COMP1006/1406 - Summer 2016 - Tutorial 5

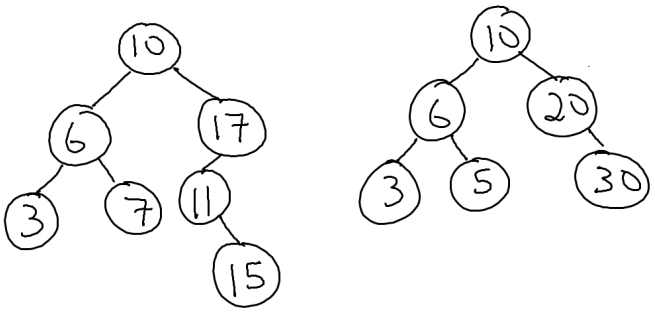
Objectives: To work with Binary (Search) Trees and Generics.

Binary Search Trees

You have seen binary trees in class. In this problem you will investigate **binary search trees**, which are special trees that satisfy the **binary search tree property**. In the following, we will assume that all trees have unique values in their nodes.

The **binary search tree property** states that for any node in the binary tree, its value is greater than all values of every node in its left subtree and its value is less than all values of every node in its right subtree.

Consider the following two binary trees:



The tree on the left is a binary search. If you look at any node all value in its left subtree are less than it and all values in its right subtree are greater. The binary tree on the right is NOT a binary search tree. In particular, the nodes with values 6 and 5 violate the binary search tree property.

As the name suggests, a binary search tree is a good data structure for searching. When searching for a item you look at the root of the tree. If you do not find it, you either look at the left or right subtree depending on if the value you are looking for is greater or smaller than the value of the root. You then repeat this recursively until you either find the item you are looking for or reach the bottom of the tree (and conclude the item is NOT in the tree).

➡ Using the BTNode class provided, implement the two missing methods: `isBST` and `search`. The first method confirms if a binary tree is a binary search tree and the second searches for a value in a binary search tree.

Generics

The binary (search) trees from the previous problem only stores an `Integer` in each node. The methods you wrote are essentially the same for any kind of data that you might store though. Using generics, you will rewrite the code using generics.

➡ Create a generic node class for binary trees called `Node`. The class should have three attributes: the data (which is generic) and two references to Nodes.

Your class should also have several constructors (like the BTNode class). Make some binary trees of with different data types.

➡ Implement generic versions of your `isBST` and `search` methods. You can make the assumption that the type `T` extends the `Comparable` interface. In order to enforce this you need to use a bounded type parameter. See

`https://docs.oracle.com/javase/tutorial/java/generics/bounded.html`

for help with this.