CS570 Spring 2020: Analysis of Algorithms Exam I

	Points
Problem 1	20
Problem 2	14
Problem 3	12
Problem 4	17
Problem 5	8
Problem 6	12
Problem 7	17
Total	100

Instructions:

1. **Sign an honor code pledge**. If it is not signed, your exam won't be graded.
2. This is a 2-hr and 20-mins open-book take-home exam.
3. You may consult the textbook, your notes, HW assignments and discussions.
4. You are **not allowed** to communicate to each other in ANY way.
5. Any kind of cheating will lead to **score 0** for the entire exam and you will be reported to AIC.
6. If a description to an algorithm or a proof is required please limit your description or proof to within 200 words, preferably not exceeding the space allotted for that question.
7. 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.
8. Do not write your answers in cursive scripts.
9. Write solutions to each problem on a separate sheet of paper.

Academic Integrity Honor Code Pledge

I pledge to uphold the highest academic standards and integrity. In accordance with USC Viterbi's Honor Code (https://viterbischool.usc.edu/academic-integrity/), I affirm that I have not used any unauthorized materials in completing this exam, and have neither given assistance to others nor received assistance from others. Further, I affirm that I have not observed any other students in this class acting to gain an unfair advantage, or I have reported to my instructor any activity I have observed that is not in accordance with USC Viterbi's Honor Code. I do so to sustain a Viterbi culture of integrity, responsibility, community and "excellence in all our endeavors." I understand that there are significant consequences for violating academic integrity (https://policy.usc.edu/scampus-part-b/) and that suspected violations will be reported to the School and the University.

First and Last Name: Xinrui Ying

USC Email: xinruiyi@usc.edu

Signature: Xinrui Ying

Date: 03/13/2020

1) 10 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/[✓ FALSE]]
Algorithm *A* has a running time of $\Theta(n^2)$ and algorithm *B* has a running time of $\Theta(n \log n)$. From this we can conclude that *A* can never run faster than *B* on the same input set.

[[✓ TRUE]/FALSE]
The shortest path between two points in a graph could change if the weight of each edge is increased by an identical number.

[TRUE/[✓ FALSE]]
In a simple, undirected, connected, weighted graph with at least three vertices and unique edge weights, the heaviest edge in the graph never belongs to the minimum spanning tree.

[TRUE/[✓ FALSE]]
The total amortized cost of a sequence of *n* operations (i.e., the sum over all operations) gives a lower bound on the total actual cost of the sequence.

[[✓ TRUE]/FALSE]
For a binomial heap with the min-heap property, upon deleteMin, the order of the largest binomial tree in the binomial heap will never increase.

[TRUE/[✓ FALSE]]
In a divide & conquer algorithm, the size of each sub-problem must be at most half the size of the original problem.

[TRUE/[✓ FALSE]]
In a dynamic programming solution, the space requirement is always at least as big as the number of unique sub problems.

[[✓ TRUE]/FALSE]
The longest simple path in a DAG can be computed by negating the cost of all the edges in the graph and then running the Bellman-Ford algorithm.

[TRUE/[✓ FALSE]]
The Bellman-Ford algorithm may not terminate in a graph with a negative cycle.

[TRUE/[✓ FALSE]]
If an operation takes O(1) amortized time, then that operation takes O(1) worst case 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 $f(n) = O(g(n))$. All logs are in base 2.

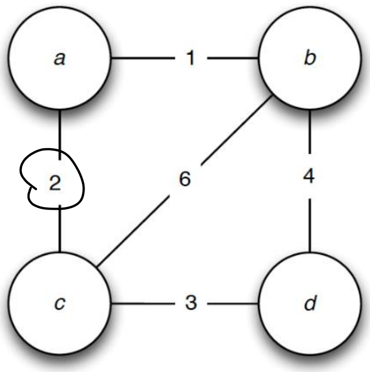
$n^{\sqrt{n}}$, $(\sqrt{2})^{\log n}$, $n\,(\log n)^3$, $2^{\sqrt{2\log n}}$, $(\log n)^n$, $n^{\frac{1}{\log n}}$, $\log(n!)$

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

3) 12 pts.

Given undirected weighted connected graph G = (*V*, *E*) and a subset of vertices $U \subset V$. We need to find the minimum spanning tree M_U in which all vertices of *U* are leaves. The tree might have other leaves as well. Design an O(*E* log *E*) runtime algorithm that computes the minimum spanning tree M_U given a subset of vertices $U \subset V$.

Consider the following example.



Let us choose *U* = {*c*}, then the minimum spanning tree M_U consists of edges (*a*, *b*), (*a*, *c*) and (*b*, *d*) with the total weight 7. Note that the regular minimum spanning tree *M* consists of edges (*a*,*b*), (*a*,*c*) and (*c*,*d*), with the total weight 6.

First, we sort all edges. Then, we loop through sorted edges in ascending weight order. We use Unionfind to keep track of connected group, similar to kruskal, in addition, if the edge has a vertex or two vertices in U, we will check if these vertices in U has already been visited using a dictionary. If the vertex of the edge is in U and has been visited, we don't add the edge. Otherwise, if the edge connects two components, we will add it to the MST and modify Unionfind and dictionary accordingly. Time: $O(E\log E + E) = O(E\log E)$

An intentionally blank page.

4) 17 pts.

Jared owns *N* grocery stores along a straight avenue of length *L* miles. He knows that his *i*-th store, which is located at x_i miles from the start point of the avenue, can attract all residents whose houses are no more than r_i miles from that store. That is, if one's house distance from the start point of the avenue is within $[x_i - r_i, x_i + r_i]$, then he/she will be attracted to Jared's *i*-th store. Jared is very proud that no matter where the resident lives, he/she will be attracted by at least one of Jared's stores. Unfortunately, Jared is short of cash now due to the stock market meltdown. He will have close some of his stores, but he still wants all the residents, no matter where they live, to be attracted by at least one of his stores.

a) Design a greedy algorithm to help Jared to find the minimum number of stores he can keep. (6 pts)

We sort all stores by $(x_i - r_i)$ and we initialize the current location to 0. The we will select the store $x_{i'}$ that has Largest $(x_i + r_i)$ and $(x_i - r_i)$ is less than current location. We then update current location to $(x_{i'} + r_{i'})$ and continue the iteration until We cover L. Each time we set x_i as current Store, we will only have to look at store that's after x_i in the sorted list we did at begining.

b) Compute the runtime complexity of your algorithm in terms of N. Explain your answer by analyzing the runtime complexity of each step in the algorithm description. (4 pts)

The time is $\Theta(N \log N)$.

First, we sort all $(x_i - r_i)$, this step takes $\Theta(N \log N)$.
Second, we simply loop through all stores in the sorted List order. We will check each store only once and the operation for each store is constant. This step takes $\Theta(N)$.
Total $\Theta(N \log N + N) = \Theta(N \log N)$

c) Prove the correctness of your algorithm. (7 pts)

Suppose our solution takes $a_1 \dots a_i$, the optimal solution takes $o_1 \dots o_j$ and $j < i$.
Because we are doing greedy approach, we can be sure that $(x_{a_1} + r_{a_1}) > (x_{o_1} + r_{o_1})$ if they both cover the starting point. Then, we are choosing a_2 because it has largest $(x_{a_2} + r_{a_2})$ when $(x_{a_2} - r_{a_2})$ is less than $(x_{a_1} + r_{a_1})$. So, $(x_{a_2} + r_{a_2}) > (x_{o_2} + r_{o_2})$.
Therefore, we know that $(x_{a_j} + r_{a_j}) > (x_{o_j} + r_{o_j})$.
However, if a_j covers L, we won't have a_i as $i > j$,

An intentionally blank page.

which conflicts with optimal solution will covers L as $(x_{a_j} + r_{a_j})$ cannot cover L. $(x_{o_j} + r_{o_j}) < (x_{a_j} + r_{a_j})$ and cannot cover L. There exists an conflict so there is not solution has less stores than ours, which makes our solution optimal.

5) 8 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. $T(n) = 16\,T(n/4) + n / \log n$ $\quad$ $T(n) = \Theta(n^2)$

2. $T(n) = 8\,T(n/3)\;\; n + n \log n$ $\quad$ $T(n) = \Theta(n^{\log_3 8})$

3. $T(n) = T(2n/3) + n! + 3n^4$ $\quad$ $T(n) = \Theta(n!)$

4. $T(n) = T(3n/2) + n$ $\quad$ $T(n) = \Theta(n)$

6) 12 pts

Given a weighted undirected graph G = (*V*, *E*) where a set of cities *V* is connected by a network of roads *E*. Each road/edge has a positive weight, *w*(*u*, *v*) between cities *u* and *v*. There is a proposal to add a new road to the network. The proposal suggests a list *C* of candidate pairs of cities between which the new road may be built. Note *C* is a list of new edges (and their weights) in the graph. Your task is to choose the road that would result in the maximum decrease in the driving distance between *given* city *s* and *given* city *t*. Design an efficient algorithm for solving this problem, and prove its complexity in terms *V*, *E* and *C*.

We will first run dijkstra on s and t, we will then have shortest path from s to every node and shortest path from t to every node. We set initial value of distance from s to t be dist(s,t) and road to make be empty. We then loop through all edges in C. For each edge (v_1, v_2), we update the distance from s to t to the minimum value among current value, $dist(s, v_1) + dist(t, v_2) +$ weight (v_1, v_2), and $dist(s, v_2) + dist(t, v_1) +$ weight (v_1, v_2). If current value changed, we update road to make to edge (v_1, v_2). We will return the final result of road to make.

Time: $O(2|E|\log|E| + |C|) = O(|E|\log|E| + |C|)$

Dijk takes $O(|E|\log|E|)$ and each iteration takes constant so total takes $O(|C|)$

7) 17 pts

Kara is the owner of a honey fried chicken restaurant. One day she receives a request for preparing and delivering meals for research conference at USC. The coordinator of the conference has collected orders for all conference participants. Unfortunately, Kara's restaurant is too small to support all of the orders made. She has to decide to forfeit some orders, but she wants to earn as much as she can. Here is the information she has:

- There are N orders from the conference. For the i-th order, Kara knows it will take t_i minutes to prepare and will need k_i pounds of chicken.
- The coordinator will pay c_i dollars if Kara can deliver the i-th order. If Kara cannot deliver that order, she could cancel it without paying any penalty.
- Kara has only T minutes and K pounds of chicken to prepare a meal.
- T, K, N, and $\underline{c_i, t_i, k_i}$ for all i are positive integers for simplicity.

Help Kara to respond to the coordinator with the list of orders she could deliver by designing a dynamic programming algorithm.

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

Let OPT[T, K, N] T be the total number of time Left, K be total number of pounds left and N be the current order.

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

$$OPT[t,k,i] = \max\{OPT[t,k,i-1],\ C_i + OPT[t-t_i, k-k_i, i-1]\}$$

If $t < t_i$ or $k < k_i$:

$$OPT[t,k,i] = OPT[t,k,i-1]$$

c) Give the pseudocode for the DP algorithm using tabulation. (4 pts)

```
Initialize OPT[T+1, K+1, N+1], max_c = 0
for i = 0 ... N:
    for t = 0 ... T:
        for k = 0 ... K:
            if i == 0 or t == 0 or k == 0:
                OPT[t,k,i] = 0.
            if t_i > t or k_i > k:
                OPT[t,k,i] = OPT[t,k,i-1]
            else:
                OPT[t,k,i] = max{ OPT[t,k,i-1],
                                  C_i + OPT[t-t_i, k-k_i, i-1]
            If i == N:
                If max_c = max{max_c, OPT[t,k,i]}   }.
return max.
```

d) Compute the runtime of the above DP algorithm in terms of *N, T* and *K*. (2 pts)

runtime will be $O(NTK)$ as each cell will take $O(1)$ to compute and we have $O(NTK)$ cells.

e) Is your DP algorithm polynomial? Explain your answer. (3 pts)

No, it's pesudo-polynomial. As it's polynomial if N, T, K are polynomial.