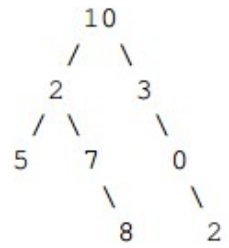
Easy

Write the preorder, inorder, and postorder traversals of the following binary tree.



Bonus questions: What is the height of a binary tree with N nodes that has the same preorder and inorder traversal? Is this also true for non-binary trees?

SomeWhat Hard

The method isBST determines whether or not a given a given TreeNode is the root of a valid BST. Assume that all values in the BST are unique.

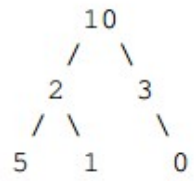
```
public class TreeNode {
    public TreeNode(int val) {
        this.val = val;
    }

    public int val;
    public TreeNode left;
    public TreeNode right;
}

public static boolean isBST(TreeNode n) {
    if (n == null) {
        return true;
    }

    if (n.left != null && n.left.val > n.val) {
        return false;
    }

    if (n.right != null && n.right.val < n.val) {
        return false;
    }

    return isBST(n.left) && isBST(n.right);
}
```

(a) Is the above method correct for all inputs? Why or why not?

(b) If you answered no to the previous part, provide a correction to isBST on another sheet of paper.

SomeWhat Harder x2

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 root, int n) that prints out all root-to-leaf paths whose values sum to n. For example, if RootNode is the binary tree rooted in 10 in the diagram below and n is 13, then the program will print out 10 2 1 on one line and 10 3 0 on another.



Provide your solution by filling in the code below:

```
void printSumPaths(Node t, int n) {
    if (t != null) {
        sumPathsHelper(                                    );
    }
}

void sumPathsHelper(                                  ) {









}
```

Bonus question: What is the worst case $\Theta(\cdot)$ runtime of your method in terms of the number of nodes Q in the given tree? The best case?

Midterm Level Question (You can expect to see this type of problem on the exam! :))

```
/** A binary tree node. The empty tree is represented by null. */
class BinNode { /** Returns my left child. */
    BinNode left() {
      ...
    }
    /** Returns my right child. */
    BinNode right() {
      ...
    }
    /** Returns my parent, or null if I am the root of the entire * tree. */
    BinNode parent() {
      ...
    }
    /** Returns my key value (label). */
    ValueType label() {
      ...
    }
    ...
}

Fill in the following functions to conform to their comments. /** True iff B is
the left child of another node. */
static boolean amLeftChild(BinNode b) {

return ________________________________;
}
```

/** Returns the node in B's tree that contains the value next larger
* than B's. Assumes that the tree I am part of is a binary search
* tree and that all its values are unique. Returns null if G is
* the node containing the largest value. Does not call label(). */

static BinNode next(BinNode b) {

}