CS110 LBA: A Day In My Life In Taipei

CS110: Problem Solving with Data Structures and Algorithms

Prof. H. Ribeiro

Minerva University

November 11, 2022

CS110 LBA: A Day In My Life In Taipei

Part 1: Setting Up

A) Table

A table with the list of tasks for my day, wherein the red tasks are the ones highlighted as cultural immersion tasks unique to Taiwan.

Task ID	Description	Duration	Dependencies	Fixed/Flexible
1	Get Ready & Meet Ethan	30 Minutes	-	09:00
2	Eat Donuts with Ethan	60 Minutes	-	09:30
3	Go to Wanlong Metro Station	15 Minutes	-	Flexible
4	Visit Memorial Park	120 Minutes	(3)	Flexible
5	Scallion Pancakes for Lunch	60 Minutes	-	13:00
6	Effective Altruism Fellowship Meeting	90 Minutes	-	19:00
7	Bubble Tea with Fellows	30 Minutes	(6)	Flexible
8	Attend Brazilian 10:01	90 Minutes	-	22:00

B) Reasons for selecting the said tasks

1. **Get Ready & Meet Ethan:** This morning, I planned to hang out with Ethan, one of the M25s from the US. Before we actually went out for doing what we had planned, I have to first wake up, get ready, and see Ethan at his door.
2. **Eat Donuts with Ethan:** This morning is Ethan's 20th birthday. So, a group of friends and I went with him to get donuts. This is something quite important to Ethan and his parents insisted on doing this as well. So, we all got donuts with Ethan.
3. **Go to Wanlong Metro Station:** I planned to visit a park while I was away with some friends with whom I had donuts at Ethan's birthday. Visiting the park required taking the Taipei Metro from Wanlong Metro Station to the NTU Hospital Metro Station.
4. **Visit 2/28 Peace Memorial Park:** This is a special and very historically important park in Taipei. It is a symbol of the Taiwanese people's fight for democracy and their resistance to authoritarian rule. The park is named after the February 28th Incident, a massacre that occurred in 1947 when the Kuomintang (KMT) government opened fire on unarmed protesters, killing thousands. The park is a reminder of the sacrifices made by the Taiwanese people in their struggle for democracy and freedom. Therefore, I made the plan to visit this park. As somebody who really enjoys history, it has been of my must-visit places in Taipei.
5. **Scallion Pancakes for Lunch:** Scallion pancakes are a staple in Taiwan and extremely popular for breakfast. They are made from flour, water, and scallions. The pancake is traditionally eaten with a dipping sauce made from soy sauce, vinegar, and chili oil, which is

how I’ve been eating it here. As a vegan, this is one of my go-to street foods in Taiwan that I eat almost every day.

6. **EA Fellowship Meeting:** I lead the Minerva Effective Altruism Student Initiative. Effective Altruism is a social movement that promotes the use of reason and evidence to identify and work on the most effective ways to benefit others. It is something that I have been deeply involved in since the beginning of this year. Every week, I facilitate a meeting of the Arete Fellowship in Taipei, where Minerva students come together to discuss the key ideas of effective altruism. On this day, we were scheduled to discuss how nuclear wars, extreme climate change, and pandemics could pose an existential threat to humanity.
7. **Bubble Tea with Fellows:** With my EA group, I have a tradition to eat food (or in Minervan lingo, “break bread”) and drinks with the fellows after nearly an hour of discussions. Since we are in Taiwan, we got bubble tea as this is the place where the drink originates from. The tea is typically made with black tea, milk, and tapioca balls, and it can be served hot or cold. Bubble tea is considered a refreshing and satisfying drink, and it is also very affordable here in Taipei.
8. **Attending Brazilian 10:01:** Finally, at the end of the day, I had to attend Brazilian 10:01. 10:01s are a unique opportunity for us, Minerva students, to learn about the culture, history, and foods of different countries around the world. Brazil is a large and culturally diverse country which I do not know much about. I have been following the recent Brazilian elections a little and have been an avid follower of the Brazilian football team, so I was

especially excited about attending this one!

Part 2: Algorithmic Strategy

A) The task scheduler is meant to create an algorithm that will plan a day with the specified tasks in the most efficient way possible while factoring in the computation of a priority value. A priority queue is a particularly well-suited data structure to prioritize tasks because it can be used to maintain a set of elements where each element has an associated key. The key in our case will be the priority value and start time of the task.

There are two types of priority queues: MinHeap (root node is minimum) and MaxHeap (root node is maximum). In each of these two queues, there are three main operations that this data structure can utilize: insert (add new tasks to the schedule), search (for the highest/lowest value by searching for the value of the root node), and extract (take out the root node of the heap for computational tasks).

Additionally, it allows for the efficient insertion and removal of elements by using a first-in, first-out algorithm. This algorithm ensures that the most important tasks are completed first and that the less important tasks are completed last. That is important as it ensures that the person's day-to-day tasks are completed in a timely and efficient manner.

Therefore, the main advantages of using priority queues are the ability to store data structures efficiently in a manner where they are associated with a key and can most efficiently retrieve data from the heaps. Besides, for a more advanced computation of dynamic priority values, this is also a better data structure. Other than that, there is the time-efficiency advantage such that inserting a new element into a priority queue will have a lesser scaling growth O(logn) in comparison to adding a new element into a sorted list and sorting it once again that will have scaling growth of O(n logn).

B) On a high level, my algorithm works in an intuitive manner of how I would normally schedule the tasks in my day. So, the algorithm would start by reviewing the current time when I wake up, checking when the next fixed task is due, and analyzing whether a flexible task can be scheduled in this time between. Then, repeat this process until there are no longer any fixed tasks left. Once all fixed tasks are completed, then complete the remaining flexible tasks, if any of them are remaining.

This can be developed into an algorithm by putting the fixed and flexible tasks into two different priority queues (both min-heap and max-heap queues could work).

C) The task scheduling will take place with two separative priority queues for fixed and flexible tasks that will employ MinHeap and MaxHeap, respectively. The MinHeap is a tree-based data structure in which the root node is the smallest element; similarly, MaxHeap is a tree-based data structure in which the root node is the largest element. Fixed tasks are the ones that need to be scheduled at a set time. For example, the Brazilian 10:01 is supposed to be scheduled at 10:00pm and cannot be set at any other time. Therefore, the tasks that have a set a start time in the input should be placed in the fixed task bucket and those without would be flexible tasks that can be scheduled at any point in the day.

In my task scheduler, I’ve used min-heap for fixed tasks (such that the tasks that have an earlier start_time can be scheduled first) and max-heap for flexible tasks to accommodate the highest priority tasks (that will be explained in the next section) first. This is to optimize for doing the tasks that take longer (which can be correlated with the importance of the task for me) can be done first. This is something that I personally implement in my own schedule and is also supported by evidence for long-term productivity (Staats, 2019).

For placing the flexible tasks in order, I have defined the priority values for each task by the amount of time it takes for the task to complete. The priority value for each task is assigned by adding the time it takes to complete this task and adding it with the time it takes to complete its dependencies. Therefore, when flexible tasks are inserted into the Maxheap, the tasks with the highest priority (or the task that takes the highest time) are scheduled first, followed by the next most prioritized tasks. This is done because the tasks with the largest duration (or dependencies with the largest duration) can be correlated with how important and valuable they are. Putting them in separate queues is important to ensure that a) the tasks do not overlap and b) they can be out into separate Min and Max heaps. It ensures that we can compare whether there is a time between the current time and the next scheduled fixed task to schedule a flexible task in the middle.

I refrained from measuring the utility values of each task because assigning arbitrary utility values is not very meaningful. The utility is a measure of preference. So, if I prefer to do task ‘a’ before task ‘b’ because ‘a’ is more important and takes more time to complete, then the higher utility (or preference) can be reflected by inserting them in order (as done by the Maxheap function for flexible tasks) without using arbitrary utility values. Additionally, I did not use prospect theory in this scheduler as prospect theory should only be used in case there is risk or uncertainty for the tasks/events. Since the tasks that are inserted in the task scheduler for my case and the other test cases that I presented it with do not include any uncertainty in their occurrence, I do not think there would be any meaningful application of prospect theory here. The cases where I think it might be meaningful and useful to include applications of prospect theory is if there are numerous tasks that are added to the task list and their durations exceed 24hrs. Therefore, those edge cases would be part of the task scheduler's limitations for this approach.

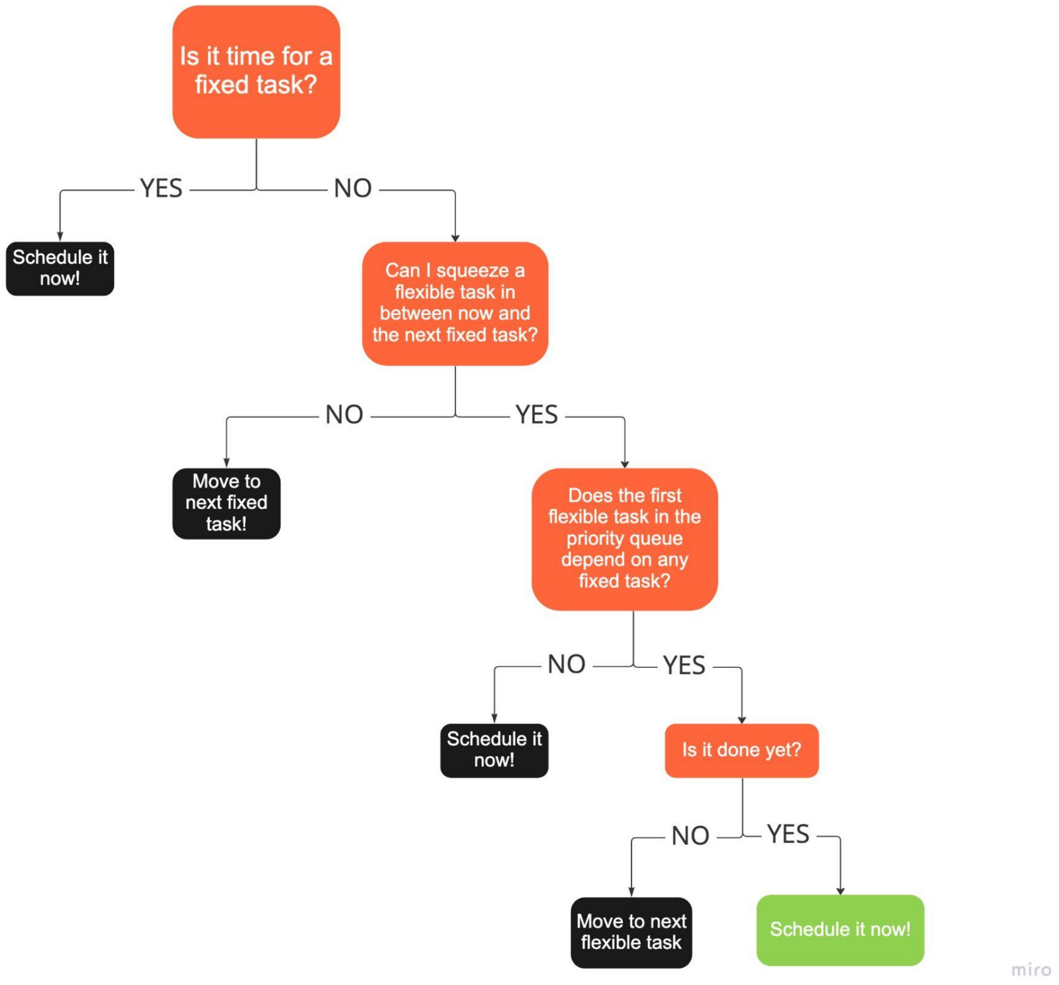
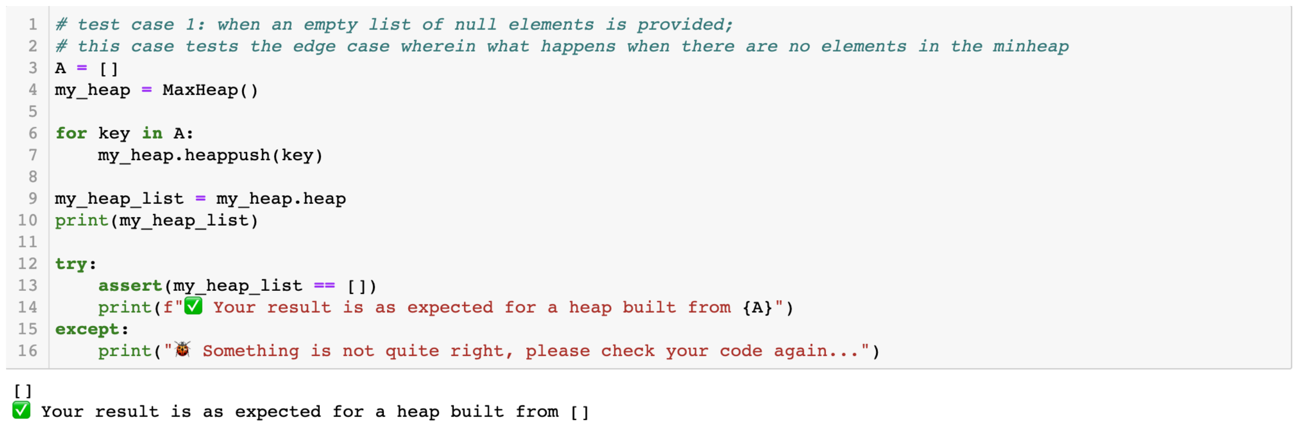

Is it time for a fixed task?
YES
NO
Schedule it now!
Can I squeeze a flexible task in between now and the next fixed task?
NO
YES
Move to next fixed task!
Does the first flexible task in the priority queue depend on any fixed task?
NO
YES
Schedule it now!
Is it done yet?
NO
YES
Move to next flexible task
Schedule it now!
miro

Figure 1. Graphical representation of the high-level algorithmic strategy.

Once these two queues have been generated in order, the algorithm can be prepared by following the high-level strategy described above in section 'B'.

Part 3: OOP Implementation and Test Cases in Jupyter Notebook

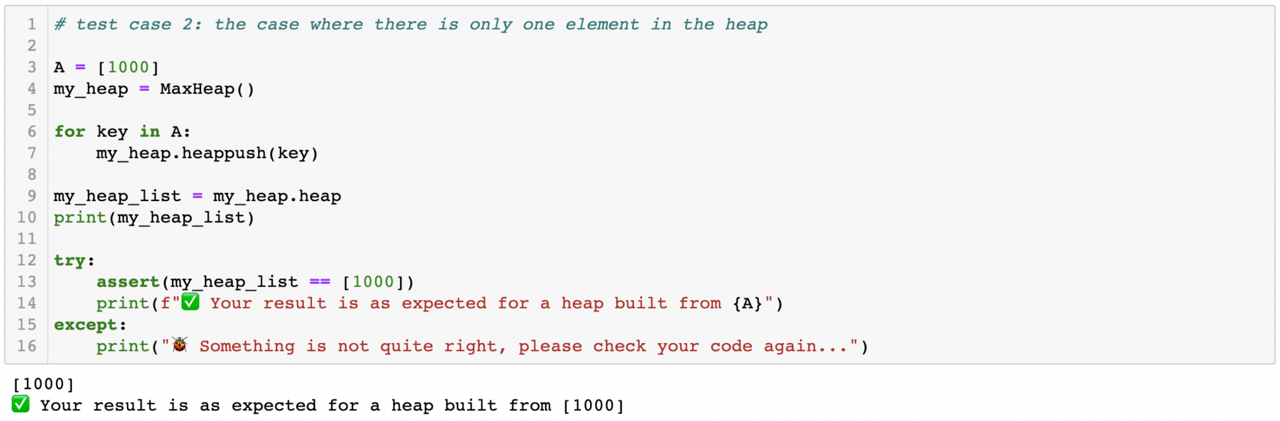
Test Case 1:



```python
# test case 1: when an empty list of null elements is provided;
# this case tests the edge case wherein what happens when there are no elements in the minheap
A = []
my_heap = MaxHeap()

for key in A:
    my_heap.heappush(key)

my_heap_list = my_heap.heap
print(my_heap_list)

try:
    assert(my_heap_list == [])
    print(f"✅ Your result is as expected for a heap built from {A}")
except:
    print("🐞 Something is not quite right, please check your code again...")
```

```
[]
✅ Your result is as expected for a heap built from []
```

Test Case 2:



```python
# test case 2: the case where there is only one element in the heap

A = [1000]
my_heap = MaxHeap()

for key in A:
    my_heap.heappush(key)

my_heap_list = my_heap.heap
print(my_heap_list)

try:
    assert(my_heap_list == [1000])
    print(f"✅ Your result is as expected for a heap built from {A}")
except:
    print("🐞 Something is not quite right, please check your code again...")
```

```
[1000]
✅ Your result is as expected for a heap built from [1000]
```

Test Case 3:

```python
# test case 3: the case where there are multiple values in the heap

A = [6,4,7,9,10,-5,-6,12,8,3,1,-10]
my_heap = MaxHeap()

for key in A:
    my_heap.heappush(key)

my_heap_list = my_heap.heap
print(my_heap_list)

try:
    assert(my_heap_list == [12, 10, 6, 9, 7, -5, -6, 4, 8, 3, 1, -10])
    print(f"✅ Your result is as expected for a heap built from {A}")
except:
    print("🐞 Something is not quite right, please check your code again...")
```

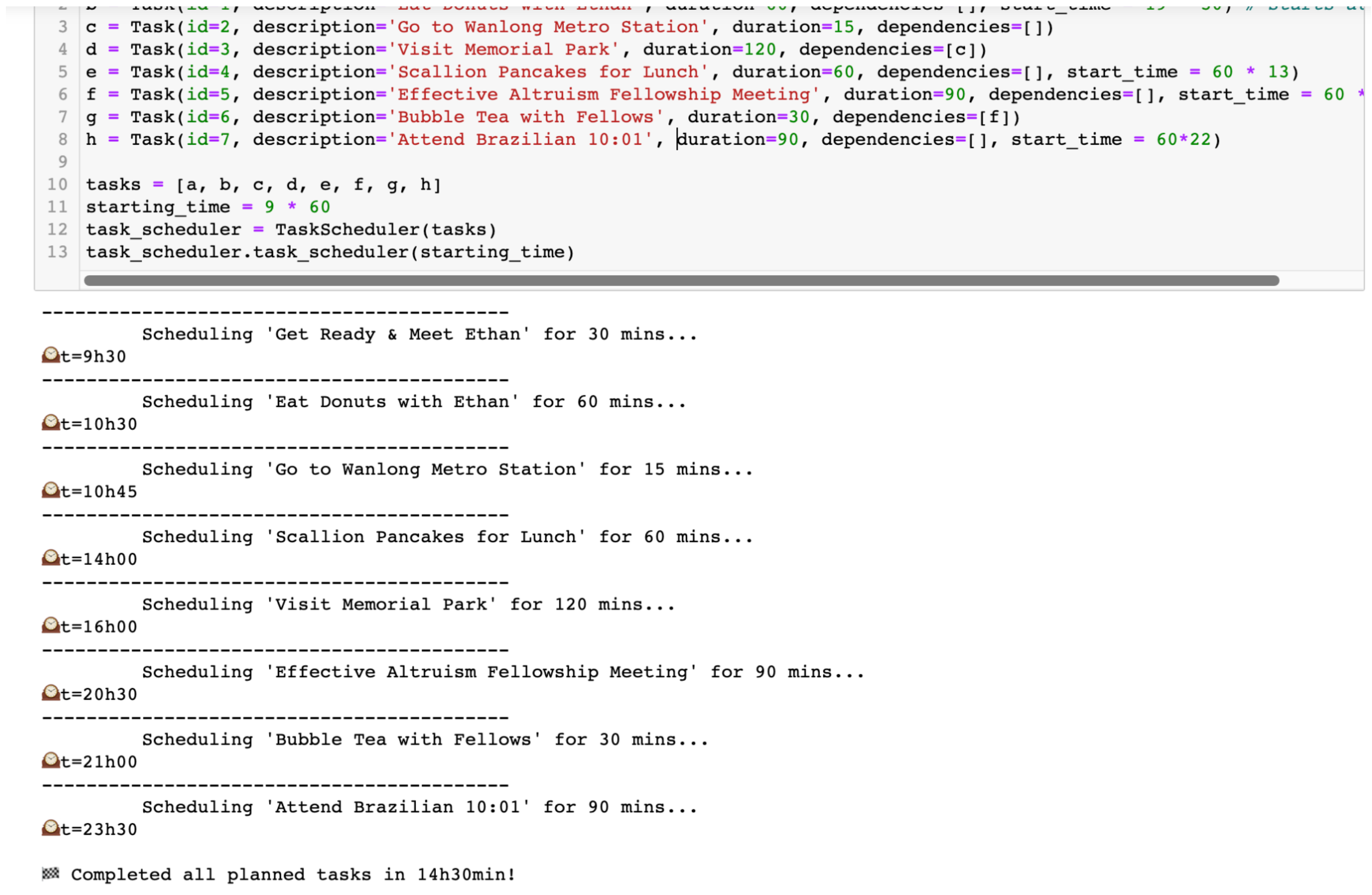
```
[12, 10, 6, 9, 7, -5, -6, 4, 8, 3, 1, -10]
✅ Your result is as expected for a heap built from [6, 4, 7, 9, 10, -5, -6, 12, 8, 3, 1, -10]
```

Test Case 4:

```python
# test case 4: when there are duplicates of elements (i.e. )
A = [0,-5,-6,8,3,1,-10,-5,-5,-6]
my_heap = MaxHeap()

for key in A:
    my_heap.heappush(key)

my_heap_list = my_heap.heap
print(my_heap_list)

try:
    assert(my_heap_list == [8, 3, 1, -5, 0, -6, -10, -5, -5, -6])
    print(f"✅ Your result is as expected for a heap built from {A}")
except:
    print("🐞 Something is not quite right, please check your code again...")
```

```
[8, 3, 1, -5, 0, -6, -10, -5, -5, -6]
✅ Your result is as expected for a heap built from [0, -5, -6, 8, 3, 1, -10, -5, -5, -6]
```

Part B)



```
c = Task(id=2, description='Go to Wanlong Metro Station', duration=15, dependencies=[])
d = Task(id=3, description='Visit Memorial Park', duration=120, dependencies=[c])
e = Task(id=4, description='Scallion Pancakes for Lunch', duration=60, dependencies=[], start_time = 60 * 13)
f = Task(id=5, description='Effective Altruism Fellowship Meeting', duration=90, dependencies=[], start_time = 60 *
g = Task(id=6, description='Bubble Tea with Fellows', duration=30, dependencies=[f])
h = Task(id=7, description='Attend Brazilian 10:01', duration=90, dependencies=[], start_time = 60*22)

tasks = [a, b, c, d, e, f, g, h]
starting_time = 9 * 60
task_scheduler = TaskScheduler(tasks)
task_scheduler.task_scheduler(starting_time)
```

```
------------------------------------------
         Scheduling 'Get Ready & Meet Ethan' for 30 mins...
🕰t=9h30
------------------------------------------
         Scheduling 'Eat Donuts with Ethan' for 60 mins...
🕰t=10h30
------------------------------------------
         Scheduling 'Go to Wanlong Metro Station' for 15 mins...
🕰t=10h45
------------------------------------------
         Scheduling 'Scallion Pancakes for Lunch' for 60 mins...
🕰t=14h00
------------------------------------------
         Scheduling 'Visit Memorial Park' for 120 mins...
🕰t=16h00
------------------------------------------
         Scheduling 'Effective Altruism Fellowship Meeting' for 90 mins...
🕰t=20h30
------------------------------------------
         Scheduling 'Bubble Tea with Fellows' for 30 mins...
🕰t=21h00
------------------------------------------
         Scheduling 'Attend Brazilian 10:01' for 90 mins...
🕰t=23h30

🏁 Completed all planned tasks in 14h30min!
```

As evident from the above, the input is processed in a way that the output provides an order for the tasks, the starting & ending time of each task, task descriptions, and their duration.

Part C)

```python
tasks1 = [a, b, c, d, e, f, g, h]
tasks2 = [b, a, e, f, c, d, h, g]

starting_time = 9 * 60
task_scheduler1 = TaskScheduler(tasks1)
task_scheduler2 = TaskScheduler(tasks2)

try:
    assert(task_scheduler1.task_scheduler(starting_time) == task_scheduler2.task_scheduler(starting_time))
    print(f"✅ Your result is as expected! ✨")
except:
    print("🐞 Something is not quite right, please check your code again...")
```

```
🏁 Completed all planned tasks in 14h30min!

🏁 Completed all planned tasks in 14h30min!
✅ Your result is as expected! ✨
```

As evident from the above, this code will print the same schedule twice and there are no AssertionErrors proving that the order of the tasks in the input does not matter.

Part 4) Let’s test-drive your scheduler

Some pictures from my day running the scheduler :D

Task	Image
Scallion Pancake for Lunch!	

Met Ethan and others for Donuts!	
Effective Altruism Meeting!	

Brazilian 10:01	

I believe that the algorithm's output was fairly efficient. However, there are certain limitations and failure modes that make it not ideal for implementation.

One, the algorithm does not take into account logical, corresponding events. For example, if I wanted to go to the park and visiting the MRT is a prerequisite for that, then they need to be scheduled right after one another. In the current algorithm, it scheduled them far apart from each other during the day.

Two, there are certain flexible tasks that can only be conducted at certain time blocks in the day, which the algorithm does not take into consideration. For example, if I were to have a flexible task of calling my mum in the day, then I cannot do that task after 10 pm. The same could be said for studying for a test, some students would want to study early in the morning or very late at night. Therefore, the algorithm should ideally be able to account for the time preferences of the user.,

Three, in the current algorithm, the fixed tasks occur with certainty at a set time. However, realistically, that is not the case. There are possibilities where a certain flexible task needs to be prioritized over another fixed task occurring at the same time. For example, if one of my flexible tasks were to complete my assignment due at 10 pm, then I would continue working on that instead of attending the Brazilian 10:01 which is a fixed task.

Finally, there is the issue of biases and errors in decision making to not being able to accurately measure the duration of certain tasks as a failure mode. For instance, if there is a flexible task for completing the pre-work for a class, then it is hard to accurately measure by the minute how long a task will take to complete.

The code worked quite efficiently for my schedule, as it was able to accommodate the number of tasks (i.e. 8) that I wanted to complete at the time that I wanted to wake up (i.e. 9 am) and sleep by midnight (i.e. 12am). Additionally, the scheduler was able to accommodate tasks in my day such that the tasks were not back to back and I had free time in the middle of the day. Thus, number of tasks scheduled and number of time taken for the tasks were adequate metrics.

C) Experiment for Efficiency

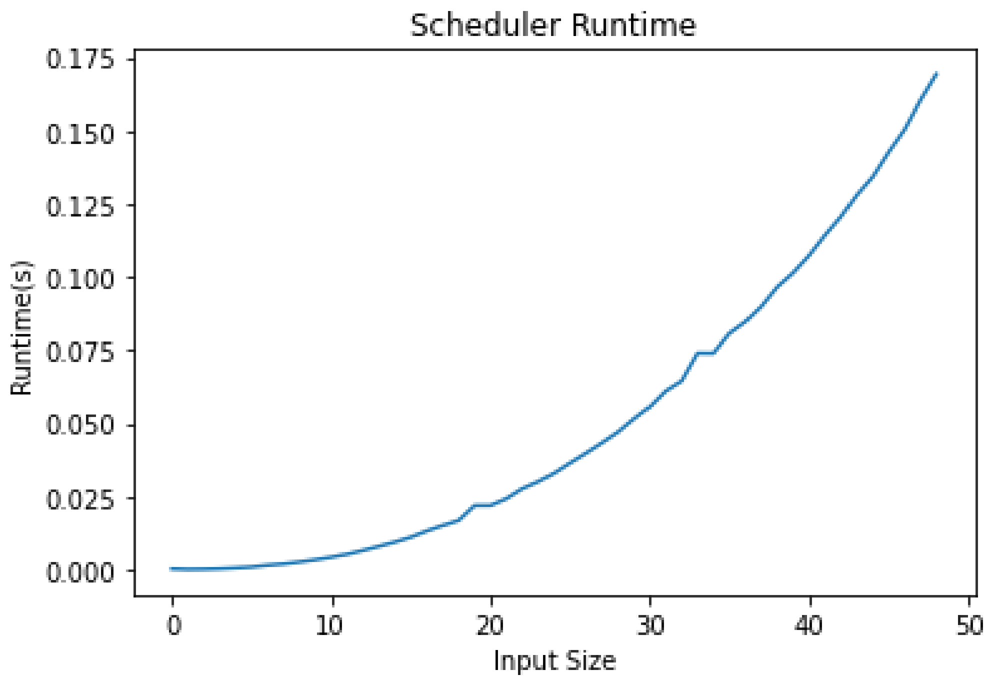


Figure 2. Scheduler Run Time Plotted Against Input Size.

The algorithm, as evident from the graph above, does not appear very efficient when analyzing the runtime of the program. It seems like the scheduler's runtime is scaling quadratically with the increasing input size, which seems plausible due to the numerous loops nested within one another in this implementation. This claim can be reinforced by comparing the runtime at 20 and 40 input sizes which are approximately 0.025 and 0.100, respectively (i.e. in a quadratic way)..

There are a couple of alternative approaches for the algorithms which would theoretically have a different scaling growth as compared to the above. One, the greedy approach would be to not associate the tasks with priority values. This would make the algorithm a lot faster with lower time complexity; however, the main drawback is that it would not be optimal as the importance of the task is not taken into account. Secondly, a brute force approach could be taken where every possible permutation of the possible tasks is taken into account. As a result, the run-time would be slower; but in this approach, the utility can be maximized if priority values are taken into account after checking all possible approaches.

Part 5: Getting back to the board

For the reasons mentioned above in Part 4a, I would not use this algorithm for my day as there is still tremendous scope for improvement. For the drawbacks mentioned above, the algorithm can be significantly improved by adapting the code to fix them and increasing their efficiency. The issue with prioritizing certain flexible tasks over fixed tasks can be potentially corrected by adding priority values to fixed tasks that will ensure scheduling a flexible task with a higher priority over a fixed task that is overlapping. Then, the issue with errors in estimation can be improved by programming a buffer time between tasks. Therefore, if a task does not have an exact duration, then including a 5-10% buffer time that is dependent on the duration of the task can reduce the margin for error in this case. Thus, a task that has an exact time (eg. attending CS110 on Forum which takes 90 minutes) that has a set duration can be scheduled timely; and tasks that do not have a set time (eg. completing the pre-class work for CS110) whose time could take more time than expected. Finally, another improvement for the program could be to program free time (which could be the total time in the day minus the duration of all the tasks scheduled). A set time (eg. 3 hours) can be inputted for free time that is introduced to the program as a constraint to not override. Alternatively, to optimize for a program that ensures free time and also better time complexity, a limit can be set on the number of tasks (eg. 10 tasks) that can be inputted in a day.

Word Count: 275 words

Part 6: Appendix Part 1: HC & LO Applications

#AlgoStratDataStruct

In Part 2 of the assignment, I have given a step-by-step process of how the algorithm functions and the algorithmic strategy of the assignment in simplified language while avoiding technical jargon in its explanation. This included justifying the choice of data structure (i.e. priority queues), creating a flowchart that explained the way the task scheduler's algorithm worked in an accessible manner, and developing an intuition for the way the algorithm should be programmed (by appropriately explaining the high-level strategy).

Word Count: 80 words

#ComputationalCritique

I have described the reasons for choosing the type of algorithm and data structure that has been used in Part 2 of the assignment. Additionally, I critiqued the efficiency of the task scheduler program in parts 4 and 5, where I ran and described why the experimental analysis showed that the algorithm is not very efficient, compared the algorithm to other possible algorithm approaches (i.e. greedy and brute force), noted the various limitations of the program, and suggested improvements to make it better.

Word Count: 85 words

#PythonProgramming

I successfully applied this LO in this assignment as all of the code in my assignment works correctly. The Min and Max heaps are implemented correctly and tested against various appropriate test cases. The Task Scheduler program uses a function that I wrote almost exclusively by myself (without most of the code given in the class session) such as the successful algorithmic approach for the task scheduler and the function for checking dependencies. Additionally, I successfully and accurately created randomized inputs whose runtimes I plotted in my experimental analysis using MatPlotLib.

Word Count: 91

#CodeReadability

I ensured that the code through my assignment followed the best practices of code readability. The functions in the code are named appropriately such that they are easy to understand, all classes and the majority of the functions have been accompanied with appropriate docstrings, and an appropriate number of comments have been placed alongside the code (an improvement from the excessive comments in my previous assignments).

Word Count: 61 words

#Algorithms

I successfully and accurately applied this HC in this assignment by thoroughly engaging with it. In the theoretical parts of the assignment, I ensured that my algorithm strategy was explained clearly, concisely, and chronologically while identifying the base cases for when it should stop running. I also supplemented my explanations with a flowchart that described precisely how the algorithm works. Then, I identified the accurate algorithmic strategy to be used in the assignment and also compared two other possible approaches in the experimental runtime analysis. Finally, I also correctly implemented the algorithmic strategy in my code along with appropriate comments that explain it.

Word Count: 102 words

Bibliography

Staats, B. (2019, November 4). Why You Should Skip the Easy Wins and Tackle the Hard Task First. Kellogg Insight. https://insight.kellogg.northwestern.edu/article/easy-or-hard-tasks-first