CSE 101 Practice Final Questions (Not to be turned in)

1. This question refers to the following algorithm. x is an integer, represented using $n = \lceil \lg(x) \rceil$ bits.

```
P(x)
for i ← 2 to ⌈√x⌉
    do if x mod i = 0
        then return 0
return 1
```

   (a) What does P(15) return?

   (b) What does P(17) return?

   (c) Give the worst-case running time using Θ notation, in terms of the number of bits n required to represent the instance x.

   (d) Is this worst-case running time polynomial in n?

   (e) Give the best-case running time using Θ notation.

2. We are given a set of n lectures (in no particular order) that are supposed to take place on a given day. Each lecture has a specified start time and finish time, on the 24 hour clock, and all lectures start and finish on the hour (that is, 9am or 1pm, but not 1:30pm, etc.). Lectures do not start before 8am (represented as the integer 8) and lectures finish by 5pm (represented as the integer 17). Only a single lecture hall is available, so the goal is to schedule as many lectures as possible into the lecture hall that day.

   For example, there might be 3 lectures, one starting at time 8 and finishing at time 10, another starting at time 9 and finishing at time 12, and a third starting at time 11 and finishing at time 13. The best schedule would pick the first and third lectures, fitting a total of two lectures into the lecture hall.

   Write an algorithm (using pseudocode) to find an optimal schedule for n lectures, where the lectures are given by an array S of start times, and an array F of finish times. Your algorithm should take $O(n)$ time. If necessary, you may make calls to any of the algorithms we have discussed in class.

3. Here is a modified version of QUICKSORT, that uses SELECT to choose the smallest element in the array, and uses this as the "pivot" element about which to partition. The modified PARTITION function used below is the same as the original, except that it allows us to pass in the pivot value x rather than setting it to be equal to the first element of the array.

```
QUICKSORT2(A,p,r)
if p < r
    then x ← SELECT(A,p, r, 1)
         q ← PARTITION(A,p,r,x)
         QUICKSORT2(A,p,q)
         QUICKSORT2(A,q + 1,r)
```

   (a) Give the *best-case* running time of QUICKSORT2, using Θ notation.

   (b) Is the problem solved by this algorithm (the problem of sorting a list of numbers) in the complexity class P? Is it in NP?

4. **(a)** Consider the following problem. We are presented with n cakes. Cake i has yumminess value y_i per kilogram, and weights w_i kilograms. We can fit a total of K kilograms of cake into our stomach, and want to maximize the total yumminess value consumed. We can feed on any subset of the cakes, and can eat any portion of the chosen cakes (it doesn't have to be the whole cake, and it doesn't have to be an integer-valued number of kilograms).

   For example, if we had a 3 kilogram cake with yumminess 2 per kilogram, and a 2.5 kilogram cake with yumminess 1 per kilogram, and we could fit a total of $K = 4.2$ kilograms into our stomach, then the optimal meal would be the entire first cake, and 1.2 kilograms of the second cake, for a total yumminess value of $2 \cdot 3 + 1.2 \cdot 1 = 7.2$, using up our total allotted weight of $3 + 1.2 = 4.2$ kilograms.

Write a greedy algorithm to solve this problem (use pseudocode). It should take in n cakes, represented by an array of yumminess values Y and an array of weights W, and a total allowed weight K, and return the fraction of each cake that should be eaten. Assume the cakes are input in decreasing sorted order, according to yumminess value per kilogram. Your algorithm should take $O(n)$ time (assume arithmetic operations take constant time). You shouldn't need to make calls to any of the algorithms we have discussed in class.

(b) Prove that your algorithm produces the optimal solution (a set of cakes that maximizes total consumed yumminess value, for the given allotted weight K).

5. Recall the FLOYD-WARSHALL algorithm, which takes a graph's edge weight matrix W, and returns the matrix D giving length of shortest paths from every vertex to every other vertex in the graph.

FLOYD-WARSHALL(W)
1 $\quad n \leftarrow rows[W]$
2 $\quad D^0 \leftarrow W$
3 $\quad$ **for** $k \leftarrow 1$ **to** n
4 $\qquad$ **do for** $i \leftarrow 1$ **to** n
5 $\qquad\quad$ **do for** $j \leftarrow 1$ **to** n
6 $\qquad\qquad d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7 $\quad$ **return** $D^{(n)}$

This algorithm finds paths that minimize a path score, where paths are scored according to the total sum of weights of edges used along the path. That is, if a path from vertex v_1 to vertex v_m uses edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{m-1}, v_m)$, the score for this path is:

$$\sum_{i=1}^{m-1} W[v_i, v_{i+1}].$$

Say that we modify the score for paths so that paths are scored according to the *product* of weights of edges used along the path. Now, the score for the above path would be:

$$\prod_{i=1}^{m-1} W[v_i, v_{i+1}].$$

Assume that all edge weights are positive, and modify FLOYD-WARSHALL so that it finds the the matrix of minimum path scores, according to this new definition of path score. The D matrix returned by your modified algorithm should contain, for each pair of vertices i, j, the minimum product $D[i, j]$ of edge weights for paths from i to j.

If you want, you can just specify modifications to the pseudocode, but be precise (for example, "modify line 3 to read... include the following line between lines 5 and 6 so that it is within the scope of the **for** on line 5...").

6. In the EDGE-COUNT problem, we are given a graph $G = (V, E)$ with vertices V and edges E, a vertex count b, and an edge count c. We are asked to decide whether there is any subset $V' \subseteq V$ of b vertices, such that there are exactly c edges in E that have both endpoints in V'. That is,

$\mathrm{EDGE-COUNT} = \{< G = (V, E), b, c >$: there exists $V' \subseteq V$ with $|V'| = b$ and $|\{(i, j) \in E : i \in V' \text{ and } j \in V'\}| = c\}$

Show that EDGE-COUNT is NP-complete. You may use reductions from 3-CNF-SAT, CLIQUE, HAM-CYCLE, or TSP.

7. Recall the activity-selection problem: we are given a set $S = \{1, 2, \ldots n\}$ of n proposed activities that wish to use a resource which can only be used by one activity at a time. Each activity i has start time s_i and finish time f_i with $s_i \leq f_i$. We are asked to find a maximum-size set of non-overlapping activities. Our greedy algorithm for this problem repeatedly chooses the *first-finishing* activity compatible (not overlapping) with the activities already chosen, and it was shown that this produces an optimal schedule.

Consider the alternative greedy strategy of choosing the *first-starting* activity compatible with the activities already chosen. That is, among activities that don't overlap with ones already chosen, we choose the activity that starts first. Give a counterexample to show that this greedy strategy does not always produce an optimal schedule. Give the start and finish times of your activities, show the schedule produced by this alternative first-starting greedy strategy, and show a better schedule to prove that the first-starting schedule is not optimal.