CS570 Spring 2024: Analysis of Algorithms Exam I

	Points		Points
Problem 1	20	Problem 5	10
Problem 2	9	Problem 6	20
Problem 3	9	Problem 7	16
Problem 4	6	Problem 8	10
	Total	**100**	

Instructions:

1. This is a 2-hr exam. Closed book. No electronic devices or internet access. One 8.5x11 cheat sheet allowed.
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.
8. This exam is printed double sided. Check and use the back of each page.

1) 20 pts (2 pts each)

Mark the following statements as **TRUE** or **FALSE** by circling the correct answer. No need to provide any justification.

[TRUE/FALSE]

A bipartite graph cannot contain an odd-length cycle.

[TRUE/FALSE]

There exists an algorithm with worst-case running time complexity $O(\log n)$ for finding the smallest element of a binary max-heap with n elements.

[TRUE/FALSE]

There exists an algorithm with worst-case running time complexity $O(n)$ for merging two binomial min-heaps of size n.

[TRUE/FALSE]

Removing any element from a binary heap (not necessarily the root) and then reheapifying to maintain the heap property can be accomplished in $O(\log n)$ time, where n is the number of elements in the heap.

[TRUE/FALSE]

If a Directed Acyclic Graph G is weakly connected and has $n \geq 2$ vertices and n-1 edges, then G must have a unique topological ordering.

[TRUE/FALSE]

Dijkstra's algorithm will work correctly as long as the graph has no more than one negative weight edge.

[TRUE/FALSE]

Every tree is a bipartite graph.

[TRUE/FALSE]

The amortized cost of an operation can be lower than the worst-case cost of that operation.

[TRUE/FALSE]

The amortized cost of an operation can be higher than the worst-case cost of that operation.

[TRUE/FALSE]

If $f(n) = \Omega(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.

2) 9 pts (3 pts each)

Circle ALL correct answers (no partial credit when missing some of the correct answers). No need to provide any justification.

i- In a graph G with n nodes and n+5 edges, where $k \geq 7$ edges have the same weight, every minimum spanning tree of G will have at least…

a) 2 edges with the same weight.
b) 3 edges with the same weight.
c) 4 edges with the same weight.
d) None of the above

ii- What is the solution to the recurrence equation $T(n) = 2T(n/2) + 3n + \text{sqrt}(n)$?

a) $T(n) = \Theta(n\ \text{sqrt}(n))$
b) $T(n) = \Theta(n \log^2 n)$
c) $T(n) = \Theta(n \log n)$
d) None of the above

iii- What is the solution to the recurrence $T(n) = 2T(n/2) + 3n \log n + 2n$?

a) $T(n) = \Theta(n \log n)$
b) $T(n) = \Theta(n)$
c) $T(n) = \Theta(n \log^2 n)$
d) $T(n) = \Theta(n^2)$

3) 9 pts

For each of the following algorithms, use the Master Method to determine its running time in terms of big-O, and if the Master Method cannot be applied explain why.

a. Algorithm *A*, which solves problems of size *n* by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions to the subproblems in linear time. (3 pts)

b. Algorithm *B*, which solves problems of size *n* by recursively solving two subproblems each of size *n*-1 and then combining the solutions in constant time. (3 pts)

c. Algorithm *C*, which solves problems of size *n* by dividing them into nine subproblems of size *n*/3, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time. (3 pts)

4) 6 pts (1 pt per correct answer)

Select all choices that correctly describe the worst-case running time complexity of the following code.

```
int a = 0;
for (i = 0; i < n; i++) {
          for (j = n; j > i; j--) {
               a = a + i + j;
          }
}
```

a) $\mathbf{O}(n^2)$

b) $\boldsymbol{\Theta}(n^2 \log n)$

c) $\Omega(n^2 \log n)$

d) $\mathbf{O}(n^2 \log n)$

e) $\Omega(n^2)$

f) $\boldsymbol{\Theta}(n^2)$

5) 10 pts

Consider a collection of n integer-valued variables $x_1, ..., x_n$ and a collection of m constraints, where in each constraint, two variables are constrained to be equal (e.g. $x_i=x_j$) or unequal (e.g. $x_i \neq x_j$).

Depending on the constraints, it may be impossible to assign values to the variables without violating at least one constraint. For example, the following collection of constraints cannot all be satisfied simultaneously: $x_1=x_2$, $x_2=x_3$, $x_3=x_4$, $x_4 \neq x_2$.

Design an algorithm (hint: graph based) that takes as input m constraints over n variables and decides in $O(m+n)$ time whether the given set of constraints can be satisfied. Analyze the worst-case running time complexity of your algorithm.

5 points for solutions that run in $O(m^2)$

6) 20 pts

Consider the Minimum Leaf-Constrained Spanning Tree (MLCST) Problem, defined below:

Given an undirected weighted graph $G = (V, E, w)$ and a subset of vertices $U \subseteq V$, find the least-weight spanning tree T in G in which each vertex in U is a leaf of T. If no such tree exists, your algorithm should indicate so.

a. Design an algorithm for this problem with worst-case running time $O(|E|\log|V|)$. (10 pts)

b. Analyze the worst-case running time complexity of your algorithm. (5 pts)

c. Give an example of a graph G and a set of vertices $U \subseteq V$ for which such an MLCST does not exist. (5 pts)

7) 16 pts

Consider two lists, L_1 of length n and L_2 of length m. We say that L_2 is a subsequence of L_1 if we can delete certain elements from L_1 so that the remaining sequence is equal to L_2. This means that there exists m indices $i_1 < \ldots < i_m$ such that $L_1[i_j] = L_2[j]$ for each j. Design an efficient algorithm that detects if L_2 is a subsequence of L_1 and outputs the indices $i_1, \ldots, i_m$ if L_2 is a subsequence of L_1 (8 pts), analyze your algorithm's worst-case running time (3 pts), and prove that your algorithm is correct (5 pts).

8) 10 pts

You are assigned *n* tasks, numbered from 1 to *n*. There are *m* dependence relations between the tasks. For the *i*-th dependence relation, (u_i, v_i) indicates that task u_i should be finished before starting task v_i. You can only process one task at a time. Design an efficient algorithm for deciding whether it is possible to finish all tasks without violating any dependence relations. Your solution should run in linear time with respect to *n* and *m*. (7pts)

Analyze the worst-case running time complexity of your solution. (3 pts)

Additional Space

Additional Space

Additional Space