CS 202, Fall 2019
Homework #3 – Heaps & AVL Trees
Due Date: November 30, 2019

Important Notes

Please do not start the assignment before reading these notes.

- Before 23:55, November 30, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as `studentID_hw3.zip`.
- Your ZIP archive should contain the following files:
  - `hw3.pdf`, the file containing the answers to Part 1,
  - `main.cpp`, any `".cpp"` and `".h"` files which contain the C++ source codes, and the `Makefile`.
  - Do not forget to put your name, student ID, and section number in all of these files. Well comment your implementation. Add a header as in Listing 1 to the beginning of each file:

Listing 1: Header style

```
/**
 * Title: Heaps & AVL Trees
 * Author: Name Surname
 * ID: 21000000
 * Section: 1
 * Assignment: 3
 * Description: description of your code
 */
```

  - 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 (for example to measure the time).

- **You should prepare the answers of Question 1 using a word processor (in other words, do not submit images of handwritten answers).**
- Please use the algorithms as exactly shown in lectures.
- Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work in a Linux environment (specifically on the **Dijkstra** 'dijkstra.ug.bcc.bilkent.edu.tr' server). We will compile your programs with the g++ compiler and test your codes in a Linux environment. Thus, you may lose significant amount of points if your C++ code does not compile or execute in a Linux environment.
- Please make sure that you are aware of the homework grading policy that is explained in the rubric for homeworks
- This homework will be graded by your TA, Alihan Okka. Thus, please contact him directly for any homework related questions.

Attention: For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL). **Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.**

Part 1 – 25 points

(a) [*5 points*] Insert values $15, 77, 79, 10, 18, 8, 11, 37, 25, 75, 9, 67$ into an initially empty AVL tree in order. Show **only the final tree** after all insertions. Then, delete $37, 15, 75$ in given order. Show **only the final tree** after all deletion operations.

(b) [*5 points*] Insert $15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1, 3$ into an empty min-heap. Show **only the final heap as a tree** (not as an array) after all insertions.

(c) [*15 points*] Consider the following statements. Prove the statement if it is **true**, or disprove it if it is **false** by giving a counter example. Answers with no explanations will not get any credits.

(I) All binary search trees are heaps.

(II) The depths of any two leaves in a max heap differ by at most 1. (Note that depth of a node is defined as the length of a simple downward path from root node to the node in question. So, a root node will have a depth of 0.)

(III) In an AVL tree, conducting a left rotation on a node and then a right rotation on the same node will not alter the overall tree structure.

Part 2 – Programming Assignment (75 points)

Priority queues are generally used in job scheduling algorithms or event-driven simulations. In this programming part, you are to write a basic discrete event simulation. Discrete event simulation is focused with processes and systems that change at discrete instants. Pushing an elevator button, starting of a motor, stopping of a motor, and turning on a light are all discrete events because there is an instant of time at which each occurs. The main idea is to analyze a complex system by computing the times that are associated with events.

Imagine yourself as in charge of the *Dijkstra* server in Bilkent University. You will estimate the **minimum number of computers** needed in the server room such that the average waiting time for a student **does not exceed a given amount of time**.

You can think about the scenario like this. A student sends a request to the server for it to run his/her code. Server processes this request and sends the output of the code back to the student. But ultimately, you are not interested in the details of the requests. You just need to abstractly **simulate** the processing of the requests by computers and calculate the **average waiting time** for the requests.

Bilkent Computer Center (BCC) has provided you with a log file containing history of job requests made to the server over some period of time. This is a simple **UNIX-style** text file. The first line of this file indicates the number of job requests in the file. The following lines after that includes in order **job id, priority, time of request (in milliseconds), process time (in milliseconds)**. Note that these will be integers and space delimited (only one space).

For instance in the sample log file below, there had been 4 requests made to the server. The first request has id 1, priority 8, was sent to the server after 3 *ms* since the start of recording, and it takes 5 *ms* to process.

```
4
1 8 3 5
2 4 30 3
3 2 44 24
4 3 57 20
```

The protocol of each computer is as follows:

- The request with the highest priority should be processed first.
- In case of having two requests with the same 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.

You can make the following assumptions in your implementation.

- 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).
- Simulation time starts at 0 ms.
- Finding and extracting the unprocessed job with highest priority and the next event in our priority queues can be done instantly.

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 question.

Hint: Use a priority queue to hold requests that are waiting for a computer and to find the request with the highest priority. Have another priority queue holding *events*. An event being a new request arriving or a computer done with processing a request. The time of the event is the *key value* for the priority queue. Then the psuedocode is:

Initialize system state

Initialize event list

Event loop

- Pull event with lowest-time off event priorty queue
- Process event
  - Decode what type of event
  - Run appropriate code
  - Compile statistics
  - Insert any new events onto queue
- Repeat.

In order to find the optimum number of computers needed, start with a single computer and repeat the simulation for increasing number of computers and return the minimum number of computers that will achieve the **maximum average waiting time constraint**. Stop repeating the simulation if the number of computers are equal to number of job requests. Display the simulation for which you find the optimum number of computers. You can save the statistics for a completed simulation and reset the statistics when a new simulation begins. You can find the output format in the sample outputs given below.

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:

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.

Sample Input – 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*.

```
21
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
19   10    130  50
20   60    150  10
21   40    150  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.

```
username@dijkstra:~$ ./simulator log.txt 50

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 11 (wait: 1 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 50 (wait: 0 ms)
Computer 0 takes request 14 at ms 95 (wait: 34 ms)
Computer 0 takes request 17 at ms 109 (wait: 4 ms)
Computer 1 takes request 10 at ms 111 (wait: 70 ms)
Computer 0 takes request 8 at ms 119 (wait: 78 ms)
Computer 2 takes request 20 at ms 150 (wait: 0 ms)
Computer 0 takes request 18 at ms 159 (wait: 39 ms)
Computer 2 takes request 21 at ms 160 (wait: 10 ms)
Computer 1 takes request 9 at ms 171 (wait: 130 ms)
Computer 0 takes request 11 at ms 173 (wait: 124 ms)
Computer 2 takes request 15 at ms 174 (wait: 93 ms)
Computer 1 takes request 16 at ms 221 (wait: 123 ms)
Computer 0 takes request 19 at ms 233 (wait: 103 ms)
Computer 2 takes request 13 at ms 234 (wait: 179 ms)

Average waiting time: 47.0476 ms
```

username@dijkstra:~$ ./simulator log.txt 20

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 1 takes request 6 at ms 35 (wait: 0 ms)
Computer 2 takes request 7 at ms 41 (wait: 0 ms)
Computer 3 takes request 10 at ms 41 (wait: 0 ms)
Computer 4 takes request 9 at ms 41 (wait: 0 ms)
Computer 0 takes request 12 at ms 50 (wait: 0 ms)
Computer 4 takes request 14 at ms 91 (wait: 30 ms)
Computer 1 takes request 8 at ms 95 (wait: 54 ms)
Computer 3 takes request 11 at ms 101 (wait: 52 ms)
Computer 4 takes request 17 at ms 105 (wait: 0 ms)
Computer 2 takes request 15 at ms 111 (wait: 30 ms)
Computer 4 takes request 16 at ms 115 (wait: 17 ms)
Computer 1 takes request 18 at ms 135 (wait: 15 ms)
Computer 1 takes request 19 at ms 149 (wait: 19 ms)
Computer 0 takes request 20 at ms 150 (wait: 0 ms)
Computer 0 takes request 21 at ms 160 (wait: 10 ms)
Computer 3 takes request 13 at ms 161 (wait: 106 ms)

Average waiting time: 15.8571 ms