, then set $n = 100$ at the end.

   (d) We assumed that the names were distributed in the boxes uniformly at random. Explain how the prisoners could augment their strategy to make this assumption unnecessary.

1. **Dynamic MSTs**

   Suppose that you already have a minimum spanning tree (MST) in a graph. Now one of the edge weights changes. Give an efficient algorithm to find an MST in the new graph.

2. **Minimum Bottleneck Trees**

   In a graph G, for any pair of vertices u, v, let bottleneck(u, v) be the maximum over all paths p_i from u to v of the minimum-weight edge along p_i. Construct a spanning tree T of G such that for each pair of vertices, their bottleneck in G is the same as their bottleneck in T.

   One way to think about it is to imagine the vertices of the graph as islands, and the edges as bridges. Each bridge has a maximum weight it can support. If a truck is carrying stuff from u to v, how much can the truck carry? We don't care what route the truck takes; the point is that the smallest-weight edge on the route will determine the load.

3. **Eulerian Tours**

   An *Eulerian tour* is a "walk along edges of a graph" (in which successive edges must have a common endpoint) that uses each edge exactly once and ends at the vertex where it starts. A graph is called Eulerian if it has an Eulerian tour.

   Prove that a connected graph is Eulerian iff each vertex has even degree.

1. **Alien Abduction**

   Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road won't be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What route should he take so that he has the least chance of being abducted?

   More formally, you are given a directed graph $G = (V, E)$, where every edge e has an independent safety probability $p(e)$. The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex s to a given target vertex t.

2. **The Only SSSP Algorithm**

   In the lecture notes, Jeff mentions that all SSSP algorithms are special cases of the following generic SSSP algorithm. Each vertex v in the graph stores two values, which describe a tentative shortest path from s to v.

   - $\text{dist}(v)$ is the length of the tentative shortest $s \rightsquigarrow v$ path.
   - $\text{pred}(v)$ is the predecessor of v in the shortest $s \rightsquigarrow v$ path.

   We call an edge *tense* if $\text{dist}(u) + w(u \to v) < \text{dist}(v)$. Our generic algorithm repeatedly finds a tense edge in the graph and *relaxes* it:

   ```
   Relax(u → v):
     dist(v) ← dist(u) + w(u → v)
     pred(v) ← u
   ```

   If there are no tense edges, our algorithm is finished, and we have our desired shortest path tree. The correctness of the relaxation algorithm follows directly from three simple claims. The first of these is below. Prove it.

   - When the algorithm halts, if $\text{dist}(v) \neq \infty$, then $\text{dist}(v)$ is the total weight of the predecessor chain ending at v:
     $$s \to \cdots \to (pred(pred(v)) \to pred(v) \to v.$$

3. **Can't find a Cut-edge**

   A cut-edge is an edge which when deleted disconnects the graph. Prove or disprove the following. Every 3-regular graph has no cut-edge. (A common approach is induction.)

1. **Max-Flow with vertex capacities**

   In a standard $s - t$ Maximum-Flow Problem, we assume edges have capacities, and there is no limit on how much flow is allowed to pass through a node. In this problem, we consider the variant of Maximum-Flow and Minimum-Cut problems with node capacities.

   More specifically, each node, n_i, has a capacity c_i. The edges have unlimited capacity. Show how you can model this problem as a standard Max-flow problem (where the weights are on the edges).

2. **Emergency evacuation**

   Due to large-scale flooding in a region, paramedics have identified a set of n injured people distributed across the region who need to be reushed to hospitals. There are k hospitals in the region, and each of the n people needs to be brought to a hospital that is within a half-hour's driving time of their current location.

   At the same time, we don't want to overload any hospital by sending too many patients its way. We'd like to distribute the people so that each hospital receives at most $\lceil n/k \rceil$ people.

   Show how to model this problem as a Max-flow problem.

3. **Tracking a Hacker**

   A computer network (with each edge weight 1) is designed to carry traffic from a source s to a destination t. Recently, a computer hacker destroyed some of the edges in the graph. Normally, the maximum $s - t$ flow in G is k. Unfortunately, there is currently no path from s to t. Fortunately, the sysadmins know that the hacker destroyed at most k edges of the graph.

   The sysadmins are trying to diagnose which of the nodes of the graph are no longer reachable. They would like to avoid testing each node. They are using a monitoring tool with the following behavior. If you use the command *ping(v)*, for a given node v, it will tell you whether there is currently a path from s to v (so *ping(t)* will return `False` but *ping(s)* will return `True`).

   Give an algorithm that accomplishes this task using only $O(k \log n)$ pings. (You may assume that any algorithm you wish to run on the original network (before the hacker destroyed edges) runs for free, since you have a model of that network on your computer.)

1. **Updating a maximum flow**

   Suppose you are given a directed graph $G = (V, E)$, with a positive integer capacity c_e on each edge e, a designated source $s \in V$, and a designated sink $t \in V$. You are also given a maximum $s - t$ flow in G, defined by a flow value f_e on each edge e. The flow $\{f_e\}$ is *acyclic*: There is no cycle in G on which all edges carry positive flow.

   Now suppose we pick a specific edge $e^* \in E$ and reduce its capacity by 1 unit. Show how to find a maximum flow in the resulting capacitated graph in time $O(m + n)$, where m is the number of edges in G and n is the number of nodes.

2. **Cooking Schedule**

   You live in a cooperative apartment with n other people. The co-op needs to schedule cooks for the next n days, so that each person cooks one day and each day there is one cook. In addition, each member of the co-op has a list of days they are available to cook (and is unavailable to cook on the other days).

   Because of your superior CS473 skills, the co-op selects you to come up with a schedule for cooking, so that everyone cooks on a day they are available.

   (a) Describe a bipartite graph G so that G has a perfect matching if and only if there is a feasible schedule for the co-op.

   (b) A friend of yours tried to help you out by coming up with a cooking schedule. Unfortunately, when you look at the schedule he created, you notice a big problem. $n - 2$ of the people are scheduled for different nights on which they are available: no problem there. But the remaining two people are assigned to cook on the same night (and no one is assigned to the last night).

   You want to fix your friend's mistake, but without having to recompute everything from scratch. Show that it's possible, using his "almost correct" schedule to decide in $O(n^2)$ time whether there exists a feasible schedule.

3. **Disjoint paths in a digraph**

   Let $G = (V, E)$ be a directed graph, and suppose that for each node v, the number of edges into v is equal to the number of edges out of v. That is, for all v,

$$|\{(u, v) : (u, v) \in E\}| = |\{(v, w) : (v, w) \in E\}|.$$

   Let x, y be two nodes of G, and suppose that there exist k mutually edge-disjoint paths from x to y. Under these conditions, does it follow that there exist k mutually edge-disjoint paths from y to x. Give a proof or a counterexample with explanation.

1. **String matching: an example**

   (a) Build a finite automata to search for the string "bababoon".

   (b) Use the automata from part (a) to build the prefix function for Knuth-Morris-Pratt.

   (c) Use the automata or the prefix function to search for "bababoon" in the string "babybaboon-buysbananasforotherbabybababoons".

2. **Cooking Schedule Strikes Back**

   You live in a cooperative apartment with n other people. The co-op needs to schedule cooks for the next $5n$ days, so that each person cooks five days and each day there is one cook. In addition, each member of the co-op has a list of days they are available to cook (and is unavailable to cook on the other days).

   Because of your success at headbanging last week, the co-op again asks you to compose a cooking schedule. Unfortunately, you realize that no such schedule is possible Give a schedule for the cooking so that no one has to cook on more than 2 days that they claim to be unavailable.

3. **String matching on Trees**

   You are given a rooted tree T (not necessarily binary), in which each node has a character. You are also given a pattern $P = p_1 p_2 \cdots p_l$. Search for the string as a subtree. In other words, search for a subtree in which p_i is on a child of the node containing p_{i-1} for each $2 \leq i \leq l$.

1. **Self-reductions**

   In each case below assume that you are given a black box which can answer the decision version of the indicated problem. Use a polynomial number of calls to the black box to construct the desired set.

   (a) Independent set: Given a graph G and an integer k, does G have a subset of k vertices that are pairwise nonadjacent?

   (b) Subset sum: Given a multiset (elements can appear more than once) $X = \{x_1, x_2, \ldots, x_k\}$ of positive integers, and a positive integer S does there exist a subset of X with sum exactly S?

2. **Lower Bounds**

   Give adversary arguments to prove the indicated lower bounds for the following problems:

   (a) Searching in a sorted array takes at least $1 + \lfloor \lg_2 n \rfloor$ queries.

   (b) Let M be an $n \times n$ array of real values that is increasing in both rows and columns. Prove that searching for a value requires at least n queries.

3. **k-coloring**

   Show that we can solve the problem of constructing a k-coloring of a graph by using a polynomial number of calls to a black box that determines whether a graph has such a k-coloring. (Hint: Try reducing via an intermediate problem that asks whether a partial coloring of a graph can be extended to a proper k-coloring.)

1. **NP-hardness Proofs: Restriction**
   Prove that each of the following problems is NP-hard. In each part, find a special case of the given problem that is equivalent to a known NP-hard problem.

   (a) **Longest Path**
   Given a graph G and a positive integer k, does G contain a path with k or more edges?

   (b) **Partition into Hamiltonian Subgraphs**
   Given a graph G and a positive integer k, can the vertices of G be partitioned into at most k disjoint sets such that the graph induced by each set has a Hamiltonian cycle?

   (c) **Set Packing**
   Given a collection of finite sets C and a positive integer k, does C contain k disjoint sets?

   (d) **Largest Common Subgraph**
   Given two graphs G_1 and G_2 and a positive integer k, does there exist a graph G_3 such that G_3 is a subgraph of both G_1 and G_2 and G_3 has at least k edges?

2. **Domino Line**
   You are given an unusual set of dominoes; each domino has a number on each end, but the numbers may be arbitarily large and some numbers appear on many dominoes, while other numbers only appear on a few dominoes. Your goal is to form a line using all the dominoes so that adjacent dominoes have the same number on their adjacent halves. Either give an efficient algorithm to solve the problem or show that it is NP-hard.

3. **Set Splitting**
   Given a finite set S and a collection of subsets C is there a partition of S into two sets S_1 and S_2 such that no subset in C is contained entirely in S_1 or S_2? Show that the problem is NP-hard. (Hint: use NAE-3SAT, which is similar to 3SAT except that a satisfying assingment does not allow all 3 variables in a clause to be true.)

You have 120 minutes to answer four of these five questions.

Write your answers in the separate answer booklet.

1. **Multiple Choice.**

   Each of the questions on this page has one of the following five answers:

A: $\Theta(1)$	B: $\Theta(\log n)$	C: $\Theta(n)$	D: $\Theta(n \log n)$	E: $\Theta(n^2)$

   Choose the correct answer for each question. Each correct answer is worth $+1$ point; each incorrect answer is worth $-\frac{1}{2}$ point; each "I don't know" is worth $+\frac{1}{4}$ point. Your score will be rounded to the nearest *non-negative* integer. You do *not* need to justify your answers; just write the correct letter in the box.

   (a) What is $\dfrac{5}{n} + \dfrac{n}{5}$?

   (b) What is $\displaystyle\sum_{i=1}^{n} \frac{n}{i}$?

   (c) What is $\displaystyle\sum_{i=1}^{n} \frac{i}{n}$?

   (d) How many bits are required to represent the nth Fibonacci number in binary?

   (e) What is the solution to the recurrence $T(n) = 2T(n/4) + \Theta(n)$?

   (f) What is the solution to the recurrence $T(n) = 16T(n/4) + \Theta(n)$?

   (g) What is the solution to the recurrence $T(n) = T(n-1) + 1/n^2$?

   (h) What is the worst-case time to search for an item in a binary search tree?

   (i) What is the worst-case running time of quicksort?

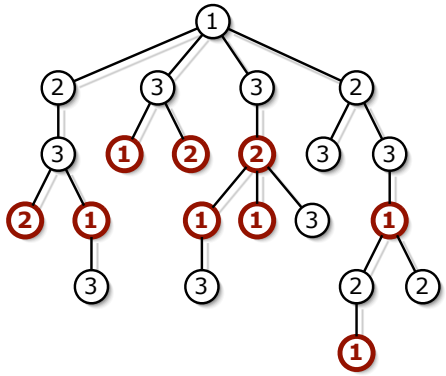
   (j) What is the running time of the fastest possible algorithm to solve Sudoku puzzles? A Sudoku puzzle consists of a 9×9 grid of squares, partitioned into nine 3×3 sub-grids; some of the squares contain digits between 1 and 9. The goal of the puzzle is to enter digits into the blank squares, so that each digit between 1 and 9 appears exactly once in each row, each column, and each 3×3 sub-grid. The initial conditions guarantee that the solution is unique.

2							4	
	7		5					
				1		9		
6		4			2			
	8						5	
			9			3		7
		1		4				
					3		8	
	5							6

A Sudoku puzzle. **Don't try to solve this during the exam!**

2. Oh, no! You have been appointed as the gift czar for Giggle, Inc.'s annual mandatory holiday party! The president of the company, who is certifiably insane, has declared that *every* Giggle employee must receive one of three gifts: (1) an all-expenses-paid six-week vacation anywhere in the world, (2) an all-the-pancakes-you-can-eat breakfast for two at Jumping Jack Flash's Flapjack Stack Shack, or (3) a burning paper bag full of dog poop. Corporate regulations prohibit any employee from receiving the same gift as his/her direct supervisor. Any employee who receives a better gift than his/her direct supervisor will almost certainly be fired in a fit of jealousy. How do you decide what gifts everyone gets if you want to minimize the number of people that get fired?

   More formally, suppose you are given a rooted tree T, representing the company hierarchy. You want to label each node in T with an integer 1, 2, or 3, such that every node has a different label from its parent.. The *cost* of an labeling is the number of nodes that have smaller labels than their parents. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree T. (Your algorithm does *not* have to compute the actual best labeling—just its cost.)



A tree labeling with cost 9. Bold nodes have smaller labels than their parents.
This is *not* the optimal labeling for this tree.

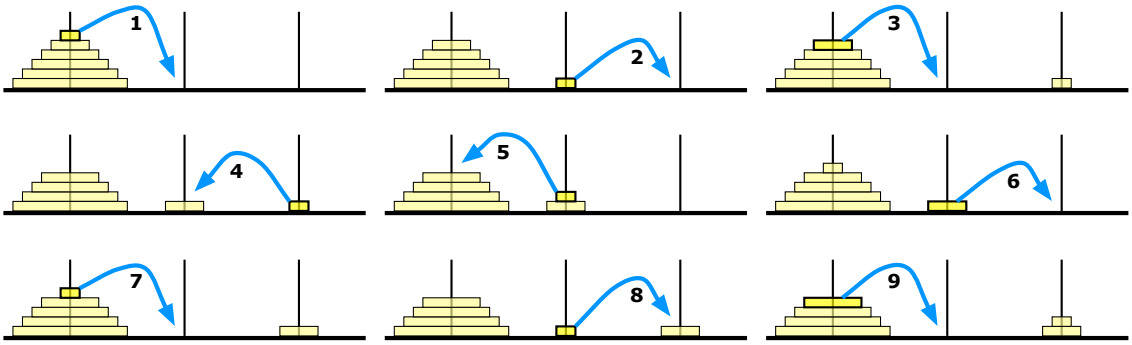
3. Suppose you are given an array $A[1..n]$ of n distinct integers, sorted in increasing order. Describe and analyze an algorithm to determine whether there is an index i such that $A[i] = i$, in $o(n)$ time. *[Hint: Yes, that's* ***little****-oh of* n*. What can you say about the sequence* $A[i] - i$*?]*

4. Describe and analyze a *polynomial-time* algorithm to compute the length of the longest common subsequence of two strings $A[1..m]$ and $B[1..n]$. For example, given the strings '`DYNAMIC`' and '`PROGRAMMING`', your algorithm would return the number 3, because the longest common subsequence of those two strings is '`AMI`'. You must give a complete, self-contained solution; don't just refer to HW1.

5. Recall that the Tower of Hanoi puzzle consists of three pegs and n disks of different sizes. Initially, all the disks are on one peg, stacked in order by size, with the largest disk on the bottom and the smallest disk on top. In a single move, you can transfer the highest disk on any peg to a different peg, except that you may never place a larger disk on top of a smaller one. The goal is to move all the disks onto one other peg.

   Now suppose the pegs are arranged in a row, and you are forbidden to transfer a disk directly between the left and right pegs in a single move; every move must involve the middle peg. How many moves suffice to transfer all n disks from the left peg to the right peg under this restriction? **Prove your answer is correct.**

   For full credit, give an *exact* upper bound. A correct upper bound using $O(\cdot)$ notation (with a proof of correctness) is worth 7 points.



The first nine moves in a restricted Towers of Hanoi solution.

1. On an overnight camping trip in Sunnydale National Park, you are woken from a restless sleep by a scream. As you crawl out of your tent to investigate, a terrified park ranger runs out of the woods, covered in blood and clutching a crumpled piece of paper to his chest. As he reaches your tent, he gasps, "Get out... while... you...", thrusts the paper into your hands, and falls to the ground. Checking his pulse, you discover that the ranger is stone dead.

   You look down at the paper and recognize a map of the park, drawn as an undirected graph, where vertices represent landmarks in the park, and edges represent trails between those landmarks. (Trails start and end at landmarks and do not cross.) You recognize one of the vertices as your current location; several vertices on the boundary of the map are labeled EXIT.

   On closer examination, you notice that someone (perhaps the poor dead park ranger) has written a real number between 0 and 1 next to each vertex and each edge. A scrawled note on the back of the map indicates that a number next to an edge is the probability of encountering a vampire along the corresponding trail, and a number next to a vertex is the probability of encountering a vampire at the corresponding landmark. (Vampires can't stand each other's company, so you'll never see more than one vampire on the same trail or at the same landmark.) The note warns you that stepping off the marked trails will result in a slow and painful death.

   You glance down at the corpse at your feet. Yes, his death certainly looked painful. Wait, was that a twitch? Are his teeth getting longer? After driving a tent stake through the undead ranger's heart, you wisely decide to leave the park immediately.

   Describe and analyze an efficient algorithm to find a path from your current location to an arbitrary EXIT node, such that the total *expected number* of vampires encountered along the path is as small as possible. *Be sure to account for* ***both*** *the vertex probabilities* ***and*** *the edge probabilities!*

2. Consider the following solution for the union-find problem, called *union-by-weight*. Each set leader $\overline{x}$ stores the number of elements of its set in the field $\mathit{weight}(\overline{x})$. Whenever we UNION two sets, the leader of the *smaller* set becomes a new child of the leader of the *larger* set (breaking ties arbitrarily).

```
MAKESET(x):
  parent(x) ← x
  weight(x) ← 1
```

```
FIND(x):
  while x ≠ parent(x)
      x ← parent(x)
  return x
```

```
UNION(x, y)
  x̄ ← FIND(x)
  ȳ ← FIND(y)
  if weight(x̄) > weight(ȳ)
      parent(ȳ) ← x̄
      weight(x̄) ← weight(x̄) + weight(ȳ)
  else
      parent(x̄) ← ȳ
      weight(x̄) ← weight(x̄) + weight(ȳ)
```

   Prove that if we use union-by-weight, the *worst-case* running time of FIND is $O(\log n)$.

3. ***Prove or disprove***[1] each of the following statements.

   (a) Let G be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of G includes the lightest edge in every cycle in G.

   (b) Let G be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of G excludes the heaviest edge in every cycle in G.

4. In Homework 2, you were asked to analyze the following algorithm to find the kth smallest element from an unsorted array. (The algorithm is presented here in iterative form, rather than the recursive form you saw in the homework, but it's exactly the same algorithm.)

```
QUICKSELECT(A[1..n], k):
  i ← 1; j ← n
  while i ≤ j
      r ← PARTITION(A[i..j], RANDOM(i, j))
      if r = k
          return A[r]
      else if r > k
          j ← r − 1
      else
          i ← r + 1
```

   The algorithm relies on two subroutines. $\text{RANDOM}(i, j)$ returns an integer chosen uniformly at random from the range $[i \,..\, j]$. $\text{PARTITION}(A[i \,..\, j], p)$ partitions the subarray $A[i \,..\, j]$ using the pivot value $A[p]$ and returns the new index of the pivot value in the partitioned array.

   What is the *exact* expected number of iterations of the main loop when $k = 1$? ***Prove*** your answer is correct. A correct $\Theta(\cdot)$ bound (with proof) is worth 7 points. You may assume that the input array $A[\,]$ contains n distinct integers.

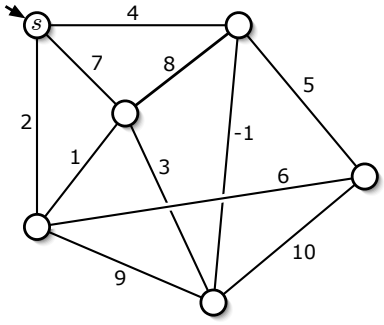
5. Find the following spanning trees for the weighted graph shown below.

   (a) A breadth-first spanning tree rooted at s.

   (b) A depth-first spanning tree rooted at s.

   (c) A shortest-path tree rooted at s.

   (d) A minimum spanning tree.



   You do *not* need to justify your answers; just clearly indicate the edges of each spanning tree. Yes, one of the edges has negative weight.

[1] But not both! If you give us *both* a proof *and* a disproof for the same statement, you will get no credit, even if one of your arguments is correct.

1. A *double-Hamiltonian* circuit in an undirected graph G is a closed walk that visits every vertex in G exactly *twice*, possibly by traversing some edges more than once. ***Prove*** that it is NP-hard to determine whether a given undirected graph contains a double-Hamiltonian circuit.

2. Suppose you are running a web site that is visited by the same set of people every day. Each visitor claims membership in one or more *demographic groups*; for example, a visitor might describe himself as male, 31-40 years old, a resident of Illinois, an academic, a blogger, a Joss Whedon fan[1], and a Sports Racer.[2] Your site is supported by advertisers. Each advertiser has told you which demographic groups should see its ads and how many of its ads you must show each day. Altogether, there are n visitors, k demographic groups, and m advertisers.

   Describe an efficient algorithm to determine, given all the data described in the previous paragraph, whether you can show each visitor exactly *one* ad per day, so that every advertiser has its desired number of ads displayed, and every ad is seen by someone in an appropriate demographic group.

3. Describe and analyze a data structure to support the following operations on an array $X[1..n]$ as quickly as possible. Initially, $X[i] = 0$ for all i.

   - Given an index i such that $X[i] = 0$, set $X[i]$ to 1.
   - Given an index i, return $X[i]$.
   - Given an index i, return the smallest index $j \geq i$ such that $X[j] = 0$, or report that no such index exists.

   For full credit, the first two operations should run in *worst-case constant* time, and the amortized cost of the third operation should be as small as possible. *[Hint: Use a modified union-find data structure.]*

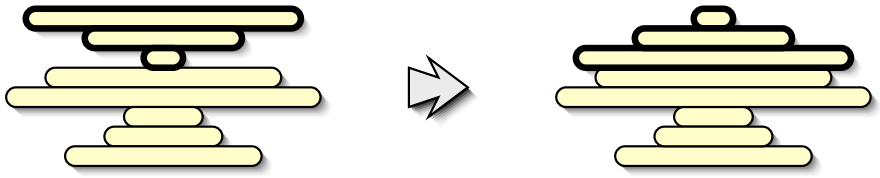
4. The next time you are at a party, one of the guests will suggest everyone play a round of Three-Way Mumbledypeg, a game of skill and dexterity that requires three teams and a knife. The official Rules of Three-Way Mumbledypeg (fixed during the Holy Roman Three-Way Mumbledypeg Council in 1625) require that (1) each team *must* have at least one person, (2) any two people on the same team *must* know each other, and (3) everyone watching the game *must* be on one of the three teams. Of course, it will be a really *fun* party; nobody will want to leave. There will be several pairs of people at the party who don't know each other. The host of the party, having heard thrilling tales of your prowess in all things algorithmic, will hand you a list of which pairs of partygoers know each other and ask you to choose the teams, while he sharpens the knife.

   Either describe and analyze a polynomial time algorithm to determine whether the partygoers can be split into three legal Three-Way Mumbledypeg teams, or prove that the problem is NP-hard.

[1] Har har har! Mine is an evil laugh! Now *die!*

[2] It's Ride the Fire Eagle Danger Day!

5. Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*—insert a spatula under the top k pancakes, for some integer k between 1 and n, and flip them all over.



Flipping the top three pancakes.

   (a) Describe an efficient algorithm to sort an arbitrary stack of n pancakes. *Exactly* how many flips does your algorithm perform in the worst case? (For full credit, your algorithm should perform as few flips as possible; an optimal $\Theta()$ bound is worth three points.)

   (b) Now suppose one side of each pancake is burned. *Exactly* how many flips do you need to sort the pancakes *and* have the burned side of every pancake on the bottom? (For full credit, your algorithm should perform as few flips as possible; an optimal $\Theta()$ bound is worth three points.)

6. Describe and analyze an efficient algorithm to find the length of the longest substring that appears both forward and backward in an input string $T[1..n]$. The forward and backward substrings must not overlap. Here are several examples:

   - Given the input string ALGORITHM, your algorithm should return 0.
   - Given the input string RECURSION, your algorithm should return 1, for the substring R.
   - Given the input string REDIVIDE, your algorithm should return 3, for the substring EDI. (The forward and backward substrings must not overlap!)
   - Given the input string DYNAMICPROGRAMMINGMANYTIMES, your algorithm should return 4, for the substring YNAM.

   For full credit, your algorithm should run in $O(n^2)$ time.

7. A *double-Eulerian* circuit in an undirected graph G is a closed walk that traverses every edge in G exactly twice. Describe and analyze a *polynomial-time* algorithm to determine whether a given undirected graph contains a double-Eulerian circuit.

CS 473G: Graduate Algorithms, Spring 2007
Homework 0

Due **in class** at 11:00am, Tuesday, January 30, 2007

Name:	
Net ID:	Alias:

☐ I understand the Course Policies.

- Neatly print your full name, your NetID, and an alias of your choice in the boxes above, and staple this page to your solution to problem 1. We will list homework and exam grades on the course web site by alias. **By providing an alias, you agree to let us list your grades; if you do not provide an alias, your grades will not be listed.** For privacy reasons, your alias should not resemble your name, your NetID, your university ID number, or (God forbid!) your Social Security number. Please use the same alias for every homework and exam.

- Read the Course Policies on the course web site, and then check the box above. Among other things, this page describes what we expect in your homework solutions, as well as policies on grading standards, regrading, extra credit, and plagiarism. In particular:

  - Submit each numbered problem separately, on its own piece(s) of paper. If you need more than one page for a problem, staple just *those* pages together, but keep different problems separate. **Do not staple your entire homework together.**
  - You may use *any* source at your disposal—paper, electronic, or human—but you *must* write your answers in your own words, and you *must* cite every source that you use.
  - Algorithms or proofs containing phrases like "and so on" or "repeat this for all n", instead of an explicit loop, recursion, or induction, are worth zero points.
  - Answering "I don't know" to any homework or exam problem is worth 25% partial credit.

  If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup.

- This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, discrete probability, graphs, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** The early chapters of Kleinberg and Tardos (or any algorithms textbook) should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks.

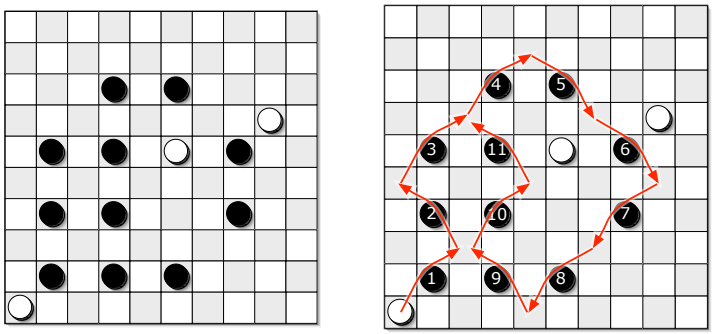
- Every homework will have five problems, each worth 10 points. Stars indicate more challenging problems. Many homeworks will also include an extra-credit problem.

---

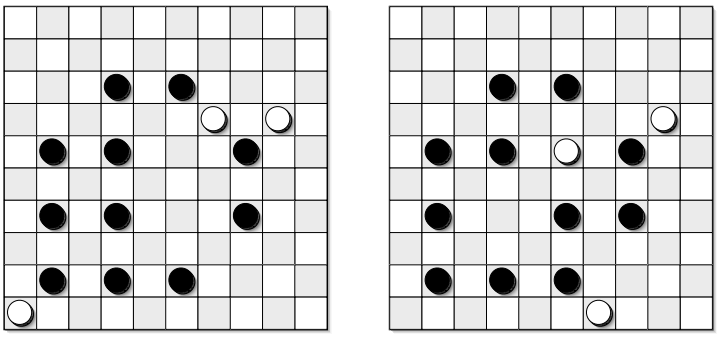
*1. Draughts/checkers is a game played on an $m \times m$ grid of squares, alternately colored light and dark. (The game is usually played on an 8×8 or 10×10 board, but the rules easily generalize to any board size.) Each dark square is occupied by at most one game piece (usually called a *checker* in the U.S.), which is either black or white; light squares are always empty. One player ("White") moves the white pieces; the other ("Black") moves the black pieces.

Consider the following simple version of the game, essentially American checkers or British draughts, but where every piece is a king.[1] Pieces can be moved in any of the four diagonal directions, either one or two steps at a time. On each turn, a player either *moves* one of her pieces one step diagonally into an empty square, or makes a series of *jumps* with one of her checkers. In a single jump, a piece moves to an empty square two steps away in any diagonal direction, but only if the intermediate square is occupied by a piece of the opposite color; this enemy piece is *captured* and immediately removed from the board. Multiple jumps are allowed in a single turn as long as they are made by the same piece. A player wins if her opponent has no pieces left on the board.

Describe an algorithm[2] that correctly determines whether White can capture every black piece, thereby winning the game, *in a single turn*. The input consists of the width of the board (m), a list of positions of white pieces, and a list of positions of black pieces. For full credit, your algorithm should run in $O(n)$ time, where n is the total number of pieces, but any algorithm that runs in time polynomial in n and m is worth significant partial credit.



White wins in one turn.



White cannot win in one turn from either of these positions.

*[Hint: The greedy strategy—make arbitrary jumps until you get stuck—does **not** always find a winning sequence of jumps even when one exists.]*

[1] Most variants of draughts have 'flying kings', which behave *very* differently than what's described here.

[2] Since you've read the Course Policies, you know what this phrase means.

2. (a) Prove that any positive integer can be written as the sum of distinct powers of 2. *[Hint: "Write the number in binary" is **not** a proof; it just restates the problem.]* For example:

$$\begin{aligned} 16+1 &= 17 &&= 2^4+2^0 \\ 16+4+2+1 &= 23 &&= 2^4+2^2+2^1+2^0 \\ 32+8+1 &= 42 &&= 2^5+2^3+2^1 \end{aligned}$$

   (b) Prove that *any* integer (positive, negative, or zero) can be written as the sum of distinct powers of -2. For example:

$$\begin{aligned} -32+16-2+1 &= -17 &&= (-2)^5+(-2)^4+(-2)^1+(-2)^0 \\ 64-32-8-2+1 &= 23 &&= (-2)^6+(-2)^5+(-2)^3+(-2)^1+(-2)^0 \\ 64-32+16-8+4-2 &= 42 &&= (-2)^6+(-2)^5+(-2)^4+(-2)^3+(-2)^2+(-2)^1 \end{aligned}$$

3. Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

   Prove that any finite set of pigeons can be arranged in a row from left to right so that every pigeon pecks the pigeon immediately to its left.

4. On their long journey from Denmark to England, Rosencrantz and Guildenstern amuse themselves by playing the following game with a fair coin. First Rosencrantz flips the coin over and over until it comes up tails. Then Guildenstern flips the coin over and over until he gets as many heads in a row as Rosencrantz got on his turn. Here are three typical games:

   Rosencrantz: H H T
   Guildenstern: H T H H

   Rosencrantz: T
   Guildenstern: (no flips)

   Rosencrantz: H H H T
   Guildenstern: T H H T H H T H T T H H H

   (a) What is the expected number of flips in one of Rosencrantz's turns?

   (b) Suppose Rosencrantz flips k heads in a row on his turn. What is the expected number of flips in Guildenstern's next turn?

   (c) What is the expected total number of flips (by both Rosencrantz and Guildenstern) in a single game?

Prove that your answers are correct. If you have to appeal to "intuition" or "common sense", your answer is almost certainly wrong! You must give *exact* answers for full credit, but a correct asymptotic bound for part (b) is worth significant credit.

5. (a) **[5 pts]** Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Assume reasonable but nontrivial base cases. If your solution requires a particular base case, say so.

$$A(n) = 3A(n/9) + \sqrt{n}$$

$$B(n) = 4B(n-1) - 4B(n-2)$$

$$C(n) = \frac{\pi\, C(n-1)}{\sqrt{2}\, C(n-2)} \qquad \textit{[Hint: This is easy!]}$$

$$D(n) = \max_{n/4<k<3n/4} \big(D(k) + D(n-k) + n\big)$$

$$E(n) = 2E(n/2) + 4E(n/3) + 2E(n/6) + n^2$$

Do *not* turn in proofs—just a list of five functions—but you should do them anyway, just for practice. *[Hint: On the course web page, you can find a handout describing several techniques for solving recurrences.]*

(b) **[5 pts]** Sort the functions in the box from asymptotically smallest to asymptotically largest, indicating ties if there are any. **Do *not* turn in proofs**—just a sorted list of 16 functions—but you should do them anyway, just for practice.

To simplify your answer, write $f(n) \ll g(n)$ to indicate that $f(n) = o(g(n))$, and write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions $n^2, n, \binom{n}{2}, n^3$ could be sorted either as $n \ll n^2 \equiv \binom{n}{2} \ll n^3$ or as $n \ll \binom{n}{2} \equiv n^2 \ll n^3$.

n	$\lg n$	$\sqrt{n}$	3^n
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$3^{\sqrt{n}}$	$\sqrt{3^n}$
$3^{\lg n}$	$\lg(3^n)$	$3^{\lg \sqrt{n}}$	$3^{\sqrt{\lg n}}$
$\sqrt{3^{\lg n}}$	$\lg(3^{\sqrt{n}})$	$\lg \sqrt{3^n}$	$\sqrt{\lg(3^n)}$

Recall that $\lg n = \log_2 n$.

CS 473G: Graduate Algorithms, Spring 2007
Homework 1

Due February 6, 2007

Remember to submit **separate, individually stapled** solutions to each of the problems.

---

1. Jeff tries to make his students happy. At the beginning of class, he passes out a questionnaire to students which lists a number of possible course policies in areas where he is flexible. Every student is asked to respond to each possible course policy with one of "strongly favor", "mostly neutral", or "strongly oppose". Each student may respond with "strongly favor" or "strongly oppose" to at most five questions. Because Jeff's students are very understanding, each student is happy if he or she prevails in just one of his or her strong policy preferences. Either describe a polynomial time algorithm for setting course policy to maximize the number of happy students or show that the problem is NP-hard.

2. Consider a variant 3SAT′ of 3SAT which asks, given a formula ϕ in conjunctive normal form in which each clause contains at most 3 literals and each variable appears in at most 3 clauses, is ϕ satisfiable? Prove that 3SAT′ is NP-complete.

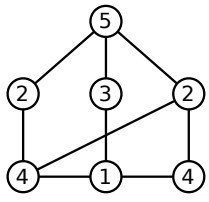
3. For each problem below, either describe a polynomial-time algorithm to solve the problem or prove that the problem is NP-complete.

   (a) A *double-Eulerian* circuit in an undirected graph G is a closed walk that traverses every edge in G exactly twice. Given a graph G, does G have a double-Eulerian circuit?

   (b) A *double-Hamiltonian* circuit in an undirected graph G is a closed walk that visits every vertex in G exactly twice. Given a graph G, does G have a double-Hamiltonian circuit?

4. Suppose you have access to a magic black box; if you give it a graph G as input, the black box will tell you, in constant time, if there is a proper 3-coloring of G. Describe a polynomial time algorithm which, given a graph G that is 3-colorable, uses the black box to compute a 3-coloring of G.

5. Let C_5 be the graph which is a cycle on five vertices. A $(5, 2)$-coloring of a graph G is a function $f : V(G) \to \{1, 2, 3, 4, 5\}$ such that every pair $\{u, v\}$ of adjacent vertices in G is mapped to a pair $\{f(u), f(v)\}$ of vertices in C_5 which are at distance two from each other.



A $(5, 2)$-coloring of a graph.

   Using a reduction from 5COLOR, prove that the problem of deciding whether a given graph G has a $(5, 2)$-coloring is NP-complete.

CS 473G: Graduate Algorithms, Spring 2007
Homework 2

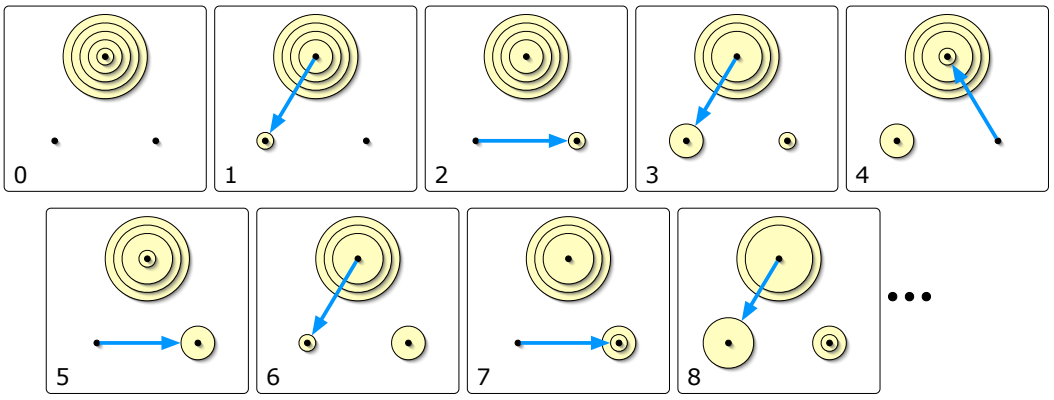
Due Tuesday, February 20, 2007

Remember to submit **separate, individually stapled** solutions to each problem.

As a general rule, a complete full-credit solution to any homework problem should fit into two typeset pages (or five hand-written pages). If your solution is significantly longer than this, you may be including too much detail.

---

1. Consider a restricted variant of the Tower of Hanoi puzzle, where the three needles are arranged in a triangle, and you are required to move each disk *counterclockwise*. Describe an algorithm to move a stack of n disks from one needle to another. *Exactly* how many moves does your algorithm perform? To receive full credit, your algorithm must perform the minimum possible number of moves. *[Hint: Your answer will depend on whether you are moving the stack clockwise or counterclockwise.]*



A top view of the first eight moves in a counterclockwise Towers of Hanoi solution

*2. You find yourself working for The Negation Company ("We Contradict Everything... Not!"), the world's largest producer of multi-bit Boolean inverters. Thanks to a recent mining discovery, the market prices for amphigen and opoterium, the key elements used in AND and OR gates, have plummeted to almost nothing. Unfortunately, the market price of inverton, the essential element required to build NOT gates, has recently risen sharply as natural supplies are almost exhausted. Your boss is counting on you to radically redesign the company's only product in response to these radically new market prices.

   Design a Boolean circuit that inverts $n = 2^k - 1$ bits, using only k NOT gates but *any* number of AND and OR gates. The input to your circuit consists of n bits $x_1, x_2, \ldots, x_n$, and the output consists of n bits $y_1, y_2, \ldots, y_n$, where each output bit y_i is the inverse of the corresponding input bit x_i. *[Hint: Solve the case $k = 2$ first.]*

3. (a) Let $X[1..m]$ and $Y[1..n]$ be two arbitrary arrays. A *common supersequence* of X and Y is another sequence that contains both X and Y as subsequences. Give a simple recursive definition for the function $scs(X, Y)$, which gives the length of the *shortest* common supersequence of X and Y.

   (b) Call a sequence $X[1..n]$ *oscillating* if $X[i] < X[i+1]$ for all even i, and $X[i] > X[i+1]$ for all odd i. Give a simple recursive definition for the function $los(X)$, which gives the length of the longest oscillating subsequence of an arbitrary array X of integers.

   (c) Call a sequence $X[1..n]$ of integers *accelerating* if $2 \cdot X[i] < X[i-1] + X[i+1]$ for all i. Give a simple recursive definition for the function $lxs(X)$, which gives the length of the longest accelerating subsequence of an arbitrary array X of integers.

   Each recursive definition should translate directly into a recursive algorithm, *but you do **not** need to analyze these algorithms.* We are looking for correctness and *simplicity*, not algorithmic efficiency. Not yet, anyway.

4. Describe an algorithm to solve 3SAT in time $O(\phi^n \operatorname{poly}(n))$, where $\phi = (1+\sqrt{5})/2 \approx 1.618034$. *[Hint: Prove that in each recursive call, either you have just eliminated a pure literal, or the formula has a clause with at most two literals. What recurrence leads to this running time?]*

5. (a) Describe an algorithm that determines whether a given set of n integers contains two distinct elements that sum to zero, in $O(n \log n)$ time.

   (b) Describe an algorithm that determines whether a given set of n integers contains *three* distinct elements that sum to zero, in $O(n^2)$ time.

   (c) Now suppose the input set X contains n integers between $-10000n$ and $10000n$. Describe an algorithm that determines whether X contains three ***distinct*** elements that sum to zero, in $O(n \log n)$ time.

   For example, if the input set is $\{-10, -9, -7, -3, 1, 3, 5, 11\}$, your algorithm for part (a) should return TRUE, because $(-3) + 3 = 0$, and your algorithms for parts (b) and (c) should return FALSE, even though $(-10) + 5 + 5 = 0$.

CS 473G: Graduate Algorithms, Spring 2007
Homework 3

Due Friday, March 9, 2007

Remember to submit **separate, individually stapled** solutions to each problem.

As a general rule, a complete, full-credit solution to any homework problem should fit into two typeset pages (or five hand-written pages). If your solution is significantly longer than this, you may be including too much detail.

---

1. (a) Let $X[1..m]$ and $Y[1..n]$ be two arbitrary arrays. A *common supersequence* of X and Y is another sequence that contains both X and Y as subsequences. Describe and analyze an efficient algorithm to compute the function $scs(X, Y)$, which gives the length of the *shortest* common supersequence of X and Y.

   (b) Call a sequence $X[1..n]$ *oscillating* if $X[i] < X[i+1]$ for all even i, and $X[i] > X[i+1]$ for all odd i. Describe and analyze an efficient algorithm to compute the function $los(X)$, which gives the length of the longest oscillating subsequence of an arbitrary array X of integers.

   (c) Call a sequence $X[1..n]$ of integers *accelerating* if $2 \cdot X[i] < X[i-1] + X[i+1]$ for all i. Describe and analyze an efficient algorithm to compute the function $lxs(X)$, which gives the length of the longest accelerating subsequence of an arbitrary array X of integers.

   *[Hint: Use the recurrences you found in Homework 2. You do not need to prove **again** that these recurrences are correct.]*

2. Describe and analyze an algorithm to solve the traveling salesman problem in $O(2^n \operatorname{poly}(n))$ time. Given an undirected n-vertex graph G with weighted edges, your algorithm should return the weight of the lightest Hamiltonian cycle in G (or ∞ if G has no Hamiltonian cycles).

3. Let G be an arbitrary undirected graph. A set of cycles $\{c_1, \ldots, c_k\}$ in G is *redundant* if it is non-empty and every edge in G appears in an even number of c_i's. A set of cycles is *independent* if it contains no redundant subsets. (In particular, the empty set is independent.) A maximal independent set of cycles is called a *cycle basis* for G.

   (a) Let C be any cycle basis for G. Prove that for any cycle γ in G **that is not an element of C**, there is a subset $A \subseteq C$ such that $A \cup \{\gamma\}$ is redundant. In other words, prove that γ is the 'exclusive or' of some subset of basis cycles.

   Solution: The claim follows directly from the definitions. A cycle basis is a *maximal* independent set, so if C is a cycle basis, then for any cycle $\gamma \notin C$, the larger set $C \cup \{\gamma\}$ cannot be an independent set, so it must contain a redundant subset. On the other hand, if C is a basis, then C is independent, so C contains no redundant subsets. Thus, $C \cup \{\gamma\}$ must have a redundant subset B that contains γ. Let $A = B \setminus \{\gamma\}$. ■

(b) Prove that the set of independent cycle sets form a matroid.

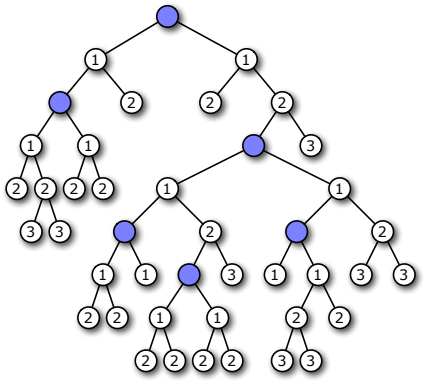
(c) Now suppose each edge of G has a weight. Define the weight of a cycle to be the total weight of its edges, and the weight of a *set* of cycles to be the total weight of all cycles in the set. (Thus, each edge is counted once for every cycle in which it appears.) Describe and analyze an efficient algorithm to compute the minimum-weight cycle basis of G.

4. Let T be a rooted binary tree with n vertices, and let $k \leq n$ be a positive integer. We would like to mark k vertices in T so that every vertex has a nearby marked ancestor. More formally, we define the *clustering cost* of a clustering of any subset K of vertices as

$$\mathit{cost}(K) = \max_v \mathit{cost}(v, K),$$

where the maximum is taken over all vertices v in the tree, and

$$\mathit{cost}(v, K) = \begin{cases} 0 & \text{if } v \in K \\ \infty & \text{if } v \text{ is the root of } T \text{ and } v \notin K \\ 1 + \mathit{cost}(\mathit{parent}(v)) & \text{otherwise} \end{cases}$$

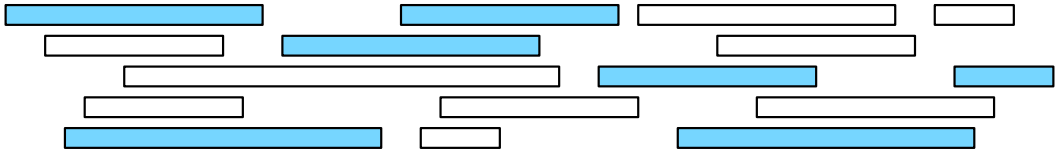


A subset of 5 vertices with clustering cost 3

Describe and analyze a dynamic-programming algorithm to compute the minimum clustering cost of any subset of k vertices in T. For full credit, your algorithm should run in $O(n^2k^2)$ time.

5. Let X be a set of n intervals on the real line. A subset of intervals $Y \subseteq X$ is called a *tiling path* if the intervals in Y cover the intervals in X, that is, any real value that is contained in some interval in X is also contained in some interval in Y. The *size* of a tiling cover is just the number of intervals.

Describe and analyze an algorithm to compute the smallest tiling path of X as quickly as possible. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X. If you use a greedy algorithm, you must prove that it is correct.



A set of intervals. The seven shaded intervals form a tiling path.

CS 473G: Graduate Algorithms, Spring 2007
Homework 4

Due March 29, 2007

Please remember to submit **separate, individually stapled** solutions to each problem.

---

1. Given a graph G with edge weights and an integer k, suppose we wish to partition the the vertices of G into k subsets $S_1, S_2, \ldots, S_k$ so that the sum of the weights of the edges that cross the partition (*i.e.*, have endpoints in different subsets) is as large as possible.

   (a) Describe an efficient $(1 - 1/k)$-approximation algorithm for this problem.

   (b) Now suppose we wish to minimize the sum of the weights of edges that do *not* cross the partition. What approximation ratio does your algorithm from part (a) achieve for the new problem? Justify your answer.

2. In class, we saw a $(3/2)$-approximation algorithm for the metric traveling salesman problem. Here, we consider computing minimum cost Hamiltonian *paths*. Our input consists of a graph G whose edges have weights that satisfy the triangle inequality. Depending upon the problem, we are also given zero, one, or two endpoints.

   (a) If our input includes zero endpoints, describe a $(3/2)$-approximation to the problem of computing a minimum cost Hamiltonian path.

   (b) If our input includes one endpoint u, describe a $(3/2)$-approximation to the problem of computing a minimum cost Hamiltonian path that starts at u.

   (c) If our input includes two endpoints u and v, describe a $(5/3)$-approximation to the problem of computing a minimum cost Hamiltonian path that starts at u and ends at v.

3. Consider the greedy algorithm for metric TSP: start at an arbitrary vertex u, and at each step, travel to the closest unvisited vertex.

   (a) Show that the greedy algorithm for metric TSP is an $O(\log n)$-approximation, where n is the number of vertices. *[Hint: Argue that the kth least expensive edge in the tour output by the greedy algorithm has weight at most $\mathrm{OPT}/(n-k+1)$; try $k = 1$ and $k = 2$ first.]*

   *(b) **[Extra Credit]** Show that the greedy algorithm for metric TSP is no better than an $O(\log n)$-approximation.

4. In class, we saw that the greedy algorithm gives an $O(\log n)$-approximation for vertex cover. Show that our analysis of the greedy algorithm is asymptotically tight by describing, for any positive integer n, an n-vertex graph for which the greedy algorithm produces a vertex cover of size $\Omega(\log n) \cdot \mathrm{OPT}$.

5. Recall the minimum makespan scheduling problem: Given an array $T[1..n]$ of processing times for n jobs, we wish to schedule the jobs on m machines to minimize the time at which the last job terminates. In class, we proved that the greedy scheduling algorithm has an approximation ratio of at most 2.

   (a) Prove that for any set of jobs, the makespan of the greedy assignment is at most $(2-1/m)$ times the makespan of the optimal assignment.

   (b) Describe a set of jobs such that the makespan of the greedy assignment is exactly $(2 - 1/m)$ times the makespan of the optimal assignment.

   (c) Describe an efficient algorithm to solve the minimum makespan scheduling problem *exactly* if every processing time $T[i]$ is a power of two.

CS 473G: Graduate Algorithms, Spring 2007
Homework 5

Due Thursday, April 17, 2007

Please remember to submit **separate, individually stapled** solutions to each problem.

Unless a problem specifically states otherwise, you can assume the function RANDOM(k), which returns an integer chosen independently and uniformly at random from the set $\{1, 2, \ldots, k\}$, in $O(1)$ time. For example, to perform a fair coin flip, you would call RANDOM(2).

---

1. Suppose we want to write an efficient function RANDOMPERMUTATION(n) that returns a permutation of the integers $\langle 1, \ldots, n \rangle$ chosen uniformly at random.

   (a) What is the expected running time of the following RANDOMPERMUTATION algorithm?

```
RandomPermutation(n):
  for i ← 1 to n
      π[i] ← Empty
  for i ← 1 to n
      j ← Random(n)
      while (π[j] ≠ Empty)
          j ← Random(n)
      π[j] ← i
  return π
```

   (b) Consider the following partial implementation of RANDOMPERMUTATION.

```
RandomPermutation(n):
  for i ← 1 to n
      A[i] ← Random(n)
  π ← SomeFunction(A)
  return π
```

   Prove that if the subroutine SOMEFUNCTION is deterministic, then this algorithm cannot be correct. *[Hint: There is a one-line proof.]*

   (c) Describe and analyze an RANDOMPERMUTATION algorithm whose expected worst-case running time is $O(n)$.

   *(d) **[Extra Credit]** Describe and analyze an RANDOMPERMUTATION algorithm that uses only fair coin flips; that is, your algorithm can't call RANDOM(k) with $k > 2$. Your algorithm should run in $O(n \log n)$ time with high probability.

2. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

   - MAKEQUEUE: Return a new priority queue containing the empty set.
   - FINDMIN(Q): Return the smallest element of Q (if any).
   - DELETEMIN(Q): Remove the smallest element in Q (if any).
   - INSERT(Q, x): Insert element x into Q, if it is not already there.
   - DECREASEKEY(Q, x, y): Replace an element $x \in Q$ with a smaller element y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q that contains x.
   - DELETE(Q, x): Delete the element $x \in Q$. The input is a pointer directly to the node in Q that contains x.
   - MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2; this operation destroys Q_1 and Q_2.

   A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```
MELD(Q1, Q2):
  if Q1 is empty, return Q2
  if Q2 is empty, return Q1

  if key(Q1) > key(Q2)
      swap Q1 ↔ Q2

  with probability 1/2
      left(Q1) ← MELD(left(Q1), Q2)
  else
      right(Q1) ← MELD(right(Q1), Q2)

  return Q1
```

   (a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (not just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. *[Hint: How long is a random root-to-leaf path in an n-node binary tree if each left/right choice is made with equal probability?]*

   (b) Prove that MELD(Q_1, Q_2) runs in $O(\log n)$ time with high probability.

   (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)

3. Prove that GUESSMINCUT returns the *second* smallest cut in its input graph with probability $\Omega(1/n^2)$. (The second smallest cut could be significantly larger than the minimum cut.)

4. A *heater* is a sort of dual treap, in which the priorities of the nodes are given by the user, but their search keys are random (specifically, independently and uniformly distributed in the unit interval $[0, 1]$).

   (a) Prove that for any r, the node with the rth smallest *priority* has expected depth $O(\log r)$.

   (b) Prove that an n-node heater has depth $O(\log n)$ with high probability.

   (c) Describe algorithms to perform the operations INSERT and DELETEMIN in a heater. What are the expected worst-case running times of your algorithms?

   You may assume all priorities and keys are distinct. *[Hint: Cite the relevant parts (but only the relevant parts!) of the treap analysis instead of repeating them.]*

5. Let n be an arbitrary positive integer. Describe a set $\mathcal{T}$ of binary search trees with the following properties:

   - Every tree in $\mathcal{T}$ has n nodes, which store the search keys $1, 2, 3, \ldots, n$.
   - For any integer k, if we choose a tree uniformly at random from $\mathcal{T}$, the expected depth of node k in that tree is $O(\log n)$.
   - Every tree in $\mathcal{T}$ has depth $\Omega(\sqrt{n})$.

   (This is why we had to prove via Chernoff bounds that the maximum depth of an n-node treap is $O(\log n)$ with high probability.)

★6. **[Extra Credit]** Recall that F_k denotes the kth Fibonacci number: $F_0 = 0$, $F_1 = 1$, and $F_k = F_{k-1} + F_{k-2}$ for all $k \geq 2$. Suppose we are building a hash table of size $m = F_k$ using the hash function

$$h(x) = (F_{k-1} \cdot x) \bmod F_k$$

   Prove that if the consecutive integers $0, 1, 2, \ldots, F_k - 1$ are inserted in order into an initially empty table, each integer is hashed into one of the largest contiguous empty intervals in the table. Among other things, this implies that there are no collisions.

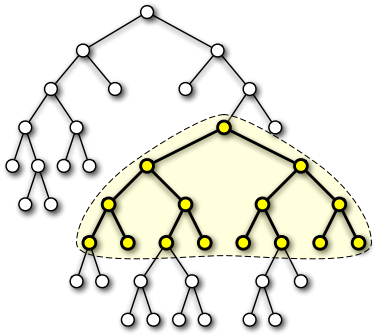
   For example, when $m = 13$, the hash table is filled as follows.

0												
0								1				
0			2					1				
0			2					1			3	
0			2			4		1			3	
0	5		2			4		1			3	
0	5		2			4		1	6		3	
0	5		2	7		4		1	6		3	
0	5		2	7		4		1	6		3	8
0	5		2	7		4	9	1	6		3	8
0	5	10	2	7		4	9	1	6		3	8
0	5	10	2	7		4	9	1	6	11	3	8
0	5	10	2	7	12	4	9	1	6	11	3	8

You have 90 minutes to answer four of these questions.
Write your answers in the separate answer booklet.
You may take the question sheet with you when you leave.

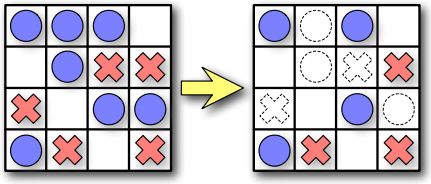
1. Recall that a binary tree is *complete* if every internal node has two children and every leaf has the same depth. An *internal subtree* of a binary tree is a connected subgraph, consisting of a node and some (possibly all or none) of its descendants.

   Describe and analyze an algorithm that computes the depth of the *largest complete internal subtree* of a given n-node binary tree. For full credit, your algorithm should run in $O(n)$ time.

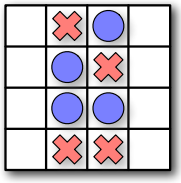


The largest complete internal subtree in this binary tree has depth 3.

2. Consider the following solitaire game. The puzzle consists of an $n \times m$ grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

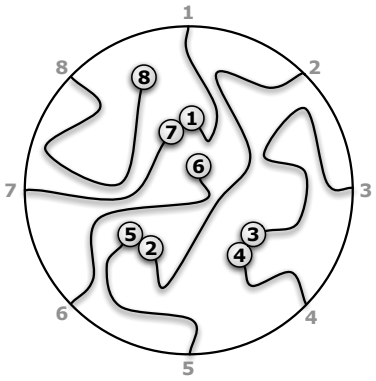
   Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

3. Suppose you are given two sorted arrays $A[1..n]$ and $B[1..n]$ and an integer k. Describe and analyze an algorithm to find the kth largest element in the union of A and B in $O(\log n)$ time. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 21], \qquad B[1..8] = [2, 4, 5, 8, 14, 17, 19, 20], \qquad k = 10,$$

   your algorithm should return 13. You can assume that the arrays contain no duplicates. *[Hint: What can you learn from comparing one element of A to one element of B?]*

4. Every year, as part of its annual meeting, the Antarctican Snail Lovers of Upper Glacierville hold a Round Table Mating Race. Several high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to n. During the race, each snail wanders around the table, leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail, even their own. If two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if the race goes on forever.



The end of a typical Antarctican SLUG race. Snails 6 and 8 never find mates.
The organizers must pay $M[3,4] + M[2,5] + M[1,7]$.

For every pair of snails, the Antarctican SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array $M[1..n, 1..n]$ posted on the wall behind the Round Table, where $M[i,j] = M[j,i]$ is the reward to be paid if snails i and j meet.

Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the array M as input.

5. SUBSETSUM and PARTITION are two closely-related NP-hard problems.

   - SUBSETSUM: Given a set X of positive integers and an integer t, determine whether there is a subset of X whose elements sum to t.
   - PARTITION: Given a set X of positive integers, determine whether X can be partitioned into two subsets whose elements sum to the same value.

   (a) Describe a polynomial-time reduction from SUBSETSUM to PARTITION.

   (b) Describe a polynomial-time reduction from PARTITION to SUBSETSUM.

   Don't forget to ***prove*** that your reductions are correct.

You have 120 minutes to answer four of these questions.
Write your answers in the separate answer booklet.
You may take the question sheet with you when you leave.

1. Consider the following algorithm for finding the smallest element in an unsorted array:

```
RANDOMMIN(A[1..n]):
  min ← ∞
  for i ← 1 to n in random order
      if A[i] < min
          min ← A[i]    (⋆)
  return min
```

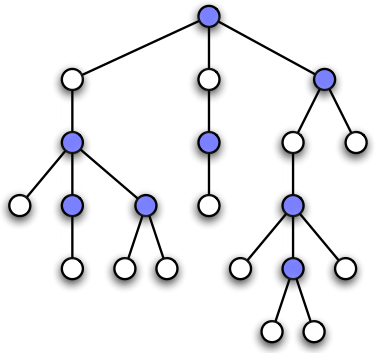
   (a) **[1 pt]** In the worst case, how many times does RANDOMMIN execute line $(\star)$?

   (b) **[3 pts]** What is the probability that line $(\star)$ is executed during the last iteration of the for loop?

   (c) **[6 pts]** What is the *exact* expected number of executions of line $(\star)$? (A correct $\Theta(\,)$ bound is worth 4 points.)

2. Describe and analyze an efficient algorithm to find the size of the smallest vertex cover of a given tree. That is, given a tree T, your algorithm should find the size of the smallest subset C of the vertices, such that every edge in T has at least one endpoint in C.

   The following hint may be helpful. Suppose C is a vertex cover that contains a leaf ℓ. If we remove ℓ from the cover and insert its parent, we get another vertex cover of the same size as C. Thus, there is a minimum vertex cover that includes none of the leaves of T (except when the tree has only one or two vertices).



A tree whose smallest vertex cover has size 8.

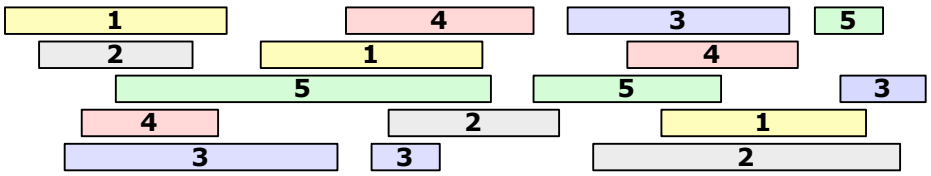
3. A *dominating set* for a graph G is a subset D of the vertices, such that every vertex in G is either in D or has a neighbor in D. The MINDOMINATINGSET problem asks for the size of the smallest dominating set for a given graph.

   Recall the MINSETCOVER problem from lecture. The input consists of a *ground set* X and a collection of subsets $S_1, S_2, \ldots, S_k \subseteq X$. The problem is to find the minimum number of subsets S_i that completely cover X. This problem is NP-hard, because it is a generalization of the vertex cover problem.

   (a) **[7 pts]** Describe a polynomial-time reduction from MINDOMINATINGSET to MINSETCOVER.

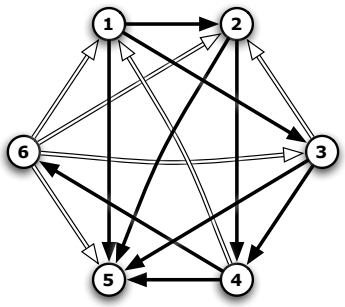
   (b) **[3 pts]** Describe a polynomial-time $O(\log n)$-approximation algorithm for MINDOMINATINGSET. *[Hint: There is a two-line solution.]*

4. Let X be a set of n intervals on the real line. A *proper coloring* of X assigns a color to each interval, so that any two overlapping intervals are assigned different colors. Describe and analyze an efficient algorithm to compute the minimum number of colors needed to properly color X. Assume that your input consists of two arrays $L[1..n]$ and $R[1..n]$, where $L[i]$ and $R[i]$ are the left and right endpoints of the ith interval. As usual, if you use a greedy algorithm, you must prove that it is correct.



A proper coloring of a set of intervals using five colors.

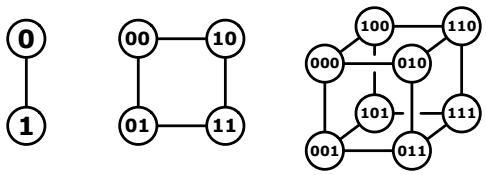
5. The *linear arrangement problem* asks, given an n-vertex directed graph as input, for an ordering $v_1, v_2, \ldots, v_n$ of the vertices that maximizes the number of *forward edges*: directed edges $v_i \to v_j$ such that $i < j$. Describe and analyze an efficient 2-approximation algorithm for this problem.



A directed graph with six vertices with nine forward edges (black) and six backward edges (white)

You have 180 minutes to answer six of these questions.
Write your answers in the separate answer booklet.

1. The d-dimensional hypercube is the graph defined as follows. There are 2^d vertices, each labeled with a different string of d bits. Two vertices are joined by an edge if and only if their labels differ in exactly one bit.



The 1-dimensional, 2-dimensional, and 3-dimensional hypercubes.

   (a) *[8 pts]* Recall that a Hamiltonian cycle is a closed walk that visits each vertex in a graph exactly once. ***Prove*** that for all $d \geq 2$, the d-dimensional hypercube has a Hamiltonian cycle.

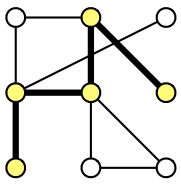
   (b) *[2 pts]* Recall that an Eulerian circuit is a closed walk that traverses each edge in a graph exactly once. Which hypercubes have an Eulerian circuit? *[Hint: This is very easy.]*

2. The University of Southern North Dakota at Hoople has hired you to write an algorithm to schedule their final exams. Each semester, USNDH offers n different classes. There are r different rooms on campus and t different time slots in which exams can be offered. You are given two arrays $E[1..n]$ and $S[1..r]$, where $E[i]$ is the number of students enrolled in the ith class, and $S[j]$ is the number of seats in the jth room. At most one final exam can be held in each room during each time slot. Class i can hold its final exam in room j only if $E[i] < S[j]$.

   Describe and analyze an efficient algorithm to assign a room and a time slot to each class (or report correctly that no such assignment is possible).

3. What is the *exact* expected number of leaves in an n-node treap? (The answer is obviously at most n, so no partial credit for writing "$O(n)$".) *[Hint: What is the probably that the node with the kth largest key is a leaf?]*

4. A *tonian path* in a graph G is a simple path in G that visits more than half of the vertices of G. (Intuitively, a tonian path is "most of a Hamiltonian path".) ***Prove*** that it is NP-hard to determine whether or not a given graph contains a tonian path.



A tonian path.

5. A *palindrome* is a string that reads the same forwards and backwards, like x, pop, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. Any string can be broken into sequence of palindromes. For example, the string bubbaseesabanana ('Bubba sees a banana.') can be broken into palindromes in several different ways; for example,

$$\mathsf{bub} + \mathsf{baseesab} + \mathsf{anana}$$
$$\mathsf{b} + \mathsf{u} + \mathsf{bb} + \mathsf{a} + \mathsf{sees} + \mathsf{aba} + \mathsf{nan} + \mathsf{a}$$
$$\mathsf{b} + \mathsf{u} + \mathsf{bb} + \mathsf{a} + \mathsf{sees} + \mathsf{a} + \mathsf{b} + \mathsf{anana}$$
$$\mathsf{b} + \mathsf{u} + \mathsf{b} + \mathsf{b} + \mathsf{a} + \mathsf{s} + \mathsf{e} + \mathsf{e} + \mathsf{s} + \mathsf{a} + \mathsf{b} + \mathsf{a} + \mathsf{n} + \mathsf{a} + \mathsf{n} + \mathsf{a}$$

Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string bubbaseesabanana, your algorithm would return the integer 3.

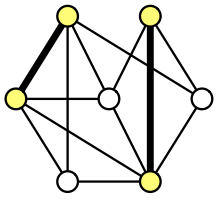
6. Consider the following modification of the 2-approximation algorithm for minimum vertex cover that we saw in class. The only real change is that we compute a set of edges instead of a set of vertices.

```
APPROXMINMAXMATCHING(G):
  M ← ∅
  while G has at least one edge
      (u, v) ← any edge in G
      G ← G \ {u, v}
      M ← M ∪ {(u, v)}
  return M
```

   (a) *[2 pts]* ***Prove*** that the output graph M is a *matching*—no pair of edges in M share a common vertex.

   (b) *[2 pts]* ***Prove*** that M is a *maximal* matching—M is not a proper subgraph of another matching in G.

   (c) *[6 pts]* ***Prove*** that M contains at most twice as many edges as the *smallest* maximal matching in G.



The smallest maximal matching in a graph.

7. Recall that in the standard maximum-flow problem, the flow through an edge is limited by the capacity of that edge, but there is no limit on how much flow can pass through a vertex. Suppose each vertex v in our input graph has a capacity $c(v)$ that limits the total flow through v, in addition to the usual edge capacities. Describe and analyze an efficient algorithm to compute the maximum (s, t)-flow with these additional constraints. *[Hint: Reduce to the standard max-flow problem.]*

CS 573: Graduate Algorithms, Fall 2008
Homework 0

Due **in class** at 12:30pm, Wednesday, September 3, 2008

Name:	
Net ID:	Alias:

☐ I understand the course policies.

- Each student must submit their own solutions for this homework. For all future homeworks, groups of up to three students may submit a single, common solution.

- Neatly print your full name, your NetID, and an alias of your choice in the boxes above, and staple this page to the front of your homework solutions. We will list homework and exam grades on the course web site by alias.

  Federal privacy law and university policy forbid us from publishing your grades, even anonymously, without your explicit written permission. ***By providing an alias, you grant us permission to list your grades on the course web site. If you do not provide an alias, your grades will not be listed.*** For privacy reasons, your alias should not resemble your name, your NetID, your university ID number, or (God forbid) your Social Security number.

- Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. Once you understand the policies, please check the box at the top of this page. In particular:

  - You may use any source at your disposal—paper, electronic, or human—but you ***must*** write your solutions in your own words, and you ***must*** cite every source that you use.
  - Unless explicitly stated otherwise, ***every*** homework problem requires a proof.
  - Answering "I don't know" to any homework or exam problem is worth 25% partial credit.
  - Algorithms or proofs containing phrases like "and so on" or "repeat this for all n", instead of an explicit loop, recursion, or induction, will receive 0 points.

- This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, discrete probability, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** The early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.

---

1. (a) **[5 pts]** Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Assume reasonable but nontrivial base cases. If your solution requires a particular base case, say so.

$$A(n) = 4A(n/8) + \sqrt{n}$$

$$B(n) = B(n/3) + 2B(n/4) + B(n/6) + n$$

$$C(n) = 6C(n-1) - 9C(n-2)$$

$$D(n) = \max_{n/3<k<2n/3} \Big(D(k) + D(n-k) + n \Big)$$

$$E(n) = \Big(E(\sqrt{n}) \Big)^2 \cdot n$$

(b) **[5 pts]** Sort the functions in the box from asymptotically smallest to asymptotically largest, indicating ties if there are any. **Do *not* turn in proofs**—just a sorted list of 16 functions—but you should do them anyway, just for practice. We use the notation $\lg n = \log_2 n$.

n	$\lg n$	$\sqrt{n}$	3^n
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$3^{\sqrt{n}}$	$\sqrt{3^n}$
$3^{\lg n}$	$\lg(3^n)$	$3^{\lg \sqrt{n}}$	$3\sqrt{\lg n}$
$\sqrt{3^{\lg n}}$	$\lg(3^{\sqrt{n}})$	$\lg \sqrt{3^n}$	$\sqrt{\lg(3^n)}$

2. Describe and analyze a data structure that stores set of n records, each with a numerical *key* and a numerical *priority*, such that the following operation can be performed quickly:

   - RANGETOP(a, z) : return the highest-priority record whose key is between a and z.

   For example, if the (key, priority) pairs are

$$(3,1),\ (4,9),\ (9,2),\ (6,3),\ (5,8),\ (7,5),\ (1,10),\ (0,7),$$

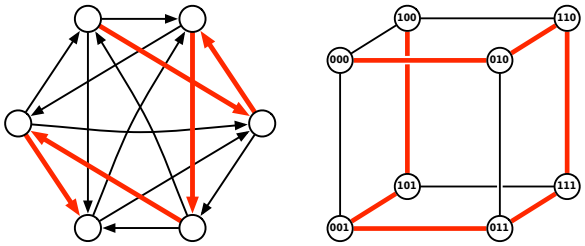
   then RANGETOP$(2,8)$ would return the record with key 4 and priority 9 (the second in the list).

   Analyze both the size of your data structure and the running time of your RANGETOP algorithm. For full credit, your space and time bounds must both be as small as possible. You may assume that no two records have equal keys or equal priorities, and that no record has a or z as its key. *[Hint: How would you compute the number of keys between a and z? How would you solve the problem if you knew that a is always $-\infty$?]*

3. A *Hamiltonian path* in G is a path that visits every vertex of G exactly once. In this problem, you are asked to prove that two classes of graphs always contain a Hamiltonian path.

   (a) **[5 pts]** A *tournament* is a directed graph with exactly one edge between each pair of vertices. (Think of the nodes in a round-robin tournament, where edges represent games, and each edge points from the loser to the winner.) Prove that every tournament contains a *directed* Hamiltonian path.

   (b) **[5 pts]** Let d be an arbitrary non-negative integer. The d-dimensional *hypercube* is the graph defined as follows. There are 2^d vertices, each labeled with a different string of d bits. Two vertices are joined by an edge if and only if their labels differ in exactly one bit. Prove that the d-dimensional hypercube contains a Hamiltonian path.



Hamiltonian paths in a 6-node tournament and a 3-dimensional hypercube.

4. Penn and Teller agree to play the following game. Penn shuffles a standard deck[1] of playing cards so that every permutation is equally likely. Then Teller draws cards from the deck, one at a time without replacement, until he draws the three of clubs (3♣), at which point the remaining undrawn cards instantly burst into flames.

   The first time Teller draws a card from the deck, he gives it to Penn. From then on, until the game ends, whenever Teller draws a card whose value is smaller than the last card he gave to Penn, he gives the new card to Penn.[2] To make the rules unambiguous, they agree beforehand that $A = 1$, $J = 11$, $Q = 12$, and $K = 13$.

   (a) What is the expected number of cards that Teller draws?

   (b) What is the expected *maximum* value among the cards Teller gives to Penn?

   (c) What is the expected *minimum* value among the cards Teller gives to Penn?

   (d) What is the expected number of cards that Teller gives to Penn?

   Full credit will be given only for *exact* answers (with correct proofs, of course). *[Hint: Let* $13 = n$*.]*

[1] In a standard deck of playing cards, each card has a *value* in the set $\{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\}$ and a *suit* in the set $\{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\}$; each of the 52 possible suit-value pairs appears in the deck exactly once. Actually, to make the game more interesting, Penn and Teller normally use razor-sharp ninja throwing cards.

[2] Specifically, he hurls them from the opposite side of the stage directly into the back of Penn's right hand. Ouch!

5. (a) The ***Fibonacci numbers F_n*** are defined by the recurrence $F_n = F_{n-1} + F_{n-2}$, with base cases $F_0 = 0$ and $F_1 = 1$. Here are the first several Fibonacci numbers:

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
0	1	1	2	3	5	8	13	21	34	55

Prove that any non-negative integer can be written as the sum of distinct, non-consecutive Fibonacci numbers. That is, if the Fibonacci number F_i appears in the sum, it appears exactly once, and its neighbors F_{i-1} and F_{i+1} do not appear at all. For example:

$$17 = F_7 + F_4 + F_2, \qquad 42 = F_9 + F_6, \qquad 54 = F_9 + F_7 + F_5 + F_3 + F_1.$$

(b) The Fibonacci sequence can be extended backward to negative indices by rearranging the defining recurrence: $F_n = F_{n+2} - F_{n+1}$. Here are the first several negative-index Fibonacci numbers:

F_{-10}	F_{-9}	F_{-8}	F_{-7}	F_{-6}	F_{-5}	F_{-4}	F_{-3}	F_{-2}	F_{-1}
−55	34	−21	13	−8	5	−3	2	−1	1

Prove that $F_{-n} = -F_n$ if and only if n is even.

(c) Prove that *any* integer—positive, negative, or zero—can be written as the sum of distinct, non-consecutive Fibonacci numbers *with negative indices*. For example:

$$17 = F_{-7} + F_{-5} + F_{-2}, \qquad -42 = F_{-10} + F_{-7}, \qquad 54 = F_{-9} + F_{-7} + F_{-5} + F_{-3} + F_{-1}.$$

[Hint: Zero is both non-negative and even. Don't use weak induction!]

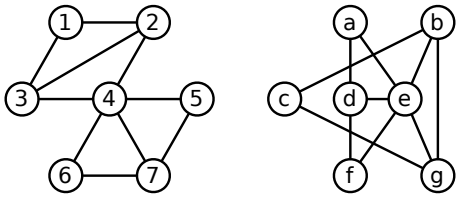
CS 573: Graduate Algorithms, Fall 2008
Homework 1

Due at 11:59:59pm, Wednesday, September 17, 2008

For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and alias (if any) of every group member on the first page of your submission.

---

1. Two graphs are said to be ***isomorphic*** if one can be transformed into the other just by relabeling the vertices. For example, the graphs shown below are isomorphic; the left graph can be transformed into the right graph by the relabeling $(1, 2, 3, 4, 5, 6, 7) \mapsto (c, g, b, e, a, f, d)$.



Two isomorphic graphs.

Consider the following related decision problems:

- GRAPHISOMORPHISM: Given two graphs G and H, determine whether G and H are isomorphic.
- EVENGRAPHISOMORPHISM: Given two graphs G and H, such that every vertex in G and H has even degree, determine whether G and H are isomorphic.
- SUBGRAPHISOMORPHISM: Given two graphs G and H, determine whether G is isomorphic to a subgraph of H.

(a) Describe a polynomial-time reduction from EVENGRAPHISOMORPHISM to GRAPHISOMORPHISM.

(b) Describe a polynomial-time reduction from GRAPHISOMORPHISM to EVENGRAPHISOMORPHISM.

(c) Describe a polynomial-time reduction from GRAPHISOMORPHISM to SUBGRAPHISOMORPHISM.

(d) Prove that SUBGRAPHISOMORPHISM is NP-complete.

(e) What can you conclude about the NP-hardness of GRAPHISOMORPHISM? Justify your answer.

[Hint: These are all easy!]

2. (a) A *tonian path* in a graph G is a path that goes through at least half of the vertices of G. Show that determining whether a graph has a tonian path is NP-complete.

   (b) A *tonian cycle* in a graph G is a cycle that goes through at least half of the vertices of G. Show that determining whether a graph has a tonian cycle is NP-complete. *[Hint: Use part (a).]*

3. The following variant of 3SAT is called either EXACT3SAT or 1IN3SAT, depending on who you ask.

   Given a boolean formula in conjunctive normal form with 3 literals per clause, is there an assignment that makes ***exactly one*** literal in each clause TRUE?

   Prove that this problem is NP-complete.

4. Suppose you are given a magic black box that can solve the MAXCLIQUE problem ***in polynomial time***. That is, given an arbitrary graph G as input, the magic black box computes the number of vertices in the largest complete subgraph of G. Describe and analyze a ***polynomial-time*** algorithm that computes, given an arbitrary graph G, a complete subgraph of G of maximum size, using this magic black box as a subroutine.

5. A boolean formula in ***exclusive-or conjunctive normal form*** (XCNF) is a conjunction (AND) of several *clauses*, each of which is the *exclusive*-or of several literals. The XCNF-SAT problem asks whether a given XCNF boolean formula is satisfiable. Either describe a polynomial-time algorithm for XCNF-SAT or prove that it is NP-complete.

*6. ***[Extra credit]*** Describe and analyze an algorithm to solve 3SAT in $O(\phi^n \operatorname{poly}(n))$ time, where $\phi = (1 + \sqrt{5})/2 \approx 1.618034$. *[Hint: Prove that in each recursive call, either you have just eliminated a pure literal, or the formula has a clause with at most two literals. What recurrence leads to this running time?]*

[0] In class, I asserted that Gaussian elimination was probably discovered by Gauss, in violation of Stigler's Law of Eponymy. In fact, a method very similar to Gaussian elimination appears in the Chinese treatise *Nine Chapters on the Mathematical Art*, believed to have been finalized before 100AD, although some material may predate emperor Qin Shi Huang's infamous 'burning of the books and burial of the scholars' in 213BC. The great Chinese mathematician Liu Hui, in his 3rd-century commentary on *Nine Chapters*, compares two variants of the method and counts the number of arithmetic operations used by each, with the explicit goal of find the more efficient method. This is arguably the earliest recorded *analysis* of any algorithm.

CS 573: Graduate Algorithms, Fall 2008
Homework 2

Due at 11:59:59pm, Wednesday, October 1, 2008

- For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and alias (if any) of every group member on the first page of your submission.
- We will use the following point breakdown to grade dynamic programming algorithms: 60% for a correct recurrence (including base cases), 20% for correct running time analysis of the memoized recurrence, 10% for correctly transforming the memoized recursion into an iterative algorithm.
- A greedy algorithm *must* be accompanied by a proof of correctness in order to receive *any* credit.

---

1. (a) Let $X[1..m]$ and $Y[1..n]$ be two arbitrary arrays of numbers. A ***common supersequence*** of X and Y is another sequence that contains both X and Y as subsequences. Describe and analyze an efficient algorithm to compute the function $scs(X, Y)$, which gives the length of the *shortest* common supersequence of X and Y.

   (b) Call a sequence $X[1..n]$ of numbers ***oscillating*** if $X[i] < X[i+1]$ for all even i, and $X[i] > X[i+1]$ for all odd i. Describe and analyze an efficient algorithm to compute the function $los(X)$, which gives the length of the longest oscillating subsequence of an arbitrary array X of integers.

   (c) Call a sequence $X[1..n]$ of numbers ***accelerating*** if $2 \cdot X[i] < X[i-1] + X[i+1]$ for all i. Describe and analyze an efficient algorithm to compute the function $lxs(X)$, which gives the length of the longest accelerating subsequence of an arbitrary array X of integers.

2. A *palindrome* is a string that reads the same forwards and backwards, like `x`, `pop`, `noon`, `redivider`, or `amanaplanacatahamayakayamahatacanalpanama`. Any string can be broken into sequence of palindromes. For example, the string `bubbaseesabanana` ('Bubba sees a banana.') can be broken into palindromes in several different ways; for example:

   ```
   bub + baseesab + anana
   b + u + bb + a + sees + aba + nan + a
   b + u + bb + a + sees + a + b + anana
   b + u + b + b + a + s + e + e + s + a + b + a + n + a + n + a
   ```

   Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string `bubbaseesabanana`, your algorithm would return the integer 3.

3. Describe and analyze an algorithm to solve the traveling salesman problem in $O(2^n \operatorname{poly}(n))$ time. Given an undirected n-vertex graph G with weighted edges, your algorithm should return the weight of the lightest Hamiltonian cycle in G, or ∞ if G has no Hamiltonian cycles. *[Hint: The obvious recursive algorithm takes $O(n!)$ time.]*

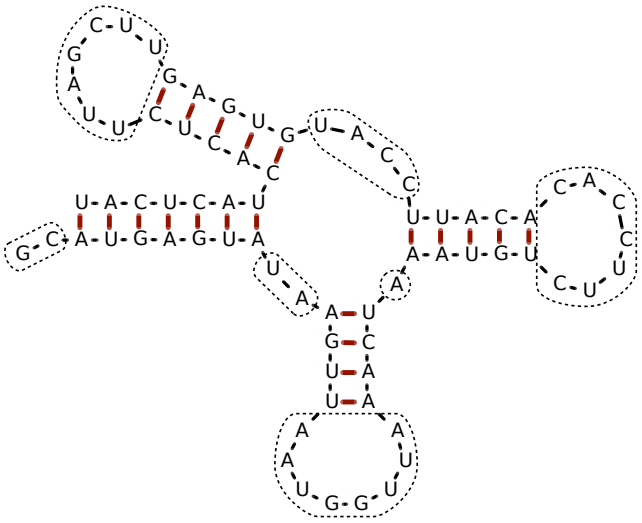
4. Ribonucleic acid (RNA) molecules are long chains of millions of nucleotides or *bases* of four different types: adenine (A), cytosine (C), guanine (G), and uracil (U). The *sequence* of an RNA molecule is a string $b[1..n]$, where each character $b[i] \in \{A, C, G, U\}$ corresponds to a base. In addition to the chemical bonds between adjacent bases in the sequence, hydrogen bonds can form between certain pairs of bases. The set of bonded base pairs is called the *secondary structure* of the RNA molecule.

   We say that two base pairs (i, j) and (i', j') with $i < j$ and $i' < j'$ ***overlap*** if $i < i' < j < j'$ or $i' < i < j' < j$. In practice, most base pairs are non-overlapping. Overlapping base pairs create so-called *pseudoknots* in the secondary structure, which are essential for some RNA functions, but are more difficult to predict.

   Suppose we want to predict the best possible secondary structure for a given RNA sequence. We will adopt a drastically simplified model of secondary structure:

   - Each base can be paired with at most one other base.
   - Only A-U pairs and C-G pairs can bond.
   - Pairs of the form $(i, i+1)$ and $(i, i+2)$ cannot bond.
   - Overlapping base pairs cannot bond.

   The last restriction allows us to visualize RNA secondary structure as a sort of fat tree.



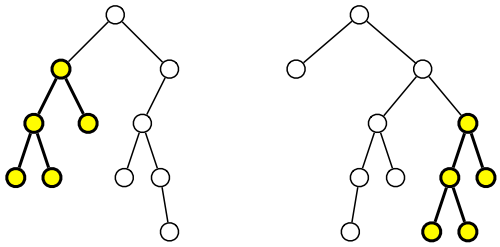
Example RNA secondary structure with 21 base pairs, indicated by heavy red lines.
Gaps are indicated by dotted curves. This structure has score $2^2 + 2^2 + 8^2 + 1^2 + 7^2 + 4^2 + 7^2 = 187$

   (a) Describe and analyze an algorithm that computes the maximum possible *number* of base pairs in a secondary structure for a given RNA sequence.

   (b) A *gap* in a secondary structure is a maximal substring of unpaired bases. Large gaps lead to chemical instabilities, so secondary structures with smaller gaps are more likely. To account for this preference, let's define the *score* of a secondary structure to be the sum of the *squares* of the gap lengths.[1] Describe and analyze an algorithm that computes the minimum possible score of a secondary structure for a given RNA sequence.

[1] This score function has absolutely no connection to reality; I just made it up. Real RNA structure prediction requires much more complicated scoring functions.

5. A *subtree* of a (rooted, ordered) binary tree T consists of a node and all its descendants. Design and analyze an efficient algorithm to compute the ***largest common subtree*** of two given binary trees T_1 and T_2; this is the largest subtree of T_1 that is isomorphic to a subtree in T_2. The contents of the nodes are irrelevant; we are only interested in matching the underlying combinatorial structure.



Two binary trees, with their largest common subtree emphasized

*6. ***[Extra credit]*** Let $D[1..n]$ be an array of digits, each an integer between 0 and 9. A ***digital subsequence*** of D is an sequence of positive integers composed in the usual way from disjoint substrings of D. For example, $3, 4, 5, 6, 23, 38, 62, 64, 83, 279$ is an increasing digital subsequence of the first several digits of π:

$$\boxed{3}, 1, \boxed{4}, 1, \boxed{5}, 9, \boxed{6}, \boxed{2,3}, 4, \boxed{3,8}, 4, \boxed{6,2}, \boxed{6,4}, 3, 3, \boxed{8,3}, \boxed{2,7,9}$$

The *length* of a digital subsequence is the number of integers it contains, *not* the number of digits; the previous example has length 10.

Describe and analyze an efficient algorithm to compute the longest increasing digital subsequence of D. *[Hint: Be careful about your computational assumptions. How long does it take to compare two k-digit numbers?]*

CS 573: Graduate Algorithms, Fall 2008
Homework 3

Due at 11:59:59pm, Wednesday, October 22, 2008

- Groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and the HW0 alias (if any) of every group member on the first page of your submission.

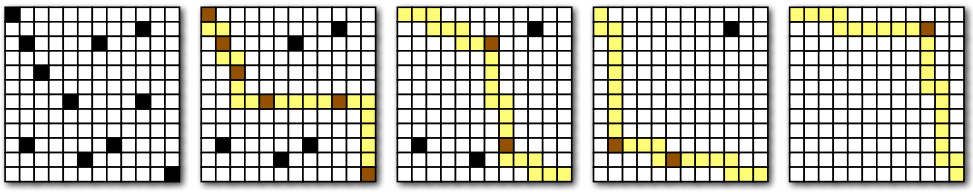
---

1. Consider an $n \times n$ grid, some of whose cells are marked. A *monotone* path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell. We want to compute the minimum number of monotone paths that cover all marked cells. The input to our problem is an array $M[1..n, 1..n]$ of booleans, where $M[i, j] = \text{TRUE}$ if and only if cell (i, j) is marked.

   One of your friends suggests the following greedy strategy:

   - Find (somehow) one "good" path π that covers the maximum number of marked cells.
   - Unmark the cells covered by π.
   - If any cells are still marked, recursively cover them.

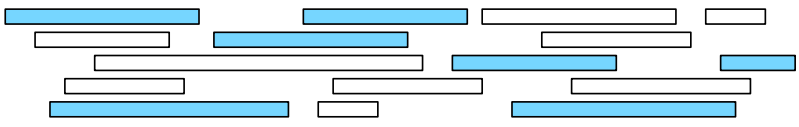
   Does this greedy strategy always compute an optimal solution? If yes, give a proof. If no, give a counterexample.



Greedily covering the marked cells in a grid with four monotone paths.

2. Let X be a set of n intervals on the real line. A subset of intervals $Y \subseteq X$ is called a ***tiling path*** if the intervals in Y cover the intervals in X, that is, any real value that is contained in some interval in X is also contained in some interval in Y. The *size* of a tiling path is just the number of intervals.

   Describe and analyze an algorithm to compute the smallest tiling path of X as quickly as possible. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X. If you use a greedy algorithm, you must prove that it is correct.



A set of intervals. The seven shaded intervals form a tiling path.

3. Given a graph G with edge weights and an integer k, suppose we wish to partition the vertices of G into k subsets $S_1, S_2, \ldots, S_k$ so that the sum of the weights of the edges that cross the partition (*i.e.*, that have endpoints in different subsets) is as large as possible.

   (a) Describe an efficient $(1-1/k)$-approximation algorithm for this problem. *[Hint: Solve the special case $k = 2$ first.]*

   (b) Now suppose we wish to minimize the sum of the weights of edges that do *not* cross the partition. What approximation ratio does your algorithm from part (a) achieve for this new problem? Justify your answer.

4. Consider the following heuristic for constructing a vertex cover of a connected graph G: ***Return the set of all non-leaf nodes of any depth-first spanning tree.*** (Recall that a depth-first spanning tree is a *rooted* tree; the root is not considered a leaf, even if it has only one neighbor in the tree.)

   (a) Prove that this heuristic returns a vertex cover of G.

   (b) Prove that this heuristic returns a 2-approximation to the minimum vertex cover of G.

   (c) Prove that for any $\varepsilon > 0$, there is a graph for which this heuristic returns a vertex cover of size at least $(2-\varepsilon) \cdot OPT$.

5. Consider the following greedy approximation algorithm to find a vertex cover in a graph:

```
GreedyVertexCover(G):
  C ← ∅
  while G has at least one edge
      v ← vertex in G with maximum degree
      G ← G \ v
      C ← C ∪ v
  return C
```

   In class we proved that the approximation ratio of this algorithm is $O(\log n)$; your task is to prove a matching lower bound. Specifically, for any positive integer n, describe an n-vertex graph G such that GreedyVertexCover(G) returns a vertex cover that is $\Omega(\log n)$ times larger than optimal. *[Hint: $H_n = \Omega(\log n)$.]*

*6. ***[Extra credit]*** Consider the greedy algorithm for metric TSP: Start at an arbitrary vertex u, and at each step, travel to the closest unvisited vertex.

   (a) Prove that this greedy algorithm is an $O(\log n)$-approximation algorithm, where n is the number of vertices. *[Hint: Show that the kth least expensive edge in the tour output by the greedy algorithm has weight at most $OPT/(n-k+1)$; try $k = 1$ and $k = 2$ first.]*

   *(b) Prove that the greedy algorithm for metric TSP is no better than an $O(\log n)$-approximation. That is, describe an infinite family of weighted graphs that satisfy the triangle inequality, such that the greedy algorithm returns a cycle whose length is $\Omega(\log n)$ times the optimal TSP tour.

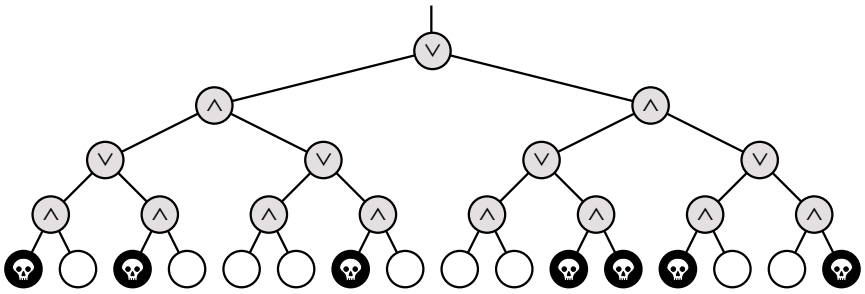
CS 573: Graduate Algorithms, Fall 2008
Homework 4

Due at 11:59:59pm, Wednesday, October 31, 2008

- Groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and the HW0 alias (if any) of every group member on the first page of your submission.
- Unless a problem explicitly states otherwise, you can assume the existence of a function RANDOM(k), which returns an integer uniformly distributed in the range $\{1, 2, \ldots, k\}$ in $O(1)$ time; the argument k must be a positive integer. For example, RANDOM(2) simulates a fair coin flip, and RANDOM(1) *always* returns 1.

---

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black leaves stand represent TRUE and FALSE inputs, respectively. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

   (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

   (b) Unfortunately, Death won't let you even look at every node in the tree. Describe and analyze a *randomized* algorithm that determines whether you can win in $O(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*

   (c) ***[Extra credit]*** Describe and analyze a *randomized* algorithm that determines whether you can win in $O(c^n)$ expected time, for some constant $c < 3$. *[Hint: You may not need to change your algorithm at all.]*

2. Consider the following randomized algorithm for choosing the largest bolt. Draw a bolt uniformly at random from the set of n bolts, and draw a nut uniformly at random from the set of n nuts. If the bolt is smaller than the nut, discard the bolt, draw a new bolt uniformly at random from the unchosen bolts, and repeat. Otherwise, discard the nut, draw a new nut uniformly at random from the unchosen nuts, and repeat. Stop either when every nut has been discarded, or every bolt except the one in your hand has been discarded.

   What is the *exact* expected number of nut-bolt tests performed by this algorithm? Prove your answer is correct. *[Hint: What is the expected number of unchosen nuts and bolts when the algorithm terminates?]*

3. (a) Prove that the expected number of proper descendants of any node in a treap is *exactly* equal to the expected depth of that node.

   (b) Why doesn't the Chernoff-bound argument for depth imply that, with high probability, *every* node in a treap has $O(\log n)$ descendants? The conclusion is obviously bogus—every n-node treap has one node with exactly n descendants!—but what is the flaw in the argument?

   (c) What is the expected number of leaves in an n-node treap? *[Hint: What is the probability that in an n-node treap, the node with kth smallest search key is a leaf?]*

4. The following randomized algorithm, sometimes called "one-armed quicksort", selects the rth smallest element in an unsorted array $A[1..n]$. For example, to find the smallest element, you would call RANDOMSELECT$(A, 1)$; to find the median element, you would call RANDOMSELECT$(A, \lfloor n/2 \rfloor)$. The subroutine PARTITION$(A[1..n], p)$ splits the array into three parts by comparing the pivot element $A[p]$ to every other element of the array, using $n-1$ comparisons altogether, and returns the new index of the pivot element.

```
RandomSelect(A[1..n], r):
  k ← Partition(A[1..n], Random(n))
  if r < k
      return RandomSelect(A[1..k-1], r)
  else if r > k
      return RandomSelect(A[k+1..n], r-k)
  else
      return A[k]
```

   (a) State a recurrence for the expected running time of RANDOMSELECT, as a function of n and r.

   (b) What is the *exact* probability that RANDOMSELECT compares the ith smallest and jth smallest elements in the input array? The correct answer is a simple function of i, j, and r. *[Hint: Check your answer by trying a few small examples.]*

   (c) Show that for any n and r, the expected running time of RANDOMSELECT is $\Theta(n)$. You can use either the recurrence from part (a) or the probabilities from part (b).

   (d) ***[Extra Credit]*** Find the *exact* expected number of comparisons executed by RANDOMSELECT, as a function of n and r.

5. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

   - MAKEQUEUE: Return a new priority queue containing the empty set.
   - FINDMIN(Q): Return the smallest element of Q (if any).
   - DELETEMIN(Q): Remove the smallest element in Q (if any).
   - INSERT(Q, x): Insert element x into Q, if it is not already there.
   - DECREASEKEY(Q, x, y): Replace an element $x \in Q$ with a smaller key y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q containing x.
   - DELETE(Q, x): Delete the element $x \in Q$. The input is a pointer directly to the node in Q containing x.
   - MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2; this operation destroys Q_1 and Q_2.

   A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```
MELD(Q1, Q2):
  if Q1 is empty return Q2
  if Q2 is empty return Q1

  if key(Q1) > key(Q2)
      swap Q1 ↔ Q2

  with probability 1/2
      left(Q1) ← MELD(left(Q1), Q2)
  else
      right(Q1) ← MELD(right(Q1), Q2)

  return Q1
```

   (a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (*not* just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where n is the total number of nodes in both trees. *[Hint: How long is a random root-to-leaf path in an n-node binary tree if each left/right choice is made with equal probability?]*

   (b) Prove that MELD(Q_1, Q_2) runs in $O(\log n)$ time with high probability. *[Hint: You don't need Chernoff bounds, but you might use the identity $\binom{ck}{k} \le (ce)^k$.]*

   (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)

*6. ***[Extra credit]*** In the usual theoretical presentation of treaps, the priorities are random real numbers chosen uniformly from the interval $[0,1]$, but in practice, computers only have access to random *bits*. This problem asks you to analyze a modification of treaps that takes this limitation into account.

Suppose the priority of a node v is abstractly represented as an infinite sequence $\pi_v[1..\infty]$ of random bits, which is interpreted as the rational number

$$\mathit{priority}(v) = \sum_{i=1}^{\infty} \pi_v[i] \cdot 2^{-i}.$$

However, only a finite number ℓ_v of these bits are actually known at any given time. When a node v is first created, *none* of the priority bits are known: $\ell_v = 0$. We generate (or 'reveal') new random bits only when they are necessary to compare priorities. The following algorithm compares the priorities of any two nodes in $O(1)$ expected time:

```
LargerPriority(v, w):
  for i ← 1 to ∞
      if i > ℓ_v
          ℓ_v ← i;  π_v[i] ← RandomBit
      if i > ℓ_w
          ℓ_w ← i;  π_w[i] ← RandomBit
      if π_v[i] > π_w[i]
          return v
      else if π_v[i] < π_w[i]
          return w
```

Suppose we insert n items one at a time into an initially empty treap. Let $L = \sum_v \ell_v$ denote the total number of random bits generated by calls to LargerPriority during these insertions.

(a) Prove that $E[L] = \Theta(n)$.

(b) Prove that $E[\ell_v] = \Theta(1)$ for any node v. *[Hint: This is equivalent to part (a). Why?]*

(c) Prove that $E[\ell_{\text{root}}] = \Theta(\log n)$. *[Hint: Why doesn't this contradict part (b)?]*

CS 573: Graduate Algorithms, Fall 2008
Homework 5

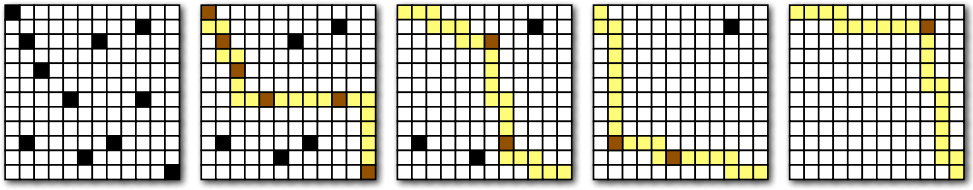
Due at 11:59:59pm, Wednesday, November 19, 2008

- Groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and the HW0 alias (if any) of every group member on the first page of your submission.

---

1. Recall the following problem from Homework 3: You are given an $n \times n$ grid, some of whose cells are marked; the grid is represented by an array $M[1..n, 1..n]$ of booleans, where $M[i, j] = \text{TRUE}$ if and only if cell (i, j) is marked. A *monotone* path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell.

   Describe and analyze an efficient algorithm to compute the smallest set of monotone paths that covers every marked cell.



Greedily covering the marked cells in a grid with four monotone paths.

2. Suppose we are given a directed graph $G = (V, E)$, two vertices s an t, and a capacity function $c : V \to \mathbb{R}^+$. A flow f is *feasible* if the total flow into every vertex v is at most $c(v)$:

$$\sum_u f(u \to v) \le c(v) \quad \text{for every vertex } v.$$

   Describe and analyze an efficient algorithm to compute a feasible flow of maximum value.

3. Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers. We want to *round* A to an integer matrix, by replacing each entry x in A with either $\lfloor x \rfloor$ or $\lceil x \rceil$, without changing the sum of entries in any row or column of A. For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longmapsto \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

   Describe an efficient algorithm that either rounds A in this fashion, or reports correctly that no such rounding is possible.

4. *Ad-hoc networks* are made up of cheap, low-powered wireless devices. In principle[1], these networks can be used on battlefields, in regions that have recently suffered from natural disasters, and in other situations where people might want to monitor conditions in hard-to-reach areas. The idea is that a large collection of cheap, simple devices could be dropped into the area from an airplane (for instance), and then they would somehow automatically configure themselves into an efficiently functioning wireless network.

   The devices can communication only within a limited range. We assume all the devices are identical; there is a distance D such that two devices can communicate if and only if the distance between them is at most D.

   We would like our ad-hoc network to be reliable, but because the devices are cheap and low-powered, they frequently fail. If a device detects that it is likely to fail, it should transmit the information it has to some other *backup* device within its communication range. To improve reliability, we require each device x to have k potential backup devices, all within distance D of x; we call these k devices the ***backup set*** of x. Also, we do not want any device to be in the backup set of too many other devices; otherwise, a single failure might affect a large fraction of the network.

   So suppose we are given the communication radius D, parameters b and k, and an array $d[1..n, 1..n]$ of distances, where $d[i, j]$ is the distance between device i and device j. Describe an algorithm that either computes a backup set of size k for each of the n devices, such that that no device appears in more than b backup sets, or reports (correctly) that no good collection of backup sets exists.

5. Let $G = (V, E)$ be a directed graph where for each vertex v, the in-degree and out-degree of v are equal. Let u and v be two vertices G, and suppose G contains k edge-disjoint paths from u to v. Under these conditions, must G also contain k edge-disjoint paths from v to u? Give a proof or a counterexample with explanation.

*6. ***[Extra credit]*** A *rooted tree* is a directed acyclic graph, in which every vertex has exactly one incoming edge, except for the *root*, which has no incoming edges. Equivalently, a rooted tree consists of a root vertex, which has edges pointing to the roots of zero or more smaller rooted trees. Describe a polynomial-time algorithm to compute, given two rooted trees A and B, the largest common rooted subtree of A and B.

   [Hint: Let $LCS(u, v)$ denote the largest common subtree whose root in A is u and whose root in B is v. Your algorithm should compute $LCS(u, v)$ for all vertices u and v using dynamic programming. This would be easy if every vertex had $O(1)$ children, and still straightforward if the children of each node were ordered from left to right and the common subtree had to respect that ordering. But for unordered trees with large degree, you need another trick to combine recursive subproblems efficiently. Don't waste your time trying to reduce the polynomial running time.]

[1] but not really in practice

CS 573: Graduate Algorithms, Fall 2008
Homework 6

Practice only

- This homework is only for practice; do not submit solutions. At least one (sub)problem (or something *very* similar) will appear on the final exam.

---

1. An *integer program* is a linear program with the additional constraint that the variables must take only integer values.

   (a) Prove that deciding whether an integer program has a feasible solution is NP-complete.

   (b) Prove that finding the optimal solution to an integer program is NP-hard.

   [Hint: Almost any NP-hard decision problem can be formulated as an integer program. Pick your favorite.]

2. Describe precisely how to dualize a linear program written in general form:

$$\begin{aligned}
\text{maximize } & \sum_{j=1}^{d} c_j x_j \\
\text{subject to } & \sum_{j=1}^{d} a_{ij} x_j \le b_i \quad \text{for each } i = 1..p \\
& \sum_{j=1}^{d} a_{ij} x_j = b_i \quad \text{for each } i = p+1..p+q \\
& \sum_{j=1}^{d} a_{ij} x_j \ge b_i \quad \text{for each } i = p+q+1..n
\end{aligned}$$

   Keep the number of dual variables as small as possible. The dual of the dual of any linear program should be *syntactically identical* to the original linear program.

3. Suppose you have a subroutine that can solve linear programs in polynomial time, but only if they are both feasible and bounded. Describe an algorithm that solves *arbitrary* linear programs in polynomial time, using this subroutine as a black box. Your algorithm should return an optimal solution if one exists; if no optimum exists, your algorithm should report that the input instance is UNBOUNDED or INFEASIBLE, whichever is appropriate. *[Hint: Add one constraint to guarantee boundedness; add one variable to guarantee feasibility.]*

4. Suppose you are given a set P of n points in some high-dimensional space $\mathbb{R}^d$, each labeled either *black* or *white*. A *linear classifier* is a d-dimensional vector c with the following properties:

   - If p is a black point, then $p \cdot c > 0$.
   - If p is a while point, then $p \cdot c < 0$.

   Describe an efficient algorithm to find a linear classifier for the given data points, or correctly report that none exists. *[Hint: This is almost trivial, but not quite.]*

Lots more linear programming problems can be found at http://www.ee.ucla.edu/ee236a/homework/problems.pdf. Enjoy!

You have 120 minutes to answer all five questions.
Write your answers in the separate answer booklet.
Please turn in your question sheet and your cheat sheet with your answers.

1. You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, several cards are dealt face up in a long row. Then you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. Each card is worth a different number of points. The player that collects the most points wins the game.

   Like most eight-year-olds who haven't studied algorithms, Elmo follows the obvious greedy strategy every time he plays: ***Elmo always takes the card with the higher point value.*** Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely *hates* it when grown-ups let him win.)

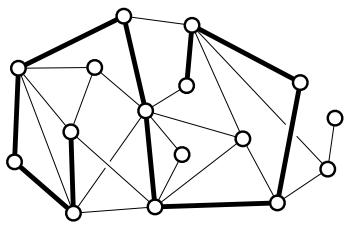
   (a) Describe an initial sequence of cards that allows you to win against Elmo, no matter who moves first, but *only* if you do *not* follow Elmo's greedy strategy.

   (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.

   Here is a sample game, where both you and Elmo are using the greedy strategy. Elmo wins 8–7. You cannot win this particular game, no matter what strategy you use.

Initial cards	2	4	5	1	3
Elmo takes the 3	2	4	5	1	~~3~~
You take the 2	~~2~~	4	5	1	
Elmo takes the 4		~~4~~	5	1	
You take the 5			~~5~~	1	
Elmo takes the 1				~~1~~	

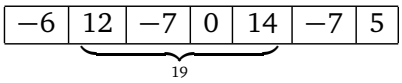
2. ***Prove*** that the following problem is NP-hard: Given an undirected graph G, find the longest path in G whose length is a multiple of 5.



This graph has a path of length 10, but no path of length 15.

3. Suppose you are given an array $A[1..n]$ of integers. Describe and analyze an algorithm that finds the largest sum of of elements in a contiguous subarray $A[i..j]$.

   For example, if the array A contains the numbers $[-6, 12, -7, 0, 14, -7, 5]$, your algorithm should return the number 19:



4. A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, 'bananaaananas' is a shuffle of 'banana' and 'ananas' in several different ways:

$$\texttt{banana}_{\texttt{ananas}} \qquad \texttt{ban}_{\texttt{ana}}\texttt{ana}_{\texttt{nas}} \qquad \texttt{b}_{\texttt{an}}\texttt{an}_{\texttt{a}}\texttt{a}_{\texttt{na}}\texttt{na}_{\texttt{s}}$$

   The strings 'prodgyrnamammiincg' and 'dyprongarmammicing' are both shuffles of 'dynamic' and 'programming':

$$\texttt{pro}^{\texttt{d}}\texttt{g}^{\texttt{y}}\texttt{r}^{\texttt{nam}}\texttt{ammi}^{\texttt{i}}\texttt{n}^{\texttt{c}}\texttt{g} \qquad \texttt{dy}_{\texttt{pro}}\texttt{n}_{\texttt{g}}\texttt{a}_{\texttt{r}}\texttt{m}_{\texttt{amm}}\texttt{ic}_{\texttt{ing}}$$

   Given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, describe and analyze an algorithm to determine whether C is a shuffle of A and B.

5. Suppose you are given two sorted arrays $A[1..m]$ and $B[1..n]$ and an integer k. Describe an algorithm to find the kth smallest element in the union of A and B in $\Theta(\log(m+n))$ time. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1..5] = [2, 5, 8, 17, 19] \qquad k = 6$$

   your algorithm should return 8. You can assume that the arrays contain no duplicates. An algorithm that works only in the special case $n = m = k$ is worth 7 points.

   [Hint: What can you learn from comparing one element of A to one element of B?]

You have 120 minutes to answer all five questions.
Write your answers in the separate answer booklet.
Please turn in your question sheet and your cheat sheet with your answers.

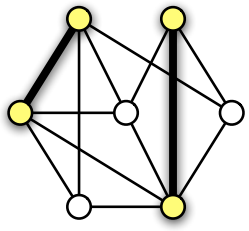
1. Consider the following modification of the 'dumb' 2-approximation algorithm for minimum vertex cover that we saw in class. The only change is that we output a set of edges instead of a set of vertices.

```
APPROXMINMAXMATCHING(G):
  M ← ∅
  while G has at least one edge
      let (u, v) be any edge in G
      remove u and v (and their incident edges) from G
      add (u, v) to M
  return M
```

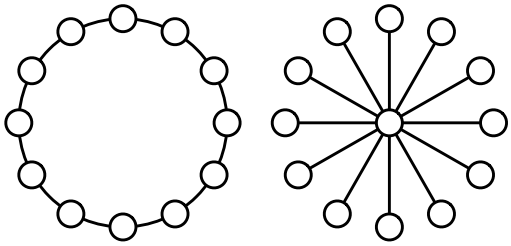
   (a) ***Prove*** that this algorithm computes a *matching*—no two edges in M share a common vertex.

   (b) ***Prove*** that M is a *maximal* matching—M is not a proper subgraph of another matching in G.

   (c) ***Prove*** that M contains at most twice as many edges as the *smallest* maximal matching in G.



The smallest maximal matching in a graph.



A cycle and a star.

2. Consider the following heuristic for computing a small vertex cover of a graph.

   - Assign a random *priority* to each vertex, chosen independently and uniformly from the real interval $[0, 1]$ (just like treaps).
   - Mark every vertex that does *not* have larger priority than *all* of its neighbors.

   For any graph G, let $OPT(G)$ denote the size of the smallest vertex cover of G, and let $M(G)$ denote the number of nodes marked by this algorithm.

   (a) ***Prove*** that the set of vertices marked by this heuristic is *always* a vertex cover.

   (b) Suppose the input graph G is a *cycle*, that is, a connected graph where every vertex has degree 2. What is the expected value of $M(G)/OPT(G)$? ***Prove*** your answer is correct.

   (c) Suppose the input graph G is a *star*, that is, a tree with one central vertex of degree $n-1$. What is the expected value of $M(G)/OPT(G)$? ***Prove*** your answer is correct.

3. Suppose we want to write an efficient function SHUFFLE($A[1..n]$) that randomly permutes the array A, so that each of the $n!$ permutations is equally likely.

   (a) ***Prove*** that the following SHUFFLE algorithm is ***not*** correct. *[Hint: There is a two-line proof.]*

   ```
   SHUFFLE(A[1..n]):
     for i = 1 to n
       swap A[i] ↔ A[RANDOM(n)]
   ```

   (b) Describe and analyze a correct SHUFFLE algorithm whose expected running time is $O(n)$. Your algorithm may call the function RANDOM(k), which returns an integer uniformly distributed in the range $\{1, 2, \ldots, k\}$ in $O(1)$ time. For example, RANDOM(2) simulates a fair coin flip, and RANDOM(1) *always* returns 1.

4. Let Φ be a legal input for 3SAT—a boolean formula in conjunctive normal form, with exactly three literals in each clause. Recall that an assignment of boolean values to the variables in Φ ***satisfies*** a clause if at least one of its literals is TRUE. The ***maximum satisfiability problem***, sometimes called MAX3SAT, asks for the maximum number of clauses that can be simultaneously satisfied by a single assignment. Solving MAXSAT exactly is clearly also NP-hard; this problem asks about approximation algorithms.

   (a) Let $MaxSat(\Phi)$ denote the maximum number of clauses that can be simultaneously satisfied by one variable assignment. Suppose we randomly assign each variable in Φ to be TRUE or FALSE, each with equal probability. ***Prove*** that the expected number of satisfied clauses is at least $\frac{7}{8}MaxSat(\Phi)$.

   (b) Let $MinUnsat(\Phi)$ denote the *minimum* number of clauses that can be simultaneously *unsatisfied* by a single assignment. ***Prove*** that it is NP-hard to approximate $MinUnsat(\Phi)$ within a factor of $10^{10^{100}}$.

5. Consider the following randomized algorithm for generating biased random bits. The subroutine FAIRCOIN returns either 0 or 1 with equal probability; the random bits returned by FAIRCOIN are mutually independent.

   ```
   ONEINTHREE:
     if FAIRCOIN = 0
       return 0
     else
       return 1 − ONEINTHREE
   ```

   (a) ***Prove*** that ONEINTHREE returns 1 with probability 1/3.

   (b) What is the *exact* expected number of times that this algorithm calls FAIRCOIN? ***Prove*** your answer is correct.

   (c) Now suppose you are *given* a subroutine ONEINTHREE that generates a random bit that is equal to 1 with probability 1/3. Describe a FAIRCOIN algorithm that returns either 0 or 1 with equal probability, using ONEINTHREE as a subroutine. **Your *only* source of randomness is ONEINTHREE; in particular, you may not use the RANDOM function from problem 3.**

   (d) What is the *exact* expected number of times that your FAIRCOIN algorithm calls ONEINTHREE? ***Prove*** your answer is correct.

You have 180 minutes to answer all seven questions.
Write your answers in the separate answer booklet.
You can keep everything except your answer booklet when you leave.

1. An *integer program* is a linear program with the additional constraint that the variables must take only integer values. ***Prove*** that deciding whether an integer program has a feasible solution is NP-complete. *[Hint: Almost any NP-hard decision problem can be formulated as an integer program. Pick your favorite.]*

2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *heater* is a priority search tree in which the priorities are given by the user, and the search keys are distributed uniformly and independently at random in the real interval $[0, 1]$. Intuitively, a heater is the 'opposite' of a treap.

   The following problems consider an n-node heater T whose node priorities are the integers from 1 to n. We identify nodes in T by their priorities; thus, 'node 5' means the node in T with priority 5. The min-heap property implies that node 1 is the root of T. Finally, let i and j be integers with $1 \le i < j \le n$.

   (a) ***Prove*** that in a random permutation of the $(i+1)$-element set $\{1, 2, \ldots, i, j\}$, elements i and j are adjacent with probability $2/(i+1)$.

   (b) ***Prove*** that node i is an ancestor of node j with probability $2/(i+1)$. *[Hint: Use part (a)!]*

   (c) What is the probability that node i is a *descendant* of node j? *[Hint: **Don't** use part (a)!]*

   (d) What is the *exact* expected depth of node j?

3. The UIUC Faculty Senate has decided to convene a committee to determine whether Chief Illiniwek should become the official ~~mascot~~ *symbol* of the University of Illinois Global Campus. Exactly one faculty member must be chosen from each academic department to serve on this committee. Some faculty members have appointments in multiple departments, but each committee member will represent only one department. For example, if Prof. Blagojevich is affiliated with both the Department of Corruption and the Department of Stupidity, and he is chosen as the Stupidity representative, then someone else must represent Corruption. Finally, University policy requires that any committee on virtual ~~mascots~~ *symbols* must contain the same number of assistant professors, associate professors, and full professors. Fortunately, the number of departments is a multiple of 3.

   Describe an efficient algorithm to select the membership of the Global Illiniwek Committee. Your input is a list of all UIUC faculty members, their ranks (assistant, associate, or full), and their departmental affiliation(s). There are n faculty members and $3k$ departments.

4. Let $\alpha(G)$ denote the number of vertices in the largest independent set in a graph G. ***Prove*** that the following problem is NP-hard: Given a graph G, return *any* integer between $\alpha(G) - 31337$ and $\alpha(G) + 31337$.

5. Let $G = (V, E)$ be a directed graph with capacities $c : E \to \mathbb{R}^+$, a source vertex s, and a target vertex t. Suppose someone hands you a function $f : E \to \mathbb{R}$. Describe and analyze a fast algorithm to determine whether f is a maximum (s, t)-flow in G.

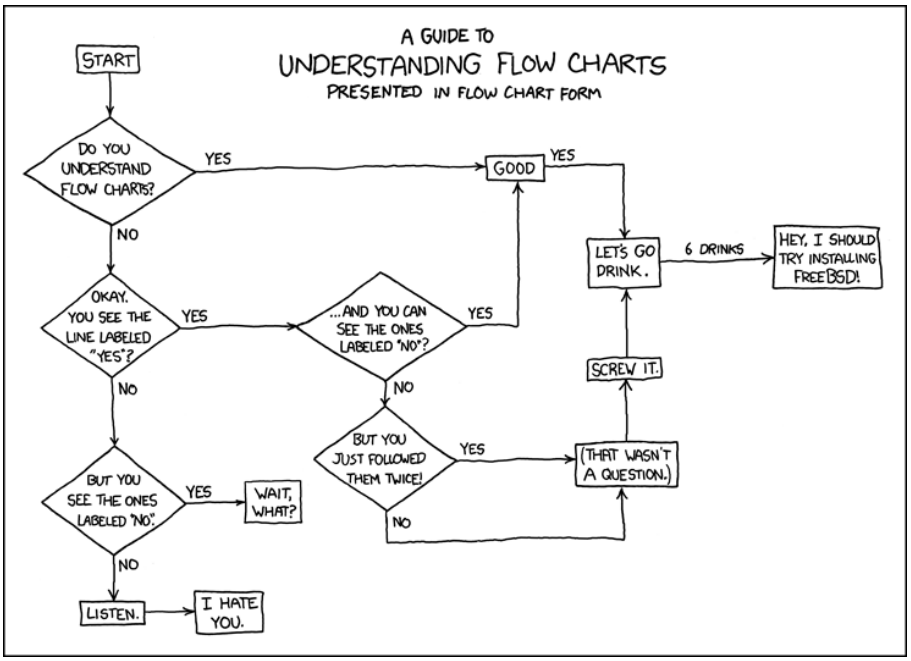
6. For some strange reason, you decide to ride your bicycle 3688 miles from Urbana to Wasilla, Alaska, to join in the annual Wasilla Mining Festival and Helicopter Wolf Hunt. The festival starts exactly 32 days from now, so you need to bike an average of 109 miles each day. Because you are a poor starving student, you can only afford to sleep at campgrounds, which are unfortunately *not* spaced exactly 109 miles apart. So some days you will have to ride more than average, and other days less, but you would like to keep the variation as small as possible. You settle on a formal scoring system to help decide where to sleep; if you ride x miles in one day, your score for that day is $(109 - x)^2$. What is the minimum possible total score for all 32 days?

   More generally, suppose you have D days to travel DP miles, there are n campgrounds along your route, and your score for traveling x miles in one day is $(x - P)^2$. You are given a sorted array $dist[1..n]$ of real numbers, where $dist[i]$ is the distance from your starting location to the ith campground; it may help to also set $dist[0] = 0$ and $dist[n + 1] = DP$. Describe and analyze a fast algorithm to compute the minimum possible score for your trip. The running time of your algorithm should depend on the integers D and n, but not on the real number P.

7. Describe and analyze efficient algorithms for the following problems.

   (a) Given a set of n integers, does it contain elements a and b such that $a + b = 0$?

   (b) Given a set of n integers, does it contain elements a, b, and c such that $a + b = c$?



— Randall Munroe, *xkcd*, December 17, 2008 (http://xkcd.com/518/)

CS 473: Undergraduate Algorithms, Spring 2009
Homework 0

Due **in class** at 11:00am, Tuesday, January 27, 2009

- This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** The early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
- Each student must submit individual solutions for this homework. For all future homeworks, groups of up to three students may submit a single, common solution.
- Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. In particular:
  - Submit five separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page. Please do not staple everything together.
  - You may use any source at your disposal—paper, electronic, or human—but you ***must*** write your solutions in your own words, and you ***must*** cite every source that you use.
  - Unless explicitly stated otherwise, ***every*** homework problem requires a proof.
  - Answering "I don't know" to any homework or exam problem (except for extra credit problems) is worth 25% partial credit.
  - Algorithms or proofs containing phrases like "and so on" or "repeat this process for all n", instead of an explicit loop, recursion, or induction, will receive 0 points.

Write the sentence "I understand the course policies." at the top of your solution to problem 1.

---

1. Professor George O'Jungle has a 27-node binary tree, in which every node is labeled with a unique letter of the Roman alphabet or the character **&**. Preorder and postorder traversals of the tree visit the nodes in the following order:

   - Preorder: **I Q J H L E M V O T S B R G Y Z K C A & F P N U D W X**
   - Postorder: **H E M L J V Q S G Y R Z B T C P U D N F W & X A K O I**

   (a) List the nodes in George's tree in the order visited by an inorder traversal.

   (b) Draw George's tree.

2. (a) **[5 pts]** Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Assume reasonable but nontrivial base cases. If your solution requires a particular base case, say so. ***Do not submit proofs***—just a list of five functions—but you should do them anyway, just for practice.

$$A(n) = 10A(n/5) + n$$

$$B(n) = 2B\left(\left\lceil \frac{n+3}{4} \right\rceil\right) + 5n^{6/7} - 8\sqrt{\frac{n}{\log n}} + 9\left\lfloor \log^{10} n \right\rfloor - 11$$

$$C(n) = 3C(n/2) + C(n/3) + 5C(n/6) + n^2$$

$$D(n) = \max_{0<k<n} (D(k) + D(n-k) + n)$$

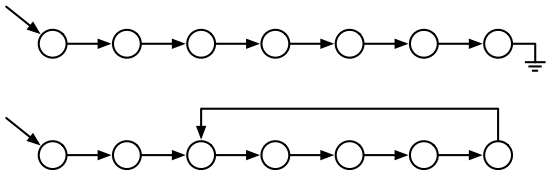
$$E(n) = \frac{E(n-1)\,E(n-3)}{E(n-2)} \qquad \textit{[Hint: Write out the first 20 terms.]}$$

(b) **[5 pts]** Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. ***Do not submit proofs***—just a sorted list of 16 functions—but you should do them anyway, just for practice.

Write $f(n) \ll g(n)$ to indicate that $f(n) = o(g(n))$, and write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. We use the notation $\lg n = \log_2 n$.

n	$\lg n$	$\sqrt{n}$	3^n
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$3^{\sqrt{n}}$	$\sqrt{3^n}$
$3^{\lg n}$	$\lg(3^n)$	$3^{\lg \sqrt{n}}$	$3\sqrt{\lg n}$
$\sqrt{3^{\lg n}}$	$\lg(3^{\sqrt{n}})$	$\lg \sqrt{3^n}$	$\sqrt{\lg(3^n)}$

3. Suppose you are given a pointer to the head of singly linked list. Normally, each node in the list has a pointer to the next element, and the last node's pointer is NULL. Unfortunately, your list might have been corrupted (by a bug in *somebody else's* code, of course), so that some node's pointer leads back to an earlier node in the list.



Top: A standard singly-linked list. Bottom: A corrupted singly linked list.

Describe an algorithm[1] that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. For full credit, your algorithm should run in $O(n)$ time, where n is the number of nodes in the list, and use $O(1)$ extra space (not counting the list itself).

[1] Since you understand the course policies, you know what this phrase means. Right?

4. (a) Prove that any integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example:

$$42 = 3^4 - 3^3 - 3^2 - 3^1$$
$$25 = 3^3 - 3^1 + 3^0$$
$$17 = 3^3 - 3^2 - 3^0$$

   (b) Prove that any integer (positive, negative, or zero) can be written in the form $\sum_i (-2)^i$, where the exponents i are distinct non-negative integers. For example:

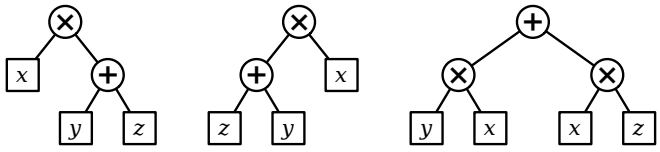
$$42 = (-2)^6 + (-2)^5 + (-2)^4 + (-2)^0$$
$$25 = (-2)^6 + (-2)^5 + (-2)^3 + (-2)^0$$
$$17 = (-2)^4 + (-2)^0$$

[Hint: Don't use weak induction. In fact, ***never*** *use weak induction.]*

5. An ***arithmetic expression tree*** is a binary tree where every leaf is labeled with a variable, every internal node is labeled with an arithmetic operation, and every internal node has exactly two children. For this problem, assume that the only allowed operations are $+$ and $\times$. Different leaves may or may not represent different variables.

   Every arithmetic expression tree represents a function, transforming input values for the leaf variables into an output value for the root, by following two simple rules: (1) The value of any $+$-node is the sum of the values of its children. (2) The value of any $\times$-node is the product of the values of its children.

   Two arithmetic expression trees are ***equivalent*** if they represent the same function; that is, the same input values for the leaf variables always leads to the same output value at both roots. An arithmetic expression tree is in ***normal form*** if the parent of every $+$-node (if any) is another $+$-node.



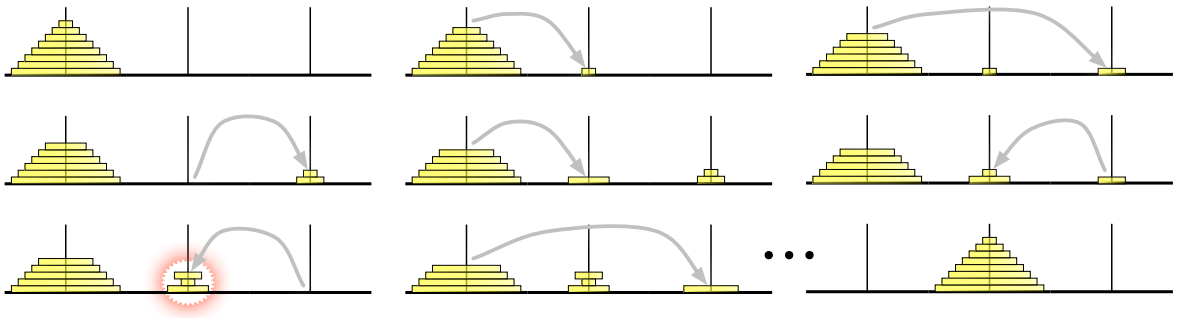
Three equivalent expression trees. Only the third is in normal form.

   Prove that for any arithmetic expression tree, there is an equivalent arithmetic expression tree in normal form.

*6. ***[Extra credit]*** You may be familiar with the story behind the famous Tower of Hanoï puzzle:

> At the great temple of Benares, there is a brass plate on which three vertical diamond shafts are fixed. On the shafts are mounted n golden disks of decreasing size. At the time of creation, the god Brahma placed all of the disks on one pin, in order of size with the largest at the bottom. The Hindu priests unceasingly transfer the disks from peg to peg, one at a time, never placing a larger disk on a smaller one. When all of the disks have been transferred to the last pin, the universe will end.

Recently the temple at Benares was relocated to southern California, where the monks are considerably more laid back about their job. At the "Towers of Hollywood", the golden disks have been replaced with painted plywood, and the diamond shafts have been replaced with Plexiglas. More importantly, the restriction on the order of the disks has been relaxed. While the disks are being moved, the bottom disk on any pin must be the *largest* disk on that pin, but disks further up in the stack can be in any order. However, after all the disks have been moved, they must be in sorted order again.



The Towers of Hollywood. The sixth move leaves the disks out of order.

Describe an algorithm that moves a stack of n disks from one pin to the another using the smallest possible number of moves. *Exactly* how many moves does your algorithm perform? *[Hint: The Hollywood monks can bring about the end of the universe considerably faster than their Benaresian counterparts.]*

CS 473: Undergraduate Algorithms, Spring 2009
Homework 1

Due Tuesday, February 3, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

---

1. The traditional Devonian/Cornish drinking song "The Barley Mow" has the following pseudolyrics, where *container*$[i]$ is the name of a container that holds 2^i ounces of beer. One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. (Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.)

```
BARLEYMOW(n):
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  "We'll drink it out of the jolly brown bowl,"
  "Here's a health to the barley-mow!"
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  for i ← 1 to n
      "We'll drink it out of the container[i], boys,"
      "Here's a health to the barley-mow!"
      for j ← i downto 1
          "The container[j],"
      "And the jolly brown bowl!"
      "Here's a health to the barley-mow!"
      "Here's a health to the barley-mow, my brave boys,"
      "Here's a health to the barley-mow!"
```

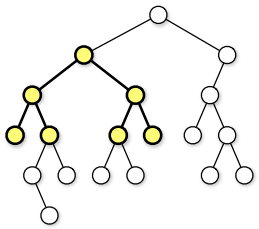
   (a) Suppose each container name *container*$[i]$ is a single word, and you can sing four words a second. How long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound.) *[Hint: Is 'barley-mow' one word or two? Does it matter?]*

   (b) If you want to sing this song for $n > 20$, you'll have to make up your own container names. To avoid repetition, these names will get progressively longer as n increases[1]. Suppose *container*$[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound.)

   (c) Suppose each time you mention the name of a container, you actually drink the corresponding amount of beer: one ounce for the jolly brown bowl, and 2^i ounces for each *container*$[i]$. Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang BARLEYMOW(n)? (Give an *exact* answer, not just an asymptotic bound.)

[1] "We'll drink it out of the hemisemidemiyottapint, boys!"

2. For this problem, a *subtree* of a binary tree means any connected subgraph; a binary tree is *complete* if every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

3. (a) Describe and analyze a recursive algorithm to reconstruct a binary tree, given its preorder and postorder node sequences (as in Homework 0, problem 1).

   (b) Describe and analyze a recursive algorithm to reconstruct a binary tree, given its preorder and *inorder* node sequences.

CS 473: Undergraduate Algorithms, Spring 2009
Homework 10

Due Tuesday, May 5, 2009 at 11:59:59pm

- Groups of up to three students may submit a single, common solution. Please clearly write every group member's name and NetID on every page of your submission.
- ***This homework is optional.*** If you submit solutions, they will be graded, and your overall homework grade will be the average of ten homeworks (Homeworks 0–10, dropping the lowest). If you do not submit solutions, your overall homework grade will be the average of *nine* homeworks (Homeworks 0–9, dropping the lowest).

---

1. Suppose you are given a magic black box that can determine ***in polynomial time***, given an arbitrary graph G, the number of vertices in the largest complete subgraph of G. Describe and analyze a ***polynomial-time*** algorithm that computes, given an arbitrary graph G, a complete subgraph of G of maximum size, using this magic black box as a subroutine.

2. PLANARCIRCUITSAT is a special case of CIRCUITSAT where the input circuit is drawn 'nicely' in the plane — no two wires cross, no two gates touch, and each wire touches only the gates it connects. (Not every circuit can be drawn this way!) As in the general CIRCUITSAT problem, we want to determine if there is an input that makes the circuit output TRUE?

   Prove that PLANARCIRCUITSAT is NP-complete. *[Hint:* XOR.*]*

3. For each problem below, either describe a polynomial-time algorithm or prove that the problem is NP-complete.

   (a) A *double-Eulerian* circuit in an undirected graph G is a closed walk that traverses every edge in G exactly twice. Given a graph G, does G have a *double-Eulerian* circuit?

   (b) A *double-Hamiltonian* circuit in an undirected graph G is a closed walk that visits every vertex in G exactly twice. Given a graph G, does G have a *double-Hamiltonian* circuit?

CS 473: Undergraduate Algorithms, Spring 2009
Homework 2

Written solutions due Tuesday, February 10, 2009 at 11:59:59pm.

- Roughly 1/3 of the students will give oral presentations of their solutions to the TAs. ***Please check Compass to check whether you are supposed give an oral presentation for this homework.*** Please see the course web page for further details.
- Groups of up to three students may submit a common solution. Please clearly write every group member's name and NetID on every page of your submission.
- Please start your solution to each numbered problem on a new sheet of paper. Please *don't* staple solutions for different problems together.
- ***For this homework only:*** These homework problems ask you to describe recursive backtracking algorithms for various problems. ***Don't*** use memoization or dynamic programming to make your algorithms more efficient; you'll get to do that on HW3. ***Don't*** analyze the running times of your algorithms. The ***only*** things you should submit for each problem are (1) a description of your recursive algorithm, and (2) a *brief* justification for its correctness.

---

1. A ***basic arithmetic expression*** is composed of characters from the set $\{1, +, \times\}$ and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1$$
$$((1+1)\times(1+1+1+1+1))+((1+1)\times(1+1))$$
$$(1+1)\times(1+1+1+1+1+1+1)$$
$$(1+1)\times(((1+1+1)\times(1+1))+1)$$

   Describe a recursive algorithm to compute, given an integer n as input, the minimum number of 1's in a basic arithmetic expression whose value is n. The number of parentheses doesn't matter, just the number of 1's. For example, when $n = 14$, your algorithm should return 8, for the final expression above.

2. A sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$ is ***bitonic*** if there is an index i with $1 < i < n$, such that the prefix $\langle a_1, a_2, \ldots, a_i \rangle$ is strictly increasing and the suffix $\langle a_i, a_{i+1}, \ldots, a_n \rangle$ is strictly decreasing. In particular, a bitonic sequence must contain at least three elements.

   Describe a recursive algorithm to compute, given a sequence A, the length of the longest bitonic subsequence of A.

3. A ***palindrome*** is a string that reads the same forwards and backwards, like x, pop, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. Any string can be broken into sequence of palindromes. For example, the string bubbaseesabanana ('Bubba sees a banana.') can be broken into palindromes in several different ways; for example:

   bub + baseesab + anana
   b + u + bb + a + sees + aba + nan + a
   b + u + bb + a + sees + a + b + anana
   b + u + b + b + a + s + e + e + s + a + b + a + n + a + n + a

   Describe a recursive algorithm to compute the minimum number of palindromes that make up a given input string. For example, given the input string bubbaseesabanana, your algorithm would return the integer 3.

CS 473: Undergraduate Algorithms, Spring 2009
Homework 3

Written solutions due Tuesday, February 17, 2009 at 11:59:59pm.

---

1. Redo Homework 2, but now with dynamic programming!

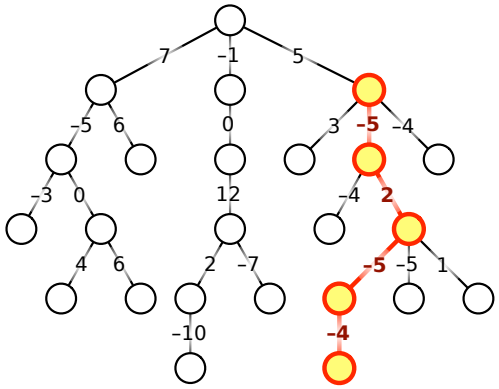
   (a) Describe and analyze an efficient algorithm to compute the minimum number of 1's in a basic arithmetic expression whose value is a given positive integer.

   (b) Describe and analyze an efficient algorithm to compute the length of the longest bitonic subsequence of a given input sequence.

   (c) Describe and analyze an efficient algorithm to compute the minimum number of palindromes that make up a given input string.

   Please see Homework 2 for more detailed descriptions of each problem. ***Solutions for Homework 2 will be posted Friday, after the HW2 oral presentations.*** You may (and should!) use anything from those solutions without justification.

2. Let T be a rooted tree with integer weights on its edges, which could be positive, negative, or zero. Design an algorithm to find the minimum-length path from a node in T down to one of its descendants. The length of a path is the sum of the weights of its edges. For example, given the tree shown below, your algorithm should return the number -12. For full credit, your algorithm should run in $O(n)$ time.



The minimum-weight downward path in this tree has weight -12.

3. Describe and analyze an efficient algorithm to compute the longest common subsequence of *three* given strings. For example, given the input strings **E**P**I**D**E**M**I**OL**O**GIST, R**E**FR**I**G**E**RAT**IO**N, and SUP**E**RCALIFRAGIL**I**STIC**E**XP**I**ALOD**O**CIOUS, your algorithm should return the number 5, because the longest common subsequence is **EIEIO**.

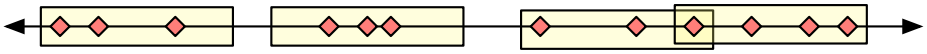
CS 473: Undergraduate Algorithms, Spring 2009
Homework 3½

Practice only

---

1. After graduating from UIUC, you are hired by a mobile phone company to plan the placement of new cell towers along a long, straight, nearly-deserted highway out west. Each cell tower can transmit the same fixed distance from its location. Federal law requires that any building along the highway must be within the broadcast range of at least one tower. On the other hand, your company wants to build as few towers as possble. Given the locations of the buildings, where should you build the towers?

   More formally, suppose you are given a set $X = \{x_1, x_2, \ldots, x_n\}$ of points on the real number line. Describe an algorithm to compute the minimum number of intervals of length 1 that can cover all the points in X. For full credit, your algorithm should run in $O(n \log n)$ time.



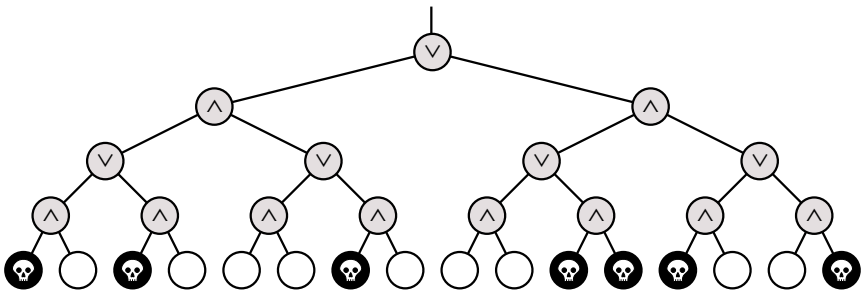
A set of points that can be covered by four unit intervals.

2. (a) The *left spine* of a binary tree is a path starting at the root and following only left-child pointers down to a leaf. What is the expected number of nodes in the left spine of an n-node treap?

   (b) What is the expected number of leaves in an n-node treap? *[Hint: What is the probability that in an n-node treap, the node with kth smallest search key is a leaf?]*

   (c) Prove that the expected number of proper descendants of any node in a treap is exactly equal to the expected depth of that node.

3. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

(a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

(b) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $O(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*

*(c) Describe a randomized algorithm that determines whether you can win in $O(c^n)$ expected time, for some constant $c < 3$. *[Hint: You may not need to change your algorithm from part (b) at all!]*

CS 473: Undergraduate Algorithms, Spring 2009
Homework 3

Written solutions due Tuesday, March 2, 2009 at 11:59:59pm.

---

1. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

   - MAKEQUEUE: Return a new priority queue containing the empty set.
   - FINDMIN(Q): Return the smallest element of Q (if any).
   - DELETEMIN(Q): Remove the smallest element in Q (if any).
   - INSERT(Q, x): Insert element x into Q, if it is not already there.
   - DECREASEKEY(Q, x, y): Replace an element $x \in Q$ with a smaller key y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q containing x.
   - DELETE(Q, x): Delete the element $x \in Q$. The input is a pointer directly to the node in Q containing x.
   - MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2; this operation destroys Q_1 and Q_2.

   A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```
MELD(Q1, Q2):
  if Q1 is empty return Q2
  if Q2 is empty return Q1

  if key(Q1) > key(Q2)
      swap Q1 ↔ Q2

  with probability 1/2
      left(Q1) ← MELD(left(Q1), Q2)
  else
      right(Q1) ← MELD(right(Q1), Q2)

  return Q1
```

   (a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (*not* just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where n is the total number of nodes in both trees. *[Hint: How long is a random root-to-leaf path in an n-node binary tree if each left/right choice is made with equal probability?]*

   (b) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ expected time.)

2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *heater* is a priority search tree in which the priorities are given by the user, and the search keys are distributed uniformly and independently at random in the real interval $[0,1]$. Intuitively, a heater is the 'opposite' of a treap.

   The following problems consider an n-node heater T whose node priorities are the integers from 1 to n. We identify nodes in T by their priorities; thus, 'node 5' means the node in T with priority 5. The min-heap property implies that node 1 is the root of T. Finally, let i and j be integers with $1 \le i < j \le n$.

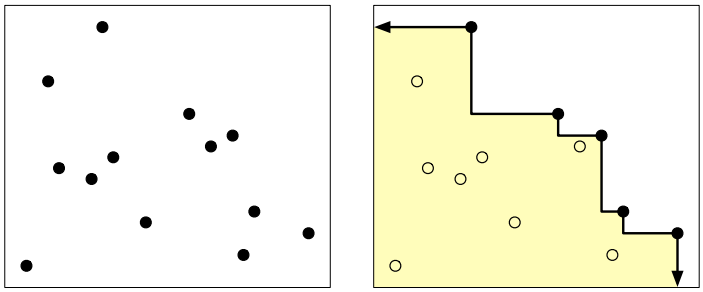
   (a) ***Prove*** that in a random permutation of the $(i+1)$-element set $\{1, 2, \ldots, i, j\}$, elements i and j are adjacent with probability $2/(i+1)$.

   (b) ***Prove*** that node i is an ancestor of node j with probability $2/(i+1)$. *[Hint: Use part (a)!]*

   (c) What is the probability that node i is a *descendant* of node j? *[Hint: **Don't** use part (a)!]*

   (d) What is the *exact* expected depth of node j?

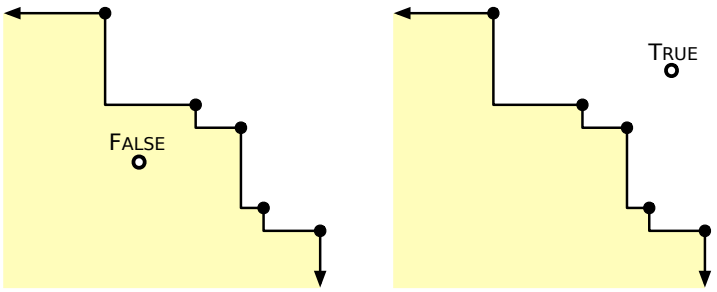
3. Let P be a set of n points in the plane. The *staircase* of P is the set of all points in the plane that have at least one point in P both above and to the right.



A set of points in the plane and its staircase (shaded).

   (a) Describe an algorithm to compute the staircase of a set of n points in $O(n \log n)$ time.

   (b) Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm ABOVE?(x, y) that returns TRUE if the point (x, y) is above the staircase, or FALSE otherwise. Your data structure should use $O(n)$ space, and your ABOVE? algorithm should run in $O(\log n)$ time.



Two staircase queries.

CS 473: Undergraduate Algorithms, Spring 2009
Homework 5

Written solutions due Tuesday, March 9, 2009 at 11:59:59pm.

---

1. Remember the difference between stacks and queues? Good.

   (a) Describe how to implement a queue using two stacks and $O(1)$ additional memory, so that the amortized time for any enqueue or dequeue operation is $O(1)$. The *only* access you have to the stacks is through the standard methods PUSH and POP.

   (b) A *quack* is an abstract data type that combines properties of both stacks and queues. It can be viewed as a list of elements written left to right such that three operations are possible:

   - **Push:** add a new item to the left end of the list;
   - **Pop:** remove the item on the left end of the list;
   - **Pull:** remove the item on the right end of the list.

   Implement a quack using *three* stacks and $O(1)$ additional memory, so that the amortized time for any push, pop, or pull operation is $O(1)$. Again, you are *only* allowed to access the stacks through the standard methods PUSH and POP.

2. In a *dirty* binary search tree, each node is labeled either *clean* or *dirty*. The lazy deletion scheme used for scapegoat trees requires us to *purge* the search tree, keeping all the clean nodes and deleting all the dirty nodes, as soon as half the nodes become dirty. In addition, the purged tree should be perfectly balanced.

   Describe and analyze an algorithm to purge an *arbitrary* n-node dirty binary search tree in $O(n)$ time, using at most $O(\log n)$ space (in addition to the tree itself). Don't forget to include the recursion stack in your space bound. An algorithm that uses $\Theta(n)$ additional space in the worst case is worth half credit.

3. Some applications of binary search trees attach a *secondary data structure* to each node in the tree, to allow for more complicated searches. Maintaining these secondary structures usually complicates algorithms for keeping the top-level search tree balanced.

   Let T be an arbitrary binary tree. Suppose every node v in T stores a secondary structure of size $O(size(v))$, which can be built in $O(size(v))$ time, where $size(v)$ denotes the number of descendants of v. Performing a rotation at any node v now requires $O(size(v))$ time, because we have to rebuild one of the secondary structures.

   (a) **[1 pt]** Overall, how much space does this data structure use *in the worst case*?

   (b) **[1 pt]** How much space does this structure use if the primary search tree T is perfectly balanced?

   (c) **[2 pts]** Suppose T is a splay tree. Prove that the *amortized* cost of a splay (and therefore of a search, insertion, or deletion) is $\Omega(n)$. *[Hint: This is easy!]*

(d) **[3 pts]** Now suppose T is a scapegoat tree, and that rebuilding the subtree rooted at v requires $\Theta(size(v) \log size(v))$ time (because we also have to rebuild the secondary structures at every descendant of v). What is the *amortized* cost of inserting a new element into T?

(e) **[3 pts]** Finally, suppose T is a treap. What's the worst-case *expected* time for inserting a new element into T?

CS 473: Undergraduate Algorithms, Spring 2009
Homework 6

Written solutions due Tuesday, March 17, 2009 at 11:59:59pm.

1. Let G be an undirected graph with n nodes. Suppose that G contains two nodes s and t, such that every path from s to t contains more than $n/2$ edges.

   (a) Prove that G must contain a vertex v that lies on *every* path from s to t.

   (b) Describe an algorithm that finds such a vertex v in $O(V + E)$ time.

2. Suppose you are given a graph G with weighted edges and a minimum spanning tree T of G.

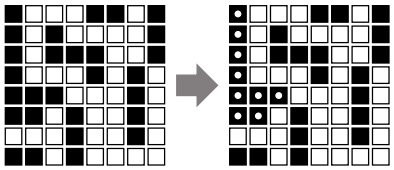
   (a) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is decreased.

   (b) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is increased.

   In both cases, the input to your algorithm is the edge e and its new weight; your algorithms should modify T so that it is still a minimum spanning tree. *[Hint: Consider the cases $e \in T$ and $e \notin T$ separately.]*

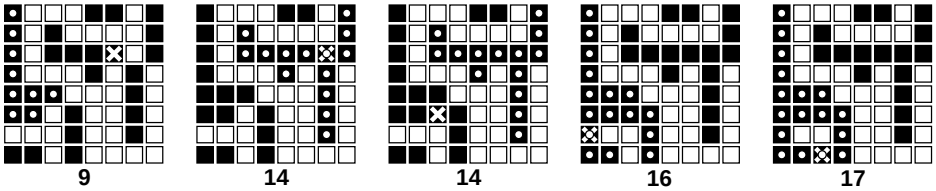
3. (a) Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an $n \times n$ bitmap $B[1..n, 1..n]$.

   For example, given the bitmap below as input, your algorithm should return the number 9, because the largest conected black component (marked with white dots on the right) contains nine pixels.



   (b) Design and analyze an algorithm BLACKEN(i, j) that colors the pixel $B[i, j]$ black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.

   For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the BLACKEN algorithm.



   (c) What is the *worst-case* running time of your BLACKEN algorithm?

CS 473: Undergraduate Algorithms, Spring 2009
Homework 6½

Practice only—do not submit solutions

---

1. In class last Tuesday, we discussed Ford's generic shortest-path algorithm—relax arbitrary tense edges until no edge is tense. This problem asks you to fill in part of the proof that this algorithm is correct.

   (a) Prove that after *every* call to RELAX, for every vertex v, either $dist(v) = \infty$ or $dist(v)$ is the total weight of some path from s to v.

   (b) Prove that for every vertex v, when the generic algorithm halts, either $pred(v) = \text{NULL}$ and $dist(v) = \infty$, or $dist(v)$ is the total weight of the predecessor chain ending at v:

   $$s \to \cdots \to pred(pred(v)) \to pred(v) \to v.$$

2. Describe a modification of Shimbel's shortest-path algorithm that actually computes a negative-weight cycle if any such cycle is reachable from s, or a shortest-path tree rooted at s if there is no such cycle. Your modified algorithm should still run in $O(VE)$ time.

3. After graduating you accept a job with Aerophobes-Я-Us, the leading traveling agency for people who hate to fly. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying (and by extension, airports), so any trip you plan needs to be as short as possible. You know all the departure and arrival times of all the flights on the planet.

   Suppose one of your customers wants to fly from city X to city Y. Describe an algorithm to find a sequence of flights that minimizes the *total time in transit*—the length of time from the initial departure to the final arrival, including time at intermediate airports waiting for connecting flights. *[Hint: Modify the input data and apply Dijkstra's algorithm.]*

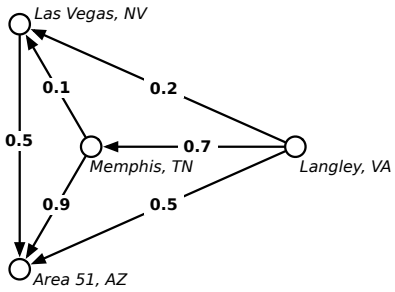
CS 473: Undergraduate Algorithms, Spring 2009
Homework 6¾

Practice only—do not submit solutions

---

1. Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road *won't* be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What route should he take so that he has the least chance of being abducted?

   More formally, you are given a directed graph $G = (V, E)$, where every edge e has an independent safety probability $p(e)$. The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex s to a given target vertex t.



   For example, with the probabilities shown above, if Mulder tries to drive directly from Langley to Area 51, he has a 50% chance of getting there without being abducted. If he stops in Memphis, he has a $0.7 \times 0.9 = 63\%$ chance of arriving safely. If he stops first in Memphis and then in Las Vegas, he has a $1 - 0.7 \times 0.1 \times 0.5 = 96.5\%$ chance of being abducted! (That's how they got Elvis, you know.)

2. Let $G = (V, E)$ be a directed graph with weighted edges; edge weights could be positive, negative, or zero. Suppose the vertices of G are partitioned into k disjoint subsets $V_1, V_2, \ldots, V_k$; that is, every vertex of G belongs to exactly one subset V_i. For each i and j, let $\delta(i, j)$ denote the minimum shortest-path distance between any vertex in V_i and any vertex in V_j:

$$\delta(i, j) = \min\{dist(u, v) \mid u \in V_i \text{ and } v \in V_j\}.$$

   Describe an algorithm to compute $\delta(i, j)$ for *all* i and j in time $O(VE + kE \log E)$. The output from your algorithm is a $k \times k$ array.

3. Recall[1] that a deterministic finite automaton (DFA) is formally defined as a tuple $M = (\Sigma, Q, q_0, F, \delta)$, where the finite set Σ is the input alphabet, the finite set Q is the set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of final (accepting) states, and $\delta : Q \times \Sigma \to Q$ is the transition function. Equivalently, a DFA is a directed (multi-)graph with labeled edges, such that each symbol in Σ is the label of exactly one edge leaving any vertex. There is a special 'start' vertex q_0, and a subset of the vertices are marked as 'accepting states'. Any string in Σ^* describes a unique walk starting at q_0.

   Stephen Kleene[2] proved that the language accepted by any DFA is identical to the language described by some regular expression. This problem asks you to develop a variant of the Floyd-Warshall all-pairs shortest path algorithm that computes a regular expression that is equivalent to the language accepted by a given DFA.

   Suppose the input DFA M has n states, numbered from 1 to n, where (without loss of generality) the start state is state 1. Let $L(i, j, r)$ denote the set of all words that describe walks in M from state i to state j, where every intermediate state lies in the subset $\{1, 2, \ldots, r\}$; thus, the language accepted by the DFA is exactly

$$\bigcup_{q \in F} L(1, q, n).$$

   Let $R(i, j, r)$ be a regular expression that describes the language $L(i, j, r)$.

   (a) What is the regular expression $R(i, j, 0)$?

   (b) Write a recurrence for the regular expression $R(i, j, r)$ in terms of regular expressions of the form $R(i', j', r - 1)$.

   (c) Describe a polynomial-time algorithm to compute $R(i, j, n)$ for all states i and j. (Assume that you can concatenate two regular expressions in $O(1)$ time.)

[1] No, really, you saw this in CS 273/373.

[2] Pronounced 'clay knee', not 'clean' or 'clean-ee' or 'clay-nuh' or 'dimaggio'.

CS 473: Undergraduate Algorithms, Spring 2009
Homework 7

Due Tuesday, April 14, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

---

1. A graph is *bipartite* if its vertices can be colored black or white such that every edge joins vertices of two different colors. A graph is *d-regular* if every vertex has degree d. A *matching* in a graph is a subset of the edges with no common endpoints; a matching is *perfect* if it touches every vertex.

   (a) Prove that every regular bipartite graph contains a perfect matching.

   (b) Prove that every d-regular bipartite graph is the union of d perfect matchings.

2. Let $G = (V, E)$ be a directed graph where for each vertex v, the in-degree of v and out-degree of v are equal. Let u and v be two vertices G, and suppose G contains k edge-disjoint paths from u to v. Under these conditions, must G also contain k edge-disjoint paths from v to u? Give a proof or a counterexample with explanation.

3. A flow f is called ***acyclic*** if the subgraph of directed edges with positive flow contains no directed cycles. A flow is *positive* if its value is greater than 0.

   (a) A *path flow* assigns positive values only to the edges of one simple directed path from s to t. Prove that every positive acyclic flow can be written as the sum of a finite number of path flows.

   (b) Describe a flow in a directed graph that *cannot* be written as the sum of path flows.

   (c) A *cycle flow* assigns positive values only to the edges of one simple directed cycle. Prove that every flow can be written as the sum of a finite number of path flows and cycle flows.

   (d) Prove that for any flow f, there is an acyclic flow with the same value as f. (In particular, this implies that some maximum flow is acyclic.)

CS 473: Undergraduate Algorithms, Spring 2009
Homework 8

Due Tuesday, April 21, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

---

1. A *cycle cover* of a given directed graph $G = (V, E)$ is a set of vertex-disjoint cycles that cover all the vertices. Describe and analyze an efficient algorithm to find a cycle cover for a given graph, or correctly report that non exists. *[Hint: Use bipartite matching!]*

2. Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers. We want to *round* A to an integer matrix, by replacing each entry x in A with either $\lfloor x \rfloor$ or $\lceil x \rceil$, without changing the sum of entries in any row or column of A. For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longmapsto \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

   Describe an efficient algorithm that either rounds A in this fashion, or reports correctly that no such rounding is possible.

3. *Ad-hoc networks* are made up of cheap, low-powered wireless devices. In principle[1], these networks can be used on battlefields, in regions that have recently suffered from natural disasters, and in other hard-to-reach areas. The idea is that several simple devices could be distributed randomly in the area of interest (for example, dropped from an airplane), and then they would somehow automatically configure themselves into an efficiently functioning wireless network.

   The devices can communicate only within a limited range. We assume all the devices are identical; there is a distance D such that two devices can communicate if and only if the distance between them is at most D.

   We would like our ad-hoc network to be reliable, but because the devices are cheap and low-powered, they frequently fail. If a device detects that it is likely to fail, it should transmit all its information to some other *backup* device within its communication range. To improve reliability, we require each device x to have k potential backup devices, all within distance D of x; we call these k devices the ***backup set*** of x. Also, we do not want any device to be in the backup set of too many other devices; otherwise, a single failure might affect a large fraction of the network.

   Suppose we are given the communication distance D, parameters b and k, and an array $d[1..n, 1..n]$ of distances, where $d[i, j]$ is the distance between device i and device j. Describe and analyze an algorithm that either computes a backup set of size k for each of the n devices, such that that no device appears in more than b backup sets, or correctly reports that no good collection of backup sets exists.

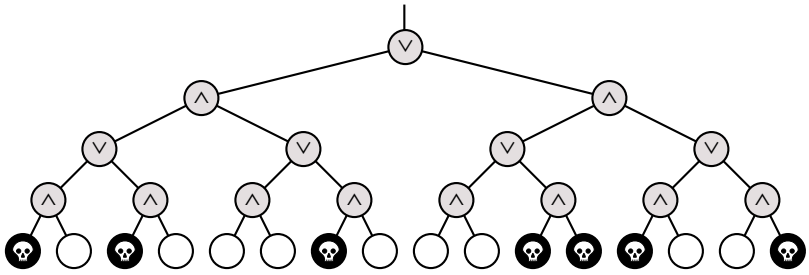
[1] but not so much in practice

CS 473: Undergraduate Algorithms, Spring 2009
Homework 9

Due Tuesday, April 28, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it is white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it is worth playing or not as follows. Imagine that the nodes at even levels (where it is your turn) are OR gates, the nodes at odd levels (where it is Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black leaves stand represent TRUE and FALSE inputs, respectively. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

   (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy.]*

   (b) Prove that *every* deterministic algorithm must examine *every* leaf of the tree in the worst case. Since there are 4^n leaves, this implies that any deterministic algorithm must take $\Omega(4^n)$ time in the worst case. Use an adversary argument; in other words, assume that Death cheats.

   (c) ***[Extra credit]*** Describe a *randomized* algorithm that runs in $O(3^n)$ expected time.

2. We say that an array $A[1..n]$ is *k-sorted* if it can be divided into k blocks, each of size n/k, such that the elements in each block are larger than the elements in earlier blocks, and smaller than elements in later blocks. The elements within each block need not be sorted.

   For example, the following array is 4-sorted:

1	2	4	3	7	6	8	5	10	11	9	12	15	13	16	14

(a) Describe an algorithm that k-sorts an arbitrary array in time $O(n \log k)$.

(b) Prove that any comparison-based k-sorting algorithm requires $\Omega(n \log k)$ comparisons in the worst case.

(c) Describe an algorithm that completely sorts an already k-sorted array in time $O(n \log(n/k))$.

(d) Prove that any comparison-based algorithm to completely sort a k-sorted array requires $\Omega(n \log(n/k))$ comparisons in the worst case.

In all cases, you can assume that n/k is an integer.

3. UIUC has just finished constructing the new Reingold Building, the tallest dormitory on campus. In order to determine how much insurance to buy, the university administration needs to determine the highest safe floor in the building. A floor is consdered *safe* if ~~a drunk student~~ ***an egg*** can fall from a window on that floor and land without breaking; if the egg breaks, the floor is considered *unsafe*. Any floor that is higher than an unsafe floor is also considered unsafe. The only way to determine whether a floor is safe is to drop an egg from a window on that floor.

You would like to find the lowest unsafe floor L by performing as few tests as possible; unfortunately, you have only a very limited supply of eggs.

(a) Prove that if you have only one egg, you can find the lowest unsafe floor with L tests. *[Hint: Yes, this is trivial.]*

(b) Prove that if you have only one egg, you must perform at least L tests in the worst case. In other words, prove that your algorithm from part (a) is optimal. *[Hint: Use an adversary argument.]*

(c) Describe an algorithm to find the lowest unsafe floor using *two* eggs and only $O(\sqrt{L})$ tests. *[Hint: Ideally, each egg should be dropped the same number of times. How many floors can you test with n drops?]*

(d) Prove that if you start with two eggs, you must perform at least $\Omega(\sqrt{L})$ tests in the worst case. In other words, prove that your algorithm from part (c) is optimal.

*(e) **[Extra credit!]** Describe an algorithm to find the lowest unsafe floor using k eggs, using as few tests as possible, and prove your algorithm is optimal for all values of k.

CS 473: Undergraduate Algorithms, Spring 2009
Head Banging Session 0

January 20 and 21, 2009

---

1. Solve the following recurrences. If base cases are provided, find an *exact* closed-form solution. Otherwise, find a solution of the form $\Theta(f(n))$ for some function f.

   - **Warmup:** You should be able to solve these almost as fast as you can write down the answers.
     (a) $A(n) = A(n-1) + 1$, where $A(0) = 0$.
     (b) $B(n) = B(n-5) + 2$, where $B(0) = 17$.
     (c) $C(n) = C(n-1) + n^2$
     (d) $D(n) = 3\,D(n/2) + n^2$
     (e) $E(n) = 4\,E(n/2) + n^2$
     (f) $F(n) = 5\,F(n/2) + n^2$

   - **Real practice:**
     (a) $A(n) = A(n/3) + 3A(n/5) + A(n/15) + n$
     (b) $B(n) = \min_{0<k<n} (B(k) + B(n-k) + n)$
     (c) $C(n) = \max_{n/4<k<3n/4} (C(k) + C(n-k) + n)$
     (d) $D(n) = \max_{0<k<n} \big(D(k) + D(n-k) + k(n-k)\big)$, where $D(1) = 0$
     (e) $E(n) = 2E(n-1) + E(n-2)$, where $E(0) = 1$ and $E(1) = 2$
     (f) $F(n) = \dfrac{1}{F(n-1)\,F(n-2)}$, where $F(0) = 1$ and $F(2) = 2$
     *(g) $G(n) = n\,G(\sqrt{n}) + n^2$

2. The *Fibonacci numbers* F_n are defined recursively as follows: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for every integer $n \geq 2$. The first few Fibonacci numbers are $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$.

   Prove that any non-negative integer can be written as the sum of distinct *non-consecutive* Fibonacci numbers. That is, if any Fibonacci number F_n appears in the sum, then its neighbors F_{n-1} and F_{n+1} do not. For example:

$$\begin{aligned} 88 = \quad & 55 + 21 + 8 + 3 + 1 && = F_{10} + F_8 + F_6 + F_4 + F_2 \\ 42 = \quad & 34 + 8 && = F_9 + F_6 \\ 17 = \quad & 13 + 3 + 1 && = F_7 + F_4 + F_2 \end{aligned}$$

3. Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

   Prove that any finite set of pigeons can be arranged in a row from left to right so that every pigeon pecks the pigeon immediately to its left. Pretty please.

1. An *inversion* in an array $A[1..n]$ is a pair of indices (i, j) such that $i < j$ and $A[i] > A[j]$. The number of inversions in an n-element array is between 0 (if the array is sorted) and $\binom{n}{2}$ (if the array is sorted backward).

   Describe and analyze an algorithm to count the number of inversions in an n-element array in $O(n \log n)$ time.

2. (a) Prove that the following algorithm actually sorts its input.

```
StoogeSort(A[0..n-1]):
  if n = 2 and A[0] > A[1]
      swap A[0] ↔ A[1]
  else if n > 2
      m = ⌈2n/3⌉
      StoogeSort(A[0..m-1])
      StoogeSort(A[n-m..n-1])
      StoogeSort(A[0..m-1])
```

   (b) Would StoogeSort still sort correctly if we replaced $m = \lceil 2n/3 \rceil$ with $m = \lfloor 2n/3 \rfloor$? Justify your answer.

   (c) State a recurrence (including base case(s)) for the number of comparisons executed by StoogeSort.

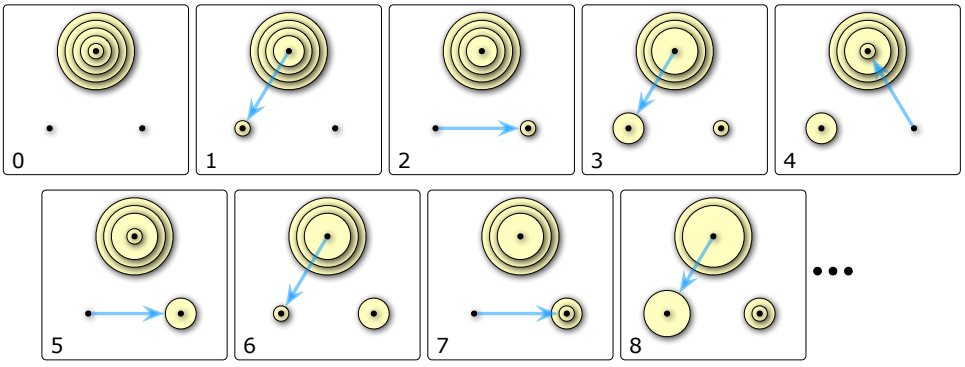
   (d) Solve this recurrence. *[Hint: Ignore the ceiling.]*

   (e) **To think about on your own:** Prove that the number of *swaps* executed by StoogeSort is at most $\binom{n}{2}$.

3. Consider the following restricted variants of the Tower of Hanoi puzzle. In each problem, the needles are numbered 0, 1, and 2, and your task is to move a stack of n disks from needle 1 to needle 2.

   (a) Suppose you are forbidden to move any disk directly between needle 1 and needle 2; *every* move must involve needle 0. Describe an algorithm to solve this version of the puzzle using as few moves as possible. *Exactly* how many moves does your algorithm make?

   (b) Suppose you are only allowed to move disks from needle 0 to needle 2, from needle 2 to needle 1, or from needle 1 to needle 0. Equivalently, Suppose the needles are arranged in a circle and numbered in clockwise order, and you are only allowed to move disks counterclockwise. Describe an algorithm to solve this version of the puzzle using as few moves as possible. *Exactly* how many moves does your algorithm make?



The first eight moves in a counterclockwise Towers of Hanoi solution

★(c) Finally, suppose you are forbidden to move any disk directly from needle 1 to 2, but any other move is allowed. Describe an algorithm to solve this version of the puzzle using as few moves as possible. *Exactly* how many moves does your algorithm make?

[Hint: This version is ***considerably*** *harder than the other two.]*

CS 473: Undergraduate Algorithms, Spring 2009
HBS 10

1. Consider the following problem, called *BOX-DEPTH*: Given a set of n axis-aligned rectangles in the plane, how big is the largest subset of these rectangles that contain a common point?

   (a) Describe a polynomial-time reduction from *BOX-DEPTH* to *MAX-CLIQUE*.

   (b) Describe and analyze a polynomial-time algorithm for *BOX-DEPTH*. [Hint: $O(n^3)$ time should be easy, but $O(n \log n)$ time is possible.]

   (c) Why don't these two results imply that $P = NP$?

2. Suppose you are given a magic black box that can determine in polynomial time, given an arbitrary weighted graph G, the length of the shortest Hamiltonian cycle in G. Describe and analyze a polynomial-time algorithm that computes, given an arbitrary weighted graph G, the shortest Hamiltonian cycle in G, using this magic black box as a subroutine.

3. Prove that the following problems are NP-complete.

   (a) Given an undirected graph G, does G have a spanning tree in which every node has degree at most 17?

   (b) Given an undirected graph G, does G have a spanning tree with at most 42 leaves?

CS 473: Undergraduate Algorithms, Spring 2009
HBS 11

1. You step in a party with a camera in your hand. Each person attending the party has some friends there. You want to have exactly one picture of each person in your camera. You want to use the following protocol to collect photos. At each step, the person that has the camera in his hand takes a picture of one of his/her friends and pass the camera to him/her. Of course, you only like the solution if it finishes when the camera is in your hand. Given the friendship matrix of the people in the party, design a polynomial algorithm that decides whether this is possible, or prove that this decision problem is NP-hard.

2. A boolean formula is in disjunctive normal form (DNF) if it is a disjunctions (OR) of several clauses, each of which is the conjunction (AND) of several literals, each of which is either a variable or its negation. For example:

   $$(a \wedge b \wedge c) \vee (\bar{a} \wedge b) \vee (\bar{c} \wedge x)$$

   Given a DNF formula give a polynomial algorithm to check whether it is satisfiable or not. Why this does not imply $P = NP$.

3. Prove that the following problems are NP-complete.

   (a) Given an undirected graph G, does G have a spanning tree in which every node has degree at most 17?

   (b) Given an undirected graph G, does G have a spanning tree with at most 42 leaves?

CS 473: Undergraduate Algorithms, Spring 2009
HBS 2

1. Consider two horizontal lines l_1 and l_2 in the plane. There are n points on l_1 with x-coordinates $A = a_1, a_2, \ldots, a_n$ and there are n points on l_2 with x-coordinates $B = b_1, b_2, \ldots, b_n$. Design an algorithm to compute, given A and B, a largest set S of non-intersecting line segments subject to the following restrictions:

   (a) Any segment in S connects a_i to b_i for some $i (1 \leq i \leq n)$.

   (b) Any two segments in S do not intersect.

2. Consider a $2^n x 2^n$ chess board with one (arbitrarily chosen) square removed. Prove that any such chessboard can be tiled without gaps or overlaps by L-shaped pieces of 3 squares each. Can you give an algorithm to do the tiling?

3. Given a string of letters $Y = y_1 y_2 \ldots y_n$, a segmentation of Y is a partition of its letters into contiguous blocks of letters (also called words). Each word has a quality that can be computed by a given oracle (e.g. you can call *quality("meet")* to get the quality of the word "meet"). The quality of a segmentation is equal to the sum over the qualities of its words. Each call to the oracle takes linear time in terms of the argument; that is quality(S) takes $O(|S|)$.

   Using the given oracle, give an algorithm that takes a string Y and computes a segmentation of maximum total quality.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 3

1. Change your recursive solutions for the following problems to efficient algorithms (Hint: use dynamic programming!).

   (a) Consider two horizontal lines l_1 and l_2 in the plane. There are n points on l_1 with x-coordinates $A = a_1, a_2, \ldots, a_n$ and there are n points on l_2 with x-coordinates $B = b_1, b_2, \ldots, b_n$. Design an algorithm to compute, given A and B, a largest set S of non-intersecting line segments subject to the following restrictions:

      i. Any segment in S connects a_i to b_i for some $i (1 \leq i \leq n)$.

      ii. Any two segments in S do not intersect.

   (b) Given a string of letters $Y = y_1 y_2 \ldots y_n$, a segmentation of Y is a partition of its letters into contiguous blocks of letters (also called words). Each word has a quality that can be computed by a given oracle (e.g. you can call *quality("meet")* to get the quality of the word "meet"). The quality of a segmentation is equal to the sum over the qualities of its words. Each call to the oracle takes linear time in terms of the argument; that is quality(S) takes $O(|S|)$.

   Using the given oracle, give an algorithm that takes a string Y and computes a segmentation of maximum total quality.

2. Give a polynomial time algorithm which given two strings A and B returns the longest sequence S that is a subsequence of A and B.

3. Consider a rooted tree T. Assume the root has a message to send to all nodes. At the beginning only the root has the message. If a node has the message, it can forward it to one of its children at each time step. Design an algorithm to find the minimum number of time steps required for the message to be delivered to all nodes.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 3.5

1. Say you are given n jobs to run on a machine. Each job has a start time and an end time. If a job is chosen to be run, then it must start at its start time and end at its end time. Your goal is to accept as many jobs as possible, regardless of the job lengths, subject to the constraint that the processor can run at most one job at any given point in time. Provide an algorithm to do this with a running time that is polynomial in n. You may assume for simplicity that no two jobs have the same start or end times, but the start time and end time of two jobs can overlap.

2. Describe and analyze an algorithm that chooses one element uniformly at random from a data stream, without knowing the length of the stream in advance. Your algorithm should spend $O(1)$ time per stream element and use $O(1)$ space (not counting the stream itself).

3. Design and analyze an algorithm that return a permutation of the integers $\{1, 2, ..., n\}$ chosen uniformly at random.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 4

1. Let x and y be two elements of a set S whose ranks differ by exactly r. Prove that in a treap for S, the expected length of the unique path from x to y is $O(\log r)$

2. Consider the problem of making change for n cents using the least number of coins.

   (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

   (b) Suppose that the available coins have the values $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

   (c) Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution, show why.

   (d) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.

3. A heater is a sort of dual treap, in which the priorities of the nodes are given, but their search keys are generate independently and uniformly from the unit interval [0,1]. You can assume all priorities and keys are distinct. Describe algorithms to perform the operations INSERT and DELETEMIN in a heater. What are the expected worst-case running times of your algorithms? In particular, can you express the expected running time of INSERT in terms of the priority rank of the newly inserted item?

CS 473: Undergraduate Algorithms, Spring 2009
HBS 5

1. Recall that the staircase of a set of points consists of the points with no other point both above and to the right. Describe a method to maintain the staircase as new points are added to the set. Specifically, describe and analyze a data structure that stores the staircase of a set of points, and an algorithm $INSERT(x, y)$ that adds the point (x, y) to the set and returns $TRUE$ or $FALSE$ to indicate whether the staircase has changed. Your data structure should use $O(n)$ space, and your INSERT algorithm should run in $O(\log n)$ amortized time.

2. In some applications, we do not know in advance how much space we will require. So, we start the program by allocating a (dynamic) table of some fixed size. Later, as new objects are inserted, we may have to allocate a larger table and copy the previous table to it. So, we may need more than $O(1)$ time for copying. In addition, we want to keep the table size small enough, avoiding a very large table to keep only few items. One way to manage a dynamic table is by the following rules:

   (a) Double the size of the table if an item is inserted into a full table

   (b) Halve the table size if a deletion causes the table to become less than $1/4$ full

   Show that, in such a dynamic table we only need $O(1)$ amortized time, per operation.

3. Consider a stack data structure with the following operations:

   - PUSH(x): adds the element x to the top of the stack
   - POP: removes and returns the element that is currently on top of the stack (if the stack is non-empty)
   - SEARCH(x): repeatedly removes the element on top of the stack until x is found or the stack becomes empty

   What is the amortized cost of an operation?

CS 473: Undergraduate Algorithms, Spring 2009
HBS 6

1. Let G be a connected graph and let v be a vertex in G. Show that T is both a DFS tree and a BFS tree rooted at v, then $G = T$.

2. An Euler tour of a graph G is a walk that starts from a vertex v, visits every edge of G exactly once and gets back to v. Prove that G has an Euler tour if and only if all the vertices of G has even degrees. Can you give an efficient algorithm to find an Euler tour of such a graph.

3. You are helping a group of ethnographers analyze some oral history data they have collected by interviewing members of a village to learn about the lives of people lived there over the last two hundred years. From the interviews, you have learned about a set of people, all now deceased, whom we will denote $P_1, P_2, \ldots, P_n$. The ethnographers have collected several facts about the lifespans of these people, of one of the following forms:

   (a) P_i died before P_j was born.

   (b) P_i and P_j were both alive at some moment.

   Naturally, the ethnographers are not sure that their facts are correct; memories are not so good, and all this information was passed down by word of mouth. So they'd like you to determine whether the data they have collected is at least internally consistent, in the sense that there could have existed a set of people for which all the facts they have learned simultaneously hold.

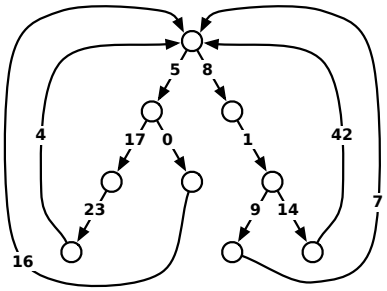
   Describe and analyze and algorithm to answer the ethnographers' problem. Your algorithm should either output possible dates of birth and death that are consistent with all the stated facts, or it should report correctly that no such dates exist.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 6.5

1. (a) Describe and analyze and algorithm to find the *second smallest spanning tree* of a given graph G, that is, the spanning tree of G with smallest total weight except for the minimum spanning tree.

   *(b) Describe and analyze an efficient algorithm to compute, given a weighted undirected graph G and an integer k, the k smallest spanning trees of G.

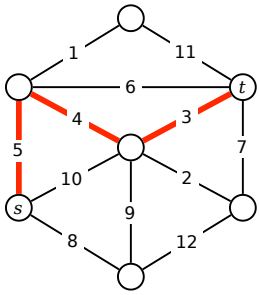
2. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight.



   (a) How much time would Dijkstra's algorithm require to compute the shortest path between two vertices u and v in a looped tree with n nodes?

   (b) Describe and analyze a faster algorithm.

3. Consider a path between two vertices s and t in an undirected weighted graph G. The *bottleneck length* of this path is the maximum weight of any edge in the path. The *bottleneck distance* between s and t is the minimum bottleneck length of any path from s to t. (If there are no paths from s to t, the bottleneck distance between s and t is ∞.)



The bottleneck distance between s and t is 5.

   Describe and analyze an algorithm to compute the bottleneck distance between *every* pair of vertices in an arbitrary undirected weighted graph. Assume that no two edges have the same weight.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 6.55

1. Suppose you are given a directed graph $G = (V, E)$ with non-negative edge lengths; $\ell(e)$ is the length of $e \in E$. You are interested in the shortest path distance between two given locations/nodes s and t. It has been noticed that the existing shortest path distance between s and t in G is not satisfactory and there is a proposal to add exactly one edge to the graph to improve the situation. The candidate edges from which one has to be chosen is given by $E' = \{e_1, e2, \ldots, e_k\}$ and you can assume that $E \cup E' = \emptyset$. The length of the e_i is $\alpha_i \geq 0$. Your goal is figure out which of these k edges will result in the most reduction in the shortest path distance from s to t. Describe an algorithm for this problem that runs in time $O((m+n)\log n + k)$ where $m = |E|$ and $n = |V|$. Note that one can easily solve this problem in $O(k(m+n)\log n)$ by running Dijkstra's algorithm k times, one for each G_i where G_i is the graph obtained by adding e_i to G.

2. Let G be an undirected graph with non-negative edge weights. Let s and t be two vertices such that the shortest path between s and t in G contains all the vertices in the graph. For each edge e, let $G \backslash e$ be the graph obtained from G by deleting the edge e. Design an $O(E \log V)$ algorithm that finds the shortest path distance between s and t in $G \backslash e$ for all e. [*Note that you need to output E distances, one for each graph* $G \backslash e$]

3. Given a Directed Acyclic Graph (DAG) and two vertices s and t you want to determine if there is an s to t path that includes at least k vertices.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 7

1. Let $G = (V, E)$ be a directed graph with non-negative capacities. Give an efficient algorithm to check whether there is a unique max-flow on G?

2. Let $G = (V, E)$ be a graph and $s, t \in V$ be two specific vertices of G. We call $(S, T = V \setminus S)$ an (s, t)-cut if $s \in S$ and $t \in T$. Moreover, it is a minimum cut if the sum of the capacities of the edges that have one endpoint in S and one endpoint in T equals the maximum (s, t)-flow. Show that, both intersection and union of two min-cuts is a min-cut itself.

3. Let $G = (V, E)$ be a graph. For each edge e let $d(e)$ be a demand value attached to it. A flow is feasible if it sends more than $d(e)$ through e. Assume you have an oracle that is capable of solving the maximum flow problem. Give efficient algorithms for the following problems that call the oracle at most once.

   (a) Find a feasible flow.

   (b) Find a feasible flow of minimum possible value.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 8

1. A box i can be specified by the values of its sides, say (i_1, i_2, i_3). We know all the side lengths are larger than 10 and smaller than 20 (i.e. $10 < i_1, i_2, i_3 < 20$). Geometrically, you know what it means for one box to nest in another: It is possible if you can rotate the smaller so that it fits inside the larger in each dimension. Of course, nesting is recursive, that is if i nests in j and j nests in k then i nests in k. After doing some nesting operations, we say a box is visible if it is not nested in any other one. Given a set of boxes (each specified by the lengthes of their sides) the goal is to find a set of nesting operations to minimize the number of visible boxes. Design and analyze an efficient algorithm to do this.

2. Let the number of papers submitted to a conference be n and the number of available reviewers be m. Each reviewer has a list of papers that he/she can review and each paper should be reviewed by three different persons. Also, each reviewer can review at most 5 papers. Design and analyze an algorithm to make the assignment or decide no feasible assignment exists.

3. Back in the euphoric early days of the Web, people liked to claim that much of the enormous potential in a company like Yahoo! was in the "eyeballs" - the simple fact that it gets millions of people looking at its pages every day. And further, by convincing people to register personal data with the site, it can show each user an extremely targeted advertisement whenever he or she visits the site, in away that TV networks or magazines could not hope to match. So if the user has told Yahoo! that he is a 20-year old computer science major from Cornell University, the site can throw up a banner ad for apartments in Ithaca, NY; on the other hand, if he is a 50-year-old investment banker from Greenwich, Connecticut, the site can display a banner ad pitching Lincoln Town Cars instead.

   But deciding on which ads to show to which people involves some serious computation behind the scenes. Suppose that the managers of a popular Web site have identified k distinct demographic groups $G_1, G_2, \ldots, G_k$. (These groups can overlap; for example G_1 can be equal to all residents of New York State, and G_2 can be equal to all people with a degree in computer science.) The site has contracts with m different advertisers, to show a certain number of copies of their ads to users of the site. Here is what the contract with the i^{th} advertiser looks like:

   (a) For a subset $X_i \subset \{G_1, \ldots, G_k\}$ of the demographic groups, advertiser i wants its ads shown only to users who belong to at least one of the demographic groups in the set X_i

   (b) For a number r_i , advertiser i wants its ads shown to at least r_i users each minute.

   Now, consider the problem of designing a good advertising policy - a way to show a single ad to each user of the site. Suppose at a given minute, there are n users visiting the site. Because we have registration information on each of these users, we know that user j(for $j = 1, 2, \ldots, n$) belongs to a subset $U_j \subset \{G_1, \ldots, G_k\}$ of the demographic groups. The problem is: is there a way to show a single ad to each user so that the site's contracts with each of the m advertisers is satisfied for this minute? (That is, for each $i = 1, 2, \ldots, m$, at least r_i of the n users, each belonging to at least one demographic group in X_i , are shown an ad provided by advertiser i.)

   Give an efficient algorithm to decide if this is possible, and if so, to actually choose an ad to show each user.

CS 473: Undergraduate Algorithms, Spring 2009
HBS 9

1. Prove that any algorithm to merge two sorted arrays, each of size n, requires at least $2n - 1$ comparisons.

2. Suppose you want to determine the largest number in an n-element set $X = \{x_1, x_2, \ldots, x_n\}$, where each element x_i is an integer between 1 and $2^m - 1$. Describe an algorithm that solves this problem in $O(n + m)$ steps, where at each step, your algorithm compares one of the elements x_i with a *constant*. In particular, your algorithm must never actually compare two elements of X! *[Hint: Construct and maintain a nested set of 'pinning intervals' for the numbers that you have not yet removed from consideration, where each interval but the largest is either the upper half or lower half of the next larger block.]*

3. Let P be a set of n points in the plane. The staircase of P is the set of all points in the plane that have at least one point in P both above and to the right. Prove that computing the staircase requires at least $\Omega(n \log n)$ comparisons in two ways,

   (a) Reduction from sorting.

   (b) Directly.

You have 90 minutes to answer four of the five questions.
Write your answers in the separate answer booklet.
You may take the question sheet with you when you leave.

1. Each of these ten questions has one of the following five answers:

   A: $\Theta(1)$ B: $\Theta(\log n)$ C: $\Theta(n)$ D: $\Theta(n \log n)$ E: $\Theta(n^2)$

   Choose the correct answer for each question. Each correct answer is worth $+1$ point; each incorrect answer is worth $-1/2$ point; each "I don't know" is worth $+1/4$ point. Your score will be rounded to the nearest *non-negative* integer.

   (a) What is $\sum_{i=1}^{n} \frac{n}{i}$?

   (b) What is $\sqrt{\sum_{i=1}^{n} i}$?

   (c) How many digits are required to write 3^n in decimal?

   (d) What is the solution to the recurrence $D(n) = D(n/\pi) + \sqrt{2}$?

   (e) What is the solution to the recurrence $E(n) = E(n - \sqrt{2}) + \pi$?

   (f) What is the solution to the recurrence $F(n) = 4F(n/2) + 3n$?

   (g) What is the worst-case time to search for an item in a binary search tree?

   (h) What is the worst-case running time of quicksort?

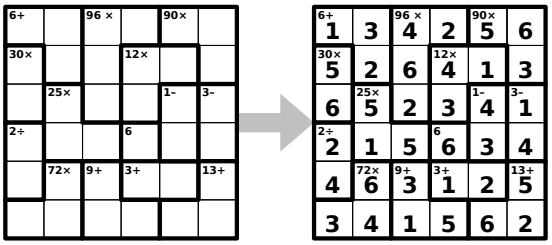
   (i) Let $H[1..n, 1..n]$ be a fixed array of numbers. Consider the following recursive function:

   $$Glub(i,j) = \begin{cases} 0 & \text{if } i = 0 \\ \infty & \text{if } i > n \text{ or } j = 0 \\ \max\{Glub(i-1,j),\ H[i,j] + Glub(i+1,j-1)\} & \text{otherwise} \end{cases}$$

   How long does it take to compute $Glub(n,n)$ using dynamic programming?

   (j) What is the running time of the fastest possible algorithm to solve KenKen puzzles?

   A KenKen puzzle is a 6×6 grid, divided into regions called *cages*. Each cage is labeled with a numerical *value* and an arithmetic *operation*: $+$, $-$, $\times$, or $\div$. (The operation can be omitted if the cage consists of a single cell.) The goal is to place an integer between 1 and 6 in each grid cell, so that no number appears twice in any row or column, and the numbers inside each cage can be combined using *only* that cage's operation to obtain that cage's value. The solution is guaranteed to be unique.

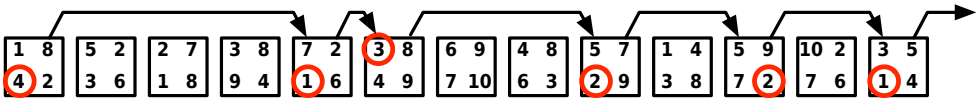


A Kenken puzzle and its solution

2. (a) Suppose $A[1..n]$ is an array of n distinct integers, sorted so that $A[1] < A[2] < \cdots < A[n]$. Each integer $A[i]$ could be positive, negative, or zero. Describe an efficient algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. An algorithm that runs in $\Theta(n)$ time is worth at most 3 points.

   (b) Now suppose $A[1..n]$ is a sorted array of n distinct ***positive*** integers. Describe an even faster algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. *[Hint: This is **really** easy!]*

3. *Moby Selene* is a solitaire game played on a row of n squares. Each square contains four positive integers. The player begins by placing a token on the leftmost square. On each move, the player chooses one of the numbers on the token's current square, and then moves the token that number of squares to the right. The game ends when the token moves past the rightmost square. The object of the game is to make as many moves as possible before the game ends.



A Moby Selene puzzle that allows six moves. (This is ***not*** the longest legal sequence of moves.)

   (a) ***Prove*** that the obvious greedy strategy (always choose the smallest number) does not give the largest possible number of moves for every Moby Selene puzzle.

   (b) Describe and analyze an efficient algorithm to find the largest possible number of legal moves for a given Moby Selene puzzle.

4. Consider the following algorithm for finding the largest element in an unsorted array:

```
RandomMax(A[1..n]):
  max ← ∞
  for i ← 1 to n in random order
      if A[i] > max
          max ← A[i]    (⋆)
  return max
```

   (a) In the worst case, how many times does RandomMax execute line $(\star)$?

   (b) What is the ***exact*** probability that line $(\star)$ is executed during the last iteration of the for loop?

   (c) What is the ***exact*** expected number of executions of line $(\star)$? (A correct $\Theta(\,)$ bound is worth half credit.)

5. *This question is taken directly from HBS 0.* Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

   Prove that any finite set of pigeons can be arranged in a row from left to right so that every pigeon pecks the pigeon immediately to its left. Pretty please.

You have 90 minutes to answer four of the five questions.
Write your answers in the separate answer booklet.
You may take the question sheet with you when you leave.

1. Recall that a ***tree*** is a connected graph with no cycles. A graph is ***bipartite*** if we can color its vertices black and white, so that every edge connects a white vertex to a black vertex.

   (a) ***Prove*** that every tree is bipartite.

   (b) Describe and analyze a fast algorithm to determine whether a given graph is bipartite.

2. Describe and analyze an algorithm SHUFFLE($A[1..n]$) that randomly permutes the input array A, so that each of the $n!$ possible permutations is equally likely. You can assume the existence of a subroutine RANDOM(k) that returns a random integer chosen uniformly between 1 and k in $O(1)$ time. For full credit, your SHUFFLE algorithm should run in $O(n)$ time. *[Hint: This problem appeared in HBS 3½.]*

3. Let G be an undirected graph with weighted edges.

   (a) Describe and analyze an algorithm to compute the *maximum* weight spanning tree of G.

   (b) A ***feedback edge set*** of G is a subset F of the edges such that every cycle in G contains at least one edge in F. In other words, removing every edge in F makes G acyclic. Describe and analyze a fast algorithm to compute the minimum weight feedback edge set of G.

   [Hint: Don't reinvent the wheel!]

4. Let $G = (V, E)$ be a connected directed graph with non-negative edge weights, let s and t be vertices of G, and let H be a subgraph of G obtained by deleting some edges. Suppose we want to reinsert exactly one edge from G back into H, so that the shortest path from s to t in the resulting graph is as short as possible. Describe and analyze an algorithm to choose the best edge to reinsert. For full credit, your algorithm should run in $O(E \log V)$ time. *[Hint: This problem appeared in HBS 6¾.]*

5. Describe and analyze an efficient data structure to support the following operations on an array $X[1..n]$ as quickly as possible. Initially, $X[i] = 0$ for all i.

   - Given an index i such that $X[i] = 0$, set $X[i]$ to 1.
   - Given an index i, return $X[i]$.
   - Given an index i, return the smallest index $j \geq i$ such that $X[j] = 0$, or report that no such index exists.

   For full credit, the first two operations should run in *worst-case constant* time, and the amortized cost of the third operation should be as small as possible.

You have 180 minutes to answer six of the seven questions.
Write your answers in the separate answer booklet.
You may take the question sheet with you when you leave.

1. SUBSETSUM and PARTITION are two closely related NP-hard problems, defined as follows.

   SUBSETSUM: Given a set X of positive integers and a positive integer k, does X have a subset whose elements sum up to k?

   PARTITION: Given a set Y of positive integers, can Y be partitioned into two subsets whose sums are equal?

   (a) **[2 pts]** ***Prove*** that PARTITION and SUBSETSUM are both in NP.

   (b) **[1 pt]** Suppose you already know that SUBSETSUM is NP-hard. Which of the following arguments could you use to prove that PARTITION is NP-hard? ***You do not need to justify your answer*** — just answer ① or ②.

   ① Given a set X and an integer k, construct a set Y in polynomial time, such that PARTITION(Y) is true if and only if SUBSETSUM(X, k) is true.

   ② Given a set Y, construct a set X and an integer k in polynomial time, such that PARTITION(Y) is true if and only if SUBSETSUM(X, k) is true.

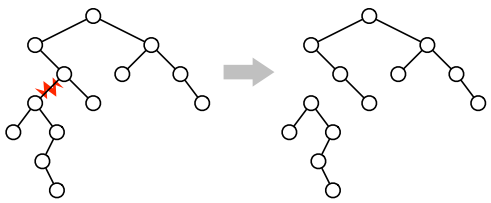
   (c) **[3 pts]** Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM. ***You do not need to prove that your reduction is correct.***

   (d) **[4 pts]** Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION. ***You do not need to prove that your reduction is correct.***

2. (a) **[4 pts]** For any node v in a binary tree, let $size(v)$ denote the number of nodes in the subtree rooted at v. Let k be an arbitrary positive number. ***Prove*** that every binary tree with at least k nodes contains a node v such that $k \leq size(v) \leq 2k$.

   (b) **[2 pts]** Removing any edge from an n-node binary tree T separates it into two smaller binary trees. An edge is called a ***balanced separator*** if both of these subtrees have at least $n/3$ nodes (and therefore at most $2n/3$ nodes). ***Prove*** that every binary tree with more than one node has a balanced separator. *[Hint: Use part (a).]*

   (c) **[4 pts]** Describe and analyze an algorithm to find a balanced separator in a given binary tree. *[Hint: Use part (a).]*



Removing a balanced separator from a binary tree.

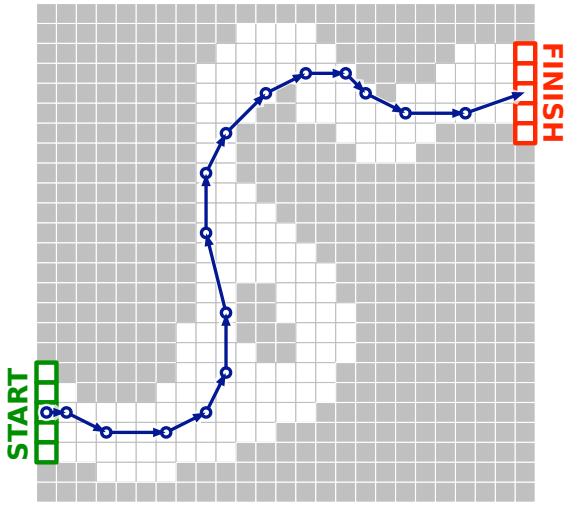
3. ***Racetrack*** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade.[1] The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

   Each car has a *position* and a *velocity*, both with integer x- and y-coordinates. The initial position is a point on the starting line, chosen by the player; the initial velocity is always $(0, 0)$. At each step, the player optionally increments or decrements either or both coordinates of the car's velocity; in other words, each component of the velocity can change by ***at most* 1** in a single step. The car's new position is then determined by adding the new velocity to the car's previous position. The new position must be inside the track; otherwise, the car crashes and that player loses the race. The race ends when the first car reaches a position ***on*** the finish line.

   Suppose the racetrack is represented by an $n \times n$ array of bits, where each 0 bit represents a grid point inside the track, each 1 bit represents a grid point outside the track, the 'starting line' is the first column, and the 'finish line' is the last column.

   Describe and analyze an algorithm to find the minimum number of steps required to move a car from the starting line to the finish line of a given racetrack. *[Hint: Build a graph. What are the vertices? What are the edges? What problem is this?]*

velocity	position
$(0,0)$	$(1,5)$
$(1,0)$	$(2,5)$
$(2,-1)$	$(4,4)$
$(3,0)$	$(7,4)$
$(2,1)$	$(9,5)$
$(1,2)$	$(10,7)$
$(0,3)$	$(10,10)$
$(-1,4)$	$(9,14)$
$(0,3)$	$(9,17)$
$(1,2)$	$(10,19)$
$(2,2)$	$(12,21)$
$(2,1)$	$(14,22)$
$(2,0)$	$(16,22)$
$(1,-1)$	$(17,21)$
$(2,-1)$	$(19,20)$
$(3,0)$	$(22,20)$
$(3,1)$	$(25,21)$



A 16-step Racetrack run, on a 25×25 track.

4. A palindrome is any string that is exactly the same as its reversal, like `I`, or `DEED`, or `RACECAR`, or `AMANAPLANACATACANALPANAMA`. Describe and analyze an algorithm to find the length of the longest *subsequence* of a given string that is also a palindrome.

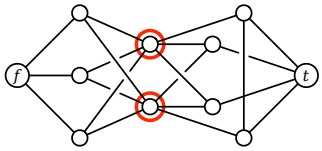
   For example, the longest palindrome subsequence of `MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM` is `MHYMRORMYHM`, so given that string as input, your algorithm should output the number 11.

[1] The actual game is a bit more complicated than the version described here.

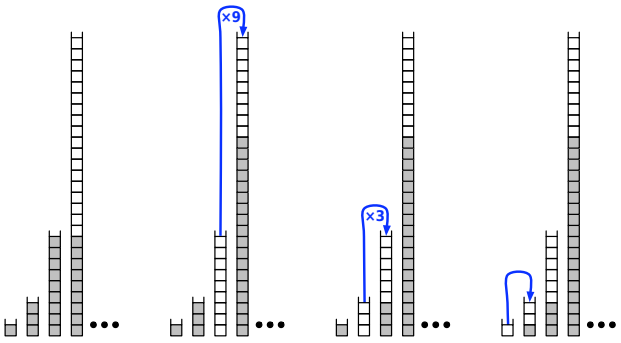
5. The Island of Sodor is home to a large number of towns and villages, connected by an extensive rail network. Recently, several cases of a deadly contagious disease (either swine flu or zombies; reports are unclear) have been reported in the village of Ffarquhar. The controller of the Sodor railway plans to close down certain railway stations to prevent the disease from spreading to Tidmouth, his home town. No trains can pass through a closed station. To minimize expense (and public notice), he wants to close down as few stations as possible. However, he cannot close the Ffarquhar station, because that would expose him to the disease, and he cannot close the Tidmouth station, because then he couldn't visit his favorite pub.

   Describe and analyze an algorithm to find the minimum number of stations that must be closed to block all rail travel from Ffarquhar to Tidmouth. The Sodor rail network is represented by an undirected graph, with a vertex for each station and an edge for each rail connection between two stations. Two special vertices f and t represent the stations in Ffarquhar and Tidmouth.

   For example, given the following input graph, your algorithm should return the number 2.



6. A *multistack* consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first pop all the elements off S_i and push them onto stack S_{i+1} to make room. (Thus, if S_{i+1} is already full, we first recursively move all its members to S_{i+2}.) Moving a single element from one stack to the next takes $O(1)$ time.



Making room for one new element in a multistack.

   (a) In the worst case, how long does it take to push one more element onto a multistack containing n elements?

   (b) ***Prove*** that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack.

7. Recall the problem 3COLOR: Given a graph, can we color each vertex with one of 3 colors, so that every edge touches two different colors? We proved in class that 3COLOR is NP-hard.

   Now consider the related problem 12COLOR: Given a graph, can we color each vertex with one of twelve colors, so that every edge touches two different colors? ***Prove*** that 12COLOR is NP-hard.

You may assume the following problems are NP-hard:

CircuitSat: Given a boolean circuit, are there any input values that make the circuit output True?

PlanarCircuitSat: Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

3Sat: Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

MaxIndependentSet: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MaxClique: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MinVertexCover: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MinSetCover: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MinHittingSet: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3Color: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

HamiltonianCycle: Given a graph G, can is there a cycle in G that visits every vertex once?

HamiltonianPath: Given a graph G, can is there a path in G that visits every vertex once?

DoubleHamiltonianCycle: Given a graph G, can is there a closed walk in G that visits every vertex twice?

DoubleHamiltonianPath: Given a graph G, can is there an open walk in G that visits every vertex twice?

MinDegreeSpanningTree: Given an undirected graph G, what is the minimum degree of any spanning tree of G?

MinLeavesSpanningTree: Given an undirected graph G, what is the minimum number of leaves in any spanning tree of G?

TravelingSalesman: Given a graph G with weighted edges, what is the minimum cost of any Hamiltonian path/cycle in G?

LongestPath: Given a graph G with weighted edges and two vertices s and t, what is the length of the longest *simple* path from s to t in G?

SubsetSum: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

Partition: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3Partition: Given a set X of n positive integers, can X be partitioned into $n/3$ three-element subsets, all with the same sum?

Minesweeper: Given a Minesweeper configuration and a particular square x, is it safe to click on x?

Tetris: Given a sequence of N Tetris pieces and a partially filled $n \times k$ board, is it possible to play every piece in the sequence without overflowing the board?

Sudoku: Given an $n \times n$ Sudoku puzzle, does it have a solution?

KenKen: Given an $n \times n$ Ken-Ken puzzle, does it have a solution?

CS 473: Undergraduate Algorithms, Spring 2010
Homework 0

Due Tuesday, January 26, 2009 **in class**

- This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** The early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
- Each student must submit individual solutions for these homework problems. For all future homeworks, groups of up to three students may submit (or present) a single group solution for each problem.
- Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. In particular:
  - Submit five separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page. Please do *not* staple everything together.
  - You may use any source at your disposal—paper, electronic, or human—but you ***must*** write your solutions in your own words, and you ***must*** cite every source that you use.
  - Unless explicitly stated otherwise, ***every*** homework problem requires a proof.
  - Answering "I don't know" to any homework or exam problem (except for extra credit problems) is worth 25% partial credit.
  - Algorithms or proofs containing phrases like "and so on" or "repeat this process for all n", instead of an explicit loop, recursion, or induction, will receive 0 points.

---

1. (a) ***Write the sentence "I understand the course policies."***

   (b) **[5 pts]** Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Assume reasonable but nontrivial base cases if none are given. ***Do not submit proofs***—just a list of five functions—but you should do them anyway, just for practice.

   - $A(n) = 3A(n-1) + 1$
   - $B(n) = B(n-5) + 2n - 3$
   - $C(n) = 4C(n/2) + \sqrt{n}$
   - $D(n) = 3D(n/3) + n^2$
   - $E(n) = E(n-1)^2 - E(n-2)^2$, where $E(0) = 0$ and $E(1) = 1$ *[Hint: This is easy!]*

   (c) **[5 pts]** Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. ***Do not submit proofs***—just a sorted list of 16 functions—but you should do them anyway, just for practice.

   Write $f(n) \ll g(n)$ to indicate that $f(n) = o(g(n))$, and write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. We use the notation $\lg n = \log_2 n$.

n	$\lg n$	$\sqrt{n}$	5^n
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$5^{\sqrt{n}}$	$\sqrt{5^n}$
$5^{\lg n}$	$\lg(5^n)$	$5^{\lg \sqrt{n}}$	$5\sqrt{\lg n}$
$\sqrt{5^{\lg n}}$	$\lg(5^{\sqrt{n}})$	$\lg \sqrt{5^n}$	$\sqrt{\lg(5^n)}$

2. **[CS 225 Spring 2009]** Suppose we build up a binary search tree by inserting elements one at a time from the set $\{1, 2, 3, \ldots, n\}$, starting with the empty tree. The structure of the resulting binary search tree depends on the order that these elements are inserted; every insertion order leads to a different n-node binary search tree.

   Recall that the *depth* of a leaf ℓ in a binary search tree is the number of *edges* between ℓ and the root, and the depth of a binary tree is the maximum depth of its leaves.

   (a) What is the maximum possible depth of an n-node binary search tree? Give an *exact* answer, and prove that it is correct.

   (b) *Exactly* how many different insertion orders result in an n-node binary search tree with maximum possible depth? Prove your answer is correct. *[Hint: Set up and solve a recurrence. Don't forget to prove that recurrence counts what you want it to count.]*

3. **[CS 173 Spring 2009]** A *binomial tree of order k* is defined recursively as follows:

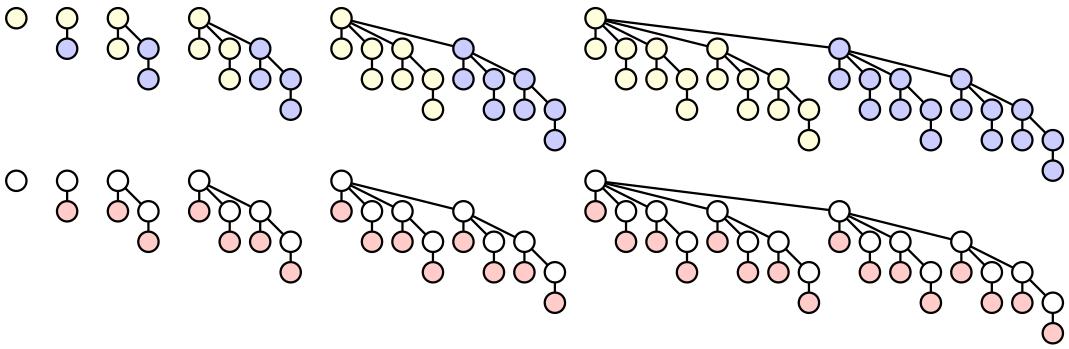
   - A binomial tree of order 0 is a single node.
   - For all $k > 0$, a binomial tree of order k consists of two binomial trees of order $k-1$, with the root of one tree connected as a new child of the root of the other. (See the figure below.)

   Prove the following claims:

   (a) For all non-negative integers k, a binomial tree of order k has exactly 2^k nodes.

   (b) For all positive integers k, attaching a leaf to every node in a binomial tree of order $k-1$ results in a binomial tree of order k.

   (c) For all non-negative integers k and d, a binomial tree of order k has exactly $\binom{k}{d}$ nodes with depth d.



Binomial trees of order 0 through 5.
Top row: the recursive definition. Bottom row: the property claimed in part (b).

4. **[CS 373 Fall 2009]** For any language $L \in \Sigma^*$, let

$$Rotate(L) := \left\{ w \in \Sigma^* \;\middle|\; w = xy \text{ and } yx \in L \text{ for some strings } x, y \in \Sigma^* \right\}$$

   For example, $Rotate(\{\texttt{OOK!}, \texttt{OOKOOK}\}) = \{\texttt{OOK!}, \texttt{OK!O}, \texttt{K!OO}, \texttt{!OOK}, \texttt{OOKOOK}, \texttt{OKOOKO}, \texttt{KOOKOO}\}$.

   Prove that if L is a regular language, then $Rotate(L)$ is also a regular language. *[Hint: Remember the power of nondeterminism.]*

5. Herr Professor Doktor Georg von den Dschungel has a 24-node binary tree, in which every node is labeled with a unique letter of the German alphabet, which is just like the English alphabet with four extra letters: **Ä**, **Ö**, **Ü**, and **ß**. (Don't confuse these with **A**, **O**, **U**, and **B!**) Preorder and postorder traversals of the tree visit the nodes in the following order:

   - Preorder: **B K Ü E H L Z I Ö R C ß T S O A Ä D F M N U G**
   - Postorder: **H I Ö Z R L E C Ü S O T A ß K D M U G N F Ä B**

   (a) List the nodes in George's tree in the order visited by an inorder traversal.

   (b) Draw George's tree.

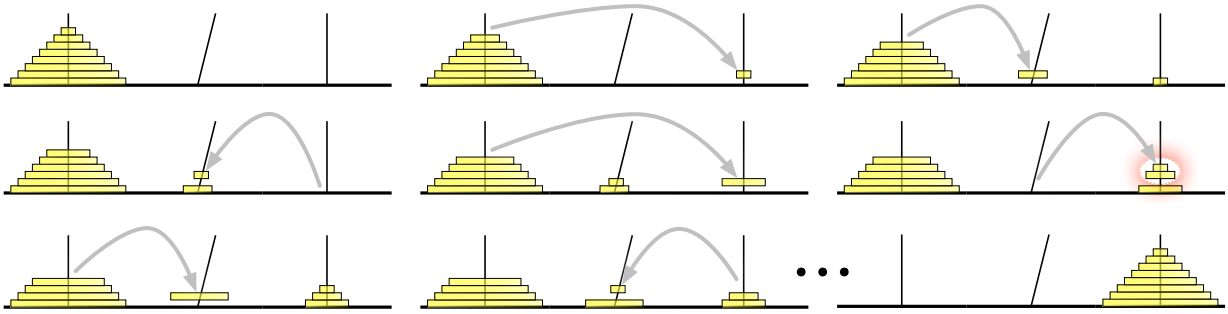
*6. ***[Extra credit]*** You may be familiar with the story behind the famous Tower of Hanoï puzzle, as related by Henri de Parville in 1884:

> In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disc resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the Tower of Bramah. Day and night unceasingly the priests transfer the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple, and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

A less familiar chapter in the temple's history is its brief relocation to Pisa in the early 13th century. The relocation was organized by the wealthy merchant-mathematician Leonardo Fibonacci, at the request of the Holy Roman Emperor Frederick II, who had heard reports of the temple from soldiers returning from the Crusades. The Towers of Pisa and their attendant monks became famous, helping to establish Pisa as a dominant trading center on the Italian peninsula.

Unfortunately, almost as soon as the temple was moved, one of the diamond needles began to lean to one side. To avoid the possibility of the leaning tower falling over from too much use, Fibonacci convinced the priests to adopt a more relaxed rule: ***Any number of disks on the leaning needle can be moved together to another needle in a single move.*** It was still forbidden to place a larger disk on top of a smaller disk, and disks had to be moved one at a time *onto* the leaning needle or between the two vertical needles.

Thanks to Fibonacci's new rule, the priests could bring about the end of the universe somewhat faster from Pisa then they could than could from Benares. Fortunately, the temple was moved from Pisa back to Benares after the newly crowned Pope Gregory IX excommunicated Frederick II, making the local priests less sympathetic to hosting foreign heretics with strange mathematical habits. Soon afterward, a bell tower was erected on the spot where the temple once stood; it too began to lean almost immediately.



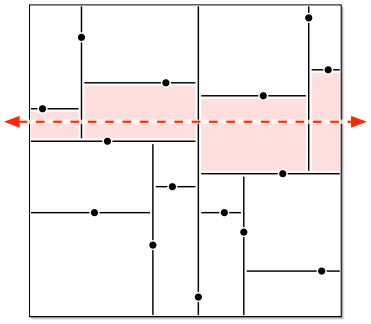
The Towers of Pisa. In the fifth move, two disks are taken off the leaning needle.

Describe an algorithm to transfer a stack of n disks from one *vertical* needle to the other *vertical* needle, using the smallest possible number of moves. *Exactly* how many moves does your algorithm perform?

- For this and all future homeworks, groups of up to three students can submit (or present) a single common solution. Please remember to write the names of all group members on every page.
- **Please fill out the online input survey linked from the course web page no later than Thursday, January 28.** Among other things, this survey asks you to identify the other members of your HW1 group, so that we can partition the class into presentation clusters without breaking up your group. We will announce the presentation clusters on Friday, January 29.
- Students in **Cluster 1** will present their solutions to Jeff or one of the TAs, on Tuesday or Wednesday of the due date (February 2 or February 3), instead of submitting written solutions. **Each homework group in Cluster 1 must sign up for a 30-minute time slot no later than Monday, February 1.** Signup sheets will be posted at 3303 Siebel Center ('The Theory Lab') later this week. Please see the course web page for more details.

---

1. Suppose we have n points scattered inside a two-dimensional box. A *kd-tree* recursively subdivides the points as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line partitions the rest of the interior points *as evenly as possible* by passing through a median point in the interior of the box (*not* on its boundary). If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.



A kd-tree for 15 points. The dashed line crosses the four shaded cells.

   (a) How many cells are there, as a function of n? Prove your answer is correct.

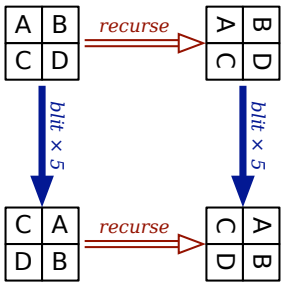
   (b) In the worst case, *exactly* how many cells can a horizontal line cross, as a function of n? Prove your answer is correct. Assume that $n = 2^k - 1$ for some integer k.

   (c) Suppose we have n points stored in a kd-tree. Describe and analyze an algorithm that counts the number of points above a horizontal line (such as the dashed line in the figure) as quickly as possible. *[Hint: Use part (b).]*

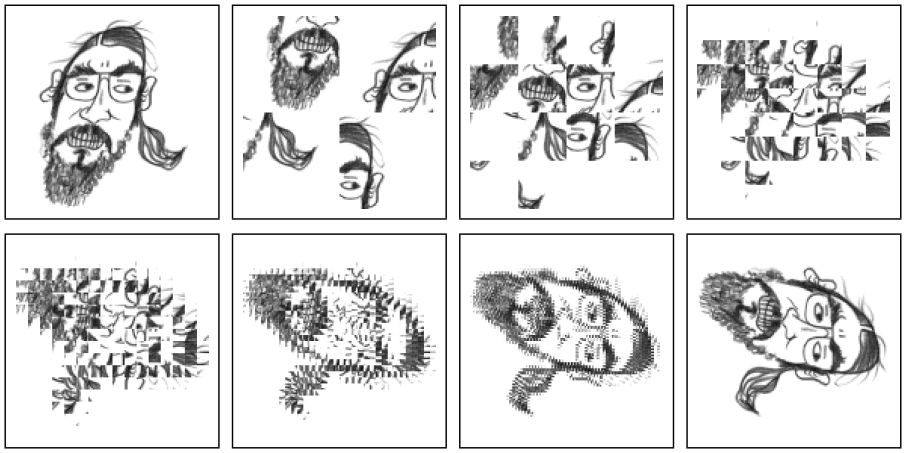
   (d) Describe an analyze an efficient algorithm that counts, given a kd-tree storing n points, the number of points that lie inside a rectangle R with horizontal and vertical sides. *[Hint: Use part (c).]*

2. Most graphics hardware includes support for a low-level operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixel map (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

   Suppose we want to rotate an $n \times n$ pixel map 90° clockwise. One way to do this, at least when n is a power of two, is to split the pixel map into four $n/2 \times n/2$ blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we could *first* recursively rotate the blocks and *then* blit them into place.



Two algorithms for rotating a pixel map.
Solid arrows indicate blitting the blocks into place; hollow arrows indicate recursively rotating the blocks.



The first rotation algorithm (blit then recurse) in action.

(a) Prove that both versions of the algorithm are correct when n is a power of two.

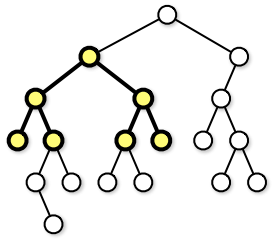
(b) *Exactly* how many blits does the algorithm perform when n is a power of two?

(c) Describe how to modify the algorithm so that it works for arbitrary n, not just powers of two. How many blits does your modified algorithm perform?

(d) What is your algorithm's running time if a $k \times k$ blit takes $O(k^2)$ time?

(e) What if a $k \times k$ blit takes only $O(k)$ time?

3. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

CS 473: Undergraduate Algorithms, Spring 2010
Homework 2

Written solutions due Tuesday, February 9, 2010 at noon

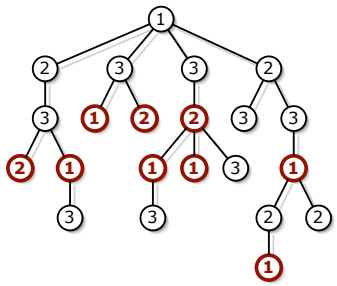
- Roughly 1/3 of the students will give oral presentations of their solutions to the TAs. ***You should have received an email telling you whether you are expected to present this homework.*** Please see the course web page for further details.
- Groups of up to three students may submit a common solution. Please clearly write every group member's name and NetID on every page of your submission. Please start your solution to each numbered problem on a new sheet of paper. Please ***don't*** staple solutions for different problems together.

---

1. A ***palindrome*** is a string that reads the same forwards and backwards, like `x`, `pop`, `noon`, `redivider`, or `"sator arepo tenet opera rotas"`, Describe and analyze an algorithm to find the length of the longest subsequence of a given string that is also a palindrome. For example, the longest palindrome subsequence of **M**A**H**D**Y**NA**M**ICP**RO**G**R**AMZLET**M**ESHOW**Y**OUT**H**E**M** is MHYMRORMYHM, so given that string as input, your algorithm should output the number 11.

2. Oh, no! You have been appointed as the gift czar for Giggle, Inc.'s annual mandatory holiday party! The president of the company, who is certifiably insane, has declared that every Giggle employee must receive one of three gifts: (1) an all-expenses-paid six-week vacation anywhere in the world, (2) an all-the-pancakes-you-can-eat breakfast for two at Jumping Jack Flash's Flapjack Stack Shack, or (3) a burning paper bag full of dog poop. Corporate regulations prohibit any employee from receiving the same gift as his/her direct supervisor. Any employee who receives a better gift than his/her direct supervisor will almost certainly be fired in a fit of jealousy. How do you decide what gifts everyone gets if you want to minimize the number of people that get fired?

   More formally, suppose you are given a rooted tree T, representing the company hierarchy. You want to label each node in T with an integer 1, 2, or 3, such that every node has a different label from its parent.. The *cost* of an labeling is the number of nodes that have smaller labels than their parents. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree T. (Your algorithm does *not* have to compute the actual best labeling—just its cost.)



A tree labeling with cost 9. Bold nodes have smaller labels than their parents.
This is ***not*** the optimal labeling for this tree.

3. After graduating from UIUC, you have decided to join the Wall Street Bank ***Boole Long Live***. The managing director of the bank, Eloob Egroeg, is a genius mathematician who worships George Boole[1] every morning before leaving for the office. The first day of every hired employee is a 'solve-or-die' day where s/he has to solve one of the problems posed by Eloob within 24 hours. Those who fail to solve the problem are fired immediately!

   Entering into the bank for the first time, you notice that the offices of the employees are organized in a straight row, with a large "T" or "F" written on the door of each office. Furthermore, between each adjacent pair of offices, there is a board marked by one of the symbols $\wedge, \vee$, or $\oplus$. When you ask about these arcane symbols, Eloob confirms that T and F represent the boolean values 'true' and 'false', and the symbols on the boards represent the standard boolean operators AND, OR, and XOR. He also explains that these letters and symbols describe whether certain combinations of employees can work together successfully. At the start of any new project, Eloob hierarchically clusters his employees by adding parentheses to the sequence of symbols, to obtain an unambiguous boolean expression. The project is successful if this parenthesized boolean expression evaluates to T.

   For example, if the bank has three employees, and the sequence of symbols on and between their doors is $T \wedge F \oplus T$, there is exactly one successful parenthesization scheme: $(T \wedge (F \oplus T))$. However, if the list of door symbols is $F \wedge T \oplus F$, there is no way to add parentheses to make the project successful.

   Eloob finally poses your solve-or-die question: Describe and algorithm to decide whether a given sequence of symbols can be parenthesized so that the resulting boolean expression evaluates to T. The input to your algorithm is an array $S[0..2n]$, where $S[i] \in \{T, F\}$ when i is even, and $S[i] \in \{\vee, \wedge, \oplus\}$ when i is odd.

[1] 1815-1864, The inventor of Boolean Logic

- For this and all future homeworks, groups of up to three students can submit (or present) a single common solution. Please remember to write the names of all group members on every page.
- Students in **Cluster 3** will present their solutions to Jeff or one of the TAs, on Tuesday or Wednesday of the due date (February 16 or February 17), instead of submitting written solutions. **Each homework group in Cluster 3 must sign up for a 30-minute time slot no later than Monday, February 15.** Signup sheets will be posted at 3304 Siebel Center ('The Theory Lab') later this week. Please see the course web page for more details.

---

1. You saw in class a correct greedy algorithm for finding the maximum number of non-conflicting courses from a given set of possible courses. This algorithm repeatedly selects the class with the earliest completion time that does not conflict with any previously selected class.

   Below are four alternative greedy algorithms. For each algorithm, either prove that the algorithm constructs an optimal schedule, or give a concrete counterexample showing that the algorithm is suboptimal.

   (a) Choose the course that *ends latest*, discard all conflicting classes, and recurse.

   (b) Choose the course that *starts first*, discard all conflicting classes, and recurse.

   (c) Choose the course with *shortest duration*, discard all conflicting classes, and recurse.

   (d) Choose a course that *conflicts with the fewest other courses* (breaking ties arbitrarily), discard all conflicting classes, and recurse.

2. You have been given the task of designing an algorithm for vending machines that computes the smallest number of coins worth any given amount of money. Your supervisors at The Area 51 Soda Company are anticipating a hostile takeover of earth by an advanced alien race that uses an unknown system of currency. So your algorithm must be as general as possible so that it will work with the alien system, whatever it turns out to be.

   Given a quantity of money x, and a set of coin denominations $b_1, \ldots, b_k$, your algorithm should compute how to make change for x with the fewest number of coins. For example, if you use the US coin denominations (1¢, 5¢, 10¢, 25¢, 50¢, and 100¢), the optimal way to make 17¢ in change uses 4 coins: one dime (10¢), one nickel (5¢), and two pennies (1¢).

   (a) Show that the following greedy algorithm does *not* work for all currency systems: If $x = 0$, do nothing. Otherwise, find the largest denomination $c \leq x$, issue one c-cent coin, and recursively give $x - c$ cents in change.

   (b) Now suppose that the system of currency you are concerned with only has coins in powers of some base b. That is, the coin denominations are $b^0, b^1, b^2, \ldots, b^k$. Show that the greedy algorithm described in part (a) does make optimal change in this currency system.

   (c) Describe and analyze an algorithm that computes optimal change for *any* set of coin denominations. (You may assume the aliens' currency system includes a 1-cent coin, so that making change is always possible.)

3. Suppose you have just purchased a new type of hybrid car that uses fuel extremely efficiently, but can only travel 100 miles on a single battery. The car's fuel is stored in a single-use battery, which must be replaced after at most 100 miles. The actual fuel is virtually free, but the batteries are expensive and can only be installed by licensed battery-replacement technicians. Thus, even if you decide to replace your battery early, you must still pay full price for the new battery to be installed. Moreover, because these batteries are in high demand, no one can afford to own more than one battery at a time.

   Suppose you are trying to get from San Francisco to New York City on the new Inter-Continental Super-Highway, which runs in a direct line between these two cities. There are several fueling stations along the way; each station charges a different price for installing a new battery. Before you start your trip, you carefully print the Wikipedia page listing the locations and prices of every fueling station on the ICSH. Given this information, how do you decide the best places to stop for fuel?

   More formally, suppose you are given two arrays $D[1..n]$ and $C[1..n]$, where $D[i]$ is the distance from the start of the highway to the ith station, and $C[i]$ is the cost to replace your battery at the ith station. Assume that your trip starts and ends at fueling stations (so $D[1] = 0$ and $D[n]$ is the total length of your trip), and that your car starts with an empty battery (so you must install a new battery at station 1).

   (a) Describe and analyze a greedy algorithm to find the minimum number of refueling stops needed to complete your trip. Don't forget to prove that your algorithm is correct.

   (b) But what you really want to minimize is the total *cost* of travel. Show that your greedy algorithm in part (a) does *not* produce an optimal solution when extended to this setting.

   (c) Describe a dynamic programming algorithm to compute the locations of the fuel stations you should stop at to minimize the cost of travel.

1. Suppose we want to write an efficient function SHUFFLE(n) that returns a permutation of the set $\{1, 2, \ldots, n\}$ chosen uniformly at random.

   (a) Prove that the following algorithm is ***not*** correct. *[Hint: Consider the case $n = 3$.]*

```
SHUFFLE(n):
  for i ← 1 to n
      π[i] ← i
  for i ← 1 to n
      swap π[i] ↔ π[RANDOM(n)]
  return π[1..n]
```

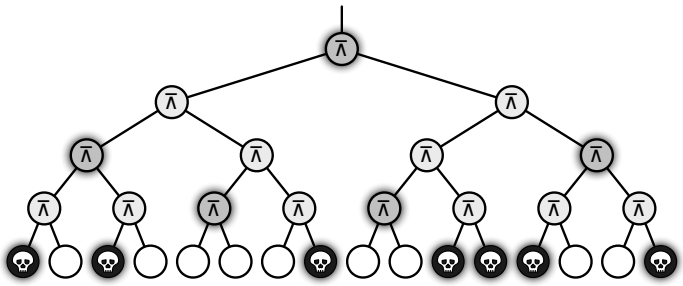
   (b) Consider the following implementation of SHUFFLE.

```
SHUFFLE(n):
  for i ← 1 to n
      π[i] ← NULL
  for i ← 1 to n
      j ← RANDOM(n)
      while (π[j] != NULL)
          j ← RANDOM(n)
      π[j] ← i
  return π[1..n]
```

   Prove that this algorithm is correct. What is its expected running time?

   (c) Describe and analyze an implementation of SHUFFLE that runs in $O(n)$ time. (An algorithm that runs in $O(n)$ *expected* time is fine, but $O(n)$ *worst-case* time is possible.)

2. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



   You can decide whether it's worth playing or not as follows. Imagine that the tree is a Boolean circuit whose inputs are specified at the leaves: white and black represent TRUE and FALSE inputs, respectively. Each internal node in the tree is a NAND gate that gets its input from its children and passes its output to its parent. (Recall that a NAND gate outputs FALSE if and only if both its inputs are TRUE.) If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead. Or maybe Battleship.

(a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

(b) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $O(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*

*(c) **[Extra credit]** Describe and analyze a randomized algorithm that determines whether you can win in $O(c^n)$ expected time, for some constant $c < 3$. *[Hint: You may not need to change your algorithm from part (b) at all!]*

3. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

- MAKEQUEUE: Return a new priority queue containing the empty set.
- FINDMIN(Q): Return the smallest element of Q (if any).
- DELETEMIN(Q): Remove the smallest element in Q (if any).
- INSERT(Q, x): Insert element x into Q, if it is not already there.
- DECREASEKEY(Q, x, y): Replace an element $x \in Q$ with a smaller key y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q containing x.
- DELETE(Q, x): Delete the element $x \in Q$. The input is a pointer directly to the node in Q containing x.
- MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2; this operation destroys Q_1 and Q_2.

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```
MELD(Q1, Q2):
  if Q1 is empty return Q2
  if Q2 is empty return Q1

  if key(Q1) > key(Q2)
      swap Q1 ↔ Q2

  with probability 1/2
      left(Q1) ← MELD(left(Q1), Q2)
  else
      right(Q1) ← MELD(right(Q1), Q2)

  return Q1
```

(a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (not just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. *[Hint: How long is a random root-to-leaf path in an n-node binary tree if each left/right choice is made with equal probability?]*

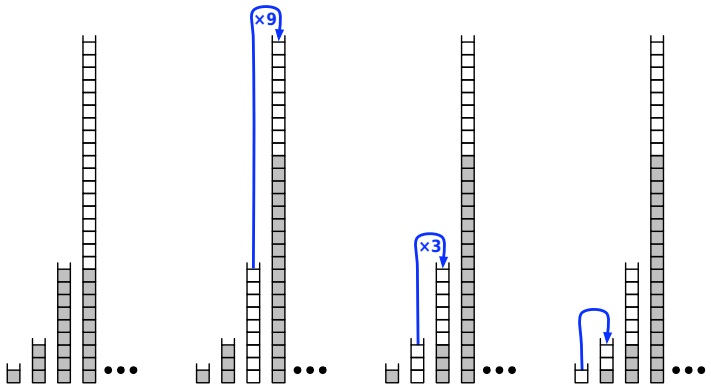
(b) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ expected time.)

1. A *multistack* consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. The user always pushes and pops elements from the smallest stack S_0. However, before any element can be pushed onto any full stack S_i, we first pop all the elements off S_i and push them onto stack S_{i+1} to make room. (Thus, if S_{i+1} is already full, we first recursively move all its members to S_{i+2}.) Similarly, before any element can be popped from any empty stack S_i, we first pop 3^i elements from S_{i+1} and push them onto S_i to make room. (Thus, if S_{i+1} is already empty, we first recursively fill it by popping elements from S_{i+2}.) Moving a single element from one stack to another takes $O(1)$ time.

   Here is pseudocode for the multistack operations MSPush and MSPop. The internal stacks are managed with the subroutines Push and Pop.

```
MPush(x):
  i ← 0
  while S_i is full
      i ← i + 1

  while i > 0
      i ← i − 1
      for j ← 1 to 3^i
          Push(S_{i+1}, Pop(S_i))

  Push(S_0, x)
```

```
MPop(x):
  i ← 0
  while S_i is empty
      i ← i + 1

  while i > 0
      i ← i − 1
      for j ← 1 to 3^i
          Push(S_i, Pop(S_{i+1}))

  return Pop(S_0)
```



Making room in a multistack, just before pushing on a new element.

   (a) In the worst case, how long does it take to push one more element onto a multistack containing n elements?

   (b) Prove that if the user never pops anything from the multistack, the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack during its lifetime.

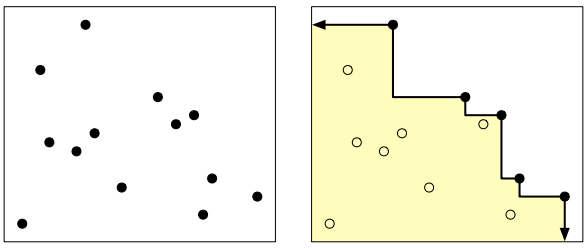
   (c) Prove that in any intermixed sequence of pushes and pops, each push or pop operation takes $O(\log n)$ amortized time, where n is the maximum number of elements in the multistack during its lifetime.

2. Design and analyze a simple data structure that maintains a list of integers and supports the following operations.

   - CREATE( ) creates and returns a new list
   - PUSH(L, x) appends x to the end of L
   - POP(L) deletes the last entry of L and returns it
   - LOOKUP(L, k) returns the kth entry of L

   Your solution may use these primitive data structures: arrays, balanced binary search trees, heaps, queues, single or doubly linked lists, and stacks. If your algorithm uses *anything* fancier, you must give an explicit implementation. Your data structure must support all operations in amortized constant time. In addition, your data structure must support each LOOKUP in *worst-case* $O(1)$ time. At all times, the size of your data structure must be linear in the number of objects it stores.

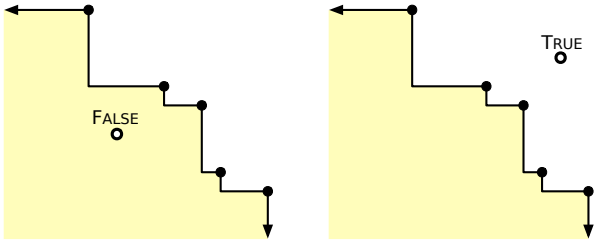
3. Let P be a set of n points in the plane. The *staircase* of P is the set of all points in the plane that have at least one point in P both above and to the right.



A set of points in the plane and its staircase (shaded).

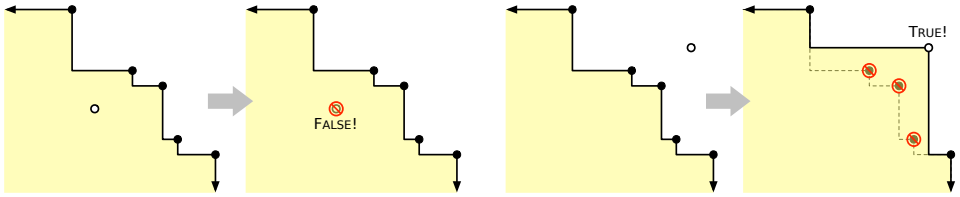
(a) Describe an algorithm to compute the staircase of a set of n points in $O(n \log n)$ time.

(b) Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm ABOVE?(x, y) that returns TRUE if the point (x, y) is above the staircase, or FALSE otherwise. Your data structure should use $O(n)$ space, and your ABOVE? algorithm should run in $O(\log n)$ time.



Two staircase queries.

(c) Describe and analyze a data structure that maintains a staircase as new points are inserted. Specifically, your data structure should support a function INSERT(x, y) that adds the point (x, y) to the underlying point set and returns TRUE or FALSE to indicate whether the staircase of the set has changed. Your data structure should use $O(n)$ space, and your INSERT algorithm should run in $O(\log n)$ amortized time.



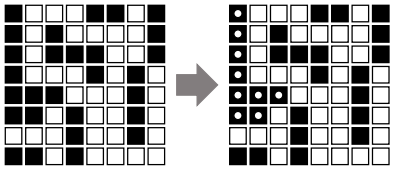
Two staircase insertions.

CS 473: Undergraduate Algorithms, Spring 2010
Homework 6

Written solutions due Tuesday, March 16, 2010 at noon

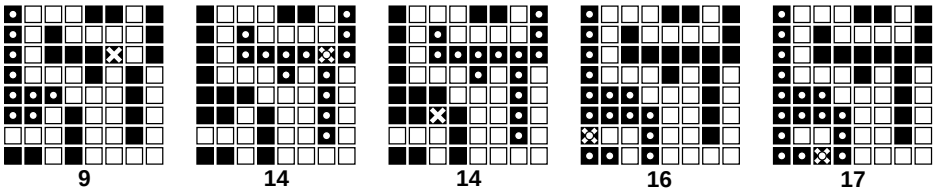
---

1. (a) Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an $n \times n$ bitmap $B[1..n, 1..n]$. For example, given the bitmap below as input, your algorithm should return the number 9, because the largest conected black component (marked with white dots on the right) contains nine pixels.



   (b) Design and analyze an algorithm BLACKEN(i, j) that colors the pixel $B[i, j]$ black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.

   For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the BLACKEN algorithm.



   (c) What is the *worst-case* running time of your BLACKEN algorithm?

2. Suppose you are given a graph G with weighted edges and a minimum spanning tree T of G.

   (a) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is *increased*.

   (b) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is *decreased*.

   In both cases, the input to your algorithm is the edge e and its new weight; your algorithms should modify T so that it is still a minimum spanning tree. *[Hint: Consider the cases $e \in T$ and $e \notin T$ separately.]*

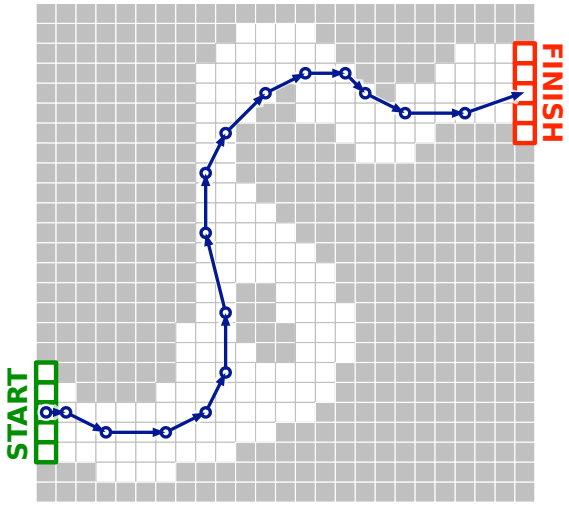
3. ***Racetrack*** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game of uncertain origin that Jeff played on the bus in 5th grade.[1] The game is played using a racetrack drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

   Each car has a *position* and a *velocity*, both with integer x- and y-coordinates. The initial position is an arbitrary point on the starting line, chosen by the player; the initial velocity is always $(0,0)$. At each step, the player optionally increments or decrements either or both coordinates of the car's velocity; in other words, each component of the velocity can change by at most 1 in a single step. The car's new position is then determined by adding the new velocity to the car's previous position. The new position must lie inside the track; otherwise, the car crashes and that player immediately loses the race. The first car that reaches a position ***on*** the finish line is the winner.

   Suppose the racetrack is represented by an $n \times n$ array of bits, where each 0 bit represents a grid point inside the track, each 1 bit represents a grid point outside the track, the 'starting line' is the first column, and the 'finish line' is the last column.

   Describe and analyze an algorithm to find *the minimum number of steps* required to move a car from the starting line to the finish line according to these rules, given a racetrack bitmap as input. *[Hint: Build a graph. What are the vertices? What are the edges? What problem is this?]*

velocity	position
$(0,0)$	$(1,5)$
$(1,0)$	$(2,5)$
$(2,-1)$	$(4,4)$
$(3,0)$	$(7,4)$
$(2,1)$	$(9,5)$
$(1,2)$	$(10,7)$
$(0,3)$	$(10,10)$
$(-1,4)$	$(9,14)$
$(0,3)$	$(9,17)$
$(1,2)$	$(10,19)$
$(2,2)$	$(12,21)$
$(2,1)$	$(14,22)$
$(2,0)$	$(16,22)$
$(1,-1)$	$(17,21)$
$(2,-1)$	$(19,20)$
$(3,0)$	$(22,20)$
$(3,1)$	$(25,21)$



A 16-step Racetrack run, on a 25×25 track. This is *not* the shortest run on this track.

[1] The actual game Jeff played was a bit more complicated than the version described in this problem. In particular, the track was a freeform curve, and by default, the entire line segment traversed by a car in a single step had to lie entirely inside the track. If a car did run off the track, it started its next turn with velocity zero, at the legal grid point closest to where it first crossed the track boundary.

1. On an overnight camping trip in Sunnydale National Park, you are woken from a restless sleep by a scream. As you crawl out of your tent to investigate, a terrified park ranger runs out of the woods, covered in blood and clutching a crumpled piece of paper to his chest. As he reaches your tent, he gasps, "Get out... while... you...", thrusts the paper into your hands, and falls to the ground. Checking his pulse, you discover that the ranger is stone dead.

   You look down at the paper and recognize a map of the park, drawn as an undirected graph, where vertices represent landmarks in the park, and edges represent trails between those landmarks. (Trails start and end at landmarks and do not cross.) You recognize one of the vertices as your current location; several vertices on the boundary of the map are labeled EXIT.

   On closer examination, you notice that someone (perhaps the poor dead park ranger) has written a real number between 0 and 1 next to each vertex and each edge. A scrawled note on the back of the map indicates that a number next to an edge is the probability of encountering a vampire along the corresponding trail, and a number next to a vertex is the probability of encountering a vampire at the corresponding landmark. (Vampires can't stand each other's company, so you'll never see more than one vampire on the same trail or at the same landmark.) The note warns you that stepping off the marked trails will result in a slow and painful death.

   You glance down at the corpse at your feet. Yes, his death certainly looked painful. Wait, was that a twitch? Are his teeth getting longer? After driving a tent stake through the undead ranger's heart, you wisely decide to leave the park immediately.

   Describe and analyze an efficient algorithm to find a path from your current location to an arbitrary EXIT node, such that the total *expected number* of vampires encountered along the path is as small as possible. *Be sure to account for* ***both*** *the vertex probabilities* ***and*** *the edge probabilities!*

2. In this problem we will discover how you, too, can be employed by Wall Street and cause a major economic collapse! The *arbitrage* business is a money-making scheme that takes advantage of differences in currency exchange. In particular, suppose that 1 US dollar buys 120 Japanese yen; 1 yen buys 0.01 euros; and 1 euro buys 1.2 US dollars. Then, a trader starting with $1 can convert his money from dollars to yen, then from yen to euros, and finally from euros back to dollars, ending with $1.44! The cycle of currencies $\$ \to ¥ \to € \to \$$ is called an ***arbitrage cycle***. Of course, finding and exploiting arbitrage cycles before the prices are corrected requires extremely fast algorithms.

   Suppose n different currencies are traded in your currency market. You are given the matrix $R[1..n, 1..n]$ of exchange rates between every pair of currencies; for each i and j, one unit of currency i can be traded for $R[i, j]$ units of currency j. (Do *not* assume that $R[i, j] \cdot R[j, i] = 1$.)

   (a) Describe an algorithm that returns an array $V[1..n]$, where $V[i]$ is the maximum amount of currency i that you can obtain by trading, starting with one unit of currency 1, assuming there are no arbitrage cycles.

   (b) Describe an algorithm to determine whether the given matrix of currency exchange rates creates an arbitrage cycle.

   (c) Modify your algorithm from part (b) to actually return an arbitrage cycle, if it exists.

3. Let $G = (V, E)$ be a directed graph with weighted edges; edge weights could be positive, negative, or zero. In this problem, you will develop an algorithm to compute shortest paths between *every* pair of vertices. The output from this algorithm is a two-dimensional array $dist[1..V, 1..V]$, where $dist[i, j]$ is the length of the shortest path from vertex i to vertex j.

   (a) How could we delete some node v from this graph, without changing the shortest-path distance between any other pair of nodes? Describe an algorithm that constructs a directed graph $G' = (V', E')$ with weighted edges, where $V' = V \setminus \{v\}$, and the shortest-path distance between any two nodes in G' is equal to the shortest-path distance between the same two nodes in G. For full credit, your algorithm should run in $O(V^2)$ time.

   (b) Now suppose we have already computed all shortest-path distances in G'. Describe an algorithm to compute the shortest-path distances from v to every other node, and from every other node to v, in the original graph G. For full credit, your algorithm should run in $O(V^2)$ time.

   (c) Combine parts (a) and (b) into an algorithm that finds the shortest paths between *every* pair of vertices in the graph. For full credit, your algorithm should run in $O(V^3)$ time.

   The lecture notes (along with most algorithms textbooks and Wikipedia) describe a dynamic programming algorithm due to Floyd and Warshall that computes all shortest paths in $O(V^3)$ time. This is *not* that algorithm.

CS 473: Undergraduate Algorithms, Spring 2010
Homework 8

Written solutions due Tuesday, April 20, 2010 in class.

---

1. Suppose you have already computed a maximum (s,t)-flow f in a flow network G with integer capacities. Let k be an arbitrary positive integer, and let e be an arbitrary edge in G whose capacity is at least k.

   (a) Suppose we *increase* the capacity of e by k units. Describe and analyze an algorithm to update the maximum flow.

   (b) Now suppose we *decrease* the capacity of e by k units. Describe and analyze an algorithm to update the maximum flow.

   For full credit, both algorithms should run in $O(Ek)$ time. *[Hint: First consider the case $k = 1$.]*

2. Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers. We want to *round* A to an integer matrix, by replacing each entry x in A with either $\lfloor x \rfloor$ or $\lceil x \rceil$, without changing the sum of entries in any row or column of A. For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longmapsto \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

   Describe an efficient algorithm that either rounds A in this fashion, or reports correctly that no such rounding is possible.

3. A *cycle cover* of a given directed graph $G = (V, E)$ is a set of vertex-disjoint cycles that cover all the vertices. Describe and analyze an efficient algorithm to find a cycle cover for a given graph, or correctly report that none exists. *[Hint: Use ipartite atching!]*

1. We say that an array $A[1..n]$ is *k-sorted* if it can be divided into k blocks, each of size n/k, such that the elements in each block are larger than the elements in earlier blocks, and smaller than elements in later blocks. The elements within each block need not be sorted.

   For example, the following array is 4-sorted:

1	2	4	3	7	6	8	5	10	11	9	12	15	13	16	14

   (a) Describe an algorithm that k-sorts an arbitrary array in time $O(n \log k)$.

   (b) Prove that any comparison-based k-sorting algorithm requires $\Omega(n \log k)$ comparisons in the worst case.

   (c) Describe an algorithm that completely sorts an already k-sorted array in time $O(n \log(n/k))$.

   (d) Prove that any comparison-based algorithm to completely sort a k-sorted array requires $\Omega(n \log(n/k))$ comparisons in the worst case.

   In all cases, you can assume that n/k is an integer and that $n! \approx \left(\frac{n}{e}\right)^n$.

2. Recall the nuts and bolts problem from the first randomized algorithms lecture. You are given n nuts and n bolts of different sizes. Each nut matches exactly one bolt and vice versa. The nuts and bolts are all almost exactly the same size, so we can't tell if one bolt is bigger than the other, or if one nut is bigger than the other. If we try to match a nut with a bolt, however, we will discover either that the nut is too big, the nut is too small, or the nut is just right for the bolt. The goal was to find the matching nut for every bolt.

   Now consider a relaxed version of the problem where the goal is to find the matching nuts for *half* of the bolts, or equivalently, to find $n/2$ matched nut-bolt pairs. (It doesn't matter *which* $n/2$ nuts and bolts are matched.) Prove that any deterministic algorithm to solve this problem must perform $\Omega(n \log n)$ nut-bolt tests in the worst case.

3. UIUC has just finished constructing the new Reingold Building, the tallest dormitory on campus. In order to determine how much insurance to buy, the university administration needs to determine the highest safe floor in the building. A floor is consdered *safe* if ~~a drunk student~~ ***an egg*** can fall from a window on that floor and land without breaking; if the egg breaks, the floor is considered *unsafe*. Any floor that is higher than an unsafe floor is also considered unsafe. The only way to determine whether a floor is safe is to drop an egg from a window on that floor.

   You would like to find the lowest unsafe floor L by performing as few tests as possible; unfortunately, you have only a very limited supply of eggs.

   (a) Prove that if you have only one egg, you can find the lowest unsafe floor with L tests. *[Hint: Yes, this is trivial.]*

   (b) Prove that if you have only one egg, you must perform at least L tests in the worst case. In other words, prove that your algorithm from part (a) is optimal. *[Hint: Use an adversary argument.]*

   (c) Describe an algorithm to find the lowest unsafe floor using *two* eggs and only $O(\sqrt{L})$ tests. *[Hint: Ideally, each egg should be dropped the same number of times. How many floors can you test with n drops?]*

   (d) Prove that if you start with two eggs, you must perform at least $\Omega(\sqrt{L})$ tests in the worst case. In other words, prove that your algorithm from part (c) is optimal.

This homework is practice only. However, there will be at least one NP-hardness problem on the final exam, so working through this homework is *strongly* recommended. Students/groups are welcome to submit solutions for feedback (but not credit) in class on May 4, after which we will publish official solutions.

---

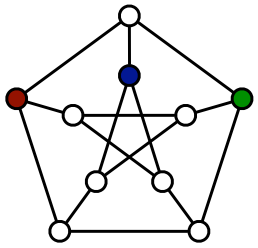
1. Recall that 3SAT asks whether a given boolean formula in conjunctive normal form, with exactly three literals in each clause, is satisfiable. In class we proved that 3SAT is NP-complete, using a reduction from CIRCUITSAT.

   Now consider the related problem **2SAT**: Given a boolean formula in conjunctive normal form, with exactly *two* literals in each clause, is the formula satisfiable? For example, the following boolean formula is a valid input to 2SAT:

   $$(x \vee y) \wedge (y \vee \overline{z}) \wedge (\overline{x} \vee z) \wedge (\overline{w} \vee y).$$

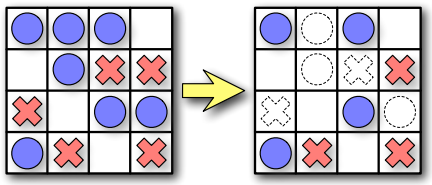
   Either prove that 2SAT is NP-hard or describe a polynomial-time algorithm to solve it. *[Hint: Recall that* $(x \vee y) \equiv (\overline{x} \rightarrow y)$*, and build a graph.]*

2. Let $G = (V, E)$ be a graph. A *dominating set* in G is a subset S of the vertices such that every vertex in G is either in S or adjacent to a vertex in S. The DOMINATINGSET problem asks, given a graph G and an integer k as input, whether G contains a dominating set of size k. Either prove that this problem is NP-hard or describe a polynomial-time algorithm to solve it.

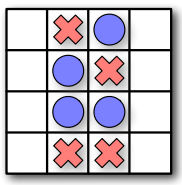


A dominating set of size 3 in the Peterson graph.

3. Consider the following solitaire game. The puzzle consists of an $n \times m$ grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

This exam lasts 120 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet with your answers.

1. Each of these ten questions has one of the following five answers:

   A: $\Theta(1)$ B: $\Theta(\log n)$ C: $\Theta(n)$ D: $\Theta(n \log n)$ E: $\Theta(n^2)$

   Choose the correct answer for each question. Each correct answer is worth $+1$ point; each incorrect answer is worth $-1/2$ point; and each "I don't know" is worth $+1/4$ point. Negative scores will be recorded as 0.

   (a) What is $\dfrac{3}{n} + \dfrac{n}{3}$?

   (b) What is $\displaystyle\sum_{i=1}^{n} \frac{i}{n}$?

   (c) What is $\sqrt{\displaystyle\sum_{i=1}^{n} i}$?

   (d) How many bits are required to write the number $n!$ (the factorial of n) in binary?

   (e) What is the solution to the recurrence $E(n) = E(n-3) + 17n$?

   (f) What is the solution to the recurrence $F(n) = 2F(n/4) + 6n$?

   (g) What is the solution to the recurrence $G(n) = 9G(n/9) + 9n$?

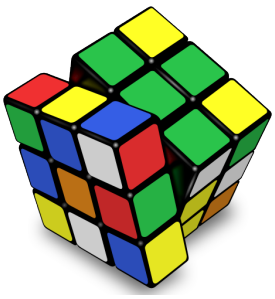
   (h) What is the worst-case running time of quicksort?

   (i) Let $X[1..n, 1..n]$ be a fixed array of numbers. Consider the following recursive function:

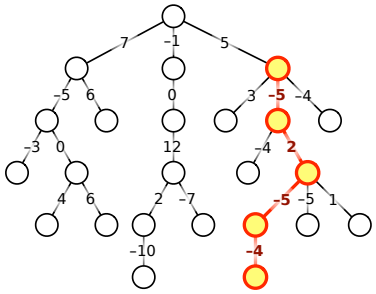
$$WTF(i,j) = \begin{cases} 0 & \text{if } \min\{i,j\} \le 0 \\ -\infty & \text{if } \max\{i,j\} > n \\ X[i,j] + \max\left\{ \begin{array}{l} WTF(i-2, j+1) \\ WTF(i-2, j-1) \\ WTF(i-1, j-2) \\ WTF(i+1, j-2) \end{array} \right\} & \text{otherwise} \end{cases}$$

   How long does it take to compute $WTF(n,n)$ using dynamic programming?

   (j) The Rubik's Cube is a mechanical puzzle invented in 1974 by Ernő Rubik, a Hungarian professor of architecture. The puzzle consists of a $3 \times 3 \times 3$ grid of 'cubelets', whose faces are covered with stickers in six different colors. In the puzzle's solved state, each face of the puzzle is one solid color. A mechanism inside the puzzle allows any face of the cube to be freely turned (as shown on the right). The puzzle can be scrambled by repeated turns. Given a scrambled Rubik's Cube, how long does it take to find the *shortest* sequence of turns that returns the cube to its solved state?



2. Let T be a rooted tree with integer weights on its edges, which could be positive, negative, or zero. The weight of a path in T is the sum of the weights of its edges. Describe and analyze an algorithm to compute the minimum weight of any path from a node in T down to one of its descendants. It is not necessary to compute the actual minimum-weight path; just its weight. For example, given the tree shown below, your algorithm should return the number -12.



The minimum-weight downward path in this tree has weight -12.

3. Describe and analyze efficient algorithms to solve the following problems:

   (a) Given a set of n integers, does it contain two elements a, b such that $a + b = 0$?

   (b) Given a set of n integers, does it contain three elements a, b, c such that $a + b = c$?

4. A *common supersequence* of two strings A and B is another string that includes both the characters of A in order and the characters of B in order. Describe and analyze and algorithm to compute the length of the *shortest* common supersequence of two strings $A[1..m]$ and $B[1..n]$. You do not need to compute an actual supersequence, just its length.

   For example, if the input strings are `ANTHROHOPOBIOLOGICAL` and `PRETERDIPLOMATICALLY`, your algorithm should output 31, because a shortest common supersequence of those two strings is `PREANTHEROHODPOBIOPLOMATGICALLY`.

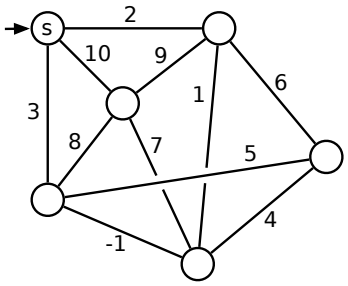
5. *[Taken directly from HBS0.]* Recall that the *Fibonacci numbers* F_n are recursively defined as follows: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for every integer $n \geq 2$. The first few Fibonacci numbers are $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$.

   Prove that any non-negative integer can be written as the sum of distinct *non-consecutive* Fibonacci numbers. That is, if any Fibonacci number F_n appears in the sum, then its neighbors F_{n-1} and F_{n+1} do not. For example:

$$\begin{aligned} 88 &= 55 + 21 + 8 + 3 + 1 &&= F_{10} + F_8 + F_6 + F_4 + F_2 \\ 42 &= 34 + 8 &&= F_9 + F_6 \\ 17 &= 13 + 3 + 1 &&= F_7 + F_4 + F_2 \end{aligned}$$

This exam lasts 120 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet with your answers.

1. Find the following spanning trees for the weighted graph shown below.



   (a) A depth-first spanning tree rooted at s.
   (b) A breadth-first spanning tree rooted at s.
   (c) ~~A shortest-path tree rooted at s.~~ ***Oops!***
   (d) A minimum spanning tree.

   You do *not* need to justify your answers; just clearly indicate the edges of each spanning tree in your answer booklet. Yes, one of the edges has negative weight.

2. *[Taken directly from HBS 6.]* An *Euler tour* of a graph G is a walk that starts and ends at the same vertex and traverses every edge of G exactly once. ***Prove*** that a connected undirected graph G has an Euler tour if and only if every vertex in G has even degree.

3. You saw in class that the standard algorithm to INCREMENT a binary counter runs in $O(1)$ amortized time. Now suppose we also want to support a second function called RESET, which resets all bits in the counter to zero.

   Here are the INCREMENT and RESET algorithms. In addition to the array $B[\ldots]$ of bits, we now also maintain the index of the most significant bit, in an integer variable msb.

```
INCREMENT(B[0..∞], msb):
  i ← 0
  while B[i] = 1
      B[i] ← 0
      i ← i + 1
  B[i] ← 1
  if i > msb
      msb ← i
```

```
RESET(B[0..∞], msb):
  for i ← 0 to msb
      B[i] ← 0
  msb ← 0
```

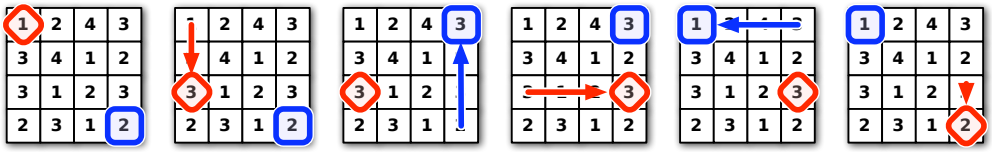
   In parts (a) and (b), let n denote the number currently stored in the counter.

   (a) What is the worst-case running time of INCREMENT, as a function of n?
   (b) What is the worst-case running time of RESET, as a function of n?
   (c) ***Prove*** that in an *arbitrary* intermixed sequence of INCREMENT and RESET operations, the amortized time for each operation is $O(1)$.

4. The following puzzle was invented by the infamous Mongolian puzzle-warrior Vidrach Itky Leda in the year 1473. The puzzle consists of an $n \times n$ grid of squares, where each square is labeled with a positive integer, and two tokens, one red and the other blue. The tokens always lie on distinct squares of the grid. The tokens start in the top left and bottom right corners of the grid; the goal of the puzzle is to swap the tokens.

   In a single turn, you may move either token up, right, down, or left *by a distance determined by the* ***other*** *token*. For example, if the red token is on a square labeled 3, then you may move the blue token 3 steps up, 3 steps left, 3 steps right, or 3 steps down. However, you may not move a token off the grid or to the same square as the other token.

   Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given Vidrach Itky Leda puzzle, or correctly reports that the puzzle has no solution. For example, given the puzzle below, your algorithm would return the number 5.



A five-move solution for a 4×4 Vidrach Itky Leda puzzle.

5. Suppose you are given an array $X[1..n]$ of real numbers chosen independently and uniformly at random from the interval $[0,1]$. An array entry $X[i]$ is called a *local maximum* if it is larger than its neighbors $X[i-1]$ and $X[i+1]$ (if they exist).

   What is the *exact* expected number of local maxima in X? ***Prove*** that your answer is correct. *[Hint: Consider the special case $n = 3$.]*

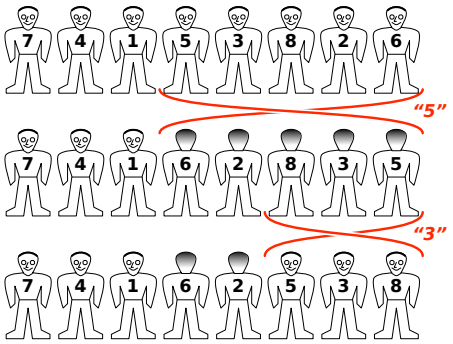
0.7	0.3	**1.0**	0.1	0.0	0.5	**0.6**	0.2	0.4	**0.9**	0.8

A randomly filled array with 4 local maxima.

This exam lasts 180 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet with your answers.

1. Choreographer Michael Flatley has hired a new dance company to perform his latest Irish step-dancing extravaganza. At their first practice session, the new dancers line up in a row on stage and practice a movement called the *Flatley Flip*: Whenever Mr. Flatley calls out any positive integer k, the k rightmost dancers rotate 180 degrees as a group, so that their order in the line is reversed.

   Each dancer wears a shirt with a positive integer printed on the front and back; different dancers have different numbers. Mr. Flatley wants to rearrange the dancers, using only a sequence of Flatley Flips, so that these numbers are sorted from left to right in increasing order.



Two Flatley flips.

   (a) Describe an algorithm to sort an arbitrary row of n numbered dancers, using $O(n)$ Flatley flips. (After sorting, the dancers may face forward, backward, or some of each.) *Exactly* how many flips does your algorithm perform in the worst case?[1]

   (b) Describe an algorithm that sorts an arbitrary row of n numbered dancers *and ensures that all dancers are facing forward*, using $O(n)$ Flatley flips. *Exactly* how many flips does your algorithm perform in the worst case?[2]

2. You're in charge of choreographing a musical for your local community theater, and it's time to figure out the final pose of the big show-stopping number at the end. ("Streetcar!") You've decided that each of the n cast members in the show will be positioned in a big line when the song finishes, all with their arms extended and showing off their best spirit fingers.

   The director has declared that during the final flourish, each cast member must either point both their arms up or point both their arms down; it's your job to figure out who points up and who points down. Moreover, in a fit of unchecked power, the director has also given you a list of arrangements that will upset his delicate artistic temperament. Each forbidden arrangement is a subset of cast members paired with arm positions; for example: "Marge may not point her arms up while Ned and Apu point their arms down."

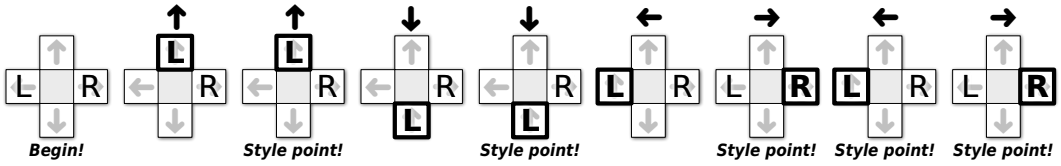
   Prove that finding an acceptable arrangement of arm positions is NP-hard. *[Hint: Describe a reduction from 3SAT.]*

[1] This is really a question about networking.
[2] This is really a question about mutating DNA.

3. ***Dance Dance Revolution*** is a dance video game, first introduced in Japan by Konami in 1998. Players stand on a platform marked with four arrows, pointing forward, back, left, and right, arranged in a cross pattern. During play, the game plays a song and scrolls a sequence of n arrows (←, ↑, ↓, or →) from the bottom to the top of the screen. At the precise moment each arrow reaches the top of the screen, the player must step on the corresponding arrow on the dance platform. (The arrows are timed so that you'll step with the beat of the song.)

   You are playing a variant of this game called "Vogue Vogue Revolution", where the goal is to play perfectly but move as little as possible. When an arrow reaches the top of the screen, if one of your feet is already on the correct arrow, you are awarded one style point for maintaining your current pose. If neither foot is on the right arrow, you must move one (and *only* one) of your feet from its current location to the correct arrow on the platform. If you ever step on the wrong arrow, or fail to step on the correct arrow, or move more than one foot at a time, all your style points are taken away and the games ends.

   How should you move your feet to maximize your total number of style points? For purposes of this problem, assume you always start with you left foot on ← and you right foot on →, and that you've memorized the entire sequence of arrows. For example, if the sequence is ↑↑↓↓←→←→, you can earn 5 style points by moving you feet as shown below:



   (a) ***Prove*** that for *any* sequence of n arrows, it is possible to earn at least $n/4 - 1$ style points.

   (b) Describe an efficient algorithm to find the maximum number of style points you can earn during a given VVR routine.[3] The input to your algorithm is an array $Arrow[1..n]$ containing the sequence of arrows. *[Hint: Build a graph!]*

4. It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the the list of n songs that the judges will play during the contest, in chronological order.

   You know all the songs, all the judges, and your own dancing ability extremely well. For each integer k, you know that if you dance to the kth song on the schedule, you will be awarded exactly $Score[k]$ points, but then you will be physically unable to dance for the next $Wait[k]$ songs (that is, you cannot dance to songs $k+1$ through $k + Wait[k]$). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

   Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays $Score[1..n]$ and $Wait[1..n]$.[4]

[3] This is really a question about paging.
[4] This is really a question about processor scheduling.

5. You're organizing the First Annual UIUC Computer Science 72-Hour Dance Exchange, to be held all day Friday, Saturday, and Sunday. Several 30-minute sets of music will be played during the event, and a large number of DJs have applied to perform. You need to hire DJs according to the following constraints.

   - Exactly k sets of music must be played each day, and thus $3k$ sets altogether.
   - Each set must be played by a single DJ in a consistent music genre (ambient, bubblegum, dubstep, horrorcore, hyphy, trip-hop, Nitzhonot, Kwaito, J-pop, Nashville country, ...).
   - Each genre must be played at most once per day.
   - Each candidate DJ has given you a list of genres they are willing to play.
   - Each DJ can play at most three sets during the entire event.

   Suppose there are n candidate DJs and g different musical genres available. Describe and analyze an efficient algorithm that either assigns a DJ and a genre to each of the $3k$ sets, or correctly reports that no such assignment is possible.

6. You've been put in charge of putting together a team for the "Dancing with the Computer Scientists" international competition. Good teams in this competition must be capable of performing a wide variety of dance styles. You are auditioning a set of n dancing computer scientists, each of whom specializes in a particular style of dance.

   Describe an algorithm to determine in $O(n)$ time if *more than half* of the n dancers specialize exactly in the same dance style. The input to your algorithm is an array of n positive integers, where each integer identifies a style: 1 = ballroom, 2 = latin, 3 = swing, 4 = b-boy, 42 = contact improv, 101 = peanut butter jelly time, and so on. *[Hint: Remember the* SELECT *algorithm!]*

7. The party you are attending is going great, but now it's time to line up for ***The Algorithm March (アルゴリズムこうしん)***! This dance was originally developed by the Japanese comedy duo Itsumo Kokokara (いつもここから) for the children's television show PythagoraSwitch (ピタゴラスイッチ). The Algorithm March is performed by a line of people; each person in line starts a specific sequence of movements one measure later than the person directly in front of them. Thus, the march is the dance equivalent of a musical round or canon, like "Row Row Row Your Boat".

   Proper etiquette dictates that each marcher must know the person directly in front of them in line, lest a minor mistake during lead to horrible embarrassment between strangers. Suppose you are given a complete list of which people at your party know each other. ***Prove*** that it is NP-hard to determine the largest number of party-goers that can participate in the Algorithm March.[5]

[5] This is really a question about ninjas.

You may assume the following problems are NP-hard:

CircuitSat: Given a boolean circuit, are there any input values that make the circuit output True?

PlanarCircuitSat: Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

3Sat: Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

MaxIndependentSet: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MaxClique: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MinVertexCover: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MinDominatingSet: Given an undirected graph G, what is the size of the smallest subset S of vertices such that every vertex in G is either in S or adjacent to a vertex in S?

MinSetCover: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MinHittingSet: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3Color: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

ChromaticNumber: Given an undirected graph G, what is the minimum number of colors needed to color its vertices, so that every edge touches vertices with two different colors?

MaxCut: Given a graph G, what is the size (number of edges) of the largest bipartite subgraph of G?

HamiltonianCycle: Given a graph G, is there a cycle in G that visits every vertex exactly once?

HamiltonianPath: Given a graph G, is there a path in G that visits every vertex exactly once?

TravelingSalesman: Given a graph G with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

SubsetSum: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

Partition: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3Partition: Given a set X of n positive integers, can X be partitioned into $n/3$ three-element subsets, all with the same sum?

Minesweeper: Given a Minesweeper configuration and a particular square x, is it safe to click on x?

Tetris: Given a sequence of N Tetris pieces and a partially filled $n \times k$ board, is it possible to play every piece in the sequence without overflowing the board?

Sudoku: Given an $n \times n$ Sudoku puzzle, does it have a solution?

KenKen: Given an $n \times n$ Ken-Ken puzzle, does it have a solution?

CS 573: Graduate Algorithms, Fall 2010
Homework 0

Due Wednesday, September 1, 2010 **in class**

- This homework tests your familiarity with prerequisite material (http://www.cs.uiuc.edu/class/fa10/cs573/stuff-you-already-know.html) to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** Fr most topics, the early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
- Each student must submit individual solutions for these homework problems. For all future homeworks, groups of up to three students may submit (or present) a single group solution for each problem.
- Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. In particular:
  - Submit five separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page. Please do *not* staple everything together.
  - You may use any source at your disposal—paper, electronic, or human—but you ***must*** write your solutions in your own words, and you ***must*** cite every source that you use. In particular, each solution should include a list of *everyone* you worked with to solve that problem.
  - Unless explicitly stated otherwise, ***every*** homework problem requires a proof.
  - Answering "I don't know" to any homework or exam problem (except for extra credit problems) is worth 25% partial credit.
  - Algorithms or proofs containing phrases like "and so on" or "repeat this process for all n" instead of an explicit loop, recursion, or induction, will receive 0 points.

---

1. (•) ***Write the sentence "I understand the course policies."***
   Solutions that omit this sentence will not be graded.

   (a) **[5 pts]** Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Assume reasonable but nontrivial base cases if none are given. ***Do not submit proofs***—just a list of five functions—but you should do them anyway, just for practice.

   - $A(n) = 4A(n-1) + 1$
   - $B(n) = B(n-3) + n^2$
   - $C(n) = 2\,C(n/2) + 3\,C(n/3) + n^2$
   - $D(n) = 2\,D(n/3) + \sqrt{n}$
   - $E(n) = \begin{cases} n & \text{if } n \le 3, \\ \dfrac{E(n-1)E(n-2)}{E(n-3)} & \text{otherwise} \end{cases}$ *[Hint: This is easier than it looks!]*

   (b) **[5 pts]** Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. ***Do not submit proofs***—just a sorted list of 16 functions—but you should do them anyway, just for practice.

   Write $f(n) \ll g(n)$ to indicate that $f(n) = o(g(n))$, and write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. We use the notation $\lg n = \log_2 n$.

n	$\lg n$	$\sqrt{n}$	7^n
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$7^{\sqrt{n}}$	$\sqrt{7^n}$
$7^{\lg n}$	$\lg(7^n)$	$7^{\lg \sqrt{n}}$	$7\sqrt{\lg n}$
$\sqrt{7^{\lg n}}$	$\lg(7^{\sqrt{n}})$	$\lg \sqrt{7^n}$	$\sqrt{\lg(7^n)}$

2. Professore Giorgio della Giungla has a 23-node binary tree, in which every node is labeled with a unique letter of the Roman alphabet, which is just like the modern English alphabet, but without the letters **J**, **U**, and **W**. Inorder and postorder traversals of the tree visit the nodes in the following order:

   - Inorder: **S V Z A T P R D B X O L F E H I Q M N G Y K C**
   - Postorder: **A Z P T X B D L E F O H R I V N M K C Y G Q S**

   (a) List the nodes in Prof. della Giungla's tree in the order visited by a *preorder* traversal.

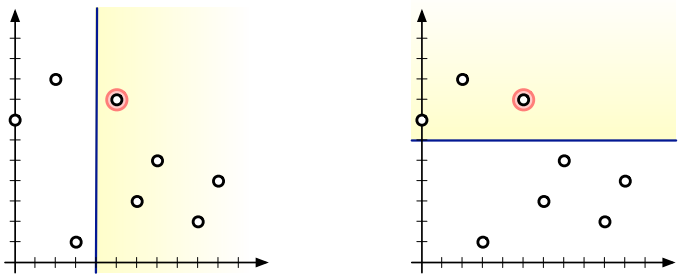
   (b) Draw Prof. della Giungla's tree.

3. The original version of this problem asked to support the mirror-image operations LOWESTTORIGHT and LEFTMOSTABOVE, which are *much* harder to support with a single data structure that stores each point at most once. We will accept $O(n)$-space data structures for either version of the problem for full credit.

   Describe a data structure that stores a set S of n points in the plane, each represented by a pair (x, y) of coordinates, and supports the following queries.

   - **HIGHESTTORIGHT(ℓ):** Return the highest point in S whose x-coordinate is greater than or equal to ℓ. If every point in S has x-coordinate less than ℓ, return NONE.
   - **RIGHTMOSTABOVE(ℓ):** Return the rightmost point in S whose y-coordinate is greater than or equal to ℓ. If every point in S has y-coordinate less than ℓ, return NONE.

   For example, if $S = \{(3,1), (1,9), (9,2), (6,3), (5,8), (7,5), (10,4), (0,7)\}$, then both HIGHESTTORIGHT(4) and RIGHTMOSTABOVE(6) should return the point $(5,8)$, and HIGHESTTORIGHT(15) should return NONE.



HIGHESTTORIGHT(4) = RIGHTMOSTABOVE(6) = (5,8)

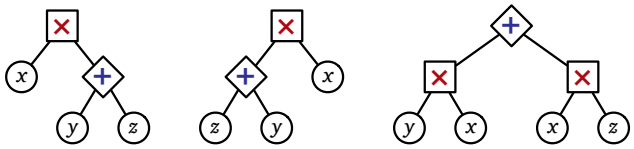
   Analyze both the size of your data structure and the running times of your query algorithms. For full credit, your data structure should use $O(n)$ space, and each query algorithm should run in $O(\log n)$ time. ***For 5 extra credit points, describe a data structure that stores each point at most once.*** You may assume that no two points in S have equal x-coordinates or equal y-coordinates.

   [Hint: Modify one of the standard data structures listed at http://www.cs.uiuc.edu/class/fa10/cs573/stuff-you-already-know.html, but just describe your changes; don't regurgitate the details of the standard data structure.]

4. An ***arithmetic expression tree*** is a binary tree where every leaf is labeled with a variable, every internal node is labeled with an arithmetic operation, and every internal node has exactly two children. For this problem, assume that the only allowed operations are $+$ and $\times$. Different leaves may or may not represent different variables.

   Every arithmetic expression tree represents a function, transforming input values for the leaf variables into an output value for the root, by following two simple rules: (1) The value of any $+$-node is the sum of the values of its children. (2) The value of any $\times$-node is the product of the values of its children.

   Two arithmetic expression trees are ***equivalent*** if they represent the same function; that is, the same input values for the leaf variables always leads to the same output value at both roots.



Three equivalent expression trees. Only the third tree is in normal form.

An arithmetic expression tree is in ***normal form*** if the parent of every +-node (if any) is another +-node.

Prove that for any arithmetic expression tree, there is an equivalent arithmetic expression tree in normal form. *[Hint: Be careful. This is trickier than it looks.]*

5. Recall that a standard (Anglo-American) deck of 52 playing cards contains 13 cards in each of four suits: spades (♠), hearts (♥), diamonds (♦), and clubs (♣). Within each suit, the 13 cards have distinct *ranks*: 2, 3, 4, 5, 6, 7, 8, 9, 10, jack (J), queen (Q), king (K), and ace (A). The ranks are ordered $2 < 3 < \cdots < 9 < 10 < J < Q < K < A$; thus, for example, the jack of spades has higher rank thank the eight of diamonds.

   Professor Jay is about to perform a public demonstration with two decks of cards, one with red backs ('the red deck') and one with blue backs ('the blue deck'). Both decks lie face-down on a table in front of Professor Jay, shuffled uniformly and independently. Thus, in each deck, every permutation of the 52 cards is equally likely.

   To begin the demonstration, Professor Jay turns over the top card from each deck. Then, while he has not yet turned over a three of clubs (3♣), the good Professor hurls the two cards he just turned over into the thick, pachydermatous outer melon layer of a nearby watermelon (that most prodigious of household fruits) and then turns over the next card from the top of each deck. Thus, if 3♣ is the last card in both decks, the demonstration ends with 102 cards embedded in the watermelon.

   (a) What is the *exact* expected number of cards that Professor Jay hurls into the watermelon?

   (b) For each of the statements below, give the *exact* probability that the statement is true of the ***first*** pair of cards Professor Jay turns over.

      i. Both cards are threes.
      ii. One card is a three, and the other card is a club.
      iii. If (at least) one card is a heart, then (at least) one card is a diamond.
      iv. The card from the red deck has higher rank than the card from the blue deck.

   (c) For each of the statements below, give the *exact* probability that the statement is true of the ***last*** pair of cards Professor Jay turns over.

      i. Both cards are threes.
      ii. One card is a three, and the other card is a club.
      iii. If (at least) one card is a heart, then (at least) one card is a diamond.
      iv. The card from the red deck has higher rank than the card from the blue deck.

   Express each of your answers as rational numbers in simplest form, like 123/4567. ***Do not submit proofs***—just a list of rational numbers—but you should do them anyway, just for practice.

CS 573: Graduate Algorithms, Fall 2010
Homework 1

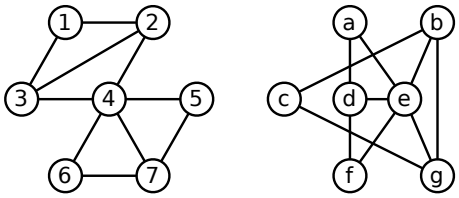
~~Due Friday, September 10, 2010 at 1pm~~

Due Monday, September 13, 2010 at 5pm
(in the homework drop boxes in the basement of Siebel)

For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name and NetID on each page of your submission.

---

1. Two graphs are said to be ***isomorphic*** if one can be transformed into the other just by relabeling the vertices. For example, the graphs shown below are isomorphic; the left graph can be transformed into the right graph by the relabeling $(1, 2, 3, 4, 5, 6, 7) \mapsto (c, g, b, e, a, f, d)$.



Two isomorphic graphs.

Consider the following related decision problems:

- GRAPHISOMORPHISM: Given two graphs G and H, determine whether G and H are isomorphic.
- EVENGRAPHISOMORPHISM: Given two graphs G and H, such that every vertex in G and H has even degree, determine whether G and H are isomorphic.
- SUBGRAPHISOMORPHISM: Given two graphs G and H, determine whether G is isomorphic to a subgraph of H.

(a) Describe a polynomial-time reduction from EVENGRAPHISOMORPHISM to GRAPHISOMORPHISM.

(b) Describe a polynomial-time reduction from GRAPHISOMORPHISM to EVENGRAPHISOMORPHISM.

(c) Describe a polynomial-time reduction from GRAPHISOMORPHISM to SUBGRAPHISOMORPHISM.

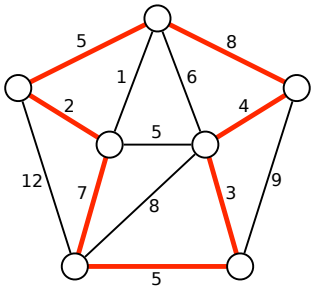
(d) Prove that SUBGRAPHISOMORPHISM is NP-complete.

(e) What can you conclude about the NP-hardness of GRAPHISOMORPHISM? Justify your answer.

[Hint: These are all easy!]

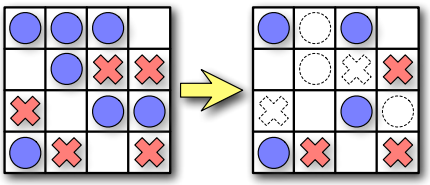
2. Suppose you are given a magic black box that can solve the 3COLORABLE problem ***in polynomial time***. That is, given an arbitrary graph G as input, the magic black box returns TRUE if G has a proper 3-coloring, and returns FALSE otherwise. Describe and analyze a ***polynomial-time*** algorithm that computes an actual proper 3-coloring of a given graph G, or correctly reports that no such coloring exists, using this magic black box as a subroutine. *[Hint: The input to the black box is a graph. Just a graph. Nothing else.]*

3. Let G be an undirected graph with weighted edges. A *heavy Hamiltonian cycle* is a cycle C that passes through each vertex of G exactly once, such that the total weight of the edges in C is at least half of the total weight of all edges in G. Prove that deciding whether a graph has a heavy Hamiltonian cycle is NP-complete.

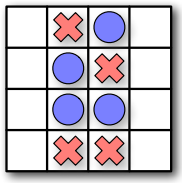


A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

4. Consider the following solitaire game. The puzzle consists of an $n \times m$ grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

5. A boolean formula in ***exclusive-or conjunctive normal form*** (XCNF) is a conjunction (AND) of several *clauses*, each of which is the *exclusive*-or of one or more literals. For example:

$$(u \oplus v \oplus \bar{w} \oplus x) \wedge (\bar{u} \oplus \bar{w} \oplus y) \wedge (\bar{v} \oplus y) \wedge (\bar{u} \oplus \bar{v} \oplus x \oplus y) \wedge (w \oplus x) \wedge y$$

The XCNF-SAT problem asks whether a given XCNF boolean formula is satisfiable. Either describe a polynomial-time algorithm for XCNF-SAT or prove that it is NP-complete.

CS 573: Graduate Algorithms, Fall 2010
Homework 2

Due Monday, September 27, 2010 at 5pm
(in the homework drop boxes in the basement of Siebel)

- For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name and NetID of every group member on the first page of your submission.
- We will use the following rubric to grade all dynamic programming algorithms:
  - 60% for a correct recurrence (including base cases and a plain-English specification); no credit for anything else if this is wrong.
  - 10% for describing a suitable memoization data structure.
  - 20% for describing a correct evaluation order. (A clear picture is sufficient.)
  - 10% point for analyzing the running time of the resulting algorithm.

  Official solutions will always include pseudocode for the final dynamic programming algorithm, but this is ***not*** required for full credit. However, if you do provide correct pseudocode for the dynamic programming algorithm, it is not necessary to separately describe the recurrence, the memoization data structure, or the evaluation order.

  It is ***not*** necessary to state a space bound. There is no penalty for using more space than the official solution, but +1 extra credit for using less space with the same (or better) running time.
- The official solution for every problem will provide a target time bound. Algorithms faster than the official solution are worth more points (as extra credit); algorithms slower than the official solution are worth fewer points. For slower algorithms, partial credit is scaled to the lower maximum score. For example, if a full dynamic programming algorithm would be worth 5 points, just the recurrence is worth 3 points. However, incorrect algorithms are worth zero points, no matter how fast they are.
- Greedy algorithms *must* be accompanied by proofs of correctness in order to receive *any* credit. Otherwise, **any correct algorithm, no matter how slow, is worth at least 2½ points**, assuming it is properly analyzed.

---

1. Suppose you are given an array $A[1..n]$ of positive integers. Describe and analyze an algorithm to find the *smallest* positive integer that is *not* an element of A in $O(n)$ time.

2. Suppose you are given an $m \times n$ bitmap, represented by an array $M[1..m, 1..n]$ whose entries are all 0 or 1. A *solid block* is a subarray of the form $M[i..i', j..j']$ in which every bit is equal to 1. Describe and analyze an efficient algorithm to find a solid block in M with maximum area.

3. Let T be a tree in which each edge e has a weight $w(e)$. A *matching* M in T is a subset of the edges such that each vertex of T is incident to at most one edge in M. The weight of a matching M is the sum of the weights of its edges. Describe and analyze an algorithm to compute a maximum weight matching, given the tree T as input.

4. For any string x and any non-negative integer k, let x^k denote the string obtained by concatenating k copies of x. For example, $\texttt{STRING}^3 = \texttt{STRINGSTRINGSTRING}$ and $\texttt{STRING}^0$ is the empty string.

   A ***repetition*** of x is a prefix of x^k for some integer k. For example, STRINGSTRINGSTRINGST and STR are both repetitions of STRING, as is the empty string.

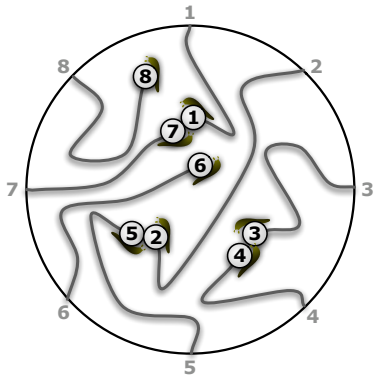
   An ***interleaving*** of two strings x and y is any string obtained by shuffling a repetition of x with a repetition of y. For example, STRWORINDGSTWORIRNGDWSTORR is an interleaving of STRING and WORD, as is the empty string.

   Describe and analyze an algorithm that accepts three strings x, y, and z as input, and decides whether z is an interleaving of x and y.

5. Every year, as part of its annual meeting, the Antarctican Snail Lovers of Upper Glacierville hold a Round Table Mating Race. Several high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to n. During the race, each snail wanders around the table, leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail, even their own. If two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if the race goes on forever.

   For every pair of snails, the Antarctican SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array $M[1..n, 1..n]$ posted on the wall behind the Round Table, where $M[i, j] = M[j, i]$ is the reward to be paid if snails i and j meet.

   Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the array M as input.



The end of a typical Antarctican SLUG race. Snails 6 and 8 never find mates.
The organizers must pay $M[3,4] + M[2,5] + M[1,7]$.

CS 573: Graduate Algorithms, Fall 2010
Homework 3

Due Monday, October 18, 2010 at 5pm
(in the homework drop boxes in the basement of Siebel)

---

1. Suppose we are given two arrays $C[1..n]$ and $R[1..n]$ of positive integers. An $n \times n$ matrix of 0s and 1s *agrees with R and C* if, for every index i, the ith row contains $R[i]$ 1s, and the ith column contains $C[i]$ 1s. Describe and analyze an algorithm that either constructs a matrix that agrees with R and C, or correctly reports that no such matrix exists.

2. Suppose we have n skiers with heights given in an array $P[1..n]$, and n skis with heights given in an array $S[1..n]$. Describe an efficient algorithm to assign a ski to each skier, so that the average difference between the height of a skier and her assigned ski is as small as possible. The algorithm should compute a permutation σ such that the expression

$$\frac{1}{n}\sum_{i=1}^{n}\left|P[i] - S[\sigma(i)]\right|$$

   is as small as possible.

3. Alice wants to throw a party and she is trying to decide who to invite. She has n people to choose from, and she knows which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints:

   - For each guest, there should be at least five other guests that they already know.
   - For each guest, there should be at least five other guests that they *don't* already know.

   Describe and analyze an algorithm that computes the largest possible number of guests Alice can invite, given a list of n people and the list of pairs who know each other.

4. Consider the following heuristic for constructing a vertex cover of a connected graph G: return the set of non-leaf nodes in any depth-first spanning tree of G.

   (a) Prove that this heuristic returns a vertex cover of G.

   (b) Prove that this heuristic returns a 2-approximation to the minimum vertex cover of G.

   (c) Describe an infinite family of graphs for which this heuristic returns a vertex cover of size $2 \cdot OPT$.

5. Suppose we want to route a set of N calls on a telecommunications network that consist of a cycle on n nodes, indexed in order from 0 to $n-1$. Each call has a source node and a destination node, and can be routed either clockwise or counterclockwise around the cycle. Our goal is to route the calls so as to minimize the overall load on the network. The load L_i on any edge $(i, (i+1) \bmod n)$ is the number of calls routed through that edge, and the overall load is $\max_i L_i$. Describe and analyze an efficient 2-approximation algorithm for this problem.

CS 573: Graduate Algorithms, Fall 2010
Homework 4

Due Monday, November 1, 2010 at 5pm
(in the homework drop boxes in the basement of Siebel)

---

1. Consider an n-node treap T. As in the lecture notes, we identify nodes in T by the ranks of their search keys. Thus, 'node 5' means the node with the 5th smallest search key. Let i, j, k be integers such that $1 \le i \le j \le k \le n$.

   (a) What is the *exact* probability that node j is a common ancestor of node i and node k?

   (b) What is the *exact* expected length of the unique path from node i to node k in T?

2. Let $M[1..n, 1..n]$ be an $n \times n$ matrix in which every row and every column is sorted. Such an array is called *totally monotone*. No two elements of M are equal.

   (a) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, compute the number of elements of M smaller than $M[i, j]$ and larger than $M[i', j']$.

   (b) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, return an element of M chosen uniformly at random from the elements smaller than $M[i, j]$ and larger than $M[i', j']$. Assume the requested range is always non-empty.

   (c) Describe and analyze a randomized algorithm to compute the median element of M in $O(n \log n)$ expected time.

3. Suppose we are given a complete undirected graph G, in which each edge is assigned a weight chosen independently and uniformly at random from the real interval $[0, 1]$. Consider the following greedy algorithm to construct a Hamiltonian cycle in G. We start at an arbitrary vertex. While there is at least one unvisited vertex, we traverse the minimum-weight edge from the current vertex to an unvisited neighbor. After $n - 1$ iterations, we have traversed a Hamiltonian path; to complete the Hamiltonian cycle, we traverse the edge from the last vertex back to the first vertex. What is the expected weight of the resulting Hamiltonian cycle? *[Hint: What is the expected weight of the first edge? Consider the case $n = 3$.]*

4. (a) Consider the following deterministic algorithm to construct a vertex cover C of a graph G.

```
VertexCover(G):
  C ← ∅
  while C is not a vertex cover
      pick an arbitrary edge uv that is not covered by C
      add either u or v to C
  return C
```

Prove that VertexCover can return a vertex cover that is $\Omega(n)$ times larger than the smallest vertex cover. You need to describe both an input graph with n vertices, for any integer n, and the sequence of edges and endpoints chosen by the algorithm.

(b) Now consider the following randomized variant of the previous algorithm.

```
RandomVertexCover(G):
  C ← ∅
  while C is not a vertex cover
      pick an arbitrary edge uv that is not covered by C
      with probability 1/2
          add u to C
      else
          add v to C
  return C
```

Prove that the expected size of the vertex cover returned by RandomVertexCover is at most $2 \cdot \text{OPT}$, where OPT is the size of the smallest vertex cover.

(c) Let G be a graph in which each vertex v has a weight $w(v)$. Now consider the following randomized algorithm that constructs a vertex cover.

```
RandomWeightedVertexCover(G):
  C ← ∅
  while C is not a vertex cover
      pick an arbitrary edge uv that is not covered by C
      with probability w(v)/(w(u) + w(v))
          add u to C
      else
          add v to C
  return C
```

Prove that the expected weight of the vertex cover returned by RandomWeightedVertexCover is at most $2 \cdot \text{OPT}$, where OPT is the weight of the minimum-weight vertex cover. A correct answer to this part automatically earns full credit for part (b).

5. (a) Suppose n balls are thrown uniformly and independently at random into m bins. For any integer k, what is the *exact* expected number of bins that contain exactly k balls?

   (b) Consider the following balls and bins experiment, where we repeatedly throw a fixed number of balls randomly into a shrinking set of bins. The experiment starts with n balls and n bins. In each round i, we throw n balls into the remaining bins, and then discard any non-empty bins; thus, only bins that are empty at the end of round i survive to round $i+1$.

```
BallsDestroyBins(n):
  start with n empty bins
  while any bins remain
      throw n balls randomly into the remaining bins
      discard all bins that contain at least one ball
```

   Suppose that in every round, *precisely* the expected number of bins are empty. Prove that under these conditions, the experiment ends after $O(\log^* n)$ rounds.[1]

   *(c) **[Extra credit]** Now assume that the balls are really thrown randomly into the bins in each round. Prove that with high probability, BallsDestroyBins(n) ends after $O(\log^* n)$ rounds.

   (d) Now consider a variant of the previous experiment in which we discard balls instead of bins. Again, the experiment n balls and n bins. In each round i, we throw the remaining balls into n bins, and then discard any ball that lies in a bin by itself; thus, only balls that collide in round i survive to round $i+1$.

```
BinsDestroySingleBalls(n):
  start with n balls
  while any balls remain
      throw the remaining balls randomly into n bins
      discard every ball that lies in a bin by itself
      retrieve the remaining balls from the bins
```

   Suppose that in every round, *precisely* the expected number of bins contain exactly one ball. Prove that under these conditions, the experiment ends after $O(\log\log n)$ rounds.

   *(e) **[Extra credit]** Now assume that the balls are really thrown randomly into the bins in each round. Prove that with high probability, BinsDestroySingleBalls(n) ends after $O(\log\log n)$ rounds.

[1] Recall that the iterated logarithm is defined as follows: $\log^* n = 0$ if $n \le 1$, and $\log^* n = 1 + \log^*(\lg n)$ otherwise.

CS 573: Graduate Algorithms, Fall 2010
Homework 5

Due Friday, November 19, 2010 at 5pm
(in the homework drop boxes in the basement of Siebel)

---

1. Suppose we are given a set of boxes, each specified by their height, width, and depth in centimeters. All three side lengths of every box lie strictly between 10cm and 20cm. As you should expect, one box can be placed inside another if the smaller box can be rotated so that its height, width, and depth are respectively smaller than the height, width, and depth of the larger box. Boxes can be nested recursively. Call a box is *visible* if it is not inside another box.

   Describe and analyze an algorithm to nest the boxes so that the number of visible boxes is as small as possible.

2. Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers. We want to *round* A to an integer matrix, by replacing each entry x in A with either $\lfloor x \rfloor$ or $\lceil x \rceil$, without changing the sum of entries in any row or column of A. For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longmapsto \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

   Describe an efficient algorithm that either rounds A in this fashion, or reports correctly that no such rounding is possible.

3. The Autocratic Party is gearing up their fund-raising campaign for the 2012 election. Party leaders have already chosen their slate of candidates for president and vice-president, as well as various governors, senators, representatives, city council members, school board members, and dog-catchers. For each candidate, the party leaders have determined how much money they must spend on that candidate's campaign to guarantee their election.

   The party is soliciting donations from each of its members. Each voter has declared the total amount of money they are willing to give each candidate between now and the election. (Each voter pledges different amounts to different candidates. For example, everyone is happy to donate to the presidential candidate,[1] but most voters in New York will not donate anything to the candidate for Trash Commissioner of Los Angeles.) Federal election law limits each person's total political contributions to $100 per day.

   Describe and analyze an algorithm to compute a donation schedule, describing how much money each voter should send to each candidate on each day, that guarantees that every candidate gets enough money to win their election. (Party members will of course follow their given schedule perfectly.[2]) The schedule must obey both Federal laws and individual voters' budget constraints. If no such schedule exists, your algorithm should report that fact.

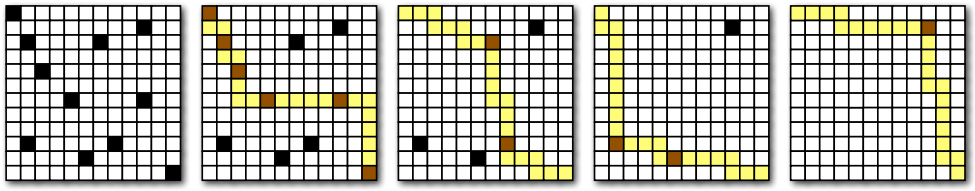
[1] or some nice men in suits will be visiting their home.
[2] It's a nice house you've got here. Shame if anything happened to it.

4. Consider an $n \times n$ grid, some of whose cells are marked. A *monotone* path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell. We want to compute the minimum number of monotone paths that cover all the marked cells.

   (a) One of your friends suggests the following greedy strategy:

   - Find (somehow) one "good" path π that covers the maximum number of marked cells.
   - Unmark the cells covered by π.
   - If any cells are still marked, recursively cover them.

   Prove that this greedy strategy does *not* always compute an optimal solution.



Greedily covering the marked cells in a grid with four monotone paths.

   (b) Describe and analyze an efficient algorithm to compute the smallest set of monotone paths that covers every marked cell. The input to your algorithm is an array $M[1..n, 1..n]$ of booleans, where $M[i, j] = \text{TRUE}$ if and only if cell (i, j) is marked.

5. Let G be a directed graph with two distinguished vertices s and t, and let r be a positive integer. Two players named Paul and Sally play the following game. Paul chooses a path P from s to t, and Sally chooses a subset S of ***at most*** r edges in G. The players reveal their chosen subgraphs simultaneously. If $P \cap S = \emptyset$, Paul wins; if $P \cap S \neq \emptyset$, then Sally wins. Both players want to maximize their chances of winning the game.

   (a) Prove that if Paul uses a deterministic strategy, and Sally knows his strategy, then Sally can guarantee that she wins.[3]

   (b) Let M be the number of edges in a minimum (s, t)-cut. Describe a deterministic strategy for Sally that guarantees that she wins when $\boldsymbol{r \geq M}$, no matter what strategy Paul uses.

   (c) Prove that if Sally uses a deterministic strategy, and Paul knows her strategy then Paul can guarantee that he wins when $r < M$.

   (d) Describe a randomized strategy for Sally that guarantees that she wins with probability at least $\min\{r/M, 1\}$, no matter what strategy Paul uses.

   (e) Describe a randomized strategy for Paul that guarantees that he loses with probability at most $\min\{r/M, 1\}$, no matter what strategy Sally uses.

   Paul and Sally's strategies are, of course, algorithms. (For example, Paul's strategy is an algorithm that takes the graph G and the integer r as input and produces a path P as output.) You do *not* need to analyze the running times of these algorithms, but you must prove all claims about their winning probabilities. Most of these questions are easy.

[3] "Good old rock. Nothing beats rock. . . . D'oh!"

CS 573: Graduate Algorithms, Fall 2010
Homework 5

Practice only — Do not submit solutions

---

1. (a) Describe how to transform any linear program written in general form into an equivalent linear program written in ***slack*** form.

$$\boxed{\begin{array}{lll} \text{maximize} & \displaystyle\sum_{j=1}^{d} c_j x_j & \\ \text{subject to} & \displaystyle\sum_{j=1}^{d} a_{ij} x_j \le b_i & \text{for each } i = 1..p \\ & \displaystyle\sum_{j=1}^{d} a_{ij} x_j = b_i & \text{for each } i = p+1..p+q \\ & \displaystyle\sum_{j=1}^{d} a_{ij} x_j \ge b_i & \text{for each } i = p+q+1..n \end{array}} \quad \Longmapsto \quad \boxed{\begin{array}{r} \max \quad c \cdot x \\ \text{s.t. } Ax = b \\ x \ge 0 \end{array}}$$

   (b) Describe precisely how to dualize a linear program written in slack form.

   (c) Describe precisely how to dualize a linear program written in general form.

   In all cases, keep the number of variables in the resulting linear program as small as possible.

2. Suppose you have a subroutine that can solve linear programs in polynomial time, but only if they are both feasible and bounded. Describe an algorithm that solves *arbitrary* linear programs in polynomial time. Your algorithm should return an optimal solution if one exists; if no optimum exists, your algorithm should report that the input instance is UNBOUNDED or INFEASIBLE, whichever is appropriate. *[Hint: Add one variable and one constraint.]*

3. An *integer program* is a linear program with the additional constraint that the variables must take only integer values.

   (a) Prove that deciding whether an integer program has a feasible solution is NP-complete.

   (b) Prove that finding the optimal feasible solution to an integer program is NP-hard.

   [Hint: Almost any NP-hard decision problem can be formulated as an integer program. Pick your favorite.]

4. Give a linear-programming formulation of the *minimum-cost feasible circulation problem*. You are given a flow network whose edges have both capacities and costs, and your goal is to find a feasible circulation (flow with value 0) whose cost is as small as possible.

5. Given points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ in the plane, the *linear regression problem* asks for real numbers a and b such that the line $y = ax + b$ fits the points as closely as possible, according to some criterion. The most common fit criterion is minimizing the L_2 *error*, defined as follows:[1]

$$\varepsilon_2(a, b) = \sum_{i=1}^{n} (y_i - ax_i - b)^2.$$

But there are several other fit criteria, some of which can be optimized via linear programming.

(a) The L_1 *error* (or *total absolute deviation*) of the line $y = ax + b$ is defined as follows:

$$\varepsilon_1(a, b) = \sum_{i=1}^{n} \left| y_i - ax_i - b \right|.$$

Describe a linear program whose solution (a, b) describes the line with minimum L_1 error.

(b) The L_∞ *error* (or *maximum absolute deviation*) of the line $y = ax + b$ is defined as follows:

$$\varepsilon_\infty(a, b) = \max_{i=1}^{n} \left| y_i - ax_i - b \right|.$$

Describe a linear program whose solution (a, b) describes the line with minimum L_∞ error.

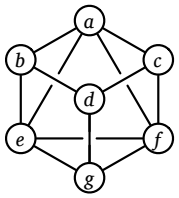
[1] This measure is also known as *sum of squared residuals*, and the algorithm to compute the best fit is normally called *(ordinary/linear) least squares fitting*.

This exam lasts 90 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet with your answers.

1. (a) Suppose $A[1..n]$ is an array of n distinct integers, sorted so that $A[1] < A[2] < \cdots < A[n]$. Each integer $A[i]$ could be positive, negative, or zero. Describe and analyze an efficient algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists.

   (b) Now suppose $A[1..n]$ is a sorted array of n distinct ***positive*** integers. Describe and analyze an even faster algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists.

2. A *double-Hamiltonian circuit* a closed walk in a graph that visits every vertex exactly *twice*. ***Prove*** that it is NP-hard to determine whether a given graph contains a double-Hamiltonian circuit.



This graph contains the double-Hamiltonian circuit $a \to b \to d \to g \to e \to b \to d \to c \to f \to a \to c \to f \to g \to e \to a$.

3. A palindrome is any string that is exactly the same as its reversal, like `I`, or `DEED`, or `HANNAH`, or `AMANAPLANACATACANALPANAMA`. Describe and analyze an algorithm to find the length of the longest subsequence of a given string that is also a palindrome.

   For example, the longest palindrome subsequence of `MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM` is `MHYMRORMYHM`, so given that string as input, your algorithm should return the integer 11.

4. Suppose you are given a magic black box that can determine **in polynomial time**, given an arbitrary graph G, the number of vertices in the largest complete subgraph of G. Describe and analyze a **polynomial-time** algorithm that computes, given an arbitrary graph G, a complete subgraph of G of maximum size, using this magic black box as a subroutine.

5. Suppose we are given a $4 \times n$ grid, where each grid cell has an integer value. Suppose we want to mark a subset of the grid cells, so that the total value of the marked cells is as large as possible. However, we are forbidden to mark any pair of grid cells that are immediate horizontal or vertical neighbors. (Marking diagonal neighbors is fine.) Describe and analyze an algorithm that computes the largest possible sum of marked cells, subject to this non-adjacency condition.

   For example, given the grid on the left below, your algorithm should return the integer 36, which is the sum of the circled numbers on the right.

4	−5	1	6
2	6	−1	8
5	4	3	3
1	−1	7	4
−3	4	5	−2

$\Longrightarrow$

④	−5	1	⑥
2	⑥	−1	8
⑤	4	3	③
1	−1	⑦	4
−3	⑤	4	−2

You may assume the following problems are NP-hard:

CircuitSat: Given a boolean circuit, are there any input values that make the circuit output True?

PlanarCircuitSat: Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

3Sat: Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

Max2Sat: Given a boolean formula in conjunctive normal form, with exactly two literals per clause, what is the largest number of clauses that can be satisfied by an assignment?

MaxIndependentSet: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MaxClique: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MinVertexCover: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MinDominatingSet: Given an undirected graph G, what is the size of the smallest subset S of vertices such that every vertex in G is either in S or adjacent to a vertex in S?

MinSetCover: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MinHittingSet: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3Color: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

ChromaticNumber: Given an undirected graph G, what is the minimum number of colors needed to color its vertices, so that every edge touches vertices with two different colors?

MaxCut: Given a graph G, what is the size (number of edges) of the largest bipartite subgraph of G?

HamiltonianCycle: Given a graph G, is there a cycle in G that visits every vertex exactly once?

HamiltonianPath: Given a graph G, is there a path in G that visits every vertex exactly once?

TravelingSalesman: Given a graph G with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

SubsetSum: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

Partition: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3Partition: Given a set X of n positive integers, can X be partitioned into $n/3$ three-element subsets, all with the same sum?

Minesweeper: Given a Minesweeper configuration and a particular square x, is it safe to click on x?

Tetris: Given a sequence of N Tetris pieces and a partially filled $n \times k$ board, is it possible to play every piece in the sequence without overflowing the board?

Sudoku: Given an $n \times n$ Sudoku puzzle, does it have a solution?

KenKen: Given an $n \times n$ Ken-Ken puzzle, does it have a solution?

This exam lasts 90 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet with your answers.

1. Assume we have access to a function RANDOM(k) that returns, given any positive integer k, an integer chosen independently and uniformly at random from the set $\{1, 2, \ldots, k\}$, in $O(1)$ time. For example, to perform a fair coin flip, we could call RANDOM(2).

   Now suppose we want to write an efficient function RANDOMPERMUTATION(n) that returns a permutation of the set $\{1, 2, \ldots, n\}$ chosen uniformly at random; that is, each permutation must be chosen with probability $1/n!$.

   (a) ***Prove*** that the following algorithm is ***not*** correct. *[Hint: Consider the case $n = 3$.]*

   ```
   RANDOMPERMUTATION(n):
     for i ← 1 to n
         π[i] ← i
     for i ← 1 to n
         swap π[i] ↔ π[RANDOM(n)]
     return π
   ```

   (b) Describe and analyze a correct RANDOMPERMUTATION algorithm that runs in $O(n)$ expected time. (In fact, $O(n)$ worst-case time is possible.)

2. Suppose we have n pieces of candy with weights $W[1..n]$ (in ounces) that we want to load into boxes. Our goal is to load the candy into as many boxes as possible, so that each box contains at least L ounces of candy. Describe an efficient 2-approximation algorithm for this problem. ***Prove*** that the approximation ratio of your algorithm is 2.

   (For 7 points partial credit, assume that every piece of candy weighs less than L ounces.)

3. The MAXIMUM-k-CUT problem is defined as follows. We are given a graph G with weighted edges and an integer k. Our goal is to partition the vertices of G into k subsets $S_1, S_2, \ldots, S_k$, so that the sum of the weights of the edges that cross the partition (that is, with endpoints in different subsets) is as large as possible.

   (a) Describe an efficient randomized approximation algorithm for MAXIMUM-k-CUT, and ***prove*** that its expected approximation ratio is ~~at most~~ $(k-1)/k$.

   (b) Now suppose we want to minimize the sum of the weights of edges that do *not* cross the partition. What expected approximation ratio does your algorithm from part (a) achieve for this new problem? ***Prove*** your answer is correct.

4. The citizens of Binaria use coins whose values are powers of two. That is, for any non-negative integer k, there are Binarian coins with value is 2^k bits. Consider the natural greedy algorithm to make x bits in change: If $x > 0$, use one coin with the largest denomination $d \leq x$ and then recursively make $x - d$ bits in change. (Assume you have an unlimited supply of each denomination.)

   (a) ***Prove*** that this algorithm uses at most one coin of each denomination.

   (b) ***Prove*** that this algorithm finds the minimum number of coins whose total value is x.

5. Any permutation π can be represented as a set of disjoint cycles, by considering the directed graph whose vertices are the integers between 1 and n and whose edges are $i \to \pi(i)$ for each i. For example, the permutation $\langle 5, 4, 2, 6, 7, 8, 1, 3, 9 \rangle$ has three cycles: $(175)(24683)(9)$.

   In the following questions, let π be a permutation of $\{1, 2, \ldots, n\}$ chosen uniformly at random, and let k be an arbitrary integer such that $1 \leq k \leq n$.

   (a) ***Prove*** that the probability that the number 1 lies in a cycle of length k in π is precisely $1/n$. *[Hint: Consider the cases $k = 1$ and $k = 2$.]*

   (b) What is the *exact* expected length of the cycle in π that contains the number 1?

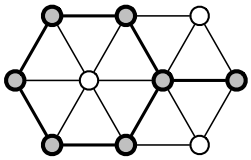
   (c) What is the *exact* expected number of cycles of length k in π?

   (d) What is the *exact* expected number of cycles in π?

   You may assume part (a) in your solutions to parts (b), (c), and (d).

This exam lasts 180 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet with your answers.

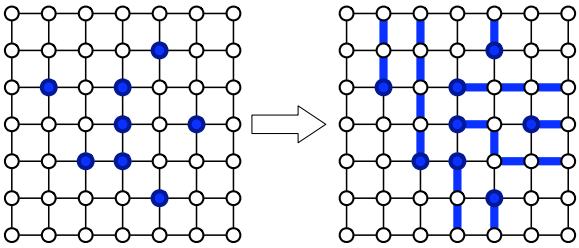
1. A subset S of vertices in an undirected graph G is called *triangle-free* if, for every triple of vertices $u, v, w \in S$, at least one of the three edges uv, uw, vw is *absent* from G. ***Prove*** that finding the size of the largest triangle-free subset of vertices in a given undirected graph is NP-hard.



A triangle-free subset of 7 vertices.
This is ***not*** the largest triangle-free subset in this graph.

2. An $n \times n$ grid is an undirected graph with n^2 vertices organized into n rows and n columns. We denote the vertex in the ith row and the jth column by (i, j). Every vertex in the grid have exactly four neighbors, except for the *boundary* vertices, which are the vertices (i, j) such that $i = 1$, $i = n$, $j = 1$, or $j = n$.

   Let $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ be distinct vertices, called *terminals*, in the $n \times n$ grid. The ***escape problem*** is to determine whether there are m vertex-disjoint paths in the grid that connect the terminals to any m distinct boundary vertices. Describe and analyze an efficient algorithm to solve the escape problem.



A positive instance of the escape problem, and its solution.

3. Consider the following problem, called UNIQUESETCOVER. The input is an n-element set S, together with a collection of m subsets $S_1, S_2, \ldots, S_m \subseteq S$, such that each element of S lies in exactly k subsets S_i. Our goal is to select some of the subsets so as to maximize the number of elements of S that lie in *exactly one* selected subset.

   (a) Fix a real number p between 0 and 1, and consider the following algorithm:

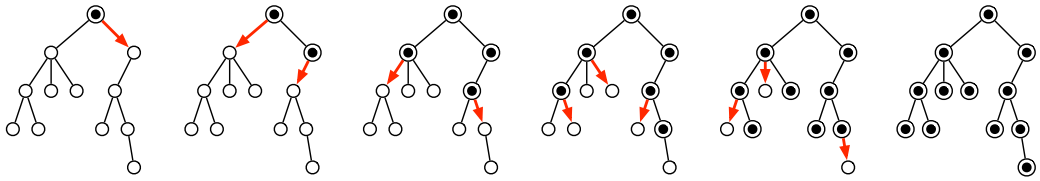
   > For each index i, select subset S_i independently with probability p.

   What is the *exact* expected number of elements that are uniquely covered by the chosen subsets? (Express your answer as a function of the parameters p and k.)

   (b) What value of p maximizes this expectation?

   (c) Describe a polynomial-time randomized algorithm for UNIQUESETCOVER whose expected approximation ratio is $O(1)$.

4. Suppose we need to distribute a message to all the nodes in a rooted tree. Initially, only the root node knows the message. In a single round, any node that knows the message can forward it to at most one of its children. Describe and analyze an efficient algorithm to compute the minimum number of rounds required for the message to be delivered to every node.



A message being distributed through a tree in five rounds.

5. Every year, Professor Dumbledore assigns the instructors at Hogwarts to various faculty committees. There are n faculty members and c committees. Each committee member has submitted a list of their *prices* for serving on each committee; each price could be positive, negative, zero, or even infinite. For example, Professor Snape might declare that he would serve on the Student Recruiting Committee for 1000 Galleons, that he would *pay* 10000 Galleons to serve on the Defense Against the Dark Arts Course Revision Committee, and that he would not serve on the Muggle Relations committee for any price.

   Conversely, Dumbledore knows how many instructors are needed for each committee, as well as a list of instructors who would be suitable members for each committee. (For example: "Dark Arts Revision: 5 members, anyone but Snape.") If Dumbledore assigns an instructor to a committee, he must pay that instructor's price from the Hogwarts treasury.

   Dumbledore needs to assign instructors to committees so that (1) each committee is full, (3) no instructor is assigned to more than three committees, (2) only suitable and willing instructors are assigned to each committee, and (4) the total cost of the assignment is as small as possible. Describe and analyze an efficient algorithm that either solves Dumbledore's problem, or correctly reports that there is no valid assignment whose total cost is finite.

6. Suppose we are given a rooted tree T, where every edge e has a non-negative *length* $\ell(e)$. Describe and analyze an efficient algorithm to assign a *stretched* length $s\ell(e) \geq \ell(e)$ to every edge e, satisfying the following conditions:

   - Every root-to-leaf path in T has the same total stretched length.
   - The total stretch $\sum_e (s\ell(e) - \ell(e))$ is as small as possible.

7. Let $G = (V, E)$ be a directed graph with edge capacities $c : E \to \mathbb{R}^+$, a source vertex s, and a target vertex t. Suppose someone hands you an *arbitrary* function $f : E \to \mathbb{R}$. Describe and analyze fast and *simple* algorithms to answer the following questions:

   (a) Is f a feasible (s, t)-flow in G?

   (b) Is f a *maximum* (s, t)-flow in G?

   (c) Is f the ***unique*** maximum (s, t)-flow in G?

Chernoff bounds:

If X is the sum of independent indicator variables and $\mu = \mathrm{E}[X]$, then

$$\Pr[X > (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \right)^{\mu} \qquad \text{for any } \delta > 0$$

$$\Pr[X > (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}} \right)^{\mu} \qquad \text{for any } 0 < \delta < 1$$

You may assume the following running times:

- Maximum flow or minimum cut: $O(E|f^*|)$ or $O(VE \log V)$
- Minimum-cost maximum flow: $O(E^2 \log^2 V)$

(These are *not* the best time bounds known, but they're close enough for the final exam.)

You may assume the following problems are NP-hard:

CircuitSat: Given a boolean circuit, are there any input values that make the circuit output True?

PlanarCircuitSat: Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

3Sat: Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

Max2Sat: Given a boolean formula in conjunctive normal form, with exactly two literals per clause, what is the largest number of clauses that can be satisfied by an assignment?

MaxIndependentSet: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MaxClique: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MinVertexCover: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MinSetCover: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MinHittingSet: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3Color: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

MaxCut: Given a graph G, what is the size (number of edges) of the largest bipartite subgraph of G?

HamiltonianCycle: Given a graph G, is there a cycle in G that visits every vertex exactly once?

HamiltonianPath: Given a graph G, is there a path in G that visits every vertex exactly once?

TravelingSalesman: Given a graph G with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

SubsetSum: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

Partition: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3Partition: Given a set X of n positive integers, can X be partitioned into $n/3$ three-element subsets, all with the same sum?

CS 473: Undergraduate Algorithms, Fall 2013
Homework 0

Due Tuesday, September 3, 2013 at 12:30pm

Quiz 0 (on the course Moodle page)
is also due Tuesday, September 3, 2013 at noon.

- **Please carefully read the course policies on the course web site.** These policies may be different than other classes you have taken. (For example: No late anything ever; "I don't know" is worth 25%, but "Repeat this for all n" is an automatic zero; ***every*** homework question requires a proof; collaboration is allowed, but you must cite your collaborators.) If you have *any* questions, please ask in lecture, in headbanging, on Piazza, in office hours, or by email.

- Homework 0 and Quiz 0 test your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** The course web page has pointers to several excellent online resources for prerequisite material. If you need help, please ask in headbanging, on Piazza, in office hours, or by email.

- ***Each student must submit individual solutions for these homework problems.*** You may use any source at your disposal—paper, electronic, or human—but you ***must*** cite ***every*** source that you use. For all future homeworks, groups of up to three students may submit joint solutions.

- **Submit your solutions on standard printer/copier paper, not notebook paper.** If you write your solutions by hand, please use the last three pages of this homework as a template. At the top of each page, please clearly print your name and NetID, and indicate your registered discussion section. Use both sides of the page. If you plan to typeset your homework, you can find a LaTeX template on the course web site; well-typeset homework will get a small amount of extra credit.

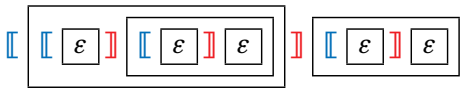
- Submit your solution to each numbered problem (stapled if necessary) in the corresponding drop box outside 1404 Siebel, ***or*** in the corresponding box in Siebel 1404 immediately before/after class. (This is the last homework we'll collect in 1404.) ***Do not staple your entire homework together.***

---

1. Consider the following recursively-defined sets of strings of left brackets ⟦ and right brackets ⟧:

   - A string x is ***balanced*** if it satisfies one of the following conditions:
     - x is the empty string, or
     - $x = ⟦\, y \,⟧\, z$, where y and z are balanced strings.

     For example, the following diagram shows that the string ⟦ ⟦ ⟧ ⟦ ⟧ ⟧ ⟦ ⟧ is balanced. Each boxed substring is balanced, and ε is the empty string.

   - A string x is ***erasable*** if it satisfies one of two conditions:
     - x is the empty string, or
     - $x = y ⟦\, ⟧\, z$, where yz is an erasable string.

     For example, we can prove that the string ⟦ ⟦ ⟧ ⟦ ⟧ ⟧ ⟦ ⟧ is erasable as follows:

     $$⟦\,⟦\,⟧\,⟦\,⟧\,⟧\,⟦\,⟧ \;\to\; ⟦\,⟦\,⟧\,⟧\,⟦\,⟧ \;\to\; ⟦\,⟧\,⟦\,⟧ \;\to\; ⟦\,⟧ \;\to\; \varepsilon$$

   Your task is to prove that these two definitions are equivalent.

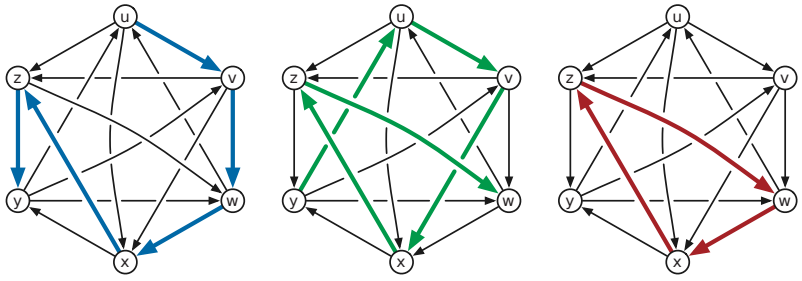
   (a) Prove that every balanced string is erasable.

   (b) Prove that every erasable string is balanced.

2. A ***tournament*** is a directed graph with exactly one directed edge between each pair of vertices. That is, for any vertices v and w, a tournament contains either an edge $v \to w$ or an edge $w \to v$, but not both. A ***Hamiltonian path*** in a directed graph G is a directed path that visits every vertex of G exactly once.

   (a) Prove that every tournament contains a Hamiltonian path.

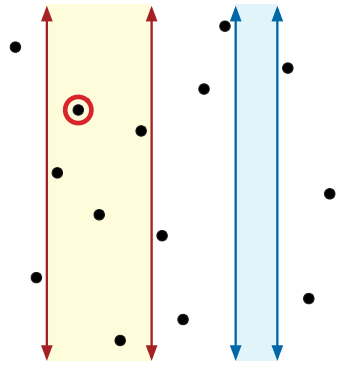
   (b) Prove that every tournament contains either *exactly one* Hamiltonian path or a directed cycle of length three.



A tournament with two Hamiltonian paths $u \to v \to w \to x \to z \to y$ and $y \to u \to v \to x \to z \to w$ and a directed triangle $w \to x \to z \to w$.

3. Suppose you are given a set $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ of n points in the plane with distinct x- and y-coordinates. Describe a data structure that can answer the following query as quickly as possible:

   > Given two numbers l and r, find the highest point in P inside the vertical slab $l < x < r$. More formally, find the point $(x_i, y_i) \in P$ such that $l < x_i < r$ and y_i is as large as possible. Return NONE if the slab does not contain any points in P.



A query with the left slab returns the indicated point.
A query with the right slab returns NONE.

To receive full credit, your solution must include (a) a concise description of your data structure, (b) a concise description of your query algorithm, (c) a proof that your query algorithm is correct, (d) a bound on the size of your data structure, and (e) a bound on the running time of your query algorithm. You do ***not*** need to describe or analyze an algorithm to construct your data structure.

Smaller data structures and faster query times are worth more points.

Starting with this homework, groups of up to three students may submit a single solution for each numbered problem. Every student in the group receives the same grade. Groups can be different for different problems.

---

1. Consider the following cruel and unusual sorting algorithm.

```
CRUEL(A[1..n]):
  if n > 1
      CRUEL(A[1..n/2])
      CRUEL(A[n/2+1..n])
      UNUSUAL(A[1..n])
```

```
UNUSUAL(A[1..n]):
  if n = 2
      if A[1] > A[2]                      ⟨⟨the only comparison!⟩⟩
          swap A[1] ↔ A[2]
  else
      for i ← 1 to n/4                    ⟨⟨swap 2nd and 3rd quarters⟩⟩
          swap A[i + n/4] ↔ A[i + n/2]
      UNUSUAL(A[1..n/2])                  ⟨⟨recurse on left half⟩⟩
      UNUSUAL(A[n/2+1..n])                ⟨⟨recurse on right half⟩⟩
      UNUSUAL(A[n/4+1..3n/4])             ⟨⟨recurse on middle half⟩⟩
```

Notice that the comparisons performed by the algorithm do not depend at all on the values in the input array; such a sorting algorithm is called ***oblivious***. Assume for this problem that the input size n is always a power of 2.

   (a) Prove that CRUEL correctly sorts any input array. *[Hint: Consider an array that contains $n/4$ 1s, $n/4$ 2s, $n/4$ 3s, and $n/4$ 4s. Why is considering this special case enough? What does* UNUSUAL *actually do?]*

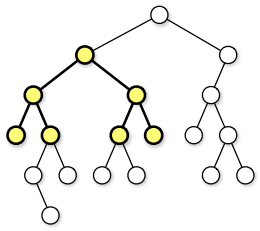
   (b) Prove that CRUEL would *not* always sort correctly if we removed the for-loop from UNUSUAL.

   (c) Prove that CRUEL would *not* always sort correctly if we swapped the last two lines of UNUSUAL.

   (d) What is the running time of UNUSUAL? Justify your answer.

   (e) What is the running time of CRUEL? Justify your answer.

2. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

3. (a) Suppose we are given two sorted arrays $A[1..n]$ and $B[1..n]$. Describe an algorithm to find the median of the union of A and B in $O(\log n)$ time. Assume the arrays contain no duplicate elements.

   (b) Now suppose we are given *three* sorted arrays $A[1..n]$, $B[1..n]$, and $C[1..n]$. Describe an algorithm to find the median element of $A \cup B \cup C$ in $O(\log n)$ time.

*4. ***Extra credit; due September 17.*** (The "I don't know" rule does not apply to extra credit problems.)

Bob Ratenbur, a new student in CS 225, is trying to write code to perform preorder, inorder, and postorder traversal of binary trees. Bob understands the basic idea behind the traversal algorithms, but whenever he tries to implement them, he keeps mixing up the recursive calls. Five minutes before the deadline, Bob submitted code with the following structure:

```
PreOrder(v):
  if v = Null
      return
  else
      print label(v)
      [redacted]Order(left(v))
      [redacted]Order(right(v))
```

```
InOrder(v):
  if v = Null
      return
  else
      [redacted]Order(left(v))
      print label(v)
      [redacted]Order(right(v))
```

```
PostOrder(v):
  if v = Null
      return
  else
      [redacted]Order(left(v))
      [redacted]Order(right(v))
      print label(v)
```

Each [redacted] represents either Pre, In, or Post. Moreover, each of the following function calls appears exactly once in Bob's submitted code:

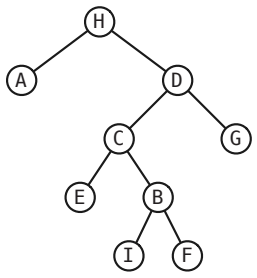
PreOrder(*left*(*v*))	InOrder(*left*(*v*))	PostOrder(*left*(*v*))
PreOrder(*right*(*v*))	InOrder(*right*(*v*))	PostOrder(*right*(*v*))

Thus, there are exactly 36 possibilities for Bob's code. Unfortunately, Bob accidentally deleted his source code after submitting the executable, so neither you nor he knows which functions were called where.

Your task is to reconstruct a binary tree T from the output of Bob's traversal algorithms, which has been helpfully parsed into three arrays $Pre[1..n]$, $In[1..n]$, and $Post[1..n]$. Your algorithm should return the unknown tree T. You may assume that the vertex labels of the unknown tree are distinct, and that every internal node has exactly two children. For example, given the input

$$Pre[1..n] = [\texttt{H A E C B I F G D}]$$
$$In[1..n] = [\texttt{A H D C E I F B G}]$$
$$Post[1..n] = [\texttt{A E I B F C D G H}]$$

your algorithm should return the following tree:



In general, the traversal sequences may not give you enough information to reconstruct Bob's code; however, to produce the example sequences above, Bob's code must look like this:

```
PreOrder(v):
  if v = Null
      return
  else
      print label(v)
      PreOrder(left(v))
      PostOrder(right(v))
```

```
InOrder(v):
  if v = Null
      return
  else
      PostOrder(left(v))
      print label(v)
      PreOrder(right(v))
```

```
PostOrder(v):
  if v = Null
      return
  else
      InOrder(left(v))
      InOrder(right(v))
      print label(v)
```

1. Suppose we are given an array $A[1..n]$ of integers, some positive and negative, which we are asked to partition into contiguous subarrays, which we call ***chunks***. The *value* of any chunk is the *square* of the sum of elements in that chunk; the value of a partition of A is the sum of the values of its chunks.

   For example, suppose $A = [3,-1,4,-1,5,-9]$. The partition $[3,-1,4],[-1,5],[-9]$ has three chunks with total value $(3-1+4)^2 + (-1+5)^2 + (-9)^2 = 6^2 + 4^2 + 9^2 = 133$, while the partition $[3,-1],[4,-1,5,-9]$ has two chunks with total value $(3-1)^2 + (4-1+5-9)^2 = 5$.

   (a) Describe and analyze an algorithm that computes the minimum-value partition of a given array of n numbers.

   (b) Now suppose we also given an integer $k > 0$. Describe and analyze an algorithm that computes the minimum-value partition of a given array of n numbers ***into at most k chunks***.

2. Consider the following solitaire form of Scrabble. We begin with a fixed, finite sequence of tiles; each tile contains a letter and a numerical value. At the start of the game, we draw the seven tiles from the sequence and put them into our hand. In each turn, we form an English word from some or all of the tiles in our hand, place those tiles on the table, and receive the total value of those tiles as points. If no English word can be formed from the tiles in our hand, the game immediately ends. Then we repeatedly draw the next tile from the start of the sequence until either (a) we have seven tiles in our hand, or (b) the sequence is empty. (Sorry, no double/triple word/letter scores, bingos, blanks, or passing.) Our goal is to obtain as many points as possible.

   For example, suppose we are given the tile sequence

   $$\boxed{I_2}\boxed{N_2}\boxed{X_8}\boxed{A_1}\boxed{N_2}\boxed{A_1}\boxed{D_3}\boxed{U_5}\boxed{D_3}\boxed{I_2}\boxed{D_3}\boxed{K_8}\boxed{U_5}\boxed{B_4}\boxed{L_2}\boxed{A_1}\boxed{K_8}\boxed{H_5}\boxed{A_1}\boxed{N_2}.$$

   Then we can earn 68 points as follows:

   - We initially draw $\boxed{I_2}\boxed{N_2}\boxed{X_8}\boxed{A_1}\boxed{N_2}\boxed{A_1}\boxed{D_3}$.
   - Play the word $\boxed{N_2}\boxed{A_1}\boxed{I_2}\boxed{A_1}\boxed{D_3}$ for 9 points, leaving $\boxed{N_2}\boxed{X_8}$ in our hand.
   - Draw the next five tiles $\boxed{U_5}\boxed{D_3}\boxed{I_2}\boxed{D_3}\boxed{K_8}$.
   - Play the word $\boxed{U_5}\boxed{N_2}\boxed{D_3}\boxed{I_2}\boxed{D_3}$ for 15 points, leaving $\boxed{K_8}\boxed{X_8}$ in our hand.
   - Draw the next five tiles $\boxed{U_5}\boxed{B_4}\boxed{L_2}\boxed{A_1}\boxed{K_8}$.
   - Play the word $\boxed{B_4}\boxed{U_5}\boxed{L_2}\boxed{K_8}$ for 19 points, leaving $\boxed{K_8}\boxed{X_8}\boxed{A_1}$ in our hand.
   - Draw the next three tiles $\boxed{H_5}\boxed{A_1}\boxed{N_2}$, emptying the list.
   - Play the word $\boxed{A_1}\boxed{N_2}\boxed{K_8}\boxed{H_5}$ for 16 points, leaving $\boxed{X_8}\boxed{A_1}$ in our hand.
   - Play the word $\boxed{A_1}\boxed{X_8}$ for 9 points, emptying our hand and ending the game.

   Design and analyze an algorithm to compute the maximum number of points that can be earned from a given sequence of tiles. ***The input consists of two arrays*** Letter$[1..n]$***, containing a sequence of letters between*** A ***and*** Z***, and*** Value$[$A$..$Z$]$***, where*** Value$[i]$ ***is the value of letter*** i. The output is a single number. Assume that you can find all English words that can be made from any seven tiles, along with the point values of those words, in $O(1)$ time.

3. ***Extra credit.*** Submit your answer to Homework 1 problem 4.

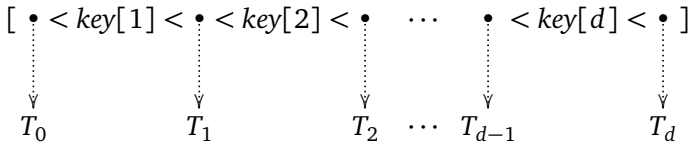
1. A standard method to improve the cache performance of search trees is to pack more search keys and subtrees into each node. A ***B-tree*** is a rooted tree in which each internal node stores up to B keys and pointers to up to $B+1$ children, each the root of a smaller B-tree. Specifically each node v stores three fields:

   - a positive integer $v.d \le B$,
   - a *sorted* array $v.key[1..v.d]$, and
   - an array $v.child[0..v.d]$ of child pointers.

   In particular, the number of child pointers is always exactly one more than the number of keys.

   Each pointer $v.child[i]$ is either NULL or a pointer to the root of a B-tree whose keys are all larger than $v.key[i]$ and smaller than $v.key[i+1]$. In particular, all keys in the leftmost subtree $v.child[0]$ are smaller than $v.key[1]$, and all keys in the rightmost subtree $v.child[v.d]$ are larger than $v.key[v.d]$.

   Intuitively, you should have the following picture in mind:

   Here T_i is the subtree pointed to by $child[i]$.

   The ***cost*** of searching for a key x in a B-tree is the number of nodes in the path from the root to the node containing x as one of its keys. A 1-tree is just a standard binary search tree.

   Fix an arbitrary positive integer $B > 0$. (I suggest $B = 8$.) Suppose we are given a sorted array $A[1,\ldots,n]$ of search keys and a corresponding array $F[1,\ldots,n]$ of frequency counts, where $F[i]$ is the number of times that we will search for $A[i]$.

   Describe and analyze an efficient algorithm to find a B-tree that minimizes the total cost of searching for n keys with a given array of frequencies.

   - For 5 points, describe a polynomial-time algorithm for the special case $B = 2$.
   - For 10 points, describe an algorithm for arbitrary B that runs in $O(n^{B+c})$ time for some fixed integer c.
   - For 15 points, describe an algorithm for arbitrary B that runs in $O(n^c)$ time for some fixed integer c that does *not* depend on B.

   Like all other homework problems, 10 points is full credit; any points above 10 will be awarded as extra credit.

   A few comments about *B*-trees. Normally, B-trees are required to satisfy two additional constraints, which guarantee a worst-case search cost of $O(\log_B n)$: Every leaf must have exactly the same depth, and every node except possibly the root must contain at least $B/2$ keys. However, in this problem, we are not interested in optimizing the *worst-case* search cost, but rather the *total* cost of a sequence of searches, so we will not impose these additional constraints.

   In most large database systems, the parameter B is chosen so that each node exactly fits in a cache line. Since the entire cache line is loaded into cache anyway, and the cost of loading a cache

line exceeds the cost of searching within the cache, the running time is dominated by the number of cache faults. This effect is even more noticeable if the data is too big to lie in RAM at all; then the cost is dominated by the number of page faults, and B should be roughly the size of a page.

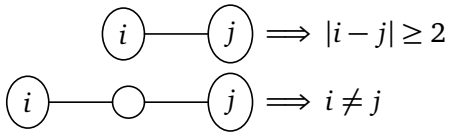
Finally, don't worry about the cache/disk performance in your homework solutions; just analyze the CPU time as usual. Designing algorithms with few cache misses or page faults is a interesting pastime; simultaneously optimizing CPU time *and* cache misses *and* page faults is even more interesting. But this kind of design and analysis requires tools we won't see in this class.

2. ***Extra credit***, because we screwed up the first version.

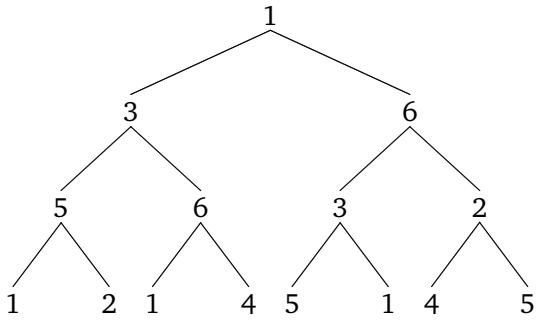
A ***k-2 coloring*** of a tree assigns each vertex a value from the set $\{1, 2, \ldots, k\}$, called its *color*, such that the following constraints are satisfied:

- A node and its parent cannot have the same color or adjacent colors.
- A node and its grandparent cannot have the same color.
- Two nodes with the same parent cannot have the same color.

The last two rules can be written more simply as "Two nodes that are two edges apart cannot have the same color." Diagrammatically, if we write the names of the colors inside the vertices,



For example, here is a valid 6-2 coloring of the complete binary tree with depth 3:



(a) Describe and analyze an algorithm that computes a 6-2 coloring of a given binary tree. The existence of such an algorithm proves that every binary tree has a 6-2 coloring.

(b) Prove that not every binary tree has a 5-2 coloring.

(c) A *ternary* tree is a rooted tree where every node has at most *three* children. What is the smallest integer k such that every ternary tree has a k-2 coloring? Prove your answer is correct.

1. A *meldable priority queue* stores a set of values, called *priorities*, from some totally-ordered universe (such as the integers) and supports the following operations:

   - MAKEQUEUE: Return a new priority queue containing the empty set.
   - FINDMIN(Q): Return the smallest element of Q (if any).
   - DELETEMIN(Q): Remove the smallest element in Q (if any).
   - INSERT(Q, x): Insert priority x into Q, if it is not already there.
   - DECREASE(Q, x, y): Replace some element $x \in Q$ with a smaller priority y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q containing x.
   - DELETE(Q, x): Delete the priority $x \in Q$. The input is a pointer directly to the node in Q containing x.
   - MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2; this operation destroys Q_1 and Q_2.

   A simple way to implement such a data structure is to use a ***heap-ordered binary tree*** — each node stores a priority, which is smaller than the priorities of its children, along with pointers to its parent and at most two children. MELD can be implemented using the following randomized algorithm:

```
MELD(Q1, Q2):
  if Q1 is empty return Q2
  if Q2 is empty return Q1

  if priority(Q1) > priority(Q2)
      swap Q1 ↔ Q2

  with probability 1/2
      left(Q1) ← MELD(left(Q1), Q2)
  else
      right(Q1) ← MELD(right(Q1), Q2)

  return Q1
```

   (a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (***not*** just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. *[Hint: How long is a random root-to-leaf path in an n-node binary tree if each left/right choice is made uniformly and independently at random?]*

   (b) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ expected time.)

2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *treap* is a priority search tree whose search keys are given by the user and whose priorities are independent random numbers.

   A ***heater*** is a priority search tree whose *priorities* are given by the user and whose *search keys* are distributed uniformly and independently at random in the real interval $[0, 1]$. Intuitively, a heater is a sort of anti-treap.[1]

[1] There are those who think that life has nothing left to chance, a host of holy horrors to direct our aimless dance.

The following problems consider an n-node heater T. We identify nodes in T by their *priority rank*; for example, "node 5" means the node in T with the 5th smallest priority. The min-heap property implies that node 1 is the root of T. You may assume all search keys and priorities are distinct. Finally, let i and j be arbitrary integers with $1 \le i < j \le n$.

(a) Prove that if we permute the set $\{1, 2, \ldots, n\}$ uniformly at random, integers i and j are adjacent with probability $2/n$.

(b) Prove that node i is an ancestor of node j with probability $2/(i+1)$. *[Hint: Use part (a)!]*

(c) What is the probability that node i is a *descendant* of node j? *[Hint: Don't use part (a)!]*

(d) What is the *exact* expected depth of node j?

(e) Describe and analyze an algorithm to insert a new item into an n-node heater.

(f) Describe and analyze an algorithm to delete the smallest priority (the root) from an n-node heater.

*3. ***Extra credit; due October 15.*** In the usual theoretical presentation of treaps, the priorities are random real numbers chosen uniformly from the interval $[0, 1]$. In practice, however, computers have access only to random *bits*. This problem asks you to analyze an implementation of treaps that takes this limitation into account.

Suppose the priority of a node v is abstractly represented as an infinite sequence $\pi_v[1..\infty]$ of random bits, which is interpreted as the rational number

$$\mathit{priority}(v) = \sum_{i=1}^{\infty} \pi_v[i] \cdot 2^{-i}.$$

However, only a finite number ℓ_v of these bits are actually known at any given time. When a node v is first created, *none* of the priority bits are known: $\ell_v = 0$. We generate (or "reveal") new random bits only when they are necessary to compare priorities. The following algorithm compares the priorities of any two nodes in $O(1)$ expected time:

```
LargerPriority(v, w):
  for i ← 1 to ∞
      if i > ℓ_v
          ℓ_v ← i;  π_v[i] ← RandomBit
      if i > ℓ_w
          ℓ_w ← i;  π_w[i] ← RandomBit
      if π_v[i] > π_w[i]
          return v
      else if π_v[i] < π_w[i]
          return w
```

Suppose we insert n items one at a time into an initially empty treap. Let $L = \sum_v \ell_v$ denote the total number of random bits generated by calls to LargerPriority during these insertions.

(a) Prove that $E[L] = \Theta(n)$.

(b) Prove that $E[\ell_v] = \Theta(1)$ for any node v. *[Hint: This is equivalent to part (a). Why?]*

(c) Prove that $E[\ell_{\text{root}}] = \Theta(\log n)$. *[Hint: Why doesn't this contradict part (b)?]*

1. Recall that a standard (FIFO) queue maintains a sequence of items subject to the following operations.

   - PUSH(x): Add item x to the end of the sequence.
   - PULL(): Remove and return the item at the beginning of the sequence.
   - SIZE(): Return the current number of items in the sequence.

   It is easy to implement a queue using a doubly-linked list, so that it uses $O(n)$ space (where n is the number of items in the queue) and the worst-case time for each of these operations is $O(1)$.

   Consider the following new operation, which removes every tenth element from the queue, starting at the beginning, in $\Theta(n)$ worst-case time.

```
DECIMATE():
  n ← SIZE()
  for i ← 0 to n − 1
      if i mod 10 = 0
          PULL()    ⟨⟨result discarded⟩⟩
      else
          PUSH(PULL())
```

   Prove that in any intermixed sequence of PUSH, PULL, and DECIMATE operations, the amortized cost of each operation is $O(1)$.

2. ***This problem is extra credit***, because the original problem statement had several confusing small errors. I believe these erors are corrected in the current revision.

   Deleting an item from an open-addressed hash table is not as straightforward as deleting from a chained hash table. The obvious method for deleting an item x simply empties the entry in the hash table that contains x. Unfortunately, the obvious method doesn't always work. (Part (a) of this question asks you to prove this.)

   Knuth proposed the following *lazy* deletion strategy. Every cell in the table stores both an *item* and a *label*; the possible labels are **EMPTY**, **FULL**, and **JUNK**. The DELETE operation marks cells as **JUNK** instead of actually erasing their contents. Then FIND pretends that **JUNK** cells are occupied, and INSERT pretends that **JUNK** cells are actually empty. In more detail:

```
FIND(H, x):
  for i ← 0 to m − 1
      j ← h_i(x)
      if H.label[j] = FULL and H.item[j] = x
          return j
      else if H.label[j] = EMPTY
          return NONE
```

```
INSERT(H, x):
  for i ← 0 to m − 1
      j ← h_i(x)
      if H.label[j] = FULL and H.item[j] = x
          return        ⟨⟨already there⟩⟩
      if H.label[j] ≠ FULL
          H.item[j] ← x
          H.label[j] ← FULL
          return
```

```
DELETE(H, x):
  j ← FIND(H, x)
  if j ≠ NONE
      H.label[j] ← JUNK
```

Lazy deletion is always *correct*, but it is only *efficient* if we don't perform too many deletions. The search time depends on the fraction of non-**EMPTY** cells, not on the number of actual items stored in the table; thus, even if the number of items stays small, the table may fill up with **JUNK** cells, causing unsuccessful searches to scan the entire table. Less significantly, the data structure may use significantly more space than necessary for the number of items it actually stores. To avoid both of these issues, we use the following rebuilding rules:

- After each INSERT operation, if less than $1/4$ of the cells are **EMPTY**, rebuild the hash table.
- After each DELETE operation, if less than $1/4$ of the cells are **FULL**, rebuild the hash table.

To rebuild the hash table, we allocate a new hash table whose size is twice the number of **FULL** cells (unless that number is smaller than some fixed constant), INSERT each item in a **FULL** cell in the old hash table into the new hash table, and then discard the old hash table, as follows:

```
REBUILD(H):
  count ← 0
  for j ← 0 to H.size − 1
      if H.label[j] = FULL
          count ← count + 1
  H′ ← new hash table of size max{2 · count, 32}
  for j ← 0 to H.size − 1
      if H.label[j] = FULL
          INSERT(H′, H.item[j])
  discard H
  return H′
```

Finally, here are your actual homework questions!

(a) Describe a ***small*** example where the "obvious" deletion algorithm is incorrect; that is, show that the hash table can reach a state where a search can return the wrong result. Assume collisions are resolved by linear probing.

(b) Suppose we use Knuth's lazy deletion strategy instead. Prove that after several INSERT and DELETE operations into a table of arbitrary size m, it is possible for a single item x to be stored in *almost half* of the table cells. (However, at most one of those cells can be labeled **FULL**.)

(c) For purposes of analysis,[1] suppose FIND and INSERT run in $O(1)$ time when at least $1/4$ of the table cells are **EMPTY**. Prove that in any intermixed sequence of INSERT and DELETE operations, using Knuth's lazy deletion strategy, the amortized time per operation is $O(1)$.

*3. ***Extra credit.*** Submit your answer to Homework 4 problem 3.

[1] In fact, FIND and INSERT run in $O(1)$ *expected* time when at least $1/4$ of the table cells are **EMPTY**, and therefore each INSERT and DELETE takes $O(1)$ *expected* amortized time. But probability doesn't play any role whatsoever in the amortized analysis, so we can safely ignore the word "expected".

1. Suppose we want to maintain an array $X[1..n]$ of bits, which are all initially subject to the following operations.

   - LOOKUP(i): Given an index i, return $X[i]$.
   - BLACKEN(i): Given an index $i < n$, set $X[i] \leftarrow 1$.
   - NEXTWHITE(i): Given an index i, return the smallest index $j \geq i$ such that $X[j] = 0$. (Because we never change $X[n]$, such an index always exists.)

   If we use the array $X[1..n]$, it is trivial to implement LOOKUP and BLACKEN in $O(1)$ time and NEXTWHITE in $O(n)$ time. But you can do better! Describe data structures that support LOOKUP in $O(1)$ worst-case time and the other two operations in the following time bounds. (We want a different data structure for each set of time bounds, not one data structure that satisfies all bounds simultaneously!)

   (a) The worst-case time for both BLACKEN and NEXTWHITE is $O(\log n)$.

   (b) The amortized time for both BLACKEN and NEXTWHITE is $O(\log n)$. In addition, the *worst-case* time for BLACKEN is $O(1)$.

   (c) The amortized time for BLACKEN is $O(\log n)$, and the worst-case time for NEXTWHITE is $O(1)$.

   (d) The worst-case time for BLACKEN is $O(1)$, and the amortized time for NEXTWHITE is $O(\alpha(n))$. *[Hint: There is no* WHITEN*.]*

2. Recall that a standard (FIFO) queue maintains a sequence of items subject to the following operations:

   - PUSH(x): Add item x to the back of the queue (the end of the sequence).
   - PULL(): Remove and return the item at the front of the queue (the beginning of the sequence).

   It is easy to implement a queue using a doubly-linked list and a counter, using $O(n)$ space altogether, so that each PUSH or PULL requires $O(1)$ time.

   (a) Now suppose we want to support the following operation instead of PULL:

   - MULTIPULL(k): Remove the first k items from the front of the queue, and return the kth item removed.

   Suppose further that we implement MULTIPULL using the obvious algorithm:

```
MULTIPULL(k):
  for i ← 1 to k
      x ← PULL()
  return x
```

   Prove that in any intermixed sequence of PUSH and MULTIPULL operations, starting with an empty queue, the amortized cost of each operation is $O(1)$. You may assume that k is never larger than the number of items in the queue.

   (b) Now suppose we ***also*** want to support the following operation instead of PUSH:

   - MULTIPUSH(x, k): Insert k copies of x into the back of the queue.

   Suppose further that we implement MULTIPUSH using the obvious algorithm:

```
MULTIPUSH(k, x):
  for i ← 1 to k
    PUSH(x)
```

Prove that for any integers ℓ and n, there is a sequence of ℓ MULTIPUSH and MULTIPULL operations that require $\Omega(n\ell)$ time, where n is the maximum number of items in the queue at any time. Such a sequence implies that the amortized cost of each operation is $\Omega(n)$.

(c) Finally, describe a data structure that supports arbitrary intermixed sequences of MULTIPUSH and MULTIPULL operations in $O(1)$ amortized cost per operation. Like a standard queue, your data structure must use only $O(1)$ space per item. *[Hint: **Don't** use the obvious algorithms!]*

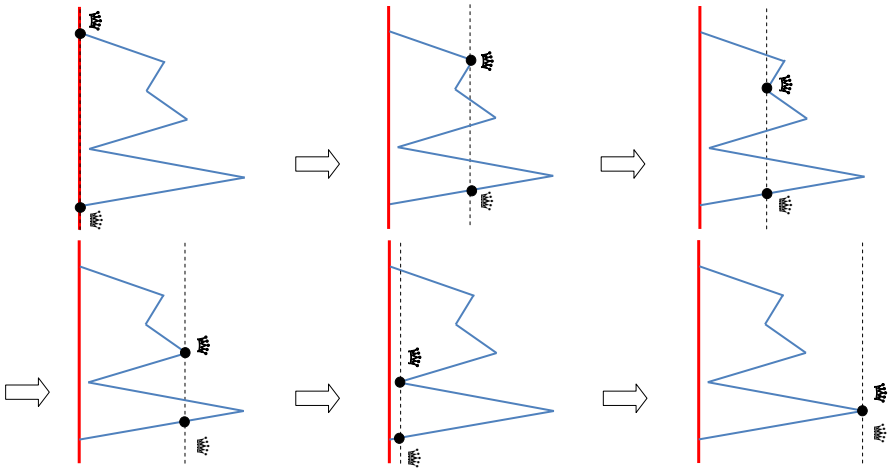
3. In every cheesy romance movie there's always that scene where the romantic couple, physically separated and looking for one another, suddenly matches eyes and then slowly approach one another with unwavering eye contact as the music rolls and in and the rain lifts and the sun shines through the clouds and kittens and puppies. . . .

Suppose a romantic couple—in grand computer science tradition, named Alice and Bob—enters a park from the northwest and southwest corners of the park, locked in dramatic eye contact. However, they can't just walk to one another in a straight line, because the paths of the park zig-zag between the northwest and southwest entrances. Instead, Alice and Bob must traverse the zig-zagging path so that their eyes are always locked perfectly in vertical eye-contact; thus, their x-coordinates must always be identical.

We can describe the zigzag path as an array $P[0..n]$ of points, which are the corners of the path in order from the southwest endpoint to the northwest endpoint, satisfying the following conditions:

- $P[i].y > P[i-1].y$ for every index i. That is, the path always moves upward.
- $P[0].x = P[n].x = 0$, and $P[i].x > 0$ for every index $1 \le i \le n-1$. Thus, the ends of the path are further to the left then any other point on the path.

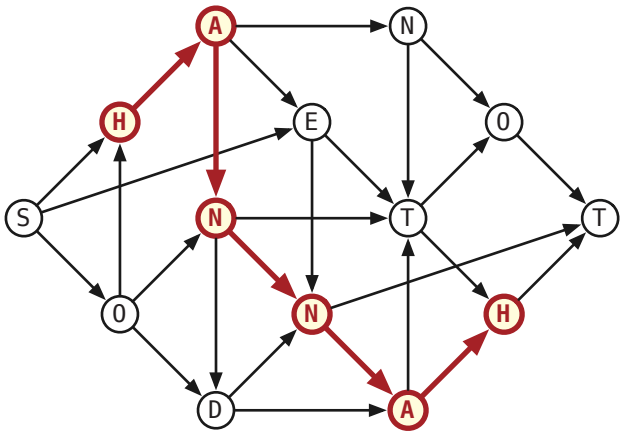
Prove that Alice and Bob can *always* meet.[1] *[Hint: Describe a graph that models all possible locations of the couple along the path. What are the vertices of this graph? What are the edges? What can we say about the degrees of the vertices?]*



[1] It follows that every cheesy romance movie (that reaches this scene) must have a happy, sappy ending.

1. Suppose we are given a directed acyclic graph G with labeled vertices. Every path in G has a label, which is a string obtained by concatenating the labels of its vertices in order. Recall that a *palindrome* is a string that is equal to its reversal.

   Describe and analyze an algorithm to find the length of the longest palindrome that is the label of a path in G. For example, given the graph below, your algorithm should return the integer 6, which is the length of the palindrome **HANNAH**.



2. Let G be a connected directed graph that contains both directions of every edge; that is, if $u \to v$ is an edge in G, its reversal $v \to u$ is also an edge in G. Consider the following non-standard traversal algorithm.

```
SpaghettiTraversal(G):
  for all vertices v in G
      unmark v
  for all edges u→v in G
      color u→v white
  s ← any vertex in G
  Spaghetti(s)
```

```
Spaghetti(v):
  mark v                                  ⟨⟨"visit v"⟩⟩
  if there is a white arc v→w
      if w is unmarked
          color w→v green
      color v→w red                       ⟨⟨"traverse v→w"⟩⟩
      Spaghetti(w)
  else if there is a green arc v→w
      color v→w red                       ⟨⟨"traverse v→w"⟩⟩
      Spaghetti(w)
  ⟨⟨else every arc v→w is red, so halt⟩⟩
```

   We informally say that this algorithm "visits" vertex v every time it marks v, and it "traverses" edge $v \to w$ when it colors that edge red. Unlike our standard graph-traversal algorithms, Spaghetti may (in fact, *will*) mark/visit each vertex more than once.

   The following series of exercises leads to a proof that Spaghetti traverses each directed edge of G exactly once. Most of the solutions are very short.

   (a) Prove that no directed edge in G is traversed more than once.

   (b) When the algorithm visits a vertex v for the kth time, exactly how many edges into v are red, and exactly how many edges out of v are red? *[Hint: Consider the starting vertex s separately from the other vertices.]*

(c) Prove each vertex v is visited at most $\deg(v)$ times, except the starting vertex s, which is visited at most $\deg(s)+1$ times. This claim immediately implies that SPAGHETTITRAVERSAL(G) terminates.

(d) Prove that when SPAGHETTITRAVERSAL(G) ends, the last visited vertex is the starting vertex s.

(e) For every vertex v that SPAGHETTITRAVERSAL(G) visits, prove that all edges incident to v (either in or out) are red when SPAGHETTITRAVERSAL(G) halts. *[Hint: Consider the vertices in the order that they are marked for the first time, starting with s, and prove the claim by induction.]*

(f) Prove that SPAGHETTITRAVERSAL(G) visits every vertex of G.

(g) Finally, prove that SPAGHETTITRAVERSAL(G) traverses every edge of G exactly once.

1. Let G be a directed graph with (possibly negative!) edge weights, and let s be an arbitrary vertex of G. Suppose every vertex $v \neq s$ stores a pointer $pred(v)$ to another vertex in G.

   Describe and analyze an algorithm to determine whether these predecessor pointers define a single-source shortest path tree rooted at s. Do ***not*** assume that the graph G has no negative cycles.

   [Hint: There is a similar problem in head-banging, where you're given distances instead of predecessor pointers.]

2. Let G be a directed graph with positive edge weights, and let s and t be an arbitrary vertices of G. Describe an algorithm to determine the *number* of different shortest paths in G from s to t. Assume that you can perform arbitrary arithmetic operations in $O(1)$ time. *[Hint: Which edges of G belong to shortest paths from s to t?]*

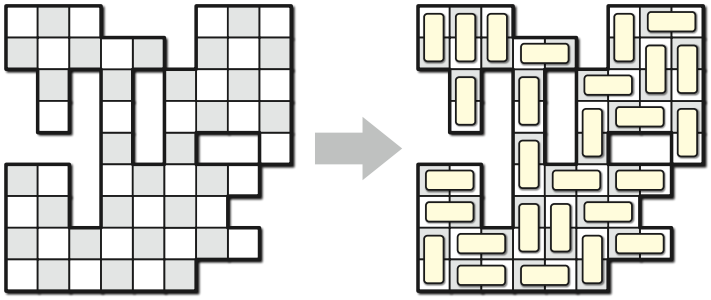
3. Describe and analyze and algorithm to find the second smallest spanning tree of a given undirected graph G with weighted edges, that is, the spanning tree of G with smallest total weight except for the minimum spanning tree.

1. You're organizing the First Annual UIUC Computer Science 72-Hour Dance Exchange, to be held all day Friday, Saturday, and Sunday. Several 30-minute sets of music will be played during the event, and a large number of DJs have applied to perform. You need to hire DJs according to the following constraints.

   - Exactly k sets of music must be played each day, and thus $3k$ sets altogether.
   - Each set must be played by a single DJ in a consistent music genre (ambient, bubblegum, dubstep, horrorcore, hyphy, trip-hop, Nitzhonot, Kwaito, J-pop, Nashville country, ...).
   - Each genre must be played at most once per day.
   - Each candidate DJ has given you a list of genres they are willing to play.
   - Each DJ can play at most three sets during the entire event.

   Suppose there are n candidate DJs and g different musical genres available. Describe and analyze an efficient algorithm that either assigns a DJ and a genre to each of the $3k$ sets, or correctly reports that no such assignment is possible.

2. Suppose you are given an $n \times n$ checkerboard with some of the squares deleted. You have a large set of dominos, just the right size to cover two squares of the checkerboard. Describe and analyze an algorithm to determine whether one can tile the board with dominos—each domino must cover exactly two undeleted squares, and each undeleted square must be covered by exactly one domino.



   Your input is a two-dimensional array $Deleted[1..n, 1..n]$ of bits, where $Deleted[i, j] = \text{TRUE}$ if and only if the square in row i and column j has been deleted. Your output is a single bit; you do ***not*** have to compute the actual placement of dominos. For example, for the board shown above, your algorithm should return TRUE.

3. Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers. We want to *round* A to an integer matrix, by replacing each entry x in A with either $\lfloor x \rfloor$ or $\lceil x \rceil$, without changing the sum of entries in any row or column of A. For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longmapsto \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

   Describe and analyze an efficient algorithm that either rounds A in this fashion, or reports correctly that no such rounding is possible.

1. For any integer k, the problem k-COLOR asks whether the vertices of a given graph G can be colored using at most k colors so that neighboring vertices does not have the same color.

   (a) Prove that k-COLOR is NP-hard, for every integer $k \geq 3$.

   (b) Now fix an integer $k \geq 3$. Suppose you are given a magic black box that can determine **in polynomial time** whether an arbitrary graph is k-colorable; the box returns TRUE if the given graph is k-colorable and FALSE otherwise. The input to the magic black box is a graph. Just a graph. Vertices and edges. Nothing else.

   Describe and analyze a **polynomial-time** algorithm that either computes a proper k-coloring of a given graph G or correctly reports that no such coloring exists, using this magic black box as a subroutine.

2. A boolean formula is in *conjunctive normal form* (or *CNF*) if it consists of a *conjuction* (AND) or several *terms*, each of which is the disjunction (OR) of one or more literals. For example, the formula

$$(\overline{x} \vee y \vee \overline{z}) \wedge (y \vee z) \wedge (x \vee \overline{y} \vee \overline{z})$$

   is in conjunctive normal form. The problem ***CNF-SAT*** asks whether a boolean formula in conjunctive normal form is satisfiable. 3SAT is the special case of CNF-SAT where every clause in the input formula must have exactly three literals; it follows immediately that CNF-SAT is NP-hard.

   Symmetrically, a boolean formula is in *disjunctive normal form* (or *DNF*) if it consists of a *disjunction* (OR) or several *terms*, each of which is the conjunction (AND) of one or more literals. For example, the formula

$$(\overline{x} \wedge y \wedge \overline{z}) \vee (y \wedge z) \vee (x \wedge \overline{y} \wedge \overline{z})$$

   is in disjunctive normal form. The problem DNF-SAT asks whether a boolean formula in disjunctive normal form is satisfiable.

   (a) Describe a polynomial-time algorithm to solve DNF-SAT.

   (b) Describe a reduction from CNF-SAT to DNF-SAT.

   (c) Why do parts (a) and (b) not imply that P=NP?

3. The 42-PARTITION problem asks whether a given set S of n positive integers can be partitioned into subsets A and B (meaning $A \cup B = S$ and $A \cap B = \emptyset$) such that

$$\sum_{a \in A} a = 42 \sum_{b \in B} b$$

   For example, we can 42-partition the set $\{1, 2, 34, 40, 52\}$ into $A = \{34, 40, 52\}$ and $B = \{1, 2\}$, since $\sum A = 126 = 42 \cdot 3$ and $\sum B = 3$. But the set $\{4, 8, 15, 16, 23, 42\}$ cannot be 42-partitioned.

   (a) Prove that 42-PARTITION is NP-hard.

   (b) Let M denote the largest integer in the input set S. Describe an algorithm to solve 42-PARTITION in time polynomial in n and M. For example, your algorithm should return TRUE when $S = \{1, 2, 34, 40, 52\}$ and FALSE when $S = \{4, 8, 15, 16, 23, 42\}$.

   (c) Why do parts (a) and (b) not imply that P=NP?

CS 473: Undergraduate Algorithms, Fall 2013

Headbanging 0: Induction!

August 28 and 29

---

1. Prove that any non-negative integer can be represented as the sum of distinct powers of 2. ("Write it in binary" is not a proof; it's just a restatement of what you have to prove.)

2. Prove that every integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example:

$$42 = 3^4 - 3^3 - 3^2 - 3^1 \qquad 25 = 3^3 - 3^1 + 3^0 \qquad 17 = 3^3 - 3^2 - 3^0$$

3. Recall that a ***full binary tree*** is either an isolated *leaf*, or an *internal node* with a left subtree and a right subtree, each of which is a full binary tree. Equivalently, a binary tree is ***full*** if every internal node has exactly two children. Give at least *three different* proofs of the following fact: *In every full binary tree, the number of leaves is exactly one more than the number of internal nodes.*

Take-home points:

- Induction is recursion. Recursion is induction.
- All induction is strong/structural induction. There is absolutely no point in using a *weak* induction hypothesis. None. Ever.
- To prove that all snarks are boojums, start with an *arbitrary* snark and remove some tentacles. Do not start with a smaller snark and try to add tentacles. Snarks don't like that.
- Every induction proof requires an exhaustive case analysis. Write down the cases. Make sure they're exhaustive.
- Do the most general cases first. Whatever is left over are the base cases.
- The empty set is the best base case.

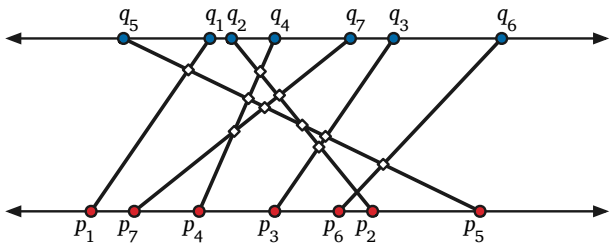
Khelm is Warsaw. Warsaw is Khelm. Khelm is Warsaw. Zay gezunt!
Warsaw is Khelm. Khelm is Warsaw. Warsaw is Khelm. For gezunt!
— Golem (feat. Amanda Palmer), "Warsaw is Khelm", *Fresh Off Boat* (2006)

1. An *inversion* in an array $A[1..n]$ is a pair of indices (i, j) such that $i < j$ and $A[i] > A[j]$. The number of inversions in an n-element array is between 0 (if the array is sorted) and $\binom{n}{2}$ (if the array is sorted backward).

   Describe and analyze a divide-and-conquer algorithm to count the number of inversions in an n-element array in $O(n \log n)$ time. Assume all the elements of the input array are distinct.

2. Suppose you are given two sets of n points, one set $\{p_1, p_2, \ldots, p_n\}$ on the line $y = 0$ and the other set $\{q_1, q_2, \ldots, q_n\}$ on the line $y = 1$. Create a set of n line segments by connect each point p_i to the corresponding point q_i. Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect, in $O(n \log n)$ time. *[Hint: Use your solution to problem 1.]*

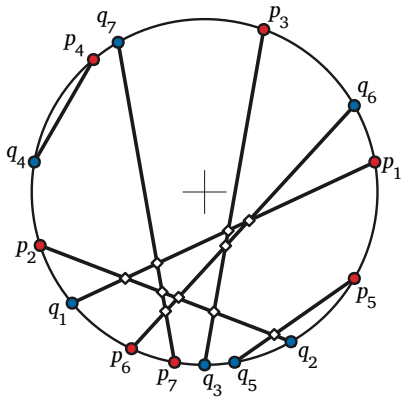
   Assume a reasonable representation for the input points, and assume the x-coordinates of the input points are distinct. For example, for the input shown below, your algorithm should return the number 10.



Ten intersecting pairs of segments with endpoints on parallel lines.

3. Now suppose you are given two sets $\{p_1, p_2, \ldots, p_n\}$ and $\{q_1, q_2, \ldots, q_n\}$ of n points *on the unit circle*. Connect each point p_i to the corresponding point q_i. Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect in $O(n \log^2 n)$ time. *[Hint: Use your solution to problem 2.]*

   Assume a reasonable representation for the input points, and assume all input points are distinct. For example, for the input shown below, your algorithm should return the number 10.



Ten intersecting pairs of segments with endpoints on a circle.

4. **To think about later:** Solve problem 3 in $O(n \log n)$ time.

1. A *longest common subsequence* of a set of strings $\{A_i\}$ is a longest string that is a subsequence of A_i for each i. For example, `alrit` is a longest common subsequence of strings

   `algorithm` and `altruistic`.

   Given two strings $A[1..n]$ and $B[1..n]$, describe and analyze a dynamic programming algorithm that computes the length of a longest common subsequence of the two strings in $O(n^2)$ time.

2. Describe and analyze a dynamic programming algorithm that computes the length of a longest common subsequence of three strings $A[1..n]$, $B[1..n]$, and $C[1..n]$ in $O(n^3)$ time. *[Hint: Try **not** to use your solution to problem 1 directly.]*

3. A *lucky-10 number* is a string $D[1..n]$ of digits from 1 to 9 (no zeros), such that the i-th digit and the last i-th digit sum up to 10; in another words, $D[i] + D[n-i+1] = 10$ for all i. For example,

   `3141592648159697` and `11599`

   are both lucky-10 numbers. Given a string of digits $D[1..n]$, describe and analyze a dynamic programming algorithm that computes the length of a longest lucky-10 subsequence of the string. *[Hint: Try to use your solution to problem 1 **directly**.]*

4. **To think about later:** Can you solve problem 1 in $O(n)$ space?

1. A ***vertex cover*** of a graph is a subset S of the vertices such that every vertex v either belongs to S or has a neighbor in S. In other words, the vertices in S cover all the edges. Finding the minimum size of a vertex cover is NP-hard, but in trees it can be found using dynamic programming.

   Given a tree T and non-negative weight $w(v)$ for each vertex v, describe an algorithm computing the minimum weight of a vertex cover of T.

2. Suppose you are given an unparenthesized mathematical expression containing n numbers, where the only operators are $+$ and $-$; for example:

$$1+3-2-5+1-6+7$$

   You can change the value of the expression by adding parentheses in different positions. For example:

$$1+3-2-5+1-6+7=-1$$
$$(1+3-(2-5))+(1-6)+7=9$$
$$(1+(3-2))-(5+1)-(6+7)=-17$$

   Design an algorithm that, given a list of integers separated by $+$ and $-$ signs, determines the maximum possible value the expression can take by adding parentheses.

   You can only insert parentheses immediately before and immediately after numbers; in particular, you are not allowed to insert implicit multiplication as in $1+3(-2)(-5)+1-6+7=33$.

3. Fix an arbitrary sequence $c_1 < c_2 < \cdots < c_k$ of coin values, all in cents. We have an infinite number of coins of each denomination. Describe a dynamic programming algorithm to determine, given an arbitrary non-negative integer x, the least number of coins whose total value is x. For simplicity, you may assume that $c_1 = 1$.

   To think about later after learning "greedy algorithms":

   (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

   (b) Suppose that the available coins have the values $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

   (c) Describe a set of 4 coin values for which the greedy algorithm does ***not*** yield an optimal solution.

Note: All the questions in this session are taken from past CS473 midterms.

1. (Fall 2006) **Multiple Choice:** Each of the questions on this page has one of the following five answers: For each question, write the letter that corresponds to your answer.

A: $\Theta(1)$	B: $\Theta(\log n)$	C: $\Theta(n)$	D: $\Theta(n \log n)$	E: $\Theta(n)$

   (a) What is $\frac{5}{n} + \frac{n}{5}$?

   (b) What is $\sum_{i=1}^{n} \frac{n}{i}$?

   (c) What is $\sum_{i=1}^{n} \frac{i}{n}$?

   (d) How many bits are required to represent the nth Fibonacci number in binary?

   (e) What is the solution to the recurrence $T(n) = 2T(n/4) + \Theta(n)$?

   (f) What is the solution to the recurrence $T(n) = 16T(n/4) + \Theta(n)$?

   (g) What is the solution to the recurrence $T(n) = T(n-1) + \frac{1}{n^2}$?

   (h) What is the worst-case time to search for an item in a binary search tree?

   (i) What is the worst-case running time of quicksort?

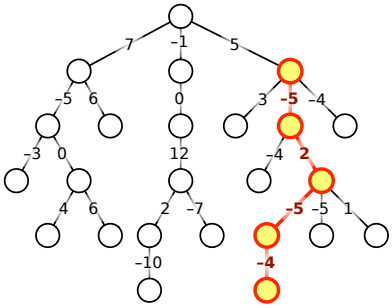
   (j) What is the running time of the fastest possible algorithm to solve Sudoku puzzles? A Sudoku puzzle consists of a 9×9 grid of squares, partitioned into nine 3×3 sub-grids; some of the squares contain digits between 1 and 9. The goal of the puzzle is to enter digits into the blank squares, so that each digit between 1 and 9 appears exactly once in each row, each column, and each 3×3 sub-grid. The initial conditions guarantee that the solution is unique.

2							4	
	7		5					
				1		9		
6		4			2			
	8						5	
			9			3		7
		1		4				
					3		8	
	5							6

A Sudoku puzzle. **Don't try to solve this during the exam!**

2. (Spring 2010) Let T be a rooted tree with integer weights on its edges, which could be positive, negative, or zero. The weight of a path in T is the sum of the weights of its edges. Describe and analyze an algorithm to compute the minimum weight of any path from a node in T down to one of its descendants. It is not necessary to compute the actual minimum-weight path; just its weight. For example, given the tree shown below, your algorithm should return the number -12.

3. (Fall 2006) Suppose you are given an array $A[1..n]$ of n distinct integers, sorted in increasing order. Describe and analyze an algorithm to determine whether there is an index i such that $A[i] = i$, in $o(n)$ time. [Hint: Yes, that's little-oh of n. What can you say about the sequence $A[i] - i$?]



The minimum-weight downward path in this tree has weight −12.

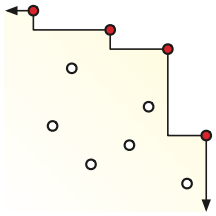
4. (Spring 2010 and Spring 2004) Describe and analyze efficient algorithms to solve the following problems:

   (a) Given a set of n integers, does it contain two elements a, b such that $a + b = 0$?

   (b) Given a set of n integers, does it contain three elements a, b, c such that $a + b = c$?

1. What is the *exact* expected number of leaves in a treap with n nodes?

2. Recall question 5 from Midterm 1:

   > Suppose you are given a set P of n points in the plane. A point $p \in P$ is *maximal* in P if no other point in P is both above and to the right of P. Intuitively, the maximal points define a "staircase" with all the other points of P below it.
   >
   > 
   >
   > A set of ten points, four of which are maximal.
   >
   > Describe and analyze an algorithm to compute the number of maximal points in P in $O(n \log n)$ time. For example, given the ten points shown above, your algorithm should return the integer 4.

   Suppose the points in P are generated independently and uniformly at random in the unit square $[0,1]^2$. What is the *exact* expected number of maximal points in P?

3. Suppose you want to write an app for your new Pebble smart watch that monitors the global Twitter stream and selects a small sample of *random* tweets. You will not know when the stream ends until your app attempts to read the next tweet and receives the error message FAILWHALE. The Pebble has only a small amount of memory, far too little to store the entire stream.

   (a) Describe an algorithm that, as soon as the stream ends, returns a single tweet chosen uniformly at random from the stream. Prove your algorithm is correct. (You may assume that the stream contains at least one tweet.)

   (b) Now fix an arbitrary positive integer k. Describe an algorithm that picks k tweets uniformly at random from the stream. Prove your algorithm is correct. (You may assume that the stream contains at least k tweets.)

Recall the following elementary data structures from CS 225.

- A *stack* supports the following operations.
  - PUSH pushes an element on top of the stack.
  - POP removes the top element from a stack.
  - ISEMPTY checks if a stack is empty.
- A *queue* supports the following operations.
  - PUSH adds an element to the back of the queue.
  - PULL removes an element from the front of the queue.
  - ISEMPTY checks if a queue is empty.
- A *deque*, or double-ended queue, supports the following operations.
  - PUSH adds an element to the back of the queue.
  - PULL removes an element from the back of the queue.
  - CUT adds an element from the front of the queue.
  - POP removes an element from the front of the queue.
  - ISEMPTY checks if a queue is empty.

Suppose you have a stack implementation that supports all stack operations in constant time.

1. Describe how to implement a queue using two stacks and $O(1)$ additional memory, so that each queue operation runs in $O(1)$ amortized time.
2. Describe how to implement a deque using three stacks and $O(1)$ additional memory, so that each deque operation runs in $O(1)$ amortized time.

1. Let P be a set of n points in the plane. Recall from the midterm that the *staircase* of P is the set of all points in the plane that have at least one point in P both above and to the right.

   (a) Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm ABOVE?(x, y) that returns TRUE if the point (x, y) is above the staircase, or FALSE otherwise. Your data structure should use $O(n)$ space, and your ABOVE? algorithm should run in $O(\log n)$ time.

   (b) Describe and analyze a data structure that maintains the staircase of a set of points as new points are inserted. Specifically, your data structure should support a function INSERT(x, y) that adds the point (x, y) to the underlying point set and returns TRUE or FALSE to indicate whether the staircase of the set has changed. Your data structure should use $O(n)$ space, and your INSERT algorithm should run in $O(\log n)$ amortized time.

2. An *ordered stack* is a data structure that stores a sequence of items and supports the following operations.

   - ORDEREDPUSH(x) removes all items smaller than x from the beginning of the sequence and then adds x to the beginning of the sequence.
   - POP deletes and returns the first item in the sequence (or NULL if the sequence is empty).

   Suppose we implement an ordered stack with a simple linked list, using the obvious ORDEREDPUSH and POP algorithms. Prove that if we start with an empty data structure, the amortized cost of each ORDEREDPUSH or POP operation is $O(1)$.

3. Consider the following solution for the union-find problem, called *union-by-weight*. Each set leader $\bar{x}$ stores the number of elements of its set in the field weight$(\bar{x})$. Whenever we UNION two sets, the leader of the *smaller* set becomes a new child of the leader of the *larger* set (breaking ties arbitrarily).

```
MAKESET(x)
  parent(x) ← x
  weight(x) ← 1
```

```
FIND(x)
  while x ≠ parent(x)
      x ← parent(x)
  return x
```

```
UNION(x, y)
  x̄ ← FIND(x)
  ȳ ← FIND(y)
  if weight(x̄) > weight(ȳ)
      parent(ȳ) ← x̄
      weight(x̄) ← weight(x̄) + weight(ȳ)
  else
      parent(x̄) ← ȳ
      weight(x̄) ← weight(x̄) + weight(ȳ)
```

   Prove that if we use union-by-weight, the *worst-case* running time of FIND(x) is $O(\log n)$, where n is the cardinality of the set containing x.

1. Let G be an undirected graph.

   (a) Suppose we start with two coins on two arbitrarily chosen nodes. At every step, each coin must move to an adjacent node. Describe an algorithm to compute the minimum number of steps to reach a configuration that two coins are on the same node.

   (b) Now suppose there are three coins, numbered 0, 1, and 2. Again we start with an arbitrary coin placement with all three coins facing up. At each step, we move each coin to an adjacent node at each step. Moreover, for every integer i, we flip coin i mod 3 at the ith step. Describe an algorithm to compute the minimum number of steps to reach a configuration that all three coins are on the same node and all facing up. What is the running time of your algorithm?

2. Let G be a directed acyclic graph with a unique source s and a unique sink t.

   (a) A *Hamiltonian path* in G is a directed path in G that contains every vertex in G. Describe an algorithm to determine whether G has a Hamiltonian path.

   (b) Suppose several nodes in G are marked to be *important*; also an integer k is given. Design an algorithm which computes all the nodes that can reach t through at least k important nodes.

   (c) Suppose the edges in G have real weights. Describe an algorithm to find a path from s to t with maximum total weight.

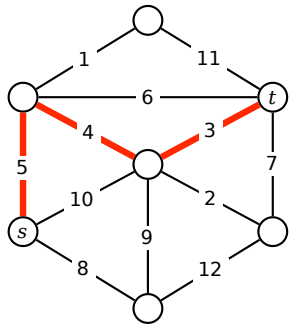
   (d) Suppose the vertices of G have labels from a fixed finite alphabet, and let $A[1..\ell]$ be a string over the same alphabet. Any directed path in G has a label, which is obtained by concatenating the labels of its vertices. Describe an algorithm to find the longest path in G whose labels are a subsequence of A.

3. Let G be a directed graph with a special source that has an edge to each other node in graph, and denote $scc(G)$ as the strong component graph of G. Let S and S' be two strongly connected components in G with $S \to S'$ an arc in $scc(G)$. (That is, if there is an arc between node $u \in S$ and $v \in S'$, then it must be $u \to v$.) Consider a fixed depth-first search performed on G starting at s; we define post$(\cdot)$ as the post-order numbering of the search.

   (a) Prove or disprove that we have $\text{post}(u) > \text{post}(u')$ for any $u \in S$ and $u' \in S'$.

   (b) Prove or disprove that we have $\max_{u \in S} \text{post}(u) > \max_{u' \in S'} \text{post}(u')$.

1. Consider a path between two vertices s and t in an undirected weighted graph G. The *bottleneck length* of this path is the maximum weight of any edge in the path. The *bottleneck distance* between s and t is the minimum bottleneck length of any path from s to t. (If there are no paths from s to t, the bottleneck distance between s and t is ∞)

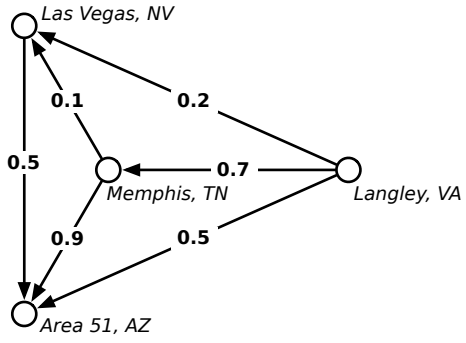


Describe an algorithm to compute the bottleneck distance between *every* pair of vertices in an arbitrary undirected weighted graph. Assume that no two edges have the same weight.

2. Let G be a directed graph with (possibly negative) edge weights, and let s be an arbitrary vertex of G. Suppose for each vertex v we are given a real number $d(v)$. Describe and analyze an algorithm to determine whether the numbers $d(v)$ on vertices are the shortest path distances from s to each vertex v. Do not assume that the graph G has no negative cycles.

3. Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road *won't* be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What rout should hs take so that he has the least chance of being abducted?

   More formally, you are given a directed graph G, possibly with cycles, where every edge e has an independent safety probability $p(e)$. The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex s to a given target vertex t.



For example, with the probabilities shown above, if Mulder tries to drive directly from Langley to Area 51, he has a 50% chance of getting there without being abducted. If he stops in Memphis, he has a $0.7 \times 0.9 = 63\%$ chance of arriving safely. If he stops first in Memphis and then in Las Vegas, he has a $1 - 0.7 \times 0.1 \times 0.5 = 96.5\%$ chance of begin abducted![1]

[1] That's how they got Elvis, you know.

Almost all these review problems from from past midterms.

1. [Fall 2002, Spring 2004] Suppose we want to maintain a set X of numbers, under the following operations:

   - INSERT(x): Add x to the set (if it isn't already there).
   - PRINT&DELETEBETWEEN(a, b): Print every element $x \in X$ such that $a \le x \le b$, in order from smallest to largest, and then delete those elements from X.

   For example, if the current set is $\{1, 5, 3, 4, 8\}$, then PRINT&DELETEBETWEEN$(4, 6)$ prints the numbers 4 and 5 and changes the set to $\{1, 3, 8\}$.

   Describe and analyze a data structure that supports these two operations, each in $O(\log n)$ *amortized* time, where n is the maximum number of elements in X.

2. [Spring 2004] Consider a random walk on a path with vertices numbered $1, 2, \ldots, n$ from left to right. We start at vertex 1. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex n.

   Prove that the probability that the walk ends by falling off the *left* end of the path is exactly $n/(n+1)$. [Hint: Set up a recurrence and verify that $n/(n+1)$ satisfies it.]

3. [Fall 2006] ***Prove or disprove*** each of the following statements.

   (a) Let G be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of G includes the lightest edge in every cycle in G.

   (b) Let G be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of G excludes the heaviest edge in every cycle in G.

4. [Fall 2012] Let $G = (V, E)$ be a connected undirected graph. For any vertices u and v, let $d_G(u, v)$ denote the length of the shortest path in G from u to v. For any sets of vertices A and B, let $d_G(A, B)$ denote the length of the shortest path in G from any vertex in A to any vertex in B:

   $$d_G(A, B) = \min_{u \in A} \min_{v \in B} d_G(u, v).$$

   Describe and analyze a fast algorithm to compute $d_G(A, B)$, given the graph G and subsets A and B as input. You do not need to prove that your algorithm is correct.

5. Let G and H be directed acyclic graphs, whose vertices have labels from some fixed alphabet, and let $A[1..\ell]$ be a string over the same alphabet. Any directed path in G has a label, which is a string obtained by concatenating the labels of its vertices.

   (a) Describe an algorithm to find the longest string that is both a label of a directed path in G and the label of a directed path in H.

   (b) Describe an algorithm to find the longest string that is both a *subsequence* of the label of a directed path in G and *subsequence* of the label of a directed path in H.

1. The Island of Sodor is home to a large number of towns and villages, connected by an extensive rail network. Recently, several cases of a deadly contagious disease (either swine flu or zombies; reports are unclear) have been reported in the village of Ffarquhar. The controller of the Sodor railway plans to close down certain railway stations to prevent the disease from spreading to Tidmouth, his home town. No trains can pass through a closed station. To minimize expense (and public notice), he wants to close down as few stations as possible. However, he cannot close the Ffarquhar station, because that would expose him to the disease, and he cannot close the Tidmouth station, because then he couldn't visit his favorite pub.

   Describe and analyze an algorithm to find the minimum number of stations that must be closed to block all rail travel from Ffarquhar to Tidmouth. The Sodor rail network is represented by an undirected graph, with a vertex for each station and an edge for each rail connection between two stations. Two special vertices f and t represent the stations in Ffarquhar and Tidmouth.

2. Given an undirected graph $G = (V, E)$, with three vertices u, v, and w, describe and analyze an algorithm to determine whether there is a path from u to w that passes through v.

3. Suppose you have already computed a maximum flow f^* in a flow network G with *integer* edge capacities.

   (a) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is increased by 1.

   (b) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is decreased by 1.

   Both algorithms should be significantly faster than recomputing the maximum flow from scratch.

1. An *American graph* is a directed graph with each vertex colored *red*, *white*, or *blue*. An *American Hamiltonian path* is a Hamiltonian path that cycles between red, white, and blue vertices; that is, every edge goes from red to white, or white to blue, or blue to red. The AMERICANHAMILTONIANPATH problem asks whether there is an American Hamiltonian path in an American graph.

   (a) Prove that AMERICANHAMILTONIANPATH is NP-complete by reducing from HAMILTONIANPATH.

   (b) In the opposite direction, reduce AMERICANHAMILTONIANPATH to HAMILTONIANPATH.

2. Given a graph G, the DEG17SPANNINGTREE problem asks whether G has a spanning tree in which each vertex of the spanning tree has degree at most 17. Prove that DEG17SPANNINGTREE is NP-complete.

3. Two graphs are *isomorphic* if one can be transformed into the other by relabeling the vertices. Consider the following related decision problems:

   - GRAPHISOMORPHISM: Given two graphs G and H, determine whether G and H are isomorphic.
   - EVENGRAPHISOMORPHISM: Given two graphs G and H, such that every vertex of G and H have even degree, determine whether G and H are isomorphic.
   - SUBGRAPHISOMORPHISM: Given two graphs G and H, determine whether G is isomorphic to a subgraph of H.

   (a) Describe a polynomial time reduction from GRAPHISOMORPHISM to EVENGRAPHISOMORPHISM.

   (b) Describe a polynomial time reduction from GRAPHISOMORPHISM to SUBGRAPHISOMORPHISM.

1. Prove that the following problem is NP-hard.

   SETCOVER: Given a collection of sets $\{S_1, \ldots, S_m\}$, find the smallest sub-collection of S_i's that contains all the elements of $\bigcup_i S_i$.

2. Given an undirected graph G and a subset of vertices S, a *Steiner tree* of S in G is a subtree of G that contains every vertex in S. If S contains every vertex of G, a Steiner tree is just a spanning tree; if S contains exactly two vertices, any path between them is a Steiner tree.

   Given a graph G, a vertex subset S, and an integer k, the *Steiner tree problem* requires us to decide whether there is a Steiner tree of S in G with at most k edges. Prove that the Steiner tree problem is NP-hard. *[Hint: Reduce from* VERTEXCOVER*, or* SETCOVER*, or* 3SAT*.]*

3. Let G be a directed graph whose edges are colored red and white. A *Canadian Hamiltonian path* is a Hamiltonian path whose edges are alternately red and white. The CANADIANHAMILTONIANPATH problem ask us to find a Canadian Hamiltonian path in a graph G. (Two weeks ago we looked for Hamiltonian paths that cycled through colors on the *vertices* instead of edges.)

   (a) Prove that CANADIANHAMILTONIANPATH is NP-Complete.

   (b) Reduce CANADIANHAMILTONIANPATH to HAMILTONIANPATH.

This exam lasts 120 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheet with your answers.

1. Each of these ten questions has one of the following five answers:

   A: $\Theta(1)$ B: $\Theta(\log n)$ C: $\Theta(n)$ D: $\Theta(n \log n)$ E: $\Theta(n^2)$

   (a) What is $\dfrac{n^5 - 3n^3 - 5n + 4}{4n^3 - 2n^2 + n - 7}$?

   (b) What is $\displaystyle\sum_{i=1}^{n} i$?

   (c) What is $\displaystyle\sum_{i=1}^{n} \sqrt{\frac{n}{i}}$?

   (d) How many bits are required to write the integer n^{10} in binary?

   (e) What is the solution to the recurrence $E(n) = E(n/2) + E(n/4) + E(n/8) + 16n$?

   (f) What is the solution to the recurrence $F(n) = 6F(n/6) + 6n$?

   (g) What is the solution to the recurrence $G(n) = 9G(\lceil n/3 \rceil + 1) + n$?

   (h) The *total path length* of a binary tree is the sum of the depths of all nodes. What is the total path length of an n-node binary tree in the worst case?

   (i) Consider the following recursive function, defined in terms of a fixed array $X[1..n]$:

$$WTF(i,j) = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \\ \max\left\{ \begin{array}{c} 2 \cdot \left[X[i] \neq X[j] \right] + WTF(i+1, j-1) \\ 1 + WTF(i+1, j) \\ 1 + WTF(i, j-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

   How long does it take to compute $WTF(1, n)$ using dynamic programming?

   (j) Voyager 1 recently became the first made-made object to reach interstellar space. Currently the spacecraft is about 18 billion kilometers (roughly 60,000 light seconds) from Earth, traveling outward at approximately 17 kilometers per second (approximately 1/18000 of the speed of light). Voyager carries a golden record containing over 100 digital images abd approximately one hour of sound recordings. In digital form, the recording would require about 1 gigabyte. Voyager can transmit data back to Earth at approximately 1400 bits per second. Suppose the engineers at JPL sent instructions to Voyager 1 to send the complete contents of the Golden Record back to Earth; how many seconds would they have to wait to receive the entire record?

2. You are a visitor at a political convention (or perhaps a faculty meeting) with n delegates; each delegate is a member of exactly one political party. It is impossible to tell which political party any delegate belongs to; in particular, you will be summarily ejected from the convention if you ask. However, you can determine whether any pair of delegates belong to the *same* party or not simply by introducing them to each other—members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.

   Suppose more than half of the delegates belong to the same political party. Describe and analyze an efficient algorithm that identifies all members of this majority party.

3. Recall that a *tree* is a connected undirected graph with no cycles. ***Prove*** that in any tree, the number of nodes is exactly one more than the number of edges.

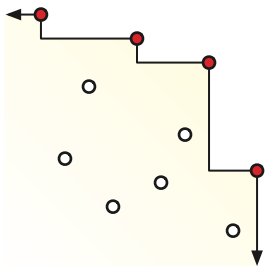
4. Next spring break, you and your friends decide to take a road trip, but before you leave, you decide to figure out *exactly* how much money to bring for gasoline. Suppose you compile a list of all gas stations along your planned route, containing the following information:

   - A sorted array $Dist[0..n]$, where $Dist[0] = 0$ and $Dist[i]$ is the number of miles from the beginning of your route to the ith gas station. Your route ends at the nth gas station.
   - A second array $Price[1..n]$, where $Price[i]$ is the price of one gallon of gasoline at the ith gas station. (Unlike in real life, these prices do not change over time.)

   You start the trip with a full tank of gas. Whenever you buy gas, you must completely fill your tank. Your car holds exactly 10 gallons of gas and travels exactly 25 miles per gallon; thus, starting with a full tank, you can travel exactly 250 miles before your car dies. Finally, $Dist[i+1] < Dist[i]+250$ for every index i, so the trip is possible.

   Describe and analyze an algorithm to determine the minimum amount of money you must spend on gasoline to guarantee that you can drive the entire route.

5. Suppose you are given a set P of n points in the plane. A point $p \in P$ is *maximal* in P if no other point in P is both above and to the right of P. Intuitively, the maximal points define a "staircase" with all the other points of P below it.



A set of ten points, four of which are maximal.

   Describe and analyze an algorithm to compute the number of maximal points in P in $O(n \log n)$ time. For example, given the ten points shown above, your algorithm should return the integer 4.

This exam lasts 120 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheet with your answers.

1. Each of these ten questions has one of the following five answers:

   A: $\Theta(1)$ B: $\Theta(\log n)$ C: $\Theta(n)$ D: $\Theta(n \log n)$ E: $\Theta(n^2)$

   (a) What is $\dfrac{n^5 - 3n^3 - 5n + 4}{3n^4 - 2n^2 + n - 7}$?

   (b) What is $\displaystyle\sum_{i=1}^{n} \frac{n}{i}$?

   (c) What is $\displaystyle\sum_{i=1}^{n} \frac{i}{n}$?

   (d) How many bits are required to write the integer 10^n in binary?

   (e) What is the solution to the recurrence $E(n) = E(n/3) + E(n/4) + E(n/5) + n/6$?

   (f) What is the solution to the recurrence $F(n) = 16F(n/4 + 2) + n$?

   (g) What is the solution to the recurrence $G(n) = G(n/2) + 2G(n/4) + n$?

   (h) The *total path length* of a binary tree is the sum of the depths of all nodes. What is the total path length of a perfectly balanced n-node binary tree?

   (i) Consider the following recursive function, defined in terms of two fixed arrays $A[1..n]$ and $B[1..n]$:

   $$WTF(i,j) = \begin{cases} 0 & \text{if } i > j \\ \max\left\{ \begin{array}{r} (A[i] - B[j])^2 + WTF(i+1, j-1) \\ A[i]^2 + WTF(i+1, j) \\ B[i]^2 + WTF(i, j-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

   How long does it take to compute $WTF(1, n)$ using dynamic programming?

   (j) Voyager 1 recently became the first made-made object to reach interstellar space. Currently the spacecraft is about 18 billion kilometers (roughly 60,000 light seconds) from Earth, traveling outward at approximately 17 kilometers per second (approximately 1/18000 of the speed of light). Voyager carries a golden record containing over 100 digital images and approximately one hour of sound recordings. In digital form, the recording would require about 1 gigabyte. Voyager can transmit data back to Earth at approximately 1400 bits per second. Suppose the engineers at JPL sent instructions to Voyager 1 to send the complete contents of the Golden Record back to Earth; how many seconds would they have to wait to receive the entire record?

2. Suppose we are given an array $A[0..n+1]$ with fencepost values $A[0] = A[n+1] = -\infty$. We say that an element $A[x]$ is a *local maximum* if it is less than or equal to its neighbors, or more formally, if $A[x-1] \le A[x]$ and $A[x] \ge A[x+1]$. For example, there are five local maxima in the following array:

$-\infty$	6	⑦	2	1	3	⑦	5	4	⑨	⑨	3	4	⑧	6	$-\infty$

We can obviously find a local maximum in $O(n)$ time by scanning through the array. Describe and analyze an algorithm that returns the index of one local maximum in $O(\log n)$ time. *[Hint: With the given boundary conditions, the array **must** have at least one local maximum. Why?]*

3. ***Prove*** that in any binary tree, the number of nodes with no children (leaves) is exactly one more than the number of nodes with two children. (Remember that a binary tree can have nodes with only one child.)

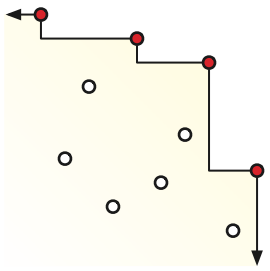
4. A string x is a *supersequence* of a string y if we can obtain x by inserting zero or more letters into y, or equivalently, if y is a subsequence of x. For example, the string **D**YN**AM**IC**P**ROG**RA**MMIN**G** is a supersequence of the string DAMPRAG.

   A *palindrome* is any string that is exactly the same as its reversal, like I, DAD, HANNAH, AIBOHPHOBIA (fear of palindromes), or the empty string.

   Describe and analyze an algorithm to find the length of the shortest supersequence of a given string that is also a palindrome.

   For example, the 11-letter string E**HE**C**ADAC**E**HE** is the shortest palindrome supersequence of HEADACHE, so given the string HEADACHE as input, your algorithm should output the number 11.

5. Suppose you are given a set P of n points in the plane. A point $p \in P$ is *maximal* in P if no other point in P is both above and to the right of P. Intuitively, the maximal points define a "staircase" with all the other points of P below it.



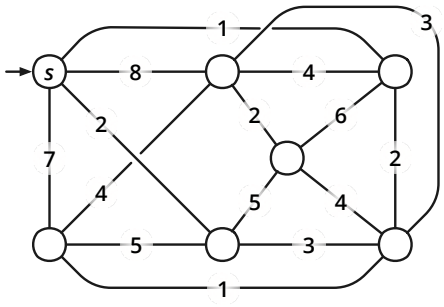
A set of ten points, four of which are maximal.

   Describe and analyze an algorithm to compute the number of maximal points in P in $O(n \log n)$ time. For example, given the ten points shown above, your algorithm should return the integer 4.

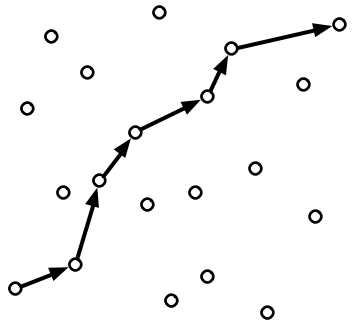
This exam lasts 120 minutes.
Write your answers in the separate answer booklet.
Please return this question sheet and your cheat sheet with your answers.

1. ***Clearly*** indicate the following spanning trees in the weighted graph pictured below. Some of these subproblems have more than one correct answer.

   (a) A depth-first spanning tree rooted at s
   (b) A breadth-first spanning tree rooted at s
   (c) A shortest-path tree rooted at s
   (d) A minimum spanning tree
   (e) A *maximum* spanning tree



2. A ***polygonal path*** is a sequence of line segments joined end-to-end; the endpoints of these line segments are called the ***vertices*** of the path. The ***length*** of a polygonal path is the sum of the lengths of its segments. A polygonal path with vertices $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ is ***monotonically increasing*** if $x_i < x_{i+1}$ and $y_i < y_{i+1}$ for every index i—informally, each vertex of the path is above and to the right of its predecessor.



A monotonically increasing polygonal path with seven vertices through a set of points

Suppose you are given a set S of n points in the plane, represented as two arrays $X[1..n]$ and $Y[1..n]$. Describe and analyze an algorithm to compute the length of the maximum-length monotonically increasing path with vertices in S. Assume you have a subroutine $\textsc{Length}(x, y, x', y')$ that returns the length of the segment from (x, y) to (x', y').

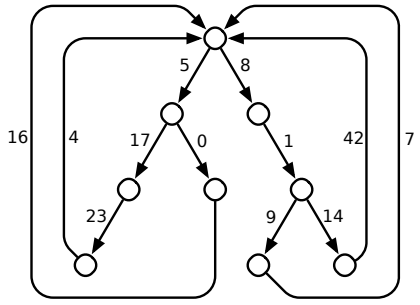
3. Suppose you are maintaining a circular array $X[0..n-1]$ of counters, each taking a value from the set $\{0,1,2\}$. The following algorithm increments one of the counters; if the counter overflows, the algorithm resets it 0 and recursively increments its two neighbors.

```
INCREMENT(i):
  X[i] ← X[i] + 1
  if X[i] = 3
      X[i] ← 0
      INCREMENT((i − 1) mod n)
      INCREMENT((i + 1) mod n)
```

   (a) Suppose $n = 5$ and $X = [2,2,2,2,2]$. What does X contain after we call INCREMENT(3)?

   (b) Suppose all counters are initially 0. ***Prove*** that INCREMENT runs in $O(1)$ amortized time.

4. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has non-negative weight.



A looped tree.

   (a) How much time would Dijkstra's algorithm require to compute the shortest path from an arbitrary vertex s to another arbitrary vertex t, in a looped tree with n vertices?

   (b) Describe and analyze a faster algorithm. Your algorithm should compute the actual shortest path, not just its length.

5. Consider the following algorithm for finding the smallest element in an unsorted array:

```
RANDOMMIN(A[1..n]):
  min ← ∞
  for i ← 1 to n in random order
      if A[i] < min
          min ← A[i]    (⋆)
  return min
```

   Assume the elements of A are all distinct.

   (a) In the worst case, how many times does RANDOMMIN execute line $(\star)$?

   (b) What is the probability that line $(\star)$ is executed during the *last* iteration of the for loop?

   (c) What is the *exact* expected number of executions of line $(\star)$?

This exam lasts 180 minutes.
Write your answers in the separate answer booklet.
Please return this question handout and your cheat sheets with your answers.

1. Suppose you are given a sorted array of n distinct numbers that has been *rotated* k steps, for some ***unknown*** integer k between 1 and $n-1$. That is, you are given an array $A[1..n]$ such that the prefix $A[1..k]$ is sorted in increasing order, the suffix $A[k+1..n]$ is sorted in increasing order, and $A[n] < A[1]$. For example, you might be given the following 16-element array (where $k = 10$):

9	13	16	18	19	23	28	31	37	42	−4	0	2	5	7	8

   Describe and analyze an algorithm to determine if the given array contains a given number x. For example, given the previous array and the number 17 as input, your algorithm should return FALSE. The index k is ***NOT*** part of the input.

2. You are hired as a cyclist for the Giggle Highway View project, which will provide street-level images along the entire US national highway system. As a pilot project, you are asked to ride the Giggle Highway-View Fixed-Gear Carbon-Fiber Bicycle from "the Giggleplex" in Portland, Oregon to "Giggleburg" in Williamsburg, Brooklyn, New York.

   You are a hopeless caffeine addict, but like most Giggle employees you are also a coffee snob; you only drink independently roasted organic shade-grown single-origin espresso. After each espresso shot, you can bike up to L miles before suffering a caffeine-withdrawal migraine.

   Giggle helpfully provides you with a map of the United States, in the form of an undirected graph G, whose vertices represent coffee shops that sell independently roasted organic shade-grown single-origin espresso, and whose edges represent highway connections between them. Each edge e is labeled with the length $\ell(e)$ of the corresponding stretch of highway. Naturally, there are espresso stands at both Giggle offices, represented by two specific vertices s and t in the graph G.

   (a) Describe and analyze an algorithm to determine whether it is possible to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine.

   (b) You discover that by wearing a more expensive fedora, you can increase the distance L that you can bike between espresso shots. Describe and analyze and algorithm to find the minimum value of L that allows you to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine.

3. Suppose you are given a collection of up-trees representing a partition of the set $\{1, 2, \ldots, n\}$ into subsets. **You have no idea how these trees were constructed.** You are also given an array $node[1..n]$, where $node[i]$ is a pointer to the up-tree node containing element i. Your task is to create a new array $label[1..n]$ using the following algorithm:

```
LABELEVERYTHING:
  for i ← 1 to n
      label[i] ← FIND(node[i])
```

Recall that there are two natural ways to implement FIND: simple pointer-chasing and pointer-chasing with path compression. Pseudocode for both methods is shown below.

```
FIND(x):
  while x ≠ parent(x)
      x ← parent(x)
  return x
```

Without path compression

```
FIND(x):
  if x ≠ parent(x)
      parent(x) ← FIND(parent(x))
  return parent(x)
```

With path compression

(a) What is the worst-case running time of LABELEVERYTHING if we implement FIND *without* path compression?

(b) ***Prove*** that if we implement FIND using path compression, LABELEVERYTHING runs in $O(n)$ time in the worst case.

4. Congratulations! You have successfully conquered Camelot, transforming the former battle-scarred kingdom with an anarcho-syndicalist commune, where citizens take turns to act as a sort of executive-officer-for-the-week, but with all the decisions of that officer ratified at a special bi-weekly meeting, by a simple majority in the case of purely internal affairs, but by a two-thirds majority, in the case of more major....

As a final symbolic act, you order the Round Table (surprisingly, an actual circular table) to be split into pizza-like wedges and distributed to the citizens of Camelot as trophies. Each citizen has submitted a request for an angular wedge of the table, specified by two angles—for example, Sir Robin the Brave might request the wedge from 23.17° to 42°. Each citizen will be happy if and only if they receive *precisely* the wedge that they requested. Unfortunately, some of these ranges overlap, so satisfying *all* the citizens' requests is simply impossible. Welcome to politics.

Describe and analyze an algorithm to find the maximum number of requests that can be satisfied.

5. The NSA has established several monitoring stations around the country, each one conveniently hidden in the back of a Starbucks. Each station can monitor up to 42 cell-phone towers, but can only monitor cell-phone towers within a 20-mile radius. To ensure that every cell-phone call is recorded even if some stations malfunction, the NSA requires each cell-phone tower to be monitored by at least 3 different stations.

   Suppose you know that there are n cell-phone towers and m monitoring stations, and you are given a function DISTANCE(i, j) that returns the distance between the ith tower and the jth station in $O(1)$ time. Describe and analyze an algorithm that either computes a valid assignment of cell-phone towers to monitoring stations, or reports correctly that there is no such assignment (in which case the NSA will build another Starbucks).

6. Consider the following closely related problems:

   - HAMILTONIANPATH: Given an undirected graph G, determine whether G contains a ***path*** that visits every vertex of G exactly once.
   - HAMILTONIANCYCLE: Given an undirected graph G, determine whether G contains a ***cycle*** that visits every vertex of G exactly once.

   Describe a polynomial-time reduction from HAMILTONIANPATH to HAMILTONIANCYCLE. ***Prove*** your reduction is correct. *[Hint: A polynomial-time reduction is allowed to call the black-box subroutine more than once.]*

7. An array $X[1..n]$ of distinct integers is ***wobbly*** if it alternates between increasing and decreasing: $X[i] < X[i+1]$ for every odd index i, and $X[i] > X[i+1]$ for every even index i. For example, the following 16-element array is wobbly:

12	13	0	16	13	31	5	7	−1	23	8	10	−4	37	17	42

   Describe and analyze an algorithm that permutes the elements of a given array to make the array wobbly.

You may use the following algorithms as black boxes:

RANDOM(k): Given any positive integer k, return an integer chosen independently and uniformly at random from the set $\{1, 2, \ldots, k\}$ in $O(1)$ time.

ORLINMAXFLOW(V, E, c, s, t): Given a directed graph $G = (V, E)$, a capacity function $c : E \to \mathbb{R}^+$, and vertices s and t, return a maximum (s, t)-flow in G in $O(VE)$ time. If the capacities are integral, so is the returned maximum flow.

Any other algorithm that we described in class.

You may assume the following problems are NP-hard:

CIRCUITSAT: Given a boolean circuit, are there any input values that make the circuit output TRUE?

PLANARCIRCUITSAT: Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output TRUE?

3SAT: Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

MAXINDEPENDENTSET: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MAXCLIQUE: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MINVERTEXCOVER: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MINSETCOVER: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MINHITTINGSET: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3COLOR: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

HAMILTONIANCYCLE: Given a graph G, is there a cycle in G that visits every vertex exactly once?

HAMILTONIANPATH: Given a graph G, is there a path in G that visits every vertex exactly once?

TRAVELINGSALESMAN: Given a graph G with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

STEINERTREE: Given an undirected graph G with some of the vertices marked, what is the minimum number of edges in a subtree of G that contains every marked vertex?

SUBSETSUM: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

PARTITION: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3PARTITION: Given a set X of $3n$ positive integers, can X be partitioned into n three-element subsets, all with the same sum?

DRAUGHTS: Given an $n \times n$ international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

DOGE: Such N. Many P. Wow.

This exam lasts 180 minutes.
Write your answers in the separate answer booklet.
Please return this question handout and your cheat sheets with your answers.

1. Suppose you are given a sorted array of n distinct numbers that has been *rotated* k steps, for some ***unknown*** integer k between 1 and $n-1$. That is, you are given an array $A[1..n]$ such that the prefix $A[1..k]$ is sorted in increasing order, the suffix $A[k+1..n]$ is sorted in increasing order, and $A[n] < A[1]$. Describe and analyze an algorithm to compute the unknown integer k.

   For example, given the following array as input, your algorithm should output the integer 10.

9	13	16	18	19	23	28	31	37	42	−4	0	2	5	7	8

2. You are hired as a cyclist for the Giggle Highway View project, which will provide street-level images along the entire US national highway system. As a pilot project, you are asked to ride the Giggle Highway-View Fixed-Gear Carbon-Fiber Bicycle from "the Giggleplex" in Portland, Oregon to "Giggleburg" in Williamsburg, Brooklyn, New York.

   You are a hopeless caffeine addict, but like most Giggle employees you are also a coffee snob; you only drink independently roasted organic shade-grown single-origin espresso. After each espresso shot, you can bike up to L miles before suffering a caffeine-withdrawal migraine.

   Giggle helpfully provides you with a map of the United States, in the form of an undirected graph G, whose vertices represent coffee shops that sell independently roasted organic shade-grown single-origin espresso, and whose edges represent highway connections between them. Each edge e is labeled with the length $\ell(e)$ of the corresponding stretch of highway. Naturally, there are espresso stands at both Giggle offices, represented by two specific vertices s and t in the graph G.

   (a) Describe and analyze an algorithm to determine whether it is possible to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine.

   (b) When you report to your supervisor (whom Giggle recently hired away from competitor Yippee!) that the ride is impossible, she demands to look at your map. "Oh, I see the problem; there are no *Starbucks* on this map!" As you look on in horror, she hands you an updated graph G' that includes a vertex for every Starbucks location in the United States, helpfully marked in Starbucks Green (Pantone® 3425 C).

   Describe and analyze an algorithm to find the minimum number of Starbucks locations you must visit to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine. More formally, your algorithm should find the minimum number of green vertices on any path in G' from s to t that uses only edges of length at most L.

3. Suppose you are given a collection of up-trees representing a partition of the set $\{1, 2, \ldots, n\}$ into subsets. **You have no idea how these trees were constructed.** You are also given an array $node[1..n]$, where $node[i]$ is a pointer to the up-tree node containing element i. Your task is to create a new array $label[1..n]$ using the following algorithm:

```
LABELEVERYTHING:
  for i ← 1 to n
      label[i] ← FIND(node[i])
```

Recall that there are two natural ways to implement FIND: simple pointer-chasing and pointer-chasing with path compression. Pseudocode for both methods is shown below.

```
FIND(x):
  while x ≠ parent(x)
      x ← parent(x)
  return x
```

Without path compression

```
FIND(x):
  if x ≠ parent(x)
      parent(x) ← FIND(parent(x))
  return parent(x)
```

With path compression

(a) What is the worst-case running time of LABELEVERYTHING if we implement FIND *without* path compression?

(b) ***Prove*** that if we implement FIND using path compression, LABELEVERYTHING runs in $O(n)$ time in the worst case.

4. Congratulations! You have successfully conquered Camelot, transforming the former battle-scarred kingdom with an anarcho-syndicalist commune, where citizens take turns to act as a sort of executive-officer-for-the-week, but with all the decisions of that officer ratified at a special bi-weekly meeting, by a simple majority in the case of purely internal affairs, but by a two-thirds majority, in the case of more major....

As a final symbolic act, you order the Round Table (surprisingly, an actual circular table) to be split into pizza-like wedges and distributed to the citizens of Camelot as trophies. Each citizen has submitted a request for an angular wedge of the table, specified by two angles—for example, Sir Robin the Brave might request the wedge from 17° to 42°. Each citizen will be happy if and only if they receive *precisely* the wedge that they requested. Unfortunately, some of these ranges overlap, so satisfying *all* the citizens' requests is simply impossible. Welcome to politics.

Describe and analyze an algorithm to find the maximum number of requests that can be satisfied.

5. The NSA has established several monitoring stations around the country, each one conveniently hidden in the back of a Starbucks. Each station can monitor up to 42 cell-phone towers, but can only monitor cell-phone towers within a 20-mile radius. To ensure that every cell-phone call is recorded even if some stations malfunction, the NSA requires each cell-phone tower to be monitored by at least 3 different stations.

   Suppose you know that there are n cell-phone towers and m monitoring stations, and you are given a function DISTANCE(i, j) that returns the distance between the ith tower and the jth station in $O(1)$ time. Describe and analyze an algorithm that either computes a valid assignment of cell-phone towers to monitoring stations, or reports correctly that there is no such assignment (in which case the NSA will build another Starbucks).

6. Consider the following closely related problems:

   - HAMILTONIANPATH: Given an undirected graph G, determine whether G contains a ***path*** that visits every vertex of G exactly once.
   - HAMILTONIANCYCLE: Given an undirected graph G, determine whether G contains a ***cycle*** that visits every vertex of G exactly once.

   Describe a polynomial-time reduction from HAMILTONIANCYCLE to HAMILTONIANPATH. ***Prove*** your reduction is correct. *[Hint: A polynomial-time reduction is allowed to call the black-box subroutine more than once.]*

7. An array $X[1..n]$ of distinct integers is ***wobbly*** if it alternates between increasing and decreasing: $X[i] < X[i+1]$ for every odd index i, and $X[i] > X[i+1]$ for every even index i. For example, the following 16-element array is wobbly:

12	13	0	16	13	31	5	7	−1	23	8	10	−4	37	17	42

   Describe and analyze an algorithm that permutes the elements of a given array to make the array wobbly.

You may use the following algorithms as black boxes:

RANDOM(k): Given any positive integer k, return an integer chosen independently and uniformly at random from the set $\{1, 2, \ldots, k\}$ in $O(1)$ time.

ORLINMAXFLOW(V, E, c, s, t): Given a directed graph $G = (V, E)$, a capacity function $c : E \to \mathbb{R}^+$, and vertices s and t, return a maximum (s, t)-flow in G in $O(VE)$ time. If the capacities are integral, so is the returned maximum flow.

Any other algorithm that we described in class.

You may assume the following problems are NP-hard:

CIRCUITSAT: Given a boolean circuit, are there any input values that make the circuit output TRUE?

PLANARCIRCUITSAT: Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output TRUE?

3SAT: Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

MAXINDEPENDENTSET: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MAXCLIQUE: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MINVERTEXCOVER: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MINSETCOVER: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MINHITTINGSET: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3COLOR: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

HAMILTONIANCYCLE: Given a graph G, is there a cycle in G that visits every vertex exactly once?

HAMILTONIANPATH: Given a graph G, is there a path in G that visits every vertex exactly once?

TRAVELINGSALESMAN: Given a graph G with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

STEINERTREE: Given an undirected graph G with some of the vertices marked, what is the minimum number of edges in a subtree of G that contains every marked vertex?

SUBSETSUM: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

PARTITION: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3PARTITION: Given a set X of $3n$ positive integers, can X be partitioned into n three-element subsets, all with the same sum?

DRAUGHTS: Given an $n \times n$ international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

DOGE: Such N. Many P. Wow.