CS 202, Fall 2017
Homework #3 – Heaps and AVL Trees
Due Date: November 27, 2017

Important Notes

Please do not start the assignment before reading these notes.

- Before 23:55, November 27, 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 and 3,
  - `MaxHeap.h`, `MaxHeap.cpp`, `MinHeap.h`, `MinHeap.cpp`, `QuickMedian.h`, `QuickMedian.cpp`, `AVLTree.h`, `AVLTree.cpp`, `analysis.h`, `analysis.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 and AVL Trees
 * 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 and 3 using a word processor (in other words, do not submit images of handwritten answers).**

- 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 using the g++ compiler). 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.
- This homework will be graded by your TA, Ilkin Safarli. 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*] 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. 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 Y, X, V, M in given order. Show **only the final heap as a tree** after all deletion operations. Use the exact algorithms shown in the lectures.

(c) [*10 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 – 70 points

Use the given file names and function signatures during implementation.

(a) [*10 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
```

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

(b) [*10 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
```

Put your code into `MinHeap.h` and `MinHeap.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) [*20 points*] Implement AVL tree data structure named as `AVLTree` for maintaining a list of integer keys with the following methods:

```
    void insert(int val); // inserts an element into the tree
    int getHeight(); // returns the height of the tree
```

(e) [*10 points*] In this part, you will analyze the average height of AVL trees and determine if different patterns of insertion affect the height of AVL trees. Write a global function, `void heightAnalysis();` which does the following:

(1) Creates 1000 random numbers and inserts them into an empty AVL tree. After inserting all elements into AVL tree, outputs the height of the tree. Repeat the experiment for the following sizes: $\{2000, 3000, 4000, \cdots, 20000\}$

(2) Instead of creating arrays of random integers, create arrays with elements in ascending order and repeat the steps in part e1.

(3) Instead of creating arrays of random integers, create arrays with elements in descending order and repeat the steps in part e1.

Put your code into `analysis.h` and `analysis.cpp` file. When `heightAnalysis` function is called, it needs to produce an output similar to the following one:

Listing 2: Sample output

```
Part e - Height analysis of AVL trees
-----------------------------------------------------------
Array Size       Random       Ascending        Descending
-----------------------------------------------------------
1000               x              x                 x
2000               x              x                 x
...
```

(f) [*0 points, mandatory*] Create a `main.cpp` file which does the followings:

- creates a max-heap object, adds the following numbers into it and output the maximum number and size by using `getMax` and `getSize` respectively: $\{40, 50, 45, 30, 60, 55, 20, 35, 10, 25\}$
- creates a min-heap object, adds the following numbers into it and output the minimum number and size by using `getMin` and `getSize` respectively: $\{40, 50, 45, 30, 60, 55, 20, 35, 10, 25\}$
- creates a `QuickMedian` object, inserts the following elements into it and finds the median of numbers by using `getMedian` method: $\{10, 40, 30, 50, 70, 60, 20, 90, 100, 110, 0, 25, 123, 11, 200\}$
- calls `heightAnalysis` function

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 2.

Question 3 – 10 points

After running your programs, you are expected to prepare a single page report about the experimental results that you obtained in Question 2 e. With the help of a spreadsheet program (Microsoft Excel, Matlab or other tools), plot *number of elements* versus *height of tree* for random, ascending and descending ordered numbers on the same figure. A sample figure is given in Figure 1 (*these values do not reflect real values*).

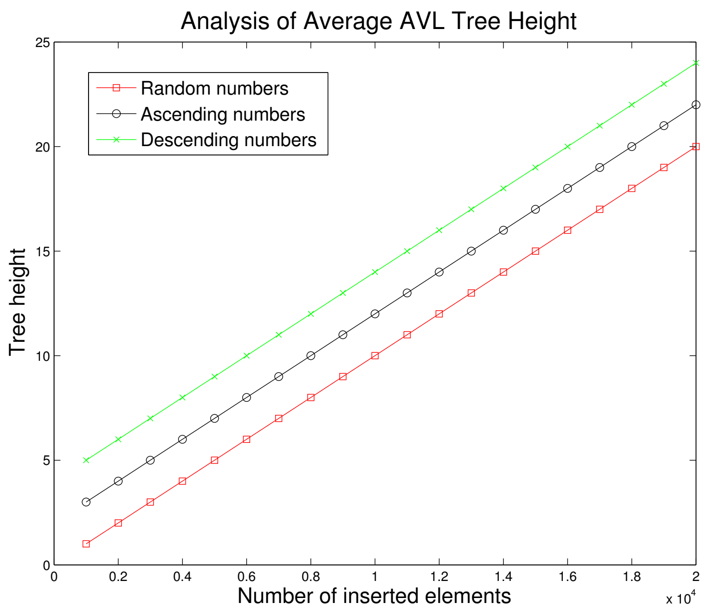

Figure 1: Sample figure

In your report, you need to discuss the following points:

- Do your findings related to average height of AVL tree agree with the theoretical results?
- Do the different patterns of insertion affect the height of AVL tree? If so, explain how. If not, explain why not.
- How would the result be if you used regular Binary Search Tree instead of AVL tree?