Binary Tree Exercise

Instructions

Check out the project called 132Spring13Week08Lab. It contains a class called BinarySearchTree. Please note that you will soon be implementing a project where the Binary Search Tree is implemented polymorphically, which is very different!

In this exercise, we use null to represent an empty tree. For example, an empty BinarySearchTree has a null root, and a leaf node has null left and right fields.

NOTE: You may not add any instance or static variables to the BinarySearchTree class. Adding auxiliary (private) methods is fine.

After writing each method below, test your code to see if it works!

1. We have written an add method that relies on an auxiliary add method (look at the code distribution). Please write the auxiliary add method. Hint: Use recursion!
2. Define a method named **height()** that returns the height of the tree.
3. Define a **recursive** method **size()** that returns the number of entries in the tree.
4. Define a recursive method named **numberOfLeavesNodes ()** that returns the number of leaf nodes in the tree.
5. Define a recursive method **getIncreasingOrderList()** that returns an ArrayList with the **data** elements of the tree inserted into the list based on increasing key order.
6. Define a **recursive** method named **preOrderTraversal** which returns a string representing a pre-order traversal of the tree.
7. Define a recursive method **isFull()** that returns true if every interior node in the tree has both a left and a right subtree.