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.

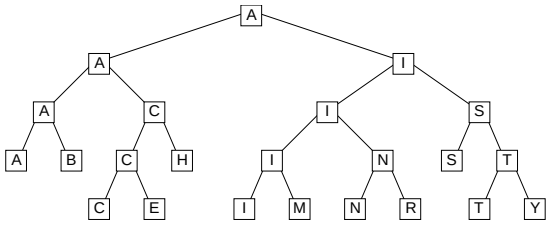
   (a) Prove that $B = R + 1$.

   (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

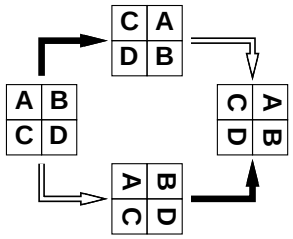
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- 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							

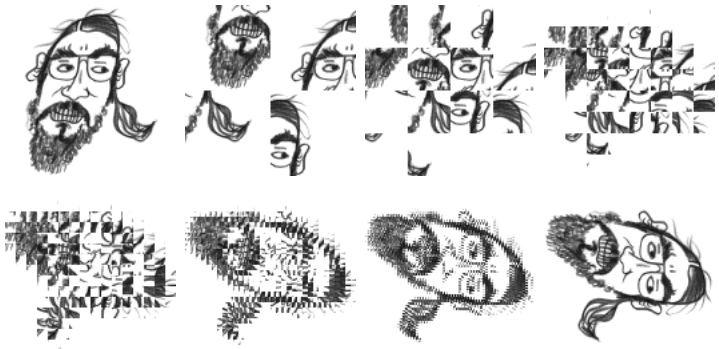
1. Some graphics hardware includes support for an operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

   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.

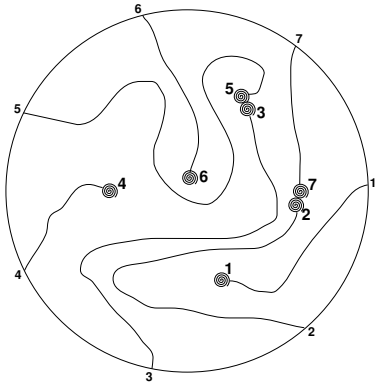
#	1	2	3	4	5	6*	Total
Score							
Grader							

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.

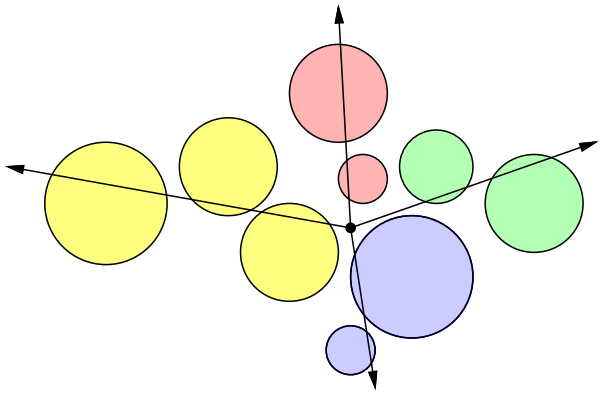
5. Consider a machine with a row of n processors numbered 1 through n. A *job* is some computational task that occupies a contiguous set of processors for some amount of time. Each processor can work on only one job at a time. Each job is represented by a pair $J_i = (n_i, t_i)$, where n_i is the number 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:

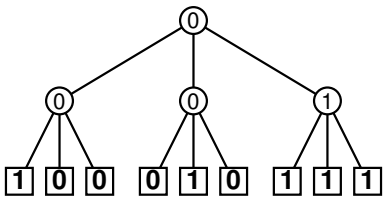
```
MELD(Q1, Q2):
  if Q1 is empty return Q2
  if Q2 is empty return Q1

  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.

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

   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.

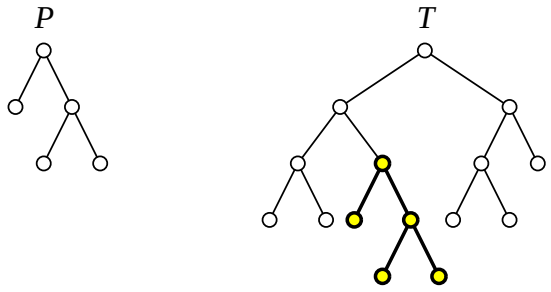
   Both proofs should be entirely self-contained. In particular, they should not use the word "tree" or any properties of trees that you saw in CS 225 or CS 273.

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.



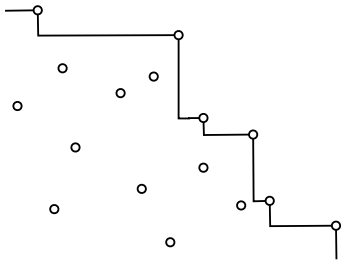
P appears exactly once as a subtree of T.

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.]*

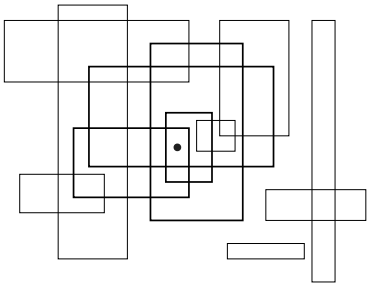
   *(c) Describe an algorithm to compute the staircase of P in $O(n \log h)$ time, where h is the number of Pareto-optimal points. *[Hint: I know of at least two different ways to do this.]*

   (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.

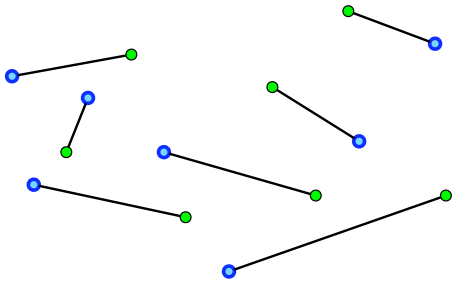


A point with depth 4 in a set of rectangles.

   (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.

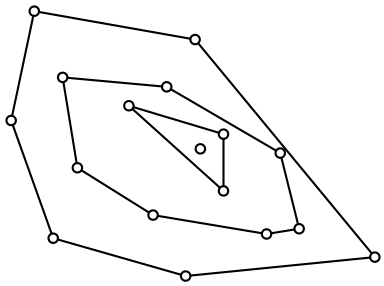
*(c) Describe and analyze an algorithm to find a line ℓ such that exactly half the ghosts and exactly half the Ghostbusters are above ℓ. (Assume n is even.)

(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

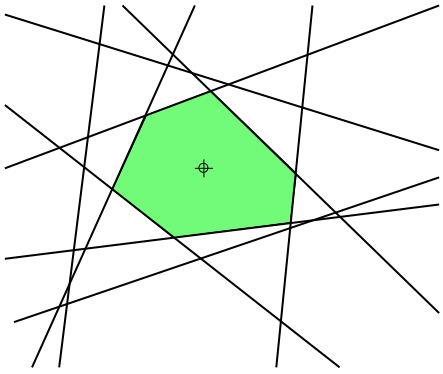
4. The *convex layers* of a point set P consist of a series of nested convex polygons. The convex layers of the empty set are empty. Otherwise, the first layer is just the convex hull of P, and the remaining layers are the convex layers of the points that are not on the convex hull of P.



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?

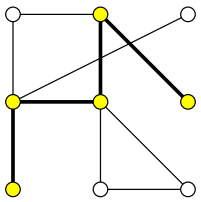
   (j) A train leaves Chicago at 8:00pm and travels south at 75 miles per hour. Another train leaves New Orleans at 1:00pm and travels north at 60 miles per hour. The conductors of both trains are playing a game of chess over the phone. After each player moves, the other player must move before his train 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 $a + b = c$?

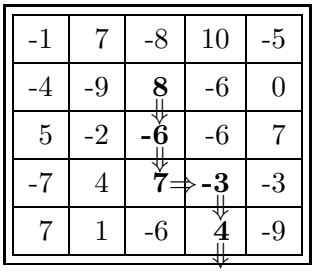
3. A *tonian path* in a graph G is a simple path in G that visits more than half of the vertices of G. (Intuitively, a tonian path is "most of a Hamiltonian path".) Prove that it is NP-hard to determine whether or not a given graph contains a tonian path.



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:

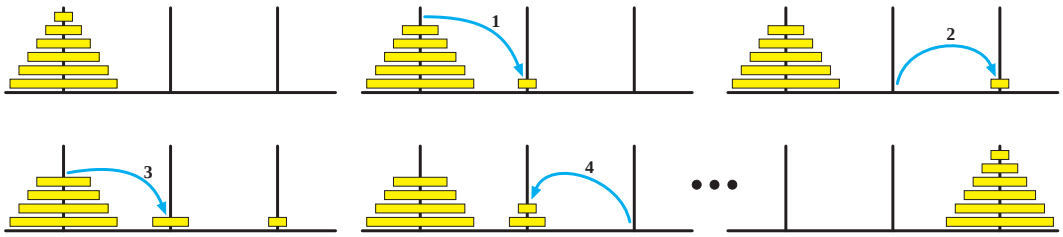
   (a) Given a set of n integers, does it contain a pair of elements a, b such that $a + b = 0$?

   (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.]*

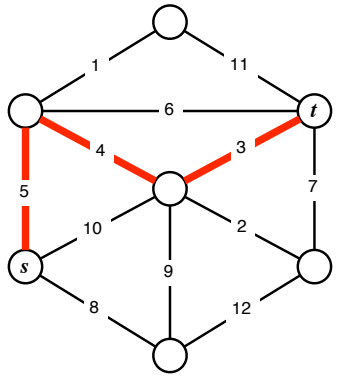
2. Consider a random walk on a path with vertices numbered $1, 2, \ldots, n$ from left to right. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left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

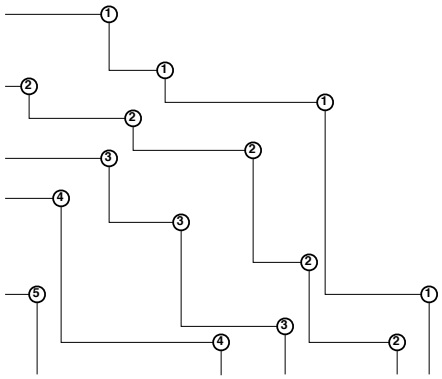
   - Given an arbitrary graph G, compute a graph H such that 5COLOR(G) is true if and only if 5COLOR±1(H) is true.
   - Given an arbitrary graph G, compute a graph H such that 5COLOR±1(G) is true if and only if 5COLOR(H) is true.

   (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).]*

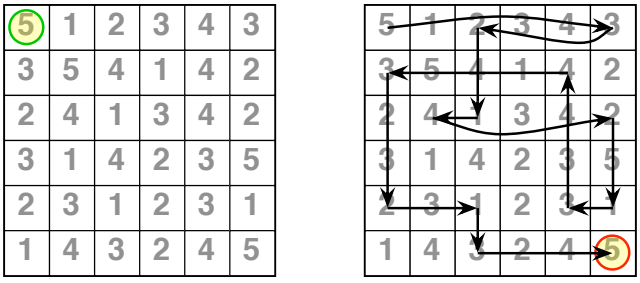
6. Let P be a set of points in the plane. Recall that a point $p \in P$ is *Pareto-optimal* if no other points in P are both above and to the right of p. Intuitively, the sequence of Pareto-optimal points forms a *staircase* with all the other points in P below and to the left. The *staircase layers* 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.

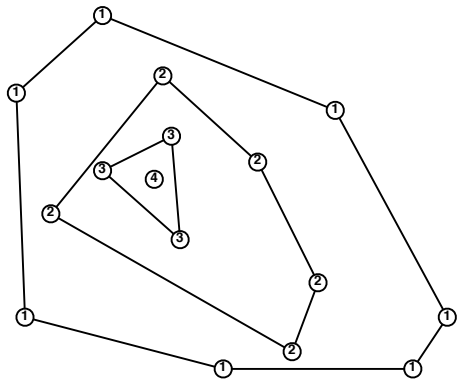
   (b) **[1 pt]** Suppose we knew that SUBSETSUM is NP-hard, and we wanted to prove that PARTITION is NP-hard. Which of the following arguments should we use?

   - 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.

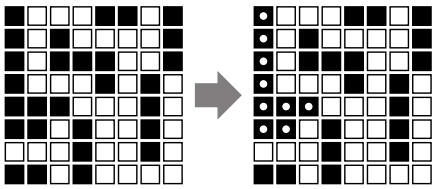


A set of points with 4 convex layers

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 $B[1..n, 1..n]$.

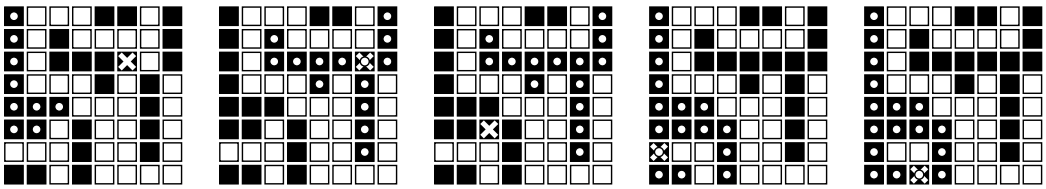
   For example, given the bitmap below as input, your algorithm should return the number 9, because the largest conected black component (marked with white dots on the right) contains nine pixels.



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?