```
public class BinarySearchTree
{
    private TreeNode root;

    public BinarySearchTree ()
    {
        root = null;
    }

    public void insert (Comparable obj)
    {
        TreeNode newNode = new TreeNode (obj, null, null);
        if (root == null)
           root = newNode;
        else
            insertNode (root, newNode );
    }

    public void insertNode (TreeNode current, TreeNode newNode )
    {
        Comparable newValue = (Comparable ) newNode. getValue ();
        Comparable currValue = (Comparable ) current. getValue ();

        if (newValue. compareTo (currValue ) < 0)
        {
            if (current. getLeft () == null)
               current. setLeft (newNode );
            else
                insertNode (current. getLeft (), newNode );
        }
        else
        {
            if (current. getRight () == null)
               current. setRight (newNode );
            else
                insertNode (current. getRight (), newNode );
        }
    }

    public void print ()
    {
        if (root != null)
            printNodes (root);
    }

    public void printNodes (TreeNode current )
    {
        if (current != null)
        {
            printNodes (current. getLeft ());
            System .out. println (current. getValue ());
            printNodes (current. getRight ());
        }
    }


    public boolean find (Comparable key)
    {
        if (root == null)
            return false;
        else
            return findNode (root, key );
    }

    public boolean findNode (TreeNode current, Comparable key)
    {
        if (current == null)
```

```
            return  false;
        else
        {
            Comparable  currValue  =  (Comparable ) current. getValue ();
            if  (key. compareTo (currValue )  ==  0)
                return  true;
            else  if  (key. compareTo (currValue )  <  0)
                return  findNode (current. getLeft (),  key);
            else
                return  findNode (current. getRight (),  key);
        }
    }

    public  void printPostorder  ()
    {
        if  (root  !=  null)
            printNodesPostorder  (root);
    }

    public  void printNodesPostorder  (TreeNode  current )
    {
        if  (current  !=  null)
        {
            printNodes (current. getLeft ());
            printNodes (current. getRight ());
            System .out. println (current. getValue ());
        }
    }

    public  int  countNodesHelper  ()
    {
        if  (root  ==  null)
            return  0;
        return  countNodes (root);
    }

    public   int  countNodes (TreeNode  root)
    {
        if  (root  ==  null)
            return  0;

        return  1  +  countNodes (root. getRight ())  +  countNodes (root. getLeft ())
    }

    // ***********************************************
    // findMin only works if the tree stores Integer objects

    public  Object  findMinValueHelper  ()
    {
        if  (root  ==  null)
            return  0;


        return  findMinValue (root);
    }

    public  Object  findMinValue  (TreeNode  root)
    {
        if  (root. getLeft ()  ==  null)
            return  root. getValue ();

        return  findMinValue (root. getLeft ());
    }

    // ****************************************
    // treeSum only works if the tree stores Integer objects

```

```java
    public int treeSumHelper ()
    {
        if (root == null)
            return 0;

        return treeSum(root);
    }

    public static int treeSum(TreeNode root)
    {
        if (root == null)
            return 0;

        return ((Integer) root.getValue()).intValue() +
treeSum(root.getLeft()) + treeSum(root.getRight());
    }

    // ***************************************************

    public int countLeafsHelper ()
    {
        if (root == null)
            return 0;

        return countLeafs (root);
    }

    public int countLeafs (TreeNode root)
    {
        if (root == null)
            return 0;
        else if (root.getLeft() == null && root.getRight() == null)
            return 1;

        return countLeafs (root.getLeft()) + countLeafs (root.getRight());
    }

    // *******************************************

    public int treeDepthHelper ()
    {
        if (root == null)
            return 0;

        return treeDepth(root);
    }

    public  int treeDepth(TreeNode root)
    {
        int depth = 0;

        if (root == null)
            return 0;

        if (root.getRight() == null && root.getLeft() == null)
            return depth;

        if (treeDepth(root.getRight()) > treeDepth(root.getLeft()))
            return depth + treeDepth(root.getRight()) + 1;

        return depth + treeDepth(root.getLeft()) + 1;
    }

    // *******************************************

    public int internalPathLengthHelper ()
    {
        if (root == null)
            return 0;

        return internalPathLength (root, 0);
    }

    public int internalPathLength (TreeNode root, int pathsSoFar )
    {
        if (root == null)
            return 0;
        else
            return pathsSoFar +
                        internalPathLength (root.getLeft (), pathsSoFar + 1)
                        internalPathLength (root.getRight (), pathsSoFar + 1)
    }

    // ****************************************

    public static boolean areSimilar (TreeNode tree1,  TreeNode tree2 )
    {
        //  two trees are similar if they have the same exact shape and
structure

        return  true;
    }

    // ****************************************

    public int distance (TreeNode node1,  TreeNode node2 )
    {
        return 0;
    }
}
```