CS570
Analysis of Algorithms
Summer 2016
Exam I

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	14	
Problem 3	16	
Problem 4	20	
Problem 5	15	
Problem 6	15	
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**. No need to provide any justification.

[/FALSE]
If f (n) = O(n^2) and g(n) = O(n^2), then f (n) = O(g(n)).

[TRUE/]
If f (n) = O(g(n)) and g(n) = O(n^2), then f (n) = O(n^2).

[/FALSE]
Every directed acyclic graph has exactly one topological ordering.

[/FALSE]
Dijkstra's algorithm may not terminate if the graph contains negative-weight edges, i.e. the iterations may continue forever.

[/FALSE]
Given a binary max-heap with n elements, the time complexity to find the second largest element from n elements is O(1).

[/FALSE]
If graph G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of any minimum spanning tree.

[TRUE/]
Consider a weighted directed graph $G = (V, E, w)$ and let X be a shortest s-t path for s, $t \in V$. If we double the weight of every edge in the graph, setting $w'(e) = 2w(e)$ for each $e \in E$, then X will still be a shortest s-t path in (V, E, w').

[TRUE/]
If T is a spanning tree of G and e an edge in G which is not in T, then the graph $T+e$ has a unique cycle.

[TRUE/]
The array [20, 15, 18, 7, 9, 5, 12, 3, 6, 2] forms a binary MaxHeap.

[TRUE/]
The worst case cost of the insert operation in a Fibonacci heap is O(1).

2) 14 pts

In each of the following situations, indicate whether $f = O(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$).

	f(n)	***g(n)***
(a)	$n^{1/2}$	$n^{2/3}$
(b)	$n \log n$	$10n \log 10n$
(c)	$10 \log n$	$\log (n^2)$
(d)	$n^2 / \log n$	$n (\log n)^2$
(e)	$n^{1/2}$	$(\log n)^3$
(f)	$n^{1/2}$	$5\text{^}(\log_2 n)$
(g)	2^n	2^{n+1}

Solution:

(a) $f=O(g)$
(b) $f=\Theta(g)$
(c) $f=\Theta(g)$
(d) $f=\Omega(g)$
(e) $f=\Omega(g)$
(f) $f=O(g)$
(g) $f=\Theta(g)$

3) 16 pts

You are given a set of m time intervals [si , fi], where $1 \leq i \leq m$. You are also given a set of n possible bomb times T = t1, ..., tn. We say that bomb j hits interval i if this bomb time lies within the interval, that is, $tj \in [si , fi]$. Your objective is to determine the minimum number of bombs from T to hit every one of the intervals. You may assume that every interval is hit by at least one bomb, so a solution exists. Present a polynomial-time algorithm to determine a minimum set of bombs to hit all the intervals. Justify your algorithm's correctness.

Solution:

A greedy algorithm:

1) Sort the intervals in non-decreasing order based on the finishing time.
2) Sort the bombs in non-decreasing order based on the time.
3) For the earliest finishing interval among the remaining intervals, pick the bomb lying in the interval with the maximum time.
4) Remove the intervals hit by the above selected bomb.
5) Repeat steps 3 and 4 until we exhaust all the intervals.

Proof of correctness:

Claim: From left to right, our bombs 1 to $k \geq 1$ will be going off no earlier than the corresponding bombs in the optimal solution.

Base case: Our first bomb will go off no earlier than the first bomb in the optimal solution. Reason: leftmost bomb has to take out first interval from left, so it cannot go off after the end of the first interval.

Inductive step:

Hypothesis: Our bomb $k \geq 1$ goes off no earlier than the corresponding bomb in the optimal solution. Need to prove that the $(k+1)^{st}$ bomb in our solution goes off no earlier than the $(k+1)^{st}$ bomb in the optimal solution.

Proof: if k^{th} bomb in the optimal solution is to the left of our k^{th} bomb, then our first k bombs will take out at least the same intervals that are taken out by the optimal solution ,and maybe more (because all those remaining intervals that have a start time before the k^{th} bomb will be taken out by the k^{th} bomb, and since our k^{th} bomb is to the right of the k^{th} bomb in the optimal solution, it can take out potentially more intervals that start after the k^{th} bomb in the optimal solution and before our k^{th} bomb). Now, say our $(k+1)^{st}$ bomb is at the end of the j^{th} interval (i.e. our solution chooses $(k+1)^{st}$ bomb for j^{th} interval). The $(k+1)^{st}$ bomb in the optimal solution will at least have to be at that position or maybe to its left, because it has to cover the j^{th} interval and there may be intervals ending earlier in the optimal solution that have not been covered by the k^{th} bomb in the optimal solution.

Applying this fact to the last bombs in the two solutions, we can see that if we need x bombs to take out all intervals, the optimal solution will need at least x bombs because the x^{th} bomb in our solution is to the right of the x^{th} bomb in the optimal solution (due to above fact) and potentially can take out intervals with later start times than those covered by the x^{th} bomb in the optimal solution.

4) 20 pts

Given a directed graph with positive edge lengths and a source node s and a sink node t, design an efficient algorithm to compute the number of shortest paths from s to t.

Solution:

We modify Dijkstra's algorithm to accomplish this.

For a vertex u, let count(u) denote the number of shortest paths from s to u. Initialize count(s) = 1 and for all $u \neq s$, count(u) = 0.

When an edge (u, v) with u in S, v not in S is relaxed, there are three cases to consider for updating count:

1. If d(v) < d(u) + l(u, v), do nothing.
2. If d(v) > d(u) + l(u, v), then update count(v) = count(u) since we have found a potential shortest path to v through u and u could be reached in count(u) ways.
3. If d(v) = d(u) + l(u, v), then set count(v) += count(u) since we have found another count(u) ways to get to v with the same length.

5) 15 pts

We are given N objects and the distance between them. We can represent these distances as edges in a weighted undirected graph. Now we are interested in "clustering" these objects into groups such that objects within a group have small distances between them, and objects across groups have large distances between them.

One way to cluster the objects is to run Kruskal's algorithm, but stop it before the MST has been formed. Suppose you stop the algorithm after K edges have been added to T. Then answer the following questions:

a. How many clusters (i.e. connected components) will you have? Explain your answer.

Solution:
N - *K* clusters.

Since adding one edge connects two clusters (no edge between objects in the same cluster will be added or a cycle will be introduced), adding one edge will result in one less cluster. For there are originally *N* clusters, adding *K* edges results in *N* - *K* clusters.

b. If the K^{th} added edge has weight W, then what can you say about the distances of the edges across clusters? Why?

Solution:
All clusters have a distance of at least *W* from one another.

Since Kruskal's algorithm adds edges in increasing order of the edge weight, edges between any two unconnected clusters must have weight greater or equal to the weight of the last added edge, which is *W*.

6) 15 pts

Create a data structure with following properties:

1- Each element has an integer key
2- Can insert a new element into the data structure in O(log n)
3- Can find the median key value in the data structure in O(1)

For simplicity, you can assume that the key values are unique. You need to describe how each of the two operations work and analyze their complexity.

If there are an even number of elements in the data structure, you should return the average of the two middle keys as the effective median. For example, the effective median of the set {1,4,5,8} will be 4.5, while the median of the set {1,4,5} will be 4.

Hint: you can use one or more data structures that you are already familiar with to implement this data structure. I.e., you don't have to create the data structure from scratch.

Solution:

We can use a max heap on left side to represent elements that are less than effective median, and a min heap on right side to represent elements that are greater than effective median.

After processing an incoming element, the number of elements in heaps can differ utmost by 1 element. Thus, we need to keep track of the number of elements in both heaps. Insertion is done as follows:

1. If the element to insert is greater than effective median, then insert the element into the min heap; otherwise, insert the element into the max heap.
2. If the number of elements in heaps differs more than 1, extract-max/min from the heap with more elements and insert the extracted element into the other heap.

Since the above two steps takes at most 3 O(log n) operations, the time complexity of insertion is O(log n).

For finding the median key, when both heaps contain same number of elements, we pick average of heaps root data as effective median. When the heaps are not balanced, we select effective median from the root of heap containing more elements. Either case is O(1).