Assignment Notes and Summary
: You have n activities that you would like to schedule in a day. These activities have all been preassigned with a profit value, which you shall justify, and they need to be organized in an optimal scheduler

My codes in :
https://github.com/Jeongwoo-KGI/Optimal-Task-Schedule-Generator

● You will need to provide a short explanation about what it means to have an optimal scheduler. Perhaps if you are planning several activities across town, you would like to save money on fuel/bus tickets, or perhaps you would like to spend the least amount of time doing a number of tasks in a single day.

- What I consider to be an optimal scheduler is that I finish what are important in the day at least. I can't work for a full day and nobody can. However, we all have plans and things that we need to finish and what we want to finish. By getting to do at least the ones that are important, we can say that we've used the day quite wisely that may not be the best perhaps. Here, I am saying that a scheduler that tells me what are the most import to the least and laying them as a timeline to show that the early ones are the most important and the least goes at the end of the day, I will be able to at least start with and hopefully finish the ones that are the most important.

- However, here the importance may have bias. So, to mitigate some bias(#biasmitigation), the code calculates the importance for the users. This is calculated by the left over days before the deadline and the importance in our life that is somewhat cut into different categories assigned by numbers mentioned in the notes of class todo.

Review your LBA Part I assignment. Using the feedback you have received then as well as everything you have learned during this semester, perform a critical analysis of that algorithm. And compare it with the new code
[word count 397]
- The previous assignment was done in OOP with heapsort. However, as CS110 class came to an end, I see that there can be more effective methods to be appended to the previous assignment for better convenience on the user part. So, what I've done in the code was to calculate the 'value' that I will be sorting the works by heap sort for the having the least time complexity, as well as storing the sorted existing list of work to do that allows me to add a new task that user just noticed and return the new list in a fast time in expense of memory.
- The assignment 2's time complexity:
    1. Heap sort : O(nlogn)
    2. Class 'todo' = constant O(1)
    3. Method sorting = 4(n + heapsort + len(duration)*len(dur) + len(duration)) + 4(n) = O(n^2)
    4. Method printout = O(n)

        5. Class make_a_todo_list = O(n^2)
- The new time complexity:
    1. Heap sort and class todo has no difference
    2. Method make_list (initial todo list)= method sorting + method printout = O(n^2 + n) = O(n^2)
    3. Method new_list (adding a new value) = O(n)
    4. Method sorting = O(method cluster + len(sequence)*2 = dependence*n^2+2n) = O(n^2)
    5. Method cluster = O(dependence*n^2) = O(n^2).
- The previous work considered the time when I woke up and started laying out the whole task into certain time. However, as I looked carefully at my own life, sometimes work do not start and finish as we've planned. There are so many variables in life that may make our 'plans' not work out perfectly, and we cannot blame that variable as that can be also another important factor in daily life as a human being. So, I've considered making the code bit simple by just listing out the tasks in order by the importance of the task calculated as a 'value' automatically, and thus the codes will output the whole list of tasks in descending order of 'value'. Thus we know what are the most important thing to do in the daily life and what is less important. This allows us to choose and focus on the most important work and feel less being chased by work when the plan does not work out as expected. Past code did not support this idea and outputs the time by hours with decimals. So, in this new assignment, this feature has been deleted when the list is printed out again with a new task to do during the day.

Combining dynamic programming and greedy techniques with your critical evaluation from question 1, propose at least two improved algorithmic strategies to solve the scheduling problem with multitasking activities: one should implement a dynamic programming strategy, whereas the other should implement a greedy one. Note that for this question, every task has the following information:

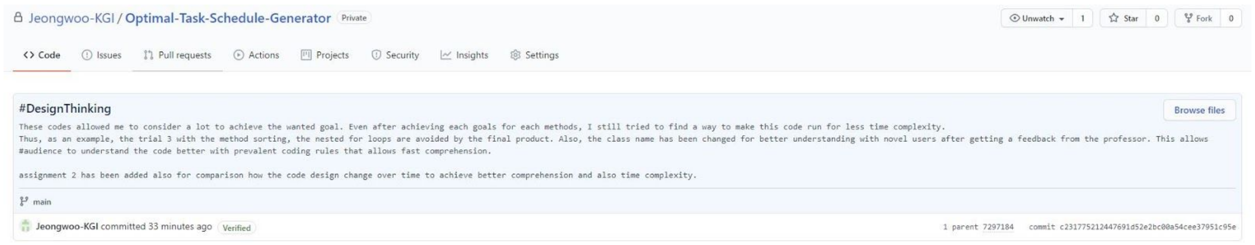
a. Design an optimal scheduler and write the new algorithmic strategies in pseudocode. Make sure to carefully explain any assumptions you make, including what data structures are best suited for each of these strategies.

- The optimal scheduler here is to show a list in order what task is the most important thing to finish to the least important task in my life. Importance here considers the d-day, importance in my life as it can be small for a day but can be magnificent after a year or two. All of them are counted inside the code by allocating each work to have specific importance from range 0 to 4 and higher importance will mean to have a higher number.
- The 'value' inside the code is represented by a float that considers the urgency(left over days to finish or submit) and the importance. The equation for this is importance*10/(urgency+1) because the urgency rate can be 0 to mean that there are 0 days left for submission. This will be used for a greedy algorithmic approach to sort the data to achieve the optimal sequence of work to do for fulfilling the most out of a day.

- The new algorithmic strategy is used here by using cluster method and shortened method ‘sorting’. The former code used to have a heapsort for 4 times on different categories and another O(n) was applied to make a full list to print out which is in total time complexity of 4nlogn+n = O(n). But now, we only need to use heapsort(Once or non) to build a fast list of things to do for a day.
- The cluster method uses queue data structure to search through all tasks to cluster the whole dependent tasks together. However, all this effort to shorten the time complexity sheds light on the topological sorting as this will allow the dependence to be in a list form and it will have O(u+v) instead of O(dependencies*n^2).

● List all the LOs and HCs you have exercised while working on this final assignment as well as a thorough justification of their application. Review your previous assignments and in-class grades, and reflect on your application of these LOS has evolved over time -> are mentioned in comments to mention below

#bias mitigation from question 1.
+
Comments to mention (also in the git repository like the below image)



#DesignThinking
```
These codes allowed me to consider a lot to achieve the wanted goal. Even after achieving each goals for each methods, I still tried to find a way to make this code run for less time complexity.
Thus, as an example, the trial 3 with the method sorting, the nested for loops are avoided by the final product. Also, the class name has been changed for better understanding with novel users after getting a feedback from the professor. This allows
#audience to understand the code better with prevalent coding rules that allows fast comprehension.

assignment 2 has been added also for comparison how the code design change over time to achieve better comprehension and also time complexity.
```
Jeongwoo-KGI committed 33 minutes ago Verified
1 parent 7297184 commit c231775212447691d52e2bc00a54cee37951c95e

```
- These codes(trials) allowed me to consider a lot to achieve the
  wanted goal. Even after achieving each goals for each methods, I
  still tried to find a way to make this code run for less time
  complexity.
  Thus, as an example, the trial 3 with the method sorting, the
  nested for loops are avoided by the final product. Also, the class
  name has been changed for better understanding with novel users
  after getting a feedback from the professor. This allows #audience
  to understand the code better with prevalent coding rules that
  allows fast comprehension.
- assignment 2 has been added also for comparison how the code design
  change over time to achieve better comprehension and also time
  complexity.
- This code schedules task according to the importance and
  d-day as well as considering the dependence.
- This is a code part without topological sorting applied,
  but to have less time complexity, I've changed the
  dependence from a list input to int input to consider only
  1 task before it should be allocated. (#heuristics)
```

- After some fixing and trying diverse trials and versions, I found this is the most effective.(only considering non-topological sorting)
- The heapsort is O(nlogn) time complexity, however, by having to cluster the activities considering dependencies and to generate a new value for such cluster, the O(n^2) time complexity was additionally taken by method 'cluster'. Thus, this whole code, when it runs for the first place it gets to have O(n^2) time complexity. However, this code is different from the ones from the start of the semester. Because it allows adding a new activity inside the existing list of tasks. This only takes O(n) time complexity in expense of memory O(n) for keeping the ready-made list after carefully considering the way to shorten the complexity from using nested for loop(or nested while loop) to a single for loop with if statements. As this is a code before getting to know the topological sorting, I've used already existing knowledge of simple coding and from the class(using queue structure) to achieve the goal of getting a result without an error. (#heuristics: availability)
- Especially, considering the cluster was a technical challenge that I've had to face to build a greedy-optimal scheduler to have dependent activities be aligned with each other closely as well as having to sort them by this cluster. (#gap analysis: knowing what I want to build and trying to find a break through from existing code from the start of the semester.)

- (#cognitive persuasion by #evidencebased)
- Time complexity has been calculated by :
- for loops/while loops as (n) or some what similar to this, it gets multiplied by other time complexities inside the loop. So, nested for loop with iterating n times each will be O(n^2). Without break, it will be big-Theta(n^2) in this example.
- if statement and other single calculations are considered as constant.
- These analysis was kept in mind as the versions upgraded to achieve the efficiency as well as achieving the wanted goal. So the newly added feature method 'new_list' was considered in expense of space(memory) the time complexity has decreased from O(n^2) to O(n). As a daily tasks are usually limited in daily life for individual human, it is

unlikely that people reach the number of n that will take millions of years to run, however, we don't want the code to run more than a minute especially during the day when we are in middle of activity and just appending another task to do. We need this result to be quick to see what to do next. So, it is preferred to have smaller time complexity. Also as the computer is developing in hardware, the memory size has increased tremendously over time. Thus, having to make some memory use is ok for future use to decrease time complexity of the codes.