CS570 Fall 2025: Analysis of Algorithms Exam I

	Points		Points
Problem 1	18	Problem 5	12
Problem 2	9	Problem 6	17
Problem 3	10	Problem 7	20
Problem 4	14		
			Total 100

First name	
Last Name	
Student ID	

Instructions:

1. This is a 2-hr exam. Closed book and notes. No electronic devices or internet access.
2. A single double sided 8.5in x 11in cheat sheet is allowed.
3. 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.
4. No space other than the pages in the exam booklet will be scanned for grading.
5. 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.
6. Do not detach any sheets from the booklet.
7. 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.
8. Do not write your answers in cursive scripts.
9. This exam is printed double sided. Check and use the back of each page.

Student ID:

1) (18 points)
Mark the following statements as **TRUE** or **FALSE** by circling the correct answer. No need to provide any justification. (2 points each)

[TRUE/FALSE]
In a stable matching problem, when running the Gale Shapley algorithm with men proposing, it is possible for men to end up with their worst valid partner.

[TRUE/FALSE]
Suppose we have functions f(n) and g(n) that take positive values for all n. Then, it is possible that neither 'f(n) = O(g(n))' nor 'g(n) = O(f(n))' holds true.

[TRUE/FALSE]
If all the edge weights in a connected graph are distinct, then the minimum spanning tree (MST) of the graph is guaranteed to be unique.

[TRUE/FALSE]
In a binary heap, the *decrease-key* operation has a worst-case time complexity of $O(\log n)$

[TRUE/FALSE]
If Algorithm A has a worst-case running time of $\Theta(2^n)$ and algorithm B has a worst-case running time of $\Theta(n^2)$, then Algorithm B always runs faster than algorithm A on any given input.

[TRUE/FALSE]
A binary heap is always a complete binary tree.

[TRUE/FALSE]
Dijkstra's algorithm can correctly output the shortest path in graphs with negative edge weights as long as there are no negative cycles.

[TRUE/FALSE]
An undirected graph G has no cycles, and is not a connected graph. Then, G must be a bipartite graph.

[TRUE/FALSE]
Consider a stable matching problem with n men and n women, for n > 2. If man M prefers woman W the least and W prefers M the least as well, then the pair (M,W) cannot be part of a stable matching.

Student ID:

2) (9 points)
For each question below, select **all** correct answers! 3 points each. **No partial credit**.

I. Which is the correct order of the following functions such that g(n) follows f(n) in the list if and only if f(n) = O(g(n))?

(A) $n^{\log n}$, $\log n!$, $\log n^n$, $\log^4 n$

(B) $\log n!$, $\log^4 n$, $n^{\log n}$, $\log n^n$

(C) $\log^4 n$, $\log n!$, $\log n^n$, $n^{\log n}$

(D) $n^{\log n}$, $\log n^n$, $\log^4 n$, $\log n!$

II. If you insert the values [12, 5, 13, 2, 5, 3, 7, 11, 18, 1, 9] into an empty binomial heap,

(A) The number of binomial trees in this binomial heap is 3.
(B) The number of binomial trees in this binomial heap is 4.
(C) The highest order of a binomial tree in this binomial heap is 3.
(D) The highest order of a binomial tree in this binomial heap is 4.

III. Suppose we consider a sequence of n operations, some of which are of type A and others type B. Operation A takes O(log n) time in the worst case, and operation B takes O(n) in the worst case. Then, which of the following is a possible bound on the amortized cost for A and B obtained by the aggregate method on the worst-case sequence of n operations of types A and B

(A) O(log n)
(B) O(√n)
(C) O(n)
(D) $O(n^2)$

Student ID:

3) (10 points)

For each part below, solve the recurrence using the Master Method—if it applies—by giving tight theta-notation bound and briefly showing the steps. If the Master Method does not apply, explain why (no need to solve the recurrence in that case).
Here, T(.) represents the running time of an algorithm.

a) $T(n) = 4\,T(n/2) + n^2 \log^2 n$

Second case: $T(n) = \theta(n^2 \log^3 n)$

b) $T(n) = 100^{100}\,T(n/2) + 3^n$

Third case: $T(n) = \theta(f(n)) = \theta(3^n)$
Regularity condition: $100^{100}\,3^{n/2} \leq c\,3^n$ should be true for some $c < 1$ and large n. This simplifies to $100^{100} / (3^{n/2}) \leq c$ which is true for any $c < 1$ and large n, as the denominator $3^{n/2}$ will grow past any constant.

c) $T(n) = T(n/2) + T(n/4) + n$

Master theorem does not apply since recurrence must have a single term of the form a*T(n/b)

d) $T(n) = 1/2\,T(n/2) + n$

Master theorem does not apply since $a < 1$

Student ID:

4) (14 points)

We are given a sequence of N positive numbers. A subsequence is non-consecutive, if it does not contain a pair of numbers that appear consecutively in the given sequence. Our goal is to find the highest sum possible for a non-consecutive subsequence.

Example: For the sequence [2, **5**, 1, 3, **8**, 10, **8**], the non-consecutive subsequence with the highest sum is [5, 8, 8] as highlighted.

Design an efficient divide-n-conquer based algorithm. Analyze its time complexity (using Master Theorem or otherwise).

Solution: Suppose function f computes the required answer. It is implemented using divide-n-conquer as follows.

f(seq S) {
Suppose size(S) = n
Make cases on whether the mid element S[n/2] is included in the answer.

Case 1) Included.
Then, the elements to the left and right of S[n/2] must be omitted.
Suppose L = S[1, …, n/2 - 2], R = S[n/2 + 2, …, n].
Answer candidate = f(L) + f(R) + S[n/2].

Case 2) Not included.
Then, the elements to the left and right of S[n/2] may be included independently.
Suppose L = S[1, …, n/2 - 1], R = S[n/2 + 1, …, n].
Answer candidate = f(L) + f(R)

Compare both cases and return higher.

Base case: if $n \leq 2$, return the highest element (or 0 if S empty) }

The resultant recurrence is $T(n) = 4\,T(n/2) + O(1)$
Using Master's Theorem, this yields $T(n) = O(n^2)$

Rubric:
Considering the cases on middle element to divide the problem (3 points)
- Other ways of dividing into subproblems may exist, slower algorithms are okay as long as correct and efficient (poly-time).

Correctly define subproblems L, R in each case & compute each answer to be returned (2+2 = 4 points)
Combine step (Compare the cases and return maximum) (2 points)
Base case (1 points)
Correct recurrence (2 points)
Correct runtime from recurrence (2 points)

Student ID:

Important: The rubric applies for solutions that are close to the one above. The most common approaches that do not work, are on the lines of "make two halves and recurse. Attempt to handle consecutive-ness during merging by comparing middle two elements and throwing away one or replacing one with "neighbor" or "second largest element etc. This does not work at all at the time of merging since, removing an element can **completely** change the solution for that half. Therefore, the most critical step is to make the cases mentioned in the solution IN THE DIVIDE STEP, making halves that can be solved independently (which is required for Divide-n-conquer). The approaches described above, will receive a good-faith credit of 5 points at most (or less, in case of other gaps/mistakes as applicable).

Student ID:

5) (12 points)
Imagine you are driving on a k-lane highway. The route can be viewed as consisting of points $p_0, \ldots, p_n$, where p_0 is the starting point, p_n the ending point, and you can switch lanes at any p_i but must stick to a lane between any p_i to p_{i+1}. Thanks to the advanced navigation technology, we have accurate estimates for how long it takes to travel a section in a particular lane: going from p_i to p_{i+1} in lane j requires T_{ij} minutes. In general, you would want to pick the fastest lane available for each section, but there is a catch - switching lanes requires slowing down and merging, which adds a delay. In particular, switching to the immediate next lane adds a delay of 2 minutes; thus, switching from lane h to lane j adds a delay of 2*|h-j| minutes. Your goal is to plan the route to compute the minimum possible travel time. Design an algorithm to achieve this, and analyze its time complexity. For full credit, your algorithm should run asymptotically faster than $\boldsymbol{\Theta}(n^2)$.

Example: Consider the route to be p_0–p_1–p_2 and k=3 lanes. Traversing p_0–p_1 via the 3 lanes takes 5,10,10 minutes respectively, and traversing p_1–p_2 takes 10,10,5 minutes respectively. Then, using lane 1 until p_1 takes 5 minutes, switching from lane 1 to lane 3 at p_1 adds **4** minutes, and going to p_2 in lane 3 takes another 5 minutes, giving a total of 14 minutes, which is the least time possible.

Solution: Create a "grid"-like graph. Each column i has k nodes corresponding to k lanes at p_i. Horizontal edges have weights T_{ij}. Vertical edges have weights 2 each. Create node S and connect to the leftmost column nodes with weight 0. Create node T and connect rightmost column nodes with weight 0. Compute the shortest S-T path using Dijkstra.

Creating the graph (9 points)
- All the required nodes (k nodes for each p_i) (2 points)
- Handled the lane-times T_{ij} ("horizontal edges") (2 points)
- Handled the lane-switching delay ("vertical edges") (3 points)
- Create S, T and connect to appropriate nodes (2 points)

Call Dijkstra (1 point)

Compute runtime (1 point for runtime of graph-creation + 1 point for that of Dijkstra's)

Student ID:

6) (17 points)
A king is planning to reward one of his servants. Since he wants to reward people based on their merits, he proposes a game to determine the servant's reward. The servant is offered n bags of coins in the beginning, with bag i having x_i coins. At each turn, the servant can choose to remove a bag and receive all the coins in it. However, after each time the servant chooses a bag to be removed, the king chooses one of the remaining bags and doubles the coins within that bag (note that this bag still remains in the game, and the servant can pick this bag in any of the following turns). The game ends when the servant plays n turns, having obtained the coins in all bags. The servant wants to **maximize** the coins obtained, and the king wants to **minimize** the coins that the servant gets.

Example: Suppose that initially there are two bags with 5, 10 coins, respectively. If the servant chooses the first bag, he gets 5 coins, then the king must choose the only remaining bag and the coins there are doubled from 10 to 20, then the servant chooses this bag to get 20 more coins, and a total of 25 coins, which is the maximum number of coins the servant can get.

a) (5 points) Describe the strategy each of them should deploy in deciding which bag to select in their respective turns to optimize their objectives (No proof required)

Servant must leave more valuable bags behind so that they can be doubled later. Thus, he always chooses the smallest one. (2.5 pts)
King also always chooses the smallest one to minimize the coins added. (2.5 pts)

b) (12 points) Assuming they play optimally as determined in part a), design an efficient algorithm to compute how many coins the servant gets in the end, and analyze its time complexity. For full credit, your algorithm should run asymptotically faster than $\boldsymbol{\Theta}(n^2)$.

Create a min-heap with all xi. Servant's total initialized to 0.
Iteratively:
1) Remove min and add to Servant's total.
2) Remove min and add its double back to min-heap.
Repeat until heap empty.

Initial heap-creation takes O(n)
Each iteration has 2 extract-min and 1 insert, thus, O(log(n)) in total. n total iterations make it O(n log n) for the entire algorithm.

Rubric:
Using min-heap with an iterative algorithm (3 points)
Correct step for servant's turn (3 points)
Correct step for king's turn (3 points)
Correct run-time analysis (3 points)

Student ID:

Student ID:

7) (20 points)
Suppose we have a graph G with positive edge-weights and an MST T. Each of the parts below describes how G is modified. In each case, either prove that T still remains an MST of the modified graph, or provide an algorithm that computes a new MST - this algorithm should be asymptotically faster than computing an MST of G from scratch.

a) 10 arbitrary edges from T are chosen and their weights are halved.
b) 10 arbitrary edges not in T are chosen and their weights are halved.
c) 10 arbitrary edges from T are chosen and their weights are doubled.
d) 10 arbitrary edges not in T are chosen and their weights are doubled.

a) and d) MST remains the same. This is because, before the change, adding an edge to T creates a cycle, with the added edge being at least as big as the others in the cycle. The same continues to hold after the change mentioned.

2 points for correct answer, 3 for explanation.

b) Each modified edge has potential to replace an existing edge of the MST. Hence, we run the following algorithm:

For each of the modified edge (u,v):
Run BFS/DFS in T to find the unique path from u to v, say P.
Find the largest edge on P, say *e*.
If *e* has a higher weight than edge (u,v), replace *e* with (u,v).

Each inner loop is in linear time, so for 10 edges, the total is in linear time.

c) Remove the 10 modified edges from T and replicate the approach used in Kruskal's. Specifically, here's the algorithm:

1) Remove all 10 doubled edges.
2) Run BFS/DFS to get connected components.
3) Repeat 10 times: Go through all remaining edges and record the smallest edge that joins different connected components, add it to T.

Close variant: Can try to replace doubled edges one at a time, i.e., remove one edge (u,v) at a time, which will create two connected components (obtained by running BFS/DFS from u and v resp.) Loop through the remaining edges and record the smallest edge that goes across the two connected components, add it to T. Repeat for each of the doubled edges.

For b) and c): 5 points for any correct algorithm that is strictly faster than computing an MST from scratch. Note that simply running Kruskal's algorithm over the new tree with modified edges is incorrect, because it does not support replacement/removal of edges; additional steps or modified algorithms are required for b) and c).

Student ID:

3 points if the algorithm is correct, and not simply computing an MST from scratch, BUT NOT FASTER than computing an MST from scratch (**this will be the case if all edges are sorted, for instance.**)

Student ID:

Additional Space

Student ID:

Additional Space