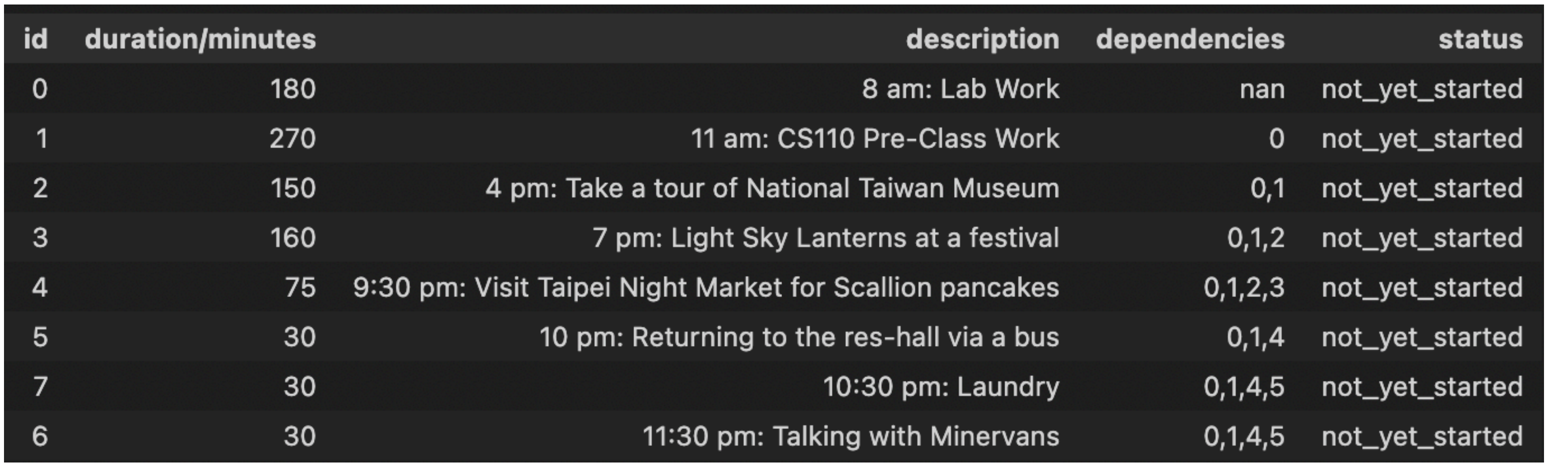
Setting up

A: Tasks



id	duration/minutes	description	dependencies	status
0	180	8 am: Lab Work	nan	not_yet_started
1	270	11 am: CS110 Pre-Class Work	0	not_yet_started
2	150	4 pm: Take a tour of National Taiwan Museum	0,1	not_yet_started
3	160	7 pm: Light Sky Lanterns at a festival	0,1,2	not_yet_started
4	75	9:30 pm: Visit Taipei Night Market for Scallion pancakes	0,1,2,3	not_yet_started
5	30	10 pm: Returning to the res-hall via a bus	0,1,4	not_yet_started
7	30	10:30 pm: Laundry	0,1,4,5	not_yet_started
6	30	11:30 pm: Talking with Minervans	0,1,4,5	not_yet_started

Fig.1 shows the tasks that we need to schedule. Each task is assigned an id, the duration it takes to complete the task in minutes, the description of the task, dependencies (the tasks need to be completed before this task), and status showing whether it is complete, in progress, or incomplete.

B: Task descriptions

The following tasks are imperative for my day-to-day life as they are the tasks that I struggle to stay productive through or scheduling them prevents any delays or missing deadlines to accomplish them. For instance, I have to go to Lab five times a day or socialize with my class fellows which as an introvert I would not do unless it is scheduled for me. My academic tasks like Pre-class work are also mandatory. Here, by Pre-class work, I also refer to the class I have to take and I am reducing two tasks that are interconnected together as I like to do Pre-class work on the same day as my class to keep the content fresh. Moreover, my cultural activities include things that can be done in a day like visiting a national museum to check out the cultural clothes and local animals that I can find. I also learned quite a bit about the influence of Japanese culture on Taiwan which is more than Chinese culture. Lastly, attending local festivals just so I can dance or light a sky lantern to make a wish which highlights the cultural integration I am trying to experience in Taipei. Each festival offers different food and an opportunity to visit different night markets to try local cuisine.

Preparing your algorithmic strategy

A:

Priority queues are well suited for scheduling tasks as using them we can assign particular priorities or weight to our tasks which makes it easier to pull out the task that we need to accomplish first or in specific time intervals. It is especially helpful as it provides access to the largest element in our queue which means it can provide more efficiency as getting the highest element has a time complexity of O(1) for tasks that are comparable (meaning in a given metric like the time one can be prioritized over the other).

We do use priority queue over sorting functions as it gives better average performance than sorting such as insertion sort. The reason is that when we need to add a new element to the list that needs to be sorted we need to compare the new element with all the elements in the list in the worst-case but for priority queues based on the priority value we assign the element will just move to its appropriate after the element which has a greater priority value than this new element.

Coming back to scheduling tasks, we might have tasks that are fixed or continuous. For this, we need to make separate instances of priority queues that deal with these categories of tasks. For instance, for my tasks like laundry and talking with Minervans, I can do both tasks at the same time so there should be instances that allow for parallel scheduling as people multi-task, and if the opportunity presents we can even eat food, and do class preparation. Another aspect can be a conflict between fixed time-constrained tasks like taking a class or an office hour, maybe a city event hosted by Minerva. For these tasks, it is important to set instances where once they are called they cannot be ignored or sent back to the priority queue as we need to understand that even when we schedule tasks some things are unpredictable like not being able to find a bus and so couldn't socialize with Minervans but things like taking a class are mandatory. Therefore, by creating separate instances we will be able to send back or readjust the priority value for flexible tasks while ensuring time-specific tasks are not affected.

B.

To implement a priority queue via python, we need to consider two main things: first, the constraint that we can only remove tasks from our box of tasks based on which is the most important for us or important at that time, and second there could be cases when there is more than one task that needs to be done at the same time so we need to pick a task based on which one we first placed in the box e.g. Tasks [wake up, run, listen to music, read a book, sleep]. In this example, we can do tasks together running and listening to music and so our scheduler should not be confused and give no answer but instead, consider that we can multitask and simultaneously work on these tasks.

After the above considerations, we need to make different groupings for types of tasks. Based on a person and their needs the scheduler will require specific needs so we will try to make a generalized scheduler that can cater to maximum needs. The groupings need to be based on fixed time and continuous tasks. For instance, you are on a bus tour to a Sky Lantern festival. The trip is long and you need to ensure that while doing this task you are not missing other tasks so traveling or even participating in this event can be considered a continuous task. During this continuous task, you can do other things like take classes, do pre-class work or even work on your lab work (like data analysis), these are fixed time-dependent tasks that you need to meet deadlines so you will have a separate groups for them.

Now it is the time for an implementation strategy. We will break this down for a single task and then repeat the process for every other task. First, we need to ensure there are tasks in the box so we need to give our scheduler the ability to add tasks by inserting tasks one by one. But while doing this we need to ensure we are adding a weight that symbolizes the importance of that task. As we might remember we have a meeting tomorrow later than adding we need to wake up tomorrow. So to allow flexibility in rearranging the position of tasks we will assign values to each task so it can organize the tasks based on what needs to be done first. We also need to consider that we are considering what our task is dependent on. As for going to the night market, we need to ensure we completed our lab work and other tasks so on a day when we forget to check mark which tasks we completed and at the end we remember to do that we can remove our night market tour from our box and that will mean we have done all the tasks that come before that for that day. This will also make it easier for our task scheduler to organize and remove tasks. For doing this it will be hard to keep track of a computer to search for names so to make it easier and more efficient we can add Task IDs based on groupings like academics or leisure. We can keep it simple and just enumerate them too. Lastly, there are two more things to do. First, we need to consider that there might be times when we have a busy schedule and we might need to add more tasks but we might not have time so that is why we need to add durations of each task and the ability to change priorities based on that. This way we will know if we have more space in a day to do things. Second, how does our computer know whether to remove the task that we have done or not until we tell it to remove it? Hence, we need to keep track of the status of our tasks and provide the ability for schedulers to update once the task timing starts or prioritized when the duration ends or maybe when we want to just remove it. Thus insertion, deletion, and reorganizing ability of the scheduler will be enough to make a robust tasks scheduler.

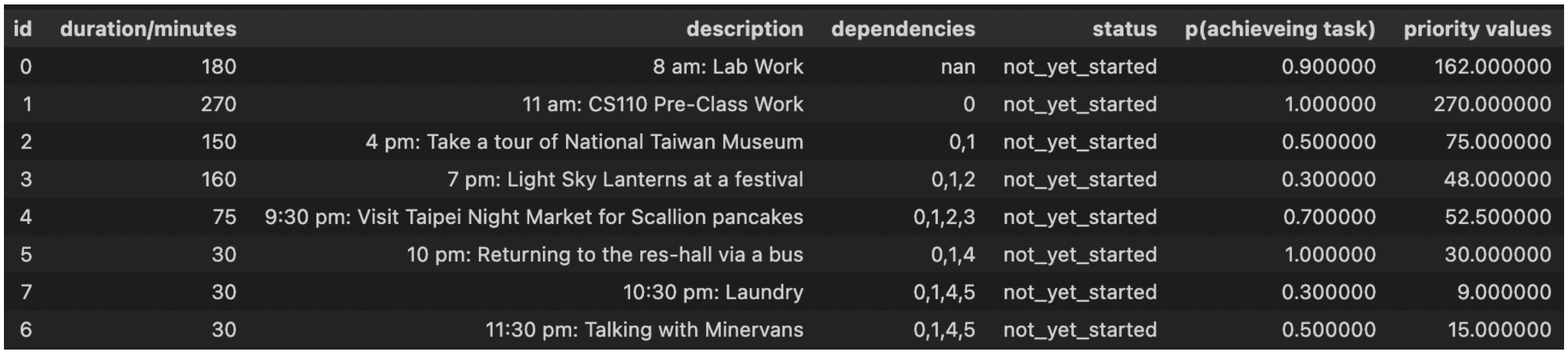
C.

Our tasks scheduler works on using priorities which can be defined as arbitrary values that can be assigned to our tasks that maintain a priority queue by organizing tasks based on these arbitrary values. The higher the arbitrary value, the more priority that task gets and will be on the top of the list. Given that we are trying to select our tasks based on the highest priorities we need to implement max-heap where the element with the highest priority is at the top of the list.

Assigning priority values can be tricky. As mentioned above, each task can be classified into different groups which means that their priority range will be different. For instance, a student taking the class and preparing for that class holds the highest priority especially when that class has a specific timing and cannot be changed. Therefore we can use an expected value theorem to generate these priority values for our given tasks.

Firstly, we need to have a utility function that describes the cost of a given task. Here, given the different grouping of tasks, it will be difficult to formulate a single function for all the tasks so we will be assigning values to each task based on the time constraints/dependencies. Fig.2 shows the priority values generated from assigning a cost and considering the probability of achieving the task that day which also indicates if the task is flexible enough to be pushed another day. The priority values are assigned based on how important a task is for instance I cannot miss pre-classwork for a class as I will be unprepared for the class; however, not going to that Light Sky Lantern won't make an impact on my life. Here, I am basing these assumptions on someone who prioritized work over leisure. The probabilities indicate whether a task has a fixed time or not I need to go to a lab from 8 am - 11 am daily as it is a professional commitment but laundry can be done on Monday if time is available or pushed to the weekend. Here we don't need the probabilities to add up to 1 as some tasks can be done on a later day and the ones with higher mean are mandatory/fixed for that day. Finally, the expected values are the final values that will be considered for prioritizing a task given by:

$$E(X) \ = \ cost(duration) \ * \ p(achieving\ a\ task)$$



id	duration/minutes	description	dependencies	status	p(achieveing task)	priority values
0	180	8 am: Lab Work	nan	not_yet_started	0.900000	162.000000
1	270	11 am: CS110 Pre-Class Work	0	not_yet_started	1.000000	270.000000
2	150	4 pm: Take a tour of National Taiwan Museum	0,1	not_yet_started	0.500000	75.000000
3	160	7 pm: Light Sky Lanterns at a festival	0,1,2	not_yet_started	0.300000	48.000000
4	75	9:30 pm: Visit Taipei Night Market for Scallion pancakes	0,1,2,3	not_yet_started	0.700000	52.500000
5	30	10 pm: Returning to the res-hall via a bus	0,1,4	not_yet_started	1.000000	30.000000
7	30	10:30 pm: Laundry	0,1,4,5	not_yet_started	0.300000	9.000000
6	30	11:30 pm: Talking with Minervans	0,1,4,5	not_yet_started	0.500000	15.000000

Fig. 2 shows the priority values (expected values), probability of achieving a task and priority value for each task.

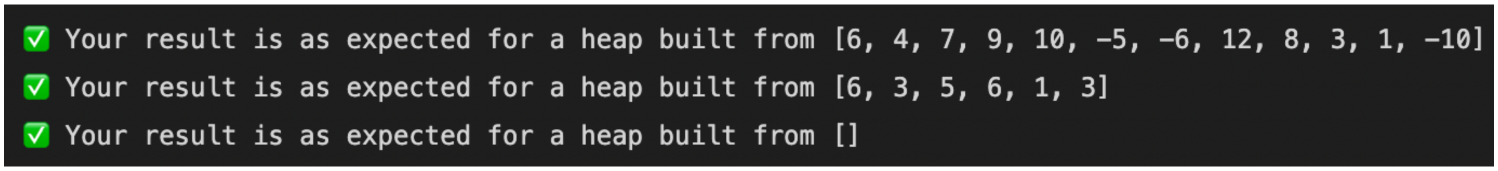
In this case, we cannot miss another important factor to determine our priority values. We need to consider, dependencies. Different tasks vary in dependencies but all the priority values above have been given while considering if a task has a higher number of priorities or not. There are cases where tasks with higher dependencies have been given higher priorities e.g. eating scallion pancakes as food is mandatory while other tasks with lower dependencies are given lower priorities as they are classified as leisure activities that can be done another week.

Python implementation

A:

Code provided in Appendix Part II: A

Output:



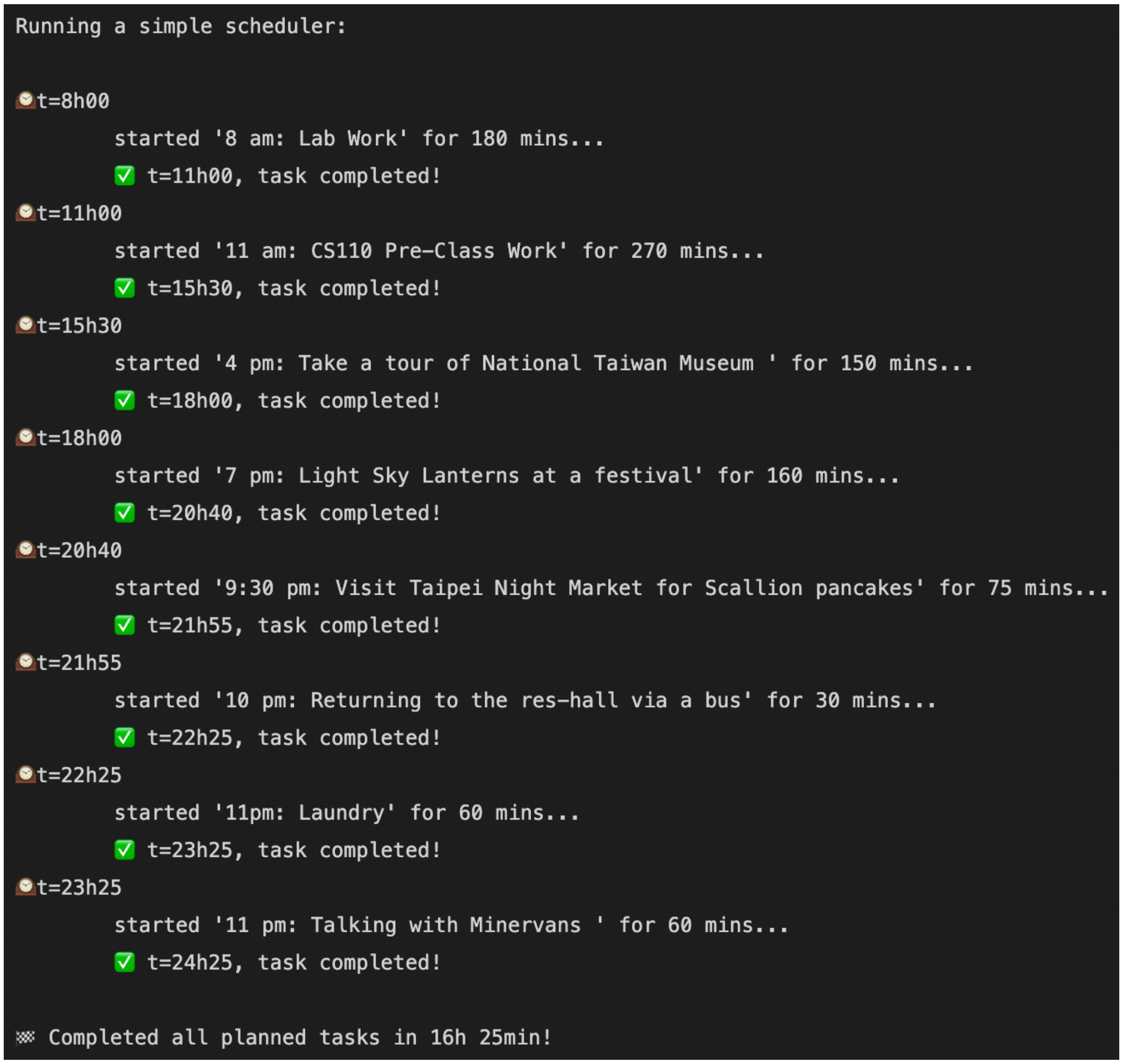
```
✅ Your result is as expected for a heap built from [6, 4, 7, 9, 10, -5, -6, 12, 8, 3, 1, -10]
✅ Your result is as expected for a heap built from [6, 3, 5, 6, 1, 3]
✅ Your result is as expected for a heap built from []
```

B:

Code provided in Appendix Part II: B

My code is utilizing dependencies as the primary mode of priority values. The assumption is that we need to finish some tasks above others and there is limited time in a day. Additionally, multitasking doesn't work unless the tasks include traveling in a bus while doing some work, and also depends on how busy the bus is and if you get a seat. Therefore, to reduce such complexity, the task scheduler only uses dependencies for prioritization.

Output:



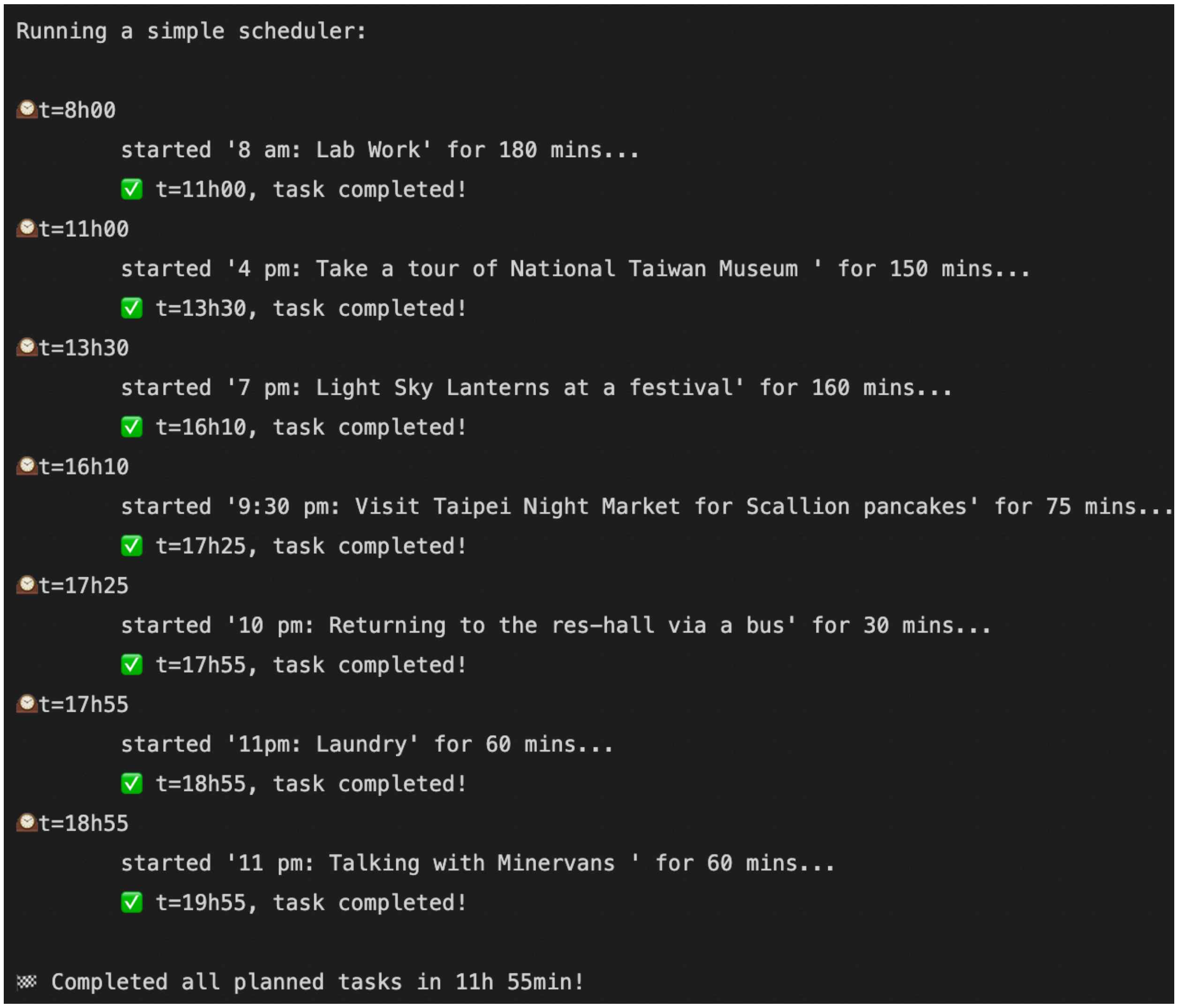
```
Running a simple scheduler:

⏰t=8h00
        started '8 am: Lab Work' for 180 mins...
        ✅ t=11h00, task completed!
⏰t=11h00
        started '11 am: CS110 Pre-Class Work' for 270 mins...
        ✅ t=15h30, task completed!
⏰t=15h30
        started '4 pm: Take a tour of National Taiwan Museum ' for 150 mins...
        ✅ t=18h00, task completed!
⏰t=18h00
        started '7 pm: Light Sky Lanterns at a festival' for 160 mins...
        ✅ t=20h40, task completed!
⏰t=20h40
        started '9:30 pm: Visit Taipei Night Market for Scallion pancakes' for 75 mins...
        ✅ t=21h55, task completed!
⏰t=21h55
        started '10 pm: Returning to the res-hall via a bus' for 30 mins...
        ✅ t=22h25, task completed!
⏰t=22h25
        started '11pm: Laundry' for 60 mins...
        ✅ t=23h25, task completed!
⏰t=23h25
        started '11 pm: Talking with Minervans ' for 60 mins...
        ✅ t=24h25, task completed!

🏁 Completed all planned tasks in 16h 25min!
```

C:

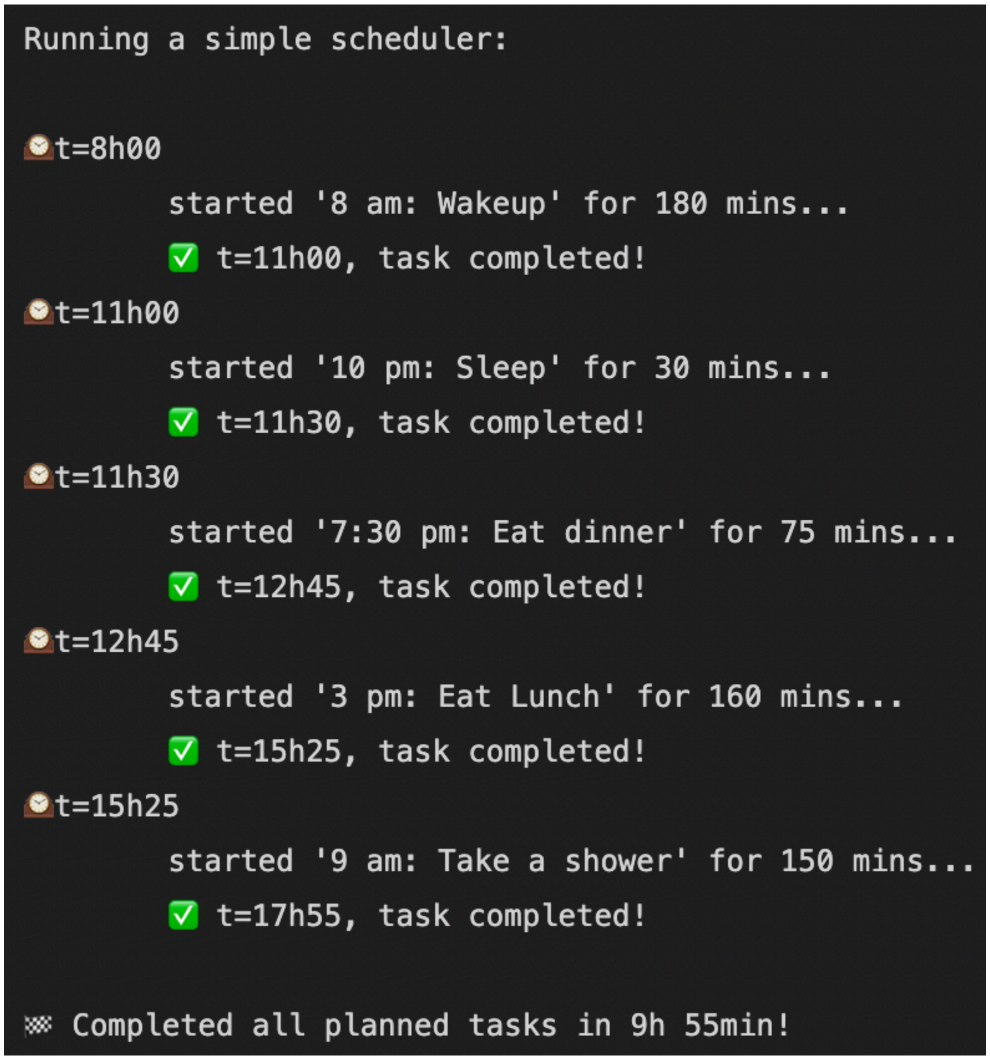
Code provided in Appendix Part II: B

To test the flexibility of the task scheduler I changed the primary mode of comparing priority values: and dependencies and switched around the tasks to see the limitations of the scheduler which will be discussed in later sections.

Output:



```
Running a simple scheduler:

🕰t=8h00
        started '8 am: Lab Work' for 180 mins...
        ✅ t=11h00, task completed!
🕰t=11h00
        started '4 pm: Take a tour of National Taiwan Museum ' for 150 mins...
        ✅ t=13h30, task completed!
🕰t=13h30
        started '7 pm: Light Sky Lanterns at a festival' for 160 mins...
        ✅ t=16h10, task completed!
🕰t=16h10
        started '9:30 pm: Visit Taipei Night Market for Scallion pancakes' for 75 mins...
        ✅ t=17h25, task completed!
🕰t=17h25
        started '10 pm: Returning to the res-hall via a bus' for 30 mins...
        ✅ t=17h55, task completed!
🕰t=17h55
        started '11pm: Laundry' for 60 mins...
        ✅ t=18h55, task completed!
🕰t=18h55
        started '11 pm: Talking with Minervans ' for 60 mins...
        ✅ t=19h55, task completed!

🏁 Completed all planned tasks in 11h 55min!
```



```
Running a simple scheduler:

🕰t=8h00
        started '8 am: Wakeup' for 180 mins...
        ✅ t=11h00, task completed!
🕰t=11h00
        started '10 pm: Sleep' for 30 mins...
        ✅ t=11h30, task completed!
🕰t=11h30
        started '7:30 pm: Eat dinner' for 75 mins...
        ✅ t=12h45, task completed!
🕰t=12h45
        started '3 pm: Eat Lunch' for 160 mins...
        ✅ t=15h25, task completed!
🕰t=15h25
        started '9 am: Take a shower' for 150 mins...
        ✅ t=17h55, task completed!

🏁 Completed all planned tasks in 9h 55min!
```

Let’s test-drive the scheduler

A:

The scheduler proved to be quite resourceful, especially for tasks when comparing academics over entertainment. It was able to reduce the certain level of procrastination to complete my pre-class work by reminding me that I have a busy schedule ahead and providing a specific time range that I should only focus on my tasks. It was quite helpful as a reminder to do smaller tasks like laundry or talking with friends as without a scheduler I would miss it or avoid it but knowing that I have a fixed time for it on my schedule I would work on it.

However, it did fail to consider multiple tasks at one time for instance I can do laundry and talk to Minervans at the same time but it didn't allow co-dependencies which might be due to the way I consider my priorities. It is also limited to consider tasks as fixed rather than continuous in some cases. For the above task nothing was continuous but things like bus rides or listening to a podcast while folding clothes cannot be used here. I tried patching some things by gap spaces that will prevent any task to be missed but for my users, I will instruct some limitations like considering such possible outcomes and booking a longer interval of times than expected for a given task in unforeseen circumstances. Additionally, for tasks where we need to go outside, a time gap can be set between each time to prevent overlapping.

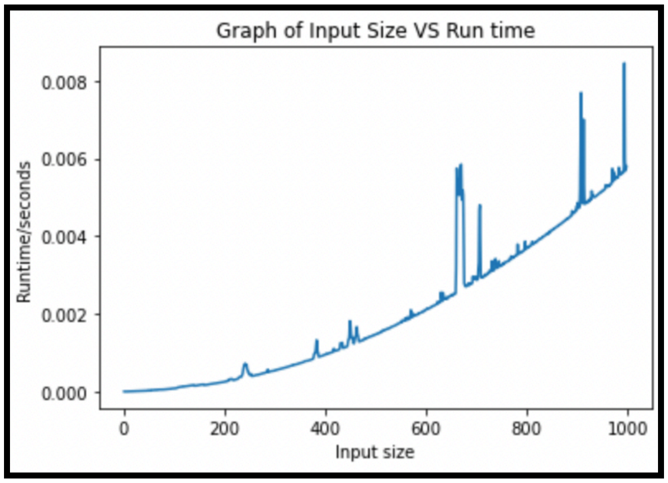
B:

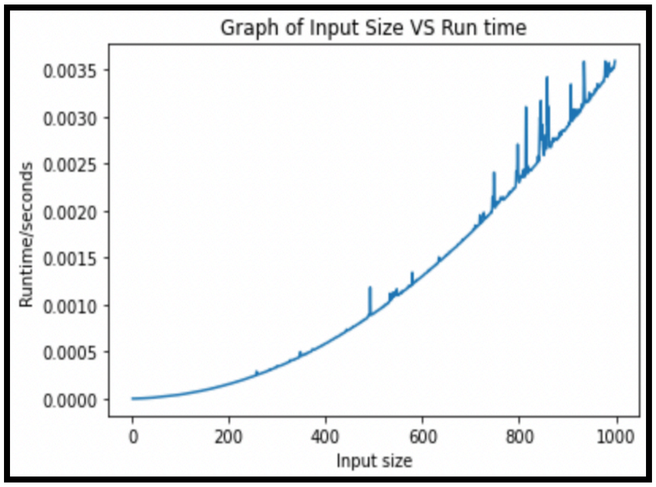
Given that, the scheduler prioritizes dependencies over anything the best metric to compare its efficiency is run-time. We need to see how efficient our scheduler is in adding tasks with long durations and multiple dependencies as we intend to make it usable where the user can insert or remove tasks instantaneously and maybe use it as a checklist of things to accomplish. For this, by comparing the run-time for random runs on different days we can check the time taken and noise produced that can indicate possible errors or wait times based on the schedule of that day and tasks with long durations. We can also check our run-time with different inputs of tasks to see if there is a relatively high difference between the efficiency when large numbers of tasks are added or removed. This feature is needed as we don't want our scheduler to break or delay the removal of tasks in the case where the user has smaller dependencies that make up a task e.g. To finish an assignment a user might want to break the 8-hour-long assignment into 10 minutes intervals. Hence, increasing tasks is also useful to check the asymptotic scaling growth. Lastly, an improved implementation of this scheduler that accounts for deadlines, and fixed and continuous tasks can also check deadlines that might overlap and the effect on the scheduler and also the duration of the tasks.

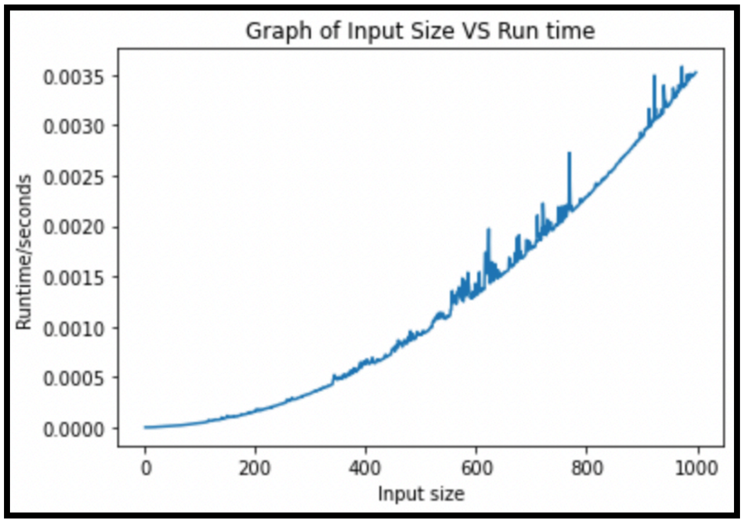
C:

Code provided in Appendix Part II: C

Output: The graphs below show three random trials to check noise in an asymptotic regime.







With an increasing number of tasks, in an asymptotic regime, it is observable that there is considerable noise in the system. The noise starts after 500 input size/tasks which might indicate that we need to improve our task scheduler to reduce delays in processing time. Another observable feature of these experiments is the inward bend of the curve with a higher input size. The inward curve represents that it takes more time to schedule tasks once the size of the task list increases asymptotically. Therefore, the use of heaps and priority queues for this scheduler is not efficient to organize tasks. This conclusion also means that we should compare the efficiency of this scheduler with the efficiency of a scheduler made using other data structures like binary search trees to find the ideal method of scheduling tasks. Lastly, the range of run-time as for the first graph indicates that based on the durations of the tasks there could be a significant difference as to how the scheduler runs, and to avoid that we need to further condense how the input of duration for tasks is processed.

Getting back to the board

A.

With the above information, I believe I would not use this task scheduler for organizing the errands of my day. Firstly, it does not account for fixed vs continuous tasks which indicates that it won't account for multitasking. Hence, it is limited to scheduling major tasks of the day. Secondly, major tasks cannot be broken down into multiple smaller tasks as over a threshold it has poor efficiency which might mean I might get poorly organized tasks or delay in removing or checking my tasks in the queue. Lastly, dependencies that are used for priority values cannot be set co-dependently so again multi-tasking is impossible. These limited features present the conclusion that this scheduler needs more features and better efficiency.

B.

As mentioned above, the task scheduler fails to consider multi-tasking as a possibility. We need to ensure that the scheduler has the ability to read two tasks simultaneously and remove them together. This multitasking can be achieved by merging the tasks which have dependencies on each other. This is useful as we will be able to fix the problem of not allowing co-dependency where two tasks have similar dependencies (mainly each other). For merging the tasks we need to check whether the task is dependent on another task that is dependent on the prior task. This can be simplified through the following equation:

$$Task_1.dependencies\ =\ Task_2.dependencies$$

Once we find that two asks are dependent on each other, similar to status update a feature like a multi-tasking update can be added: Multi-tasking. status = True/False. If a task shows True for multi-tasking it will be merged with its partner to be considered as one task and the ID can be changed to the task with the id that comes first (smaller ID). Now, we are also saving space as we don’t have to store two separate tasks with different dependencies and overall the time complexity of removing the task and its dependencies will become smaller as we delete one task but get rid of two tasks that were added, hence reducing run-time when there are multiple tasks. Finally, for the output, we can make a new list of tasks that consider both the merged and individual tasks. Hence, we have a simple, efficient, and multi-tasking task scheduler.

Word Count: 261 words

References

Q3 A:

[7.2] Heaps and priority queues. Pre-Class Work.

https://sle-collaboration.minervaproject.com/?id=06646410-f55c-4e96-87b9-668cab17cb69&userId=11949&name=Hassan+Bukhari&avatar=https%3A//s3.amazonaws.com/picasso.fixtures/Syed_Hassan%20Bukhari_11949_2021-08-30T02%3A58%3A10.807Z&iframed=1&readOnly=0&isInstructor=0&enableSavingIndicators=1&signature=f094960cb55702032ee41361b81df7e6e08db36cc16488f1201334a1ad034a15

Q3 B:

[7.2] Heaps and priority queues. Breakout room.

https://sle-collaboration.minervaproject.com/?id=14caae4d-9085-4ce0-9555-284834dc8e62&userId=11949&name=Hassan+Bukhari&avatar=https%3A//s3.amazonaws.com/picasso.fixtures/Syed_Hassan%20Bukhari_11949_2021-08-30T02%3A58%3A10.807Z&isInstructor=0&signature=20b021307bd97dd1e07ae5d1b197fd3e71b142f537a4923ea885c11b1e4366d0