CS 575
Homework 5
Total: 100 points

The written part of assignment should be handed in person at the beginning of class on the submission date.

Written Part (25 points)

1. Show the result of inserting 13, 8, 5, 9, 4, 6, 12, 2, 1 and 3 in that order into an initially empty AVL tree. Show the tree after each insertion, clearly labeling which tree is which**. (10 points)**

2. Give pseudocode for a linear-time algorithm that verifies that an AVL tree is correctly maintained. Assume every node has fields key, data, height, left, and right and that keys can be compared with <, ==, and >. The algorithm should verify all of the following:
   - The tree is a binary search tree.
   - The height information stored in each node is correct.
   - Every node is balanced.

Your code should have a Boolean return type, and return true if the tree is a valid AVL tree, or false if it violates any of the above properties. **(15 points)**

Programming Part (75 points)

What to hand in?

1. Submit code with in-line documentation
2. Run your code on your local machine as well as on 'remote'. A readme file outlining how your code should be run.

You need to write the assignment in C++. For problems 1 to 3, use the skeleton code is provided to you.

Consider the BST.h, BST.cpp and treetester.cpp files provided to you. These files only contain the insert, search and empty member functions. Please make any changes necessary to compile and run the code.

Problem 1: (15 points)
Write new member functions called

a. **preOrder()** that implements the preorder traversal algorithm of a binary search tree. Your member function should display each node's data on the screen.

b. **inOrder()** that implements the inorder traversal algorithm of a binary search tree. Your member function should display each node's data on the screen.

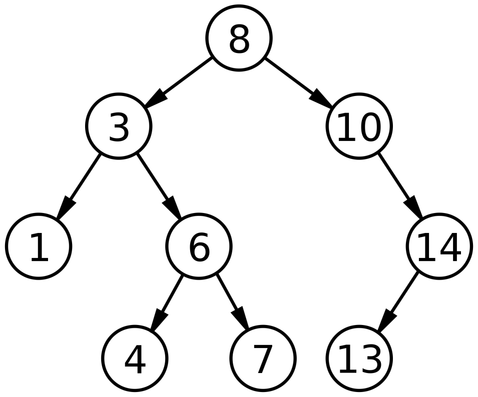
c. **nodeCount()** to count the number of nodes in a binary search tree. In this function, you should use a recursive function. You can't just use a variable such as "mySize".

Problem 2: (15 points)

In this problem, you will write the delete member function to delete a node from a BST. Recall that to delete a node from a BST you need to consider the three cases that we discussed in class.

a. The node has no children (easiest case)
b. The node has only one child (easy case)
c. The node has two children (hard case)

As an example consider the following BST. You need to create this BST by inserting these elements from the treetester.cpp file.



Delete nodes 13, 10 and 3 in order. Every time you delete a node, use the inorder() member function you wrote to show that the BST property is being maintained. Inorder() will print out the elements of a BST in sorted order.

Test your code with other BSTs to see if the delete member function is working correctly.

Problem 3: (30 points)

Modify the provided files and write additional methods to the class and convert the Binary Search Tree to an AVL tree. Note that the tree should be balanced both after insertion and deletion.

Problem 4: (15 points)

Implement and test the longest common subsequence (LCS) problem. For this problem, you are required to maintain a two dimensional arrays for storing the tables. You need to output the LCS along with the length. You need to implement the algorithm covered in the lectures. You are not allowed to search for solutions on the web.