CS570 Spring 2019: Analysis of Algorithms Exam I

	Points		Points
Problem 1	20	Problem 5	18
Problem 2	16	Problem 6	12
Problem 3	16	Problem 7	8
Problem 4	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.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.

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

[TRUE/FALSE]
Any depth-first search in a directed graph will produce the same number of tree edges in the DFS tree.

[TRUE/FALSE]
For any connected graph with *n* vertices and *m* edges we can say that m+n = O(m).

[TRUE/FALSE]
In a binomial heap, for a given set of input values, the number of binomial trees in the heap will depend on the order of insertions

[TRUE/FALSE]
The amortized cost of an operation can be as high as the worst-case cost of an operation in a data structure.

[TRUE/FALSE]
A recurrence equation of the type T(n) = a * T(n/b) + f(n) where a * f(n/b) = 2 * f(n) cannot be solved using the Master Theorem.

[TRUE/FALSE]
For a given problem, an algorithm with a Θ (n^2) worst case performance can run faster than an algorithm with a Θ (n log n) worst case performance.

[TRUE/FALSE]
The shortest path between two nodes in a weighted directed graph can be found using BFS if all edges except for those leaving the starting point have the same weight.

[TRUE/FALSE]
Suppose that G is a directed, acyclic graph and has a topological ordering with v_1 the first node in the ordering and v_n the last node in the ordering. Then there is a path in G from v_1 to v_n.

[TRUE/FALSE]
Consider an execution of Kruskal's algorithm on graph G with n nodes where all the first n-1 edges that we pick from the sorted list end up in the MST. Then we can conclude that G has no cycles.

[TRUE/FALSE]
Kruskal's algorithm for minimum weight spanning trees is an example of a greedy algorithm.

2) 16 pts

Consider a special case of the Minimum Spanning Tree problem where the graph G has edge costs of either 1, 3, or 5. We know that we can find an MST in this graph using Prim's algorithm in $O(m \log n)$. Modify Prim's algorithm (and/or the data structures used in Prim's) to create an algorithm that has a worst-case performance strictly better than $O(m \log n)$ on this graph. That is, your modified Prim's must have a worst-case running time of $O(m \log n)$ but not $\theta(m\, log\; n)$.

Solution: Run Prim's algorithm, but instead of having a min-heap, have three buckets for edges, just like in the previous answer. Adding to the not-a-heap takes O(1) and finding min takes O(1). This gives us O(m) time.

Rubric :
If you are using Prim's Algorithm — 2 points
indicating data structure running θ(1) in push/pop/find-min with correct usage (e.g. push/pop elements in the Prim process **instead of** preprocessing) — 4 points
proposing such a correct data structure — 8 points
correct analysis of time complexity — 2 points
Best score you can get: 16 pts

If you are using Kruskal's Algorithm — 0 point
sorting all edges in θ(m) time — 4 points
correct operations of inserting edges — 4 points
mentioning Union-Find data structure (Optimization in P155-P157 of the textbook) — 2 points
correct analysis of time complexity (**θ(α(n))** for each step) — 4 points
Best score you can get: 14 pts

If you are using other algorithms — 0 point
correctly solve the problem in time faster than θ(m*log(n)) — 12 points
correct analysis of time complexity — 2 points
Best score you can get: 14 pts

All self-designed algorithms that fall into Prim/Kruskal's general ideas will be regarded as using them; points will be given **step by step**, i.e. if you fail to achieve a former step, points of following steps will never be given. Partial credit may rarely be given but it depends case by case.

Common Incorrect Example Cases:
a. Fibonacci heap takes θ(m+n*log(n)), but m can be in O(n), so the time complexity is not strictly better than θ(m*log(n)), thus at most 6 points rewarded.
b. General Kruskal takes θ(log(n)) in combining two sets of nodes with U-F data structure, thus overall time complexity is θ(m*log(n)). At most 8 points rewarded. If using **optimized** U-F data structure (P155-P157 in the textbook), then 14 points.
c. Using BFS with edges of length 1, 3, 5 respectively then combining them is an incorrect algorithm. BFS on edges of length 1 then adding edges of length 3 into the graph and BFS again and so for 5 is incorrect, either. Both will get 0 point.

16 pts

Q3. Suppose you are organizing a company party. The corporation has a hierarchical ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node, and so on in this fashion. To keep the party fun for all involved, you will not invite any employee whose immediate superior is invited. Every employee is equally valued, so we would like to maximize the total number of employees invited.

A) Provide a greedy algorithm to solve this problem. (8 pts)

Solution:
Algorithm: invite every leaf node; remove them and parents from the tree, and repeat recursively.

Rubric:
For full credit: Need to use greedy method which does NOT involve using a bipartite graph. Greedy approach involves making a greedy choice at every stage, not in one. The answer needs to indicate the involvement of leaves, not layers
Need to mention: Starts by including leaves. Takes care of leaves on different layers, needs to exclude (delete or skip the immediate parents). Does it recursively till the root is reached.
Partial credits for solutions where the approach yields a result considerably close to the actual solution.
The run time is not considered while marking.

B) Prove that your solution is correct. (8 pts)

Proof of correctness: we need to establish that any solution missing any particular leaf can be made to include that leaf without decreasing the solution quality. The rest of the proof will follow inductively. Consider any leaf node l. Either l or its parent must be invited in an optimal solution: any proposed solution with neither can be easily improved by adding l (as none of l's neighbors its parent is the only neighbor will be included, this is valid to do). If we have any solution that has the parent but not l, we can swap the two (remove the parent from the solution, introduce l into the solution) with a net change of zero to the solution quality (although this may open up the possibility of other omitted vertices, you did not need to note that).
This establishes that any solution missing any leaf can be made to include that leaf without decreasing solution quality, and the rest of the proof follows inductively.

Rubric:
6 points for framing the argument correctly.
2 points for mentioning inductive extension to the whole graph.
If the algorithm is not yielding a correct result or very close to correct result, this proof is not considered since it is essentially proving a different solution (from the required one) correct.

3) 10 pts

Arrange these functions under the O notation using only = or $\subset$ (strict subset of):

E.g. for the functions $n,\ n+1,\ n^2$, the answer should be $O(n+1)=O(n)\subset O(n^2)$.

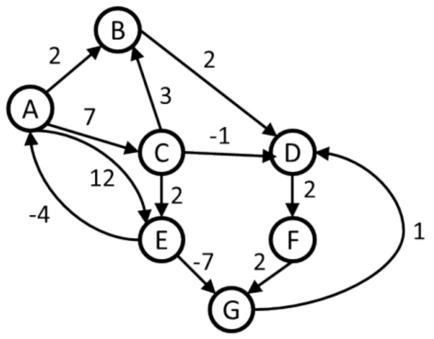
n^3-613, $\ln(n+4)$, $\sin(n^4)$, $\log_2(n^{88n})$, 9999, n^n, $n\log_3(n^3)$, $n^{1.1}$

Solution:

$O(\sin(n^4)) \subset O(9999) \subset O(\ \ln(n+4)\ ) \subset O(\ \log_2(n^{88n})\ ) = O(\ n\log_3(n^3)\ ) \subset O(n^{1.1}) \subset O(N^3-613) \subset O(n^n)$

Rubric: -2 pts for every expression in incorrect order.

4) 18 pts



Consider the following directed, weighted graph:

(a) Even though the graph has negative weight edges, step through Dijkstra's algorithm to calculate the supposedly shortest paths from A to every other vertex. Show your steps in the table below. Start with initial distances in the top row of the table and cross out old values (distances) and write in new ones in the row below whenever distances change.
(6 pts)

A	B	C	D	E	F	G
0	2	7	-	12	-	-
0	2	7	4	12	6	-
0	2	7	4	12	6	-
0	2	7	4	12	6	8
0	2	7	4	9	6	8
0	2	7	4	9	6	8
0	2	7	4	9	6	8

- Each column has +1 Other than column A.
- If only final distances from A are correct, you get 2 points and intermediate steps are not given. No partial points for incorrect distances without intermediate steps
- -1 for additional steps in each column beyond what Dijkstra's would find.

(b) List the vertices in the order which you marked them known. (3 pts)

ABDFCGE

- +3 if all nodes are in order
- -1 for every node not in order (minimum possible score is 0)
- If any node is repeated then -1. For instance solution, ABDFGACE will get only 2 points
- If edges are given instead of nodes, then 0

(c) Dijkstra's algorithm found the wrong path to some of the vertices. For just the vertices where the wrong path was computed, indicate both the path that was computed and the correct path. (6 pts)

Computed path to G is A,B,D,F,G but shortest path is A,C,E,G.
Computed path to D is A,B,D but shortest path is A,C,E,G,D.
Computed path to F is A,B,D,F but shortest path is A,C,E,G,D,F.

- If old path is correctly given +1 for each old path
- If new path is correctly given +1 for each new path
- If only old and new distances are given for ALL the three paths then +2. No partial credit for giving old and new distances for only 1 or 2 paths
- If the old and new paths are interchanged, you get a 0.

(d) What single edge could be removed from the graph such that Dijkstra's algorithm would happen to compute correct answers for all vertices in the remaining graph? (3 pts)

EG

- +3 for EG
- If any other edge is given -1 for each additional edge

5) 12 pts

No partial grading for each part. If answer part is correct, full 3 points, otherwise zero.

Use the Master's Theorem to solve the following recurrence equations (if it applies). Indicate if the Master's Theorem does not apply and give a reason why.

a) $T(n) = 3T(n/4) + O(n)$ (3 pts)

Case 3. $T(n) = O(n)$ / $\Theta(n)$

For $T'(n) = 3T(n/4) + \Theta(n)$, master theorem case 3 gives:

$T'(n) = \Theta(n)$

$O(n) \leq \Theta(n)$, so $T(n) \leq T'(n)$, so the upper bound of $T(n)$ is less than or equal to the upper bound of $T'(n)$.

So $T(n) = O(n)$

$T(n) = \Theta(n)$ strictly is not correct, since the lower bound can not be decided by $O(n)$. But we accept $T(n) = \Theta(n)$ as a correct answer, since this is the original version of the Master Theorem. The same also applies to b) and d).

b) $T(n) = 8T(n/4) + O(n^2)$ (3 pts)

Case 3. $T(n) = O(n^2)$ / $\Theta(n^2)$

c) $T(n) = 2^nT(n/2) + O(n)$ (3 pts)

Does not apply.

d) $T(n) = 9T(n/3) + O(n^2)$ (3 pts)

Case 2 .

$T(n) = O(n^2 \log n)$ / $\Theta(n^2 \log n)$

For $T'(n) = 9T(n/3) + \Theta(n^2)$, master theorem case 2 gives:

$T'(n) = \Theta(n^2 \log n)$

$O(n^2) < \Theta(n^2)$, so $T(n) \leq T'(n)$, so the upper bound of $T(n)$ is less than or equal to the upper bound of $T'(n)$.

So $T(n) = O(n^2 \log n)$

$T(n) = \Theta(n^2 \log n)$ strictly is not correct, since the lower bound can not be decided. But we accept $T(n) = \Theta(n^2 \log n)$ as a correct answer, since this is the original version of the Master Theorem.

6) 8 pts

Assume you are creating an array data structure that has a fixed size of n. You want to backup this array after every so many insertion operations. Unfortunately, the backup operation is quite expensive, it takes exactly n operations to do the backup. Insertions without a backup only takes one operation to perform.

a) What is the amortized cost of the insertion operation (after n insertions) if you perform a backup after every n/5 insertions? (4 pts)

Solution:
Use the aggregate method
Total time to insert all n elements is n + 5n = 6n
Amortized time = 6n/n = O(1)

b) What is the amortized cost of the insertion operation if you perform a backup after every 5 insertions? (4 pts)

Solution:
Use the aggregate method
Total time to insert all n elements is $n + n/5 * n = n + n^2/5$
Amortized time = $(n + n^2/5)/n = O(n)$

Rubric:

-2 points if amortized answer is incorrect, but the total time to insert elements is correctly calculated. -4 otherwise.

It is not important for the students to show how they came up with the answer, e.g., if someone just writes "O(1)" as answer for part a, s/he still gets full points.