CS570 Spring 2020: Analysis of Algorithms 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(lg n).

False. The smallest element should be among the leaf nodes. Consider a full binary tree of n nodes. It has $(n+1)/2$ leaves. Then the worst case of finding the smallest element of a full binary tree (heap) is $O(n)$.

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

False. It always makes the choice that seems to be the best at that moment.

[TRUE/FALSE]
Suppose that for some graph G, the average edge weight is A. Then a minimum spanning tree of Gwill have weight at most $(n-1) \times A$.

False. We may be forced to select edges with weight much higher than average. For example, consider a graph G consisting of a complete graph G' on 4 nodes, with all edges having weight 1 and another vertex u, connected to one of the vertices of G' by an edge of weight 8. The average weight is (8 + 6)/7 = 2. Therefore, we would expect the spanning tree to have weight at most 4*2 = 8. But the spanning tree has weight more than 8 because the unique edge incident on u must be selected.

(See presentation for graph.)

[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 edges E such that for any edge $e \in E$ we have $w_2(e) = (w_1(e))^2$. For any two vertices $u, v \in V$, any shortest path between u and v in G_2 is also a shortest path in G_1.

False. Counterexample: Consider two paths p1 and p2 from u to v with two edges each: p1 has weights 5 and 6 and p2 has 1 and 9. p2 is shorter (10 instead of 11). When squared, we compare 25+36=61 to 1+81=82. Here, p1 is shorter.

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

True. Any spanning tree of a graph having only unit cost edges is also a MST, because the weight is always n-1 units. Of course, BFS gives a spanning tree in the connected graph.

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

False. We go by depth and don't break down the problem into smaller problems.

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

False. Counterexample: Consider a graph with a cycle of 80-90-90 between three vertices. All three edges are also directly connected to a fourth vertex by edges that cost 1 each. Then 80 is not contained.

(See presentation for graph.)

[TRUE/FALSE]

0/1 knapsack problem can be solved using dynamic programming in polynomial time.

False. The runtime is pseudo-polynomial. O(nW) with n = number of items and W = capacity. Difference is between the length of input in bits and its value.
$O(n2^{bits\ in\ W})$.

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

False. Depends on the order in which the nodes are traversed.

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

True. The idea is to use Topological Sorting. We initialize distances to all vertices as infinite and distance to source as 0, then we find a topological sorting of the graph. We one by one process all vertices in topological order. For every vertex being processed, we update distances of its adjacent using distance of current vertex.

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 $f(n) = O(g(n))$

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

Bring down the exponent by using logarithm.

$$\log\left(2^{(\log n)^{\frac{1}{3}}}\right) = (\log n)^{\frac{1}{3}} \log 2$$

$$\log((\log n)^{\log n}) = \log n \cdot \log\log n$$

$$\log(n\,(\log n)^3) = \log n + 3\log\log n$$

$$\log\left(\frac{\sqrt{\log n}}{\log\log n}\right) = \frac{1}{2}\log\log n - \log\log\log n$$

$$\log\left(2^{2^n}\right) = 2^n \log 2$$

$$\log(n^{1.001}) = 1.001 \log n$$

$$\log(n!) = \log\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right) = n\log\left(\frac{n}{e}\right) + \frac{1}{2}\log(2\pi n)$$ (Stirling's formula)

So we have:

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

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)

Solution:
Since the graph is simple, the maximum number of the edges can be reached when every vertex in the graph is connected to all the other vertices, which makes the graph a complete graph. Thus, the total number of the maximum edges is $\frac{V(V-1)}{2}$, which is the number of the edges of complete graph K_V. (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)

Solution:
We can prove by contradiction. If G is disconnected, then we can split the graph into two subgraphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, such that there is no edge connecting vertices between V_1 and V_2. (3 pts)

By part (a), we know that we can have the largest number of edges when G_1 and G_2 are both complete graphs. (1pt)

In this case, we let p_1 and p_2 be the number of vertices of G_1 and G_2, respectively. Then, we can show that in this case, G can be generated by removing edges connected V_1 and V_2 from a complete graph K_p. Thus, the number of the edges in G is $\frac{V(V-1)}{2} - V_1V_2$, where $V_1 + V_2 = V$. It is easy to find that the minimum of V_1V_2 is $V - 1$, where $V_1 = 1, V_2 = p - 1$. (This is also the condition for the maximum number of the edges in G.) (2pts)

(Alternative: Without loss of generality, we can assume $V_1 \leq V_2$. If $V_1 \geq 2$, then we can move one vertex from V_1 to V_2. We will remove $V_1 - 1$ edges in G_1 but will add new V_2 edges in G_2. The total number of edges in G will increase $V_2 - (V_1 + 1) > 0$. After keeping applying this, we can reach the state where $V_1 = 1, V_2 = p - 1$, where the number of total edges reached the maximum.)

Thus, the maximum number of edges of G is

$$\frac{p(p-1)}{2} - (p-1) = \frac{(p-1)(p-2)}{2}. \quad (1 \text{ pts})$$

This contradicts the given condition: $q > \frac{(p-1)(p-2)}{2}$. Thus, G must be connected. (2pts)

4) 12 pts.

Businessman Daniel has n containers $c_1, c_2, \ldots, c_n$ of several varieties of fruits to ship. Each container has different cost and depreciation expense. Initial value of each container is $v_1, v_2, \ldots, v_n$ and depreciation expense is given by $d_1, d_2, \ldots, d_n$. 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.

Please refer to our review video for the solution.

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) =

1. $a = 16,\ b = \frac{4}{3},\ c_{crit} = \log_{\frac{4}{3}} 16.$

   So $c_{crit} > \log_4 16 = 2 > 1$ (Case 1), $T(n) = \Theta\left(n^{\log_{\frac{4}{3}} 16}\right)$.

2. $a = b = 3,\ c_{crit} = \log_3 3 = 1.$

So $c_{crit} > \frac{1}{2}$ (Case 1), $T(n) = \Theta(n)$.

3. $a = 1,\ b = 2,\ c_{crit} = \log_2 1 = 0.$

   Pick $c = 0.5$, then $c_{crit} < c$ and $f(n) = \Omega(n^c)$ (Case 3), we still need to check the regularity condition.

   However, when $n = 2\pi m$, where m is an odd integer:

   $\text{left} = m\pi(2 - (-1)) = 3m\pi$, $\text{right} = 2\pi m(2 - 1) = 2\pi m$

   $\text{left} = \frac{3}{2} \cdot \text{right}$, so the regularity condition does not hold.

   Then the master theorem does not apply.

4. $a = b = 2,\ c_{crit} = \log_2 2 = 1,\ f(n) = \Theta(n \log^2 n).$

   So $T(n) = \Theta(n \log^3 n)$.

5. $a = 8,\ b = 3,\ c_{crit} = \log_3 8 < 2.$

   Pick $c = 2,$ then $c > c_{crit}$ and $f(n) = \Omega(n^c)$(Case 3), we still need to check the regularity condition.

$$8f\left(\frac{n}{3}\right) = -\frac{8}{9}n^2 + \frac{8}{9}n^2 \log\left(\frac{1}{3}n\right) = \frac{8}{9}\left(-n^2 + n^2 \log\left(\frac{1}{3}n\right)\right)$$
$$< \frac{8}{9}(-n^2 + n^2 \log n)$$

$$f(n) = -n^2 + n^2 \log n$$

   So $8f\left(\frac{n}{3}\right) \le \frac{8}{9}f(n)$, $k = \frac{8}{9}$, so the regularity condition holds true,

   Then $T(n) = \Theta(f(n)) = \Theta(n^2 \log n)$.

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)

Solution:
We can construct a directed graph $G = (V, E)$, where V represents all the resting spots (including the peak) and E represents all the skiing lanes. The weight of each edge, $w(u, v)$ is the time Kuang needs to pass the lane starting from u to v. Given the statements, we know that G is a directed acyclic graph (DAG). Now we need to find the minimum distance from the vertex representing the peak, s, to all the other vertices in this DAG. (3 pts for constructing the graph correctly)

We can use DFS to find a topology sort of all vertices. (3 pts)

Let $d(v)$ be the minimum distance from the peak of the mountain, s to the destination resting spot, v. We have $d(s) = 0$.

Then, following the order of topological sort (2 pts),
we can calculate $d(v)$ for all vertices by the following recurrence.

$$d(v) = \min_{e=(u,v)\in E}(d(u) + w(e)) \quad (2\text{pts})$$

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

Solution:

Construction of the graph will takes O(m)
Topological sort takes $O(n+m)$ (2pts)

For calculating $d(v)$, we need to traverse all cities, which takes $O(m)$. For each city v, we need to visit all edges whose destination is v. Thus, each edge in the graph has been visited at most once, which takes $O(n)$. The time complexity is also $O(n+m)$. (2pts)

Thus, the total time complexity is $O(n+m)$ as well. (1pt)

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 (Note that this is **NOT** an edit operations), 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)

$OPT(i,j)$ is the edit distance between $X = X_1 \dots X_i$ and $Y = Y_1 \dots Y_j$

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

$OPT(i,j) = OPT(i-1, j-1)$ if $X_i = Y_j$,
$OPT(i,j) = \min\{OPT(i-1,j-1), OPT(i-1,j), OPT(i,j-1)\} + 1$ otherwise.

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

$for\ i = 0, \dots, n$:
    $OPT(i,0) = i$
$for\ j = 1, \dots, m$:
    $OPT(0,j) = j$
$for\ i = 1, \dots, n$:
    $for\ j = 1, \dots, m$:

$If\ (X_i = Y_j)$:

$OPT(i,j) = OPT(i-1, j-1)$

Else:

$OPT(i,j) = \min\{OPT(i-1,j-1), OPT(i-1,j), OPT(i,j-1)\} + 1$

$Return\ OPT(n,m)$

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

$O(nm)$

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

If last characters of two strings are the same, ignore them and go for remaining strings.

If last characters are different, consider all three operations (Delete $OPT(i-1,j)$, Insert $OPT(i,j-1)$ and Replace $OPT(i-1,j-1)$) on last character of the first string, compute the minimum cost and take the corresponding operation (arbitrarily break ties).