# CS570 Spring 2020: Analysis of Algorithms Practice Exam I

| | Points |
|---|---|
| Problem 1 | 20 |
| Problem 2 | 14 |
| Problem 3 | 12 |
| Problem 4 | 12 |
| Problem 5 | 10 |
| Problem 6 | 15 |
| Problem 7 | 17 |
| Total | 100 |

Instructions:
1. This is a 2-hr and 20-mins exam. Closed book and notes.
2. Any kind of cheating will lead to **score 0** for the entire exam and be reported to SJACS.
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. 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]**
Given a binary max-heap with n elements, the time complexity of finding the smallest element is $O(\log n)$.

**[TRUE/FALSE]**
A greedy algorithm considers the entire search space when making each step.

**[TRUE/FALSE]**
Suppose that for a graph $G = (V, E)$ the average edge weight is $w$. Then a minimum spanning tree of $G$ will have weight at most $(V - 1)\, w$.

**[TRUE/FALSE]**
Consider two positively weighted graphs $G_1 = (V, E, w_1)$ and $G_2 = (V, E, w_2)$ with the same vertices $V$ and same edges $E$ such that for any edge $e \in E$ we have $w_2(e) = (w_1(e)^2$. For any two vertices $u \in V$ and $v \in V$ any shortest path between $u$ and $v$ in $G_2$ is also a shortest path in $G_1$.

**[TRUE/FALSE]**
If all edges in a connected undirected graph have unit cost, then you can find the MST using BFS.

**[TRUE/FALSE]**
Breadth first search is an example of a divide-and-conquer algorithm.

**[TRUE/FALSE]**
For any cycle in a graph, the cheapest edge in the cycle is in a minimum spanning tree.

**[TRUE/FALSE]**
0/1 knapsack problem can be solved using dynamic programming in polynomial time.

**[TRUE/FALSE]**
DFS finds the longest paths from start vertex $s$ to each vertex $v$ in the graph.

**[TRUE/FALSE]**
The shortest path in a weighted DAG can be found in linear time.

2) 14 pts

Arrange the following functions in increasing order of growth rate with $g(n)$ following $f(n)$ in your list if and only if $f(n) = O(g(n))$:

$$2^{(\log n)^{\frac{1}{3}}}, \quad (\log n)^{\log n}, \quad n\,(\log n)^3, \quad \frac{\sqrt{\log n}}{\log\log n}, \quad 2^{2^n}, \quad n^{1.001}, \quad n!$$

3) 12 pts.

Suppose that we have an undirected graph $G = (V, E)$. Prove the following statements.

a) If $G$ is a simple connected graph, then $E \leq \frac{V(V-1)}{2}$. (3 pts)

b) If $G$ is a simple graph with $V > 3$ and $E > \frac{(V-1)(V-2)}{2}$, then $G$ is a connected graph. (9 pts)

4) 12 pts.

Businessman Daniel has $n$ containers $c_1$, $c_2$, …, $c_n$ of several varieties of fruits to ship. Each container has different cost and depreciation expense. Initial value of each container is $v_1$, $v_2$, …, $v_n$ and depreciation expense is given by $d_1$, $d_2$, …, $d_n$. Daniel has only one ship, so he can transport only one container at a time. Therefore, if container $c_i$ happens to be in the $j$-th shipment, its value will depreciate to $v_i/(j * d_i)$. Can you help Daniel to ship all containers and maximize total value of containers after depreciation? Provide proof of correctness and state the complexity of your algorithm.

5) 10 pts.

For each of the following recurrences, give an expression in the Theta notation for the runtime $T(n)$ if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

1. $T(n) = 16T\left(\frac{3n}{4}\right) + n$ T($n$) =

2. $T(n) = 3T\left(\frac{n}{3}\right) + \sqrt{n}$ T($n$) =

3. $T(n) = T\left(\frac{n}{2}\right) + n\,(2 - \cos n)$ T($n$) =

4. $T(n) = 2T\left(\frac{n}{2}\right) + n\log^2 n$ T($n$) =

5. $T(n) = 8T\left(\frac{n}{3}\right) - n^2 + n^2\log n$ T($n$) =

6) 15 pts

Kuang is planning to go skiing in Big Bear Mountain. The resort has *n* trails amd *m* resting bases. As a professional skier, he tries to find the minimum time he needs to get from the peak of the mountain to different resting bases. Due to the safety issue, Kuang must follow the following restrictions:

- He starts skiing from the peak.
- There are many ski trails connecting different resting bases, as well as the peak, however there is at most one trail connecting two of them.
- For each ski trail in the mountain, Kuang knows the time he needs to get downhill.
- Kuang cannot ski uphill.

a) Design a linear time algorithm to find the minimum time to get from the peak to all the other resting bases. (10 pts)

b) Show the time complexity of the algorithm. (5 pts)

7) 17 pts

There are several ways to formalize the notion of distance between strings. One common formalization is called edit distance, that focuses on transforming one string into the other by a series of edit operations. The permitted operations are **D**elete, **I**nsert and **R**eplace of a character in the first string. We also permit **M**atch, denoting that two characters match each other. Here is one possible alignment of DYNAMIC and STATIC

```
R   R   D   M   R   M   M
-------------------------
d   y   n   a   m   i   c
s   t       a   t   i   c
```

The edit distance between two strings is defined as the minimum number of edit operations. A transformation in terms of **D**, **I**, **M**, **R** of one string to another is called an edit transcript. The edit distance problem is to compute the edit distance between two given strings along with an optimal edit transcript.

Design a dynamic programming algorithm that calculates the edit distance between string *X* of length *n* and another string *Y* of length *m*.

a) Define (in plain English) subproblems to be solved. (3 pts)

b) Write the recurrence relation for subproblems. (5 pts)

c) Write pseudo-code to compute the minimum number of edit operations. (4 pts)

d) Compute the runtime of the above DP algorithm in terms of *n*. (2 pts)

e) Discuss how you would recover the edit transcript based on the DP table created in part c) (3 pts)