Introduction to Analysis of Algorithms CS4820/5820 Fall 2021	Prelim 1 Solutions

PART I: SHORT ANSWER QUESTIONS (20 POINTS) Short answer questions. Each of these questions asks for a True/False answer or a short answer to a question. Follow the instructions for each question; you may be asked to give a brief (one sentence) explanation. **No proofs are required in this section.**

1. (6 points)

   (a) (2 points) Consider the stable matching problem with hospitals h_1, h_2, h_3 and residents r_1, r_2, r_3, with the following preferences of the hospitals and residents (where 1st is the most preferred option, and 3rd is the least preferred option).

	1st	2nd	3rd
h_1	r_1	r_3	r_2
h_2	r_1	r_2	r_3
h_3	r_2	r_1	r_3

	1st	2nd	3rd
r_1	h_3	h_2	h_1
r_2	h_2	h_3	h_1
r_3	h_2	h_1	h_3

   Give the stable matching found by the Gale-Shapley algorithm with hospitals proposing.

   Solution: $\{(h_1, r_3), (h_2, r_1), (h_3, r_2)\}$

   (b) (4 points)

   Consider the Minimum Cost Spanning Tree (MST) problem in an undirected graph $G = (V, E)$ with distinct edge costs c_e for $e \in E$.

   Determine whether the following statement is true or false. If false, give a counterexample; make sure to explain why your example contradicts the statement. If true, give a short (one phrase or one sentence) explanation why.

   True or false: If there exists a cycle in the graph in which e is the cheapest edge, then e *must be* in the Minimum Cost Spanning Tree.

   Solution: False. For example, suppose we have nodes A, B, C, D with edges $(A, B), (B, C), (C, D), (D, A)$ and (A, C) and costs $c_{AB} = 1, c_{BC} = 2, c_{CD} = 4, c_{DA} = 5, c_{AC} = 3$. Then (A, C) is the cheapest edge in the cycle on A, C, D, but it is also the most expensive edge in the cycle on A, B, C; so by the cycle property it does not occur in the MST.

2. (6 points)

   In class, we considered an algorithm for finding the k-th smallest value in an unsorted set S of all different elements, in which we choose an element x uniformly at random from S, and recurse if its index i (in the sorted set S) is in the middle half: that is,

$n/4 \le i \le 3n/4$. Here we consider a version where instead we recurse if the index i falls in the middle *third*: $n/3 \le i \le 2n/3$. Below is the code for the algorithm:

QuickSelectIII(S, k)
If $|S| \le 3$: sort and return the kth element
Else:
  Set $i = \infty$, and $n = |S|$
  While i does not satisfy $n/3 \le i \le 2n/3$:
    Select $x \in S$ uniformly at random.
    Let $S_- = \{y \in S : y < x\}$ and $S_+ = \{y \in S : y > x\}$
    Set $i = |S_-| + 1$
Endwhile
If $i = k$: Return x
Else if $i > k$: QuickSelectIII(S_-, k)
Else: QuickSelectIII$(S_+, k - i)$

Answer the following questions, with no explanation necessary:

(a) (2 points) What is the probability that the `While` loop terminates in just one iteration? **Solution:** $1/3$.

(b) (2 points) What is the expected number of comparisons done by the `While` loop (so find the number of comparisons per pass multiplied by the expected number of times we pass through the loop)? **Solution:** There are $n - 1$ comparisons in one pass, and the expected number of passes is $1/(1/3) = 3$, so the expected number of comparisons by the loop is $3(n - 1)$.

(c) (2 points) If $T(n)$ is the expected number of comparisons used by the algorithm, what is the recurrence to compute $T(n)$? Give an equation in terms of n, using $T(n/p)$ for a $p > 1$ (that is, problem of size smaller than n) to describe a recursive call of this algorithm (where (as you saw in class) n/p should be an upper bound on the subproblem's size; not the expected size of the subproblem). You do not have to solve the recurrence, just state it. **Solution:** $T(n) = T(\frac{2}{3}n) + 3(n-1)$.

3. (8 points) In this question, we consider the shortest path problem in a directed graph $G = (V, E)$ where each edge (u, v) has an associated cost/length c_{uv} (the costs may be negative). The term shortest path is used to indicate the path with the lowest sum of costs (not the fewest edges).

   (a) (4 points) Suppose $V = \{t, v_1, v_2, v_3, v_4\}$. The following table contains $D(i, v) =$ *cost of the shortest path from* v *to* t *using at most* i *edges* for every $v \in V$, and $i = 0, \ldots, 5$.

$D(i,v)$					
i \ v	t	v_1	v_2	v_3	v_4
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	$+\infty$	$+\infty$	$+\infty$	7
2	0	9	11	5	7
3	0	7	10	5	4
4	0	6	8	2	4
5	0	0	7	2	1

What does the table allow you to conclude about whether or not the graph has a negative cycle? Explain.

Solution: The graph must have a negative cycle. Note that $n = 5$ and the table shows that $D(n,v) < D(n-1,v)$ for $v = v_1, v_2, v_4$. Take for example v_1, if we trace back from $D(n,v_1)$, we find the min-cost path from v_1 to t using at most $n = 5$ edges. The path must have 5 edges (because $D(4,v_1) > D(5,v_1)$), so must contain a cycle because there are only 5 nodes in the graph, and the cycle must be negative, as deleting the cycle we get a path with at most 4 edges, and $D(4,v_1) > D(5,v_1)$ implies that this path must have a larger cost.

Alternatively, we could reason about the fact that the table shows that the only edge into t is (v_4,t) with cost 7. But $D(3,v_4) < 7$, so the path of 3 edges from v_4 to t must go through a cycle of negative cost before taking the edge (v_4,t).

(b) (4 points) Suppose you've run Bellman-Ford twice (on a graph that *does not have negative cycles*) to generate two tables, containing the following for every $v \in V$ and $i = 0, \ldots, n$:

$d(i,v)$ = cost of shortest s-to-v path with at most i edges
$D(i,v)$ = cost of shortest v-to-t path with at most i edges

Let w be a node in V; you would like to know whether there is a shortest s-to-t path that goes through w. Give an algorithm that uses the given data to answer this question. What is the running time of your algorithm (when used for *one* node w only, and excluding the time to compute the two tables)?

You do **not** need to argue correctness, but for full credit, your algorithm should be as fast as possible.

Solution: The shortest path from s to t has length $d(n-1,t) = D(n-1,s)$. There is a shortest s-to-t path that goes through w if and only if $d(n-1,w) + D(n-1,w) = d(n-1,t)$.

The running time of the algorithm is $O(1)$, because it just needs to look up three values and check if $d(n-1,w) + D(n-1,w) = d(n-1,t)$.

PART II: LONG ANSWER QUESTIONS (30 POINTS)

4. (16 points) You have n items you want to paint. For each item i, you know the time p_i that it takes to paint the item, and the time d_i it takes the item to dry. You can paint only one item at a time, but (obviously) they can dry simultaneously. You want to determine in which order to paint the items, so that the time when all items are painted and dry is minimized.

   Below, you are given two potential algorithms to solve the problem. For **each** proposed algorithm, **either** prove that it returns an optimal solution, **OR** give a example input on which the algorithm fails to return an optimal solution.

   (a) Order the items by increasing painting time (so the item with the smallest painting time gets done first).

   (b) Order the items by decreasing drying time (so the item with the largest drying time gets done first).

Solution: We are trying to minimize the time when all items are painted and dry; in other words, if C_i is the time when item i is done, then we want to minimize $\max_{i=1,\ldots,n} C_i$.

(a) **Not optimal**

For example, suppose we have items $1, 2$ with $p_1 = 1, d_1 = 1, p_2 = 2, d_2 = 2$. Then ordering by increasing painting time puts 1 first, and gives completion times $C_1 = 2, C_2 = 5$ so the objective is 5, whereas ordering by decreasing drying time puts 1 last, and gives completion times $C'_1 = 4, C'_2 = 4$ with objective 4.

(b) **Optimal**

Proof using an exchange argument. Let O be an optimal ordering of the items. We will show that we can modify O into the ordering by decreasing drying time without increasing the objective. If O is different from the ordering by decreasing drying time, there must be two items j, k that are ordered consecutively in O such that $d_j \leq d_k$. Let s be the time when in O we start painting j, then $C_j = s + p_j + d_j$ and $C_k = s + p_j + p_k + d_k$. If we switch j and k in O, the completion time C_i for any item $i \neq j, k$ is unchanged, and we have

$$\begin{aligned} C'_k &= s + p + k + d_k \leq s + p_j + p_k + d_k = C_k \\ C'_j &= s + p_j + p_k + d_j \leq s + p_j + p_k + d_k = C_k \end{aligned}$$

Let O' be the new ordering after switching j and k in O, then maximum completion time of O' is not more than the maximum completion time of O. Also, O' has one fewer pair of items that are out of order with respect to the ordering by decreasing drying time. We can repeate this argument until O is identical to the ordering by decreasing drying time without increasing the maximum completion time. Thus the ordering by decreasing drying time is optimal.

Comment on the grading:

You should have interpreted this question as asking to minimize the maximum completion time (minmax objective). Then (a) is wrong and (b) is correct, and you could get up to 5 points for (a) and 11 points for (b). If you interpreted the question as minimizing the sum of completion times (minsum objective), you did not solve the correct problem (and the problem becomes simpler, because the drying times become irrelevant as they add a fixed constant to the objective). With this interpretation, (a) is right and (b) is wrong, and you could get up to 10 points for (a), and up to 3 points for (b).

5. (14 points) A famous long-distance hiking trail in Europe consists of n "stages", numbered $1, \ldots, n$. The stages need to be walked consecutively, and each stage ends in a town where you can spend the night. You are given a charm-rating for each of the towns; the town at the end of stage i has rating $v_i > 0$.

   You are planning to spend k days (and nights) to hike the route. You will hike between 1 and r stages each day, and you must complete the last stage on the k-th day. You may assume $n < kr$ (so it is possible to complete the entire tour within the restrictions). A feasible plan is thus a sequence $(i_1, i_2, \ldots, i_k)$ (which we can interpret as saying that you spend the j-th night at the town at the end of stage i_j) such that $i_1 < i_2 < \ldots < i_k$, $i_k = n$, $1 \leq i_1 \leq r$, and $i_j - i_{j-1} \in \{1, 2, \ldots, r\}$ for $j = 2, \ldots, k$.

   Give a polynomial time algorithm to find the total charm of the feasible plan that maximizes the total charm of the towns where they spend the night. You do not need to return the actual towns selected, just their total charm. You should **give the algorithm and analyze the running time**, but you do **not** have to give a proof of correctness. Pseudocode is acceptable, as long as you explain in English the intended meaning of your variables, and you explain the reasoning behind your algorithm.

   Example Suppose the route has $n = 10$ stages, you want to stay for $k = 4$ nights, and you can hike between 1 and $r = 3$ stages per day.

Town	1	2	3	4	5	6	7	8	9	10
Charm	5	2	0	3	8	2	7	3	6	2

   In this case, you could stay in town 1, town 4, town 7, and town 10, for a total charm value of 17. The optimal solution in this example would be to stay in town 2, 5, 7, and 10, for a total charm value of 19.

   Solution: Note that the assumption that charm is positive means that you will always choose to spend exactly m nights.

Solution I: Dynamic Programming Define $OPT(i, m)$ as the *maximum total charm of the towns where you spend the night if you spend m nights and hike stages 1 to i, and spend the last night in town i.*

$OPT(i, 1) = v_i$ if $i \leq r$, $OPT(i, 1) = -\infty$ if $i > r$.

For $i = 1$ to n

For $m = 2$ to k

$OPT(i, m) = v_i + \max_{\max\{0, i-r\} \leq j \leq i-1} \{OPT(j, m-1)\}$

Return $OPT(n, k)$

Running time is $O(nkr)$: we are computing nk values, and each takes $O(r)$ time because it compares r options.

Solution II: Reduction to shortest path

Create a node i for the town at the end of stage i, for $i = 1, \ldots, n$, plus a start node 0. Create an edge (i, j) for every $i \in \{0, \ldots, n-1\}$ and $j = i+1, i+2, \ldots, i+r$ of cost $-v_j$. Use Bellman-Ford to find the shortest path from 0 to n using at most (or exactly) k edges. Return $D[k, n]$ (length of the shortest path from 0 to n using (at most or exactly) k edges).

Running time is $O(n + nr)$ to create the graph with $n + 1$ nodes and nr edges. Running Bellman-Ford on a graph with N nodes and M edges to find the shortest path of at most k edges is $O(NM)$ (also OK if you say $O(N^3)$ or $O(N^2 k)$) so the total running time is $O(n^2 r)$ if using the first bound on Bellman-Ford (or $O(n^3 r^3)$ or $O(n^2 r^2 k)$ if using the second or third bound).