Q1- Setting up:

A . The shared code and the text file are included in the zip file

B.

I added ‘wake-up’ to start the day and then ‘have my breakfast’ which is a usual daily tasks. Then dress-up to get ready and start exploring. I added the tasks ‘going to Myeondong’, ‘study at cafe there’, and ‘try korean food’ as cultural experience tasks to explore the neighbourhood and trying the cultural food and drinks in every new area. Also, I added boba becuase I am at Asian city and I like to have it from one day to another. Then, taking the class after doing the pre-class work at any cafe and come back to the res hall.

Table(1) shows the tasks of the day

Id	Description	Duration	Dependencies	Profit	Happiness	Healthiness
0	Have breakfast	30	[1]	0	Happy	Healthy
1	Wake-up	30	[]	0	Neutral	Neutral
2	Dress up to go out	15	[0]	0	Neutral	Neutral
3	Go back to the res hall	15	[7]	-2	Neutral	Neutral
4	Try Korean food at Myeondong	60	[8]	-15	Very happy	Healthy
5	Get boba	30	[]	-6	Very happy	Neutral
6	Study at cafe	120	[8]	0	Happy	Neutral
7	Take the class	90	[6]	0	Neutral	Neutral
8	Take the bus to Myeondong	15	[2]	-2	Neutral	Neutral

Q2- Running the program:

A. I anticipate some issues with the algorithmic strategy of the code:

The algorithmic strategy is based on the priority that is calculated according to the satisfaction, duration, and dependencies as in the utility function. Since the length of the dependencies are the same in all the tasks, I guess it is going to have a problem with the tasks that have happiness (neutral), healthiness (neutral), and 0 profit. As the satisfaction isn't going to affect the priority calculation and the shorter the task, the more important it is going to be executed first which isn't realistic to prioritize the tasks based on the duration.

The modification in the get-tasks-ready method to include all tasks, even those with dependencies, introduces a significant issue. This change could result in tasks that are dependent on the completion of other tasks being scheduled and executed before their prerequisite tasks. This sequence would disrupt the logical flow of task execution, as tasks that should logically

follow others might be prematurely addressed, leading to potential inefficiencies or the inability to complete certain tasks as scheduled.

B.

```
Running a daily scheduler:

🕛t=11h00
        started 'get boba' for 30 mins...
        ✅ Task completed at 11h30 with priority=0.6055300708194983
🕛t=11h30
        started 'wake-up' for 30 mins...
        ✅ Task completed at 12h00 with priority=0.18257418583505536
🕛t=12h00
        started 'have breakfast' for 30 mins...
        ✅ Task completed at 12h30 with priority=0.8366600265340756
🕛t=12h30
        started 'try korean food at Myeondong' for 60 mins...
        ✅ Task completed at 13h30 with priority=-2.2949219304078006
🕛t=13h30
        started 'dress up' for 15 mins...
        ✅ Task completed at 13h45 with priority=0.2581988897471611
🕛t=13h45
        started 'study at cafe' for 120 mins...
        ✅ Task completed at 15h45 with priority=-2.575525836277581
🕛t=15h45
        started 'take the bus to Myeondong' for 15 mins...
        ✅ Task completed at 16h00 with priority=-1.0327955589886444
🕛t=16h00
        started 'Take the class' for 90 mins...
        ✅ Task completed at 17h30 with priority=0.10540925533894598
🕛t=17h30
        started 'go back to the res hall' for 15 mins...
        ✅ Task completed at 17h45 with priority=-1.0327955589886444

🏁 Completed all planned tasks in 6h45min!

🤦‍♂️ Your scheduler has total utility of -4.9476664563879345 utils
```

Figure(1) shows the output of running the scheduler)

A. Is the schedule that it returns feasible?

No, the schedule isn't feasible as it starts with 'getting boba' before even 'wake up'. Also, it executes 'try korean food' which is supposed to be at Myeondong before 'take the bus to Myeondong'. Also, it executes trying the food before dressing up. Also, it gives 'take the bus to Myeondong' after 'study at cafe' and it was supposed to be after going to Myeondong to explore the cafes there. Also, 'take the class' after 'go to Myeondong' isn't true since taking the class depends on studying at cafe that is in Myeondong which means after going to Myeondong. The only tasks that is in the correct order or feasible is 'have breakfast' as it ends on waking up and 'go back to the res hall' after taking the class as it depends on it.

B. Do you envisage running into trouble while test-driving the instructions you are getting?

Yes, because I can't get boba before waking up. Also, I can't try the korean food after having the breakfast without going to Myeondong because the restaurants are there. Also, I can't dress up and get ready to go out after trying the korean food. Also, messing up with going to the cafe to study and then go to Myeondong to take the class is a trouble since I am supposed to study at cafe in Myeondong so I can't go study before going to Myeondong neither taking the class after going to Myeondong directly.

C. Confirm that the priority values are calculated meaningfully and correctly.

The prioirty value (utility function)is calculated based on the satisfaction, duration, and number of dependencies. The satisfaction is based on the profit value, healthiness, and happiness scales. I think the profit scale shouldn't be the same as healthiness and happiness because it has more cost and more important in the real life. For example, two hours of boring work-study gives you 40USD and being very happy to get boba aren't the same for an international student so maybe increasing the scale of profit over the happiness and healthiness level is more meaningful. Then, the prioirty value depends on the duration of the task, satisfaction, and length of dependencies is correct. Then, the final value based on the loss aversion is clear and thoughtful but if two tasks have the same number of dependencies and same satisfaction value, it is just going to depend on the duration. The longer the task value, the less important according to the final utility value which means if a task has longer time, it is going to be executed late which isn't helpful in the real life.

To make sure the calculation is correct, I am going to calculate the utility of 'wake-up' task:

Dependency = 0

Healthiness= 0 (neutral)

Happiness = 0 (neutral)

Profit = 0

The satisfaction = 0

Duration = 30

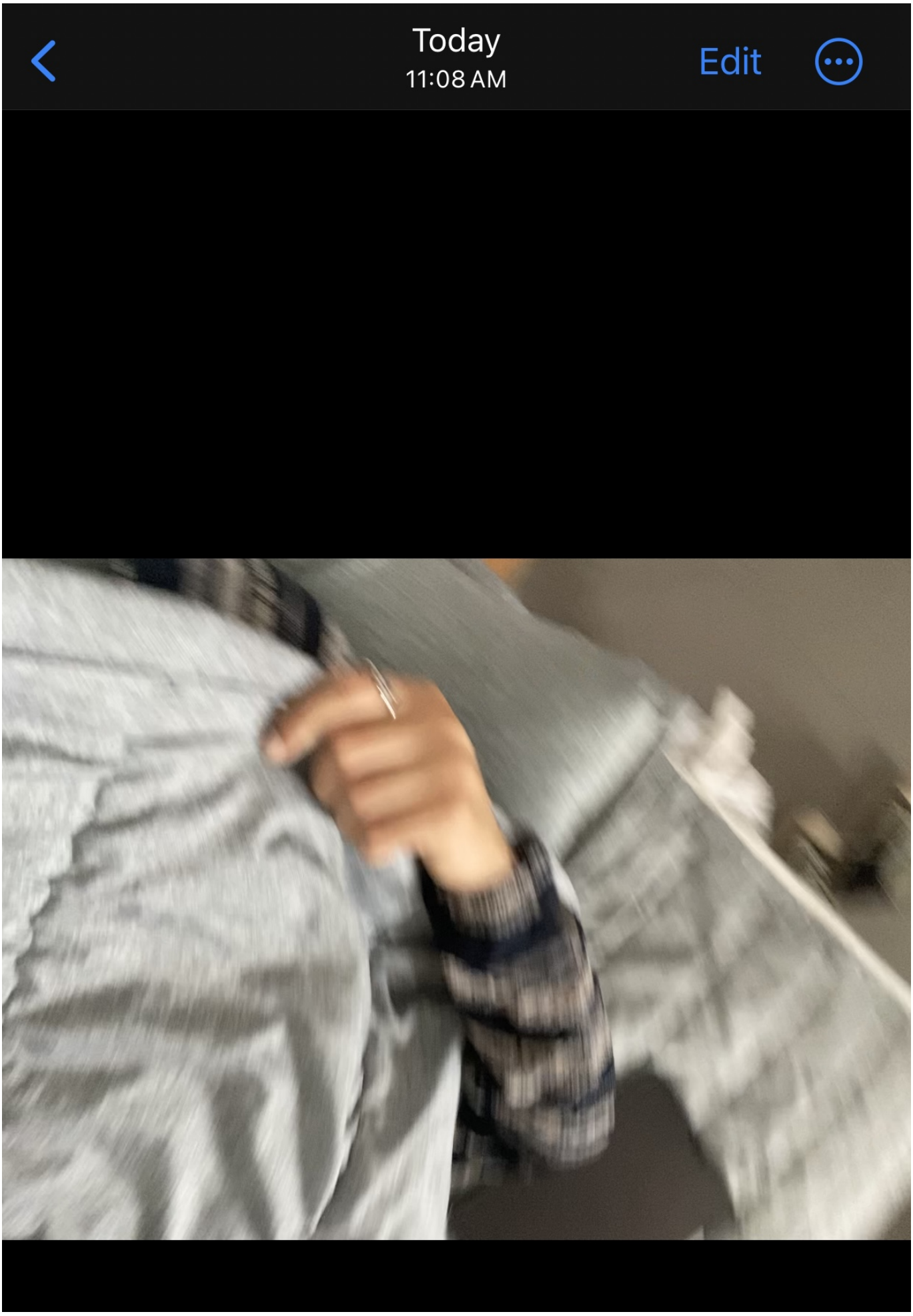
Then the function is going to be = satisfaction + 1/duration - 2(dependencies)

```
= 0 +1/30 + 0 = 1/30. --> +ve value
utility = sqrt(1/30)= 0.18257
```

comparing with the code, the value is the same which means the the priority value is calculated correctly.

My scheduler isn't restricted with fixed time

First, waking-up


Today
11:08 AM
Edit

Having breakfast

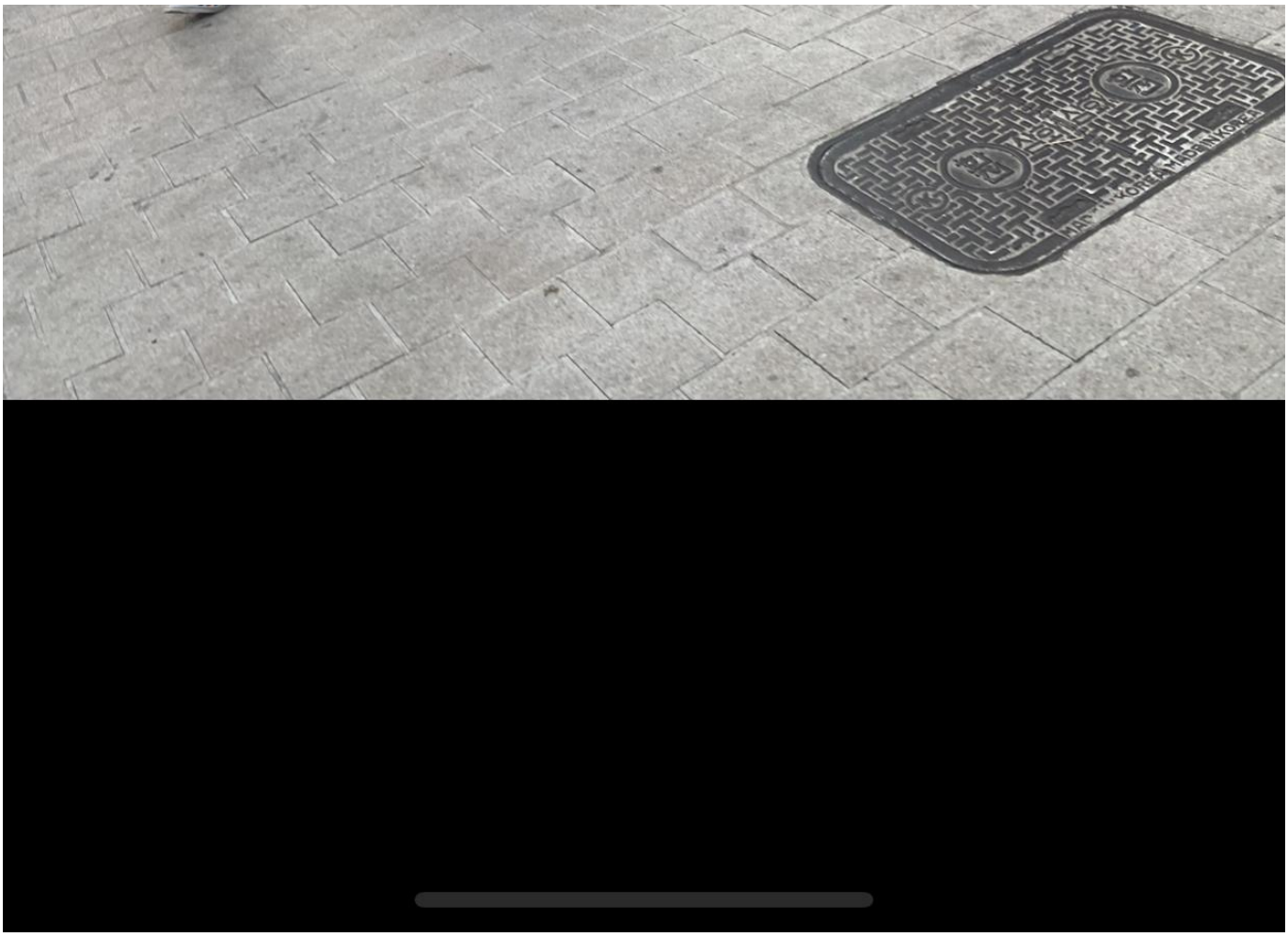

Today
11:36 AM
Edit

Dress up


9:10
Today
1:31 PM
Edit

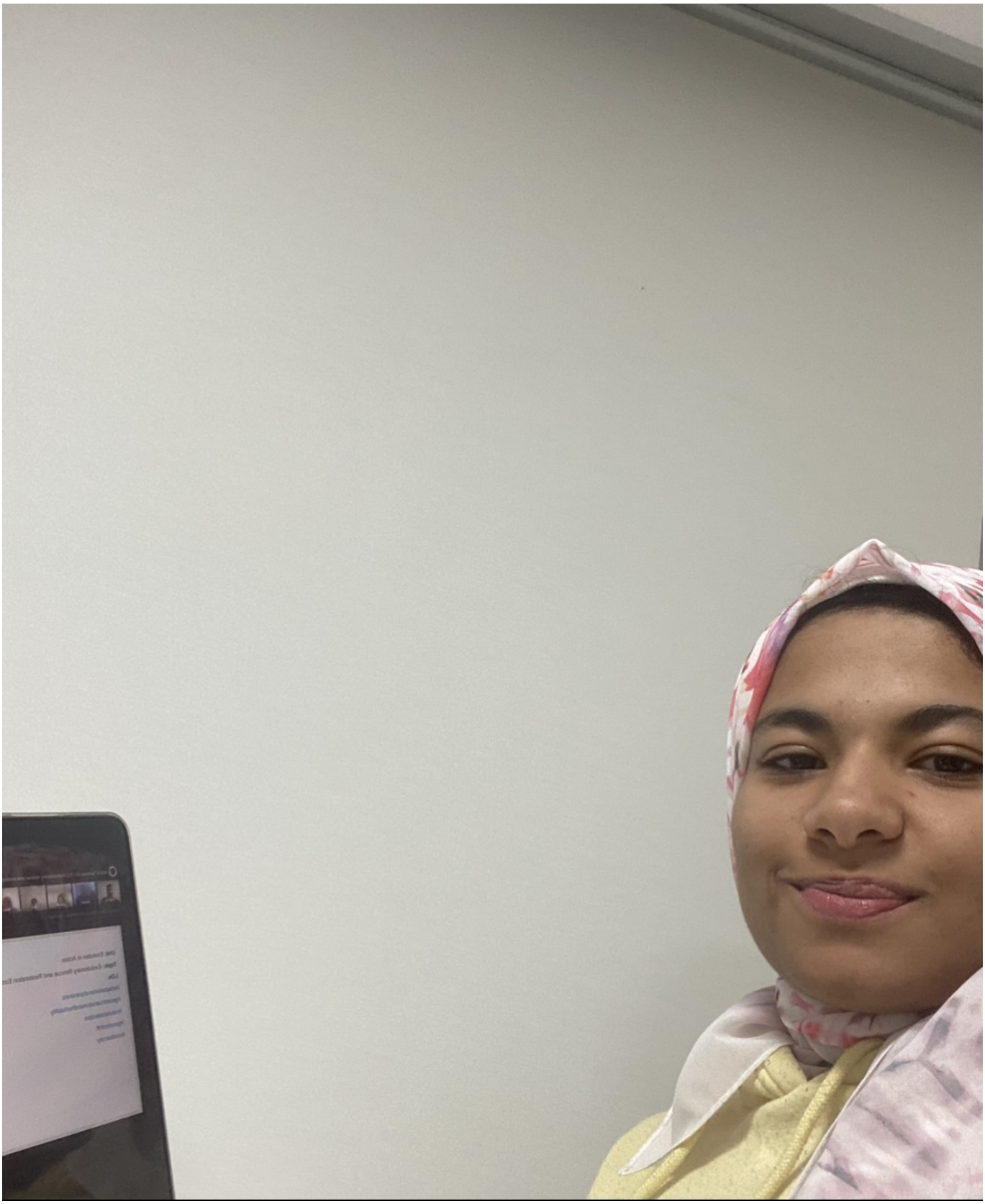
Going to Myeondong


TONYMOLY
LINE
LINE
Skin care
Body care
TOPBROWS
TONYMOLY
LENS TOWN

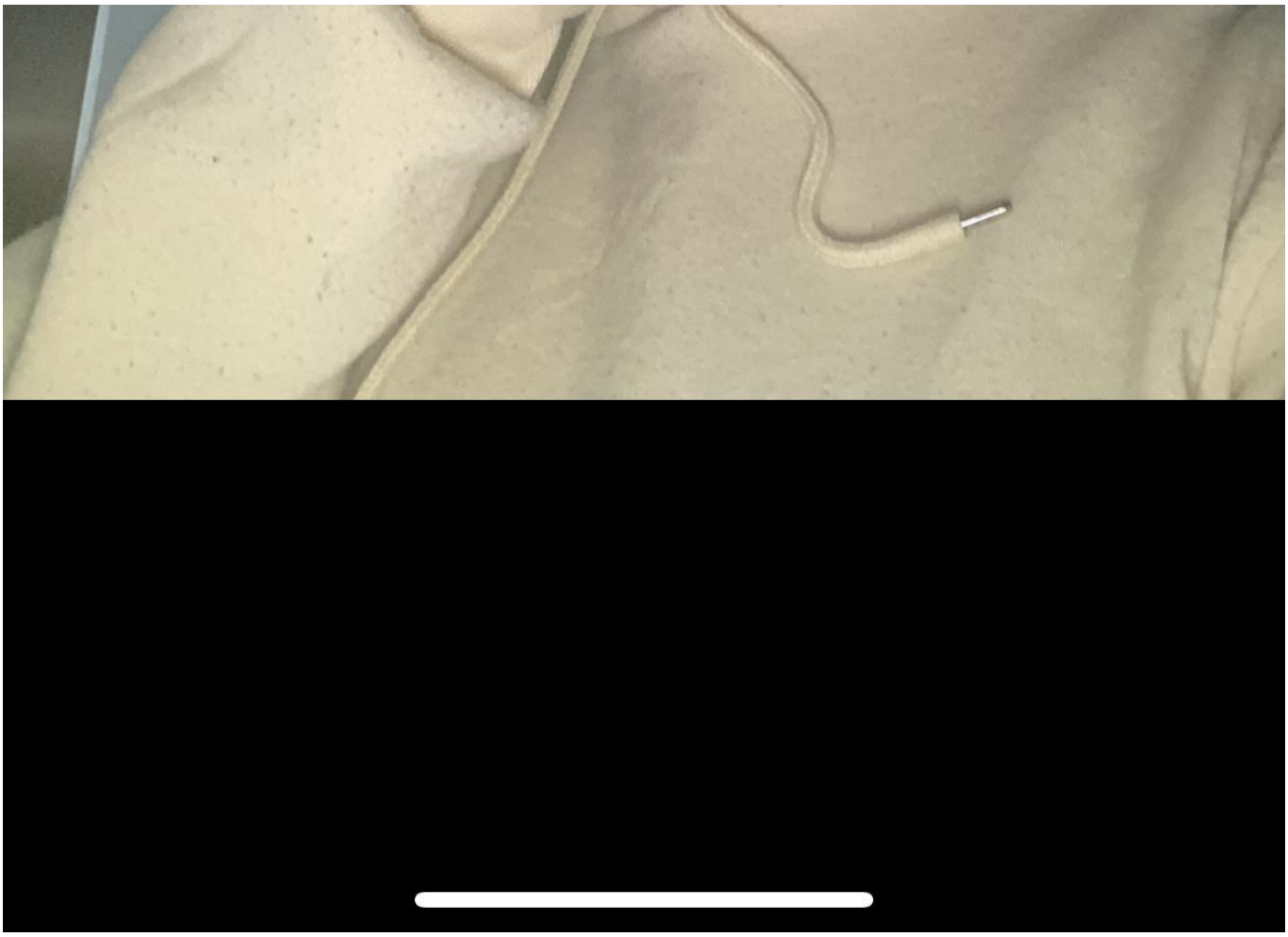


I forget to take a selfie because the area was amazinggg

Take the class



Go back to the res hall



Q4–Analysis:

The benefits of using this scheduler:

This algorithms is good in general for flexible tasks which could encourage the users to engage in activities that are mental and physical well-being. This scheduler is going to be perfect if the tasks are all about hobbies and free day to do which is going to guarantee the happiest and healthiest options in short time. Also, it is going to help people releasing the stress of having too many tasks by starting with the ones with shorter duration and more happiness over long boring tasks which could give the users courage and energy to go through all the other tasks. For example,'going for a walk' is happy option for the user and takes 15 mins vs 'cook' which is unhappy option and takes longer time. There is no specific time to cook as it is flexible so the scheduler gives the user to go for the walk first (relaxing and gaining energy) and then cook where you can see in appendix part II. It is based on the loss aversion where the pain of losing is greater than the utility of an equivalent gain. It can lead to a schedule that minimizes potential negative outcomes,

increasing overall satisfaction with task completion (based on the happiness, healthness, and profit values).

```
Running a daily scheduler:

🕛t=14h00
        started 'go for a walk' for 15 mins...
        ✅ Task completed at 14h15 with priority=0.6324555320336759
🕛t=14h15
        started 'cook' for 30 mins...
        ✅ Task completed at 14h45 with priority=-1.0954451150103321

🏁 Completed all planned tasks in 0h45min!

👨‍🎨 Your scheduler has total utility of -0.46298958297665627 utils
```

The limitations and failure:

- In get-tasks-ready, all the tasks with dependencies are pushed into the prioirty queue which means that there are some tasks that are executed before their prerequisite as you can see in my scheduler that although 'try korean food' depends on going to Myeondong, it has been executed before going to Myeondong which is practically impossible .This leads to a significant flaw: tasks are executed out of the required logical sequence. I wasn't able to get the boba before waking up because it is impossible, too.

- Also, this code can't go with tasks that have fixed time. It doesn't have a second heap to sort the tasks with fixed time so they aren't going to run on time. Also, there is no attribute for the fixed time.

- This scheduler had the scale of very unhealthy to be 2 which is the same value as very healthy which means that the tasks with 'very unhealthy' and 'very healthy' values are going to have the same value for the happiness and going to mess the schedule if all the other attributes have the same utility value and are going to be organized at the order of the given tasks as you can see in fig(2) and fig(3).

```
Running a daily scheduler:

🕛t=14h00
        started 'A' for 15 mins...
        ✅ Task completed at 14h15 with priority=0.8563488385776752
🕛t=14h15
        started 'B' for 15 mins...
        ✅ Task completed at 14h30 with priority=0.8563488385776752

🏁 Completed all planned tasks in 0h30min!

👨‍🎨 Your scheduler has total utility of 1.7126976771553504 utils
```

Fig(2) shows the output of two tasks have same attributes except the healthiness scales. A is very healthy and B is very unhealthy in the order of [B, A]

```
Running a daily scheduler:

🕛t=14h00
        started 'B' for 15 mins...
        ✅ Task completed at 14h15 with priority=0.8563488385776752
🕛t=14h15
        started 'A' for 15 mins...
        ✅ Task completed at 14h30 with priority=0.8563488385776752

🏁 Completed all planned tasks in 0h30min!

👨‍🎨 Your scheduler has total utility of 1.7126976771553504 utils
```

Fig(3) shows the output of two tasks have same attributes except the healthiness scales. A is very healthy and B is very unhealthy in the order of [A, B]

- Giving the profit the same scale as the happiness and healthiness isn't reliable in the real-life. Regardless of the healthiness and dependencies, If you have two tasks : work-study that is going to give you a profit of 40USD but is very sad and takes 2hrs vs watch movie which is going to make you very happy, duration of 2hrs as well and the profit is zero. Now, since the -2 value of very unhappy is going to cancel the 2 of the profit of 40USD (since all the profits above 20 takes a value of 2), the scale of 2 for very happy in 'watch movie' is going to be higher and the scheduler is going to execute 'watch movie' first as you can see in fig(4).

```
Running a daily scheduler:

🕛t=14h00
        started 'watch movie' for 120 mins...
        ✅ Task completed at 16h00 with priority=0.8215838362577491
🕛t=16h00
        started 'work-study' for 120 mins...
        ✅ Task completed at 18h00 with priority=0.09128709291752768

🏁 Completed all planned tasks in 4h00min!

👨‍🎨 Your scheduler has total utility of 0.9128709291752768 utils
```

Fig(4) shows the output of two tasks: 'watch movie' and 'work-study'

I moved from the task of 'getting boba' to the subsequent task because it wasn't possible to get boba before waking up. Similarly, trying Korean food in Myeongdong wasn't feasible before actually arriving in Myeongdong, nor was dressing up to go out. Additionally, starting work at a cafe in Myeongdong wasn't an option since I hadn't yet arrived there. Following the schedule, I woke up, had breakfast, and skipped trying the Korean food, moving directly to getting dressed to go out. I also bypassed the 'study at cafe' task, took the bus to Myeongdong, attended my class there, and then returned to the residence hall.

Basically, the output of my scheduler gave all the tasks in the table without skipping any tasks but the tasks that are actually feasible and I was able to finish it is just 6 tasks/8 total tasks which is 3/4 --> 75%. This isn't a bad efficiency. This is bad for me since I missed most of the cultural experience tasks (trying korean food and going to cafe). Also, I missed studying before the class which had bad consequences on me in the class that I didn't fully understand the material due the absence of preparing. If the tasks had similar importance, 75% isn't a bad start for the scheduler.

We can calculate the running time of this scheduler for random tasks therotically:
The time complexity of the methods in the Task_scheduler class for each method:

1. **init** Method:

The time complexity of this method is O(1) because it is an intialization of the attributes that takes constant time operation whatever the input size changes.

2. _print__self Method:

It has time complexity of O(n) since it iterates once over all the tasks and has single for loop in the code.

3. remove_dependency Method:

Also O(n) single it has a single for loop and iterates over all the tasks to update the dependencies.

4. get_tasks_ready Method:

It has a time complexity of O(nlogn) since it iterates over all the tasks O(n) and performs a heap operation which is heappush O(logn) for each task that meets the condition in the code. Since heappush is
O(logn), and it's inside a loop that can go up to n times, the combined complexity is O(nlogn).

5. check_unscheduled_tasks Method:

It has time complexity of O(n) since it has single for loop and iterates over all the tasks to check their status.

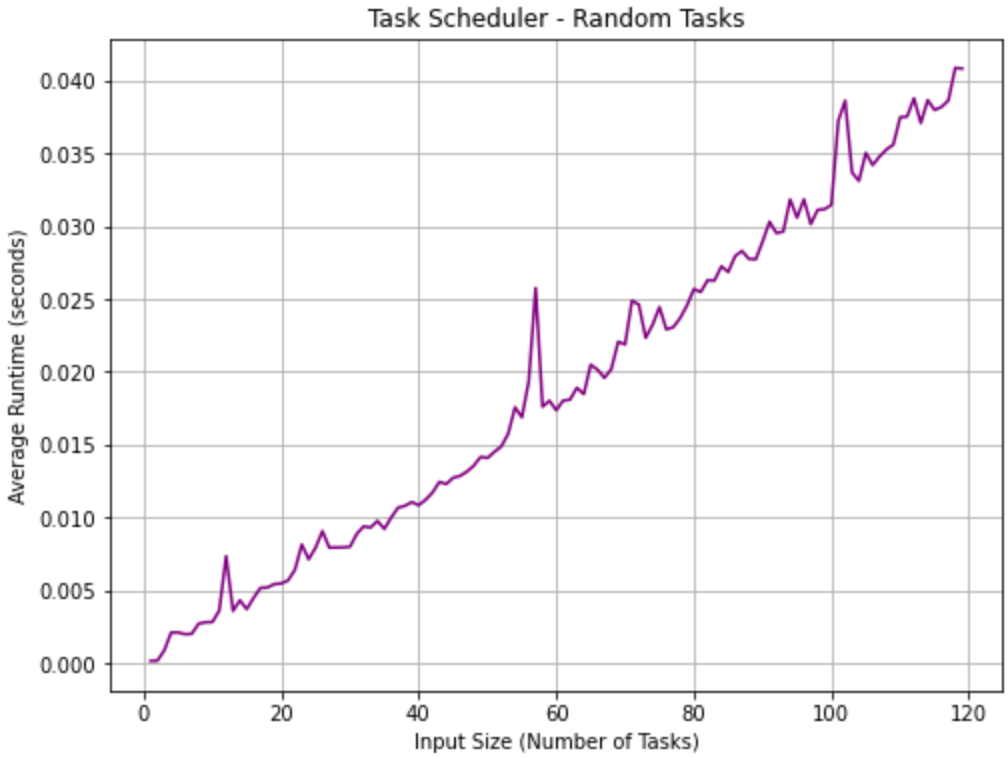
6. format_time Method:

It is just a simple mathmatical operation that has constant time complexity O(1)

7. run_task_scheduler Method:

This method has a while loop and inside it; there is get tasks ready which is O(nlogn), heappop which is O(log k) which k is the number of items in the heap, updating time and utility is O(1), remove dependencies O(n). Overall complexity per one iteration= O(nlogn)+O(logk)+O(1)+O(n) for n iteraton = n*(O(nlogn+logk+n))
However, since O(n^2logn) is typically the dominant factor in such a combination, especially for larger n, we can consider **O(n^2logn)** as the leading term for the overall complexity.

Then, we can test it experimentally by plotting the graph of the running time for random input:



Fig(5) It is a line graph where the x-axis represents the number of tasks (num_tasks_list) and the y-axis represents the average running time (avg_runtimes).This graph will show how the running time of your task scheduler scales with the number of tasks.

we can take two point on the x-axis and two points on the y-axis from the graph to see the ratio between them experimentally

when x1= 20 , y1= 0.006

when x2= 100 , y2= 0.032

then x2/x1 = 100/20 = 5

y2/y1= 0.32/0.006 = 5.33

nlog(n)= 5log(5)= 11

then 5.33 is larger than 5 which is n and less than 11 which is nlogn.
Another point:

x1= 20, y1= 0.006

x2=80 , y2= 0.026

x2/x1= 80/20 = 4

y2/y1 = 0.026/ 0.006 = 4.33
4log(4)= 8

4 < 4.33 < 8

then :n < value of y < nlogn

The values propose that the time complexity of this scheduler by experiment is between O(n) and O(nlogn).It doesn't contradict the therotical analysis since O(n^2logn) is the upper bound which the scheduler isn't going to approach everytime. To approach O(n^logn) means that we should push every task in n iterations which doesn't happen.

So, I am not going to use this scheduler for my day since I said how I missed many important tasks because I went with the organization of this schedule. I missed some academic and cultural tasks which have huge consequences on me like getting 2 in the poll because I didn't study for the class when I went with the scheduler.

This scheduler prioritizes executing tasks without dependencies first, distinguishing them from tasks with dependencies. In the Task-Scheduler class, both types of tasks are considered in the utility function. However, the impact of dependencies on the utility is not overly significant, as it has a scale factor of just 2. Once a task is completed, it's marked as such and removed from any dependency lists. This approach minimizes the occurrence of tasks with multiple unfulfilled dependencies which means that the lenght of the dependencies in the utility function isn't magnificent. The scheduler needs to ensure that tasks without dependencies are addressed first, which is essential for correctly sequencing tasks based on their prerequisites. This prioritization is achieved by modifying the get_tasks_ready function to first handle tasks without dependencies.

As a result, tasks that do have dependencies are only executed after their prerequisites have been met, maintaining an orderly and logical task flow. Also, removing the length of the dependencies from the utility function since now we are dealing with the tasks without dependencies.

Additionally, the scales assigned to ‘very healthy’ and ‘very unhealthy’ are identical, as noted in the discussion about the algorithm’s limitations. These scales ought to be distinct to reflect different utility values for tasks that result in ‘very healthy’ versus ‘very unhealthy’ outcomes. Clearly, the value assigned to ‘very healthy’ should be greater (such as 2), to differentiate it from the ‘unhealthy’ rating. This distinction is essential for the algorithm to accurately assess and prioritize tasks based on their impact on the user’s healthiness.

Also, this scheduler could be improved by adding one more heap for the tasks with fixed time like the classes or the meetings. In my case, it was a coincidence that it executes the class at 4pm which is actually my class time but when I change the starting time or the duration of any task, it is going to change the time of the class. By implementing a separate heap dedicated to these fixed-time tasks, the scheduler can ensure that they are executed at their predetermined times, regardless of changes in other variable-time tasks.

I implemented the suggested modification to the get_tasks_ready function, ensuring it prioritizes tasks without dependencies. After rerunning my scheduler, as demonstrated below, it’s evident that tasks with dependencies are now correctly sequenced after their prerequisites. For instance, the task of dressing up now precedes going to Myeongdong, and visiting Myeongdong comes before trying Korean food and studying at a café there. This adjustment effectively resolves the issue with task dependencies, making the schedule more practical and logical.

```
get boba
wake-up
🕛t=11h00
        started 'get boba' for 30 mins...
        ✅ Task completed at 11h30 with priority=0.6055300708194983
wake-up
🕛t=11h30
        started 'wake-up' for 30 mins...
        ✅ Task completed at 12h00 with priority=0.18257418583505536
have breakfast
🕛t=12h00
        started 'have breakfast' for 15 mins...
        ✅ Task completed at 12h15 with priority=0.8563488385776752
dress up
🕛t=12h15
        started 'dress up' for 15 mins...
        ✅ Task completed at 12h30 with priority=0.2581988897471611
take the bus to Myeondong
🕛t=12h30
        started 'take the bus to Myeondong' for 15 mins...
        ✅ Task completed at 12h45 with priority=-1.0327955589886444
try korean food at Myeondong
study at cafe
🕛t=12h45
        started 'try korean food at Myeondong' for 60 mins...
        ✅ Task completed at 13h45 with priority=0.8266397845091497
study at cafe
🕛t=13h45
        started 'study at cafe' for 120 mins...
        ✅ Task completed at 15h45 with priority=0.5845225972250061
Take the class
🕛t=15h45
        started 'Take the class' for 90 mins...
        ✅ Task completed at 17h15 with priority=0.10540925533894598
go back to the res hall
🕛t=17h15
        started 'go back to the res hall' for 15 mins...
        ✅ Task completed at 17h30 with priority=-1.0327955589886444
```

Fig(6) shows the output after modifing get_tasks_ready function

However, there's a minor inconsistency where 'getting boba' is scheduled before 'waking up.' This occurs because 'getting boba' is independent and contributes to happiness, but it's not logically feasible before waking up. To address this, 'getting boba' can be set to depend on 'waking up' and 'dress up' in the task input. This change is going to guarantee that 'wake-up' is going to be the first task to be executed and then the 'dress up' before getting boba.

```
wake-up
🕛t=11h00
        started 'wake-up' for 30 mins...
        ✅ Task completed at 11h30 with priority=0.18257418583505536
have breakfast
🕛t=11h30
        started 'have breakfast' for 15 mins...
        ✅ Task completed at 11h45 with priority=0.8563488385776752
dress up
🕛t=11h45
        started 'dress up' for 15 mins...
        ✅ Task completed at 12h00 with priority=0.2581988897471611
get boba
take the bus to Myeondong
🕛t=12h00
        started 'get boba' for 30 mins...
        ✅ Task completed at 12h30 with priority=0.6055300708194983
take the bus to Myeondong
🕛t=12h30
        started 'take the bus to Myeondong' for 15 mins...
        ✅ Task completed at 12h45 with priority=-1.0327955589886444
try korean food at Myeondong
study at cafe
🕛t=12h45
        started 'try korean food at Myeondong' for 60 mins...
        ✅ Task completed at 13h45 with priority=0.8266397845091497
study at cafe
🕛t=13h45
        started 'study at cafe' for 120 mins...
        ✅ Task completed at 15h45 with priority=0.5845225972250061
Take the class
🕛t=15h45
        started 'Take the class' for 90 mins...
        ✅ Task completed at 17h15 with priority=0.10540925533894598
go back to the res hall
🕛t=17h15
        started 'go back to the res hall' for 15 mins...
        ✅ Task completed at 17h30 with priority=-1.0327955589886444

🏁 Completed all planned tasks in 6h30min!

💁 Your scheduler has total utility of 1.3536325040752026 utils
```

Fig(7) shows the output after modifing ‘get boba’ to depend on the ‘wake-up’ and ‘dress-up’

I also fixed the scale of ‘very unhealthy’ to be -2 instead of 2 and that solved the third problem in the failure section.

```
Running a daily scheduler:

🕛t=14h00
        started 'A' for 15 mins...
        ✅ Task completed at 14h15 with priority=0.8563488385776752
🕛t=14h15
        started 'B' for 15 mins...
        ✅ Task completed at 14h30 with priority=-1.5491933384829668

🏁 Completed all planned tasks in 0h30min!

💁 Your scheduler has total utility of -0.6928444999052916 utils
```

Fig(8) shows the output after fixing the scale of ‘very unhealthy’ to -2

AI statement:

I didn't use any AI tools other than Grammarly to check the grammar.

Appendix Part I:

Learning outcomes:

#AlgoStratDataStruct:

I talked about how to improve the code's improvement by modifying the get_tasks_ready method and the utility function. I addressed why I made this improvement and how it will change my output. I also defined why this is an issue and why I removed the part of the length of dependencies in the utility function and showed how it affected the whole algorithm strategy by putting the tasks without dependencies in the method instead of putting all the tasks.

#ComplexityAnalysis:
I discussed the theoretical and empirical analysis of the scheduler's algorithm strategy. I defined the theoretical running time step by step for every method in the class and how I calculated that. I implemented a graph of the average running time for the random input size. Then, I used two points on the x_axis and y_axis to calculate the ratio between them to know the running time from the graph. Lastly, I stated how the experimental analysis is consistent with the theoretical analysis and why it differs. I also stated how the theoretical running time is an upper bound and when it happens.

#ComputationalCritique:
I critique the scheduler's output based on whether it is feasible. I set how it affects my day and the efficiency of this scheduler for organizing the tasks of my day. I stated my stance toward the scheduler, whether I will use it daily or not, and what consequences may arise because of this scheduler. I also critique the program's algorithm strategy and why it has an issue. I set the failures I got from the test cases and explained why they failed and how we could improve them. I defined how we can use this scheduler to get a high benefit if the tasks are flexible and more about hobbies and activities that have scale in the happiness and healthiness because other tasks will always be neutral. I talked about improving the code by modifying the get_tasks_ready

method and the utility function. I addressed why I made this improvement and how it will change my output. I also defined why this is an issue and why I removed the part of the length of dependencies in the utility function.

#PythonProgramming:

I used assertion statements for the test cases I added to show whether the code was working before and after my modification. I used different test cases to test the code in different situations and with different edge cases to find the failure and success of the code. I fixed the scale of 'very unhealthy' to be -2 so the code can differentiate between the tasks 'very unhealthy' and 'very healthy' so that it changes the output of the scheduler.

#CodeReadability:

I made a clear change in the get_tasks_ready method and correct code line. I added comments for every test case to make it easy for the reader to know what specific parts of the algorithm this test case tests and what they represent. I provided comments for the code of the running time, too.

#Professionalism:
Working with codes is hard since the code may clash many times. I tried to add the test cases in different code cells other than the main code cell. I tried to follow organized presentation in the Jupyter notebook. I also ignored any typos that may happen in the code or the paragraph. I also added the graph of the time complexity in another code cell so that it is easier to see the graph direct without scrolling the whole output.

HC

#dataviz:

I added a description to every figure in the assignment. I added the description of the table of tasks before the table to follow the APA style. I added different labels for the x_axis, y_axis, and title for the graph of the time complexity. I also added a description of what the graph represents. I defined the type of the graph (line graph) to visualize how the running time changes with the input size.

Appendix part II :

```
### Table with 'regular tasks' and 'rotation-city-centered tasks'
import pandas as pd
from tabulate import tabulate

# Create a table with my tasks
data = [
    {'id': 5, 'description': 'get boba', 'duration': 30, 'dependencies': [1],
     'profit':-6, 'happiness': 'very happy', 'healthiness': 'neutral'},
    {'id': 0, 'description': 'have breakfast', 'duration': 15, 'dependencies': [1],
     'profit':0, 'happiness': 'happy', 'healthiness': 'healthy'},
    {'id': 1, 'description': 'wake-up', 'duration': 30, 'dependencies': [],
     'profit':0, 'happiness': 'neutral', 'healthiness': 'neutral'},
    {'id': 2, 'description': 'dress up', 'duration': 15, 'dependencies': [0],
     'profit':0, 'happiness': 'neutral', 'healthiness': 'neutral'},
    {'id': 3, 'description': 'go back to the res hall', 'duration': 15, 'dependenci
     'profit':-2, 'happiness': 'neutral', 'healthiness': 'neutral'},
    {'id': 4, 'description': 'try korean food at Myeondong', 'duration': 60, 'deper
     'profit':-15, 'happiness': 'very happy', 'healthiness': 'healthy'},
    {'id': 6, 'description': 'study at cafe', 'duration': 120, 'dependencies': [8],
     'profit':0, 'happiness': 'happy', 'healthiness': 'neutral'},
    {'id': 7, 'description': 'Take the class', 'duration': 90, 'dependencies': [6],
     'profit':0, 'happiness': 'neutral', 'healthiness': 'neutral'},
    {'id': 8, 'description': 'take the bus to Myeondong', 'duration': 15, 'depender
     'profit':-2, 'happiness': 'neutral', 'healthiness': 'neutral'},
]

# Takes data from dictionary for the column's names and values
df = pd.DataFrame(data)
df.set_index('id', inplace=True)

# Print the table
print(tabulate(df, headers='keys', tablefmt='fancy_grid'))
```

```
╒══════╤═══════════════════════════╤════════════╤════════════════╤══════════╤═══
│   id │ description               │   duration │ dependencies   │   profit │ ha
╞══════╪═══════════════════════════╪════════════╪════════════════╪══════════╪═══
│    5 │ get boba                  │         30 │ [1]            │       -6 │ ve
├──────┼───────────────────────────┼────────────┼────────────────┼──────────┼───
│    0 │ have breakfast            │         15 │ [1]            │        0 │ ha
├──────┼───────────────────────────┼────────────┼────────────────┼──────────┼───
│    1 │ wake-up                   │         30 │ []             │        0 │ ne
```

2	dress up	15	[0]	0	ne
3	go back to the res hall	15	[7]	-2	ne
4	try korean food at Myeondong	60	[8]	-15	ve
6	study at cafe	120	[8]	0	ha
7	Take the class	90	[6]	0	ne
8	take the bus to Myeondong	15	[2]	-2	ne

```
# This code was retrived from my PCW for Session 13[7.2]

class MaxHeapq:
    """
    A class that implements properties and methods
        that support a max priority queue data structure

    Attributes:
    ----------
    heap : arr
        A Python list where key values in the max heap are stored
    heap_size: int
        An integer counter of the number of keys present in the max heap
    """

    def __init__(self):
        """
        Initializes the class attributes

        Parameters:
        ----------
        None
        """
        self.heap       = []
        self.heap_size  = 0

    def left(self, i):
        """
```

```
        Takes the index of the parent node
        and returns the index of the left child node

        Parameters:
        ----------
        i: int
          Index of parent node

        Returns:
        ----------
        int
          Index of the left child node
        """
        return 2 * i + 1

    def right(self, i):
        """
        Takes the index of the parent node
        and returns the index of the right child node

        Parameters:
        ----------
        i: int
            Index of parent node

        Returns:
        ----------
        int
            Index of the right child node
        """
        return 2 * i + 2

    def parent(self, i):
        """
        Takes the index of the child node
        and returns the index of the parent node

        Parameters:
        ----------
        i: int
            Index of child node

        Returns:
```

```
        ----------
        int
            Index of the parent node
        """
        return (i - 1)//2

    def maxk(self):
        """
        Returns the highest key in the priority queue.

        Parameters:
        ----------
        None

        Returns:
        ----------
        int
            the highest key in the priority queue
        """
        return self.heap[0]

    def heappush(self, key):
        """
        Insert a key into a priority queue

        Parameters:
        ----------
        key: int
            The key value to be inserted

        Returns:
        --------
        None
        """
        self.heap.append(-float("inf"))
        self.increase_key(self.heap_size,key)
        self.heap_size+=1

    def increase_key(self, i, key):
        """
        Modifies the value of a key in a max priority queue
        with a higher value and maintains the max-heap property
```

```
        Parameters:
        ----------
        i: int
            The index of the key to be modified
        key: int
            The new key value

        Returns:
        --------
        None
        """
        if key < self.heap[i]:
            raise ValueError('new key is smaller than the current key')
        self.heap[i] = key
        while i > 0 and self.heap[self.parent(i)] < self.heap[i]:
            j = self.parent(i)
            holder = self.heap[j]
            self.heap[j] = self.heap[i]
            self.heap[i] = holder
            i = j

    def heapify(self, i):
        """
        Creates a max heap from the index given

        Parameters:
        ----------
        i: int
            The index of of the root node of the subtree to be heapify

        Returns:
        --------
        None
        """
        l = self.left(i)
        r = self.right(i)
        heap = self.heap
        if l <= (self.heap_size-1) and heap[l]>heap[i]:
            largest = l
        else:
            largest = i
        if r <= (self.heap_size-1) and heap[r] > heap[largest]:
            largest = r
```

```
            if largest != i:
                heap[i], heap[largest] = heap[largest], heap[i]
                self.heapify(largest)

        def heappop(self):
            """
            Returns the largest key in the max priority queue,
            removes it from the max priority queue and
            mantains min-heap property

            Parameters:
            ----------
            None

            Returns:
            --------
            int
                the max value in the heap that is extracted
            """
            if self.heap_size < 1:
                raise ValueError('Heap underflow: There are no keys in the priority que
            maxk = self.heap[0]
            self.heap[0] = self.heap[-1]
            self.heap.pop()
            self.heap_size-=1
            self.heapify(0)
            return maxk


    #I modified in get-tasks-ready and priority value methods
    import math as m

    class Task:
        """
        A class with attributes and methods to describe
        and manage a task.

        Attributes:
        -----------
        id (int):
            Task id (a reference number)
        description (str):
            Task short description
```

```
        duration (int):
            Task duration in minutes
        dependencies (list):
            List of task ids that need to precede this task
        status (str):
            Current status of the task
        profit (int):
            Task profit or cost in dollars
        happiness (str):
            Task qualitative rate (very happy, happy, neutral, sad, or very sad)
        healthiness (str):
            Task qualitative rate (very healthy, healthy, neutral, unhealthy, or very u
        """

        def __init__(self, id, description, duration,
                     dependencies, status="N", profit=0,
                     happiness='neutral', healthiness='neutral'):
            '''
            Initializes a new intance of os the class Task with its attributes.

            Parameters
            ----------
            id: int
                Task id (a reference number)
            description: str
                Task short description
            duration: int
                Task duration in minutes
            dependencies: list
                List of task ids that need to preceed this task
            status: str, default='N'
                Current status of the task
            profit: int, default=0
                Task profit or cost in dollars
            happiness: str, default='neutral'
                Task qualitative rate (very happy, happy, neutral, sad, or very sad)
            healthiness: str, default='neutral'
                Task qualitative rate (very healthy, healthy, neutral, unhealthy, or ve
            '''
            self.id = id
            self.description = description
            self.duration = duration
            self.dependencies = dependencies
```

```
        self.status = status
        self.profit = profit
        self.happiness = happiness
        self.healthiness = healthiness

    def satisfaction(self):
        '''
        Calculate overall satisfaction value on happiness, healthiness, and profit.

        Parameters:
        -----------
        None

        Returns:
        -------
        satisfaction: float
            A value (positive or negative) for the average satisfaction
        '''
        happiness_rates = {'very happy':2, 'happy':1, 'neutral':0, 'sad':-1, 'very
        happiness_point = happiness_rates[self.happiness]

        healthiness_rates = {'very healthy':2, 'healthy':1, 'neutral':0, 'unhealthy
        healthiness_point = healthiness_rates[self.healthiness]

        if self.profit > 20:
            profit_point = 2

        if 0 < self.profit <= 20:
            profit_point = 1

        if self.profit == 0:
            profit_point = 0

        if -20 <= self.profit < 0:
            profit_point = -1

        if self.profit < -20:
            profit_point = -2

        satisfaction = (1*happiness_point + 1*healthiness_point + 1*profit_point)/3

        return satisfaction
```

```
    def priority_value(self):
        '''
        Combines satisfaction, duration, and dependencies in multivariable functior
        Calculates the overall utility of a task (priority value) based on Loss Ave

        Parameters:
        -----------
        None

        Returns
        --------
        utility: float
            The calculated priority value in utils
        '''
        x = self.satisfaction()
        y = (1/self.duration)

        # Combining aspect on f(x,y,z)
        f = 1*x + 1*y

        # Calculate utility
        if f >= 0:
            utility = m.sqrt(f)

        if f < 0:
            utility = -2*m.sqrt(-f)

        return utility


    def __lt__(self, other):
        '''
        Compares two instances' priority values and checks which one has the smalle

        Parameters:
        ----------
        other: Task
            Another Task instance to compare with.

        Returns:
        --------
        bool:
```

```
            True if this task's priority value is smaller than the other's priority
            False otherwise or if the input is not a Task instance.
        '''
        if isinstance(other, Task):
            return self.priority_value() < other.priority_value()
        else:
            return False

class TaskScheduler:
    """
    A daily task scheduler using priority queues.

    Attributes:
    ----------
    tasks: list
        a list of tasks with Task() attributes
    priority_queue: heap
        a max heap priority queue that takes a task priority value as key

    Methods:
    --------
    __init__(self,tasks):
        initialize the attributes
    print_self(self):
        print tasks added to the scheduler
    remove_dependency(seld,id):
        update dependency lists if a task is completed
    get_tasks_ready(self):
        push tasks not started into the queue and update their status
    check_unscheduled_tasks(self):
        check for classes out of the scheduler
    format_time(self, time):
        transform time from minutes to hours in a day
    run_task_scheduler(self, starting_time):
        gets tasks in the queue and creates scheduler based on priority value

    """
    # Assign string for each status
    NOT_STARTED = 'N'
    IN_PRIORITY_QUEUE = 'I'
    COMPLETED = 'C'

    def __init__(self, tasks):
```

```
        '''
        Initializes a new instance of the TaskScheduler class with its attributes.

        Parameters:
        -----------
        - tasks: list of Task
            A list of Task instances to be managed by the scheduler.
        '''
        self.tasks = tasks
        self.priority_queue = MaxHeapq()

    def print_self(self):
        '''
        Print the tasks added to the daily scheduler, including their descriptions,
        durations, and the presence of dependencies

        Parameters:
        -----------
        None

        Returns:
        --------
        str
            A formatted string with the tasks' information
        '''
        print("Tasks added to the daily scheduler:")
        print("-------------------------------------")
        for t in self.tasks:
            print(f"➡️ '{t.description}', duration = {t.duration} mins.")
            if len(t.dependencies)>0:
                print(f"\t ⚠️ This task depends on others!")

    def remove_dependency(self, id):
        '''
        Update the dependency lists of tasks if a task with a given id is completed

        Parameters:
        -----------
        id: int
            Task id (a reference number)

        Returns:
        --------
```

```
            None
            '''
            for t in self.tasks:
                if t.id != id and id in t.dependencies:
                    t.dependencies.remove(id)

        def get_tasks_ready(self):
            '''
            Push tasks that are not started into the priority queue and update status.

            Parameters:
            -----------
            None

            Returns:
            --------
            None
            '''
            for task in self.tasks:
                # I removed the condition to rely only on no dependencies tasks
                if task.status == self.NOT_STARTED and not task.dependencies:
                    task.status = self.IN_PRIORITY_QUEUE
                    self.priority_queue.heappush(task)
            for task in self.priority_queue.heap:
                    print(task.description)
        def check_unscheduled_tasks(self):
            '''
            Check if there are unscheduled tasks that haven't started yet.

            Parameters:
            -----------
            None

            Returns:
            --------
            bool:
                True if there are unscheduled tasks, False otherwise.
            '''
            for task in self.tasks:
                if task.status == self.NOT_STARTED:
                    return True
            return False
```

```
    def format_time(self, time):
        '''
        Transform current time of scheduler from minutes to hours and minutes in a

        Parameters:
        -----------
        time: int
            Time in minutes

        Returns:
        --------
        str:
            Formatted time in the "hh:mm" format
        '''
        return f"{time//60}h{time%60:02d}"

    def run_task_scheduler(self, starting_time):
        '''
        Run the task scheduler to manage and execute tasks based on their priority

        Parameters:
        -----------
        starting_time: int
            The starting time of the scheduler in minutes.

        Returns:
        --------
        str:
            Formatted string with the daily scheduler that informs the start/end ti
            the priority value of each task, and the overall utility of the schedul
        '''
        # Start counting time and utility
        current_time = starting_time
        current_utils = 0

        print("Running a daily scheduler:\n")

        while self.check_unscheduled_tasks() or self.priority_queue:
            # Push tasks ready into the priority queue
            self.get_tasks_ready()

            # Check for tasks in the priority queue
            if len(self.priority_queue.heap) > 0:
```

```
                # Get the task on top of the priority queue
                task = self.priority_queue.heappop()
                print(f"🕛 t={self.format_time(current_time)}")
                print(f"\tstarted '{task.description}' for {task.duration} mins..."

                # Update time and utility
                current_time += task.duration
                current_utils += task.priority_value()
                print(f"\t✅ Task completed at {self.format_time(current_time)} wit

                # If the task is done, remove it from the dependency list
                self.remove_dependency(task.id)
                task.status = self.COMPLETED

            elif len(self.priority_queue.heap) == 0:
                break

        total_time = current_time - starting_time
        total_utils = current_utils
        print(f"\n🏁 Completed all planned tasks in {total_time//60}h{total_time%60
        print(f"\n🤷‍♂️ Your scheduler has total utility of {total_utils} utils")

# Creating a list of tasks from data
task_list = [Task(**task_data) for task_data in data]

# Using the list to create an instance of TaskScheduler()
task_scheduler = TaskScheduler(task_list)

# Print the scheduler's input
task_scheduler.print_self()
```

```
Tasks added to the daily scheduler:
-------------------------------------
➡️ 'get boba', duration = 30 mins.
        ⚠️ This task depends on others!
➡️ 'have breakfast', duration = 15 mins.
        ⚠️ This task depends on others!
➡️ 'wake-up', duration = 30 mins.
➡️ 'dress up', duration = 15 mins.
        ⚠️ This task depends on others!
➡️ 'go back to the res hall', duration = 15 mins.
```

```
        ⚠️ This task depends on others!
➡️ 'try korean food at Myeondong', duration = 60 mins.
        ⚠️ This task depends on others!
➡️ 'study at cafe', duration = 120 mins.
        ⚠️ This task depends on others!
➡️ 'Take the class', duration = 90 mins.
        ⚠️ This task depends on others!
➡️ 'take the bus to Myeondong', duration = 15 mins.
        ⚠️ This task depends on others!


# Convert starting time in minutes
start_scheduler_at = 11*60

task_scheduler.run_task_scheduler(start_scheduler_at)


Running a daily scheduler:

wake-up
🕛 t=11h00
        started 'wake-up' for 30 mins...
        ✅ Task completed at 11h30 with priority=0.18257418583505536
have breakfast
get boba
🕛 t=11h30
        started 'have breakfast' for 15 mins...
        ✅ Task completed at 11h45 with priority=0.8563488385776752
get boba
dress up
🕛 t=11h45
        started 'get boba' for 30 mins...
        ✅ Task completed at 12h15 with priority=0.6055300708194983
dress up
🕛 t=12h15
        started 'dress up' for 15 mins...
        ✅ Task completed at 12h30 with priority=0.2581988897471611
take the bus to Myeondong
🕛 t=12h30
        started 'take the bus to Myeondong' for 15 mins...
        ✅ Task completed at 12h45 with priority=-1.0327955589886444
try korean food at Myeondong
study at cafe
🕛 t=12h45
```

```
        started 'try korean food at Myeondong' for 60 mins...
        ✅ Task completed at 13h45 with priority=0.8266397845091497
study at cafe
🕐 t=13h45
        started 'study at cafe' for 120 mins...
        ✅ Task completed at 15h45 with priority=0.5845225972250061
Take the class
🕐 t=15h45
        started 'Take the class' for 90 mins...
        ✅ Task completed at 17h15 with priority=0.10540925533894598
go back to the res hall
🕐 t=17h15
        started 'go back to the res hall' for 15 mins...
        ✅ Task completed at 17h30 with priority=-1.0327955589886444

🏁 Completed all planned tasks in 6h30min!

👨‍🎨 Your scheduler has total utility of 1.3536325040752026 utils
```

Test cases:

```
# test case for the duration
task6 = Task(6, "cook",duration= 30, dependencies= [], happiness = 'sad')
task7 = Task(7, "go for a walk", duration= 15, dependencies= [], happiness= 'happy'
task_list = [task6, task7]

# Priority values before the scheduler runs
task6_value = task6.priority_value()
task7_value = task7.priority_value()


# Create a task scheduler and run it
scheduler = TaskScheduler(task_list)
scheduler.run_task_scheduler(14*60)  # Starting time at 2:00 PM
```

```
Running a daily scheduler:

go for a walk
cook
```

```
🕛 t=14h00
        started 'go for a walk' for 15 mins...
        ✅ Task completed at 14h15 with priority=0.6324555320336759
cook
🕛 t=14h15
        started 'cook' for 30 mins...
        ✅ Task completed at 14h45 with priority=-1.0954451150103321

🏁 Completed all planned tasks in 0h45min!

🤦 Your scheduler has total utility of -0.46298958297665627 utils
```

```
# test case after modifying the scale of very unhealthy to -2 instead of 2
task2 = Task(2, "A",duration= 15, dependencies= [], healthiness = 'very healthy')
task3 = Task(3, "B", duration= 15, dependencies= [], healthiness= 'very unhealthy')
task_list = [task3, task2]

# Priority values before the scheduler runs
task2_value = task2.priority_value()
task3_value = task3.priority_value()


# Create a task scheduler and run it
scheduler = TaskScheduler(task_list)
A = scheduler.run_task_scheduler(14*60)  # Starting time at 2:00 PM
```

```
Running a daily scheduler:

A
B
🕛 t=14h00
        started 'A' for 15 mins...
        ✅ Task completed at 14h15 with priority=0.8563488385776752
B
🕛 t=14h15
        started 'B' for 15 mins...
        ✅ Task completed at 14h30 with priority=-1.5491933384829668

🏁 Completed all planned tasks in 0h30min!
```

```
👨‍🎨 Your scheduler has total utility of -0.6928444999052916 utils

# test case for the healthiness scales before modifying
task2 = Task(2, "A",duration= 15, dependencies= [], healthiness = 'very healthy')
task3 = Task(3, "B", duration= 15, dependencies= [], healthiness= 'very unhealthy')
task_list = [task3, task2]

# Priority values before the scheduler runs
task2_value = task2.priority_value()
task3_value = task3.priority_value()


# Create a task scheduler and run it
scheduler = TaskScheduler(task_list)
B = scheduler.run_task_scheduler(14*60)  # Starting time at 2:00 PM

assert A==B

Running a daily scheduler:

A
B
🕛 t=14h00
        started 'A' for 15 mins...
        ✅ Task completed at 14h15 with priority=0.8563488385776752
B
🕛 t=14h15
        started 'B' for 15 mins...
        ✅ Task completed at 14h30 with priority=-1.5491933384829668

🏁 Completed all planned tasks in 0h30min!

👨‍🎨 Your scheduler has total utility of -0.6928444999052916 utils


# test case for the profit one
task2 = Task(2, "watch movie",profit = 0, dependencies= [], happiness= 'very happy'
task3 = Task(3, "work-study", profit = 40, dependencies= [], happiness= 'very sad',
task_list = [task3, task2]

# Priority values before the scheduler runs
task2_value = task2.priority_value()
```

```
task3_value = task3.priority_value()


# Create a task scheduler and run it
scheduler = TaskScheduler(task_list)
C = scheduler.run_task_scheduler(14*60)  # Starting time at 2:00 PM
```

```
Running a daily scheduler:

watch movie
work-study
🕛 t=14h00
        started 'watch movie' for 120 mins...
        ✅ Task completed at 16h00 with priority=0.8215838362577491
work-study
🕛 t=16h00
        started 'work-study' for 120 mins...
        ✅ Task completed at 18h00 with priority=0.09128709291752768

🏁 Completed all planned tasks in 4h00min!

👨‍🎨 Your scheduler has total utility of 0.9128709291752768 utils
```

```
# test case for the profit one
task2 = Task(2, "watch movie",profit = 0, dependencies= [], happiness= 'very happy'
task3 = Task(3, "work-study", profit = 40, dependencies= [], happiness= 'very sad',
task_list = [task2, task3]

# Priority values before the scheduler runs
task2_value = task2.priority_value()
task3_value = task3.priority_value()


# Create a task scheduler and run it
scheduler = TaskScheduler(task_list)
D = scheduler.run_task_scheduler(14*60)  # Starting time at 2:00 PM
assert C==D
```

```
Running a daily scheduler:

watch movie
work-study
🕛 t=14h00
        started 'watch movie' for 120 mins...
        ✅ Task completed at 16h00 with priority=0.8215838362577491
work-study
🕛 t=16h00
        started 'work-study' for 120 mins...
        ✅ Task completed at 18h00 with priority=0.09128709291752768

🏁 Completed all planned tasks in 4h00min!

💁 Your scheduler has total utility of 0.9128709291752768 utils
```

Time complexity graph

```
# Importing packages
import time
import numpy as np
import matplotlib.pyplot as plt
# Initialize lists and variables for the trials
num_tasks_list = []
avg_runtimes = []
experiments = 50
input_size = 120
interval = 1

def generate_random_task_data(task_id):
    # Example function to generate random task data
    # Replace this with your actual data generation logic
    task_data = {
        "id": task_id,
        "description": f"Task {task_id}",
        "duration": np.random.randint(1, 60),  # Random duration between 1 and 60 m
        "dependencies": [],  # You can add logic to generate dependencies if needed
        # Add other necessary attributes here
    }
    return task_data
```

```
# Loop to test the scheduler's performance with increasing number of tasks
for size in range(1, input_size, interval):  # Adjust the range as needed
    num_tasks_list.append(size)
    total_runtime = 0    # Initialize total runtime for averaging

    for i in range(experiments):
        # Generate a list of random tasks
        random_tasks = [Task(**generate_random_task_data(task_id)) for task_id in r

        # Measure the runtime for trials of running the scheduler
        start_time = time.time()
        random_scheduler = TaskScheduler(random_tasks)
        random_scheduler.run_task_scheduler(360) # Start at 6:00 AM
        end_time = time.time()
        runtime = end_time - start_time
        total_runtime += runtime

    avg_runtimes.append(total_runtime/experiments)


plt.figure(figsize=(8, 6))
plt.plot(num_tasks_list, avg_runtimes, linestyle='-', color='purple')
plt.title('Task Scheduler - Random Tasks')
plt.xlabel('Input Size (Number of Tasks)')
plt.ylabel('Average Runtime (seconds)')
plt.grid(True)
plt.show()
```