Find largest value less than or equal to k

https://practice.geeksforgeeks.org/problems/closest-neighbor-in-bst/1?utm_source=geeksforgeeks&utm_medium=article_practice_tab&utm_campaign=article_practice_
(https://practice.geeksforgeeks.org/problems/closest-neighbor-in-bst/1?utm_source=geeksforgeeks&utm_medium=article_practice_tab&utm_campaign=article_practice_

```
TreeNode {
    left
    right
    val
}

int lessThanEqualN(TreeNode* root, int n) {

}
```

In [ ]:

```

```

Using InOrder Traversal

```
int ans = -1;
public void sol(TreeNode root , int n ){
    if(root == null) return;

    sol(root.left, n);

    if(root.val <= n){
        ans = root.val;
    } else {
        return
    }

    sol(root.right, n);
}
int lessThanEqualK(TreeNode* root, int n) {
    sol(root, n);
    return ans;
}
```

In [ ]:

```

```

```
**Check if tree is balanced**
```

```
https://leetcode.com/problems/balanced-binary-tree/
```

```
public int getHeight(TreeNode root){
        if(root == null) return 0;
        int leftHeight = getHeight(root.left);
        int rightHeight = getHeight(root.right);

        return 1 + Math.max(leftHeight , rightHeight);
    }
    public boolean isBalanced(TreeNode root) {
        if(root == null) return true;
        if(root.left == null && root.right == null) return true;

        int diffHeight = Math.abs(getHeight(root.left)-getHeight(root.right));

        if(diffHeight < 2 && isBalanced(root.left) && isBalanced(root.right)){
            return true;
        } else {
            return false;
        }
    }
```

```
def isBalanced(self, root: Optional[TreeNode]) -> bool:
        if root == None:
            return True
        def Depth(root):
            if not root:
                return 0
            return 1 + max(Depth(root.left),Depth(root.right))
        lh = Depth(root.left)
        rh = Depth(root.right)
        if abs(lh-rh) > 1:
            return False
        left = self.isBalanced(root.left)
        right = self.isBalanced(root.right)
        if left and right:
            return True
        return False
```

```
public int sol(TreeNode root , boolean [] isState){
        if(root == null) return 0;

        int leftHeight = sol(root.left , isState);
        int rightHeight = sol(root.right ,isState );

        if(Math.abs(leftHeight - rightHeight) >= 2 ) {
            isState[0] = true;
        }

        return 1 + Math.max(leftHeight , rightHeight);
    }
    public boolean isBalanced(TreeNode root) {

        boolean [ ] isState = new boolean[1];
        sol(root , isState);
        return  isState[0] == true ? false : true;
    }
```

```
class Solution {
public:
    int verify_balance(TreeNode* root, bool& res) {
        if (!root || !res) return 0;

        int lheight = verify_balance(root->left, res);
        int rheight = verify_balance(root->right, res);

        if (abs(lheight - rheight) > 1)
            res = false;

        return max(lheight, rheight) + 1;
    }
    bool isBalanced(TreeNode* root) {
        bool res = true;
        verify_balance(root, res);

        return res;
    }
};
```