CS 202, Fall 2018
Homework #3 – Heaps and AVL Trees
Due Date: November 27, 2018

Important Notes

Please do not start the assignment before reading these notes.

- Before 23:55 on the due date, 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(b) and 3,
  - Source files for each class (e.g `AVLTree.h`, `AVLTree.cpp`, `analysis.h`, `analysis.cpp`, `main.cpp`) 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).**
  - Use the exact algorithms as 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 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, Mubashira Zaman. Thus, please contact her 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 – 25 points

(a) [*10 points*] Insert $9, 12, 10, 5, 1, 8, 20, 15, 13, 25$ to an empty AVL tree. Explain each insertion step in a single line. Show **only the final tree** after all insertions.

(b) [*10 points*] Build a **max-heap** from the following array: $\{41, 75, 63, 33, 49\}$. Apply and trace the heap sort step by step as shown in your lecture slides to obtain the ascending sort of the heap.

(c) [*5 points*] Draw all valid **max-heaps** containing $8, 10, 2, 6, 4$.

Question 2 – 60 points

Use the given file names and function signatures during implementation.

(a) [*35 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 a value into the key
void delete(int val); //deletes a value from the tree
void printTree(); // prints the tree (the same function as hw 2)
```

You can reuse your Binary Search Tree implementation from the previous homework assignment by inheriting from it. In other words, AVLTree would extend the PbBST class, whereas AVLTreeNode would extend the PbBSTNode class.

(b) [*15 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 4000 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: $\{8000, 12000, 16000, \cdots, 80000\}$

(2) Instead of creating arrays of random integers, create arrays with elements in ascending order and repeat the steps in part b1.

(3) Instead of creating arrays of random integers, create arrays with elements in descending order and repeat the steps in part b1.

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

Also add the **screenshot** of this output to the pdf file.

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

- creates an `AVLTree` object, inserts the following elements into it: $\{5, 10, 15, 60, 8, 40, 2, 30, 17, 12, 74\}$, then deletes the following elements: $\{30, 10, 8, 74, 5\}$. Also prints the tree exactly 2 times; after the final insertion and after the final deletion.
- calls `heightAnalysis` function

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

Question 3 – 15 points

After running your programs, you are expected to prepare a single page report about the experimental results that you obtained in Question 2(b). 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*).

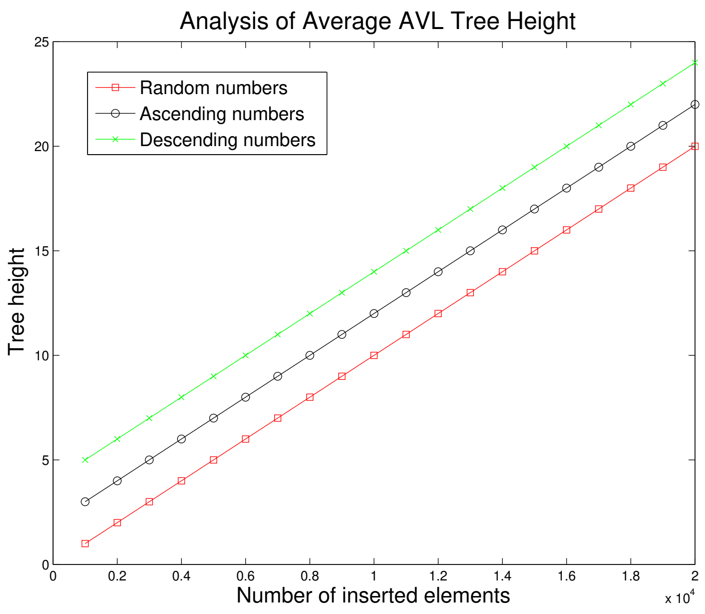

Figure 1: Sample figure

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

- Interpret and compare your empirical results with the theoretical ones. Explain any differences between the empirical and theoretical results, if any.
- What would be the worst case scenario in terms of time complexity while performing the insertion and deletion operations in an AVL tree?