CS526 O2 – Spring 2020
Homework Assignment 4

This assignment has 5 parts.

Part 1. The goal of this part is to give students an opportunity to implement an application that uses a priority queue. The application to be implemented is a simulation of a process scheduler of a computer system. This simulated scheduler is a small, simplified version, which reflects some of the basic operations of a typical scheduler.

Name the file with the main method as *ProcessScheduling.java*. An incomplete code named *ProcessSchedulingIncomplete.java* is posted on Blackboard, for your reference. This code includes Process class definition, a method that reads processes from an input file, and a method that prints a priority queue entry. You can use the code as it is or you may modify any part of the code.

An example simulation program that uses a priority queue is also posted on Blackboard. The name of the program is *PQSimulationExample.java*. Note that the behavior of this simulation program is not exactly the same as that of the process scheduling simulation program. However, careful study of this program will help you to write the process scheduling simulation program. **I strongly suggest that you study this example program before you start this part**.

The following describes the scheduling system that is simulated.

> Processes arrive at a computer system and the computer system executes the processes one at a time based on a priority criterion. Each process has a *process id*, *priority*, *arrival time*, and *duration*. The *duration* of a process is the amount of time it takes to completely execute the process. The system keeps a priority queue to keep arriving processes and prioritize the execution of processes. When a process arrives, it is inserted into the priority queue. Then, each time the system is ready to execute a process, the system removes a process with the ***smallest priority*** from the priority queue and executes it for the *duration* of the process.

For the purpose of this simulation, we assume the following:

- We use the priority queue implemented in the *HeapPriorityQueue.java*. This class implements a priority queue that uses a heap data structure.
- Each entry in the priority queue keeps (*K*, *V*) pair, which represents a process.
  - *K* is the *priority* of the process and it is of Integer type. The value of *priority* is between 1 and 10, inclusively. A process with a smaller *priority* is executed before a process with a larger *priority*.
  - *V* is the reference to the process.
- Each process must have, at the minimum, the following attributes:

  pr: Integer // priority of the process
  id: integer // process id

```
arrivalTime: integer   // the time when the process arrives at the system
duration: integer      // execution of the process takes this amount of time
```

The simulation program uses a logical time to keep track of the simulation process and the same logical time is used to represent the *arrivalTime* and *duration*. The simulation goes through a series of iterations and each iteration represents the passage of one logical time unit (in what follows we will use *time unit* to refer to *logical time unit*). At the beginning, the current time is set to time 0. Each iteration implements what occurs during one time unit and, at the end of each iteration, the current time is incremented.

The following describes the simulation program:

- All processes are stored in a certain data structure *D*, which is supposed to be external to the computer system.
- In each iteration, the following occurs:
  - We compare the current time with the arrival time of a process with the earliest arrival time in *D*. If the arrival time of that process is equal to or smaller than the current time, we remove the process from *D* and insert it into the priority queue *Q* (this represents the arrival of a process at the system).
  - If no process is being executed at this time and there is at least one process in *Q*, then a process with the *smallest priority* is removed from *Q* and executed.
  - **Note:** When there is more than one process with the same *smallest priority*, it would be reasonable that the one with the earliest *arrivalTime* is removed from *Q* and executed. However, for this assignment, you don't need to implement this and you just need to remove an entry with the smallest *priority* as determined by the code within *HeapPriorityQueue.java*.
  - The current time is increased by one time unit.
- The above is repeated until *D* is empty. At this time, all processes have arrived at the system. Some of them may have been completed and removed from *Q* and some may still wait in *Q*.
- If there are any remaining processes in *Q*, these processes are removed and executed one at a time. Again, a process with the smallest priority is removed and executed first.

A pseudocode of the simulation is given below. You must consider this pseudocode as a high-level description and you need to figure out details and convert the pseudocode to a Java program. In the pseudocode, *Q* is a priority queue, which stores <*priority*, *process*> pairs. There is a Boolean variable *running* in the pseudocode. It is used to indicate whether the system is currently executing a process or not. It is ***true*** if the system is currently executing a process and ***false*** otherwise.

```
Read all processes from the input file and store them in an appropriate data structure, D
currentTime = 0
running = false
create an empty priority queue Q

While D is not empty // while loop runs once for every time unit until D is empty
```

```
        Get (don't remove) a process p from D that has the earliest arrival time
        If the arrival time of p <= currentTime
            Remove p from D and insert it into Q
        If Q is not empty and the flag running is false
            Remove a process with the smallest priority from Q
            Set a flag running to true
        currentTime = currentTime + 1
        If currently running process has finished
            Set a flag running to false
    End of While loop

    // At this time all processes in D have been moved to Q.
    While there is a process waiting in Q
        Remove a process with the smallest priority from Q and execute it
```

As mentioned in the first line of the pseudocode, an input file stores information about all processes. The name of the input file is *process_scheduling_in.txt*. A sample input file is shown below (this sample input is posted on Blackboard):

```
1 10 25 10
2 3 15 17
3 1 17 26
4 9 17 30
5 10 9 40
6 6 14 47
7 7 18 52
8 5 18 70
9 2 16 93
10 8 20 125
```

Each line in the input file represents a process and it has four integers separated by a space(s). The four integers are the *process id*, *priority*, *duration*, and *arrival time*, respectively, of a process. Your program must read all processes from the input file and store them in an appropriate data structure. You can use any data structure that you think is appropriate.

However, for the priority queue, you must use the priority queue implemented in *HeapPriorityQueue.java*. The *HeapPriorityQueue* class inherits from or implements a few superclasses and interfaces. You may want to study some or all of these superclasses and interfaces to better understand the *HeapPriorityQueue* class. On Blackboard, I will post only the *HeapPriorityQueue.java* file. You can obtain other files from the textbook's source code collection.

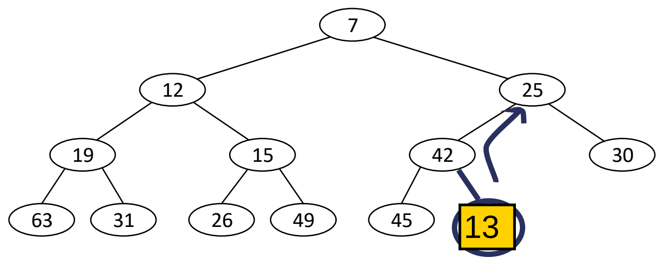
While your program is performing the simulation, whenever a process is removed from the priority queue (to be executed), your program must display information about the removed process. Your program also needs to print other information as shown in the sample output below. After your program finishes the simulation of the execution of all processes in the input file, it must display the average waiting time of all processes. A sample output is shown below, which is the expected output for the above sample input.

```
Id = 1, priority = 10, duration = 25, arrival time = 10
Id = 2, priority = 3, duration = 15, arrival time = 17
Id = 3, priority = 1, duration = 17, arrival time = 26
Id = 4, priority = 9, duration = 17, arrival time = 30
Id = 5, priority = 10, duration = 9, arrival time = 40
Id = 6, priority = 6, duration = 14, arrival time = 47
Id = 7, priority = 7, duration = 18, arrival time = 52
Id = 8, priority = 5, duration = 18, arrival time = 70
Id = 9, priority = 2, duration = 16, arrival time = 93
Id = 10, priority = 8, duration = 20, arrival time = 125

Process removed from queue is: id = 1, at time 10, wait time = 0 Total wait time = 0.0
Process id = 1
        Priority = 10
        Arrival = 10
        Duration  = 25
Process 1 finished at time 35

Process removed from queue is: id = 3, at time 35, wait time = 9 Total wait time = 9.0
Process id = 3
        Priority = 1
        Arrival = 26
        Duration  = 17
Process 3 finished at time 52

Process removed from queue is: id = 2, at time 52, wait time = 35 Total wait time = 44.0
Process id = 2
        Priority = 3
        Arrival = 17
        Duration  = 15
Process 2 finished at time 67

Process removed from queue is: id = 6, at time 67, wait time = 20 Total wait time = 64.0
Process id = 6
        Priority = 6
        Arrival = 47
        Duration  = 14
Process 6 finished at time 81

Process removed from queue is: id = 8, at time 81, wait time = 11 Total wait time = 75.0
Process id = 8
        Priority = 5
        Arrival = 70
```

```
            Duration  = 18
Process 8 finished at time 99

Process removed from queue is: id = 9, at time 99, wait time = 6 Total wait time = 81.0
Process id = 9
            Priority = 2
            Arrival = 93
            Duration  = 16
Process 9 finished at time 115

Process removed from queue is: id = 7, at time 115, wait time = 63 Total wait time = 144.0
Process id = 7
            Priority = 7
            Arrival = 52
            Duration  = 18

D becomes empty at time 125

Process 7 finished at time 133

Process removed from queue is: id = 10, at time 133, wait time = 8 Total wait time = 152.0
Process id = 10
            Priority = 8
            Arrival = 125
            Duration  = 20
Process 10 finished at time 153

Process removed from queue is: id = 4, at time 153, wait time = 123 Total wait time = 275.0
Process id = 4
            Priority = 9
            Arrival = 30
            Duration  = 17
Process 4 finished at time 170

Process removed from queue is: id = 5, at time 170, wait time = 130 Total wait time = 405.0
Process id = 5
            Priority = 10
            Arrival = 40
            Duration  = 9
Process 5 finished at time 179

Total wait time = 405.0
Average wait time = 40.5
```

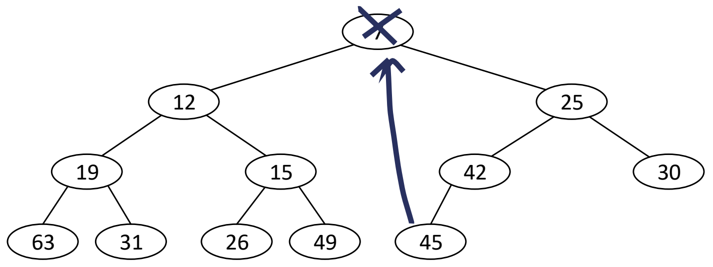
Your program must write the output to an output file named *process_scheduling_out.txt*.

Part 2. Consider the following heap that implements a minimum-oriented priority queue:



Suppose you insert 13 (or an entry with key = 13). Show the resulting heap. You must use the method we discussed in the class.

Part 3. Consider the following heap that implements a minimum-oriented priority queue:



Suppose you perform the removeMin( ) operation. Show the resulting heap. You must use the method we discussed in the class.

Part 4. Consider the following hash table, which contains some keys.

0	1	2	3	4	5	6	7	8	9	10	11	12
	79			17	98		59		22		37	

If you insert k = 14 to this hash table, where is it placed? Assume that collision is resolved using the open addressing method with linear probing.

Part 5. Suppose that your hash function resolves collisions using the open addressing method with double hashing. The double hashing method uses two hash functions h and h'.

Assume that the table size N = 13, $h(k) = k \bmod 13$, $h'(k) = 1 + (k \bmod 11)$, and the current content of the hash table is:

0	1	2	3	4	5	6	7	8	9	10	11	12
	79			17	98		59		22		37	

If you insert *k* = 14 to this hash table, where is it placed?

Documentation

No separate documentation is needed. However, you must include sufficient inline comments within your program. Your inline comments must include a specification for each method in your program. A specification of a method must include at least the following:

- Brief description of what the method does
- Input parameters: Brief description of parameters and their names and types, or none
- Output: Brief description and the type of the return value of the method, or none

Deliverables

You must submit the following files:

- ProcessScheduling.java
- All other files that are necessary to compile and run your program
- A pdf file that has answers to part 2 through part 5. Name this file *hw4.pdf*.

Combine all files into a single archive file and name it *LastName_FirstName_hw*4.*EXT*, where *EXT* is an appropriate file extension (such as *zip* or *rar*). Upload this file to Blackboard.

Grading

Part 1. Your simulation program will be tested with a test input and points will be deducted as follows:

- If processes are not removed (from *Q*) in the correct order, up to 10 points will be deducted.
- If the removal times (from *Q*) are incorrect, 2 points will be deducted for each occurrence of wrong removal time up to 10 points.
- If the average wait time is wrong, 10 points will be deducted.

If you do not include method specifications or sufficient inline comments, points will be deducted up to 20 points.

Part 2. Up to 6 points will be deducted if your answer is wrong.

Part 3. Up to 6 points will be deducted if your answer is wrong.

Part 4. Up to 6 points will be deducted if your answer is wrong.

Part 5. Up to 6 points will be deducted if your answer is wrong.