

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:

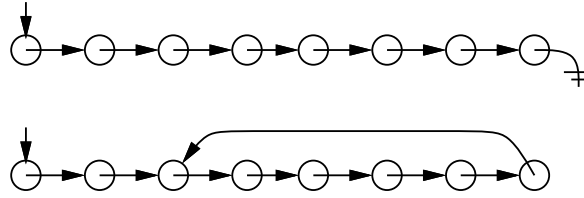
$$\begin{array}{ccccc} 1 & n & n^2 & \lg n & \lg(n \lg n) \\ \lg^* n & \lg^* 2^n & 2^{\lg^* n} & \lg \lg^* n & \lg^* \lg n \\ n^{\lg n} & (\lg n)^n & (\lg n)^{\lg n} & n^{1/\lg n} & n^{\lg \lg n} \\ \log_{1000} n & \lg^{1000} n & \lg^{(1000)} n & \left(1 + \frac{1}{n}\right)^n & n^{1/1000} \end{array}$$

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.

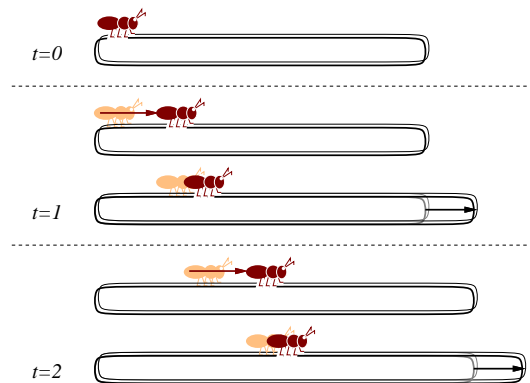


Top: A standard singly-linked list. Bottom: A corrupted singly linked list.

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?

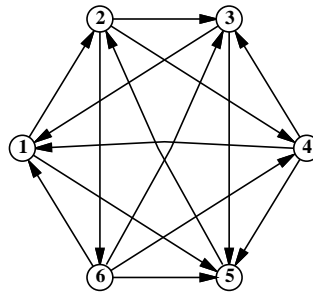
²Since you've read the Homework Instructions, you know what the phrase "describe an algorithm" means. Right?

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).

- 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 .

- 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.



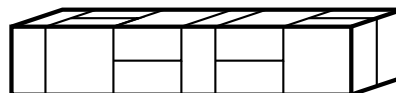
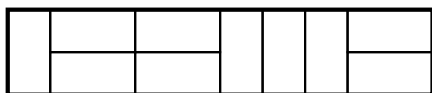
A six-vertex tournament containing the Hamiltonian path $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$.

- (a) Prove the following identity by induction:

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

- 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 $2^{2^{\lfloor \lg n \rfloor}} = \Theta(2^{2^{\lceil \lg n \rceil}})$? 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.

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

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

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.

```

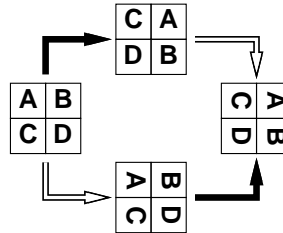
RANDOMSELECT( $A[1..n], r$ ):
   $p \leftarrow \text{RANDOM}(1, p)$ 
   $k \leftarrow \text{PARTITION}(A[1..n], p)$ 
  if  $r < k$ 
    return RANDOMSELECT( $A[1..k-1], r$ )
  else if  $r > k$ 
    return RANDOMSELECT( $A[k+1..n], r-k$ )
  else
    return  $A[k]$ 

```

- (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?

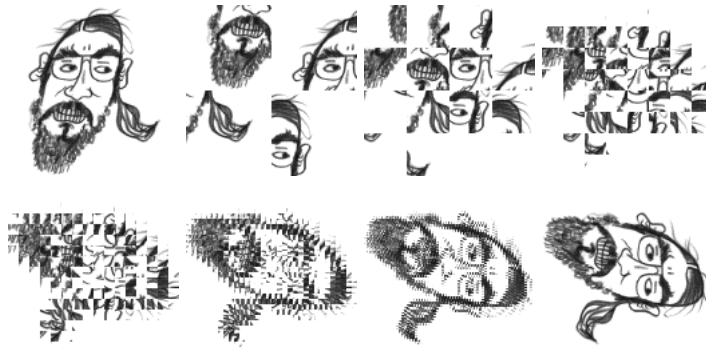
4. Some graphics hardware includes support for an operation called *blit*, or **block transfer**, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

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 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 $container[i]$ 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 $container[i]$ 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 $container[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound.)
- (c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and 2^i ounces for each $container[i]$. Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang BARLEYMOW(n)? (Give an *exact* answer, not just an asymptotic bound.)

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 hemisemidemiyoctapint, 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.
- (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 obvious 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.
 - (d) Describe a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.
 - (e) 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.
- (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, 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 $key[x]$.

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.

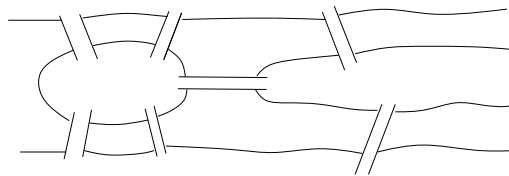
Required Problems

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

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

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 $5^{2^{29}5^0 + 23^41 + 17^32 + 11^23 + 5^14} \pmod{6}$?
- (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 2000

Homework 4 (due October 26, 2000 at midnight)

Name:		
Net ID:	Alias:	U ³ / ₄ 1

Name:		
Net ID:	Alias:	U ³ / ₄ 1

Name:		
Net ID:	Alias:	U 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, 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:

$$F_1 = b$$

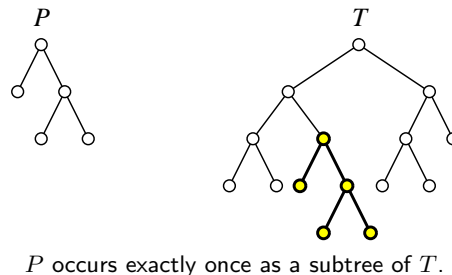
$$F_2 = a$$

$$F_n = F_{n-1}F_{n-2} \quad \text{for all } n > 2$$

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 $j \rightarrow fail[j] \rightarrow fail[fail[j]] \rightarrow \dots$ 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) 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) 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.]



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.

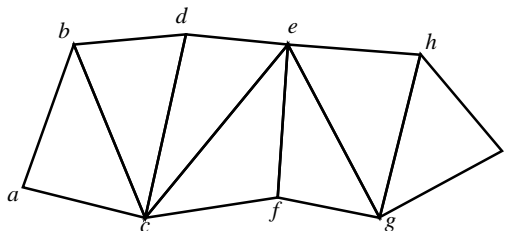
Required Problems

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?

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 $(a, b, c), (b, c, d), (d, c, e), (c, e, f), (f, e, g), (g, h, i)$.



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

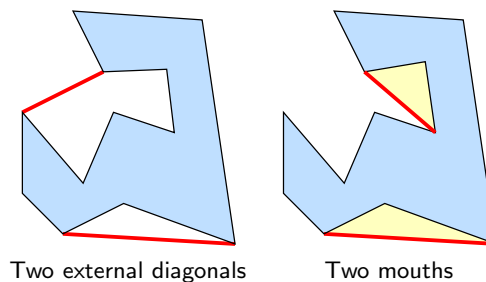
Give an $O(n \log n)$ algorithm to test whether any two circles in a set of size n intersect.

3. Basic polygon computations (assume *exact* arithmetic)
 - (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.

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.
 - (a) 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:	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

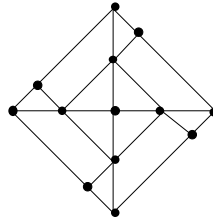
- Prove that $P \subseteq \text{co-NP}$
 - Show that if $\text{NP} \neq \text{co-NP}$, then *no* NP-complete problem is a member of co-NP
- 2SAT is a special case of the formula satisfiability problem, where the input formula is in conjunctive normal form and every clause has at most *two* literals. Prove that 2SAT is in P
- 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.

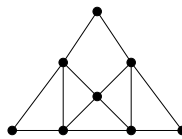
Practice Problems

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 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 $T(n) = 3T(\lceil \frac{n+51}{3} \rceil) + 17n - \sqrt[3]{\lg \lg n} - 2^{2^{\log^* n}} + \pi$?

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

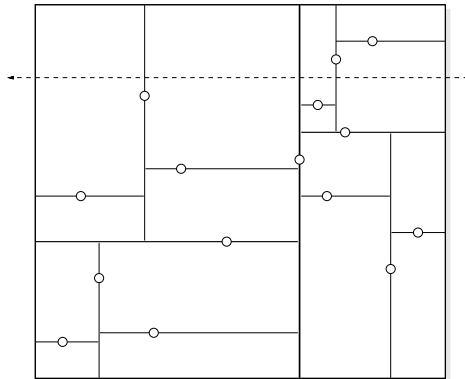
The expected time for inserting one item into an n -node randomized treap is $O(\log n)$.
What is the worst-case time for a sequence of n insertions into an initially empty treap?

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*.

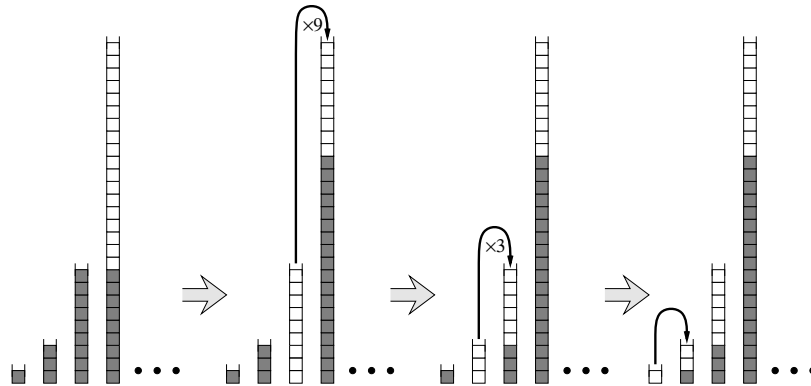


A kd-tree for 15 points. The dashed line crosses four cells.

- (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.



Making room for one new element in a multistack.

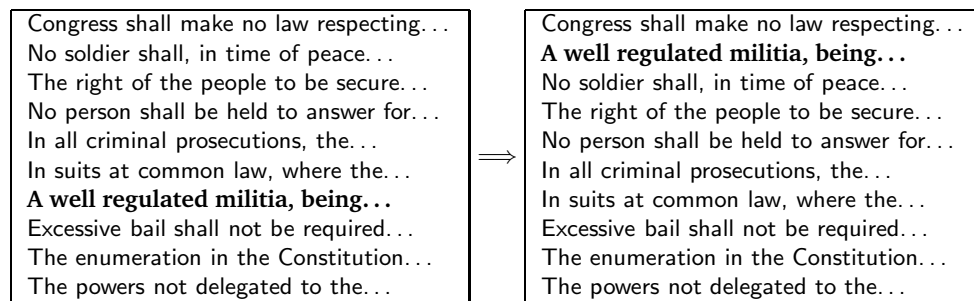
- (a) [1 point] In the worst case, how long does it take to push one more element onto a multistack containing n elements?
- (b) [9 points] Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack. You can use any method you like.

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.

- $\text{COMPAREPARAGRAPHS}(i, j)$ 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!
- $\text{MOVEPARAGRAPH}(i, j)$ ‘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 $\text{MOVEPARAGRAPH}(7, 2)$:



[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 $\text{RANDOM}(i, j)$ 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
RECURSION 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 numerological 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:

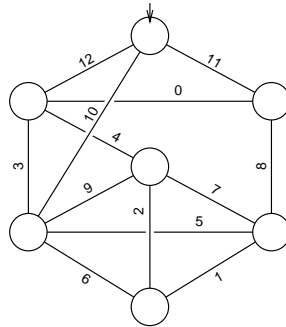
$373 \text{ (base 14)} = 689 \text{ (base 10)}$

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:

$689 - 23 = 666$

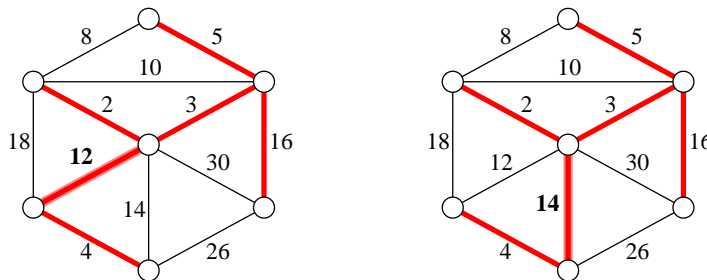
QED.

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!

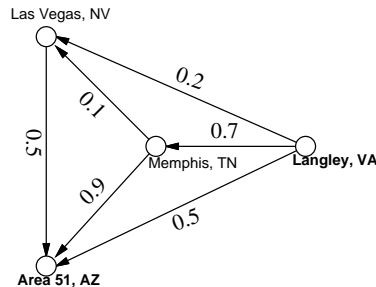


The minimum spanning tree and the second-minimum spanning tree of a graph.

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 `flippityfloppitydongdong`. 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) $\text{NP} \neq \text{co-NP}$

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.

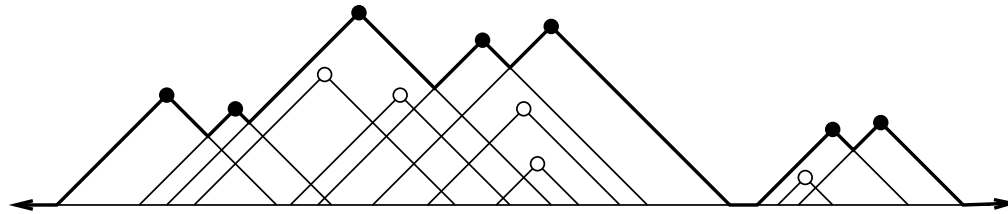
<i>G</i>	<i>H</i>	<i>R</i>	<i>R</i>	<i>G</i>	<i>G</i>	<i>R</i>	<i>G</i>	<i>H</i>	<i>H</i>	<i>R</i>	<i>S</i>	<i>R</i>	<i>R</i>	<i>H</i>	<i>G</i>	<i>S</i>	<i>H</i>	<i>G</i>	<i>G</i>
Harry	Ann	Bob	Tina	Chad	Bill	Lisa	Ekta	Bart	Jim	John	Jeff	Liz	Mary	Dawn	Nick	Kim	Fox	Dana	Mel

↓

<i>G</i>	<i>G</i>	<i>G</i>	<i>G</i>	<i>G</i>	<i>G</i>	<i>G</i>	<i>H</i>	<i>H</i>	<i>H</i>	<i>H</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>R</i>	<i>S</i>	<i>S</i>	
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 an 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).



The skyline formed by these 12 pyramids has 16 vertices.

- 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.]
- 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.

- Show that DNF-SAT is in P.
- 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 two 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.