CS570 Spring 2020: Analysis of Algorithms Exam I

GRADING RUBRIC

1) 20 pts
Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

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

[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.

[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.

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

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

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

[TRUE]
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.

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

[FALSE]
If an operation takes O(1) amortized time, then that operation takes O(1) worst case time.

Rubric: No partial credit – all or nothing.

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.

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

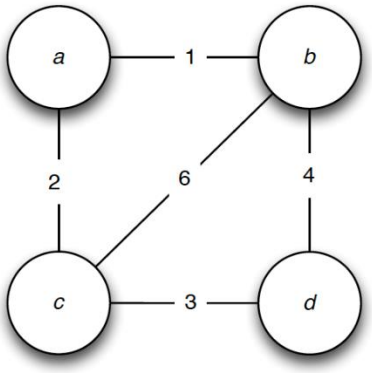
$$n^{\frac{1}{\log n}} < 2^{\sqrt{2\log n}} < (\sqrt{2})^{\log n} < \log(n!) < n\,(\log n)^3 < n^{\sqrt{n}} < (\log n)^n$$

Rubric: The grader will try to find the longest common subsequence between the student's solution and our solution. If the length of the longest common subsequence is L, the student will get 2*L points. Student can get 12 points if the student gets everything right but the order in reverse.

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.

(i) Use Kruskal's algorithm to find the minimum spanning tree (or forest if the graph is disconnected) of the induced graph on nodes V \ U
(ii) For each node $u \in U$, add incident edge (u, v) with the smallest weight (break ties arbitrarily).
Running Time: The running time is the O(|E| log |E|) since the running time of Kruskal's algorithm is O(|E| log |E|) and it takes O(|E|) time to find the edge of smallest weight for each u.

Correctness: For the correctness, we first argue that, without loss of generality, the optimum solution includes the minimum spanning forest in the induced graph on nodes V \ U. Note that the optimal solution must include a maximal acyclic graph on V \ U since adding only a single edge to each $u \in U$ can not complete a path between two nodes in V \ U. The cheapest maximal acyclic graph is the minimum spanning forest. Since we need to add exactly one edge between V \ U and each $u \in U$, to minimize the total weight we should add the cheapest such edge.

Rubric:

5 points - for algorithm description

4 points - for explaining its runtime complexity

3 points - for proving algorithm correctness

NOTE: Regrade requests on proof of correctness and time complexity analysis part will not be considered if not explicitly shown in the submission.

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.

General guidelines:

- If the students complete their solution which is correct, but some mistakes (for example, using "<" rather than " ≤ " in Part c)) can be found, the student could get 1 pt deducted for each part of Q4 (that is, Part a), b), and c))

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

1. For ith store, we construct the interval $[x_i - r_i, x_i + r_i]$, denoted as $[s_i, f_i]$. The problem now is to find the minimum number of intervals that can cover $[0, L]$.
2. Sort all interval by the increasing order of s_i and re-index the stores by this sorting result. (2 pts)
3. Let S be the set of intervals we have chosen and p be the minimum point in $[0, L]$ that intervals in S is not covered. At first, we have $S = \emptyset$ and $p = 0$. Repeat the following steps until $p \geq L$.
    - Iterate the sorted list of intervals and find all intervals $[s_i, f_i]$ such that $s_i \leq p$. Since the list is sorted, we can stop when $s_i > p$. (2 pts)
    - Among the intervals we choose from the first step, we add the interval with the maximum ending point into S. If there are multiple candidates, we can choose an arbitrary one. (1 pts)
    - Then, we remove all the intervals that is not chosen in the second step. (1pt)

Additional Notes:

- If the students describe the algorithm of traversal in English, they can also get full points. However, they should describe the following to get credits:

- What are the candidates of intervals (2pts)
- Which candidate will be chosen from the candidates (1pt)
- What to do after adding a new interval into the greedy solution. (1pt)

- If the students give a wrong greedy algorithm that none of the rubrics above are covered, then the students will get 2 pts at most.
- If the students give an algorithm without using greedy strategy, then the students will get 0 pt.
- If the students didn't mention sorting in Part a), but mention it in Part b), then the students can get the points for sorting.
- If the students give another correct greedy algorithm, but of time complexity higher than O(N log N) , then the students get at most 4 points for Part a)

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)

- Step 1 will take $O(N)$ time to construct intervals. (Not required)
- Step 2 will take $O(N \log N)$ for sorting. (1pt)
- Step 3: We will visit each interval at most three times: one for finding the intervals covering p, one for finding the maximum ending point, and another one for deleting from the list. Therefore, the time complexity of step 3 is $O(N)$. (2pts)

Thus, the total time complexity is $O(N \log N)$. (1pt)

Additional Notes:

- If the students give a wrong algorithm in Part a), it will not affect the credits they can get in the question. That is, if the students give the correct runtime complexity of their wrong algorithms, then they can still get 4 pts in Part b).

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

We consider the sorted list after Step 2 of our algorithm. Let $\{i_1, i_2, \dots, i_k\}$ the indices intervals chosen by our algorithms.

Feasibility: Since the original N intervals can cover $[0, L]$, our greedy algorithm can stop with a set of intervals that can cover $[0, L]$. Otherwise, we will find some point $x \in [0, L]$ that cannot be cover by any interval, which contradicts the given condition.

Optimality: Suppose that $\{j_1, j_2, \dots, j_m\}$ is the indices of the intervals in the optimal subset that can cover $[0, L]$. Since it is optimal solution, we have $m \leq k$. We can assume that $i_1 < i_2 < \cdots < i_k$ and $j_1 < j_2 < \cdots < j_m$.

Part A: We can prove by induction that $e_{i_p} \geq e_{j_p}$ for all $p \leq m$.
Proof: (4pts for the total proof of Part A)
Base case: $p = 1$. It is true because our greedy algorithm will always choose the one whose ending point is the greatest. (2pt)
Induction Hypothesis: If $e_{i_{p-1}} \geq e_{j_{p-1}}$,
Induction step: Now we will prove $e_{i_p} \geq e_{j_p}$. We know that to make sure all the intervals have covered $[0, L]$, we must have $e_{i_{p-1}} \geq s_{j_p}$, thus $e_{i_p} \geq s_{j_p}$.Therefore, $[s_{j_p}, e_{j_p}]$ must be considered during the greedy decision of the pth interval, then we have $e_{i_p} \geq e_{j_p}$ since we always choose the interval with the maximum ending point. (2pts)

Part B: Now, we are going to prove $m = k$ by contradiction. If $m < k$, by the property we have just proved, $e_{j_m} \leq e_{i_m}$. However, by our algorithm, we know that $e_{i_m} < L$. Thus, the optimal subset cannot cover $[0, L]$, which is a contradiction. (3pts)

Thus, our algorithm is the optimal.

Additional Notes:

- Some students can paraphrase the statement of Part A as the following:

  The intervals chosen by the greedy solution can cover more points/houses/range than the correspondent interval in the optimal solution.

  The student can get 1pt in Part A if the students wrote equivalent statements correctly and 3 pts if they prove part A correctly. However, if the students did not show or give reason why covering more can implies fewer number of used intervals, than the students cannot get 3pts in Part B.

- If the students give wrong algorithms in Part a) of Q4,
  - If the students give an algorithm that is similar to the rubrics in Q1, but not all parts are the same (for example, missing sorting or choosing the wrong first stores), then we should grade them with the normal rubrics.
  - If they prove the greedy choice is always ahead, but does not imply fewer intervals, then they could get at most 4 points.
  - Otherwise, they could get at most 2 points if they write something that follows the proof of greedy algorithms in the class.

The wrong greedy algorithms: (WA)

1. Sort by the starting point by the increasing order, choose the first one that can cover the starting point and skip all the following intervals that are already covered. Continue doing this until covering all [0, L].
    a. Counterexamples: [0, 4], [1, 5], [2, 8] when L = 8.
    b. The algorithm will choose [0, 4], [1, 5] and [2, 8], but [0, 4], [2, 8] is better
2. Sort by the x_i increasingly, choose the first one that can cover the left-most uncovered interval and skip all the following intervals that are already covered.
    a. Counterexamples: [0, 4] (x_1 = 2), [0, 6] (x_2 = 3), [4, 6] (x_3 = 5) when L = 6
    b. The algorithm will choose [0, 4] and [0, 6] while [0, 6] is the optimal solution.
3. Sort by the range ($2r_i$) / radius (r_i) decreasingly, choose the first one and skip all the stores that have been covered.
    a. Counterexamples: [0, 6], [1, 7], [2, 8] (The range length is 6) and [0, 4], [4, 8] (the range length is 4) when L = 8
    b. The algorithm will choose the first threes and ignore the last two since their range length is small and [0, 6] and [2, 8] can cover them.
4. Sort by the starting point by the increasing order, choose the one whose starting point is rightmost to the currently covered intervals. Continue doing this until covering all [0, L]
    a. Counterexample:
    b. The algorithm will choose the first threes and ignore the last one since their ending point is the rightmost of the currently covered intervals.
5. Sort by x_i (or starting point) increasingly, choose the one with the largest x_i (or starting point) that can cover the left-most uncovered interval and skip all the following intervals that are already covered. Continue doing this until all is covered
    a. Counterexample: [-5, 7] (x_1= 1), [0, 6] (x_2= 3), [6, 7] (x_3 = 3.5), where L = 7.
    c. The algorithm will choose the last two and ignore the last one since the x_2 is farther than x_1.

Partially Correct Greedy Algorithms:

1. A similar algorithm with the one in this answer, except that choosing the one with the minimum $x_i - r_i$ for the first store.
    a. This is a solution to finding the smallest subset of intervals whose union is the same as the union of all intervals.

b. However, these two problems are different. We only need to cover from the starting point of the avenue, i.e. point 0. Thus, it will affect the proof of the base case in the correctness of the algorithm.

2. Sort all intervals by r_i decreasingly, repeat the following: Choose the one with the maximum coverage area. Then, update all other intervals' effective coverage by removing the area that has been covered by the one we have chosen. This is correct but of high time complexity. For each time updating the whole interval list, we need to iterate all elements and will take $O(N)$. The total time complexity will be at least $O(N^2)$

Alternative Greedy Algorithms:

- Construct sets of point $S = \{0, L\} \cup \{x_i \pm r_i \mid i = 1,2,\ldots,N\}$. Sort S in increasing order and remove the duplicate points. Relabel all the points with their indices. Now we can assume the locations involved are 0, 1, ..., M-1, where M is the number of elements in S. Now, we could solve this problem with all locations as integer.

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

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

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

4. $T(n) = T(3n/2) + n$ $T(n) = NA$

Rubric:

No partial credit – all or nothing.

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*.

Algorithm (10pts):

1. Run Dijkstra algorithm from s to calculate shortest distances from s to all other cities
2. Run Dijkstra algorithm from t to calculate shortest distance from t to all other cities
3. For every candidate pair of cities {u,v}, the shortest path distance between s and t which covers road $u \rightarrow v$ is **min**(dist(s,u) + dist(t,v) + length(u,v), dist(s,v) + dist(t,u) + length(u,v)).
4. Choose the shortest distances from loop-3.

- If the selected distance is longer than original dist(s,t), any candidate road cannot decrease distance between s and t.
- Else, choose the {u,v} pair that produces shortest new distance. In the case of tie, choose one arbitrarily.

[He: -2pts for all cases that forget to min between (u,v) and (v,u).]
[He: -5pts for all "baseline" solution: just adding the edge and run Dijkstra. I really want to give 0 pt but....]

Complexity (2pts): Complexity of running Dijkstra's algorithm is $\Theta(2 \times |E| \log|V|)$, the complexity of running step-3 is $\Theta(2 \times |C|)$, thus total complexity is $\Theta(|E| \log|V| + |C|)$.

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 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)

$OPT[i, t, k]$ is the maximum payment Kara can earn given t minutes and k pounds of chicken if we consider the first i orders.

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

If $i > 0$, then

$$OPT[i,t,k] = \begin{cases} \max(OPT[i-1,t,k], OPT[i-1,t-t_i,k-k_i] + c_i), & t \geq t_i \wedge k \geq k_i \\ OPT[i-1,t,k], & otherwise \end{cases}$$

(2pts for the first case, 1pt for the second case)

If $i = 0$, then $OPT[i,t,k] = 0$, if (2 pts)

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

```
for(int i = 0; i <= N; i++)
  for(int t_ = 0; t_ <= T; t_++)
     for(int k_ = 0; k_ <= K; k_++) {

       if (i == 0) {
         OPT[i][t_][k_] = 0;
       } else {
         if (t_ >= t[i] && k_ >= k[i])
             OPT[i][t_][k_] = max(
                          OPT[i-1][t_][k_],
                          OPT[i-1][t_ - t[i]][k_ - k_i]);
         else
             OPT[i][t_][k_] = OPT[i-1][t_][k_];
     }
```

Rubrics:

The students can receive the full credit if the pseudocode they give are understandable for the graders. However,

- If the grader finds that during computing some subproblem, the subproblems it depends has never been initialized or computed, then 3 points will be deducted.
- If the students provide wrong code that mismatch the recurrence relation in part

b), 2 points will be deducted for each mistake the students make in (subproblem calculation/if-condition/for-loop condition) until all points in this part have been deducted.

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

$$O(NTK)$$

No partial credit for this part.

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

No partial credit for this part.

Required statements, direct or indirect:

- The algorithm is not polynomial, but rather pseudo-polynomial or exponential in the lengths of both T and K.
- Explanation that this is so because these two input variables represent bits or are numbers that do not represent the size of the input.

Equations are optional but are checked if given.

T and K are not constants, and C is not an input for the Big-O calculation.