CMPE 242
Spring 2020
Programming Homework #5

This assignment is due by 23:55 on Thursday, 11 June 2020.

Partner Selection – due by 23:55 Friday, 1 June 2020:

Late submissions: 20 points will be deducted for each late day.

For this assignment, you may work with one partner. If you do so, the two of you will turn in only one assignment and, except in extraordinary situations, receive the same grade (and, if applicable, have the same number of late days applied). Working with a partner is optional; it is fine to complete the assignment by yourself.

If you choose to work with a partner, one of you must upload the names of your group to Moodle before the partner-selection due date. If you choose to work alone, also inform us over Moodle. After the partner selection deadline, you are **not** allowed to make any change on how you will work on the assignment.

If you choose to work with a partner, you may divide the work however you wish, but both partners must understand and be responsible for everything that is submitted. The TA may meet with you to go over your code. Beyond working with a partner, all the usual collaboration policies apply.

You do not need to prepare a PDF report, just document your code well. We will ask you to also explain your code as a face-to-face demo session. The schedule will be announced by the TA later.

Homework

In this programming assignment, you are asked to implement a software solution that processes a list of jobs in parallel. Operating systems such as Linux, MacOS or Windows all have special programs in them called schedulers which do exactly this with the programs on your computer.

You have a program which is parallelized and uses n independent threads to process the given list of m jobs. Each running thread increase CPU power consumption and its temperature. Therefore, to make the overall system as energy efficient as possible, you need to optimize this number. Your program should use jobs data to calculate average waiting times and find out the minimum number of threads needed to meet the average waiting time requirement.

The data are stored in a plain text file. The first line of the input contains m integers ti — the service times (separated by one or more whitespace) in seconds it takes any thread to process i-th job. Jobs ID same as indexed that starting from 0 in second line list. The subsequent lines contain two integers, each separated by one or more whitespace characters (space or

tab). These denote, respectively, the job priority (from 1 to 10, higher is more urgent), arrival time (in minutes from a given point [e.g. 12:00 am]).

For example, from the input file content given below, we understand there are 5 jobs. The first line include jobs service time. According to the first line job with 3 service time is in 0 index and so ID is 0 and job with 4 service time in 3 index so ID is 3. In subsequent lines, the first job (service time is 3 from the first line) with id 0 has priority of 10, arrives at the system at minute 2. The second job with id 1 has priority of 2, arrives at the system at minute 3, and service time is 8 minutes. The third job with id 2 has priority of 7, arrives at the system at minute 6, and the service time is 1 minutes.

Sample Input:

```
3 8 1 4 5

10 2

2 3

7 6

5 3

1 4
```

In this assignment, you are asked to write a simulation program that reads jobs data from an input file and calculates the minimum number of threads required for a given maximum average waiting time.

In your implementation you make the following assumptions:

- Threads take jobs with the highest priority order they are given in the input.
- In case of having two jobs with the same highest priority, the job who has waited longer should be selected first.
- If there is a free thread, it immediately takes the next job from the list.
- If several threads try to take jobs from the list simultaneously, the thread with smaller index takes the job.
- If a thread has started processing a job, it doesn't interrupt or stop until it finishes processing the job.
- After the processing of that job ends, the thread becomes available immediately.
- The waiting time of a job is the duration (difference) between the arrival time of the job and the time he is assigned to a thread.

- The data file will always be valid. All data are composed of integers.
- There may be at most 50 jobs in the data file.

In your implementation, you MUST use a heap-based priority queue to store jobs who are waiting for a thread (i.e., to store jobs who have arrived at the system but have not been process yet). If you do not use such a heap-based priority queue to store these jobs, then you will get no points from this question.

You should implement your own heap data structure or use the textbook implementation of the heap. You should study the code at Sedgewick book to understand the swim/sink operations, and also try to write your own heap class.

You may not use other data structures or implementations provided by Java Collections Framework or any other implementation.

The name of the input file and the maximum average waiting time will be provided as command line arguments to your program. Thus, your program should run using two command line arguments. Thus, the application interface is simple and given as follows:

```
% java Simulator <filename> <avgwaitingtime>
```

Assuming that you have a class called "Simulator", this command calls the executable with two command line arguments. The first one is the name of the file from which your program reads the job data. The second one is the maximum average waiting time; your program should calculate the **minimum number of threads** required for meeting this `avgwaitingtime`. You may assume that the maximum average waiting time is given as an integer. (You should use the parameters of the main() method for this purpose – check the web on how to do that).

HINT:

Use the heap data structure to hold jobs that are waiting for a thread and to find the job with the highest priority. Think about how to apply priority queue to simulate processing of these events in the required order. Update the heap whenever a new job arrives or a job processing starts. In order to find the optimum number of threads needed, repeat the simulation for increasing number of threads and return the minimum number of threads that will achieve the maximum average waiting time constraint. You need to determine for each job which thread will process it and when will it start processing. Display the simulation for which you find the maximum average waiting time.

SAMPLE OUTPUT:

Suppose that you have the following input file consisting of the job data. Also suppose that the name of the file is jobs.txt.

```
10 14 6 5 10 14 10 14 6 5 10 14
20    1
40    1
10    1
10    1
20    4
40    7
20    9
40    11
10    13
10    14
20    15
40    17
```

The output for this input file is given as follows for different thread number. Please check your program with this input file as well as the others that you will create. Please note that we will use other input files when grading your assignments.

```
% java Simulator jobs.txt 5

Minimum number of threads required: 4

Simulation with 4 threads:
Thread 0 takes job 2 at minute 1 (wait: 0 mins)
Thread 1 takes job 1 at minute 1 (wait: 0 mins)
Thread 2 takes job 3 at minute 1 (wait: 0 mins)
Thread 3 takes job 4 at minute 1 (wait: 0 mins)
Thread 3 takes job 5 at minute 6 (wait: 2 mins)
Thread 2 takes job 6 at minute 7 (wait: 0 mins)
Thread 1 takes job 8 at minute 11 (wait: 0 mins)
Thread 0 takes job 7 at minute 15 (wait: 6 mins)
Thread 3 takes job 11 at minute 16 (wait: 1 mins)
Thread 2 takes job 12 at minute 21 (wait: 4 mins)
Thread 0 takes job 9 at minute 25 (wait: 12 mins)
Thread 1 takes job 10 at minute 25 (wait: 11 mins)

Average waiting time: 3 minutes

% Java Simulator jobs.txt 10

Minimum number of threads required: 3

Simulation with 3 threads:
Thread 0 takes job 2 at minute 1 (wait: 0 mins)
```

```
Thread 1 takes job 1 at minute 1 (wait: 0 mins)
Thread 2 takes job 3 at minute 1 (wait: 0 mins)
Thread 2 takes job 6 at minute 7 (wait: 0 mins)
Thread 1 takes job 8 at minute 11 (wait: 0 mins)
Thread 0 takes job 5 at minute 15 (wait: 11 mins)
Thread 2 takes job 12 at minute 21 (wait: 4 mins)
Thread 0 takes job 7 at minute 25 (wait: 16 mins)
Thread 1 takes job 11 at minute 25 (wait: 10 mins)
Thread 0 takes job 4 at minute 35 (wait: 34 mins)
Thread 1 takes job 9 at minute 35 (wait: 22 mins)
Thread 2 takes job 10 at minute 35 (wait: 21 mins)

Average waiting time: 9.83333 minutes
```

Question 2 (bonus, optional, Extra 10 points):

This is a bonus assignment, and you are not required to implement it for full grade. However, if your implementation runs correctly without errors, you will get **+10** pts bonus.

Now suppose that your simulation was to be run for the google deepMind (google deep learning API) service with many potential thread (N) and many, many more jobs. Would it still be a good idea to try the simulation starting from 1 thread and increasing until you found the right number $K \leq N$?

In this optional question, you should implement a more efficient strategy for finding the optimum number of threads in such a case. You are expected to document (using asymptotical analysis or experimentally) why your new method is more efficient.

IMPORTANT

IMPORTANT NOTES: Do not start your homework before reading these notes!!!

1. You should submit your homework to course Moodle page before the deadline. No hardcopy submission is needed. You should send files and any additional files if you wrote additional classes in your solution as a single archive file (e.g., .zip, .rar).

2. The standard rules about late homework submissions apply (20 points will be deducted for each late day). Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.

3. You ARE NOT ALLOWED to modify the given method names. However, if necessary, you may define additional data members and member functions.

4. For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides; or those that you develop yourself. You ARE NOT ALLOWED to

use any codes from somewhere else (e.g., from the internet, other text books, other slides ...).

5. The submissions that do not obey these rules will not be graded.

6. To increase the efficiency of the grading process as well as the readability of your code, you have to follow the following instructions about the format and general layout of your program.

7. Do not forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your Java files. Example:

```
//-------------------------------------------------------
// Title: BST
// Author: Hamid AHMADLOUEI
// ID: 2100000000
// Section: 0
// Assignment: 1
// Description: This class defines a BST
//-------------------------------------------------------
```

8. Since your codes 1will be checked without your observation, you should report everything about your implementation. Well comment your classes, functions, declarations etc. Make sure that you explain each function in the beginning of your function structure. Example:

```
    void setVariable(char varName, int varValue)
    //-------------------------------------------------------
    // Summary: Assigns a value to the variable whose
    // name is given.
    // Precondition: varName is a char and varValue is an
    // integer
    // Postcondition: The value of the variable is set.
    //-------------------------------------------------------
    {
        // body of the function
}
```

   - Indentation, indentation, indentation...

9. This homework will be graded by your TAs, İbrahim İleri and Hamid Ahmadlouei (hamid.ahmadlouei@tedu.edu.tr, ibrahim.ileri@tedu.edu.tr). Thus, you may ask them your homework related questions through HW forum on Moodle course page. You are also welcome to ask your course instructor Tolga Çapın and Tansel Dökeroğlu for help.

GRADING RUBRICs

Performance Element	Weight	Master (4/4)	Advanced (3/4)	Developing (2/4)	Beginner (1/4)	Insufficient (0/4)
Information	5%	Information (id, name, section, assignment number) given in each file.	Missing minor details.	Missing few details (e.g. section id).	Only name given.	None.
Documentation	5%	Every class and method has a detailed explanation (pre- and post-conditions, sample I/O, etc).	Most classes and methods have an explanation, but missing in some parts.	Attempted to document the classes and methods, but they are not clear.	Only few comments.	Not even an attempt.
Code Design	5%	Modular source code and code format. Complete submission of classes and methods.	Methods make sense. Includes constructor that initializes carefully.	Uses set/get methods as necessary.	Class does very little; most functions remain in one main (driver) class.	Methods not properly defined.
Abstract Data Type: heap-based priority queue	10%	Use heap-based priority queue data structure implementation. And provides all functionality.	Uses heap-based priority queue data structure for implementation. And provides most of expected functionalities.	Implements heap with major deviations from the specification.	Used different data structure from heap to solve this problem.	Not even an attempt.
Functionality	65%	All functions are implemented with no missing functionality. Runs without any crash.	Missing some minor features or minor output problems. Runs without any crash.	Attempted to implement all functions but some of them do not work.	Only few functions are implemented correctly. Compiles but several warnings.	No working solution or does not compile.
Testing: Test data creation & generation	10%	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution	Provided a tester class. Able to identify key test points. Clear on what aspects of the solution are being tested	Provided a tester class. Able to identify key test points.	Sporadic.	No test case presented.

		are being tested with each set. Able to generate test data automatically when necessary.	with each set.			