CS 202, Fall 2017
Homework #2 – Binary Search Trees
Due Date: November 6, 2017

Important Notes

Please do not start the assignment before reading these notes.

- Before 23:55, November 6, 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:
  - `hw2.pdf`, the file containing the answers to Questions 1 and 3,
  - `AbBinaryNode.h`, `AbBinaryNode.cpp`, `AbBST.h`, `AbBST.cpp`, `PbBinaryNode.h`, `PbBinaryNode.cpp`, `PbBST.h`, `PbBST.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: Binary Search Trees
 * Author: Name Surname
 * ID: 21000000
 * Section: 0
 * Assignment: 2
 * 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 a significant amount of points if your C++ code does not compile or execute in a Linux environment.

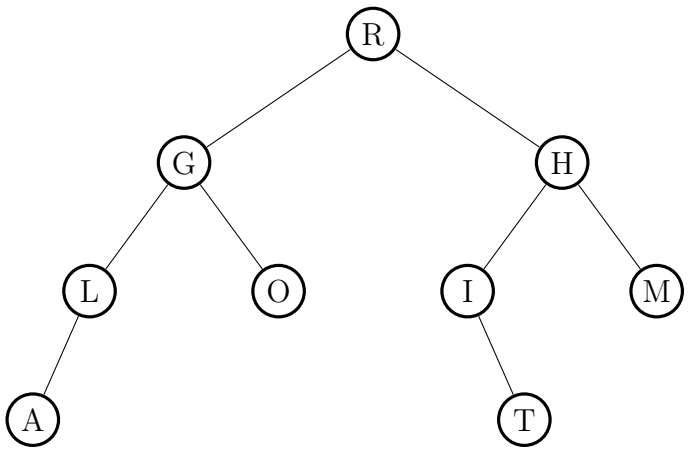
- This homework will be graded by your TA, Ilkin Safarli. Thus, please contact him directly for any homework related questions.

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 – 15 points

(a) [*5 points*] What are the preorder, inorder, and postorder traversals of the binary tree below:



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

order. Show **only the final tree** after all deletion operations. Use the exact algorithms shown in lectures. Verify your answers by using this visualization tool.

(c) [*5 points*] A binary tree has a pre-order traversal of B, I, N, A, R, Y and postorder traversal of N, A, I, R, Y, B. What is its inorder traversal? Reconstruct the tree from those traversals and draw it.

Question 2 – 70 points

(a) [*15 points*] Write an array-based implementation of Binary Search Tree (BST) named as `AbBST` for maintaining a list of integer keys. Implement only `insert` and `getHeight` methods for `AbBST` class. The initial size of the array is two. During insertion, if there is no more empty space left in the array, you should double the array size. Put your code into `AbBinaryNode.h`, `AbBinaryNode.cpp`, `AbBST.h` and `AbBST.cpp` files. Prototypes of required methods:

```
void insert(int val);
int getHeight();
```

(b) [*15 points*] Write a pointer-based implementation of Binary Search Tree named as `PbBST` for maintaining a list of integer keys. Implement only `insert` and `getHeight` methods for `PbBST` class. Put your code into `PbBinaryNode.h`, `PbBinaryNode.cpp`, `PbBST.h` and `PbBST.cpp` files. Prototypes of required methods:

```
void insert(int val);
int getHeight();
```

(c) [*10 points*] Write a method for `PbBST` class with the following prototype to check if the tree is a valid BST or not: `bool isValidBST();`

(d) [*10 points*] Write another method for `PbBST` class to return the median of numbers in that BST *in linear time* (linear in the number of items). Your method should have the following prototype:

```
int medianOfBST();
```

(e) [*10 points*] In this part of the homework, you will analyze the performance of array-based and pointer-based implementations of Binary Search Trees. Write a global function, `void performanceAnalysis()`, which does the following:

- Creates 2000 random numbers and inserts them into an empty array-based BST and an empty pointer-based BST. Calculate the elapsed times to insert all of those numbers into each BST and output them (use `clock` from `ctime`

for calculating elapsed time). Repeat the experiment for the following sizes: {4000, 6000, 8000, 10000, 12000, 14000, 16000, 18000, 20000}

Put your code in a file named `analysis.cpp`. When `performanceAnalysis` function is called, it needs to produce an output similar to the following one:

Listing 2: Sample output

```
Part e - Performance analysis of BST implementations
-------------------------------------------------------------
Array Size       Array Based      Pointer Based
-------------------------------------------------------------
1000                  x ms             x ms
2000                  x ms             x ms
...
```

(f) [*10 points*] Height of BST is a very important property which affects the performance of search, delete, and insert operations directly. In this part, you will analyze how different patterns of insertion affect the height of a BST. Write a global function, `void heightAnalysis();` which does the following:

(1) Creates 2000 random numbers and inserts them into an empty pointer-based BST. After inserting all elements into BST, output the height of the tree. Repeat the experiment for the following sizes: {4000, 6000, 8000, 10000, 12000, 14000, 16000, 18000, 20000}

(2) Instead of creating arrays of random integers, create arrays with elements in ascending order and repeat the steps in part f1.

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

Listing 3: Sample output

```
Part f - Analysis of BST height
-------------------------------------------------------------
Array Size       Random Numbers       Ascending Numbers
-------------------------------------------------------------
1000                  x                    x
2000                  x                    x
```

. . .

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

- creates a pointer-based BST and adds the following numbers into it: $\{40, 50, 45, 30, 60, 55, 20, 35, 10, 25\}$
- check if this tree is valid or not by using `isValidBST` method
- find the median of numbers by using `medianOfBST` method
- calls `performanceAnalysis` function
- calls `heightAnalysis` function

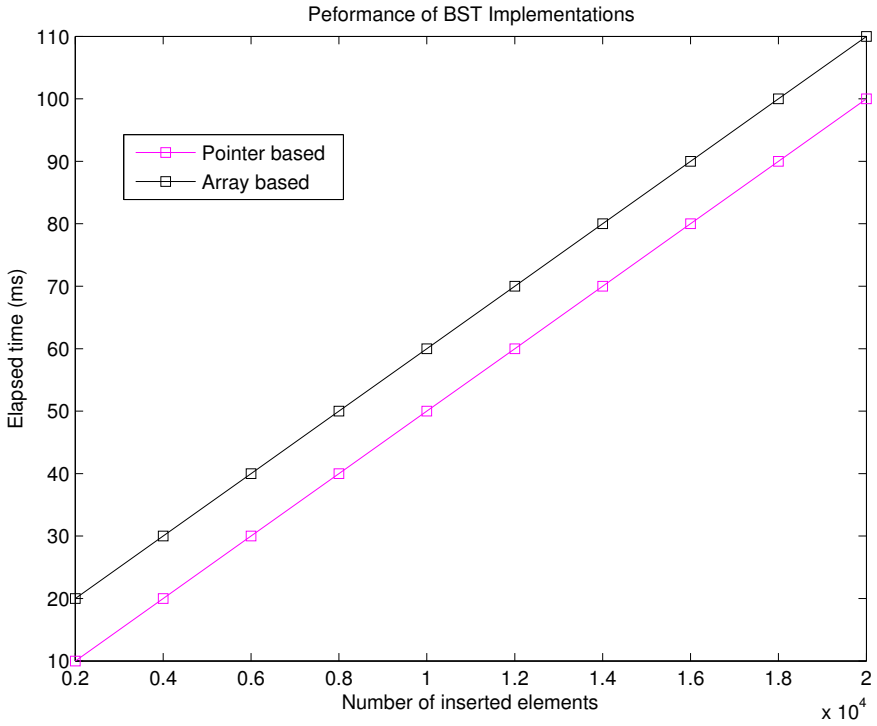
At the end, write a basic Makefile which compiles all your code and creates an executable file named `hw2`. Check out these tutorials for writing a simple make file: tutorial 1, tutorial 2.

You are free to write helper functions to accomplish the tasks required from the above functions. Use the given file names and function signatures during implementation. Please make sure that your `Makefile` works properly, otherwise you will not get any points from Question 2.

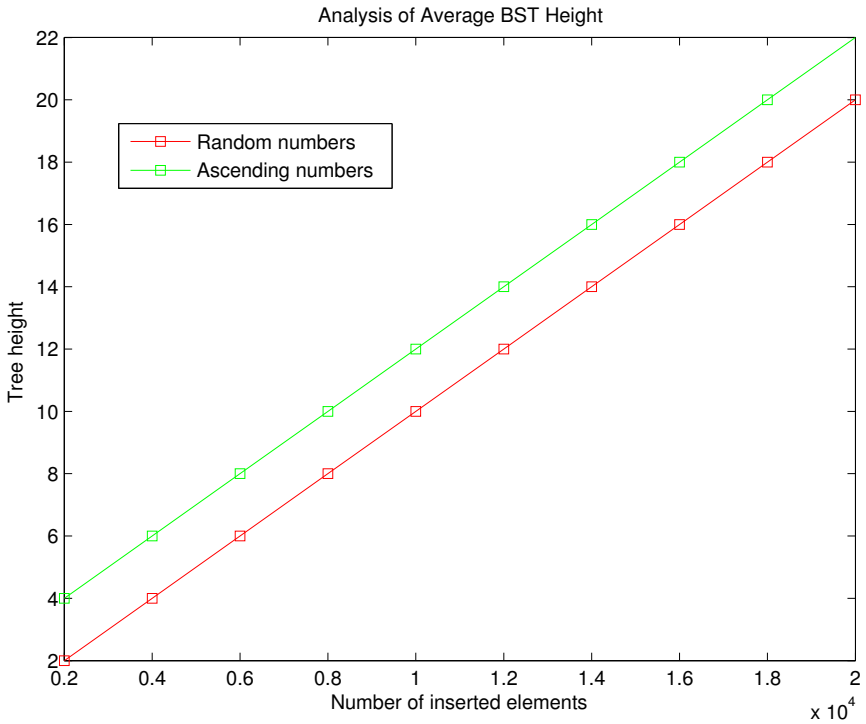
Question 3 – 15 points

After running your programs, you are expected to prepare a $3-4$ page report about the experimental results that you obtained in Question 2 e and f. First, with the help of a spreadsheet program (Microsoft Excel, Matlab or other tools), plot *number of elements* versus *elapsed time* for both implementation on the same figure. Then plot *number of elements* versus *height* for both random and ascending ordered numbers. Sample figures are given in Figure 1 (*these values do not reflect real values*).

Interpret and compare your empirical results with the theoretical ones for each implementation of BST. Explain any differences between the empirical and theoretical results, if any. Discuss which implementation of BST is better and why. In practice, what would be the average height (in terms of n) of a BST based on your excremental results? What could be done to make sure that the worst case scenario does not happen when inserting a predefined set of data into BST?



(a) Peformance of BST implementations



(b) Analysis of average BST height

Figure 1: Sample figures