CS570
Analysis of Algorithms
Summer 2017
Exam I

Name: ______________________
Student ID: __________________
Email Address:_________________

_____Check if DEN Student

	Maximum	Received
Problem 1	20	
Problem 2	15	
Problem 3	15	
Problem 4	15	
Problem 5	15	
Problem 6	20	
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/[✓ FALSE]]
If $f(n) = O(g(n))$, then there is some value of n where $f(n) \geq g(n)$.

[TRUE/[✓ FALSE]]
If $f(n) = O(g(n))$, then there is some value n_0 beyond which $g(n) \geq f(n)$.

[TRUE/[✓ FALSE]]
If G is a strongly connected directed graph, then there is a vertex such that removing that vertex leaves a graph that is also strongly connected.

[[✓ TRUE]/FALSE]
It takes O(n) time to find the minimum element in a binary max heap of n elements.

[[✓ TRUE]/FALSE]
It takes O(log n) time to find the minimum element in a binary min heap of n elements.

[TRUE/[✓ FALSE]]
Given a connected graph G with at least two edges having the same cost, there will be at least two distinct minimum spanning trees in G.

[TRUE/[✓ FALSE]]
In the divide and conquer approach, the sizes of sub-problems at each level are exactly the same.

[TRUE/[✓ FALSE]]
If an algorithm needs to run an operation n times for an input of size n, the time complexity of this algorithm is O(1).

[[✓ TRUE]/FALSE]
For any cycle C in the graph, if the weight of an edge e of C is larger than the individual weights of all other edges of C, then this edge cannot belong to a MST.

[TRUE/[✓ FALSE]]
The Master's theorem can be applied to the following recurrence equation:
$T(n) = T(n/3) + T(n/6) + O(n \lg n)$

2) 15 pts

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle values.

For example:	In a list [2,3,4], the median is 3.
	In a list [2,3], the median is (2+3) / 2 = 2.5

Design an algorithm and its data structures that can accommodate an incoming stream of data with following time complexities:

- Storing one of the elements of the incoming data stream takes O(lg n) time
- Finding the median of all elements input so far takes O(1) time.

We maintain two heaps to store incoming data, one is max heap and one is min heap. The max heap stores the smaller half of data while the min heap stores the larger half of data. The sizes of these two heaps need to be balanced each time when adding a new number so that their size will not be different by more than 1. So, the median of all elements input so far will be the top element of the heap which has one more element (if max-heap and min-heap have different sizes), or the average of the two tops (if max-heap and min-heap have equal sizes).

Now we show how to maintain these two heaps. When adding a new number, there will be four situations:

1) If both heap are empty, just arbitrarily inserted it into a heap
2) If min-heap has more elements, we need to compare the new number with the top of the min-heap. If it is larger than that, then the new number belongs to the larger half and it should be added to the min-heap. But since we have to balance the heap, we should move the top element of the min-heap to the max-heap. For the min-heap, we inserted a new number but removed the original top, its size won't change. For the max-heap, we inserted a new element (the top of the min-heap) so its size will increase by 1.
3) If max-heap has more elements, we did the similar thing as 2).
4) If they have the same size, we just compare the new number with one of the top to determine which heap the new number should be inserted. .

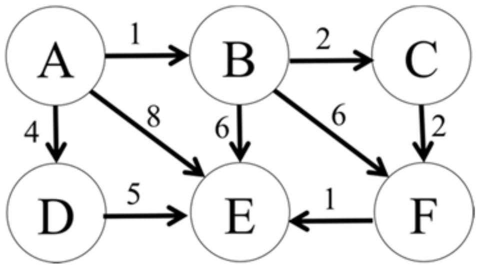
It is obvious that adding a new number takes O(logn) time complexity and finding the median of takes O(1).

Grading:

- The design of max heap and min heap earns 5 points
- Correct adding operation earns 5 points
- Correct finding median operation earns 5 points

3) 15 pts

Run Dijkstra's algorithm on the following directed graph, to compute distances from node A to all the other nodes. Fill in the entries of the table, showing the distance of each node from the starting point after each iteration.



	Iteration					
Node	0	1	2	3	4	5
A	0	0	0	0	0	0
B	∞	1	1	1	1	1
C	∞	∞	3	3	3	3
D	∞	4	4	4	4	4
E	∞	8	7	7	7	6
F	∞	∞	7	5	5	5

Grading:

- Each correct value worth 0.5 points

4) 15 pts

Suppose there are three alternatives for dividing a problem of size *n* into subproblems of smaller size:

1- If you solve 3 subproblems of size *n*/2, then the cost for combining the solutions of the subproblems to obtain a solution for the original problem is $\Theta(n^2\sqrt{n})$

2- If you solve 4 subproblems of size *n*/2, then the cost for combining the solutions is $\Theta(n^2)$

3- If you solve 5 subproblems of size *n*/2, then the cost for combining the solutions is $\Theta(n\ log\ n)$.

Assuming that the problem can be divided into subproblems in constant time, which alternative do you prefer and why?

The first recurrence is $T(n) = 3T(n/2) + \Theta(n^{2.5})$; since $\log_2 3 < 2.5$, the master theorem tells us that $T(n) = \Theta(n^{2.5})$.

The second recurrence is $T(n) = 4T(n/2) + \Theta(n^2)$; since $\log_2 4 = 2$, the master theorem says $T(n) = \Theta(n^2 log\ n)$.

The third recurrence is $T(n) = 5T(n/2) + \Theta(n\ log\ n)$; since $\log_2 5 > 2$, the master theorem says $T(n) = \Theta(n^{log_2 5})$.

Therefore, the second alternative is the best.

Grading:

- Correctly computing the complexity of each option (4 points each)
- Correctly comparing the three complexities (3 points)
    - If the original complexities are not computed correctly, but the final comparison is correct, you'll only get 2 out of 3 points for the comparison.

5) 15 pts

You have been hired to manage the translation process for some documentation. Unfortunately, different sections of the documentation were written in different languages: n languages in total. Your boss wants the entire documentation to be available in all n languages.

There are m different translators for hire. Each translator knows exactly two different languages and can translate back and forth between them. Each translator has a non-negative hiring cost. Unfortunately, your budget is too small to hire one translator for each pair of languages. Instead, you must rely on chains of translators: an English-Spanish translator and a Spanish-French translator, working together, can translate between English and French. Given a document with n languages and m translators where the cost of hiring each translator is given by *c(m)*, give an efficient algorithm that gives the minimum cost of hiring translators that can translate the entire document.

We construct a graph $G = (V, E, w)$ as follows:

- $V =$ For each n languages, we add one vertex such that v_a represents language a.
- $E =$ For each translator who knows languages a and b, we add an edge (v_a, v_b) to the graph.
- $w =$ Assuming edge e_m represents translator m in the graph, we set $w(e_m) = c(m)$ (cost of hiring translator m).

Any spanning tree of G represents one configuration for translating the entire document. More specifically for a spanning tree T, hiring all the translators whose corresponding edges are in T, would suffice to translate the entire document (Between every two nodes (languages) in any spanning tree, there exists a path (chain of translators). The sum of weights of edges in T, gives the cost of hiring the translators whose edges are represented in T. To minimize this cost, we need to find the minimum spanning tree (MST) of G. We can do this efficiently by running Prim's or Kruskal's algorithms which run in $O(m \lg n)$.

Grading:

- Any algorithm that generates the correct output in $O(m \lg n)$ complexity, gets full points:
- Not computing the complexity of your algorithm (- 3 points)
- If your algorithm doesn't output a set of translators that can translate the document, but it is not necessarily the minimum cost set (7.5 points)
- If your algorithm is correct but not efficient (- 5 points)

6) 20 pts

Suppose there are *n* children at the birthday party, and the birthday cake is sliced into *n* slices with sizes s_1, s_2, … s_n (where s_i is the weight of slice *i*) The cake was not perfectly sliced, so no slice has the same weight as others. Given the appetite of each child a_1, a_2, …, a_n (where a_j is a measure of child *j*'s appetite in terms of cake weight), we want to distribute the slices (one slice per child) to fulfill every child's appetite so they won't have to fight over the piñata candy. (said differently, a child's appetite is fulfilled if his/her slice weighs greater than or equal to his/her appetite)

a) Suggest an algorithm to determine whether it is possible for such a distribution of the slices. (8 pts)

A greedy algorithm

1) Sort the size of sliced cakes in ascending order

2) Sort the appetite of children in ascending order

3) Possible if no $a_i > s_i$, otherwise not possible.

Grading:

- If your algorithm is correct but not efficient (- 4 points)
- If your algorithm does not return true/false (- 1 points)

b) Prove the correctness of your solution. (8 pts)

Suppose there is an optimal solution where all children's appetites are fulfilled. We will order children in that solution in ascending order of their appetite (i.e. $a_j > a_i$ if $j>i$. We then define an inversion as follows: child i has a bigger slice than child j but appears before child j. It is obvious that by removing this inversion the solution will still remain optimal (because $s_i > s_j$ so child j will be fulfilled, and since $a_j > a_i$ it must be that $s_j > a_i$ so giving child j's slice to child i will also fulfill child i.)

So, if there is an optimal solution that has inversions in it we can remove all inversions one by one resulting in an optimal solution with no inversions. This solution is the same as our solution since our solution has no inversions. So our solution is also optimal.

Grading:

- Proof of correctness using inversion gets full points.
- Wrong approach with explanation (- 4 points)

c) Determine the complexity of your algorithm (4 pts)

$O(n) + 2*O(n \log n) = O(n \log n)$

Grading:

- If you incorrectly mention the time complexity of sorting (- 2 points)
- If you incorrectly added the time complexity (- 2 points)
- Missing O( ) notation (- 1 point)

Additional Space

Additional Space