Name: Param Somane **Access ID:** pss5256

Problem 1 **Points:**

1. Order of edges added to MST when Kruskal's algorithm is run on the graph:

   1. (a, c) – 2
   2. (a, b) – 3
   3. (b, e) – 3
   4. (c, f) – 3
   5. (d, e) – 4
   6. (f, g) – 6

2. Order of vertices added to MST when Prim's algorithm is run on the graph:

   1. a
   2. c
   3. b
   4. e
   5. f
   6. d
   7. g

Problem 2 **Points:**

Let $G = (V, E)$ be an undirected graph with edge weights $w(e)$ for all $e \in E$. Let $T = (V^*, E^*)$ be the MST of G. Let $e' \in E$ be some edge. First, we assume that $G' = (V, E \setminus \{e'\})$ is connected, that is the graph resulting from the deletion of e' from G consists of a single connected component and thereby has a spanning tree. Depending on whether or not $e' \in E^*$, we consider the following two cases.

If $e' \notin E^*$, that is e' is not in the MST, then the deletion of e' from G does not affect the MST of G because $V^* = V$ and $|E^*| = |V| - 1$ remain unaffected by the deletion. Thus, the MST of the resulting graph is T itself.

If $e' \in E^*$, that is e' is part of the MST of G, then the deletion of e' from G, and thereby from T, splits T into two disjoint, connected components (subtrees). As we have assumed that G' is connected, there must be at least one edge $e'' \in E \setminus \{e'\}$ that connects these two subtrees. The MST of G' is then the union of these two subtrees with e'', where e'' is chosen such that $w(e'')$

is minimum among all viable candidate edges. The resulting tree $T' = (V^{*\prime}, E^{*\prime})$, where $V^{*\prime} = V$ and $E^{*\prime} = (E^* \setminus \{e'\}) \cup \{e''\}$ is the MST of G' because it is connected by construction, $|E^{*\prime}| = |E^*| - 1 + 1 = |V| - 1$, and $\sum_{e \in E^{*\prime}} w(e) = \sum_{e \in E^*} w(e) - w(e') + w(e'')$ is minimized. Note that this follows from the cut property of minimum spanning trees as $E^* \setminus \{e'\}$ is a partial MST of G and consequently, $E^{*\prime}$ must be an MST of G' as $E^{*\prime} \subseteq E \setminus \{e'\}$.

Running time: Checking whether or not $e' \in E^*$ takes $O(|E^*|) = O(|V|)$ time and in the case that $e' \in E^*$, searching for the required e'' with minimum possible weight takes $O(|E|)$ times as we need to cycle through all the edges in $E \setminus E^*$. Since $|V| < |E|$, the algorithm runs in $O(|E|)$ time.

Problem 3 **Points:**

Let $G = (V, E)$ be an undirected graph with distinct edge weights $w(e)$ for each $e \in E$.

1. Let C be a cycle of G and $e^* := arg\ \max_{e \in C} w(e)$ be the edge with the largest weight in C.
Proof. Suppose, toward a contradiction, that $e^* = (u, v)$ appears in a minimum spanning tree $T = (V^*, E^*)$ of G. Now, if we omit e^* from G, then we will split T into two disjoint subtrees because $e^* \in E^*$ by assumption. Since C is a cycle, there exists an alternative path from u to v that does not pass through e^*. In particular, there exists an edge $e' \in C$ that connects the above two subtrees and yields $T' = (V^*, (E^* \setminus \{e^*\}) \cup \{e'\} =: E^{*\prime})$ as another spanning tree of G because $V^* = V$, $|E^{*\prime}| = |E^*| - 1 + 1 = |V| - 1$, and it is connected. Moreover, $\sum_{e \in E^{*\prime}} w(e) = \sum_{e \in E^*} w(e) - w(e^*) + w(e') < \sum_{e \in E^*} w(e)$ because $w(e') < w(e^*)$ by the maximality of the weight of e^* in C and the distinctness of weights yields a strict inequality. This contradicts the minimality of T as an MST of G. Ergo, e^* cannot appear in any minimum spanning tree of G. ■

2. It is assumed that G is connected and so has an MST. Suppose that G satisfies $|E| = |V| + 9$. Recall that if $T = (V^*, E^*)$ is an MST of G, then $V^* = V$ and $|E^*| = |V| - 1$. This implies that we need to remove $(|V| + 9) - (|V| - 1) = 10$ suitable edges from E to obtain E^*. Observe that we must have a cycle in G because $|E| \neq |V| - 1$ and G is connected. Subsequently, we run the DFS algorithm on G to find the first occurrence of a cycle by detecting presence of a back-edge and we alsom keep track of the edge in that cycle with the maximum weight. We delete this edge from E. This process is then repeated ten times and during each iteration, we are guaranteed to detect a cycle in the current graph because it has more edges than $|V| - 1$. At the end of this process, we are left with a subgraph $T = (V^*, E^*)$ of G that is connected, has $|V| - 1$ edges, and spans all the V vertices of G. Thus, T must be a spanning tree of G. Observe that the edges with the maximum weight in any cycle of G cannot appear in T by 3 (1) and we have ommitted precisely these edges from T by construction. Therefore, $\sum_{e \in E^*} w(e)$ is minimal and so T is a minimum spanning tree of G.

Running time: Accounting for all the ten iterations, the total running time of the algorithm is

$$O(|V| + |E|) + O(|V| + |E| - 1) + \cdots + O(|V| + |E| - 9) = O(10 \cdot |V| + 10 \cdot |E| - 45)$$
$$= O(20 \cdot |V| + 45) = O(|V|).$$

Problem 4 **Points:**

Let $A[1 \cdots n]$ be an array of n intervals, where $A[i] = [L_i, R_i]$ for each $1 \leq i \leq n$. The idea is to first sort the intervals in increasing order of L_i. Then the left endpoint of the last interval, $L_n \geq L_i$ for all i and so we may choose L_n to be in S. We examine the intervals in reverse order (descending order of L_i) and skip any intervals which contain L_n. At the first occurrence of an interval $A[k]$ which does not contain L_n, we include L_k in S because $A[k]$ is disjoint from $[L_n, R_n]$ as $R_k < L_n$. This process is repeated until all the n intervals have been examined and the resulting set S contains exactly one integer inside every interval $A[i]$; thus, $|S|$ is minimal as required. The pseudocode below illustrates this idea.

```
Algorithm INTERVAL-GREEDY (A[1 ··· n])
    sort A in increasing order of L_i;
    init j = n; //keep track of previously added interval
    init S = {L_n};
    for i = n − 1 → 1
        if R_i < L_j //last element of S ∉ A[i]
            S.add(L_i);
            set j = i;
        end if;
    end for;
    report |S| as the least possible size of S;
end algorithm;
```

Running time: The running time of the algorithm is dominated by the sorting of intervals according to the left endpoints because the for-loop runs in linear time; thus, the running time of the algorithm is $O(n \log n)$.

Problem 5 **Points:**

Let the list of n items $L = \{p_1, \cdots, p_n\}$ with distinct importance rates $w_I \geq 0$ be given. Let $C[1 \cdots n]$ be the list of optimal costs that sum up to the minimized total money required to buy all the items subject to the stated two constraints. Observe that for any given $1 \leq i \leq n$, whenever valid, we have that $w_{i-1} < w_i \implies C[i-1] < C[i]$ and $w_i > w_{i+1} \implies C[i] > C[i+1]$. Thus, we first traverse the rates w_i in the forward direction and mark the costs for the sequences of increasing rates, while neglecting any decreasing sequence. Next, we traverse the rates in the reverse direction and mark the costs for sequences of decreasing rates, while neglecting any increasing sequence (with respect to the forward direction). Lastly, we take the maximum of the increasing and decreasing costs as the final cost for each item. The resulting array of costs preserves both the increasing as well as the decreasing order of importance rates while assigning costs as both of the above inequalities hold true for all i.

The greedy portion of the algorithm lies in setting to 1 (the lowest possible cost) any decreasing sequence with respect to either direction of traversal. Hence, the resulting array of costs sums up

to the minimized total money needed to be spent because if it did not, then one of the $C[i]$'s would need to take on a lower value than its assigned value to lower the total sum and subsequently, its cost would not satisfy the second required condition with respect to its adjacent neighbors. The algorithm below portrays this idea.

Algorithm GREEDY-ITEMS ($\forall 1 \leq i \leq n$, $w_i \geq 0$ distinct importance rates)
  init $C[1 \cdots n]$;
  init $incC[1 \cdots n]$;
  init $decC[1 \cdots n]$;
  set $incC[1] = 1$, $decC[n] = 1$;
  for $i = 2 \rightarrow n$
    if $w_{i-1} < w_i$: $incC[i] = incC[i-1] + 1$;
    else: $incC[i] = 1$;
  end for;
  for $i = n-1 \rightarrow 1$
    if $w_{i+1} < w_i$: $decC[i] = decC[i+1] + 1$;
    else: $decC[i] = 1$;
  end for;
  for $i = 1 \rightarrow n$
    $C[i] = \max\{incC[i], decC[i]\}$;
  end for;
  report $sum(C)$ as the minimized total money for buying all items;
end algorithm;

Running time: The algorithm implements three for-loops, each of which runs in linear time. Thus, the algorithm runs in $O(n)$ time.

Bonus Problem

We wish to find an order I of jobs which minimizes $\sum_{i=1}^{n} s_i \cdot f_i =: \xi(I)$. For this purpose, we sort the n jobs in descending order of significance to processing time ratio s_i/t_i because this permits us to isolate jobs with a greater significance per unit time and assign them to be earlier in the order; also, this greedy strategy works in the proof provided below. We claim that this sorted order minimizes $\sum_{i=1}^{n} s_i \cdot f_i$.

Running time: Sorting by s_i/t_i takes $O(n \log n)$ time, which dominates in the algorithm.

Correctness: Let $I = \{i_1, i_2, \cdots, i_n\}$ be an optimal order of jobs (sorted in descending order of ratios as mentioned above) and suppose that job j is immediately before job i in I. Hence, $s_j/t_j \geq s_i/t_i$. Let J be another order of jobs obtained by swapping positions of jobs i and j in I. Let T be

the sum of processing times of all jobs prior to job j in the order I. Observe that

$$\begin{aligned}\xi(J) - \xi(I) &= (s_i(T+t_i) + s_j(T+t_i+t_j)) - (s_j(T+t_j) + s_i(T+t_j+t_i)) \\ &= s_j t_i - s_i t_j = t_i t_j \left(\frac{s_j}{t_j} - \frac{s_i}{t_i}\right) \\ &\geq 0 \quad (\because j \text{ is before } i \text{ in descending order } I)\end{aligned}$$

Ergo, $\xi(I) \leq \xi(J)$, that is the sum of products of significance and finishing-time of jobs in J is at least as much as that of the jobs in I; hence it is not possible that I is not an optimal order of jobs that minimizes $\sum_{i=1}^{n} s_i \cdot f_i$.