CS 202, Fall 2020
Homework #2 – Binary Search Trees
Due Date: November 9, 2020

Important Notes

Please do not start the assignment before reading these notes.

- Before 23:55, November 9, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as `studentID_hw2.zip`.
- Your ZIP archive should contain the following files:
  - `hw2.pdf`, the file containing the answers to Questions 1 and 3,
  - `PbBST.h`, `PbBST.cpp`, `PbBSTNode.h`, `PbBSTNode.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).

  - 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. Thus, you may lose a significant amount of points if your C++ code does not compile or execute on the dijkstra server.

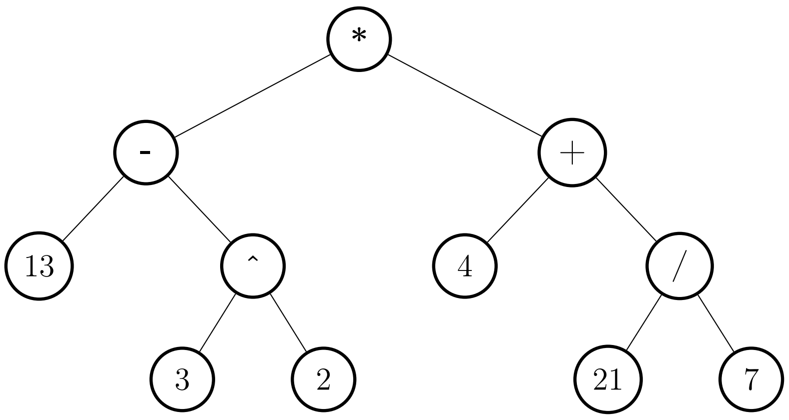
- This homework will be graded by your TA, Mubashira Zaman. Thus, please contact her directly for any homework related questions.

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 – 20 points

(a) [*5 points*] What are the preorder, inorder, and postorder traversals of the binary algebraic expression tree drawn below? Use the inorder traversal to compute the solution of the expression.



(b) [*10 points*] Insert $30, 19, 24, 66, 44, 39, 88, 63, 92, 69, 51$ into an empty binary search tree. Show **only the final tree** after all insertions. Then delete $44, 92, 19, 30$ in given order. Show the tree **after each delete operation**. Verify your answers by using this visualization tool.

(c) [*5 points*] The postorder traversal of a full binary tree is D, A, R, S, G, N, O. What is its inorder traversal? Reconstruct the tree from its traversal and draw it.

Question 2 – 65 points

Write a pointer-based implementation of Binary Search Tree named as PbBST for maintaining a list of integer keys. As mentioned in your lecture notes, use the TreeNode class for creating nodes for the tree. You are allowed to create the necessary helper functions. Put your code into PbBST.h, PbBST.cpp, PbBSTNode.h and PbBSTNode.cpp files.

(a) [*10 points*] Implement insertKey and deleteKey methods for PbBST class. Prototypes of required methods are:

```
void PbBST::insertKey(int key); // 5 points
void PbBST::deleteKey(int key); // 5 points
```

(b) [*15 points*] Implement a method `findNodesRequired` that finds the number of nodes required to convert a tree into a full binary tree of its current height. Call the methods `getHeight` and `getNodeCount` inside the `findNodesRequired` method to perform this task:

```
int PbBST::getHeight(); // 5 points
int PbBST::getNodeCount(); // 5 points
int PbBST::findNodesRequired(); // 5 points
```

(c) [*15 points*] Implement a method `mirrorTree` that swaps the right and left pointers of each node. Also write a method that prints the contents of the tree using preorder traversal. The output of the `mirrorTree` method should yield results as shown in Figure 1.

```
void PbBST::mirrorTree(); // 10 points
void PbBST::preorderTraversal(); // 5 points
```

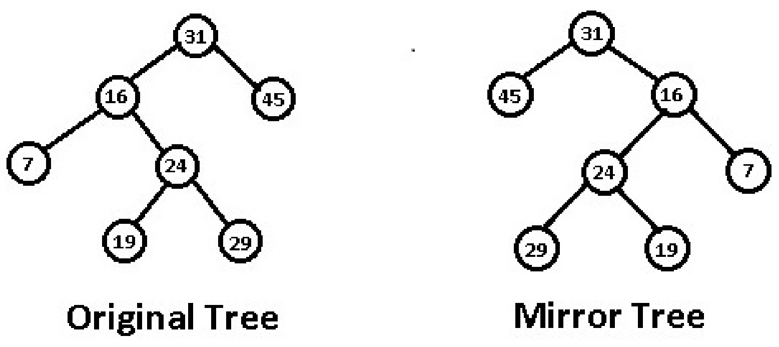


Figure 1: Tree mirroring

(d) [*10 points*] Write another method to return the median of numbers in the binary search tree in linear time (linear in the number of items). Your method should have the following prototype:

```
int medianOfBST(); // 10 points
```

(e) [*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 the height of BST changes as you insert and delete random numbers into/from the tree. Write a global function, `void heightAnalysis()`, which does the following:

(1) Creates an array of 15000 random numbers and starts inserting them into an empty pointer based BST. At each 1500 insertions, outputs the height of the tree.

(2) Shuffles the array created in part e1. Then iterates over it and deletes the numbers from the tree. After each 1500 deletions, outputs the height of the tree.

Add 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 Binary Search Tree - part 1
---------------------------------------------------------
Tree Size       Tree Height
---------------------------------------------------------
1500              x ms
3000              x ms
...

Part e - Height analysis of Binary Search tree - part 2
---------------------------------------------------------
Tree Size       Tree Height
---------------------------------------------------------
13500             x ms
12000             x ms
...
```

(f) [*5 points, mandatory*] Create a `main.cpp` file which does the following in order:

- creates a pointer based binary search tree and insert the following numbers into it: $\{42, 19, 22, 35, 56, 11, 94, 32, 28, 8, 6, 81, 63, 13, 45\}$
- calls the `findNodesRequired` method
- deletes 56 and 19 from the tree
- calls the `medianOfBST` method
- creates a mirrored copy of the tree and prints its preorder traversal

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. 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 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* after each 1500 insertions and deletions. On the same figure, plot *number of elements* versus *theoretical height* after each 1500 insertions and deletions. A sample figure is given in Figure 2 (*these values do not reflect real values*).

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.
- How would the height of the tree change if you inserted sorted numbers into it instead of randomly generated numbers?

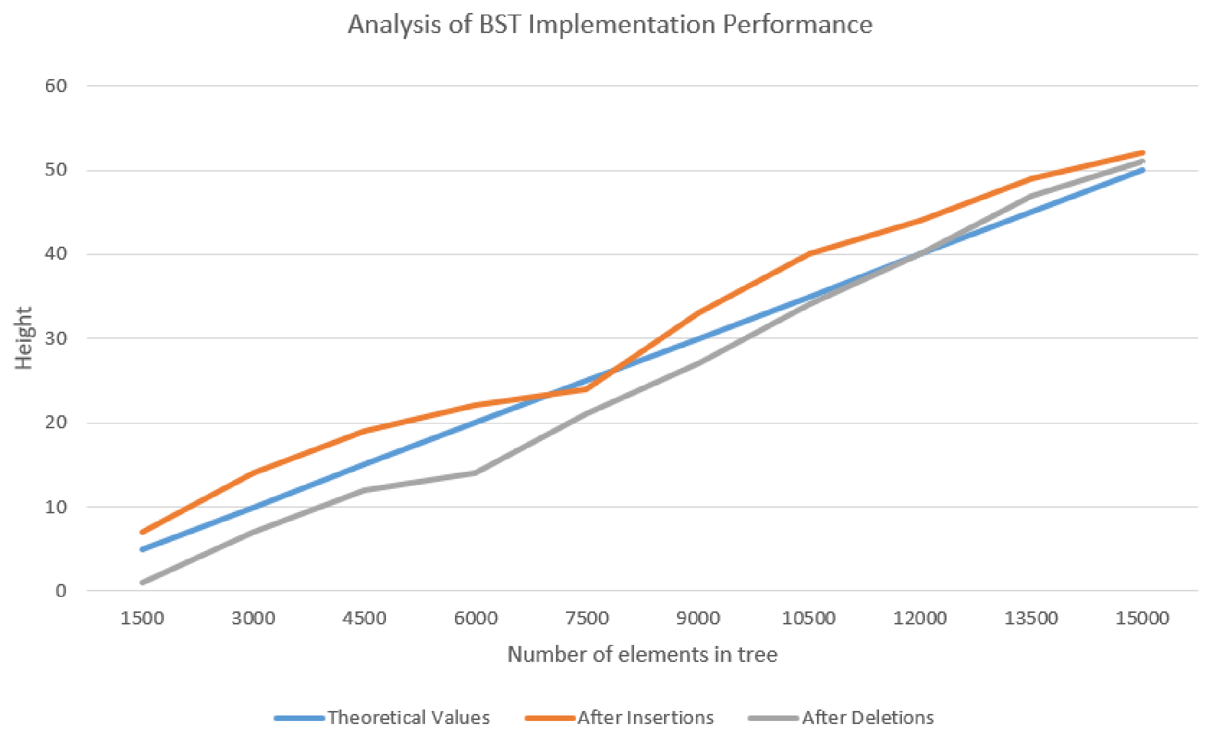


Figure 2: Sample figure for BST Performance Analysis