CS5800: Algorithms — Spring ’21 — Virgil Pavlu

Homework 4
Submit via Gradescope

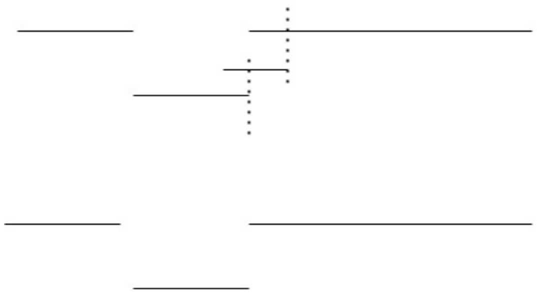
Name: Ketaki Nitin Kolhatkar
Collaborators: Madhusudan Malhar Deshpande

Instructions:

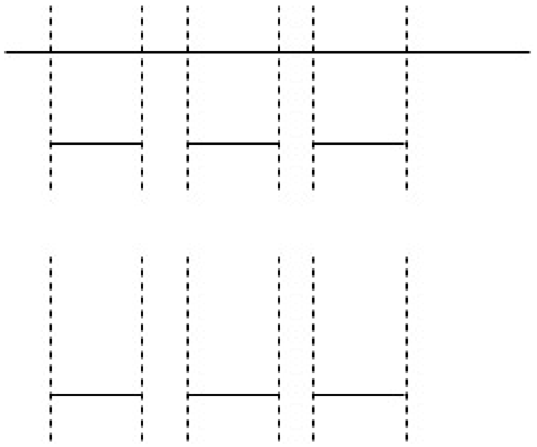
- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. ***(15 points)*** *Exercise 16.1-3*

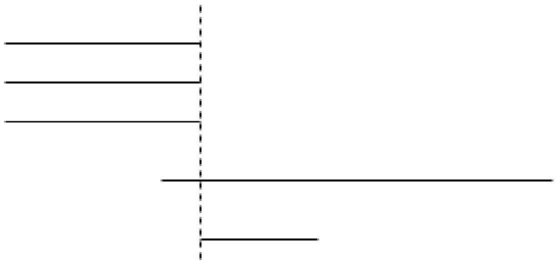
Solution:
The following example shows that selecting the least duration activity as the greedy criteria which does not always work. The least duration activity gives us two overlaps, while if we choose to ignore the least duration activity we get three activities.



Choosing earliest start time does not always give us the the optimal solution. As we can seen in the following example, the earliest start time activity gives us three overlaps while if we ignore the earliest start time activity and take the others we get a more optimal result.



Taking the following example, we either choose the one with two overlaps or one with 4 overlaps and the one with 3 overlaps. Hence minimum number of overlaps don't always work.



2. ***(15 points)*** *Exercise 16.2-3. Use induction to argue correctness.*

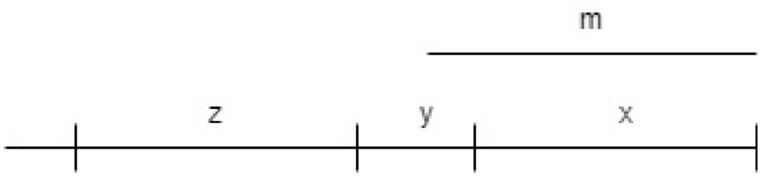
Solution:
Induction proof: The items are arranged in the decreasing order by their value and the increasing order by their weight. If we get an optimal solution on the kth element after adding it to the knapsack using the greedy algorithm. Then the k+1th term would also give an optimal solution if we follow the greedy algorithm.

```
1. Merge Sort the element based on their values and weights. According to the problem,
when we sort the values in the ascending order (increasing order), the values come in
the descending order. This will give us a time complexity of (nlogn).
2. We select the greedy criteria next. Here we select the max value/weight criteria to
get an optimum result.
3. Now make another array as the knapsack.
4. We start inserting elements with maximum value/weight in the knapsack. We keep adding
the elements until we reach the limit of the knapsack.
5. It takes O(n) time in the worst case to loop through the array for selecting which
element to insert in the knapsack.
6. The total time complexity we get in this algorithm is O(nlogn) + O(n).
```

3. ***(15 points)*** *Exercise 16.2-4.*

Solution:

```
1. In this problem, our greedy criteria is going the maximum distance without having
to refill our water.
2. We check at the beginning the distance between the start and the next stop. If this
distance is greater than 'm', that is the distance until we can retain the water, then
we decide to refill our water at the next stop,
3. If the distance is less than 'm' then we need to consider if we would be able to
sustain the water until a stop ahead of the next stop. Or we would run out of water
even if the distance is less than m, but the distance between the stop ahead of the next
stop should also be less than or equal to m.
4. Here we use the stay ahead strategy to prove that the greedy algorithm gives us an
optimal solution. If we can reach a stop ahead of the next
stop, that gives us the most optimal solution. We always try to find the maximum we can
go to find the optimal solution.
```

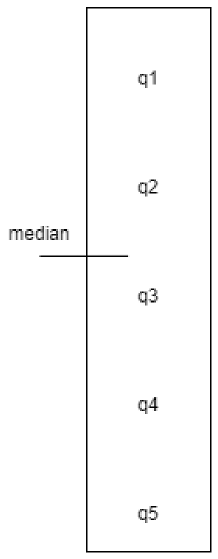


4. ***(15 points)*** *Exercise 16.2-5.*

Solution:
Since we have to take the smallest set with unit length interval, we take our greedy criteria as choosing the leftmost element. Then we select a set with a difference of a unit length after the left most element. This will be the first bucket. The remaining sub problem would be solved in the same fashion. We solve this using the stay ahead strategy where if we choose the starting point one element left to the leftmost digit, greedy would work as expected.

5. ***(15 points)*** *Exercise 16.2-6.*

Solution:
We assume that the elements are in a sorted order and are in a stack. Then we calculate the median of the elements and then sum up the elements less than the median. Let's call the sum as W and median as M. If we get W ¡ M, we will continue adding elements in the knapsack, until we reach its capacity. If we get W ¿ M, we take the elements less than the median and calculate its median. The time complexity in total that the algorithm takes is, T(n) = T(n/2) + O(n). We take only half the input in the knapsack and hence the time reduces by half. We take O(n) to find the median of the elements. Hence the time complexity required is O(n).



6. ***(15 points)*** *Exercise 16.3-3 (Extra Credit). First prove by induction that* $\sum_{i=0}^{n-1} F(i) = F(n+1) - 1$

Solution:

7. ***(20 points)*** *Problem 16-1, (a), (b) and (c).*

Solution:(a) Assume we have the following set of pennies, 1, 5, 10, 25. To find the most optimum solution, we consider our greedy criteria to choose the greatest value penny first, to minimize the number of pennies to be chosen.
If we take the example of getting a total of 25 pennies, according our greedy criteria, we choose the 25 - greatest valued penny. While if we do not use our greedy method, there is are many

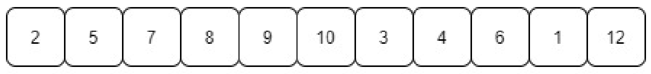
permutations by which we can get 25 pennies. (10+10+5) gives us 25, so does (10+5+5+5). The number of pennies used in each of these examples is more than the one we got using the greedy criteria, Using our greedy method we only required one penny, while with the other examples we required more than that.
This will be the same when we consider any example like we need a total of 30 pennies. According to greedy, we first choose 25, then 5. If we consider other permutations, that would give us (10+10+10) or (10+10+5+5). Hence, greedy gives us the most optimal solution.

(b) The number 1443 has denominations 1^3, 1^2, 1^1 and 1^0.
1443 = $((1^3)*1+(1^2)*4+(1^1)*4+(1^0)*3$ Our greedy choice here would be choosing the highest denomination to make the number. Hence we choose 1 of 1^3 to make up 1443. If we don't go with the greedy strategy and choose a denomination that is not the highest, we would have to overcompensate for the highest one with the one we choose. For example, we take 10^2 instead of 10^3 in this example, we would have to take 10 more of 10^2s than before to get 1443. In all we would take 14 10^2s. More generally, if we take $(c^3)*1$ in the greedy algorithm, we would have to take $(c^2)*c$ in the example we discussed.

(c) Let's take us a set of denominations = (1, 4, 5, 8, 9, 10) Using the greedy algorithm, we try to get as less coins as possible in every transaction. Hence our greedy criteria would be choosing the highest denomination coin, so that we would have to choose lesser coins of lower values.
We take an instance where we want to 17 pennies.
With our greedy algorithm, we would get an output of (10, 5, 1, 1). (Since we prefer choosing the penny with the highest denomination first in any problem.)
But this is a counter example for greedy algorithm, since we have a better solution of choosing just two pennies (9, 8) which would the optimal solution.

8. ***(15 points)*** *Exercise 15.4-5. Hint: try to solve this problem using a greedy approach -it may not work; if it doesn't work, it means you must use DP and you can leave it for HW5.*

Solution:
If we take the following example, then using the greedy criteria of taking a number just bigger than the least number, we get the subsequence = [2, 3, 4, 6, 12]. But we also get a counter example with a longer array, subsequence = [2, 5, 7, 8, 9, 10, 12] when we don't follow the greedy criteria of taking the next element just greater than the least element. Hence, this problem can not be solved using the greedy algorithm.



Greedy = 2, 3, 4, 6, 12

Optimum solution = 2, 5, 7, 8, 9, 10, 12