CS 373: Combinatorial Algorithms, Spring 1999

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

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

Name:	
Net ID:	Alias:

Neatly print your name (first name first, with no comma), your network ID, and a short alias into the boxes above. Do not *sign* your name. Do not write your Social Security number. Staple this sheet of paper to the top of your homework. Grades will be listed on the course web site by alias, so your alias should not resemble your name (or your Net ID). If you do not give yourself an alias, you will be stuck with one we give you, no matter how much you hate it.

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

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

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

Undergrad/.75U Grad/1U Grad Problems

►1. **[173/273]**

(a) Prove that any positive integer can be written as the sum of distinct powers of 2. (For example: $42 = 2^5 + 2^3 + 2^1$, $25 = 2^4 + 2^3 + 2^0$, $17 = 2^4 + 2^0$.)

(b) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. (For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, $17 = F_7 + F_4 + F_2$.)

(c) Prove that *any* integer can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. (For example: $42 = 3^4 - 3^3 - 3^2 - 3^1$, $25 = 3^3 - 3^1 + 3^0$, $17 = 3^3 - 3^2 - 3^0$.)

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

[To simplify notation, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$ and $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions n^2, n, $\binom{n}{2}$, n^3 could be sorted as follows: $n \ll n^2 \equiv \binom{n}{2} \ll n^3$.]

3. **[273/225]** Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. Assume reasonable (nontrivial) base cases. Extra credit will be given for more exact solutions.

   ►(a) $A(n) = A(n/2) + n$

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

   ►(c) $C(n) = 3C(n/2) + n$

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

   ►(e) $E(n) = \min_{0<k<n} \big(E(k) + E(n-k) + 1\big)$

   (f) $F(n) = 4F(\lfloor n/2 \rfloor + 5) + n$

   ►(g) $G(n) = G(n-1) + 1/n$

   *(h) $H(n) = H(n/2) + H(n/4) + H(n/6) + H(n/12) + n$ [Hint: $\frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{12} = 1$.]

   ►*(i) $I(n) = 2I(n/2) + n/\lg n$

   *(j) $J(n) = \dfrac{J(n-1)}{J(n-2)}$

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

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

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

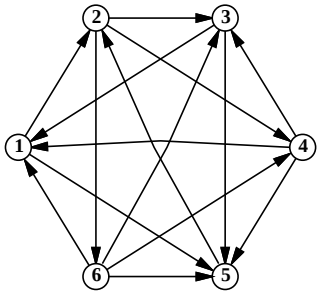
preorder: `M N H C R S K W T G D X I Y A J P O E Z V B U L Q F`
postorder: `C W T K S G R H D N A O E P J Y Z I B Q L F U V X M`

Draw George's binary tree.

Only 1U Grad Problems

▷*1. **[225/273]** A *tournament* is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A *Hamiltonian path* is a sequence of directed edges, joined end to end, that visits every vertex exactly once.

Prove that every tournament contains at least one Hamiltonian path.



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

Practice Problems

1. **[173/273]** Recall the standard recursive definition of the Fibonacci numbers: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Prove the following identities for all positive integers n and m.

   (a) F_n is even if and only if n is divisible by 3.

   (b) $\sum_{i=0}^{n} F_i = F_{n+2} - 1$

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

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

2. **[225/273]**

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

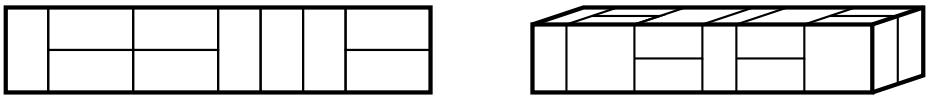
   (b) Is $2^{\lfloor \lg n \rfloor} = \Theta\big(2^{\lceil \lg n \rceil}\big)$? Justify your answer.

   (c) Is $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta\big(2^{2^{\lceil \lg \lg n \rceil}}\big)$? Justify your answer.

3. **[273]**

   (a) A *domino* is a 2×1 or 1×2 rectangle. How many different ways are there to completely fill a $2 \times n$ rectangle with n dominos?

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



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

4. **[273]** Penn and Teller have a special deck of fifty-two cards, with no face cards and nothing but clubs—the ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ..., 52 of clubs. (They're big cards.) Penn shuffles the deck until each each of the 52! possible orderings of the cards is equally likely. He then takes cards one at a time from the top of the deck and gives them to Teller, stopping as soon as he gives Teller the three of clubs.

(a) On average, how many cards does Penn give Teller?

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

*(c) On average, what is the largest-numbered card that Penn gives Teller?

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

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

```
SLOWEUCLID(a, b) :
  if b > a
      return SLOWEUCLID(b, a)
  else if b == 0
      return a
  else
      return SLOWEUCLID(a, b − a)
```

```
FASTEUCLID(a, b) :
  if b == 0
      return a
  else
      return FASTEUCLID(b, a mod b)
```

CS 373: Combinatorial Algorithms, Spring 1999

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

Homework 1 (due February 9, 1999 by noon)

Name:	
Net ID:	Alias:

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

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

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

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

Undergrad/.75U Grad/1U Grad Problems

►1. Consider the following sorting algorithm:

```
StupidSort(A[0..n-1]):
  if n = 2 and A[0] > A[1]
      swap A[0] ↔ 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 $m = \lceil 2n/3 \rceil$ with $m = \lfloor 2n/3 \rfloor$? Justify your answer.

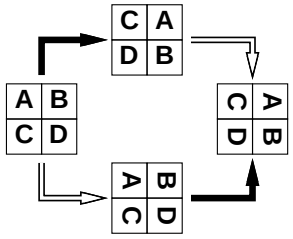
(c) State a recurrence (including the base case(s)) for the number of comparisons executed by StupidSort.

(d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?

*(e) Show that the number of *swaps* executed by STUPIDSORT is at most $\binom{n}{2}$.

►2. 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.

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

(c) What is the algorithm's running time if a $k \times k$ blit takes $O(k^2)$ time?

(d) What if a $k \times k$ blit takes only $O(k)$ time?

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

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

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

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

(a) Describe an efficient deterministic algorithm that finds the median bolt. How many nut-bolt comparisons does your algorithm perform in the worst case?

(b) Describe an efficient *randomized* algorithm that finds the median bolt.

i. State a recurrence for the expected number of nut/bolt comparisons your algorithm performs.

ii. What is the probability that your algorithm compares the ith largest bolt with the jth largest nut?

iii. What is the expected number of nut-bolt comparisons made by your algorithm? [Hint: Use your answer to either of the previous two questions.]

Only 1U Grad Problems

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

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

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

*(c) Suppose you don't know how many parties there are, but you do know that one party has a plurality, and at least p people in the plurality party are present. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)

★(d) Finally, suppose you don't know how many parties are represented at the convention, and you don't know how big the plurality is. Give an efficient algorithm to identify a member of the plurality party. How is the running time of your algorithm affected by the number of parties (k)? By the size of the plurality (p)?

Practice Problems

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

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

3. Dynamic Programming: Coin Changing
   Consider the problem of making change for n cents using the least number of coins.

   (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

   (b) Suppose that the available coins have the values $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

   (c) Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution, show why.

   (d) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.

   (e) Prove that, with only two coins a, b whose gcd is 1, the smallest value n for which change *can* be given for all values greater than or equal to n is $(a-1)(b-1)$.

   ★(f) For only three coins a, b, c whose gcd is 1, give an algorithm to determine the smallest value n for which change *can* be given for all values greater than n. (note: this problem is currently unsolved for $n > 4$.

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

   (a) Give a dynamic programming algorithm to do this.

   (b) Prove that if the neatness function is linear, a linear time greedy algorithm will give an optimum 'neatness'.

5. Comparison of Amortized Analysis Methods
   A sequence of n operations is performed on a data structure. The 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 1999

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

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

Name:	
Net ID:	Alias:

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

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

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

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

Undergrad/.75U Grad/1U Grad Problems

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

▶2. SELECT(A, k)
Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.

(a) Show that a rotation in an augmented binary tree can be performed in constant time.

(b) Describe an algorithm SCAPEGOATSELECT(k) that selects the kth smallest item in an augmented scapegoat tree in $O(\log n)$ *worst-case* time. (The scapegoat trees presented in class were already augmented.)

(c) Describe an algorithm 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. Scapegoat trees

(a) Prove that only one tree gets rebalanced at any insertion.

(b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree ($I(v) = max(0, |\hat{v}| - |\check{v}|)$, where $\hat{v}$ is the child of greater height and $\check{v}$ the child of lesser height, $|v|$ is the number of nodes in subtree v, and perfectly balanced means each subtree has as close to half the leaves as possible and is perfectly balanced itself.

⋆(c) Show that you can rebuild a fully balanced binary tree in $O(n)$ time using only $O(1)$ additional memory.

▶4. Memory Management

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

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

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

Only 1U Grad Problems

▷1. Detecting overlap

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

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

Practice Problems

1. Amortization

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

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

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

*2. Amortization
Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of "fits", where the 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.]

3. Rotations

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

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

4. Fibonacci Heaps: SECONDMIN
We want to find the second smallest of a set efficiently.

   (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.

   *(b) Modify the Fibonacci Heap data structure to implement SECONDMIN in constant time.

5. Give an efficient implementation of the operation **Fib-Heap-Change-Key(**H, x, k**)**, which changes the key of a node x in a Fibonacci heap H to the value k. The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which k is greater than, less than, or equal to $key[x]$.

CS 373: Combinatorial Algorithms, Spring 1999

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

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

Name:	
Net ID:	Alias:

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

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

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

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

Undergrad/.75U Grad/1U Grad Problems

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

▶2. SELECT(A, k)
Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.

(a) Show that a rotation in an augmented binary tree can be performed in constant time.

(b) Describe an algorithm SCAPEGOATSELECT(k) that selects the kth smallest item in an augmented scapegoat tree in $O(\log n)$ *worst-case* time. (The scapegoat trees presented in class were already augmented.)

(c) Describe an algorithm 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. Scapegoat trees

(a) Prove that only one tree gets rebalanced at any insertion.

(b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree ($I(v) = max(0, |\hat{v}| - |\check{v}|)$, where $\hat{v}$ is the child of greater height and $\check{v}$ the child of lesser height, $|v|$ is the number of nodes in subtree v, and perfectly balanced means each subtree has as close to half the leaves as possible and is perfectly balanced itself.

⋆(c) Show that you can rebuild a fully balanced binary tree in $O(n)$ time using only $O(1)$ additional memory.

▶4. Memory Management

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

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

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

Only 1U Grad Problems

▷1. Detecting overlap

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

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

Practice Problems

1. Amortization

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

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

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

*2. Amortization
Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of "fits", where the 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.]

3. Rotations

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

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

4. Fibonacci Heaps: SECONDMIN
We want to find the second smallest of a set efficiently.

   (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.

   *(b) Modify the Fibonacci Heap data structure to implement SECONDMIN in constant time.

5. Give an efficient implementation of the operation **Fib-Heap-Change-Key(**H, x, k**)**, which changes the key of a node x in a Fibonacci heap H to the value k. The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which k is greater than, less than, or equal to $key[x]$.

CS 373: Combinatorial Algorithms, Spring 1999

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

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

Name:	
Net ID:	Alias:

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

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

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

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

Undergrad/.75U Grad/1U Grad Problems

▶1. Hashing

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

(b) (3 pts) Let the hash function for a table of size m be

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

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

►3. (5 pts) Makefiles:
In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called 'make' that only recompiles those files that were changed, *and* any intermediate files in the compiliation that depend on those changed. Design an algorithm to recompile only those necessary.

►4. (5 pts) Shortest Airplane Trip:
A person wants to fly from city A to city B in the shortest possible time. S/he turns to the traveling agent who knows all the departure and arrival times of all the flights on the planet. Give an algorithm that will allow the agent to choose an optimal route. Hint: rather than modify Dijkstra's algorithm, modify the data. The time is from departure to arrival at the destination, so it will include layover time (time waiting for a connecting flight).

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

(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.

Only 1U Grad Problems

▷1. Nesting Envelopes
You are given an unlimited number of each of n different types of envelopes. The dimensions of envelope type i are $x_i \times y_i$. In nesting envelopes inside one another, you can place envelope A inside envelope B if and only if the dimensions A are *strictly smaller* than the dimensions of B. Design and analyze an algorithm to determine the largest number of envelopes that can be nested inside one another.

Practice Problems

1. The incidence matrix of an undirected graph $G = (V, E)$ is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that
$$b_{ij} = \begin{cases} 1 & (i,j) \in E, \\ 0 & (i,j) \notin E. \end{cases}$$
   (a) Describe what all the entries of the matrix product BB^T represent (B^T is the matrix transpose). Justify.
   (b) Describe what all the entries of the matrix product $B^T B$ 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).

2. $o(V^2)$ Adjacency Matrix Algorithms
   (a) Give an $O(V)$ algorithm to decide whether a directed graph contains a *sink* in an adjacency matrix representation. A sink is a vertex with in-degree $V - 1$.
   (b) An undirected graph is a scorpion if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree $V - 2$ (the body) connected to the other $V - 3$ vertices (the feet). Some of the feet may be connected to other feet.
   Design an algorithm that decides whether a given adjacency matrix represents a scorpion by examining only $O(V)$ of the entries.
   (c) Show that it is impossible to decide whether G has at least one edge in $O(V)$ time.

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

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

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

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

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

(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 1999

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

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

Name:	
Net ID:	Alias:

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

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

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

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

Undergrad/.75U Grad/1U Grad Problems

►1. (5 pts total) Collinearity
Give an $O(n^2 \log n)$ algorithm to determine whether any three points of a set of n points are collinear. Assume two dimensions and *exact* arithmetic.

►2. (4 pts, 2 each) Convex Hull Recurrence
Consider the following generic recurrence for convex hull algorithms that divide and conquer:

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

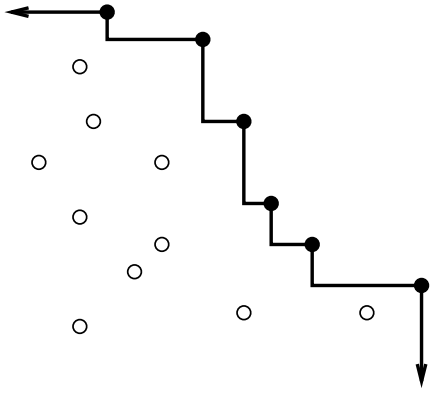
where $n \geq n_1 + n_2, h = h_1 + h_2$ and $n \geq h$. This means that the time to compute the convex hull is a function of both n, the number of input points, and h, the number of convex hull vertices. The splitting and merging parts of the divide-and-conquer algorithm take $O(n)$ time. When n is a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

(a) $h_1, h_2 < \frac{3}{4}h$

(b) $n_1, n_2 < \frac{3}{4}n$

►3. (5 pts) Circle Intersection
Give an $O(n \log n)$ algorithm to test whether any two circles in a set of size n intersect.

►4. (5 pts total) Staircases
You are given a set of points in the first quadrant. A *left-up* point of this set is defined to be a point that has no points both greater than it in both coordinates. The left-up subset of a set of points then forms a *staircase* (see figure).



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

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

Only 1U Grad Problems

▷1. (6 pts, 2 each) Ghostbusters and Ghosts
A group of n ghostbusters is battling n ghosts. Each ghostbuster can shoot a single energy beam at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits a ghost. The ghostbusters must all fire at the same time and no two energy beams may cross. The positions of the ghosts and ghostbusters is fixed in the plane (assume that no three points are collinear).

(a) Prove that for any configuration ghosts and ghostbusters there exists such a non-crossing matching.

(b) Show that there exists a line passing through one ghostbuster and one ghost such that the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Give an efficient algorithm to find such a line.

(c) Give an efficient divide and conquer algorithm to pair ghostbusters and ghosts so that no two streams cross.

Practice Problems

1. Basic Computation (assume two dimensions and *exact* arithmetic)

   (a) Intersection: Extend the basic algorithm to determine if two line segments intersect by taking care of *all* degenerate cases.

   (b) Simplicity: Give an $O(n \log n)$ algorithm to determine whether an n-vertex polygon is simple.

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

   (d) Inside: Give an algorithm to determine whether a point is within a simple n-polygon (not necessarily convex) in $O(n)$ time.

2. External Diagonals and Mouths

   (a) A pair of polygon vertices defines an *external diagonal* if the line segment between them is completely outside the polygon. Show that every nonconvex polygon has at least one external diagonal.

   (b) Three 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.

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

4. Another On-Line Convex Hull Algorithm

   (a) Given an n-polygon and a point outside the polygon, give an algorithm to find a tangent.

   *(b) Suppose you have found both tangents. Give an algorithm to remove the points from the polygon that are within the angle formed by the tangents (as segments!) and the opposite side of the polygon.

   (c) Use the above to give an algorithm to compute the convex hull on-line in $O(n \log n)$

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

   (a) Prove that if a set of points is chosen randomly within a given rectangle. then the average size of the convex hull is $O(\log n)$.

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

CS 373: Combinatorial Algorithms, Spring 1999

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

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

Name:	
Net ID:	Alias:

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

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

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

Undergrad/.75U Grad/1U Grad Problems

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

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

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

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

(a) (2 pts) Prove that in any Fibonacci string there are no two b's adjacent and no three a's.

(b) (2 pts) Give the unoptimized and optimized 'prefix' (fail) function for F_7.

(c) (3 pts) Prove that, in searching for a Fibonacci string of length m using unoptimized KMP, it may shift up to $\lceil \log_\phi m \rceil$ times, where $\phi = (1+\sqrt{5})/2$, is the golden ratio. (Hint: Another way of saying the above is that we are given the string F_n and we may have to shift n times. Find an example text T that gives this number of shifts).

(d) (3 pts) What happens here when you use the optimized prefix function? Explain.

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

▶4. (4 pts, 2 each) Lower Bounds on Adjacency Matrix Representations of Graphs

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

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

Only 1U Grad Problems

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

Practice Problems

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

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

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

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

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

6. Some elementary reductions

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

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

7. There is no Proof but We are pretty Sure
   Justify (prove) the following logical rules of inference:

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

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

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

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

CS 373: Combinatorial Algorithms, Spring 1999

Midterm 1 (February 23, 1999)

Name:	
Net ID:	Alias:

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

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

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

#	Score	Grader
1		
2		
3		
4		
5		

1. **Multiple Choice**

   Every question below has one of the following answers.

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

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

   ☐ What is $\sum_{i=1}^{n} i$?

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

   ☐ What is the solution of the recurrence $T(n) = T\left(\sqrt{n}\right) + n$?

   ☐ What is the solution of the recurrence $T(n) = T(n-1) + \lg n$?

   ☐ What is the solution of the recurrence $T(n) = 2T\left(\left\lceil \frac{n+27}{2} \right\rceil\right) + 5n - 7\sqrt{\lg n} + \frac{1999}{n}$?

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

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

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

   ☐ How many bits are there in the binary representation of the nth Fibonacci number?

   ☐ What is the worst-case cost of merging two arbitrary splay trees with n items total into a single splay tree with n items.

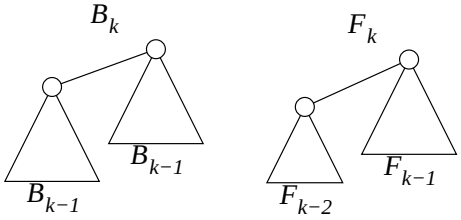
   ☐ Suppose you correctly identify three of the answers to this question as obviously wrong. If you pick one of the two remaining answers at random, what is your expected score for this problem?

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

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

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

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



Recursive definitions of binomial trees and Fibonacci trees.

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

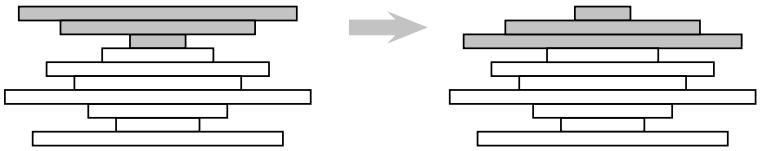
```
RANDOMMIN(A[1..n]):
  min ← ∞
  for i ← 1 to n in random order
      if A[i] < min
          min ← A[i]    (⋆)
      return min
```

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

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

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

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



Flipping the top three pancakes

   (a) **[3 pt]** Describe an algorithm to sort an arbitrary stack of n pancakes.

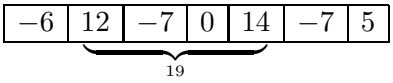
   (b) **[3 pt]** Prove that your algorithm is correct.

   (c) **[2 pt]** Exactly how many flips does your algorithm perform in the worst case? (A correct $\Theta()$ bound is worth one point.)

   (d) **[2 pt]** Suppose one side of each pancake is burned. Exactly how many flips do you need to sort the pancakes *and* have the burned side of every pancake on the bottom? (A correct $\Theta()$ bound is worth one point.)

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

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



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

CS 373: Combinatorial Algorithms, Spring 1999

Midterm 2 (April 6, 1999)

Name:	
Net ID:	Alias:

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

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

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

#	Score	Grader
1		
2		
3		
4		
5		

1. **Bipartite Graphs**

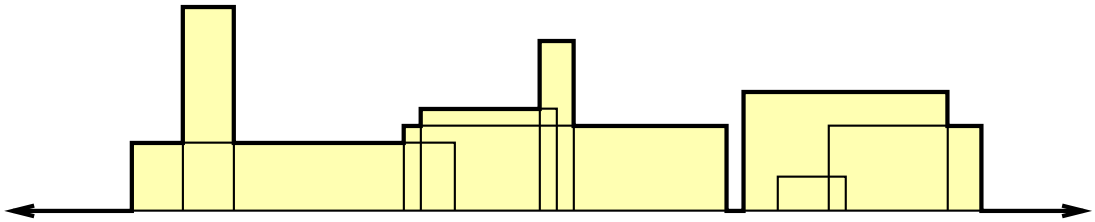
   A graph (V, E) is *bipartite* if the vertices V can be partitioned into two subsets L and R, such that every edge has one vertex in L and the other in R.

   (a) Prove that every tree is a bipartite graph.

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

2. **Manhattan Skyline**

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



A set of rectangles and its skyline.

3. **Least Cost Vertex Weighted Path**

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

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

4. **Union-Find with Alternate Rule**

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

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

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

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

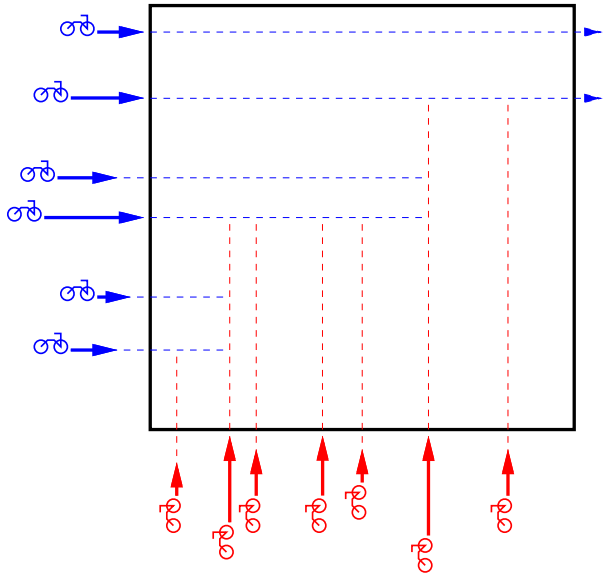
[Hint: First prove it without path compression, and then argue that path compression doesn't matter (for this problem).]

5. **Motorcycle Collision**

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

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

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



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

(a) Solve the case $n = 1$. Given only two motorcycles moving perpendicular to each other, determine which one of them falls over and where in $O(1)$ time.

(b) Describe an efficient algorithm to find the set of all points where motorcycles fall over.

5. **Motorcycle Collision (continued)**

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

CS 373: Combinatorial Algorithms, Spring 1999

Final Exam (May 7, 1999)

Name:	
Net ID:	Alias:

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

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

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

#	Score	Grader
1		
2		
3		
4		
5		
6		
7		

1. **Short Answer**

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

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

(a) Given a Champaign phone book, find your own phone number.

(b) Given a collection of n rectangles in the plane, determine whether any two intersect in $O(n \log n)$ time.

(c) Given an undirected graph G and an integer k, determine if G has a complete subgraph with k edges.

(d) Given an undirected graph G, determine if G has a triangle — a complete subgraph with three vertices.

(e) Prove that any n-vertex graph with minimum degree at least $n/2$ has a Hamiltonian cycle.

(f) Given a graph G and three distinguished vertices u, v, and w, determine whether G contains a path from u to v that passes through w.

(g) Given a graph G and two distinguished vertices u and v, determine whether G contains a path from u to v that passes through at most 17 edges.

(h) Solve the recurrence $T(n) = 5T(n/17) + O(n^{4/3})$.

(i) Solve the recurrence $T(n) = 1/n + T(n-1)$, where $T(0) = 0$.

(j) Given an array of n integers, find the integer that appears most frequently in the array.

(a) _____ (f) _____

(b) _____ (g) _____

(c) _____ (h) _____

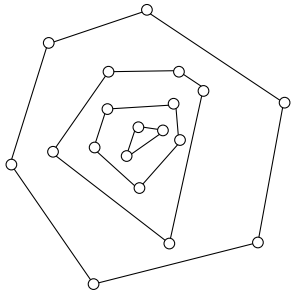
(d) _____ (i) _____

(e) _____ (j) _____

2. **Convex Layers**

Given a set Q of points in the plane, define the *convex layers* of Q inductively as follows: The first convex layer of Q is just the convex hull of Q. For all $i > 1$, the 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. [Partial credit for a correct slower algorithm; extra credit for a correct faster algorithm.]



A set of points with four convex layers.

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

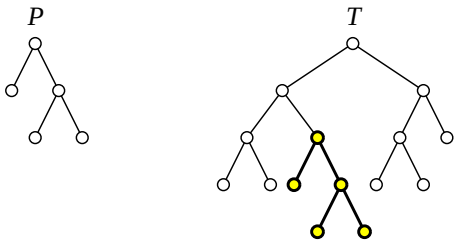
   (a) **[3 pts]** Describe an $O(n)$-time algorithm for the following problem:
   Find two numbers from the list that add up to zero, or report that there is no such pair. In other words, find two numbers a and b such that $a + b = 0$.

   (b) **[7 pts]** Describe an $O(n^2)$-time algorithm for the following problem:
   Find *three* numbers from the list that add up to zero, or report that there is no such triple. In other words, find three numbers a, b, and c, such that $a + b + c = 0$. [Hint: Use something similar to part (a) as a subroutine.]

4. **Pattern Matching**

   (a) **[4 pts]** A *cyclic rotation* of a string is obtained by chopping off a prefix and gluing it at the end of the string. For example, ALGORITHM is a cyclic shift of RITHMALGO. Describe and analyze an algorithm that determines whether one string $P[1..m]$ is a cyclic rotation of another string $T[1..n]$.

   (b) **[6 pts]** Describe and analyze an algorithm that decides, given any two binary trees P and T, whether P equals a subtree of T. [Hint: First transform both trees into strings.]



P occurs exactly once as a subtree of T.

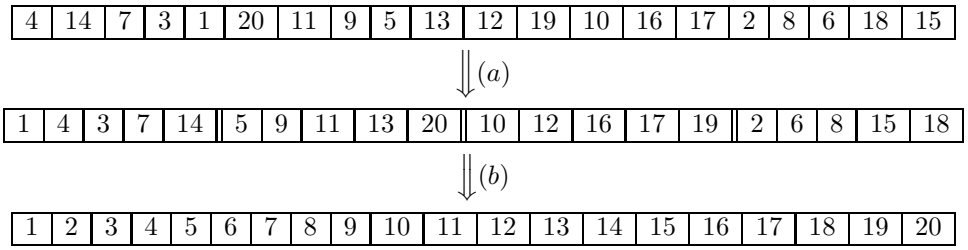
5. **Two-stage Sorting**

   (a) **[1 pt]** Suppose we are given an array $A[1..n]$ of distinct integers. Describe an algorithm that splits A into n/k subarrays, each with k elements, such that the elements of each subarray $A[(i-1)k+1..ik]$ are sorted. Your algorithm should run in $O(n \log k)$ time.

   (b) **[2 pts]** Given an array $A[1..n]$ that is already split into n/k sorted subarrays as in part (a), describe an algorithm that sorts the entire array in $O(n \log(n/k))$ time.

   (c) **[3 pts]** Prove that your algorithm from part (a) is optimal.

   (d) **[4 pts]** Prove that your algorithm from part (b) is optimal.



6. **SAT Reduction**

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

7. **Knapsack**

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

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

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

(b) **[5 pts]** Describe and analyze a dynamic programming algorithm to solve the knapsack problem in $O(nK)$ time. Prove your algorithm is correct.