CS 202, Fall 2020
Homework #3 – Heaps and Balanced Search Trees
Due Date: November 30, 2020

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.zip`.
- Your ZIP archive should contain the following files:
  - `hw3.pdf`, the file containing the answers to Questions 1, 2 and 4,
  - `MinHeap.h`, `MinHeap.cpp`, `MaxHeap.h`, `MaxHeap.cpp`, `QuickMedian.h`, `QuickMedian.cpp`, `main.cpp` 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
 * Author: Name Surname
 * ID: 21000000
 * Section: 0
 * 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 Questions 1, 2 and 4 using a word processor (in other words, do not submit images of handwritten answers).**
  - Use the exact algorithms 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 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 homeworks.

- This homework will be graded by your TA, Hasan Balci. 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.

Question 1 – 20 points

(a) [*5 points*] Draw all valid min-heaps containing these 4 elements 5, 7, 6, 1.

(b) [*5 points*] Insert $40, 50, 45, 30, 60, 55, 20, 35, 10, 25$ to an empty AVL tree. Show **only the final tree** after all insertions. Then, delete $10, 40, 50$ in given order. Show **only the final tree** after all deletion operations.

(c) [*5 points*] What is the maximum number of keys that a 2-3 tree of height h can hold?

(d) [*5 points*] If you start with an empty 2-3-4 tree and insert the letters in English alphabet A, B, C, D, ... in alphabetical order, the first time the tree would grow to height 2 would be after inserting D. After inserting which letter would the tree grow to height 3 for the first time? Show the 2-3-4 tree before and after inserting that letter.

Question 2 – 10 points

Fill in the below table appropriately by giving *expected* running times in big-O notation without providing any explanations for each operation. **insert** places a new item to the data structure and **extractMin** retrieves the item with minimum key and deletes it from the data structure.

Data Structure	**insert**	**extractMin**
unsorted array		
AVL tree		
min-heap		
unsorted linked list		
sorted linked list		

Question 3 – 60 points

Use the given file names and function signatures during implementation.

(a) [*15 points*] Implement min-heap data structure named as `MinHeap` for maintaining a list of integer keys with the following methods:

```
void insert(int val); // inserts an element into heap
int getMin(); // retrieves the minimum element
int removeMin(); // retrieves and removes the minimum element
int getSize(); // returns the number of elements in heap in O(1) time
int getHeight(); // returns the height of heap in O(1) time
int* getLessThan(int val); // returns an array of integer elements that
are less than given value by doing a preorder traversal on the heap
```

Put your code into `MinHeap.h` and `MinHeap.cpp` files.

(b) [*15 points*] Implement max-heap data structure named as `MaxHeap` for maintaining a list of integer keys with the following methods:

```
void insert(int val); // inserts an element into heap
int getMax(); // retrieves maximum element
int removeMax(); // retrieves and removes the maximum element
int getSize(); // returns the number of elements in heap in O(1) time
int getHeight(); // returns the height of heap in O(1) time
int* getGreaterThan(int val); // returns an array of integer elements
that are greater than given value by doing a preorder traversal on the heap
```

Put your code into `MaxHeap.h` and `MaxHeap.cpp` files.

(c) [*20 points*] Reusing classes from parts a and b, design and implement a data structure for maintaining a list of integers in which `insert` operation is processed in $\mathcal{O}(\log n)$ and finding median is processed in $\mathcal{O}(1)$. Name your data structure as `QuickMedian` and implement the following methods:

```
void insert(int val); // inserts an element into QuickMedian
double getMedian(); // returns the median of elements
```

Make sure your new data structure hides any re-used classes though the use of encapsulation. Put your code into `QuickMedian.h` and `QuickMedian.cpp` files.

(d) [*10 points*] Create a `main.cpp` file which does the followings:

- creates a min-heap object, adds the following numbers $\{15, 50, 45, 30, 60, 55, 20, 35, 10, 25, 65\}$ into it and output the minimum number, size, height and values less than 40 by using `getMin`, `getSize`, `getHeight` and `getLessThan`, respectively.
- creates a max-heap object, adds the following numbers $\{15, 50, 45, 30, 60, 55, 20, 35, 10, 25, 65\}$ into it and output the maximum number, size, height and values greater than 40 by using `getMax`, `getSize`, `getHeight` and `getGreaterThan`, respectively.
- creates a `QuickMedian` object, inserts the following elements $\{10, 40, 30, 50, 70, 60, 20, 90, 100, 110, 0, 25, 123, 11, 200\}$ into it and finds the median of numbers by using `getMedian` method.

At the end, write a basic Makefile which compiles all your code and creates an executable file named `hw3`. Check out these tutorials for writing a simple make file: tutorial 1, tutorial 2. Please make sure that your `Makefile` works properly, otherwise you will not get any points from Question 3.

Question 4 – 10 points

- What are the *expected* running times of getLessThan and getGreaterThan methods used in MinHeap and MaxHeap data structures in big-O notation? Can this be done to execute asymptotically better?
- Explain the data structure you designed for Q3(c) in detail. How does your data structure process insertion in $\mathcal{O}(\log n)$ and find median in $\mathcal{O}(1)$?