1 PreOrder and Friends

(a) Write the preorder, inorder, and postorder traversals of the following binary search tree.

```
    10
   / \
  3   12
 / \  /  \
1  7 13 15
```

(b) Draw the result of deleting 3 and then 10 from the binary search tree shown above (using the deletion strategy shown in lecture).

2 Is This a BST?

The following code is meant to check if a given binary tree is a binary search tree. However, for some binary trees it is returning the wrong answer.

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

(a) Give an example of a binary tree for which the method fails.
(b) Rewrite isBST so that it is correct. You may find it helpful to define a helper method.

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 printSumPaths(TreeNode T, int k) that prints out all root-to-leaf paths whose values sum to k. For example, if RootNode is the binary tree rooted in 10 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.

```
  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(
    );
}
}
```

```java
public static void sumPaths( ) {

}
```

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