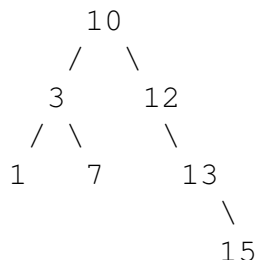


## 1 Law and Order

Write the DFS pre-order, DFS in-order, DFS post-order, and BFS traversals of the following binary search tree. For all traversals, process child nodes left to right.



## 2 Is This a BST?

(a) The following code should check if a given binary tree is a BST. However, for some trees, it returns the wrong answer. Give an example of a binary tree for which `brokenIsBST` fails.

```

public static boolean brokenIsBST(TreeNode T) {
    if (T == null) {
        return true;
    } else if (T.left != null && T.left.val > T.val) {
        return false;
    } else if (T.right != null && T.right.val < T.val) {
        return false;
    } else {
        return brokenIsBST(T.left) && brokenIsBST(T.right);
    }
}
  
```

(b) Now, write `isBST` that fixes the error encountered in part (a).

*Hint:* You will find `Integer.MIN_VALUE` and `Integer.MAX_VALUE` helpful.

```

public static boolean isBST(TreeNode T) {
    return isBSTHelper(
    );
}

public static boolean isBSTHelper(
    ) {

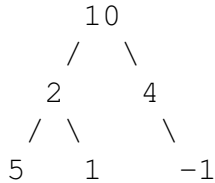
}
  
```



## 4 Extra Binary Tree Practice: Sum Paths

---

Define a root-to-leaf path as a sequence of nodes from the root of a tree to one of its leaves. Write a method `printSumPaths(TreeNode T, int k)` that prints out all root-to-leaf paths whose values sum to `k`. For example, if `T` is the binary tree in the diagram below and `k` is 13, then the program will print out `10 2 1` on one line and `10 4 -1` on another.



(a) Provide your solution by filling in the code below:

```
public static void printSumPaths(TreeNode T, int k) {
    if (T != null) {
        sumPaths(
    }
}

public static void sumPaths(TreeNode T, int k, String path) {

}

}
```

(b) What is the worst case runtime of `printSumPaths` in terms of  $N$ , the number of nodes in the tree? What is the worst case runtime in terms of  $h$ , the height of the tree?