Name: Mauro Vargas Jr

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 3B (45 points) **Summer 2020, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed**, we cannot accept hand-written solutions. Here's a short intro to **Latex**.
- In this homework we denote the asymptomatic *Big-O* notation by $\mathcal{O}$ and *Small-O* notation is represented as o.
- We recommend using online Latex editor **Overleaf**. Download the **.tex** file from Canvas and upload it on overleaf to edit.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Canvas if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name: Mauro Vargas Jr
ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 3B (45 points) **Summer 2020, CU-Boulder**

Piazza threads for hints and further discussion

Piazza Threads
Question 1
Question 2

Recommended reading
Greedy Algorithms: Ch. 16 16.1, 16.2, 16.3; Ch. 2 2.1, 2.2

Name: Mauro Vargas Jr

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 3B (45 points) **Summer 2020, CU-Boulder**

---

1. ($7.5 \times 2 = 15$ *pts*) *In this question we will look at the* ***Interval scheduling*** *problem. The problem consists of a set of tasks. Each of these tasks need to be executed in a specific time interval. Two tasks are set to be compatible if their time intervals do not overlap. The goal is to find the maximum set of compatible tasks so that as many tasks possible can be executed. For example consider the case of just three tasks*

Task	start time	end time
Task 1	1	10
Task 2	2	5
Task 3	7	9

Here the set consisting of Task 2 and Task 3 is the set consisting of maximum number of compatible tasks. It is maximum because if we select Task 1, then we cannot select either Task 2 or Task 3 as both of them have overlapping intervals with Task 1.

   (a) *Consider a greedy algorithm which always selects the shortest appointment first. Provide an example in tabular form with at least 5 tasks where this algorithm fails. List the order in which the algorithm selects the intervals, and also write down a larger subset of non-overlapping intervals than the subset output by the greedy algorithm.*

   Note

   A comment on level of justification to help us understand your thinking, it is worth writing a little about the order in which the algorithm selects the tasks. For example, The algorithm consider tasks in the order [Task 3, Task 2, Task 1]: first the algorithm selects Task 3 because that is the shortest. Task 1 conflicts Task3, so it is removed. In the next step the shortest and only remaining Task 2 is selected.

Name: Mauro Vargas Jr
ID: 107212593

Task	start time	end time	difference
Task 1	5	7	-2
Task 2	0	6	-6
Task 3	6	9	-3
Task 4	9	13	-4
Task 5	10	11	-1

Since the Greedy algorithm always selects the shortest appointment first Task 5 will be added to our schedule array [Task 5]. Next it will check for the next shortest appointment which is Task 1 and add it to the schedule array [Task 5, Task 1]. The algorithm will then check for the next shortest Task 2, but it fails since it conflicts with Task 1 and looking at the table we can also see that Task 3 and 4 will also conflict with Task 1 and 5. Therefore that algorithm will return the schedule array [Task 5, Task 1]. Although the Greedy algorithm did produce a correct answer, it did not produce the most optimal solution. Because we can produce another array that hold more Task that do not overlap optimal array [Task 2, Task 3, Task 4].

(b) *Consider a greedy algorithm which always selects the longest task first. Provide an example in tabular form with at least 5 tasks where this algorithm fails. Show the order in which the algorithm selects the intervals, and also show a larger subset of non-overlapping intervals than the subset output by the greedy algorithm.*
*The **Note** provided in 2a applies here in terms of the level of explanation.*

Task	start time	end time	difference
Task 1	5	7	-2
Task 2	0	6	-6
Task 3	7	10	-3
Task 4	9	13	-4
Task 5	10	11	-1

Since the algorithm always selects the longest task we add [Task 2] first. Next the algorithm will pick the next longest task which is Task 4 and add it to the schedule array [Task 2, Task 4]. The algorithm will fail to add the next longest task because Task 4 conflicts with Task 3 and looking at the table we can see that Task 1 and Task 5 will fail to add because it also conflict with Task 2 and Task 4. We can see that this schedule array [Task 2, Task 4] is not the optimal solution since we can create another array that has more non overlapping Task optimal array [Task 2, Task 3, Task 5].

Name: Mauro Vargas Jr
ID: 107212593

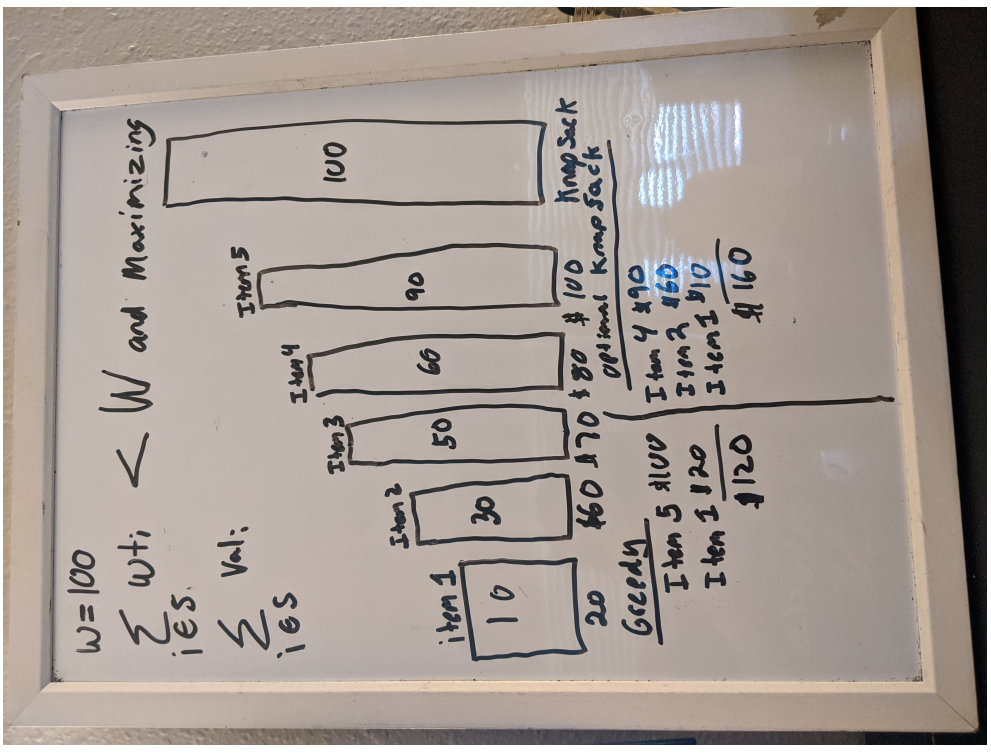
2. $(10 \times 3 = 30)$ *In this question we'll consider weighted problems.*

   (a) *Consider the weighted interval scheduling problem. In this problem, the input is a list of n tasks and weights, each of which is specified by* $(start_i; end_i; w_i)$*. The goal is now to find a subset of the given intervals in which no two overlap and to maximize the sum of the weights, rather than the total number of intervals in your subset. That is, if your list has length n, the goal is to find* $S \subseteq \{1, 2, ...n\}$ *such that for any* $i, j \in S$*, interval i and interval j do not overlap, and maximizing* $\sum_{i \in S} w_i t_i$*. Consider the greedy algorithm for interval scheduling from recitation, which selects the job with the earliest end time first. Give an example of weighted interval scheduling with at least 5 intervals where this greedy algorithm fails. Show the order in which the algorithm selects the intervals, and also show a higher-weight subset of non-overlapping intervals than the subset output by the greedy algorithm. The* ***Note*** *provided in 2a applies here in terms of the level of explanation.*

Task	start time	end time	difference	w_i
Task 1	5	7	-2	2
Task 2	0	6	-6	4
Task 3	6	9	-3	7
Task 4	9	13	-4	9
Task 5	10	11	-1	3

Using the same table from problem 1(a) and the Greedy algorithm. The program will select the shortest appointment first Task 5 and adds it to our schedule array [Task 5]. The program also sets the **MAX** value variable to 3 to keep track of the total amount weight for every task in our array. Next it will check for the next shortest appointment which is Task 1 and adds it to the schedule array [Task 5, Task 1] and computes the new **MAX** value of 5. The algorithm will then check for the next shortest task Task 2, but it fails since it conflicts with Task 1 and looking at the table we can see that Task 3 and 4 will also conflict with Task 1 and 5. Therefore that algorithm will return the schedule array [Task 5, Task 1] and have a **MAX** value variable weight of 5. The Greedy algorithm again produces the correct answer but it is not produce the most optimal solution because we can produce another array that hold more tasks that do not overlap with a optimal array of [Task 2, Task 3, Task 4] and with a **MAX** value of 20. The Greedy algorithm also does not ensure us that we will have a optimal **MAX** value for our schedule array.

(b) *Consider the Knapsack problem: the input is a list of n items, each with a value and weight $(val_i; wt_i)$, and a threshold weight W. All values and weights are strictly positive. The goal is to select a subset S of the items such that $\sum_{i \in S} w_i t_i < W$ and maximizing $\sum_{i \in S} val_i$. (Note that, unlike change-making, here there is only one of each item, whereas in change-making you in principle have an unlimited number of each type of coin.) Consider a greedy algorithm for this problem which makes greedy choices as follows: among the remaining items, choose the item of maximum value such that it will not make the total weight exceed the threshold W. Give an example of knapsack with at least 5 items where this greedy algorithm fails. Show the order in which the algorithm selects the items, and also show a higher-value subset whose weight does not exceed the threshold. The **Note** provided in 2a applies here in terms of the level of explanation.*

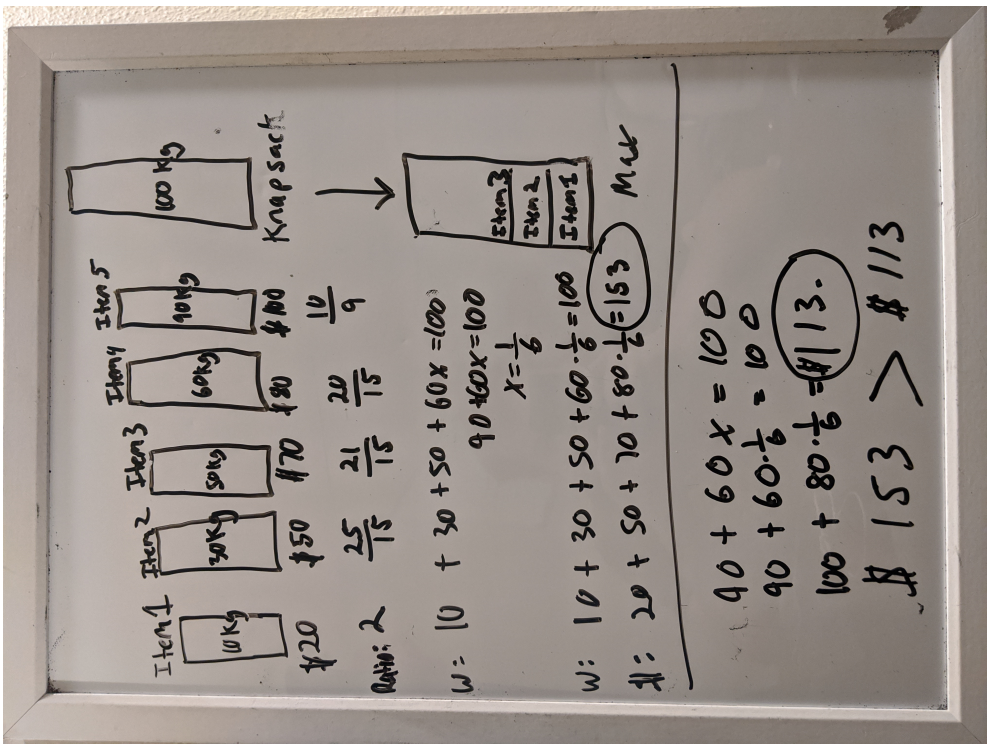


Since were using Greedy algorithm with weights and values the program is going to select all the Items that have the highest value first without going over the knapsack limit. You can see from the picture that Item 5 and 1 will be added with out going over the limit. but you can see to the right that there is a solution that can fit Item 1,2,4 with a value of 160 showing the that program fails to come up with the most optimal solution.

Name: Mauro Vargas Jr
ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 3B (45 points) **Summer 2020, CU-Boulder**

(c) *Now consider the following algorithm for the knapsack problem. Call the relative value of item i the ratio* $\frac{val_i}{wt_i}$*. Consider the greedy algorithm which, among the remaining items, chooses the item of maximum relative value such that it will not make the total weight exceed the threshold* W*. Give an example of knapsack where this greedy algorithm fails. Show the order in which the algorithm selects the items, and also show a higher-value subset whose weight does not exceed the threshold. The* ***Note*** *provided in 2a applies here in terms of the level of explanation.*



Using the Greedy algorithm and fractional knapsack method the program takes the items in order from greatest value per kg and yields a optimal solution. From the picture we add Items 1, 2, 3, and 1/6th of Items 4's weight and value to the program and it comes up with a max value of 153 dollars. To test this I calculated the value for Item 5 and a 1/6th fraction of Item 4 and still came up with a smaller max value concluding that when we are allowed to break Items we can come up with a greedy algorithm that always comes out to be a optimal solution.

Name: Mauro Vargas Jr
ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 3B (45 points) **Summer 2020, CU-Boulder**

3. ***Extra Credit (5% of total homework grade)*** *For this extra credit question, please refer the leetcode link provided below or click here. Multiple solutions exist to this question ranging from brute force to the most optimal one. Points will be provided based on Time and Space Complexities relative to that of the most optimal solution.*

   Please provide your solution with proper comments which carries points as well.

   https://leetcode.com/problems/minimum-cost-for-tickets/

   ```
   Replace this text with your source code inside of the .tex document
   ```