1 Law and Order

Write the pre-order, in-order, post-order, DFS, and BFS traversal of the following binary search tree. Assume for DFS and BFS, process child nodes left to right.

    10
   /  
  3   12
  / 
 1 7 13
  
  15

2 Is This a BST?

The following code should check if a given binary tree is a BST. However, for some binary trees, it is returning the wrong answer. Think about an example of a binary tree for which the method fails. Then, write isBSTGood so that it is correct. Hint: You will find Integer.MIN_VALUE and Integer.MAX_VALUE helpful.

```java
public static boolean isBSTBad(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 isBSTBad(T.left) && isBSTBad(T.right);
    }
}

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

public static boolean isBSTHelper() {
}
```
3 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 \texttt{printSumPaths(TreeNode T, int k)} that prints out all root-to-leaf paths whose values sum to \texttt{k}. For example, if \texttt{RootNode} is the binary tree rooted in 10 in the diagram below and \texttt{k} is 13, then the program will print out \texttt{10 2 1} on one line and \texttt{10 4 -1} on another.

```
10
 /   \
2   4
 / \   \
5   1   -1
```

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

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

public static void sumPaths(TreeNode T, int k, String path) {
    // Code implementation
}
```

(b) What is the worst case running time of the \texttt{printSumPaths} in terms of \texttt{N}, the number of nodes in the tree? What is the worst case running time in terms of \texttt{h}, the height of the tree?