CS202, Fall 2023

Homework 2 - AVL Trees and Heaps

Due: 19/03/2024

Before you start your homework, please read the following instructions carefully:

FAILURE TO FULFILL ANY OF THE FOLLOWING REQUIREMENTS WILL RESULT IN A GRADE SCORE OF 0 (zero) WITHOUT ANY CHANCE OF REDEMPTION.

- See the course page for any late submission policies and Honor Code for Assignments.
- Upload your solutions in a **single ZIP archive** using the Moodle submission form. Name the file as `studentID_name_surname_hw2.zip`.
- Your ZIP archive should contain **only** the following files:
  - Your `.cpp`, `.h`, `.pdf files`, and **Makefile** **(No Makefile results in 50% deduction in points)**.
  - Do not forget to put your name, student ID, and section number in all of these files. Add a header (see below) to the beginning of each file:

    ```
    /*
     * Author : Name & Surname
     * ID: 12345678
     * Section : 1
     * Homework : 2
     * Description : description of your code
     */
    ```

  - Do not add unnecessary files, such as the auxiliary files generated from your preferred IDE.
- Your code must **compile**.
- Your code must be **complete**.
- Your code must contain the proper **comments** needed to understand its function.
- Your code must run on the **dijkstra.cs.bilkent.edu.tr** server.
- We will test your code using the **google test library**. Therefore, the output message for each operation in Question 2 **MUST** match the format shown in the output of the example code.
- For any questions related to the homework, use the “hw2 questions” forum in Moodle.

DO NOT START YOUR HOMEWORK BEFORE READING THIS SECTION CAREFULLY! THE INSTRUCTIONS MAY DIFFER FROM HOMEWORK TO HOMEWORK!

Question 1 – 20 points

Answers to this question must be hand-written. You may either write on a piece of paper and then scan it or use a tablet and export a PDF of your hand-written answers.

(a) ***[10 points]*** Insert 40, 50, 45, 30, 60, 55, 20, 35, 10, 25 to an empty AVL tree. Show the tree after all insertions. Then, delete 10, 40, 50 in the given order. Show the tree after all deletion operations. Use the exact algorithms shown in the lectures.

(b) ***[5 points]*** Insert M, L, X, A, B, D, V, C, Y, K, into an empty max-heap. Show only the final heap as a tree (not as an array) after all insertions. Then, delete L, A, Y, K in the given order. Show only the final heap as a tree after all deletion operations. Use the exact algorithms shown in the lectures.

(c) ***[5 points]*** You would like to store a set of numbers either in a max-heap or a sorted array. For the following applications, explain which data structure is better, or discuss if it does not matter which one is used. Answers without explanation will get 0 points.

(a) finding maximum element quickly

(b) finding minimum element quickly

(c) finding median of elements quickly

(d) deleting an element quickly

(e) forming the structure quickly

Question 2 – 80 points

In this programming assignment, you will estimate the **minimum number of computers** needed in a web hosting firm so that the average waiting time of a client for each HTTP request **does not exceed a given amount of time**.

In order to estimate the minimum number of computers needed, you will be given the logs of the HTTP requests and use the data to simulate the processing of the requests by the computers and **calculate the average waiting time** for the requests. Using the calculated average waiting time, you will estimate the minimum number of computers needed for the future.

The log file is stored in a simple text file, which is a Unix-style file, and the end of line character is denoted by “\n”. The first line of the file gives the number of requests in the log file. The subsequent lines include the **id, priority, request time (denotes the time in milliseconds since the log file is created) and process time (denotes the processing time of the request in milliseconds)**. Each of these are stored as integer numbers and separated by a white space.

For example, from the file content given below, we see 4 requests made to the server and we can extract the information about each request. For instance, the first request has **id 1** and **priority 9**.

Additionally, we can see that it is **sent after 3 ms** of log file creation and can be **processed in 5 ms**. Similar information can also be obtained for other requests as well.

Input file:

```
4
1    9    3    5
2    5    70   10
3    3    82   70
5    1    100  5
```

In this assignment, you will **assume** that each computer follows the given protocol below:

- ➔ The request with the highest priority should be processed first.
- ➔ In case of having two requests with the same highest priority, the request which is sent earlier should be selected.
- ➔ If more than one computer is available, then the computer with lower id will process the request.
- ➔ Once a request is assigned to a computer, the computer immediately starts processing the request and is not available during the processing time for that request. After the process of that request ends, the computer becomes available immediately.

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 requests are sorted according to their 'sent' times.
- ➔ There may be any number of requests in the log file. For example, in the input file given above, this number is 4. *(Hint: use dynamic memory for allocations and release it before the program ends)*.

In your implementation, you **MUST use a heap-based priority queue** to store requests that are waiting for a computer (i.e., to store requests that were sent but have not been processed yet). If you do not use such a heap-based priority queue to store these requests, then **you will get no points from this assignment**.

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:

```
username@dijkstra:~>./simulator <filename> <avgwaitingtime>
```

Assuming that you have an executable 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 log data. The second one is the maximum average waiting time; your program should calculate the minimum number of computers required for meeting this avgwaitingtime. You may assume that the maximum average waiting time is given as a double value.

Hint:

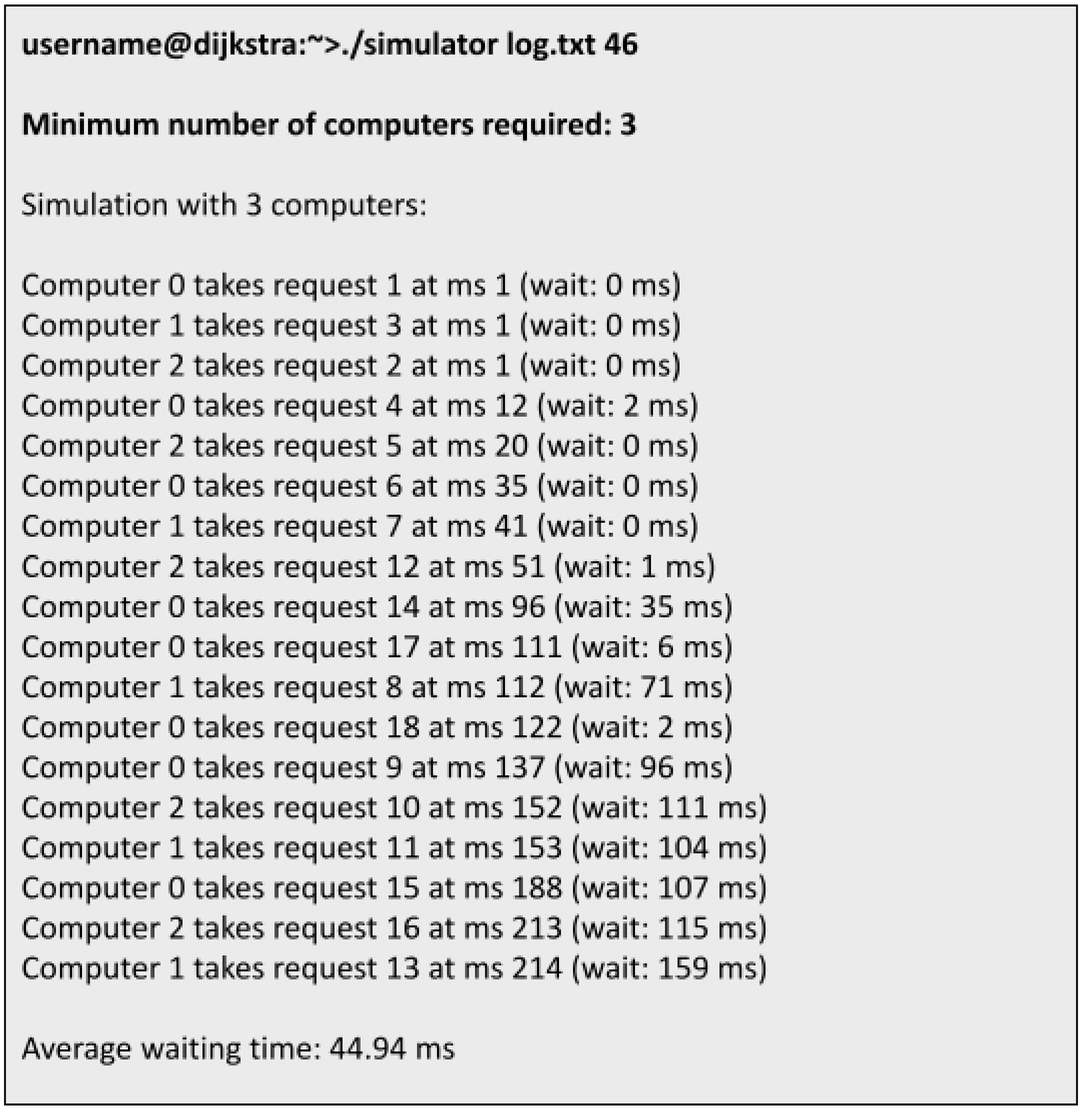
Use the heap data structure to hold requests that are waiting for a computer and to find the request with the highest priority. Update the heap whenever a new request arrives or a request processing starts. In order to find the optimum number of computers needed, repeat the simulation for increasing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

SAMPLE OUTPUT:

Suppose that you have the following input file consisting of the request data. Also suppose that the name of the file is *log.txt*:

```
18
1    20   1    10
2    5    1    10
3    10   1    34
4    10   10   21
5    60   20   30
6    10   35   60
7    10   41   70
8    10   41   40
9    10   41   50
10   10   41   60
11   10   49   60
12   40   50   100
13   5    55   10
14   40   61   14
15   10   81   60
16   10   98   50
17   20   105  10
18   40   120  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 <u>we will use other input files when grading</u> your assignments.



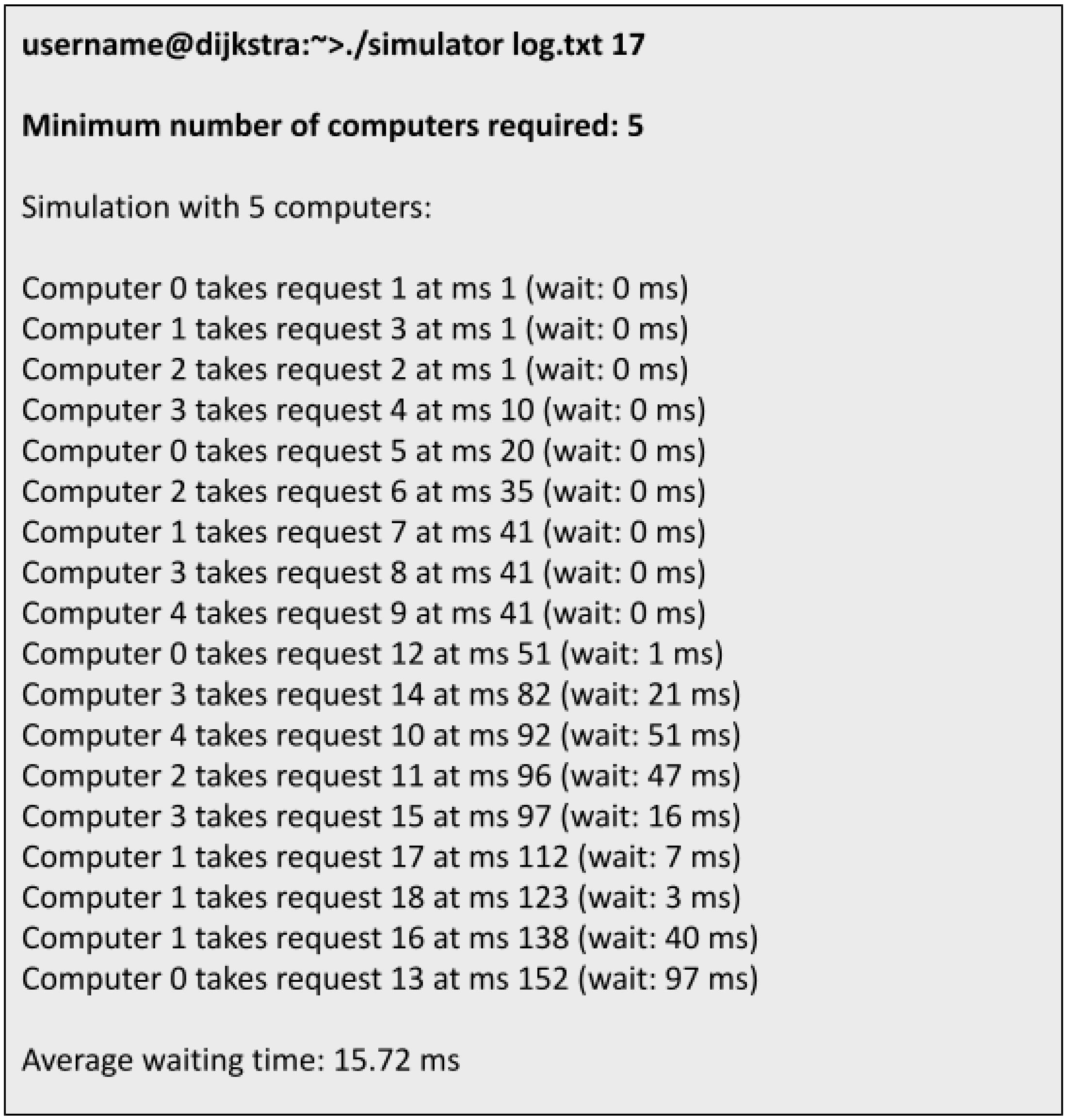
```
username@dijkstra:~>./simulator log.txt 46

Minimum number of computers required: 3

Simulation with 3 computers:

Computer 0 takes request 1 at ms 1 (wait: 0 ms)
Computer 1 takes request 3 at ms 1 (wait: 0 ms)
Computer 2 takes request 2 at ms 1 (wait: 0 ms)
Computer 0 takes request 4 at ms 12 (wait: 2 ms)
Computer 2 takes request 5 at ms 20 (wait: 0 ms)
Computer 0 takes request 6 at ms 35 (wait: 0 ms)
Computer 1 takes request 7 at ms 41 (wait: 0 ms)
Computer 2 takes request 12 at ms 51 (wait: 1 ms)
Computer 0 takes request 14 at ms 96 (wait: 35 ms)
Computer 0 takes request 17 at ms 111 (wait: 6 ms)
Computer 1 takes request 8 at ms 112 (wait: 71 ms)
Computer 0 takes request 18 at ms 122 (wait: 2 ms)
Computer 0 takes request 9 at ms 137 (wait: 96 ms)
Computer 2 takes request 10 at ms 152 (wait: 111 ms)
Computer 1 takes request 11 at ms 153 (wait: 104 ms)
Computer 0 takes request 15 at ms 188 (wait: 107 ms)
Computer 2 takes request 16 at ms 213 (wait: 115 ms)
Computer 1 takes request 13 at ms 214 (wait: 159 ms)

Average waiting time: 44.94 ms
```



```
username@dijkstra:~>./simulator log.txt 17

Minimum number of computers required: 5

Simulation with 5 computers:

Computer 0 takes request 1 at ms 1 (wait: 0 ms)
Computer 1 takes request 3 at ms 1 (wait: 0 ms)
Computer 2 takes request 2 at ms 1 (wait: 0 ms)
Computer 3 takes request 4 at ms 10 (wait: 0 ms)
Computer 0 takes request 5 at ms 20 (wait: 0 ms)
Computer 2 takes request 6 at ms 35 (wait: 0 ms)
Computer 1 takes request 7 at ms 41 (wait: 0 ms)
Computer 3 takes request 8 at ms 41 (wait: 0 ms)
Computer 4 takes request 9 at ms 41 (wait: 0 ms)
Computer 0 takes request 12 at ms 51 (wait: 1 ms)
Computer 3 takes request 14 at ms 82 (wait: 21 ms)
Computer 4 takes request 10 at ms 92 (wait: 51 ms)
Computer 2 takes request 11 at ms 96 (wait: 47 ms)
Computer 3 takes request 15 at ms 97 (wait: 16 ms)
Computer 1 takes request 17 at ms 112 (wait: 7 ms)
Computer 1 takes request 18 at ms 123 (wait: 3 ms)
Computer 1 takes request 16 at ms 138 (wait: 40 ms)
Computer 0 takes request 13 at ms 152 (wait: 97 ms)

Average waiting time: 15.72 ms
```

NOTES:

- ★ You may use codes provided in your textbook or lecture slides however, you cannot use external/pre-existing heap implementations.
- ★ Please refrain from asking questions directly related to your code/implementation. Example: Sending screenshots of non-working code asking how to fix it, Sending code with errors asking where the problem is, etc.
- ★ Your codes will be checked for similarities with each other, and you will be held responsible if high enough similarity is found(see Honor Code).