MinorI (COL 351)

Read the instructions carefully:

- The algorithms should be explained in English (you can use pseudocode, but it should be easy to understand). Do NOT use examples/figures to explain your algorithm. We will NOT read any such explanation.
- The proofs should be brief and statements in the proof should follow a logical sequence. If using induction, you must clearly state the induction hypothesis. Do NOT use examples or special cases to explain the proof. We will NOT read any such part of the proof.

1. You are given a bag with capacity B, and a set of n items. For each item i, you are given its size s_i and profit p_i (to be more precise, you are given two arrays, where the first array contains the sizes of the n items, and the second array contains profits of the n items). You are required to pick a subset of these items such that their total size does not exceed B, and the profit should be maximized. You are allowed to pick *fraction* of an item – if you pick a fraction f of an item i, then you will get $f \cdot p_i$ profit for this item, and the size occupied by the item will be $f \cdot s_i$.

   Example: Suppose the bag has capacity 5, and there are two items – item 1 has size 6 and profit 8 and item 2 has size 3 and profit 9. So we can pick half of item 1 and 2/3 of item 2. The total size occupied is $\frac{1}{2} \cdot 6 + \frac{2}{3} \cdot 3 = 5$, which is at most (in fact, equal to) the bag's capacity. The profit earned is $\frac{1}{2} \cdot 8 + \frac{2}{3} \cdot 9 = 10$.

   Prove that the following greedy algorithm is optimal: sort the items in decreasing order of the ratio p_i/s_i. Start placing items in this order in the bag, till the bag overflows. The last item which causes the bag to overflow is picked fractionally so that we do not exceed the bag's capacity. In the example above, the ratios are 8/6 and 9/3, and so we first pack all of item 2 in the bag. Now we can only pack 1/3 of item 1. The profit is $9 + 8/3$. (**7 marks**)

   Order the items in decreasing order of density (ratio of profit to size), and let this ordering be $i_1, i_2, \ldots, i_n$. Let G be the greedy algorithm and suppose it picks items $i_1, i_2, \ldots, i_k$ in this order, where i_k was picked partially to a fractional extent of f_k (f_k could be 1). We prove the following statement by induction on $j, 0 \leq j \leq k$: there is an optimal solution which picks $i_1, \ldots, i_j$ completely (or if $j = k$, then picks i_k to f_k fraction) (**3 marks**). The base case for $j = 0$ is true trivially. Suppose it is true for some $j < k$. Let O be such an optimal solution, which has picked $i_1, \ldots, i_j$. If it picks i_{j+1} to the maximum possible extent (i.e., completely if $j+1 < k$ or i_k to f_k fractional extent), then we are done because O also satisfies the induction hypothesis for $j+1$ (**1 mark**). So suppose it does not pick i_{j+1} to this extent. Let f_{j+1} be f_k if $j+1 = k$ or 1 otherwise. Say O picks i_{j+1} to fraction $f'_{j+1} < f_{j+1}$. Then it is filling at least $s_{j+1} \cdot (f_{j+1} - f'_{j+1})$ volume of the bag with items $i_{k+1}, \ldots, i_n$ (**1 mark**). We now show that if we replace this volume with i_{j+1}, the profit cannot go down (**1 mark**). Indeed, for every unit of volume if we replace an item $i_\ell, \ell > j+1$ with i_{j+1}, then the change in profit is $p_{j+1}/s_{j+1} - p_\ell/s_\ell \geq 0$. Since, we can fill i_{j+1} completely by removing fractions of items with smaller density, we can only increase the profit of our solution (**1 mark**). This proves the induction hypothesis.

   Common Mistakes: Many students considered optimum as an ordering of items. But there is no ordering here, a solution just needs to say which items are in and which are out. Many students looked at optimum which are of the form – take some items completely, and take one item fractionally. But without prior justification, it is not clear why this is a valid assumption. Many items could be taken fractionally. Similarly, in the induction hypothesis, many have claimed that suppose optimum

also takes the first i items as in the greedy algorithm. Now suppose at the next step, they differ and take different items. But again, they could differ by taking different fractions of the same item. Finally, many have made the assumption that p_i/s_i are distinct – this need not be the case, two different items could have the same ratio, and so, there could be multiple optimum solutions.

2. Give an $O(n)$ time algorithm to find an optimal solution to the problem described in Question 1 above. (**7 marks**)

   This can be done by divide and conquer. Using the notation above, we just need to know the last item which is fractionally (or completely) assigned to the bag (**2 marks**). First find the median item (according to density) – this takes $O(n)$ time (**1 mark**). Let this item be i_m. We can easily tell whether $k \geq m$ or $k < m$ – indeed, try filling all jobs with density at most that of i_m. If we fill the bag before reaching i_m, it means $k < m$ or else, $k \geq m$. If $k < m$, we recurse with the set of items being $i_1, \ldots, i_{m-1}$, otherwise we recurse on $i_m, \ldots, i_n$ (with the size of the bag being B minus total size of items in $i_1, \ldots, i_{m-1}$) (**3 marks**). Note that the recurrence is $T(n) = T(n/2) + O(n)$, and so, $T(n) = O(n)$ (**1 mark**).

3. You are given an undirected connected graph G with n vertices and $n + 10$ edges. Each edge has a positive weight, and you can assume that the weights are distinct. Give an $O(n)$ time algorithm to find the minimum spanning tree of G. Justify why your algorithm is correct. (**6 marks**)

   We know that if there is cycle, then the maximum weight edge in this cycle cannot be part of the minimum spanning tree (no need to prove this) (**2 marks**). So, we repeat the following steps: if there is a cycle, remove the largest weight edge from it (**2 marks**). This can be done in $O(n)$ time, because cycle can be found in $O(n)$ time by BFS/DFS and then the adjacency list can be updated in $O(n)$ time (**1 mark**). We need to perform this 11 times only because any spanning tree has $n - 1$ edges (**1 mark**).

   Common Mistakes: First common mistake is to look at *all* cycles, and eliminate largest edges. Note that there could be many more than 11 cycles in the graph, and DFS does not enumerate **all** the cycles. It just finds a few. If you want DFS to output all the cycles, it could take more than linear time (at least it is not clear directly from the description of DFS how it will do so in linear time). The other common mistakes are algorithms like this – look at edges in decreasing order of weight. When we look at an edge, discard it if it forms a cycle. This again, cannot be implemented in linear time. Similarly, all algorithms which look at the 11 edges with the highest weight and so some processing there after are either wrong or don't run in linear time. Checking is an edge creates a cycle or not will need at least $\log n$ time.