CS570
Analysis of Algorithms
Fall 2015
Exam I

Name: _____

Student ID: _____

Email Address:_____

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	12	
Problem 4	9	
Problem 5	12	
Problem 6	9	
Problem 7	10	
Problem 8	13	
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.

[TRUE/]

In a connected undirected graph, and using the **same** starting point, the depth of any DFS tree is at least as much as the depth of any BFS tree.

[/FALSE]

Algorithm A has a running time of $\Theta(n^2)$ and algorithm B has a running time of $\Theta(n \log n)$. From this we can conclude that A can never run faster than B on the same input set.

[/FALSE]

Let T be a complete binary tree with n nodes. Finding a path from the root of T to a given vertex $v \in T$ using breadth-first search takes O(log n) time.

[TRUE/]

Amortized cost of operations in a Fibonacci heap is at least as good as the worst case cost of those same operations in a binomial heap.

[/FALSE]

Master's Theorem can be used to calculate the running time of any recursive function.

[/FALSE]

Dijkstra's shortest path algorithm can be used to find shortest path in graphs with any edge weights.

[/FALSE]

Function $f(n)= 5n^2 4^n + 6n^4 3^n$ is $O(n^4 3^n)$.

[TRUE/]

Stable Matching: Suppose Jack prefers Rose to others, and Rose prefers Jack to others. The pair (Jack, Rose) exists in every stable matching.

[TRUE/]

A DFS tree is a spanning tree.

[TRUE/]

A binary max-heap can be built using an unsorted list of elements in O(n) time

2) 15 pts

Suppose that you want to get from vertex *s* to vertex *t* in a connected undirected graph G = (V; E) with positive edge costs, but you would like to stop by vertex *u* if it is possible to do so without increasing the length of your path by more than a factor of α.

a- Describe an efficient algorithm in O(|E| log |V|) time that would determine an optimal s-t path given your preference for stopping at *u* along the way if doing so is not prohibitively costly. (In other words, your algorithm should either return the shortest path from *s* to *t*, or the shortest path from *s* to *t* containing *u*, depending on the situation.) If it helps, imagine that there are free burgers at *u*!

b- Show that your algorithm runs in O(|E| log |V|).

Since the graph has positive edges, one can use dijkstra algorithm for the shortest paths computation. We run dijkstra twice, once from s to find the shortest paths s to u and s to t and once from u to find the shortest path u to t. The shortest path from s to t containing u is composed of the shortest path from s to u and the shortest path from u to t. We can now compare the length of this path to the length of the shortest path from s to t, and choose the one to return based on their lengths. Since we are using dijkstra algorithm and the graph is connected the total running time is 2*O(|E| log |V|) which is equivalent to O(|E| log |V|)

3) 12 pts

Assume the coastline of the country Lyniera is an infinite straight line. There are islands off the coastline of Lyniera. In order to keep a close eye on these islands, the king of Lyniera decided to set up some radar bases along the coastline. Each radar base is a point on the coastline which can cover a circular area around itself with radius *d*. Let's use the x-axis in a coordinate system as the coastline (horizontal axis), with the sea on the upper side of the x-axis. Each island is a point in the sea with coordinates (*x*, *y*). Design a greedy algorithm to find the minimum number of radar bases needed to cover all the islands and give their locations on the coastline. Prove the correctness of your algorithm. (Assume each island is within distance *d* of the coast.)

Solution:

For each island, draw a circle around it with radius d and we can get an interval on the x-axis where we are able to place a radar base to cover this island (the intersection of the circle with the x-axis). So, translate the input to a set of intervals along the x-axis. The problem is to find the minimum number of points so that each interval has a point inside it.

Greedy strategy:

Put a radar base at the right-most point of the interval with smallest right end-point which has not been covered yet. Remove the intervals which includes the chosen end point. Recursively do the same thing for the remaining intervals until all the intervals are removed.

Proof:

First, prove the first radar base is put correctly. Let the left-most radar base in the optimal solution S^* has x-coordinate r^*. Then the left-most radar base in our greedy algorithm S has x-coordinate $r \geq r^*$; otherwise, the optimal solution does not cover this corresponding island. Note that, as we placed our radar base at the right-end point of the interval with smallest right end-point, the set of islands covered by the base at r must include the set of islands covered by base at r^*. Thus, we can make an exchange: replace the base at r^* in the optimal solution by the base at r, and we form another strategy S', which should also be an optimal solution because S' still covers all the islands. Therefore, there is an optimal solution that makes the same first choice as the greedy strategy does, so our first choice is correct.

Now consider the small subproblem of covering all islands which have not been covered by our first base. Inductively we can show that all future decisions must also be correct, as they can be rephrased as finding the first base on a smaller subproblem. Thus, our greedy algorithm is optimal.

4) 9 pts

Solve the following recurrences using the Master Method. You do not need to justify your answers, but any justification that you provide will help when assigning partial credit.

i. $T(n)=8T(n/2)+n\log n$

ii. $T(n)=4T(n/2)+n^2(\log n)^2$

iii. $T(n)=2T(n/2)+ n^3(\log n)^3$

i. Case 1 of the Master Method: $T(n)= \Theta(n^3)$

ii. Case 2 of the Master Method: $T(n)= \Theta(n^2(\log n)^3)$

iii. Case 3 of the Master Method to obtain $T(n)= \Theta(n^3(\log n)^3)$, checking that the regularity condition $2f(n/2)=2(n/2)^3(\log(n/2))^3 \leq c\, n^3(\log n)^3$ for some $c<1$. Condition holds for any $c=1/4$

5) 12 pts
In the MERGE-SORT algorithm we merge two sorted lists into one sorted list in O(n) time. Describe an O(n log k)-time algorithm to merge k sorted lists into one sorted list, where n is the total number of elements in all the input lists. Be sure to explain why your algorithm runs in O(n log k)-time.

First we remove the smallest element from each sorted list and we build a min-priority queue (using a min-heap) out of these elements in O(k) time. Then we repeat the following steps: we extract the minimum from the min-priority queue (in O(log k) time) and this will be the next element in the sorted order. From the original sorted list where this element came from we remove the next smallest element (if it exists) and insert it to the min-priority queue (in O(log k) time) . We are done when the queue becomes empty and at this point we have all the numbers in our sorted list. The total running time is O(k + n log k) = O(n log k).

6) 9 pts

Indicate for each pair of expressions (A,B) in the table below, whether A is O, Ω, or Θ of B (in other words, whether A=O(B), A= Ω(B), or A= Θ (B)). Assume that k and C are positive constants. You can mark each box with **Yes** or **No**. No justification needed.

(Note: log is base 2)

A	B	O	Ω	Θ
n^3+logn+n^2	Cn^3	Y	Y	Y
n^2	Cn*2^log(n)	Y	Y	Y
2^n*2^k	n^(2k)	N	Y	N

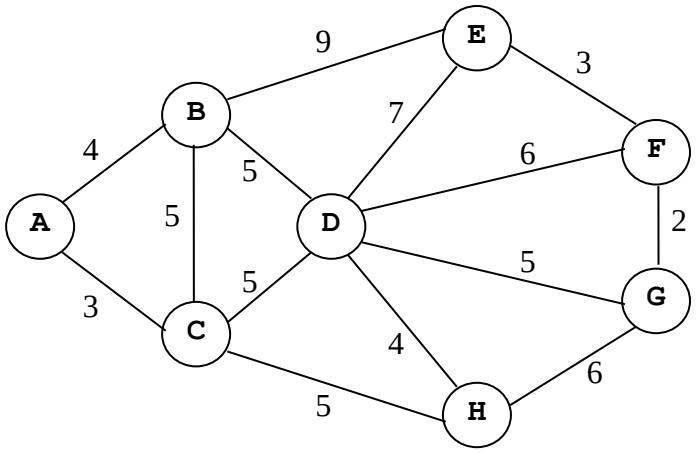
7) 10 pts

Design an algorithm which, given an undirected graph $G = (V, E)$ and a particular edge $e \in E$ determines whether G has a cycle containing e. The running time should be bounded by $O(|V| + |E|)$. Explain why your algorithm runs in $O(|V| + |E|)$ time.

Let s and t denote the ending nodes of the edge e.
Delete e from G to obtain a new graph G', run the DFS or BFS algorithm starting from s, if t can be traversed, then G has a cycle containing e, otherwise G does not have such a cycle.

In the worst case, all the edges and nodes are traversed, resulting in $O(|V| + |E|)$ time; or you can say the running time of DFS or BFS is $O(|V| + |E|)$.

8) 13 pts
Given the undirected graph shown below:



a- Use Prim's algorithm to obtain an MST of this graph. Use A as your starting point. Show the final MST and indicate the order in which you selected the edges

MST: (A,C), (A,B), (B,D), (D,H), (D,G), (G,F), (F,E)

(B,D) could also be (C,D)

b- Use Kruskal's algorithm to obtain an MST. Show the final MST and indicate the order in which you selected the edges

MST: (G,F), (E,F), (A,C), (A,B), (D,H), (B,D),(D,G)

Sequence of (E,F), (A,C), and (D,H), (A,B), and (B,D), (D,G) could be changeable. (B,D) could also be (C,D).

c- Is the MST in this graph unique? If yes, explain why. If no, show edges that can be part of an MST but don't have to be part of every MST.

No. (B,D) , (C,D) , and (C,H)