CS 373: Combinatorial Algorithms, Fall 2002
Homework 0, due September 5, 2002 at the beginning of class

Name:		
Net ID:	Alias:	U G

Neatly print your name (first name first, with no comma), your network ID, and an alias of your choice into the boxes above. Circle U if you are an undergraduate, and G if you are a graduate student. **Do not sign your name. Do not write your Social Security number.** Staple this sheet of paper to the top of your homework.

Grades will be listed on the course web site by alias give us, so your alias should not resemble your name or your Net ID. If you don't give yourself an alias, we'll give you one that you won't like.

Before you do anything else, please read the Homework Instructions and FAQ on the CS 373 course web page (http://www-courses.cs.uiuc.edu/~cs373/hwx/faq.html) and then check the box below. There are 300 students in CS 373 this semester; we are quite serious about giving zeros to homeworks that don't follow the instructions.

☐ I have read the CS 373 Homework Instructions and FAQ.

Every CS 373 homework has the same basic structure. There are six required problems, some with several subproblems. Each problem is worth 10 points. Only graduate students are required to answer problem 6; undergraduates can turn in a solution for extra credit. There are several practice problems at the end. Stars indicate problems we think are hard.

This homework tests your familiarity with the prerequisite material from CS 173, CS 225, and CS 273, primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Rosen (the 173/273 textbook), CLRS (especially Chapters 1–7, 10, 12, and A–C), and the lecture notes on recurrences should be sufficient review, but you may want to consult other texts as well.

Required Problems

1. Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. Please *don't* turn in proofs, but you should do them anyway to make sure you're right (and for practice).

$$
\begin{array}{ccccc}
1 & n & n^2 & \lg n & n \lg n \\[2mm]
n^{\lg n} & (\lg n)^n & (\lg n)^{\lg n} & n^{\lg \lg n} & n^{1/\lg n} \\[2mm]
\log_{1000} n & \lg^{1000} n & \lg^{(1000)} n & \lg(n^{1000}) & \left(1 + \frac{1}{1000}\right)^n
\end{array}
$$

To simplify notation, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$ and $f(n) \equiv g(n)$ 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. Solve these recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Please *don't* turn in proofs, but you should do them anyway just for practice. Assume reasonable but nontrivial base cases, and state them if they affect your solution. Extra credit will be given for more exact solutions. *[Hint: Most of these are <u>very</u> easy.]*

$$
\begin{aligned}
A(n) &= 2A(n/2) + n & F(n) &= 9F(\lfloor n/3 \rfloor + 9) + n^2 \\
B(n) &= 3B(n/2) + n & G(n) &= 3G(n-1)/5G(n-2) \\
C(n) &= 2C(n/3) + n & H(n) &= 2H(\sqrt{n}) + 1 \\
D(n) &= 2D(n-1) + 1 & I(n) &= \min_{1 \le k \le n/2} \big(I(k) + I(n-k) + k \big) \\
E(n) &= \max_{1 \le k \le n/2} \big(E(k) + E(n-k) + n \big) & {}^*J(n) &= \max_{1 \le k \le n/2} \big(J(k) + J(n-k) + k \big)
\end{aligned}
$$

3. Recall that a binary tree is *full* if every node has either two children (an internal node) or no children (a leaf). Give at least *four different* proofs of the following fact:

   > In any full binary tree, the number of leaves is exactly one more than the number of internal nodes.

   For full credit, each proof must be self-contained, the proof must be substantially different from each other, and at least one proof must *not* use induction. For each n, your nth correct proof is worth n points, so you need four proofs to get full credit. Each correct proof beyond the fourth earns you extra credit. *[Hint: I know of at least <u>six</u> different proofs.]*

4. Most of you are probably familiar with the story behind the Tower of Hanoï puzzle:[1]

> At the great temple of Benares, there is a brass plate on which three vertical diamond shafts are fixed. On the shafts are mounted n golden disks of decreasing size.[2] At the time of creation, the god Brahma placed all of the disks on one pin, in order of size with the largest at the bottom. The Hindu priests unceasingly transfer the disks from peg to peg, one at a time, never placing a larger disk on a smaller one. When all of the disks have been transferred to the last pin, the universe will end.

Recently the temple at Benares was relocated to southern California, where the monks are considerably more laid back about their job. At the "Towers of Hollywood", the golden disks were replaced with painted plywood, and the diamond shafts were replaced with Plexiglas. More importantly, the restriction on the order of the disks was relaxed. While the disks are being moved, the *bottom* disk on any pin must be the *largest* disk on that pin, but disks further up in the stack can be in any order. However, after all the disks have been moved, they must be in sorted order again.

The Towers of Hollywood.
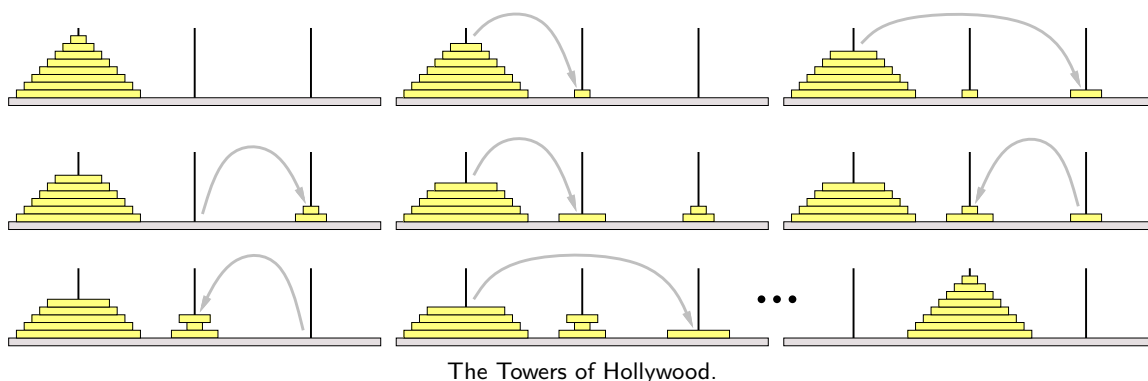
Describe an algorithm[3] that moves a stack of n disks from one pin to the another using the smallest possible number of moves. For full credit, your algorithm should be non-recursive, but a recursive algorithm is worth significant partial credit. *Exactly* how many moves does your algorithm perform? *[Hint: The Hollywood monks can bring about the end of the universe quite a bit faster than the original monks at Benares could.]*

The problem of computing the minimum number of moves was posed in the most recent issue of the *American Mathematical Monthly* (August/September 2002). No solution has been published yet.

---

[1]The puzzle and the accompanying story were both invented by the French mathematician Eduoard Lucas in 1883. See http://www.cs.wm.edu/~pkstoc/toh.html

[2]In the original legend, $n = 64$. In the 1883 wooden puzzle, $n = 8$.

[3]Since you've read the Homework Instructions, you know exactly what this phrase means.

5. On their long journey from Denmark to England, Rosencrantz and Guildenstern amuse themselves by playing the following game with a fair coin. First Rosencrantz flips the coin over and over until it comes up tails. Then Guildenstern flips the coin over and over until he gets as many heads in a row as Rosencrantz got on his turn. Here are three typical games:

> Rosencrantz: <u>H H</u> T
> Guildenstern: H T <u>H H</u>
>
> Rosencrantz: T
> Guildenstern: (no flips)
>
> Rosencrantz: <u>H H H</u> T
> Guildenstern: T H H T H H T H T <u>H H H</u>

   (a) What is the expected number of flips in one of Rosencrantz's turns?

   (b) Suppose Rosencrantz flips k heads in a row on his turn. What is the expected number of flips in Guildenstern's next turn?

   (c) What is the expected total number of flips (by both Rosencrantz and Guildenstern) in a single game?

   Prove your answers are correct. If you have to appeal to "intuition" or "common sense", your answer is almost certainly wrong! You must give exact answers for full credit, but asymptotic bounds are worth significant partial credit.

6. *[This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]*

   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. Thus, the first two arrangements below are traditional, but not the third.
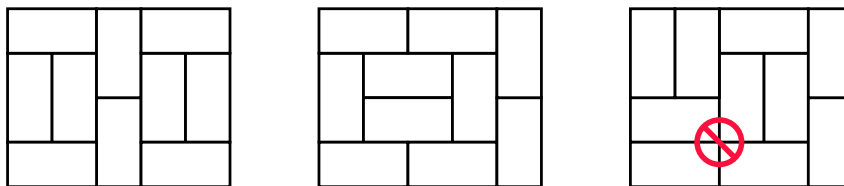
   Two traditional tatami arrangements and one non-traditional arrangement.

   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) [**5 points extra credit**] How many different *traditional* ways are there to tile an $n \times n$ square with 1×2 tatami mats? Prove your answer is correct.

Practice Problems

These problems are only for your benefit; other problems can be found in previous semesters' homeworks on the course web site. You are *strongly* encouraged to do some of these problems as additional practice. Think of them as potential exam questions (hint, hint). Feel free to ask about any of these questions on the course newsgroup, during office hours, or during review sessions.

1. Removing any edge from a binary tree with n nodes partitions it into two smaller binary trees. If both trees have at least $\lceil (n-1)/3 \rceil$ nodes, we say that the partition is *balanced*.

    (a) Prove that every binary tree with more than one vertex has a balanced partition. *[Hint: I know of at least two different proofs.]*

    (b) If each smaller tree has more than $\lfloor n/3 \rfloor$ nodes, we say that the partition is *strictly balanced*. Show that for every n, there is an n-node binary tree with *no* strictly balanced partition.

2. Describe an algorithm COUNTTOTENTOTHE(n) that prints the integers from 1 to 10^n.

    Assume you have a subroutine PRINTDIGIT(d) that prints any integer d between 0 and 9, and another subroutine PRINTSPACE that prints a space character. Both subroutines run in $O(1)$ time. You may want to write (and analyze) a separate subroutine PRINTINTEGER to print an arbitrary integer.

    Since integer variables cannot store arbitrarily large values in most programming languages, your algorithm must not store any value larger than $\max\{10, n\}$ in any single integer variable. Thus, the following algorithm is **not correct**:

    > $\underline{\text{BOGUSCOUNTTOTENTOTHE}(n)\text{:}}$
    >   for $i \leftarrow 1$ to POWER($10, n$)
    >   PRINTINTEGER(i)

    (So what exactly *can* you pass to PRINTINTEGER?)

    What is the running time of your algorithm (as a function of n)? How many digits and spaces does it print? How much space does it use?

3. I'm sure you remember the following simple rules for taking derivatives:

    - Simple cases: $\frac{d}{dx}\alpha = 0$ for any constant α, and $\frac{d}{dx}x = 1$
    - Linearity: $\frac{d}{dx}(f(x) + g(x)) = f'(x) + g'(x)$
    - The product rule: $\frac{d}{dx}(f(x) \cdot g(x)) = f'(x) \cdot g(x) + f(x) \cdot g'(x)$
    - The chain rule: $\frac{d}{dx}(f(g(x)) = f'(g(x)) \cdot g'(x)$

    Using *only* these rules and induction, prove that $\frac{d}{dx}x^c = cx^{c-1}$ for any integer $c \neq -1$. Do not use limits, integrals, or any other concepts from calculus, except for the simple identities listed above. *[Hint: Don't forget about negative values of c!]*

4. This problem asks you to calculate the total resistance between two points in a series-parallel resistor network. Don't worry if you haven't taken a circuits class; everything you need to know can be summed up in two sentences and a picture.

- The total resistance of two resistors in *series* is the sum of their individual resistances.
- The total resistance of two resistors in *parallel* is the reciprocal of the sum of the reciprocals of their individual resistances.

Equivalence laws for series-parallel resistor networks.

What is the *exact* total resistance[4] of the following resistor networks as a function of n? Prove your answers are correct. *[Hint: Use induction. Duh.]*

(a) A complete binary tree with depth n, with a 1Ω resistor at every node, and a common wire joining all the leaves. Resistance is measured between the root and the leaves.
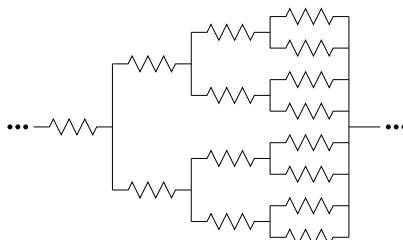

A balanced binary resistor tree with depth 3.

(b) A totally unbalanced full binary tree with depth n (every internal node has two children, one of which is a leaf) with a 1Ω resistor at every node, and a common wire joining all the leaves. Resistance is measured between the root and the leaves.
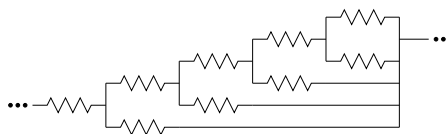
A totally unbalanced binary resistor tree with depth 4.

*(c) A ladder with n rungs, with a 1Ω resistor on every edge. Resistance is measured between the bottom of the legs.
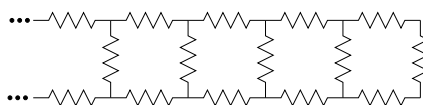
A resistor ladder with 5 rungs.

---

[4]The ISO standard unit of resistance is the Ohm, written with the symbol Ω. Don't confuse this with the asymptotic notation $\Omega(f(n))$!

CS 373: Combinatorial Algorithms, Fall 2002
Homework 1, due September 17, 2002 at 23:59:59

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Undergrads	Grads

This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

Required Problems

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

> 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 $i \leftarrow 1$ to n
>   "We'll drink it out of the $container[i]$, boys,"
>   "Here's a health to the barley-mow!"
>   for $j \leftarrow i$ downto 1
>   "The $container[j]$,"
>   "And the jolly brown bowl!"
>   "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.)

---

[1] 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.

(b) 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 over 21, *exactly* how many ounces of beer would you drink if you sang BARLEYMOW(n)? (Give an *exact* answer, not just an asymptotic bound.)

2. Suppose you have a set S of n numbers. Given two elements you *cannot* determine which is larger. However, you are given an oracle that will tell you the median of a set of three elements.

(a) Give a linear time algorithm to find the pair of the largest and smallest numbers in S.

(b) Give an algorithm to sort S in $O(n \lg n)$ time.

3. Given a black and white pixel image $A[1 \ldots m][1 \ldots n]$, our task is to represent A with a search tree T. Given a query (x, y), a simple search on T should return the color of pixel $A[x][y]$. The algorithm to construct T will be as follows.


CONSTRUCTSEARCHTREE($A[1 \ldots m][1 \ldots n]$):
    //Base Case
    if A contains only one color
        return a leaf node labeled with that color

    //Recurse on Subtrees
    $(i, j) \leftarrow$ CHOOSECUT($A[1 \ldots m][1 \ldots n]$)
    $T_1 \leftarrow$ CONSTRUCTSEARCHTREE($A[1 \ldots i][1 \ldots j]$)
    $T_2 \leftarrow$ CONSTRUCTSEARCHTREE($A[1 \ldots i][j + 1 \ldots n]$)
    $T_3 \leftarrow$ CONSTRUCTSEARCHTREE($A[i + 1 \ldots m][1 \ldots j]$)
    $T_4 \leftarrow$ CONSTRUCTSEARCHTREE($A[i + 1 \ldots m][j + 1 \ldots n]$)

    //Construct the Root
    $T.cut \leftarrow (i, j)$
 $T.children \leftarrow T_1, T_2, T_3, T_4$
    return T

That is, this algorithm divides a multicolor image into quadrants and recursively constructs the search tree for each quadrant. Upon a query (x, y) of T (assuming $1 \le x \le m$ and $1 \le y \le n$), the appropriate subtree is searched. When the correct leaf node is reached, the pixel color is returned. Here's a toy example.
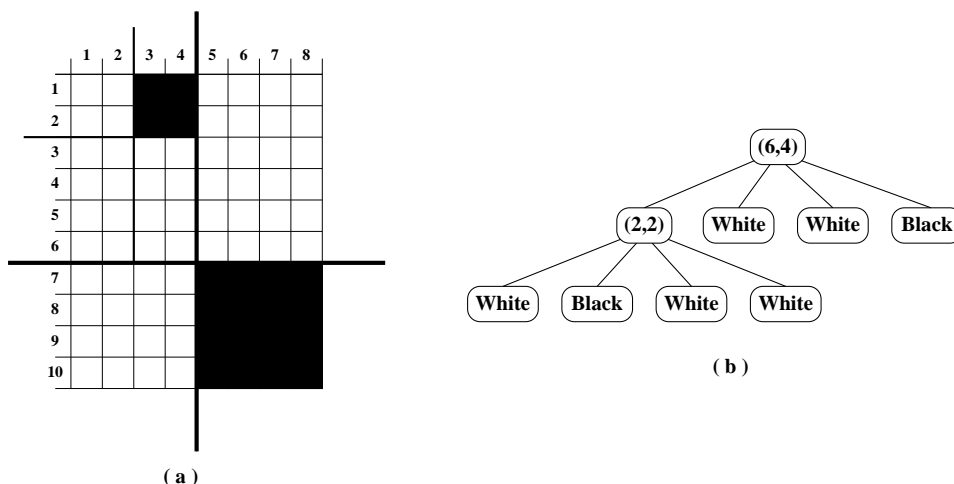
(a) An image A and the chosen cuts. (b) The corresponding search tree.

Your job in this problem is to give an algorithm for CHOOSECUT. The sequence of chosen cuts must result in an optimal search tree T. That is, the expected search depth of a uniformly chosen pixel must be minimized. You may use any external data structures (*i.e.* a global table) that you find necessary. You may also preprocess in order to initialize these structures before the initial call to CONSTRUCTSEARCHTREE($A[1 \ldots m][1 \ldots n]$).

4. Let A be a set of n positive integers, all of which are no greater than some constant $M > 0$. Give an $O(n^2 M)$ time algorithm to determine whether or not it is possible to split A into two subsets such that the sum of the numbers in each subset are equal.

5. Let S and T be two binary trees. A *matching* of S and T is a tree M which is isomorphic to some subtree in each of S and T. Here's an illustration.
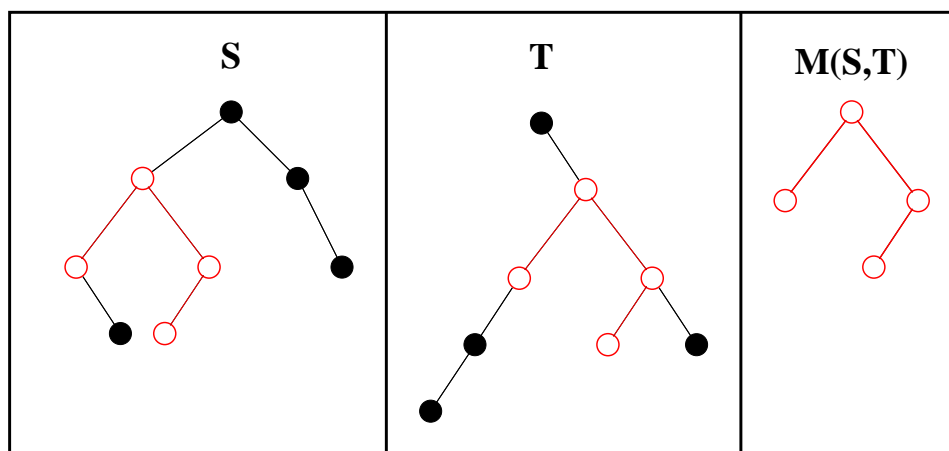

A matching $M(S, T)$ of binary trees S and T.

A *maximal* matching is a matching which contains at least as many vertices as any other matching. Give an algorithm to compute a maximal matching given the roots of two binary trees. Your algorithm should return the size of the match as well as the two roots of the matched subtrees of S and T.

6. *[This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]*

   Let $P[1, \ldots, n]$ be a set of n *convex* points in the plane. Intuitively, if a rubber band were stretched to surround P then each point would touch the rubber band. Furthermore, suppose that the points are labeled such that $P[1], \ldots, P[n]$ is a simple path along the convex hull (*i.e.* $P[i]$ is adjacent to $P[i+1]$ along the rubber band).

   (a) Give a simple algorithm to compute a shortest *cyclic* tour of P.

   (b) A *monotonic* tour of P is a tour that never crosses itself. Here's an illustration.
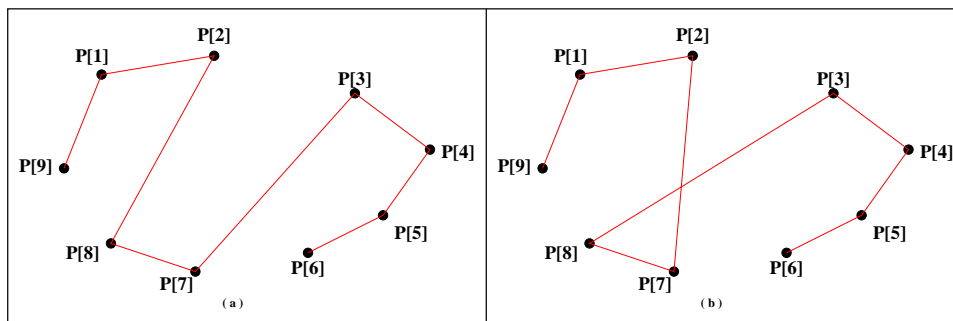
   (a) A monotonic tour of P. (b) A non-monotonic tour of P.

   Prove that any shortest tour of P must be monotonic.

   (c) Given an algorithm to compute a shortest tour of P starting at point $P[1]$ and finishing on point $P[\lfloor \frac{n}{2} \rfloor]$.

4

Practice Problems

These remaining practice problems are entirely for your benefit. Don't turn in solutions—we'll just throw them out—but feel free to ask us about these questions during office hours and review sessions. Think of these as potential exam questions (hint, hint).

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

   (a) the square immediately above,

   (b) the square that is one up and one left (but only if the checker is not already in the leftmost column),

   (c) the square that is one up and one right (but only if the checker is not already in the rightmost column).

   Each time you move from square x to square y, you receive $p(x, y)$ dollars. You are given $p(x, y)$ for all pairs (x,y) for which a move from x to y is legal. Do not assume that $p(x, y)$ is positive.

   Give an algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. Your algorithm is free to pick any square along the bottom edge as a starting point and any square along the top edge as a destination in order to maximize the number of dollars gathered along the way. What is the running time of your algorithm?

2. (CLRS 15-1) The *euclidean traveling-salesman problem* is the problem of determining the shortest closed tour that connects a given set of n points in the plane. Figure (a) below shows the solution to a 7-point problem. The general problem is NP-complete, and its solution is therefore believed to require more than polynomial time.

   J.L. Bentley has suggested that we simplify the problem by restricting our attention to *bitonic tours* (Figure (b) below). That is, tours that start at the leftmost point, go strictly left to right to the rightmost point, and then go strictly right to left back to the starting point. In this case, a polynomial-time algorithm is possible.
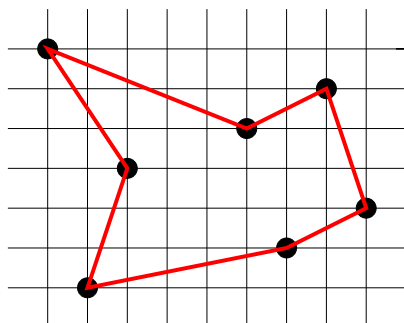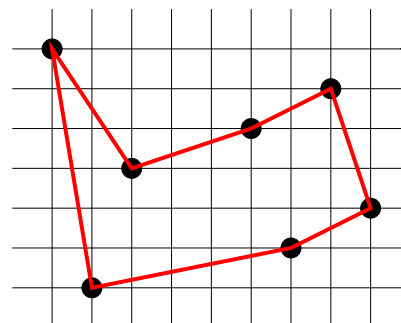
(a)   (b)

Seven points in the plane, shown on a unit grid. (a) The shortest closed tour, with length approximately 24.89. This tour is not bitonic. (b) The shortest bitonic tour for the same set of points. It's length is approximately 25.58.

Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour. You may assume that no two points have the same x-coordinate. [Hint: Scan left to right, maintaining optimal possibilities for the two parts of the tour.]

3. You are given a polygonal line γ made out of n vertices in the plane. Namely, you are given a list of n points in the plane $p_1, \ldots, p_n$, where $p_i = (x_i, y_i)$. You need to display this polygonal line on the screen, however, you realize that you might be able to draw a polygonal line with considerably less vertices that looks identical on the screen (because of the limited resolution of the screen). It is crucial for you to minimize the number of vertices of the polygonal line. (Because, for example, your display is a remote Java applet running on the user computer, and for each vertex of the polygon you decide to draw, you need to send the coordinates of the points through the network which takes a *long long long time*. So the fewer vertices you send, the snappier your applet would be.)

So, given such a polygonal line γ, and a parameter k, you would like to select k vertices of γ that yield the "best" polygonal line that looks like γ.
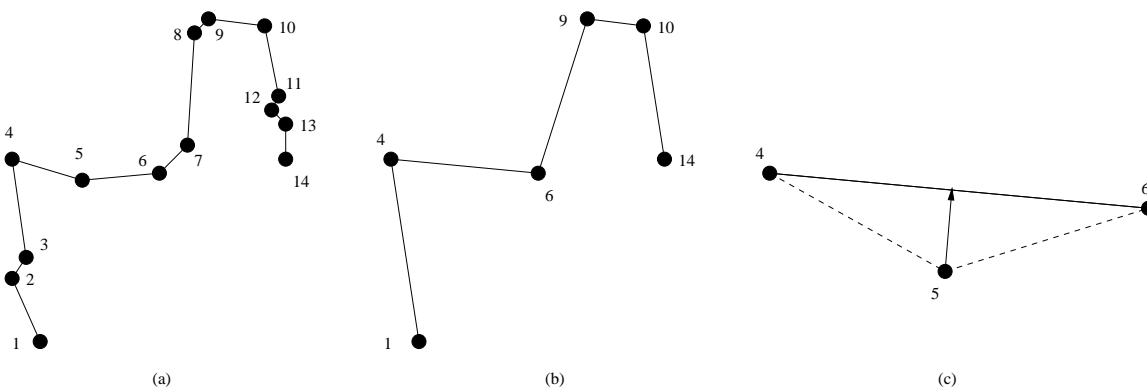
(a) The original polygonal line with 14 vertices. (b) A new polygonal line with 6 vertices. (c) The distance between p_5 on the original polygonal line and the simplification segment $p_4 p_6$. The error of p_5 is
$$\text{error}(p_5) = \text{dist}(p_5, p_4 p_6).$$

Namely, you need to build a new polygonal line γ' and minimize the difference between the two polygonal-lines. The polygonal line γ' is built by selecting k vertices $\{p_{i_1}, p_{i_2}, ..., p_{i_k}\}$ from γ. It is required that $i_1 = 1$, $i_k = n$, and $i_j < i_{j+1}$ for $j = 1, 2, ..., k-1$.

We define the error between γ and γ' by how far from γ' are the vertices of γ. More formally, The difference between the two polygonal lines is

$$error(\gamma, \gamma') = \sum_{j=1}^{k-1} \sum_{m=i_j+1}^{i_{j+1}-1} \text{dist}(p_m, p_{i_j} p_{i_{j+1}}).$$

Namely, for every vertex not in the simplification, its associated error, is the distance to the corresponding simplified segment (see (c) in above figure). The overall error is the sum over all vertices.

You can assume that you are provided with a subroutine that can calculate $\text{dist}(u, vw)$ in constant time, where $\text{dist}(u, vw)$ is the distance between the point u and the segment vw.

Give an $O(n^3)$ time algorithm to find the γ' that minimizes error(γ, γ').

CS 373: Combinatorial Algorithms, Fall 2002
Homework 2 (due Thursday, September 26, 2002 at 11:59:59 p.m.)

Name:		
Net ID:	Alias:	U G

Name:		
Net ID:	Alias:	U G

Name:		
Net ID:	Alias:	U G

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. Since graduate students are required to solve problems that are worth extra credit for other students, **Grad students may not be on the same team as undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate or 1-unit grad student by circling U or G, respectively. Staple this sheet to the top of your homework. **NOTE: You must use different sheet(s) of paper for each problem assigned.**

Required Problems

1. For each of the following problems, the input is a set of n nuts and n bolts. For each bolt, there is exactly one nut of the same size. Direct comparisons between nuts or between bolts are not allowed, but you can compare a nut and a bolt in constant time.

   (a) Describe and analyze a deterministic algorithm to find the largest bolt. *Exactly* how many comparisons does your algorithm perform in the worst case? *[Hint: This is very easy.]*

   (b) Describe and analyze a randomized algorithm to find the largest bolt. What is the *exact* expected number of comparisons performed by your algorithm?

   (c) Describe and analyze an algorithm to find the largest and smallest bolts. Your algorithm can be either deterministic or randomized. What is the *exact* worst-case expected number of comparisons performed by your algorithm? *[Hint: Running part (a) twice is definitely not the most efficient algorithm.]*

   In each case, to receive **full** credit, you need to describe the most efficient algorithm possible.

2. Consider the following algorithm:

```
SlowShuffle(A[1 .. n]) :
      for i ← 1 to n
            B[i] ←Null
      for i ← 1 to n
            index ← Random(1,n)
            while B[index] ≠ Null
                  index ← Random(1,n)
            B[index] ← A[i]
      for i ← 1 to n
            A[i] ← B[i]
```

Suppose that Random(i,j) will return a random number between i and j inclusive in constant time. SlowShuffle will shuffle the input array into a random order such that every permutation is equally likely.

(a) What is the expected running time of the above algorithm. Justify your answer and give a tight asymptotic bound.

(b) Describe an algorithm that randomly shuffles an n-element array, so that every permutation is *equally* likely, in $O(n)$ time.

3. Suppose we are given an undirected graph $G = (V, E)$ together with two distinguished vertices s and t. An **s-t min-cut** is a set of edges that once removed from the graph, will disconnect s from t. We want to find such a set with the minimum cardinality(The smallest number of edges). In other words, we want to find the smallest set of edges that will seperate s and t

To do this we repeat the following step $|V| - 2$ times: Uniformly at random, pick an edge from the set E which contains all edges in the graph excluding those that directly connects vertices s and t. Merge the two vertices that is connected by this randomly selected edge. If as a result there are several edges between some pair of vertices, retain them all. Edges that are between the two merged vertices are removed so that there are never any self-loops. We refer to this process of merging the two end-points of an edge into a single vertex as the *contraction* of that edge. Notice with each contraction the number of vertices of G decreases by one.

As this algorithm proceeds, the vertex s may get merged with a new vertex as the result of an edge being contracted. We call this vertex the s-vertex. Similarly, we have a t-vertex. During the contraction algorithm, we ensure that we never contract an edge between the s-vertex and the t-vertex.
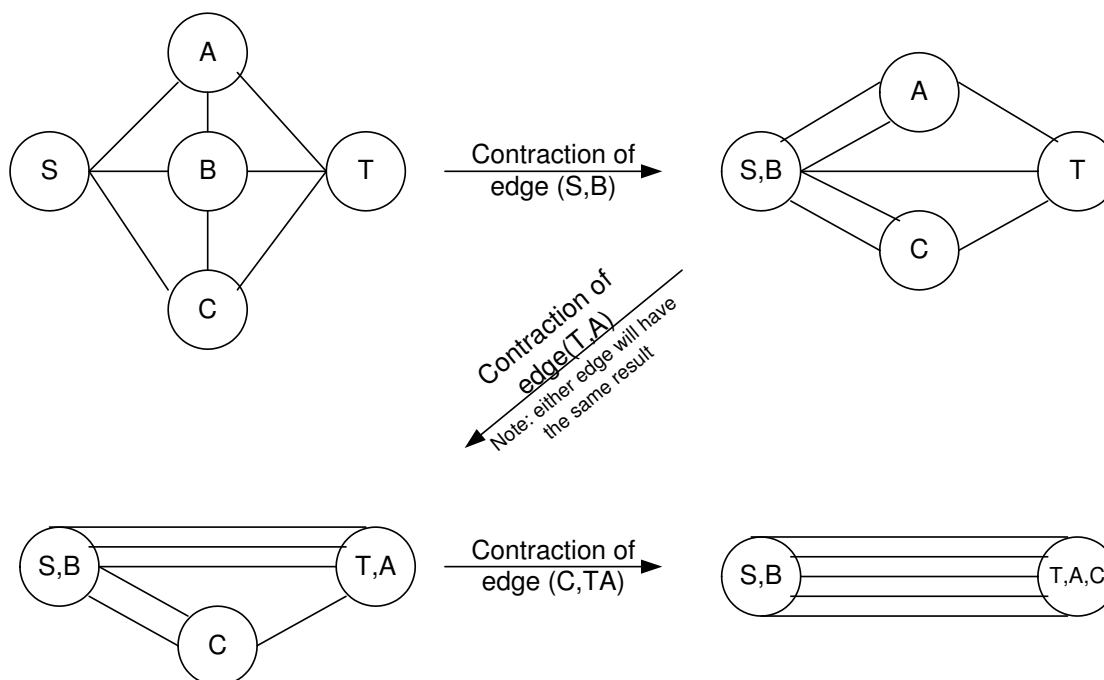
(a) Give an example of a graph in which the probability that this algorithm finds an s-t min-cut is exponentially small($O(1/a^n)$). Justify your answers.
(*Hint: Think multigraphs*)

(b) Give an example of a graph such that there are $O(2^n)$ number of s-t min-cuts. Justify your answers.

4. Describe a modification of treaps that supports the following operations, each in $O(\log n)$ expected time:

- INSERT(x): Insert a new element x into the data structure.
- DELETE(x): Delete an element x from the data structure.
- COMPUTERANK(x): Return the number of elements in the data structure less than or equal to x.
- FINDBYRANK(r): Return the kth smallest element in the data structure.

Describe and analyze the algorithms that implement each of these operations. *[Hint: Don't reinvent the wheel!]*

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

- MAKEQUEUE: Return a new priority queue storing the empty set.
- FINDMIN(Q): Return the smallest element stored in Q (if any).
- DELETEMIN(Q): Delete the smallest element stored in Q (if any).
- INSERT(Q, x): Insert element x into Q.
- MELD(Q_1, Q_2): Return a new priority queue containing all the elements stored in Q_1 and Q_2. The component priority queues are destroyed.
- DECREASEKEY(Q, x, y): Replace an element x of Q with a smaller key y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q storing x.
- DELETE(Q, x): Delete an element $x \in Q$. The input is a pointer directly to the node in Q storing x.

A simple way to implement this data structure is to use a heap-ordered binary tree, where each node stores an element, a pointer to its left child, a pointer to its right child, and a pointer to its parent. MELD(Q_1, Q_2) can be implemented with the following randomized algorithm.

- If either one of the queues is empty, return the other one.
- If the root of Q_1 is smaller than the root of Q_2, then recursively MELD Q_2 with either right(Q_1) or left(Q_1), each with probability $1/2$.
- Similarly, if the root of Q_2 is smaller than the root of Q_1, then recursively MELD Q_1 with a randomly chosen child of Q_2.

(a) Prove that for *any* heap-ordered trees Q_1 and Q_2, 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 path in an n-node binary tree, if each left/right choice is made with equal probability?]* For extra credit, prove that the running time is $O(\log n)$ *with high probability.*

(b) Show that each of the operations DELETEMIN, INSERT, DECREASEKEY, and DELETE can be implemented with one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ with high probability.)

6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

The following randomized algorithm selects the rth smallest element in an unsorted array $A[1,..,n]$. For example, to find the smallest element, you would call RANDOMSELECT($A, 1$); to find the median element, you would call RANDOMSELECT($A, \lfloor n/2 \rfloor$). Recall from lecture that PARTITION splits the array into three parts by comparing the pivot element $A[p]$ to every other element of the array, using $n - 1$ comparisons altogether, and returns the new index of the pivot element.

RANDOMSELECT($A[1..n], r$) :
     $p \leftarrow$ RANDOM$(1, n)$
     $k \leftarrow$ PARTITION$(A[1..n], p)$

     if $r < k$
         return RANDOMSELECT$(A[1..k-1], r)$
     else if $r > k$
         return RANDOMSELECT$(A[k+1..n], r-k)$
     else
         return $A[k]$

(a) State a recurrence for the expected running time of RANDOMSELECT, as a function of both n and r.

(b) What is the *exact* probability that RANDOMSELECT compares the ith smallest and jth smallest elements in the input array? The correct answer is a simple function of i, j, and r. [*Hint: Check your answer by trying a few small examples.*]

(c) Show that for any n and r, the expected running time of RANDOMSELECT is $\Theta(n)$. You can use either the recurrence from part (a) or the probabilities from part (b). For extra credit, find the *exact* expected number of comparisons, as a function of n and r.

(d) What is the expected number of times that RANDOMSELECT calls itself recursively?

Practice Problems

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.

   You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are *and* gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

   (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

   (b) Unfortunately, Death won't let you even look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $\Theta(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*
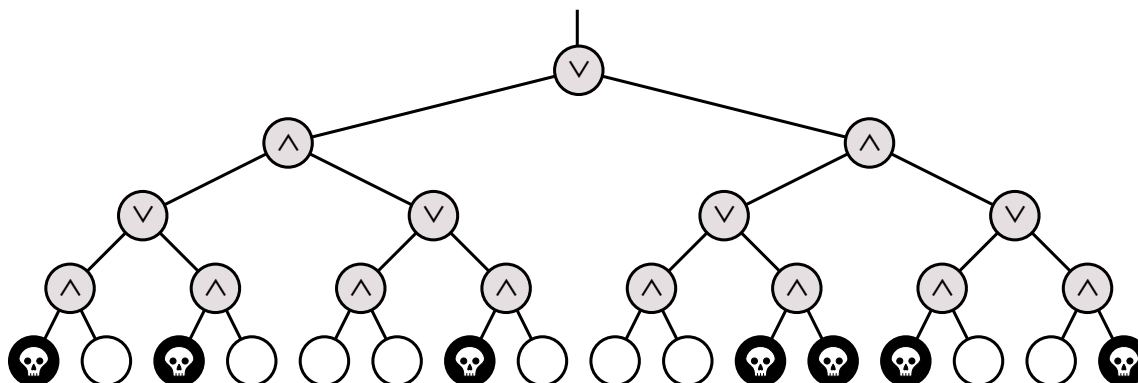
2. WHat is the *exact* number of nodes in a skip list storing n keys, *not* counting the sentinel nodes at the beginning and end of each level? Justify your answer.

3. Suppose we are given two sorted arrays $A[1..n]$ and $B[1..n]$ and an integer k. Describe an algorithm to find the kth smallest element in the union of A and B. (For example, if $k = 1$, your algorithm should return the smallest element of $A \cup B$; if $k = n$, our algorithm should return the median of $A \cup B$.) You can assume that the arrays contain no duplicates. Your algorithm should be able to run in $\Theta(\log n)$ time. *[Hint: First try to solve the special case $k = n$.]*

CS 373: Combinatorial Algorithms, Fall 2002
Homework 3, due October 17, 2002 at 23:59:59

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Undergrads	Grads

This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

Required Problems

1. (a) Prove that only one subtree gets rebalanced in a scapegoat tree insertion.

    (b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree. (Recall that $I(v) = \max\{0, |T| - |s| - 1\}$, where T is the child of greater height and s the child of lesser height, and $|v|$ is the number of nodes in subtree v. A perfectly balanced tree has two perfectly balanced subtrees, each with as close to half the nodes as possible.)

    *(c) Show that you can rebuild a fully balanced binary tree from an unbalanced tree in $O(n)$ time using only $O(\log n)$ additional memory.

2. 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 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 (it makes it much more difficult).

3. A stack is a FILO/LIFO data structure that represents a stack of objects; access is only allowed at the top of the stack. In particular, a stack implements two operations:

    - PUSH(x): adds x to the top of the stack.

- POP: removes the top element and returns it.

A queue is a FIFO/LILO data structure that represents a row of objects; elements are added to the front and removed from the back. In particular, a queue implements two operations:

- ENQUEUE(x): adds x to the front of the queue.
- DEQUEUE: removes the element at the back of the queue and returns it.

Using two stacks and no more than $O(1)$ additional space, show how to simulate a queue for which the operations ENQUEUE and DEQUEUE run in constant amortized time. You should treat each stack as a black box (i.e., you may call PUSH and POP, but you do not have access to the underlying stack implementation). Note that each PUSH and POP performed by a stack takes $O(1)$ time.

4. A data structure is *insertion-disabled* if there is no way to add elements to it. For the purposes of this problem, further assume that an insertion-disabled data structure implements the following operations with the given running times:

   - INITIALIZE(S): Return an insertion-disabled data structure that contains the elements of S. Running time: $O(n \log n)$.
   - SEARCH(D, x): Return TRUE if x is in D; return FALSE if not. Running time: $O(\log n)$.
   - RETURNALL(D, x): Return an unordered set of all elements in D. Running time: $O(n)$.
   - DELETE(D, x): Remove x from D if x is in D. Running time: $O(\log n)$.

   Using an approach known as the Bentley-Saxe Logarithmic Method (BSLM), it is possible to represent a dynamic (i.e., supports insertions) data structure with a collection of insertion-disabled data structures, where each insertion-disabled data structure stores a number of elements that is a distinct power of two. For example, to store $39 = 2^0 + 2^1 + 2^2 + 2^5$ elements in a BSLM data structure, we use four insertion-disabled data structures with 2^0, 2^1, 2^2, and 2^5 elements.

   To find an element in a BSLM data structure, we search the collection of insertion-disabled data structures until we find (or don't find) the element.

   To insert an element into a BSLM data structure, we think about adding a 2^0-size insertion-disabled data structure. However, an insertion-disabled data structure with 2^0 elements may already exist. In this case, we can combine two 2^0-size structures into a single 2^1-size structure. However, there may already be a 2^1-size structure, so we will need to repeat this process. In general, we do the following: Find the smallest i such that for all nonnegative $k < i$, there is a 2^k-sized structure in our collection. Create a 2^i-sized structure that contains the element to be inserted and all elements from 2^k-sized data structures for all $k < i$. Destroy all 2^k-sized data structures for $k < i$.

Inserting an element into a BSLM data structure.
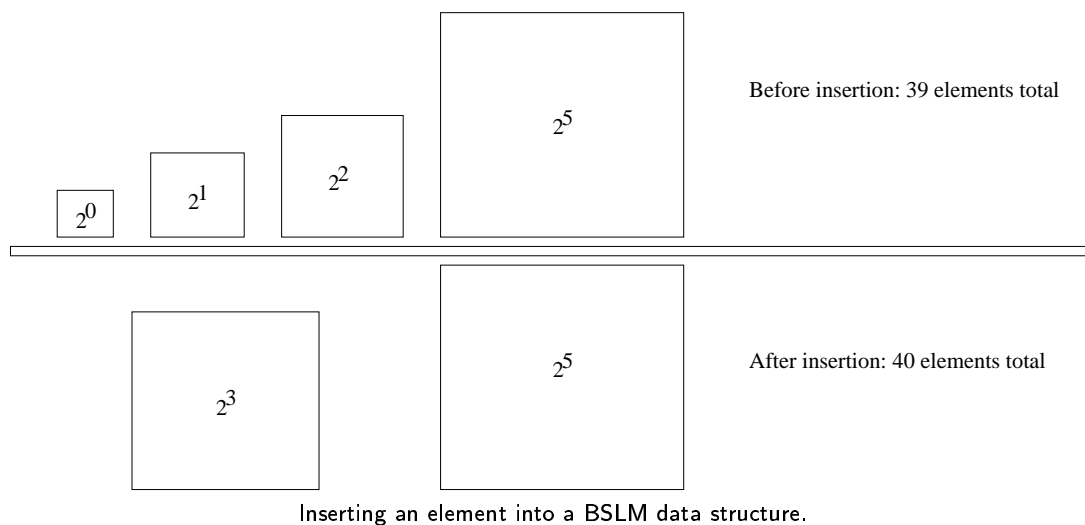
We delete elements from the BSLM data structure lazily. To delete an element, we first search the collection of insertion-disabled data structures for it. Then we call DELETE to remove the element from its insertion-disabled data structure. This means that a 2^i-sized insertion-disabled data structure might store less than 2^i elements. That's okay; we just say that it stores 2^i elements and say that 2^i is its *pretend* size. We keep track of a single variable, called $Waste$, which is initially 0 and is incremented by 1 on each deletion. If $Waste$ exceeds three-quarters of the total pretend size of all insertion-disabled data structures in our collection (i.e., the total number of elements stored), we rebuild our collection of insertion-disabled data structures. In particular, we create a 2^m-sized insertion-disabled data structure, where 2^m is the smallest power that is greater than or equal to the total number of elements stored. All elements are stored in this 2^m-sized insertion-disabled data structure, and all other insertion-disabled data structures in our collection are destroyed. $Waste$ is reset to $2^m - n$, where n is the total number of elements stored in the BSLM data structure.

Your job is to prove the running times of the following three BSLM operations:

- SEARCHBSLM(D, x): Search for x in the collection of insertion-disabled data structures that represent the BSLM data structure D. Running time: $O(\log^2 n)$ worst-case.

- INSERTBSLM(D, x): Insert x into the collection of insertion-disabled data structures that represent the BSLM data structure D, modifying the collection as necessary. Running time: $O(\log^2 n)$ amortized.

- DELETEBSLM(D, x): Delete x from the collection of insertion-disabled data structures that represent the BSLM data structure D, rebuilding when there is a lot of wasted space. Running time: $O(\log^2 n)$ amortized.

5. Except as noted, the following sub-problems refer to a Union-Find data structure that uses both path compression and union by rank.

  (a) Prove that in a set of n elements, a sequence of n consecutive FIND operations takes $O(n)$ total time.

  (b) Show that any sequence of m MAKESET, FIND, and UNION operations takes only $O(m)$ time if all of the UNION operations occur before any of the FIND operations.

(c) Now consider part b with a Union-Find data structure that uses path compression but does *not* use union by rank. Is $O(m)$ time still correct? Prove your answer.

6. *[This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]*

Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of 'fits', where the ith least significant fit indicates whether the sum includes the ith Fibonacci number F_i. For example, the fit string 101110 represents the number $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$. Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string.]

Practice Problems

These remaining practice problems are entirely for your benefit. Don't turn in solutions—we'll just throw them out—but feel free to ask us about these questions during office hours and review sessions. Think of these as potential exam questions (hint, hint).

1. A *multistack* consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first move all the elements in S_i to stack S_{i+1} to make room. But if S_{i+1} is already full, we first move all its members to S_{i+2}, and so on. Moving a single element from one stack to the next takes $O(1)$ time.

Making room for one new element in a multistack.
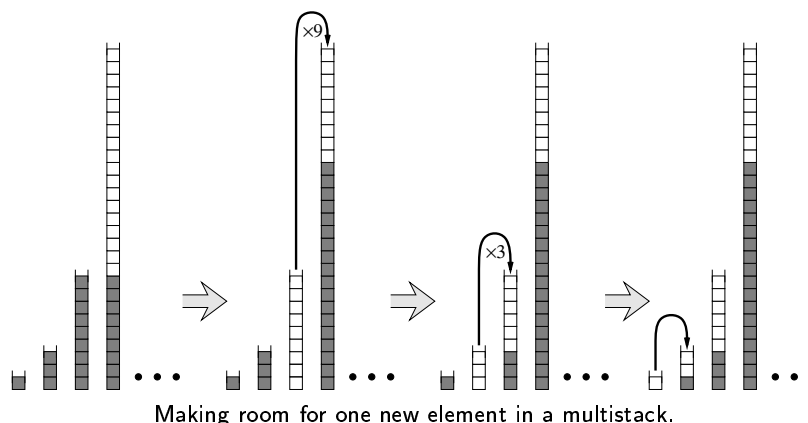
   (a) In the worst case, how long does it take to push one more element onto a multistack containing n elements?

   (b) Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack. You can use any method you like.

2. A hash table of size m is used to store n items with $n \leq m/2$. Open addressing is used for collision resolution.

   (a) Assuming uniform hashing, show that for $i = 1, 2, \ldots, n$, the probability that the ith insertion requires strictly more than k probes is at most 2^{-k}.

   (b) Show that for $i = 1, 2, \ldots, n$, the probability that the ith insertion requires more than $2 \lg n$ probes is at most $1/n^2$.

   Let the random variable X_i denote the number of probes required by the ith insertion. You have shown in part (b) that $\Pr\{X_i > 2 \lg n\} \leq 1/n^2$. Let the random variable $X = \max_{1 \leq i \leq n} X_i$ denote the maximum number of probes required by any of the n insertions.

   (c) Show that $\Pr\{X > 2 \lg n\} \leq 1/n$.

   (d) Show that the expected length of the longest probe sequence is $E[X] = O(\lg n)$.

3. A sequence of n operations is performed on a data structure. The ith operation costs i if i is an exact power of 2, and 1 otherwise. That is operation i costs $f(i)$, where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

Determine the amortized cost per operation using the following methods of analysis:

(a) Aggregate method

(b) Accounting method

*(c) Potential method

CS 373: Combinatorial Algorithms, Fall 2002
Homework 4, due Thursday, October 31, 2002 at 23:59.99

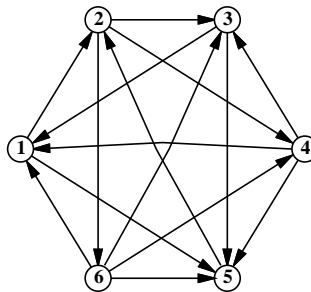| Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Undergrads	Grads

This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

Required Problems

1. Tournament:

   A *tournament* is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A *Hamiltonian path* is a sequence of directed edges, joined end to end, that visits every vertex exactly once.

   Prove that every tournament contains at least one Hamiltonian path.

   A six-vertex tournament containing the Hamiltonian path $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$.

2. Acrophobia:

   Consider a graph $G = (V, E)$ whose nodes are cities, and whose edges are roads connecting the cities. For each edge, the weight is assigned by h_e, the maximum altitude encountered when traversing the specified road. Between two cities s and t, we are interested in those paths whose maximum altitude is as low as possible. We will call a subgraph, G', of G an *acrophobic friendly subgraph*, if for any two nodes s and t the path of minimum altitude is always

included in the subgraph. For simplicity, assume that the maximum altitude encountered on each road is unique.

   (a) Prove that every graph of n nodes has an acrophobic friendly subgraph that has only $n - 1$ edges.

   (b) Construct an algorithm to find an acrophobic friendly subgraph given a graph $G = (V, E)$.

3. Refer to the lecture notes on single-source shortest paths. The GENERICSSSP algorithm described in class can be implemented using a stack for the 'bag'. Prove that the resulting algorithm, given a graph with n nodes as input, could perform $\Omega(2^n)$ relaxation steps before stopping. You need to describe, for any positive integer n, a specific weighted directed n-vertex graph that forces this exponential behavior. The easiest way to describe such a family of graphs is using an *algorithm*!

4. Neighbors:
Two spanning trees T and T' are defined as *neighbors* if T' can be obtained from T by swapping a single edge. More formally, there are two edges e and f such that T' is obtained from T by adding edge e and deleting edge f.

   (a) Let T denote the minimum cost spanning tree and suppose that we want to find the second cheapest tree T' among all trees. Assuming unique costs for all edges, prove that T and T' are neighbors.

   (b) Given a graph $G = (V, E)$, construct an algorithm to find the second cheapest tree, T'.

   (c) Consider a graph, H, whose vertices are the spanning trees of the graph G. Two vertices are connected by an edge if and only if they are neighbors as previously defined. Prove that for any graph G this new graph H is connected.

5. Network Throughput:
Suppose you are given a graph of a (tremendously simplified) computer network $G = (V, E)$ such that a weight, b_e, is assigned to each edge representing the communication bandwidth of the specified channel in Kb/s and each node is assigned a value, l_v, representing the server latency measured in seconds/packet. Given a fixed packet size, and assuming all edge bandwidth values are a multiple of the packet size, your job is to build a system to decide which paths to route traffic between specfied servers.
More formally, a person wants to route traffic from server s to server t along the path of maximum throughput. Give an algorithm that will allow a network design engineer to choose an optimal path by which to route data traffic.

6. All-Pairs-Shortest-Path:

   [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

   Given an undirected, unweighted, connected graph $G = (V, E)$, we wish to solve the distance version of the all-pairs-shortest-path problem. The algorithm APD takes the $n \times n$ 0-1 adjacency matrix A and returns an $n \times n$ matrix D such that d_{ij} represents the shortest path between vertices i and j.

   $\underline{\text{APD}(A)}$
         $Z \leftarrow A \cdot A$
         let B be an $n \times n$ matrix, where $b_{ij} = 1$ iff $i \neq j$ and $(a_{ij} = 1$ or $z_{ij} > 0)$
         if $b_{ij} = 1$ for all $i \neq j$
               return $D \leftarrow 2B - A$
         $T \leftarrow \text{APD}(B)$
         $X \leftarrow T \cdot A$
         foreach x_{ij}
               if $x_{ij} \geq t_{ij} \cdot degree(j)$
                     $d_{ij} \leftarrow 2t_{ij}$
               else
                     $d_{ij} \leftarrow 2t_{ij} - 1$
         return D

   (a) In the APD algorithm above, what do the matrices Z, B, T, and X represent? Justify your answers.

   (b) Prove that the APD algorithm correctly computes the matrix of shortest path distances. In other words, prove that in the output matrix D, each entry d_{ij} represents the shortest path distance between node i and node j.

   (c) Suppose we can multiply two $n \times n$ matrices in $M(n)$ time, where $M(n) = \Omega(n^2)$.[1] Prove that APD runs in $O(M(n) \log n)$ time.

   ---

   [1]The matrix multiplication algorithm you already know runs in $O(n^3)$ time, but this is not the fastest known. The current record is $M(n) = O(n^{2.376})$, due to Don Coppersmith and Shmuel Winograd. Determinign the smallest possible value of $M(n)$ is a long-standing open problem.

Practice Problems

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

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

2. The incidence matrix of an undirected graph $G = (V, E)$ is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that
$$b_{ij} = \begin{cases} 1 & \text{if vertex } v_i \text{ is an endpoint of edge } e_j \\ 0 & \text{otherwise} \end{cases}$$

   (a) Describe what all the entries of the matrix product BB^T represent (B^T is the matrix transpose). Justify.

   (b) Describe what all the entries of the matrix product $B^T B$ represent. Justify.

   ★(c) Let $C = BB^T - 2A$. Let C' be C with the first row and column removed. Show that $\det C'$ is the number of spanning trees. (A is the adjacency matrix of G, with zeroes on the diagonal).

3. Reliable Network:
   Suppose you are given a graph of a computer network $G = (V, E)$ and a function $r(u, v)$ that gives a reliability value for every edge $(u, v) \in E$ such that $0 \le r(u, v) \le 1$. The reliability value gives the probability that the network connection corresponding to that edge will *not* fail. Describe and analyze an algorithm to find the most reliable path from a given source vertex s to a given target vertex t.

4. Aerophobia:
   After graduating you find a job with Aerophobes-R'-Us, the leading traveling agency for aerophobic people. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying so the trip should be as short as possible.

   In other words, a person wants to fly from city A to city B in the shortest possible time. S/he turns to the traveling agent who knows all the departure and arrival times of all the flights on the planet. Give an algorithm that will allow the agent to choose an optimal route to minimize the total time in transit. Hint: rather than modify Dijkstra's algorithm, modify the data. The total transit time is from departure to arrival at the destination, so it will include layover time (time waiting for a connecting flight).

5. The Seven Bridges of Königsberg:
   During the eighteenth century the city of Königsberg in East Prussia was divided into four sections by the Pregel river. Seven bridges connected these regions, as shown below. It was said that residents spent their Sunday walks trying to find a way to walk about the city so as to cross each bridge exactly once and then return to their starting point.
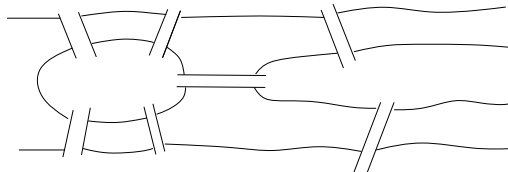
   (a) Show how the residents of the city could accomplish such a walk or prove no such walk exists.

   (b) Given any undirected graph $G = (V, E)$, give an algorithm that finds a cycle in the graph that visits every edge exactly once, or says that it can't be done.

6. Given an undirected graph $G = (V, E)$ with costs $c_e \geq 0$ on the edges $e \in E$ give an $O(|E|)$ time algorithm that tests if there is a minimum cost spanning tree that contains the edge e.

7. Combining Boruvka and Prim:
   Give an algorithm that find the MST of a graph G in $O(m \log \log n)$ time by combining Boruvka's and Prim's algorithm.

8. Minimum Spanning Tree changes:
   Suppose you have a graph G and an MST of that graph (i.e. the MST has already been constructed).

   (a) Give an algorithm to update the MST when an edge is added to G.

   (b) Give an algorithm to update the MST when an edge is deleted from G.

   (c) Give an algorithm to update the MST when a vertex (and possibly edges to it) is added to G.

9. Nesting Envelopes
   You are given an unlimited number of each of n different types of envelopes. The dimensions of envelope type i are $x_i \times y_i$. In nesting envelopes inside one another, you can place envelope A inside envelope B if and only if the dimensions A are *strictly smaller* than the dimensions of B. Design and analyze an algorithm to determine the largest number of envelopes that can be nested inside one another.

10. $o(V^2)$ Adjacency Matrix Algorithms

    (a) Give an $O(V)$ algorithm to decide whether a directed graph contains a *sink* in an adjacency matrix representation. A sink is a vertex with in-degree $V - 1$.

(b) An undirected graph is a scorpion if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree $V - 2$ (the body) connected to the other $V - 3$ vertices (the feet). Some of the feet may be connected to other feet.

Design an algorithm that decides whether a given adjacency matrix represents a scorpion by examining only $O(V)$ of the entries.

(c) Show that it is impossible to decide whether G has at least one edge in $O(V)$ time.

11. Shortest Cycle:
Given an **undirected** graph $G = (V, E)$, and a weight function $f : E \rightarrow \mathbf{R}$ on the **edges**, give an algorithm that finds (in time polynomial in V and E) a cycle of smallest weight in G.

12. Longest Simple Path:
Let graph $G = (V, E)$, $|V| = n$. A *simple path* of G, is a path that does not contain the same vertex twice. Use dynamic programming to design an algorithm (not polynomial time) to find a simple path of maximum length in G. Hint: It can be done in $O(n^c 2^n)$ time, for some constant c.

13. Minimum Spanning Tree:
Suppose all edge weights in a graph G are equal. Give an algorithm to compute an MST.

14. Transitive reduction:
Give an algorithm to construct a *transitive reduction* of a directed graph G, i.e. a graph G^{TR} with the fewest edges (but with the same vertices) such that there is a path from a to b in G iff there is also such a path in G^{TR}.

15. (a) What is $5^{2^{29^{5^0} + 23^{4^1} + 17^{3^2} + 11^{2^3} + 5^{1^4}}} \mod 6$?

(b) What is the capital of Nebraska? Hint: It is not Omaha. It is named after a famous president of the United States that was not George Washington. The distance from the Earth to the Moon averages roughly 384,000 km.

CS 373: Combinatorial Algorithms, Fall 2002

Homework 5 (due Thur. Nov. 21, 2002 at 11:59 pm)

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. (10 points) Given two arrays, $A[1..n]$ and $B[1..m]$ we want to determine whether there is an $i \geq 0$ such that $B[1] = A[i+1]$,$B[2] = A[i+2]$,...,$B[m] = A[i+m]$. In other words, we want to determine if B is a substring of A. Show how to solve this problem in $O(n \log n)$ time with high probability.

2. (5 points) Let $a, b, c \in \mathbb{Z}^+$.

   (a) Prove that $\gcd(a,b) \cdot \mathrm{lcm}(a,b) = ab$.
   (b) Prove $\mathrm{lcm}(a,b,c) = \mathrm{lcm}(\mathrm{lcm}(a,b),c)$.
   (c) Prove $\gcd(a,b,c)\mathrm{lcm}(ab,ac,bc) = abc$.

3. (5 points) Describe an efficient algorithm to compute multiplicative inverses modulo a prime p. Does your algorithm work if the modulos is composite?

4. (10 points) Describe an efficient algorithm to compute $F_n \bmod m$, given integers n and m as input.

5. (10 points) Let n have the prime factorization $p_1^{k_1} p_2^{k_2} \cdots p_t^{k_t}$, where the primes p_i are distinct and have exponents $k_i > 0$. Prove that

$$\phi(n) = \prod_{i=1}^{t} p_i^{k_i - 1}(p_i - 1).$$

Conclude that $\phi(n)$ can be computed in polynomial time given the prime factorization of n.

6. (10 points) Suppose we want to compute the Fast Fourier Transform of an intger vector $P[0\mathrel{..}n-1]$. We could choose an integer m larger than any coefficient $P[i]$, and then perform all arithmetic modulo m (or more formally, in the ring $\mathbb{Z}_m$). In order to make the FFT algorithm work, we need to find an integer that functions as a "primitive nth root of unity modulo m".

    For this problem, let's assume that $m = 2^{n/2} + 1$, where as usual n is a power of two.

  (a) Prove that $2^n \equiv 1 \pmod{m}$.

  (b) Prove that $\sum_{k=0}^{n-1} 2^k \equiv 0 \pmod{m}$. These two conditions imply that 2 is a primitive nth root of unity in $\mathbb{Z}_m$.

  (c) Given (a), (b), and (c), *briefly* argue that the "FFT modulo m" of P is well-defined and be computed in $O(n \log n)$ arithmetic operations.

  (d) Prove that n has a multiplicative inverse in $\mathbb{Z}_m$. *[Hint: n is a power of 2, and m is odd.]* We need this property to implement the inverse FFT modulo m.

  (e) What is the FFT of the sequence $[3, 1, 3, 3, 7, 3, 7, 3]$ modulo 17?

7. (10 points) *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

  (a) Prove that for any integer $n > 1$, if the n-th Fibonacci number F_n is prime then either n is prime or $n = 4$.

  (b) Prove that if a divides b, then F_a divides F_b.

  (c) Prove that $\gcd(F_a, F_b) = F_{\gcd(a,b)}$. This immediately implies parts (a) and (b), so if you solve this part, you don't have to solve the other two.

Practice Problems

1. Let $a, b, n \in Z \setminus \{0\}$. Assume $\gcd(a,b)|n$. Prove the entire set of solutions to the equation

$$n = ax + by$$

   is given by:

$$\Gamma = \{x_o + \frac{tb}{\gcd(a,b)}, y_0 - \frac{ta}{\gcd(a,b)} : t \in Z\}.$$

2. Show that in the RSA cryptosystem the decryption exponent d can be chosen such that $de \equiv 1$ mod $\mathrm{lcm}(p-1, q-1)$.

3. Let (n, e) be a public RSA key. For a plaintext $m \in \{0, 1, ..., n-1\}$, let $c = m^e$ mod n be the corresponding ciphertext. Prove that there is a positive integer k such that

$$m^{e^k} \equiv m \bmod n.$$

   For such an integer k, prove that

$$c^{e^{k-1}} \equiv m \bmod n.$$

   Is this dangerous for RSA?

4. Prove that if Alice's RSA public exponent e is 3 and an adversary obtains Alice's secret exponent d, then the adversary can factor Alice's modulus n in time polynomial in the number of bits in n.

CS 373: Combinatorial Algorithms, Fall 2002

Homework 6 (Do not hand in!)

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. (10 points) Prove that SAT is still a NP-complete problem even under the following constraints: each variable must show up once as a positive literal and once or twice as a negative literal in the whole expression. For instance, $(A \lor \bar{B}) \land (\bar{A} \lor C \lor D) \land (\bar{A} \lor B \lor \bar{C} \lor \bar{D})$ satisfies the constraints, while $(A \lor \bar{B}) \land (\bar{A} \lor C \lor D) \land (A \lor B \lor \bar{C} \lor \bar{D})$ does not, because positive literal A appears twice.

2. (10 points) A domino is 2×1 rectanble divided into two squares, with a certain number of pips(dots) in each square. In most domino games, the players lay down dominos at either end of a single chain. Adjacent dominos in the chain must have matching numbers. (See the figure below.)

   Describe and analyze an efficient algorithm, or prove that it is NP-complete, to determine wheter a given set of n dominos can be lined up in a single chain. For example, for the sets of dominos shown below, the correct output is TRUE.
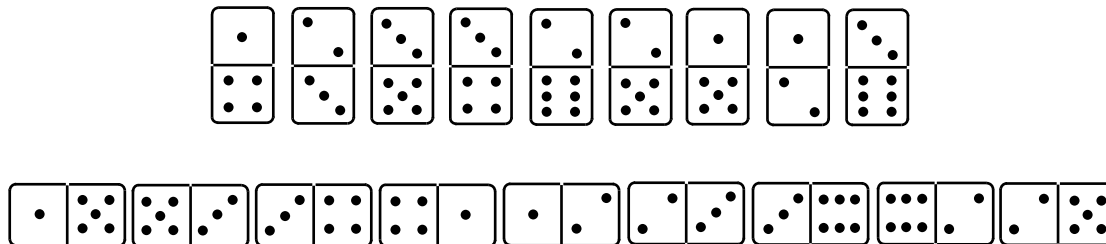
Top: A set of nine dominos
Bottom:The entire set lined up in a single chain

3. (10 points) Prove that the following 2 problems are NP-complete. Given an undirected Graph $G = (V, E)$, a subset of vertices $V' \subseteq V$, and a positive integer k:

   (a) determine whether there is a spanning tree T of G whose leaves are the same as V'.

   (b) determine whether there is a spanning tree T of G whose degree of vertices are all less than k.

4. (10 points) An optimized version of Knapsack problem is defined as follows. Given a finite set of elements U where each element of the set $u \in U$ has its own size $s(u) > 0$ and the value $v(u) > 0$, maximize $A(U') = \sum_{u \in U'} v(u)$ under the condition $\sum_{u \in U'} s(u) \leq B$ and $U' \subseteq U$. This problem is NP-hard. Consider the following polynomial time approximation algorithm. Determine the worst case approximation ratio $R(U) = \max_{U} Opt(U)/Approx(U)$ and prove it.

GREEDY:
      Put all the elements $u \in U$ into an array $A[i]$
      Sort $A[i]$ by $v(u)/s(u)$ in a decreasing order
      $S \leftarrow 0$
      $V \leftarrow 0$
      for $i \leftarrow 0$ to NumOfElements
          if $(S + s(u[i]) > B)$
              break
          $S \leftarrow S + s(u[i])$
          $V \leftarrow V + v(u[i])$
      return V

APPROXIMATIONALGORITHM:
      $A_1 \leftarrow Greedy()$
      $A_2 \leftarrow SingleElement()$
      return max$(A1, A2)$

SINGLEELEMENT:
      Put all the elements $u \in U$ into an array $A[i]$
      $V \leftarrow 0$
      for $i \leftarrow 0$ to NumOfElements
          if $(s(u[i]) \leq B \;\&\; V < v(u[i]))$
              $V \leftarrow v(u[i])$
      return V

5. (10 points) The recursion fairy's distant cousin, the reduction genie, shows up one day with a magical gift for you: a box that determines in constant time whether or not a graph is 3-colorable.(A graph is 3-colorable if you can color each of the vertices red, green, or blue, so that every edge has do different colors.) The magic box does not tell you how to color the graph, just wheter or not it can be done. Devise and analyze an algorithm to 3-color any graph in **polynomial time** using the magic box.

6. (10 points) The following is an NP-hard version of PARTITION problem.

PARTITION(NP-HARD):
      Given a set of n positive integers $S = \{a_i | i = 0 ... n - 1\}$,

      minimize max $\left(\sum_{a_i \in T} a_i, \sum_{a_i \in S - T} a_i \right)$

      where T is a subset of S.

A polynomial time approximation algorithm is given in what follows. Determine the worst case approximation ratio $\min_{S} Approx(S)/Opt(S)$ and prove it.

```
APPROXIMATIONALGORITHM:
      Sort S in an increasing order
      s1 ← 0
      s2 ← 0
      for i ← 0 to n
            if s1 ≤ s2
                  s1 ← s1 + a_i
            else
                  s2 ← s2 + a_i
      result ← max(s1, s2)
```

Practice Problems

1. Construct a linear time algorithm for 2 SAT problem.

2. Assume that $P \neq NP$. Prove that there is no polynomial time approximation algorithm for an optimized version of Knapsack problem, which outputs $A(I)$ s.t. $|Opt(I) - A(I)| \leq K$ for any instance I, where K is a constant.

3. Your friend Toidi is planning to hold a party for the coming Christmas. He wants to take a picture of all the participants including himself, but he is quite **shy** and thus cannot take a picture of a person whom he does not know very well. Since he has only **shy** friends, every participant coming to the party is also **shy**. After a long struggle of thought he came up with a seemingly good idea:

   - At the beginning, he has a camera.
   - A person, holding a camera, is able to take a picture of another participant whom the person knows very well, and pass a camera to that participant.
   - Since he does not want to waste films, everyone has to be taken a picture exactly once.

   Although there can be some people whom he does not know very well, he knows completely who knows whom well. Therefore, in theory, given a list of all the participants, he can determine if it is possible to take all the pictures using this idea. Since it takes only linear time to take all the pictures if he is brave enough (say "Say cheese!" N times, where N is the number of people), as a student taking CS373, you are highly expected to give him an advice:

   - show him an efficient algorithm to determine if it is possible to take pictures of all the participants using his idea, given a list of people coming to the party.
   - or prove that his idea is essentially facing a NP-complete problem, make him give up his idea, and give him an efficient algorithm to practice saying "Say cheese!":

     e.g.,
     ```
     for i ← 0 to N
           Make him say "Say cheese!" 2^i times
     ```
     oops, it takes exponential time...

4. Show, given a set of numbers, that you can decide wheter it has a subset of size 3 that adds to zero in polynomial time.

5. Given a CNF-normalized form that has at most one negative literal in each clause, construct an efficient algorithm to solve the satisfiability problem for these clauses. For instance,

$$(A \lor B \lor \bar{C}) \land (B \lor \bar{A}),$$
$$(A \lor \bar{B} \lor C) \land (B \lor \bar{A} \lor D) \land (A \lor D),$$
$$(\bar{A} \lor B) \land (B \lor \bar{A} \lor C) \land (C \lor D)$$

satisfy the condition, while

$$(\bar{A} \lor B \lor \bar{C}) \land (B \lor \bar{A}),$$
$$(A \lor \bar{B} \lor C) \land (B \lor \bar{A} \lor \bar{D}) \land (A \lor D),$$
$$(\bar{A} \lor B) \land (B \lor \bar{A} \lor C) \land (\bar{C} \lor \bar{D})$$

do not.

6. The ExactCoverByThrees problem is defined as follows: given a finite set X and a collection C of 3-element subsets of X, does C contain an exact cover for X, that is, a sub-collection $C' \subseteq C$ where every element of X occurs in exactly one member of C'? Given that ExactCoverByThrees is NP-complete, show that the similar problem ExactCoverByFours is also NP-complete.

7. The *LongestSimpleCycle* problem is the problem of finding a simple cycle of maximum length in a graph. Convert this to a formal definition of a decision problem and show that it is NP-complete.

<div style="border:1px solid black; text-align:center;">

Write your answers in the separate answer booklet.

</div>

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 $\sum_{i=1}^{n} \frac{i}{n}$?

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

   (c) How many bits do you need to write 10^n in binary?

   (d) What is the solution of the recurrence $T(n) = 9T(n/3) + n$?

   (e) What is the solution of the recurrence $T(n) = T(n-2) + \frac{3}{n}$?

   (f) What is the solution of the recurrence $T(n) = 5T\left(\left\lceil \frac{n-17}{25} \right\rceil - \lg \lg n\right) + \pi n + 2^{\sqrt{\log^* n}} - 6$?

   (g) What is the worst-case running time of randomized quicksort?

   (h) The expected time for inserting one item into a randomized treap is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty treap?

   (i) Suppose STUPIDALGORITHM produces the correct answer to some problem with probability $1/n$. How many times do we have to run STUPIDALGORITHM to get the correct answer with high probability?

   (j) Suppose you correctly identify three of the possible answers to this question as obviously wrong. If you choose one of the three remaining answers at random, each with equal probability, what is your expected score for this question?

2. Consider the following algorithm for finding the smallest element in an unsorted array:

   <div style="border:1px solid black; display:inline-block;">

   $\underline{\text{RANDOMMIN}(A[1..n]):}$
       $min \leftarrow \infty$
       for $i \leftarrow 1$ to n in random order
          if $A[i] < min$
             $min \leftarrow A[i]$ $(\star)$
       return min

   </div>

   (a) [**1 point**] In the worst case, how many times does RANDOMMIN execute line $(\star)$?

   (b) [**3 points**] What is the probability that line $(\star)$ is executed during the nth iteration of the for loop?

   (c) [**6 points**] What is the exact expected number of executions of line $(\star)$? (A correct $\Theta()$ bound is worth 4 points.)

3. 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 Phobos-orbits[2] 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.
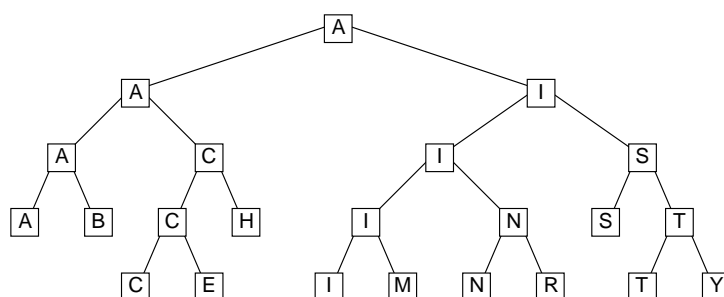
A Phobian binary search tree for the set $\{M, A, R, T, I, N, B, Y, S, C, H, E\}$.

(a) [**2 points**] 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) [**2 points**] Show how to perform a rotation in a Phobian binary search tree in $O(1)$ time.

(c) [**6 points**] 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.

4. 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?]*

---

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

[2] 1000 Phobos orbits $\approx$ 1 Earth year

5. *[Graduate students must answer this question.]*

   A *common supersequence* of two strings A and B is a string of minimum total length that includes both the characters of A in order and the characters of B in order. Design and analyze and algorithm to compute the length of the *shortest* common supersequence of two strings $A[1..m]$ and $B[1..n]$. For example, if the input strings are ANTHROHOPOBIOLOGICAL and PRETERDIPLOMATICALLY, your algorithm should output 31, since a shortest common supersequence of those two strings is <u>PRE</u><u>ANT</u>H<u>ER</u>O<u>H</u>O<u>D</u>P<u>OB</u>I<u>O</u>P<u>LOMAT</u>G<u>ICALLY</u>. You do not need to compute an actual supersequence, just its length. For full credit, your algorithm must run in $\Theta(nm)$ time.

> **Write your answers in the separate answer booklet.**
> **This is a 90-minute exam. The clock started when you got the questions.**

1. Professor Quasimodo has built a device that automatically rings the bells in the tower of the Cathédrale de Notre Dame de Paris so he can finally visit his true love Esmerelda. Every hour exactly on the hour (when the minute hand is pointing at the 12), the device rings at least one of the n bells in the tower. Specifically, the ith bell is rung once every i hours.

   For example, suppose $n = 4$. If Quasimodo starts his device just after midnight, then his device rings the bells according to the following twelve-hour schedule:
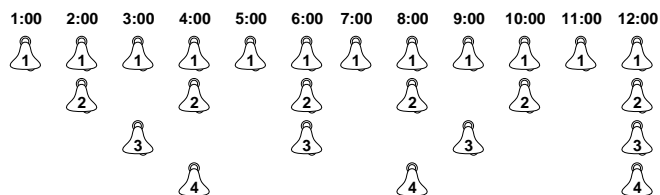
   What is the *amortized* number of bells rung per hour, as a function of n? For full credit, give an exact closed-form solution; a correct $\Theta()$ bound is worth 5 points.

2. Let G be a directed graph, where every edge $u \to v$ has a weight $w(u \to v)$. To compute the shortest paths from a start vertex s to every other node in the graph, the generic single-source shortest path algorithm calls INITSSSP once and then repeatedly calls RELAX until there are no more tense edges.

   $$\boxed{\begin{array}{l} \underline{\text{INITSSSP}(s):} \\ \quad \text{dist}(s) \leftarrow 0 \\ \quad \text{pred}(s) \leftarrow \text{NULL} \\ \quad \text{for all vertices } v \neq s \\ \quad\quad \text{dist}(v) \leftarrow \infty \\ \quad\quad \text{pred}(v) \leftarrow \text{NULL} \end{array}} \qquad \boxed{\begin{array}{l} \underline{\text{RELAX}(u \to v):} \\ \quad \text{if dist}(v) > \text{dist}(u) + w(u \to v) \\ \quad\quad \text{dist}(v) \leftarrow \text{dist}(u) + w(u \to v) \\ \quad\quad \text{pred}(v) \leftarrow u \end{array}}$$

   Suppose the input graph has no negative cycles. Let v be an arbitrary vertex in the input graph. **Prove** that after every call to RELAX, if $\text{dist}(v) \neq \infty$, then $\text{dist}(v)$ is the total weight of some path from s to v.

3. Suppose we want to maintain a dynamic set of values, subject to the following operations:

   - INSERT(x): Add x to the set (if it isn't already there).
   - PRINT&DELETERANGE(a, b): Print and delete every element x in the range $a \leq x \leq b$. For example, if the current set is $\{1, 5, 3, 4, 8\}$, then PRINT&DELETERANGE($4, 6$) prints the numbers 4 and 5 and changes the set to $\{1, 3, 8\}$.

   Describe and analyze a data structure that supports these operations, each with amortized cost $O(\log n)$.

4. (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 $B[1 \mathinner{.\,.} n, 1 \mathinner{.\,.} n]$.

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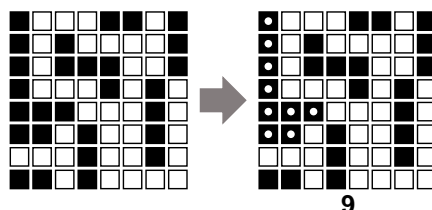
(b) [**4 pts**] Design and analyze an algorithm $\textsc{Blacken}(i, j)$ that colors the pixel $B[i, j]$ black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.

   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 $\textsc{Blacken}$ algorithm.

    9   14   14   16   17
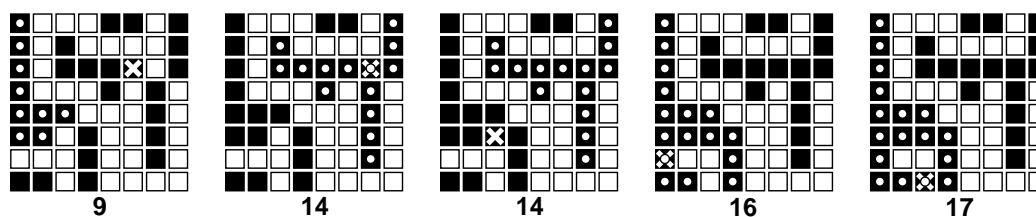
(c) [**2 pts**] What is the *worst-case* running time of your $\textsc{Blacken}$ algorithm?

5. *[Graduate students must answer this question.]*

   After a grueling 373 midterm, you decide to take the bus home. Since you planned ahead, you have a schedule that lists the times and locations of every stop of every bus in Champaign-Urbana. Unfortunately, there isn't a single bus that visits both your exam building and your home; you must transfer between bus lines at least once.

   Describe and analyze an algorithm to determine the sequence of bus rides that will get you home as early as possible, assuming there are b different bus lines, and each bus stops n times per day. Your goal is to minimize your *arrival time*, not the time you actually spend travelling. Assume that the buses run exactly on schedule, that you have an accurate watch, and that you are too tired to walk between bus stops.

> **Write your answers in the separate answer booklet.**
> **This is a 180-minute exam. The clock started when you got the questions.**

1. The *d-dimensional hypercube* is the graph defined as follows. There are 2^d vertices, each labeled with a different string of d bits. Two vertices are joined by an edge if their labels differ in exactly one bit.
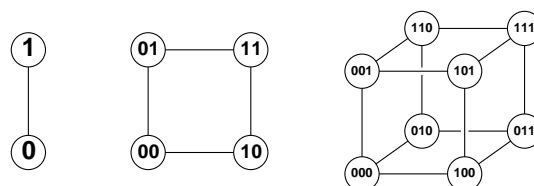
The 1-dimensional, 2-dimensional, and 3-dimensional hypercubes.

   (a) [**8 pts**] Recall that a Hamiltonian cycle passes through every vertex in a graph exactly once. **Prove** that for all $d \geq 2$, the d-dimensional hypercube has a Hamiltonian cycle.

   (b) [**2 pts**] Which hypercubes have an Eulerian circuit (a closed walk that visits every edge exactly once)? *[Hint: This is very easy.]*

2. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight. The number of nodes in the graph is n.
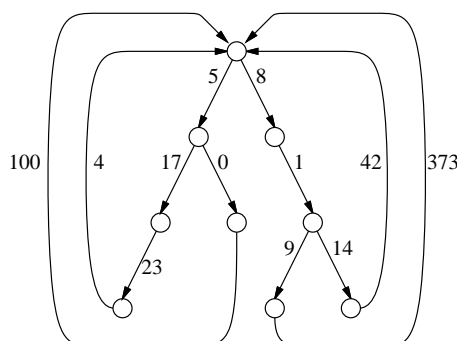
   (a) How long would it take Dijkstra's algorithm to compute the shortest path between two vertices u and v in a looped tree?

   (b) Describe and analyze a faster algorithm.

3. Prove that $(x + y)^p \equiv x^p + y^p \pmod{p}$ for any prime number p.

4. A *palindrome* is a string that reads the same forwards and backwards, like X, 373, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. *Any* string can be written as a sequence of palindromes. For example, the string bubbaseesabanana ('Bubba sees a banana.') can be decomposed in several ways; for example:

$$\text{bub} + \text{baseesab} + \text{anana}$$
$$\text{b} + \text{u} + \text{bb} + \text{a} + \text{sees} + \text{aba} + \text{nan} + \text{a}$$
$$\text{b} + \text{u} + \text{bb} + \text{a} + \text{sees} + \text{a} + \text{b} + \text{anana}$$
$$\text{b} + \text{u} + \text{b} + \text{b} + \text{a} + \text{s} + \text{e} + \text{e} + \text{s} + \text{a} + \text{b} + \text{a} + \text{n} + \text{a} + \text{n} + \text{a}$$

Describe an efficient algorithm to find the *minimum* number of palindromes that make up a given input string. For example, given the input string bubbaseesabanana, your algorithm would return the number 3.

5. Your boss wants you to find a *perfect* hash function for mapping a known set of n items into a table of size m. A hash function is perfect if there are *no* collisions; each of the n items is mapped to a different slot in the hash table. Of course, this requires that $m \geq n$.

After cursing your 373 instructor for not teaching you about perfect hashing, you decide to try something simple: repeatedly pick *random* hash functions until you find one that happens to be perfect. A random hash function h satisfies two properties:

- $\Pr\big[h(x) = h(y)\big] = \frac{1}{m}$ for any pair of items $x \neq y$.
- $\Pr\big[h(x) = i\big] = \frac{1}{m}$ for any item x and any integer $1 \leq i \leq m$.

(a) [**2 pts**] Suppose you pick a random hash function h. What is the *exact* expected number of collisions, as a function of n (the number of items) and m (the size of the table)? Don't worry about how to *resolve* collisions; just count them.

(b) [**2 pts**] What is the *exact* probability that a random hash function is perfect?

(c) [**2 pts**] What is the *exact* expected number of different random hash functions you have to test before you find a perfect hash function?

(d) [**2 pts**] What is the *exact* probability that none of the first N random hash functions you try is perfect?

(e) [**2 pts**] How many random hash functions do you have to test to find a perfect hash function *with high probability*?

To get full credit for parts (a)–(d), give *exact* closed-form solutions; correct $\Theta(\cdot)$ bounds are worth significant partial credit. Part (e) requires only a $\Theta(\cdot)$ bound; an exact answer is worth extra credit.

6. Your friend Toidi is planning to hold a Christmas party. He wants to take a picture of all the participants, including himself, but he is quite shy and thus cannot take a picture of a person whom he does not know very well. Since he has only shy friends[1], everyone at the party is also shy. After thinking hard for a long time, he came up with a seemingly good idea:

   - Toidi brings a disposable camera to the party.
   - Anyone holding the camera can take a picture of someone they know very well, and then pass the camera to that person.
   - In order not to waste any film, every person must have their picture taken exactly once.

   Although there can be some people Toidi does not know very well, he knows completely who knows whom well. Thus, *in principle*, given a list of all the participants, he can determine whether it is possible to take all the pictures using this idea. But how quickly?

   Either describe an efficient algorithm to solve Toidi's problem, or show that the problem is NP-complete.

7. The recursion fairy's cousin, the reduction genie, shows up one day with a magical gift for you: a box that can solve the NP-complete PARTITION problem in constant time! Given a set of positive integers as input, the magic box can tell you in constant time it can be split into two subsets whose total weights are equal.

   For example, given the set $\{1, 4, 5, 7, 9\}$ as input, the magic box cheerily yells "YES!", because that set can be split into $\{1, 5, 7\}$ and $\{4, 9\}$, which both add up to 13. Given the set $\{1, 4, 5, 7, 8\}$, however, the magic box mutters a sad "Sorry, no."

   The magic box does not tell you *how* to partition the set, only whether or not it can be done. Describe an algorithm to actually split a set of numbers into two subsets whose sums are equal, **in polynomial time**, using this magic box.[2]

---

[1]Except you, of course. Unfortunately, you can't go to the party because you're taking a final exam. Sorry!
[2]Your solution to problem 4 in homework 1 does *not* solve this problem in polynomial time.