**Lab 4**

**Goal: This lab would give you practice in using binary search trees**

In this lab, you would be writing a BinarySearchTree and BinaryTreeNode. As a refresher: the *nodes contained in the left subtree must be LESS THAN* the parent and *nodes contained in the right subtree must be GREATER* the parent (for simplicity, you can assume all the integers are unique). Here is some code to help you get started:

public class BinaryTreeNode

{

int item;

BinaryTreeNode parent;

BinaryTreeNode left;

BinaryTreeNode right;

}

As you can see, you are writing a Tree for storing integers…☺ You will be implementing the binary search tree:

public class BinarySearchTree

{

BinaryTreeNode root;

int size;

…

}

You have to implement the following functions in BinarySearchTree class:

**insert:** Should take in the integer to be inserted and return nothing. It needs to insert this int such that the tree remains a BST (see the italicized text above).

**remove:** If the node has 2 children, replace with the node of the maximum value in the left child of the node. If the value to be removed is not present, return null.

**search:** Should take in the integer to be searched for and should return a reference to the node that contains the integer or null if one is not found.

**getPreOrderStr**: Should return the pre-order String representation of the BinarySearchTree. i.e. the following tree should produce “3, 1, 0, 2, 5, 4, 6” (comma and space delimited with no comma and space at the end)

**getInOrderStr**: Should return the in-order String representation of the BinarySearchTree. i.e. the following tree should produce “0, 1, 2, 3, 4, 5, 6” (comma and space delimited with no comma and space at the end)

**getPostOrderStr**: Should return the pre-order String representation of the BinarySearchTree. i.e. the following tree should produce “0, 2, 1, 4, 6, 5, 3” (comma and space delimited with no comma and space at the end)

Diagram

Description automatically generated

**Test Cases**

Using Junit, please provide test cases that test the accuracy of your methods. You should test insert, remove, search, and your traversal functions. Remember, your tests should have 100% code coverage. If you are not close to 100% coverage within a few cases, we will deduct points.

**Grading Rubric**

* Attendance: 1pt
* Test cases: 2 pts
* insert: 2 pts
* remove: 4 pts
* search: 2 pts
* getPreOrderStr: 2 pts
* getInOrderStr: 2 pts
* getPostOrderStr: 2 pts

We will be looking to see if your methods follow the expected time complexities (all are O(n)) 😊

**Submit to Canvas (not zipped)**

* BinarySearchTree.java
* BinarySearchTreeTest.java