CS570
Analysis of Algorithms
Spring 2008
Final Exam (B)

Name: _____

Student ID: _____

Section:__2:00-5:00 or __5:00-8:00

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	15	
Problem 4	20	
Problem 5	10	
Problem 6	20	
Total	100	

Note: The exam is closed book closed notes.

1) 20 pts
Mark the following statements as **TRUE**, **FALSE, or UNKNOWN**. No need to provide any justification.

[TRUE/FALSE]FALSE
In a flow network whose edges have capacity 1, the maximum flow always corresponds to the maximum degree of a vertex in the network**.**

[TRUE/FALSE]FALSE
If all edge capacities of a flow network are unique, then the min cut is also unique.

[TRUE/FALSE]TRUE
A minimum weight edge in a graph G must be in one minimum spanning tree of G.

[TRUE/FALSE]TRUE
When the size of the input grows, any polynomial algorithm will eventually become more efficient than any exponential one.

[TRUE/FALSE/UNKNOWN]FALSE
NP is the class of problems that are not solvable in polynomial time.

[TRUE/FALSE/UNKNOWN]FALSE
If a problem is not solvable in polynomial time, it is in the NP-Complete class.

[TRUE/FALSE/UNKNOWN]TRUE
Linear programming can be solved in polynomial time.

[TRUE/FALSE] FALSE
$10^{2 \log 4n+3} + 9^{2 \log 3n+21}$ is O(n).

[TRUE/FALSE]FALSE
f(n) = O(g(n)) implies g(n) = O(f(n)).

[TRUE/FALSE]FALSE
If X can be reduced in polynomial time to Y and Z can be reduced in polynomial time to Y , then X can be reduced in polynomial time to Z.

2) 15 pts

Suppose you are given a number x and an array A with n entries, each being a distinct number. Also it is known that the sequence of values A[1],A[2], ...,A[n] is **unimodal**. In other words for some unknown index p between 1 and n, we have A[1] < ...<A[p-1]<A[p]>A[p + 1] > ... > A[n]. (Note that A[p] holds the peak value in the array).

Give a algorithm with running time O(log n) to determine if x belongs to A, if yes the algorithm should return the index j such that A[j] = x. You should justify both the correctness of your algorithm and the running time.

Solution:

The idea is to first find out p and then break A into two separated sorted arrays, then use binary search on these two arrays to check if x is belong to A.

Let FindPeak() be the function of finding the peak in A . Then FindPeak(A [1 : n]) works as follows:

Look at A [n / 2], there are 3 cases:

(1) If A [n / 2 - 1] < A [n / 2] < A [n / 2] + 1, then the peak must come strictly after n / 2. Run FindPeak(A [n / 2 : n]).

(2) If A [n / 2 - 1] > A [n / 2] > A [n / 2] + 1, then the peak must come strictly before n / 2. Run FindPeak(A [1 : n / 2]).

(3) If A [n / 2 - 1] < A [n / 2] > A [n / 2] + 1, then the peak is A [n / 2], return n/2.

Now we know the peak index(p value). Then we can run binary search on A [1 : p] and A [p +1 : n] to see if x belong to them because they are both sorted.

In the procedure of finding p, we halve the size of the array in each recurrence. The running time T(n) satisfies T (n) = T (n/2) + O(1) . Thus T (n) = O (log n). Also both binary search has running time at most O (log n), so total running time is O (log n).

3) 15 pts

You are given two sequences a[1],...,a[m] and b[1],...,b[n]. You need to find their longest common subsequence; that is, find a subsequence $a[i_1], ..., a[i_k]$ and $b[j_1],...,b[j_k]$, such that $a[i_1] = b[j_1], ..., a[i_k] = b[j_k]$ with k as large as possible. You need to show the running time of your algorithm.

Solutions:

We use dynamic programming to solve this problem.
Notation X(i)= the prefix sequence <x(1), x(2), …, x(i)>
Let Z be a LCS of X and Y, |Z|=k.
If the last character of X and Y are different, Z is a LCS of either X(m-1) and Y(n) or X(m) and Y(n-1).
If the last character of X and Y are the same, Z(k-1) is a LCS of X(m-1) and Y(n-1)
State Transition Equation:

$$OPT(i, j) = \begin{cases} OPT(i-1, j-1)+1 & if\ x(i) = y(j) \\ \max\{OPT(i-1, j), OPT(i, j-1)\} & if\ x(i) \neq y(j) \end{cases}$$

The largest k is OPT(m, n).

Since computing each OPT(i, j) costs O(1), the running time is O(mn).

4) 20 pts
In Linear Programming, variables are allowed to be real numbers. Consider that you are restricting variables to be only integers, keeping everything else the same. This is called Integer Programming. Integer Programming is nothing but a Linear Programming with the added constraint that variables be integers. Prove that integer programming is NP-Hard

Solutions:

Proof by reduction from Satisfiability

Any SAT instance has boolean variables and clauses. Our Integer Programming problem will have twice as many variables, one for each variable and its compliment, as well as the following inequalities:
$0 \le v_i \le 1$ and $0 \le \neg v_i \le 1$
$1 \le v_i + \neg v_i \le 1$
for each clause $C = \{v_1, \neg v_2, \ldots v_i\} : v_1 + \neg v_2 + \ldots + v_i \ge 1$

1. Any SAT problem has a solution in IP.
In any SAT solution, a TRUE literal corresponds to a 1 in IP since, if the expression is SATISFIED, at least one literal per clause is TRUE, so the inequality sum is > 1.

2. Any IP solution gives a SAT solution.
Given a solution to this IP instance, all variables will be 0 or 1. Set the literals corresponding to 1 as TRUE and 0 as FALSE. No boolean variable and its complement will both be true, so it is a legal assignment with also must satisfy the clauses.

Concluding 1 and 2, Satisfiability $<_p$ Integer Programming. Since Satisfiability is NP-complete, Integer Programming is NP-hard

5) 10 pts

A carpenter makes tables and bookshelves, and he wants to determine how many tables and bookshelves he should make each week for a maximum profit. The carpenter knows that a net profit for each table is $25 and a net profit for each bookshelf is $30. The wooden material available each week is 690 units, its working hours are 120 hours per week. The estimated wood and working hours for making a table are 20 units and 5 hours respectively, while for making a bookshelf are 30 units and 4 hours. Formulate the problem using any technique we have covered in class. You do not need to solve it numerically.

Solutions:

We formulate the problem using linear programming
Suppose that the carpenter decides to make x tables and y bookshelves.

Based on the description in the problem, we want to maximize $25x + 30y$ subject to
$20x + 30y \leq 690$, and
$5x + 4y \leq 120$ where x, y are integers

6) 20 pts
A variation of the satisfiability problem is the MIN 2-SAT problem. The goal in the MIN 2-SAT problem is to find a truth assignment that minimizes the number of satisfied clauses. Give the best approximation algorithm that you can find for the problem.

Solution:

We assume that no clause contains a variable as well the complement of that variable. (Such clauses are satisfied regardless of the truth assignment used.)

We give a 2-approximation algorithm as follows:

Let I be an instance of MINSAT consisting of the clause set C_I and variable set X_I. Construct an auxiliary graph $G_I(V_I, E_I)$ corresponding to I as follows. The node set V_I is in one-to-one correspondence with the clause set C_I. For any two nodes v_i and v_j in V_I, the edge (v_i, v_j) is in E_I if and only if the corresponding clauses c_i and c_j are such that there is a variable x belongs to X_I that appears in uncomplemented form in c_i and complemented form in c_j, or vice versa.

To construct a truth assignment, we construct an approximate vertex cover V' for G_I such that $|V'|$ is at most twice that of a minimum vertex cover for G_I. Then, construct a truth assignment that causes all clauses in $V_I - V'$ to be false. (For a method to find an approximate vertex cover, please refer to section 11.4 in the textbook.)