Project 1: Design a Scheduler

Minerva University

CS110 - Problem Solving with Data Structures and Algorithms

Prof. H. Ribeiro

October 31, 2023

I. Setting up.

A. Table of tasks for the scheduler at the starting time (8:30 AM)

Task ID	Task Description	Task Duration (minutes)	Task Dependencies	Fixed Time	Deadline	# of people involved	Status
1	Research about K-Economy	120					Not yet started
2	Get ready	30			9:00 AM		Not yet started
3	Bus to Seoul Museum	35	Task 2				Not yet started
4	Breakfast with Sookmyung buddy	40	Task 2,3	10:20 AM		2	Not yet started
5	Museum tour	90	Task 1,2,3,4	11:00 AM		2	Not yet started
6	Take pictures	15	Task 1,2,3,4,5				Not yet started
7	Submit reflection and report	60	Task 1,2,3,4,5,6		4:00 PM		Not yet started
8	Walk around Gyeongbokgung Palace	30	Task 1,2,3,4,5,6				Not yet started
9	Get street food and practice Korean	45					Not yet started
10	Metro back to the residence	30	Task 1,2,3,4,5,6,8				Not yet started
11	Get groceries with roommates	20	Task 1,2,3,4,5,6,8, 9,10		6:00 PM	3	Not yet started

B. This semester, I set a goal to either hang out with my buddy from Sookmyung University or my Minerva friends to discover and enjoy Seoul on weekends. These tasks are an example of what a Saturday would look like for me. The numbers are to indicate activities that showcase my immersion into the city.

Task ID	Description	Relevance
1	Research about Korean Economy	Expand my understanding of the city, and increase my research skills. It is interesting to read about the rich history of the various dynasties in Korean history
2	Get ready	A task that I do every day. It has a deadline of 9:00 AM because it is a personal goal I set for myself to be ready at 9 AM every day.
3	Bus to Seoul Museum	1) The transportation system in Seoul is quite complicated and figuring this system out is another daily task of mine
4	Breakfast with Sookmyung buddy	I enjoy Korean food a lot and enjoy it even more when my company is a Korean friend who often tells me about the meaning of the food. It has a set time at 10:20 AM as we plan to meet then.
5	Museum tour	2) Activity to immerse myself in the city's culture. The museum tour has a fixed time at 11 AM and if I'm late, I'll miss it.
6	Take pictures for memories and proof of participation	This is a hobby and it's fun. I also take pictures of the city every day

7	Write a report and reflection	This can either be for my own journaling or a city experience report. The deadline for the form is at 4:00 PM. However, I aim to submit it early if possible.
8	Walk around Gyeongbokgung Palace	Going on walks is a crucial part of my day, and I love walking in different parks and historical places
9	Get street food and practice Korean	3) Street food is a Korean specialty. It is affordable, delicious, accessible, and a great way for me to practice Korean by ordering
10	Metro back to the residence	4) Using different modes of transportation to get to different destinations is representative of my daily tasks
11	Get groceries with roommates	A usual activity, but with a deadline at 6:00 PM because my roommates have evening classes. It also involves 2 more people, thus, the time has to be precise to not affect other people's schedules.

II. Preparing my algorithmic strategy.

A. *Why priority queue?*

a. 3 different classes: MaxHeap, Task, and Scheduler

- Class MaxHeap: Sets up the foundation for managing tasks' priority values and efficiently selects the tasks with the higher priority value to be executed first. Suitable for our requirements to select tasks with highest priority to be executed (element with the maximum value in the root node)
- Class Task: Includes attributes of a task (ID, description, duration, dependencies, status, fixed_time, deadlines, people). Its method includes calculating the priority value and

managing task status.

- Class Scheduler: Executes the scheduling process, satisfies constraints, and prints output

A priority queue allows efficient insertions and deletions, especially when dealing with tasks with dependencies or fixed times. We can insert or remove tasks from the queue with a time complexity of O(log n), suitable for a frequently changing task list. The 3 classes also provide readability, clarity, and better management. The Scheduler class can utilize the MaxHeap to prioritize tasks, and the Task class allows manipulation of each task's attributes.

Moreover, if we decide to upgrade the algorithm and make it interactive, we can have streaming data throughout the day from users instead of fixed data in the beginning like the current version. Priority queues are particularly suitable to handle this because of their efficient insertion and deletion operations as stated earlier.

b. 1 queue with 2 constraints:

Instead of 2 priority queues, I chose to do 1 queue of priority queues and 2 constraints of fixed time and deadlines. Two priority queues would require more time and space complexity to operate compared to my method.

Benefits of using constraints and 1 priority queue include:

- Reduces the overhead of maintaining and storing two separate queues
- Avoids the time complexity of comparing tasks from two separate queues and simplifies the process of selecting the highest priority task to be scheduled
- A single priority queue can also incorporate multiple constraints without complicating the data structure. When a task is inserted into the queue, it can be placed into the appropriate 'sub-queue' based on its constraints (fixed time or deadline)

c. Constraints:

1. <u>Fixed time</u>:

If I'm late to a fixed time event, here's a formula to cut down the duration of the task in order to finish on time and avoid the domino effect of continuously delaying subsequent events.

Original end time = Fixed time + Duration

New duration = Original end time - Current time

Example: if my friend and I agreed to meet at 10:20 AM (fixed time) and have breakfast for 40 minutes (duration), which means we planned to finish by 11:00 AM (original end time).
However, I came late at 10:45 AM (current time) because I woke up late (dependency); thus, it makes sense to eat breakfast quicker in 11:00 - 10:45 = 15 minutes (new duration), because we do not want to delay other plans later in the day.
Because datetime is being utilized for the algorithm, I will also make sure that the fixed time and current time are on the same date. I will also use timedelta to calculate the difference between the current time and the fixed time or the deadline. This way, the algorithm can effectively determine whether there is enough time to complete a task before a deadline or the duration of a task should be cut down to finish on time.

2. <u>Deadline</u>:

How this constraint is taken into account is explained in Part B below.

B. <u>How my scheduler works in a high-level description:</u>

Imagine you have a list of tasks to complete during the day, and each task has a certain duration and dependencies on other tasks; some tasks have constraints such as deadlines, and/ or happen at a fixed time that can't be changed. The goal of this scheduler is to determine the best order to complete all the tasks based on their priority values given the constraints while maximizing your productivity.

The fairest way to determine the priority value of each task is by computing it through a formula with various factors with their significance weighted differently. This formula is further explained below. Using this value, the algorithm schedules tasks with higher priority first and leaves tasks with lower priority later. Moreover, if a task depends on other tasks that haven't been completed, it has to wait. Otherwise, it gets added to the priority queue. If two tasks have the same priority value, the algorithm sorts them by their ID (as inputted).

There are two constraints that this algorithm was built to satisfy.

1. Fixed time:

- If a task happens at a fixed time, it will be scheduled at that time, regardless of its priority value.
- If, because of its dependencies, a task is scheduled later than its fixed time, the scheduler reduces the duration of the task to the remaining time till the original finish time. This aims to help finish the task on time, as planned, even if we start it late, and avoid delaying the long list of remaining tasks for the rest of the day.
- If the remaining time till the original finish time is less than or equal to 0, the scheduler will skip this task and move on to the next one in the priory queue. For example, the museum tour starts at 11:00 AM (fixed time) with a duration of 60 minutes, which means it ends at 12:00 PM. I get there at 12:05 PM, meaning the remaining time is negative (12:00 - 12:05 = -5 minutes), thus it does not make sense to reschedule the task because

there is only one available time slot for the tour and I missed it.

2. Deadlines:

For deadlines, if it has passed, the code prints a message stating that the task missed the deadline, marks the task as completed, and proceeds to the next task. If at the current time, there is not enough time to complete the task by the deadline, the code will calculate the new scheduled time to ensure it can be completed by the deadline. Otherwise, it starts the task as soon as possible to make the most use of the time before the deadline.

Execution: The scheduler takes the most important task from the priority queue and executes it. The algorithm then updates the time, marks the task as completed, removes its dependencies, and adds its duration to the total time. The output includes the schedule, the total time to complete the tasks, and the total utility value gained.

C. *How I define and compute the priority value of each task:*

a. Factors that influence the priority value:

- Task dependencies: Tasks with more dependencies will have higher priority because they depend on the completion of other tasks, which means they could also be critical milestones that allow other tasks to proceed in a series of tasks that depend on each other. Weight 0.5
    - For example, to get the bus to the museum (3), I have to be ready (2); and to have breakfast with my friend (4), I have to catch the bus to get there (3). Thus, task (4) is dependent on both tasks (3) and (2).
    - Even if we entered task 3 as the only dependency for task 4, the algorithm would

still schedule task 4 after tasks 3 and 2, as task 3 depends on task 2. However, we are calculating the priority value based on the number of dependencies that a task has, we need to input all accumulated dependencies, instead of just the one that the task is directly dependent on.

- Task duration: I personally prefer to complete longer tasks first since I have more energy earlier in the day and tend to get tired after a long day. However, this factor is not necessarily as important as resolving small tasks can also help celebrate small wins; thus, its weight = 0.2
- Number of people involved: The more people are involved in a task, the more prioritized it should be, because this task not only affects me but also other people. Because this impact can be on a larger scale, it should have the highest weight = 0.7

Based on these factors, I created a formula:

Priority Value = (0.5*Task Dependencies) + (0.2* Task Duration) + (0.7 * Number of People Involved)

b. Constraints:

- Fixed time slots/ deadlines: A museum tour has a fixed time slot that I have to sign up for in advance or the city experience report has a fixed deadline. I make sure that this factor can be handled in multiple ways (ie: skipping the task or cutting down on durations if late) and can override other factors. This is because the time is fixed and can affect other people's schedules and result in negative consequences if late/ missed. I also want to avoid the domino effect.
- Deadline: There are tasks that have deadlines which means that the task has to be

completed by that time. To avoid unexpected delays, low-quality products, and anxiety, I want to start tasks with deadlines as soon as possible and finish before the deadline.

III. Python Implementation.

A. OOP implementation of MaxHeap:

```
#adopted this class from Session 13 preclass-work
class MaxHeapq:
    """
    A class to represent a max-heap

    Attributes
    ----------
    heap: arr
        A Python list where key values in the max heap are stored
    length: int
        the number of keys present in the max heap

    Methods:
    ----------
    left(i):
        Returns the index of the left child of the input index, if exists
    right(i):
        Returns the index of the right child of the input index, if exists
    parent(i):
        Returns the index of the parent node of the input index, if exists
    heappush(key):
        Adds an element to the heap
    heappop():
        Removes the root node of the heap
    increase_key(i, key):
        Increase the priority of the ith element in the max heap
    heapify(i):
        Puts the i-th element in the heap to one of its correct position
    """
```

… other methods

```python
    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

        largest = i
        if l <= (self.heap_size - 1) and heap[l] > heap[i]:
            largest = l
        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)
```

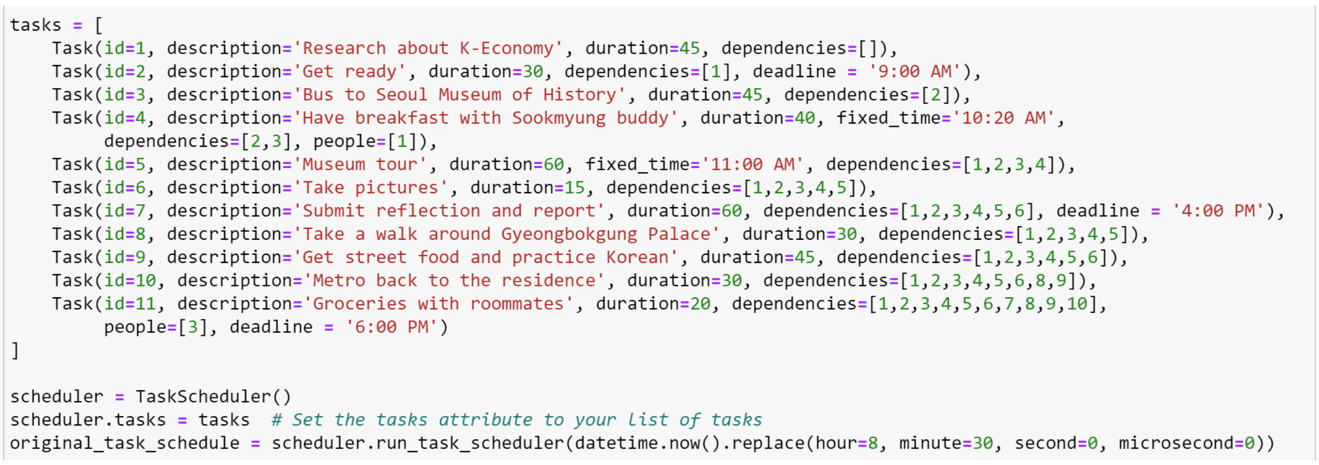
```python
# A and B were retrieved from pre-class work
A = [4, 3, 6, 8, 2, -5, 100]
my_heap = MaxHeapq()
[my_heap.heappush(k) for k in A]
assert(my_heap.heap == [100, 6, 8, 3, 2, -5, 4])

B = [6,4,7,9,10,-5,-6,12,8,3,1,-10]
her_heap = MaxHeapq()
[her_heap.heappush(k) for k in B]
assert(her_heap.heap == [12, 10, 6, 9, 7, -5, -6, 4, 8, 3, 1, -10])

C = [8, 9, 10, 45, -5, 123, 7]
our_heap = MaxHeapq()
[our_heap.heappush(k) for k in C]
assert our_heap.heap == [123, 10, 45, 8, -5, 9, 7]
```

B. Inputs and Outputs:

- Input 1:



```
tasks = [
    Task(id=1, description='Research about K-Economy', duration=45, dependencies=[]),
    Task(id=2, description='Get ready', duration=30, dependencies=[1], deadline = '9:00 AM'),
    Task(id=3, description='Bus to Seoul Museum of History', duration=45, dependencies=[2]),
    Task(id=4, description='Have breakfast with Sookmyung buddy', duration=40, fixed_time='10:20 AM',
         dependencies=[2,3], people=[1]),
    Task(id=5, description='Museum tour', duration=60, fixed_time='11:00 AM', dependencies=[1,2,3,4]),
    Task(id=6, description='Take pictures', duration=15, dependencies=[1,2,3,4,5]),
    Task(id=7, description='Submit reflection and report', duration=60, dependencies=[1,2,3,4,5,6], deadline = '4:00 PM'),
    Task(id=8, description='Take a walk around Gyeongbokgung Palace', duration=30, dependencies=[1,2,3,4,5]),
    Task(id=9, description='Get street food and practice Korean', duration=45, dependencies=[1,2,3,4,5,6]),
    Task(id=10, description='Metro back to the residence', duration=30, dependencies=[1,2,3,4,5,6,8,9]),
    Task(id=11, description='Groceries with roommates', duration=20, dependencies=[1,2,3,4,5,6,7,8,9,10],
         people=[3], deadline = '6:00 PM')
]

scheduler = TaskScheduler()
scheduler.tasks = tasks  # Set the tasks attribute to your list of tasks
original_task_schedule = scheduler.run_task_scheduler(datetime.now().replace(hour=8, minute=30, second=0, microsecond=0))
```

- Output 1:

In Task 4 “Have breakfast”, the duration was supposed to be 40 minutes and was due to finish at 10:20 AM + 40 minutes = 11:00 AM. However, because its dependencies just finished at the current time, 10h30, the algorithm cut down the duration to 30 minutes, in order for the task to still finish at 11:00 AM. Thus, we were able to get to the museum tour on time, without experiencing the domino effect of continuously being late to subsequent tasks because of one delayed task.

Tasks without fixed times and deadlines were scheduled based on their priority values (example: tasks 6 & 7 and tasks 9 & 10).

```
🕰t=8h30
        started 'Research about K-Economy' for 45 mins, with priority value = 540.7
        ✅ t=9h15, task completed!
❌ Task 'Get ready' missed the deadline at 9:00 AM!
🕰t=9h15
(Deadline: 9:00 AM)
        started 'Get ready' for 30 mins, with priority value = 360.7
        ✅ t=9h45, task completed!
🕰t=9h45
        started 'Bus to Seoul Museum of History' for 45 mins, with priority value = 540.7
        ✅ t=10h30, task completed!
🕰t=10h30
(Fixed Time: 10:20 AM)
        started 'Have breakfast with Sookmyung buddy' for 30 mins, with priority value = 480.7
        ✅ t=11h00, task completed!
🕰t=11h00
(Fixed Time: 11:00 AM)
        started 'Museum tour' for 60 mins, with priority value = 720.7
        ✅ t=12h00, task completed!
🕰t=12h00
        started 'Take a walk around Gyeongbokgung Palace' for 30 mins, with priority value = 360.7
        ✅ t=12h30, task completed!
🕰t=12h30
        started 'Take pictures' for 15 mins, with priority value = 180.7
        ✅ t=12h45, task completed!
🕰t=12h45
(Deadline: 4:00 PM)
        started 'Submit reflection and report' for 60 mins, with priority value = 720.7
        ✅ t=13h45, task completed!
🕰t=13h45
        started 'Get street food and practice Korean' for 45 mins, with priority value = 540.7
        ✅ t=14h30, task completed!
🕰t=14h30
        started 'Metro back to the residence' for 30 mins, with priority value = 360.7
        ✅ t=15h00, task completed!
🕰t=15h00
(Deadline: 6:00 PM)
        started 'Groceries with roommates' for 20 mins, with priority value = 240.7
        ✅ t=15h20, task completed!

🏁 Completed all planned tasks in 6h50.0min, with a total utility value of 5047.699999999999!
```

- Input 2:

The duration of task 3 is changed from 45 to 80 minutes (because there was a protest and multiple roads were blocked).

```
tasks = [
    Task(id=1, description='Research about K-Economy', duration=45, dependencies=[]),
    Task(id=2, description='Get ready', duration=30, dependencies=[1], deadline = '9:00 AM'),
    Task(id=3, description='Bus to Seoul Museum of History', duration=80, dependencies=[2]),
    Task(id=4, description='Have breakfast with Sookmyung buddy', duration=40, fixed_time='10:20 AM',
         dependencies=[2,3], people=[1]),
    Task(id=5, description='Museum tour', duration=60, fixed_time='11:00 AM', dependencies=[1,2,3,4]),
    Task(id=6, description='Take pictures', duration=15, dependencies=[1,2,3,4,5]),
    Task(id=7, description='Submit reflection and report', duration=60, dependencies=[1,2,3,4,5,6], deadline = '4:00 PM'),
    Task(id=8, description='Take a walk around Gyeongbokgung Palace', duration=30, dependencies=[1,2,3,4,5]),
    Task(id=9, description='Get street food and practice Korean', duration=45, dependencies=[1,2,3,4,5,6]),
    Task(id=10, description='Metro back to the residence', duration=30, dependencies=[1,2,3,4,5,6,8,9]),
    Task(id=11, description='Groceries with roommates', duration=20, dependencies=[1,2,3,4,5,6,7,8,9,10],
         people=[3], deadline = '6:00 PM')
]

scheduler = TaskScheduler()
scheduler.tasks = tasks  # Set the tasks attribute to your list of tasks
original_task_schedule = scheduler.run_task_scheduler(datetime.now().replace(hour=8, minute=30, second=0, microsecond=0))
```

- Output 2:

The original finish time for Task 4 “Have breakfast with Sookmying buddy” was supposed to be 11:00 AM (as calculated in output1).

The remaining till the original finish time = 11:00 AM - 11:05 AM = -5 minutes

Because the remaining time till the original finish time is less than 0, the scheduler skipped this task, marked it as completed, and moved on to Task 5.

Next, in Task 5 “Museum tour”, the original duration was 60 minutes but was reduced to 55 minutes because we started 5 minutes late (11h05 instead of 11h).

```
⏰t=8h30
        started 'Research about K-Economy' for 45 mins, with priority value = 540.7
        ✅ t=9h15, task completed!
❌ Task 'Get ready' missed the deadline at 9:00 AM!
⏰t=9h15
(Deadline: 9:00 AM)
        started 'Get ready' for 30 mins, with priority value = 360.7
        ✅ t=9h45, task completed!
⏰t=9h45
        started 'Bus to Seoul Museum of History' for 80 mins, with priority value = 960.7
        ✅ t=11h05, task completed!
⏰t=11h05
(Fixed Time: 10:20 AM)
        started 'Have breakfast with Sookmyung buddy' for 0 mins, with priority value = 480.7
        ✅ t=11h05, task completed!
⏰t=11h05
(Fixed Time: 11:00 AM)
        started 'Museum tour' for 55 mins, with priority value = 720.7
        ✅ t=12h00, task completed!
⏰t=12h00
        started 'Take a walk around Gyeongbokgung Palace' for 30 mins, with priority value = 360.7
        ✅ t=12h30, task completed!
⏰t=12h30
        started 'Take pictures' for 15 mins, with priority value = 180.7
        ✅ t=12h45, task completed!
⏰t=12h45
(Deadline: 4:00 PM)
        started 'Submit reflection and report' for 60 mins, with priority value = 720.7
        ✅ t=13h45, task completed!
⏰t=13h45
        started 'Get street food and practice Korean' for 45 mins, with priority value = 540.7
        ✅ t=14h30, task completed!
⏰t=14h30
        started 'Metro back to the residence' for 30 mins, with priority value = 360.7
        ✅ t=15h00, task completed!
⏰t=15h00
(Deadline: 6:00 PM)
        started 'Groceries with roommates' for 20 mins, with priority value = 240.7
        ✅ t=15h20, task completed!

🏁 Completed all planned tasks in 6h50.0min, with a total utility value of 5467.699999999999!
```

C. Reorder of input:

```
tasks = [
    Task(id=1, description='Research about K-Economy', duration=45, dependencies=[]),
    Task(id=2, description='Get ready', duration=30, dependencies=[1], deadline = '9:00 AM'),
    Task(id=3, description='Bus to Seoul Museum of History', duration=20, dependencies=[2]),
    Task(id=4, description='Have breakfast with Sookmyung buddy', duration=40, fixed_time='10:20 AM',
         dependencies=[2,3], people=[1]),
    Task(id=5, description='Museum tour', duration=60, fixed_time='11:00 AM', dependencies=[1,2,3,4]),
    Task(id=6, description='Take pictures', duration=15, dependencies=[1,2,3,4,5]),
    Task(id=7, description='Submit reflection and report', duration=60, dependencies=[1,2,3,4,5,6], deadline = '4:00 PM'),
    Task(id=8, description='Take a walk around Gyeongbokgung Palace', duration=30, dependencies=[1,2,3,4,5]),
    Task(id=9, description='Get street food and practice Korean', duration=45, dependencies=[1,2,3,4,5,6]),
    Task(id=10, description='Metro back to the residence', duration=30, dependencies=[1,2,3,4,5,6,8,9]),
    Task(id=11, description='Groceries with roommates', duration=20, dependencies=[1,2,3,4,5,6,7,8,9,10],
         people=[3], deadline = '6:00 PM')
]

tasks_reorder = [
    Task(id=1, description='Research about K-Economy', duration=45, dependencies=[]),
    Task(id=8, description='Take a walk around Gyeongbokgung Palace', duration=30, dependencies=[1,2,3,4,5]),
    Task(id=10, description='Metro back to the residence', duration=30, dependencies=[1,2,3,4,5,6,8,9]),
    Task(id=4, description='Have breakfast with Sookmyung buddy', duration=40, fixed_time='10:20 AM',
         dependencies=[2,3], people=[1]),
    Task(id=7, description='Submit reflection and report', duration=60, dependencies=[1,2,3,4,5,6], deadline='4:00 PM'),
    Task(id=2, description='Get ready', duration=30, dependencies=[1], deadline='9:00 AM'),
    Task(id=5, description='Museum tour', duration=60, fixed_time='11:00 AM', dependencies=[1,2,3,4]),
    Task(id=3, description='Bus to Seoul Museum of History', duration=20, dependencies=[2]),
    Task(id=9, description='Get street food and practice Korean', duration=45, dependencies=[1,2,3,4,5,6]),
    Task(id=6, description='Take pictures', duration=15, dependencies=[1,2,3,4,5]),
    Task(id=11, description='Groceries with roommates', duration=20, dependencies=[1,2,3,4,5,6,7,8,9,10],
         people=[3], deadline='6:00 PM')
]

scheduler = TaskScheduler()
scheduler.tasks = tasks  # Set the tasks attribute to your list of tasks
original_task_schedule = scheduler.run_task_scheduler(datetime.now().replace(hour=8, minute=30, second=0, microsecond=0))

scheduler2 = TaskScheduler()
scheduler2.tasks = tasks_reorder  # Set the tasks attribute to the reorder list of tasks
reordered_task_schedule = scheduler2.run_task_scheduler(datetime.now().replace(hour=8, minute=30, second=0, microsecond=0))

assert original_task_schedule == reordered_task_schedule
```

IV. Algorithm Analysis.

A. Benefits and Strengths:

- Minimizes idle time: The scheduler is designed to schedule tasks that minimize gaps between them. It ensures that tasks dependent on others are scheduled to start as soon as their dependencies are completed, given their priority value. This increases efficiency and maximizes the time in a day.

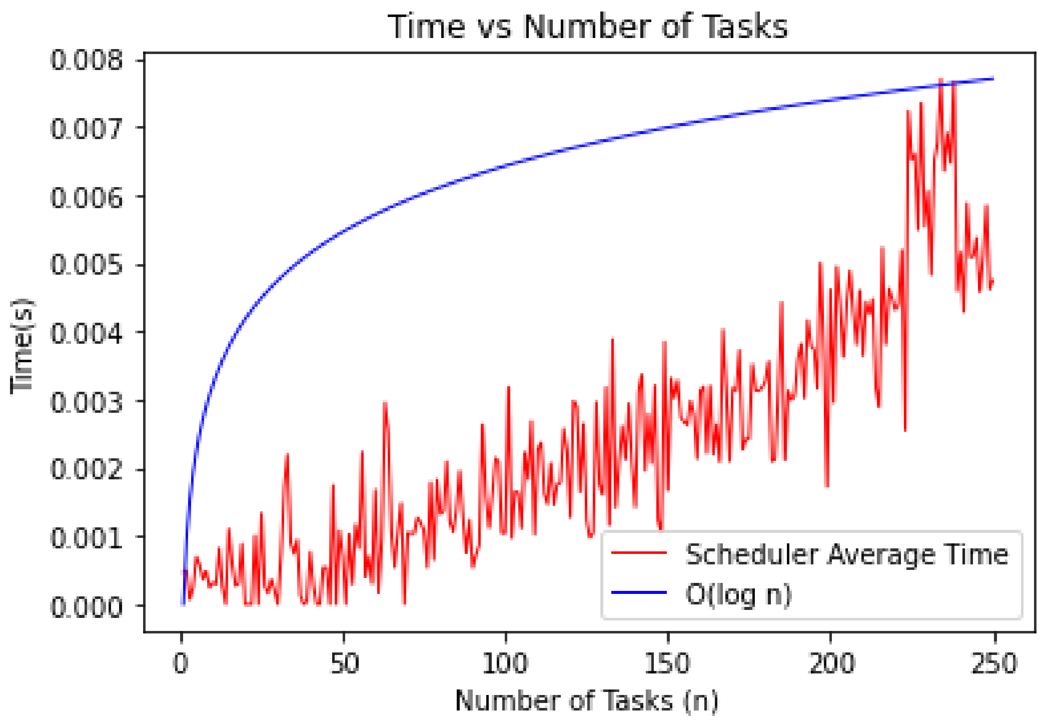
- Prioritization: Based on the priority value that was computed based on a justified formula, the scheduler effectively prioritizes tasks based on their dependencies, duration, and the number of people involved. This ensures that important tasks are addressed first.
- Fixed time considerations: For tasks with fixed time, they will always be scheduled at that time, regardless of the priority value. This makes sense because some tasks, though giving me low utility, still have to be done at the planned time because they involve other people or are based on a rule/ policy.
- Deadline considerations: The scheduler prints a statement if the deadline has been missed (because of dependencies, etc). It reschedules if it notices that there is not enough time to complete the task if started at the current time. Otherwise, it schedules the task to start at the earliest. This ensures the deadlines are met as it is one of the most important factors while planning a day.

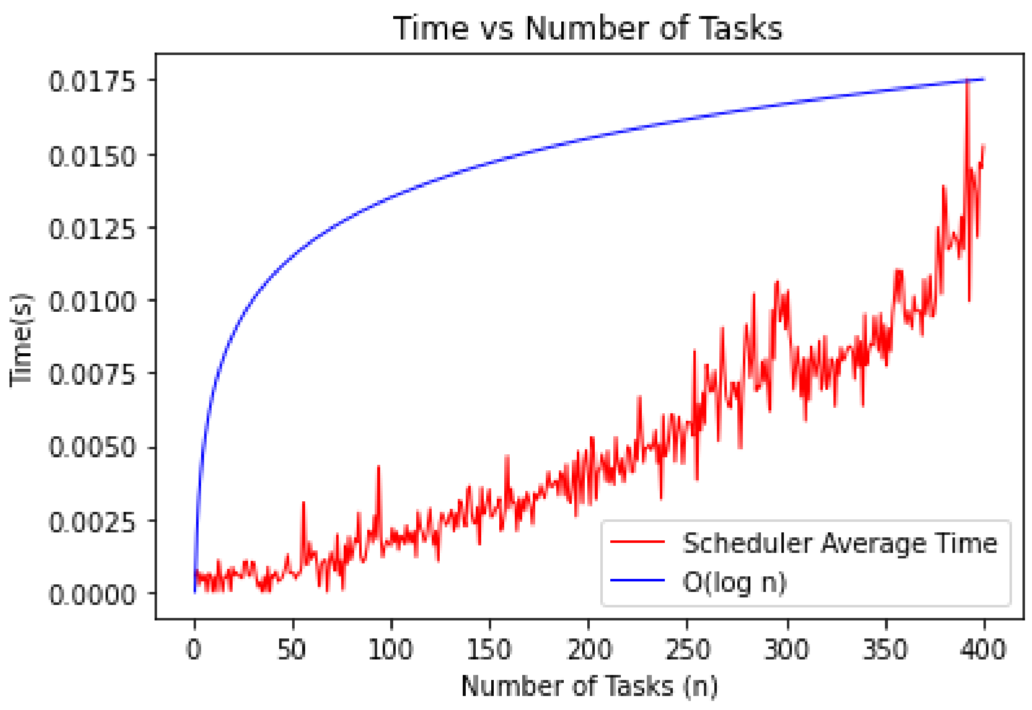
B. Limitations:

Limitation	Description	Potential solution
Multitasking	For example, one can take pictures while walking around Gyeongbokgung Palace.	Boolean of true/ false whether a task can be multitasked. Tasks that have the value of "true" can be merged into a single task and the new priority value would be the average of their original values.
Splitting tasks	Some tasks can be split and done in noncontinuous time slots, such as researching about Korean economy for the tour. However, the schedule assumes that all	Allow tasks to be divided into sub-tasks with their own durations. The algorithm will subtract the duration completed to get the remaining time.

	tasks must be completed at once, given the duration.	
No break consideration	While this can be efficient, it does not account for breaks, unexpected delays, or traveling time. If one activity takes longer than planned/ or the user cannot continuously work, it could affect subsequent tasks, and the scheduler is currently leaving very little room for those.	Implement and add a break task at regular intervals or, **Make the scheduler interactive**. We can build a function that frequently asks for information from the user to know whether to reschedule a task. The schedule should be able to accept streaming data throughout the day instead of fixed data at the beginning. This solution can be beneficial to solve all the limitations listed. Especially, given how a priority queue is well-suited for handling streaming data because of its efficient insertion and deletion operations
Conflicting events	If one of the people in the task changes one's schedule, that task would need to be rescheduled and may overlap with another task.	
Limited visual representation	The scheduler provides detailed output, but the visual representation can be improved to make it easier to follow/ reschedule.	A calendar format with time frames, and different colors of different statuses, or even drag-and-drop functionality for users to manually adjust task schedules.

C. Complexity analysis:





Graph:

The number of tasks (n) is set to 250 and 400, and the algorithm iterates 15 times for each input size. A list of tasks is generated each time, with attributes of ID, description, duration, dependencies, fixed time, and deadline. The ‘run_task_scheduler’ method is called to execute. The start and end times are recorded before and after running the task scheduling method for each iteration. The ‘scheduler_time list’ stores the time taken for all iterations. To calculate the average execution time, the algorithm sums up the individual execution times for each iteration and divides this total by the number of iterations (15, which is a large enough number). This returns the scheduler's average time, measured in seconds. The blue line represents the log2() of the number of inputs as a reference point for O(log n) scaling.

A graph to calculate the average run time over iterations rather than a single measurement was specifically chosen to reduce outliers, produce an assessment of how the algorithm behaves on average, and smooth out variations (system load, environment influences), resulting in more reliable and realistic insights.

Theoretical analysis:

- Priority queue: Operations of a heap data structure often has a logarithmic time complexity for inserting a task or removing the root task into/ from the priority queue, O(log n), using heappush and heappop.
- Calculating priority value: This takes O(1) because it takes a constant time to calculate and return a value for every task.

- Constraints (fixed time and deadlines): Checking a task has a constraint and comparing it with the current time is a constant-time operation, as well as calculating the time difference or the remaining time. O(n), where n is the number of tasks with constraints.
- Thus, theoretically, this scheduling algorithm should take O(log n).

Empirical analysis:

- Based on the graph, the algorithm's running time seems to scale linearly O(n) as the input size (number of tasks) increases.
- The operations handling fixed time and deadline constraints are a dominant factor in the algorithm's running time, which takes up O(n) time complexity. Scanning and resolving dependencies may also involve linear operations.
- While constraints, dependencies, and other factors may have linear aspects, they are secondary to the core operations on the priority queue, which is the most time-consuming operation in this algorithm, O(log n).

V. My thought process.

🙂 Loom Video

VI. Appendices.

Part I: LO and HC applications

1. #ComputationalCritique

The priority value was based on various factors with their own weights and the choices were justified. The constraints of fixed time and deadlines were taken into consideration. The critique analyzed the algorithm’s advantages as well as how the current version fails to overcome some limitations such as multitasking, breaks, and unexpected delays, as well as suggesting potential solutions to solve them.

Word count: 57

2. **#ComplexityAnalysis**

Improving from the last assignment, I specified what the input size is in this case and included the “average scheduling time vs number of tasks” graph with n = 250 and 400 (number of tasks). Theoretical and empirical analyses were both examined, with justification from the code and insights from the graph. Although I could not justify why the empirical results differed from the theoretical ones, I explained my thought process with evidence.

Word count: 72

3. **#CodeReadability**

Variable names are meaningful while being concise and precise. The comments provide insights into the logic and purpose of important lines of code. The docstrings ensure that collaborators can understand the purpose and usage of different classes/ methods. I learned from the last assignment to stay consistent with my comments, instead of including a lot of comments for a few code blocks and leaving some ambiguous.

Word count: 65

4. **#PythonProgramming**

The code employs an object-oriented programming approach with well-organized classes (MaxHeap, Task, and Scheduler). I also included 3 test cases for the MaxHeap class to ensure that the code works properly. The test cases for the scheduler include changing the task duration of different tasks and examining how the algorithm adjusts subsequent tasks. I also switched the order of tasks and included an assert statement to make sure the algorithm works based on the priority queue and constraints instead of the task ID. All tests passed and the tasks were scheduled as expected.

Word count: 93

5. **#AlgoStratDataStruct**

My code utilizes the advantages of a priority queue for effective task scheduling, I justified the choice of a MaxHeap and a priority queue by highlighting their benefits of handling tasks with dependencies, frequent data updates, as well as efficient insertion and deletion operations. The MaxHeap class forms the foundation for managing task priorities. The Task class presents individual tasks with various attributes, while the Scheduler class executes the scheduling process. This object-oriented approach enhances the maintainability and reusability of code.

Word count: 81

6. **#constraints:**

The two constraints (fixed time and deadline) were considered in multiple cases (on time, late) and how to satisfy them in each case (reschedule, skip, cut down duration). Not only were the constraints defined, they were satisfied in multiple ways which helped guide the scheduling algorithm logic, besides the priority queue.

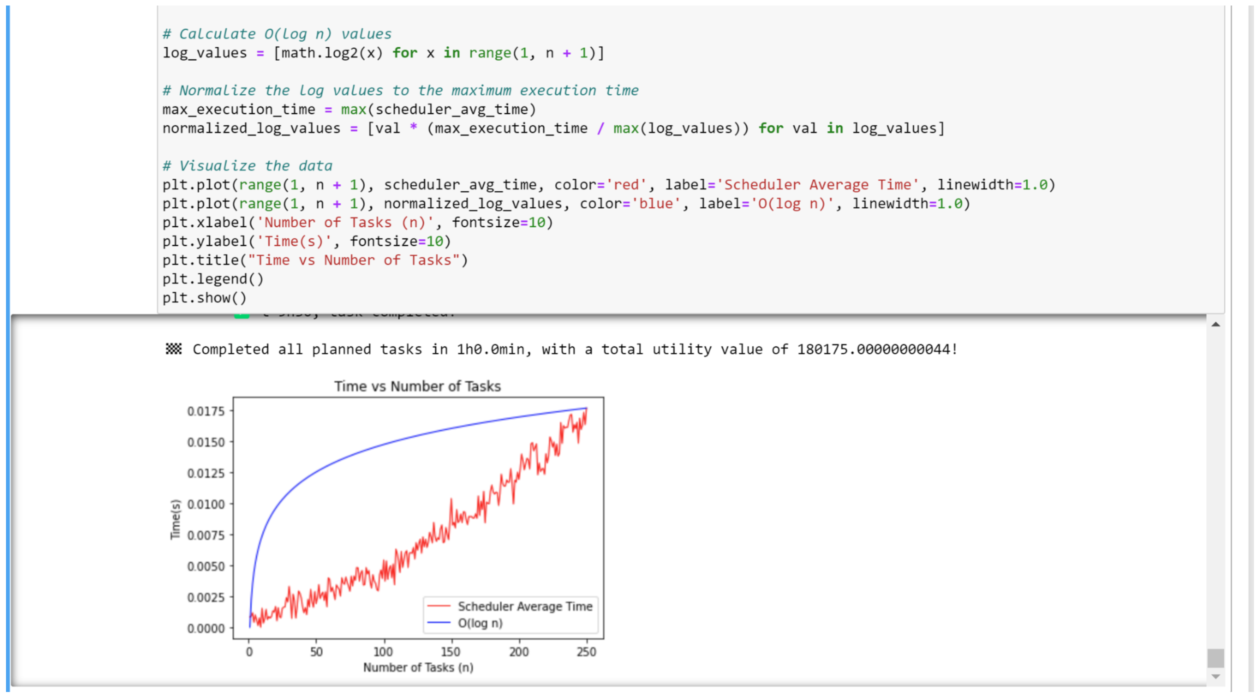
Word count: 51

Part II: AI Statement

I wanted to calculate the time difference and learned about datetime.timedelta through this article. Then, I used ChatGPT to help me implement it for the deadline function. However, all of the formulas/ calculation logic were created and justified by me. Other than that, all my work was based on Session 13 pre-class work/ breakout code and my implementation.

Part III: Python code

I could not include the picture of the graph in the markdown file, so I'm inserting a screenshot here. I hope this is fine.



```python
# Calculate O(log n) values
log_values = [math.log2(x) for x in range(1, n + 1)]

# Normalize the log values to the maximum execution time
max_execution_time = max(scheduler_avg_time)
normalized_log_values = [val * (max_execution_time / max(log_values)) for val in log_values]

# Visualize the data
plt.plot(range(1, n + 1), scheduler_avg_time, color='red', label='Scheduler Average Time', linewidth=1.0)
plt.plot(range(1, n + 1), normalized_log_values, color='blue', label='O(log n)', linewidth=1.0)
plt.xlabel('Number of Tasks (n)', fontsize=10)
plt.ylabel('Time(s)', fontsize=10)
plt.title("Time vs Number of Tasks")
plt.legend()
plt.show()
```

```
🏁 Completed all planned tasks in 1h0.0min, with a total utility value of 180175.00000000044!
```

```
from datetime import datetime, timedelta

#adopted this class from Session 13 preclass-work
class MaxHeapq:
    """
    A class to represent a max-heap

    Attributes
    ----------
    heap: arr
        A Python list where key values in the max heap are stored
    length: int
        the number of keys present in the max heap

    Methods:
    ----------
    left(i):
        Returns the index of the left child of the input index, if exists
    right(i):
        Returns the index of the right child of the input index, if exists
    parent(i):
        Returns the index of the parent node of the input index, if exists
    heappush(key):
        Adds an element to the heap
    heappop():
        Removes the root node of the heap
    increase_key(i, key):
        Increase the priority of the ith element in the max heap
    heapify(i):
        Puts the i-th element in the heap to one of its correct position
    """

    def __init__(self):
        """
        Initiate the attributes for the max-heap

        Paremeters
        ---------
        None
        """
        self.heap =  [] #defining the heap list
        self.heap_size = 0 #defining the heap size
    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(key)
        self.heap_size += 1
        self.increase_key(self.heap_size - 1, key)


    def heappop(self):
        """
        Returns the larest key in the max priority queue
        and remove it from the max priority queue

        Parameters
        ----------
        None

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

    def increase_key(self, i, key):
        """
        Modifies the value of a key in a max priority queue
        with a higher value

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

        Returns
        ----------
        None
        """
        while i > 0 and key > self.heap[self.parent(i)]:
            self.heap[i] = self.heap[self.parent(i)]
            i = self.parent(i)
        self.heap[i] = key
    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

        largest = i
        if l <= (self.heap_size - 1) and heap[l] > heap[i]:
            largest = l
        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)

# A and B were retrieved from pre-class work
A = [4, 3, 6, 8, 2, -5, 100]
my_heap = MaxHeapq()
[my_heap.heappush(k) for k in A]
assert(my_heap.heap == [100, 6, 8, 3, 2, -5, 4])

B = [6,4,7,9,10,-5,-6,12,8,3,1,-10]
her_heap = MaxHeapq()
[her_heap.heappush(k) for k in B]
assert(her_heap.heap == [12, 10, 6, 9, 7, -5, -6, 4, 8, 3, 1, -10])

C = [8, 9, 10, 45, -5, 123, 7]
our_heap = MaxHeapq()
[our_heap.heappush(k) for k in C]
assert our_heap.heap == [123, 10, 45, 8, -5, 9, 7]
```

```
class Task:
    """
    A class to represent the Task object

    Parameters
    -----------
    _id: Any
        An identification element
    description: str
        A string describing the task
    duration: int
        The time in minutes the activity takes
    dependencies: list
        A list of other tasks the current task depends on
    status: str
        The status of the task
    fixed_time: str
        The fixed time that the task must be executed at
    deadline: str
        The time that the task must be completed by
    people: int
        The number of people involve in the task

    Attributes
    ----------
    priority_value:int
        A number representing the tasks priority rank among other tasks

    Methods
    ----------
    calculate_priority_value(current_time)
        Calculate the priority value of a task based on its dependencies, duration,
    change_status(new_status)
        Set a new status of a task
    """

    def __init__(self, id, description, duration, dependencies, status="not_yet_sta
        """
        Initializes the class variables
        """
        self.id = id
        self.description = description
        self.duration = duration
        self.dependencies = dependencies
        self.status = status
        self.fixed_time = fixed_time # Add a 'fixed time' attribute
        self.deadline = deadline  # Add a 'deadline' attribute
        self.people = people # Add a 'people' attribute
        self.priority_value = None  # Initialize priority value as None


    def calculate_priority_value (self, current_time):
        """
        Calculate the priority value of a task based on its dependencies, duration,

        Parameters
        ----------
        current_time: datetime
            The current time to consider when calculating priority.
        """
        task_dependencies = len(self.dependencies)
        task_duration = timedelta(minutes=self.duration)
        task_people = len(self.people) if self.people else 1  # If people is not pr

        # Formula to calculate priority value
        self.priority_value = (0.5 * task_dependencies) + (0.2 * task_duration.tota

    def __gt__(self, other):
        """
        Compare two tasks and prioritize tasks by ID if they have the same priority

        Parameters
        ----------
        other: Task
            Another task to compare against

        Returns
        -------
        bool
            True if this task has higher priority than the other task, False otherw
        """
        # Check if the the tasks have the same priority value:
        if self.priority_value == other.priority_value:
            # Compare and prioritize by IDs
            return self.id > other.id
        return self.priority_value > other.priority_value


    def change_status(self, new_status):
        """
        Set a new status of an event
        """
        self.status = new_status



class TaskScheduler:
    """
    Class object representing the Scheduler. Inherited from MaxHeap class.

    Attributes
    ----------
    tasks:lst
        A list that stores the tasks to be scheduled
    priority_queue:
        An instance of the MaxHeap class, representing the priority queue
        to manage tasks based on their priority value

    Methods
    ----------
    __init__():
        Initialize the Scheduler class
    remove_dependency(id):
        Remove a task's dependencies with given id
    get_tasks_ready(current_time):
        Identify tasks that are ready to be scheduled
    check_unscheduled_tasks():
        Check if there are unscheduled tasks
    format_time(time):
        Formate the time as a string in 'hh:mm' format
    run_task_scheduler(starting_time):
        Execute the scheduler to schedule tasks given fixed time and deadlines and
    """

    NOT_STARTED = 'not_yet_started'
    IN_PRIORITY_QUEUE = 'in_priority_queue'
    COMPLETED = 'completed'
    def __init__(self):
        """
        Initialize the Scheduler class
        """
        self.tasks = []  # Initialize with an empty list of tasks
        self.priority_queue = MaxHeapq() # An attribute as a instance of the MaxHea

    def remove_dependency(self, id):
        """
        Remove a task's dependencies with the given ID
        """
        for task in self.tasks:
            if task.id != id and id in task.dependencies:
                task.dependencies.remove(id)

    def get_tasks_ready(self, current_time):
        """
        Identify tasks that are ready to be scheduled
        """
        for task in self.tasks:

            # Check if the task is not yet started and all its dependencies have be
            if task.status == self.NOT_STARTED and all(dependency.status == self.CC

                # Calculates the priorty value of the task
                task.calculate_priority_value(current_time)

                # Changes the task's status to "in priority queue"
                task.status = self.IN_PRIORITY_QUEUE

                # Pushes the task into the priorty queue
                self.priority_queue.heappush(task)

    def check_unscheduled_tasks(self):
        """
        Check if there are unscheduled tasks
        """
        return any(task.status == self.NOT_STARTED for task in self.tasks)

    def format_time(self, time):
        """
        Formate the time as a string in 'hh:mm' format
        Parameters:
        ----------
        time: datetime
            the time to format

        Returns:
        str: the formatted time
        """
        hours, minutes = time.hour, time.minute
        return f"{hours}h{minutes:02d}"


    def run_task_scheduler(self, starting_time):
        """
        Execute the scheduler to schedule tasks given fixed time and deadlines
        """
        current_time = starting_time
        total_time = timedelta(0) # Intitalizes the total time as a timedelta objec

        for task in self.tasks:
            task.status = self.NOT_STARTED

        while self.check_unscheduled_tasks():
            # Find tasks that are ready to be scheduled at the current time
            self.get_tasks_ready(current_time)

            if self.priority_queue and self.priority_queue.heap_size > 0:
                # Get the task with the highest priority value from the priority qu
                task = self.priority_queue.heappop()
                task.calculate_priority_value(current_time)


                if task.fixed_time:
                    scheduled_time = datetime.strptime(task.fixed_time, '%I:%M %p')
                    fixed_time = datetime.strptime(task.fixed_time, '%I:%M %p')

                    if current_time > scheduled_time:
                        scheduled_time = current_time
                        current_date = current_time.date()
                        # Ensure both times are on the same date
                        scheduled_time = scheduled_time.replace(year=current_date.y
                        fixed_time = fixed_time.replace(year=current_date.year, mor
                        # Calculate the new duration to finish on time
                        original_end_time = fixed_time + timedelta(minutes=task.dur
                        remaining_time = original_end_time - current_time
                        if remaining_time.total_seconds() < 0:
                            # Skip the task
                            task.duration = 0
                        else:
                            task.duration = int(remaining_time.total_seconds() / 6

                else:
                    scheduled_time = current_time


                if task.deadline:
                    current_date = current_time.date()
                    task_deadline = datetime.strptime(task.deadline, '%I:%M %p')
                    task_deadline = task_deadline.replace(year=current_date.year, m
                    time_to_deadline = task_deadline - current_time
                    if time_to_deadline.total_seconds() < 0:
                        # The deadline has passed, it is marked as missed
                        if task.status != self.COMPLETED:
                            print(f"❌ Task '{task.description}' missed the deadli
                            task.status = self.COMPLETED
                    elif time_to_deadline.total_seconds() < task.duration * 60:
                        # Not enough time to complete the task before the deadline;
                        scheduled_time = task_deadline - timedelta(minutes=task.dur
                    else:
                        # Schedule the task to start as soon as possible
                        if current_time < task_deadline - timedelta(minutes=task.du
                            scheduled_time = current_time

                current_time = scheduled_time

                print(f"🕰t={self.format_time(current_time)}")
                if task.deadline:
                    print(f"(Deadline: {task.deadline})")
                if task.fixed_time:
                    print(f"(Fixed Time: {task.fixed_time})")

                print(f"\tstarted '{task.description}' for {task.duration} mins, wi

                current_time += timedelta(minutes=task.duration)
                total_time += timedelta(minutes=task.duration)
```

```
                print(f"\t✅ t={self.format_time(current_time)}, task completed!")
                self.remove_dependency(task.id)
                task.status = self.COMPLETED

        total_seconds = total_time.total_seconds()

        # Calculate hours and remaining seconds
        hours = total_seconds // 3600
        remaining_seconds = total_seconds % 3600

        # Convert the remaining seconds to minutes
        minutes = remaining_seconds // 60

        total_priority_value = 0
        for task in self.tasks:
            total_priority_value += task.priority_value

        print(f"\n🏁 Completed all planned tasks in {int(hours)}h{minutes}min, witl



tasks = [
    Task(id=1, description='Research about K-Economy', duration=45, dependencies=[]
    Task(id=2, description='Get ready', duration=30, dependencies=[1], deadline = '
    Task(id=3, description='Bus to Seoul Museum of History', duration=80, dependenc
    Task(id=4, description='Have breakfast with Sookmyung buddy', duration=40, fixe
         dependencies=[2,3], people=[1]),
    Task(id=5, description='Museum tour', duration=60, fixed_time='11:00 AM', deper
    Task(id=6, description='Take pictures', duration=15, dependencies=[1,2,3,4,5]),
    Task(id=7, description='Submit reflection and report', duration=60, dependencie
    Task(id=8, description='Take a walk around Gyeongbokgung Palace', duration=30,
    Task(id=9, description='Get street food and practice Korean', duration=45, depe
    Task(id=10, description='Metro back to the residence', duration=30, dependencie
    Task(id=11, description='Groceries with roommates', duration=20, dependencies=[
         people=[3], deadline = '6:00 PM')
]

scheduler = TaskScheduler()
scheduler.tasks = tasks  # Set the tasks attribute to your list of tasks
original_task_schedule = scheduler.run_task_scheduler(datetime.now().replace(hour=8
⏰t=8h30
        started 'Research about K-Economy' for 45 mins, with priority value = 540.7
        ✅ t=9h15, task completed!
❌ Task 'Get ready' missed the deadline at 9:00 AM!
⏰t=9h15
(Deadline: 9:00 AM)
        started 'Get ready' for 30 mins, with priority value = 360.7
        ✅ t=9h45, task completed!
⏰t=9h45
        started 'Bus to Seoul Museum of History' for 80 mins, with priority value =
        ✅ t=11h05, task completed!
⏰t=11h05
(Fixed Time: 10:20 AM)
        started 'Have breakfast with Sookmyung buddy' for 0 mins, with priority val
        ✅ t=11h05, task completed!
⏰t=11h05
(Fixed Time: 11:00 AM)
        started 'Museum tour' for 55 mins, with priority value = 720.7
        ✅ t=12h00, task completed!
⏰t=12h00
        started 'Take a walk around Gyeongbokgung Palace' for 30 mins, with priorit
        ✅ t=12h30, task completed!
⏰t=12h30
        started 'Take pictures' for 15 mins, with priority value = 180.7
        ✅ t=12h45, task completed!
⏰t=12h45
(Deadline: 4:00 PM)
        started 'Submit reflection and report' for 60 mins, with priority value = 7
        ✅ t=13h45, task completed!
⏰t=13h45
        started 'Get street food and practice Korean' for 45 mins, with priority va
        ✅ t=14h30, task completed!
⏰t=14h30
        started 'Metro back to the residence' for 30 mins, with priority value = 36
        ✅ t=15h00, task completed!
⏰t=15h00
(Deadline: 6:00 PM)
        started 'Groceries with roommates' for 20 mins, with priority value = 240.7
        ✅ t=15h20, task completed!

🏁 Completed all planned tasks in 6h50.0min, with a total utility value of 5467.69
```

```
tasks = [
    Task(id=1, description='Research about K-Economy', duration=45, dependencies=[]
    Task(id=2, description='Get ready', duration=30, dependencies=[1], deadline = '
    Task(id=3, description='Bus to Seoul Museum of History', duration=20, dependenc
    Task(id=4, description='Have breakfast with Sookmyung buddy', duration=40, fixe
         dependencies=[2,3], people=[1]),
    Task(id=5, description='Museum tour', duration=60, fixed_time='11:00 AM', deper
    Task(id=6, description='Take pictures', duration=15, dependencies=[1,2,3,4,5]),
    Task(id=7, description='Submit reflection and report', duration=60, dependencie
    Task(id=8, description='Take a walk around Gyeongbokgung Palace', duration=30,
    Task(id=9, description='Get street food and practice Korean', duration=45, depe
    Task(id=10, description='Metro back to the residence', duration=30, dependencie
    Task(id=11, description='Groceries with roommates', duration=20, dependencies=[
         people=[3], deadline = '6:00 PM')
]

tasks_reorder = [
    Task(id=1, description='Research about K-Economy', duration=45, dependencies=[]
    Task(id=8, description='Take a walk around Gyeongbokgung Palace', duration=30,
    Task(id=10, description='Metro back to the residence', duration=30, dependencie
    Task(id=4, description='Have breakfast with Sookmyung buddy', duration=40, fixe
         dependencies=[2,3], people=[1]),
    Task(id=7, description='Submit reflection and report', duration=60, dependencie
    Task(id=2, description='Get ready', duration=30, dependencies=[1], deadline='9:
    Task(id=5, description='Museum tour', duration=60, fixed_time='11:00 AM', deper
    Task(id=3, description='Bus to Seoul Museum of History', duration=20, dependenc
    Task(id=9, description='Get street food and practice Korean', duration=45, depe
    Task(id=6, description='Take pictures', duration=15, dependencies=[1,2,3,4,5]),
    Task(id=11, description='Groceries with roommates', duration=20, dependencies=[
         people=[3], deadline='6:00 PM')
]

scheduler = TaskScheduler()
scheduler.tasks = tasks  # Set the tasks attribute to your list of tasks
original_task_schedule = scheduler.run_task_scheduler(datetime.now().replace(hour=8

scheduler2 = TaskScheduler()
scheduler2.tasks = tasks_reorder  # Set the tasks attribute to the reorder list of
reordered_task_schedule = scheduler2.run_task_scheduler(datetime.now().replace(hour

assert original_task_schedule == reordered_task_schedule
```

```
🕰t=8h30
        started 'Research about K-Economy' for 45 mins, with priority value = 540.7
            ✅ t=9h15, task completed!
❌ Task 'Get ready' missed the deadline at 9:00 AM!
🕰t=9h15
(Deadline: 9:00 AM)
        started 'Get ready' for 30 mins, with priority value = 360.7
            ✅ t=9h45, task completed!
🕰t=9h45
        started 'Bus to Seoul Museum of History' for 20 mins, with priority value =
            ✅ t=10h05, task completed!
🕰t=10h05
(Fixed Time: 10:20 AM)
        started 'Have breakfast with Sookmyung buddy' for 55 mins, with priority va
            ✅ t=11h00, task completed!
🕰t=11h00
(Fixed Time: 11:00 AM)
        started 'Museum tour' for 60 mins, with priority value = 720.7
            ✅ t=12h00, task completed!
🕰t=12h00
        started 'Take a walk around Gyeongbokgung Palace' for 30 mins, with priorit
            ✅ t=12h30, task completed!
🕰t=12h30
        started 'Take pictures' for 15 mins, with priority value = 180.7
            ✅ t=12h45, task completed!
🕰t=12h45
(Deadline: 4:00 PM)
        started 'Submit reflection and report' for 60 mins, with priority value = 7
            ✅ t=13h45, task completed!
🕰t=13h45
        started 'Get street food and practice Korean' for 45 mins, with priority va
            ✅ t=14h30, task completed!
🕰t=14h30
        started 'Metro back to the residence' for 30 mins, with priority value = 36
            ✅ t=15h00, task completed!
🕰t=15h00
(Deadline: 6:00 PM)
        started 'Groceries with roommates' for 20 mins, with priority value = 240.7
            ✅ t=15h20, task completed!

🏁 Completed all planned tasks in 6h50.0min, with a total utility value of 4747.69
🕰t=8h30
```

```
        started 'Research about K-Economy' for 45 mins, with priority value = 540.7
        ✅ t=9h15, task completed!
❌ Task 'Get ready' missed the deadline at 9:00 AM!
🕰t=9h15
(Deadline: 9:00 AM)
        started 'Get ready' for 30 mins, with priority value = 360.7
        ✅ t=9h45, task completed!
🕰t=9h45
        started 'Bus to Seoul Museum of History' for 20 mins, with priority value =
        ✅ t=10h05, task completed!
🕰t=10h05
(Fixed Time: 10:20 AM)
        started 'Have breakfast with Sookmyung buddy' for 55 mins, with priority va
        ✅ t=11h00, task completed!
🕰t=11h00
(Fixed Time: 11:00 AM)
        started 'Museum tour' for 60 mins, with priority value = 720.7
        ✅ t=12h00, task completed!
🕰t=12h00
        started 'Take a walk around Gyeongbokgung Palace' for 30 mins, with priorit
        ✅ t=12h30, task completed!
🕰t=12h30
        started 'Take pictures' for 15 mins, with priority value = 180.7
        ✅ t=12h45, task completed!
🕰t=12h45
(Deadline: 4:00 PM)
        started 'Submit reflection and report' for 60 mins, with priority value = 7
        ✅ t=13h45, task completed!
🕰t=13h45
        started 'Get street food and practice Korean' for 45 mins, with priority va
        ✅ t=14h30, task completed!
🕰t=14h30
        started 'Metro back to the residence' for 30 mins, with priority value = 36
        ✅ t=15h00, task completed!
🕰t=15h00
(Deadline: 6:00 PM)
        started 'Groceries with roommates' for 20 mins, with priority value = 240.7
        ✅ t=15h20, task completed!

🏁 Completed all planned tasks in 6h50.0min, with a total utility value of 4747.69
```

```
import random
import numpy as np
from datetime import datetime
import time
from matplotlib import pyplot as plt
import math  # Import the math module

def task_generator(id=None, description=None, duration=60, dependencies=[],fixed_ti
    """
    Generate a task, with defualt attributes provided wehre necessary.
    Parameters
    -----------
    _id:Any
    An identification element of the task
    description:str
    A string describing the task
    duration:int
    The time in minutes the activity takes
    dependencies:list
    18
    A list of other tasks the current task depends on
    utility:int
    The utility of the task
    Return
    ------
    Task
    A `Task` object
    """
    #generate random id if not provided
    if id is None:
        id = random.randint(1000000, 10000000)
    #generate description based on id if description is not provided
    if description is None:
        description = "Task {id}".format(id=id)
    return Task(id, description, duration, dependencies,fixed_time,deadline)

random.seed(42)

n = 250  # Maximum number of tasks to generate
iterations = 15  # Number of iterations for each input size

# Create tasks
```

```
tasks = []
for i in range(n):
    dependencies = []
    if i > 5:
        dependencies = random.sample(tasks, 3)
    task = task_generator(id=i, dependencies=dependencies,)  # Corrected the functi
    tasks.append(task)


# Create an empty list to store the execution times for different input sizes
scheduler_avg_time = []

# Iterate n times, increasing tasks size and timing the scheduler run method runtim
for n in range(1, n + 1):
    scheduler_time = []
    for trial in range(iterations):
        # Create tasks with controlled input size (1 to n)
        tasks = [task_generator(id=i, dependencies=[]) for i in range(1, n + 1)]

        scheduler = TaskScheduler()
        scheduler.tasks = tasks  # Set the tasks attribute to the generated tasks

        start_time = time.time()
        scheduler.run_task_scheduler(datetime.now().replace(hour=8, minute=30, secc
        end_time = time.time()

        scheduler_time.append(end_time - start_time)

    # Calculate the average execution time for the current input size
    scheduler_avg_time.append(np.sum(scheduler_time) / iterations)

# Calculate O(log n) values
log_values = [math.log2(x) for x in range(1, n + 1)]

# Normalize the log values to the maximum execution time
max_execution_time = max(scheduler_avg_time)
normalized_log_values = [val * (max_execution_time / max(log_values)) for val in lc

# Visualize the data
plt.plot(range(1, n + 1), scheduler_avg_time, color='red', label='Scheduler Average
plt.plot(range(1, n + 1), normalized_log_values, color='blue', label='O(log n)', li
plt.xlabel('Number of Tasks (n)', fontsize=10)
plt.ylabel('Time(s)', fontsize=10)
```

```
    plt.title("Time vs Number of Tasks")
    plt.legend()
    plt.show()
```