CS570 Practice Exam 1

T/F Question 1-10 (Random Questions) [Any **10** from the below 15 Questions]

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

[TRUE/FALSE]
Assume that no two men have the same highest-ranking woman. If the women carried out the proposal to men, then the Gale-Shapley algorithm will contain a matching set where every man gets their highest-ranking woman.

[TRUE/FALSE]
Using the master's theorem, the asymptotic bounds for the recurrence $2T(n/4) + n$ is $\Theta(n)$.

[TRUE/FALSE]
Breadth first search finds the shortest distance to a node from the starting point in unweighted graphs

[TRUE/FALSE]
Inserting an element into a binary min-heap takes O(1) time if the new element is greater than all the existing elements in the min heap.

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

[TRUE/FALSE]
An array with following sequence of terms [16, 14, 10, 10, 12, 9, 3, 2, 4, 1] is a binary max-heap.

[TRUE/FALSE]
In any graph we have that $|E| = \Theta(|V|^2)$

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

[TRUE/FALSE]
Given a graph G. If the edge e is not part of any MST of G, then it must be the maximum weight edge on some cycle in G.

[TRUE/FALSE]
Any dynamic programming algorithm with n^2 subproblems will require at least $O(n^2)$ space.

[TRUE/FALSE]
Kruskal's algorithm can fail in the presence of negative cost edges.

[TRUE/FALSE]
Any problem that can be solved using dynamic programming has a polynomial time worst case time complexity with respect to its input size.

[TRUE/FALSE]
Consider a version of the interval scheduling problem where all intervals are of the same size. A greedy algorithm based on earliest start time will always select the maximum number of non-overlapping intervals.

[TRUE/FALSE]
Suppose we have a weighted graph G = (V, E, w) and let S be a shortest s – t path for s , t in V. If the weight of every edge in G is doubled (i.e., w'(e) = 2w(e) for each e in E), then S will still be a shortest s – t path in (V, E, w')

[TRUE/FALSE]
In the aggregate analysis different operations may have different amortized costs, while in the accounting method all operations have the same amortized cost.

Greedy Algorithm Problem

Suppose you are the registrar for USC and you need to schedule classrooms to be used for various classes. There are n available classrooms and n classes that need rooms at a given time. Each classroom i has a maximum capacity c_i , and each class j has s_j students. A classroom can't take a class that is over its maximum capacity. Assume that there exists a perfect matching of classrooms and classes.

a). Design a greedy algorithm which returns a perfect matching of classes to appropriate classrooms

Solution:
Sort classrooms by capacities c_i and classes by students s_j in the same order (both ascending or both descending, either works). Schedule classrooms for classes in that order.

b). Give the asymptotic worst-case complexity

Solution:
Sorting both classes and classrooms takes O(nlog n) each.

c). Prove that your algorithm produces correct results

Solution:
Let's consider classrooms in descending order of their capacities $\{c_1 , c_2 , \ldots . c_n\}$, then our solution schedules them with classes having students $\{s_1 , s_2 , \ldots . s_n\}$ in the same order (descending). To prove that our algorithm works, we just need to show that it produces a perfect matching.

Assume that a perfect matching OPT exists, which might schedule some classes differently than our solution. We define an inversion to be the case when in OPT there is a class with students s_j scheduled before another class with students $s_i > s_j$ (considering classrooms in descending order of their capacities). Thus, OPT can have at most n choose 2 inversions.

Suppose there exists an inversion in the OPT. Our solution must have s_i before s_j , whereas OPT has s_j before s_i for some $i < j$. Since OPT schedules c_j with s_i , then $c_j \geq s_i$ .We also know that $c_i \geq c_j$ and $s_i \geq s_j$. Therefore, $c_i \geq c_j \geq s_i \geq s_j$ and we can swap s_i and s_j in OPT to create OPT' which also satisfies the perfect matching condition (A classroom can't take a class that is over its maximum capacity).

If we continue removing inversions, we eliminate all the differences between OPT and our solution without violating the perfect matching condition. In other words, our solution is a perfect matching.

Shortest Path problem

Suppose that you want to get from vertex s to vertex t in a connected undirected graph G = (V; E) with positive edge costs, but you would like to stop by vertex u if it is possible to do so without increasing the length of your path by more than a factor of α.

a) Describe an efficient algorithm in O(|E| log |V|) time that would determine an optimal s-t path given your preference for stopping at u along the way if doing so is not prohibitively costly. (In other words, your algorithm should either return the shortest path from s to t, or the shortest path from s to t containing u, depending on the situation.) If it helps, imagine that there are free burgers at u!
b) Show that your algorithm runs in O(|E| log |V|).

Solution:

a). Since the graph has positive edges, one can use Dijkstra algorithm for the shortest paths computation. We run Dijkstra twice, once from s and once from u. The shortest path from s to t containing u is composed of the shortest path from s to u and the shortest path from u to t. We can now compare the length of this path to the length of the shortest path from s to t and choose the one to return based on their lengths.

b). Since we are using Dijkstra algorithm the total running time is O((|V| +|E|) log |V|) and since the graph is connected it is simplified to O(|E| log |V|)

Asymptotic Notations Problem (all correct to receive a full score or 0**)**

3.1) Rank the following functions in order from smallest asymptotic complexity to largest. No justification needed. (log is log base 2 by convention)

$$\log n^{\log n}$$
$$\log n^{10}$$
$$3n^2$$
$$n^{\log n}$$
$$10^{\log n}$$
$$\log n^{2n}$$
$$3^n$$

Solution:

$\log n^{10} < \log n^{\log n} < \log n^{2n} < 3n^2 < 10^{\log n} < n^{\log n} < 3^n$

Dynamic Programming Problem - 1

There is a series of activities lined up one after the other, $J_1, J_2, \ldots, J_n$. The $i\text{-}th$ activity takes T_i units of time, and you are given Miamount of money for it. Also for the $i\text{-}th$ activity, you are given N_i , which is the number of immediately following activities that you cannot take if you perform that $i\text{-}th$ activity.

a). Give a dynamic programming solution to maximize the amount of money one can make in T units of time. Note that an activity has to be completed in order to make any money on it.

Solution:

Recurrence formula:

If $T_i > t$ then $opt(i, t) = opt(i+1, t)$,

Otherwise, $opt(i, t) = \max(opt(i+1, t), opt(i + N_i + 1, t - T_i) + M_i)$

Boundary conditions: $opt(i, t) = 0$ for $i > n$ and all t,

$opt(i, t) = 0$ for $t < 1$ and all i

To find the value of the optimal solution:

```
for i=n to 1 by -1
    for t= 1 to T
        If Ti > t then opt(i, t)= opt(i+1, t),
        Otherwise, opt(i, t)= max(opt(i+1, t), opt(i + Ni + 1, t - Ti) + Mi)
    end for
end for
```

opt(1, T) will hold the value of the optimal solution (maximum money that can be made).
Complexity is O(nT)

Given all the values in the two dimensional opt( ) array, the optimal set of activities can be found in O(n)

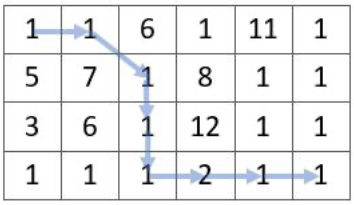
b). State the runtime of your algorithm.

Solution:

Overall complexity of the algorithm is O(nT) which is pseudopolynomial.

Dynamic Programming Problem II

Given an m x n grid filled with non-negative numbers, use dynamic programming to find a path from top left to bottom right which minimizes the sum of all numbers along its path. At any point, you can either go to the square on your right, the square below you, or the one diagonally to the right and below you as shown in this diagram.



a). Define in plain English sub-problems to be solved.

For every square searched, find the smallest weight among the squares either to the left, up, or diagonal above the square, and add the cost of the current square to find the minimum path to the current square.

b). Write the recurrence relation for sub-problems

OPT(m,n) = min(OPT(m-1, n), OPT(m, n-1), OPT(m-1, n-1))

c). Using the recurrence formula in the last question, write pseudo-code to compute the minimum cost of the path

```
Create a two-dimensional array with size m x n
Initialize the first row OPT(0, n) and the first column OPT(m, 0) with
accumulated values For i = 1 to m-1
        For j = 1 to n-1
                OPT(i, j) = min(OPT(i-1, j), OPT(i, j-1), OPT(i-1, j-1))
        End for
End for
Return OPT(m-1, n-1)
```

d). What is the runtime complexity of your solution?

O(mn)

Minimum Spanning Tree Problem

Let T be a minimum spanning tree for G with edge weights given by weight function w. Choose one edge $(x, y) \in T$ and a positive number k, and define the weight function w' by:
$w'(u, v) = w(u, v)$, if $(u, v) \neq (x, y)$
$w'(u, v) = w(u, v) - k$, if $(u, v) = (x, y)$

Show that T is also a minimum spanning tree for G with edge weights given by w'.

Solution:

Proof by contradiction. Note: T is a spanning tree for G with weight function w' and $w'(T) = w(T) - k$.

Suppose T is NOT a MST for G with w'. Hence, there exists a tree T' which is an MST for G with w'. So we have

$$w'(T') < w'(T) \quad \text{--- [1]}$$

We have two cases to analyze:

Case 1:

If $(x, y) \in T'$, then $w'(T') = w(T') - k$.
From the above we have $w(T') < w(T)$.

Now that means that T' is a spanning tree for G with w which is **better** than T which is a MST for G with w.
A contradiction.

Case 2:

If $(x,y) \notin T'$, then $w'(T') = w(T')$.
So by [1] we have $w(T') < w'(T) = w(T) - k$.
A similar contradiction again.

Hence T is a MST for G with w'.