CS570 Fall 2019: Analysis of Algorithms Exam I

	Points		Points
Problem 1	20	Problem 5	20
Problem 2	15	Problem 6	10
Problem 3	15	Problem 7	10
Problem 4	10		
	Total	**100**	

[redacted]

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]
If path *P* is the shortest path from *u* to *v* and *w* is a node on the path, then the part of path *P* from *u* to *w* is also the shortest path.

[TRUE/FALSE]
Consider an alternate version of the interval scheduling problem where there is a positive reward for each job/interval. The following greedy algorithm always gets the maximum possible reward: Sort the jobs by reward and schedule them one by one starting with the highest reward, rejecting any that overlap with jobs already scheduled.

[TRUE/FALSE]
Dijkstra's algorithm may not terminate if the graph contains negative-weight edges.

[TRUE/FALSE]
If a data structure supports an operation 'X' such that a sequence of n 'X' operations takes $\Theta(nlogn)$ time to perform in the worst case, then the amortized time of the 'X' operation is $\Theta(logn)$, while the actual time of a single 'X' operation could be as high as $\Theta(nlogn)$.

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

[TRUE/FALSE]
Starting with a node u in a graph G, run DFS and BFS to obtain search trees T and T' respectively. The number of children of u in T cannot be greater than the number of children of u in T'.

[TRUE/FALSE]
In class, we proved that a binary heap can be constructed in linear time by showing that the amortized cost of the insert operation in a binary heap is O(1).

[TRUE/FALSE]
Let T and T' be distinct Minimum Spanning Trees of a graph G. Then T and T' must have at least one common edge

[TRUE/FALSE]
Let G be a connected bipartite graph with n nodes and m edges. Then,
$m \log m = O(n^2 \log n)$

[TRUE/FALSE]
We know that algorithm A has a worst case running time of $O(nlogn)$ and algorithm B has a worst case running time of $O(n^2)$. It is possible for algorithm B to run faster than algorithm A when $N \rightarrow \infty$

2) 15 pts
Which of the following equalities are true and why?

a. $3n^2 + 6n = O(n^2 logn)$

True, LS<= $6n^2 \log n$ for any n>=2. Hence holds

b. $3^n = O(2^n)$

False. $3^n = O(\frac{3}{2})^n O(2^n) > O(2^n)$. More precisely given any n_0 and c, $3^n > c * 2^n$ when n>=Max($n_0, \log_{3/2} c$)

c. $\log n = O((\log\log n)^4)$

False. Taking log preserves inequality. If $\log n \leq c((\log\log n)^4)$, then log log n<= log c +4 log log log n. Clearly log log n >= log log log n, hence it is false.

d. $n = O((\log n)^{\log n})$

True. Note that taking log preserves inequality. If n<= $c(\log n)^{\log n}$, then log n = log c + log n(log log n). Clearly this holds.

e. $n^{100} = O(2^n)$

True. n^{100} <= $c(2^n)$, then $\log n^{100}$ =100 log n <= log c +$\log 2^n$ = log c + n log 2. This holds.

Rubric: For each subpart-

1) -3 if you get true/false incorrect
2) +2 if you get the true/false correct
3) -1 if you mention no reasoning or if you do not mention the correct reasoning
4) Using the inequality for growth of functions (logarithmic < polynomial < exponential) is okay
5) Reasoning using limit n->infinity is correct if you get the limit calculations correct

3) 15 pts
Given a $n \times n$ matrix where each of the rows and columns are sorted in ascending order, find the k-th smallest element in the matrix using a min-heap data structure. You may assume that $k < n$. Your algorithm must run in time $O(n + k \log n)$.

Solution:
1. Build a min heap containing the first element of each row and then run deleteMin to return the smallest element. (can also be done with first element of each column)
2. If the element from the i-th row was deleted, then Insert another remaining element in the same row to the min heap. Repeat the procedure k times.
3. The total running time is $O(n + k \log n)$. Minheap is built in $O(n)$. Insert and deleteMin operations both take $\Theta(\log n)$ since there are n elements in the heap. To find the kth smallest elements we require k-1 delete and insert operations.

Rubric:

1) Do not explicitly explain the algorithm; -8 points.
2) Not clear about the min-heap; -5 points.
3) Wrong calculation of time complexity; -5 points.
4) Build the heap without explain the algorithm; < 5 points

4) 10 pts

Prove or disprove the following statement:
For a given stable matching problem, if m and w appear as a pair when men propose and they also appear as a pair when women propose, then m and w must be paired in all possible stable matchings.

Proof: In this case, m and w are each other's best and worst valid partners. This means that they only have one valid partner.

Alternatively: Because m and w are each other's best valid partners, proof by contradiction that they are each other's only valid partners. Assume a stable matching exists where m and w are not paired. We know m is w's best valid partner, so w must prefer m to her current partner. Likewise, w is m's best valid partner, so m must prefer w to his current partner. Thus an instability exists, contradicting our assumption. Thus, no stable matching exists where m and w aren't paired, and thus they are each other's only valid partner.

Rubric:

- Deciding that the statement is true +3pts
- Mentioning that they are each other's best and worst valid partners +4pts
    - If only mention one of these +2pts
- Concluding from that statement that they are each other's only valid partner +3pts
- If any unclear wording and for every erroneous statement -1pt

Rubric (Alternate):

- Deciding that the statement is true +3pts
- Mentioning that they are each other's best valid partners +3pts
- Valid, complete contradiction proof +4pts
- If any unclear wording and for every erroneous statement -1pt

Assuming the statement is false and attempting to disprove: 0pts

5) 20 pts
Tom is looking to buy a new smartphone, and is looking at the upcoming phone releases. Each phone *i* releases to the public at some time t_i and is given software support for some number of years s_i. Tom wants to buy as few phones as possible over the rest of his (unfortunately finite) lifetime. Assuming that we know the date of Tom's demise and all the phone release data until that time,

a) design an efficient algorithm to minimize the number of phones Tom needs to buy for the rest of his life while ensuring that he never goes without an unsupported phone. (10 pts)

b) Prove the correctness of your solution. (10 pts)

Solution + Rubric

I) Algorithm

Sort the phones on t_i. Say t_1 <= t_2 <= t_n
Let T denote the current time, initialized to today. Initialize i=1.

```
While T < Tom's demise
    Var max_i, max_ts;
    While t_i <= T
            IF (t_i + s_i > max_ts) THEN Max_ts = t_i + s_i; max_i = i;
            i++;
    Buy phone P_(max_i); T = max_ts;
```

Complexity: nlogn for the sorting, n for the loop, total n log n.

A text description as simple as this will work: "Starting today, choose from among the available phones that maximizes (t+s). Reset the current time to this t+s and repeat until Tom's Demise".

Alternate solution starts in the opposite direction, i.e. "starting from Tom's demise, choose among phones that have "t+s" more than current time, one that has the lowest t_i. Buy such a phone i and reset current time to t_i."

Rubric:

- Full points even if it just **clearly states the text description.**
  No penalty if complexity missing or complexity O(n^2).
- Deduct a point if the 1st phone picked is the one with earliest t_i, but the rest is correct.
- Deduct a point if it does not say to look for "availability" at the time of buying (i.e. just says maximize "t+s")
- Deduct only a point if it just tries to maximize 's' instead of "t+s" but it's clear that they misunderstood 's' to mean support end date rather than support duration.

- Thus, a solution that roughly has this idea right, gets 8-10 points

- 4/5/6 points for algorithms that intuitively make sense: for example, “latest release” or “max support duration”. (6 if clearly explained, 4 if not clear/has mistakes)
- 3 points for an algorithm that produces a FEASIBLE solution, i.e. a set of phones that covers Tom’s lifetime but is not at all close: for example “earliest release” which will make him buy way too many phones
- 3 points for an algorithm that produces “non-overlapping intervals”, e.g. “Buy a phone with longest support. Then buy a phone immediately AFTER it’s unsupported” etc.
- 1 point minimum (for attempting anything)

2) Proof of correctness:

Let g_1, …, g_k be the phones bought in the greedy solution and h_1, …, h_m be an optimal solution.

Show any of the following by induction:

1. The stay-ahead argument: For all i <= k, (g_1, …, g_i) covers at least as much time as (h_1, …, h_i)

 Here, in the inductive step, we show that, since t+s for phone g_i is bigger than that for h_i (by inductive hypothesis), h_{i+1} is a ‘valid pick’ for the i+1’th step in our algo, which implies t+s is bigger for g_{i+1} since that was maximized. Hence greedy “stays ahead” of OPT after i+1’th step.

2. The inversion argument: For all i <= m, (g_1, …, g_i, h_{i+1}, …, h_m) is an optimal solution

 Here, in the inductive step we argue the following. Since (g_1, …, g_i, h_{i+1}, …, h_m) is an optimal solution, h_{i+1} is a ‘valid pick’ to buy after g_i. Hence, as per our algo, t+s is bigger for g_{i+1} than for h_{i+1}. Hence, if h_{i+1} is swapped with g_{i+1}, h_{i+2} can still be a “valid pick” after, making (g_1, …, g_i, g_{i+1}, h_{i+2}, …, h_m) a feasible solution, but also optimal since it uses m phones.

After having shown either of these, it follows that g_1, …, g_m is an optimal solution. Finally, as per the stop condition of our algo, m cannot be smaller than k.

Rubric:

- Full points for correctly showing the stay-ahead or the inversion argument.
- Deduct 1-2 points each for minor erroneous claims e.g. “optimal solution MUST buy the same 1st phone as the greedy” or “if at a certain stage, a phone expires sooner in a solution S as compared to S’, it must mean S buys more phones or is non-optimal”. Max deductions 3 points as long as the idea is right.
- 7 points if the inductive argument is there but the inductive step is just hand-wavy or not correct.
- 5 points for not showing any formal inductive argument (e.g. just saying h_1 can be swapped with g_1 and saying “we can similarly do this for all phones”)

- When the **algorithm itself is incorrect**: 3-4 points for reasonable inductive proof attempts.
- 1-2 points for any attempt

6) 10 pts

Consider the divide and conquer solution described in class to find the closest pair of points in a 2D plane. Assume that we did not have a driver routine to sort the points. So our recursive function did not receive the points in sorted orders of their X and Y coordinates and the sorting had to be done for each subproblem (at every level). What would be the worst-case complexity of this algorithm assuming that the rest of the algorithm remains the same?

Solution:

Divide step and combine step will now take $\Theta(n \log n)$.

$F(n) = \Theta(n \log n)$

$n^{\wedge}(\log_b a) = n^{\wedge}(\log_2 2) = \Theta(n)$

This will fall under the general version of case 2: So, $T(n) = \Theta(n (\log n)^{\wedge}2)$

Rubric:

- $2T(n/2)$ 2 points
- $F(n)$ 3 points
- Correct use of Master theorem (or any other correct explanation) 5 points
- If f(n) is guessed incorrectly but master theorem is applied correctly: up to 5 points (If you're using case 3, 1 point is for checking the regularity condition: $af(n/b)<kf(n)$)
- Response is correct, analysis is wrong: 5 or 7 points.

10 pts

a) Sam is trying to compute the complexity of merge-sort. In writing the recurrence relation, he erroneously considers the complexity of the merging step to be O(n^2) instead of O(n). Assuming no other mistakes, what is the complexity of merge-sort he ends up with? (5 points)

Solution: T(n) = 2 T(n/2) + O(n^2) => T(n) = O(n^2)

b) Show that the number of nodes in the highest-order binomial tree in a binomial heap with n elements is Θ(n). (5 points)

Solution:

- The number of nodes in the highest order tree < total number of node in the heap, so it is O(n)
- The number of nodes in the highest order tree is at least equal to n/2, so it is Ω(n)
- If it is both O(n) and Ω(n), then it is Θ(n).

In a binomial heap, the highest-order binomial tree, also known as the tree with the most nodes, corresponds to the largest binomial tree. The number of nodes in the largest binomial tree in a binomial heap with n elements is indeed Θ(n).

To show this, let's analyze the structure of binomial trees in a binomial heap:

1. A binomial tree of order k has exactly 2^k nodes. This is because each binomial tree starts with a single node, and when you merge two trees of the same order, you get a tree of the next higher order with twice as many nodes.

2. In a binomial heap, the number of nodes with a specific order (k) is at most one. This ensures that the number of nodes in the highest-order binomial tree is determined by the highest order of any binomial tree in the heap.

3. When you have n elements in a binomial heap, the maximum order (k_max) of any binomial tree is log2(n), as you can't have more than log2(n) trees with distinct orders in the heap. This is because each binomial tree order is a power of 2.

Therefore, the highest-order binomial tree in a binomial heap with n elements has at most 2^(log2(n)) = n nodes. Hence, the number of nodes in the highest-order binomial tree is Θ(n).

Additional Space

Additional Space