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.

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

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) $B(n) = \min_{0<k<n} \big(B(k) + B(n-k) + 1\big)$.

   (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?]*

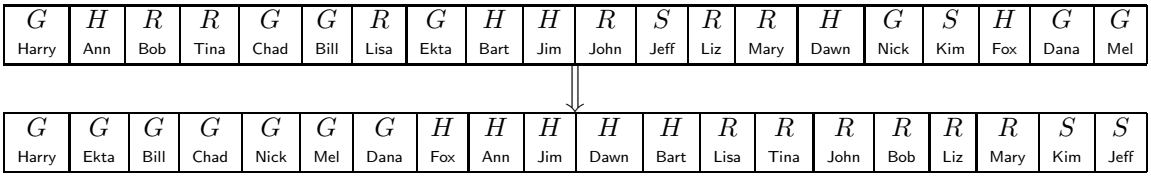
   (c) Suppose $\gamma_0 = p$, $\gamma_1 = q$, and $\gamma_n = (\gamma_{n-2} \Rightarrow \gamma_{n-1})$ for all $n \geq 2$. Simplify γ_n as much as possible. *[Hint: What is γ_5?]*

   (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 algorithm[1] that rearranges the items into four clusters in $O(n)$ time using only $O(1)$ extra space.



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!

[1]Since 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) $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$

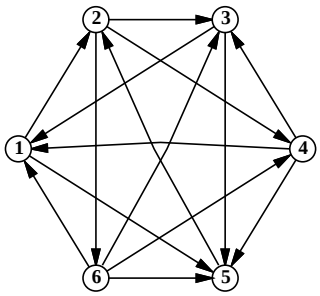
   ★(d) If n is an integer multiple of m, then F_n is an integer multiple of F_m.

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

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) $C(n) = n + \frac{1}{2}\big(C(n-1) + C(3n/4)\big)$

   (d) $D(n) = \max_{n/3<k<2n/3} \big(D(k) + D(n-k) + n\big)$

   ⋆(e) $E(n) = 2E(n/2) + n/\lg n$

   ⋆(f) $F(n) = \dfrac{F(n-1)}{F(n-2)}$, 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 $F(0) = F(1) = 1$

   ⋆(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: $2^{\lfloor \lg n \rfloor} = \Theta\big(2^{\lceil \lg n \rceil}\big)$.

   (c) Prove or disprove: $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta\big(2^{2^{\lceil \lg \lg n \rceil}}\big)$.

   (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.

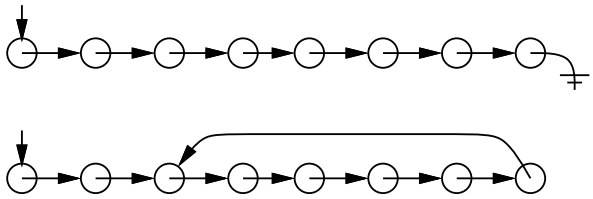
6. Evaluate the following summations; simplify your answers as much as possible. Significant partial credit will be given for answers in the form $\Theta(f(n))$ for some recognizable function $f(n)$.

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

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

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

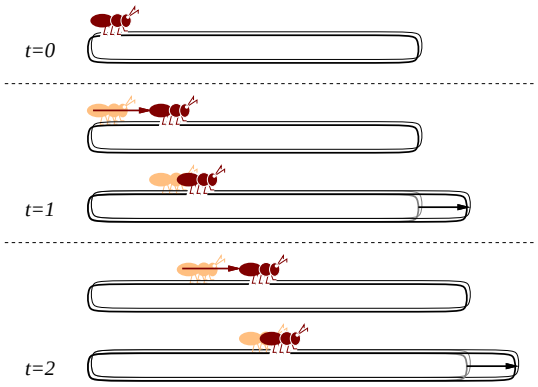
7. Suppose you have a pointer to the head of singly linked list. Normally, each node in the list only has a pointer to the next element, and the last node's pointer is NULL. Unfortunately, your list might have been corrupted by a bug in somebody else's code[2], so that the last node has a pointer back 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.

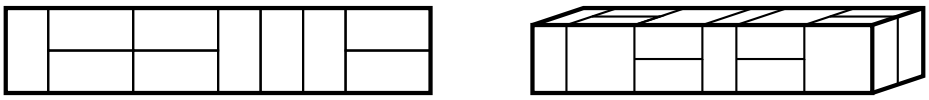
(a) How far has the ant moved after t seconds, as a function of n and t? Set up a recurrence and (for full credit) give an *exact* closed-form solution. [Hint: What *fraction* of the rubber band's length has the ant walked?]

(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.

[2] After 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 mod 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 ¾ 1

Name:		
Net ID:	Alias:	U ¾ 1

Name:		
Net ID:	Alias:	U ¾ 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, ¾, 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 $1 = c[1] < c[2] < \cdots < c[n]$. (In the U.S., for example, $n = 6$ and the values are $1, 5, 10, 25, 50$ and 100 cents.) Your beloved and belevolent 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] ↔ A[1]
  else if n > 2
      m ← ⌈2n/3⌉
      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 rth smallest element in an unsorted array $A[1..n]$. For example, to find the smallest element, you would call RandomSelect$(A, 1)$; to find the median element, you would call RandomSelect$(A, \lfloor n/2 \rfloor)$. Recall from lecture that Partition splits the array into three parts by comparing the pivot element $A[p]$ to every other element of the array, using $n-1$ comparisons altogether, and returns the new index of the pivot element.

```
RandomSelect(A[1..n], r):
  p ← Random(1, n)
  k ← 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 ith smallest and jth smallest elements in the input array? The correct answer is a simple function of i, j, and r. [Hint: Check your answer by trying a few small examples.]

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

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

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 pseudolyrics[1], where *container*$[i]$ is the name of a container that holds 2^i ounces of beer.[2]

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

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

  for i ← 1 to n
      "We'll drink it out of the container[i], boys,"
      "Here's a health to the barley-mow!"
      for j ← i downto 1
          "The container[j],"
      "And the jolly brown bowl!"
      "Here's a health to the barley-mow!"
      "Here's a health to the barley-mow, my brave boys,"
      "Here's a health to the barley-mow!"
```

(a) Suppose each container name *container*$[i]$ is a single word, and you can sing four words a second. How long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound.)

(b) If you want to sing this song for $n > 20$, you'll have to make up your own container names, and to avoid repetition, these names will get progressively longer as n increases[3]. Suppose *container*$[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound.)

(c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and 2^i ounces for each *container*$[i]$. Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang BARLEYMOW(n)? (Give an *exact* answer, not just an asymptotic bound.)

[1] Pseudolyrics are to lyrics as pseudocode is to code.

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

[3] "We'll drink it out of the hemisemidemiyottapint, boys!"

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 ith 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.)

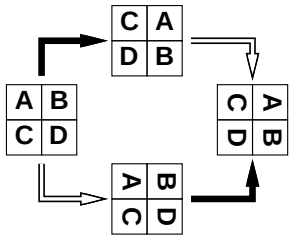
   (a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.

   (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?]

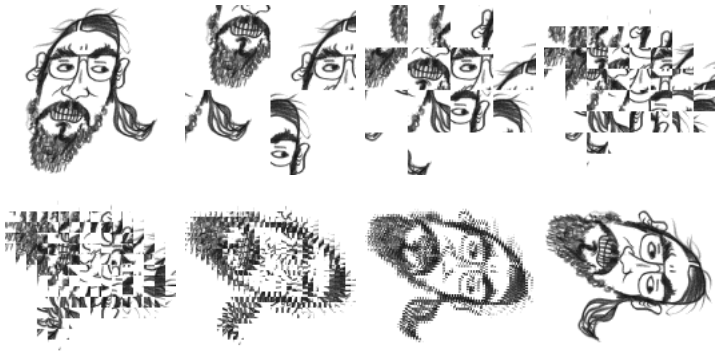
4. Some graphics hardware includes support for an operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

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



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

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



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

(a) Prove that both versions of the algorithm are correct. [Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]

(b) *Exactly* how many blits does the algorithm perform?

(c) What is the algorithm's running time if 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 kth 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 $1, 2, 4, 8, \ldots, 2^k$, $k \geq 1$. 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 ¾ 1

Name:		
Net ID:	Alias:	U ¾ 1

Name:		
Net ID:	Alias:	U ¾ 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, ¾, 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 kth smallest element in the union of A and B. (For example, if $k = 1$, your algorithm should return the smallest element of $A \cup B$; if $k = n$, our algorithm should return the median of $A \cup B$.) You can assume that the arrays contain no duplicates. 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 kth smallest item in an augmented scapegoat tree in $O(\log n)$ *worst-case* time.

   (c) Describe an algorithm SplaySelect(k) that selects the kth smallest item in an augmented splay tree in $O(\log n)$ *amortized* time.

(d) Describe an algorithm TREAPSELECT(k) that selects the kth 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.)

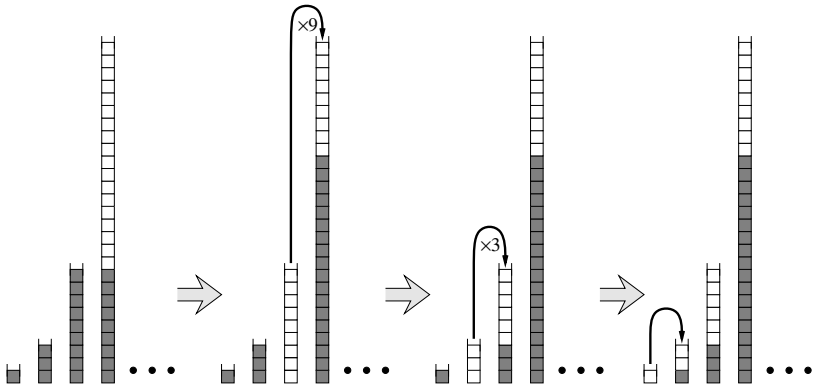
   *(c) Show that you can rebuild a fully balanced binary tree from an unbalanced tree in $O(n)$ time using only $O(\log n)$ additional memory.

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



Making room for one new element in a multistack.

   (a) *[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.

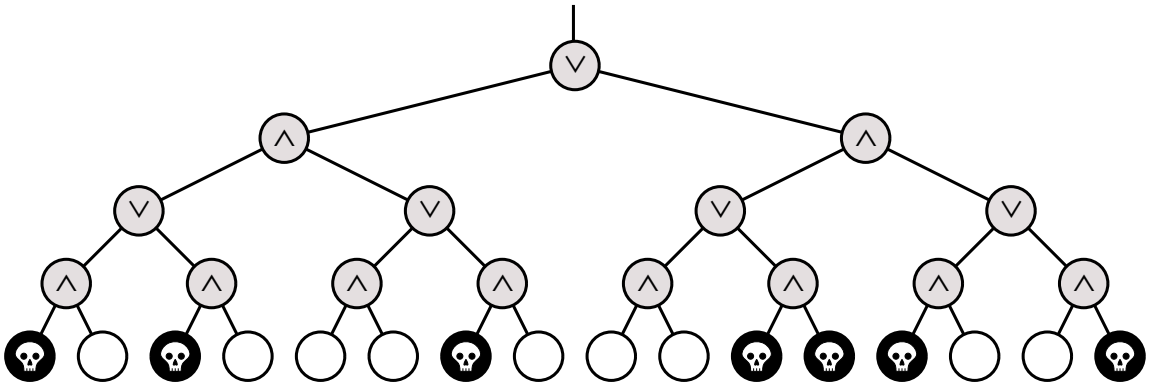
6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   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 ith least significant fit indicates whether the sum includes the ith Fibonacci number F_i. For example, the fit string 101110 represents the number $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$. Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string. 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 whet her 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 ith operation costs i if i is an exact power of 2, and 1 otherwise. That is operation i costs $f(i)$, where:

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

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

   (a) Aggregate method

   (b) Accounting method

   *(c) Potential method

CS 373: Combinatorial Algorithms, Spring 2001

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

Name:		
Net ID:	Alias:	U ¾ 1

Name:		
Net ID:	Alias:	U ¾ 1

Name:		
Net ID:	Alias:	U ¾ 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, ¾, 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.

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

   (b) Show that for $i = 1, 2, \ldots, n$, 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 $\Pr\{X > 2 \lg n\} \leq 1/n$.

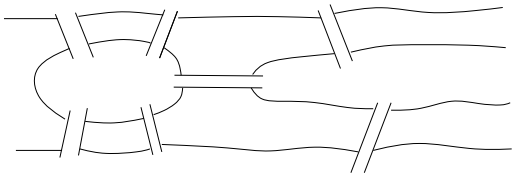
   (d) Show that the expected length of the longest probe sequence is $E[X] = O(\lg n)$.

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

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

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.

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 $B^T B$ represent. Justify.

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

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 $f : E \to \mathbf{R}$ 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 $5^{2^{29^{5^0}+23^{4^1}+17^{3^2}+11^{2^3}+5^{1^4}}} \bmod 6$?

(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 ¾ 1

Name:		
Net ID:	Alias:	U ¾ 1

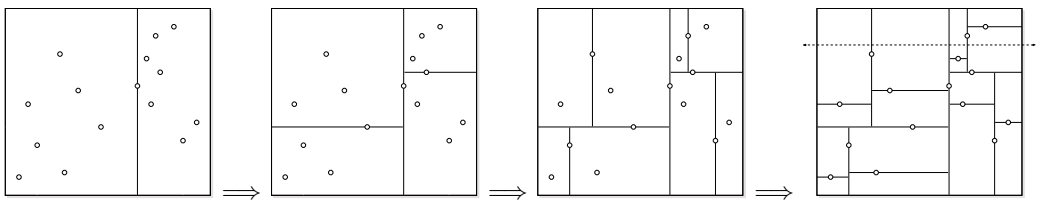
Name:		
Net ID:	Alias:	U ¾ 1

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.**

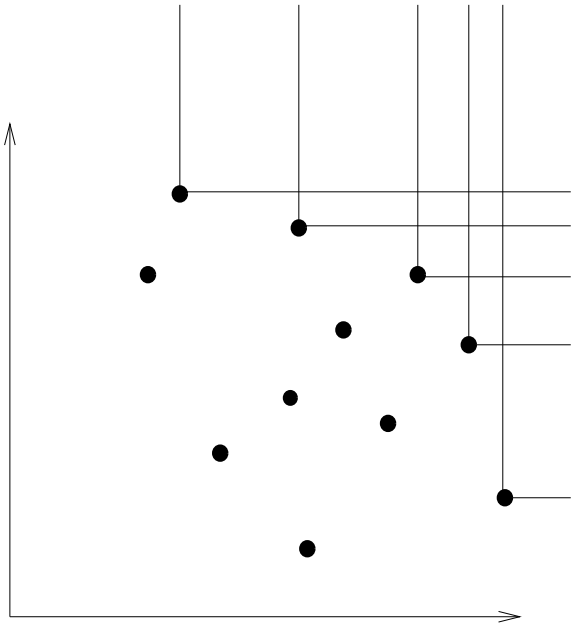
Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, ¾, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. 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.

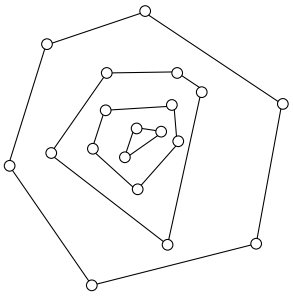
(b) Give an $O(n \log n)$ algorithm to find the staircase of a set of points.

(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 ith 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 evey 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.

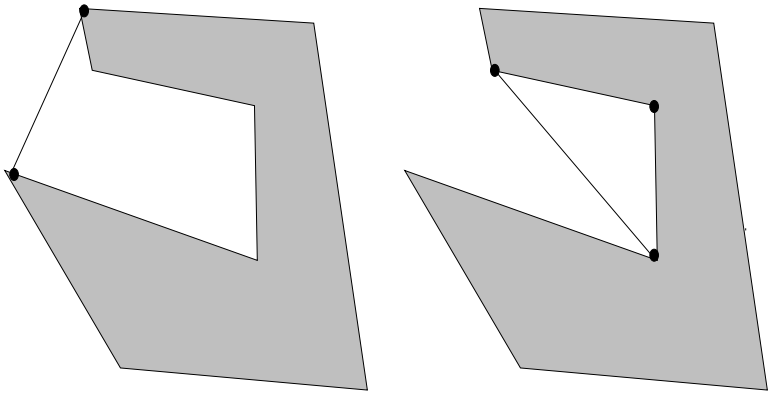
   (c) Area: Give an algorithm to compute the area of a simple n-polygon (not necessarily convex) in $O(n)$ time.

   (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 consective 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.



An external diagonal A 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 ¾ 1

Name:		
Net ID:	Alias:	U ¾ 1

Name:		
Net ID:	Alias:	U ¾ 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, ¾, or 1, respectively. Staple this sheet to the top of your homework.

RequiredProblems

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:

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

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 nth 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 $m \rightarrow fail[m] \rightarrow fail[fail[m]] \rightarrow \ldots$ 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?

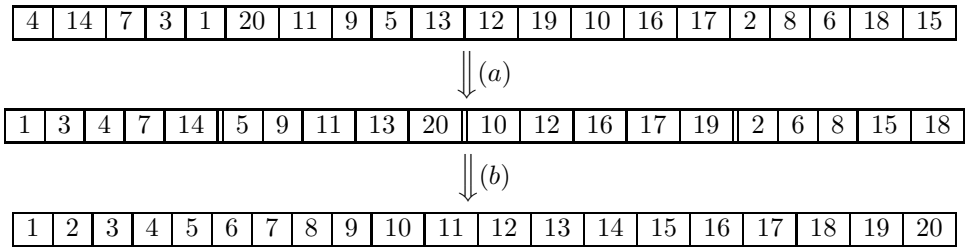
3. Two-stage sorting

(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 alogrithm from part (b) is optimal.

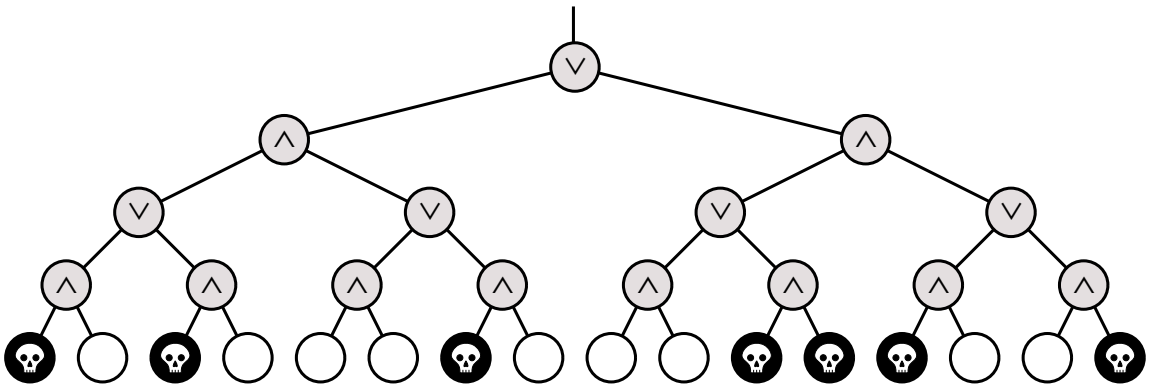


4. Show how to extend the Rabin-Karp fingerprinting method to handle the problem of looking for a given $m \times m$ pattern in an $n \times n$ array of characters. (The pattern may be shifted horizontally and vertically, but it may not be rotated.)

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 toke at the root of the tree. To play the game, you and Death took turns movin 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 challenge 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

   (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 $\Omega(V^2)$.

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).

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 ¾ 1

Name:		
Net ID:	Alias:	U ¾ 1

Name:		
Net ID:	Alias:	U ¾ 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, ¾, 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 $\subseteq$ co-NP.

   (b) Show that if NP $\neq$ 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 $S = s_1, s_2 \ldots, s_n$, 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 $S = s_1, s_2 \ldots, s_n$ 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 $U_1, \ldots, U_k$ 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 $\{-6, 7, -4, -13, -2, 5, 13\}$.

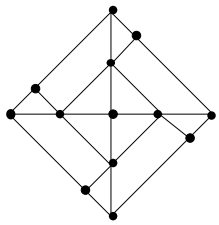


Figure 1. Gadget for PLANAR-3-COLOR.

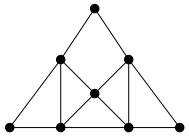


Figure 2. Gadget for DEGREE-4-PLANAR-3-COLOR.

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 defines 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 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 NP $\cap$ co-NP. Hint: co-NP is easy (what's a certificate for showing that a number is composite?). For NP, consider a certificate involving primitive roots and recursively their primitive roots. Show that 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 $T(n) = 4T\left(\left\lceil \frac{n+51}{4} \right\rceil - \sqrt{n}\right) + 17n - 2^{8 \log^*(n^2)} + 6$?

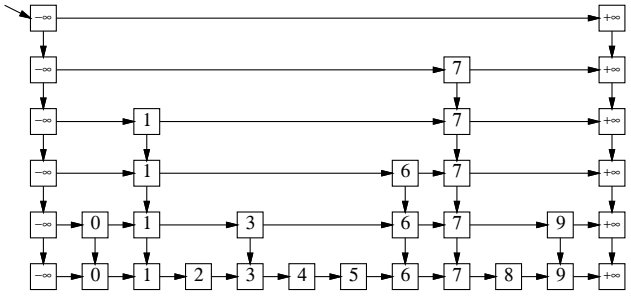
   (g) What is the worst-case running time of randomized quicksort?

   (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?

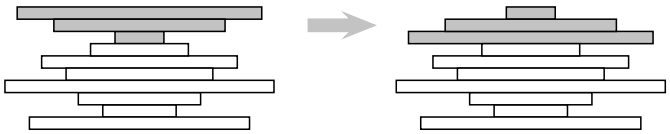
   (i) The amortized time for inserting one item into an n-node splay tree is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty splay tree?

   (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:

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

$$\mathsf{banana}_{\mathsf{ananas}} \qquad \mathsf{ban}_{\mathsf{ana}}\mathsf{ana}_{\mathsf{nas}} \qquad \mathsf{b}_{\mathsf{an}}\mathsf{an}_{\mathsf{a}}\mathsf{a}_{\mathsf{na}}\mathsf{na}_{\mathsf{s}}$$

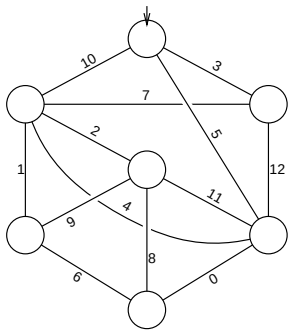
   The strings 'prodgyrnamammiincg' and 'dyprongarmammicing' are both shuffles of 'dynamic' and 'programming':

$$\mathsf{pro}^{\mathsf{d}}{}_{\mathsf{g}}{}^{\mathsf{y}}{}_{\mathsf{r}}\mathsf{nam}_{\mathsf{ammi}}{}^{\mathsf{i}}{}_{\mathsf{n}}{}^{\mathsf{c}}{}_{\mathsf{g}} \qquad {}^{\mathsf{dy}}{}_{\mathsf{pro}}{}^{\mathsf{n}}{}_{\mathsf{g}}{}^{\mathsf{a}}{}_{\mathsf{r}}{}^{\mathsf{m}}{}_{\mathsf{amm}}{}^{\mathsf{ic}}{}_{\mathsf{ing}}$$

   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) 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 **maximum** spanning tree.

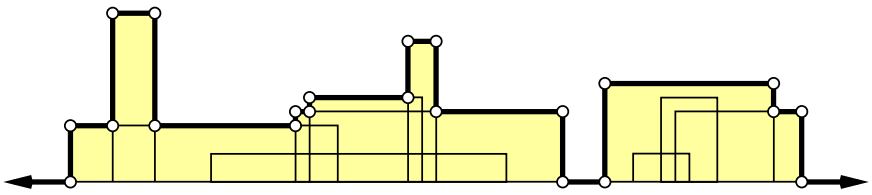


2. (a) *[4 pts]* Prove that a connected acyclic undirected graph with V vertices has exactly $V-1$ edges. ("It's a tree!" is not a proof.)

   (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 builings). 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.



A set of rectangles and its skyline. Compute the sequence of white points.

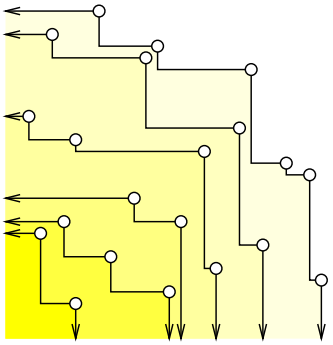
4. Suppose we model a computer network as a weighted undirected graph, where each vertex represents a computer and each edge represents a *direct* network connection between two computers. The weight of each edge represents the *bandwidth* of that connection—the number of bytes that can flow from one 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 communciate, 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 ith 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.

[1] Notice 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!

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

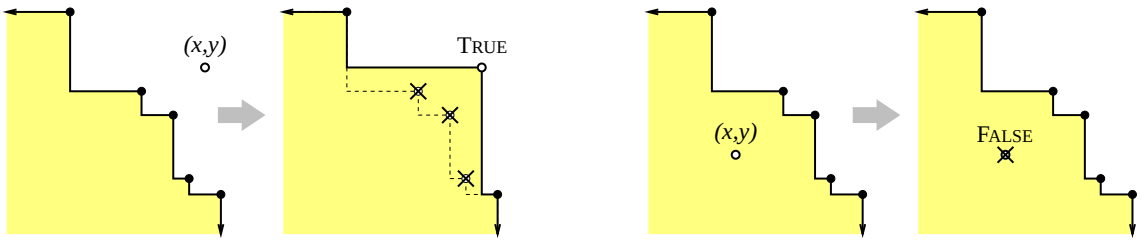
You must turn in this question sheet with your answers.

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 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]

   (a) 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 `HYAKUGOJYUUICHI`[2] is the single letter `S`. For full credit, your algorithm should run in $O(n)$ time.

   (b) 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 `NIAAHAAIN`), and the longest palindrome subsequence of `HYAKUGOJYUUICHI` is `HUUUH`[3] (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.

[1] Finnish for 'soap dealer'.
[2] Japanese for 'one hundred fifty-one'.
[3] English for 'What the heck are you talking about?'

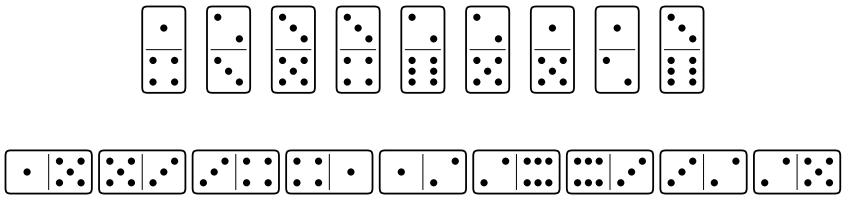
4. **Toute votre base sont appartiennent à nous**

   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.



Top: A set of nine dominos
Bottom: The entire set lined up in a single chain

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.

   (a) Describe and analyze a reduction of one of these problems to the other.

   (b) 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.

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