CS570 Practice Exam 1
Fall 2020

T/F Question 1-10 (Random Questions) [Any **10** from the below 12 Questions]

[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 Θ(n).

[TRUE/FALSE]
BFS 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| = Θ(|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]
Kruskal's algorithm can fail in the presence of negative cost edges.

[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.

[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')

Asymptotic Notations Problem

Rank the following functions in order from smallest asymptotic complexity to largest. No justification needed. (log is log base 2 by convention)
Note: 1 = smallest , ..., 7 = largest

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

Greedy 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(nlogn) each.

c) Prove that your algorithm produces correct results

Solution:
Let's consider classrooms in descending order of their capacities $\{c_1 , c_2 , ... c_n\}$, then our solution schedules them with classes having students $\{s_1 , s_2 , ... s_n\}$ 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.

Minimum Spanning Tree

Let T be a minimum spanning tree for G with edge weights given by weight function w. Choose one edge (x,y) ∈ T and a positive number k, and define the weight function w' by:

$$w'(u,v) = \begin{cases} \mathrm{w(u,v)}, \text{ if } \mathrm{(u,v)} \neq \mathrm{(x,y)} \\ \mathrm{w(u,v)} - k, \text{ if } \mathrm{(u,v)} = \mathrm{(x,y)} \end{cases}$$

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

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

We have two cases to analyze:

Case 1:
If (x, y) ∈ 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) ∉ 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’.

Shortest Path

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|)

Divide and Conquer

Assume we have 2 databases each containing n numbers. Each database has a function ***get-kth-element(k)*** that returns the k-th element of the database. Also, each database is storing the numbers sorted. Since these databases are stored at remote locations, the cost of each function call is high. Our goal is to find the k-th item in the aggregate of two databases. That is, if we had all the numbers of both databases, what would the k-th number be? For simplicity assume all the numbers are different. Give the algorithm to solve this problem. Your algorithm should be optimized. Compute the complexity of your algorithm.
a) Give an algorithm to solve this problem. Your algorithm should be optimized.
b) Compute the complexity of your algorithm. (**Note:** if you want to use any of these special characters in your answer, copy it over: Θ, Ω)

Solution:

a) The optimal algorithm is divide and conquer. Similar to the binary search, we can attempt to find the k-th comparing the k/2 element of arrays.
Pseudo-code:

```
i = k/2
j = k - i
step = k/4
while step > 0
    if a[i] > b[j]
        i -= step
        j += step
    else
        i += step
        j -= step
    step /= 2

if a[i] > b[j]
    return a[i]
else
    return b[j]
```

b) Step is k/4 at the beginning and in the while loop step gets divided by two in each iteration until it gets 0, hence the runtime of algorithm is log(k)

Master Theorem

A divide and conquer algorithm that operates on a graph with n nodes and m edges (i.e., a problem of size n, m) has the following characteristics:

- It takes Cn^2 time to divide a problem of size (n, m)
- It takes $\mathrm{Cm}log^2\mathrm{m}$ to combine a problem of size (n, m)
- At each step we divide the problem into 25 subproblems of size (n/5, m/5)

Use the Master's method to determine a tight upper bound on the performance of this algorithm for the following two cases:

a) A very sparse graph (i.e., $m=Cn$).

b) A very dense graph.

Solution:
We can formulate a recurrence equation:

$$T(n,m) = 25T\left(\frac{n}{5},\frac{m}{5}\right) + Cn^2 + \mathrm{Cm}log^2\mathrm{m}$$

a) We know $m = Cn$, then

$$T(n) = 25T\left(\frac{n}{5}\right) + Cn^2 + Cnlog^2n$$

$$f(n) = Cn^2 + Cnlog^2n = \theta(n^2)$$

$$a = 25, b = 5$$

$$n^{log_b^a} = n^2$$

By applying the Master's theorem case 2

$$T(n) = \theta(n^2logn)$$

b) We know $m \approx n^2$, then

$$T(n) = 25T\left(\frac{n}{5}\right) + Cn^2 + Cn^2log^2(n^2)$$

$$f(n) = Cn^2 + Cn^2log^2(n^2) = Cn^2 + Cn^2 * 4log^2(n) = \theta(n^2log^2n)$$

$$a = 25, b = 5$$

$$n^{log_b^a} = n^2$$

By applying the generalized Master's theorem

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