CS570
Analysis of Algorithms
Summer 2016

Exam III

Name: ______________________

Student ID: __________________

Email Address:________________

_____Check if DEN Student

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

Instructions:
1. This is a 2-hr exam. Closed book and notes
2. If a description to an algorithm or a proof is required please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.

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

[[✓ TRUE]/FALSE/UNKNOWN]
If a problem is in P, it must also be in NP.

[TRUE/FALSE/[✓ UNKNOWN]]
If a problem is in NP, it must also be in P.

[[✓ TRUE]/FALSE/UNKNOWN]
If a problem is NP-complete, it must also be in NP.

[[✓ TRUE]/FALSE/UNKNOWN]
If a problem is NP-complete, it must also be NP-hard.

[TRUE/[✓ FALSE]/UNKNOWN]
If a problem is not in P, it must be NP-complete.

[[✓ TRUE]/FALSE]
If the edge weights of a weighted graph are doubled, then the number of minimum spanning trees of the graph remains unchanged.

[[✓ TRUE]/FALSE]
A greedy algorithm follows the heuristic of making a locally optimum choice at each stage with the hope of finding the global optimum.

[TRUE/[✓ FALSE]]
If $Y \leq_P X$ and we have a 2-approximation algorithm to solve X, we can use that to find a 2-approximation to Y.

[[✓ TRUE]/FALSE]
For a given problem with input size n, there must be some N that when n>N, a $\Theta(n\log n)$ algorithm runs faster than a $\Theta(n^2)$ algorithm.

[TRUE/[✓ FALSE]]
If in a flow network all edge capacities are distinct, then there exists a unique min-cut.

2) 10 pts
A manufacturer makes wooden desks (X) and tables (Y). Each desk requires 2.5 hours to assemble, 3 hours for buffing, and 1 hour to crate. Each table requires 1 hour to assemble, 3 hours to buff, and 2 hours to crate. The firm can do only up to 20 hours of assembling, 30 hours of buffing, and 16 hours of crating per week. Profit is \$3 per desk and \$4 per table. Our objective is to maximize the profit. Formulate this problem as a linear programming problem. You do not have to solve it numerically.

Solution:

x is the number of desks
y is the number of tables

We are to maximize the objective function: profit = $3x + 4y$

subject to the following constraints:
Assembling: $2.5x + y \leq 20$
Buffing: $3x + 3y \leq 30$
Crating: $x + 2y \leq 16$
Non-negativity: $x \geq 0$, $y \geq 0$

3) 20 pts

Suppose you are given two sets A and B, each containing n positive integers. You can choose to reorder each set however you like. After reordering, let a_i be the i^{th} element of set A, and let b_i be the i^{th} element of set B. You then receive a payoff of $\prod_1^n a_i^{b_i}$.

Give an algorithm that will maximize your payoff. Prove that your algorithm maximizes the payoff, and state its running time.

Solution:

Algorithm: sort A and B into monotonically decreasing order.

Proof:
Consider any indices i, j, p, q such that i < j and p < q, and consider the terms ai^bp and aj^bq. We want to show that it is no worse to include these terms in the payoff than to include ai^bq and aj^bp, that is, ai^bp * aj^bq ≥ ai^bq * aj^bp.
Since A and B are sorted into monotonically decreasing order and i < j, p < q, we have ai ≥ aj and bp ≥ bq . Since ai and aj are positive and bp − bq is nonnegative, we have

ai^(bp-bq) ≥ aj^(bp-bq).

Multiplying both sides by ai^bq * aj^bq yields ai^bp * aj^bq ≥ ai^bq * aj^bp.
Since the order of multiplication does not matter, sorting A and B into monotonically increasing order works as well.

Complexity: sorting 2 sets of n positive integers is O(nlogn).

4) 20 pts
In a certain town, there are many clubs, and every adult belongs to at least one club. The townspeople would like to simplify their social life by disbanding as many clubs as possible, but they want to make sure that afterwards everyone will still belong to at least one club. Prove that the Redundant Clubs problem is NP-complete.
PROBLEM NAME: Redundant Clubs
INPUT: List of people; list of clubs; list of members of each club; number K.
OUTPUT: Yes if there exists a set of K clubs such that, after disbanding all clubs in this set, each person still belongs to at least one club. No otherwise.

Solution:
First, we must show that Redundant Clubs is in NP: if we are given a set of K clubs, it is straightforward to check in polynomial time whether each person is a member of another club outside this set.

Next, we reduce from a known NP-complete problem, Set Cover. We translate inputs of Set Cover to inputs of Redundant Clubs, so we need to specify how each Redundant Clubs input Element is formed from the Set Cover instance. We use the Set Cover's elements as our translated list of people, and make a list of clubs, one for each member of the Set Cover family. The members of each club are just the elements of the corresponding family. To finish specifying the Redundant Clubs input, we need to say what K is: we let $K = F - K_{SC}$ where F is the number of families in the Set Cover instance and K_{SC} is the value k from the set cover instance. This translation can clearly be done in polynomial time (it just involves copying some lists and a single subtraction).

Finally, we need to show that the translation preserves truth values. If we have a yes-instance of Set Cover, that is, an instance with a cover consisting of K_{SC} subsets, the other K subsets form a solution to the translated Redundant Clubs problem, because each person belongs to a club in the cover. Conversely, if we have K redundant clubs, the remaining K_{SC} clubs form a cover. So the answer to the Set Cover instance is yes if and only if the answer to the translated Redundant Clubs instance is yes.

5) 20 pts

Consider an array of integers $A = [a_1, . . . , a_n]$ with $a_1 < a_n$.

a) Show that there exists an index $i \in \{1, . . . , n - 1\}$ such that $a_i < a_{i+1}$. In other words, there exist two consecutive entries of the array where the first is smaller than the second. 6 pts

Solution:

Assume not. Then $a_1 \geq a_2 \geq a_3 \geq \ldots \geq a_n$, contradicting the assumption that $a_1 < a_n$.

b) Exhibit an algorithm which finds such a pair of entries in O(log n) time. Your algorithm should output $i \in \{1, . . . , n - 1\}$ such that $a_i < a_{i+1}$. You may assume that the array A has already been loaded into memory. You may also assume that comparing two entries of the array is a primitive operation (i.e., takes constant time). 14 pts

Solution:

```
Function FindIndex(A, i, j)
        if i+1=j then
                return i
        else
                m=(i+j)/2       /*here m is an integer, / is integer division*/
                if A[i] ≥ A[m] then
                        return FindIndex(A, m, j)
                else
                        return FindIndex(A, i, m)
                end
        end
```

The return value of the FindIndex(A, 1, n) is the index we want to find.

6) 10 pts

Suppose you are given a flow network with unit capacity edges, i.e. you have a directed graph G = (V, E), a source vertex s and a sink vertex t, and each edge has capacity 1. You are also given an integer parameter k.

Your goal is to delete a set of k edges from G in order to reduce the maximum s-t flow as much as possible. In other words, you want to find a subset of edges F ⊆ E consisting of exactly k edges to delete from G, giving a new flow network G , and the maximum s-t flow in G is as small as possible.

Describe an efficient algorithm for performing this task. Analyze the complexity of your algorithm.

Solution:

First find a maximum flow in G. This also allows you to find a minimum cut (A, B) where s ∈ A and t ∈ B: consider the residual network associated with a maximum flow, then find all vertices reachable from s in that residual network. That is one set in a minimum cut, the remaining set of vertices form the other set in the cut.

Consider the set of edges E' that cross the cut (A, B), i.e. all of the forwarding edges that start in A and end in B. If E' has k or more edges in it, then the set F can consist of any k edges of E'. If E' has fewer than k edges, then for the set F, we can take all of the edges of E', together with any other edges of G, so that F has exactly k edges in the overall set. We claim that the set F described does the job, and you should convince yourself why that is the case. (Hint: What is the capacity of the cut (A, B) in the graph G obtained from G by deleting the edges in the set F?)

Using the Ford-Fulkerson maximum flow algorithm, this procedure will work in time O(nm) where n is the number of vertices of G and m is the number of edges. (Finding the maximum flow is the time-consuming part here. Then identifying the cut and the edges crossing the cut can be performed in time O(n+m) in the residual graph (by BFS, say) and in time O(m), respectively.)