CS 202, Spring 2020
Homework 3 – Heaps and Priority Queues

Due: 23:55, April 24, 2020

Important Notes

Please do not start the assignment before reading these notes.

1. Before 23:55, April 24, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as studentID_secNo_hw1.zip.
2. Your ZIP archive should contain the following files:
   - **maxHeap1.h** and **maxHeap1.cpp**, which contain the class definition and implementation for the first question, respectively.
   - **maxHeap2.h** and **maxHeap2.cpp**, which contain the class definition and implementation for the second question, respectively. Use the same function names with the first question. Thus, you can use the same driver function for the first and second questions.
   - **simulator.cpp** that includes the driver function for simulation as well as your main function.
   - **Makefile** that creates two executables, one for the first question and one for the second question. Name these executables as **simulator_Q1** and **simulator_Q2**, respectively.
   - Do not put any unnecessary files such as the auxiliary files generated from your favorite IDE. Be careful to avoid using any OS dependent utilities.
3. **In your implementation, you MUST use a heap-based priority queue to store patients who are waiting for a doctor (i.e., to store patients who have arrived at the clinic but have not been examined yet). If you do not use such a heap-based priority queue to store these patients, then you will get no points from this assignment**.
4. Although you may use any platform or any operating system to implement your algorithms, your code should work on the dijkstra server (dijkstra.ug.bcc.bilkent.edu.tr). We will compile and test your programs on that server. Please make sure that you are aware of the homework grading policy that is explained in the rubric for homework assignments. Please check the following website https://docs.google.com/document/d/1jyGik6lsghu7KdSk75wwAcjTwo__4nbtIXrWK4_L75w/edit.
5. This homework will be graded by your TA, Can Taylan Sarı. Thus, please contact him directly (can.sari at bilkent edu tr) for any homework related questions.

Attention: For this assignment, you may use the codes given in your textbook and the slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the internet, and belonging to your classmates). Furthermore, you ARE NOT ALLOWED to use any data structure or function from the C++ standard template library (STL).

Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.

Question 1 – 75 points

In this programming assignment, you will find a software solution to the problem of the director of Covid-19 Clinic. She is trying to figure out how many doctors should work in this clinic. For each doctor in this clinic, the expense increases; however, according to the standards, the average waiting time for all patients should not exceed a given amount of time. So, she needs to optimize this number and asks for your help in this task. The clinic has the data of past patients. Your program should use these data to calculate average waiting times and find out the minimum number of doctors needed to meet the average waiting time requirement.

The data are stored in a plain text file[1]. The first line of the file contains the number of patients. The subsequent lines contain four integers, each separated by one or more whitespace characters (space or tab). These denote, respectively, the patient id, the patient priority (from 1 to 10, higher is more urgent), arrival time (in minutes from a given point—e.g., 12:00 am) and service time (in minutes).

For example, from the file content given below, we understand that there are 3 patients. The first patient with id 1 has priority of 9, arrives at the clinic at minute 1, and his examination lasts for 5 minutes. The second patient with id 14 has priority of 3, arrives at the clinic at minute 70, and his examination lasts for 10 minutes. The third patient with id 5 has priority of 3, arrives at the clinic at minute 82, and his examination lasts for 70 minutes.

Sample input file:

```
3
1       9       1       5
14      3       70      10
5       3       82      70
```

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

In your implementation, you may make the following assumptions:

- The data file will always be valid. All data are composed of integers.
- In the data file, the patients are sorted according to their arrival times.
- There may be at most 2000 patients in the data file.

Your implementation must obey the following requirements:

- The patient with the highest priority should be examined first.
- In case of having two patients with the same highest priority, the patient who has waited longer should be selected first.
- If more than one doctor is available at a given time, the patient is assigned to the doctor with a lower id.
- Once a patient is assigned to a doctor, the doctor immediately starts examining that patient and is not available during the examination period given for that patient. After the examination of that patient ends, the doctor becomes available immediately.
- The waiting time of a patient is the duration (difference) between the arrival time of the patient and the time he is assigned to a doctor.

[1] The file is a UNIX-style text file with the end-of-line denoted by a single \n (ASCII 0x0A)

Put the class definition and implementation of the heap data structure in **maxHeap1.h** and **maxHeap1.cpp**, respectively. Put the function that drives your simulation as well as your main function in a file called **simulator.cpp** function.

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. The application interface is simple and given as follows:

dijkstra@hostname:~>./simulator_Q1 <filename> <avgwaitingtime>

Assuming that you have an executable called "simulator_Q1", 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 patient data. The second one is the maximum average waiting time; your program should calculate the minimum number of doctors required for meeting this **`avgwaitingtime.`** You may assume that the maximum average waiting time is given as an integer.

Hint

Use the heap data structure to hold patients that are waiting for a doctor and to find the patient with the highest priority. Update the heap whenever a new patient arrives or a patient examination starts. In order to find the optimum number of doctors needed, repeat the simulation for increasing number of doctors and return the minimum number of doctors that will achieve the maximum average waiting time constraint. Display the simulation for which you find the optimum number of doctors.

SAMPLE OUTPUT:

Suppose that you have the following input file consisting of the patient data. Also suppose that the name of the file is `patients.txt`.

```
12
1       2       1       10
2       4       1       14
3       1       1       6
4       1       1       5
5       2       4       10
6       4       7       14
7       2       9       10
8       4       11      14
9       1       13      6
10      1       14      5
11      2       15      10
12      4       17      14
```

The output for this input file is given as follows for different maximum average waiting times. 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.

```
dijkstra@hostname:~>./simulator_Q1   patients.txt   5

Minimum number of doctors required: 4

Simulation with 4 doctors:

Doctor 0 takes patient 2 at minute 1 (wait: 0 mins)
Doctor 1 takes patient 1 at minute 1 (wait: 0 mins)
Doctor 2 takes patient 3 at minute 1 (wait: 0 mins)
Doctor 3 takes patient 4 at minute 1 (wait: 0 mins)
Doctor 3 takes patient 5 at minute 6 (wait: 2 mins)
Doctor 2 takes patient 6 at minute 7 (wait: 0 mins)
Doctor 1 takes patient 8 at minute 11 (wait: 0 mins)
Doctor 0 takes patient 7 at minute 15 (wait: 6 mins)
Doctor 3 takes patient 11 at minute 16 (wait: 1 mins)
Doctor 2 takes patient 12 at minute 21 (wait: 4 mins)
Doctor 0 takes patient 9 at minute 25 (wait: 12 mins)
Doctor 1 takes patient 10 at minute 25 (wait: 11 mins)

Average waiting time: 3.00 minutes

dijkstra@hostname:~>./simulator_Q1   patients.txt   10

Minimum number of doctors required: 3

Simulation with 3 doctors:

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

Average waiting time: 9.83 minutes
```

Question 2 – 25 points

In the previous question, there is 2000 limit for the number of patients. Now extend your program such that there is no such kind of limit. For that, first dynamically allocate an array with a size of 10 when you construct a heap. Then, double the size (i.e., extend the array by reallocating) if you exceed this number after an insertion. Likewise, halve the size (i.e., this time, shrink the array again by reallocating) if the number of items in the heap falls below the half of the allocated size after a deletion.

Put the class definition and implementation of this extended version in **maxHeap2.h** and **maxHeap2.cpp**, respectively. Use the same function names with the implementation that you use in the first question. So, you can use the same driver function given in **simulator.cpp**.