In [11]:

```
class Node:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right


class Tree:
    def __init__(self):
        self.root = None

    def add(self, data):
        if self.root is None:
            self.root = Node(data)
            return

        temp = self.root
        while True:
            if temp.data <= data:
                if temp.right is None:
                    temp.right = Node(data)
                    break
                else:
                    temp = temp.right
            else:
                if temp.left is None:
                    temp.left = Node(data)
                    break
                else:
                    temp = temp.left

    def inorder(self):
        print()

        def _inorder(root):
            if root is None:
                return
            _inorder(root.left)
            print(root.data, end=' ')
            _inorder(root.right)
        _inorder(self.root)


    def preorder(self):
        print()

        def _preorder(root):
            if root is None:
                return
            print(root.data, end=' ')
            _preorder(root.left)
            _preorder(root.right)
        _preorder(self.root)

#     def remove(self, data):
```

```
    def find(self, data):

        def _find(root, data):
            if root is None:
                return False

            if root.data == data:
                return True

            if root.data < data:
                return _find(root.right, data)
            else:
                return _find(root.left, data)

        return _find(self.root, data)

t1 = Tree()
print("Find in empty tree:", t1.find(10))
t1.add(10)
t1.add(20)
t1.add(5)
t1.add(30)
t1.add(11)
t1.add(15)

t1.inorder()
t1.preorder()

print()
print("Find in tree:", t1.find(10))
print("Find in tree:", t1.find(100))
```

```
Find in empty tree: False

5 10 11 15 20 30
10 5 20 11 15 30
Find in tree: True
Find in tree: False
```

In [ ]:

In [ ]:

BST

Binary Search Tree:

- Search: searching efficient: Best Case - O(log N); Worst Case - O(N)
- Left, root, Right: Left < root < Right
- Inorder Traversal of a BST gives data in sorted order.
- TC of finding min and max element in a BST (compare to heap)

- Height : O(H)
- Numberr of nodes: Balanced O(log N) Skew: O(N)

Balanced BST:

- Uses a balancing algorithm to keep left height and right height of the tree balanced
- Ex: RB Tree, AVL Tree
- Gives worst, avg case complexity = O(log N)

In [ ]:

Search a value in BST
https://leetcode.com/problems/search-in-a-binary-search-tree/ (https://leetcode.com/problems/search-in-a-binary-search-tree/)

In [ ]:
```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode searchBST(TreeNode root, int val) {

        if(root == null)
            return null;

        if(root.val == val)
            return root;
        if(root.val < val)
            return searchBST(root.right, val);
        else
            return searchBST(root.left, val);


    }
}
```

In [ ]:
```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode searchBST(TreeNode root, int val) {
        if (root == null)
            return root;
        if (root.val == val)
            return root;
        if(root.val > val)
            return searchBST(root.left, val);
        if(root.val < val)
            return searchBST(root.right, val);
        return root;
    }
}
```

In [ ]:
```java
class Solution {
    public TreeNode searchBST(TreeNode root, int val) {
        if (root  null || root.val  val) {
            return root;
        }
        if (val < root.val) {
            return searchBST(root.left, val);
        } else {
            return searchBST(root.right, val);
        }
    }
}
```

In [ ]:
```
```

Check if tree is BST

Solution-1: Pass a range to recursive function calls

Solution-2: Do inorder traversal and compare previous value with current

https://leetcode.com/problems/validate-binary-search-tree/ (https://leetcode.com/problems/validate-binary-search-tree/)

In [ ]:
```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isValidBST(TreeNode root) {
        List<Integer> ans = new ArrayList<>();

        collect(root, ans);
        for(int i = 1; i < ans.size(); i++){
            if(ans.get(i-1) >= ans.get(i))
                return false;
        }
        return true;
    }
    private void collect(TreeNode root, List<Integer> ans){
        if(root == null)
            return;

        collect(root.left, ans);
        ans.add(root.val);
        collect(root.right, ans);
    }
}
```

In [ ]:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
 * };
 */
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        vector<int> buffer;
        inorder(root, buffer);

        for(int i = 1; i < buffer.size(); i++) {
            if (buffer[i-1] >= buffer[i])
                return false;
        }
        return true;
    }

    void inorder(TreeNode* root, vector<int> &buffer) {
        if (root == NULL) {
            return;
        }

        inorder(root->left, buffer);
        buffer.push_back(root->val);
        inorder(root->right, buffer);
    }
};
```

In [ ]:
```
public boolean isValidBST(TreeNode root) {
        if (root == null)
            return true;
        Stack<TreeNode> stack = new Stack<>();
        TreeNode pre = null;

        while (root != null || !stack.isEmpty()) {
            while (root != null) {
                stack.push(root);
                root = root.left;
            }

            root = stack.pop();
            if (pre != null && root.val <= pre.val)
                return false;
            pre = root;
            root = root.right;
        }

        return true;
    }
```

In [ ]:

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isValidBST(TreeNode root) {

        List<Integer> li = new ArrayList<>();
        answer(root, li);

        for(int i=1; i<li.size(); i++){
            if(li.get(i-1) > li.get(i)){
                return false;
            }
        }
        return true;

    }

    public void answer(TreeNode node, List<Integer> li){

        if(node == null){
            return;
        }

      answer(node.left, li);
       li.add(node.val);
       answer(node.right, li);

    }
}
```

In [ ]:
```python
class Solution:
    def isValidBST(self, root: Optional[TreeNode]) -> bool:
        prev = float('-inf')
        def inorder(node):
            nonlocal prev
            if not node:
                return True
            if not (inorder(node.left) and prev < node.val):
                return False
            prev = node.val
            return inorder(node.right)
        return inorder(root)
```

In [ ]:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
 * };
 */
class Solution {
    TreeNode* prev;
public:
    bool isValidBST(TreeNode* root) {

        prev = NULL;
        return inorder(root);
    }

    bool inorder(TreeNode* root) {
        if (root == NULL) {
            return true;
        }

        if (inorder(root->left) == false)
            return false;

        if (prev != NULL && prev->val >= root->val) {
            return false;
        }
        prev = root;


        if (inorder(root->right) == false)
            return false;

        return true;
    }
};
```

In [ ]:
```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
 * };
 */
class Solution {
    TreeNode* prev;
public:
    bool isValidBST(TreeNode* root) {

        long long min = INT_MIN;
        long long max = INT_MAX;
        min--;
        max++;
        return isValidBSTUtil(root, min, max);
    }

    bool isValidBSTUtil(TreeNode* root, long long min, long long max) {
        if (root == NULL) {
            return true;
        }

        if (root->val <= min || root->val >=max) {
            return false;
        }

        return isValidBSTUtil(root->left, min, root->val) && isValidBSTUtil(ro
    }
};
```

In [ ]:

In [ ]:

Inorder traversal and BST

Kth smallest element
Solution-1 Store inorder traversal in array
Solution-2 Simple in order traversal with a counter
https://leetcode.com/problems/kth-smallest-element-in-a-bst/
(https://leetcode.com/problems/kth-smallest-element-in-a-bst/)

In [ ]:

2 Sum in BST
https://leetcode.com/problems/two-sum-iv-input-is-a-bst/m (https://leetcode.com/problems/two-sum-iv-input-is-a-bst/m)

1. All data in hashmap: TC: O(N) SC: O(N)
2. Inorder-> put in array -> solve using 2 pointers: TC: O(N) SC:O(N)
3. Traverse Each node -> Find (k-curr.val) in tree TC: O(N log N) SC: O(log N)

In [ ]: