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?

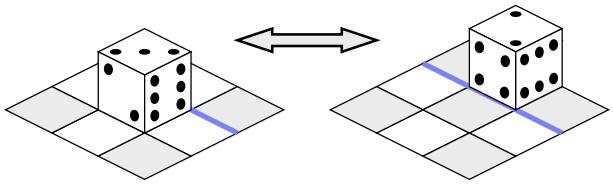
   (c) What is the expected *minimum* value among the cards Teller gives to Penn?

   (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.

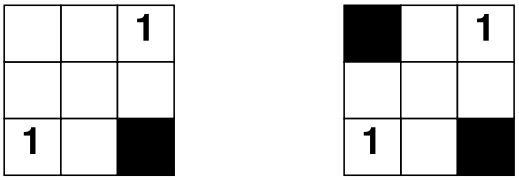
3. A *rolling die maze* is a puzzle involving a standard six-sided die[2] and a grid of squares. You should imagine the grid lying on top of a table; the die always rests on and exactly covers one square. In a single step, you can *roll* the die 90 degrees around one of its bottom edges, moving it to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

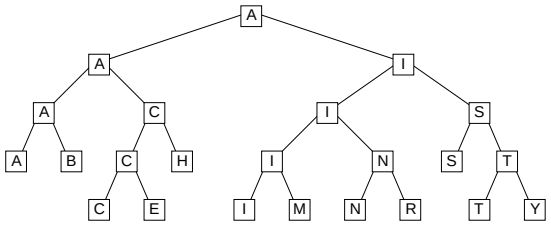
   (b) On average, what is the smallest-numbered card that Penn gives Teller?

   ⋆(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 $\textsc{Find}(x, T)$ that returns $\textsc{True}$ if x is stored in the Phobian binary search tree T, and $\textsc{False}$ 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

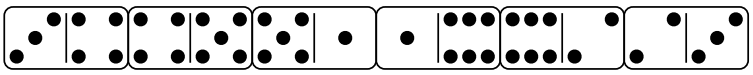
   (a) Show that the problem of deciding whether one graph is a subgraph of another is NP-complete.

   (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.]*

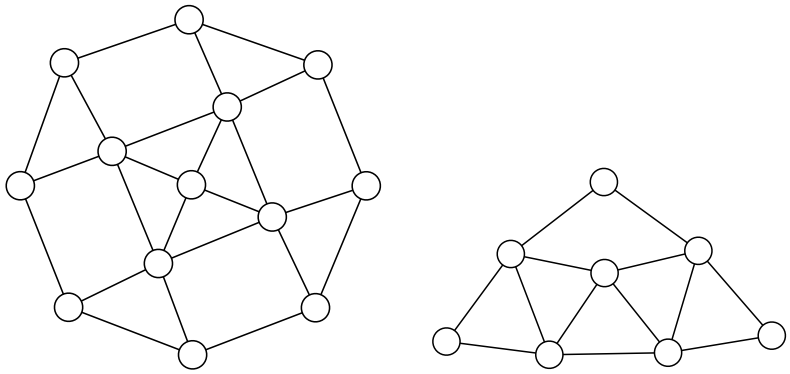


Figure 1. (a) Gadget for planar 3-colorability. (b) Gadget for degree-4 planar 3-colorability.

   (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.)

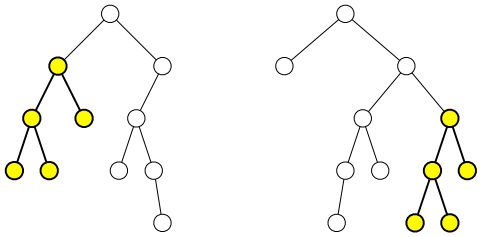
   Also, Xryltconian society is strictly governed by a number of sacred cultural traditions.

   - 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.

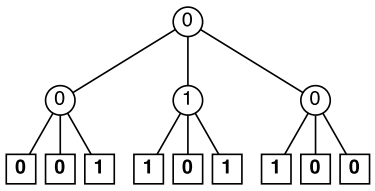
4. Let $X[1..n]$ be an array of n distinct real numbers, and let $N[1..n]$ be an array of indices with the following property: If $X[i]$ is the largest element of X, then $X[N[i]]$ is the smallest element of X; otherwise, $X[N[i]]$ is the smallest element of X that is larger than $X[i]$.

   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.

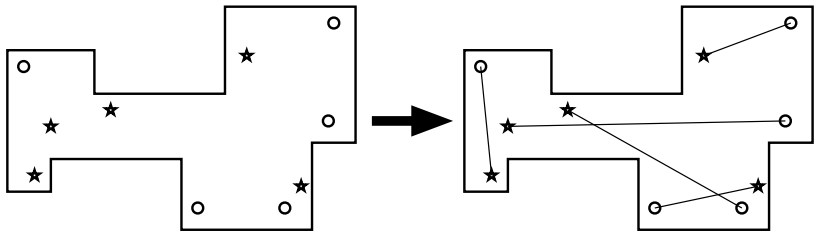
   (b) Prove that the minimum spanning tree of G does not contain any dangerous edge.

   (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.)

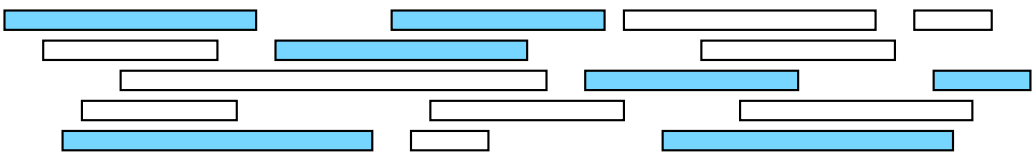
   (a) Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do *not* follow the same greedy strategy as Elmo.

   (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.

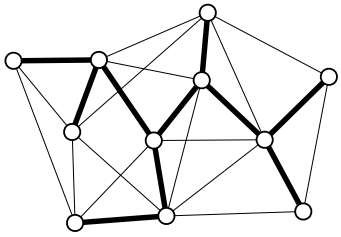
3. Let X be a set of n intervals on the real line. A subset of intervals $Y \subseteq X$ is called a *tiling path* if the intervals in Y cover the intervals in X, that is, any real value that is contained in some interval in X is also contained in some interval in Y. The *size* of a tiling cover is just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

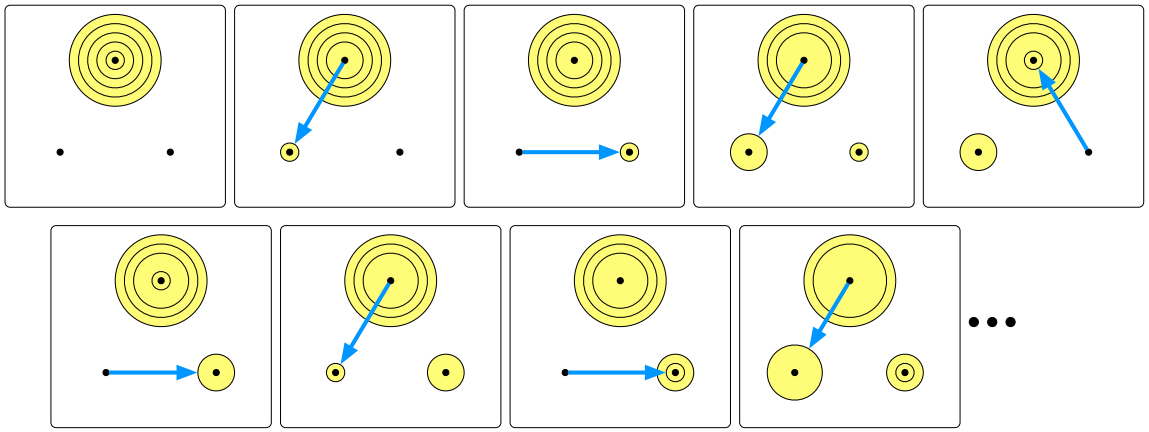


A graph with a spanning tree of maximum degree 3.

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.

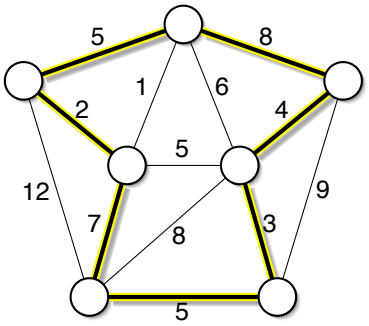
You may take the question sheet with you when you leave.

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.

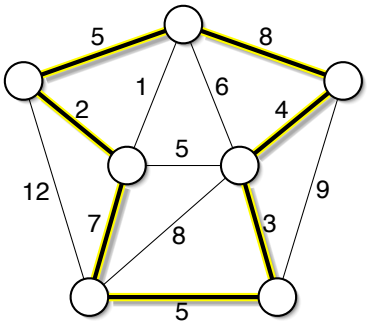
You have 180 minutes to answer six of these questions.

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.