Greedy Lab V2 LAB

By Ishaan Bhattacharya

INTRODUCTION:

The office hours problem has my boy Professor Arup Guha and his office hours. He has a window for his office hours consists of a continuous **m** hours. Students come in 5 minute intervals and can be served by it takes Professor Guha only 5 minutes to deal with their problems. So We are then given n (the number of students) and the when they enter the office hours room and the duration for which they stay before they leave. The question is what is the Maximum number of students Guha can serve.

Naïve solutions are solutions that yield the correct answer but have something missing like efficiency. Some naïve solutions can be just test every single time interval with each student object and when an invalid state arises backtrack and choose a different answer. This will use a backtracking method to find the total number of students served. Then just sort all possible answers to have highest first and then use that answer to answer the problem. This will yield the right answer but the expected time will be way longer than O(nlog(n)) for large input sizes. The solution is obviously bad due to the higher run time than my solution(I hope the ideal solution). The runtime for this solution might be O(N!) checking each student for each time interval.

Method:

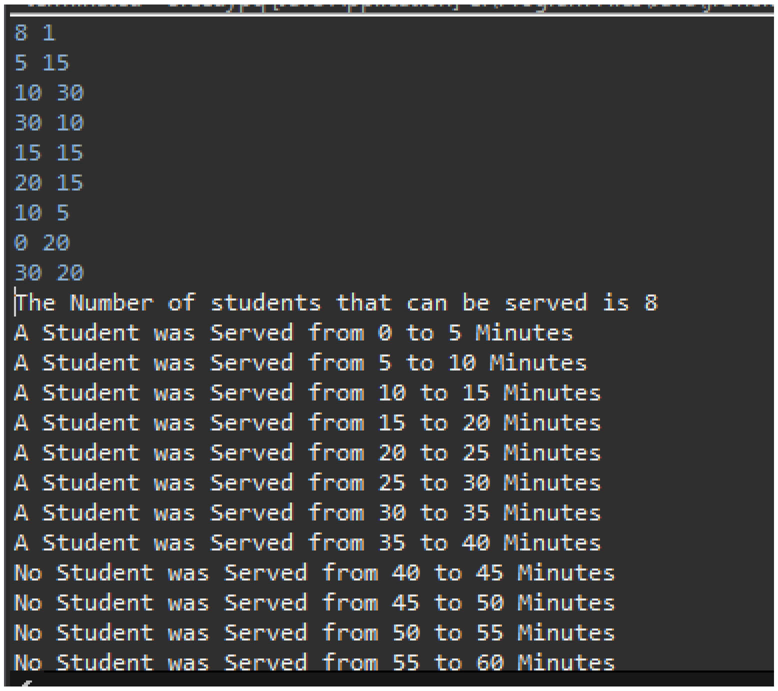
I decided to use a greedy method to improve upon the naïve solution above. I first made an object for each student and inputted that into an array of this type of object. For each object I recorded their start time and their duration. I also created another variable for the Exit time for the student which is just start time plus the duration of the time they are at office hours. I then used this element the exit time to input these objects into a object priority queue. I used this because retrieval in a priority queue is o(logn). And insertion is logn for each element so nlogn. Then after, I check the top of the priority queue and see the entering time. Then I place that object into an array at the time/5 as the index. If there are two for this index I check the leaving time and input that object into the next valid slot. This valid slot occurs when there is an open interval of 5 minutes between the starting time and exit time of the respective student.

If there is a collision, say a student must be placed in an already taken interval for them to be placed. We just poll that student out of the PQ and don't record them in the array. Then we just move on to the next top element in the PQ. We do this until the PQ is empty. At that point just traverse the array and if holds contents at an index then increment your students-served by 1. This will yield the total number of students that Arup can serve.

Results:

1. The method written before is O(nlogn) as that is the highest factor in the problem. That is for retrieving and placing n elements into the priority queue. But in big o we get rid of constants. I have many n runtimes things happening like filling arrays and traversing arrays and stuff but the overall big o runtime is nlogn.

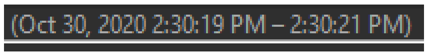
` 2. The following is my method being run through my code on the given table.



```
8 1
5 15
10 30
30 10
15 15
20 15
10 5
0 20
30 20
The Number of students that can be served is 8
A Student was Served from 0 to 5 Minutes
A Student was Served from 5 to 10 Minutes
A Student was Served from 10 to 15 Minutes
A Student was Served from 15 to 20 Minutes
A Student was Served from 20 to 25 Minutes
A Student was Served from 25 to 30 Minutes
A Student was Served from 30 to 35 Minutes
A Student was Served from 35 to 40 Minutes
No Student was Served from 40 to 45 Minutes
No Student was Served from 45 to 50 Minutes
No Student was Served from 50 to 55 Minutes
No Student was Served from 55 to 60 Minutes
```

The first two inputs are n and m where n is number of students and m is the duration of the entire office hours being provided by Professor Guha in hours. In my code the m must be large enough such that all students could possibly fit into the time frame. As we can see the answer I got was 8 and that means all students can be served and the order is seen as well. This means no student is served after the 40 minute mark.

The solution was near instant but the input size was of course low.



(Oct 30, 2020 2:30:19 PM – 2:30:21 PM)

But I did this in the ide and it took a second to copy and paste the input.

This was done in Windows 10, in the eclipse ide and then through the terminal. I do not know how to give memory to the JVM. I made 1 larger test case and then used the test cases given during lab to check and they outputted the correct answers. But for the large test case I just kept putting the same window until it was not feasible to serve that many students.

Conclusion:

My solution I believe is completely correct. The students are placed in the index based on the interval that they are best suited for. The only thing that could be weird is if there are ties of students with the same entry time and duration. The solution of mine would still be correct but based on the way those tied students are inputted could lead to a lot of ways there are put into the open slots. For example, a1 = 0,15 a2=0,15 a3=0,15. There are 3! Ways of inputting these people into the 0-15 slots. But then again, no matter which way they are put in the solution for the problem remains the same. So, objects could be in different orders, but the validity of the solution is always true.

Greedies can be used to solve a variety of question. For example, given a bunch of coins of different value which 4 can u pick to maximize total value. From what I understand greedies solve optimization problems and generally use a sort before making a decision.

Some changes done that could make this greedy difficult are changing the inputs and values such that they are difficult to be sorted properly. Another is asking to make multiple decisions after sorting array which would make a greedy certainly more difficult to impossible.

Appendix:

I used java 1.8 like in class to make and compile this thing. So keep that in mind but normally I would just do the following below to comile and stuff.

HOW TO RUN:

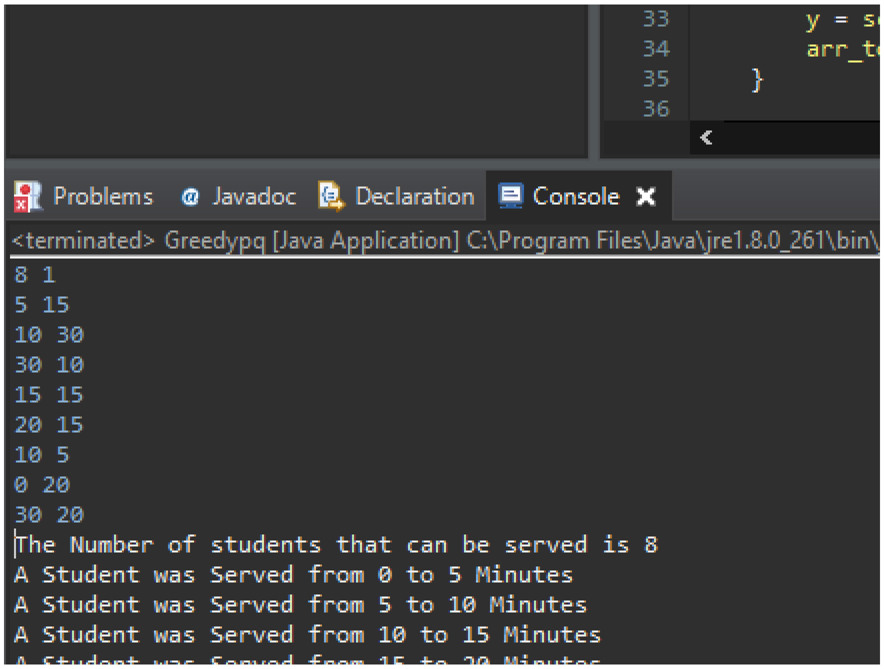
1. go to command prompt and go to the location where the file is saved.
2. Enter thee following into the command line: **javac Greedypq.java** to compile
3. Then enter : **java Greedypq** to run the thing.

INPUT AFTER RUNNING:

1. **first line enter 2 ints : 1 for number of students u will be entering and second for the number of hours Guha will stay at office hours.** The second my always be large enough for all students to come As Guha is a great professor.

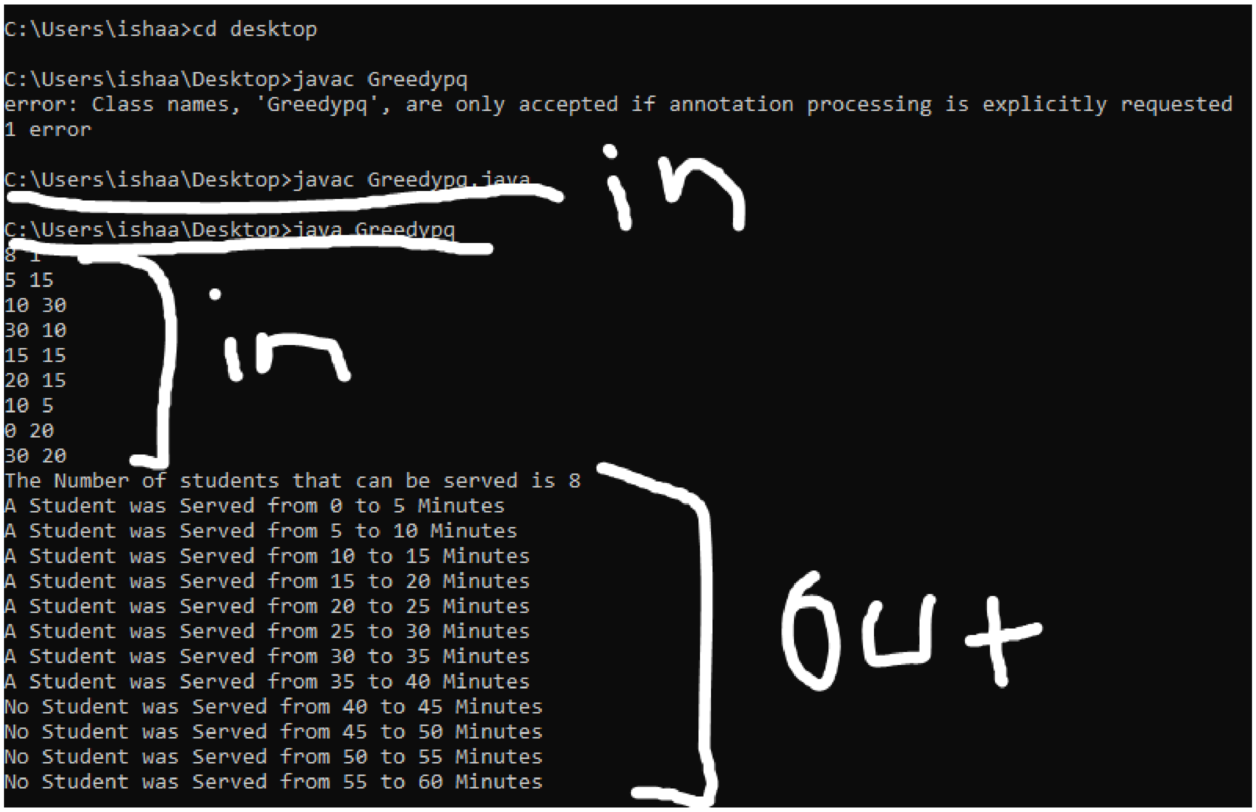
2. in the next n lines input 2 inputs the starting time and duration of the students.

If you need some visual refer to the blueish text in the code below or on the next page.



```
        y = s
        arr_t
    }
```

```
<terminated> Greedypq [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\
8 1
5 15
10 30
30 10
15 15
20 15
10 5
0 20
30 20
The Number of students that can be served is 8
A Student was Served from 0 to 5 Minutes
A Student was Served from 5 to 10 Minutes
A Student was Served from 10 to 15 Minutes
A Student was Served from 15 to 20 Minutes
```



```
C:\Users\ishaa>cd desktop

C:\Users\ishaa\Desktop>javac Greedypq
error: Class names, 'Greedypq', are only accepted if annotation processing is explicitly requested
1 error

C:\Users\ishaa\Desktop>javac Greedypq.java

C:\Users\ishaa\Desktop>java Greedypq
8 1
5 15
10 30
30 10
15 15
20 15
10 5
0 20
30 20
The Number of students that can be served is 8
A Student was Served from 0 to 5 Minutes
A Student was Served from 5 to 10 Minutes
A Student was Served from 10 to 15 Minutes
A Student was Served from 15 to 20 Minutes
A Student was Served from 20 to 25 Minutes
A Student was Served from 25 to 30 Minutes
A Student was Served from 30 to 35 Minutes
A Student was Served from 35 to 40 Minutes
No Student was Served from 40 to 45 Minutes
No Student was Served from 45 to 50 Minutes
No Student was Served from 50 to 55 Minutes
No Student was Served from 55 to 60 Minutes
```