Algorithm Homework and Test Problems

Steven S. Skiena
Department of Computer Science
State University of New York
Stony Brook, NY 11794-4400
skiena@cs.sunysb.edu

January 29, 2006

All of the midterm and final exam problems for this semester will be drawn from this list. Thus trying to solve the problems associated with topics covered on the exam is the best possible way to study.

Note that this document contains two groups of problems, one group from my personal problem list and another selected from Bill Gasarch's problem list. Be sure to look at both sets. In particular, I promise *at least* one problem from Gasarch's list on each midterm and final.

1 Mathematical Preliminaries and Analysis

In this section, we should have all the problems dealing with the big oh notation, recurrence relations, logarithms, etc.

1.1 Summations

1.2 Big Oh

1. Graph the following expressions. For each expression, state for which values of n that expression is the most efficient.

   For the following functions:

$$4n^2 \qquad \log_3 n \qquad 3^n \qquad 20n \qquad 2 \qquad \log_2 n \qquad n^{2/3}$$

   (a) Graph all seven functions on a single plot.

(b) Partition the integers $n \geq 1$ into groups such that all the integers in a group have the same function which yields the smallest values of $f(n)$.

(c) Partition the integers $n \geq 1$ into groups such that all the integers in a group have the same function which yields the largest values of $f(n)$.

(d) Arrange the expressions by asymptotic growth rate from slowest to fastest.

2. Show that for any real constants a and b, $b > 0$,

$$(n + a)^b = \Theta(n^b)$$

(*)

3. (a) Is $2^{n+1} = O(2^n)$?

   (b) Is $2^{2n} = O(2^n)$?

   (*)

4. For each of the following, compute how large a problem instance do you need before algorithm A is faster than algorithm B. How much time do the algorithms take on that instance?

   (a) Algorithm A takes n^2 days to solve a problem of size n. Algorithm B takes n^3 seconds on the same problem.

   (b) Algorithm A takes n^2 days to solve a problem of size n. Algorithm B takes 2^n seconds on the same problem.

5. Let P be a problem. The worst-case time complexity of P is $O(n^2)$. The worst-case time complexity of P is also $\Omega(n \lg n)$. Let A be an algorithm that solves P. Which subset of the following statements are consistent with this information about the complexity of P?

   - A has worst-case time complexity $O(n^2)$.
   - A has worst-case time complexity $O(n^{3/2})$.
   - A has worst-case time complexity $O(n)$.
   - A has worst-case time complexity $\Theta(n^2)$.
   - A has worst-case time complexity $\Theta(n^3)$.

6. For each of the following pairs of functions, either $f(n)$ is in $O(g(n))$, $f(n)$ is in $\Omega(g(n))$, or $f(n) = \Theta(g(n))$. Determine which relationship is correct and briefly explain why.

   (a) $f(n) = \log n^2$; $g(n) = \log n + 5$

   (b) $f(n) = \sqrt{n}$; $g(n) = \log n^2$

(c) $f(n) = \log^2 n$; $g(n) = \log n$

(d) $f(n) = n$; $g(n) = \log^2 n$

(e) $f(n) = n \log n + n$; $g(n) = \log n$

(f) $f(n) = 10$; $g(n) = \log 10$

(g) $f(n) = 2^n$; $g(n) = 10n^2$

(h) $f(n) = 2^n$; $g(n) = 3^n$

(*)

7. List the functions below from the lowest to the highest order. If any two or more are of the same order, indicate which.

n	2^n	$n \lg n$	$\ln n$
$n - n^3 + 7n^5$	$\lg n$	$\sqrt{n}$	e^n
$n^2 + \lg n$	n^2	2^{n-1}	$\lg \lg n$
n^3	$(\lg n)^2$	$n!$	$n^{1+\varepsilon}$ where $0 < \varepsilon < 1$

(*)

8. List the functions below from the lowest to the highest order. If any two or more are of the same order, indicate which.

$\sqrt{n}$	n	2^n
$n \log n$	$n - n^3 + 7n^5$	$n^2 + \log n$
n^2	n^3	$\log n$
$n^{\frac{1}{3}} + \log n$	$(\log n)^2$	$n!$
$\ln n$	$\frac{n}{\log n}$	$\log \log n$
$(\frac{1}{3})^n$	$(\frac{3}{2})^n$	6

(*)

9. For each of the following pairs of functions $f(n)$ and $g(n)$, determine whether $f(n) = O(g(n))$, $g(n) = O(f(n))$, or both.

(a) $f(n) = (n^2 - n)/2$, $g(n) = 6n$

(b) $f(n) = n + 2\sqrt{n}$, $g(n) = n^2$

(c) $f(n) = n \log n$, $g(n) = n\sqrt{n}/2$

(d) $f(n) = n + \log n$, $g(n) = \sqrt{n}$

(e) $f(n) = 2(\log n)^2$, $g(n) = \log n + 1$

(f) $f(n) = 4n \log n + n$, $g(n) = (n^2 - n)/2$

10. Prove that $n^3 - 3n^2 - n + 1 = \Theta(n^3)$.

11. Prove that

    (a) $2^n = O(n!)$
    (b) $n! = \Omega(2^n)$

12. For each of the following pairs of functions $f(n)$ and $g(n)$, give an appropriate positive constant c such that $f(n) \leq c \cdot g(n)$ for all $n > 1$.

    (a) $f(n) = n^2 + n + 1$, $g(n) = 2n^3$
    (b) $f(n) = n\sqrt{n} + n^2$, $g(n) = n^2$
    (c) $f(n) = n^2 - n + 1$, $g(n) = n^2/2$

13. Prove that if $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

14. Prove that if $f_1(N) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_1(n))$, then $f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$.

15. Prove that if $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

16. Prove or disprove. $O((x + y)^2) = O(x^2) + O(y^2)$

    (*)

17. Show that big-O is transitive. That is, if $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.

    problem

18. True or False?

    (a) $2n^2 + 1 = O(n^2)$
    (b) $\sqrt{n} = O(\log n)$
    (c) $\log n = O(\sqrt{n})$
    (d) $n^2(1 + \sqrt{n}) = O(n^2 \log n)$
    (e) $3n^2 + \sqrt{n} = O(n^2)$
    (f) $\sqrt{n} \log n = O(n)$
    (g) $\log n = O(n^{-1/2})$

    (*)

19. For each of the following pairs of functions $f(n)$ and $g(n)$, state whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$, or none of the above

(a) $f(n) = n^2 + 3n + 4$, $g(n) = 6n + 7$

(b) $f(n) = n\sqrt{n}$, $g(n) = n^2 - n$

(c) $f(n) = 2^n - n^2$, $g(n) = n^4 + n^2$

(*)

20. Is $f(cn) = \Theta(f(n))$ for all functions f and positive constants c? (Hint: Consider particularly fast-growing functions.) (*)

1.3 Recurrence Relations

1.4 Induction

1.5 Proof by Contradiction

1.6 Number Theory

1.7 Binomial Coefficients and Counting

1.8 Logarithms

1. (a) Prove that $\log_a(xy) = \log_a x + \log_a y$

   (b) Prove that $\log_a x^y = y \log_a x$

   (c) Prove that $\log_a x = \frac{\log_b x}{\log_b a}$

   (d) Prove that $x^{\log_b y} = y^{\log_b x}$

   (*)

2. Prove that that the binary representation of $n \geq 1$ has $\lfloor \lg_2 n \rfloor + 1$ bits.

3. In one of my research papers I give a comparison-based sorting algorithm that runs in $O(n \log(\sqrt{n}))$. Given the existence of an $\Omega(n \log n)$ lower bound for sorting, how can this be possible? (*)

1.9 Estimation

1. Do all the books you own total at least one million pages? How many total pages are stored in your school library?

2. How many words are there in the textbook?

3. How many hours are one million seconds? How many days? Answer these questions by doing all arithmetic in your head.

4. Estimate how many cities and towns there are in the United States.
5. Estimate how many cubic miles of water flow out of the mouth of the Mississippi River each day? Do not look up any supplemental facts. Describe all assumptions you made in arriving at your answer.
6. A sorting algorithm takes 1 second to sort 1000 items on your local machine. How long will it take to sort 10,000 items...
   (a) if you believe that the algorithm takes time proportional to n^2, and
   (b) if you believe that the algorithm takes time roughly proportional to $n \log n$?

1.10 Probability

1.11 Program Analysis

1. What value is returned by the following function? Express your answer as a function of n. Give the worst-case running time using the big-Oh notation.

```
function mystery(n)
1.    r := 0
2.    for i := 1 to n − 1 do
3.        for j := i + 1 to n do
4.            for k := 1 to j do
5.                r := r + 1
6.    return(r)
```

2. What value is returned by the following function? Express your answer as a function of n. Give the worst-case running time using big-Oh notation.

```
function pesky(n)
1.    r := 0
2.    for i := 1 to n do
3.        for j := 1 to i do
4.            for k := j to i + j do
5.                r := r + 1
6.    return(r)
```

3. What value is returned by the following function? Express your answer as a function of n. Give the worst-case running time using big-Oh notation.

```
function prestiferous($n$)
1.    $r := 0$
2.    for $i := 1$ to $n$ do
3.      for $j := 1$ to $i$ do
4.        for $k := j$ to $i + j$ do
5.          for $l := 1$ to $i + j - k$ do
6.            $r := r + 1$
7.    return($r$)   (*)
```

4. What value is returned by the following function? Express your answer as a function of n. Give the worst-case running time using big-Oh notation.

```
function conundrum($n$)
1.    $r := 0$
2.    for $i := 1$ to $n$ do
3.      for $j := i + 1$ to $n$ do
4.        for $k := i + j - 1$ to $n$ do
5.          $r := r + 1$
6.    return($r$)   (**)
```

1.12 Proofs of Correctness

1. (a) Prove the correctness of the following recursive algorithm for incrementing natural numbers:

```
function increment(y)
    comment Return y+1
1.    if $y = 0$ then return(1) else
2.      if ($y$ mod 2) = 1 then
3.        return($2 \cdot increment(\lfloor y/2 \rfloor)$)
4.      else return($y + 1$)
```

   (b) Analyze this algorithm. How many right-shifts does it use, i.e., how many times is line 2 executed in the worst case?

2. (a) Prove the correctness of the following recursive algorithm to multiply two natural numbers is correct, for all integer constants $c \geq 2$.

```
function multiply($y, z$)
```

```
        comment Return the product $yz$.
1.          if $z = 0$ then return(0) else
2.          return(multiply($cy$, $\lfloor z/c \rfloor$) + $y \cdot (z \bmod c)$)
```

(b) Analyze this algorithm. How many additions does it use, i.e., how many times is line 2 executed in the worst case?

1.13 Programming Exercises

2 Data Structures

2.1 Simple Data Structures

1. For each of the four types of linked lists in the following table, what is the asymptotic worst-case running time for each dynamic-set operation listed?

	singly unsorted	singly sorted	doubly unsorted	doubly sorted
Search(L, k)				
Insert(L, x)				
Delete(L, x)				
Successor(L, x)				
Predecessor(L, x)				
Minimum(L)				
Maximum(L)				

(*)

2. A common problem for compilers and text editors is determining whether the parentheses (or other brackets) in a string are balanced and properly nested. For example, the string ((())())() contains properly nested pairs of parentheses, which the strings)()(and ()) do not.

   (a) Give an algorithm that returns true if a string contains properly nested and balanced parentheses, and false if otherwise. *Hint:* At some point while scanning an illegal string from left to right will you have encountered more right parentheses than left parentheses.

   (b) Give an algorithm that returns the position of the first offending parenthesis if the string is not properly nested and balanced. That is, if an excess right parenthesis is found, return its position; if there are too many left parentheses, return the position of the first left parenthesis. Return -1 if the string is properly balanced and nested.

3. Write a program to reverse the direction of a given singly-linked list. In other words, after the reversal all pointers should now point backwards.

4. Suppose that we are given a stack of n elements which we would like to sort, by returning a stack containing the records in sorted order (with the smallest value on top). We are allowed to use only the following operations to manipulate the data:

   - *Pop-Push*(s_1,s_2) – pop the top item from stack s_1 and push it onto stack s_2.
   - *Compare*(s_1,s_2) – test whether the top element of stack s_1 is less than the top element of stack s_2.

   (a) Give a $\Theta(n^2)$ sorting algorithm using just these operations and three stacks.

   (b) Give an initial stack which cannot be sorted just using these operations and two stacks.

   (c) Give a $\Theta(n^2)$ sorting algorithm using just these operations and two stacks, where you are allowed to temporarily park elements in a constant number of additional registers. (hint: use insertion sort)

   (*)

5. Design a data structure that allows one to search, insert, and delete an integer X in $O(1)$ time (ie constant time, independent of the total number of integers stored). Assume that $1 \leq X \leq n$ and that there are $m + n$ units of space available for the symbol table, where m is the maximum number of integers that can be in the table at any one time. (Hint: use two arrays $A[1..n]$ and $B[1..m]$.) You are not allowed to initialize either A or B, as that would take $O(m)$ or $O(n)$ operations. This means the arrays are full of random garbage to begin with, so you must be very careful.

   (**)

2.2 Heaps

1. Devise an algorithm for finding the k smallest elements of an unsorted set of n integers in $O(n + k \lg n)$.

2. Given an array-based heap on n elements and a real number x, efficiently determine whether the kth smallest in the heap is greater than or equal to x. Your algorithm should be $O(k)$ in the worst-case, independent of the size of the heap. Hint: you not have to find the kth smallest element; you need only determine its relationship to x. (*)

3. Give an $O(n \lg k)$-time algorithm which merges k sorted lists with a total of n elements into one sorted list. (hint: use a heap to speed up the elementary $O(kn)$-time algorithm).

   (*)

4. Devise a data structure that supports the following operations on an n-element set: *insert* in time $O(\lg n)$, and *delete max* in $O(1)$ time. Note that *delete max* takes $O(\lg n)$ in a conventional heap. (*)

5. (a) Give an efficient algorithm to find the second-largest key among n keys. You can do better than $2n - 3$ comparisons.

   (b) Give an efficient algorithm to find the third-largest key among n keys. How many key comparisons does you algorithm do in the worst case? Must your algorithm determine which key is largest and second-largest in the process?

   (*)

2.3 Union-Find

1. (a) Show a *Union-Find* program of size n that requires $\Theta(n^2)$ time if *Find* without path compression and arbitrary *Union* are performed.

   (b) Show a *Union-Find* program of size n that requires $\Theta(n \lg n)$ time if *Find* without path compression and size-weighted *Union* are performed.

2. There is well-known data structure for *union* and *find* operations that supports these operations in in worst-case time $O(\log n)$ and amortized time $O(\log^* n)$.

   Design a data structure that can perform a sequence m *union* and *find* operations on a universal set of n elements, consisting of a sequence of all *unions* followed by a sequence of all *finds*, in time $O(m + n)$. (*)

2.4 Trees and Other Dictionary Structures

1. Design a dictionary data structure in which insertions, deletions and membership queries can be all processed in $O(1)$ time in the worst case. You may assume the set elements are integers drawn from a finite set $1, 2, .., n$, and initialization can take $O(n)$ time.

2. Describe how to modify any balanced tree data structure such that search, insert, delete, minimum, and maximum still take $O(\lg n)$ time each, but successor and predecessor now take $O(1)$ time each. Which operations have to be modified to support this?

   (*)

3. Suppose you have access to a balanced dictionary data structure, which supports each of the operations search, insert, delete, minimum, maximum, successor, and predecessor in $O(\log n)$ time. Explain how to modify the insert and delete operations so they still take $O(\log n)$ but now minimum and maximum take $O(1)$ time. (Hint: think in terms of using the abstract dictionary operations, instead of mucking about with pointers and the like.)

(*)

4. You are given the task of reading in n numbers and then printing them out in sorted order. Suppose you have access to a balanced dictionary data structure, which supports each of the operations search, insert, delete, minimum, maximum, successor, and predecessor in $O(\log n)$ time.

   - Explain how you can use this dictionary to sort in $O(n \log n)$ time using only the following abstract operations: minimum, successor, insert, search.
   - Explain how you can use this dictionary to sort in $O(n \log n)$ time using only the following abstract operations: minimum, insert, delete, search.
   - Explain how you can use this dictionary to sort in $O(n \log n)$ time using only the following abstract operations: insert and in-order traversal.

   (*)

5. Given the pre-order and in-order traversals of a binary tree, is it possible to reconstruct the tree? If so, sketch an algorithm to do it. If not, give a counterexample. Repeat the problem if you are given the pre-order and post-order traversals.

6. Design a data structure to support the following operations:

   - *insert(x,T)* – Insert item x into the set T.
   - *delete(k,T)* – Delete the kth smallest element from T.
   - *member(x,T)* – Return true iff $x \in T$.

   All operations must take $O(\log n)$ time on an n-element set. (*)

7. In the *bin-packing problem*, we are given n metal objects, each weighing between zero and one kilogram. We also have a collection of large, but fragile bins. Our goal is to find the smallest number of bins that will hold the n objects, with no bin holding more than one kilogram.

   - The *any-fit heuristic* for bin packing is as follows. Take each of the objects in the order in which they are given. For each object, place it into *any* partially filled bin which has room. If no such bin exists, start a new bin. Design an algorithm that implements the first-fit heuristic (taking as input the n weights $w_1, w_2, ..., w_n$ and outputting the number of bins needed when using first-fit) in $O(n \log n)$ time.
   - Repeat the above using the *best-fit heuristic*, where we put the next object in the partially filled bin with the smallest amount of extra room *after* the object is inserted.

- Repeat the above using the *worst-fit heuristic*, where we put the next object in the partially filled bin with the largest amount of extra room *after* the object is inserted.

(*)

8. Let $A[1..n]$ be an array of real numbers. Design an algorithm to perform any sequence of the following operations:

   - *Add(i,y)* – add the value y to the ith number.
   - *Partial-sum(i)* – return the sum of the first i numbers, i.e. $\sum_{i=1}^{n} A[i]$.

   There are no insertions or deletions; the only change is to the values of the numbers. Each operation should take $O(\log n)$ steps. You may use one additional array of size n as a work space. (*)

9. Extend the data structure of the previous problem to support insertions and deletions. Each element now has both a *key* and a *value*. An element is accessed by its key. The addition operation applied to the values, but the element are accessed by its key. The Partial_sum operation is different.

   - *Add(k,y)* – add the value y to the item with key k.
   - *Insert(k,y)* – insert a new item with key k and value y.
   - *Delete(k)* – delete the item with key k.
   - *Partial-sum(k)* – return the sum of all the elements currently in the set whose key is less than y, i.e. $\sum_{x_j<y} x_i$.

   The worst case running time should still be $O(n \log n)$ for any sequence of $O(n)$ operations. (**)

10. Suppose that we are given a sequence of n values $x_1, x_2, ..., x_n$ and seek to quickly answer repeated queries of the form: given i and j, find the smallest value in $x_i, \ldots, x_j$.

    (a) Design a data structure that uses $O(n^2)$ space and answers queries in $O(1)$ time.

    (b) Design a data structure that uses $O(n)$ space and answers queries in $O(\log n)$ time. For partial credit, your data structure can use $O(n \lg n)$ space and have $O(\log n)$ query time.

    (*)

2.5 Programming Exercises

3 Sorting and Searching

3.1 Applications of Sorting

1. Newt Gingrich is given the job of partitioning $2n$ players into two teams of n players each. Each player has a numerical rating that measures how good he/she is at the game. Newt seeks to divide the players as *unfairly* as possible, so as to create the biggest possible talent imbalance between team A and team B. Show how Newt can do the job in $O(n \lg n)$ time.

2. Given two sets S_1 and S_2 (each of size n), and a number x, describe an $O(n \log n)$ algorithm for finding whether there exists a pair of elements, one from S_1 and one from S_2, that add up to x. (For partial credit, give a $\Theta(n^2)$ algorithm for this problem.)

3. Given a set of S containing n real numbers, and a real number x. We seek an algorithm to determine whether there are two elements of S whose sum is exactly x.

   (a) Assume that S is unsorted. Give an $O(n \log n)$ algorithm for the problem.

   (b) Assume that S is sorted. Give an $O(n)$ algorithm for the problem.

4. For each of the following problems, give an algorithm that finds the desired numbers within the given amount of time. To keep your answers brief, feel free to use algorithms from the book as subroutines. For the example, $S = \{6, 13, 19, 3, 8\}$, $19 - 3$ maximizes the difference, while $8 - 6$ minimizes the difference.

   (a) Let S be an *unsorted* array of n integers. Give an algorithm that finds the pair $x, y \in S$ that *maximizes* $|x - y|$. Your algorithm must run in $O(n)$ worst-case time.

   (b) Let S be a *sorted* array of n integers. Give an algorithm that finds the pair $x, y \in S$ that *maximizes* $|x - y|$. Your algorithm must run in $O(1)$ worst-case time.

   (c) Let S be an *unsorted* array of n integers. Give an algorithm that finds the pair $x, y \in S$ that *minimizes* $|x - y|$, for $x \neq y$. Your algorithm must run in $O(n \lg n)$ worst-case time.

   (d) Let S be a *sorted* array of n integers. Give an algorithm that finds the pair $x, y \in S$ that *minimizes* $|x - y|$, for $x \neq y$. Your algorithm must run in $O(n)$ worst-case time.

5. Give an efficient algorithm to determine whether two sets (of size m and n) are disjoint. Analyze the complexity of your algorithm in terms of m and n. Be sure to consider the case where m is substantially smaller than n.

6. Give an efficient algorithm to compute the union of sets A and B, where $n = \max(|A|, |B|)$. The output should be an array of distinct elements that form the union of the sets, such that appears more than once in the union.

(a) Assume that A and B are unsorted. Give an $O(n \log n)$ algorithm for the problem.

(b) Assume that A and B are sorted. Give an $O(n)$ algorithm for the problem.

7. The *nuts and bolts* problem is defined as follows. You are given a collection of n bolts of different widths, and n corresponding nuts. You can test whether a given nut and bolt together, from which you learn whether the nut is too large, too small, or an exact match for the bolt. The differences in size between pairs of nuts or bolts can be too small to see by eye, so you cannot rely on comparing the sizes of two nuts or two bolts directly. You are to match each bolt to each nut.

   (a) Give an $O(n^2)$ algorithm to solve the nuts and bolts problem.

   (b) Suppose that instead of matching all of the nuts and bolts, you wish to find the smallest bolt and its corresponding nut. Show that this can be done in only $2n-2$ comparisons.

   (c) Show that any algorithm for the nuts and bolts problem must take $\Omega(n \log n)$ comparisons in the worst case. (Hint: sorting n items takes $\Omega(n \log n)$ comparisons in the worst case.)

   (*)

3.2 Quicksort

1. The running time for Quicksort depends upon both the data being sorted and the partition rule used to select the pivot. Although Quicksort is $O(n \log n)$ on average, certain partition rules cause Quicksort to take $\Theta(n^2)$ time if the array is already sorted.

   (a) Suppose we always pick the pivot element to be the key from the *last* position of the subarray. On a sorted array, does Quicksort now take $\Theta(n)$, $\Theta(n \log n)$, or $\Theta(n^2)$?

   (b) Suppose we always pick the pivot element to be the key from the *middle* position of the subarray. On a sorted array, does Quicksort now take $\Theta(n)$, $\Theta(n \log n)$, or $\Theta(n^2)$?

   (c) Suppose we always pick the pivot element to be the key of the *median* element of the *first three keys* of the subarray. (The median of three keys is the middle value, so the median of 5, 3, 8 is five.) On a sorted array, does Quicksort now take $\Theta(n)$, $\Theta(n \log n)$, or $\Theta(n^2)$?

   (d) Suppose we always pick the pivot element to be the key of the *median* element of the first, last, and middle elements of the subarray. On a sorted array, does Quicksort now take $\Theta(n)$, $\Theta(n \log n)$, or $\Theta(n^2)$?

2. Suppose an array A consists of n elements, each of which is *red*, *white*, or *blue*. We seek to sort the elements so that all the *reds* come before all the *whites*, which come before all the *blues* The only operation permitted on the keys are

   - *Examine(A,i)* – report the color of the ith element of A.

- *Swap(A,i,j)* – swap the ith element of A with the jth element.

Find a correct and efficient algorithm for red-white-blue sorting. There is a linear-time solution. (*)

3.3 Mergesort

3.4 Other Sorting Algorithm

1. An *inversion* of a permutation is a pair of elements which are out of order.
   (a) Show that a permutation of n items has at most $n(n-1)/2$ inversions. Which permutation(s) have exactly $n(n-1)/2$ inversions?
   (b) Let P be a permutation and P^r be the reversal of this permutation. Show that P and P^r have a total of exactly $n(n-1)/2$ inversions.
   (c) Use the previous result to argue that the expected number of inversions in a random permutation is $n(n-1)/4$.
2. Show that n positive integers in the range 1 to k can be sorted in $O(n \log k)$ time.
3. We seek to sort a sequence S of n integers with many duplications, such that the number of distinct integers in S is $O(\log n)$.
   (a) Give an $O(n \log \log n)$ worst-case time algorithm to sort such subsequences.
   (b) Why doesn't this violate the $\Omega(n \log n)$ lower bound for sorting?

   (*)

3.5 Searching

1. Consider the numerical 20 questions game. In this game, player 1 thinks of a number in the range 1 to n. Player 2 has to figure out this number by asking the fewest number of true/false questions. Assume that nobody cheats.
   (a) What is an optimal strategy if n in known?
   (b) What is a good strategy is n is not known?
2. Let M be an $n \times m$ integer matrix in which the entries of each row are sorted in increasing order(from left to right) and the entries in each column are in increasing order (from top to bottom). Give an efficient algorithm to find the position of an integer x in M, or to determine that x is not there. How many comparisons of x with matrix entries does your algorithm use in worst case? (*)
3. Prove that a successful sequential search for a random element in an n element list performs, on average, $(n+1)/2$ comparisons.

4. The *mode* of a set of numbers is the number that occurs most frequently in the set. The set $(4, 6, 2, 4, 3, 1)$ has a mode of 4.

   (a) Give an efficient and correct algorithm to compute the mode of a set of n numbers.

   (b) Suppose we know that there is an (unknown) element that occurs $n/2 + 1$ times in the set. Give a worst-case linear-time algorithm to find the mode. For partial credit, your algorithm may run in expected linear time.

   (*)

5. Suppose you are given an input set S of n numbers, and a black box that if given any sequence of real numbers and an integer k instantly and correctly answers whether there is a subset of input sequence whose sum is exactly k. Show how to use the black box $O(n)$ times to find a subset of S that adds up to k. (*)

3.6 Programming Exercises

4 Divide and Conquer

4.1 Binary Search

1. Suppose you are given an array A of n sorted numbers that has been *circularly shifted* k positions to the right. For example, $\{35, 42, 5, 15, 27, 29\}$ is a sorted array that has been circularly shifted $k = 2$ positions, while $\{27, 29, 35, 42, 5, 15\}$ has been shifted $k = 4$ positions.

   - Suppose you know what k is. Give an $O(1)$ algorithm to find the largest number in A.
   - Suppose you *do not* know what k is. Give an $O(\lg n)$ algorithm to find the largest number in A. For partial credit, you may give an $O(n)$ algorithm.

2. Suppose that you are given a sorted sequence of *distinct* integers $\{a_1, a_2, \ldots, a_n\}$. Give an $O(\lg n)$ algorithm to determine whether there exists an index i such at $a_i = i$. For example, in $\{-10, -3, 3, 5, 7\}$, $a_3 = 3$. In $\{2, 3, 4, 5, 6, 7\}$, there is no such i.

   (*)

3. Given an array storing numbers ordered by value, modify the binary search routine to return the position of the first number with value x in the situation where x can appear multiple times in the array. Be sure that your algorithm is $\Theta(\log n)$, that is, do *not* resort to sequential search once an occurrence of x is found. (*)

4. Suppose that S and T are sorted arrays, each containing n distinct elements.

   (a) Give an $O((\lg n)^2)$ algorithm to find the nth smallest of the $2n$ element.

(b) Give a lower bound or faster algorithm for the problem.

(*)

4.2 Recursive Algorithms

1. A max-min algorithm finds both the largest and smallest elements in an array of n values. Design and analyze a divide-conquer max-min algorithm that uses $\lceil 3n/2 \rceil - 2$ comparisons for any integer n. (Hint: first consider the case where n is a power of 2.) (*)

2. Give an efficient divide-and-conquer algorithm to find the kth largest element in the merge of two sorted sequences S_1 and S_2. The best algorithm runs in time $O(\log(\max(m, n)))$, where $|S_1| = n$ and $|S_2| = m$. (*)

3. Given an array of n real numbers, consider the problem of finding the maximum sum in any contiguous subvector of the input. For example, in the array

$$\{31, -41, 59, 26, -53, 58, 97, -93, -23, 84\}$$

the maximum is achieved by summing the third through seventh elements, where $59 + 26 + (-53) + 58 + 97 = 187$. When all numbers are positive, the entire array is the answer, while when all numbers are negative, the empty array maximizes the total at 0.

   - Give a simple, clear, and correct $\Theta(n^2)$-time algorithm to find the maximum contiguous subvector.
   - Now give a $\Theta(n)$-time dynamic programming algorithm for this problem. To get partial credit, you may instead give a *correct* $O(n \log n)$ divide-and-conquer algorithm.

   (*)

4. Let $X = \{x_1, x_3, ..., x_n\}$ be a sequence of arbitrary real numbers. Give a linear time algorithm to find the subsequence of consecutive elements $x_i, x_{i+1}, ..., x_j$ whose product is maximum over all consecutive subsequences. The product of the empty subsequence is defined as 1. Observe that X can contain reals less than 1, including negative numbers. (*)

4.3 Programming Exercises

5 Dynamic Programming

5.1 Greedy Algorithms

1. The natural greedy algorithm for making change of n units using the smallest number of coins is as follows. Give the customer one unit of the highest denomination coin of at most n units, say d units. Now repeat to make change of the remaining $n - d$ units.

   For each of the following nations coinage, establish whether or not this greedy algorithm always minimizes the number of coins returned in change. If so, prove it, if not give a counter example.

   (a) United States coinage, which consists of half dollars (50 cents), quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies.

   (b) English coinage before the decimalization, which consisted of half-crowns (30 pence), florins(24 pence), shillings (12 pence), sixpence (6 pence), threepence (3 pence), pennies (1 pence), half pennies (1/2 pence), and farthings (1/4 pence).

   (c) Portuguese coinage, which includes coins for 1,$2\frac{1}{2}$,5,10,20,25 and 50 escudos. You need only consider change for an integer number of escudos.

   (d) Martian coinage, where the available denominations are 1,$p, p^2, \ldots, p^k$, where $p > 1$ and $n \geq 0$ are integers.

   (*)

2. Consider the problem of storing n books on shelves in a library. The order of the books is fixed by the cataloging system and so cannot be rearranged. Therefore, we can speak of a book b_i, where $1 \leq i \leq n$, that has a thickness t_i and height h_i. The length of each bookshelf at this library is L.

   Suppose all the books have the same height h (ie $h = h_i = h_j$ for all i, j) and the shelves are all separated by a distance of greater than h, so any book fits on any shelf. The greedy algorithm would fill the first shelf with as many books as we can until we get the smallest i such that b_i does not fit, and then repeat with subsequent shelves. Show that the greedy algorithm always finds the optimal shelf placement, and analyze its time complexity.

3. This is a generalization of the previous problem. Now consider the case where the height of the books is not constant, but we have the freedom to adjust the height of each shelf to that of the tallest book on the shelf. Thus the cost of a particular layout is the sum of the heights of the largest book on each shelf.

   - Give an example to show that the greedy algorithm of stuffing each shelf as full as possible does not always give the minimum overall height.

- Give an algorithm for this problem, and analyze its time complexity. Hint: use dynamic programming.

(*)

5.2 Fibonacci Numbers

5.3 Approximate and Exact String Matching

1. The longest common *substring* (not subsequence) of two strings X and Y is the longest string which appears as a run of consecutive letters in both strings. For example, the longest common substring of *photograph* and *tomography* is *ograph*.

   (a) Let $n = |X|$ and $m = |Y|$. Give a $\Theta(nm)$ dynamic programming algorithm for longest common substring based on the longest common subsequence / edit distance algorithm.

   (b) Give a simpler $\Theta(nm)$ which does not rely on dynamic programming.

   (c) (hard) Give an $O(n + m)$ algorithm for longest common substring using suffix trees.

   (*)

2. Let $A = a_1a_2 \cdots a_n$ and $B = b_1b_2 \cdots b_n$ be two character strings of the same length. Give an $O(n)$ algorithm to determine whether B is a cyclic shift of A. B is a cyclic shift of A if there exists an index k, $1 \leq k \leq n$ such that $a_i = b_{(k+1)\mathrm{m}odn}$, for all i, $1 \leq i \leq n$. (Hint: recall that testing whether string S is a substring of T can be done in $O(|S| + |T|)$ time.) (*)

3. Give an $O(n^2)$ algorithm to find the longest montonically increasing sequence in a sequence of n numbers.

   (*)

4. Suppose you are given three strings of characters: X, Y, and Z, where $|X| = n$, $|Y| = m$, and $|Z| = n + m$. Z is said to be a *shuffle* of X and Y iff Z can be formed by interleaving the characters from X and Y in a way that maintains the left-to-right ordering of the characters from each string.

   (a) Show that *cchocohilaptes* is a shuffle of *chocolate* and *chips*, but *chocochilatspe* is not.

   (b) Give an efficient dynamic-programming algorithm that determines whether Z is a shuffle of X and Y. Hint: The values the dynamic programming matrix you construct should be Boolean, not numeric.

   (*)

5.4 Design Problems

1. Consider a city whose streets are defined by an $X \times Y$ grid. We are interested in walking from the upper left-hand corner of the grid to the lower right-hand corner.

   Unfortunately, the city has bad neighborhoods, which are defined as intersections we do not want to walk in. We are given an $X \times Y$ matrix *BAD*, where *BAD[i,j]* = *"yes"* if and only if the intersection between streets i and j is somewhere we want to avoid.

   (a) Give an example of the contents of *BAD* such that there is no path across the grid avoiding bad neighborhoods.

   (b) Give an $O(XY)$ algorithm to find a path across the grid that avoids bad neighborhoods.

   (c) Give an $O(XY)$ algorithm to find the *shortest* path across the grid that avoids bad neighborhoods. You may assume that all blocks are of equal length. For partial credit, give an $O(X^2Y^2)$ algorithm.

   (*)

2. Consider the same situation as the previous problem. We have a city whose streets are defined by an $X \times Y$ grid. We are interested in walking from the upper left-hand corner of the grid to the lower right-hand corner. We are given an $X \times Y$ matrix *BAD*, where *BAD[i,j]* = *"yes"* if and only if the intersection between streets i and j is somewhere we want to avoid.

   If there were no bad neighborhoods to contend with, the shortest path across the grid would have length $(X - 1) + (Y - 1)$ blocks, and indeed there would be many such paths across the grid. Each path would consist of only rightward and downward moves.

   Give an algorithm that takes the array *BAD* and returns the *number* of safe paths of length $X + Y - 2$. For full credit, your algorithm must run in $O(XY)$.

   (*)

3. The *knapsack problem* is as follows: given a set of integers $S = \{s_1, s_2, \ldots, s_n\}$, and a given target number T, find a subset of S which adds up exactly to T. For example, within $S = \{1, 2, 5, 9, 10\}$ there is a subset which adds up to $T = 22$ but not $T = 23$.

   Find counterexamples to each of the following algorithms for the knapsack problem. That is, give an S and T such that the subset is selected using the algorithm does not leave the knapsack completely full, even though a such a solution exists.

   (a) Put the elements of S in the knapsack in left to right order if they fit, i.e. the first-fit algorithm.

   (b) Put the elements of S in the knapsack from smallest to largest, i.e. the best-fit algorithm.

   (c) Put the elements of S in the knapsack from largest to smallest, i.e. the worst-fit algorithm.

Now give a correct dynamic programming algorithm which runs in $O(nT)$ time.

4. *Arbitrage* is the use of discrepancies in currency-exchange rates to make a profit. For example, there may be a small window of time during which 1 U.S dollar buys a 0.75 British pounds, 1 British pound buys 2 Australian dollars, and 1 Australian dollar buys 0.70 US dollars. At such a time, a smart trader can trade one U.S. dollar and end up with $0.75 \times 2 \times 0.7 = 1.05$ U.S. dollars, a profit of 5%. Suppose that there are n currencies $c_1, ..., c_n$ and an $n \times n$ table R of exchange rates, such that one unit of currency c_i buys $R[i, j]$ units of currency c_j. Devise and analyze a dynamic algorithm to determine the maximum value of

$$R[c_1, c_{i1}] \cdot R[c_{i1}, c_{i2}] \cdots R[c_{ik-1}, c_{ik}] \cdot R[c_{ik}, c_1]$$

   Hint: think all-pairs shortest path. (*)

5. We wish to compute the laziest way to dial given n-digit number on a standard push-button telephone using two fingers. We assume that the two fingers start out on the * and # keys, and that the effort required to move a finger from one button to another is proportional to the Euclidean distance between them. Design and analyze an algorithm that computes in time $O(n)$ the method of dialing that involves moving your fingers the smallest amount of total distance. (*)

6. You have n objects that you wish to put in order using the relations " $<$ " and " $=$ ". For example, with three objects 13 different orderings are possible:

$$\begin{array}{lllll} a=b=b & a=b<c & a<b=c & a<b<c & a<c<b \\ a=c<b & b<a=c & b<a<c & b<c<a & b=c<a \\ c<a=b & c<a<b & c<b<a & & \end{array}$$

   Give a dynamic-programming algorithm that can calculate, as a function of n, the number of different possible orderings. Your algorithm should take $O(n^2)$ time and $O(n)$ space.

   (*)

7. A certain string processing language allows the programmer to break a string into two pieces. Since this involves copying the old string, it costs n units of time to break a string of n characters into two pieces. Suppose a programmer wants to break a string into many pieces. The order in which the breaks are made can affect the total amount of time used. For example suppose we wish to break a 20 character string after characters 3,8, and 10. If the breaks are made in left-right order, then the first break costs 20 units of time, the second break costs 17 units of time and the third break costs 12 units of time, a total of 49 steps. If the breaks are made in right-left order, the first break costs 20 units of time, the second break costs 10 units of time, and the third break costs 8 units of time, a total of only 38 steps.

Give a dynamic programming algorithm that, given the list of character positions after which to break, determines the cheapest break cost in $O(n^2)$ time. (*)

8. Consider the problem of examining a string $x = x_1 x_2 \ldots x_n$ of characters from an alphabet on k symbols, and a multiplication table over this alphabet, and deciding whether or not it is possible to parenthesize x in such a way that the value of the resulting expression is a, where a belongs to the alphabet. The multiplication table is neither commutative or associative, so the order of multiplication matters.

	a	b	c
a	a	c	c
b	a	a	b
c	c	c	c

For example, consider the following multiplication table and the string $bbbba$. Parenthesizing it $(b(bb))(ba)$ gives a, but $((((bb)b)b)a)$ gives c.

Give an algorithm, with time polynomial in n and k, to decide whether such a parenthesization exists for a given string, multiplication table, and goal element.

(**)

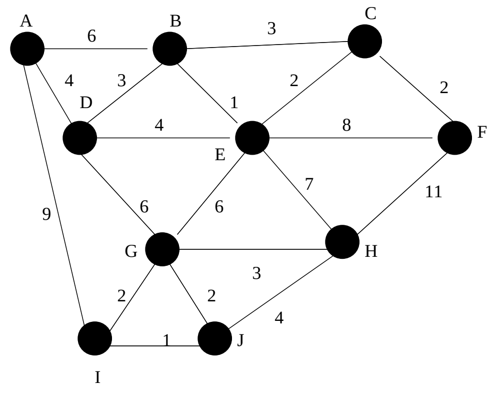
problem

5.5 Programming Exercises

6 Graph Algorithms

6.1 Simulating Graph Algorithms

1. For the following graph G_1:



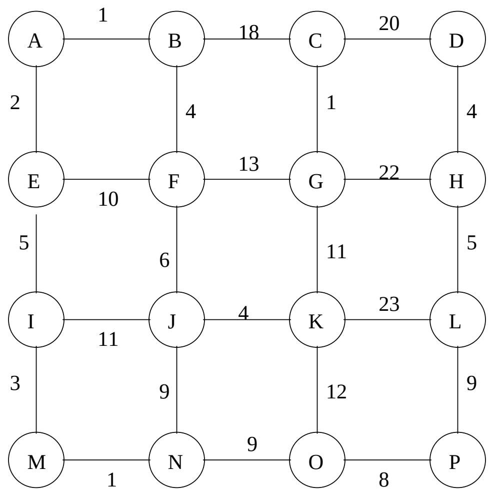
(a) Report the order of the vertices encountered on a breadth-first search of G_1 starting from vertex A. Break all ties by picking the vertices in alphabetical order (i.e A before Z).

(b) Report the order of the vertices encountered on a depth-first search of G_1 starting from vertex A. Break all ties by picking the vertices in alphabetical order (i.e A before Z).

(c) Find the minimum spanning tree of G_1.

(d) Find the shortest path spanning tree of G_1 rooted in A.

2. For the following graph G_2:



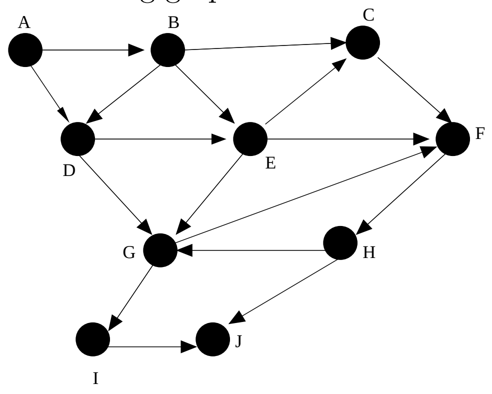
(a) Report the order of the vertices encountered on a breadth-first search of G_2 starting from vertex A. Break all ties by picking the vertices in alphabetical order (i.e A before Z).

(b) Report the order of the vertices encountered on a depth-first search of G_2 starting from vertex A. Break all ties by picking the vertices in alphabetical order (i.e A before Z).

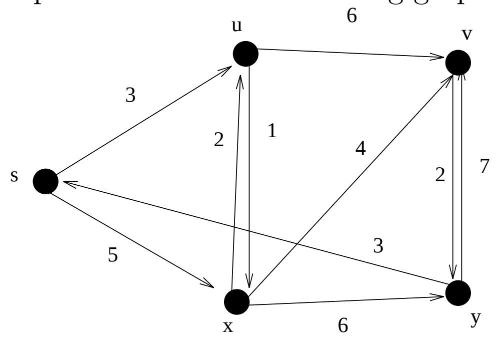
(c) Find the minimum spanning tree of G_2.

(d) Find the shortest path spanning tree of G_2 rooted in A.

3. Do a topological sort of following graph G:



4. Give two more shortest path trees for the following graph:



(*)

6.2 Graph Theory

1. A connected graph is *vertex biconnected* if there is no vertex whose removal disconnects the graph. A connected graph is *edge biconnected* if there is no edge whose removal disconnects the graph.

   Give a proof or counterexample for each for the following statements:

   (a) A vertex biconnected graph is edge biconnected.

   (b) An edge biconnected graph is vertex biconnected.

2. Given two spanning trees T and R of graph $G = (V, E)$, show how to find the shortest sequence of trees $T_0, T_1, \ldots, T_k$ such that $T_0 = T$, $T = R$, and each tree T_i differs from the previous one T_{i-1} by an addition of one edge and a deletion of one edge. (*)

3. A *matching* of an undirected graph $G = (V, E)$ is a set of edges no two of which have a vertex in common. A *perfect matching* is a matching in which all vertices are matched.

   (a) Construct a graph G with $2n$ vertices and n^2 edges such that G has an an exponential number of perfect matchings.

   (b) Construct a graph G with $2n$ vertices and n^2 edges such that G has exactly one unique perfect matching.

   (**)

4. Consider an $n \times n$ board of checkerboard B of alternating black and white squares. Assume that n is even. We seek to cover this checkerboard with rectangular dominos of size 2×1.

   (a) Show how to cover the board with $n \times n/2$ dominos.

   (b) Remove the upper left and lower right corners from B. Show that you *cannot* cover the remaining board with $n \times n/2 - 1$ dominos.

(c) Remove one arbitrary black square and one arbitrary white square from B. Show that the rest of the board can be covered with $n \times n/2 - 1$ dominos. (Hint: think about a Hamiltonian cycle in the dual graph of B).

(*)

6.3 BFS / DFS

1. Give a depth-first search algorithm which outputs the list of the edges in the depth-first search tree.

2. Present correct and efficient algorithms to convert between the following graph data structures, for an undirected graph G with n vertices and m edges. You must give the time complexity of each algorithm.

   (a) Convert from an adjacency matrix to adjacency lists.

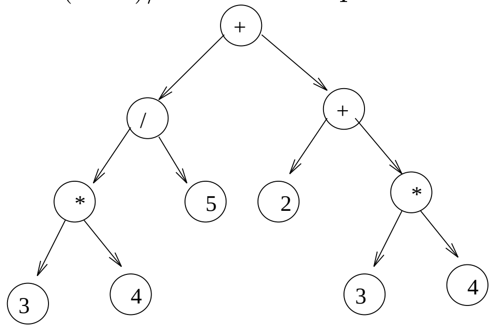
   (b) Convert from an adjacency list to an incidence matrix. An incidence matrix M has a row for each vertex and a column for each edge, such that $M[i, j] = 1$ if vertex i is part of edge j, otherwise $M[i, j] = 0$.

   (c) Convert from an incidence matrix to adjacency lists.

3. Prove that if G is an undirected connected graph, then each of its edges is either in the depth-first search tree or is a back edge.

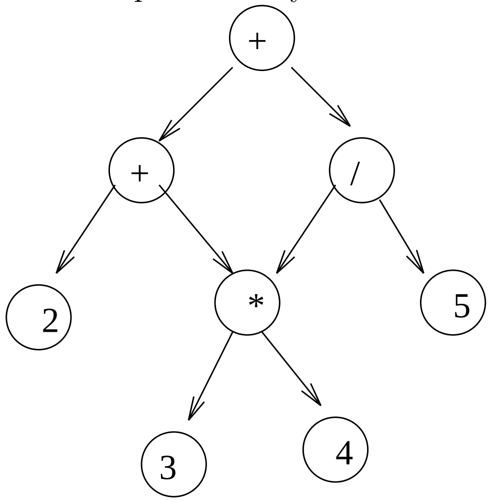
4. Suppose G is a connected undirected graph. An edge e whose removal disconnects the graph is called a *bridge*. Must every bridge e be an edge in a depth-first search tree of G, or can e be a back edge? Give a proof or a counterexample.

5. Suppose an arithmetic expression is given as a tree. Each leaf is an integer and each internal node is one of the standard arithmetical operations $(+, -, *, /)$. For example, the expression $2 + 3 * 4 + (3 * 4)/5$ could be represented by the tree in Figure



   Give an $O(n)$ algorithm for evaluating such an expression, where there are n nodes in the tree.

6. Suppose an arithmetic expression is given as a DAG (directed acyclic graph) with common subexpressions removed. Each leaf is an integer and each internal node is one of the standard arithmetical operations $(+, -, *, /)$. For example, the expression $2 + 3 * 4 + (3 * 4)/5$ could be represented by the DAG below:



Give an $O(n + m)$ algorithm for evaluating such a DAG, where there are n nodes and m edges in the DAG. Hint: modify an algorithm for the tree case to achieve the desired efficiency.

(*)

7. (a) Give a linear-time algorithm to determine whether an undirected graph on n vertices contains a cycle.

   (b) Give a linear-time algorithm to determine whether a directed graph on n vertices contains a cycle. How does this differ from the previous algorithm?

8. A *bipartite* graph is a graph whose vertices can be partitioned into two subsets such that there is no edge between any two vertices in the same subset. Give a linear-time algorithm to determine if a graph G is bipartite.

9. Give a linear algorithm to compute the chromatic number of graphs where each vertex has degree at most 2. Must such graphs be bipartite?

10. The problem of testing whether a graph G contains a Hamiltonian path is NP-hard, where a Hamiltonian *path* P is a path that visits each vertex exactly once. There does not have to be an edge in G from the ending vertex to the starting vertex of P, unlike in the Hamiltonian cycle problem.

    Given a directed acyclic graph G (a DAG), give an $O(n + m)$-time algorithm to test whether or not it contains a Hamiltonian *path*. (Hint: think about topological sorting and DFS.)

    (*)

11. Design a linear-time algorithm to eliminate each vertex v of degree 2 from a graph by replacing edges (u, v) and (v, w) by an edge (u, w). We also seek to eliminate multiple copies of edges by replacing them with a single edge. Note that removing multiple copies of an edge may create a new vertex of degree 2, which has to be removed, and that removing a vertex of degree 2 may create multiple edges, which must be also be removed.

12. Your job is to arrange n rambunctious children in a straight line, facing front. You are given a list of m statements of the form "i hates j". If i hates j, then you do not want put i somewhere behind j, because then i is capable of throwing something at j.

    (a) Give an algorithm that orders the line, (or says that it is not possible) in $O(m+n)$ time.

    (b) Suppose instead you want to arrange the children in rows, such that if i hates j then i must be in a lower numbered row than j. Give an efficient algorithm to find the minimum number of rows needed, if it is possible.

    (*)

13. A *mother* vertex in a directed graph $G = (V, E)$ is a vertex v such that all other vertices G can be reached by a directed path from v.

    (a) Give an $O(n + m)$ algorithm to test whether a given vertex v is a mother of G, where $n = |V|$ and $m = |E|$.

    (b) Give an $O(n + m)$ algorithm to test whether graph G contains a mother vertex.

    (*)

14. An articulation vertex of a graph G is a vertex whose deletion disconnects G. Let G be a graph with n vertices and m edges. Give a simple $O(n + m)$ algorithm for finding a vertex of G that is *not* an articulation vertex, ie whose deletion does not disconnect G.

    (*)

15. Following up on the previous problem, give an $O(n+m)$ algorithm that finds a deletion order for the n vertices such that no deletion disconnects the graph. (Hint: think DFS/BFS.) (*)

6.4 Minimum Spanning Tree

1. Prove that if the weights of the edges of a connected graph G are distinct, then G has a unique minimum spanning tree.

2. Can Prim's and Kruskal's algorithm yield different minimum spanning trees? Explain why or why not.

3. Does either Prim's and Kruskal's algorithm work if there are negative edge weights? Explain why or why not.

4. Is the path between a pair of vertices in a minimum spanning tree necessarily a shortest path between the two vertices in the full graph? Give a proof or a counterexample.

5. Assume that all edges in the graph have distinct edge weights (ie no pair of edges have the same weight). Is the path between a pair of vertices in a minimum spanning tree necessarily a shortest path between the two vertices in the full graph? Give a proof or a counterexample.

6. Suppose we are *given* the minimum spanning tree T of a given graph G (with n vertices and m edges) and a new edge $e = (u, v)$ of weight w that we will add to G. Give an efficient algorithm to find the minimum spanning tree of the graph $G + e$. Your algorithm should run in $O(n)$ time to receive full credit, although slower but correct algorithms will receive partial credit. (*)

7. In the minimum *product* spanning tree problem, the cost of a tree is the product of all the edge weights in the tree, instead of the sum of the weights. You may assume that all edges have positive weight.

   (a) Give a graph whose minimum product spanning tree is different than the minimum weight spanning tree.

   (b) Give an efficient algorithm to compute the minimum product spanning tree. (Hint: think logarithms).

   (*)

8. Modify Prim's algorithm so that it runs in time $O(n \log k)$ on a graph that has only k different edges costs. (*)

6.5 Shortest Path

1. The *single-destination shortest path* problem for a directed graph is to find the shortest path *from* every vertex to a specified vertex v. Give an efficient algorithm to solve the single-destination shortest paths problem.

2. Let $G = (V, E)$ be an undirected weighted graph, and let T be the shortest-path spanning tree rooted at a vertex v. Suppose now that all the edge weights in G are increased by a constant number c. Is T still the shortest-path spanning tree from v?

3. Can we modify Dijkstra's algorithm to solve the single-source *longest* path problem by changing *minimum* to *maximum*? If so, then prove your algorithm correct. If not, then provide a counterexample. (*)

4. Let $G = (V, E)$ be a weighted acyclic directed graph with possibly negative edge weights. Design a linear-time algorithm to solve the single-source shortest-paths problem from a given source v. (*)

6.6 Design Problems

1. A matching in a graph is a set of disjoint edges, i.e. edges which do not share any vertices in common. Give a linear-time algorithm to find a maximum matching in a tree.

2. Given an undirected graph G with n vertices and m edges, and an integer k, give an $O(m+n)$ algorithm that finds the maximum induced subgraph H of G such that each vertex in H has degree $\geq k$, or prove that no such graph exists. An induced subgraph $F = (U, R)$ of a graph $G = (V, E)$ is a subset of U of the vertices V of G, and all edges R of G such that both vertices of each edge are in U.

   (*)

3. The *square* of a directed graph $G = (V, E)$ is the graph $G^2 = (V, E^2)$ such that $(u, w) \in E^2$ iff for some $v \in V$, both $(u, v) \in E$ and $(v, w) \in E$; ie. there is a path of exactly two edges.

   Give efficient algorithms for both adjacency lists and matricies.

   (*)

4. A *vertex cover* of an undirected graph $G = (V, E)$ is a set of vertices U such that each edge in E is incident on at least one vertex of U.

   (a) Give an efficient algorithm to find a minimum-size vertex cover if G is a tree.

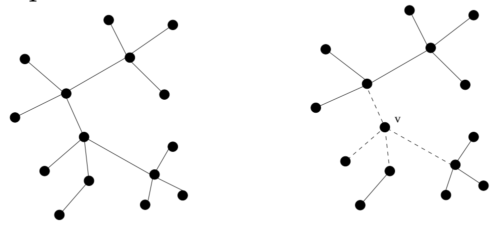
   (b) Let $G = (V, E)$ be a tree with weights associated with the vertices such that the weight of each vertex is equal to the degree of that vertex. Give an efficient algorithm to find a minimum-weight vertex cover of G.

   (c) Let $G = (V, E)$ be a tree with arbitrary weights associated with the vertices. Give an efficient algorithm to find a minimum-weight vertex cover of G.

5. An *interval graph* is an undirected graph $G = (V, E)$ whose vertices correspond to intervals on the real line, where each interval is specified by a leftmost value v_1 and a rightmost value v_2. Two vertices in G are connected iff the corresponding intervals overlap. Let G be an interval graph whose corresponding intervals are provided. Give an efficient algorithm to find a maximum independent set in G.

6. Let $G = (V, E)$ be a binary tree. The distance between two vertices in G is the length of the path connecting these two vertices, and the *diameter* of G is the maximal distance over all pairs of vertices. Give a linear-time algorithm to find the diameter of a given tree. (*)

7. When a vertex and its incident edges are removed from a tree, a collection of subtrees remains, as in the example below:



Give a linear-time algorithm that, for any tree with n vertices, finds a vertex whose removal leaves no subtree with more than $n/2$ vertices. (*)

problem

6.7 Other Graph Problems

6.8 Programming Exercises

7 Special Topics

7.1 Lower Bounds

1. Mr. B. C. Dull claims to have developed a new data structure for priority queues that supports the operations *Insert*, *Maximum*, and *Extract-Max*, all in $O(1)$ worst-case time. Prove that he is mistaken. (Hint: the argument does not involve a lot of gory details–just think about what this would imply about the $\Omega(n \log n)$ lower bound for sorting.)

   (*)

2. Show that there is no sorting algorithm which sorts at least $(1/2^n) \times n!$ instances in $O(n)$ time. Show that there is no sorting algorithm which sorts at least $(1/2^n) \times n!$ instances in $O(n)$ time. (*)

7.2 Computational Geometry

7.3 Programming Exercises

8 Backtracking and Combinatorial Search

8.1 Backtracking

8.2 Combinatorial Generation

8.3 Programming Exercises

9 Intractable Problems and Approximation Algorithms

9.1 Reductions

1. Give the 3-SAT formula that results from applying the reduction of SAT to 3-SAT for the formula:

$$(x + y + \overline{z} + w + u + \overline{v}) \cdot (\overline{x} + \overline{y} + z + \overline{w} + u + v) \cdot (x + \overline{y} + \overline{z} + w + u + \overline{v}) \cdot (x + \overline{y})$$

2. Draw the graph that results from the reduction of 3-SAT to vertex cover for the expression

$$(x + \overline{y} + z) \cdot (\overline{x} + y + \overline{z}) \cdot (\overline{x} + y + z) \cdot (x + \overline{y} + \overline{x})$$

3. The *baseball card collector problem* is as follows. Given packets $P_1, \ldots, P_m$, each of which contains a subset of that year's baseball cards, is it possible to collect all the year's cards by buying $\leq k$ packets?

   For example, if the players are $\{Aaron, Mays, Ruth, Skiena\}$ and the packets are

$$\{\{Aaron, Mays\}, \{Mays, Ruth\}, \{Skiena\}, \{Mays, Skiena\}\},$$

   there does not exist a solution for $k = 2$ but there does for $k = 3$, such as

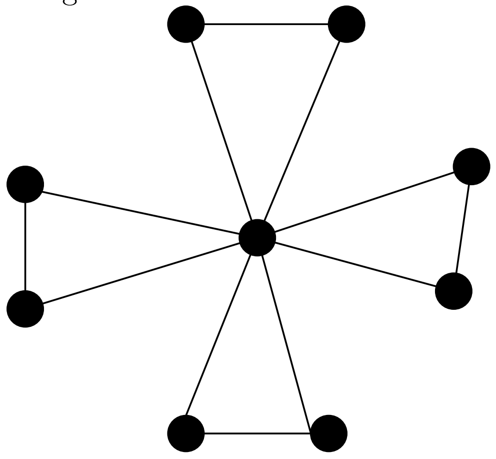
$$\{Aaron, Mays\}, \{Mays, Ruth\}, \{Skiena\}$$

   Prove that the baseball card collector problem is NP-hard using a reduction from vertex cover.

4. An *Eulerian cycle* is a tour that visits every edge in a graph exactly once. An *Eulerian subgraph* is a subset of the edges and vertices of a graph that has an Eulerian cycle. Prove that the problem of finding the number of edges in the largest Eulerian subgraph of a graph is NP-hard. (Hint: the Hamiltonian circuit problem is NP-hard even if each vertex in the graph is incident upon exactly three edges.)

   (*)

5. The *low degree spanning tree problem* is as follows. Given a graph G and an integer k, does G contain a spanning tree such that all vertices in the tree have degree *at most* k (obviously, only tree edges count towards the degree)? For example, in the following graph, there is no spanning tree such that all vertices have degree less than three.



   (a) Prove that the low degree spanning tree problem is NP-hard with a reduction from Hamiltonian *path*.

   (b) Now consider the *high degree spanning tree problem*, which is as follows. Given a graph G and an integer k, does G contain a spanning tree whose highest degree vertex is *at least* k? In the previous example, there exists a spanning tree of highest degree 8. Give an efficient algorithm to solve the high degree spanning tree problem, and an analysis of its time complexity.

6. Prove that subgraph isomorphism is NP-complete.

   (*)

7. Prove that the *clique, no-clique* problem is NP-hard:

   Input: An undirected graph $G = (V, E)$ and an integer k.

   Output: Does G contain a clique of size k and an independent set of size k.

8. (a) Give a linear-time reduction of the problem of finding the maximum element in an array to sorting. What does this reduction tell us about the upper and lower bounds to the problem of finding the maximum element in a sequence?

(b) Can we not reduce the problem of sorting to that of finding the maximum element in linear time?

9.2 Complexity Theory

9.3 Approximation Algorithms

9.4 Programming Exercises

10 Randomized Algorithms

10.1 Random Numbers

10.2 Design Problems

10.3 Programming Exercises

11 Implementation Issues

11.1 The RAM Model

11.2 Thought Problems

11.3 Programming Exercises

12 Gararch's Problems

12.1 Analysis and Asymptotics

1. a. Assume that Christmas has n days. Exactly how many presents did my "true love" send me? [If you do not understand this question, you should do some research.]

   b. Using part (a), exactly how many presents did my "true love" send during the twelve days of Christmas?

   (*)

2. Consider the following code fragment.

```
for i=1 to n do
    for j=i to 2*i do
        output 'foobar'
```

   Let $T(n)$ denote the number of times 'foobar' is printed as a function of n.

   a. Express $T(n)$ as a summation (actually two nested summations).

   b. Simplify the summation. Show your work.

   (*)

3. Consider the following code fragment.

```
for i=1 to n/2 do
    for j=i to n-i do
        for k=1 to j do
            output 'foobar'
```

   Assume n is even. Let $T(n)$ denote the number of times 'foobar' is printed as a function of n.

   (a) Express $T(n)$ as three nested summations.

   (b) Simplify the summation. Show your work.

   (*)

4. For each of the following expressions $f(n)$ find a simple $g(n)$ such that $f(n) = \Theta(g(n))$.

   (a) $f(n) = \sum_{i=1}^{n} \frac{1}{i}$.

   (b) $f(n) = \sum_{i=1}^{n} \lceil \frac{1}{i} \rceil$.

   (c) $f(n) = \sum_{i=1}^{n} \log i$.

(d) $f(n) = \log(n!)$.

(*)

5. Place the following functions into increasing asymptotic order.

   $f_1(n) = n^2 \log_2 n$, $f_2(n) = n(\log_2 n)^2$, $f_3(n) = \sum_{i=0}^{n} 2^i$, $f_4(n) = \log_2(\sum_{i=0}^{n} 2^i)$.

   (*)

6. Place the following functions into increasing asymptotic order. If two or more of the functions are of the same asymptotic order then indicate this.

   $f_1(n) = \sum_{i=1}^{n} \sqrt{i}$, $f_2(n) = (\sqrt{n}) \log n$, $f_3(n) = n\sqrt{\log n}$, $f_4(n) = 12n^{\frac{3}{2}} + 4n$,

   (*)

7. For each of the following expressions $f(n)$ find a simple $g(n)$ such that $f(n) = \Theta(g(n))$. (You should be able to prove your result by exhibiting the relevant parameters, but this is not required for the hw.)

   (a) $f(n) = \sum_{i=1}^{n} 3i^4 + 2i^3 - 19i + 20$.

   (b) $f(n) = \sum_{i=1}^{n} 3(4^i) + 2(3^i) - i^{19} + 20$.

   (c) $f(n) = \sum_{i=1}^{n} 5^i + 3^{2i}$.

   (*)

8. Which of the following are true?

   (a) $\sum_{i=1}^{n} 3^i = \Theta(3^{n-1})$.

   (b) $\sum_{i=1}^{n} 3^i = \Theta(3^n)$.

   (c) $\sum_{i=1}^{n} 3^i = \Theta(3^{n+1})$.

   (*)

9. For each of the following functions f find a simple function g such that $f(n) = \Theta(g(n))$.

   (a) $f_1(n) = (1000)2^n + 4^n$.

   (b) $f_2(n) = n + n\log n + \sqrt{n}$.

   (c) $f_3(n) = \log(n^{20}) + (\log n)^{10}$.

   (d) $f_4(n) = (0.99)^n + n^{100}$.

   (*)

10. For each pair of expressions (A, B) below, indicate whether A is O, o, Ω, ω, or Θ of B. Note that zero, one or more of these relations may hold for a given pair; list all correct ones.

	A	B
(a)	n^{100}	2^n
(b)	$(\lg n)^{12}$	$\sqrt{n}$
(c)	$\sqrt{n}$	$n^{\cos(\pi n/8)}$
(d)	10^n	100^n
(e)	$n^{\lg n}$	$(\lg n)^n$
(f)	$\lg(n!)$	$n \lg n$

(*)

12.2 Data Structures

1. Recall that a heap allows inserts and deletes in $O(\log n)$ steps, and FINDMAX in $O(1)$ steps. Devise a structure that allows inserts and deletes in $O(\log n)$ steps and both FINDMAX and FINDMIN in $O(1)$ steps.

   (*)

2. Devise a Data Structure for a weighted Directed Graph on n nodes that can do the following operations quickly.

   (a) Merge two components,

   (b) locate which component a vertex is in,

   (c) retrieve minimum edge in a component.

   (*)

12.3 Sorting

1. Let $A[1..n]$ be an array such that the first $n - \sqrt{n}$ elements are already sorted (though we know nothing about the remaining elements. Write an algorithm that will sort A in substantially better than $n \log n$ steps.

   (*)

2. For what values of α, $0 \leq \alpha \leq 1$ is the following statement TRUE: Given an array $A[1..n]$ where the first $n - n^\alpha$ are sorted (though we know nothing about the remaining elements), A can be sorted in linear time (i.e., $O(n)$ steps).

   (*)

3. You wish to store a set of n numbers in either a heap or a sorted array. (Heap is MAX-heap, has MAX element on top, etc.) For each of the following desired properties state whether you are better off using a heap or an ordered array, or that it does not matter. Justify your answers.

   (a) Want to find the MAX quickly.

   (b) Want to be able to delete an element quickly.

   (c) Want to be able to form the structure quickly.

   (d) Want to find the MIN quickly.

   (*)

4. Given n elements we are going to, in the course of time, (1) do an add or delete a times, (2) try to find MAX b times, (3) try to find MIN c times. We can assume a, b, c are much smaller than n. For what values of a, b, c are we better off using a heap? For what values are we better off using a sorted array. (You must also take into consideration the time spent forming these structures.)

   (*)

5. Assume that n numbers are in a MAX-heap.

   (a) Show that the third largest element can be found in $O(1)$ steps.

   (b) Show that the ith largest element can be found in $O(i)$ steps

   (*)

6. Assume that the array $A[1..n]$ only has numbers from $\{1, \ldots, n^2\}$ but that at most $\log\log n$ of these numbers ever appear. Devise an algorithm that sorts A in substantially less than $O(n \log n)$.

   (*)

7. Consider the problem of sorting a sequence of n 0's and 1's using comparisons. For each comparison of two values x and y, the algorithm learns which of $x < y$, $x = y$, or $x > y$ holds.

   (a) Give an algorithm to sort in $n - 1$ comparisons in the worst case. Show that your algorithm is optimal.

   (b) Give an algorithm to sort in $2n/3$ comparisons in the average case (assuming each of the n inputs is 0 or 1 with equal probability). Show that your algorithm is optimal.

   (*)

8. Consider an $n \times n$ array A containing integer elements (positive, negative, and zero). Assume that the elements in each row of A are in strictly increasing order, and the elements if each column of A are in strictly decreasing order. (Hence there cannot be two zeroes in the same row or the same column.) Describe an efficient algorithm that counts the number of occurences of the element 0 in A. Analyze its running time.

   (*)

12.4 Graphs

1. For each of the following combinations of properties either exhibit a connected graph on 10 vertices that exhibits these properties, or show that no such graph can exist.

   (a) Eulerian but not Hamiltonian.

   (b) Hamiltonian but not Eulerian.

   (c) Hamiltonian and 2-colorable.

   (*)

2. Describe concisely and accurately a linear-time algorithm for solving the single-source shortest paths problem in a *weighted* DAG. Give a rigorous, concise proof that your algorithm is correct.

   Hints: Topological sort; relaxation lemma; try your alg. on examples.

   (*)

3. Consider the following problem: There is a set of movies $M_1, M_2, \ldots, M_k$. There is also a set of customers, with each one indicating the two movies they would like to see this weekend. (Assume that customer i specifies a subset S_i of the two movies that he/she would like to see. Movies are shown on saturday evening and sunday evening. Multiple movies may be screened at the same time.)

   You need to decide which movies should be televised on saturday, and which movies should be televised on sunday, so that every customer gets to see the two movies they desire. The question is: is there a schedule so that each movie is shown at most once? Design an efficient algorithm to find such a schedule if one exists?

   (One naive solution would be to show all movies on both days, so that each customer can see one desired movie on each day. We would need k channels if k is the number of movies in this solution – hence if there is a solution in which each movie is shown at most once, we would like to find it.)

   (*)

12.5 Dynamic Programming

1. Let $k, n \in N$. The (k, n)-*egg problem* is as follows: There is a building with n floors. You are told that there is a floor f such that if an egg is dropped off the fth floor then it will break, but if it is dropped off the $(f-1)$st floor it will not. (It is possible that $f = 1$ so the egg always breaks, or $f = n+1$ in which case the egg never breaks.) If an egg breaks when dropped from some floor then it also breaks if dropped from a higher floor. If an egg does not break when dropped from some floor then it also does not break if dropped from a lower floor. Your goal is, given k eggs, to find f. The only operation you can perform is to drop an egg off some floor and see what happens. If an egg breaks then it cannot be reused. You would like to drop eggs as few times as possible. Let $E(k, n)$ be the minimal number of egg-droppings that will always suffice.

   (a) Show that $E(1, n) = n$.

   (b) Show that $E(k, n) = \Theta(n^{\frac{1}{k}})$.

   (c) Find a recurrence for $E(k, n)$. Write a dynamic program to find $E(k, n)$. How fast does it run?

   (d) Write a dynamic program that can be used to actually yield the optimal strategy for finding the floor f. How fast does it run?

   (*)

2. Let α and β be constants. Assume that in a tree it costs α to go left, and β to go right. Devise an algorithm that will, given keys $k_1, \ldots, k_n$ which have probabilities of being searched $p_1, \ldots, p_n$, builds a tree with optimal worst case cost.

   (*)

3. The traditional world chess championship is a match of 24 games. The current champion retains the title in case the match is a tie. Not only are there wins and losses, but some games end in a draw (tie). Wins count as 1, losses as 0, and draws as 1/2. The players take turns playing white and black. White has an advantage, because he moves first. Assume the champion is white in the first game, has probabilities w_{w}, w_{d}, and w_{l} of winning, drawing, and losing playing white, and has probabilities b_{w}, b_{d}, and b_{l} of winning, drawing, and losing playing black.

   (a) Write down a recurrence for the probability that the champion retains the title. Assume that there are g games left to play in the match and that the champion needs to win i games (which may end in a 1/2).

   (b) Based on your recurrence, give a dynamic programming to calculate the champion's probability of retaining the title.

   (c) Analyze its running time assuming that the match has n games.

   (*)

12.6 Greedy Algorithms

1. While walking on the beach one day you find a treasure trove. Inside there are n treasures with weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$. Unfortunately you have a knapsack that only holds a total weight M. Fortunately there is a knife handy so that you can cut treasures if necessary; a cut treasure retains its fractional value (so, for example, a third of treasure i has weight $w_i/3$ and value $v_i/3$).

   (a) Describe a $\Theta(n \lg n)$ time greedy algorithm to solve this problem.

   (b) Prove that your algorithm works correctly.

   (c) Improve the running time of your algorithm to $\Theta(n)$.

   (*)

2. Assume that we have a network (a connected undirected graph) in which each edge e_i has an associated bandwidth b_i. If we have a path P, from s to v, then the capacity of the path is defined to be the minimum bandwidth of all the edges that belong to the path P. We define $\mathit{capacity}(s, v) = \max_{P(s,v)} \mathit{capacity}(P)$. (Essentially, $\mathit{capacity}(s, v)$ is equal to the maximum capacity path from s to v.) Give an efficient algorithm to compute $\mathit{capacity}(s, v)$, for each vertex v; where s is some fixed source vertex. Show that your algorithm is "correct", and analyze its running time.
   (Design something that is no more than $O(n^2)$, and with the right data structures takes $O(m \lg n)$ time.)

   (*)

3. A *coloring* of an undirected graph is assignment of colors to the nodes so that no two neighboring nodes have the same color. In the *graph coloring problem*, we desire to find the minimum number of colors needed to color an undirected graph.

   One possible greedy algorithm for this problem is to start with an arbitrary node and color it "1". Then to consider *every* other node one at a time and color it "1" if it does not neighbor a node already colored "1". Now do the same for all of the remaining nodes using the color "2". Continue in this fashion until all of the nodes in the graph are colored.

   (a) How would you implement this algorithm efficiently? Be brief but clear. What is your running time as a function of the number of vertices $|V|$, edges $|E|$, and colors $|C|$.

   (b) Prove that for every graph, the algorithm might get lucky and color it with as few colors as possible.

   (c) Prove that the algorithm can be arbitrarily bad. In other words, for every constant $\alpha > 0$ the ratio between the number of colors used by the greedy algorithm and by the optimal algorithm could be at least α.

   (*)

12.7 NP-Completeness

1. Consider the problem DENSE SUBGRAPH: Given G, does it contain a subgraph H that has exactly K vertices and at least Y edges ? Prove that this problem is NP-complete. You could pick any problem to reduce from.

   (*)

2. Prove that the following problem is NP-complete.
   SPANNING TREE WITH A FIXED NUMBER OF LEAVES: Given a graph $G(V, E)$, and an integer K, does G contain a spanning tree with *exactly* K leaves ?

   (a) First prove that the problem is in NP by describing a "guessing" algorithm for it. (Write out the algorithm in detail.)

   (b) Now prove that the problem is NP-hard, by reducing UNDIRECTED HAMILTONIAN PATH to it.
   HAMILTONIAN PATH asks for a simple path that starts at some vertex and goes to some other vertex, going through each remaining vertex exactly once.

   (*)

3. Assume that you are given access to function CLIQUE(G, K) that returns "True" if G contains a clique of size at least K, and "False" otherwise.

   (a) Use the function CLIQUE to determine, in polynomial time, the size of the largest clique by making. (Try and solve the problem by asking the function CLIQUE as few questions as possible.)

   (b) How can you use the function CLIQUE to *actually* compute a clique that is largest in size ? (Try and solve the problem by asking the function CLIQUE as few questions as possible.)

   (*)

4. (a) Let
$$A = \{G \mid \text{ either } G \text{ or } \overline{G} \text{ has a Hamiltonian Cycle}\}.$$
   Either show $A \in P$ or that A is NP-complete.

   (b) Make up and solve similar problems to the one above where you take some property of graphs and ask if the graph or its compliment has it.

   (*)