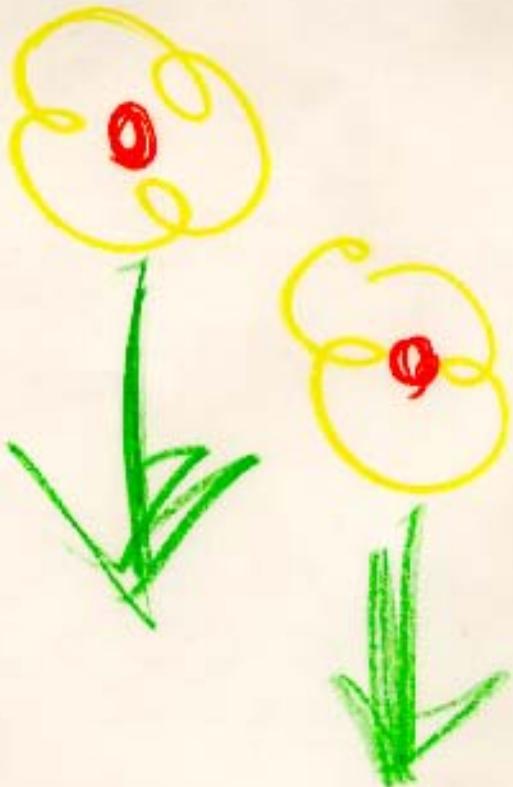
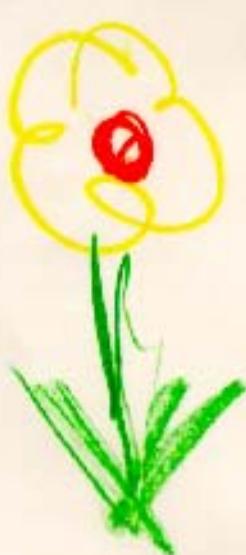


CS 373

Name: Johnny
alias: pikachu

HW1



1

Trees have no cycles but must be connected.
Tournaments are cliques with their edges directed.
Hamilton circuits, Eulerian paths,
These are a few of my favorite graphs.

Matchings and bicliques and blossoms and bases,
Kempe chains, hypercubes, forests, and faces,
AKS networks that split into halves,
These are a few of my favorite graphs.

Short paths of co-authors leading to Erdos,
Large neural networks that translate from Kurdish,
Finite projective planes - they make me laugh
These are a few of my favorite graphs.

Chorus: Propositions, corollaries,
Problems that are starred,
I simply remember my favorite graphs
And then they don't seem so hard.

If there's no K_5 or $K_{3,3}$ minor,
Old Kuratowski says it'll be 'plinor'.
Four's enough colors if there are no gaffes!
These are a few of my favorite graphs.

Quadrangles, thrackles, and triangulations,
Minor-closed families and sparsifications,
Voronoi diagrams found on giraffes,
These are a few of my favorite graphs.

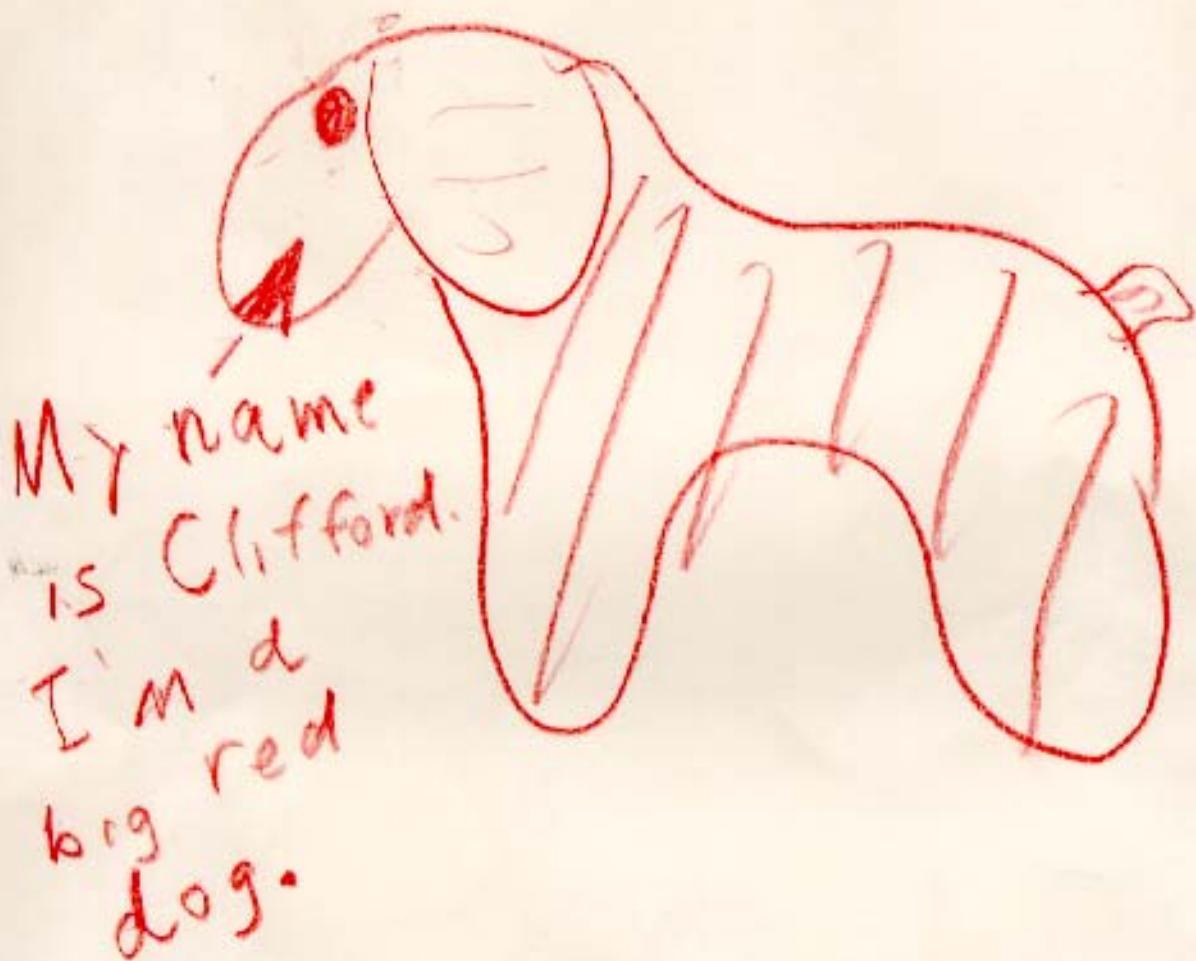
Chorus: Propositions, corollaries,
When they're just too deep,
I simply remember my favorite graphs
And then I go right to sleep.

Look! I
minimized
the zlop of
my favourite
graph song.



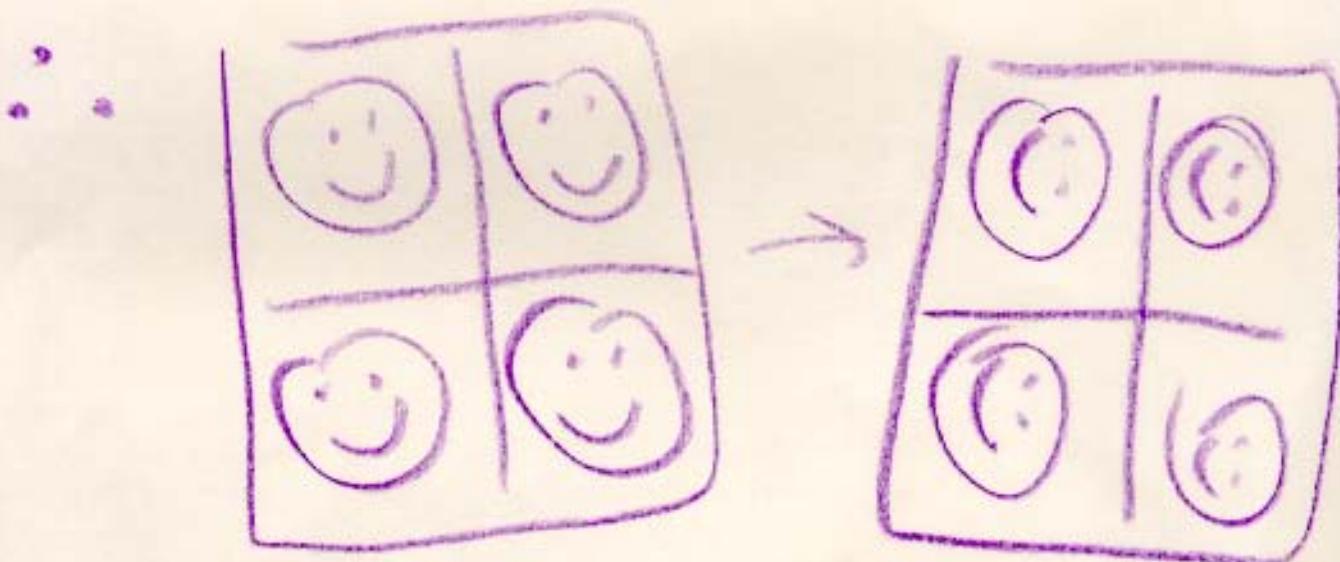
2.

Mamma always said:
2tupid is as stupid does.



3. This problem is the work of
Satan. It is very scan.
I was too afraid to do it.
Sorry.

4. I betcha that:



QED.

5. I am running this
algorithm with MGD. Will
report results...

much beer drinking
more analysis needed

in Ign is like a cheese log
if I'm hungry, call it
and have a slice
beer.

CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/cs373>

Homework 0 (due January 26, 1999 by the beginning of class)

Name:	
Net ID:	Alias:

Neatly print your name (first name first, with no comma), your network ID, and a short alias into the boxes above. 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, so your alias should not resemble your name (or your Net ID). If you do not give yourself an alias, you will be stuck with one we give you, no matter how much you hate it.

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

This homework tests your familiarity with the prerequisite material from CS 225 and CS 273 (and *their* prerequisites)—many of these problems appeared on homeworks and/or exams in those classes—primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.**

Undergrad/.75U Grad/1U Grad Problems

►1. [173/273]

- (a) Prove that any positive integer can be written as the sum of distinct powers of 2. (For example: $42 = 2^5 + 2^3 + 2^1$, $25 = 2^4 + 2^3 + 2^0$, $17 = 2^4 + 2^0$.)
- (b) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. (For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, $17 = F_7 + F_4 + F_2$.)
- (c) Prove that *any* integer can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. (For example: $42 = 3^4 - 3^3 - 3^2 - 3^1$, $25 = 3^3 - 3^1 + 3^0$, $17 = 3^3 - 3^2 - 3^0$.)

►2. [225/273] Sort the following functions from asymptotically smallest to largest, indicating ties if there are any: n , $\lg n$, $\lg \lg^* n$, $\lg^* \lg n$, $\lg^* n$, $n \lg n$, $\lg(n \lg n)$, $n^{n/\lg n}$, $n^{\lg n}$, $(\lg n)^n$, $(\lg n)^{\lg n}$, $2^{\sqrt{\lg n \lg \lg n}}$, 2^n , $n^{\lg \lg n}$, $\sqrt[1000]{n}$, $(1 + \frac{1}{1000})^n$, $(1 - \frac{1}{1000})^n$, $\lg^{1000} n$, $\lg^{(1000)} n$, $\log_{1000} n$, $\lg^n 1000$, 1 .

[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 as follows: $n \ll n^2 \equiv \binom{n}{2} \ll n^3$.]

3. [273/225] 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 (nontrivial) base cases. Extra credit will be given for more exact solutions.

- (a) $A(n) = A(n/2) + n$
- (b) $B(n) = 2B(n/2) + n$
- (c) $C(n) = 3C(n/2) + n$
- (d) $D(n) = \max_{n/3 < k < 2n/3} (D(k) + D(n - k) + n)$
- (e) $E(n) = \min_{0 < k < n} (E(k) + E(n - k) + 1)$
- (f) $F(n) = 4F(\lfloor n/2 \rfloor + 5) + n$
- (g) $G(n) = G(n - 1) + 1/n$
- *(h) $H(n) = H(n/2) + H(n/4) + H(n/6) + H(n/12) + n$ [Hint: $\frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{12} = 1$.]
- *(i) $I(n) = 2I(n/2) + n/\lg n$
- *(j) $J(n) = \frac{J(n - 1)}{J(n - 2)}$

►4. [273] Alice and Bob each have a fair n -sided die. Alice rolls her die once. Bob then repeatedly throws his die until the number he rolls is at least as big as the number Alice rolled. Each time Bob rolls, he pays Alice \$1. (For example, if Alice rolls a 5, and Bob rolls a 4, then a 3, then a 1, then a 5, the game ends and Alice gets \$4. If Alice rolls a 1, then no matter what Bob rolls, the game will end immediately, and Alice will get \$1.)

Exactly how much money does Alice expect to win at this game? Prove that your answer is correct. (If you have to appeal to “intuition” or “common sense”, your answer is probably wrong.)

►5. [225] George has a 26-node binary tree, with each node labeled by a unique letter of the alphabet. The preorder and postorder sequences of nodes are as follows:

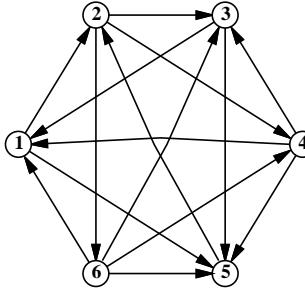
preorder: M N H C R S K W T G D X I Y A J P O E Z V B U L Q F
 postorder: C W T K S G R H D N A O E P J Y Z I B Q L F U V X M

Draw George’s binary tree.

Only 1U Grad Problems

►*1. [225/273] 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 $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$.

Practice Problems

1. [173/273] 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 .

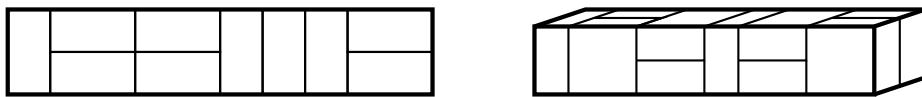
- (a) F_n is even if and only if n is divisible by 3.
- (b) $\sum_{i=0}^n F_i = F_{n+2} - 1$
- (c) $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$
- ★(d) If n is an integer multiple of m , then F_n is an integer multiple of F_m .

2. [225/273]

- (a) Prove that $2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor} / n = \Theta(n)$.
- (b) Is $2^{\lfloor \lg n \rfloor} = \Theta(2^{\lceil \lg n \rceil})$? Justify your answer.
- (c) Is $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta(2^{2^{\lceil \lg \lg n \rceil}})$? Justify your answer.

3. [273]

- (a) A *domino* is a 2×1 or 1×2 rectangle. How many different ways are there to completely fill a $2 \times n$ rectangle with n dominos?
- (b) A *slab* is a three-dimensional box with dimensions $1 \times 2 \times 2$, $2 \times 1 \times 2$, or $2 \times 2 \times 1$. How many different ways are there to fill a $2 \times 2 \times n$ box with n slabs? Set up a recurrence relation and give an *exact* closed-form solution.



A 2×10 rectangle filled with ten dominos, and a $2 \times 2 \times 10$ box filled with ten slabs.

4. [273] 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.]

5. [273/225] Prove that for any nonnegative parameters a and b , the following algorithms terminate and produce identical output.

SLOWEUCLID(a, b) :

if $b > a$
 return SLOWEUCLID(b, a)
else if $b == 0$
 return a
else
 return SLOWEUCLID($a, b - a$)

FASTEUCLID(a, b) :

if $b == 0$
 return a
else
 return FASTEUCLID($b, a \bmod b$)

CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 1 (due February 9, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
2. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
3. Justify the correctness of your algorithm, including termination if that is not obvious.
4. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

- ▶1. Consider the following sorting algorithm:

```

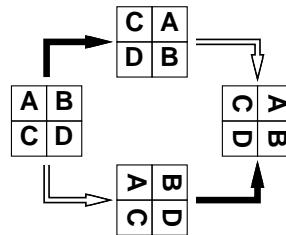
STUPIDSORT( $A[0..n - 1]$ ) :
  if  $n = 2$  and  $A[0] > A[1]$ 
    swap  $A[0] \leftrightarrow A[1]$ 
  else if  $n > 2$ 
     $m = \lceil 2n/3 \rceil$ 
    STUPIDSORT( $A[0..m - 1]$ )
    STUPIDSORT( $A[n - m..n - 1]$ )
    STUPIDSORT( $A[0..m - 1]$ )
  
```

- (a) Prove that STUPIDSORT actually sorts its input.
- (b) Would the algorithm still sort correctly if we replaced $m = \lceil 2n/3 \rceil$ with $m = \lfloor 2n/3 \rfloor$? Justify your answer.
- (c) State a recurrence (including the base case(s)) for the number of comparisons executed by STUPIDSORT.

- (d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?
 - *(e) Show that the number of *swaps* executed by STUPID SORT is at most $\binom{n}{2}$.

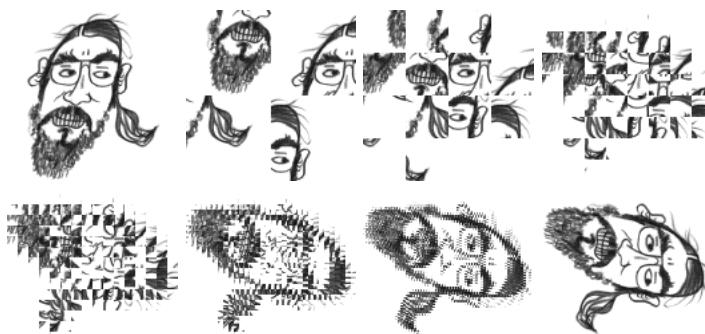
► 2. Some graphics hardware includes support for an operation called *blit*, or **block transfer**, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

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



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

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



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

- (a) Prove that both versions of the algorithm are correct.
 - (b) Exactly how many blits does the algorithm perform?
 - (c) What is the algorithm's running time if a $k \times k$ blit takes $O(k^2)$ time?
 - (d) What if a $k \times k$ blit takes only $O(k)$ time?

►3. Dynamic Programming: The Company Party

A company is planning a party for its employees. The organizers of the party want it to be a fun party, and so have assigned a ‘fun’ rating to every employee. The employees are organized into a strict hierarchy, i.e. a tree rooted at the president. There is one restriction, though, on the guest list to the party: both an employee and their immediate supervisor (parent in the tree) cannot both attend the party (because that would be no fun at all). Give an algorithm that makes a guest list for the party that maximizes the sum of the ‘fun’ ratings of the guests.

►4. Dynamic Programming: Longest Increasing Subsequence (LIS)

Give an $O(n^2)$ algorithm to find the longest increasing subsequence of a sequence of numbers. Note: the elements of the subsequence need not be adjacent in the sequence. For example, the sequence (1, 5, 3, 2, 4) has an LIS (1, 3, 4).

►5. Nut/Bolt Median

You are given a set of n nuts and n bolts of different sizes. Each nut matches exactly one bolt (and vice versa, of course). The sizes of the nuts and bolts are so similar that you cannot compare two nuts or two bolts to see which is larger. You can, however, check whether a nut is too small, too large, or just right for a bolt (and vice versa, of course).

In this problem, your goal is to find the median bolt (*i.e.*, the $\lfloor n/2 \rfloor$ th largest) as quickly as possible.

- (a) Describe an efficient deterministic algorithm that finds the median bolt. How many nut/bolt comparisons does your algorithm perform in the worst case?
- (b) Describe an efficient *randomized* algorithm that finds the median bolt.
 - i. State a recurrence for the expected number of nut/bolt comparisons your algorithm performs.
 - ii. What is the probability that your algorithm compares the i th largest bolt with the j th largest nut?
 - iii. What is the expected number of nut-bolt comparisons made by your algorithm? [Hint: Use your answer to either of the previous two questions.]

Only 1U Grad Problems

►1. You are at a political convention with n delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party a delegate belongs to. However, you can check whether any two delegates are in the *same* party or not by introducing them to each other. (Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.)

- (a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.
- *(b) Suppose exactly k political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Give an efficient algorithm that identifies a member of the plurality party.

- *(c) Suppose you don't know how many parties there are, but you do know that one party has a plurality, and at least p people in the plurality party are present. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)
- ★(d) Finally, suppose you don't know how many parties are represented at the convention, and you don't know how big the plurality is. Give an efficient algorithm to identify a member of the plurality party. How is the running time of your algorithm affected by the number of parties (k)? By the size of the plurality (p)?

Practice Problems

1. Second Smallest

Give an algorithm that finds the *second* smallest of n elements in at most $n + \lceil \lg n \rceil - 2$ comparisons. Hint: divide and conquer to find the smallest; where is the second smallest?

2. Linear in-place 0-1 sorting

Suppose that you have an array of records whose keys to be sorted consist only of 0's and 1's. Give a simple, linear-time $O(n)$ algorithm to sort the array in place (using storage of no more than constant size in addition to that of the array).

3. Dynamic Programming: Coin Changing

Consider the problem of making change for n cents using the least number of coins.

- (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.
- (b) Suppose that the available coins have the values c^0, c^1, \dots, c^k for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.
- (c) Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution, show why.
- (d) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.
- (e) Prove that, with only two coins a, b whose gcd is 1, the smallest value n for which change *can* be given for all values greater than or equal to n is $(a - 1)(b - 1)$.
- ★(f) For only three coins a, b, c whose gcd is 1, give an algorithm to determine the smallest value n for which change *can* be given for all values greater than n . (note: this problem is currently unsolved for $n > 4$.)

4. Dynamic Programming: Paragraph Justification

Consider the problem of printing a paragraph neatly on a printer (with fixed width font). The input text is a sequence of n words of lengths l_1, l_2, \dots, l_n . The line length is M (the maximum # of characters per line). We expect that the paragraph is left justified, that all first words on a line start at the leftmost position and that there is exactly one space between any two words on the same line. We want the uneven right ends of all the lines to be together as ‘neat’ as possible. Our criterion of neatness is that we wish to minimize the sum, over all lines except the last, of the cubes of the numbers of extra space characters at the ends of the lines. Note: if a printed line contains words i through j , then the number of spaces at the end of the line is $M - j + i - \sum_{k=i}^j l_k$.

- (a) Give a dynamic programming algorithm to do this.
- (b) Prove that if the neatness function is linear, a linear time greedy algorithm will give an optimum ‘neatness’.

5. Comparison of Amortized Analysis Methods

A sequence of n operations is performed on a data structure. The i th 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, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 2 (due Thu. Feb. 18, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
2. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
3. Justify the correctness of your algorithm, including termination if that is not obvious.
4. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

▶ 1. Faster Longest Increasing Subsequence (LIS)

Give an $O(n \log n)$ algorithm to find the longest increasing subsequence of a sequence of numbers. Hint: In the dynamic programming solution, you don't really have to look back at all previous items.

▶ 2. $\text{SELECT}(A, k)$

Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.

- (a) Show that a rotation in an augmented binary tree can be performed in constant time.
- (b) Describe an algorithm $\text{SCAPEGOATSELECT}(k)$ that selects the k th smallest item in an augmented scapegoat tree in $O(\log n)$ worst-case time. (The scapegoat trees presented in class were already augmented.)
- (c) Describe an algorithm $\text{SPLAYSELECT}(k)$ that selects the k th smallest item in an augmented splay tree in $O(\log n)$ amortized time.
- (d) Describe an algorithm $\text{TREAPSELECT}(k)$ that selects the k th smallest item in an augmented treap in $O(\log n)$ expected time.

►3. Scapegoat trees

- (a) Prove that only one tree gets rebalanced at any insertion.
- (b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree ($I(v) = \max(0, |\hat{v}| - |\check{v}|)$, where \hat{v} is the child of greater height and \check{v} the child of lesser height, $|v|$ is the number of nodes in subtree v , and perfectly balanced means each subtree has as close to half the leaves as possible and is perfectly balanced itself).
- *(c) Show that you can rebuild a fully balanced binary tree in $O(n)$ time using only $O(1)$ additional memory.

►4. Memory Management

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

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

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do not use the potential method (it makes it much more difficult).

Only 1U Grad Problems

▷1. Detecting overlap

- (a) You are given a list of ranges represented by min and max (e.g. [1,3], [4,5], [4,9], [6,8], [7,10]). Give an $O(n \log n)$ -time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.
- (b) You are given a list of rectangles represented by min and max x - and y - coordinates. Give an $O(n \log n)$ -time algorithm that decides whether or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.

Practice Problems

1. Amortization

- (a) Modify the binary double-counter (see class notes Feb. 2) to support a new operation Sign, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

[Hint: Suppose p is the number of significant bits in P , and n is the number of significant bits in N . For example, if $P = 17 = 10001_2$ and $N = 0$, then $p = 5$ and $n = 0$. Then $p - n$ always has the same sign as $P - N$. Assume you can update p and n in $O(1)$ time.]

- *(b) Do the same but now you can't assume that p and n can be updated in $O(1)$ time.

*2. Amortization

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 i th least significant fit indicates whether the sum includes the i th 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. This is *not* the same representation as on Homework 0.]

3. Rotations

- (a) Show that it is possible to transform any n -node binary search tree into any other n -node binary search tree using at most $2n - 2$ rotations.
- *(b) Use fewer than $2n - 2$ rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most $2n - 6$ rotations, and there are pairs of trees that are $2n - 10$ rotations apart. These are the best bounds known.

4. Fibonacci Heaps: SECONDMIN

We want to find the second smallest of a set efficiently.

- (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.
- *(b) Modify the Fibonacci Heap data structure to implement SECONDMIN in constant time.
5. Give an efficient implementation of the operation **Fib-Heap-Change-Key**(H, x, k), which changes the key of a node x in a Fibonacci heap H to the value k . The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which k is greater than, less than, or equal to $\text{key}[x]$.

CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 2 (due Thu. Feb. 18, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
2. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
3. Justify the correctness of your algorithm, including termination if that is not obvious.
4. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

▶ 1. Faster Longest Increasing Subsequence (LIS)

Give an $O(n \log n)$ algorithm to find the longest increasing subsequence of a sequence of numbers. Hint: In the dynamic programming solution, you don't really have to look back at all previous items.

▶ 2. $\text{SELECT}(A, k)$

Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.

- (a) Show that a rotation in an augmented binary tree can be performed in constant time.
- (b) Describe an algorithm $\text{SCAPEGOATSELECT}(k)$ that selects the k th smallest item in an augmented scapegoat tree in $O(\log n)$ worst-case time. (The scapegoat trees presented in class were already augmented.)
- (c) Describe an algorithm $\text{SPLAYSELECT}(k)$ that selects the k th smallest item in an augmented splay tree in $O(\log n)$ amortized time.
- (d) Describe an algorithm $\text{TREAPSELECT}(k)$ that selects the k th smallest item in an augmented treap in $O(\log n)$ expected time.

►3. Scapegoat trees

- (a) Prove that only one tree gets rebalanced at any insertion.
- (b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree ($I(v) = \max(0, |\hat{v}| - |\check{v}|)$, where \hat{v} is the child of greater height and \check{v} the child of lesser height, $|v|$ is the number of nodes in subtree v , and perfectly balanced means each subtree has as close to half the leaves as possible and is perfectly balanced itself).
- *(c) Show that you can rebuild a fully balanced binary tree in $O(n)$ time using only $O(1)$ additional memory.

►4. Memory Management

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

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

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do not use the potential method (it makes it much more difficult).

Only 1U Grad Problems

▷1. Detecting overlap

- (a) You are given a list of ranges represented by min and max (e.g. [1,3], [4,5], [4,9], [6,8], [7,10]). Give an $O(n \log n)$ -time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.
- (b) You are given a list of rectangles represented by min and max x - and y - coordinates. Give an $O(n \log n)$ -time algorithm that decides whether or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.

Practice Problems

1. Amortization

- (a) Modify the binary double-counter (see class notes Feb. 2) to support a new operation Sign, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

[Hint: Suppose p is the number of significant bits in P , and n is the number of significant bits in N . For example, if $P = 17 = 10001_2$ and $N = 0$, then $p = 5$ and $n = 0$. Then $p - n$ always has the same sign as $P - N$. Assume you can update p and n in $O(1)$ time.]

- *(b) Do the same but now you can't assume that p and n can be updated in $O(1)$ time.

*2. Amortization

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 i th least significant fit indicates whether the sum includes the i th 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. This is *not* the same representation as on Homework 0.]

3. Rotations

- (a) Show that it is possible to transform any n -node binary search tree into any other n -node binary search tree using at most $2n - 2$ rotations.
- *(b) Use fewer than $2n - 2$ rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most $2n - 6$ rotations, and there are pairs of trees that are $2n - 10$ rotations apart. These are the best bounds known.

4. Fibonacci Heaps: SECONDMIN

We want to find the second smallest of a set efficiently.

- (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.
- *(b) Modify the Fibonacci Heap data structure to implement SECONDMIN in constant time.
5. Give an efficient implementation of the operation **Fib-Heap-Change-Key**(H, x, k), which changes the key of a node x in a Fibonacci heap H to the value k . The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which k is greater than, less than, or equal to $\text{key}[x]$.

CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 3 (due Thu. Mar. 11, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. (New!) If not already done, model the problem appropriately. Often the problem is stated in real world terms; give a more rigorous description of the problem. This will help you figure out what is assumed (what you know and what is arbitrary, what operations are and are not allowed), and find the tools needed to solve the problem.
2. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
3. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
4. Justify the correctness of your algorithm, including termination if that is not obvious.
5. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

▶ 1. Hashing

- (a) (2 pts) Consider an open-address hash table with uniform hashing and a load factor $\alpha = 1/2$. What is the expected number of probes in an unsuccessful search? Successful search?
- (b) (3 pts) Let the hash function for a table of size m be

$$h(x) = \lfloor Ax \rfloor \bmod m$$

where $A = \hat{\phi} = \frac{\sqrt{5}-1}{2}$. Show that this gives the best possible spread, i.e. if the x are hashed in order, $x+1$ will be hashed in the largest remaining contiguous interval.

►2. (5 pts) Euler Tour:

Given an **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.

►3. (5 pts) 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, *and* any intermediate files in the compilation that depend on those changed. Design an algorithm to recompile only those necessary.

►4. (5 pts) Shortest Airplane Trip:

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. Hint: rather than modify Dijkstra’s algorithm, modify the data. The time is from departure to arrival at the destination, so it will include layover time (time waiting for a connecting flight).

►5. (9 pts, 3 each) Minimum Spanning Tree changes Suppose you have a graph G and an MST of that graph (i.e. the MST has already been constructed).

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

Only 1U Grad Problems

▷1. 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.

Practice Problems

1. The incidence matrix of an undirected graph $G = (V, E)$ is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = \begin{cases} 1 & (i, j) \in E, \\ 0 & (i, j) \notin E. \end{cases}$$

- (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^TB 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).

2. $o(V^2)$ 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.

3. 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 .

4. 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 .

5. Minimum Spanning Tree:

Suppose all edge weights in a graph G are equal. Give an algorithm to compute an MST.

6. 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} .

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

- (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, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 4 (due Thu. Apr. 1, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. (New!) If not already done, model the problem appropriately. Often the problem is stated in real world terms; give a more rigorous description of the problem. This will help you figure out what is assumed (what you know and what is arbitrary, what operations are and are not allowed), and find the tools needed to solve the problem.
2. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
3. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
4. Justify the correctness of your algorithm, including termination if that is not obvious.
5. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

▶ 1. (5 pts total) Collinearity

Give an $O(n^2 \log n)$ algorithm to determine whether any three points of a set of n points are collinear. Assume two dimensions and exact arithmetic.

▶ 2. (4 pts, 2 each) Convex Hull Recurrence

Consider the following generic recurrence for convex hull algorithms that divide and conquer:

$$T(n, h) = T(n_1, h_1) + T(n_2, h_2) + O(n)$$

where $n \geq n_1 + n_2$, $h = h_1 + h_2$ and $n \geq h$. This means that the time to compute the convex hull is a function of both n , the number of input points, and h , the number of convex hull vertices. The splitting and merging parts of the divide-and-conquer algorithm take $O(n)$ time. When n is a constant, $T(n, h)$ is $O(1)$, but when h is a constant, $T(n, h)$ is $O(n)$. Prove that for both of the following restrictions, the solution to the recurrence is $O(n \log h)$:

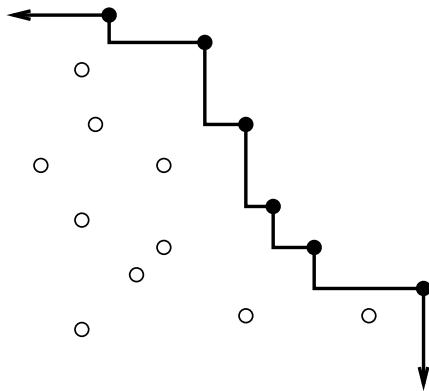
- (a) $h_1, h_2 < \frac{3}{4}h$
- (b) $n_1, n_2 < \frac{3}{4}n$

►3. (5 pts) Circle Intersection

Give an $O(n \log n)$ algorithm to test whether any two circles in a set of size n intersect.

►4. (5 pts total) Staircases

You are given a set of points in the first quadrant. A *left-up* point of this set is defined to be a point that has no points both greater than it in both coordinates. The left-up subset of a set of points then forms a *staircase* (see figure).



- (a) (3 pts) Give an $O(n \log n)$ algorithm to find the staircase of a set of points.

- (b) (2 pts) Assume that points are chosen uniformly at random within a rectangle. What is the average number of points in a staircase? Justify. Hint: you will be able to give an exact answer rather than just asymptotics. You have seen the same analysis before.

Only 1U Grad Problems

►1. (6 pts, 2 each) Ghostbusters and Ghosts

A group of n ghostbusters is battling n ghosts. Each ghostbuster can shoot a single energy beam at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits a ghost. The ghostbusters must all fire at the same time and no two energy beams may cross. The positions of the ghosts and ghostbusters is fixed in the plane (assume that no three points are collinear).

- (a) Prove that for any configuration ghosts and ghostbusters there exists such a non-crossing matching.
- (b) Show that there exists a line passing through one ghostbuster and one ghost such that the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Give an efficient algorithm to find such a line.
- (c) Give an efficient divide and conquer algorithm to pair ghostbusters and ghosts so that no two streams cross.

Practice Problems

1. Basic Computation (assume two dimensions and *exact* arithmetic)
 - (a) Intersection: Extend the basic algorithm to determine if two line segments intersect by taking care of *all* degenerate cases.
 - (b) Simplicity: Give an $O(n \log n)$ algorithm to determine whether an n -vertex polygon is simple.
 - (c) Area: Give an algorithm to compute the area of a simple n -polygon (not necessarily convex) in $O(n)$ time.
 - (d) Inside: Give an algorithm to determine whether a point is within a simple n -polygon (not necessarily convex) in $O(n)$ time.

2. External Diagonals and Mouths
 - (a) A pair of polygon vertices defines an *external diagonal* if the line segment between them is completely outside the polygon. Show that every nonconvex polygon has at least one external diagonal.
 - (b) Three consecutive polygon vertices p, q, r form a *mouth* if p and r define an external diagonal. Show that every nonconvex polygon has at least one mouth.

3. On-Line Convex Hull

We are given the set of points one point at a time. After receiving each point, we must compute the convex hull of all those points so far. Give an algorithm to solve this problem in $O(n^2)$ (We could obviously use Graham's scan n times for an $O(n^2 \log n)$ algorithm). Hint: How do you maintain the convex hull?

4. Another On-Line Convex Hull Algorithm
 - (a) Given an n -polygon and a point outside the polygon, give an algorithm to find a tangent.
 - *(b) Suppose you have found both tangents. Give an algorithm to remove the points from the polygon that are within the angle formed by the tangents (as segments!) and the opposite side of the polygon.
 - (c) Use the above to give an algorithm to compute the convex hull on-line in $O(n \log n)$

- ★5. Order of the size of the convex hull

The convex hull on $n \geq 3$ points can have anywhere from 3 to n points. The average case depends on the distribution.

 - (a) Prove that if a set of points is chosen randomly within a given rectangle, then the average size of the convex hull is $O(\log n)$.
 - (b) Prove that if a set of points is chosen randomly within a given circle, then the average size of the convex hull is $O(\sqrt{n})$.

CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 5 (due Thu. Apr. 22, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: You will be held accountable for the appropriate responses for answers (e.g. give models, proofs, analyses, etc)

Undergrad/.75U Grad/1U Grad Problems

- ▶1. (5 pts) Show how to find the occurrences of pattern P in text T by computing the prefix function of the string PT (the concatenation of P and T).

- ▶2. (10 pts total) Fibonacci strings and KMP matching
Fibonacci strings are defined as follows:

$$F_1 = \text{"b"}, \quad F_2 = \text{"a"}, \quad \text{and } F_n = F_{n-1}F_{n-2}, (n > 2)$$

where the recursive rule uses concatenation of strings, so F_2 is "ab", F_3 is "aba". Note that the length of F_n is the n th Fibonacci number.

- (a) (2 pts) Prove that in any Fibonacci string there are no two b's adjacent and no three a's.
- (b) (2 pts) Give the unoptimized and optimized 'prefix' (fail) function for F_7 .
- (c) (3 pts) Prove that, in searching for a Fibonacci string of length m using unoptimized KMP, it may shift up to $\lceil \log_\phi m \rceil$ times, where $\phi = (1 + \sqrt{5})/2$, is the golden ratio. (Hint: Another way of saying the above is that we are given the string F_n and we may have to shift n times. Find an example text T that gives this number of shifts).
- (d) (3 pts) What happens here when you use the optimized prefix function? Explain.

- ▶3. (5 pts) Prove that finding the second smallest of n elements takes $n + \lceil \lg n \rceil - 2$ comparisons in the worst case. Prove for both upper and lower bounds. Hint: find the (first) smallest using an elimination tournament.

- ▶4. (4 pts, 2 each) Lower Bounds on Adjacency Matrix Representations of Graphs
 - (a) Prove that the time to determine if an undirected graph has a cycle is $\Omega(V^2)$.

- (b) Prove that the time to determine if there is a path between two nodes in an undirected graph is $\Omega(V^2)$.

Only 1U Grad Problems

- ▷ 1. (5 pts) Prove that $\lceil 3n/2 \rceil - 2$ comparisons are necessary in the worst case to find both the minimum and maximum of n numbers. Hint: Consider how many are potentially either the min or max.

Practice Problems

1. String matching with wild-cards

Suppose you have an alphabet for patterns that includes a ‘gap’ or wild-card character; any length string of any characters can match this additional character. For example if ‘*’ is the wild-card, then the pattern ‘foo*bar*nad’ can be found in ‘foofoowangbarnad’. Modify the computation of the prefix function to correctly match strings using KMP.

2. Prove that there is no comparison sort whose running time is linear for at least $1/2$ of the $n!$ inputs of length n . What about at least $1/n$? What about at least $1/2^n$?

3. Prove that $2n - 1$ comparisons are necessary in the worst case to merge two sorted lists containing n elements each.

4. Find asymptotic upper and lower bounds to $\lg(n!)$ without Stirling’s approximation (Hint: use integration).

5. Given a sequence of n elements of n/k blocks (k elements per block) all elements in a block are less than those to the right in sequence, show that you cannot have the whole sequence sorted in better than $\Omega(n \lg k)$. Note that the entire sequence would be sorted if each of the n/k blocks were individually sorted in place. Also note that combining the lower bounds for each block is not adequate (that only gives an upper bound).

6. Some elementary reductions

- (a) Prove that if you can decide whether a graph G has a clique of size k (or less) then you can decide whether a graph G' has an independent set of size k (or more).

- (b) Prove that if you can decide whether one graph G_1 is a subgraph of another graph G_2 then you can decide whether a graph G has a clique of size k (or less).

7. There is no Proof but We are pretty Sure

Justify (prove) the following logical rules of inference:

- (a) Classical - If $a \rightarrow b$ and a hold, then b holds.

- (b) Fuzzy - Prove: If $a \rightarrow b$ holds, and a holds with probability p , then b holds with probability less than p . Assume all probabilities are independent.

- (c) Give formulas for computing the probabilities of the fuzzy logical operators ‘and’, ‘or’, ‘not’, and ‘implies’, and fill out truth tables with the values T (true, $p = 1$), L (likely, $p = 0.9$), M (maybe, $p = 0.5$), N (not likely, $p = 0.1$), and F (false, $p = 0$).

- (d) If you have a poly time (algorithmic) reduction from problem B to problem A (i.e. you can solve B using A with a poly time conversion), and it is very unlikely that A has better than lower bound $\Omega(2^n)$ algorithm, what can you say about problem A . Hint: a solution to A implies a solution to B .

CS 373: Combinatorial Algorithms, Spring 1999

Midterm 1 (February 23, 1999)

Name:

Net ID:

Alias:

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your $8\frac{1}{2}'' \times 11''$ cheat sheet, please leave it at the front of the classroom.

- **Don't panic!**

- Print your name, netid, and alias in the boxes above, and print your name at the top of every page.
- **Answer four of the five questions on the exam.** Each question is worth 10 points. If you answer more than four questions, the one with the lowest score will be ignored. **1-unit graduate students must answer question #5.**
- Please write your answers on the front of the exam pages. Use the backs of the pages as scratch paper. Let us know if you need more paper.
- Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
- Write *something* down for every problem. Don't panic and erase large chunks of work. Even if you think it's nonsense, it might be worth partial credit.

- **Don't panic!**

#	Score	Grader
1		
2		
3		
4		
5		

1. Multiple Choice

Every question below has one of the following answers.

- (a) $\Theta(1)$ (b) $\Theta(\log n)$ (c) $\Theta(n)$ (d) $\Theta(n \log n)$ (e) $\Theta(n^2)$

For each question, write the letter that corresponds to your answer. You do not need to justify your answer. Each correct answer earns you 1 point, but each incorrect answer *costs* you $\frac{1}{2}$ point. (You cannot score below zero.)

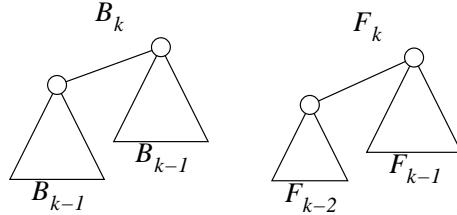
- What is $\sum_{i=1}^n i$?
- What is $\sum_{i=1}^n \frac{1}{i}$?
- What is the solution of the recurrence $T(n) = T(\sqrt{n}) + n$?
- What is the solution of the recurrence $T(n) = T(n - 1) + \lg n$?
- What is the solution of the recurrence $T(n) = 2T\left(\lceil \frac{n+27}{2} \rceil\right) + 5n - 7\sqrt{\lg n} + \frac{1999}{n}$?
- The amortized time for inserting one item into an n -node splay tree is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty splay tree?
- The expected time for inserting one item into an n -node randomized treap is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty treap?
- What is the worst-case running time of randomized quicksort?
- How many bits are there in the binary representation of the n th Fibonacci number?
- What is the worst-case cost of merging two arbitrary splay trees with n items total into a single splay tree with n items.
- Suppose you correctly identify three of the answers to this question as obviously wrong. If you pick one of the two remaining answers at random, what is your expected score for this problem?

2. (a) [5 pt] Recall that a binomial tree of order k , denoted B_k , is defined recursively as follows. B_0 is a single node. For any $k > 0$, B_k consists of two copies of B_{k-1} linked together.

Prove that the degree of any node in a binomial tree is equal to its height.

- (b) [5 pt] Recall that a Fibonacci tree of order k , denoted F_k , is defined recursively as follows. F_1 and F_2 are both single nodes. For any $k > 2$, F_k consists of an F_{k-2} linked to an F_{k-1} .

Prove that for any node v in a Fibonacci tree, $\text{height}(v) = \lceil \text{degree}(v)/2 \rceil$.



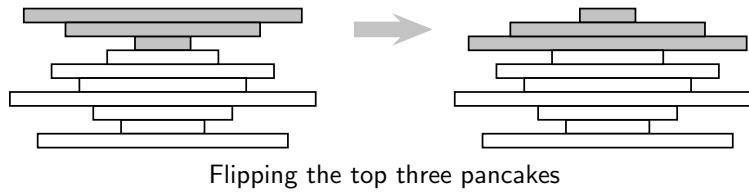
Recursive definitions of binomial trees and Fibonacci trees.

3. Consider the following randomized algorithm for computing the smallest element in an array.

```
RANDOMMIN( $A[1..n]$ ):  
     $min \leftarrow \infty$   
    for  $i \leftarrow 1$  to  $n$  in random order  
        if  $A[i] < min$   
             $min \leftarrow A[i]$  (*)  
    return  $min$ 
```

- (a) [1 pt] In the worst case, how many times does RANDOMMIN execute line (*)?
- (b) [3 pt] What is the probability that line (*) is executed during the n th iteration of the for loop?
- (c) [6 pt] What is the exact expected number of executions of line (*)? (A correct $\Theta()$ bound is worth 4 points.)

4. Suppose we have a stack of n pancakes of different sizes. We want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation we can perform is a *flip* — insert a spatula under the top k pancakes, for some k between 1 and n , and flip them all over.



- (a) [3 pt] Describe an algorithm to sort an arbitrary stack of n pancakes.
- (b) [3 pt] Prove that your algorithm is correct.
- (c) [2 pt] Exactly how many flips does your algorithm perform in the worst case? (A correct $\Theta()$ bound is worth one point.)
- (d) [2 pt] Suppose one side of each pancake is burned. Exactly how many flips do you need to sort the pancakes *and* have the burned side of every pancake on the bottom? (A correct $\Theta()$ bound is worth one point.)

5. You are given an array $A[1..n]$ of integers. Describe and analyze an algorithm that finds the largest sum of elements in a contiguous subarray $A[i..j]$.

For example, if the array contains the numbers $(-6, 12, -7, 0, 14, -7, 5)$, then the largest sum is $19 = 12 - 7 + 0 + 14$.

-6	12	-7	0	14	-7	5
19						

To get full credit, your algorithm must run in $\Theta(n)$ time — there are at least three different ways to do this. An algorithm that runs in $\Theta(n^2)$ time is worth 7 points.

CS 373: Combinatorial Algorithms, Spring 1999

Midterm 2 (April 6, 1999)

Name:

Net ID:

Alias:

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your $8\frac{1}{2}'' \times 11''$ cheat sheet, please leave it at the front of the classroom.

-
- Don't panic!
 - Print your name, netid, and alias in the boxes above, and print your name at the top of every page.
 - Answer four of the five questions on the exam. Each question is worth 10 points. If you answer more than four questions, the one with the lowest score will be ignored. **1-unit graduate students must answer question #5.**
 - Please write your answers on the front of the exam pages. You can use the backs of the pages as scratch paper. Let us know if you need more paper.
 - Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
 - Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
 - Write *something* down for every problem. Don't panic and erase large chunks of work. Even if you think it's nonsense, it might be worth partial credit.
 - Don't panic!
-

#	Score	Grader
1		
2		
3		
4		
5		

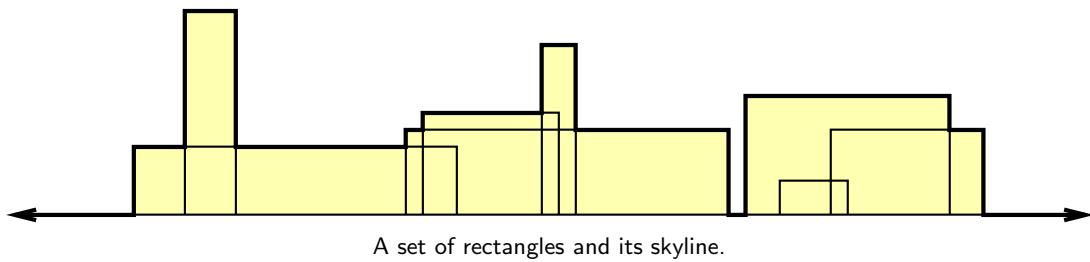
1. Bipartite Graphs

A graph (V, E) is *bipartite* if the vertices V can be partitioned into two subsets L and R , such that every edge has one vertex in L and the other in R .

- (a) Prove that every tree is a bipartite graph.
- (b) Describe and analyze an efficient algorithm that determines whether a given connected, undirected graph is bipartite.

2. Manhattan Skyline

The purpose of the following problem is to compute the outline of a projection of rectangular buildings. You are given the height, width, and left x -coordinate of n rectangles. The bottom of each rectangle is on the x -axis. Describe and analyze an efficient algorithm to compute the vertices of the “skyline”.



3. Least Cost Vertex Weighted Path

Suppose you want to drive from Champaign to Los Angeles via a network of roads connecting cities. You don't care how long it takes, how many cities you visit, or how much gas you use. All you care about is how much money you spend on food. Each city has a possibly different, but fixed, value for food.

More formally, you are given a directed graph $G = (V, E)$ with nonnegative weights on the vertices $w: V \rightarrow \mathbb{R}^+$, a source vertex $s \in V$, and a target vertex $t \in V$. Describe and analyze an efficient algorithm to find a minimum-weight path from s to t . [Hint: Modify the graph.]

4. Union-Find with Alternate Rule

In the UNION-FIND data structure described in CLR and in class, each set is represented by a rooted tree. The UNION algorithm, given two sets, decides which set is to be the parent of the other by comparing their ranks, where the rank of a set is an upper bound on the height of its tree.

Instead of rank, we propose using the *weight* of the set, which is just the number of nodes in the set. Here's the modified UNION algorithm:

```
UNION(A, B):
    if weight(A) > weight(B)
        parent(B) ← A
        weight(A) ← weight(A) + weight(B)
    else
        parent(A) ← B
        weight(B) ← weight(A) + weight(B)
```

Prove that if we use this method, then after any sequence of n MAKESETS, UNIONS, and FINDS (with path compression), the *height* of the tree representing any set is $O(\log n)$.

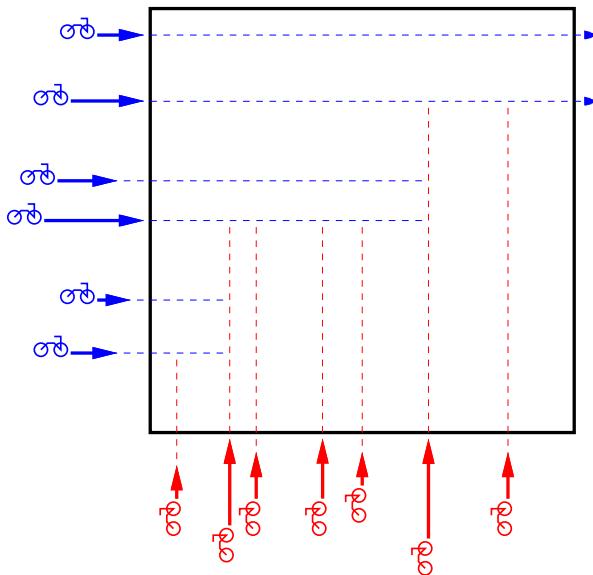
[Hint: First prove it without path compression, and then argue that path compression doesn't matter (for this problem).]

5. Motorcycle Collision

One gang, Hell's Ordinates, start west of the arena facing directly east; the other, The Vicious Abscissas of Death, start south of the arena facing due north. All the motorcycles start moving simultaneously at a prearranged signal. Each motorcycle moves at a fixed speed—no speeding up, slowing down, or turning is allowed. Each motorcycle leaves an oil trail behind it. If another motorcycle crosses that trail, it falls over and stops leaving a trail.

More formally, we are given two sets H and V , each containing n motorcycles. Each motorcycle is represented by three numbers (s, x, y) : its speed and the x - and y -coordinates of its initial location. Bikes in H move horizontally; bikes in V move vertically.

Assume that the bikes are infinitely small points, that the bike trails are infinitely thin lie segments, that a bike crashes stops *exactly* when it hits a oil trail, and that no two bikes collide with each other.



Two sets of motorcycles and the oil trails they leave behind.

- (a) Solve the case $n = 1$. Given only two motorcycles moving perpendicular to each other, determine which one of them falls over and where in $O(1)$ time.
- (b) Describe an efficient algorithm to find the set of all points where motorcycles fall over.

5. Motorcycle Collision (continued)

Incidentally, the movie *Tron* is being shown during Roger Ebert's Forgotten Film Festival at the Virginia Theater in Champaign on April 25. Get your tickets now!

CS 373: Combinatorial Algorithms, Spring 1999

Final Exam (May 7, 1999)

Name:

Net ID:

Alias:

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your two $8\frac{1}{2}'' \times 11''$ cheat sheets, please leave it at the front of the classroom.

-
- Print your name, netid, and alias in the boxes above, and print your name at the top of every page.
 - **Answer six of the seven questions on the exam.** Each question is worth 10 points. If you answer every question, the one with the lowest score will be ignored. **1-unit graduate students must answer question #7.**
 - Please write your answers on the front of the exam pages. Use the backs of the pages as scratch paper. Let us know if you need more paper.
 - Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
 - Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
 - Write *something* down for every problem. Don't panic and erase large chunks of work. Even if you think it's nonsense, it might be worth partial credit.
-

#	Score	Grader
1		
2		
3		
4		
5		
6		
7		

1. Short Answer

sorting	induction	Master theorem	divide and conquer
randomized algorithm	amortization	brute force	hashing
binary search	depth-first search	splay tree	Fibonacci heap
convex hull	sweep line	minimum spanning tree	shortest paths
shortest path	adversary argument	NP-hard	reduction
string matching	evasive graph property	dynamic programming	H_n

Choose from the list above the best method for solving each of the following problems. We do *not* want complete solutions, just a short description of the proper solution technique! Each item is worth 1 point.

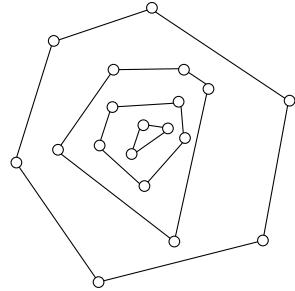
- (a) Given a Champaign phone book, find your own phone number.
- (b) Given a collection of n rectangles in the plane, determine whether any two intersect in $O(n \log n)$ time.
- (c) Given an undirected graph G and an integer k , determine if G has a complete subgraph with k edges.
- (d) Given an undirected graph G , determine if G has a triangle — a complete subgraph with three vertices.
- (e) Prove that any n -vertex graph with minimum degree at least $n/2$ has a Hamiltonian cycle.
- (f) Given a graph G and three distinguished vertices u , v , and w , determine whether G contains a path from u to v that passes through w .
- (g) Given a graph G and two distinguished vertices u and v , determine whether G contains a path from u to v that passes through at most 17 edges.
- (h) Solve the recurrence $T(n) = 5T(n/17) + O(n^{4/3})$.
- (i) Solve the recurrence $T(n) = 1/n + T(n - 1)$, where $T(0) = 0$.
- (j) Given an array of n integers, find the integer that appears most frequently in the array.

- (a) _____ (f) _____
- (b) _____ (g) _____
- (c) _____ (h) _____
- (d) _____ (i) _____
- (e) _____ (j) _____

2. Convex Layers

Given a set Q of points in the plane, define the *convex layers* of Q inductively as follows: The first convex layer of Q is just the convex hull of Q . For all $i > 1$, the i th convex layer is the convex hull of Q after the vertices of the first $i - 1$ layers have been removed.

Give an $O(n^2)$ -time algorithm to find all convex layers of a given set of n points. [Partial credit for a correct slower algorithm; extra credit for a correct faster algorithm.]



A set of points with four convex layers.

3. Suppose you are given an array of n numbers, sorted in increasing order.

- (a) [3 pts] Describe an $O(n)$ -time algorithm for the following problem:

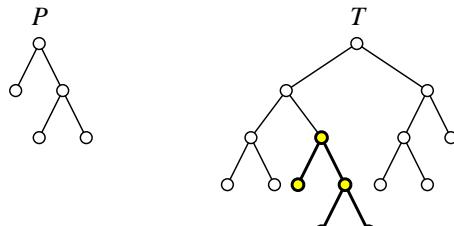
Find two numbers from the list that add up to zero, or report that there is no such pair.
In other words, find two numbers a and b such that $a + b = 0$.

- (b) [7 pts] Describe an $O(n^2)$ -time algorithm for the following problem:

Find *three* numbers from the list that add up to zero, or report that there is no such triple.
In other words, find three numbers a , b , and c , such that $a + b + c = 0$. [Hint: Use something similar to part (a) as a subroutine.]

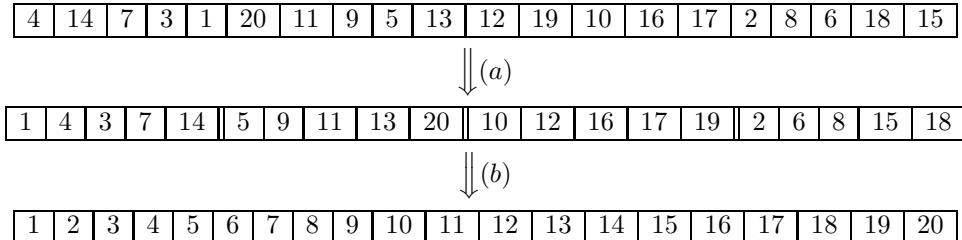
4. Pattern Matching

- (a) [4 pts] A *cyclic rotation* of a string is obtained by chopping off a prefix and gluing it at the end of the string. For example, ALGORITHM is a cyclic shift of RITHMALGO. Describe and analyze an algorithm that determines whether one string $P[1..m]$ is a cyclic rotation of another string $T[1..n]$.
- (b) [6 pts] Describe and analyze an algorithm that decides, given any two binary trees P and T , whether P equals a subtree of T . [Hint: First transform both trees into strings.]



5. Two-stage Sorting

- (a) [1 pt] Suppose we are given an array $A[1..n]$ of distinct integers. Describe an algorithm that splits A into n/k subarrays, each with k elements, such that the elements of each subarray $A[(i-1)k+1..ik]$ are sorted. Your algorithm should run in $O(n \log k)$ time.
- (b) [2 pts] Given an array $A[1..n]$ that is already split into n/k sorted subarrays as in part (a), describe an algorithm that sorts the entire array in $O(n \log(n/k))$ time.
- (c) [3 pts] Prove that your algorithm from part (a) is optimal.
- (d) [4 pts] Prove that your algorithm from part (b) is optimal.



6. SAT Reduction

Suppose you are have a black box that magically solves SAT (the formula satisfiability problem) in constant time. That is, given a boolean formula of variables and logical operators (\wedge, \vee, \neg), the black box tells you, in constant time, whether or not the formula can be satisfied. Using this black box, design and analyze a **polynomial-time** algorithm that computes an assignment to the variables that satisfies the formula.

7. Knapsack

You're hiking through the woods when you come upon a treasure chest filled with objects. Each object has a different size, and each object has a price tag on it, giving its value. There is no correlation between an object's size and its value. You want to take back as valuable a subset of the objects as possible (in one trip), but also making sure that you will be able to carry it in your knapsack which has a limited size.

In other words, you have an integer capacity K and a target value V , and you want to decide whether there is a subset of the objects whose total size is *at most* K and whose total value is *at least* V .

- (a) [5 pts] Show that this problem is NP-hard. [Hint: Restate the problem more formally, then reduce from the NP-hard problem PARTITION: Given a set S of nonnegative integers, is there a partition of S into disjoint subsets A and B (where $A \cup B = S$) whose sums are equal, *i.e.*, $\sum_{a \in A} a = \sum_{b \in B} b$.]
- (b) [5 pts] Describe and analyze a dynamic programming algorithm to solve the knapsack problem in $O(nK)$ time. Prove your algorithm is correct.

CS 373: Combinatorial Algorithms, Fall 2000

Homework 0, due August 31, 2000 at the beginning of class

Name:	
Net ID:	Alias:

Neatly print your name (first name first, with no comma), your network ID, and a short alias into the boxes above. **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, read the Homework Instructions and FAQ on the CS 373 course web page (<http://www-courses.cs.uiuc.edu/~cs373/hw/faq.html>), and then check the box below. This web page gives instructions on how to write and submit homeworks—staple your solutions together in order, write your name and netID on every page, don't turn in source code, analyze everything, use good English and good logic, and so forth.

I have read the CS 373 Homework Instructions and FAQ.

This homework tests your familiarity with the prerequisite material from CS 173, CS 225, and CS 273—many of these problems have appeared on homeworks or exams in those classes—primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Parberry and Chapters 1–6 of CLR should be sufficient review, but you may want to consult other texts as well.

Required Problems

- Sort the following 25 functions from asymptotically smallest to asymptotically largest, indicating ties if there are any:

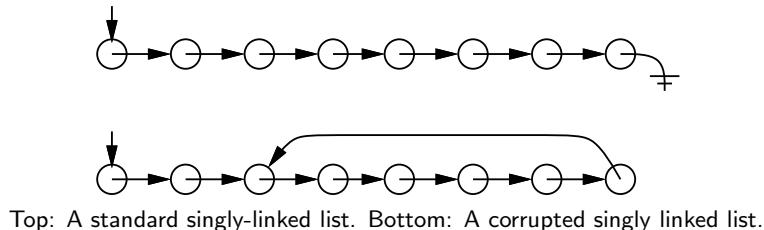
$$\begin{array}{ccccc} 1 & n & n^2 & \lg n & \lg(n \lg n) \\ \lg^* n & \lg^* 2^n & 2^{\lg^* n} & \lg \lg^* n & \lg^* \lg n \\ n^{\lg n} & (\lg n)^n & (\lg n)^{\lg n} & n^{1/\lg n} & n^{\lg \lg n} \\ \log_{1000} n & \lg^{1000} n & \lg^{(1000)} n & \left(1 + \frac{1}{n}\right)^n & n^{1/1000} \end{array}$$

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. (a) Prove that any positive integer can be written as the sum of distinct powers of 2. For example: $42 = 2^5 + 2^3 + 2^1$, $25 = 2^4 + 2^3 + 2^0$, $17 = 2^4 + 2^0$. [Hint: “Write the number in binary” is *not* a proof; it just restates the problem.]
- (b) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, $17 = F_7 + F_4 + F_2$.
- (c) Prove that *any* integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example: $42 = 3^4 - 3^3 - 3^2 - 3^1$, $25 = 3^3 - 3^1 + 3^0$, $17 = 3^3 - 3^2 - 3^0$.
3. 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. If no base cases are given, assume something reasonable but nontrivial. Extra credit will be given for more exact solutions.
- (a) $A(n) = 3A(n/2) + n$
 - (b) $B(n) = \max_{n/3 < k < 2n/3} (B(k) + B(n - k) + n)$
 - (c) $C(n) = 4C(\lfloor n/2 \rfloor + 5) + n^2$
 - *(d) $D(n) = 2D(n/2) + n/\lg n$
 - *(e) $E(n) = \frac{E(n-1)}{E(n-2)}$, where $E(1) = 1$ and $E(2) = 2$.
4. 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?

[Hint: Solve for an n -card deck, and then set $n = 52$.] Prove that your answers are correct. If you have to appeal to “intuition” or “common sense”, your answers are probably wrong!

5. Suppose you have a pointer to the head of singly linked list. Normally, each node in the list only has a pointer to the next element, and the last node's pointer is `NULL`. Unfortunately, your list might have been corrupted by a bug in somebody else's code¹, so that the last node has a pointer back to some other node in the list instead.

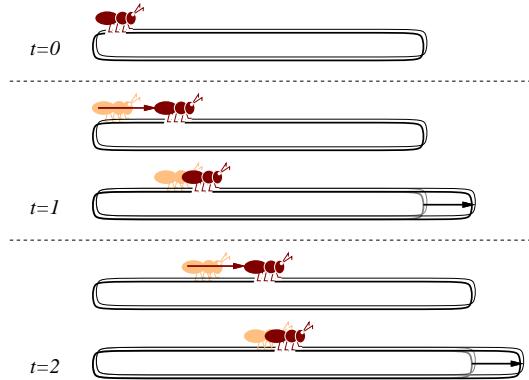


Top: A standard singly-linked list. Bottom: A corrupted singly linked list.

Describe an algorithm² that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. For full credit, your algorithm should run in $O(n)$ time, where n is the number of nodes in the list, and use $O(1)$ extra space (not counting the list itself).

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.]

An ant is walking along a rubber band, starting at the left end. Once every second, the ant walks one inch to the right, and then you make the rubber band one inch longer by pulling on the right end. The rubber band stretches uniformly, so stretching the rubber band also pulls the ant to the right. The initial length of the rubber band is n inches, so after t seconds, the rubber band is $n + t$ inches long.



Every second, the ant walks an inch, and then the rubber band is stretched an inch longer.

- (a) How far has the ant moved after t seconds, as a function of n and t ? Set up a recurrence and (for full credit) give an *exact* closed-form solution. [Hint: What fraction of the rubber band's length has the ant walked?]
- *(b) How long does it take the ant to get to the right end of the rubber band? For full credit, give an answer of the form $f(n) + \Theta(1)$ for some explicit function $f(n)$.

¹After all, *your* code is always completely 100% bug-free. Isn't that right, Mr. Gates?

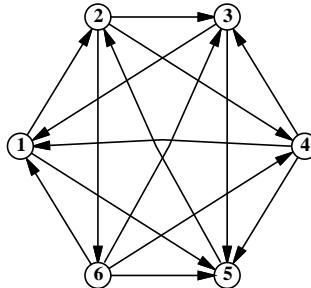
²Since you've read the Homework Instructions, you know what the phrase "describe an algorithm" means. Right?

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. 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 .

2. 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.



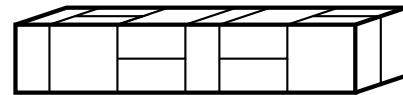
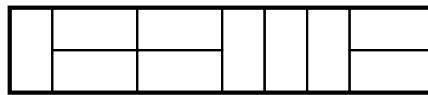
A six-vertex tournament containing the Hamiltonian path $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$.

3. (a) Prove the following identity by induction:

$$\binom{2n}{n} = \sum_{k=0}^n \binom{n}{k} \binom{n}{n-k}.$$

- (b) Give a non-inductive combinatorial proof of the same identity, by showing that the two sides of the equation count exactly the same thing in two different ways. There is a correct one-sentence proof.

4. (a) Prove that $2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor} / n = \Theta(n)$.
 (b) Is $2^{\lfloor \lg n \rfloor} = \Theta(2^{\lceil \lg n \rceil})$? Justify your answer.
 (c) Is $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta(2^{2^{\lceil \lg \lg n \rceil}})$? Justify your answer.
 (d) Prove that if $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$. Justify your answer.
 (e) Prove that $f(n) = O(g(n))$ does *not* imply that $\log(f(n)) = O(\log(g(n)))$?.
 *(f) Prove that $\log^k n = o(n^{1/k})$ for any positive integer k .
5. 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. If no base cases are given, assume something reasonable (but nontrivial). Extra credit will be given for more exact solutions.
- (a) $A(n) = A(n/2) + n$
 (b) $B(n) = 2B(n/2) + n$
 (c) $C(n) = \min_{0 < k < n} (C(k) + C(n - k) + 1)$, where $C(1) = 1$.
 (d) $D(n) = D(n - 1) + 1/n$
 *(e) $E(n) = 8E(n - 1) - 15E(n - 2) + 1$
 *(f) $F(n) = (n - 1)(F(n - 1) + F(n - 2))$, where $F(0) = F(1) = 1$
 ★(g) $G(n) = G(n/2) + G(n/4) + G(n/6) + G(n/12) + n$ [Hint: $\frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{12} = 1$.]
6. (a) A *domino* is a 2×1 or 1×2 rectangle. How many different ways are there to completely fill a $2 \times n$ rectangle with n dominos? Set up a recurrence relation and give an *exact* closed-form solution.
 (b) A *slab* is a three-dimensional box with dimensions $1 \times 2 \times 2$, $2 \times 1 \times 2$, or $2 \times 2 \times 1$. How many different ways are there to fill a $2 \times 2 \times n$ box with n slabs? Set up a recurrence relation and give an *exact* closed-form solution.



A 2×10 rectangle filled with ten dominos, and a $2 \times 2 \times 10$ box filled with ten slabs.

7. Professor George O'Jungle has a favorite 26-node binary tree, whose nodes are labeled by letters of the alphabet. The preorder and postorder sequences of nodes are as follows:

preorder: M N H C R S K W T G D X I Y A J P O E Z V B U L Q F
 postorder: C W T K S G R H D N A O E P J Y Z I B Q L F U V X M

Draw Professor O'Jungle's binary tree, and give the inorder sequence of nodes.

8. Alice and Bob each have a fair n -sided die. Alice rolls her die once. Bob then repeatedly throws his die until he rolls a number at least as big as the number Alice rolled. Each time Bob rolls, he pays Alice \$1. (For example, if Alice rolls a 5, and Bob rolls a 4, then a 3, then a 1, then a 5, the game ends and Alice gets \$4. If Alice rolls a 1, then no matter what Bob rolls, the game will end immediately, and Alice will get \$1.)

Exactly how much money does Alice expect to win at this game? Prove that your answer is correct. If you have to appeal to “intuition” or “common sense”, your answer is probably wrong!

9. Prove that for any nonnegative parameters a and b , the following algorithms terminate and produce identical output.

<u>SLOWEUCLID(a, b) :</u>
if $b > a$
return SLOWEUCLID(b, a)
else if $b = 0$
return a
else
return SLOWEUCLID($b, a - b$)

<u>FASTEUCLID(a, b) :</u>
if $b = 0$
return a
else
return FASTEUCLID($b, a \bmod b$)

CS 373: Combinatorial Algorithms, Fall 2000

Homework 1 (due September 12, 2000 at midnight)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

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. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or 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, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

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

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

2. Consider the following sorting algorithm:

```

STUPIDSORT( $A[0..n-1]$ ) :
  if  $n = 2$  and  $A[0] > A[1]$ 
    swap  $A[0] \leftrightarrow A[1]$ 
  else if  $n > 2$ 
     $m \leftarrow \lceil 2n/3 \rceil$ 
    STUPIDSORT( $A[0..m-1]$ )
    STUPIDSORT( $A[n-m..n-1]$ )
    STUPIDSORT( $A[0..m-1]$ )

```

- (a) Prove that STUPIDSORT actually sorts its input.
 - (b) Would the algorithm still sort correctly if we replaced the line $m \leftarrow \lceil 2n/3 \rceil$ with $m \leftarrow \lfloor 2n/3 \rfloor$? Justify your answer.
 - (c) State a recurrence (including the base case(s)) for the number of comparisons executed by STUPIDSORT.
 - (d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?
 - *(e) Show that the number of swaps executed by STUPIDSORT is at most $\binom{n}{2}$.
3. The following randomized algorithm selects the r th 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.

```

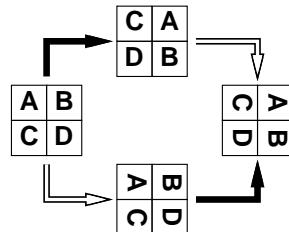
RANDOMSELECT( $A[1..n], r$ ) :
   $p \leftarrow \text{RANDOM}(1, p)$ 
   $k \leftarrow \text{PARTITION}(A[1..n], p)$ 
  if  $r < k$ 
    return RANDOMSELECT( $A[1..k-1], r$ )
  else if  $r > k$ 
    return RANDOMSELECT( $A[k+1..n], r-k$ )
  else
    return  $A[k]$ 

```

- (a) State a recurrence for the expected running time of RANDOMSELECT, as a function of n and r .
- (b) What is the *exact* probability that RANDOMSELECT compares the i th smallest and j th 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) What is the expected running time of RANDOMSELECT, as a function of n and r ? You can use either the recurrence from part (a) or the probabilities from part (b). For extra credit, give the *exact* expected number of comparisons.
- (d) What is the expected number of times that RANDOMSELECT calls itself recursively?

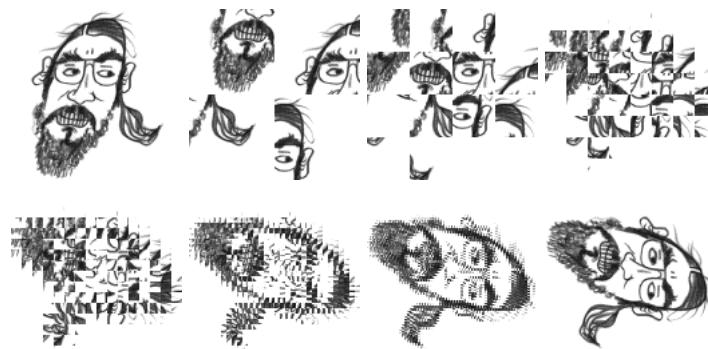
4. Some graphics hardware includes support for an operation called *blit*, or **block transfer**, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcp()`.

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



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

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



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

- Prove that both versions of the algorithm are correct. [Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]
- Exactly how many blits does the algorithm perform?
- What is the algorithm's running time if a $k \times k$ blit takes $O(k^2)$ time?
- What if a $k \times k$ blit takes only $O(k)$ time?

5. The traditional Devonian/Cornish drinking song “The Barley Mow” has the following pseudolyrics¹, where $\text{container}[i]$ is the name of a container that holds 2^i ounces of beer.²

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

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

  for  $i \leftarrow 1$  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!"
      "Here's a health to the barley-mow, my brave boys,"
      "Here's a health to the barley-mow!"
```

- (a) Suppose each container name $\text{container}[i]$ is a single word, and you can sing four words a second. How long would it take you to sing $\text{BARLEYMOW}(n)$? (Give a tight asymptotic bound.)
- (b) If you want to sing this song for $n > 20$, you’ll have to make up your own container names, and to avoid repetition, these names will get progressively longer as n increases³. Suppose $\text{container}[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing $\text{BARLEYMOW}(n)$? (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 $\text{container}[i]$. Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang $\text{BARLEYMOW}(n)$? (Give an *exact* answer, not just an asymptotic bound.)

¹Pseudolyrics are to lyrics as pseudocode is to code.

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

³“We'll drink it out of the hemisemidemiyottapint, boys!”

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.]

A company is planning a party for its employees. The employees in the company are organized into a strict hierarchy, that is, a tree with the company president at the root. The organizers of the party have assigned a real number to each employee measuring how ‘fun’ the employee is. In order to keep things social, there is one restriction on the guest list: an employee cannot attend the party if their immediate supervisor is present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it’s her company, after all. Give an algorithm that makes a guest list for the party that maximizes the sum of the ‘fun’ ratings of the guests.

Practice Problems

1. Give an $O(n^2)$ algorithm to find the longest increasing subsequence of a sequence of numbers. The elements of the subsequence need not be adjacent in the sequence. For example, the sequence $\langle 1, 5, 3, 2, 4 \rangle$ has longest increasing subsequence $\langle 1, 3, 4 \rangle$.

2. You are at a political convention with n delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party a delegate belongs to. However, you can check whether any two delegates are in the *same* party or not by introducing them to each other. (Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.)
 - (a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.
 - (b) Suppose exactly k political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)

3. Give an algorithm that finds the *second* smallest of n elements in at most $n + \lceil \lg n \rceil - 2$ comparisons. [Hint: divide and conquer to find the smallest; where is the second smallest?]

4. Suppose that you have an array of records whose keys to be sorted consist only of 0’s and 1’s. Give a simple, linear-time $O(n)$ algorithm to sort the array in place (using storage of no more than constant size in addition to that of the array).

5. Consider the problem of making change for n cents using the least number of coins.
- Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.
 - Suppose that the available coins have the values c^0, c^1, \dots, c^k for some integers $c > 1$ and $k \geq 1$. Show that the obvious greedy algorithm always yields an optimal solution.
 - Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution.
 - Describe a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.
 - Suppose we have only two types of coins whose values a and b are relatively prime. Prove that any value of greater than $(a - 1)(b - 1)$ can be made with these two coins.
- ★(f) For only three coins a, b, c whose greatest common divisor is 1, give an algorithm to determine the smallest value n such that change *can* be given for all values greater than n . [Note: this problem is currently unsolved for more than four coins!]
6. Suppose you have a subroutine that can find the median of a set of n items (i.e., the $\lfloor n/2 \rfloor$ smallest) in $O(n)$ time. Give an algorithm to find the k th biggest element (for arbitrary k) in $O(n)$ time.
7. You're walking along the beach and you stub your toe on something in the sand. You dig around it and find that it is a treasure chest full of gold bricks of different (integral) weight. Your knapsack can only carry up to weight n before it breaks apart. You want to put as much in it as possible without going over, but you *cannot* break the gold bricks up.
- Suppose that the gold bricks have the weights $1, 2, 4, 8, \dots, 2^k$, $k \geq 1$. Describe and prove correct a greedy algorithm that fills the knapsack as much as possible without going over.
 - Give a set of 3 weight values for which the greedy algorithm does *not* yield an optimal solution and show why.
 - Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of gold brick values.

CS 373: Combinatorial Algorithms, Fall 2000

Homework 2 (due September 28, 2000 at midnight)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

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. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or 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, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Faster Longest Increasing Subsequence (15 pts)

Give an $O(n \log n)$ algorithm to find the longest increasing subsequence of a sequence of numbers. [Hint: In the dynamic programming solution, you don't really have to look back at all previous items. There was a practice problem on HW 1 that asked for an $O(n^2)$ algorithm for this. If you are having difficulty, look at the HW 1 solutions.]

2. SELECT(A, k) (10 pts)

Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.

- Show that a rotation in an augmented binary tree can be performed in constant time.
- Describe an algorithm `SCAPEGOATSELECT(k)` that selects the k th smallest item in an augmented scapegoat tree in $O(\log n)$ worst-case time.
- Describe an algorithm `SPLAYSELECT(k)` that selects the k th smallest item in an augmented splay tree in $O(\log n)$ amortized time.

- (d) Describe an algorithm $\text{TREAPSELECT}(k)$ that selects the k th smallest item in an augmented treap in $O(\log n)$ *expected* time.

[Hint: The answers for (b), (c), and (d) are almost exactly the same!]

3. Scapegoat trees (15 pts)

- (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. For 5 extra credit points, use only $O(1)$ additional memory.

4. Memory Management (10 pts)

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

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

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method—it makes the problem much too hard!

5. Fibonacci Heaps: SECONDMIN (10 pts)

- (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.
- *(b) Modify the Fibonacci Heap data structure to implement the SECONDMIN operation in constant time, without degrading the performance of any other Fibonacci heap operation.

Practice Problems

1. Amortization

- (a) Modify the binary double-counter (see class notes Sept 12) to support a new operation SIGN, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

[Hint: Suppose p is the number of significant bits in P , and n is the number of significant bits in N . For example, if $P = 17 = 10001_2$ and $N = 0$, then $p = 5$ and $n = 0$. Then $p - n$ always has the same sign as $P - N$. Assume you can update p and n in $O(1)$ time.]

- *(b) Do the same but now you can't assume that p and n can be updated in $O(1)$ time.

*2. Amortization

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 i th least significant fit indicates whether the sum includes the i th 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. This is not the same representation as on Homework 0!]]

3. Rotations

- (a) Show that it is possible to transform any n -node binary search tree into any other n -node binary search tree using at most $2n - 2$ rotations.
- *(b) Use fewer than $2n - 2$ rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most $2n - 6$ rotations, and there are pairs of trees that are $2n - 10$ rotations apart. These are the best bounds known.
4. Give an efficient implementation of the operation CHANGEKEY(x, k), which changes the key of a node x in a Fibonacci heap to the value k . The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which k is greater than, less than, or equal to $\text{key}[x]$.

5. Detecting overlap

- (a) You are given a list of ranges represented by min and max (e.g., [1,3], [4,5], [4,9], [6,8], [7,10]). Give an $O(n \log n)$ -time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.

- (b) You are given a list of rectangles represented by min and max x - and y -coordinates. Give an $O(n \log n)$ -time algorithm that decides whether or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). [Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.]

6. Comparison of Amortized Analysis Methods

A sequence of n operations is performed on a data structure. The i th 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 2000

Homework 3 (due October 17, 2000 at midnight)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

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. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or 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, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

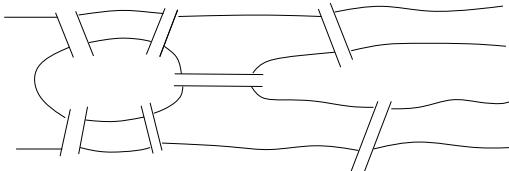
1. Suppose you have to design a dictionary that holds 2048 items.
 - (a) How many probes are used for an unsuccessful search if the dictionary is implemented as a sorted array? Assume the use of Binary Search.
 - (b) How large a hashtable do you need if your goal is to have 2 as the expected number of probes for an unsuccessful search?
 - (c) How much more space is needed by the hashtable compared to the sorted array? Assume that each pointer in a linked list takes 1 word of storage.
2. 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. Don't worry about the details of parsing a Makefile.

3. A person wants to fly from city A to city B in the shortest possible time. She 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 a route with the minimum total travel time—initial takeoff to final landing, including layovers. [*Hint: Modify the data and call a shortest-path algorithm.*]

4. 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.

5. 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 .

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.*]

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.

Practice Problems

- ★1. Let the hash function for a table of size m be

$$h(x) = \lfloor Ax \rfloor \bmod m$$

where $A = \hat{\phi} = \frac{\sqrt{5}-1}{2}$. Show that this gives the best possible spread, i.e. if the x are hashed in order, $x+1$ will be hashed in the largest remaining contiguous interval.

2. The incidence matrix of an undirected graph $G = (V, E)$ is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = [(i, j) \in E] = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if } (i, j) \notin E. \end{cases}$$

- (a) Describe what all the entries of the matrix product BB^T represent (B^T is the matrix transpose).
 - (b) Describe what all the entries of the matrix product B^TB represent.
 - ★(c) Let $C = BB^T - 2A$, where A is the adjacency matrix of G , with zeroes on the diagonal. Let C' be C with the first row and column removed. Show that $\det C'$ is the number of spanning trees.
3. (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.
4. Given an *undirected* graph $G = (V, E)$, and a weight function $f : E \rightarrow \mathbb{R}$ on the edges, give an algorithm that finds (in time polynomial in V and E) a cycle of smallest weight in G .
5. Let $G = (V, E)$ be a graph with n vertices. 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 .
6. Suppose all edge weights in a graph G are equal. Give an algorithm to compute a minimum spanning tree of G .

7. 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} .

8. (a) What is $5^{29^5^0 + 23^4^1 + 17^3^2 + 11^2^3 + 5^1^4} \pmod{6}$?
- (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 2000

Homework 4 (due October 26, 2000 at midnight)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 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 grad. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or 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, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

- (10 points) A certain algorithms professor once claimed that the height of an n -node Fibonacci heap is of height $O(\log n)$. Disprove his claim by showing that for a positive integer n , a sequence of Fibonacci heap operations that creates a Fibonacci heap consisting of just one tree that is a (downward) linear chain of n nodes.
- (20 points) *Fibonacci strings* are defined as follows:

$$\begin{aligned}F_1 &= b \\F_2 &= a \\F_n &= F_{n-1}F_{n-2} \quad \text{for all } n > 2\end{aligned}$$

where the recursive rule uses concatenation of strings, so $F_3 = ab$, $F_4 = aba$, and so on. Note that the length of F_n is the n th Fibonacci number.

- Prove that in any Fibonacci string there are no two b's adjacent and no three a's.

- (b) Give the unoptimized and optimized failure function for F_7 .
- (c) Prove that, in searching for the Fibonacci string F_k , the unoptimized KMP algorithm may shift $\lceil k/2 \rceil$ times on the same text character. In other words, prove that there is a chain of failure links $j \rightarrow \text{fail}[j] \rightarrow \text{fail}[\text{fail}[j]] \rightarrow \dots$ of length $\lceil k/2 \rceil$, and find an example text T that would cause KMP to traverse this entire chain on the same position in the text.
- (d) What happens here when you use the optimized prefix function? Explain.
3. (10 points) Show how to extend the Rabin-Karp fingerprinting method to handle the problem of looking for a given $m \times m$ pattern in an $n \times n$ array of characters. The pattern may be shifted horizontally and vertically, but it may not be rotated.
4. (10 points)
- A *cyclic rotation* of a string is obtained by chopping off a prefix and gluing it at the end of the string. For example, ALGORITHM is a cyclic shift of RITHMALGO. Describe and analyze an algorithm that determines whether one string $P[1..m]$ is a cyclic rotation of another string $T[1..n]$.
 - Describe and analyze an algorithm that decides, given any two binary trees P and T , whether P equals a subtree of T . We want an algorithm that compares the *shapes* of the trees. There is no data stored in the nodes, just pointers to the left and right children. [Hint: First transform both trees into strings.]
-
- P
- T
- P occurs exactly once as a subtree of T .
5. (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.]

Refer to the notes for lecture 11 for this problem. The GENERICSSSP algorithm described in class can be implemented using a stack for the ‘bag’. Prove that the resulting algorithm can be forced to perform in $\Omega(2^n)$ relaxation steps. To do this, 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*!

Practice Problems

1. String matching with wild-cards

Suppose you have an alphabet for patterns that includes a ‘gap’ or wild-card character; any length string of any characters can match this additional character. For example if ‘*’ is the wild-card, then the pattern foo*bar*nad can be found in foofoowangbarnad. Modify the computation of the prefix function to correctly match strings using KMP.

2. Prove that there is no comparison sort whose running time is linear for at least $1/2$ of the $n!$ inputs of length n . What about at least $1/n$? What about at least $1/2^n$?

3. Prove that $2n - 1$ comparisons are necessary in the worst case to merge two sorted lists containing n elements each.

4. Find asymptotic upper and lower bounds to $\lg(n!)$ without Stirling’s approximation (Hint: use integration).

5. Given a sequence of n elements of n/k blocks (k elements per block) all elements in a block are less than those to the right in sequence, show that you cannot have the whole sequence sorted in better than $\Omega(n \lg k)$. Note that the entire sequence would be sorted if each of the n/k blocks were individually sorted in place. Also note that combining the lower bounds for each block is not adequate (that only gives an upper bound).

6. Show how to find the occurrences of pattern P in text T by computing the prefix function of the string PT (the concatenation of P and T).

7. Lower Bounds on Adjacency Matrix Representations of Graphs

(a) Prove that the time to determine if an undirected graph has a cycle is $\Omega(V^2)$.

(b) Prove that the time to determine if there is a path between two nodes in an undirected graph is $\Omega(V^2)$.

CS 373: Combinatorial Algorithms, Fall 2000

Homework 1 (due November 16, 2000 at midnight)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

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. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

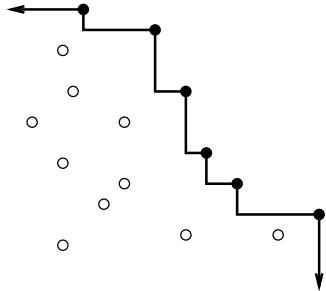
Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Give an $O(n^2 \log n)$ algorithm to determine whether any three points of a set of n points are collinear. Assume two dimensions and *exact* arithmetic.

2. We are given an array of n bits, and we want to determine if it contains two consecutive 1 bits. Obviously, we can check every bit, but is this always necessary?
 - (a) (4 pts) Show that when $n \bmod 3 = 0$ or 2, we must examine every bit in the array. That is, give an adversary strategy that forces any algorithm to examine every bit when $n = 2, 3, 5, 6, 8, 9, \dots$.
 - (b) (4 pts) Show that when $n = 3k + 1$, we only have to examine $n - 1$ bits. That is, describe an algorithm that finds two consecutive 1s or correctly reports that there are none after examining at most $n - 1$ bits, when $n = 1, 4, 7, 10, \dots$.
 - (c) (2 pts) How many n -bit strings are there with two consecutive ones? For which n is this number even or odd?

3. You are given a set of points in the plane. A point is *maximal* if there is no other point both above and to the right. The subset of maximal points of points then forms a *staircase*.

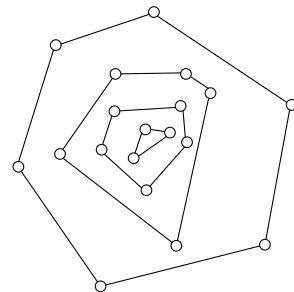


The staircase of a set of points. Maximal points are black.

- (a) (0 pts) Prove that maximal points are *not* necessarily on the convex hull.
- (b) (6 pts) Give an $O(n \log n)$ algorithm to find the maximal points.
- (c) (4 pts) Assume that points are chosen uniformly at random within a rectangle. What is the average number of maximal points? Justify. Hint: you will be able to give an exact answer rather than just asymptotics. You have seen the same analysis before.

4. Given a set Q of points in the plane, define the *convex layers* of Q inductively as follows: The first convex layer of Q is just the convex hull of Q . For all $i > 1$, the i th convex layer is the convex hull of Q after the vertices of the first $i - 1$ layers have been removed.

Give an $O(n^2)$ -time algorithm to find all convex layers of a given set of n points.



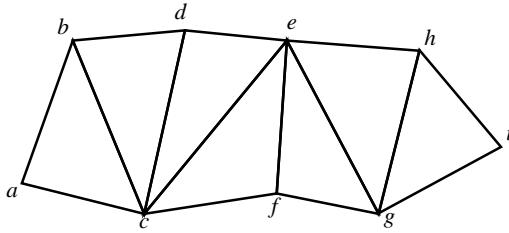
A set of points with four convex layers.

5. Prove that finding the second smallest of n elements takes $n + \lceil \lg n \rceil - 2$ comparisons in the worst case. Prove for both upper and lower bounds. Hint: find the (first) smallest using an elimination tournament.

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.]

Almost all computer graphics systems, at some level, represent objects as collections of triangles. In order to minimize storage space and rendering time, many systems allow objects to be stored as a set of *triangle strips*. A triangle strip is a sequence of vertices $\langle v_1, v_2, \dots, v_k \rangle$, where each contiguous triple of vertices v_i, v_{i+1}, v_{i+2} represents a triangle. As the rendering system reads the sequence of vertices and draws the triangles, it keeps the two most recent vertices in a cache.

Some systems allow triangle strips to contain *swaps*: special flags indicating that the order of the two cached vertices should be reversed. For example, the triangle strip $\langle a, b, c, d, \text{swap}, e, f, \text{swap}, g, h, i \rangle$ represents the sequence of triangles $(a, b, c), (b, c, d), (d, c, e), (c, e, f), (f, e, g)$.



Two triangle strips are *disjoint* if they share no triangles (although they may share vertices). The *length* of a triangle strip is the length of its vertex sequence, including swaps; for example, the example strip above has length 11. A *pure* triangle strip is one with no swaps. The adjacency graph of a triangle strip is a simple path. If the strip is pure, this path alternates between left and right turns.

Suppose you are given a set S of interior-disjoint triangles whose adjacency graph is a tree. (In other words, S is a triangulation of a simple polygon.) Describe a linear-time algorithm to decompose S into a set of disjoint triangle strips of minimum total length.

Practice Problems

1. Consider the following generic recurrence for convex hull algorithms that divide and conquer:

$$T(n, h) = T(n_1, h_1) + T(n_2, h_2) + O(n)$$

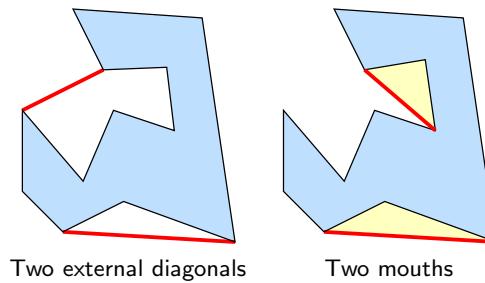
where $n \geq n_1 + n_2$, $h = h_1 + h_2$ and $n \geq h$. This means that the time to compute the convex hull is a function of both n , the number of input points, and h , the number of convex hull vertices. The splitting and merging parts of the divide-and-conquer algorithm take $O(n)$ time. When n is a constant, $T(n, h) = O(1)$, but when h is a constant, $T(n, h) = O(n)$. Prove that for both of the following restrictions, the solution to the recurrence is $O(n \log h)$:

- (a) $h_1, h_2 < \frac{3}{4}h$
- (b) $n_1, n_2 < \frac{3}{4}n$

2. Circle Intersection

Give an $O(n \log n)$ algorithm to test whether any two circles in a set of size n intersect.

3. Basic polygon computations (assume *exact* arithmetic)
- Intersection: Extend the basic algorithm to determine if two line segments intersect by taking care of *all* degenerate cases.
 - Simplicity: Give an $O(n \log n)$ algorithm to determine whether an n -vertex polygon is simple.
 - Area: Give an algorithm to compute the area of a simple n -polygon (not necessarily convex) in $O(n)$ time.
 - Inside: Give an algorithm to determine whether a point is within a simple n -polygon (not necessarily convex) in $O(n)$ time.
4. We are given the set of points one point at a time. After receiving each point, we must compute the convex hull of all those points so far. Give an algorithm to solve this problem in $O(n^2)$ total time. (We could obviously use Graham's scan n times for an $O(n^2 \log n)$ -time algorithm). Hint: How do you maintain the convex hull?
5. *(a) Given an n -polygon and a point outside the polygon, give an algorithm to find a tangent.
(b) Suppose you have found both tangents. Give an algorithm to remove the points from the polygon that are within the angle formed by the tangents (as segments!) and the opposite side of the polygon.
(c) Use the above to give an algorithm to compute the convex hull on-line in $O(n \log n)$
6. (a) A pair of polygon vertices defines an *external diagonal* if the line segment between them is completely outside the polygon. Show that every nonconvex polygon has at least one external diagonal.
(b) Three consecutive polygon vertices p, q, r form a *mouth* if p and r define an external diagonal. Show that every nonconvex polygon has at least one mouth.



7. A group of n ghostbusters is battling n ghosts. Each ghostbuster can shoot a single energy beam at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits the ghost. The ghostbusters all fire at the same time and no two energy beams may cross. The positions of the ghosts and ghostbusters are fixed points in the plane.
- Prove that for any configuration of ghosts and ghostbusters, there is such a non-crossing matching. (Assume that no three points are collinear.)

- (b) Show that there is a line passing through one ghostbuster and one ghost such that the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Give an efficient algorithm to find such a line.
- (c) Give an efficient divide and conquer algorithm to pair ghostbusters and ghosts so that no two streams cross.

CS 373: Combinatorial Algorithms, Fall 2000

Homework 6 (due December 7, 2000 at midnight)

Name:		
Net ID:	Alias:	<input type="checkbox"/> U <input checked="" type="checkbox"/> <input type="checkbox"/> 1

Name:		
Net ID:	Alias:	<input type="checkbox"/> U <input checked="" type="checkbox"/> <input type="checkbox"/> 1

Name:		
Net ID:	Alias:	<input type="checkbox"/> U <input checked="" type="checkbox"/> <input type="checkbox"/> 1

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. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or 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, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

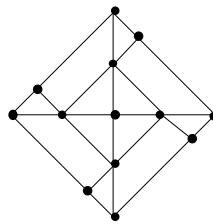
1. (a) Prove that $P \subseteq \text{co-NP}$.
(b) Show that if $NP \neq \text{co-NP}$, then *no* NP-complete problem is a member of co-NP.
2. 2SAT is a special case of the formula satisfiability problem, where the input formula is in conjunctive normal form and every clause has at most two literals. Prove that 2SAT is in P.
3. Describe an algorithm that solves the following problem, called 3SUM, as quickly as possible: Given a set of n numbers, does it contain three elements whose sum is zero? For example, your algorithm should answer TRUE for the set $\{-5, -17, 7, -4, 3, -2, 4\}$, since $-5+7+(-2)=0$, and FALSE for the set $\{-6, 7, -4, -13, -2, 5, 13\}$.

4. (a) Show that the problem of deciding whether one undirected graph is a subgraph of another is NP-complete.
(b) Show that the problem of deciding whether an unweighted undirected graph has a path of length greater than k is NP-complete.
 5. (a) Consider the following problem: Given a set of axis-aligned rectangles in the plane, decide whether any point in the plane is covered by k or more rectangles. Now also consider the CLIQUE problem. Describe and analyze a reduction of one problem to the other.
(b) Finding the largest clique in an arbitrary graph is NP-hard. What does this fact imply about the complexity of finding a point that lies inside the largest number of rectangles?
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.]

PARTITION is the problem of deciding, given a set $S = \{s_1, s_2, \dots, s_n\}$ of numbers, whether there is a subset T containing half the 'weight' of S , i.e., such that $\sum T = \frac{1}{2} \sum S$. SUBSETSUM is the problem of deciding, given a set $S = \{s_1, s_2, \dots, s_n\}$ of numbers and a target sum t , whether there is a subset $T \subseteq S$ such that $\sum T = t$. Give two reductions between these two problems, one in each direction.

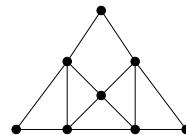
Practice Problems

1. What is the *exact* worst case number of comparisons needed to find the median of 5 numbers?
For 6 numbers?
2. 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 subcollection $C' \subseteq C$ 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.
3. Using 3COLOR and the ‘gadget’ below, prove that the problem of 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.]



Crossing gadget for PLANAR3COLOR.

4. Using the previous result, and the ‘gadget’ below, prove that the problem of deciding whether a planar graph with no vertex of degree greater than four 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.]



Degree gadget for DEGREE4PLANAR3COLOR

5. Show that an algorithm that makes at most a constant number of calls to polynomial-time subroutines runs in polynomial time, but that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.
6. (a) Prove that if G is an undirected bipartite graph with an odd number of vertices, then G is nonhamiltonian. Give a polynomial time algorithm for finding a hamiltonian cycle in an undirected bipartite graph or establishing that it does not exist.
(b) Show that the hamiltonian-path problem can be solved in polynomial time on directed acyclic graphs.
(c) Explain why the results in previous questions do not contradict the fact that both HAMILTONIANCYCLE and HAMILTONIANPATH are NP-complete problems.
7. Consider the following pairs of problems:

- (a) MIN SPANNING TREE and MAX SPANNING TREE
- (b) SHORTEST PATH and LONGEST PATH
- (c) TRAVELING SALESMAN and VACATION TOUR (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?

*8. Consider the problem of deciding whether one graph is isomorphic to another.

- (a) Give a brute force algorithm to decide this.
- (b) Give a dynamic programming algorithm to decide this.
- (c) Give an efficient probabilistic algorithm to decide this.
- ★(d) Either prove that this problem is NP-complete, give a poly time algorithm for it, or prove that neither case occurs.

*9. Prove that PRIMALITY (Given n , is n prime?) is in $\text{NP} \cap \text{co-NP}$. Showing that PRIMALITY is in 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 checked to be correct and used to show that n is prime, and that this check takes polynomial time.

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

CS 373: Combinatorial Algorithms, Fall 2000

Midterm 1 — October 3, 2000

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your $8\frac{1}{2}'' \times 11''$ cheat sheet, please leave it at the front of the classroom.

-
- Print your name, netid, and alias in the boxes above. Circle U if you are an undergrad, $\frac{3}{4}$ if you are a 3/4-unit grad student, or 1 if you are a 1-unit grad student. Print your name at the top of every page (in case the staple falls out!).
 - **Answer four of the five questions on the exam.** Each question is worth 10 points. If you answer more than four questions, the one with the lowest score will be ignored. **1-unit graduate students must answer question 5.**
 - Please write your final answers on the front of the exam pages. Use the backs of the pages as scratch paper. Let us know if you need more paper.
 - Unless we specifically say otherwise, proofs are not required. However, they may help us give you partial credit.
 - Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
 - Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
 - Write something down for every problem. Don't panic and erase large chunks of work. Even if you think it's absolute nonsense, it might be worth partial credit.
 - Relax. Breathe. Kick some ass.

#	Score	Grader
1		
2		
3		
4		
5		
Total		

1. Multiple Choice

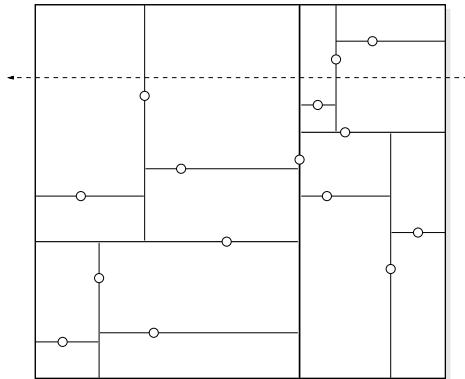
Every question below has one of the following answers.

- (a) $\Theta(1)$ (b) $\Theta(\log n)$ (c) $\Theta(n)$ (d) $\Theta(n \log n)$ (e) $\Theta(n^2)$

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, but each incorrect answer *costs* you $\frac{1}{2}$ point. You cannot score below zero.

- What is $\sum_{i=1}^n \log i$?
- What is $\sum_{i=1}^n \frac{n}{i}$?
- How many digits do you need to write 2^n in decimal?
- What is the solution of the recurrence $T(n) = 25T(n/5) + n$?
- What is the solution of the recurrence $T(n) = T(n - 1) + \frac{1}{2^n}$?
- What is the solution of the recurrence $T(n) = 3T\left(\lceil \frac{n+51}{3} \rceil\right) + 17n - \sqrt[3]{\lg \lg n} - 2^{2^{\log^* n}} + \pi$?
- What is the worst-case running time of randomized quicksort?
- The expected time for inserting one item into an n -node randomized treap is $O(\log n)$.
What is the worst-case time for a sequence of n insertions into an initially empty treap?
- The amortized time for inserting one item into an n -node scapegoat tree is $O(\log n)$.
What is the worst-case time for a sequence of n insertions into an initially empty scapegoat tree?
- In the worst case, how many nodes can be in the root list of a Fibonacci heap storing n keys, immediately after a DECREASEKEY operation?
- Every morning, an Amtrak train leaves Chicago for Champaign, 200 miles away. The train can accelerate or decelerate at 10 miles per hour per second, and it has a maximum speed of 60 miles an hour. Every 50 miles, the train must stop for five minutes while a school bus crosses the tracks. Every hour, the conductor stops the train for a union-mandated 10-minute coffee break. How long does it take the train to reach Champaign?

2. Suppose we have n points scattered inside a two-dimensional box. A *kd-tree* recursively subdivides the rectangle as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line passes through some point inside the box (*not* on the boundary) and partitions the rest of the interior points as evenly as possible. If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.

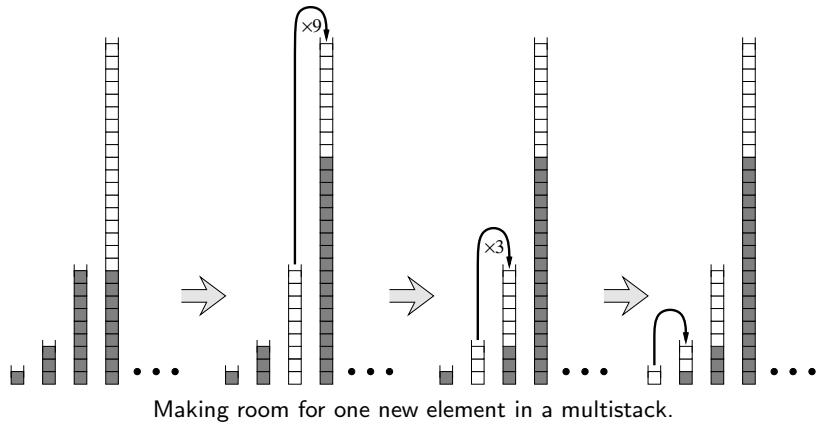


A kd-tree for 15 points. The dashed line crosses four cells.

- (a) [2 points] How many cells are there, as a function of n ? Prove your answer is correct.
- (b) [8 points] In the worst case, exactly how many cells can a horizontal line cross, as a function of n ? Prove your answer is correct. Assume that $n = 2^k - 1$ for some integer k .
[For full credit, you must give an exact answer. A tight asymptotic bound (with proof) is worth 5 points. A correct recurrence is worth 3 points.]
- (c) [5 points extra credit] In the worst case, how many cells can a diagonal line cross?

Incidentally, ‘kd-tree’ originally meant ‘ k -dimensional tree’—for example, the specific data structure described here used to be called a ‘2d-tree’—but current usage ignores this etymology. The phrase ‘ d -dimensional kd-tree’ is now considered perfectly standard, even though it’s just as redundant as ‘ATM machine’, ‘PIN number’, ‘HIV virus’, ‘PDF format’, ‘Mt. Fujiyama’, ‘Sahara Desert’, ‘The La Brea Tar Pits’, or ‘and etc.’ On the other hand, ‘BASIC code’ is *not* redundant; ‘Beginner’s All-Purpose Instruction Code’ is a backronym. Hey, aren’t you supposed to be taking a test?

3. A *multistack* consists of an infinite series of stacks S_0, S_1, S_2, \dots , where the i th 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.



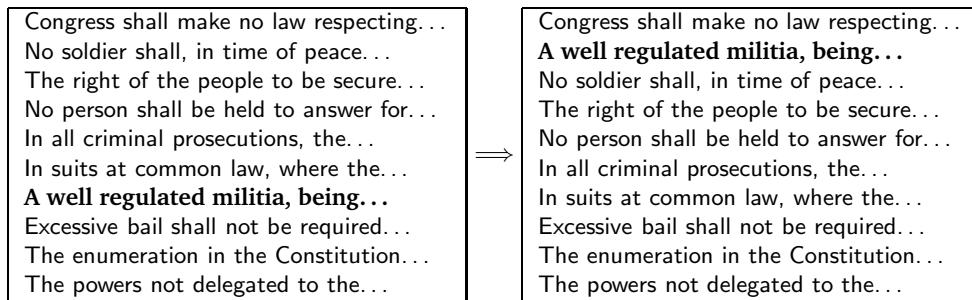
- (a) [1 point] In the worst case, how long does it take to push one more element onto a multistack containing n elements?
- (b) [9 points] Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack. You can use any method you like.

4. After graduating with a computer science degree, you find yourself working for a software company that publishes a word processor. The program stores a document containing n characters, grouped into p paragraphs. Your manager asks you to implement a ‘Sort Paragraphs’ command that rearranges the paragraphs into alphabetical order.

Design and analyze an efficient paragraph-sorting algorithm, using the following pair of routines as black boxes.

- COMPAREPARAGRAPHS(i, j) compares the i th and j th paragraphs, and returns i or j depending on which paragraph should come first in the final sorted output. (Don’t worry about ties.) This function runs in $O(1)$ time, since almost any two paragraphs can be compared by looking at just their first few characters!
- MOVEPARAGRAPH(i, j) ‘cuts’ out the i th paragraph and ‘pastes’ it back in as the j th paragraph. This function runs in $O(n_i)$ time, where n_i is the number of characters in the i th paragraph. (So in particular, $n_1 + n_2 + \dots + n_p = n$.)

Here is an example of MOVEPARAGRAPH(7, 2):



[Hint: For full credit, your algorithm should run in $o(n \log n)$ time when $p = o(n)$.]

5. [1-unit grad students must answer this question.]

Describe and analyze an algorithm to randomly shuffle an array of n items, so that each of the $n!$ possible permutations is equally likely. Assume that you have a function RANDOM(i, j) that returns a random integer from the set $\{i, i + 1, \dots, j\}$ in constant time.

[Hint: As a sanity check, you might want to confirm that for $n = 3$, all six permutations have probability $1/6$. For full credit, your algorithm must run in $\Theta(n)$ time. A correct algorithm that runs in $\Theta(n \log n)$ time is worth 7 points.]

From: "Josh Pepper" <jwpepper@uiuc.edu>
To: "Chris Neihengen" <neihenge@uiuc.edu>
Subject: FW: proof
Date: Fri, 29 Sep 2000 09:34:56 -0500

thought you might like this.

Problem: To prove that computer science 373 is indeed the work of Satan.

Proof: First, let us assume that everything in "Helping Yourself with Numerology", by Helyn Hitchcock, is true.

Second, let us apply divide and conquer to this problem. There are main parts:

1. The name of the course: "Combinatorial Algorithms"
2. The most important individual in the course, the "Recursion Fairy"
3. The number of this course: 373.

We examine these sequentially.

The name of the course. "Combinatorial Algorithms" can actually be expressed as a single integer - 23 - since it has 23 letters.

The most important individual, the Recursion Fairy, can also be expressed as a single integer - 14 - since it has 14 letters. In other words:

COMBINATORIAL ALGORITHMS = 23
RECUSION FAIRY = 14

As a side note, a much shorter proof has already been published showing that the Recursion Fairy is Lucifer, and that any class involving the Fairy is from Lucifer, however, that proofs numerical significance is slight.

Now we can move on to an analysis of the number of course, which holds great meaning. The first assumption we make is that the number of the course, 373, is not actually a base 10 number. We can prove this inductively by making a reasonable guess for the actual base, then finding a new way to express the nature of the course, and if the answer

confirms what we assumed, then we're right. That's the way induction works.

What is a reasonable guess for the base of the course? The answer is trivial, since the basest of all beings is the Recursion Fairy, the base is 14. So a true base 10 representation of 373 (base 14) is 689. So we see:

373 (base 14) = 689 (base 10)

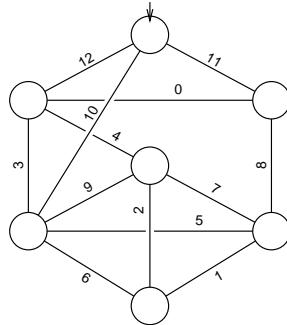
Now since the nature of the course has absolutely nothing to do with combinatorial algorithms (instead having much to do with the work of the devil), we can subtract from the above result everything having to do with combinatorial algorithms just by subtracting 23. Here we see that:

689 - 23 = 666

QED.

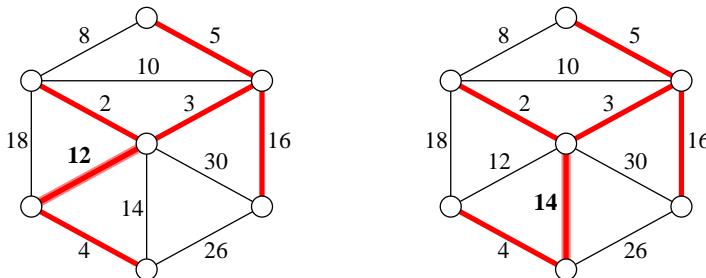
1. Using any method you like, compute the following subgraphs for the weighted graph below. Each subproblem is worth 3 points. Each incorrect edge costs you 1 point, but you cannot get a negative score for any subproblem.

- (a) a depth-first search tree, starting at the top vertex;
- (b) a breadth-first search tree, starting at the top vertex;
- (c) a shortest path tree, starting at the top vertex;
- (d) the minimum spanning tree.



2. Suppose you are given a weighted undirected graph G (represented as an adjacency list) and its minimum spanning tree T (which you already know how to compute). Describe and analyze an algorithm to find the *second-minimum spanning tree* of G , i.e., the spanning tree of G with smallest total weight except for T .

The minimum spanning tree and the second-minimum spanning tree differ by exactly one edge. But *which* edge is different, and *how* is it different? That's what your algorithm has to figure out!

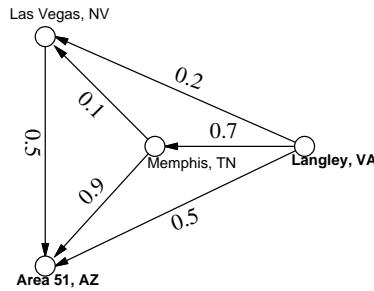


The minimum spanning tree and the second-minimum spanning tree of a graph.

3. (a) [4 pts] Prove that a connected acyclic graph with V vertices has exactly $V - 1$ edges. (“It’s a tree!” is not a proof.)
- (b) [4 pts] Describe and analyze an algorithm that determines whether a given graph is a tree, where the graph is represented by an adjacency list.
- (c) [2 pts] What is the running time of your algorithm from part (b) if the graph is represented by an adjacency matrix?

4. Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road *won't* be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What route should he take so that he has the least chance of being abducted?

More formally, you are given a directed graph $G = (V, E)$, where every edge e has an independent safety probability $p(e)$. The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex s to a given target vertex t .



With the probabilities shown above, if Mulder tries to drive directly from Langley to Area 51, he has a 50% chance of getting there without being abducted. If he stops in Memphis, he has a $0.7 \times 0.9 = 63\%$ chance of arriving safely. If he stops first in Memphis and then in Las Vegas, he has a $1 - 0.7 \times 0.1 \times 0.5 = 96.5\%$ chance of being abducted!¹

5. [1-unit grad students must answer this question.]

Many string matching applications allow the following *wild card* characters in the pattern.

- The wild card ? represents an arbitrary single character. For example, the pattern s?r?ng matches the strings string, sprung, and sarong.
- The wild card * represents an arbitrary string of zero or more characters. For example, the pattern te*st* matches the strings test, tensest, and technostructuralism.

Both wild cards can occur in a single pattern. For example, the pattern f*a?? matches the strings face, football, and flippyfloppydingdongdang. On the other hand, neither wild card can occur in the text.

Describe how to modify the Knuth-Morris-Pratt algorithm to support patterns with these wild cards, and analyze the modified algorithm. Your algorithm should find the first substring in the text that matches the pattern. An algorithm that supports only one of the two wild cards is worth 5 points.

¹That's how they got Elvis, you know.

1. True, False, or Maybe

Indicate whether each of the following statements is always true, sometimes true, always false, or unknown. Some of these questions are deliberately tricky, so read them carefully. Each correct choice is worth +1, and each incorrect choice is worth -1. **Guessing will hurt you!**

- (a) Suppose SMARTALGORITHM runs in $\Theta(n^2)$ time and DUMBALGORITHM runs in $\Theta(2^n)$ time for all inputs of size n . (Thus, for each algorithm, the best-case and worst-case running times are the same.) SMARTALGORITHM is faster than DUMBALGORITHM.

True False Sometimes Nobody Knows

- (b) QUICKSORT runs in $O(n^6)$ time.

True False Sometimes Nobody Knows

- (c) $\lfloor \log_2 n \rfloor \geq \lceil \log_2 n \rceil$

True False Sometimes Nobody Knows

- (d) The recurrence $F(n) = n + 2\sqrt{n} \cdot F(\sqrt{n})$ has the solution $F(n) = \Theta(n \log n)$.

True False Sometimes Nobody Knows

- (e) A Fibonacci heap with n nodes has depth $\Omega(\log n)$.

True False Sometimes Nobody Knows

- (f) Suppose a graph G is represented by an adjacency matrix. It is possible to determine whether G is an independent set without looking at every entry of the adjacency matrix.

True False Sometimes Nobody Knows

- (g) $NP \neq co\text{-}NP$

True False Sometimes Nobody Knows

- (h) Finding the smallest clique in a graph is NP-hard.

True False Sometimes Nobody Knows

- (i) A polynomial-time reduction from X to 3SAT proves that X is NP-hard.

True False Sometimes Nobody Knows

- (j) The correct answer for exactly three of these questions is “False”.

True False

2. Convex Hull

Suppose you are given the convex hull of a set of n points, and one additional point (x, y) . The convex hull is represented by an array of vertices in counterclockwise order, starting from the leftmost vertex. Describe how to test in $O(\log n)$ time whether or not the additional point (x, y) is inside the convex hull.

3. Finding the Largest Block

In your new job, you are working with screen images. These are represented using two dimensional arrays where each element is a 1 or a 0, indicating whether that position of the screen is illuminated. Design and analyze an efficient algorithm to find the largest rectangular block of ones in such an array. For example, the largest rectangular block of ones in the array shown below is in rows 2–4 and columns 2–3. [Hint: Use dynamic programming.]

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

4. The Hogwarts Sorting Hat

Every year, upon their arrival at Hogwarts School of Witchcraft and Wizardry, new students are sorted into one of four houses (Gryffindor, Hufflepuff, Ravenclaw, or Slytherin) by the Hogwarts Sorting Hat. The student puts the Hat on their head, and the Hat tells the student which house they will join. This year, a failed experiment by Fred and George Weasley filled almost all of Hogwarts with sticky brown goo, mere moments before the annual Sorting. As a result, the Sorting had to take place in the basement hallways, where there was so little room to move that the students had to stand in a long line.

After everyone learned what house they were in, the students tried to group together by house, but there was too little room in the hallway for more than one student to move at a time. Fortunately, the Sorting Hat took CS 373 many years ago, so it knew how to group the students as quickly as possible. What method did the Sorting Hat use?

More formally, you are given an array of n items, where each item has one of four possible values, possibly with a pointer to some additional data. Design and analyze an algorithm that rearranges the items into four clusters in $O(n)$ time using only $O(1)$ extra space.

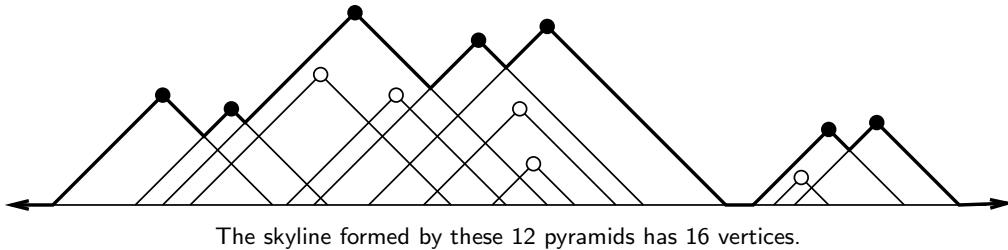
The diagram shows two horizontal arrays of 20 items each. The top array represents the initial state, and the bottom array represents the final state after sorting. Each item is a pair of letters (e.g., G, H, R, S) followed by a name. In the initial array, the items are mixed. In the final array, the items are grouped into four distinct clusters: G (Harry, Ekta, Bill, Chad, Nick, Mel, Dana, Fox, Ann, Jim, Dawn, Bart, Lisa, Tina, John, Bob, Liz, Mary, Kim, Jeff), H (Dawn, Bart, Lisa, Tina, John, Bob, Liz, Mary, Kim, Jeff), R (Harry, Ekta, Bill, Chad, Nick, Mel, Dana, Fox, Ann, Jim, Dawn, Bart, Lisa, Tina, John, Bob, Liz, Mary, Kim, Jeff), and S (Nick, Kim, Fox, Dana, Mel).

G	H	R	R	G	G	R	G	H	H	R	R	S	R	R	H	G	S	H	G	G
Harry	Ann	Bob	Tina	Chad	Bill	Lisa	Ekta	Bart	Jim	John	Jeff	Liz	Mary	Dawn	Nick	Kim	Fox	Dana	Mel	

G	G	G	G	G	G	G	H	H	H	H	H	R	R	R	R	R	R	R	S	S
Harry	Ekta	Bill	Chad	Nick	Mel	Dana	Fox	Ann	Jim	Dawn	Bart	Lisa	Tina	John	Bob	Liz	Mary	Kim	Jeff	

5. The Egyptian Skyline

Suppose you are given a set of n pyramids in the plane. Each pyramid is a isosceles triangle with two 45° edges and a horizontal edge on the x -axis. Each pyramid is represented by the x - and y -coordinates of its topmost point. Your task is to compute the “skyline” formed by these pyramids (the dark line shown below).



- (a) Describe and analyze an algorithm that determines which pyramids are visible on the skyline. These are the pyramids with black points in the figure above; the pyramids with white points are not visible. [Hint: You've seen this problem before.]
- (b) Once you know which pyramids are visible, how would you compute the *shape* of the skyline? Describe and analyze an algorithm to compute the left-to-right sequence of skyline vertices, including the vertices between the pyramids and on the ground.

6. DNF-SAT

A boolean formula is in *disjunctive normal form* (DNF) if it consists of clauses of conjunctions (ANDs) joined together by disjunctions (ORs). For example, the formula

$$(\bar{a} \wedge b \wedge \bar{c}) \vee (b \wedge c) \vee (a \wedge \bar{b} \wedge \bar{c})$$

is in disjunctive normal form. DNF-SAT is the problem that asks, given a boolean formula in disjunctive normal form, whether that formula is satisfiable.

- (a) Show that DNF-SAT is in P.
- (b) What is wrong with the following argument that P=NP?

Suppose we are given a boolean formula in conjunctive normal form with at most three literals per clause, and we want to know if it is satisfiable. We can use the distributive law to construct an equivalent formula in disjunctive normal form. For example,

$$(a \vee b \vee \bar{c}) \wedge (\bar{a} \vee \bar{b}) \iff (a \wedge \bar{b}) \vee (b \wedge \bar{a}) \vee (\bar{c} \wedge \bar{a}) \vee (\bar{c} \wedge \bar{b})$$

Now we can use the answer to part (a) to determine, in polynomial time, whether the resulting DNF formula is satisfiable. We have just solved 3SAT in polynomial time! Since 3SAT is NP-hard, we must conclude that P=NP.

7. Magic 3-Coloring [1-unit graduate students must answer this question.]

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 different colors.) The magic box does not tell you *how* to color the graph, just whether or not it can be done. Devise and analyze an algorithm to 3-color any graph **in polynomial time** using this magic box.

CS 373: Combinatorial Algorithms, Spring 2001

Homework 0, due January 23, 2001 at the beginning of class

Name:	
Net ID:	Alias:

Neatly print your name (first name first, with no comma), your network ID, and a short alias into the boxes above. **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.

This homework tests your familiarity with the prerequisite material from CS 173, CS 225, and CS 273—many of these problems have appeared on homeworks or exams in those classes—primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Parberry and Chapters 1–6 of CLR should be sufficient review, but you may want to consult other texts as well.

Before you do anything else, read the Homework Instructions and FAQ on the CS 373 course web page (<http://www-courses.cs.uiuc.edu/~cs373/hw/faq.html>), and then check the box below. This web page gives instructions on how to write and submit homeworks—staple your solutions together in order, write your name and netID on every page, don't turn in source code, analyze everything, use good English and good logic, and so forth.

I have read the CS 373 Homework Instructions and FAQ.

Required Problems

1. (a) Prove that any positive integer can be written as the sum of distinct powers of 2. For example: $42 = 2^5 + 2^3 + 2^1$, $25 = 2^4 + 2^3 + 2^0$, $17 = 2^4 + 2^0$. [Hint: ‘Write the number in binary’ is not a proof; it just restates the problem.]
- (b) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, $17 = F_7 + F_4 + F_2$.
- (c) Prove that *any* integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example: $42 = 3^4 - 3^3 - 3^2 - 3^1$, $25 = 3^3 - 3^1 + 3^0$, $17 = 3^3 - 3^2 - 3^0$.

2. Sort the following 20 functions 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.

$$\begin{array}{ccccc}
 1 & n & n^2 & \lg n & \lg^* n \\
 2^{2^{\lg \lg n + 1}} & \lg^* 2^n & 2^{\lg^* n} & \lfloor \lg(n!) \rfloor & \lfloor \lg n \rfloor! \\
 n^{\lg n} & (\lg n)^n & (\lg n)^{\lg n} & n^{1/\lg n} & n^{\lg \lg n} \\
 \log_{1000} n & \lg^{1000} n & \lg^{(1000)} n & \left(1 + \frac{1}{1000}\right)^n & n^{1/1000}
 \end{array}$$

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 $n \ll \binom{n}{2} \equiv n^2 \ll n^3$.

3. 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 none are supplied. Extra credit will be given for more exact solutions.

- (a) $A(n) = 5A(n/3) + n \log n$
- (b) $B(n) = \min_{0 < k < n} (B(k) + B(n - k) + 1)$.
- (c) $C(n) = 4C(\lfloor n/2 \rfloor + 5) + n^2$
- (d) $D(n) = D(n - 1) + 1/n$
- *(e) $E(n) = n + 2\sqrt{n} \cdot E(\sqrt{n})$

4. This problem asks you to simplify some recursively defined boolean formulas as much as possible. In each case, prove that your answer is correct. Each proof can be just a few sentences long, but it must be a *proof*.
- (a) Suppose $\alpha_0 = p$, $\alpha_1 = q$, and $\alpha_n = (\alpha_{n-2} \wedge \alpha_{n-1})$ for all $n \geq 2$. Simplify α_n as much as possible. [Hint: What is α_5 ?]
 - (b) Suppose $\beta_0 = p$, $\beta_1 = q$, and $\beta_n = (\beta_{n-2} \Leftrightarrow \beta_{n-1})$ for all $n \geq 2$. Simplify β_n as much as possible. [Hint: What is β_5 ?]
 - (c) Suppose $\gamma_0 = p$, $\gamma_1 = q$, and $\gamma_n = (\gamma_{n-2} \Rightarrow \gamma_{n-1})$ for all $n \geq 2$. Simplify γ_n as much as possible. [Hint: What is γ_5 ?]
 - (d) Suppose $\delta_0 = p$, $\delta_1 = q$, and $\delta_n = (\delta_{n-2} \bowtie \delta_{n-1})$ for all $n \geq 2$, where \bowtie is some boolean function with two arguments. Find a boolean function \bowtie such that $\delta_n = \delta_m$ if and only if $n - m$ is a multiple of 4. [Hint: There is only one such function.]

5. Every year, upon their arrival at Hogwarts School of Witchcraft and Wizardry, new students are sorted into one of four houses (Gryffindor, Hufflepuff, Ravenclaw, or Slytherin) by the Hogwarts Sorting Hat. The student puts the Hat on their head, and the Hat tells the student which house they will join. This year, a failed experiment by Fred and George Weasley filled almost all of Hogwarts with sticky brown goo, mere moments before the annual Sorting. As a result, the Sorting had to take place in the basement hallways, where there was so little room to move that the students had to stand in a long line.

After everyone learned what house they were in, the students tried to group together by house, but there was too little room in the hallway for more than one student to move at a time. Fortunately, the Sorting Hat took CS 373 many years ago, so it knew how to group the students as quickly as possible. What method did the Sorting Hat use?

More formally, you are given an array of n items, where each item has one of four possible values, possibly with a pointer to some additional data. Describe an algorithm¹ that rearranges the items into four clusters in $O(n)$ time using only $O(1)$ extra space.

G Harry	H Ann	R Bob	R Tina	G Chad	G Bill	R Lisa	G Ekta	H Bart	H Jim	R John	S Jeff	R Liz	R Mary	H Dawn	G Nick	S Kim	H Fox	G Dana	G Mel
↓																			
G Harry	G Ekta	G Bill	G Chad	G Nick	G Mel	G Dana	H Fox	H Ann	H Jim	H Dawn	H Bart	R Lisa	R Tina	R John	R Bob	R Liz	R Mary	S Kim	S Jeff

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.]

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?

[Hint: Solve for an n -card deck and then set $n = 52$.] In each case, give *exact* answers and prove that they are correct. If you have to appeal to “intuition” or “common sense”, your answers are probably wrong!

¹Since you’ve read the Homework Instructions, you know what the phrase ‘describe an algorithm’ means. Right?

Practice Problems

The remaining problems are entirely for your benefit; similar questions will appear in every homework. Don't turn in solutions—we'll just throw them out—but feel free to ask us about practice questions during office hours and review sessions. Think of them as potential exam questions (hint, hint). We'll post solutions to *some* of the practice problems after the homeworks are due.

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

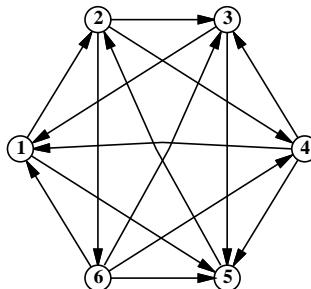
- ★(d) If n is an integer multiple of m , then F_n is an integer multiple of F_m .

2. (a) Prove the following identity by induction:

$$\binom{2n}{n} = \sum_{k=0}^n \binom{n}{k} \binom{n}{n-k}.$$

- (b) Give a non-inductive combinatorial proof of the same identity, by showing that the two sides of the equation count exactly the same thing in two different ways. There is a correct one-sentence proof.

3. 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 $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$.

4. 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 none are supplied. Extra credit will be given for more exact solutions.

- (a) $A(n) = A(n/2) + n$
- (b) $B(n) = 2B(n/2) + n$
- ★(c) $C(n) = n + \frac{1}{2}(C(n-1) + C(3n/4))$
- (d) $D(n) = \max_{n/3 < k < 2n/3} (D(k) + D(n-k) + n)$
- *(e) $E(n) = 2E(n/2) + n/\lg n$
- *(f) $F(n) = \frac{F(n-1)}{F(n-2)}$, where $F(1) = 1$ and $F(2) = 2$.
- *(g) $G(n) = G(n/2) + G(n/4) + G(n/6) + G(n/12) + n$ [Hint: $\frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{12} = 1$.]
- *(h) $H(n) = n + \sqrt{n} \cdot H(\sqrt{n})$
- *(i) $I(n) = (n-1)(I(n-1) + I(n-2))$, where $I(0) = I(1) = 1$
- *(j) $J(n) = 8J(n-1) - 15J(n-2) + 1$

5. (a) Prove that $2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor} = \Theta(n^2)$.

- (b) Prove or disprove: $2^{\lfloor \lg n \rfloor} = \Theta(2^{\lceil \lg n \rceil})$.
- (c) Prove or disprove: $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta(2^{2^{\lceil \lg \lg n \rceil}})$.
- (d) Prove or disprove: If $f(n) = O(g(n))$, then $\log(f(n)) = O(\log(g(n)))$.
- (e) Prove or disprove: If $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$.
- *(f) Prove that $\log^k n = o(n^{1/k})$ for any positive integer k .

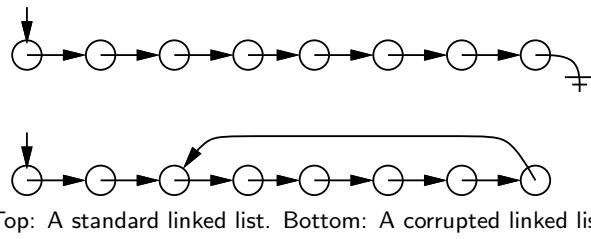
6. Evaluate the following summations; simplify your answers as much as possible. Significant partial credit will be given for answers in the form $\Theta(f(n))$ for some recognizable function $f(n)$.

$$(a) \sum_{i=1}^n \sum_{j=1}^i \sum_{k=j}^i \frac{1}{i}$$

$$*(b) \sum_{i=1}^n \sum_{j=1}^i \sum_{k=j}^i \frac{1}{j}$$

$$(c) \sum_{i=1}^n \sum_{j=1}^i \sum_{k=j}^i \frac{1}{k}$$

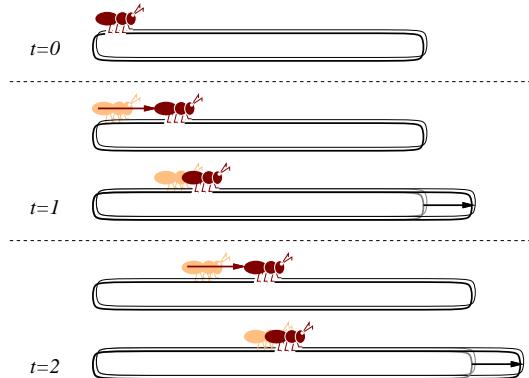
7. Suppose you have a pointer to the head of singly linked list. Normally, each node in the list only has a pointer to the next element, and the last node's pointer is `NULL`. Unfortunately, your list might have been corrupted by a bug in somebody else's code², so that the last node has a pointer back to some other node in the list instead.



Top: A standard linked list. Bottom: A corrupted linked list.

Describe an algorithm that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. For full credit, your algorithm should run in $O(n)$ time, where n is the number of nodes in the list, and use $O(1)$ extra space (not counting the list itself).

- *8. An ant is walking along a rubber band, starting at the left end. Once every second, the ant walks one inch to the right, and then you make the rubber band one inch longer by pulling on the right end. The rubber band stretches uniformly, so stretching the rubber band also pulls the ant to the right. The initial length of the rubber band is n inches, so after t seconds, the rubber band is $n + t$ inches long.

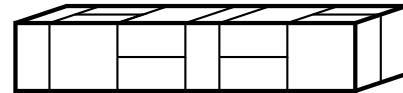


Every second, the ant walks an inch, and then the rubber band is stretched an inch longer.

- (a) How far has the ant moved after t seconds, as a function of n and t ? Set up a recurrence and (for full credit) give an *exact* closed-form solution. [Hint: What *fraction* of the rubber band's length has the ant walked?]
- (b) How long does it take the ant to get to the right end of the rubber band? For full credit, give an answer of the form $f(n) + \Theta(1)$ for some explicit function $f(n)$.
9. (a) A *domino* is a 2×1 or 1×2 rectangle. How many different ways are there to completely fill a $2 \times n$ rectangle with n dominos? Set up a recurrence relation and give an *exact* closed-form solution.

²After all, *your* code is always completely 100% bug-free. Isn't that right, Mr. Gates?

- (b) A *slab* is a three-dimensional box with dimensions $1 \times 2 \times 2$, $2 \times 1 \times 2$, or $2 \times 2 \times 1$. How many different ways are there to fill a $2 \times 2 \times n$ box with n slabs? Set up a recurrence relation and give an *exact* closed-form solution.



A 2×10 rectangle filled with ten dominos, and a $2 \times 2 \times 10$ box filled with ten slabs.

10. Professor George O'Jungle has a favorite 26-node binary tree, whose nodes are labeled by letters of the alphabet. The preorder and postorder sequences of nodes are as follows:

preorder: M N H C R S K W T G D X I Y A J P O E Z V B U L Q F
 postorder: C W T K S G R H D N A O E P J Y Z I B Q L F U V X M

Draw Professor O'Jungle's binary tree, and give the inorder sequence of nodes.

11. Alice and Bob each have a fair n -sided die. Alice rolls her die once. Bob then repeatedly throws his die until he rolls a number at least as big as the number Alice rolled. Each time Bob rolls, he pays Alice \$1. (For example, if Alice rolls a 5, and Bob rolls a 4, then a 3, then a 1, then a 5, the game ends and Alice gets \$4. If Alice rolls a 1, then no matter what Bob rolls, the game will end immediately, and Alice will get \$1.)

Exactly how much money does Alice expect to win at this game? Prove that your answer is correct. If you have to appeal to ‘intuition’ or ‘common sense’, your answer is probably wrong!

12. Prove that for any nonnegative parameters a and b , the following algorithms terminate and produce identical output.

SLOWEUCLID(a, b) :

```

if  $b > a$ 
    return SLOWEUCLID( $b, a$ )
else if  $b = 0$ 
    return  $a$ 
else
    return SLOWEUCLID( $b, a - b$ )
  
```

FASTEUCLID(a, b) :

```

if  $b = 0$ 
    return  $a$ 
else
    return FASTEUCLID( $b, a \bmod b$ )
  
```

CS 373: Combinatorial Algorithms, Spring 2001

Homework 1 (due Thursday, February 1, 2001 at 11:59:59 p.m.)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

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. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or 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, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Suppose you are a simple shopkeeper living in a country with n different types of coins, with values $1 = c[1] < c[2] < \dots < c[n]$. (In the U.S., for example, $n = 6$ and the values are 1, 5, 10, 25, 50 and 100 cents.) Your beloved and benevolent dictator, El Generalissimo, has decreed that whenever you give a customer change, you must use the smallest possible number of coins, so as not to wear out the image of El Generalissimo lovingly engraved on each coin by servants of the Royal Treasury.
 - (a) In the United States, there is a simple greedy algorithm that always results in the smallest number of coins: subtract the largest coin and recursively give change for the remainder. El Generalissimo does not approve of American capitalist greed. Show that there is a set of coin values for which the greedy algorithm does *not* always give the smallest possible of coins.
 - (b) Describe and analyze a dynamic programming algorithm to determine, given a target amount A and a sorted array $c[1..n]$ of coin values, the smallest number of coins needed to make A cents in change. You can assume that $c[1] = 1$, so that it is possible to make change for any amount A .

2. Consider the following sorting algorithm:

```

STUPIDSORT( $A[0..n-1]$ ) :
  if  $n = 2$  and  $A[0] > A[1]$ 
    swap  $A[0] \leftrightarrow A[1]$ 
  else if  $n > 2$ 
     $m \leftarrow \lceil 2n/3 \rceil$ 
    STUPIDSORT( $A[0..m-1]$ )
    STUPIDSORT( $A[n-m..n-1]$ )
    STUPIDSORT( $A[0..m-1]$ )

```

- (a) Prove that STUPIDSORT actually sorts its input.
- (b) Would the algorithm still sort correctly if we replaced the line $m \leftarrow \lceil 2n/3 \rceil$ with $m \leftarrow \lfloor 2n/3 \rfloor$? Justify your answer.
- (c) State a recurrence (including the base case(s)) for the number of comparisons executed by STUPIDSORT.
- (d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?
- *(e) Show that the number of *swaps* executed by STUPIDSORT is at most $\binom{n}{2}$.

3. The following randomized algorithm selects the r th 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( $A[1..n], r$ ) :
   $p \leftarrow \text{RANDOM}(1, n)$ 
   $k \leftarrow \text{PARTITION}(A[1..n], p)$ 
  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 n and r .
- (b) What is the *exact* probability that RANDOMSELECT compares the i th smallest and j th 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?

4. What excitement! The Champaign Spinners and the Urbana Dreamweavers have advanced to meet each other in the World Series of Basketweaving! The World Champions will be decided by a best-of- $2n - 1$ series of head-to-head weaving matches, and the first to win n matches will take home the coveted Golden Basket (for example, a best-of-7 series requiring four match wins, but we will keep the generalized case). We know that for any given match there is a constant probability p that Champaign will win, and a subsequent probability $q = 1 - p$ that Urbana will win.

Let $P(i, j)$ be the probability that Champaign will win the series given that they still need i more victories, whereas Urbana needs j more victories for the championship. $P(0, j) = 1$, $1 \leq j \leq n$, because Champaign needs no more victories to win. $P(i, 0) = 0$, $1 \leq i \leq n$, as Champaign cannot possibly win if Urbana already has. $P(0, 0)$ is meaningless. Champaign wins any particular match with probability p and loses with probability q , so

$$P(i, j) = p \cdot P(i - 1, j) + q \cdot P(i, j - 1)$$

for any $i \geq 1$ and $j \geq 1$.

Create and analyze an $O(n^2)$ -time dynamic programming algorithm that takes the parameters n , p and q and returns the probability that Champaign will win the series (that is, calculate $P(n, n)$).

5. The traditional Devonian/Cornish drinking song “The Barley Mow” has the following pseudolyrics¹, where $\text{container}[i]$ is the name of a container that holds 2^i ounces of beer.²

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

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

for  $i \leftarrow 1$  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!"
        "Here's a health to the barley-mow, my brave boys,"
        "Here's a health to the barley-mow!"
```

- (a) Suppose each container name $\text{container}[i]$ is a single word, and you can sing four words a second. How long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound.)
- (b) If you want to sing this song for $n > 20$, you’ll have to make up your own container names, and to avoid repetition, these names will get progressively longer as n increases³. Suppose $\text{container}[n]$ 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 $\text{container}[i]$. Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang BARLEYMOW(n)? (Give an *exact* answer, not just an asymptotic bound.)

¹Pseudolyrics are to lyrics as pseudocode is to code.

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

³“We'll drink it out of the hemisemidemiyottapint, boys!”

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.]

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

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

Practice Problems

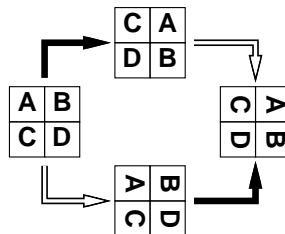
1. Give an $O(n^2)$ algorithm to find the longest increasing subsequence of a sequence of numbers. The elements of the subsequence need not be adjacent in the sequence. For example, the sequence $\langle 1, 5, 3, 2, 4 \rangle$ has longest increasing subsequence $\langle 1, 3, 4 \rangle$.

2. You are at a political convention with n delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party a delegate belongs to. However, you can check whether any two delegates are in the *same* party or not by introducing them to each other. (Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.)
 - (a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.
 - (b) Suppose exactly k political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)

3. Give an algorithm that finds the *second* smallest of n elements in at most $n + \lceil \lg n \rceil - 2$ comparisons. [Hint: divide and conquer to find the smallest; where is the second smallest?]

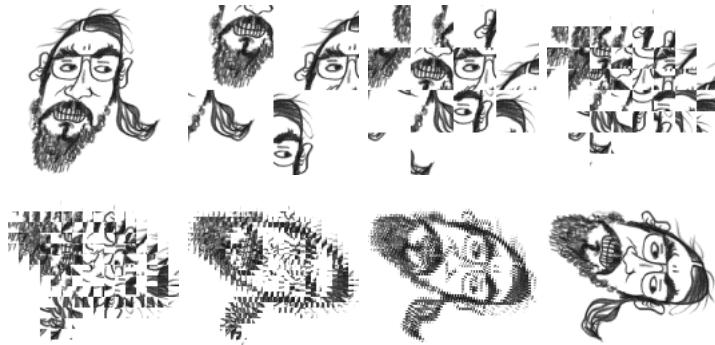
4. Some graphics hardware includes support for an operation called *blit*, or **block transfer**, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

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



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

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



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

- (a) Prove that both versions of the algorithm are correct. [Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]
 - (b) *Exactly* how many blits does the algorithm perform?
 - (c) What is the algorithm's running time if a $k \times k$ blit takes $O(k^2)$ time?
 - (d) What if a $k \times k$ blit takes only $O(k)$ time?
-
5. A company is planning a party for its employees. The employees in the company are organized into a strict hierarchy, that is, a tree with the company president at the root. The organizers of the party have assigned a real number to each employee measuring how ‘fun’ the employee is. In order to keep things social, there is one restriction on the guest list: an employee cannot attend the party if their immediate supervisor is present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it’s her company, after all. Give an algorithm that makes a guest list for the party that maximizes the sum of the ‘fun’ ratings of the guests.

 6. Suppose you have a subroutine that can find the median of a set of n items (*i.e.*, the $\lfloor n/2 \rfloor$ smallest) in $O(n)$ time. Give an algorithm to find the k th biggest element (for arbitrary k) in $O(n)$ time.

 7. You’re walking along the beach and you stub your toe on something in the sand. You dig around it and find that it is a treasure chest full of gold bricks of different (integral) weight. Your knapsack can only carry up to weight n before it breaks apart. You want to put as much in it as possible without going over, but you *cannot* break the gold bricks up.
 - (a) Suppose that the gold bricks have the weights $1, 2, 4, 8, \dots, 2^k$, $k \geq 1$. Describe and prove correct a greedy algorithm that fills the knapsack as much as possible without going over.
 - (b) Give a set of 3 weight values for which the greedy algorithm does *not* yield an optimal solution and show why.
 - (c) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of gold brick values.

CS 373: Combinatorial Algorithms, Spring 2001

Homework 2 (due Thu. Feb. 15, 2001 at 11:59 PM)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

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. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or 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, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

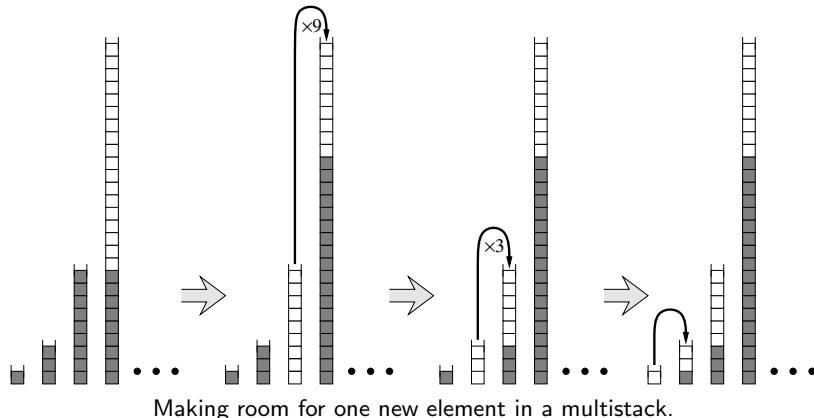
Required Problems

1. Suppose we are given two sorted arrays $A[1..n]$ and $B[1..n]$ and an integer k . Describe an algorithm to find the k th 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. For full credit, your algorithm should run in $\Theta(\log n)$ time. [Hint: First try to solve the special case $k = n$.]
2. Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.
 - (a) Show that a rotation in an augmented binary tree can be performed in constant time.
 - (b) Describe an algorithm SCAPEGOATSELECT(k) that selects the k th smallest item in an augmented scapegoat tree in $O(\log n)$ worst-case time.
 - (c) Describe an algorithm SPLAISELECT(k) that selects the k th smallest item in an augmented splay tree in $O(\log n)$ amortized time.

- (d) Describe an algorithm TREAPSELECT(k) that selects the k th smallest item in an augmented treap in $O(\log n)$ *expected* time.
3. (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.
4. Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:
- After an insertion, if the table is more than $3/4$ full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
 - After a deletion, if the table is less than $1/4$ full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method (it makes it much more difficult).

5. A *multistack* consists of an infinite series of stacks S_0, S_1, S_2, \dots , where the i th 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.



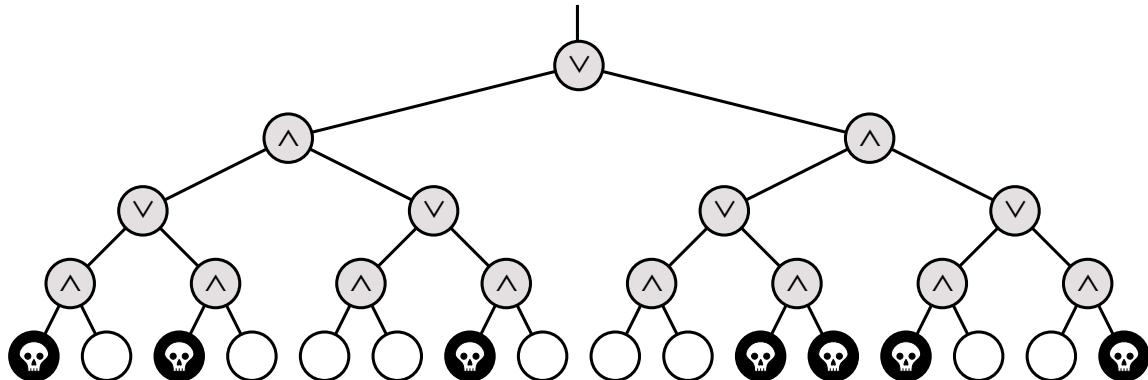
- (a) [1 point] In the worst case, how long does it take to push one more element onto a multistack containing n elements?
- (b) [9 points] Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack. You can use any method you like.

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.]

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) (2 pts) Describe and analyze a deterministic algorithm to determine whether or not you can win. [Hint: This is easy!]
- (b) (8 pts) 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 $n = 1$.]



Practice Problems

1. (a) Show that it is possible to transform any n -node binary search tree into any other n -node binary search tree using at most $2n - 2$ rotations.

*(b) Use fewer than $2n - 2$ rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most $2n - 6$ rotations, and there are pairs of trees that are $2n - 10$ rotations apart. These are the best bounds known.

2. Faster Longest Increasing Subsequence(LIS)

Give an $O(n \log n)$ algorithm to find the longest increasing subsequence of a sequence of numbers. [Hint: In the dynamic programming solution, you don't really have to look back at all previous items. There was a practice problem on HW 1 that asked for an $O(n^2)$ algorithm for this. If you are having difficulty, look at the solution provided in the HW 1 solutions.]

3. Amortization

- (a) Modify the binary double-counter (see class notes Sept 12) to support a new operation SIGN, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

[Hint: Suppose p is the number of significant bits in P , and n is the number of significant bits in N . For example, if $P = 17 = 10001_2$ and $N = 0$, then $p = 5$ and $n = 0$. Then $p - n$ always has the same sign as $P - N$. Assume you can update p and n in $O(1)$ time.]

- *(b) Do the same but now you can't assume that p and n can be updated in $O(1)$ time.

- *4. Amortization

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 i th least significant fit indicates whether the sum includes the i th 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. This is *not* the same representation as on Homework 0.]

5. Detecting overlap

- (a) You are given a list of ranges represented by min and max (*e.g.*, [1,3], [4,5], [4,9], [6,8], [7,10]). Give an $O(n \log n)$ -time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.

- (b) You are given a list of rectangles represented by min and max x - and y -coordinates. Give an $O(n \log n)$ -time algorithm that decides whether or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). [Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.]

6. Comparison of Amortized Analysis Methods

A sequence of n operations is performed on a data structure. The i th 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, Spring 2001

Homework 3 (due Thursday, March 8, 2001 at 11:59.99 p.m.)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

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. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or 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, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Hashing:

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

- Assuming uniform hashing, show that for $i = 1, 2, \dots, n$, the probability that the i^{th} insertion requires strictly more than k probes is at most 2^{-k} .
- Show that for $i = 1, 2, \dots, n$, the probability that the i^{th} 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 i^{th} 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.

- Show that $\Pr\{X > 2 \lg n\} \leq 1/n$.
- Show that the expected length of the longest probe sequence is $E[X] = O(\lg n)$.

2. 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 $0 \leq r(u, v) \leq 1$. 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 .

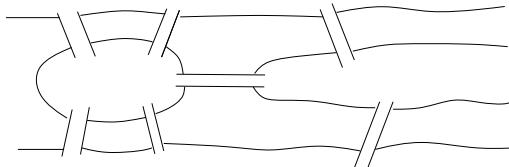
3. Aerophobia:

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).

4. 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.
5. 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 .

6. Nesting Envelopes

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

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. Let the hash function for a table of size m be

$$h(x) = \lfloor Ax \rfloor \bmod m$$

where $A = \hat{\phi} = \frac{\sqrt{5}-1}{2}$. Show that this gives the best possible spread, i.e. if the x are hashed in order, $x + 1$ will be hashed in the largest remaining contiguous interval.

3. The incidence matrix of an undirected graph $G = (V, E)$ is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = \begin{cases} 1 & (i, j) \in E, \\ 0 & (i, j) \notin E. \end{cases}$$

(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^TB 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).

4. $o(V^2)$ 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.

5. 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 .

6. 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 .

7. Minimum Spanning Tree:

Suppose all edge weights in a graph G are equal. Give an algorithm to compute an MST.

8. 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} .

9. (a) What is $5^{2^{29}5^0 + 23^41 + 17^32 + 11^23 + 5^14} \pmod{6}$?

- (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, Spring 2001

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 4 (due Thu. March 29, 2001 at 11:59:59 pm)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

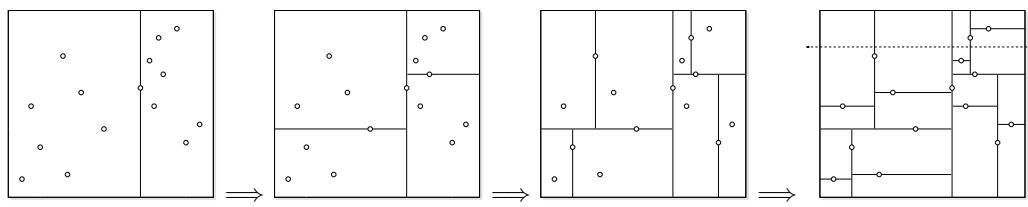
Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

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. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

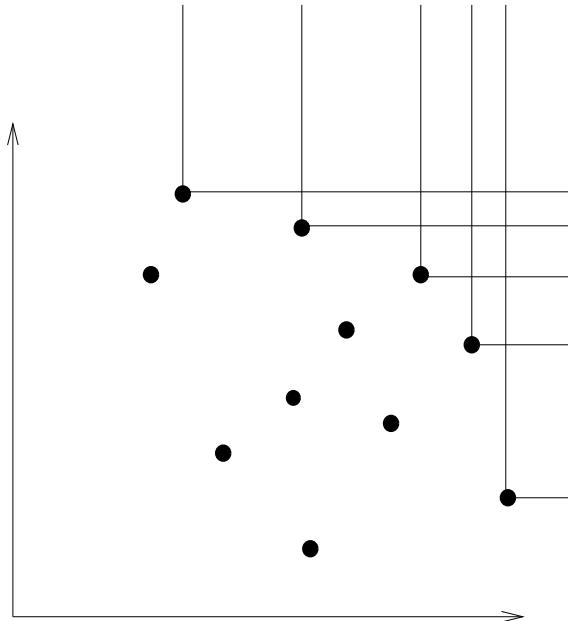
Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

- Suppose we have n points scattered inside a two-dimensional box. A *kd-tree* recursively subdivides the rectangle as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line partitions the rest of the interior points *as evenly as possible* by passing through a median point inside the box (*not* on the boundary). If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.



Successive divisions of a kd-tree for 15 points. The dashed line crosses four cells.



An example staircase as in problem 3.

- (a) How many cells are there, as a function of n ? Prove your answer is correct.
- (b) In the worst case, *exactly* how many cells can a horizontal line cross, as a function of n ? Prove your answer is correct. Assume that $n = 2^k - 1$ for some integer k .
- (c) Suppose we have n points stored in a kd-tree. Describe an algorithm that counts the number of points above a horizontal line (such as the dashed line in the figure) in $O(\sqrt{n})$ time.
- *(d) [Optional: 5 pts extra credit] Find an algorithm that counts the number of points that lie inside a rectangle R and show that it takes $O(\sqrt{n})$ time. You may assume that the sides of the rectangle are parallel to the sides of the box.

2. Circle Intersection [*This problem is worth 20 points*]

Describe an algorithm to decide, given n circles in the plane, whether any two of them intersect, in $O(n \log n)$ time. Each circle is specified by three numbers: its radius and the x - and y -coordinates of its center.

We only care about intersections between circle boundaries; concentric circles do not intersect. What general position assumptions does your algorithm require? [Hint: Modify an algorithm for detecting line segment intersections, but describe your modifications very carefully! There are at least two very different solutions.]

3. Staircases

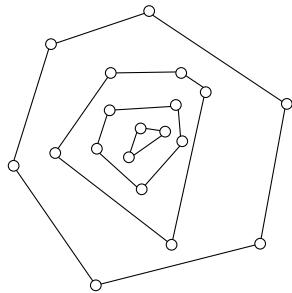
You are given a set of points in the first quadrant. A *left-up* point of this set is defined to be a point that has no points both greater than it in both coordinates. The left-up subset of a set of points then forms a *staircase* (see figure).

- (a) Prove that left-up points do not necessarily lie on the convex hull.
- (b) Give an $O(n \log n)$ algorithm to find the staircase of a set of points.

- (c) Assume that points are chosen uniformly at random within a rectangle. What is the average number of points in a staircase? Justify. Hint: you will be able to give an exact answer rather than just asymptotics. You have seen the same analysis before.
4. Convex Layers

Given a set Q of points in the plane, define the *convex layers* of Q inductively as follows: The first convex layer of Q is just the convex hull of Q . For all $i > 1$, the i th convex layer is the convex hull of Q after the vertices of the first $i - 1$ layers have been removed.

Give an $O(n^2)$ -time algorithm to find all convex layers of a given set of n points.



A set of points with four convex layers.

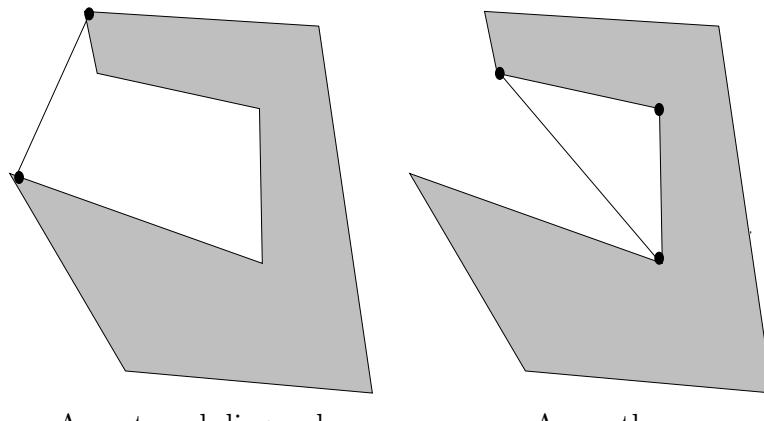
5. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.] Solve the travelling salesman problem for points in convex position (ie, the vertices of a convex polygon). Finding the shortest cycle that visits every point is easy – it's just the convex hull. Finding the shortest path that visits every point is a little harder, because the path can cross through the interior.

- (a) Show that the optimal path cannot be one that crosses itself.
 (b) Describe an $O(n^2)$ time dynamic programming algorithm to solve the problem.

Practice Problems

1. Basic Computation (assume two dimensions and *exact* arithmetic)
 - (a) Intersection: Extend the basic algorithm to determine if two line segments intersect by taking care of *all* degenerate cases.
 - (b) Simplicity: Give an $O(n \log n)$ algorithm to determine whether an n -vertex polygon is simple.
 - (c) Area: Give an algorithm to compute the area of a simple n -polygon (not necessarily convex) in $O(n)$ time.
 - (d) Inside: Give an algorithm to determine whether a point is within a simple n -polygon (not necessarily convex) in $O(n)$ time.

2. External Diagonals and Mouths
 - (a) A pair of polygon vertices defines an *external diagonal* if the line segment between them is completely outside the polygon. Show that every nonconvex polygon has at least one external diagonal.
 - (b) Three consecutive polygon vertices p, q, r form a *mouth* if p and r define an external diagonal. Show that every nonconvex polygon has at least one mouth.



3. On-Line Convex Hull

We are given the set of points one point at a time. After receiving each point, we must compute the convex hull of all those points so far. Give an algorithm to solve this problem in $O(n^2)$ (We could obviously use Graham's scan n times for an $O(n^2 \log n)$ algorithm). Hint: How do you maintain the convex hull?

4. Another On-Line Convex Hull Algorithm
 - (a) Given an n -polygon and a point outside the polygon, give an algorithm to find a tangent.
 - *(b) Suppose you have found both tangents. Give an algorithm to remove the points from the polygon that are within the angle formed by the tangents (as segments!) and the opposite side of the polygon.

- (c) Use the above to give an algorithm to compute the convex hull on-line in $O(n \log n)$
5. Order of the size of the convex hull
The convex hull on $n \geq 3$ points can have anywhere from 3 to n points. The average case depends on the distribution.
- (a) Prove that if a set of points is chosen randomly within a given rectangle then the average size of the convex hull is $O(\log n)$.
 - ★(b) Prove that if a set of points is chosen randomly within a given circle then the average size of the convex hull is $O(n^{1/3})$.
6. Ghostbusters and Ghosts
A group of n ghostbusters is battling n ghosts. Each ghostbuster can shoot a single energy beam at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits a ghost. The ghostbusters must all fire at the same time and no two energy beams may cross (it would be bad). The positions of the ghosts and ghostbusters is fixed in the plane (assume that no three points are collinear).
- (a) Prove that for any configuration of ghosts and ghostbusters there exists such a non-crossing matching.
 - (b) Show that there exists a line passing through one ghostbuster and one ghost such that the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Give an efficient algorithm to find such a line.
 - (c) Give an efficient divide and conquer algorithm to pair ghostbusters and ghosts so that no two streams cross.

CS 373: Combinatorial Algorithms, Spring 2001

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 5 (due Tue. Apr. 17, 2001 at 11:59 pm)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 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, $\frac{3}{4}$ -unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Prove that finding the second smallest of n elements takes EXACTLY $n + \lceil \lg n \rceil - 2$ comparisons in the worst case. Prove for both upper and lower bounds. Hint: find the (first) smallest using an elimination tournament.
2. *Fibonacci strings* are defined as follows:

$$F_1 = "b", \quad F_2 = "a", \quad \text{and } F_n = F_{n-1}F_{n-2}, (n > 2)$$

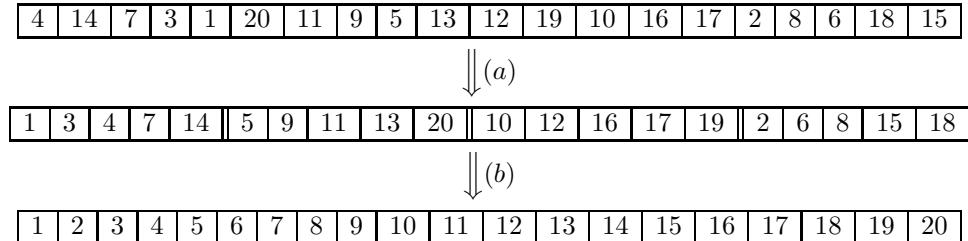
where the recursive rule uses concatenation of strings, so F_3 is “ab”, F_4 is “aba”. Note that the length of F_n is the n th Fibonacci number.

- (a) Prove that in any Fibonacci string there are no two b’s adjacent and no three a’s.
- (b) Give the unoptimized and optimized ‘prefix’ (fail) function for F_7 .
- (c) Prove that, in searching for the Fibonacci string F_k , the unoptimized KMP algorithm can shift $\lceil k/2 \rceil$ times in a row trying to match the last character of the pattern. In other words, prove that there is a chain of failure links $m \rightarrow \text{fail}[m] \rightarrow \text{fail}[\text{fail}[m]] \rightarrow \dots$ of length $\lceil k/2 \rceil$, and find an example text T that would cause KMP to traverse this entire chain at a single text position.

- (d) Prove that the unoptimized KMP algorithm can shift $k - 2$ times in a row at the same text position when searching for F_k . Again, you need to find an example text T that would cause KMP to traverse this entire chain on the same text character.
- (e) How do the failure chains in parts (c) and (d) change if we use the optimized failure function instead?

3. Two-stage sorting

- (a) Suppose we are given an array $A[1..n]$ of distinct integers. Describe an algorithm that splits A into n/k subarrays, each with k elements, such that the elements of each subarray $A[(i-1)k+1..ik]$ are sorted. Your algorithm should run in $O(n \log k)$ time.
- (b) Given an array $A[1..n]$ that is already split into n/k sorted subarrays as in part (a), describe an algorithm that sorts the entire array in $O(n \log(n/k))$ time.
- (c) Prove that your algorithm from part (a) is optimal.
- (d) Prove that your algorithm from part (b) is optimal.

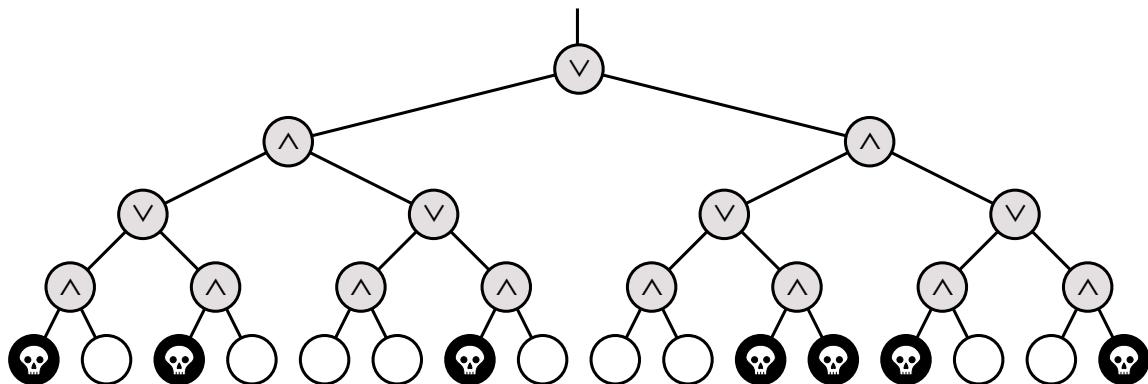


4. Show how to extend the Rabin-Karp fingerprinting method to handle the problem of looking for a given $m \times m$ pattern in an $n \times n$ array of characters. (The pattern may be shifted horizontally and vertically, but it may not be rotated.)

5. Death knocks on your door once more on a warm spring day. He remembers that you are an algorithms student and that you soundly defeated him last time and are now living out your immortality. Death is in a bit of a quandry. He has been losing a lot and doesn't know why. He wants you to prove a lower bound on your deterministic algorithm so that he can reap more souls. If you have forgotten, the game goes like this: It is a complete binary tree with 4^n leaves, each colored black or white. There is a token at the root of the tree. To play the game, you and Death took turns moving the token from its current node to one of its children. The game ends after $2n$ moves, when the token lands on a leaf. If the final leaf is black, the player dies; if it's white, you will live forever. You move first, so Death gets the last turn.

You decided 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've challenged Death to a game of Twister instead.

Prove that *any* deterministic algorithm must examine *every* leaf of the tree in the worst case. Since there are 4^n leaves, this implies that any deterministic algorithm must take $\Omega(4^n)$ time in the worst case. Use an adversary argument, or in other words, assume Death cheats.



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.]

Lower Bounds on Adjacency Matrix Representations of Graphs

- Prove that the time to determine if an undirected graph has a cycle is $\Omega(V^2)$.
- Prove that the time to determine if there is a path between two nodes in an undirected graph is $\Omega(V^2)$.

Practice Problems

- String matching with wild-cards

Suppose you have an alphabet for patterns that includes a ‘gap’ or wild-card character; any length string of any characters can match this additional character. For example if ‘*’ is the wild-card, then the pattern ‘foo*bar*nad’ can be found in ‘foofoowangbarnad’. Modify the computation of the prefix function to correctly match strings using KMP.

2. Prove that there is no comparison sort whose running time is linear for at least $1/2$ of the $n!$ inputs of length n . What about at least $1/n$? What about at least $1/2^n$?
3. Prove that $2n - 1$ comparisons are necessary in the worst case to merge two sorted lists containing n elements each.
4. Find asymptotic upper and lower bounds to $\lg(n!)$ without Stirling's approximation (Hint: use integration).
5. Given a sequence of n elements of n/k blocks (k elements per block) all elements in a block are less than those to the right in sequence, show that you cannot have the whole sequence sorted in better than $\Omega(n \lg k)$. Note that the entire sequence would be sorted if each of the n/k blocks were individually sorted in place. Also note that combining the lower bounds for each block is not adequate (that only gives an upper bound).
6. Show how to find the occurrences of pattern P in text T by computing the prefix function of the string PT (the concatenation of P and T).

CS 373: Combinatorial Algorithms, Spring 2001

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 6 (due Tue. May 1, 2001 at 11:59.99 p.m.)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

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. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or 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, 3/4-unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Note: You will be held accountable for the appropriate responses for answers (e.g. give models, proofs, analyses, etc). For NP-complete problems you should prove everything rigorously, i.e. for showing that it is in NP, give a description of a certificate and a poly time algorithm to verify it, and for showing NP-hardness, you must show that your reduction is polytime (by similarly proving something about the algorithm that does the transformation) and proving both directions of the ‘if and only if’ (a solution of one is a solution of the other) of the many-one reduction.

Required Problems

1. Complexity
 - (a) Prove that $P \subseteq \text{co-NP}$.
 - (b) Show that if $NP \neq \text{co-NP}$, then *every* NP-complete problem is *not* a member of co-NP.
2. 2-CNF-SAT
Prove that deciding satisfiability when all clauses have at most 2 literals is in P.
3. Graph Problems

(a) SUBGRAPH-ISOMORPHISM

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

(b) LONGEST-PATH

Show that the problem of deciding whether an unweighted undirected graph has a path of length greater than k is NP-complete.

4. PARTITION, SUBSET-SUM

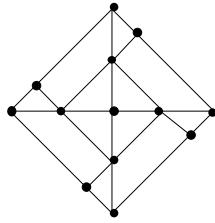
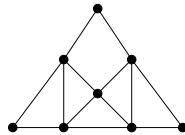
PARTITION is the problem of deciding, given a set of numbers, whether there exists a subset whose sum equals the sum of the complement, i.e. given $S = s_1, s_2 \dots, s_n$, does there exist a subset S' such that $\sum_{s \in S'} s = \sum_{t \in S - S'} t$. SUBSET-SUM is the problem of deciding, given a set of numbers and a target sum, whether there exists a subset whose sum equals the target, i.e. given $S = s_1, s_2 \dots, s_n$ and k , does there exist a subset S' such that $\sum_{s \in S'} s = k$. Give two reduction, one in both directions.

5. BIN-PACKING Consider the bin-packing problem: given a finite set U of n items and the positive integer size $s(u)$ of each item $u \in U$, can U be partitioned into k disjoint sets U_1, \dots, U_k such that the sum of the sizes of the items in each set does not exceed B ? Show that the bin-packing problem is NP-Complete. [Hint: Use the result from the previous problem.]

6. 3SUM

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

Describe an algorithm that solves the following problem as quickly as possible: Given a set of n numbers, does it contain three elements whose sum is zero? For example, your algorithm should answer TRUE for the set $\{-5, -17, 7, -4, 3, -2, 4\}$, since $-5 + 7 + (-2) = 0$, and FALSE for the set $\{-6, 7, -4, -13, -2, 5, 13\}$.

**Figure 1.** Gadget for PLANAR-3-COLOR.**Figure 2.** Gadget for DEGREE-4-PLANAR-3-COLOR.

Practice Problems

1. 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. Justify (exhibit a set that cannot be done in one less comparisons). Do the same for 6 numbers.

2. EXACT-COVER-BY-4-SETS

The EXACT-COVER-BY-3-SETS problem is defined as the following: given a finite set X with $|X| = 3q$ and a collection C of 3-element subsets of X , does C contain an *exact cover* for X , that is, a subcollection $C' \subseteq C$ such that every element of X occurs in exactly one member of C' ?

Given that EXACT-COVER-BY-3-SETS is NP-complete, show that EXACT-COVER-BY-4-SETS is also NP-complete.

3. PLANAR-3-COLOR

Using 3-COLOR, and the ‘gadget’ in figure 3, prove that the problem of 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.

4. DEGREE-4-PLANAR-3-COLOR

Using the previous result, and the ‘gadget’ in figure 4, prove that the problem of deciding whether a planar graph with no vertex of degree greater than four 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. Poly time subroutines can lead to exponential algorithms

Show that an algorithm that makes at most a constant number of calls to polynomial-time subroutines runs in polynomial time, but that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.

6. (a) Prove that if G is an undirected bipartite graph with an odd number of vertices, then G is nonhamiltonian. Give a polynomial time algorithm for finding a **hamiltonian cycle** in an undirected bipartite graph or establishing that it does not exist.
(b) Show that the **hamiltonian-path** problem can be solved in polynomial time on directed acyclic graphs by giving an efficient algorithm for the problem.
(c) Explain why the results in previous questions do not contradict the facts that both HAM-CYCLE and HAM-PATH are NP-complete problems.
7. 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?
- ★8. GRAPH-ISOMORPHISM
Consider the problem of deciding whether one graph is isomorphic to another.
 - (a) Give a brute force algorithm to decide this.
 - (b) Give a dynamic programming algorithm to decide this.
 - (c) Give an efficient probabilistic algorithm to decide this.
 - (d) Either prove that this problem is NP-complete, give a poly time algorithm for it, or prove that neither case occurs.
9. Prove that PRIMALITY (Given n , is n prime?) is in $\text{NP} \cap \text{co-NP}$. 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 knowing this tree of primitive roots can be checked to be correct and used to show that n is prime, and that this check takes poly time.
10. How much wood would a woodchuck chuck if a woodchuck could chuck wood?

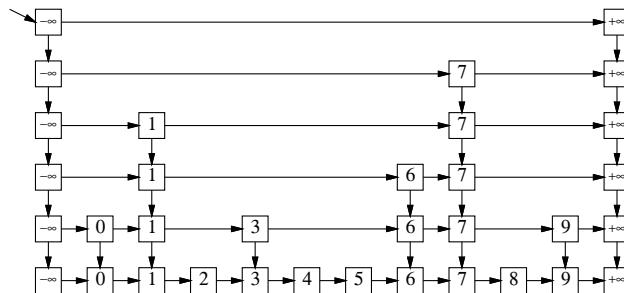
Write your answers in the separate answer booklet.

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

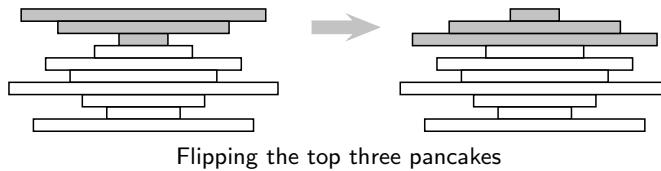
(a) $\Theta(1)$ (b) $\Theta(\log n)$ (c) $\Theta(n)$ (d) $\Theta(n \log n)$ (e) $\Theta(n^2)$

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, but each incorrect answer costs you $\frac{1}{2}$ point. You cannot score below zero.

- (a) What is $\sum_{i=1}^n H_i$?
 - (b) What is $\sum_{i=1}^{\lg n} 2^i$?
 - (c) How many digits do you need to write $n!$ in decimal?
 - (d) What is the solution of the recurrence $T(n) = 16T(n/4) + n$?
 - (e) What is the solution of the recurrence $T(n) = T(n - 2) + \lg n$?
 - (f) What is the solution of the recurrence $T(n) = 4T\left(\lceil \frac{n+51}{4} \rceil - \sqrt{n}\right) + 17n - 2^{8 \log^*(n^2)} + 6$?
 - (g) What is the worst-case running time of randomized quicksort?
 - (h) The expected time for inserting one item into a treap is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty treap?
 - (i) The amortized time for inserting one item into an n -node splay tree is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty splay tree?
 - (j) In the worst case, how long does it take to solve the traveling salesman problem for 10,000,000,000,000,000 cities?
2. What is the *exact* expected 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. A correct $\Theta()$ bound (with justification) is worth 5 points.



3. Suppose we have a stack of n pancakes of all different sizes. We want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation we can perform is a *flip* — insert a spatula under the top k pancakes, for some k between 1 and n , turn them all over, and put them back on top of the stack.



Flipping the top three pancakes

- (a) (3 pts) Describe an algorithm to sort an arbitrary stack of n pancakes using flips.
 - (b) (3 pts) Prove that your algorithm is correct.
 - (c) (2 pts) Exactly how many flips does your sorting algorithm perform in the worst case? A correct $\Theta()$ bound is worth one point.
 - (d) (2 pts) Suppose one side of each pancake is burned. Exactly how many flips do you need to sort the pancakes, so that the burned side of every pancake is on the bottom? A correct $\Theta()$ bound is worth one point.
4. Suppose we want to maintain a set of values in a data structure subject to the following operations:

- $\text{INSERT}(x)$: Add x to the set (if it isn't already there).
- $\text{DELETERANGE}(a, b)$: Delete every element x in the range $a \leq x \leq b$. For example, if the set was $\{1, 5, 3, 4, 8\}$, then $\text{DELETERANGE}(4, 6)$ would change the set to $\{1, 3, 8\}$.

Describe and analyze a data structure that supports these operations, such that the amortized cost of either operation is $O(\log n)$. [Hint: Use a data structure you saw in class. If you use the same INSERT algorithm, just say so—you don't need to describe it again in your answer.]

5. [1-unit grad students must answer this question.]

A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, ‘bananaananas’ is a shuffle of ‘banana’ and ‘ananas’ in several different ways.

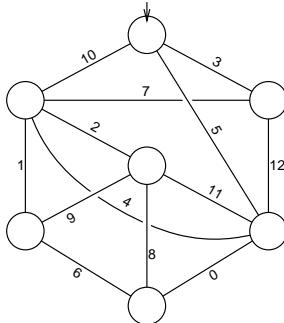
banana ananas ban ana ana nas b an an a na na s

The strings ‘prodgyrnammamiiincg’ and ‘dyprongarmammicing’ are both shuffles of ‘dynamic’ and ‘programming’:

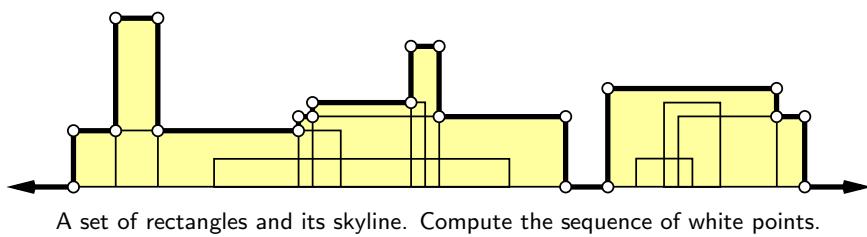
pro^dg y r nam ammi i n c g dy proⁿ g a r m amm ic ing

Given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, describe and analyze an algorithm to determine whether C is a shuffle of A and B . For full credit, your algorithm should run in $\Theta(mn)$ time.

1. Using any method you like, compute the following subgraphs for the weighted graph below. Each subproblem is worth 3 points. Each incorrect edge costs you 1 point, but you cannot get a negative score for any subproblem.
- a depth-first search tree, starting at the top vertex;
 - a breadth-first search tree, starting at the top vertex;
 - a shortest path tree, starting at the top vertex;
 - the **maximum** spanning tree.



2. (a) [4 pts] Prove that a connected acyclic undirected graph with V vertices has exactly $V - 1$ edges. ("It's a tree!" is not a proof.)
- (b) [4 pts] Describe and analyze an algorithm that determines whether a given undirected graph is a tree, where the graph is represented by an adjacency list.
- (c) [2 pts] What is the running time of your algorithm from part (b) if the graph is represented by an adjacency matrix?
3. Suppose we want to sketch the Manhattan skyline (minus the interesting bits like the Empire State and Chrysler buildings). You are given a set of n rectangles, each rectangle represented by its left and right x -coordinates and its height. The bottom of each rectangle is on the x -axis. Describe and analyze an efficient algorithm to compute the vertices of the skyline.



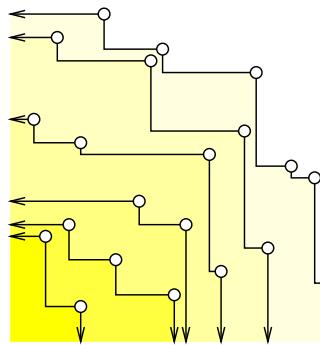
4. Suppose we model a computer network as a weighted undirected graph, where each vertex represents a computer and each edge represents a *direct* network connection between two computers. The weight of each edge represents the *bandwidth* of that connection—the number of bytes that can flow from one computer to the other in one second.¹ We want to implement a point-to-point network protocol that uses a single dedicated path to communicate between any pair of computers. Naturally, when two computers need to communicate, we should use the path with the highest bandwidth. The bandwidth of a *path* is the *minimum* bandwidth of its edges.

Describe an algorithm to compute the maximum bandwidth path between *every* pair of computers in the network. Assume that the graph is represented as an adjacency list.

5. [1-unit grad students must answer this question.]

Let P be a set of points in the plane. Recall that the *staircase* of P contains all the points in P that have no other point in P both above and to the right. We can define the *staircase layers* of P recursively as follows. The first staircase layer is just the staircase; for all $i > 1$, the i th staircase layer is the staircase of P after the first $i - 1$ staircase layers have been deleted.

Describe and analyze an algorithm to compute the staircase layers of P in $O(n^2)$ time.² Your algorithm should label each point with an integer describing which staircase layer it belongs to. You can assume that no two points have the same x - or y -coordinates.



A set of points and its six staircase layers.

¹Notice the bandwidth is symmetric; there are no cable modems or wireless phones. Don't worry about systems-level stuff like network load and latency. After all, this is a theory class!

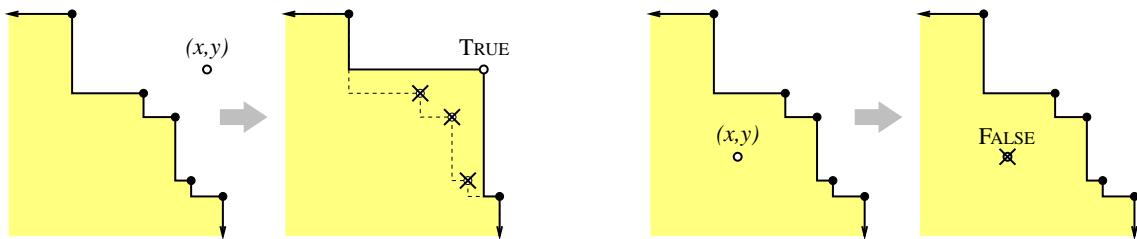
²This is *not* the fastest possible running time for this problem.

You must turn in this question sheet with your answers.**1. Déjà vu**

Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, and $17 = F_7 + F_4 + F_2$. You must give a complete, self-contained proof, not just a reference to the posted homework solutions.

2. L'esprit d'escalier

Recall that the *staircase* of a set of points consists of the points with no other point both above and to the right. Describe a method to maintain the staircase as new points are added to the set. Specifically, describe and analyze a data structure that stores the staircase of a set of points, and an algorithm $\text{INSERT}(x, y)$ that adds the point (x, y) to the set and returns TRUE or FALSE to indicate whether the staircase has changed. Your data structure should use $O(n)$ space, and your INSERT algorithm should run in $O(\log n)$ amortized time.

**3. Engage le jeu que je le gagne**

A palindrome is a text string that is exactly the same as its reversal, such as DEED, RACECAR, or SAIPPUAKAUPPIAS.¹

- Describe and analyze an algorithm to find the longest *prefix* of a given string that is also a palindrome. For example, the longest palindrome prefix of ILLINOISURBANACHAMPAIGN is ILLI, and the longest palindrome prefix of HYAKUGOJYUUICHI² is the single letter S. For full credit, your algorithm should run in $O(n)$ time.
- Describe and analyze an algorithm to find the length of the longest *subsequence* of a given string that is also a palindrome. For example, the longest palindrome subsequence of ILLINOISURBANACHAMPAIGN is NIAACAAIN (or NIAAHAIN), and the longest palindrome subsequence of HYAKUGOJYUUICHI is HUUUH³ (or HUGUH or HUYUH or...). You do not need to compute the actual subsequence; just its length. For full credit, your algorithm should run in $O(n^2)$ time.

¹Finnish for ‘soap dealer’.

²Japanese for ‘one hundred fifty-one’.

³English for ‘What the heck are you talking about?’

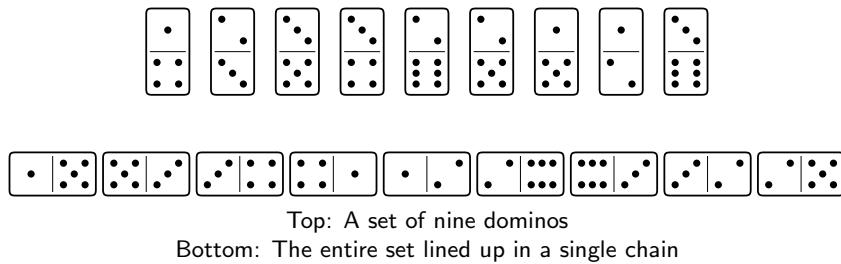
4. Toute votre base sont appartientent à nous

Prove that *exactly* $2n - 1$ comparisons are required in the worst case to merge two sorted arrays, each with n distinct elements. Describe and analyze an algorithm to prove the upper bound, and use an adversary argument to prove the lower bound. *You must give a complete, self-contained solution, not just a reference to the posted homework solutions.*⁴

5. Plus ça change, plus ça même chose

A domino is a 2×1 rectangle 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-hard, to determine whether a given set of n dominos can be lined up in a single chain. For example, for the set of dominos shown below, the correct output is TRUE.



6. Ceci n'est pas une pipe

Consider the following pair of problems:

- **BOXDEPTH:** Given a set of n axis-aligned rectangles in the plane and an integer k , decide whether any point in the plane is covered by k or more rectangles.
 - **MAXCLIQUE:** Given a graph with n vertices and an integer k , decide whether the graph contains a clique with k or more vertices.
- Describe and analyze a reduction of one of these problems to the other.
 - MAXCLIQUE is NP-hard. What does your answer to part (a) imply about the complexity of BOXDEPTH?

7. C'est magique! [1-unit graduate students must answer this question.]

The recursion fairy's cousin, the reduction genie, shows up one day with a magical gift for you—a box that determines in constant time the size of the largest clique in any given graph. (Recall that a clique is a subgraph where every pair of vertices is joined by an edge.) The magic box does not tell you *where* the largest clique is, only its size. Describe and analyze an algorithm to actually find the largest clique in a given graph **in polynomial time**, using this magic box.

⁴The posted solution for this Homework 5 practice problem was incorrect. So don't use it!

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).

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

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 very easy.]

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

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:

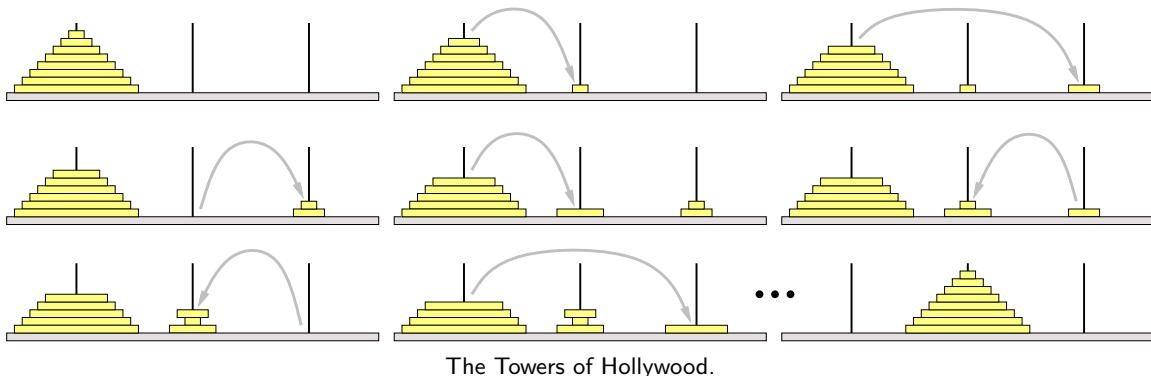
In any full binary tree, the number of leaves is exactly one more than the number of internal nodes.

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 n th 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 six different proofs.]

4. Most of you are probably familiar with the story behind the Tower of Hanoi 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.



Describe an algorithm³ 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.

¹The 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>

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

³Since 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: H H T

Guildenstern: H T H H

Rosencrantz: T

Guildenstern: (no flips)

Rosencrantz: H H H T

Guildenstern: T H H T H H T H T H H H

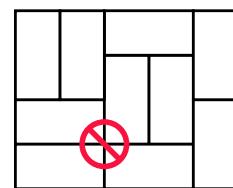
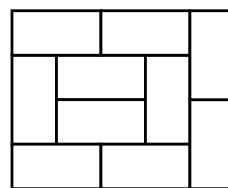
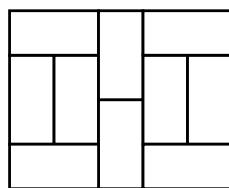
- (a) What is the expected number of flips in one of Rosencrantz's turns?
- (b) Suppose Rosencrantz flips k heads in a row on his turn. What is the expected number of flips in Guildenstern's next turn?
- (c) What is the expected total number of flips (by both Rosencrantz and Guildenstern) in a single game?

Prove 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**:

```
BOGUSCOUNTTOTENTOTHE( $n$ ):
  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.

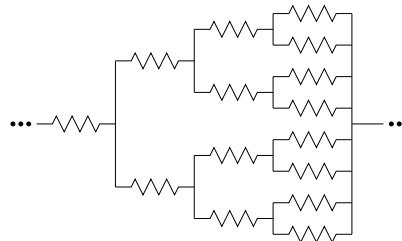
- The total resistance of two resistors in *series* is the sum of their individual resistances.
- The total resistance of two resistors in *parallel* is the reciprocal of the sum of the reciprocals of their individual resistances.

$$\dots - \begin{smallmatrix} \nearrow \\ \searrow \end{smallmatrix} \begin{smallmatrix} x \\ y \end{smallmatrix} - \begin{smallmatrix} \nearrow \\ \searrow \end{smallmatrix} - \dots = \dots - \begin{smallmatrix} \nearrow \\ \searrow \end{smallmatrix} \begin{smallmatrix} x+y \\ \end{smallmatrix} - \dots \quad \dots - \boxed{\begin{smallmatrix} \nearrow \\ \searrow \end{smallmatrix} \begin{smallmatrix} x \\ y \end{smallmatrix}} - \dots = \dots - \begin{smallmatrix} \nearrow \\ \searrow \end{smallmatrix} \frac{I}{I/x + I/y} - \dots$$

Equivalence laws for series-parallel resistor networks.

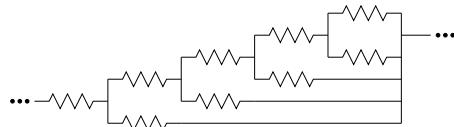
What is the *exact* total resistance⁴ of the following resistor networks as a function of n ? Prove your answers are correct. [Hint: Use induction. Duh.]

- (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.



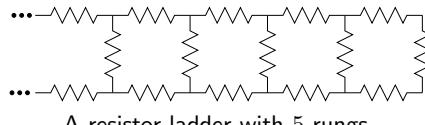
A balanced binary resistor tree with depth 3.

- (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.

⁴The 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

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 container¹ 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 $\text{BARLEYMOW}(n)$? (Give a tight asymptotic bound.)

¹One 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 $\text{container}[n]$ 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 $\text{container}[i]$. 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.
- Give a linear time algorithm to find the pair of the largest and smallest numbers in S .
 - Give an algorithm to sort S in $O(n \lg n)$ time.
3. Given a black and white pixel image $A[1 \dots m][1 \dots n]$, 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.

```

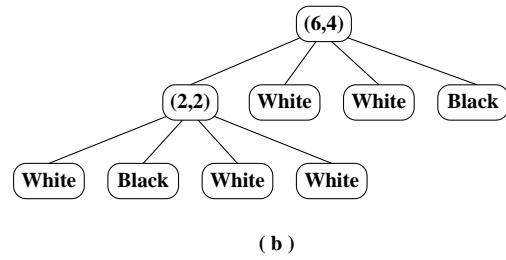
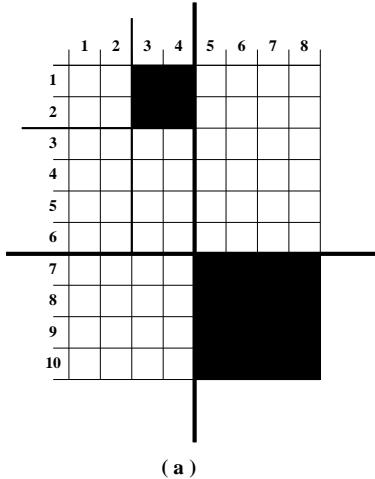
CONSTRUCTSEARCHTREE( $A[1 \dots m][1 \dots n]$ ):
  //Base Case
  if  $A$  contains only one color
    return a leaf node labeled with that color

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

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

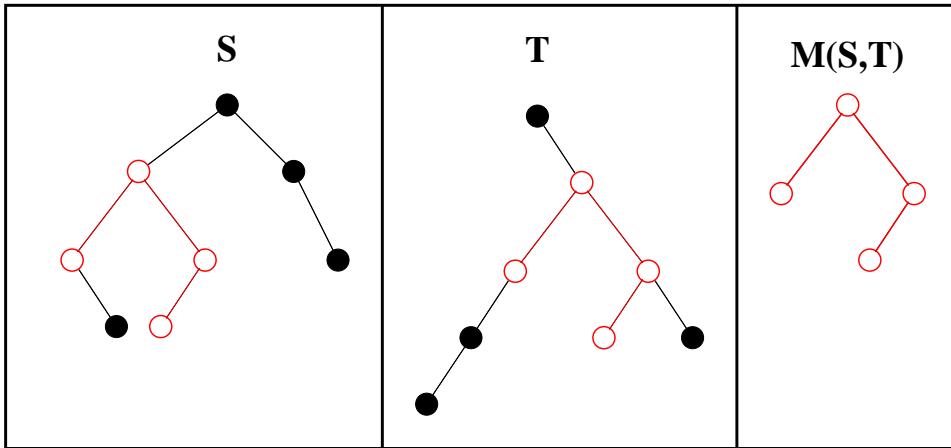
```

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 $1 \leq x \leq m$ and $1 \leq y \leq n$), 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($A[1 \dots m][1 \dots n]$).

4. Let A be a set of n positive integers, all of which are no greater than some constant $M > 0$. Give an $O(n^2M)$ 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.

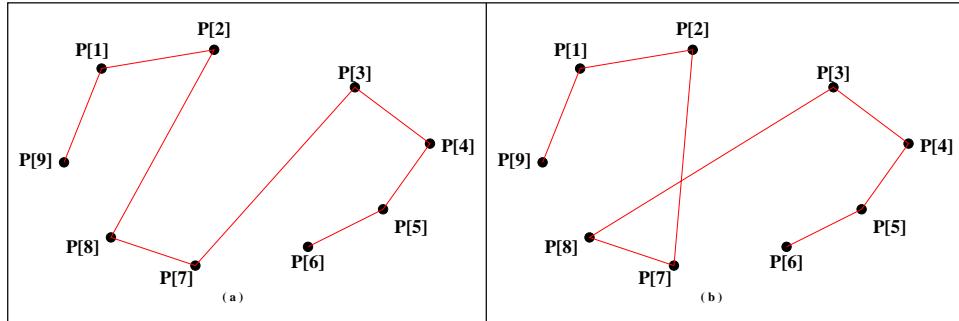


A *maximal* matching is a matching which contains at least as many vertices as any other matching. Give an algorithm to compute a maximal matching given the roots of two binary trees. Your algorithm should return the size of the match as well as the two roots of the matched subtrees of S and T .

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

Let $P[1, \dots, n]$ 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 $P[1], \dots, P[n]$ 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 $P[\lfloor \frac{n}{2} \rfloor]$.

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).

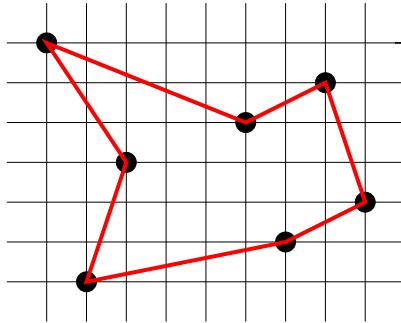
1. Suppose that you are given an $n \times n$ checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:
 - (a) the square immediately above,
 - (b)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

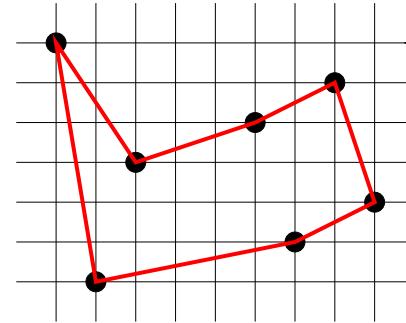
Give an algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. Your algorithm is free to pick any square along the bottom edge as a starting point and any square along the 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)



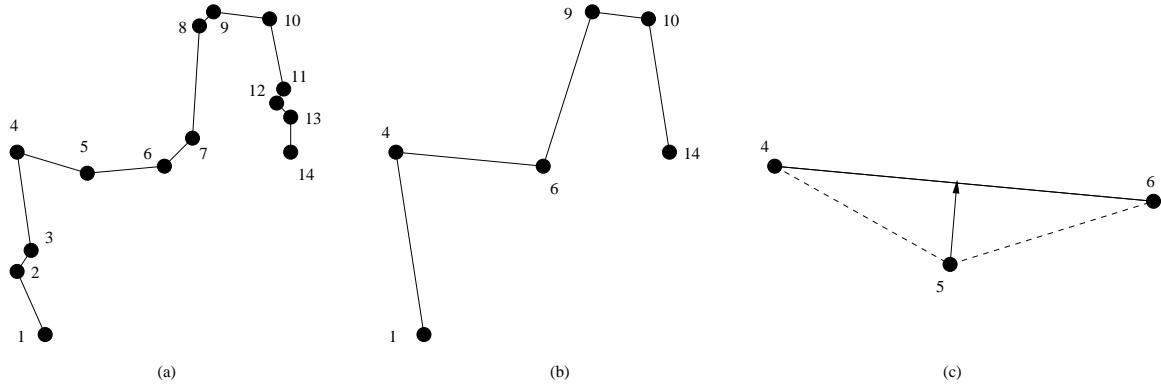
(b)

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. Its 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, \dots, 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_4p_6 . The error of p_5 is $\text{error}(p_5) = \text{dist}(p_5, p_4p_6)$.

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 $\{p_{i_1}, p_{i_2}, \dots, p_{i_k}\}$ from γ . It is required that $i_1 = 1$, $i_k = n$, and $i_j < i_{j+1}$ for $j = 1, 2, \dots, k - 1$.

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

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

Namely, for every vertex not in the simplification, its associated error, is the distance to the corresponding simplification 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 $\text{error}(\gamma, \gamma')$.

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  $i \leftarrow 1$  to  $n$ 
     $B[i] \leftarrow \text{Null}$ 
  for  $i \leftarrow 1$  to  $n$ 
    index  $\leftarrow \text{Random}(1,n)$ 
    while  $B[\text{index}] \neq \text{Null}$ 
      index  $\leftarrow \text{Random}(1,n)$ 
     $B[\text{index}] \leftarrow A[i]$ 
  for  $i \leftarrow 1$  to  $n$ 
     $A[i] \leftarrow B[i]$ 

```

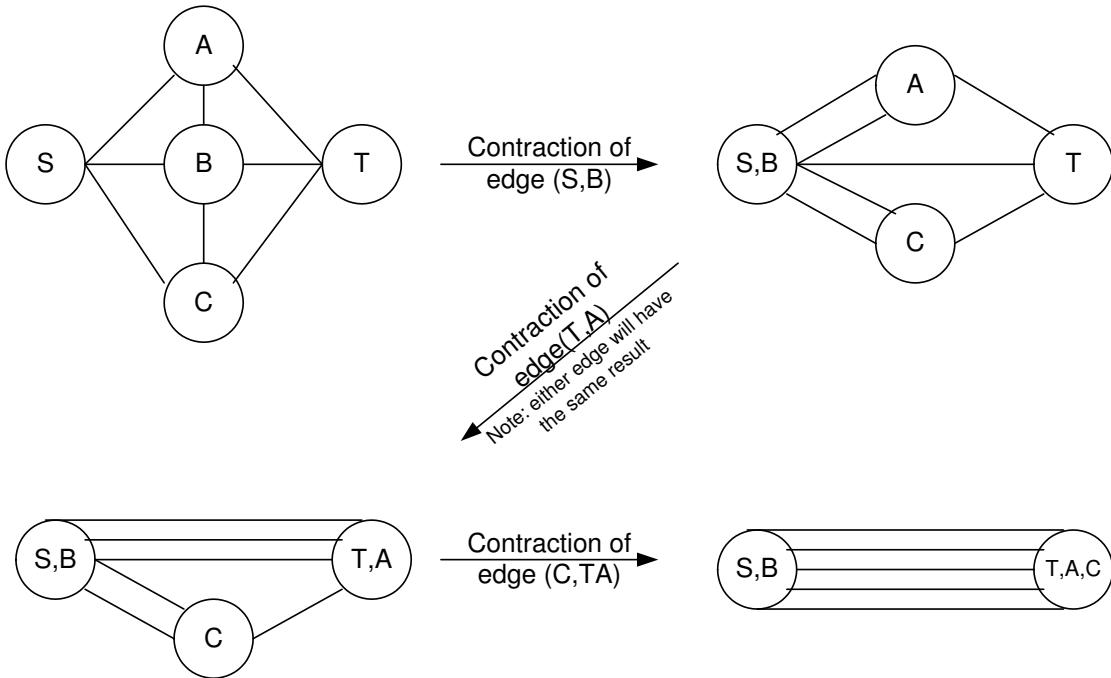
Suppose that $\text{Random}(i,j)$ will return a random number between i and j inclusive in constant time. **SLOWSHUFFLE** will shuffle the input array into a random order such that every permutation is equally likely.

- (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 separate 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 are 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 k th 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 r th 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( $A[1..n], r$ ) :  
     $p \leftarrow \text{RANDOM}(1, n)$   
     $k \leftarrow \text{PARTITION}(A[1..n], p)$   
    if  $r < k$   
        return RANDOMSELECT( $A[1..k - 1], r$ )  
    else if  $r > k$   
        return RANDOMSELECT( $A[k + 1..n], r - k$ )  
    else  
        return  $A[k]$ 
```

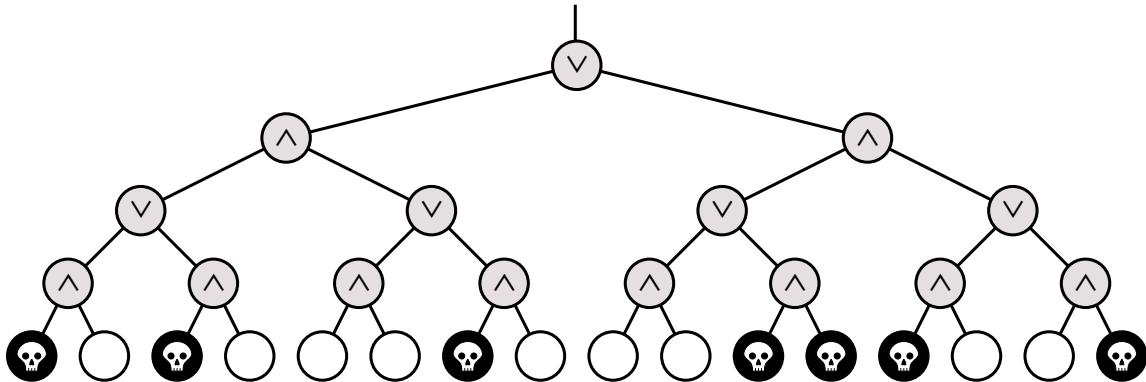
- (a) State a recurrence for the expected running time of RANDOMSELECT, as a function of both n and r .
- (b) What is the *exact* probability that RANDOMSELECT compares the i th smallest and j th smallest elements in the input array? The correct answer is a simple function of i , j , and r . [Hint: Check 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 k th 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 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:

 - $\text{PUSH}(x)$: adds x to the top of the stack.

- **POP:** removes the top element and returns it.

A queue is a FIFO/LIFO 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.

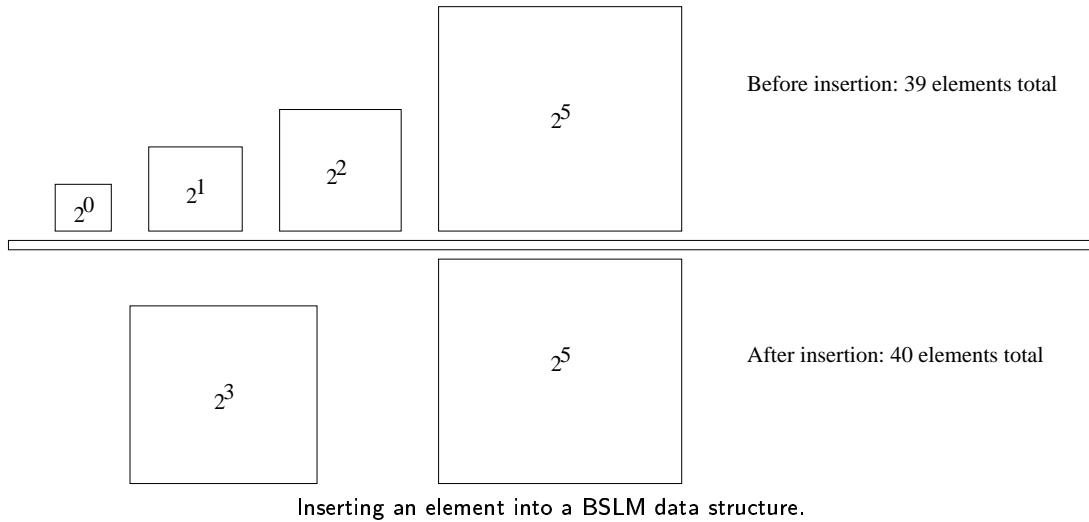
4. A data structure is *insertion-disabled* if there is no way to add elements to it. For the purposes of this problem, further assume that an insertion-disabled data structure implements the following operations with the given running times:

- **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 $k < i$.



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.

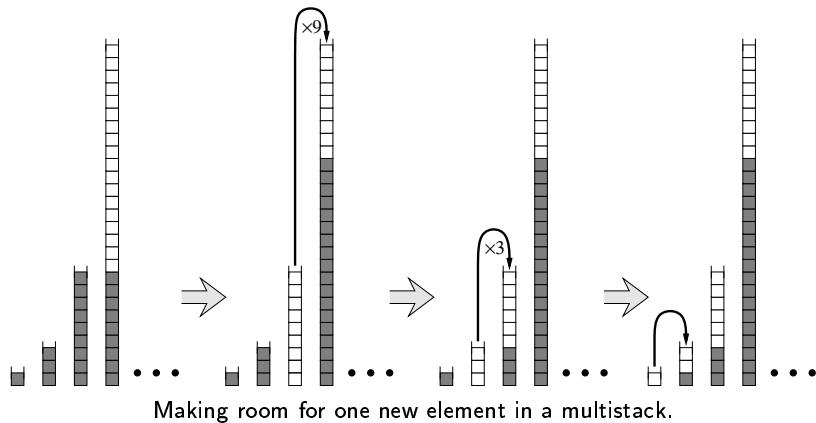
- (c) Now consider part b with a Union-Find data structure that uses path compression but does *not* use union by rank. Is $O(m)$ time still correct? Prove your answer.
6. [This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]

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 i th least significant fit indicates whether the sum includes the i th 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, \dots , where the i th 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.



- (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 $i = 1, 2, \dots, n$, the probability that the i^{th} insertion requires strictly more than k probes is at most 2^{-k} .
 - (b) Show that for $i = 1, 2, \dots, n$, the probability that the i^{th} 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 i^{th} 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 i th 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

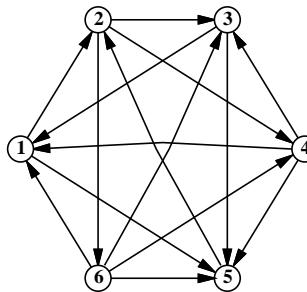
This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

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 specified 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 .

```

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  $D \leftarrow 2B - A$ 
 $T \leftarrow APD(B)$ 
 $X \leftarrow T \cdot A$ 
foreach  $x_{ij}$ 
    if  $x_{ij} \geq t_{ij} \cdot \text{degree}(j)$ 
         $d_{ij} \leftarrow 2t_{ij}$ 
    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.

¹The 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. Determining 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^TB 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 $0 \leq r(u, v) \leq 1$. 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 .

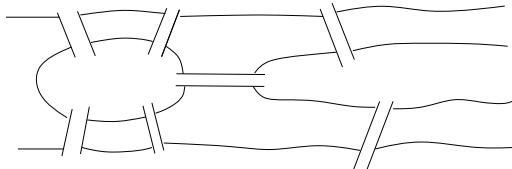
4. Aerophobia:

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 finds 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 $5^{2^{29}5^0 + 23^41 + 17^32 + 11^23 + 5^14} \pmod{6}$?

- (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

<http://www-courses.cs.uiuc.edu/~cs373>

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

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 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, $\frac{3}{4}$ -unit grad student, or 1-unit grad student by circling U, $\frac{3}{4}$, 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 $B[1] = A[i + 1], B[2] = A[i + 2], \dots, B[m] = A[i + m]$. 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 $\gcd(a, b) \cdot \text{lcm}(a, b) = ab$.
 - (b) Prove $\text{lcm}(a, b, c) = \text{lcm}(\text{lcm}(a, b), c)$.
 - (c) Prove $\gcd(a, b, c)\text{lcm}(ab, ac, bc) = abc$.
3. (5 points) Describe an efficient algorithm to compute multiplicative inverses modulo a prime p . Does your algorithm work if the modulus 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 integer vector $P[0..n-1]$. 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 n th 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 n th 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 can 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 $a, b, n \in \mathbb{Z} \setminus \{0\}$. Assume $\gcd(a, b) | n$. Prove the entire set of solutions to the equation

$$n = ax + by$$

is given by:

$$\Gamma = \left\{ x_0 + \frac{tb}{\gcd(a, b)}, y_0 - \frac{ta}{\gcd(a, b)} : t \in \mathbb{Z} \right\}.$$

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

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

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

For such an integer k , prove that

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

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

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 6 (Do not hand in!)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

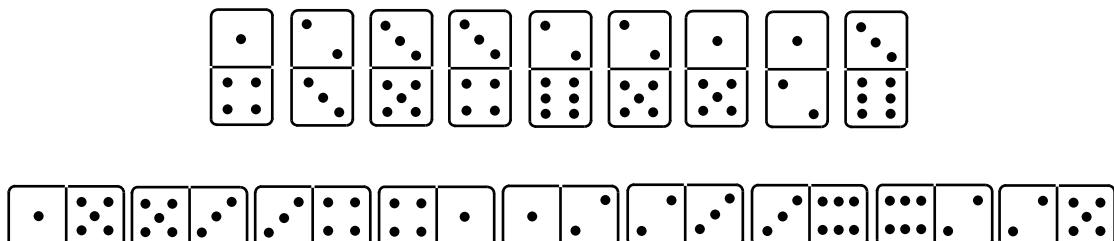
Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

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, $\frac{3}{4}$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

- (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, $(A \vee \bar{B}) \wedge (\bar{A} \vee C \vee D) \wedge (\bar{A} \vee B \vee \bar{C} \vee \bar{D})$ satisfies the constraints, while $(A \vee \bar{B}) \wedge (\bar{A} \vee C \vee D) \wedge (A \vee B \vee \bar{C} \vee \bar{D})$ does not, because positive literal A appears twice.
- (10 points) A domino is 2×1 rectangle 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 whether 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.

```
APPROXIMATION ALGORITHM:
A1 ← Greedy()
A2 ← SingleElement()
return max(A1, A2)
```

```
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  $V$ 
```

```
SINGLE ELEMENT:
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  $S = \{a_i | i = 0 \dots n - 1\}$ ,
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.

APPROXIMATION ALGORITHM:

```

Sort S in an increasing order
s1 ← 0
s2 ← 0
for i ← 0 to n
    if s1 ≤ s2
        s1 ← s1 + ai
    else
        s2 ← s2 + ai
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 $i \leftarrow 0$ 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 whether 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,

$$\begin{aligned}(A \vee B \vee \bar{C}) \wedge (B \vee \bar{A}), \\ (A \vee \bar{B} \vee C) \wedge (B \vee \bar{A} \vee D) \wedge (A \vee D), \\ (\bar{A} \vee B) \wedge (B \vee \bar{A} \vee C) \wedge (C \vee D)\end{aligned}$$

satisfy the condition, while

$$\begin{aligned}(\bar{A} \vee B \vee \bar{C}) \wedge (B \vee \bar{A}), \\ (A \vee \bar{B} \vee C) \wedge (B \vee \bar{A} \vee \bar{D}) \wedge (A \vee D), \\ (\bar{A} \vee B) \wedge (B \vee \bar{A} \vee C) \wedge (\bar{C} \vee \bar{D})\end{aligned}$$

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.

Write your answers in the separate answer booklet.

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

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

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

- (a) What is $\sum_{i=1}^n \frac{i}{n}$?
- (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 $T(n) = 5T\left(\lceil \frac{n-17}{25} \rceil - \lg \lg n\right) + \pi n + 2\sqrt{\log^* n} - 6$?
- (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:

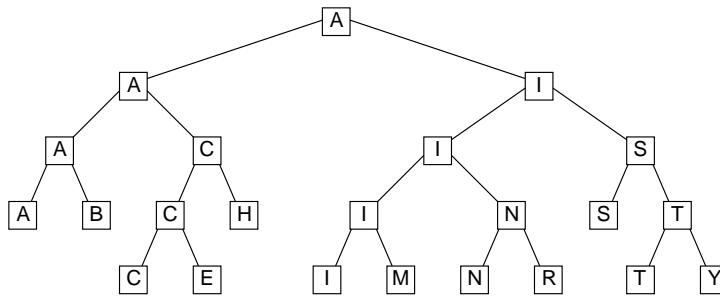
```
RANDOMMIN( $A[1..n]$ ):
     $min \leftarrow \infty$ 
    for  $i \leftarrow 1$  to  $n$  in random order
        if  $A[i] < min$ 
             $min \leftarrow A[i]$  (*)
```

return min

- (a) [1 point] In the worst case, how many times does RANDOMMIN execute line (*)?
- (b) [3 points] What is the probability that line (*) is executed during the n th iteration of the for loop?
- (c) [6 points] What is the exact expected number of executions of line (*)? (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-orbits² 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 $\text{FIND}(x, T)$ 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?]

¹Greek for “fear” and “panic”, respectively. Doesn’t that make you feel better?

²1000 Phobos orbits ≈ 1 Earth year

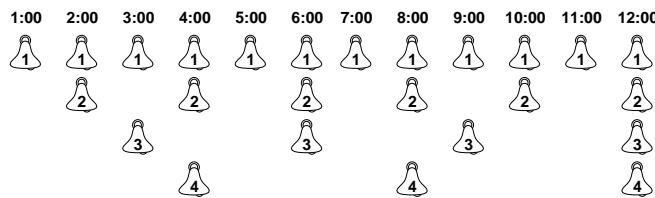
5. [Graduate students must answer this question.]

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 an 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 PREANTHEROHODPOBIOPLOMATGICALLY. 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 i th 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 $u \rightarrow v$ has a weight $w(u \rightarrow v)$. 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.

INITSSSP(s):

```

dist( $s$ )  $\leftarrow 0$ 
pred( $s$ )  $\leftarrow \text{NULL}$ 
for all vertices  $v \neq s$ 
    dist( $v$ )  $\leftarrow \infty$ 
    pred( $v$ )  $\leftarrow \text{NULL}$ 

```

RELAX($u \rightarrow v$):

```

if dist( $v$ )  $>$  dist( $u$ ) +  $w(u \rightarrow v)$ 
    dist( $v$ )  $\leftarrow$  dist( $u$ ) +  $w(u \rightarrow v)$ 
    pred( $v$ )  $\leftarrow u$ 

```

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 .

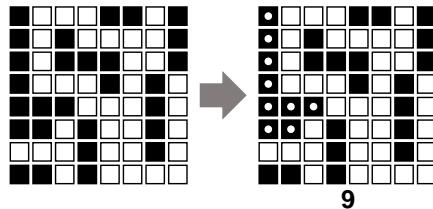
3. Suppose we want to maintain a dynamic set of values, subject to the following operations:

- **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 $O(\log n)$.

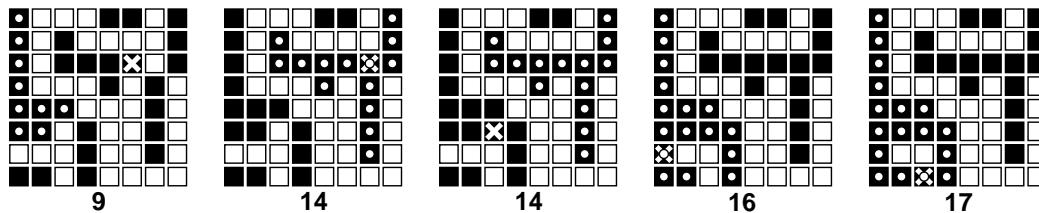
4. (a) [4 pts] Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an $n \times n$ bitmap $B[1..n, 1..n]$.

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



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

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



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

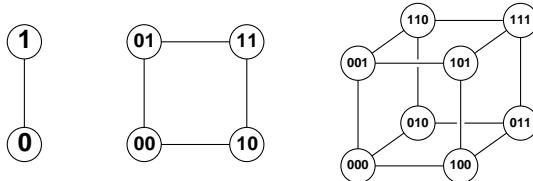
5. [Graduate students must answer this question.]

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.

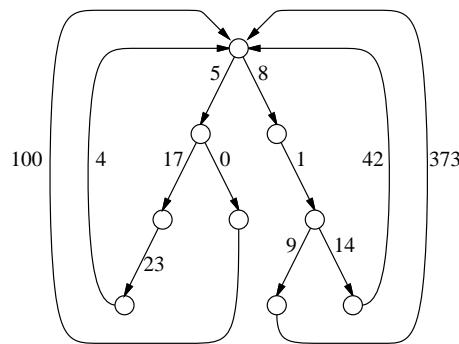
Write your answers in the separate answer booklet.
This is a 180-minute exam. The clock started when you got the questions.

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 **bubbasesabananana** ('Bubba sees a banana.') can be decomposed in several ways; for example:

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

Describe an efficient algorithm to find the *minimum* number of palindromes that make up a given input string. For example, given the input string **bubbasesabananana**, 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:

- $\Pr[h(x) = h(y)] = \frac{1}{m}$ for any pair of items $x \neq y$.
- $\Pr[h(x) = i] = \frac{1}{m}$ 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 friends¹, 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

¹Except you, of course. Unfortunately, you can't go to the party because you're taking a final exam. Sorry!

²Your solution to problem 4 in homework 1 does *not* solve this problem in polynomial time.

CS 373U: Combinatorial Algorithms, Spring 2004

Homework 0

Due January 28, 2004 at noon

Name:	
Net ID:	Alias:



I understand the Homework Instructions and FAQ.

-
- Neatly print your full name, your NetID, and an alias of your choice in the boxes above. Grades will be listed on the course web site by alias; for privacy reasons, your alias should not resemble your name or NetID. By providing an alias, you agree to let us list your grades; if you do not provide an alias, your grades will not be listed. *Never give us your Social Security number!*
 - Before you do anything else, read the Homework Instructions and FAQ on the course web page, and then check the box above. This web page gives instructions on how to write and submit homeworks—staple your solutions together in order, start each numbered problem on a new sheet of paper, write your name and NetID one every page, don't turn in source code, analyze and prove everything, use good English and good logic, and so on. See especially the policies regarding the magic phrases “I don’t know” and “and so on”. If you have *any* questions, post them to the course newsgroup or ask in lecture.
 - This homework tests your familiarity with prerequisite material—basic data structures, big-Oh notation, recurrences, discrete probability, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Chapters 1–10 of CLRS should be sufficient review, but you may also want consult your discrete mathematics and data structures textbooks.
 - Every homework will have five required problems and one extra-credit problem. Each numbered problem is worth 10 points.
-

#	1	2	3	4	5	6*	Total
Score							
Grader							

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

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

(a)
$$\boxed{2^{\sqrt{\lg n}} \quad 2^{\lg \sqrt{n}} \quad \sqrt{2^{\lg n}} \quad \sqrt{2^{\lg n}} \quad \lg 2^{\sqrt{n}} \quad \lg \sqrt{2^n} \quad \lg \sqrt{2^n} \quad \sqrt{\lg 2^n}}$$

*(b)
$$\boxed{\lg(\sqrt{n}!) \quad \lg(\sqrt{n}!) \quad \sqrt{\lg(n!)} \quad (\lg \sqrt{n})! \quad (\sqrt{\lg n})! \quad \sqrt{(\lg n)!}}$$

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

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

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

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

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

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

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

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

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

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

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

★(j) $J(n) = \sqrt{n} \cdot J(2\sqrt{n}) + n$

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

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

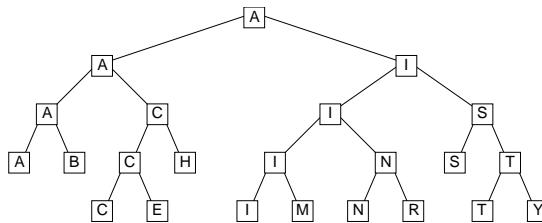
- Each red nertex has two children, exactly one of which is green.
- Each green nertex has exactly one child, which is not green.
- Blue nertices have no children.

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

- (a) Prove that $B = R + 1$.
- (b) Prove that either $G = R$ or $G = B$.
- (c) Prove that $G = B$ if and only if Rudy is green.

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

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



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

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

¹Greek for “fear” and “panic”, respectively. Doesn’t that make you feel better?

5. Penn and Teller agree to play the following game. Penn shuffles a standard deck² 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 ($3\clubsuit$), at which point the remaining undrawn cards instantly burst into flames.

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

- (a) What is the expected number of cards that Teller draws?
- (b) What is the expected *maximum* value among the cards Teller gives to Penn?
- (c) What is the expected *minimum* value among the cards Teller gives to Penn?
- (d) What is the expected number of cards that Teller gives to Penn?

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

*6. [Extra credit]⁴

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

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

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

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

- (a) Prove that in any lazy binary number, between any two 2s there is at least one 0, and between two 0s there is at least one 2.
- (b) Prove that for any natural number N , the sum of the digits of the lazy binary representation of N is exactly $\lfloor \lg(N + 1) \rfloor$.

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

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

⁴The “I don’t know” rule does not apply to extra credit problems. There is no such thing as “partial extra credit”.

CS 373U: Combinatorial Algorithms, Spring 2004

Homework 1

Due Monday, February 9, 2004 at noon

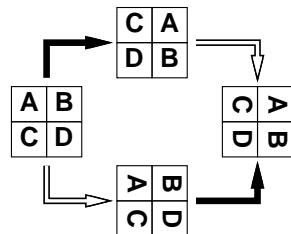
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

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

#	1	2	3	4	5	6*	Total
Score							
Grader							

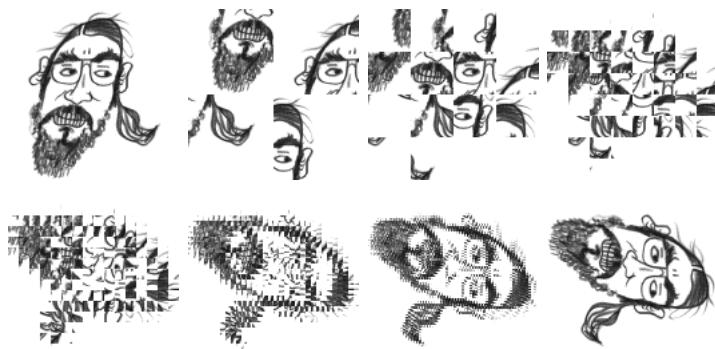
1. Some graphics hardware includes support for an operation called *blit*, or **block transfer**, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

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



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

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



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

- Prove that both versions of the algorithm are correct. [Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]
- Exactly how many blits does the algorithm perform?
- What is the algorithm's running time if each $k \times k$ blit takes $O(k^2)$ time?
- What if each $k \times k$ blit takes only $O(k)$ time?

2. The traditional Devonian/Cornish drinking song “The Barley Mow” has the following pseudolyrics¹, where $\text{container}[i]$ is the name of a container that holds 2^i ounces of beer.²

BARLEYMOW(n):

“Here’s a health to the barley-mow, my brave boys,”

“Here’s a health to the barley-mow!”

“We’ll drink it out of the jolly brown bowl,”

“Here’s a health to the barley-mow!”

“Here’s a health to the barley-mow, my brave boys,”

“Here’s a health to the barley-mow!”

for $i \leftarrow 1$ 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!*”

 “*Here’s a health to the barley-mow, my brave boys,*”

 “*Here’s a health to the barley-mow!*”

- (a) Suppose each container name $\text{container}[i]$ is a single word, and you can sing four words a second. How long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound.)
- (b) If you want to sing this song for $n > 20$, you’ll have to make up your own container names, and to avoid repetition, these names will get progressively longer as n increases³. Suppose $\text{container}[n]$ 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 $\text{container}[i]$. Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang BARLEYMOW(n)? (Give an *exact* answer, not just an asymptotic bound.)

¹Pseudolyrics are to lyrics as pseudocode is to code.

²One 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 anklers, and the last three units are just silly.

³“We’ll drink it out of the hemisemidemiyottapint, boys!”

3. In each of the problems below, you are given a ‘magic box’ that can solve one problem quickly, and you are asked to construct an algorithm that uses the magic box to solve a different problem.
- (a) **3-Coloring:** A graph is *3-colorable* if it is possible to color each vertex red, green, or blue, so that for every edge, its two vertices have two different colors. Suppose you have a magic box that can tell you whether a given graph is 3-colorable in constant time. Describe an algorithm that constructs a 3-coloring of a given graph (if one exists) as quickly as possible.
 - (b) **3SUM:** The 3SUM problem asks, given a set of integers, whether any three elements sum to zero. Suppose you have a magic box that can solve the 3SUM problem in constant time. Describe an algorithm that actually finds, given a set of integers, three elements that sum to zero (if they exist) as quickly as possible.
 - (c) **Traveling Salesman:** A *Hamiltonian cycle* in a graph is a cycle that visits every vertex exactly once. Given a complete graph where every edge has a weight, the *traveling salesman* cycle is the Hamiltonian cycle with minimum total weight; that is, the sum of the weight of the edges is smaller than for any other Hamiltonian cycle. Suppose you have a magic box that can tell you the weight of the traveling salesman cycle of a weighted graph in constant time. Describe an algorithm that actually constructs the traveling salesman cycle of a given weighted graph as quickly as possible.
4. (a) Describe and analyze an algorithm to sort an array $A[1..n]$ by calling a subroutine $\text{SQRT SORT}(k)$, which sorts the subarray $A[k+1..k + \lceil \sqrt{n} \rceil]$ in place, given an arbitrary integer k between 0 and $n - \lceil \sqrt{n} \rceil$ as input. Your algorithm is *only* allowed to inspect or modify the input array by calling SQRT SORT ; in particular, your algorithm must not directly compare, move, or copy array elements. How many times does your algorithm call SQRT SORT in the worst case?
- (b) Prove that your algorithm from part (a) is optimal up to constant factors. In other words, if $f(n)$ is the number of times your algorithm calls SQRT SORT , prove that no algorithm can sort using $o(f(n))$ calls to SQRT SORT .
- (c) Now suppose SQRT SORT is implemented recursively, by calling your sorting algorithm from part (a). For example, at the second level of recursion, the algorithm is sorting arrays roughly of size $n^{1/4}$. What is the worst-case running time of the resulting sorting algorithm? (To simplify the analysis, assume that the array size n has the form 2^{2^k} , so that repeated square roots are always integers.)

5. In a previous incarnation, you worked as a cashier in the lost Antarctic colony of Nadira, spending the better part of your day giving change to your customers. Because paper is a very rare and valuable resource on Antarctica, cashiers were required by law to use the fewest bills possible whenever they gave change. Thanks to the numerological predilections of one of its founders, the currency of Nadira, called Dream Dollars, was available in the following denominations: \$1, \$4, \$7, \$13, \$28, \$52, \$91, \$365.⁴
- (a) The greedy change algorithm repeatedly takes the largest bill that does not exceed the target amount. For example, to make \$122 using the greedy algorithm, we first take a \$91 bill, then a \$28 bill, and finally three \$1 bills. Give an example where this greedy algorithm uses more Dream Dollar bills than the minimum possible.
 - (b) Describe and analyze a recursive algorithm that computes, given an integer k , the minimum number of bills needed to make k Dream Dollars. (Don't worry about making your algorithm fast; just make sure it's correct.)
 - (c) Describe a dynamic programming algorithm that computes, given an integer k , the minimum number of bills needed to make k Dream Dollars. (This one needs to be fast.)

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

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

- (a) Suppose the graph is just a cycle of length n . Give a simple and complete characterization of the solvable light configurations in this case. (What we're really looking for here is a *fast* algorithm to decide whether a given configuration is solvable or not.) [Hint: For which cycle lengths is **every** configuration solvable?]
- *(b) Characterize the set of solvable light configurations when the graph is an arbitrary tree.
- *(c) A *grid graph* is a graph whose vertices are a regular $h \times w$ grid of integer points, with edges between immediate vertical or horizontal neighbors. Characterize the set of solvable light configurations for an arbitrary grid graph. (For example, the original Lights Out puzzle can be modeled as a 5×5 grid graph.)

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

CS 373U: Combinatorial Algorithms, Spring 2004

Homework 2

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

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

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

#	1	2	3	4	5	6*	Total
Score							
Grader							

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

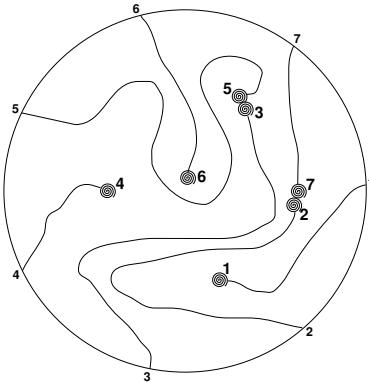
```

LENGTHOFLIS( $A[1..n]$ ):
   $A[n+1] = \infty$ 
  for  $i \leftarrow 1$  to  $n+1$ 
     $L[i] \leftarrow 1$ 
    for  $j \leftarrow 1$  to  $i-1$ 
      if  $A[j] < A[i]$  and  $1 + L[j] < L[i]$ 
         $L[i] \leftarrow 1 + L[j]$ 
  return  $L[n+1] - 1$ 

```

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

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



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

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

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

3. Describe and analyze a *polynomial-time* algorithm to determine whether a boolean formula in conjunctive normal form, with exactly *two* literals in each clause, is satisfiable.
4. This problem asks you to prove that four different variants of the minimum spanning tree problem are NP-hard. In each case, the input is a connected undirected graph G with weighted edges. Each problem considers a certain subset of the possible spanning trees of G , and asks you to compute the spanning tree with minimum total weight in that subset.
- Prove that finding the minimum-weight *depth first search* tree is NP-hard. (To remind yourself what depth first search is, and why it computes a spanning tree, see Jeff's introductory notes on graphs or Chapter 22 in CLRS.)
 - Suppose a subset S of the nodes in the input graph are marked. Prove that it is NP-hard to compute the minimum spanning tree whose leaves are all in S . [Hint: First consider the case $|S| = 2$.]
 - Prove that for any integer $\ell \geq 2$, it is NP-hard to compute the minimum spanning tree with exactly ℓ leaves. [Hint: First consider the case $\ell = 2$.]
 - Prove that for any integer $d \geq 2$, it is NP-hard to compute the minimum spanning tree with maximum degree d . [Hint: First consider the case $d = 2$. By now this should start to look familiar.]

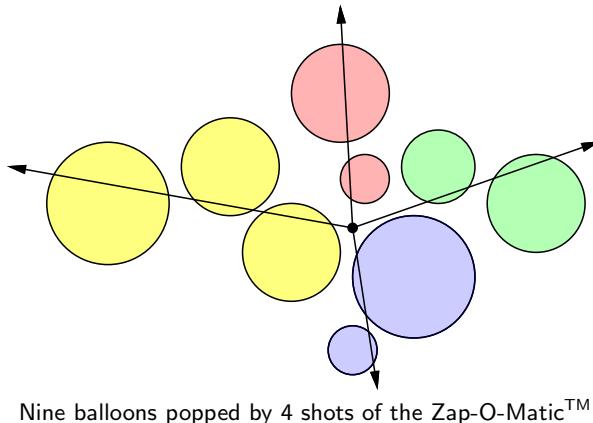
You're welcome to use reductions among these four problems. For example, even if you can't solve part (d), if you can prove that (d) implies (b), you will get full credit for (b). Just don't argue circularly.

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

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

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

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



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

- (a) Describe and analyze a greedy algorithm whose output is within 1 of optimal. That is, if m is the minimum number of rays required to hit every circle in the input, then your greedy algorithm must output either m or $m + 1$. (Of course, you must prove this fact.)
- (b) Describe an algorithm that solves the minimum zap problem in $O(n^2)$ time.
- *(c) Describe an algorithm that solves the minimum zap problem in $O(n \log n)$ time.

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

CS 373U: Combinatorial Algorithms, Spring 2004

Homework 3

Due Friday, March 12, 2004 at noon

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

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

#	1	2	3	4	5	6*	Total
Score							
Grader							

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

- (a) Describe and analyze a deterministic algorithm that computes the Pareto-optimal points in S in $O(n \log n)$ time.
- (b) Suppose each point in S is chosen independently and uniformly at random from the unit square $[0, 1] \times [0, 1]$. What is the *exact* expected number of Pareto-optimal points in S ?

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

- (a) Consider the following implementation of RANDOMPERMUTATION .

```
RANDOMPERMUTATION( $n$ ):
    for  $i = 1$  to  $n$ 
         $\pi[i] \leftarrow \text{NULL}$ 
    for  $i = 1$  to  $n$ 
         $j \leftarrow \text{RANDOM}(n)$ 
        while ( $\pi[j] \neq \text{NULL}$ )
             $j \leftarrow \text{RANDOM}(n)$ 
         $\pi[j] \leftarrow i$ 
    return  $\pi$ 
```

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

- (b) Consider the following partial implementation of RANDOMPERMUTATION .

```
RANDOMPERMUTATION( $n$ ):
    for  $i = 1$  to  $n$ 
         $A[i] \leftarrow \text{RANDOM}(n)$ 
     $\pi \leftarrow \text{SOMEFUNCTION}(A)$ 
    return  $\pi$ 
```

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

- (c) Consider a correct implementation of $\text{RANDOMPERMUTATION}(n)$ with the following property: whenever it calls $\text{RANDOM}(k)$, the argument k is at most m . Prove that this algorithm **always** calls RANDOM at least $\Omega(\frac{n \log n}{\log m})$ times.
- (d) Describe and analyze an implementation of RANDOMPERMUTATION that runs in expected worst-case time $O(n)$.

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

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

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

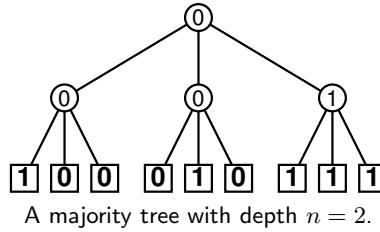
```

MELD( $Q_1, Q_2$ ):
    if  $Q_1$  is empty return  $Q_2$ 
    if  $Q_2$  is empty return  $Q_1$ 
    if  $key(Q_1) > key(Q_2)$ 
        swap  $Q_1 \leftrightarrow Q_2$ 
    with probability  $1/2$ 
         $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
    else
         $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
    return  $Q_1$ 

```

- (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 $\text{MELD}(Q_1, Q_2)$ is $O(\log n)$, where $n = |Q_1| + |Q_2|$. [Hint: How long is a random root-to-leaf path in an n -node binary tree if each left/right choice is made with equal probability?]
- (b) [Extra credit] Prove that $\text{MELD}(Q_1, Q_2)$ runs in $O(\log n)$ time with high probability.
- (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)

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



- (a) Prove that *any* deterministic algorithm that computes the value of the root of a majority tree *must* examine every leaf. [Hint: Consider the special case $n = 1$. Recurse.]
- (b) Describe and analyze a randomized algorithm that computes the value of the root in worst-case expected time $O(c^n)$ for some constant $c < 3$. [Hint: Consider the special case $n = 1$. Recurse.]
5. Suppose n lights labeled $0, \dots, n - 1$ are placed clockwise around a circle. Initially, each light is set to the off position. Consider the following random process.

```

LIGHTTHECIRCLE( $n$ ):
 $k \leftarrow 0$ 
turn on light 0
while at least one light is off
  with probability  $1/2$ 
     $k \leftarrow (k + 1) \bmod n$ 
  else
     $k \leftarrow (k - 1) \bmod n$ 
  if light  $k$  is off, turn it on
  
```

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

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

CS 373U: Combinatorial Algorithms, Spring 2004

Homework 4

Due Friday, April 2, 2004 at noon

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

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

#	1	2	3	4	5	6*	Total
Score							
Grader							

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

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

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

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

- (a) Describe how to implement a queue using two stacks and $O(1)$ additional memory, so that the amortized time for any enqueue or dequeue operation is $O(1)$. The *only* access you have to the stacks is through the standard subroutines PUSH and POP.
- (b) A *quack* is a data structure combining properties of both stacks and queues. It can be viewed as a list of elements written left to right such that three operations are possible:
 - **Push:** add a new item to the left end of the list;
 - **Pop:** remove the item on the left end of the list;
 - **Pull:** remove the item on the right end of the list.

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

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

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

- (a) [1 pt] Overall, how much space does this data structure use in the worst case?
- (b) [1 pt] How much space does this structure use if the top-level search tree T is balanced?
- (c) [2 pt] Suppose T is a splay tree. Prove that the *amortized* cost of a splay (and therefore of a search, insertion, or deletion) is $\Omega(n)$. [*Hint: This is easy!*]
- (d) [3 pts] Now suppose T is a scapegoat tree, and that rebuilding the subtree rooted at v requires $\Theta(|v| \log |v|)$ time (because we also have to rebuild all the secondary structures). What is the *amortized* cost of inserting a new element into T ?
- (e) [3 pts] Finally, suppose T is a treap. What's the worst-case *expected* time for inserting a new element into T ?

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

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

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

LAZYINCREMENT($B[0..n]$):

```

 $B[0] \leftarrow B[0] + 1$ 
for  $i \leftarrow 1$  to  $n - 1$ 
  if  $B[i] = 2$ 
     $B[i] \leftarrow 0$ 
     $B[i + 1] \leftarrow B[i + 1] + 1$ 
return
  
```

LAZYDECREMENT($B[0..n]$):

```

 $B[0] \leftarrow B[0] - 1$ 
for  $i \leftarrow 1$  to  $n - 1$ 
  if  $B[i] = -1$ 
     $B[i] \leftarrow 1$ 
     $B[i + 1] \leftarrow B[i + 1] - 1$ 
return
  
```

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

```

0  $\xrightarrow{++}$  1  $\xrightarrow{++}$  10  $\xrightarrow{++}$  11  $\xrightarrow{++}$  20  $\xrightarrow{++}$  101  $\xrightarrow{++}$  110  $\xrightarrow{++}$  111  $\xrightarrow{++}$  120  $\xrightarrow{++}$  201  $\xrightarrow{++}$  210
 $\xrightarrow{++}$  1011  $\xrightarrow{++}$  1020  $\xrightarrow{++}$  1101  $\xrightarrow{++}$  1110  $\xrightarrow{++}$  1111  $\xrightarrow{++}$  1120  $\xrightarrow{++}$  1201  $\xrightarrow{++}$  1210  $\xrightarrow{++}$  2011  $\xrightarrow{++}$  2020
 $\xrightarrow{--}$  2011  $\xrightarrow{--}$  2010  $\xrightarrow{--}$  2001  $\xrightarrow{--}$  2000  $\xrightarrow{--}$  2011  $\xrightarrow{--}$  2110  $\xrightarrow{--}$  2101  $\xrightarrow{--}$  1100  $\xrightarrow{--}$  1111  $\xrightarrow{--}$  1010
 $\xrightarrow{--}$  1001  $\xrightarrow{--}$  1000  $\xrightarrow{--}$  1011  $\xrightarrow{--}$  1110  $\xrightarrow{--}$  1101  $\xrightarrow{--}$  1010  $\xrightarrow{--}$  100  $\xrightarrow{--}$  111  $\xrightarrow{--}$  10  $\xrightarrow{--}$  1  $\xrightarrow{--}$  0
  
```

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

¹That was for a slightly different problem anyway.

6. [Extra credit] My wife is teaching a class² where students work on homeworks in groups of exactly three people, subject to the following rule: *No two students may work together on more than one homework.* At the beginning of the semester, it was easy to find homework groups, but as the course progresses, it is becoming harder and harder to find a legal grouping. Finally, in despair, she decides to ask a computer scientist to write a program to find the groups for her.

- (a) We can formalize this homework-group-assignment problem as follows. The input is a graph, where the vertices are the n students, and two students are joined by an edge if they have not yet worked together. Every node in this graph has the same degree; specifically, if there have been k homeworks so far, each student is connected to exactly $n - 1 - 2k$ other students. The goal is to find $n/3$ disjoint triangles in the graph, or conclude that no such triangles exist. Prove (or disprove!) that this problem is NP-hard.
- (b) Suppose my wife had planned ahead and assigned groups for every homework at the beginning of the semester. How many homeworks can she assign, as a function of n , without violating the no-one-works-together-twice rule? Prove the best upper and lower bounds you can. To prove the upper bound, describe an algorithm that actually assigns the groups for each homework.

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

CS 373U: Combinatorial Algorithms, Spring 2004

Homework 5

Due Wednesday, April 28, 2004 at noon

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

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

#	1	2	3	4	5	Total
Score						
Grader						

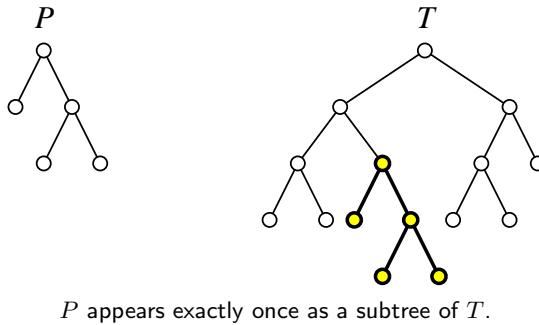
1. (a) Prove that every graph with the same number of vertices and edges has a cycle.
(b) Prove that every graph with exactly two fewer edges than vertices is disconnected.

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

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

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

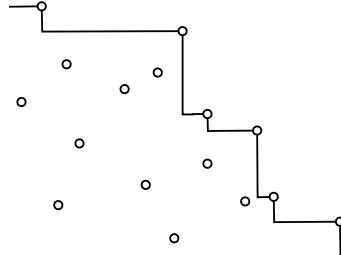
3. Describe and analyze an algorithm that decides, given two binary trees P and T , whether T is a subtree of P . There is no actual *data* stored in the nodes—these are not binary *search* trees or binary *heaps*. You are only trying to match the *shape* of the trees.



4. Describe and analyze an algorithm that computes the *second* smallest spanning tree of a given connected, undirected, edge-weighted graph.
5. Show that if the input graph is allowed to have negative edges (but no negative cycles), Dijkstra's algorithm¹ runs in exponential time in the worst case. Specifically, describe how to construct, for every integer n , a weighted directed graph G_n without negative cycles that forces Dijkstra's algorithm to perform $\Omega(2^n)$ relaxation steps. Give your description in the form of an algorithm! [Hint: Towers of Hanoi.]

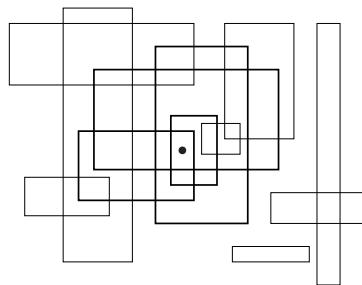
¹This refers to the version of Dijkstra's algorithm described in Jeff's lecture notes. The version in CLRS is always fast, but sometimes gives incorrect results for graphs with negative edges.

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



The staircase of a set of points

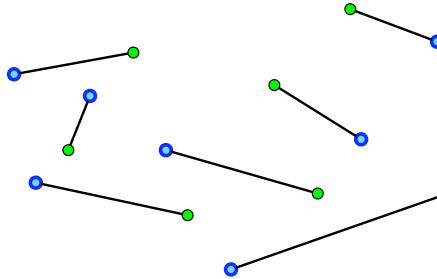
- (a) Describe an algorithm to compute the staircase of P in $O(nh)$ time, where h is the number of Pareto-optimal points.
 - (b) Describe a divide-and-conquer algorithm to compute the staircase of P in $O(n \log n)$ time. [Hint: I know of at least two different ways to do this.]
 - *(c) Describe an algorithm to compute the staircase of P in $O(n \log h)$ time, where h is the number of Pareto-optimal points. [Hint: I know of at least two different ways to do this.]
 - (d) Finally, suppose the points in P are already given in sorted order from left to right. Describe an algorithm to compute the staircase of P in $O(n)$ time. [Hint: I know of at least two different ways to do this.]
2. Let R be a set of n rectangles in the plane.
- (a) Describe and analyze a plane sweep algorithm to decide, in $O(n \log n)$ time, whether any two rectangles in R intersect.
 - *(b) The *depth* of a point is the number of rectangles in R that contain that point. The *maximum depth* of R is the maximum, over all points p in the plane, of the depth of p . Describe a plane sweep algorithm to compute the maximum depth of R in $O(n \log n)$ time.



A point with depth 4 in a set of rectangles.

- (c) Describe and analyze a polynomial-time reduction from the maximum depth problem in part (b) to MAXCLIQUE: Given a graph G , how large is the largest clique in G ?
- (d) MAXCLIQUE is NP-hard. So does your reduction imply that P=NP? Why or why not?

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



A non-crossing matching between 7 ghosts and 7 Ghostbusters

- (a) Prove that the Ghostbusters can succeed. More formally, prove that there is a collection of n non-intersecting line segments, each joining one point in G to one point in B . [Hint: Think about the set of joining segments with minimum total length.]
- (b) Describe and analyze an algorithm to find a line ℓ that passes through one ghost and one Ghostbuster, so that same number of ghosts as Ghostbusters are above ℓ .
- *(c) Describe and analyze an algorithm to find a line ℓ such that exactly half the ghosts and exactly half the Ghostbusters are above ℓ . (Assume n is even.)
- (d) Using your algorithm for part (b) or part (c) as a subroutine, describe and analyze an algorithm to find the line segments described in part (a). (Assume n is a power of two if necessary.)

Spengler: *Don't cross the streams.*

Venkman: *Why?*

Spengler: *It would be bad.*

Venkman: *I'm fuzzy on the whole good/bad thing. What do you mean "bad"?*

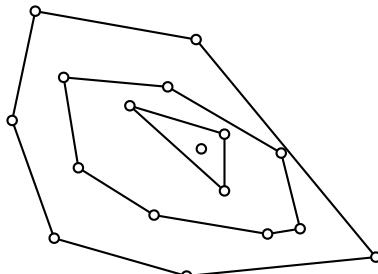
Spengler: *Try to imagine all life as you know it stopping instantaneously and every molecule in your body exploding at the speed of light.*

Stantz: *Total protonic reversal!*

Venkman: *That's bad. Okay. Alright, important safety tip, thanks Egon.*

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

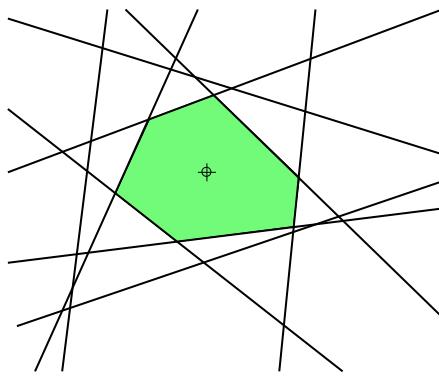
4. The *convex layers* of a point set P consist of a series of nested convex polygons. The convex layers of the empty set are empty. Otherwise, the first layer is just the convex hull of P , and the remaining layers are the convex layers of the points that are not on the convex hull of P .



The convex layers of a set of points.

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

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



The cell containing the origin in an arrangement of lines.

Write your answers in the separate answer booklet.

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

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

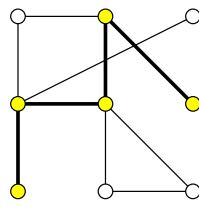
For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point. Each X earns you $\frac{1}{4}$ point. **Each incorrect answer costs you $\frac{1}{2}$ point.** Your total score will be rounded **down** to an integer.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

- (a) What is $\sum_{i=1}^n \lg i$?
- (b) What is $\sum_{i=1}^{\lg n} i 2^i$?
- (c) How many decimal digits are required to write the n th Fibonacci number?
- (d) What is the solution of the recurrence $T(n) = 4T(n/8) + n \log n$?
- (e) What is the solution of the recurrence $T(n) = T(n - 3) + \frac{5}{n}$?
- (f) What is the solution of the recurrence $T(n) = 5T\left(\lceil \frac{n+13}{3} \rceil + \lfloor \sqrt{n} \rfloor\right) + (10n - 7)^2 - \frac{\lg^3 n}{\lg \lg n}$?
- (g) How long does it take to construct a Huffman code, given an array of n character frequencies as input?
- (h) How long does it take to sort an array of size n using quicksort?
- (i) Given an unsorted array $A[1..n]$, how long does it take to construct a binary search tree for the elements of A ?
- (j) A train leaves Chicago at 8:00pm and travels south at 75 miles per hour. Another train leaves New Orleans at 1:00pm and travels north at 60 miles per hour. The conductors of both trains are playing a game of chess over the phone. After each player moves, the other player must move before his train has traveled five miles. How many moves do the two players make before their trains pass each other (somewhere near Memphis)?

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

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

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



A tonian path in a 9-vertex graph.

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

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

-1	7	-8	10	-5
-4	-9	8	-6	0
5	-2	-6	-6	7
-7	4	7 \Rightarrow -3	-3	-3
7	1	-6	4	-9

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

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

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

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

Write your answers in the separate answer booklet.

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

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

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

- What is $\sum_{i=1}^n \frac{n}{i}$?
- What is $\sum_{i=1}^{\lg n} 4^i$?
- How many bits are required to write $n!$ in binary?
- What is the solution of the recurrence $T(n) = 4T(n/2) + n \log n$?
- What is the solution of the recurrence $T(n) = T(n - 3) + \frac{5}{n}$?
- What is the solution of the recurrence $T(n) = 9T\left(\lceil \frac{n+13}{3} \rceil\right) + 10n - 7\sqrt{n} - \frac{\lg^3 n}{\lg \lg n}$?
- How long does it search for a value in an n -node binary search tree?
- Given a sorted array $A[1..n]$, how long does it take to construct a binary search tree for the elements of A ?
- How long does it take to construct a Huffman code, given an array of n character frequencies as input?
- A train leaves Chicago at 8:00pm and travels south at 75 miles per hour. Another train leaves New Orleans at 1:00pm and travels north at 60 miles per hour. The conductors of both trains are playing a game of chess over the phone. After each player moves, the other player must move before his train has traveled five miles. How many moves do the two players make before their trains pass each other (somewhere near Memphis)?

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

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

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

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

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

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

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

- SUBSETSUM: Given a set X of integers and an integer k , does X have a subset whose elements sum up to k ?
- PARTITION: Given a set X of integers and an integer k , can X be partitioned into two subsets whose sums are equal?
 - (a) Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION.
 - (b) Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM.

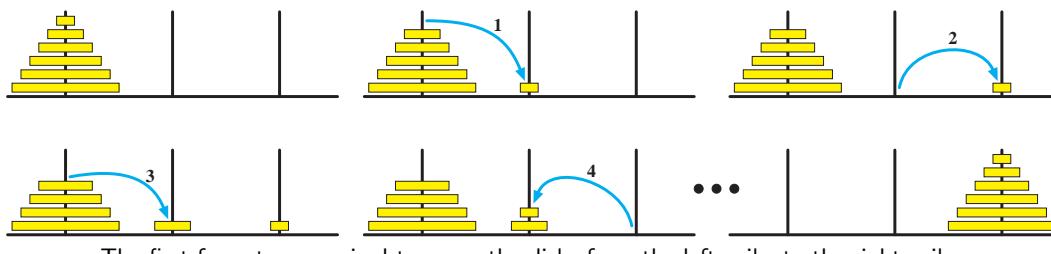
5. Describe and analyze efficient algorithms to solve the following problems:

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

Write your answers in the separate answer booklet.

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

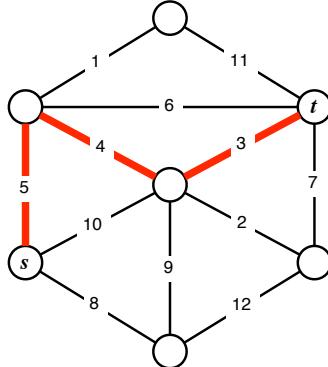
Hmmm.... You've probably known how to solve this problem since CS 125, so make it more interesting, let's add another constraint: The three spikes are arranged in a row, and we are also forbidden to move a disk directly from the left spike to the right spike or vice versa. In other words, we *must* move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



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

- (a) [4 pts] Describe an algorithm that moves the stack of n disks from the left needle to the right needle in as few steps as possible.
- (b) [6 pts] Exactly how many steps does your algorithm take to move all n disks? A correct Θ -bound is worth 3 points. [Hint: Set up and solve a recurrence.]
-
2. Consider a random walk on a path with vertices numbered $1, 2, \dots, n$ from left to right. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex n . In Midterm 2, you were asked to prove that if we start at vertex 1, the probability that the walk ends by falling off the *left* end of the path is exactly $n/(n+1)$.
- (a) [6 pts] Prove that if we start at vertex 1, the expected number of steps before the random walk ends is exactly n . [Hint: Set up and solve a recurrence. Use the result from Midterm 2.]
- (b) [4 pts] Suppose we start at vertex $n/2$ instead. State a tight Θ -bound on the expected length of the random walk in this case. **No proof is required.** [Hint: Set up and solve a recurrence. Use part (a), even if you can't prove it.]
-
3. Prove that any connected acyclic graph with n vertices has exactly $n - 1$ edges. Do not use the word "tree" or any well-known properties of trees; your proof should follow entirely from the definitions.

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



The bottleneck distance between s and t is 5.

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

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

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

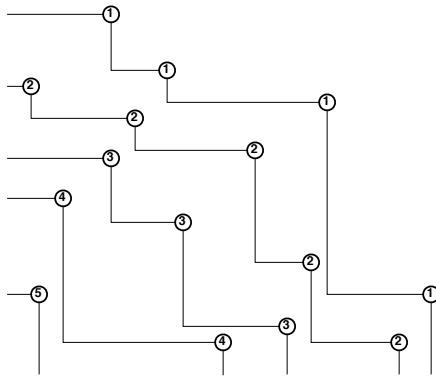
- (a) [2 pts] Show that 5COLOR ± 1 is in NP.
- (b) [1 pt] To prove that 5COLOR ± 1 is NP-hard (and therefore NP-complete), we must describe a polynomial time algorithm for *one* of the following problems. Which one?
 - Given an arbitrary graph G , compute a graph H such that 5COLOR(G) is true if and only if 5COLOR ± 1 (H) is true.
 - Given an arbitrary graph G , compute a graph H such that 5COLOR ± 1 (G) is true if and only if 5COLOR(H) is true.

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

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

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

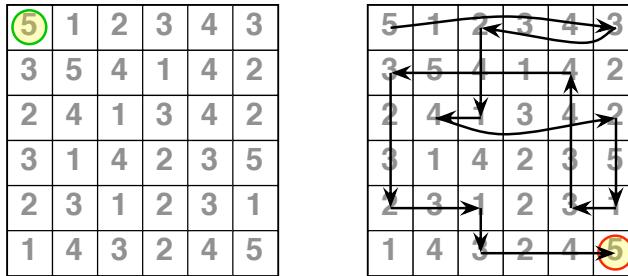
6. Let P be a set of points in the plane. Recall that a point $p \in P$ is *Pareto-optimal* if no other points in P are both above and to the right of p . Intuitively, the sequence of Pareto-optimal points forms a *staircase* with all the other points in P below and to the left. The *staircase layers*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



A set of points with 5 staircase layers

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

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



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

- (a) [4 pts] Describe and analyze an algorithm to determine, given an $n \times n$ array of labels and two squares s and t , whether there is a sequence of legal moves that takes the token from s to t .
- (b) [6 pts] Suppose you are only given the $n \times n$ array of labels. Describe how to preprocess these values, so that afterwards, given any two squares s and t , you can determine in $O(1)$ time whether there is a sequence of legal moves from s to t .

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

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

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

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

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

- (a) [6 pts] **Prove** that if we start at vertex 1, the expected number of steps before the random walk ends is exactly n . [Hint: Set up and solve a recurrence. Use the result from Midterm 2.]
- (b) [4 pts] Suppose we start at vertex $n/2$ instead. State and **prove** a tight Θ -bound on the expected length of the random walk in this case. [Hint: Set up and solve a recurrence. Use part (a), even if you can't prove it.]

3. **Prove** that any connected acyclic graph with $n \geq 2$ vertices has at least two vertices with degree 1. Do not use the words “tree” or “leaf”, or any well-known properties of trees; your proof should follow entirely from the definitions.
4. Consider the following sketch of a “reverse greedy” algorithm. The input is a connected undirected graph G with weighted edges, represented by an adjacency list.

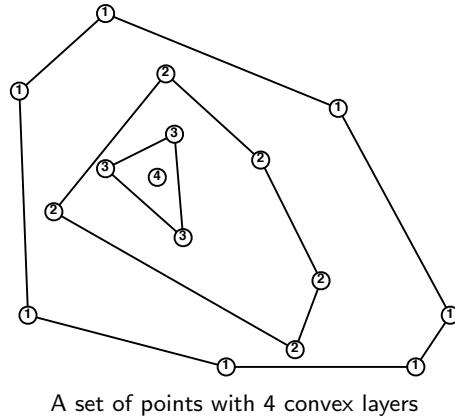
REVERSEGREEDYMST(G):
sort the edges E of G by weight
for $i \leftarrow 1$ to $ E $
$e \leftarrow$ i th heaviest edge in E
if $G \setminus e$ is connected
remove e from G

- (a) [4 pts] What is the worst-case running time of this algorithm? (Answering this question will require fleshing out a few details.)
- (b) [6 pts] **Prove** that the algorithm transforms G into its minimum spanning tree.

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

- SUBSETSUM: Given a set X of integers and an integer k , does X have a subset whose elements sum up to k ?
- PARTITION: Given a set X of integers, can X be partitioned into two subsets whose sums are equal?
 - [2 pts] Prove that PARTITION and SUBSETSUM are both in NP.
 - [1 pt] Suppose we knew that SUBSETSUM is NP-hard, and we wanted to prove that PARTITION is NP-hard. Which of the following arguments should we use?
 - Given a set X and an integer k , compute a set Y such that $\text{PARTITION}(Y)$ is true if and only if $\text{SUBSETSUM}(X, k)$ is true.
 - Given a set X , construct a set Y and an integer k such that $\text{PARTITION}(X)$ is true if and only if $\text{SUBSETSUM}(Y, k)$ is true.
 - [3 pts] Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM. (See part (b).)
 - [4 pts] Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION. (See part (b).)

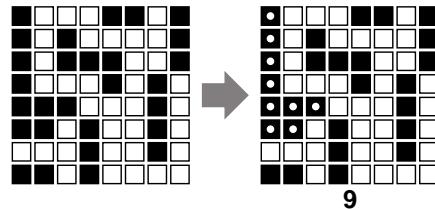
6. Let P be a set of points in the plane. The *convex layers* of P are defined recursively as follows. If P is empty, it has no convex layers. Otherwise, the first convex layer is the convex hull of P , and the remaining convex layers are the convex layers of P minus its convex hull.



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

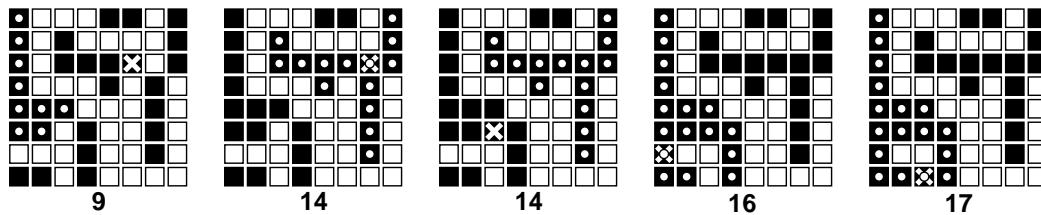
7. (a) [4 pts] Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an $n \times n$ bitmap $B[1..n, 1..n]$.

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



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

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



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

Write your answers in the separate answer booklet.

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

```
DOsomethingINTERESTING(stream  $S$ ):
repeat
     $x \leftarrow$  next item in  $S$ 
     $\langle\!\langle$  do something fast with  $x$   $\rangle\!\rangle$ 
until  $S$  ends
return  $\langle\!\langle$  something  $\rangle\!\rangle$ 
```

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

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

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

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

MAKESET(x):
$\text{parent}(x) \leftarrow x$
$\text{size}(x) \leftarrow 1$

FIND(x):
while $x \neq \text{parent}(x)$
$x \leftarrow \text{parent}(x)$
return x

UNION(x, y):
$\bar{x} \leftarrow \text{FIND}(x)$
$\bar{y} \leftarrow \text{FIND}(y)$
if $\text{size}(\bar{x}) < \text{size}(\bar{y})$
$\text{parent}(\bar{x}) \leftarrow \bar{y}$
$\text{size}(\bar{x}) \leftarrow \text{size}(\bar{x}) + \text{size}(\bar{y})$
else
$\text{parent}(\bar{y}) \leftarrow \bar{x}$
$\text{size}(\bar{y}) \leftarrow \text{size}(\bar{x}) + \text{size}(\bar{y})$

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

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

- (a) [7 pts] Describe and analyze an algorithm that solves SUBSETSUM in time $O(nk)$.
- (b) [3 pts] SUBSETSUM is NP-hard. Does part (a) imply that P=NP? Justify your answer.

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

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

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

- (a) [6 pts] Describe and analyze a data structure that supports these two operations, each in $O(\log n)$ *amortized* time, where n is the maximum number of elements in X .
- (b) [2 pts] What is the running time of your INSERT algorithm in the worst case?
- (c) [2 pts] What is the running time of your $\text{PRINTANDDELETEBETWEEN}$ algorithm in the worst case?

CS 473G: Combinatorial Algorithms, Fall 2005

Homework 0

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

Name:	
Net ID:	Alias:



I understand the Homework Instructions and FAQ.

-
- Neatly print your full name, your NetID, and an alias of your choice in the boxes above. Grades will be listed on the course web site by alias. Please write the same alias on every homework and exam! For privacy reasons, your alias should not resemble your name or NetID. By providing an alias, you agree to let us list your grades; if you do not provide an alias, your grades will not be listed. ***Never give us your Social Security number!***
 - Read the “Homework Instructions and FAQ” on the course web page, and then check the box above. This page describes what we expect in your homework solutions—start each numbered problem on a new sheet of paper, write your name and NetID on every page, don’t turn in source code, analyze and prove everything, use good English and good logic, and so on—as well as policies on grading standards, regrading, and plagiarism. **See especially the course policies regarding the magic phrases “I don’t know” and “and so on”.** If you have *any* questions, post them to the course newsgroup or ask during lecture.
 - Don’t forget to submit this cover sheet with the rest of your homework solutions.
 - This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, discrete probability, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Chapters 1–10 of CLRS should be sufficient review, but you may also want consult your discrete mathematics and data structures textbooks.
 - Every homework will have five required problems. Most homeworks will also include one extra-credit problem and several practice (no-credit) problems. Each numbered problem is worth 10 points.
-

#	1	2	3	4	5	6*	Total
Score							
Grader							

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

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

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

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

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

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

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

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

*(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 deck¹ 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 ($3\clubsuit$), at which point the remaining undrawn cards instantly burst into flames and the game is over.

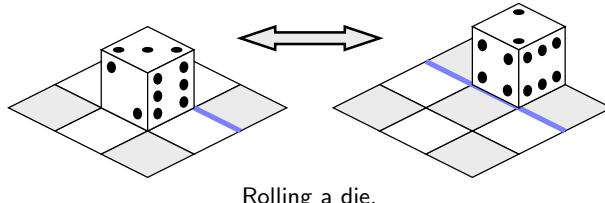
The first time Teller draws a card from the deck, he gives it to Penn. From then on, until the game ends, whenever Teller draws a card whose value is smaller than the previous card he gave to Penn, he gives the new card to Penn. To make the rules unambiguous, they agree on the numerical values $A = 1$,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

¹In 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.

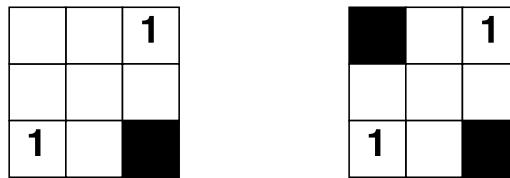
3. A *rolling die maze* is a puzzle involving a standard six-sided die² and a grid of squares. You should imagine the grid lying on top of a table; the die always rests on and exactly covers one square. In a single step, you can *roll* the die 90 degrees around one of its bottom edges, moving it to an 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 i th row and j th 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.]

²A 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 choose 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 $\lfloor \lg(N + 1) \rfloor$.

³The “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.

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

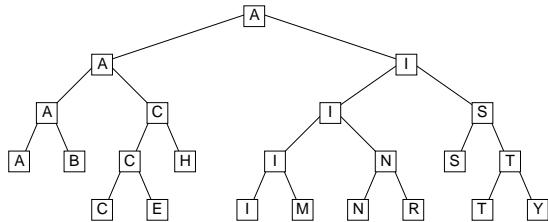
To simplify your answers, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$, and write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions $n^2, n, \binom{n}{2}, n^3$ could be sorted either as $n \ll n^2 \equiv \binom{n}{2} \ll n^3$ or as $n \ll \binom{n}{2} \equiv n^2 \ll n^3$.

- 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 .
 - F_n is even if and only if n is divisible by 3.
 - $\sum_{i=0}^n F_i = F_{n+2} - 1$
 - $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$
 - If n is an integer multiple of m , then F_n is an integer multiple of F_m .
- 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.
 - On average, how many cards does Penn give Teller?
 - On average, what is the smallest-numbered card that Penn gives Teller?
 - 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 $\text{FIND}(x, T)$ that returns TRUE if x is stored in the Phobian binary search tree T , and FALSE otherwise.
 - (b) A *Deimoid* binary search tree is almost exactly the same as its Phobian counterpart, except that the *largest* element is stored at the root, and both subtrees are Deimoid binary search trees. Describe and analyze an algorithm to transform an n -node Phobian binary search tree into a Deimoid binary search tree in $O(n)$ time, using as little additional space as possible.
5. Tatami are rectangular mats used to tile floors in traditional Japanese houses. Exact dimensions of tatami mats vary from one region of Japan to the next, but they are always twice as long in one dimension than in the other. (In Tokyo, the standard size is 180cm×90cm.)
- (a) How many different ways are there to tile a $2 \times n$ rectangular room with 1×2 tatami mats? Set up a recurrence and derive an *exact* closed-form solution. [Hint: The answer involves a *familiar recursive sequence*.]
 - (b) According to tradition, tatami mats are always arranged so that four corners never meet. How many different *traditional* ways are there to tile a $3 \times n$ rectangular room with 1×2 tatami mats? Set up a recurrence and derive an *exact* closed-form solution.
 - *(c) How many different *traditional* ways are there to tile an $n \times n$ square with 1×2 tatami mats? Prove your answer is correct.

⁴Greek 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.

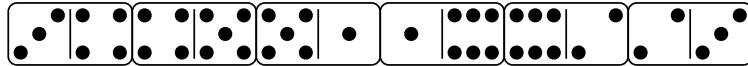
Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your answer to problem 1.

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 receive 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.

¹These 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

- Given an initial configuration consisting of an undirected graph $G = (V, E)$ and a function $p : V \rightarrow \mathbb{N}$ 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.
 - 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.
 - 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 S sum to t .
 - Describe a polynomial-time reduction from SUBSETSUM to PARTITION.
 - Describe a polynomial-time reduction from PARTITION to SUBSETSUM.
 - 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.
 - 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.]

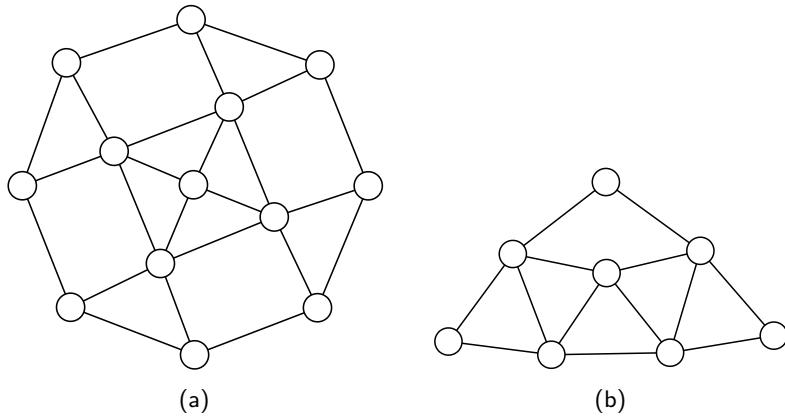


Figure 1. (a) Gadget for planar 3-colorability. (b) Gadget for degree-4 planar 3-colorability.

- (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 $\text{NP} \cap \text{co-NP}$. [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 and 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

¹For 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 Antarctic 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.)

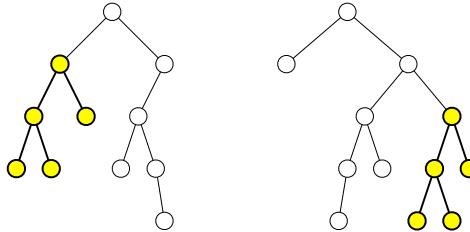
Also, Xryltconian society is strictly governed by a number of sacred cultural traditions.

- 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 weirdness value of each vode. Design and analyze an efficient algorithm to decide which vodes the scientists should invite to maximize the sum of the weirdness values of the vodes in the room. Be careful to respect all of the vodes' cultural taboos.

²For 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 $O(4^n \text{ poly}(n))$ 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 $O(3^n \text{ poly}(n))$ 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 $O((1+3^{1/3})^n \text{ poly}(n))$ 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 $O(\phi^n \text{poly}(n))$, 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 $\mathbf{A} = \{A_1, A_2, \dots, A_k\}$ 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 :

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

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 efficient 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 i th 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:

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

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

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

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 $k + \dots + 3 + 2 + 1$, which is the only partition that has roughness 0. For example, the smoothest partitions of 7 are $4 + 2 + 1$ and $3 + 2 + 1 + 1$:

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

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( $G$ ):  
     $C \leftarrow \emptyset$   
    while  $G$  has at least one edge  
         $v \leftarrow$  vertex in  $G$  with maximum degree  
         $G \leftarrow G \setminus v$   
         $C \leftarrow C \cup v$   
    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 $\text{GREEDYVERTEXCOVER}(G)$ 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 $A(G) > OPT(G) + k$, 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( $G$ ):
 $T \leftarrow$  any spanning tree of  $G$ 
Color the tree  $T$  with two colors
 $c \leftarrow 2$ 
for each edge  $(u, v) \in G \setminus T$ 
     $T \leftarrow T \cup \{(u, v)\}$ 
    if  $\text{color}(u) = \text{color}(v)$     «Try recoloring  $u$  with an existing color»
        for  $i \leftarrow 1$  to  $c$ 
            if no neighbor of  $u$  in  $T$  has color  $i$ 
                 $\text{color}(u) \leftarrow i$ 
    if  $\text{color}(u) = \text{color}(v)$     «Try recoloring  $v$  with an existing color»
        for  $i \leftarrow 1$  to  $c$ 
            if no neighbor of  $v$  in  $T$  has color  $i$ 
                 $\text{color}(v) \leftarrow i$ 
    if  $\text{color}(u) = \text{color}(v)$     «Give up and use a new color»
         $c \leftarrow c + 1$ 
         $\text{color}(u) \leftarrow c$ 
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 $\text{TREECOLOR}(\mathcal{G}) < \alpha \cdot \text{OPT}(\mathcal{G})$ for any graph \mathcal{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( $W[1..n]$ ):
```

```

 $b \leftarrow 0$ 
 $Total[0] \leftarrow \infty$ 
for  $i \leftarrow 1$  to  $n$ 
  if  $Total[b] + W[i] > 1$ 
     $b \leftarrow b + 1$ 
     $Total[b] \leftarrow W[i]$ 
  else
     $Total[b] \leftarrow Total[b] + W[i]$ 
return  $b$ 
```

```
FIRSTFIT( $W[1..n]$ ):
```

```

 $b \leftarrow 0$ 
for  $i \leftarrow 1$  to  $n$ 
   $j \leftarrow 1$ ;  $found \leftarrow \text{FALSE}$ 
  while  $j \leq b$  and  $found = \text{FALSE}$ 
    if  $Total[j] + W[i] \leq 1$ 
       $Total[j] \leftarrow Total[j] + W[i]$ 
       $found \leftarrow \text{TRUE}$ 
     $j \leftarrow j + 1$ 
  if  $found = \text{FALSE}$ 
     $b \leftarrow b + 1$ 
     $Total[b] = W[i]$ 
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 $\text{RANDOM}(m)$ returns an integer chosen uniformly at random from the set $\{1, 2, \dots, m\}$ in $O(1)$ time. For example, a fair coin flip is obtained by calling $\text{RANDOM}(2)$.

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( $Q_1, Q_2$ ):
  if  $Q_1$  is empty return  $Q_2$ 
  if  $Q_2$  is empty return  $Q_1$ 
  if  $key(Q_1) > key(Q_2)$ 
    swap  $Q_1 \leftrightarrow Q_2$ 
  with probability  $1/2$ 
     $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
  else
     $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
  return  $Q_1$ 
```

- (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 $\text{MELD}(Q_1, Q_2)$ 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 $\text{MELD}(Q_1, Q_2)$ runs in $O(\log n)$ time with high probability.
- (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)

3. 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.

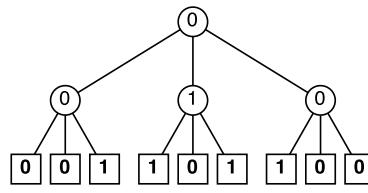
4. Let $X[1..n]$ be an array of n distinct real numbers, and let $N[1..n]$ be an array of indices with the following property: If $X[i]$ is the largest element of X , then $X[N[i]]$ is the smallest element of X ; otherwise, $X[N[i]]$ is the smallest element of X that is larger than $X[i]$.

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

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

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  $i \leftarrow 1$  to  $\infty$ 
    if  $i > \ell_v$ 
         $\ell_v \leftarrow i$ ;  $\pi_v[i] \leftarrow \text{RANDOMBIT}$ 
    if  $i > \ell_w$ 
         $\ell_w \leftarrow i$ ;  $\pi_w[i] \leftarrow \text{RANDOMBIT}$ 
    if  $\pi_v[i] > \pi_w[i]$ 
        return  $v$ 
    else if  $\pi_v[i] < \pi_w[i]$ 
        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 \heartsuit 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.

- \heartsuit 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 $\tilde{w} : E(G) \rightarrow \mathbb{R}$, the positive integer k , and the edge e .

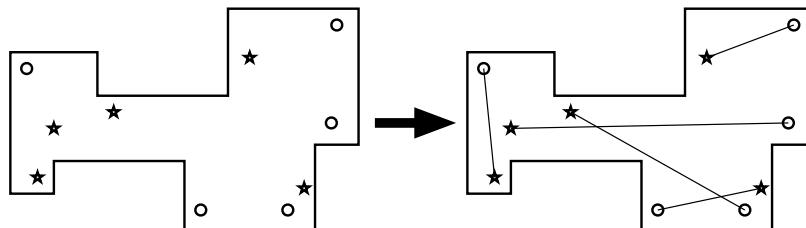
2. Most classical minimum-spanning-tree algorithms use the notions of ‘safe’ and ‘useless’ edges described in the lecture notes, but there is an alternate formulation. Let G be a weighted undirected graph, where the edge weights are distinct. We say that an edge e is *dangerous* if it is the longest 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.]*

γ 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.

γ 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 or 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.*]
 - Υ (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.*]

¹Without 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 $\ell : E \rightarrow \mathbb{R}$, 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 $u \rightarrow v$ has an associated nonnegative *capacity* $c(u \rightarrow v) \geq 0$ and an associated *cost* $\alpha(u \rightarrow v)$. 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 $f_i : E \rightarrow \mathbb{R}_{\geq 0}$ 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* $F : E \rightarrow \mathbb{R}$ is defined as the sum of these individual flows: $F(u \rightarrow v) = \sum_{i=1}^k f_i(u \rightarrow v)$. The aggregate flow $F(u \rightarrow v)$ on any edge must not exceed the capacity $c(u \rightarrow v)$. The goal is to find an aggregate flow whose total *cost* $\sum_{u \rightarrow v} F(u \rightarrow v) \cdot \alpha(u \rightarrow v)$ 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:

$$\begin{array}{ccc} \text{Primal (II)} & \iff & \text{Dual (II)} \\ \boxed{\begin{array}{ll} \max & c \cdot x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0 \end{array}} & & \boxed{\begin{array}{ll} \min & y \cdot b \\ \text{s.t.} & yA \geq c \\ & y \geq 0 \end{array}} \end{array}$$

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

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

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:

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

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:

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

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 $Ax \geq b$ 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.)

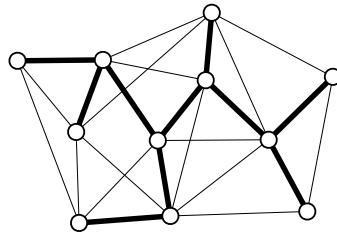
- (a) Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do *not* follow the same greedy strategy as Elmo.
 (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that 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 .



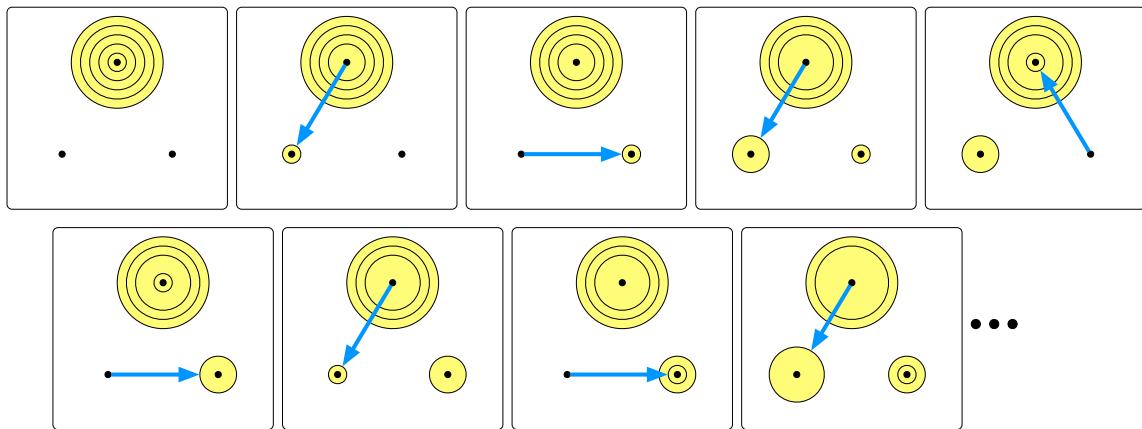
A set of intervals. The seven shaded intervals form a tiling path.

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

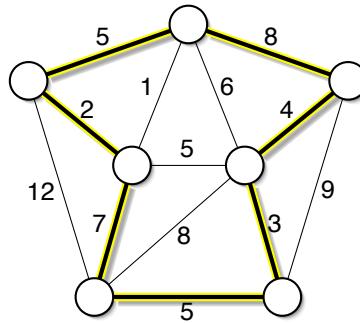
Write your answers in the separate answer booklet.

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

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

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

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 $\text{RANDOM}(m)$ returns an integer chosen uniformly at random from the set $\{1, 2, \dots, m\}$ 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 $\langle a_1, a_2, a_3, \dots, a_n \rangle$ 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 $c : E \rightarrow \mathbb{Z}_+$, and a maximum flow $F : E \rightarrow \mathbb{Z}$ from some vertex s to some other vertex t in G . Describe and analyze efficient algorithms for the following operations:
 - (a) $\text{INCREMENT}(e)$ — Increase the capacity of edge e by 1 and update the maximum flow F .
 - (b) $\text{DECREMENT}(e)$ — 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 {red, green, blue}. Prove that the expected number of interesting edges is at least $\frac{2}{3}wow(G)$.

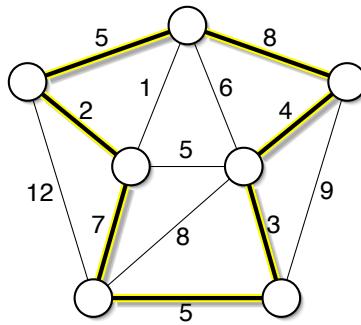
7.

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 $\text{RANDOM}(m)$ returns an integer chosen uniformly at random from the set $\{1, 2, \dots, m\}$ 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 $c : E \rightarrow \mathbb{Z}_+$ and a maximum flow $F : E \rightarrow \mathbb{Z}$ from some vertex s to some other vertex t in G . Describe and analyze efficient algorithms for the following operations:

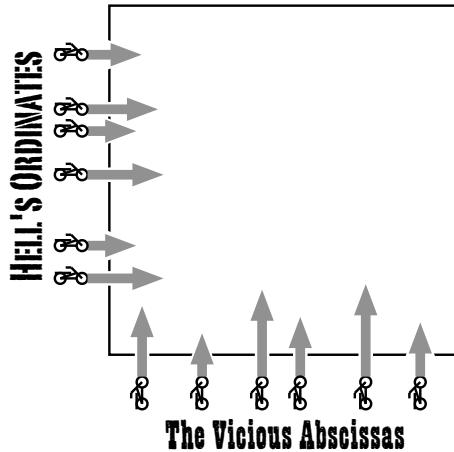
- (a) $\text{INCREMENT}(e)$ — Increase the capacity of edge e by 1 and update the maximum flow F .
- (b) $\text{DECREMENT}(e)$ — Decrease the capacity of edge e by 1 and update the maximum flow F .

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 $\langle a_1, a_2, a_3, \dots, a_n \rangle$ 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.
- 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 $10^{10^{100}}$.
 - 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 {red, green, blue}. Prove that the expected number of interesting edges is at least $\frac{2}{3}wow(G)$.
7. It's time for the 3rd Quasi-Annual Champaign-Urbana Ice Motorcycle Demolition Derby Race-O-Rama and Spaghetti Bake-Off! The main event is a competition between two teams of n motorcycles in a huge square ice-covered arena. All of the motorcycles have spiked tires so that they can ride on the ice. Each motorcycle drags a long metal chain behind it. Whenever a motorcycle runs over a chain, the chain gets caught in the tire spikes, and the motorcycle crashes. Two motorcycles can also crash by running directly into each other. All the motorcycle start simultaneously. Each motorcycle travels in a straight line at a constant speed until it either crashes or reaches the opposite wall—no turning, no braking, no speeding up, no slowing down. The Vicious Abscissas start at the south wall of the arena and ride directly north (vertically). Hell's Ordinates start at the west wall of the arena and ride directly east (horizontally). If any motorcycle completely crosses the arena, that rider's entire team wins the competition.

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



CS 473U: Undergraduate Algorithms, Fall 2006

Homework 0

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

Name:	
Net ID:	Alias:

I understand the Homework Instructions and FAQ.

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

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

- Please carefully read the Homework Instructions and FAQ on the course web page, and then check the box above. This page describes what we expect in your homework solutions—start each numbered problem on a new sheet of paper, write your name and NetID on every page, don’t turn in source code, analyze and prove everything, use good English and good logic, and so on—as well as policies on grading standards, regrading, and plagiarism. **See especially the policies regarding the magic phrases “I don’t know” and “and so on”.** If you have *any* questions, post them to the course newsgroup or ask in lecture.
 - This homework tests your familiarity with prerequisite material—basic data structures, big-Oh notation, recurrences, discrete probability, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Each numbered problem is worth 10 points; not all subproblems have equal weight.
-

#	1	2	3	4	5	6*	Total
Score							
Grader							

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

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

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

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

In case you've forgotten:

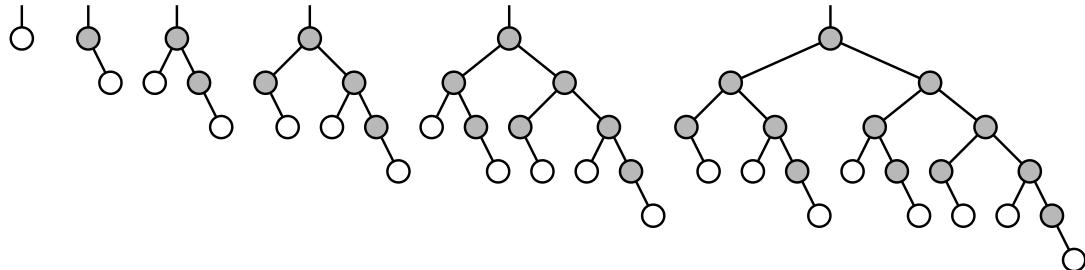
- $\lg n = \log_2 n \neq \ln n = \log_e n$
- $\lg^3 n = (\lg n)^3 \neq \lg \lg \lg n$.
- The harmonic numbers: $H_n = \sum_{i=1}^n 1/i \approx \ln n + 0.577215\dots$
- The Fibonacci numbers: $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$

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

- (a) $A(n) = 2A(n/4) + \sqrt{n}$
- (b) $B(n) = 3B(n/3) + n/\lg n$
- (c) $C(n) = \frac{2C(n-1)}{C(n-2)}$ [Hint: This is easy!]
- (d) $D(n) = D(n-1) + 1/n$
- (e) $E(n) = E(n/2) + D(n)$
- (f) $F(n) = 4F\left(\left\lceil \frac{n-8}{2} \right\rceil + \left\lfloor \frac{3n}{\log_\pi n} \right\rfloor\right) + 6\binom{n+5}{2} - 42n \lg^7 n + \sqrt{13n-6} + \frac{\lg \lg n + 1}{\lg n \lg \lg \lg n}$
- (g) $G(n) = 2G(n-1) - G(n-2) + n$
- (h) $H(n) = 2H(n/2) - 2H(n/4) + 2^n$
- (i) $I(n) = I(n/2) + I(n/4) + I(n/6) + I(n/12) + n$
- ★(j) $J(n) = \sqrt{n} \cdot J(2\sqrt{n}) + n$
[Hint: First solve the secondary recurrence $j(n) = 1 + j(2\sqrt{n})$.]

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

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



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

- Prove that the number of leaves in \mathcal{F}_n is precisely the n th Fibonacci number: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$.
- How many nodes does \mathcal{F}_n have? For full credit, give an *exact*, closed-form answer in terms of Fibonacci numbers, and prove your answer is correct.
- Prove that the left subtree of \mathcal{F}_n is a copy of \mathcal{F}_{n-2} .

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

$\text{RANGETOP}(a, z) : \text{return the highest-priority record whose key is between } a \text{ and } z.$

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

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

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

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

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

5. Penn and Teller agree to play the following game. Penn shuffles a standard deck¹ 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 ($3\clubsuit$), at which point the remaining undrawn cards instantly burst into flames.

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

- (a) What is the expected number of cards that Teller draws?
- (b) What is the expected *maximum* value among the cards Teller gives to Penn?
- (c) What is the expected *minimum* value among the cards Teller gives to Penn?
- (d) What is the expected number of cards that Teller gives to Penn?

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

*6. [Extra credit]³

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

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

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

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

- (a) Prove that in any lazy binary number, between any two 2s there is at least one 0, and between two 0s there is at least one 2.
- (b) Prove that for any natural number N , the sum of the digits of the lazy binary representation of N is exactly $\lfloor \lg(N + 1) \rfloor$.

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

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

³The “I don’t know” rule does not apply to extra credit problems. There is no such thing as “partial extra credit”.

CS 473U: Undergraduate Algorithms, Fall 2006

Homework 1

Due Tuesday, September 12, 2006 in 3229 Siebel Center

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

1. Recall from lecture that a *subsequence* of a sequence A consists of a (not necessarily contiguous) collection of elements of A , arranged in the same order as they appear in A . If B is a subsequence of A , then A is a *supersequence* of B .
 - (a) Describe and analyze a **simple** recursive algorithm to compute, given two sequences A and B , the length of the *longest common subsequence* of A and B . For example, given the strings ALGORITHM and ALTRUISTIC, your algorithm would return 5, the length of the longest common subsequence ALRIT.
 - (b) Describe and analyze a **simple** recursive algorithm to compute, given two sequences A and B , the length of a *shortest common supersequence* of A and B . For example, given the strings ALGORITHM and ALTRUISTIC, your algorithm would return 14, the length of the shortest common supersequence ALGTORUISTHIMC.
 - (c) Let $|A|$ denote the length of sequence A . For any two sequences A and B , let $\text{lcs}(A, B)$ denote the length of the longest common subsequence of A and B , and let $\text{scs}(A, B)$ denote the length of the shortest common supersequence of A and B .
Prove that $|A| + |B| = \text{lcs}(A, B) + \text{scs}(A, B)$ for all sequences A and B . [Hint: There is a simple non-inductive proof.]

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

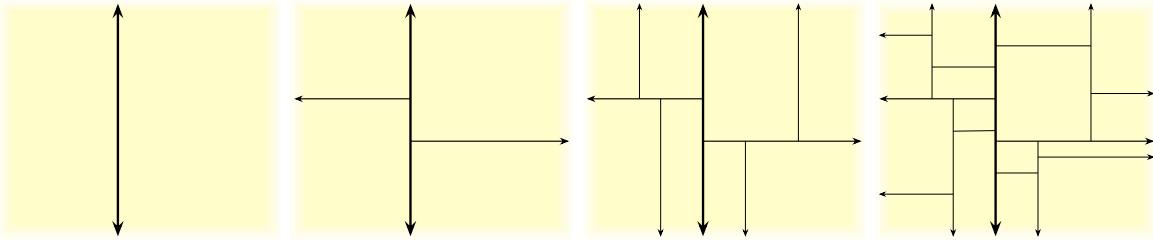
2. You are a contestant on a game show, and it is your turn to compete in the following game. You are presented with an $m \times n$ grid of boxes, each containing a unique number. It costs \$100 to open a box. Your goal is to find a box whose number is larger than its neighbors in the grid (above, below, left, and right). If you spend less money than your opponents, you win a week-long trip for two to Las Vegas and a year's supply of Rice-A-Roni™, to which you are hopelessly addicted.
 - (a) Suppose $m = 1$. Describe an algorithm that finds a number that is bigger than any of its neighbors. How many boxes does your algorithm open in the worst case?
 - (b) Suppose $m = n$. Describe an algorithm that finds a number that is bigger than any of its neighbors. How many boxes does your algorithm open in the worst case?
 - (c) [**Extra credit**]¹ Prove that your solution to part (b) is asymptotically optimal.

¹The “I don’t know” rule does not apply to extra credit problems. There is no such thing as “partial extra credit”.

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

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

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



The first four levels of a typical kd-tree.

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

- Consider the horizontal line $y = t$, where $t \neq p_v$ for all nodes v in T . Exactly how many leaf boxes of T does this line intersect? [Hint: The parity of the root node matters.] Prove your answer is correct. A correct $\Theta(\cdot)$ bound is worth significant partial credit.
- Describe and analyze an efficient algorithm to compute, given T and an arbitrary horizontal line ℓ , the number of leaf boxes of T that lie entirely above ℓ .

CS 473U: Undergraduate Algorithms, Fall 2006

Homework 2

Due Tuesday, September 19, 2006 in 3229 Siebel Center

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

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

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

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

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

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

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

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

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

CS 473U: Undergraduate Algorithms, Fall 2006

Homework 3

Due Wednesday, October 4, 2006 in 3229 Siebel Center

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

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

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

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

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

- Let H_1 be the input heap with greater priority, and H_2 the input heap with lower priority. (That is, the priority of $\text{root}(H_1)$ is greater than the priority of $\text{root}(H_2)$.) Let H_L be the left subtree of $\text{root}(H_1)$ and H_R be the right subtree of $\text{root}(H_1)$.
 - We set $\text{root}(H) = \text{root}(H_1)$.
 - We now flip a coin that comes up either “Left” or “Right” with equal probability.
 - If it comes up “Left”, we set the left subtree of $\text{root}(H)$ to be H_L , and the right subtree of $\text{root}(H)$ to be $\text{meld}(H_R, H_2)$ (defined recursively).
 - If the coin comes up “Right”, we set the right subtree of $\text{root}(H)$ to be H_R , and the left subtree of $\text{root}(H)$ to be $\text{meld}(H_L, H_2)$.
 - As a base case, melding any heap H_1 with an empty heap gives H_1 .
- (a) Analyze the expected running time of $\text{meld}(H_a, H_b)$ if H_a is a (meldable) heap with n elements, and H_b is a (meldable) heap with m elements.
- (b) Describe how to perform each of the following operations using only melds, and give the running time of each.
- $\text{DeleteMax}(H)$, which deletes the element with greatest priority.
 - $\text{Insert}(H, x)$, which inserts the element x into the heap H .
 - $\text{Delete}(H, x)$, which - given a pointer to element x in heap H - returns the heap with x deleted.

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

```

QuickSelect(Array A, n, k)
pivot ← Random(1, n)
S ← { $x \mid x \in A, x < A[pivot]$ }
s ← |S|
L ← { $x \mid x \in A, x > A[pivot]$ }
if ( $k = s + 1$ )
    return  $A[pivot]$ 
else if ( $k \leq s$ )
    return QuickSelect( $S, s, k$ )
else
    return QuickSelect( $L, n - (s + 1), k - (s + 1)$ )

```

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

- (a) Write a recurrence relation for the expected running time of QuickSelect.
 - (b) Given any two elements $x, y \in A$, what is the probability that x and y will be compared?
 - (c) Either from part (a) or part (b), find the expected running time of QuickSelect.
4. [Extra Credit]: In the previous problem, we found a $\Theta(n)$ algorithm for selecting the k th smallest element, but the constant hidden in the $\Theta(\cdot)$ notation is somewhat large. It is easy to find the *smallest* element using at most n comparisons; we would like to be able to extend this to larger k . Can you find a randomized algorithm that uses $n + \Theta(k \log k \log n)$ ¹ expected comparisons? (Note that there is no constant multiplying the n .)

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

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

CS 473U: Undergraduate Algorithms, Fall 2006

Homework 4

Due Tuesday, October 10, 2006 in 3229 Siebel Center

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

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

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

2. Design and analyze a simple data structure that maintains a list of integers and supports the following operations.
 - (a) $\text{CREATE}()$: creates and returns a new list L
 - (b) $\text{PUSH}(L, x)$: appends x to the end of L
 - (c) $\text{POP}(L)$: deletes the last entry of L and returns it
 - (d) $\text{LOOKUP}(L, k)$: returns the k th entry of L

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

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

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

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

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

- (a) $\text{CREATED}()$: creates a new data structure D containing no points. It should have a worst-case constant running time.
- (b) $\text{INSERT}_D(D, p)$: inserts p into D . It should run in amortized $O(\log n) \cdot T(n)/n$ time.
- (c) $\text{QUERY}_D(D, R)$: returns the number of points in D that are contained in the region R . It should have a worst-case running time of $O(\log n) \cdot Q(n)$.

CS 473U: Undergraduate Algorithms, Fall 2006

Homework 5

Due Tuesday, October 24, 2006 in 3229 Siebel Center

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

1. Makefiles:

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

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

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

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

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

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

CS 473U: Undergraduate Algorithms, Fall 2006

Homework 6

Due Wednesday, November 8, 2006 in 3229 Siebel Center

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

1. Dijkstra's algorithm can be used to determine shortest paths on graphs with some negative edge weights (as long as there are no negative cycles), but the worst-case running time is much worse than the $O(E + V \log V)$ it takes when the edge weights are all positive. Construct an infinite family of graphs - with negative edge weights - for which the asymptotic running time of Dijkstra's algorithm is $\Omega(2^{|V|})$.
2. It's a cold and rainy night, and you have to get home from Siebel Center. Your car has broken down, and it's too windy to walk, which means you have to take a bus. To make matters worse, there is no bus that goes directly from Siebel Center to your apartment, so you have to change buses some number of times on your way home. Since it's cold outside, you want to spend as little time as possible waiting in bus shelters.

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

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

CS 473U: Undergraduate Algorithms, Fall 2006

Homework 6

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

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

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

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

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

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

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

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

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

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

CS 473U: Undergraduate Algorithms, Fall 2006

Homework 8

Due Wednesday, December 6, 2006 in 3229 Siebel Center

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

1. Given an array $A[1..n]$ of $n \geq 2$ distinct integers, we wish to find the second largest element using as few comparisons as possible.
 - (a) Give an algorithm which finds the second largest element and uses at most $n + \lceil \lg n \rceil - 2$ comparisons in the worst case.
 - *(b) Prove that every algorithm which finds the second largest element uses at least $n + \lceil \lg n \rceil - 2$ comparisons in the worst case.
2. Let R be a set of rectangles in the plane. For each point p in the plane, we say that the *rectangle depth* of p is the number of rectangles in R that contain p .
 - (a) (Step 1: Algorithm Design) Design and analyze a polynomial-time algorithm which, given R , computes the maximum rectangle depth.
 - (b) (Step 2: ???) Describe and analyze a polynomial-time reduction from the maximum rectangle depth problem to the maximum clique problem.
 - (c) (Step 3: Profit!) In 2000, the Clay Mathematics Institute described the Millennium Problems: seven challenging open problems which are central to ongoing mathematical research. The Clay Institute established seven prizes, each worth one million dollars, to be awarded to anyone who solves a Millennium problem. One of these problems is the $P = NP$ question. In (a), we developed a polynomial-time algorithm for the maximum rectangle depth problem. In (b), we found a reduction from this problem to an NP-complete problem. We know from class that if we find a polynomial-time algorithm for any NP-complete problem, then we have shown $P = NP$. Why hasn't Jeff used (a) and (b) to show $P = NP$ and become a millionaire?
3. Let G be a complete graph with integer edge weights. If C is a cycle in G , we say that the *cost* of C is the sum of the weights of edges in C . Given G , the traveling salesman problem (TSP) asks us to compute a Hamiltonian cycle of minimum cost. Given G , the traveling salesman cost problem (TSCP) asks us to compute the cost of a minimum cost Hamiltonian cycle. Given G and an integer k , the traveling salesman decision problem (TSDP) asks us to decide if there is a Hamiltonian cycle in G of cost at most k .
 - (a) Describe and analyze a polynomial-time reduction from TSP to TSCP.
 - (b) Describe and analyze a polynomial-time reduction from TSCP to TSDP.
 - (c) Describe and analyze a polynomial-time reduction from TSDP to TSP.

- (d) What can you conclude about the relative computational difficulty of TSP, TSCP, and TSDP?
4. Let G be a graph. A set S of vertices of G is a *dominating set* if every vertex in G is either in S or adjacent to a vertex in S . Show that, given G and an integer k , deciding if G contains a dominating set of size at most k is NP-complete.

1. Probability

- (a) n people have checked their hats with a hat clerk. The clerk is somewhat absent-minded and returns the hats uniformly at random (with no regard for whether each hat is returned to its owner). On average, how many people will get back their own hats?
 - (b) Let S be a uniformly random permutation of $\{1, 2, \dots, n-1, n\}$. As we move from the left to the right of the permutation, let X denote the smallest number seen so far. On average, how many different values will X take?
2. A *tournament* is a directed graph where each pair of distinct vertices u, v has either the edge uv or the edge vu (but not both). A *Hamiltonian path* is a (directed) path that visits each vertex of the (di)graph. Prove that every tournament has a Hamiltonian path.
3. Describe and analyze a data structure that stores a set of n records, each with a numerical *key*, such that the following operation can be performed quickly:

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

For example, if the keys are:

3 4 9 6 5 8 7 1 0

then **Foo(2)** would return 42, since 3, 4, 5, 6, 7, 8, 9 are all larger than 2 and $3 + 4 + 5 + 6 + 7 + 8 + 9 = 42$. You may assume that no two records have equal keys, and that no record has a key equal to a . Analyze both the size of your data structure and the running time of your **Foo** algorithm. Your data structure must be as small as possible and your **Foo** algorithm must be as fast as possible.

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

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

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

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

- (a) Show that it is possible to cover all but one square of a 64×64 checkerboard using L-shape trominoes. (In your covering, each tromino should cover three squares and no square should be covered more than once.)
- (b) Show that you can leave *any* single square uncovered.
- (c) Can you cover all but one square of a 64×64 checkerboard using *line* trominoes? If so, which squares can you leave uncovered?

1. Moving on a Checkerboard

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

- 1) the square immediately above
- 2) the square that is one up and one to the left (but only if the checker is not already in the leftmost column)
- 3) the square that is one up and one to the right (but only if the checker is not already in the rightmost column)

Each time you move from square x to square y , you receive $p(x, y)$ dollars. You are given a list of the values $p(x, y)$ for each pair (x, y) for which a move from x to y is legal. Do not assume that $p(x, y)$ is positive.

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

2. Maximizing Profit

You are given lists of values h_1, h_2, \dots, h_k and l_1, l_2, \dots, l_k . For each i you can choose $j_i = h_i$, $j_i = l_i$, or $j_i = 0$; the only catch is that if $j_i = h_i$ then j_{i-1} must be 0 (except for $i = 1$). Your goal is to maximize $\sum_{i=1}^k j_i$.

Give an efficient algorithm that returns the maximum possible value of $\sum_{i=1}^k j_i$.

3. Maximum alternating subsequence

An *alternating sequence* is a sequence a_1, a_2, \dots such that no three consecutive terms of the sequence satisfy $a_i > a_{i+1} > a_{i+2}$ or $a_i < a_{i+1} < a_{i+2}$.

Given a sequence, efficiently find the longest alternating subsequence it contains. What is the running time of your algorithm?

1. Championship Showdown

What excitement! The Champaign Spinners and the Urbana Dreamweavers have advanced to meet each other in the World Series of Basketweaving! The World Champions will be decided by a best of $2n - 1$ series of head-to-head weaving matches, and the first to win n matches will take home the coveted Golden Basket (for example, a best-of-7 series requires four match wins, but we will keep the generalized case). We know that for any given match there is a constant probability p that Champaign will win, and a subsequent probability $q = 1 - p$ that Urbana will win.

Let $P(i, j)$ be the probability that Champaign will win the series given that they still need i more victories, whereas Urbana needs j more victories for the championship. $P(0, j) = 1$, $1 \leq j \leq n$, because Champaign needs no more victories to win. $P(i, 0) = 0$, $1 \leq i \leq n$, as Champaign cannot possibly win if Urbana already has. $P(0, 0)$ is meaningless. Champaign wins any particular match with probability p and loses with probability q , so

$$P(i, j) = p \cdot P(i - 1, j) + q \cdot P(i, j - 1)$$

for any $i \geq 1$ and $j \geq 1$.

Create and analyze an $O(n^2)$ -time dynamic programming algorithm that takes the parameters n, p , and q and returns the probability that Champaign will win the series (that is, calculate $P(n, n)$).

2. Making Change

Suppose you are a simple shopkeeper living in a country with n different types of coins, with values $1 = c[1] < c[2] < \dots < c[n]$. (In the U.S., for example, $n = 6$ and the values are 1, 5, 10, 25, 50, and 100 cents.) Your beloved benevolent dictator, El Generalissimo, has decreed that whenever you give a customer change, you must use the smallest possible number of coins, so as not to wear out the image of El Generalissimo lovingly engraved on each coin by servants of the Royal Treasury.

Describe and analyze a dynamic programming algorithm to determine, given a target amount A and a sorted array $c[1..n]$ of coin values, the smallest number of coins needed to make A cents in change. You can assume that $c[1] = 1$, so that it is possible to make change for any amount A .

3. Knapsack

You are a thief, who is trying to choose the best collection of treasure (some subset of the n treasures, numbered 1 through n) to steal. The weight of item i is $w_i \in \mathbb{N}$ and the profit is $p_i \in \mathbb{R}$. Let $C \in \mathbb{N}$ be the maximum weight that your knapsack can hold. Your goal is to choose a subset of elements $S \subseteq \{1, 2, \dots, n\}$ that maximizes your total profit $P(S) = \sum_{i \in S} p_i$, subject to the constraint that the sum of the weights $W(S) = \sum_{i \in S} w_i$ is not more than C .

Give an algorithm that runs in time $O(Cn)$.

1. Randomized Edge Cuts

We will randomly partition the vertex set of a graph G into two sets S and T . The algorithm is to flip a coin for each vertex and with probability $1/2$, put it in S ; otherwise put it in T .

- (a) Show that the expected number of edges with one endpoint in S and the other endpoint in T is exactly half the edges in G .
- (b) Now say the edges have weights. What can you say about the sum of the weights of the edges with one endpoint in S and the other endpoint in T ?

2. Skip Lists

A *skip list* is built in layers. The bottom layer is an ordinary sorted linked list. Each higher layer acts as an “express lane” for the lists below, where an element in layer i appears in layer $i+1$ with some fixed probability p .

```

1
1-----4---6
1---3-4---6-----9
1-2-3-4-5-6-7-8-9-10

```

- (a) What is the probability a node reaches height h .
- (b) What is the probability any node is above $c \log n$ (for some fixed value of c)?
Compute the value explicitly when $p = 1/2$ and $c = 4$.
- (c) To search for an entry x , scan the top layer until you find the last entry y that is less than or equal to x . If $y < x$, drop down one layer and in this new layer (beginning at y) find the last entry that is less than or equal to x . Repeat this process (dropping down a layer, then finding the last entry less than or equal to x) until you either find x or reach the bottom layer and confirm that x is not in the skip list. What is the expected search time?
- (d) Describe an efficient method for insertion. What is the expected insertion time?

3. Clock Solitaire

In a standard deck of 52 cards, put 4 face-down in each of the 12 ‘hour’ positions around a clock, and 4 face-down in a pile in the center. Turn up a card from the center, and look at the number on it. If it’s number x , place the card face-up next to the face-down pile for x , and turn up the next card in the face-down pile for x (that is, the face-down pile corresponding to hour x). You win if, for each $\text{Ace} \leq x \leq \text{Queen}$, all four cards of value x are turned face-up before all four Kings (the center cards) are turned face-up.

What is the probability that you win a game of Clock Solitaire?

1. Simulating Queues with Stacks

A *queue* is a first-in-first-out data structure. It supports two operations *push* and *pop*. Push adds a new item to the back of the queue, while pop removes the first item from the front of the queue. A *stack* is a last-in-first-out data structure. It also supports push and pop. As with a queue, push adds a new item to the back of the queue. However, pop removes the last item from the back of the queue (the one most recently added).

Show how you can simulate a queue by using two stacks. Any sequence of pushes and pops should run in amortized constant time.

2. Multistacks

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

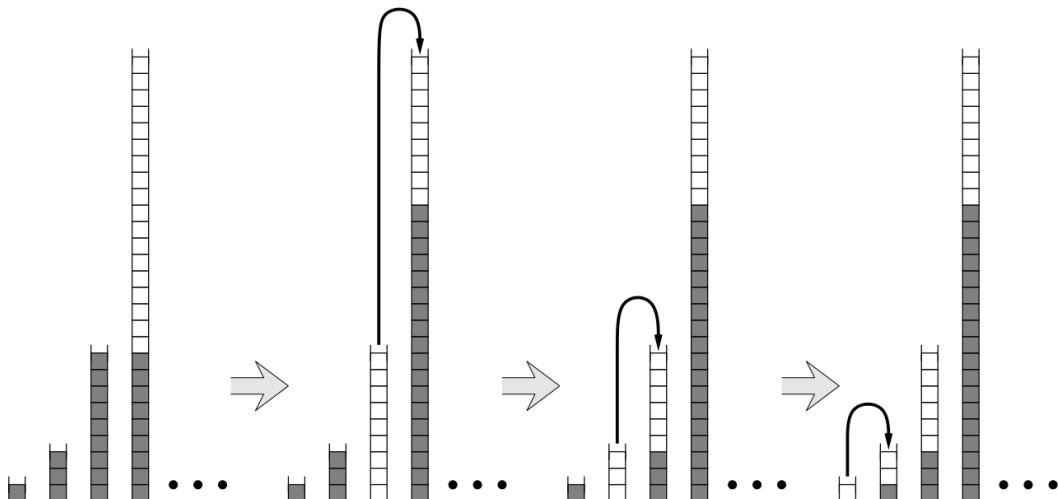


Figure 1. Making room for one new element in a multistack.

- (a) In the worst case, how long does it take to push one more element onto a multistack containing n elements?
- (b) Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack.

3. Powerhungry function costs

A sequence of n operations is performed on a data structure. The i th operation costs i if i is an exact power of 2, and 1 otherwise. Determine the amortized cost of the operation.

1. Representation of Integers

- (a) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, $17 = F_7 + F_4 + F_2$.
- (b) Prove that any integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example $42 = 3^4 - 3^3 - 3^2 - 3^1$, $25 = 3^3 - 3^1 + 3^0$, $17 = 3^3 - 3^2 - 3^0$.

2. Minimal Dominating Set

Suppose you are given a rooted tree T (not necessarily binary). You want to label each node in T with an integer 0 or 1, such that every node either has the label 1 or is adjacent to a node with the label 1 (or both). The *cost* of a labeling is the number of nodes with label 1. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree T .

3. Names in Boxes

The names of 100 prisoners are placed in 100 wooden boxes, one name to a box, and the boxes are lined up on a table in a room. One by one, the prisoners are led into the room; each may look in at most 50 boxes, but must leave the room exactly as he found it and is permitted no further communication with the others.

The prisoners have a chance to plot their strategy in advance, and they are going to need it, because unless *every single prisoner finds his own name* all will subsequently be executed. Find a strategy for them which has probability of success exceeding 30%. You may assume that the names are distributed in the boxes uniformly at random.

- (a) Calculate the probability of success if each prisoner picks 50 boxes uniformly at random.
- *(b) Consider the following strategy.

The prisoners number themselves 1 to 100. Prisoner i begins by looking in box i . There he finds the name of prisoner j . If $j \neq i$, he continues by looking in box j . As long as prisoner i has not found his name, he continues by looking in the box corresponding to the last name he found.

Describe the set of permutations of names in boxes for which this strategy will succeed.

- *(c) Count the number of permutations for which the strategy above succeeds. Use this sum to calculate the probability of success. You may find it useful to do this calculation for general n , then set $n = 100$ at the end.
- (d) We assumed that the names were distributed in the boxes uniformly at random. Explain how the prisoners could augment their strategy to make this assumption unnecessary.

1. Dynamic MSTs

Suppose that you already have a minimum spanning tree (MST) in a graph. Now one of the edge weights changes. Give an efficient algorithm to find an MST in the new graph.

2. Minimum Bottleneck Trees

In a graph G , for any pair of vertices u, v , let $\text{bottleneck}(u, v)$ be the maximum over all paths p_i from u to v of the minimum-weight edge along p_i . Construct a spanning tree T of G such that for each pair of vertices, their bottleneck in G is the same as their bottleneck in T .

One way to think about it is to imagine the vertices of the graph as islands, and the edges as bridges. Each bridge has a maximum weight it can support. If a truck is carrying stuff from u to v , how much can the truck carry? We don't care what route the truck takes; the point is that the smallest-weight edge on the route will determine the load.

3. Eulerian Tours

An *Eulerian tour* is a “walk along edges of a graph” (in which successive edges must have a common endpoint) that uses each edge exactly once and ends at the vertex where it starts. A graph is called Eulerian if it has an Eulerian tour.

Prove that a connected graph is Eulerian iff each vertex has even degree.

1. Alien Abduction

Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road won't be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What route should he take so that he has the least chance of being abducted?

More formally, you are given a directed graph $G = (V, E)$, where every edge e has an independent safety probability $p(e)$. The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex s to a given target vertex t .

2. The Only SSSP Algorithm

In the lecture notes, Jeff mentions that all SSSP algorithms are special cases of the following generic SSSP algorithm. Each vertex v in the graph stores two values, which describe a tentative shortest path from s to v .

- $\text{dist}(v)$ is the length of the tentative shortest $s \rightsquigarrow v$ path.
- $\text{pred}(v)$ is the predecessor of v in the shortest $s \rightsquigarrow v$ path.

We call an edge *tense* if $\text{dist}(u) + w(u \rightarrow v) < \text{dist}(v)$. Our generic algorithm repeatedly finds a tense edge in the graph and *relaxes* it:

$$\boxed{\begin{array}{c} \text{Relax}(u \rightarrow v): \\ \hline \text{dist}(v) \leftarrow \text{dist}(u) + w(u \rightarrow v) \\ \text{pred}(v) \leftarrow u \end{array}}$$

If there are no tense edges, our algorithm is finished, and we have our desired shortest path tree. The correctness of the relaxation algorithm follows directly from three simple claims. The first of these is below. Prove it.

- When the algorithm halts, if $\text{dist}(v) \neq \infty$, then $\text{dist}(v)$ is the total weight of the predecessor chain ending at v :

$$s \rightarrow \dots \rightarrow (\text{pred}(\text{pred}(v))) \rightarrow \text{pred}(v) \rightarrow v.$$

3. Can't find a Cut-edge

A cut-edge is an edge which when deleted disconnects the graph. Prove or disprove the following. Every 3-regular graph has no cut-edge. (A common approach is induction.)

1. Max-Flow with vertex capacities

In a standard $s - t$ Maximum-Flow Problem, we assume edges have capacities, and there is no limit on how much flow is allowed to pass through a node. In this problem, we consider the variant of Maximum-Flow and Minimum-Cut problems with node capacities.

More specifically, each node, n_i , has a capacity c_i . The edges have unlimited capacity. Show how you can model this problem as a standard Max-flow problem (where the weights are on the edges).

2. Emergency evacuation

Due to large-scale flooding in a region, paramedics have identified a set of n injured people distributed across the region who need to be reashed to hospitals. There are k hospitals in the region, and each of the n people needs to be brought to a hospital that is within a half-hour's driving time of their current location.

At the same time, we don't want to overload any hospital by sending too many patients its way. We'd like to distribute the people so that each hospital receives at most $\lceil n/k \rceil$ people.

Show how to model this problem as a Max-flow problem.

3. Tracking a Hacker

A computer network (with each edge weight 1) is designed to carry traffic from a source s to a destination t . Recently, a computer hacker destroyed some of the edges in the graph. Normally, the maximum $s - t$ flow in G is k . Unfortunately, there is currently no path from s to t . Fortunately, the sysadmins know that the hacker destroyed at most k edges of the graph.

The sysadmins are trying to diagnose which of the nodes of the graph are no longer reachable. They would like to avoid testing each node. They are using a monitoring tool with the following behavior. If you use the command $\text{ping}(v)$, for a given node v , it will tell you whether there is currently a path from s to v (so $\text{ping}(t)$ will return `False` but $\text{ping}(s)$ will return `True`).

Give an algorithm that accomplishes this task using only $O(k \log n)$ pings. (You may assume that any algorithm you wish to run on the original network (before the hacker destroyed edges) runs for free, since you have a model of that network on your computer.)

1. Updating a maximum flow

Suppose you are given a directed graph $G = (V, E)$, with a positive integer capacity c_e on each edge e , a designated source $s \in V$, and a designated sink $t \in V$. You are also given a maximum $s - t$ flow in G , defined by a flow value f_e on each edge e . The flow $\{f_e\}$ is *acyclic*: There is no cycle in G on which all edges carry positive flow.

Now suppose we pick a specific edge $e^* \in E$ and reduce its capacity by 1 unit. Show how to find a maximum flow in the resulting capacitated graph in time $O(m + n)$, where m is the number of edges in G and n is the number of nodes.

2. Cooking Schedule

You live in a cooperative apartment with n other people. The co-op needs to schedule cooks for the next n days, so that each person cooks one day and each day there is one cook. In addition, each member of the co-op has a list of days they are available to cook (and is unavailable to cook on the other days).

Because of your superior CS473 skills, the co-op selects you to come up with a schedule for cooking, so that everyone cooks on a day they are available.

- (a) Describe a bipartite graph G so that G has a perfect matching if and only if there is a feasible schedule for the co-op.
- (b) A friend of yours tried to help you out by coming up with a cooking schedule. Unfortunately, when you look at the schedule he created, you notice a big problem. $n - 2$ of the people are scheduled for different nights on which they are available: no problem there. But the remaining two people are assigned to cook on the same night (and no one is assigned to the last night).

You want to fix your friend's mistake, but without having to recompute everything from scratch. Show that it's possible, using his "almost correct" schedule to decide in $O(n^2)$ time whether there exists a feasible schedule.

3. Disjoint paths in a digraph

Let $G = (V, E)$ be a directed graph, and suppose that for each node v , the number of edges into v is equal to the number of edges out of v . That is, for all v ,

$$|\{(u, v) : (u, v) \in E\}| = |\{(v, w) : (v, w) \in E\}|.$$

Let x, y be two nodes of G , and suppose that there exist k mutually edge-disjoint paths from x to y . Under these conditions, does it follow that there exist k mutually edge-disjoint paths from y to x . Give a proof or a counterexample with explanation.

1. String matching: an example

- (a) Build a finite automata to search for the string “bababoon”.
- (b) Use the automata from part (a) to build the prefix function for Knuth-Morris-Pratt.
- (c) Use the automata or the prefix function to search for “bababoon” in the string “babybaboonbuysbananasforotherbababoons”.

2. Cooking Schedule Strikes Back

You live in a cooperative apartment with n other people. The co-op needs to schedule cooks for the next $5n$ days, so that each person cooks five days and each day there is one cook. In addition, each member of the co-op has a list of days they are available to cook (and is unavailable to cook on the other days).

Because of your success at headbanging last week, the co-op again asks you to compose a cooking schedule. Unfortunately, you realize that no such schedule is possible Give a schedule for the cooking so that no one has to cook on more than 2 days that they claim to be unavailable.

3. String matching on Trees

You are given a rooted tree T (not necessarily binary), in which each node has a character. You are also given a pattern $P = p_1p_2 \cdots p_l$. Search for the string as a subtree. In other words, search for a subtree in which p_i is on a child of the node containing p_{i-1} for each $2 \leq i \leq l$.

1. Self-reductions

In each case below assume that you are given a black box which can answer the decision version of the indicated problem. Use a polynomial number of calls to the black box to construct the desired set.

- (a) Independent set: Given a graph G and an integer k , does G have a subset of k vertices that are pairwise nonadjacent?
- (b) Subset sum: Given a multiset (elements can appear more than once) $X = \{x_1, x_2, \dots, x_k\}$ of positive integers, and a positive integer S does there exist a subset of X with sum exactly S ?

2. Lower Bounds

Give adversary arguments to prove the indicated lower bounds for the following problems:

- (a) Searching in a sorted array takes at least $1 + \lfloor \lg_2 n \rfloor$ queries.
- (b) Let M be an $n \times n$ array of real values that is increasing in both rows and columns. Prove that searching for a value requires at least n queries.

3. k -coloring

Show that we can solve the problem of constructing a k -coloring of a graph by using a polynomial number of calls to a black box that determines whether a graph has such a k -coloring. (Hint: Try reducing via an intermediate problem that asks whether a partial coloring of a graph can be extended to a proper k -coloring.)

1. NP-hardness Proofs: Restriction

Prove that each of the following problems is NP-hard. In each part, find a special case of the given problem that is equivalent to a known NP-hard problem.

(a) **Longest Path**

Given a graph G and a positive integer k , does G contain a path with k or more edges?

(b) **Partition into Hamiltonian Subgraphs**

Given a graph G and a positive integer k , can the vertices of G be partitioned into at most k disjoint sets such that the graph induced by each set has a Hamiltonian cycle?

(c) **Set Packing**

Given a collection of finite sets C and a positive integer k , does C contain k disjoint sets?

(d) **Largest Common Subgraph**

Given two graphs G_1 and G_2 and a positive integer k , does there exist a graph G_3 such that G_3 is a subgraph of both G_1 and G_2 and G_3 has at least k edges?

2. Domino Line

You are given an unusual set of dominoes; each domino has a number on each end, but the numbers may be arbitrarily large and some numbers appear on many dominoes, while other numbers only appear on a few dominoes. Your goal is to form a line using all the dominoes so that adjacent dominoes have the same number on their adjacent halves. Either give an efficient algorithm to solve the problem or show that it is NP-hard.

3. Set Splitting

Given a finite set S and a collection of subsets C is there a partition of S into two sets S_1 and S_2 such that no subset in C is contained entirely in S_1 or S_2 ? Show that the problem is NP-hard. (Hint: use NAE-3SAT, which is similar to 3SAT except that a satisfying assignment does not allow all 3 variables in a clause to be true.)

You have 120 minutes to answer four of these five questions.

Write your answers in the separate answer booklet.

1. Multiple Choice.

Each of the questions on this page has one of the following five answers:

A: $\Theta(1)$	B: $\Theta(\log n)$	C: $\Theta(n)$	D: $\Theta(n \log n)$	E: $\Theta(n^2)$
----------------	---------------------	----------------	-----------------------	------------------

Choose the correct answer for each question. Each correct answer is worth +1 point; each incorrect answer is worth $-\frac{1}{2}$ point; each “I don’t know” is worth $+\frac{1}{4}$ point. Your score will be rounded to the nearest *non-negative* integer. You do *not* need to justify your answers; just write the correct letter in the box.

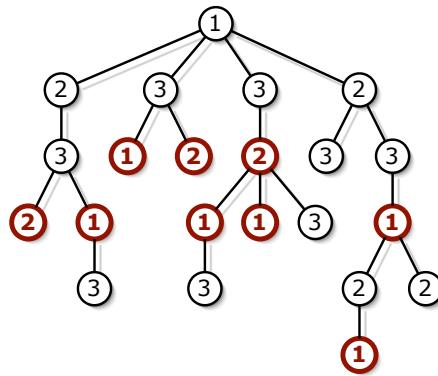
- (a) What is $\frac{5}{n} + \frac{n}{5}$?
- (b) What is $\sum_{i=1}^n \frac{n}{i}$?
- (c) What is $\sum_{i=1}^n \frac{i}{n}$?
- (d) How many bits are required to represent the n th Fibonacci number in binary?
- (e) What is the solution to the recurrence $T(n) = 2T(n/4) + \Theta(n)$?
- (f) What is the solution to the recurrence $T(n) = 16T(n/4) + \Theta(n)$?
- (g) What is the solution to the recurrence $T(n) = T(n - 1) + 1/n^2$?
- (h) What is the worst-case time to search for an item in a binary search tree?
- (i) What is the worst-case running time of quicksort?
- (j) What is the running time of the fastest possible algorithm to solve Sudoku puzzles?
A Sudoku puzzle consists of a 9×9 grid of squares, partitioned into nine 3×3 sub-grids; some of the squares contain digits between 1 and 9. The goal of the puzzle is to enter digits into the blank squares, so that each digit between 1 and 9 appears exactly once in each row, each column, and each 3×3 sub-grid. The initial conditions guarantee that the solution is unique.

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

A Sudoku puzzle. **Don't try to solve this during the exam!**

2. Oh, no! You have been appointed as the gift czar for Giggle, Inc.'s annual mandatory holiday party! The president of the company, who is certifiably insane, has declared that *every* Giggle employee must receive one of three gifts: (1) an all-expenses-paid six-week vacation anywhere in the world, (2) an all-the-pancakes-you-can-eat breakfast for two at Jumping Jack Flash's Flapjack Stack Shack, or (3) a burning paper bag full of dog poop. Corporate regulations prohibit any employee from receiving the same gift as his/her direct supervisor. Any employee who receives a better gift than his/her direct supervisor will almost certainly be fired in a fit of jealousy. How do you decide what gifts everyone gets if you want to minimize the number of people that get fired?

More formally, suppose you are given a rooted tree T , representing the company hierarchy. You want to label each node in T with an integer 1, 2, or 3, such that every node has a different label from its parent.. The *cost* of an labeling is the number of nodes that have smaller labels than their parents. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree T . (Your algorithm does *not* have to compute the actual best labeling—just its cost.)



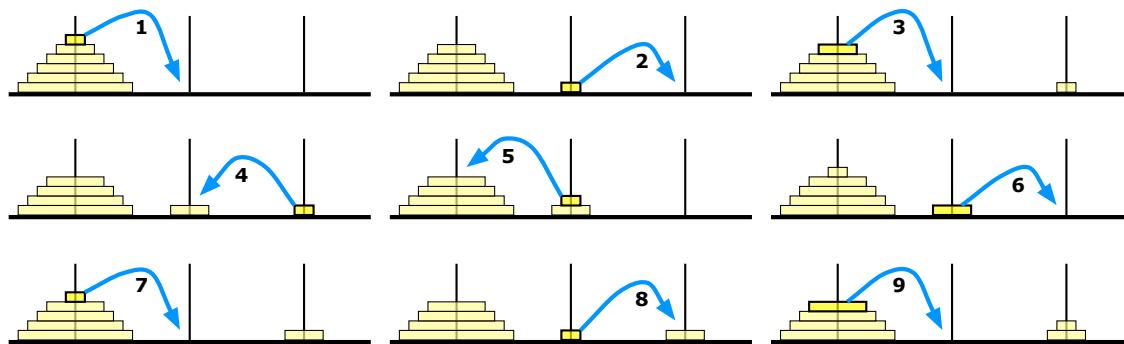
A tree labeling with cost 9. Bold nodes have smaller labels than their parents.
This is *not* the optimal labeling for this tree.

3. Suppose you are given an array $A[1..n]$ of n distinct integers, sorted in increasing order. Describe and analyze an algorithm to determine whether there is an index i such that $A[i] = i$, in $o(n)$ time. [Hint: Yes, that's **little-oh** of n . What can you say about the sequence $A[i] - i$?]
4. Describe and analyze a *polynomial-time* algorithm to compute the length of the longest common subsequence of two strings $A[1..m]$ and $B[1..n]$. For example, given the strings 'DYNAMIC' and 'PROGRAMMING', your algorithm would return the number 3, because the longest common subsequence of those two strings is 'AMI'. You must give a complete, self-contained solution; don't just refer to HW1.

5. Recall that the Tower of Hanoi puzzle consists of three pegs and n disks of different sizes. Initially, all the disks are on one peg, stacked in order by size, with the largest disk on the bottom and the smallest disk on top. In a single move, you can transfer the highest disk on any peg to a different peg, except that you may never place a larger disk on top of a smaller one. The goal is to move all the disks onto one other peg.

Now suppose the pegs are arranged in a row, and you are forbidden to transfer a disk directly between the left and right pegs in a single move; every move must involve the middle peg. How many moves suffice to transfer all n disks from the left peg to the right peg under this restriction? **Prove your answer is correct.**

For full credit, give an *exact* upper bound. A correct upper bound using $O(\cdot)$ notation (with a proof of correctness) is worth 7 points.



The first nine moves in a restricted Towers of Hanoi solution.

1. On an overnight camping trip in Sunnydale National Park, you are woken from a restless sleep by a scream. As you crawl out of your tent to investigate, a terrified park ranger runs out of the woods, covered in blood and clutching a crumpled piece of paper to his chest. As he reaches your tent, he gasps, “Get out... while... you...”, thrusts the paper into your hands, and falls to the ground. Checking his pulse, you discover that the ranger is stone dead.

You look down at the paper and recognize a map of the park, drawn as an undirected graph, where vertices represent landmarks in the park, and edges represent trails between those landmarks. (Trails start and end at landmarks and do not cross.) You recognize one of the vertices as your current location; several vertices on the boundary of the map are labeled EXIT.

On closer examination, you notice that someone (perhaps the poor dead park ranger) has written a real number between 0 and 1 next to each vertex and each edge. A scrawled note on the back of the map indicates that a number next to an edge is the probability of encountering a vampire along the corresponding trail, and a number next to a vertex is the probability of encountering a vampire at the corresponding landmark. (Vampires can’t stand each other’s company, so you’ll never see more than one vampire on the same trail or at the same landmark.) The note warns you that stepping off the marked trails will result in a slow and painful death.

You glance down at the corpse at your feet. Yes, his death certainly looked painful. Wait, was that a twitch? Are his teeth getting longer? After driving a tent stake through the undead ranger’s heart, you wisely decide to leave the park immediately.

Describe and analyze an efficient algorithm to find a path from your current location to an arbitrary EXIT node, such that the total *expected number* of vampires encountered along the path is as small as possible. *Be sure to account for both the vertex probabilities and the edge probabilities!*

2. Consider the following solution for the union-find problem, called *union-by-weight*. Each set leader \bar{x} stores the number of elements of its set in the field $weight(\bar{x})$. Whenever we UNION two sets, the leader of the *smaller* set becomes a new child of the leader of the *larger* set (breaking ties arbitrarily).

MAKESET(x):
$parent(x) \leftarrow x$
$weight(x) \leftarrow 1$

FIND(x):
while $x \neq parent(x)$
$x \leftarrow parent(x)$
return x

UNION(x, y)
$\bar{x} \leftarrow FIND(x)$
$\bar{y} \leftarrow FIND(y)$
if $weight(\bar{x}) > weight(\bar{y})$
$parent(\bar{y}) \leftarrow \bar{x}$
$weight(\bar{x}) \leftarrow weight(\bar{x}) + weight(\bar{y})$
else
$parent(\bar{x}) \leftarrow \bar{y}$
$weight(\bar{x}) \leftarrow weight(\bar{x}) + weight(\bar{y})$

Prove that if we use union-by-weight, the *worst-case* running time of FIND is $O(\log n)$.

3. *Prove or disprove*¹ each of the following statements.

- (a) Let G be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of G includes the lightest edge in every cycle in G .
 - (b) Let G be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of G excludes the heaviest edge in every cycle in G .
4. In Homework 2, you were asked to analyze the following algorithm to find the k th smallest element from an unsorted array. (The algorithm is presented here in iterative form, rather than the recursive form you saw in the homework, but it's exactly the same algorithm.)

```

QUICKSELECT( $A[1..n], k$ ):
     $i \leftarrow 1; j \leftarrow n$ 
    while  $i \leq j$ 
         $r \leftarrow \text{PARTITION}(A[i..j], \text{RANDOM}(i, j))$ 
        if  $r = k$ 
            return  $A[r]$ 
        else if  $r > k$ 
             $j \leftarrow r - 1$ 
        else
             $i \leftarrow r + 1$ 

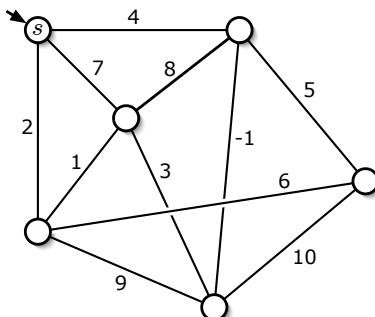
```

The algorithm relies on two subroutines. $\text{RANDOM}(i, j)$ returns an integer chosen uniformly at random from the range $[i..j]$. $\text{PARTITION}(A[i..j], p)$ partitions the subarray $A[i..j]$ using the pivot value $A[p]$ and returns the new index of the pivot value in the partitioned array.

What is the *exact* expected number of iterations of the main loop when $k = 1$? *Prove* your answer is correct. A correct $\Theta(\cdot)$ bound (with proof) is worth 7 points. You may assume that the input array $A[]$ contains n distinct integers.

5. Find the following spanning trees for the weighted graph shown below.

- (a) A breadth-first spanning tree rooted at s .
- (b) A depth-first spanning tree rooted at s .
- (c) A shortest-path tree rooted at s .
- (d) A minimum spanning tree.



You do *not* need to justify your answers; just clearly indicate the edges of each spanning tree. Yes, one of the edges has negative weight.

¹But not both! If you give us *both* a proof and a disproof for the same statement, you will get no credit, even if one of your arguments is correct.

1. A *double-Hamiltonian* circuit in an undirected graph G is a closed walk that visits every vertex in G exactly *twice*, possibly by traversing some edges more than once. *Prove* that it is NP-hard to determine whether a given undirected graph contains a double-Hamiltonian circuit.

2. Suppose you are running a web site that is visited by the same set of people every day. Each visitor claims membership in one or more *demographic groups*; for example, a visitor might describe himself as male, 31-40 years old, a resident of Illinois, an academic, a blogger, a Joss Whedon fan¹, and a Sports Racer.² Your site is supported by advertisers. Each advertiser has told you which demographic groups should see its ads and how many of its ads you must show each day. Altogether, there are n visitors, k demographic groups, and m advertisers.

Describe an efficient algorithm to determine, given all the data described in the previous paragraph, whether you can show each visitor exactly *one* ad per day, so that every advertiser has its desired number of ads displayed, and every ad is seen by someone in an appropriate demographic group.

3. Describe and analyze a data structure to support the following operations on an array $X[1..n]$ as quickly as possible. Initially, $X[i] = 0$ for all i .
 - Given an index i such that $X[i] = 0$, set $X[i]$ to 1.
 - Given an index i , return $X[i]$.
 - Given an index i , return the smallest index $j \geq i$ such that $X[j] = 0$, or report that no such index exists.

For full credit, the first two operations should run in *worst-case constant* time, and the amortized cost of the third operation should be as small as possible. [Hint: Use a modified union-find data structure.]

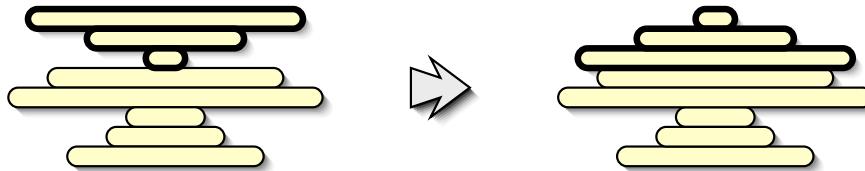
4. The next time you are at a party, one of the guests will suggest everyone play a round of Three-Way Mumbledypeg, a game of skill and dexterity that requires three teams and a knife. The official Rules of Three-Way Mumbledypeg (fixed during the Holy Roman Three-Way Mumbledypeg Council in 1625) require that (1) each team *must* have at least one person, (2) any two people on the same team *must* know each other, and (3) everyone watching the game *must* be on one of the three teams. Of course, it will be a really *fun* party; nobody will want to leave. There will be several pairs of people at the party who don't know each other. The host of the party, having heard thrilling tales of your prowess in all things algorithmic, will hand you a list of which pairs of partygoers know each other and ask you to choose the teams, while he sharpens the knife.

Either describe and analyze a polynomial time algorithm to determine whether the partygoers can be split into three legal Three-Way Mumbledypeg teams, or prove that the problem is NP-hard.

¹Har har har! Mine is an evil laugh! Now die!

²It's Ride the Fire Eagle Danger Day!

5. Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*—insert a spatula under the top k pancakes, for some integer k between 1 and n , and flip them all over.



Flipping the top three pancakes.

- (a) Describe an efficient algorithm to sort an arbitrary stack of n pancakes. *Exactly* how many flips does your algorithm perform in the worst case? (For full credit, your algorithm should perform as few flips as possible; an optimal $\Theta()$ bound is worth three points.)
 - (b) Now suppose one side of each pancake is burned. *Exactly* how many flips do you need to sort the pancakes *and* have the burned side of every pancake on the bottom? (For full credit, your algorithm should perform as few flips as possible; an optimal $\Theta()$ bound is worth three points.)
6. Describe and analyze an efficient algorithm to find the length of the longest substring that appears both forward and backward in an input string $T[1..n]$. The forward and backward substrings must not overlap. Here are several examples:

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

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

7. A *double-Eulerian* circuit in an undirected graph G is a closed walk that traverses every edge in G exactly twice. Describe and analyze a *polynomial-time* algorithm to determine whether a given undirected graph contains a double-Eulerian circuit.

CS 473G: Graduate Algorithms, Spring 2007

Homework 0

Due in class at 11:00am, Tuesday, January 30, 2007

Name:	
Net ID:	Alias:

I understand the Course Policies.

-
- Neatly print your full name, your NetID, and an alias of your choice in the boxes above, and staple this page to your solution to problem 1. **By providing an alias, you agree to let us list your grades; if you do not provide an alias, your grades will not be listed.** For privacy reasons, your alias should not resemble your name, your NetID, your university ID number, or (God forbid!) your Social Security number. Please use the same alias for every homework and exam.
 - Read the Course Policies on the course web site, and then check the box above. Among other things, this page describes what we expect in your homework solutions, as well as policies on grading standards, regrading, extra credit, and plagiarism. In particular:
 - Submit each numbered problem separately, on its own piece(s) of paper. If you need more than one page for a problem, staple just *those* pages together, but keep different problems separate. **Do not staple your entire homework together.**
 - You may use *any* source at your disposal—paper, electronic, or human—but you *must* write your answers in your own words, and you *must* cite every source that you use.
 - Algorithms or proofs containing phrases like “and so on” or “repeat this for all n ”, instead of an explicit loop, recursion, or induction, are worth zero points.
 - Answering “I don’t know” to any homework or exam problem is worth 25% partial credit.

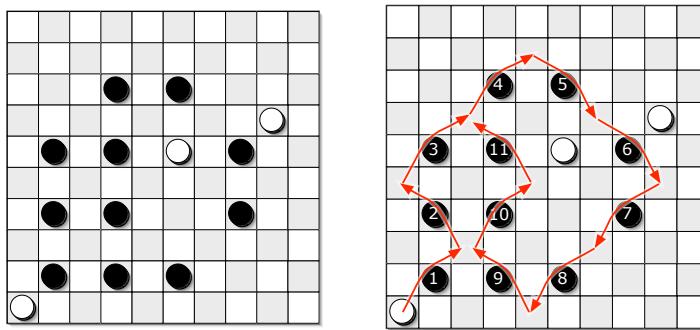
If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup.

- This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, discrete probability, graphs, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** The early chapters of Kleinberg and Tardos (or any algorithms textbook) should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks.
 - Every homework will have five problems, each worth 10 points. Stars indicate more challenging problems. Many homeworks will also include an extra-credit problem.
-

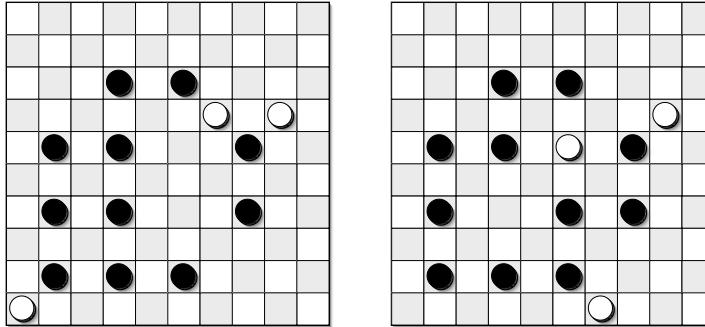
*1. Draughts/checkers is a game played on an $m \times m$ grid of squares, alternately colored light and dark. (The game is usually played on an 8×8 or 10×10 board, but the rules easily generalize to any board size.) Each dark square is occupied by at most one game piece (usually called a *checker* in the U.S.), which is either black or white; light squares are always empty. One player (“White”) moves the white pieces; the other (“Black”) moves the black pieces.

Consider the following simple version of the game, essentially American checkers or British draughts, but where every piece is a king.¹ Pieces can be moved in any of the four diagonal directions, either one or two steps at a time. On each turn, a player either moves one of her pieces one step diagonally into an empty square, or makes a series of *jumps* with one of her checkers. In a single jump, a piece moves to an empty square two steps away in any diagonal direction, but only if the intermediate square is occupied by a piece of the opposite color; this enemy piece is *captured* and immediately removed from the board. Multiple jumps are allowed in a single turn as long as they are made by the same piece. A player wins if her opponent has no pieces left on the board.

Describe an algorithm² that correctly determines whether White can capture every black piece, thereby winning the game, *in a single turn*. The input consists of the width of the board (m), a list of positions of white pieces, and a list of positions of black pieces. For full credit, your algorithm should run in $O(n)$ time, where n is the total number of pieces, but any algorithm that runs in time polynomial in n and m is worth significant partial credit.



White wins in one turn.



White cannot win in one turn from either of these positions.

[Hint: The greedy strategy—make arbitrary jumps until you get stuck—does **not** always find a winning sequence of jumps even when one exists.]

¹Most variants of draughts have ‘flying kings’, which behave very differently than what’s described here.

²Since you’ve read the Course Policies, you know what this phrase means.

2. (a) Prove that any positive integer can be written as the sum of distinct powers of 2. [*Hint: “Write the number in binary” is **not** a proof; it just restates the problem.*] For example:

$$\begin{aligned}16 + 1 &= 17 = 2^4 + 2^0 \\16 + 4 + 2 + 1 &= 23 = 2^4 + 2^2 + 2^1 + 2^0 \\32 + 8 + 1 &= 42 = 2^5 + 2^3 + 2^1\end{aligned}$$

- (b) Prove that *any* integer (positive, negative, or zero) can be written as the sum of distinct powers of -2 . For example:

$$\begin{aligned}-32 + 16 - 2 + 1 &= -17 = (-2)^5 + (-2)^4 + (-2)^1 + (-2)^0 \\64 - 32 - 8 - 2 + 1 &= 23 = (-2)^6 + (-2)^5 + (-2)^3 + (-2)^1 + (-2)^0 \\64 - 32 + 16 - 8 + 4 - 2 &= 42 = (-2)^6 + (-2)^5 + (-2)^4 + (-2)^3 + (-2)^2 + (-2)^1\end{aligned}$$

3. Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

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

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

Rosencrantz: H H T

Guildenstern: H T H H

Rosencrantz: T

Guildenstern: (no flips)

Rosencrantz: H H H T

Guildenstern: T H H T H H T H T H H H

- (a) What is the expected number of flips in one of Rosencrantz’s turns?
- (b) Suppose Rosencrantz flips k heads in a row on his turn. What is the expected number of flips in Guildenstern’s next turn?
- (c) What is the expected total number of flips (by both Rosencrantz and Guildenstern) in a single game?

Prove that your answers are correct. If you have to appeal to “intuition” or “common sense”, your answer is almost certainly wrong! You must give *exact* answers for full credit, but a correct asymptotic bound for part (b) is worth significant credit.

5. (a) [5 pts] Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Assume reasonable but nontrivial base cases. If your solution requires a particular base case, say so.

$$A(n) = 3A(n/9) + \sqrt{n}$$

$$B(n) = 4B(n - 1) - 4B(n - 2)$$

$$C(n) = \frac{\pi C(n - 1)}{\sqrt{2} C(n - 2)} \quad [\text{Hint: This is easy!}]$$

$$D(n) = \max_{n/4 < k < 3n/4} (D(k) + D(n - k) + n)$$

$$E(n) = 2E(n/2) + 4E(n/3) + 2E(n/6) + n^2$$

Do not turn in proofs—just a list of five functions—but you should do them anyway, just for practice. [*Hint: On the course web page, you can find a handout describing several techniques for solving recurrences.*]

- (b) [5 pts] Sort the functions in the box from asymptotically smallest to asymptotically largest, indicating ties if there are any. **Do not turn in proofs**—just a sorted list of 16 functions—but you should do them anyway, just for practice.

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

n	$\lg n$	\sqrt{n}	3^n
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$3\sqrt{n}$	$\sqrt{3^n}$
$3^{\lg n}$	$\lg(3^n)$	$3^{\lg \sqrt{n}}$	$3^{\sqrt{\lg n}}$
$\sqrt{3^{\lg n}}$	$\lg(3^{\sqrt{n}})$	$\lg \sqrt{3^n}$	$\sqrt{\lg(3^n)}$

Recall that $\lg n = \log_2 n$.

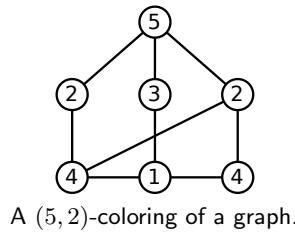
CS 473G: Graduate Algorithms, Spring 2007

Homework 1

Due February 6, 2007

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

1. Jeff tries to make his students happy. At the beginning of class, he passes out a questionnaire to students which lists a number of possible course policies in areas where he is flexible. Every student is asked to respond to each possible course policy with one of “strongly favor”, “mostly neutral”, or “strongly oppose”. Each student may respond with “strongly favor” or “strongly oppose” to at most five questions. Because Jeff’s students are very understanding, each student is happy if he or she prevails in just one of his or her strong policy preferences. Either describe a polynomial time algorithm for setting course policy to maximize the number of happy students or show that the problem is NP-hard.
2. Consider a variant 3SAT' of 3SAT which asks, given a formula ϕ in conjunctive normal form in which each clause contains at most 3 literals and each variable appears in at most 3 clauses, is ϕ satisfiable? Prove that 3SAT' is NP-complete.
3. For each problem below, either describe a polynomial-time algorithm to solve the problem or prove that the problem is NP-complete.
 - (a) A *double-Eulerian* circuit in an undirected graph G is a closed walk that traverses every edge in G exactly twice. Given a graph G , does G have a double-Eulerian circuit?
 - (b) A *double-Hamiltonian* circuit in an undirected graph G is a closed walk that visits every vertex in G exactly twice. Given a graph G , does G have a double-Hamiltonian circuit?
4. Suppose you have access to a magic black box; if you give it a graph G as input, the black box will tell you, in constant time, if there is a proper 3-coloring of G . Describe a polynomial time algorithm which, given a graph G that is 3-colorable, uses the black box to compute a 3-coloring of G .
5. Let C_5 be the graph which is a cycle on five vertices. A $(5, 2)$ -coloring of a graph G is a function $f : V(G) \rightarrow \{1, 2, 3, 4, 5\}$ such that every pair $\{u, v\}$ of adjacent vertices in G is mapped to a pair $\{f(u), f(v)\}$ of vertices in C_5 which are at distance two from each other.



Using a reduction from 5COLOR, prove that the problem of deciding whether a given graph G has a $(5, 2)$ -coloring is NP-complete.

CS 473G: Graduate Algorithms, Spring 2007

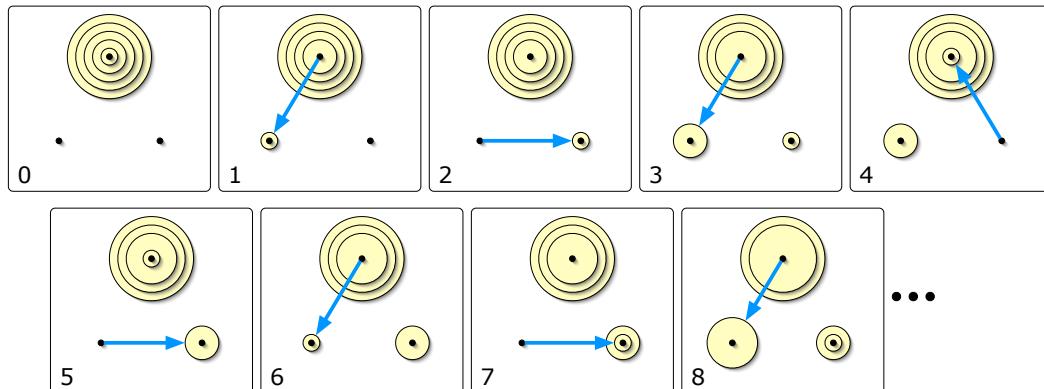
Homework 2

Due Tuesday, February 20, 2007

Remember to submit **separate, individually stapled** solutions to each problem.

As a general rule, a complete full-credit solution to any homework problem should fit into two typeset pages (or five hand-written pages). If your solution is significantly longer than this, you may be including too much detail.

1. Consider a restricted variant of the Tower of Hanoi puzzle, where the three needles are arranged in a triangle, and you are required to move each disk *counterclockwise*. Describe an algorithm to move a stack of n disks from one needle to another. *Exactly* how many moves does your algorithm perform? To receive full credit, your algorithm must perform the minimum possible number of moves. [Hint: Your answer will depend on whether you are moving the stack clockwise or counterclockwise.]



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

- *2. You find yourself working for The Negation Company (“We Contradict Everything...Not!”), the world’s largest producer of multi-bit Boolean inverters. Thanks to a recent mining discovery, the market prices for amphigen and opoterium, the key elements used in AND and OR gates, have plummeted to almost nothing. Unfortunately, the market price of inverton, the essential element required to build NOT gates, has recently risen sharply as natural supplies are almost exhausted. Your boss is counting on you to radically redesign the company’s only product in response to these radically new market prices.

Design a Boolean circuit that inverts $n = 2^k - 1$ bits, using only k NOT gates but *any* number of AND and OR gates. The input to your circuit consists of n bits x_1, x_2, \dots, x_n , and the output consists of n bits y_1, y_2, \dots, y_n , where each output bit y_i is the inverse of the corresponding input bit x_i . [Hint: Solve the case $k = 2$ first.]

3. (a) Let $X[1..m]$ and $Y[1..n]$ be two arbitrary arrays. A *common supersequence* of X and Y is another sequence that contains both X and Y as subsequences. Give a simple recursive definition for the function $scs(X, Y)$, which gives the length of the *shortest* common supersequence of X and Y .
(b) Call a sequence $X[1..n]$ *oscillating* if $X[i] < X[i+1]$ for all even i , and $X[i] > X[i+1]$ for all odd i . Give a simple recursive definition for the function $los(X)$, which gives the length of the longest oscillating subsequence of an arbitrary array X of integers.
(c) Call a sequence $X[1..n]$ of integers *accelerating* if $2 \cdot X[i] < X[i-1] + X[i+1]$ for all i . Give a simple recursive definition for the function $lxs(X)$, which gives the length of the longest accelerating subsequence of an arbitrary array X of integers.

Each recursive definition should translate directly into a recursive algorithm, *but you do not need to analyze these algorithms*. We are looking for correctness and *simplicity*, not algorithmic efficiency. Not yet, anyway.

4. Describe an algorithm to solve 3SAT in time $O(\phi^n \text{ poly}(n))$, where $\phi = (1 + \sqrt{5})/2 \approx 1.618034$.
[Hint: Prove that in each recursive call, either you have just eliminated a pure literal, or the formula has a clause with at most two literals. What recurrence leads to this running time?]
5. (a) Describe an algorithm that determines whether a given set of n integers contains two distinct elements that sum to zero, in $O(n \log n)$ time.
(b) Describe an algorithm that determines whether a given set of n integers contains three distinct elements that sum to zero, in $O(n^2)$ time.
(c) Now suppose the input set X contains n integers between $-10000n$ and $10000n$. Describe an algorithm that determines whether X contains three **distinct** elements that sum to zero, in $O(n \log n)$ time.

For example, if the input set is $\{-10, -9, -7, -3, 1, 3, 5, 11\}$, your algorithm for part (a) should return TRUE, because $(-3) + 3 = 0$, and your algorithms for parts (b) and (c) should return FALSE, even though $(-10) + 5 + 5 = 0$.

CS 473G: Graduate Algorithms, Spring 2007

Homework 3

Due Friday, March 9, 2007

Remember to submit **separate, individually stapled** solutions to each problem.

As a general rule, a complete, full-credit solution to any homework problem should fit into two typeset pages (or five hand-written pages). If your solution is significantly longer than this, you may be including too much detail.

1. (a) Let $X[1..m]$ and $Y[1..n]$ be two arbitrary arrays. A *common supersequence* of X and Y is another sequence that contains both X and Y as subsequences. Describe and analyze an efficient algorithm to compute the function $scs(X, Y)$, which gives the length of the shortest common supersequence of X and Y .
- (b) Call a sequence $X[1..n]$ *oscillating* if $X[i] < X[i+1]$ for all even i , and $X[i] > X[i+1]$ for all odd i . Describe and analyze an efficient algorithm to compute the function $los(X)$, which gives the length of the longest oscillating subsequence of an arbitrary array X of integers.
- (c) Call a sequence $X[1..n]$ of integers *accelerating* if $2 \cdot X[i] < X[i-1] + X[i+1]$ for all i . Describe and analyze an efficient algorithm to compute the function $lcs(X)$, which gives the length of the longest accelerating subsequence of an arbitrary array X of integers.

[Hint: Use the recurrences you found in Homework 2. You do not need to prove again that these recurrences are correct.]

2. Describe and analyze an algorithm to solve the traveling salesman problem in $O(2^n \text{poly}(n))$ time. Given an undirected n -vertex graph G with weighted edges, your algorithm should return the weight of the lightest Hamiltonian cycle in G (or ∞ if G has no Hamiltonian cycles).
3. Let G be an arbitrary undirected graph. A set of cycles $\{c_1, \dots, c_k\}$ in G is *redundant* if it is non-empty and every edge in G appears in an even number of c_i 's. A set of cycles is *independent* if it contains no redundant subsets. (In particular, the empty set is independent.) A maximal independent set of cycles is called a *cycle basis* for G .

- (a) Let C be any cycle basis for G . Prove that for any cycle γ in G **that is not an element of C** , there is a subset $A \subseteq C$ such that $A \cup \{\gamma\}$ is redundant. In other words, prove that γ is the ‘exclusive or’ of some subset of basis cycles.

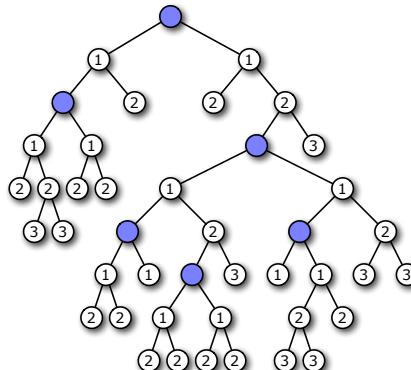
Solution: The claim follows directly from the definitions. A cycle basis is a *maximal* independent set, so if C is a cycle basis, then for any cycle $\gamma \notin C$, the larger set $C \cup \{\gamma\}$ cannot be an independent set, so it must contain a redundant subset. On the other hand, if C is a basis, then C is independent, so C contains no redundant subsets. Thus, $C \cup \{\gamma\}$ must have a redundant subset B that contains γ . Let $A = B \setminus \{\gamma\}$. ■

- (b) Prove that the set of independent cycle sets form a matroid.
- (c) Now suppose each edge of G has a weight. Define the weight of a cycle to be the total weight of its edges, and the weight of a set of cycles to be the total weight of all cycles in the set. (Thus, each edge is counted once for every cycle in which it appears.) Describe and analyze an efficient algorithm to compute the minimum-weight cycle basis of G .
4. Let T be a rooted binary tree with n vertices, and let $k \leq n$ be a positive integer. We would like to mark k vertices in T so that every vertex has a nearby marked ancestor. More formally, we define the *clustering cost* of a clustering of any subset K of vertices as

$$\text{cost}(K) = \max_v \text{cost}(v, K),$$

where the maximum is taken over all vertices v in the tree, and

$$\text{cost}(v, K) = \begin{cases} 0 & \text{if } v \in K \\ \infty & \text{if } v \text{ is the root of } T \text{ and } v \notin K \\ 1 + \text{cost}(\text{parent}(v)) & \text{otherwise} \end{cases}$$

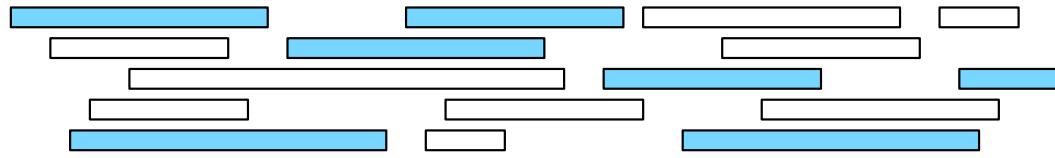


A subset of 5 vertices with clustering cost 3

Describe and analyze a dynamic-programming algorithm to compute the minimum clustering cost of any subset of k vertices in T . For full credit, your algorithm should run in $O(n^2k^2)$ time.

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

Describe and analyze an algorithm to compute the smallest tiling path of X as quickly as possible. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X . If you use a greedy algorithm, you must prove that it is correct.



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

CS 473G: Graduate Algorithms, Spring 2007

Homework 4

Due March 29, 2007

Please remember to submit **separate, individually stapled** solutions to each problem.

1. Given a graph G with edge weights and an integer k , suppose we wish to partition the vertices of G into k subsets S_1, S_2, \dots, S_k so that the sum of the weights of the edges that cross the partition (*i.e.*, have endpoints in different subsets) is as large as possible.
 - (a) Describe an efficient $(1 - 1/k)$ -approximation algorithm for this problem.
 - (b) Now suppose we wish to minimize the sum of the weights of edges that do *not* cross the partition. What approximation ratio does your algorithm from part (a) achieve for the new problem? Justify your answer.
2. In class, we saw a $(3/2)$ -approximation algorithm for the metric traveling salesman problem. Here, we consider computing minimum cost Hamiltonian *paths*. Our input consists of a graph G whose edges have weights that satisfy the triangle inequality. Depending upon the problem, we are also given zero, one, or two endpoints.
 - (a) If our input includes zero endpoints, describe a $(3/2)$ -approximation to the problem of computing a minimum cost Hamiltonian path.
 - (b) If our input includes one endpoint u , describe a $(3/2)$ -approximation to the problem of computing a minimum cost Hamiltonian path that starts at u .
 - (c) If our input includes two endpoints u and v , describe a $(5/3)$ -approximation to the problem of computing a minimum cost Hamiltonian path that starts at u and ends at v .
3. Consider the greedy algorithm for metric TSP: start at an arbitrary vertex u , and at each step, travel to the closest unvisited vertex.
 - (a) Show that the greedy algorithm for metric TSP is an $O(\log n)$ -approximation, where n is the number of vertices. [*Hint: Argue that the k th least expensive edge in the tour output by the greedy algorithm has weight at most $\text{OPT}/(n - k + 1)$; try $k = 1$ and $k = 2$ first.*]
 - *(b) **[Extra Credit]** Show that the greedy algorithm for metric TSP is no better than an $O(\log n)$ -approximation.
4. In class, we saw that the greedy algorithm gives an $O(\log n)$ -approximation for vertex cover. Show that our analysis of the greedy algorithm is asymptotically tight by describing, for any positive integer n , an n -vertex graph for which the greedy algorithm produces a vertex cover of size $\Omega(\log n) \cdot \text{OPT}$.

5. Recall the minimum makespan scheduling problem: Given an array $T[1..n]$ of processing times for n jobs, we wish to schedule the jobs on m machines to minimize the time at which the last job terminates. In class, we proved that the greedy scheduling algorithm has an approximation ratio of at most 2.

- (a) Prove that for any set of jobs, the makespan of the greedy assignment is at most $(2 - 1/m)$ times the makespan of the optimal assignment.
- (b) Describe a set of jobs such that the makespan of the greedy assignment is exactly $(2 - 1/m)$ times the makespan of the optimal assignment.
- (c) Describe an efficient algorithm to solve the minimum makespan scheduling problem *exactly* if every processing time $T[i]$ is a power of two.

CS 473G: Graduate Algorithms, Spring 2007

Homework 5

Due Thursday, April 17, 2007

Please remember to submit **separate, individually stapled** solutions to each problem.

Unless a problem specifically states otherwise, you can assume the function $\text{RANDOM}(k)$, which returns an integer chosen independently and uniformly at random from the set $\{1, 2, \dots, k\}$, in $O(1)$ time. For example, to perform a fair coin flip, you would call $\text{RANDOM}(2)$.

1. Suppose we want to write an efficient function $\text{RANDOMPERMUTATION}(n)$ that returns a permutation of the integers $\langle 1, \dots, n \rangle$ chosen uniformly at random.

- (a) What is the expected running time of the following RANDOMPERMUTATION algorithm?

```
RANDOMPERMUTATION( $n$ ):  
    for  $i \leftarrow 1$  to  $n$   
         $\pi[i] \leftarrow \text{EMPTY}$   
    for  $i \leftarrow 1$  to  $n$   
         $j \leftarrow \text{RANDOM}(n)$   
        while ( $\pi[j] \neq \text{EMPTY}$ )  
             $j \leftarrow \text{RANDOM}(n)$   
         $\pi[j] \leftarrow i$   
    return  $\pi$ 
```

- (b) Consider the following partial implementation of RANDOMPERMUTATION .

```
RANDOMPERMUTATION( $n$ ):  
    for  $i \leftarrow 1$  to  $n$   
         $A[i] \leftarrow \text{RANDOM}(n)$   
     $\pi \leftarrow \text{SOMEFUNCTION}(A)$   
    return  $\pi$ 
```

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

- (c) Describe and analyze an RANDOMPERMUTATION algorithm whose expected worst-case running time is $O(n)$.
- *(d) [Extra Credit] Describe and analyze an RANDOMPERMUTATION algorithm that uses only fair coin flips; that is, your algorithm can't call $\text{RANDOM}(k)$ with $k > 2$. Your algorithm should run in $O(n \log n)$ time with high probability.

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

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

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

```

MELD( $Q_1, Q_2$ ):
  if  $Q_1$  is empty, return  $Q_2$ 
  if  $Q_2$  is empty, return  $Q_1$ 
  if  $key(Q_1) > key(Q_2)$ 
    swap  $Q_1 \leftrightarrow Q_2$ 
  with probability  $1/2$ 
     $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
  else
     $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
  return  $Q_1$ 
```

- (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 $\text{MELD}(Q_1, Q_2)$ 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 $\text{MELD}(Q_1, Q_2)$ runs in $O(\log n)$ time with high probability.
 - (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)
3. Prove that GUESSMINCUT returns the *second* smallest cut in its input graph with probability $\Omega(1/n^2)$. (The second smallest cut could be significantly larger than the minimum cut.)

4. A *heater* is a sort of dual treap, in which the priorities of the nodes are given by the user, but their search keys are random (specifically, independently and uniformly distributed in the unit interval $[0, 1]$).

- (a) Prove that for any r , the node with the r th smallest priority has expected depth $O(\log r)$.
- (b) Prove that an n -node heater has depth $O(\log n)$ with high probability.
- (c) Describe algorithms to perform the operations INSERT and DELETEMIN in a heater. What are the expected worst-case running times of your algorithms?

You may assume all priorities and keys are distinct. [Hint: Cite the relevant parts (but only the relevant parts!) of the treap analysis instead of repeating them.]

5. Let n be an arbitrary positive integer. Describe a set \mathcal{T} of binary search trees with the following properties:

- Every tree in \mathcal{T} has n nodes, which store the search keys $1, 2, 3, \dots, n$.
- For any integer k , if we choose a tree uniformly at random from \mathcal{T} , the expected depth of node k in that tree is $O(\log n)$.
- Every tree in \mathcal{T} has depth $\Omega(\sqrt{n})$.

(This is why we had to prove via Chernoff bounds that the maximum depth of an n -node treap is $O(\log n)$ with high probability.)

*6. [Extra Credit] Recall that F_k denotes the k th Fibonacci number: $F_0 = 0$, $F_1 = 1$, and $F_k = F_{k-1} + F_{k-2}$ for all $k \geq 2$. Suppose we are building a hash table of size $m = F_k$ using the hash function

$$h(x) = (F_{k-1} \cdot x) \bmod F_k$$

Prove that if the consecutive integers $0, 1, 2, \dots, F_k - 1$ are inserted in order into an initially empty table, each integer is hashed into one of the largest contiguous empty intervals in the table. Among other things, this implies that there are no collisions.

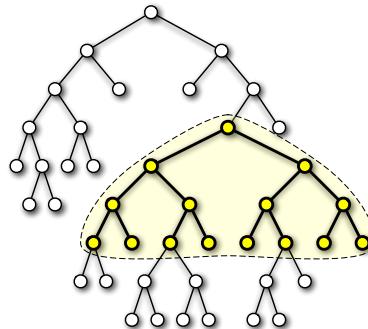
For example, when $m = 13$, the hash table is filled as follows.

0												
0									1			
0			2					1				
0			2					1			3	
0			2			4		1			3	
0	5		2			4		1			3	
0	5		2			4		1	6		3	
0	5		2	7		4		1	6		3	8
0	5		2	7		4	9	1	6		3	8
0	5	10	2	7		4	9	1	6		3	8
0	5	10	2	7	12	4	9	1	6	11	3	8
0	5	10	2	7	12	4	9	1	6	11	3	8

You have 90 minutes to answer four of these questions.
Write your answers in the separate answer booklet.
 You may take the question sheet with you when you leave.

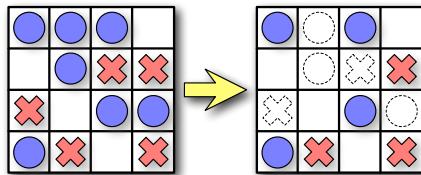
1. Recall that a binary tree is *complete* if every internal node has two children and every leaf has the same depth. An *internal subtree* of a binary tree is a connected subgraph, consisting of a node and some (possibly all or none) of its descendants.

Describe and analyze an algorithm that computes the depth of the *largest complete internal subtree* of a given n -node binary tree. For full credit, your algorithm should run in $O(n)$ time.

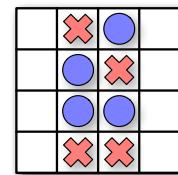


The largest complete internal subtree in this binary tree has depth 3.

2. Consider the following solitaire game. The puzzle consists of an $n \times m$ grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

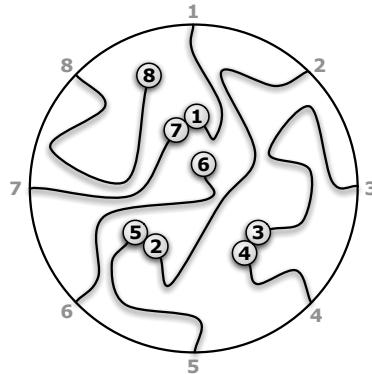
Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

3. Suppose you are given two sorted arrays $A[1..n]$ and $B[1..n]$ and an integer k . Describe and analyze an algorithm to find the k th largest element in the union of A and B in $O(\log n)$ time. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 21], \quad B[1..8] = [2, 4, 5, 8, 14, 17, 19, 20], \quad k = 10,$$

your algorithm should return 13. You can assume that the arrays contain no duplicates. [Hint: What can you learn from comparing one element of A to one element of B ?]

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



The end of a typical Antarctic SLUG race. Snails 6 and 8 never find mates.

The organizers must pay $M[3, 4] + M[2, 5] + M[1, 7]$.

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

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

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

- SUBSETSUM: Given a set X of positive integers and an integer t , determine whether there is a subset of X whose elements sum to t .
- PARTITION: Given a set X of positive integers, determine whether X can be partitioned into two subsets whose elements sum to the same value.
 - (a) Describe a polynomial-time reduction from SUBSETSUM to PARTITION.
 - (b) Describe a polynomial-time reduction from PARTITION to SUBSETSUM.

Don't forget to **prove** that your reductions are correct.

You have 120 minutes to answer four of these questions.
Write your answers in the separate answer booklet.
 You may take the question sheet with you when you leave.

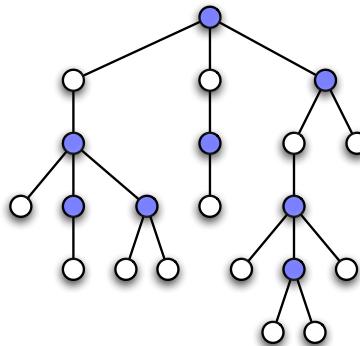
1. Consider the following algorithm for finding the smallest element in an unsorted array:

```
RANDOMMIN( $A[1..n]$ ):
   $min \leftarrow \infty$ 
  for  $i \leftarrow 1$  to  $n$  in random order
    if  $A[i] < min$ 
       $min \leftarrow A[i]$  (*)
  return  $min$ 
```

- (a) [1 pt] In the worst case, how many times does RANDOMMIN execute line (*)?
- (b) [3 pts] What is the probability that line (*) is executed during the last iteration of the for loop?
- (c) [6 pts] What is the *exact* expected number of executions of line (*)? (A correct $\Theta()$ bound is worth 4 points.)

2. Describe and analyze an efficient algorithm to find the size of the smallest vertex cover of a given tree. That is, given a tree T , your algorithm should find the size of the smallest subset C of the vertices, such that every edge in T has at least one endpoint in C .

The following hint may be helpful. Suppose C is a vertex cover that contains a leaf ℓ . If we remove ℓ from the cover and insert its parent, we get another vertex cover of the same size as C . Thus, there is a minimum vertex cover that includes none of the leaves of T (except when the tree has only one or two vertices).

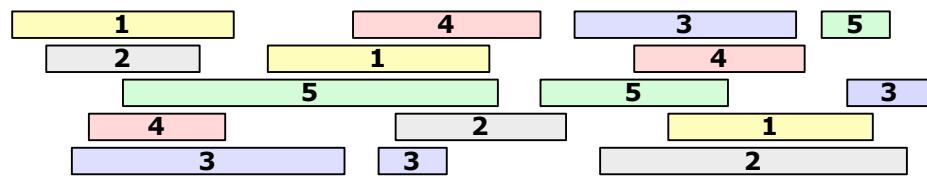


A tree whose smallest vertex cover has size 8.

3. A *dominating set* for a graph G is a subset D of the vertices, such that every vertex in G is either in D or has a neighbor in D . The MINDOMINATINGSET problem asks for the size of the smallest dominating set for a given graph.

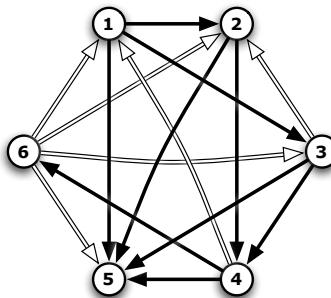
Recall the MINSETCOVER problem from lecture. The input consists of a *ground set* X and a collection of subsets $S_1, S_2, \dots, S_k \subseteq X$. The problem is to find the minimum number of subsets S_i that completely cover X . This problem is NP-hard, because it is a generalization of the vertex cover problem.

- (a) [7 pts] Describe a polynomial-time reduction from MINDOMINATINGSET to MINSET-COVER.
 - (b) [3 pts] Describe a polynomial-time $O(\log n)$ -approximation algorithm for MINDOMINATINGSET.
[Hint: There is a two-line solution.]
4. Let X be a set of n intervals on the real line. A *proper coloring* of X assigns a color to each interval, so that any two overlapping intervals are assigned different colors. Describe and analyze an efficient algorithm to compute the minimum number of colors needed to properly color X . Assume that your input consists of two arrays $L[1..n]$ and $R[1..n]$, where $L[i]$ and $R[i]$ are the left and right endpoints of the i th interval. As usual, if you use a greedy algorithm, you must prove that it is correct.



A proper coloring of a set of intervals using five colors.

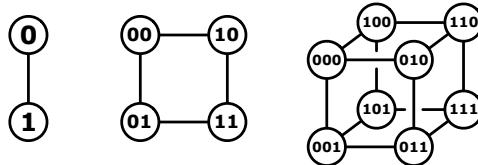
5. The *linear arrangement problem* asks, given an n -vertex directed graph as input, for an ordering v_1, v_2, \dots, v_n of the vertices that maximizes the number of *forward edges*: directed edges $v_i \rightarrow v_j$ such that $i < j$. Describe and analyze an efficient 2-approximation algorithm for this problem.



A directed graph with six vertices with nine forward edges (black) and six backward edges (white)

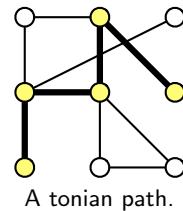
You have 180 minutes to answer six of these questions.
Write your answers in the separate answer booklet.

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



The 1-dimensional, 2-dimensional, and 3-dimensional hypercubes.

- (a) [8 pts] Recall that a Hamiltonian cycle is a closed walk that visits each vertex in a graph exactly once. **Prove** that for all $d \geq 2$, the d -dimensional hypercube has a Hamiltonian cycle.
- (b) [2 pts] Recall that an Eulerian circuit is a closed walk that traverses each edge in a graph exactly once. Which hypercubes have an Eulerian circuit? [Hint: This is very easy.]
2. The University of Southern North Dakota at Hoople has hired you to write an algorithm to schedule their final exams. Each semester, USNDH offers n different classes. There are r different rooms on campus and t different time slots in which exams can be offered. You are given two arrays $E[1..n]$ and $S[1..r]$, where $E[i]$ is the number of students enrolled in the i th class, and $S[j]$ is the number of seats in the j th room. At most one final exam can be held in each room during each time slot. Class i can hold its final exam in room j only if $E[i] < S[j]$. Describe and analyze an efficient algorithm to assign a room and a time slot to each class (or report correctly that no such assignment is possible).
3. What is the *exact* expected number of leaves in an n -node treap? (The answer is obviously at most n , so no partial credit for writing " $O(n)$ ".) [Hint: What is the probability that the node with the k th largest key is a leaf?]
4. A *tonian path* in a graph G is a simple path in G that visits more than half of the vertices of G . (Intuitively, a tonian path is “most of a Hamiltonian path”.) **Prove** that it is NP-hard to determine whether or not a given graph contains a tonian path.



A tonian path.

5. A *palindrome* is a string that reads the same forwards and backwards, like x, pop, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. Any string can be broken into sequence of palindromes. For example, the string bubbasesabana ('Bubba sees a banana.') can be broken into palindromes in several different ways; for example,

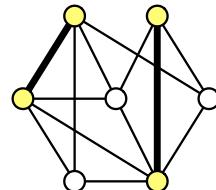
$$\begin{aligned} &\text{bub} + \text{baseesab} + \text{anana} \\ &\text{b} + \text{u} + \text{bb} + \text{a} + \text{sees} + \text{aba} + \text{nan} + \text{a} \\ &\text{b} + \text{u} + \text{bb} + \text{a} + \text{sees} + \text{a} + \text{b} + \text{anana} \\ &\text{b} + \text{u} + \text{b} + \text{b}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

Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string bubbasesabana, your algorithm would return the integer 3.

6. Consider the following modification of the 2-approximation algorithm for minimum vertex cover that we saw in class. The only real change is that we compute a set of edges instead of a set of vertices.

```
APPROXMINMAXMATCHING( $G$ ):
   $M \leftarrow \emptyset$ 
  while  $G$  has at least one edge
     $(u, v) \leftarrow$  any edge in  $G$ 
     $G \leftarrow G \setminus \{u, v\}$ 
     $M \leftarrow M \cup \{(u, v)\}$ 
  return  $M$ 
```

- (a) [2 pts] **Prove** that the output graph M is a *matching*—no pair of edges in M share a common vertex.
- (b) [2 pts] **Prove** that M is a *maximal matching*— M is not a proper subgraph of another matching in G .
- (c) [6 pts] **Prove** that M contains at most twice as many edges as the *smallest maximal matching* in G .



The smallest maximal matching in a graph.

7. Recall that in the standard maximum-flow problem, the flow through an edge is limited by the capacity of that edge, but there is no limit on how much flow can pass through a vertex. Suppose each vertex v in our input graph has a capacity $c(v)$ that limits the total flow through v , in addition to the usual edge capacities. Describe and analyze an efficient algorithm to compute the maximum (s, t) -flow with these additional constraints. [Hint: Reduce to the standard max-flow problem.]

CS 573: Graduate Algorithms, Fall 2008

Homework 0

Due in class at 12:30pm, Wednesday, September 3, 2008

Name:	
Net ID:	Alias:

I understand the course policies.

-
- Each student must submit their own solutions for this homework. For all future homeworks, groups of up to three students may submit a single, common solution.
 - Neatly print your full name, your NetID, and an alias of your choice in the boxes above, and staple this page to the front of your homework solutions. We will list homework and exam grades on the course web site by alias.

Federal privacy law and university policy forbid us from publishing your grades, even anonymously, without your explicit written permission. *By providing an alias, you grant us permission to list your grades on the course web site. If you do not provide an alias, your grades will not be listed.* For privacy reasons, your alias should not resemble your name, your NetID, your university ID number, or (God forbid) your Social Security number.

- Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. Once you understand the policies, please check the box at the top of this page. In particular:
 - You may use any source at your disposal—paper, electronic, or human—but you **must** write your solutions in your own words, and you **must** cite every source that you use.
 - Unless explicitly stated otherwise, **every** homework problem requires a proof.
 - Answering “I don’t know” to any homework or exam problem is worth 25% partial credit.
 - Algorithms or proofs containing phrases like “and so on” or “repeat this for all n ”, instead of an explicit loop, recursion, or induction, will receive 0 points.
 - This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, discrete probability, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** The early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
-

1. (a) [5 pts] Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Assume reasonable but nontrivial base cases. If your solution requires a particular base case, say so.

$$A(n) = 4A(n/8) + \sqrt{n}$$

$$B(n) = B(n/3) + 2B(n/4) + B(n/6) + n$$

$$C(n) = 6C(n-1) - 9C(n-2)$$

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

$$E(n) = (E(\sqrt{n}))^2 \cdot n$$

- (b) [5 pts] Sort the functions in the box from asymptotically smallest to asymptotically largest, indicating ties if there are any. **Do not turn in proofs**—just a sorted list of 16 functions—but you should do them anyway, just for practice. We use the notation $\lg n = \log_2 n$.

n	$\lg n$	\sqrt{n}	3^n
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$3^{\sqrt{n}}$	$\sqrt{3^n}$
$3^{\lg n}$	$\lg(3^n)$	$3^{\lg \sqrt{n}}$	$3^{\sqrt{\lg n}}$
$\sqrt{3^{\lg n}}$	$\lg(3^{\sqrt{n}})$	$\lg \sqrt{3^n}$	$\sqrt{\lg(3^n)}$

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

- $\text{RANGE TOP}(a, z)$: return the highest-priority record whose key is between a and z .

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

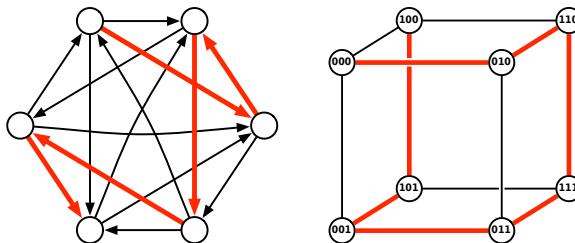
$$(3, 1), (4, 9), (9, 2), (6, 3), (5, 8), (7, 5), (1, 10), (0, 7),$$

then $\text{RANGE TOP}(2, 8)$ would return the record with key 4 and priority 9 (the second in the list).

Analyze both the size of your data structure and the running time of your RANGE TOP algorithm. For full credit, your space and time bounds must both be as small as possible. You may assume that no two records have equal keys or equal priorities, and that no record has a or z as its key. [*Hint: How would you compute the number of keys between a and z ? How would you solve the problem if you knew that a is always $-\infty$?*]

3. A *Hamiltonian path* in G is a path that visits every vertex of G exactly once. In this problem, you are asked to prove that two classes of graphs always contain a Hamiltonian path.

- (a) [5 pts] A *tournament* is a directed graph with exactly one edge between each pair of vertices. (Think of the nodes in a round-robin tournament, where edges represent games, and each edge points from the loser to the winner.) Prove that every tournament contains a *directed* Hamiltonian path.
- (b) [5 pts] Let d be an arbitrary non-negative integer. The d -dimensional *hypercube* is the graph defined as follows. There are 2^d vertices, each labeled with a different string of d bits. Two vertices are joined by an edge if and only if their labels differ in exactly one bit. Prove that the d -dimensional hypercube contains a Hamiltonian path.



Hamiltonian paths in a 6-node tournament and a 3-dimensional hypercube.

4. Penn and Teller agree to play the following game. Penn shuffles a standard deck¹ 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 ($3\clubsuit$), at which point the remaining undrawn cards instantly burst into flames.

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

- (a) What is the expected number of cards that Teller draws?
- (b) What is the expected *maximum* value among the cards Teller gives to Penn?
- (c) What is the expected *minimum* value among the cards Teller gives to Penn?
- (d) What is the expected number of cards that Teller gives to Penn?

Full credit will be given only for *exact* answers (with correct proofs, of course). [Hint: Let $13 = n$.]

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

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

5. (a) The **Fibonacci numbers** F_n are defined by the recurrence $F_n = F_{n-1} + F_{n-2}$, with base cases $F_0 = 0$ and $F_1 = 1$. Here are the first several Fibonacci numbers:

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
0	1	1	2	3	5	8	13	21	34	55

Prove that any non-negative integer can be written as the sum of distinct, non-consecutive Fibonacci numbers. That is, if the Fibonacci number F_i appears in the sum, it appears exactly once, and its neighbors F_{i-1} and F_{i+1} do not appear at all. For example:

$$17 = F_7 + F_4 + F_2, \quad 42 = F_9 + F_6, \quad 54 = F_9 + F_7 + F_5 + F_3 + F_1.$$

- (b) The Fibonacci sequence can be extended backward to negative indices by rearranging the defining recurrence: $F_n = F_{n+2} - F_{n+1}$. Here are the first several negative-index Fibonacci numbers:

F_{-10}	F_{-9}	F_{-8}	F_{-7}	F_{-6}	F_{-5}	F_{-4}	F_{-3}	F_{-2}	F_{-1}
-55	34	-21	13	-8	5	-3	2	-1	1

Prove that $F_{-n} = -F_n$ if and only if n is even.

- (c) Prove that *any* integer—positive, negative, or zero—can be written as the sum of distinct, non-consecutive Fibonacci numbers *with negative indices*. For example:

$$17 = F_{-7} + F_{-5} + F_{-2}, \quad -42 = F_{-10} + F_{-7}, \quad 54 = F_{-9} + F_{-7} + F_{-5} + F_{-3} + F_{-1}.$$

[Hint: Zero is both non-negative and even. Don't use weak induction!]

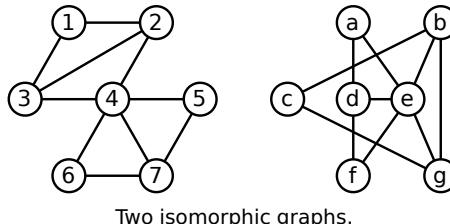
CS 573: Graduate Algorithms, Fall 2008

Homework 1

Due at 11:59:59pm, Wednesday, September 17, 2008

For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and alias (if any) of every group member on the first page of your submission.

1. Two graphs are said to be **isomorphic** if one can be transformed into the other just by relabeling the vertices. For example, the graphs shown below are isomorphic; the left graph can be transformed into the right graph by the relabeling $(1, 2, 3, 4, 5, 6, 7) \mapsto (c, g, b, e, a, f, d)$.



Consider the following related decision problems:

- **GRAPHISOMORPHISM:** Given two graphs G and H , determine whether G and H are isomorphic.
 - **EVENGRAPHISOMORPHISM:** Given two graphs G and H , such that every vertex in G and H has even degree, determine whether G and H are isomorphic.
 - **SUBGRAPHISOMORPHISM:** Given two graphs G and H , determine whether G is isomorphic to a subgraph of H .
- (a) Describe a polynomial-time reduction from **EVENGRAPHISOMORPHISM** to **GRAPHISOMORPHISM**.
 - (b) Describe a polynomial-time reduction from **GRAPHISOMORPHISM** to **EVENGRAPHISOMORPHISM**.
 - (c) Describe a polynomial-time reduction from **GRAPHISOMORPHISM** to **SUBGRAPHISOMORPHISM**.
 - (d) Prove that **SUBGRAPHISOMORPHISM** is NP-complete.
 - (e) What can you conclude about the NP-hardness of **GRAPHISOMORPHISM**? Justify your answer.

[Hint: These are all easy!]

2. (a) A *tonian path* in a graph G is a path that goes through at least half of the vertices of G . Show that determining whether a graph has a tonian path is NP-complete.
(b) A *tonian cycle* in a graph G is a cycle that goes through at least half of the vertices of G . Show that determining whether a graph has a tonian cycle is NP-complete. [Hint: Use part (a).]
3. The following variant of 3SAT is called either EXACT3SAT or 1IN3SAT, depending on who you ask.

Given a boolean formula in conjunctive normal form with 3 literals per clause, is there an assignment that makes **exactly one** literal in each clause TRUE?

Prove that this problem is NP-complete.

4. Suppose you are given a magic black box that can solve the MAXCLIQUE problem *in polynomial time*. That is, given an arbitrary graph G as input, the magic black box computes the number of vertices in the largest complete subgraph of G . Describe and analyze a **polynomial-time** algorithm that computes, given an arbitrary graph G , a complete subgraph of G of maximum size, using this magic black box as a subroutine.
 5. A boolean formula in *exclusive-or conjunctive normal form* (XCNF) is a conjunction (AND) of several *clauses*, each of which is the *exclusive-or* of several literals. The XCNF-SAT problem asks whether a given XCNF boolean formula is satisfiable. Either describe a polynomial-time algorithm for XCNF-SAT or prove that it is NP-complete.
- *6. [Extra credit] Describe and analyze an algorithm to solve 3SAT in $O(\phi^n \text{poly}(n))$ time, where $\phi = (1 + \sqrt{5})/2 \approx 1.618034$. [Hint: Prove that in each recursive call, either you have just eliminated a pure literal, or the formula has a clause with at most two literals. What recurrence leads to this running time?]

⁰In class, I asserted that Gaussian elimination was probably discovered by Gauss, in violation of Stigler's Law of Eponymy. In fact, a method very similar to Gaussian elimination appears in the Chinese treatise *Nine Chapters on the Mathematical Art*, believed to have been finalized before 100AD, although some material may predate emperor Qin Shi Huang's infamous 'burning of the books and burial of the scholars' in 213BC. The great Chinese mathematician Liu Hui, in his 3rd-century commentary on *Nine Chapters*, compares two variants of the method and counts the number of arithmetic operations used by each, with the explicit goal of finding the more efficient method. This is arguably the earliest recorded analysis of any algorithm.

CS 573: Graduate Algorithms, Fall 2008

Homework 2

Due at 11:59:59pm, Wednesday, October 1, 2008

-
- For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and alias (if any) of every group member on the first page of your submission.
 - We will use the following point breakdown to grade dynamic programming algorithms: 60% for a correct recurrence (including base cases), 20% for correct running time analysis of the memoized recurrence, 10% for correctly transforming the memoized recursion into an iterative algorithm.
 - A greedy algorithm *must* be accompanied by a proof of correctness in order to receive *any* credit.
-

1. (a) Let $X[1..m]$ and $Y[1..n]$ be two arbitrary arrays of numbers. A **common supersequence** of X and Y is another sequence that contains both X and Y as subsequences. Describe and analyze an efficient algorithm to compute the function $scs(X, Y)$, which gives the length of the shortest common supersequence of X and Y .
(b) Call a sequence $X[1..n]$ of numbers **oscillating** if $X[i] < X[i + 1]$ for all even i , and $X[i] > X[i + 1]$ for all odd i . Describe and analyze an efficient algorithm to compute the function $los(X)$, which gives the length of the longest oscillating subsequence of an arbitrary array X of integers.
(c) Call a sequence $X[1..n]$ of numbers **accelerating** if $2 \cdot X[i] < X[i - 1] + X[i + 1]$ for all i . Describe and analyze an efficient algorithm to compute the function $lxs(X)$, which gives the length of the longest accelerating subsequence of an arbitrary array X of integers.
2. A **palindrome** is a string that reads the same forwards and backwards, like x, pop, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. Any string can be broken into sequence of palindromes. For example, the string **bubbasesabana** ('Bubba sees a banana.') can be broken into palindromes in several different ways; for example:

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

Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string **bubbasesabana**, your algorithm would return the integer 3.

3. Describe and analyze an algorithm to solve the traveling salesman problem in $O(2^n \text{poly}(n))$ time. Given an undirected n -vertex graph G with weighted edges, your algorithm should return the weight of the lightest Hamiltonian cycle in G , or ∞ if G has no Hamiltonian cycles. [Hint: The obvious recursive algorithm takes $O(n!)$ time.]

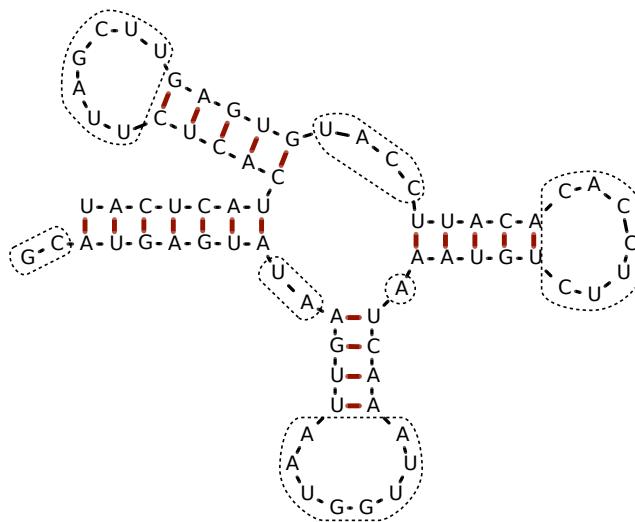
4. Ribonucleic acid (RNA) molecules are long chains of millions of nucleotides or *bases* of four different types: adenine (A), cytosine (C), guanine (G), and uracil (U). The *sequence* of an RNA molecule is a string $b[1..n]$, where each character $b[i] \in \{A, C, G, U\}$ corresponds to a base. In addition to the chemical bonds between adjacent bases in the sequence, hydrogen bonds can form between certain pairs of bases. The set of bonded base pairs is called the *secondary structure* of the RNA molecule.

We say that two base pairs (i, j) and (i', j') with $i < j$ and $i' < j'$ **overlap** if $i < i' < j < j'$ or $i' < i < j' < j$. In practice, most base pairs are non-overlapping. Overlapping base pairs create so-called *pseudoknots* in the secondary structure, which are essential for some RNA functions, but are more difficult to predict.

Suppose we want to predict the best possible secondary structure for a given RNA sequence. We will adopt a drastically simplified model of secondary structure:

- Each base can be paired with at most one other base.
 - Only A-U pairs and C-G pairs can bond.
 - Pairs of the form $(i, i + 1)$ and $(i, i + 2)$ cannot bond.
 - Overlapping base pairs cannot bond.

The last restriction allows us to visualize RNA secondary structure as a sort of fat tree.

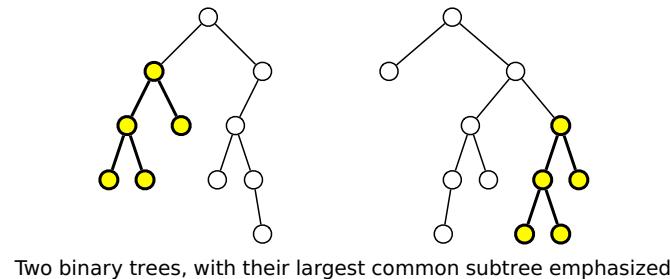


Example RNA secondary structure with 21 base pairs, indicated by heavy red lines. Gaps are indicated by dotted curves. This structure has score $2^2 + 2^2 + 8^2 + 1^2 + 7^2 + 4^2 + 7^2 = 187$

- (a) Describe and analyze an algorithm that computes the maximum possible *number* of base pairs in a secondary structure for a given RNA sequence.
 - (b) A *gap* in a secondary structure is a maximal substring of unpaired bases. Large gaps lead to chemical instabilities, so secondary structures with smaller gaps are more likely. To account for this preference, let's define the *score* of a secondary structure to be the sum of the *squares* of the gap lengths.¹ Describe and analyze an algorithm that computes the minimum possible score of a secondary structure for a given RNA sequence.

¹This score function has absolutely no connection to reality; I just made it up. Real RNA structure prediction requires much more complicated scoring functions.

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



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

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

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

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

CS 573: Graduate Algorithms, Fall 2008

Homework 3

Due at 11:59:59pm, Wednesday, October 22, 2008

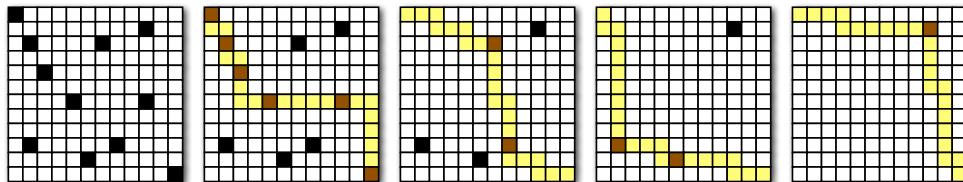
-
- Groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and the HW0 alias (if any) of every group member on the first page of your submission.
-

- Consider an $n \times n$ grid, some of whose cells are marked. A *monotone* path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell. We want to compute the minimum number of monotone paths that cover all marked cells. The input to our problem is an array $M[1..n, 1..n]$ of booleans, where $M[i, j] = \text{TRUE}$ if and only if cell (i, j) is marked.

One of your friends suggests the following greedy strategy:

- Find (somehow) one “good” path π that covers the maximum number of marked cells.
- Unmark the cells covered by π .
- If any cells are still marked, recursively cover them.

Does this greedy strategy always compute an optimal solution? If yes, give a proof. If no, give a counterexample.



Greedily covering the marked cells in a grid with four monotone paths.

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

Describe and analyze an algorithm to compute the smallest tiling path of X as quickly as possible. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X . If you use a greedy algorithm, you must prove that it is correct.



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

3. Given a graph G with edge weights and an integer k , suppose we wish to partition the vertices of G into k subsets S_1, S_2, \dots, S_k so that the sum of the weights of the edges that cross the partition (i.e., that have endpoints in different subsets) is as large as possible.
- Describe an efficient $(1 - 1/k)$ -approximation algorithm for this problem. [Hint: Solve the special case $k = 2$ first.]
 - Now suppose we wish to minimize the sum of the weights of edges that do *not* cross the partition. What approximation ratio does your algorithm from part (a) achieve for this new problem? Justify your answer.
4. Consider the following heuristic for constructing a vertex cover of a connected graph G : ***Return the set of all non-leaf nodes of any depth-first spanning tree.*** (Recall that a depth-first spanning tree is a *rooted* tree; the root is not considered a leaf, even if it has only one neighbor in the tree.)
- Prove that this heuristic returns a vertex cover of G .
 - Prove that this heuristic returns a 2-approximation to the minimum vertex cover of G .
 - Prove that for any $\varepsilon > 0$, there is a graph for which this heuristic returns a vertex cover of size at least $(2 - \varepsilon) \cdot OPT$.
5. Consider the following greedy approximation algorithm to find a vertex cover in a graph:

```

GREEDYVERTEXCOVER( $G$ ):
   $C \leftarrow \emptyset$ 
  while  $G$  has at least one edge
     $v \leftarrow$  vertex in  $G$  with maximum degree
     $G \leftarrow G \setminus v$ 
     $C \leftarrow C \cup v$ 
  return  $C$ 

```

In class we proved that the approximation ratio of this algorithm is $O(\log n)$; your task is to prove a matching lower bound. Specifically, for any positive integer n , describe an n -vertex graph G such that $\text{GREEDYVERTEXCOVER}(G)$ returns a vertex cover that is $\Omega(\log n)$ times larger than optimal. [Hint: $H_n = \Omega(\log n)$.]

- *6. [*Extra credit*] Consider the greedy algorithm for metric TSP: Start at an arbitrary vertex u , and at each step, travel to the closest unvisited vertex.
- Prove that this greedy algorithm is an $O(\log n)$ -approximation algorithm, where n is the number of vertices. [Hint: Show that the k th least expensive edge in the tour output by the greedy algorithm has weight at most $OPT/(n - k + 1)$; try $k = 1$ and $k = 2$ first.]
 - Prove that the greedy algorithm for metric TSP is no better than an $O(\log n)$ -approximation. That is, describe an infinite family of weighted graphs that satisfy the triangle inequality, such that the greedy algorithm returns a cycle whose length is $\Omega(\log n)$ times the optimal TSP tour.

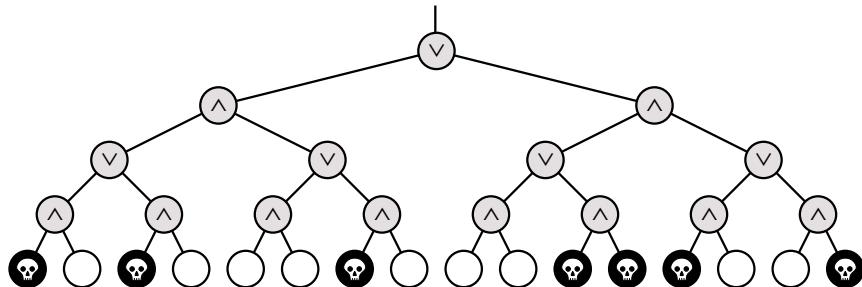
CS 573: Graduate Algorithms, Fall 2008

Homework 4

Due at 11:59:59pm, Wednesday, October 31, 2008

-
- Groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and the HW0 alias (if any) of every group member on the first page of your submission.
 - Unless a problem explicitly states otherwise, you can assume the existence of a function $\text{RANDOM}(k)$, which returns an integer uniformly distributed in the range $\{1, 2, \dots, k\}$ in $O(1)$ time; the argument k must be a positive integer. For example, $\text{RANDOM}(2)$ simulates a fair coin flip, and $\text{RANDOM}(1)$ always returns 1.
-

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



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

- Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*
- Unfortunately, Death won't let you even look at every node in the tree. Describe and analyze a randomized algorithm that determines whether you can win in $O(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*
- [Extra credit]** Describe and analyze a randomized algorithm that determines whether you can win in $O(c^n)$ expected time, for some constant $c < 3$. *[Hint: You may not need to change your algorithm at all.]*

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

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

3. (a) Prove that the expected number of proper descendants of any node in a treap is *exactly* equal to the expected depth of that node.
- (b) Why doesn't the Chernoff-bound argument for depth imply that, with high probability, *every* node in a treap has $O(\log n)$ descendants? The conclusion is obviously bogus—every n -node treap has one node with exactly n descendants!—but what is the flaw in the argument?
- (c) What is the expected number of leaves in an n -node treap? [Hint: What is the probability that in an n -node treap, the node with k th smallest search key is a leaf?]
4. The following randomized algorithm, sometimes called “one-armed quicksort”, selects the r th smallest element in an unsorted array $A[1..n]$. For example, to find the smallest element, you would call $\text{RANDOMSELECT}(A, 1)$; to find the median element, you would call $\text{RANDOMSELECT}(A, \lfloor n/2 \rfloor)$. The subroutine $\text{PARTITION}(A[1..n], p)$ splits the array into three parts by comparing the pivot element $A[p]$ to every other element of the array, using $n - 1$ comparisons altogether, and returns the new index of the pivot element.

```
RANDOMSELECT( $A[1..n], r$ ) :
   $k \leftarrow \text{PARTITION}(A[1..n], \text{RANDOM}(n))$ 
  if  $r < k$ 
    return RANDOMSELECT( $A[1..k - 1], r$ )
  else if  $r > k$ 
    return RANDOMSELECT( $A[k + 1..n], r - k$ )
  else
    return  $A[k]$ 
```

- (a) State a recurrence for the expected running time of RANDOMSELECT , as a function of n and r .
- (b) What is the *exact* probability that RANDOMSELECT compares the i th smallest and j th smallest elements in the input array? The correct answer is a simple function of i , j , and r . [Hint: Check your answer by trying a few small examples.]
- (c) Show that for any n and r , the expected running time of RANDOMSELECT is $\Theta(n)$. You can use either the recurrence from part (a) or the probabilities from part (b).
- *(d) [*Extra Credit*] Find the *exact* expected number of comparisons executed by RANDOMSELECT , as a function of n and r .

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

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

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

```

MELD( $Q_1, Q_2$ ):
  if  $Q_1$  is empty return  $Q_2$ 
  if  $Q_2$  is empty return  $Q_1$ 
  if  $key(Q_1) > key(Q_2)$ 
    swap  $Q_1 \leftrightarrow Q_2$ 
  with probability 1/2
     $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
  else
     $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
  return  $Q_1$ 
```

- (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 $\text{MELD}(Q_1, Q_2)$ is $O(\log n)$, where n is the total number of nodes in both trees. [Hint: How long is a random root-to-leaf path in an n -node binary tree if each left/right choice is made with equal probability?]
- (b) Prove that $\text{MELD}(Q_1, Q_2)$ runs in $O(\log n)$ time with high probability. [Hint: You don't need Chernoff bounds, but you might use the identity $\binom{ck}{k} \leq (ce)^k$.]
- (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)

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

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

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

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: $ℓ_v = 0$. 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  $i \leftarrow 1$  to  $\infty$   
  if  $i > ℓ_v$   
     $ℓ_v \leftarrow i$ ;  $\pi_v[i] \leftarrow \text{RANDOMBIT}$   
  if  $i > ℓ_w$   
     $ℓ_w \leftarrow i$ ;  $\pi_w[i] \leftarrow \text{RANDOMBIT}$   
  if  $\pi_v[i] > \pi_w[i]$   
    return  $v$   
  else if  $\pi_v[i] < \pi_w[i]$   
    return  $w$ 
```

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

- (a) Prove that $E[L] = \Theta(n)$.
- (b) Prove that $E[\ell_v] = \Theta(1)$ for any node v . [Hint: This is equivalent to part (a). Why?]
- (c) Prove that $E[\ell_{\text{root}}] = \Theta(\log n)$. [Hint: Why doesn’t this contradict part (b)?]

CS 573: Graduate Algorithms, Fall 2008

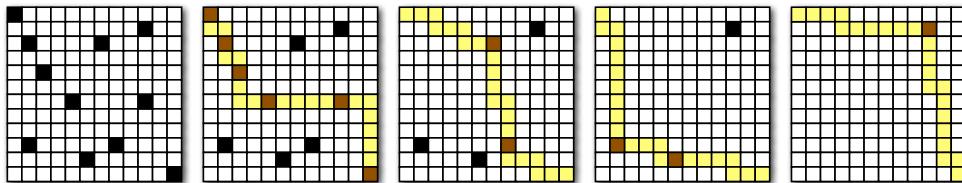
Homework 5

Due at 11:59:59pm, Wednesday, November 19, 2008

- Groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and the HW0 alias (if any) of every group member on the first page of your submission.

- Recall the following problem from Homework 3: You are given an $n \times n$ grid, some of whose cells are marked; the grid is represented by an array $M[1..n, 1..n]$ of booleans, where $M[i, j] = \text{TRUE}$ if and only if cell (i, j) is marked. A *monotone* path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell.

Describe and analyze an efficient algorithm to compute the smallest set of monotone paths that covers every marked cell.



Greedyly covering the marked cells in a grid with four monotone paths.

- Suppose we are given a directed graph $G = (V, E)$, two vertices s and t , and a capacity function $c: V \rightarrow \mathbb{R}^+$. A flow f is *feasible* if the total flow into every vertex v is at most $c(v)$:

$$\sum_u f(u \rightarrow v) \leq c(v) \quad \text{for every vertex } v.$$

Describe and analyze an efficient algorithm to compute a feasible flow of maximum value.

- Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers. We want to *round* A to an integer matrix, by replacing each entry x in A with either $\lfloor x \rfloor$ or $\lceil x \rceil$, without changing the sum of entries in any row or column of A . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

Describe an efficient algorithm that either rounds A in this fashion, or reports correctly that no such rounding is possible.

4. *Ad-hoc networks* are made up of cheap, low-powered wireless devices. In principle¹, these networks can be used on battlefields, in regions that have recently suffered from natural disasters, and in other situations where people might want to monitor conditions in hard-to-reach areas. The idea is that a large collection of cheap, simple devices could be dropped into the area from an airplane (for instance), and then they would somehow automatically configure themselves into an efficiently functioning wireless network.

The devices can communicate only within a limited range. We assume all the devices are identical; there is a distance D such that two devices can communicate if and only if the distance between them is at most D .

We would like our ad-hoc network to be reliable, but because the devices are cheap and low-powered, they frequently fail. If a device detects that it is likely to fail, it should transmit the information it has to some other *backup* device within its communication range. To improve reliability, we require each device x to have k potential backup devices, all within distance D of x ; we call these k devices the *backup set* of x . Also, we do not want any device to be in the backup set of too many other devices; otherwise, a single failure might affect a large fraction of the network.

So suppose we are given the communication radius D , parameters b and k , and an array $d[1..n, 1..n]$ of distances, where $d[i, j]$ is the distance between device i and device j . Describe an algorithm that either computes a backup set of size k for each of the n devices, such that that no device appears in more than b backup sets, or reports (correctly) that no good collection of backup sets exists.

5. Let $G = (V, E)$ be a directed graph where for each vertex v , the in-degree and out-degree of v are equal. Let u and v be two vertices G , and suppose G contains k edge-disjoint paths from u to v . Under these conditions, must G also contain k edge-disjoint paths from v to u ? Give a proof or a counterexample with explanation.

- *6. [*Extra credit*] A *rooted tree* is a directed acyclic graph, in which every vertex has exactly one incoming edge, except for the *root*, which has no incoming edges. Equivalently, a rooted tree consists of a root vertex, which has edges pointing to the roots of zero or more smaller rooted trees. Describe a polynomial-time algorithm to compute, given two rooted trees A and B , the largest common rooted subtree of A and B .

[Hint: Let $LCS(u, v)$ denote the largest common subtree whose root in A is u and whose root in B is v . Your algorithm should compute $LCS(u, v)$ for all vertices u and v using dynamic programming. This would be easy if every vertex had $O(1)$ children, and still straightforward if the children of each node were ordered from left to right and the common subtree had to respect that ordering. But for unordered trees with large degree, you need another trick to combine recursive subproblems efficiently. Don't waste your time trying to reduce the polynomial running time.]

¹but not really in practice

CS 573: Graduate Algorithms, Fall 2008

Homework 6

Practice only

-
- This homework is only for practice; do not submit solutions. At least one (sub)problem (or something *very* similar) will appear on the final exam.
-

1. An *integer program* is a linear program with the additional constraint that the variables must take only integer values.
 - (a) Prove that deciding whether an integer program has a feasible solution is NP-complete.
 - (b) Prove that finding the optimal solution to an integer program is NP-hard.

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

2. Describe precisely how to dualize a linear program written in general form:

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

Keep the number of dual variables as small as possible. The dual of the dual of any linear program should be *syntactically identical* to the original linear program.

3. Suppose you have a subroutine that can solve linear programs in polynomial time, but only if they are both feasible and bounded. Describe an algorithm that solves *arbitrary* linear programs in polynomial time, using this subroutine as a black box. Your algorithm should return an optimal solution if one exists; if no optimum exists, your algorithm should report that the input instance is UNBOUNDED or INFEASIBLE, whichever is appropriate. *[Hint: Add one constraint to guarantee boundedness; add one variable to guarantee feasibility.]*

4. Suppose you are given a set P of n points in some high-dimensional space \mathbb{R}^d , each labeled either *black* or *white*. A *linear classifier* is a d -dimensional vector c with the following properties:

- If p is a black point, then $p \cdot c > 0$.
- If p is a white point, then $p \cdot c < 0$.

Describe an efficient algorithm to find a linear classifier for the given data points, or correctly report that none exists. [*Hint: This is almost trivial, but not quite.*]

Lots more linear programming problems can be found at <http://www.ee.ucla.edu/ee236a/homework/problems.pdf>. Enjoy!

You have 120 minutes to answer all five questions.
Write your answers in the separate answer booklet.
Please turn in your question sheet and your cheat sheet with your answers.

1. You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, several cards are dealt face up in a long row. Then you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. Each card is worth a different number of points. The player that collects the most points wins the game.

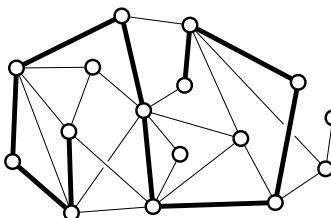
Like most eight-year-olds who haven't studied algorithms, Elmo follows the obvious greedy strategy every time he plays: ***Elmo always takes the card with the higher point value.*** Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely *hates* it when grown-ups let him win.)

- (a) Describe an initial sequence of cards that allows you to win against Elmo, no matter who moves first, but *only* if you do *not* follow Elmo's greedy strategy.
- (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.

Here is a sample game, where both you and Elmo are using the greedy strategy. Elmo wins 8–7. You cannot win this particular game, no matter what strategy you use.

Initial cards	2	4	5	1	3
Elmo takes the 3	2	4	5	1	\$
You take the 2	\$	4	5	1	
Elmo takes the 4		\$	5	1	
You take the 5			\$	1	
Elmo takes the 1				\$	

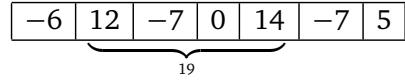
2. **Prove** that the following problem is NP-hard: Given an undirected graph G , find the longest path in G whose length is a multiple of 5.



This graph has a path of length 10, but no path of length 15.

3. Suppose you are given an array $A[1..n]$ of integers. Describe and analyze an algorithm that finds the largest sum of elements in a contiguous subarray $A[i..j]$.

For example, if the array A contains the numbers $[-6, 12, -7, 0, 14, -7, 5]$, your algorithm should return the number 19:



4. A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, ‘bananaanananas’ is a shuffle of ‘banana’ and ‘ananas’ in several different ways:

banana_ananas ban_ana_ana_anas b_an_aa_ana_as

The strings ‘prodgyrnamatammiiincg’ and ‘dyprongarmammicing’ are both shuffles of ‘dynamic’ and ‘programming’:

pro^dg^yrⁿa^ma^m_ii^cg dy_proⁿg^ar^ma^m_ic_ing

Given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, describe and analyze an algorithm to determine whether C is a shuffle of A and B .

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

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

your algorithm should return 8. You can assume that the arrays contain no duplicates. An algorithm that works only in the special case $n = m = k$ is worth 7 points.

[Hint: What can you learn from comparing one element of A to one element of B ?]

You have 120 minutes to answer all five questions.
Write your answers in the separate answer booklet.
Please turn in your question sheet and your cheat sheet with your answers.

1. Consider the following modification of the ‘dumb’ 2-approximation algorithm for minimum vertex cover that we saw in class. The only change is that we output a set of edges instead of a set of vertices.

```
APPROXMINMAXMATCHING( $G$ ):  

 $M \leftarrow \emptyset$   

while  $G$  has at least one edge  

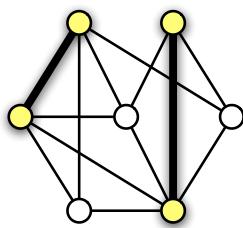
    let  $(u, v)$  be any edge in  $G$   

    remove  $u$  and  $v$  (and their incident edges) from  $G$   

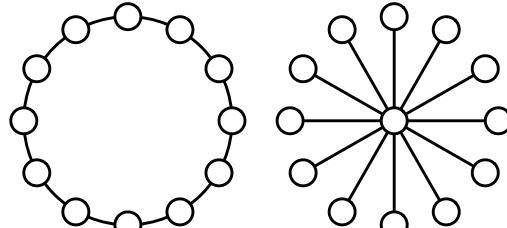
    add  $(u, v)$  to  $M$   

return  $M$ 
```

- (a) **Prove** that this algorithm computes a *matching*—no two edges in M share a common vertex.
- (b) **Prove** that M is a *maximal matching*— M is not a proper subgraph of another matching in G .
- (c) **Prove** that M contains at most twice as many edges as the *smallest maximal matching* in G .



The smallest maximal matching in a graph.



A cycle and a star.

2. Consider the following heuristic for computing a small vertex cover of a graph.

- Assign a random *priority* to each vertex, chosen independently and uniformly from the real interval $[0, 1]$ (just like treaps).
- Mark every vertex that does *not* have larger priority than *all* of its neighbors.

For any graph G , let $OPT(G)$ denote the size of the smallest vertex cover of G , and let $M(G)$ denote the number of nodes marked by this algorithm.

- (a) **Prove** that the set of vertices marked by this heuristic is *always* a vertex cover.
- (b) Suppose the input graph G is a *cycle*, that is, a connected graph where every vertex has degree 2. What is the expected value of $M(G)/OPT(G)$? **Prove** your answer is correct.
- (c) Suppose the input graph G is a *star*, that is, a tree with one central vertex of degree $n - 1$. What is the expected value of $M(G)/OPT(G)$? **Prove** your answer is correct.

3. Suppose we want to write an efficient function $\text{SHUFFLE}(A[1..n])$ that randomly permutes the array A , so that each of the $n!$ permutations is equally likely.

- (a) *Prove* that the following SHUFFLE algorithm is *not* correct. [Hint: There is a two-line proof.]

```

SHUFFLE( $A[1..n]$ ):
  for  $i = 1$  to  $n$ 
    swap  $A[i] \leftrightarrow A[\text{RANDOM}(n)]$ 
```

- (b) Describe and analyze a correct SHUFFLE algorithm whose expected running time is $O(n)$.

Your algorithm may call the function $\text{RANDOM}(k)$, which returns an integer uniformly distributed in the range $\{1, 2, \dots, k\}$ in $O(1)$ time. For example, $\text{RANDOM}(2)$ simulates a fair coin flip, and $\text{RANDOM}(1)$ *always* returns 1.

4. Let Φ be a legal input for 3SAT—a boolean formula in conjunctive normal form, with exactly three literals in each clause. Recall that an assignment of boolean values to the variables in Φ **satisfies** a clause if at least one of its literals is TRUE. The **maximum satisfiability problem**, sometimes called MAX3SAT, asks for the maximum number of clauses that can be simultaneously satisfied by a single assignment. Solving MAXSAT exactly is clearly also NP-hard; this problem asks about approximation algorithms.

- (a) Let $\text{MaxSat}(\Phi)$ denote the maximum number of clauses that can be simultaneously satisfied by one variable assignment. Suppose we randomly assign each variable in Φ to be TRUE or FALSE, each with equal probability. *Prove* that the expected number of satisfied clauses is at least $\frac{7}{8}\text{MaxSat}(\Phi)$.
- (b) Let $\text{MinUnsat}(\Phi)$ denote the *minimum* number of clauses that can be simultaneously *unsatisfied* by a single assignment. *Prove* that it is NP-hard to approximate $\text{MinUnsat}(\Phi)$ within a factor of $10^{10^{100}}$.

5. Consider the following randomized algorithm for generating biased random bits. The subroutine FAIRCOIN returns either 0 or 1 with equal probability; the random bits returned by FAIRCOIN are mutually independent.

```

ONEINTHREE:
  if FAIRCOIN = 0
    return 0
  else
    return 1 - ONEINTHREE
```

- (a) *Prove* that ONEINTHREE returns 1 with probability $1/3$.
- (b) What is the *exact* expected number of times that this algorithm calls FAIRCOIN? *Prove* your answer is correct.
- (c) Now suppose you are given a subroutine ONEINTHREE that generates a random bit that is equal to 1 with probability $1/3$. Describe a FAIRCOIN algorithm that returns either 0 or 1 with equal probability, using ONEINTHREE as a subroutine. **Your only source of randomness is ONEINTHREE; in particular, you may not use the RANDOM function from problem 3.**
- (d) What is the *exact* expected number of times that your FAIRCOIN algorithm calls ONEINTHREE? *Prove* your answer is correct.

You have 180 minutes to answer all seven questions.
Write your answers in the separate answer booklet.
You can keep everything except your answer booklet when you leave.

1. An *integer program* is a linear program with the additional constraint that the variables must take only integer values. **Prove** that deciding whether an integer program has a feasible solution is NP-complete. [Hint: Almost any NP-hard decision problem can be formulated as an integer program. Pick your favorite.]
2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *heater* is a priority search tree in which the priorities are given by the user, and the search keys are distributed uniformly and independently at random in the real interval $[0, 1]$. Intuitively, a heater is the ‘opposite’ of a treap.

The following problems consider an n -node heater T whose node priorities are the integers from 1 to n . We identify nodes in T by their priorities; thus, ‘node 5’ means the node in T with priority 5. The min-heap property implies that node 1 is the root of T . Finally, let i and j be integers with $1 \leq i < j \leq n$.

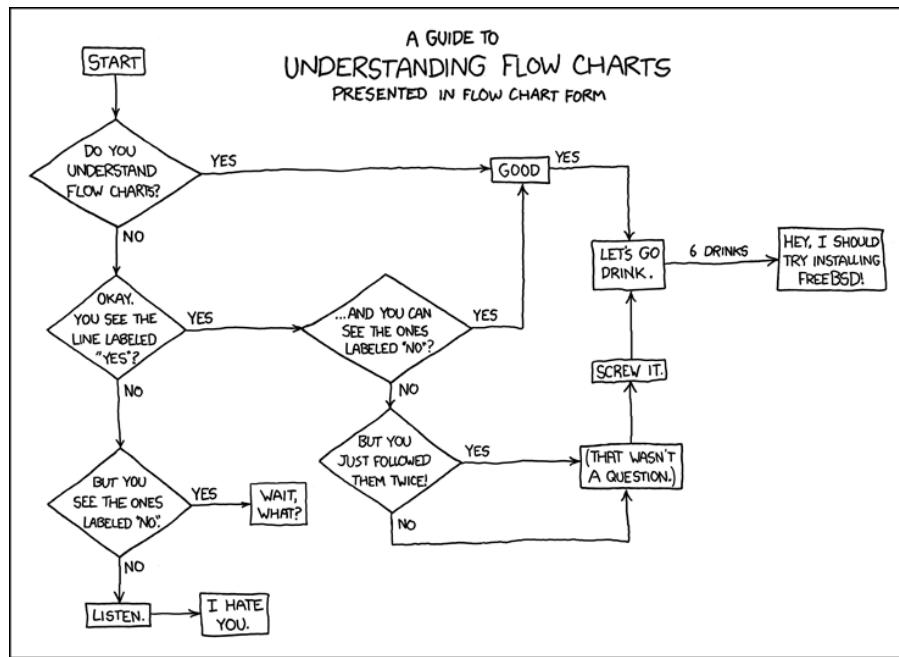
- (a) **Prove** that in a random permutation of the $(i + 1)$ -element set $\{1, 2, \dots, i, j\}$, elements i and j are adjacent with probability $2/(i + 1)$.
 - (b) **Prove** that node i is an ancestor of node j with probability $2/(i + 1)$. [Hint: Use part (a)!]
 - (c) What is the probability that node i is a descendant of node j ? [Hint: Don’t use part (a)!]
 - (d) What is the exact expected depth of node j ?
-
3. The UIUC Faculty Senate has decided to convene a committee to determine whether Chief Illiniwek should become the official *maseot symbol* of the University of Illinois Global Campus. Exactly one faculty member must be chosen from each academic department to serve on this committee. Some faculty members have appointments in multiple departments, but each committee member will represent only one department. For example, if Prof. Blagojevich is affiliated with both the Department of Corruption and the Department of Stupidity, and he is chosen as the Stupidity representative, then someone else must represent Corruption. Finally, University policy requires that any committee on virtual *maseots symbols* must contain the same number of assistant professors, associate professors, and full professors. Fortunately, the number of departments is a multiple of 3.
- Describe an efficient algorithm to select the membership of the Global Illiniwek Committee. Your input is a list of all UIUC faculty members, their ranks (assistant, associate, or full), and their departmental affiliation(s). There are n faculty members and $3k$ departments.
-
4. Let $\alpha(G)$ denote the number of vertices in the largest independent set in a graph G . **Prove** that the following problem is NP-hard: Given a graph G , return *any* integer between $\alpha(G) - 31337$ and $\alpha(G) + 31337$.

5. Let $G = (V, E)$ be a directed graph with capacities $c: E \rightarrow \mathbb{R}^+$, a source vertex s , and a target vertex t . Suppose someone hands you a function $f: E \rightarrow \mathbb{R}$. Describe and analyze a fast algorithm to determine whether f is a maximum (s, t) -flow in G .
6. For some strange reason, you decide to ride your bicycle 3688 miles from Urbana to Wasilla, Alaska, to join in the annual Wasilla Mining Festival and Helicopter Wolf Hunt. The festival starts exactly 32 days from now, so you need to bike an average of 109 miles each day. Because you are a poor starving student, you can only afford to sleep at campgrounds, which are unfortunately *not* spaced exactly 109 miles apart. So some days you will have to ride more than average, and other days less, but you would like to keep the variation as small as possible. You settle on a formal scoring system to help decide where to sleep; if you ride x miles in one day, your score for that day is $(109 - x)^2$. What is the minimum possible total score for all 32 days?

More generally, suppose you have D days to travel DP miles, there are n campgrounds along your route, and your score for traveling x miles in one day is $(x - P)^2$. You are given a sorted array $dist[1..n]$ of real numbers, where $dist[i]$ is the distance from your starting location to the i th campground; it may help to also set $dist[0] = 0$ and $dist[n + 1] = DP$. Describe and analyze a fast algorithm to compute the minimum possible score for your trip. The running time of your algorithm should depend on the integers D and n , but not on the real number P .

7. Describe and analyze efficient algorithms for the following problems.

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



— Randall Munroe, *xkcd*, December 17, 2008 (<http://xkcd.com/518/>)

CS 473: Undergraduate Algorithms, Spring 2009

Homework 0

Due in class at 11:00am, Tuesday, January 27, 2009

- This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** The early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
- Each student must submit individual solutions for this homework. For all future homeworks, groups of up to three students may submit a single, common solution.
- Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. In particular:
 - Submit five separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page. Please do not staple everything together.
 - You may use any source at your disposal—paper, electronic, or human—but you **must** write your solutions in your own words, and you **must** cite every source that you use.
 - Unless explicitly stated otherwise, **every** homework problem requires a proof.
 - Answering “I don’t know” to any homework or exam problem (except for extra credit problems) is worth 25% partial credit.
 - Algorithms or proofs containing phrases like “and so on” or “repeat this process for all n ”, instead of an explicit loop, recursion, or induction, will receive 0 points.

Write the sentence “I understand the course policies.” at the top of your solution to problem 1.

1. Professor George O’Jungle has a 27-node binary tree, in which every node is labeled with a unique letter of the Roman alphabet or the character &. Preorder and postorder traversals of the tree visit the nodes in the following order:
 - Preorder: I Q J H L E M V O T S B R G Y Z K C A & F P N U D W X
 - Postorder: H E M L J V Q S G Y R Z B T C P U D N F W & X A K O I
 - (a) List the nodes in George’s tree in the order visited by an inorder traversal.
 - (b) Draw George’s tree.

2. (a) [5 pts] Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Assume reasonable but nontrivial base cases. If your solution requires a particular base case, say so. ***Do not submit proofs***—just a list of five functions—but you should do them anyway, just for practice.

$$A(n) = 10A(n/5) + n$$

$$B(n) = 2B\left(\left\lceil \frac{n+3}{4} \right\rceil\right) + 5n^{6/7} - 8\sqrt{\frac{n}{\log n}} + 9\lfloor \log^{10} n \rfloor - 11$$

$$C(n) = 3C(n/2) + C(n/3) + 5C(n/6) + n^2$$

$$D(n) = \max_{0 < k < n} (D(k) + D(n - k) + n)$$

$$E(n) = \frac{E(n-1)E(n-3)}{E(n-2)}$$

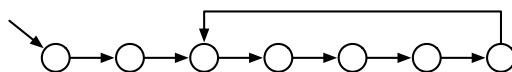
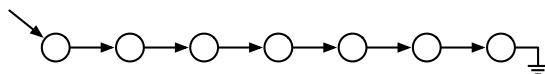
[Hint: Write out the first 20 terms.]

- (b) [5 pts] Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. ***Do not submit proofs***—just a sorted list of 16 functions—but you should do them anyway, just for practice.

Write $f(n) \ll g(n)$ to indicate that $f(n) = o(g(n))$, and write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. We use the notation $\lg n = \log_2 n$.

n	$\lg n$	\sqrt{n}	3^n
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$3^{\sqrt{n}}$	$\sqrt{3^n}$
$3^{\lg n}$	$\lg(3^n)$	$3^{\lg \sqrt{n}}$	$3^{\sqrt{\lg n}}$
$\sqrt{3^{\lg n}}$	$\lg(3^{\sqrt{n}})$	$\lg \sqrt{3^n}$	$\sqrt{\lg(3^n)}$

3. Suppose you are given a pointer to the head of singly linked list. Normally, each node in the list has a pointer to the next element, and the last node's pointer is NULL. Unfortunately, your list might have been corrupted (by a bug in *somebody else's code*, of course), so that some node's pointer leads back to an earlier node in the list.



Top: A standard singly-linked list. Bottom: A corrupted singly-linked list.

Describe an algorithm¹ that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. For full credit, your algorithm should run in $O(n)$ time, where n is the number of nodes in the list, and use $O(1)$ extra space (not counting the list itself).

¹Since you understand the course policies, you know what this phrase means. Right?

4. (a) Prove that any integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example:

$$42 = 3^4 - 3^3 - 3^2 - 3^1$$

$$25 = 3^3 - 3^1 + 3^0$$

$$17 = 3^3 - 3^2 - 3^0$$

- (b) Prove that any integer (positive, negative, or zero) can be written in the form $\sum_i (-2)^i$, where the exponents i are distinct non-negative integers. For example:

$$42 = (-2)^6 + (-2)^5 + (-2)^4 + (-2)^0$$

$$25 = (-2)^6 + (-2)^5 + (-2)^3 + (-2)^0$$

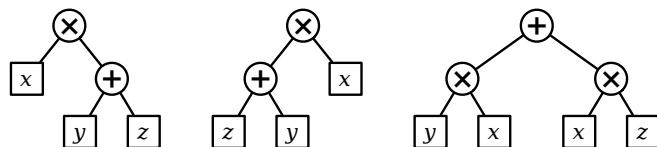
$$17 = (-2)^4 + (-2)^0$$

[Hint: Don't use weak induction. In fact, **never** use weak induction.]

5. An **arithmetic expression tree** is a binary tree where every leaf is labeled with a variable, every internal node is labeled with an arithmetic operation, and every internal node has exactly two children. For this problem, assume that the only allowed operations are $+$ and \times . Different leaves may or may not represent different variables.

Every arithmetic expression tree represents a function, transforming input values for the leaf variables into an output value for the root, by following two simple rules: (1) The value of any $+$ -node is the sum of the values of its children. (2) The value of any \times -node is the product of the values of its children.

Two arithmetic expression trees are **equivalent** if they represent the same function; that is, the same input values for the leaf variables always leads to the same output value at both roots. An arithmetic expression tree is in **normal form** if the parent of every $+$ -node (if any) is another $+$ -node.



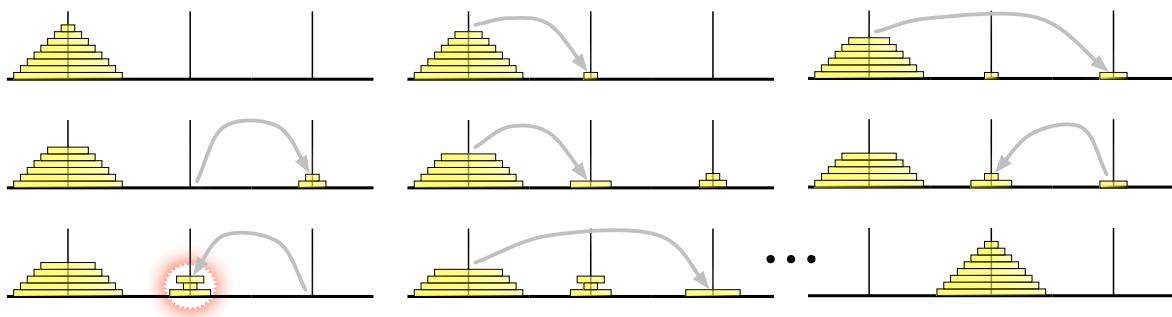
Three equivalent expression trees. Only the third is in normal form.

Prove that for any arithmetic expression tree, there is an equivalent arithmetic expression tree in normal form.

*6. [Extra credit] You may be familiar with the story behind the famous Tower of Hanoi puzzle:

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

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



The Towers of Hollywood. The sixth move leaves the disks out of order.

Describe an algorithm that moves a stack of n disks from one pin to the another using the smallest possible number of moves. Exactly how many moves does your algorithm perform? [Hint: The Hollywood monks can bring about the end of the universe considerably faster than their Benaresian counterparts.]

CS 473: Undergraduate Algorithms, Spring 2009

Homework 1

Due Tuesday, February 3, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

- The traditional Devonian/Cornish drinking song "The Barley Mow" has the following pseudolyrics, where $\text{container}[i]$ is the name of a container that holds 2^i ounces of beer. One version of the song uses the following containers: nippertkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. (Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.)

`BARLEYMOW(n):`

```

"Here's a health to the barley-mow, my brave boys,"
"Here's a health to the barley-mow!"

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

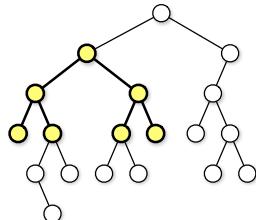
for  $i \leftarrow 1$  to  $n$ 
    "We'll drink it out of the container[ $i$ ], boys,"
    "Here's a health to the barley-mow!"
    for  $j \leftarrow i$  downto 1
        "The container[j],"
        "And the jolly brown bowl!"
        "Here's a health to the barley-mow!"
        "Here's a health to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

```

- Suppose each container name $\text{container}[i]$ is a single word, and you can sing four words a second. How long would it take you to sing $\text{BARLEYMOW}(n)$? (Give a tight asymptotic bound.) [Hint: Is 'barley-mow' one word or two? Does it matter?]
- If you want to sing this song for $n > 20$, you'll have to make up your own container names. To avoid repetition, these names will get progressively longer as n increases¹. Suppose $\text{container}[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing $\text{BARLEYMOW}(n)$? (Give a tight asymptotic bound.)
- Suppose each time you mention the name of a container, you actually drink the corresponding amount of beer: one ounce for the jolly brown bowl, and 2^i ounces for each $\text{container}[i]$. Assuming for purposes of this problem that you are at least 21 years old, exactly how many ounces of beer would you drink if you sang $\text{BARLEYMOW}(n)$? (Give an exact answer, not just an asymptotic bound.)

¹"We'll drink it out of the hemisemidemiyottapint, boys!"

2. For this problem, a *subtree* of a binary tree means any connected subgraph; a binary tree is *complete* if every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

3. (a) Describe and analyze a recursive algorithm to reconstruct a binary tree, given its preorder and postorder node sequences (as in Homework 0, problem 1).
(b) Describe and analyze a recursive algorithm to reconstruct a binary tree, given its preorder and *inorder* node sequences.

CS 473: Undergraduate Algorithms, Spring 2009

Homework 10

Due Tuesday, May 5, 2009 at 11:59:59pm

- Groups of up to three students may submit a single, common solution. Please clearly write every group member's name and NetID on every page of your submission.
 - ***This homework is optional.*** If you submit solutions, they will be graded, and your overall homework grade will be the average of ten homeworks (Homeworks 0–10, dropping the lowest). If you do not submit solutions, your overall homework grade will be the average of *nine* homeworks (Homeworks 0–9, dropping the lowest).
-

1. Suppose you are given a magic black box that can determine ***in polynomial time***, given an arbitrary graph G , the number of vertices in the largest complete subgraph of G . Describe and analyze a ***polynomial-time*** algorithm that computes, given an arbitrary graph G , a complete subgraph of G of maximum size, using this magic black box as a subroutine.
2. PLANARCIRCUITSAT is a special case of CIRCUITSAT where the input circuit is drawn ‘nicely’ in the plane — no two wires cross, no two gates touch, and each wire touches only the gates it connects. (Not every circuit can be drawn this way!) As in the general CIRCUITSAT problem, we want to determine if there is an input that makes the circuit output TRUE?

Prove that PLANARCIRCUITSAT is NP-complete. [*Hint: XOR.*]

3. For each problem below, either describe a polynomial-time algorithm or prove that the problem is NP-complete.
 - (a) A ***double-Eulerian*** circuit in an undirected graph G is a closed walk that traverses every edge in G exactly twice. Given a graph G , does G have a ***double-Eulerian*** circuit?
 - (b) A ***double-Hamiltonian*** circuit in an undirected graph G is a closed walk that visits every vertex in G exactly twice. Given a graph G , does G have a ***double-Hamiltonian*** circuit?

CS 473: Undergraduate Algorithms, Spring 2009

Homework 2

Written solutions due Tuesday, February 10, 2009 at 11:59:59pm.

- Roughly 1/3 of the students will give oral presentations of their solutions to the TAs. *Please check Compass to check whether you are supposed give an oral presentation for this homework.* Please see the course web page for further details.
 - Groups of up to three students may submit a common solution. Please clearly write every group member's name and NetID on every page of your submission.
 - Please start your solution to each numbered problem on a new sheet of paper. Please *don't* staple solutions for different problems together.
 - **For this homework only:** These homework problems ask you to describe recursive backtracking algorithms for various problems. *Don't* use memoization or dynamic programming to make your algorithms more efficient; you'll get to do that on HW3. *Don't* analyze the running times of your algorithms. The **only** things you should submit for each problem are (1) a description of your recursive algorithm, and (2) a brief justification for its correctness.
-

1. A **basic arithmetic expression** is composed of characters from the set $\{1, +, \times\}$ and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$\begin{aligned}
 & 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 \\
 & ((1 + 1) \times (1 + 1 + 1 + 1 + 1)) + ((1 + 1) \times (1 + 1)) \\
 & (1 + 1) \times (1 + 1 + 1 + 1 + 1 + 1 + 1) \\
 & (1 + 1) \times (((1 + 1 + 1) \times (1 + 1)) + 1)
 \end{aligned}$$

Describe a recursive algorithm to compute, given an integer n as input, the minimum number of 1's in a basic arithmetic expression whose value is n . The number of parentheses doesn't matter, just the number of 1's. For example, when $n = 14$, your algorithm should return 8, for the final expression above.

2. A sequence $A = \langle a_1, a_2, \dots, a_n \rangle$ is **bitonic** if there is an index i with $1 < i < n$, such that the prefix $\langle a_1, a_2, \dots, a_i \rangle$ is strictly increasing and the suffix $\langle a_i, a_{i+1}, \dots, a_n \rangle$ is strictly decreasing. In particular, a bitonic sequence must contain at least three elements.

Describe a recursive algorithm to compute, given a sequence A , the length of the longest bitonic subsequence of A .

3. A *palindrome* is a string that reads the same forwards and backwards, like x, pop, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. Any string can be broken into sequence of palindromes. For example, the string bubbbaseesabana ('Bubba sees a banana.') can be broken into palindromes in several different ways; for example:

bub + baseesab + anana
b + u + bb + a + sees + aba + nan + a
b + u + bb + a + sees + a + b + anana
b + u + b + b + a + s + e + e + s + a + b + a + n + a + n + a

Describe a recursive algorithm to compute the minimum number of palindromes that make up a given input string. For example, given the input string bubbbaseesabana, your algorithm would return the integer 3.

CS 473: Undergraduate Algorithms, Spring 2009

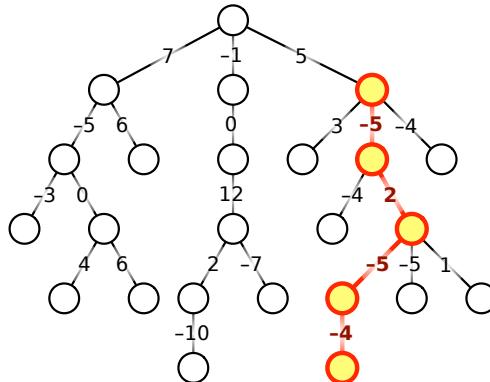
Homework 3

Written solutions due Tuesday, February 17, 2009 at 11:59:59pm.

1. Redo Homework 2, but now with dynamic programming!
 - (a) Describe and analyze an efficient algorithm to compute the minimum number of 1's in a basic arithmetic expression whose value is a given positive integer.
 - (b) Describe and analyze an efficient algorithm to compute the length of the longest bitonic subsequence of a given input sequence.
 - (c) Describe and analyze an efficient algorithm to compute the minimum number of palindromes that make up a given input string.

Please see Homework 2 for more detailed descriptions of each problem. ***Solutions for Homework 2 will be posted Friday, after the HW2 oral presentations.*** You may (and should!) use anything from those solutions without justification.

2. Let T be a rooted tree with integer weights on its edges, which could be positive, negative, or zero. Design an algorithm to find the minimum-length path from a node in T down to one of its descendants. The length of a path is the sum of the weights of its edges. For example, given the tree shown below, your algorithm should return the number -12 . For full credit, your algorithm should run in $O(n)$ time.



The minimum-weight downward path in this tree has weight -12 .

3. Describe and analyze an efficient algorithm to compute the longest common subsequence of *three* given strings. For example, given the input strings EPIDEMIOLOGIST, REFRIGERATION, and SUPERCALIFRAGILISTICEXPIALODOCIOUS, your algorithm should return the number 5, because the longest common subsequence is EIEIO.

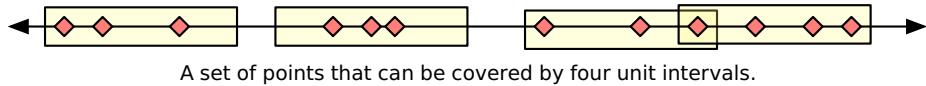
CS 473: Undergraduate Algorithms, Spring 2009

Homework 3½

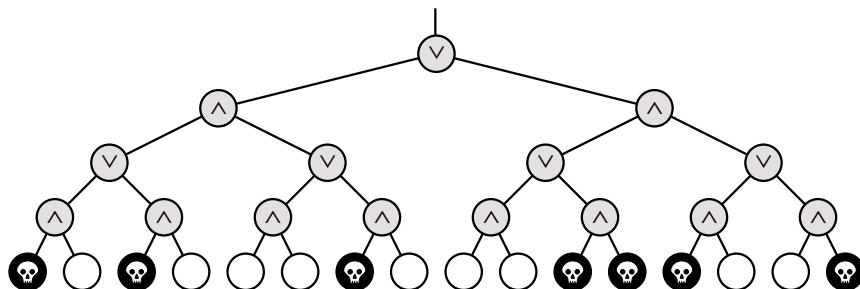
Practice only

1. After graduating from UIUC, you are hired by a mobile phone company to plan the placement of new cell towers along a long, straight, nearly-deserted highway out west. Each cell tower can transmit the same fixed distance from its location. Federal law requires that any building along the highway must be within the broadcast range of at least one tower. On the other hand, your company wants to build as few towers as possible. Given the locations of the buildings, where should you build the towers?

More formally, suppose you are given a set $X = \{x_1, x_2, \dots, x_n\}$ of points on the real number line. Describe an algorithm to compute the minimum number of intervals of length 1 that can cover all the points in X . For full credit, your algorithm should run in $O(n \log n)$ time.



2. (a) The *left spine* of a binary tree is a path starting at the root and following only left-child pointers down to a leaf. What is the expected number of nodes in the left spine of an n -node treap?
- (b) What is the expected number of leaves in an n -node treap? [Hint: What is the probability that in an n -node treap, the node with k th smallest search key is a leaf?]
- (c) Prove that the expected number of proper descendants of any node in a treap is exactly equal to the expected depth of that node.
3. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



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

- (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. [*Hint: This is easy!*]
- (b) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a randomized algorithm that determines whether you can win in $O(3^n)$ expected time. [*Hint: Consider the case $n = 1$.*]
- *(c) Describe a randomized algorithm that determines whether you can win in $O(c^n)$ expected time, for some constant $c < 3$. [*Hint: You may not need to change your algorithm from part (b) at all!*]

CS 473: Undergraduate Algorithms, Spring 2009

Homework 3

Written solutions due Tuesday, March 2, 2009 at 11:59:59pm.

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

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

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

```

MELD( $Q_1, Q_2$ ):
    if  $Q_1$  is empty return  $Q_2$ 
    if  $Q_2$  is empty return  $Q_1$ 
    if  $key(Q_1) > key(Q_2)$ 
        swap  $Q_1 \leftrightarrow Q_2$ 
    with probability 1/2
         $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
    else
         $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
    return  $Q_1$ 

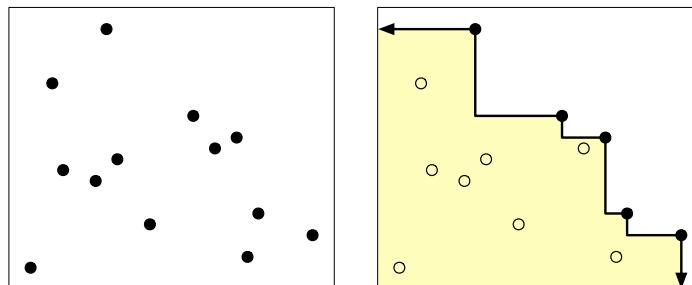
```

- (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 $\text{MELD}(Q_1, Q_2)$ is $O(\log n)$, where n is the total number of nodes in both trees. [Hint: How long is a random root-to-leaf path in an n -node binary tree if each left/right choice is made with equal probability?]
- (b) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ expected time.)

2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *heater* is a priority search tree in which the priorities are given by the user, and the search keys are distributed uniformly and independently at random in the real interval $[0, 1]$. Intuitively, a heater is the ‘opposite’ of a treap.

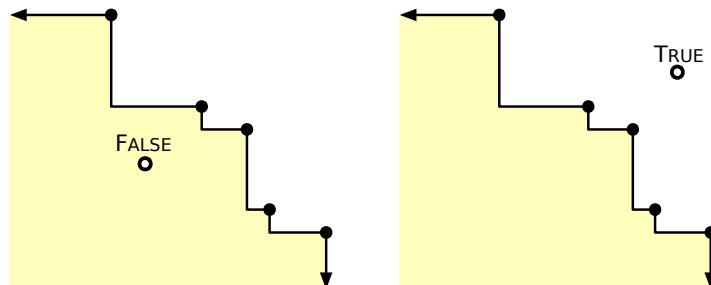
The following problems consider an n -node heater T whose node priorities are the integers from 1 to n . We identify nodes in T by their priorities; thus, ‘node 5’ means the node in T with priority 5. The min-heap property implies that node 1 is the root of T . Finally, let i and j be integers with $1 \leq i < j \leq n$.

- (a) **Prove** that in a random permutation of the $(i + 1)$ -element set $\{1, 2, \dots, i, j\}$, elements i and j are adjacent with probability $2/(i + 1)$.
 - (b) **Prove** that node i is an ancestor of node j with probability $2/(i + 1)$. [Hint: Use part (a)!]
 - (c) What is the probability that node i is a descendant of node j ? [Hint: Don’t use part (a)!]
 - (d) What is the exact expected depth of node j ?
3. Let P be a set of n points in the plane. The *staircase* of P is the set of all points in the plane that have at least one point in P both above and to the right.



A set of points in the plane and its staircase (shaded).

- (a) Describe an algorithm to compute the staircase of a set of n points in $O(n \log n)$ time.
- (b) Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm ABOVE?(x, y) that returns TRUE if the point (x, y) is above the staircase, or FALSE otherwise. Your data structure should use $O(n)$ space, and your ABOVE? algorithm should run in $O(\log n)$ time.



Two staircase queries.

CS 473: Undergraduate Algorithms, Spring 2009

Homework 5

Written solutions due Tuesday, March 9, 2009 at 11:59:59pm.

1. Remember the difference between stacks and queues? Good.
 - (a) Describe how to implement a queue using two stacks and $O(1)$ additional memory, so that the amortized time for any enqueue or dequeue operation is $O(1)$. The *only* access you have to the stacks is through the standard methods PUSH and POP.
 - (b) A *quack* is an abstract data type that combines properties of both stacks and queues. It can be viewed as a list of elements written left to right such that three operations are possible:
 - **Push:** add a new item to the left end of the list;
 - **Pop:** remove the item on the left end of the list;
 - **Pull:** remove the item on the right end of the list.
 Implement a quack using *three* stacks and $O(1)$ additional memory, so that the amortized time for any push, pop, or pull operation is $O(1)$. Again, you are *only* allowed to access the stacks through the standard methods PUSH and POP.

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

Describe and analyze an algorithm to purge an *arbitrary* n -node dirty binary search tree in $O(n)$ time, using at most $O(\log n)$ space (in addition to the tree itself). Don't forget to include the recursion stack in your space bound. An algorithm that uses $\Theta(n)$ additional space in the worst case is worth half credit.

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

Let T be an arbitrary binary tree. Suppose every node v in T stores a secondary structure of size $O(\text{size}(v))$, which can be built in $O(\text{size}(v))$ time, where $\text{size}(v)$ denotes the number of descendants of v . Performing a rotation at any node v now requires $O(\text{size}(v))$ time, because we have to rebuild one of the secondary structures.

 - (a) [1 pt] Overall, how much space does this data structure use *in the worst case*?
 - (b) [1 pt] How much space does this structure use if the primary search tree T is perfectly balanced?
 - (c) [2 pts] Suppose T is a splay tree. Prove that the *amortized* cost of a splay (and therefore of a search, insertion, or deletion) is $\Omega(n)$. [*Hint: This is easy!*]

- (d) [3 pts] Now suppose T is a scapegoat tree, and that rebuilding the subtree rooted at v requires $\Theta(\text{size}(v) \log \text{size}(v))$ time (because we also have to rebuild the secondary structures at every descendant of v). What is the *amortized* cost of inserting a new element into T ?
- (e) [3 pts] Finally, suppose T is a treap. What's the worst-case *expected* time for inserting a new element into T ?

CS 473: Undergraduate Algorithms, Spring 2009

Homework 6

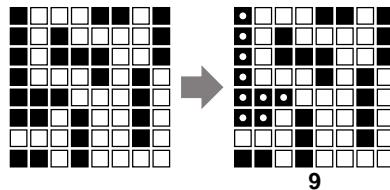
Written solutions due Tuesday, March 17, 2009 at 11:59:59pm.

1. Let G be an undirected graph with n nodes. Suppose that G contains two nodes s and t , such that every path from s to t contains more than $n/2$ edges.
 - (a) Prove that G must contain a vertex v that lies on *every* path from s to t .
 - (b) Describe an algorithm that finds such a vertex v in $O(V + E)$ time.

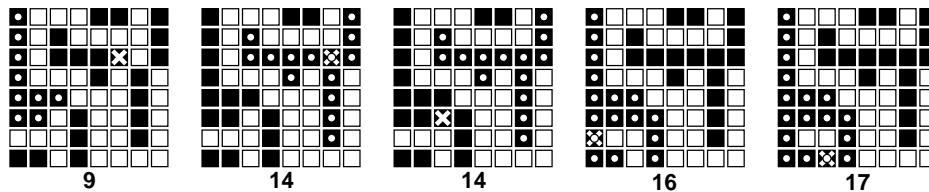
2. Suppose you are given a graph G with weighted edges and a minimum spanning tree T of G .
 - (a) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is decreased.
 - (b) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is increased.

In both cases, the input to your algorithm is the edge e and its new weight; your algorithms should modify T so that it is still a minimum spanning tree. [Hint: Consider the cases $e \in T$ and $e \notin T$ separately.]

3. (a) Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an $n \times n$ bitmap $B[1..n, 1..n]$.
 For example, given the bitmap below as input, your algorithm should return the number 9, because the largest connected black component (marked with white dots on the right) contains nine pixels.



- (b) Design and analyze an algorithm $\text{BLACKEN}(i, j)$ that colors the pixel $B[i, j]$ black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.
 For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the BLACKEN algorithm.



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

CS 473: Undergraduate Algorithms, Spring 2009

Homework 6½

Practice only—do not submit solutions

1. In class last Tuesday, we discussed Ford's generic shortest-path algorithm—relax arbitrary tense edges until no edge is tense. This problem asks you to fill in part of the proof that this algorithm is correct.
 - (a) Prove that after *every* call to RELAX, for every vertex v , either $dist(v) = \infty$ or $dist(v)$ is the total weight of some path from s to v .
 - (b) Prove that for every vertex v , when the generic algorithm halts, either $pred(v) = \text{NULL}$ and $dist(v) = \infty$, or $dist(v)$ is the total weight of the predecessor chain ending at v :

$$s \rightarrow \dots \rightarrow pred(pred(v)) \rightarrow pred(v) \rightarrow v.$$

2. Describe a modification of Shimbels shortest-path algorithm that actually computes a negative-weight cycle if any such cycle is reachable from s , or a shortest-path tree rooted at s if there is no such cycle. Your modified algorithm should still run in $O(VE)$ time.
3. After graduating you accept a job with Aerophobes-Я-Us, the leading traveling agency for people who hate to fly. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying (and by extension, airports), so any trip you plan needs to be as short as possible. You know all the departure and arrival times of all the flights on the planet.

Suppose one of your customers wants to fly from city X to city Y . Describe an algorithm to find a sequence of flights that minimizes the *total time in transit*—the length of time from the initial departure to the final arrival, including time at intermediate airports waiting for connecting flights. [Hint: Modify the input data and apply Dijkstra's algorithm.]

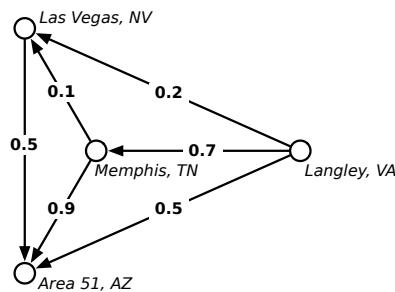
CS 473: Undergraduate Algorithms, Spring 2009

Homework 6 $\frac{3}{4}$

Practice only—do not submit solutions

1. Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road *won't* be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What route should he take so that he has the least chance of being abducted?

More formally, you are given a directed graph $G = (V, E)$, where every edge e has an independent safety probability $p(e)$. The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex s to a given target vertex t .



For example, with the probabilities shown above, if Mulder tries to drive directly from Langley to Area 51, he has a 50% chance of getting there without being abducted. If he stops in Memphis, he has a $0.7 \times 0.9 = 63\%$ chance of arriving safely. If he stops first in Memphis and then in Las Vegas, he has a $1 - 0.7 \times 0.1 \times 0.5 = 96.5\%$ chance of being abducted! (That's how they got Elvis, you know.)

2. Let $G = (V, E)$ be a directed graph with weighted edges; edge weights could be positive, negative, or zero. Suppose the vertices of G are partitioned into k disjoint subsets V_1, V_2, \dots, V_k ; that is, every vertex of G belongs to exactly one subset V_i . For each i and j , let $\delta(i, j)$ denote the minimum shortest-path distance between any vertex in V_i and any vertex in V_j :

$$\delta(i, j) = \min\{\text{dist}(u, v) \mid u \in V_i \text{ and } v \in V_j\}.$$

Describe an algorithm to compute $\delta(i, j)$ for all i and j in time $O(VE + kE \log E)$. The output from your algorithm is a $k \times k$ array.

3. Recall¹ that a deterministic finite automaton (DFA) is formally defined as a tuple $M = (\Sigma, Q, q_0, F, \delta)$, where the finite set Σ is the input alphabet, the finite set Q is the set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of final (accepting) states, and $\delta: Q \times \Sigma \rightarrow Q$ is the transition function. Equivalently, a DFA is a directed (multi-)graph with labeled edges, such that each symbol in Σ is the label of exactly one edge leaving any vertex. There is a special ‘start’ vertex q_0 , and a subset of the vertices are marked as ‘accepting states’. Any string in Σ^* describes a unique walk starting at q_0 .

Stephen Kleene² proved that the language accepted by any DFA is identical to the language described by some regular expression. This problem asks you to develop a variant of the Floyd-Warshall all-pairs shortest path algorithm that computes a regular expression that is equivalent to the language accepted by a given DFA.

Suppose the input DFA M has n states, numbered from 1 to n , where (without loss of generality) the start state is state 1. Let $L(i, j, r)$ denote the set of all words that describe walks in M from state i to state j , where every intermediate state lies in the subset $\{1, 2, \dots, r\}$; thus, the language accepted by the DFA is exactly

$$\bigcup_{q \in F} L(1, q, n).$$

Let $R(i, j, r)$ be a regular expression that describes the language $L(i, j, r)$.

- (a) What is the regular expression $R(i, j, 0)$?
- (b) Write a recurrence for the regular expression $R(i, j, r)$ in terms of regular expressions of the form $R(i', j', r - 1)$.
- (c) Describe a polynomial-time algorithm to compute $R(i, j, n)$ for all states i and j . (Assume that you can concatenate two regular expressions in $O(1)$ time.)

¹No, really, you saw this in CS 273/373.

²Pronounced ‘clay knee’, not ‘clean’ or ‘clean-ee’ or ‘clay-nuh’ or ‘dimaggio’.

CS 473: Undergraduate Algorithms, Spring 2009

Homework 7

Due Tuesday, April 14, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.
-
1. A graph is *bipartite* if its vertices can be colored black or white such that every edge joins vertices of two different colors. A graph is *d-regular* if every vertex has degree d . A *matching* in a graph is a subset of the edges with no common endpoints; a matching is *perfect* if it touches every vertex.
 - (a) Prove that every regular bipartite graph contains a perfect matching.
 - (b) Prove that every d -regular bipartite graph is the union of d perfect matchings.
 2. Let $G = (V, E)$ be a directed graph where for each vertex v , the in-degree of v and out-degree of v are equal. Let u and v be two vertices G , and suppose G contains k edge-disjoint paths from u to v . Under these conditions, must G also contain k edge-disjoint paths from v to u ? Give a proof or a counterexample with explanation.
 3. A flow f is called *acyclic* if the subgraph of directed edges with positive flow contains no directed cycles. A flow is *positive* if its value is greater than 0.
 - (a) A *path flow* assigns positive values only to the edges of one simple directed path from s to t . Prove that every positive acyclic flow can be written as the sum of a finite number of path flows.
 - (b) Describe a flow in a directed graph that *cannot* be written as the sum of path flows.
 - (c) A *cycle flow* assigns positive values only to the edges of one simple directed cycle. Prove that every flow can be written as the sum of a finite number of path flows and cycle flows.
 - (d) Prove that for any flow f , there is an acyclic flow with the same value as f . (In particular, this implies that some maximum flow is acyclic.)

CS 473: Undergraduate Algorithms, Spring 2009

Homework 8

Due Tuesday, April 21, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

- A *cycle cover* of a given directed graph $G = (V, E)$ is a set of vertex-disjoint cycles that cover all the vertices. Describe and analyze an efficient algorithm to find a cycle cover for a given graph, or correctly report that none exists. [*Hint: Use bipartite matching!*]
- Suppose we are given an array $A[1..m][1..n]$ of non-negative real numbers. We want to *round* A to an integer matrix, by replacing each entry x in A with either $\lfloor x \rfloor$ or $\lceil x \rceil$, without changing the sum of entries in any row or column of A . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

Describe an efficient algorithm that either rounds A in this fashion, or reports correctly that no such rounding is possible.

- Ad-hoc networks* are made up of cheap, low-powered wireless devices. In principle¹, these networks can be used on battlefields, in regions that have recently suffered from natural disasters, and in other hard-to-reach areas. The idea is that several simple devices could be distributed randomly in the area of interest (for example, dropped from an airplane), and then they would somehow automatically configure themselves into an efficiently functioning wireless network.

The devices can communicate only within a limited range. We assume all the devices are identical; there is a distance D such that two devices can communicate if and only if the distance between them is at most D .

We would like our ad-hoc network to be reliable, but because the devices are cheap and low-powered, they frequently fail. If a device detects that it is likely to fail, it should transmit all its information to some other *backup* device within its communication range. To improve reliability, we require each device x to have k potential backup devices, all within distance D of x ; we call these k devices the **backup set** of x . Also, we do not want any device to be in the backup set of too many other devices; otherwise, a single failure might affect a large fraction of the network.

Suppose we are given the communication distance D , parameters b and k , and an array $d[1..n, 1..n]$ of distances, where $d[i, j]$ is the distance between device i and device j . Describe and analyze an algorithm that either computes a backup set of size k for each of the n devices, such that no device appears in more than b backup sets, or correctly reports that no good collection of backup sets exists.

¹but not so much in practice

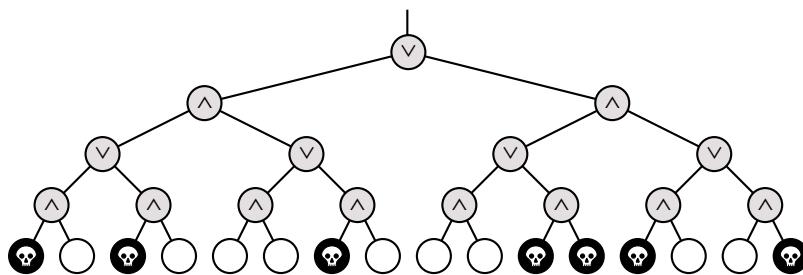
CS 473: Undergraduate Algorithms, Spring 2009

Homework 9

Due Tuesday, April 28, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

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



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

- (a) Describe and analyze a deterministic algorithm to determine whether or not you can win.
[Hint: This is easy.]
 - (b) Prove that *every* deterministic algorithm must examine *every* leaf of the tree in the worst case. Since there are 4^n leaves, this implies that any deterministic algorithm must take $\Omega(4^n)$ time in the worst case. Use an adversary argument; in other words, assume that Death cheats.
 - (c) *[Extra credit]* Describe a *randomized* algorithm that runs in $O(3^n)$ expected time.
-
- We say that an array $A[1..n]$ is *k-sorted* if it can be divided into k blocks, each of size n/k , such that the elements in each block are larger than the elements in earlier blocks, and smaller than elements in later blocks. The elements within each block need not be sorted.

For example, the following array is 4-sorted:

1	2	4	3		7	6	8	5		10	11	9	12		15	13	16	14
---	---	---	---	--	---	---	---	---	--	----	----	---	----	--	----	----	----	----

- (a) Describe an algorithm that k -sorts an arbitrary array in time $O(n \log k)$.
- (b) Prove that any comparison-based k -sorting algorithm requires $\Omega(n \log k)$ comparisons in the worst case.
- (c) Describe an algorithm that completely sorts an already k -sorted array in time $O(n \log(n/k))$.
- (d) Prove that any comparison-based algorithm to completely sort a k -sorted array requires $\Omega(n \log(n/k))$ comparisons in the worst case.

In all cases, you can assume that n/k is an integer.

3. UIUC has just finished constructing the new Reingold Building, the tallest dormitory on campus. In order to determine how much insurance to buy, the university administration needs to determine the highest safe floor in the building. A floor is considered *safe* if a ~~drunk~~ student *an egg* can fall from a window on that floor and land without breaking; if the egg breaks, the floor is considered *unsafe*. Any floor that is higher than an unsafe floor is also considered unsafe. The only way to determine whether a floor is safe is to drop an egg from a window on that floor.

You would like to find the lowest unsafe floor L by performing as few tests as possible; unfortunately, you have only a very limited supply of eggs.

- (a) Prove that if you have only one egg, you can find the lowest unsafe floor with L tests. [*Hint: Yes, this is trivial.*]
- (b) Prove that if you have only one egg, you must perform at least L tests in the worst case. In other words, prove that your algorithm from part (a) is optimal. [*Hint: Use an adversary argument.*]
- (c) Describe an algorithm to find the lowest unsafe floor using *two* eggs and only $O(\sqrt{L})$ tests. [*Hint: Ideally, each egg should be dropped the same number of times. How many floors can you test with n drops?*]
- (d) Prove that if you start with two eggs, you must perform at least $\Omega(\sqrt{L})$ tests in the worst case. In other words, prove that your algorithm from part (c) is optimal.
- *(e) [**Extra credit!**] Describe an algorithm to find the lowest unsafe floor using k eggs, using as few tests as possible, and prove your algorithm is optimal for all values of k .

CS 473: Undergraduate Algorithms, Spring 2009

Head Banging Session 0

January 20 and 21, 2009

1. Solve the following recurrences. If base cases are provided, find an *exact* closed-form solution. Otherwise, find a solution of the form $\Theta(f(n))$ for some function f .

- **Warmup:** You should be able to solve these almost as fast as you can write down the answers.

- (a) $A(n) = A(n - 1) + 1$, where $A(0) = 0$.
- (b) $B(n) = B(n - 5) + 2$, where $B(0) = 17$.
- (c) $C(n) = C(n - 1) + n^2$
- (d) $D(n) = 3D(n/2) + n^2$
- (e) $E(n) = 4E(n/2) + n^2$
- (f) $F(n) = 5F(n/2) + n^2$

- **Real practice:**

- (a) $A(n) = A(n/3) + 3A(n/5) + A(n/15) + n$
- (b) $B(n) = \min_{0 < k < n} (B(k) + B(n - k) + n)$
- (c) $C(n) = \max_{n/4 < k < 3n/4} (C(k) + C(n - k) + n)$
- (d) $D(n) = \max_{0 < k < n} (D(k) + D(n - k) + k(n - k))$, where $D(1) = 0$
- (e) $E(n) = 2E(n - 1) + E(n - 2)$, where $E(0) = 1$ and $E(1) = 2$
- (f) $F(n) = \frac{1}{F(n - 1)F(n - 2)}$, where $F(0) = 1$ and $F(2) = 2$
- (g) $G(n) = nG(\sqrt{n}) + n^2$

2. The *Fibonacci numbers* F_n are defined recursively as follows: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for every integer $n \geq 2$. The first few Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,

Prove that any non-negative integer can be written as the sum of distinct *non-consecutive* Fibonacci numbers. That is, if any Fibonacci number F_n appears in the sum, then its neighbors F_{n-1} and F_{n+1} do not. For example:

$$\begin{aligned}
 88 &= 55 + 21 + 8 + 3 + 1 &= F_{10} + F_8 + F_6 + F_4 + F_2 \\
 42 &= &= F_9 + F_6 \\
 17 &= 13 + 3 + 1 &= F_7 + F_4 + F_2
 \end{aligned}$$

3. Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

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

1. An *inversion* in an array $A[1..n]$ is a pair of indices (i, j) such that $i < j$ and $A[i] > A[j]$. The number of inversions in an n -element array is between 0 (if the array is sorted) and $\binom{n}{2}$ (if the array is sorted backward).

Describe and analyze an algorithm to count the number of inversions in an n -element array in $O(n \log n)$ time.

2. (a) Prove that the following algorithm actually sorts its input.

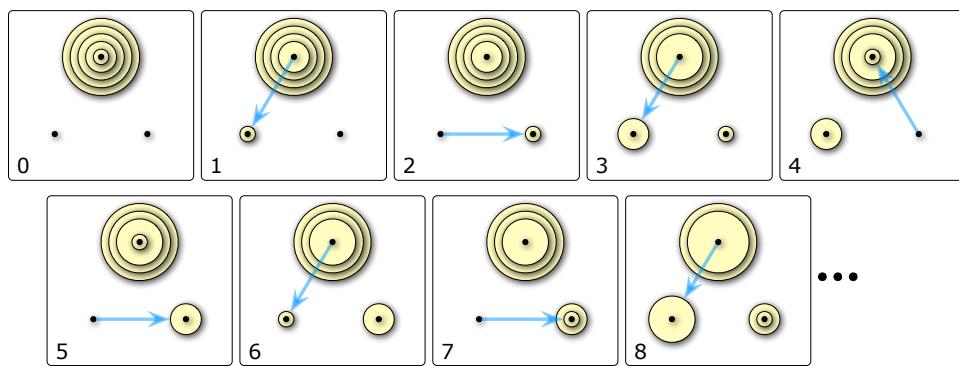
```

STOOGESORT( $A[0..n-1]$ ) :
  if  $n = 2$  and  $A[0] > A[1]$ 
    swap  $A[0] \leftrightarrow A[1]$ 
  else if  $n > 2$ 
     $m = \lceil 2n/3 \rceil$ 
    STOOGESORT( $A[0..m-1]$ )
    STOOGESORT( $A[n-m..n-1]$ )
    STOOGESORT( $A[0..m-1]$ )

```

- (b) Would STOOGESORT still sort correctly if we replaced $m = \lceil 2n/3 \rceil$ with $m = \lfloor 2n/3 \rfloor$? Justify your answer.
- (c) State a recurrence (including base case(s)) for the number of comparisons executed by STOOGESORT.
- (d) Solve this recurrence. [Hint: Ignore the ceiling.]
- (e) **To think about on your own:** Prove that the number of swaps executed by STOOGESORT is at most $\binom{n}{2}$.
3. Consider the following restricted variants of the Tower of Hanoi puzzle. In each problem, the needles are numbered 0, 1, and 2, and your task is to move a stack of n disks from needle 1 to needle 2.

- (a) Suppose you are forbidden to move any disk directly between needle 1 and needle 2; every move must involve needle 0. Describe an algorithm to solve this version of the puzzle using as few moves as possible. Exactly how many moves does your algorithm make?
- (b) Suppose you are only allowed to move disks from needle 0 to needle 2, from needle 2 to needle 1, or from needle 1 to needle 0. Equivalently, Suppose the needles are arranged in a circle and numbered in clockwise order, and you are only allowed to move disks counterclockwise. Describe an algorithm to solve this version of the puzzle using as few moves as possible. Exactly how many moves does your algorithm make?



The first eight moves in a counterclockwise Towers of Hanoi solution

- ★(c) Finally, suppose you are forbidden to move any disk directly from needle 1 to 2, but any other move is allowed. Describe an algorithm to solve this version of the puzzle using as few moves as possible. *Exactly* how many moves does your algorithm make?

[Hint: This version is considerably harder than the other two.]

CS 473: Undergraduate Algorithms, Spring 2009

HBS 10

1. Consider the following problem, called *BOX-DEPTH*: Given a set of n axis-aligned rectangles in the plane, how big is the largest subset of these rectangles that contain a common point?
 - (a) Describe a polynomial-time reduction from *BOX-DEPTH* to *MAX-CLIQUE*.
 - (b) Describe and analyze a polynomial-time algorithm for *BOX-DEPTH*. [Hint: $O(n^3)$ time should be easy, but $O(n \log n)$ time is possible.]
 - (c) Why don't these two results imply that $P = NP$?
2. Suppose you are given a magic black box that can determine in polynomial time, given an arbitrary weighted graph G , the length of the shortest Hamiltonian cycle in G . Describe and analyze a polynomial-time algorithm that computes, given an arbitrary weighted graph G , the shortest Hamiltonian cycle in G , using this magic black box as a subroutine.
3. Prove that the following problems are NP-complete.
 - (a) Given an undirected graph G , does G have a spanning tree in which every node has degree at most 17?
 - (b) Given an undirected graph G , does G have a spanning tree with at most 42 leaves?

CS 473: Undergraduate Algorithms, Spring 2009

HBS 11

1. You step in a party with a camera in your hand. Each person attending the party has some friends there. You want to have exactly one picture of each person in your camera. You want to use the following protocol to collect photos. At each step, the person that has the camera in his hand takes a picture of one of his/her friends and pass the camera to him/her. Of course, you only like the solution if it finishes when the camera is in your hand. Given the friendship matrix of the people in the party, design a polynomial algorithm that decides whether this is possible, or prove that this decision problem is NP-hard.
2. A boolean formula is in disjunctive normal form (DNF) if it is a disjunctions (OR) of several clauses, each of which is the conjunction (AND) of several literals, each of which is either a variable or its negation. For example:

$$(a \wedge b \wedge c) \vee (\bar{a} \wedge b) \vee (\bar{c} \wedge x)$$

Given a DNF formula give a polynomial algorithm to check whether it is satisfiable or not. Why this does not imply $P = NP$.

3. Prove that the following problems are NP-complete.
 - (a) Given an undirected graph G , does G have a spanning tree in which every node has degree at most 17?
 - (b) Given an undirected graph G , does G have a spanning tree with at most 42 leaves?

CS 473: Undergraduate Algorithms, Spring 2009

HBS 2

1. Consider two horizontal lines l_1 and l_2 in the plane. There are n points on l_1 with x -coordinates $A = a_1, a_2, \dots, a_n$ and there are n points on l_2 with x -coordinates $B = b_1, b_2, \dots, b_n$. Design an algorithm to compute, given A and B , a largest set S of non-intersecting line segments subject to the following restrictions:
 - (a) Any segment in S connects a_i to b_i for some $i (1 \leq i \leq n)$.
 - (b) Any two segments in S do not intersect.
2. Consider a $2^n \times 2^n$ chess board with one (arbitrarily chosen) square removed. Prove that any such chessboard can be tiled without gaps or overlaps by L-shaped pieces of 3 squares each. Can you give an algorithm to do the tiling?
3. Given a string of letters $Y = y_1 y_2 \dots y_n$, a segmentation of Y is a partition of its letters into contiguous blocks of letters (also called words). Each word has a quality that can be computed by a given oracle (e.g. you can call `quality("meet")` to get the quality of the word "meet"). The quality of a segmentation is equal to the sum over the qualities of its words. Each call to the oracle takes linear time in terms of the argument; that is `quality(S)` takes $O(|S|)$.

Using the given oracle, give an algorithm that takes a string Y and computes a segmentation of maximum total quality.

CS 473: Undergraduate Algorithms, Spring 2009

HBS 3

1. Change your recursive solutions for the following problems to efficient algorithms (Hint: use dynamic programming!).
 - (a) Consider two horizontal lines l_1 and l_2 in the plane. There are n points on l_1 with x -coordinates $A = a_1, a_2, \dots, a_n$ and there are n points on l_2 with x -coordinates $B = b_1, b_2, \dots, b_n$. Design an algorithm to compute, given A and B , a largest set S of non-intersecting line segments subject to the following restrictions:
 - i. Any segment in S connects a_i to b_i for some i ($1 \leq i \leq n$).
 - ii. Any two segments in S do not intersect.
 - (b) Given a string of letters $Y = y_1y_2\dots y_n$, a segmentation of Y is a partition of its letters into contiguous blocks of letters (also called words). Each word has a quality that can be computed by a given oracle (e.g. you can call *quality("meet")* to get the quality of the word "meet"). The quality of a segmentation is equal to the sum over the qualities of its words. Each call to the oracle takes linear time in terms of the argument; that is *quality*(S) takes $O(|S|)$.Using the given oracle, give an algorithm that takes a string Y and computes a segmentation of maximum total quality.
2. Give a polynomial time algorithm which given two strings A and B returns the longest sequence S that is a subsequence of A and B .
3. Consider a rooted tree T . Assume the root has a message to send to all nodes. At the beginning only the root has the message. If a node has the message, it can forward it to one of its children at each time step. Design an algorithm to find the minimum number of time steps required for the message to be delivered to all nodes.

CS 473: Undergraduate Algorithms, Spring 2009

HBS 3.5

1. Say you are given n jobs to run on a machine. Each job has a start time and an end time. If a job is chosen to be run, then it must start at its start time and end at its end time. Your goal is to accept as many jobs as possible, regardless of the job lengths, subject to the constraint that the processor can run at most one job at any given point in time. Provide an algorithm to do this with a running time that is polynomial in n . You may assume for simplicity that no two jobs have the same start or end times, but the start time and end time of two jobs can overlap.
2. Describe and analyze an algorithm that chooses one element uniformly at random from a data stream, without knowing the length of the stream in advance. Your algorithm should spend $O(1)$ time per stream element and use $O(1)$ space (not counting the stream itself).
3. Design and analyze an algorithm that return a permutation of the integers $\{1, 2, \dots, n\}$ chosen uniformly at random.

CS 473: Undergraduate Algorithms, Spring 2009

HBS 4

1. Let x and y be two elements of a set S whose ranks differ by exactly r . Prove that in a treap for S , the expected length of the unique path from x to y is $O(\log r)$
2. Consider the problem of making change for n cents using the least number of coins.
 - (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.
 - (b) Suppose that the available coins have the values c^0, c^1, \dots, c^k for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.
 - (c) Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution, show why.
 - (d) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.
3. A heater is a sort of dual treap, in which the priorities of the nodes are given, but their search keys are generated independently and uniformly from the unit interval $[0,1]$. You can assume all priorities and keys are distinct. Describe algorithms to perform the operations INSERT and DELETEMIN in a heater. What are the expected worst-case running times of your algorithms? In particular, can you express the expected running time of INSERT in terms of the priority rank of the newly inserted item?

CS 473: Undergraduate Algorithms, Spring 2009

HBS 5

1. Recall that the staircase of a set of points consists of the points with no other point both above and to the right. Describe a method to maintain the staircase as new points are added to the set. Specifically, describe and analyze a data structure that stores the staircase of a set of points, and an algorithm $INSERT(x, y)$ that adds the point (x, y) to the set and returns *TRUE* or *FALSE* to indicate whether the staircase has changed. Your data structure should use $O(n)$ space, and your $INSERT$ algorithm should run in $O(\log n)$ amortized time.
2. In some applications, we do not know in advance how much space we will require. So, we start the program by allocating a (dynamic) table of some fixed size. Later, as new objects are inserted, we may have to allocate a larger table and copy the previous table to it. So, we may need more than $O(1)$ time for copying. In addition, we want to keep the table size small enough, avoiding a very large table to keep only few items. One way to manage a dynamic table is by the following rules:
 - (a) Double the size of the table if an item is inserted into a full table
 - (b) Halve the table size if a deletion causes the table to become less than 1/4 fullShow that, in such a dynamic table we only need $O(1)$ amortized time, per operation.
3. Consider a stack data structure with the following operations:
 - $PUSH(x)$: adds the element x to the top of the stack
 - POP : removes and returns the element that is currently on top of the stack (if the stack is non-empty)
 - $SEARCH(x)$: repeatedly removes the element on top of the stack until x is found or the stack becomes empty

What is the amortized cost of an operation?

CS 473: Undergraduate Algorithms, Spring 2009

HBS 6

1. Let G be a connected graph and let v be a vertex in G . Show that T is both a DFS tree and a BFS tree rooted at v , then $G = T$.
2. An Euler tour of a graph G is a walk that starts from a vertex v , visits every edge of G exactly once and gets back to v . Prove that G has an Euler tour if and only if all the vertices of G has even degrees. Can you give an efficient algorithm to find an Euler tour of such a graph.
3. You are helping a group of ethnographers analyze some oral history data they have collected by interviewing members of a village to learn about the lives of people lived there over the last two hundred years. From the interviews, you have learned about a set of people, all now deceased, whom we will denote P_1, P_2, \dots, P_n . The ethnographers have collected several facts about the lifespans of these people, of one of the following forms:
 - (a) P_i died before P_j was born.
 - (b) P_i and P_j were both alive at some moment.

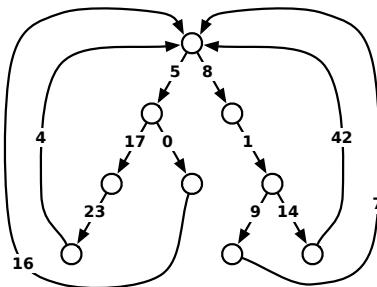
Naturally, the ethnographers are not sure that their facts are correct; memories are not so good, and all this information was passed down by word of mouth. So they'd like you to determine whether the data they have collected is at least internally consistent, in the sense that there could have existed a set of people for which all the facts they have learned simultaneously hold.

Describe and analyze an algorithm to answer the ethnographers' problem. Your algorithm should either output possible dates of birth and death that are consistent with all the stated facts, or it should report correctly that no such dates exist.

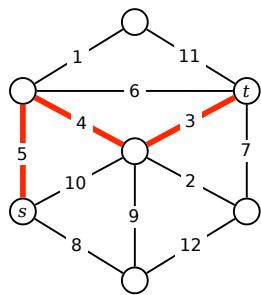
CS 473: Undergraduate Algorithms, Spring 2009

HBS 6.5

1. (a) Describe and analyze an algorithm to find the *second smallest spanning tree* of a given graph G , that is, the spanning tree of G with smallest total weight except for the minimum spanning tree.
- *(b) Describe and analyze an efficient algorithm to compute, given a weighted undirected graph G and an integer k , the k smallest spanning trees of G .
2. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight.



- (a) How much time would Dijkstra's algorithm require to compute the shortest path between two vertices u and v in a looped tree with n nodes?
- (b) Describe and analyze a faster algorithm.
3. Consider a path between two vertices s and t in an undirected weighted graph G . The *bottleneck length* of this path is the maximum weight of any edge in the path. The *bottleneck distance* between s and t is the minimum bottleneck length of any path from s to t . (If there are no paths from s to t , the bottleneck distance between s and t is ∞ .)



The bottleneck distance between s and t is 5.

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

CS 473: Undergraduate Algorithms, Spring 2009

HBS 6.55

1. Suppose you are given a directed graph $G = (V, E)$ with non-negative edge lengths; $\ell(e)$ is the length of $e \in E$. You are interested in the shortest path distance between two given locations/nodes s and t . It has been noticed that the existing shortest path distance between s and t in G is not satisfactory and there is a proposal to add exactly one edge to the graph to improve the situation. The candidate edges from which one has to be chosen is given by $E' = \{e_1, e_2, \dots, e_k\}$ and you can assume that $E \cup E' = \emptyset$. The length of the e_i is $\alpha_i \geq 0$. Your goal is figure out which of these k edges will result in the most reduction in the shortest path distance from s to t . Describe an algorithm for this problem that runs in time $O((m + n) \log n + k)$ where $m = |E|$ and $n = |V|$. Note that one can easily solve this problem in $O(k(m + n) \log n)$ by running Dijkstra's algorithm k times, one for each G_i where G_i is the graph obtained by adding e_i to G .
2. Let G be an undirected graph with non-negative edge weights. Let s and t be two vertices such that the shortest path between s and t in G contains all the vertices in the graph. For each edge e , let $G \setminus e$ be the graph obtained from G by deleting the edge e . Design an $O(E \log V)$ algorithm that finds the shortest path distance between s and t in $G \setminus e$ for all e . [Note that you need to output E distances, one for each graph $G \setminus e$]
3. Given a Directed Acyclic Graph (DAG) and two vertices s and t you want to determine if there is an s to t path that includes at least k vertices.

CS 473: Undergraduate Algorithms, Spring 2009

HBS 7

1. Let $G = (V, E)$ be a directed graph with non-negative capacities. Give an efficient algorithm to check whether there is a unique max-flow on G ?
2. Let $G = (V, E)$ be a graph and $s, t \in V$ be two specific vertices of G . We call $(S, T = V \setminus S)$ an (s, t) -cut if $s \in S$ and $t \in T$. Moreover, it is a minimum cut if the sum of the capacities of the edges that have one endpoint in S and one endpoint in T equals the maximum (s, t) -flow. Show that, both intersection and union of two min-cuts is a min-cut itself.
3. Let $G = (V, E)$ be a graph. For each edge e let $d(e)$ be a demand value attached to it. A flow is feasible if it sends more than $d(e)$ through e . Assume you have an oracle that is capable of solving the maximum flow problem. Give efficient algorithms for the following problems that call the oracle at most once.
 - (a) Find a feasible flow.
 - (b) Find a feasible flow of minimum possible value.

CS 473: Undergraduate Algorithms, Spring 2009

HBS 8

1. A box i can be specified by the values of its sides, say (i_1, i_2, i_3) . We know all the side lengths are larger than 10 and smaller than 20 (i.e. $10 < i_1, i_2, i_3 < 20$). Geometrically, you know what it means for one box to nest in another: It is possible if you can rotate the smaller so that it fits inside the larger in each dimension. Of course, nesting is recursive, that is if i nests in j and j nests in k then i nests in k . After doing some nesting operations, we say a box is visible if it is not nested in any other one. Given a set of boxes (each specified by the lengths of their sides) the goal is to find a set of nesting operations to minimize the number of visible boxes. Design and analyze an efficient algorithm to do this.

2. Let the number of papers submitted to a conference be n and the number of available reviewers be m . Each reviewer has a list of papers that he/she can review and each paper should be reviewed by three different persons. Also, each reviewer can review at most 5 papers. Design and analyze an algorithm to make the assignment or decide no feasible assignment exists.

3. Back in the euphoric early days of the Web, people liked to claim that much of the enormous potential in a company like Yahoo! was in the "eyeballs" - the simple fact that it gets millions of people looking at its pages every day. And further, by convincing people to register personal data with the site, it can show each user an extremely targeted advertisement whenever he or she visits the site, in away that TV networks or magazines could not hope to match. So if the user has told Yahoo! that he is a 20-year old computer science major from Cornell University, the site can throw up a banner ad for apartments in Ithaca, NY; on the other hand, if he is a 50-year-old investment banker from Greenwich, Connecticut, the site can display a banner ad pitching Lincoln Town Cars instead.

But deciding on which ads to show to which people involves some serious computation behind the scenes. Suppose that the managers of a popular Web site have identified k distinct demographic groups G_1, G_2, \dots, G_k . (These groups can overlap; for example G_1 can be equal to all residents of New York State, and G_2 can be equal to all people with a degree in computer science.) The site has contracts with m different advertisers, to show a certain number of copies of their ads to users of the site. Here is what the contract with the i^{th} advertiser looks like:

- (a) For a subset $X_i \subset \{G_1, \dots, G_k\}$ of the demographic groups, advertiser i wants its ads shown only to users who belong to at least one of the demographic groups in the set X_i
- (b) For a number r_i , advertiser i wants its ads shown to at least r_i users each minute.

Now, consider the problem of designing a good advertising policy - a way to show a single ad to each user of the site. Suppose at a given minute, there are n users visiting the site. Because we have registration information on each of these users, we know that user j (for $j = 1, 2, \dots, n$) belongs to a subset $U_j \subset \{G_1, \dots, G_k\}$ of the demographic groups. The problem is: is there a way to show a single ad to each user so that the site's contracts with each of the m advertisers is satisfied for this minute? (That is, for each $i = 1, 2, \dots, m$, at least r_i of the n users, each belonging to at least one demographic group in X_i , are shown an ad provided by advertiser i .)

Give an efficient algorithm to decide if this is possible, and if so, to actually choose an ad to show each user.

CS 473: Undergraduate Algorithms, Spring 2009

HBS 9

1. Prove that any algorithm to merge two sorted arrays, each of size n , requires at least $2n - 1$ comparisons.
2. Suppose you want to determine the largest number in an n -element set $X = \{x_1, x_2, \dots, x_n\}$, where each element x_i is an integer between 1 and $2^m - 1$. Describe an algorithm that solves this problem in $O(n + m)$ steps, where at each step, your algorithm compares one of the elements x_i with a constant. In particular, your algorithm must never actually compare two elements of X !
[Hint: Construct and maintain a nested set of ‘pinning intervals’ for the numbers that you have not yet removed from consideration, where each interval but the largest is either the upper half or lower half of the next larger block.]
3. Let P be a set of n points in the plane. The staircase of P is the set of all points in the plane that have at least one point in P both above and to the right. Prove that computing the staircase requires at least $\Omega(n \log n)$ comparisons in two ways,
 - (a) Reduction from sorting.
 - (b) Directly.

You have 90 minutes to answer four of the five questions.
Write your answers in the separate answer booklet.
 You may take the question sheet with you when you leave.

1. Each of these ten questions has one of the following five answers:

A: $\Theta(1)$ B: $\Theta(\log n)$ C: $\Theta(n)$ D: $\Theta(n \log n)$ E: $\Theta(n^2)$

Choose the correct answer for each question. Each correct answer is worth +1 point; each incorrect answer is worth $-1/2$ point; each "I don't know" is worth $+1/4$ point. Your score will be rounded to the nearest *non-negative* integer.

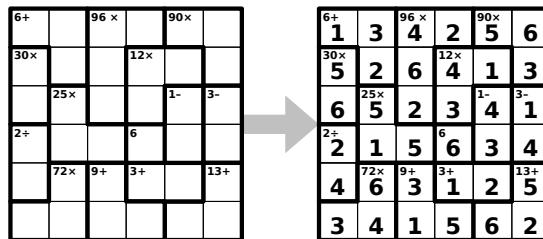
- (a) What is $\sum_{i=1}^n \frac{n}{i}$?
- (b) What is $\sqrt{\sum_{i=1}^n i}$?
- (c) How many digits are required to write 3^n in decimal?
- (d) What is the solution to the recurrence $D(n) = D(n/\pi) + \sqrt{2}$?
- (e) What is the solution to the recurrence $E(n) = E(n - \sqrt{2}) + \pi$?
- (f) What is the solution to the recurrence $F(n) = 4F(n/2) + 3n$?
- (g) What is the worst-case time to search for an item in a binary search tree?
- (h) What is the worst-case running time of quicksort?
- (i) Let $H[1..n, 1..n]$ be a fixed array of numbers. Consider the following recursive function:

$$Glub(i, j) = \begin{cases} 0 & \text{if } i = 0 \\ \infty & \text{if } i > n \text{ or } j = 0 \\ \max \{Glub(i - 1, j), H[i, j] + Glub(i + 1, j - 1)\} & \text{otherwise} \end{cases}$$

How long does it take to compute $Glub(n, n)$ using dynamic programming?

- (j) What is the running time of the fastest possible algorithm to solve KenKen puzzles?

A KenKen puzzle is a 6×6 grid, divided into regions called *cages*. Each cage is labeled with a numerical *value* and an arithmetic *operation*: $+$, $-$, \times , or \div . (The operation can be omitted if the cage consists of a single cell.) The goal is to place an integer between 1 and 6 in each grid cell, so that no number appears twice in any row or column, and the numbers inside each cage can be combined using *only* that cage's operation to obtain that cage's value. The solution is guaranteed to be unique.

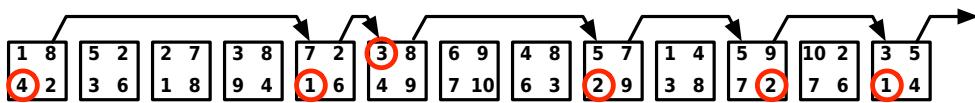


		96 ×	90 ×		
6+					
		12x			
30x					
25x			1-	3-	
2+		6			
2+ 72x	9+	3+	13+		

1	3	4	2	5	6
5	2	6	4	1	3
6	5	2	3	4	1
2	1	5	6	3	4
4	6	3	1	2	5
3	4	1	5	6	2

A Kenken puzzle and its solution

2. (a) Suppose $A[1..n]$ is an array of n distinct integers, sorted so that $A[1] < A[2] < \dots < A[n]$. Each integer $A[i]$ could be positive, negative, or zero. Describe an efficient algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. An algorithm that runs in $\Theta(n)$ time is worth at most 3 points.
- (b) Now suppose $A[1..n]$ is a sorted array of n distinct **positive** integers. Describe an even faster algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. [Hint: This is **really** easy!]
3. *Moby Selene* is a solitaire game played on a row of n squares. Each square contains four positive integers. The player begins by placing a token on the leftmost square. On each move, the player chooses one of the numbers on the token's current square, and then moves the token that number of squares to the right. The game ends when the token moves past the rightmost square. The object of the game is to make as many moves as possible before the game ends.



A Moby Selene puzzle that allows six moves. (This is **not** the longest legal sequence of moves.)

- (a) **Prove** that the obvious greedy strategy (always choose the smallest number) does not give the largest possible number of moves for every Moby Selene puzzle.
- (b) Describe and analyze an efficient algorithm to find the largest possible number of legal moves for a given Moby Selene puzzle.
4. Consider the following algorithm for finding the largest element in an unsorted array:

<pre> RANDOMMAX($A[1..n]$): $max \leftarrow \infty$ for $i \leftarrow 1$ to n in random order if $A[i] > max$ $max \leftarrow A[i]$ (*)</pre>
--

- (a) In the worst case, how many times does RANDOMMAX execute line (*)?
- (b) What is the **exact** probability that line (*) is executed during the last iteration of the for loop?
- (c) What is the **exact** expected number of executions of line (*)? (A correct $\Theta()$ bound is worth half credit.)
5. *This question is taken directly from HBS 0.* Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

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

You have 90 minutes to answer four of the five questions.
Write your answers in the separate answer booklet.
You may take the question sheet with you when you leave.

1. Recall that a *tree* is a connected graph with no cycles. A graph is *bipartite* if we can color its vertices black and white, so that every edge connects a white vertex to a black vertex.
 - (a) **Prove** that every tree is bipartite.
 - (b) Describe and analyze a fast algorithm to determine whether a given graph is bipartite.
2. Describe and analyze an algorithm $\text{SHUFFLE}(A[1..n])$ that randomly permutes the input array A , so that each of the $n!$ possible permutations is equally likely. You can assume the existence of a subroutine $\text{RANDOM}(k)$ that returns a random integer chosen uniformly between 1 and k in $O(1)$ time. For full credit, your SHUFFLE algorithm should run in $O(n)$ time. [Hint: This problem appeared in HBS 3½.]
3. Let G be an undirected graph with weighted edges.
 - (a) Describe and analyze an algorithm to compute the *maximum weight spanning tree* of G .
 - (b) A *feedback edge set* of G is a subset F of the edges such that every cycle in G contains at least one edge in F . In other words, removing every edge in F makes G acyclic. Describe and analyze a fast algorithm to compute the minimum weight feedback edge set of G .

[Hint: Don't reinvent the wheel!]
4. Let $G = (V, E)$ be a connected directed graph with non-negative edge weights, let s and t be vertices of G , and let H be a subgraph of G obtained by deleting some edges. Suppose we want to reinsert exactly one edge from G back into H , so that the shortest path from s to t in the resulting graph is as short as possible. Describe and analyze an algorithm to choose the best edge to reinsert. For full credit, your algorithm should run in $O(E \log V)$ time. [Hint: This problem appeared in HBS 6¾.]
5. Describe and analyze an efficient data structure to support the following operations on an array $X[1..n]$ as quickly as possible. Initially, $X[i] = 0$ for all i .
 - Given an index i such that $X[i] = 0$, set $X[i]$ to 1.
 - Given an index i , return $X[i]$.
 - Given an index i , return the smallest index $j \geq i$ such that $X[j] = 0$, or report that no such index exists.

For full credit, the first two operations should run in *worst-case constant* time, and the amortized cost of the third operation should be as small as possible.

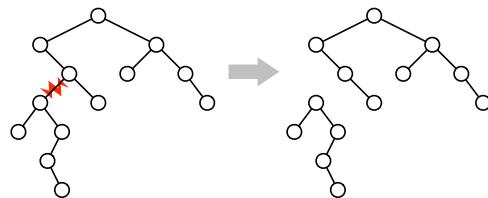
You have 180 minutes to answer six of the seven questions.
 Write your answers in the separate answer booklet.
 You may take the question sheet with you when you leave.

1. SUBSETSUM and PARTITION are two closely related NP-hard problems, defined as follows.

SUBSETSUM: Given a set X of positive integers and a positive integer k , does X have a subset whose elements sum up to k ?

PARTITION: Given a set Y of positive integers, can Y be partitioned into two subsets whose sums are equal?

- (a) [2 pts] **Prove** that PARTITION and SUBSETSUM are both in NP.
 - (b) [1 pt] Suppose you already know that SUBSETSUM is NP-hard. Which of the following arguments could you use to prove that PARTITION is NP-hard? **You do not need to justify your answer** — just answer ① or ②.
 - ① Given a set X and an integer k , construct a set Y in polynomial time, such that $\text{PARTITION}(Y)$ is true if and only if $\text{SUBSETSUM}(X, k)$ is true.
 - ② Given a set Y , construct a set X and an integer k in polynomial time, such that $\text{PARTITION}(Y)$ is true if and only if $\text{SUBSETSUM}(X, k)$ is true.
 - (c) [3 pts] Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM. **You do not need to prove that your reduction is correct.**
 - (d) [4 pts] Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION. **You do not need to prove that your reduction is correct.**
2. (a) [4 pts] For any node v in a binary tree, let $\text{size}(v)$ denote the number of nodes in the subtree rooted at v . Let k be an arbitrary positive number. **Prove** that every binary tree with at least k nodes contains a node v such that $k \leq \text{size}(v) \leq 2k$.
- (b) [2 pts] Removing any edge from an n -node binary tree T separates it into two smaller binary trees. An edge is called a **balanced separator** if both of these subtrees have at least $n/3$ nodes (and therefore at most $2n/3$ nodes). **Prove** that every binary tree with more than one node has a balanced separator. [Hint: Use part (a).]
- (c) [4 pts] Describe and analyze an algorithm to find a balanced separator in a given binary tree. [Hint: Use part (a).]



Removing a balanced separator from a binary tree.

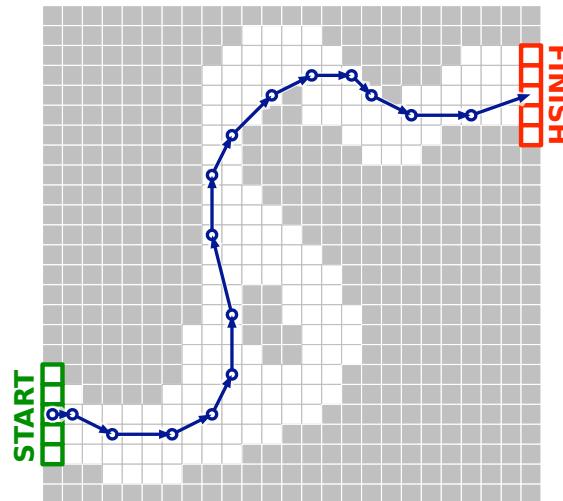
3. **Racetrack** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade.¹ The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

Each car has a *position* and a *velocity*, both with integer *x*- and *y*-coordinates. The initial position is a point on the starting line, chosen by the player; the initial velocity is always $(0, 0)$. At each step, the player optionally increments or decrements either or both coordinates of the car's velocity; in other words, each component of the velocity can change by at most 1 in a single step. The car's new position is then determined by adding the new velocity to the car's previous position. The new position must be inside the track; otherwise, the car crashes and that player loses the race. The race ends when the first car reaches a position on the finish line.

Suppose the racetrack is represented by an $n \times n$ array of bits, where each 0 bit represents a grid point inside the track, each 1 bit represents a grid point outside the track, the 'starting line' is the first column, and the 'finish line' is the last column.

Describe and analyze an algorithm to find the minimum number of steps required to move a car from the starting line to the finish line of a given racetrack. [Hint: Build a graph. What are the vertices? What are the edges? What problem is this?]

velocity	position
$(0, 0)$	$(1, 5)$
$(1, 0)$	$(2, 5)$
$(2, -1)$	$(4, 4)$
$(3, 0)$	$(7, 4)$
$(2, 1)$	$(9, 5)$
$(1, 2)$	$(10, 7)$
$(0, 3)$	$(10, 10)$
$(-1, 4)$	$(9, 14)$
$(0, 3)$	$(9, 17)$
$(1, 2)$	$(10, 19)$
$(2, 2)$	$(12, 21)$
$(2, 1)$	$(14, 22)$
$(2, 0)$	$(16, 22)$
$(1, -1)$	$(17, 21)$
$(2, -1)$	$(19, 20)$
$(3, 0)$	$(22, 20)$
$(3, 1)$	$(25, 21)$



A 16-step Racetrack run, on a 25×25 track.

4. A palindrome is any string that is exactly the same as its reversal, like I, or DEED, or RACECAR, or AMANAPLANACATACANALPANAMA. Describe and analyze an algorithm to find the length of the longest *subsequence* of a given string that is also a palindrome.

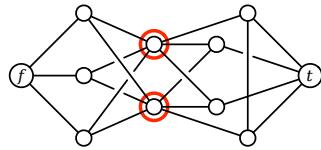
For example, the longest palindrome subsequence of MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM is MHYMRORMYHM, so given that string as input, your algorithm should output the number 11.

¹The actual game is a bit more complicated than the version described here.

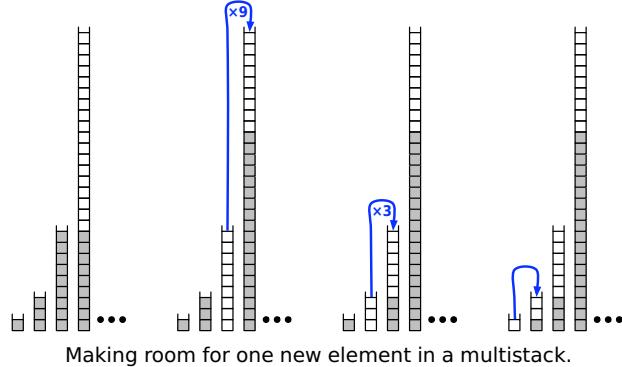
5. The Island of Sodor is home to a large number of towns and villages, connected by an extensive rail network. Recently, several cases of a deadly contagious disease (either swine flu or zombies; reports are unclear) have been reported in the village of Ffarquhar. The controller of the Sodor railway plans to close down certain railway stations to prevent the disease from spreading to Tidmouth, his home town. No trains can pass through a closed station. To minimize expense (and public notice), he wants to close down as few stations as possible. However, he cannot close the Ffarquhar station, because that would expose him to the disease, and he cannot close the Tidmouth station, because then he couldn't visit his favorite pub.

Describe and analyze an algorithm to find the minimum number of stations that must be closed to block all rail travel from Ffarquhar to Tidmouth. The Sodor rail network is represented by an undirected graph, with a vertex for each station and an edge for each rail connection between two stations. Two special vertices f and t represent the stations in Ffarquhar and Tidmouth.

For example, given the following input graph, your algorithm should return the number 2.



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



- (a) In the worst case, how long does it take to push one more element onto a mystack 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 mystack.
7. Recall the problem 3COLOR: Given a graph, can we color each vertex with one of 3 colors, so that every edge touches two different colors? We proved in class that 3COLOR is NP-hard.

Now consider the related problem 12COLOR: Given a graph, can we color each vertex with one of twelve colors, so that every edge touches two different colors? **Prove** that 12COLOR is NP-hard.

You may assume the following problems are NP-hard:

CIRCUITSAT: Given a boolean circuit, are there any input values that make the circuit output True?

PLANAR CIRCUITSAT: Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

3SAT: Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

MAXINDEPENDENTSET: Given an undirected graph G , what is the size of the largest subset of vertices in G that have no edges among them?

MAXCLIQUE: Given an undirected graph G , what is the size of the largest complete subgraph of G ?

MINVERTEXCOVER: Given an undirected graph G , what is the size of the smallest subset of vertices that touch every edge in G ?

MINSETCOVER: Given a collection of subsets S_1, S_2, \dots, S_m of a set S , what is the size of the smallest subcollection whose union is S ?

MINHITTINGSET: Given a collection of subsets S_1, S_2, \dots, S_m of a set S , what is the size of the smallest subset of S that intersects every subset S_i ?

3COLOR: Given an undirected graph G , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

HAMILTONIANCYCLE: Given a graph G , can is there a cycle in G that visits every vertex once?

HAMILTONIANPATH: Given a graph G , can is there a path in G that visits every vertex once?

DOUBLEHAMILTONIANCYCLE: Given a graph G , can is there a closed walk in G that visits every vertex twice?

DOUBLEHAMILTONIANPATH: Given a graph G , can is there an open walk in G that visits every vertex twice?

MINDEGREESPANNINGTREE: Given an undirected graph G , what is the minimum degree of any spanning tree of G ?

MINLEAVESPANNINGTREE: Given an undirected graph G , what is the minimum number of leaves in any spanning tree of G ?

TRAVELINGSalesMAN: Given a graph G with weighted edges, what is the minimum cost of any Hamiltonian path/cycle in G ?

LONGESTPATH: Given a graph G with weighted edges and two vertices s and t , what is the length of the longest *simple* path from s to t in G ?

SUBSETSUM: Given a set X of positive integers and an integer k , does X have a subset whose elements sum to k ?

PARTITION: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3PARTITION: Given a set X of n positive integers, can X be partitioned into $n/3$ three-element subsets, all with the same sum?

MINESWEEPER: Given a Minesweeper configuration and a particular square x , is it safe to click on x ?

TETRIS: Given a sequence of N Tetris pieces and a partially filled $n \times k$ board, is it possible to play every piece in the sequence without overflowing the board?

SUDOKU: Given an $n \times n$ Sudoku puzzle, does it have a solution?

KENKEN: Given an $n \times n$ Ken-Ken puzzle, does it have a solution?