CS570 Summer 2018: Analysis of Algorithms Exam I

	Points
Problem 1	20
Problem 2	16
Problem 3	16
Problem 4	16
Problem 5	20
Problem 6	12
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]
Let $d(u,v)$ denote the distance from node u to node v in weighted graph G. If $d(s,u) + d(u,t) = d(s,t)$, then u is on at least one shortest path from s to t. (All weights are positive.)

[TRUE/FALSE]
Let $d(u,v)$ denote the distance from node u to node v in weighted graph G. Suppose that P is a path between u and v and l_p denote the length of P. If $d(s,u) + l_p = d(s,v)$, then P is a shortest path between u and v.

[TRUE/FALSE]
$f(n) = 2n^2 + n \in O(n^4)$.

[TRUE/FALSE]
In every stable matching that Gale–Shapley algorithm may end up with when men propose, there is a man who is matched to his highest-ranked woman.

[TRUE/FALSE]
There is no edge in an undirected graph that jumps more than one level of any BFS tree of the graph.

[TRUE/FALSE]
In an unweighted graph where the distance between any two vertices is at most T, any BFS tree has depth at most T, but a DFS tree might have larger depth.

[TRUE/FALSE]
In a connected undirected graph where all edge weights are equal to 1, any BFS tree is an MST.

[TRUE/FALSE]
A binomial heap of size 17 is constructed using only two binomial trees.

[TRUE/FALSE]
Prim's and Kruskal's algorithms have the same worst case run time complexity when applied to sparse graphs.

[TRUE/FALSE]
The worst case run time complexity of the Insert operation in a Fibonacci heap is O(1).

2) 16 pts.

Suppose you have to choose among three algorithms to solve a problem:

A. Algorithm A solves an instance of size n by recursively solving eight instances of size $n/2$, and then combining their solutions in time $O(n^3)$.
B. Algorithm B solves an instance of size n by recursively solving twenty instances of size $n/3$, and then combining their solutions in time $O(n^2)$.
C. Algorithm C solves an instance of size n by recursively solving two instances of size $2n$, and then combining their solutions in time $O(n)$.

Which algorithm has the best run time complexity?

Solution:
Algorithm C does not even terminate, because it recursively calls itself on instances of increasing size. Hence it does not have a running time (i.e., its running time is "infinite"). By the Master theorem, Algorithm A's running time is $O(n^3 \log n)$, and Algorithm B's running time is $O(n^{\log_3^{20}})$. Since $\log_3^{20} < 3$, Algorithm B is preferable.

Rubric:
Determining Algorithm C never stops: 3 points
Time complexity of Algorithm A: 5 points
Time complexity of Algorithm B: 5 points
Comparison and choosing algorithm B: 3 points

3) 16 pts.

You are given an unsorted array of n distinct integers, along with m queries. Each query is a search for an integer in the array. The output of a query is "found" if the integer is found, or "not found" if the search is unsuccessful. Queries have to be processed in the order they are given.

a) Assuming that $m = \lfloor \sqrt{n} \rfloor$, would you answer each query via a linear search of the unsorted array, or would you rather first sort the array to speed up your searches? Briefly justify your answer. (8 pts)

Doing m linear searches requires $\theta(mn)$ time. On the other hand, sorting the array takes $\theta(n \log n)$ time, and doing m binary searches on the sorted array requires $\theta(m \log n)$ time. Hence sorting is convenient (or equivalent to not sorting) if and only if $m = \Omega(\log n)$. Therefore, if $\boxed{m = \lfloor \sqrt{n} \rfloor}$, sorting is convenient.

Time complexity analysis of sorting: 4 points
Time complexity analysis of linear search and comparison: 4 points

b) What if $m = \lfloor \sqrt{\log n} \rfloor$ instead? Briefly justify your answer. (8 pts)

If $\boxed{m = \lfloor \sqrt{\log n} \rfloor \neq \Omega(\log n)}$, then sorting is not convenient.

Time complexity analysis of sorting: 4 points
Time complexity analysis of linear search and comparison: 4 points

4) 16 pts

We have a new scheduling problem. Each job has a length and a preferred finish time, and we must schedule all the jobs on a single machine without overlapping. (There is a single start time at which all the jobs become available at once.) If we complete a job before the preferred time, we get a reward equal to the time we have saved. If we complete it after the preferred time, we pay a penalty equal to the amount of time that we are late. Our total reward (which we want to maximize) is the sum of all rewards for early completion minus the sum of all penalties for late completion.

a) Argue carefully that there is an optimal schedule that has no idle time (actually, we want to argue that every optimal schedule has no idle time). That is, every optimal schedule always has one job start at the same time the previous job finishes. (6 pts)

Consider any schedule S with a block of idle time of length x in front of some job J. Make a new schedule S' by switching J and the block of idle time, leaving all the other jobs in the same place. The only reward that changes is that of J, which increases by x, so S cannot have been optimal.
If we continue making such swaps with every block of idle time that is in front of a job, we keep increasing the reward and we eventually reach a schedule where there is no idle time because all those blocks occur after all those jobs. This is then a schedule T with no idle time that is better than S.

b) Prove that a greedy algorithm that schedules jobs in non-decreasing order of length without any idle time will always produce an optimal solution. (10 pts)

we consider any schedule that has no idle time (using part a) and is not ordered by length, and consider one of its inversions where job J is just before job I but is longer than I. We improve this schedule by switching I and J, leaving all the other jobs exactly where they are. We increase the reward of I by the length of J, and decrease the reward of J by the length of I. Since J is longer than I, this is a positive net gain. Repeating this process leads us to a schedule that is sorted by length. Only in such a schedule can we not improve the reward by swapping to remove an inversion.
Note that with this reward system, the preferred finish times have no effect on the schedule, because moving a preferred finish time of a job adds or subtracts the same amount to the net reward of any schedule.

Rubric:

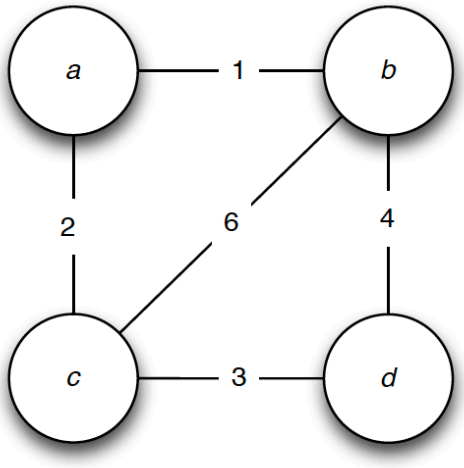
a. Mentions change of reward/penalty with removed idle time:4
   Example to explain: 2
b. Shows the maximization in reward: 8
   Described solution with (inductive and/or inversion) steps: 2

5) 20 pts

Sometimes we need to find spanning trees with additional special properties. Here's an example of such a problem.

Input: Undirected, connected graph $G = (V, E)$ with edge weights w_e. We are also given U which is a subset of vertices of the graph ($U \subset V$).

Output: Find the spanning tree of minimum total weight such that all nodes of U are leaves (there might be other leaves in this tree as well).



a. In the above graph, what is

i) the minimum weight of a spanning tree (4 pts)

6

ii) the minimum weight of a spanning tree in which node a is a leaf. The value of each edge represents the length of that edge (4 pts)

8

b. Give an algorithm for this problem which runs in O(|E| log |E|) time. Hint: You may assume the correctness and running time of any algorithm analyzed in class. (12 pts)

Algorithm:
(i) Use Kruskal's algorithm to find the minimum spanning tree (or forest if the graph is disconnected) of the induced graph on nodes V \ U
(ii) For each node $u \in U$, add incident edge (u, v) with the smallest weight (break ties arbitrarily).
Running Time: The running time is the O(|E| log |E|) since the running time of Kruskal's algorithm is O(|E| log |E|) and it takes O(|E|) time to find the edge of smallest weight for each u.
Correctness: For the correctness, we first argue that, without loss of generality, the optimum solution includes the minimum spanning forest in the induced graph on nodes V \ U. Note that the optimal solution must include a maximal acyclic graph on V \ U since adding only a single edge to each $u \in U$ can not complete a path between two nodes in V \ U. The cheapest maximal acyclic graph is the minimum spanning forest. Since we need to add exactly one edge between V \ U and each $u \in U$, to minimize the total weight we should add the cheapest such edge.

Rubric:

a. i. and ii. Correctly mentions the sum/draws the correct MST with labelled edges: 4 and 4
b. Writing the algorithm correctly: 8
   Running time analysis: 2
   Proof of correctness discussed: 2

6) 12 pts

Suppose we are to receive a set of n jobs, each with a deadline, and we need to schedule them on a single machine. Whenever a job finishes on the machine, we must begin the available job with the earliest deadline. But we may receive the jobs in any order and at any time, and we may not begin a job before we receive it. Indicate an algorithm to schedule the jobs (providing the correct available job whenever a new job is needed) using O(n log n) total computation time for the scheduling. Explain why your algorithm meets this time bound.

Solution:

When we receive a job, we place it in a priority queue based on its deadline. When the machine becomes available, we remove the front job from the queue. We thus have n inserts and n extract-min operations, and the size of the queue never exceeds n. We know that with a heap, we can implement a priority queue so that the insert and extract-min operations take O(log n) time. Thus we have O(n) operations taking O(log n) time each for O(n log n) total time. There also might be O(1) overhead to handle each job, adding another O(n) which goes away when added to O(n log n).

Rubric:

Algorithm 6 points
Time complexity analysis 6 points
If the algorithm is incorrect, you may not receive more than 6 points in total even if the time complexity analysis is correct for the given algorithm.

Additional Space

Additional Space