CS202, Summer 2025

Homework 1 - Binary Search Trees

Due: 23:55, 25/06/2025

Before you start your homework please read the following instructions carefully:

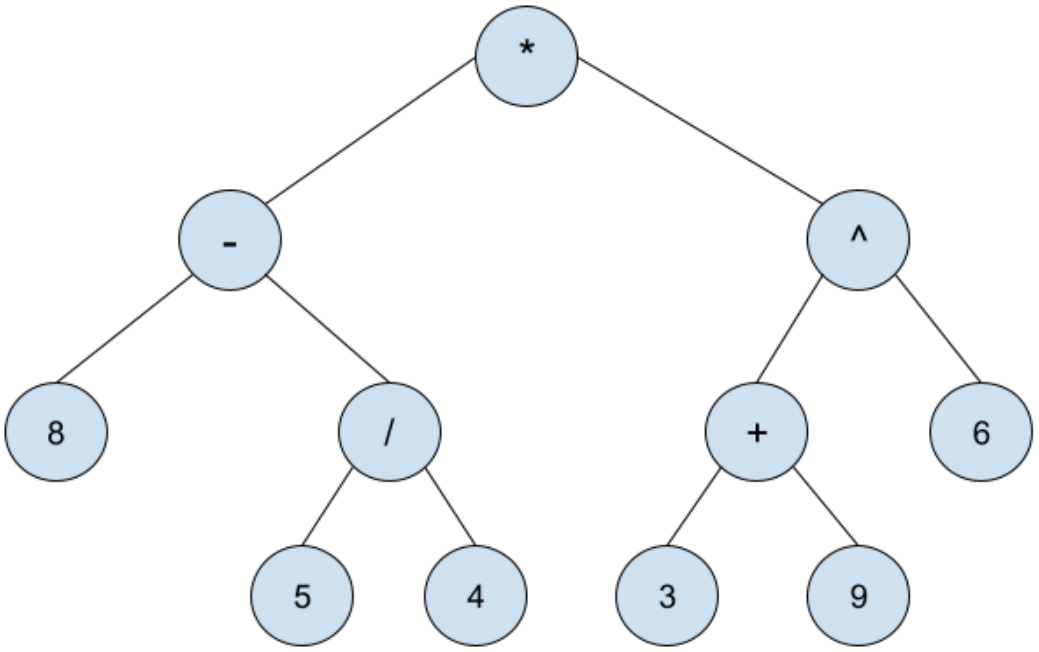
FAILURE TO FULFIL ANY OF THE FOLLOWING REQUIREMENTS WILL RESULT IN A GRADE SCORE OF 0 (zero) WITHOUT ANY CHANCE OF REDEMPTION.

- See the course page for any late submission policies and Honor Code for Assignments.
- Upload your solutions in a single ZIP archive using the Moodle submission form. Name the file as studentID_name_surname_hw1.zip.
- Your ZIP archive should contain **only** the following files:
  - **studentID_name_surname_hw1.pdf**, the file containing the answers to Questions 1, and 3.
  - In this assignment, you must have separate interface and implementation files (i.e., separate .h
  - and .cpp files) for your class. Your class name MUST BE **BST** and your file names MUST BE
  - **BST.h** and **BST.cpp**. Note that you may write additional class(es) in your solution.
  - **analysis.h** and **analysis.cpp** files related to the time analysis of Question 3.
  - Do not forget to put your name, student id, and section number in all of these files. Comment your implementation well. Add a header (see below) to the beginning of each file:

    ```
    /**
     * Title: Binary Search Trees
     * Author : Name & Surname
     * ID: 12345678
     * Section : 1
     * Homework : 1
     * Description : description of your code
     */
    ```
  - Do not put any unnecessary files such as the auxiliary files generated from your preferred IDE.
  - Please do not use **Turkish** letters in your file and folder names.
- Your code must compile.
- Your code must be complete.
- You ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL) or any other external libraries.
- The code (main function) given in Question 2 is just an example. We will test your implementation using different scenarios, which will contain different function calls. Thus, do not test your implementation only by using this example code. We recommend you to write your own driver files to make extra tests. However, you MUST NOT submit these test codes (we will use our own test code). In other words, do not submit a file that contains a function called **main**.
- We will test your code using the **google test library**. Therefore, the output message for each operation in Question 2 MUST match the format shown in the output of the example code.
- Your code must run on the **dijkstra.cs.bilkent.edu.tr** server.
- For any question related to the homework contact your TA: *saeed.karimi@bilkent.edu.tr*

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 54, 12, 87, 33, 68, 95, 25, 72, 41,21 into an empty binary search tree. Then delete 33, 72, 41, 68, 21. Show the tree after each insertion and deletion.

c) [5 points] The postorder traversal of a full binary tree is Q, A, R, F, O, C, M, K. What is its inorder traversal? Reconstruct the tree from its traversals and draw it.

Question 2 - 60 points

Implement a pointer-based Binary Search Tree whose keys are integers. Your solution must be implemented in a class called **BST**. Below is the required public part of the **BST** class. The interface for the class must be written in a file called **BST.h** and its implementation must be written in a file called **BST.cpp**. You can define additional public and private member functions and data members in this class. You can also define additional classes in your solution.

```
class BST{

public:
        BST(int keys[], int size);
        ~BST();
        void insertKey(int key);
        void deleteKey(int key);
        void displayInorder();
        void findFullBTLevel();
        void lowestCommonAncestor(int A, int B);
        void maximumSumPath();
        void maximumWidth();
        void pathFromAtoB(int A, int B);
        void mirror();

};
```

The member functions are defined as follows:
BST: Constructor. Reads the integer keys of the array one by one, and inserts them into the binary search tree one by one. You can assume that there are no identical keys in the array.
insertKey: Inserts a key to the BST, if it does not exist in the BST before.
deleteKey: Deletes a key from the BST, if it exists in the BST.
displayInorder: Displays an inorder traversal of the BST.
findFullBTLevel: Finds the maximum level at which the tree is a full binary tree, and displays the level number. Assume that root is level 1 in all the level related questions.
lowestCommonAncestor: Finds the common Ancestor of keys A and B that has the highest level. Assume that a node A is the ancestor of itself.
maximumSumPath: Finds the path from root to a leaf node which has the maximum sum of the keys included in the path, and displays the keys of the path from its root to the leaf. You can assume that there is only one path with the maximum Sum path in the tree.
maximumWidth: Finds the level of the tree which has the maximum number of nodes and displays its keys from left to right. In case of multiple maximum widths select the lowest level.
pathFromAtoB: Finds the path from key A to key B, and displays the keys included in the path.
mirror: Creates the mirror of the BST (swap left/right subtrees recursively). Then prints the preorder traversal of the mirror tree.

Figure 1 is an example of a BST tree used in our test example.

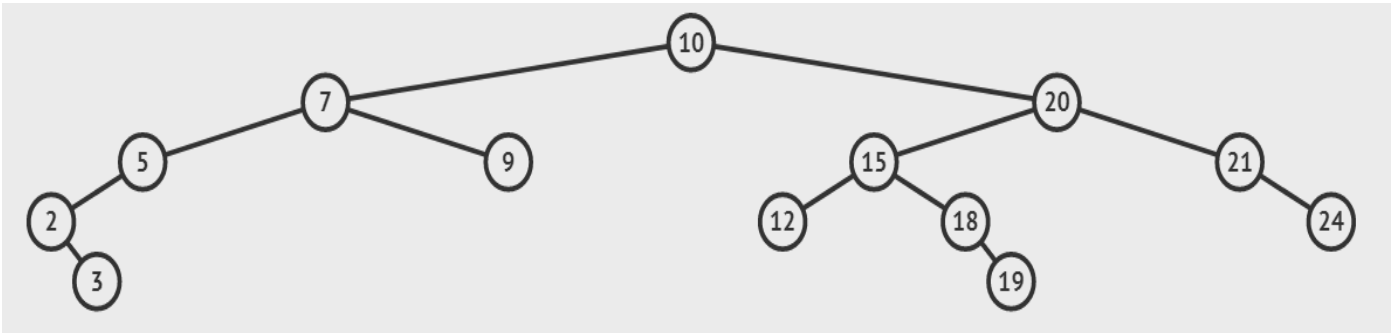


Figure 1: BST example

Example test code:

```
#include "BST.h"

int main() {

	int keys[] = {10,7,20,5,9,15,21,2,12,18,24,3,19};
	int size = 13;
	BST bst(keys, size);
	bst.findFullBTLevel();
	bst.lowestCommonAncestor(3,9);
	bst.lowestCommonAncestor(12,15);
	bst.maximumSumPath();
	bst.maximumWidth();
	bst.pathFromAtoB(2,21);
	bst.mirror();
	bst.insertKey(8);
	bst.insertKey(7);
	bst.deleteKey(10);
	bst.deleteKey(11);
	bst.displayInorder();

	return 0;
	}
```

Output of Example test code:

```
BST with size 13 is created.
Full binary tree level is: 3
Lowest common ancestor of 3 and 9 is: 7
Lowest common ancestor of 12 and 15 is: 15
Maximum sum path is: 10, 20, 15, 18, 19
Maximum level is: 5, 9, 15, 21
Path from 2 to 21 is: 2, 5, 7, 10, 20, 21
Preorder mirror is: 10, 20, 21, 24, 15, 18, 19, 12, 7, 9, 5, 2, 3
Key 8 is added.
Key 7 is not added. It exists!
Key 10 is deleted.
Key 11 is not deleted. It does not exist!
Inorder display is: 2, 3, 5, 7, 8, 9, 12, 15, 18, 19, 20, 21, 24
```

Question 3 - 20 points

In this question, you will analyze the time performance of the pointer based implementation of binary search trees. Write a function, **void timeAnalysis()** and add your code into **analysis.h** and **analysis.cpp** file. This function generates a list of 10,000 random numbers and begins inserting them into an empty pointer-based Binary Search Tree (BST).

After every 1000 insertions, displays: a) the time taken for those insertions (using the clock function from the **ctime** library to calculate elapsed time), and b) the height of the tree.

After running your program, you are expected to prepare a single-page report about the experimental results obtained from the timeAnalysis function. With the help of a spreadsheet program (Google Sheets, Matlab, or other tools):

1. Plot BST height versus the number of nodes in the BST after each period of insertions to the BST. What does this function look like? Is it a linear function or a logarithmic function? Is this the expected behaviour? Why?

2. Plot elapsed time versus the number of nodes in the tree after each period of insertions to the BST. How would the time complexity of your program change if you inserted sorted integers into it instead of randomly generated ones?