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
  - 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 [ ]:
```
1
```

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/)

```
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.left,val);
        } else  {
            return searchBST(root.right,val);
        }
    }
```

```cpp
class Solution {
public:
    TreeNode* searchBST(TreeNode* root, int val) {
        while(root != NULL) {
            if (val > root->val)
                root = root->right;
            else if (val < root->val)
                root = root->left;
            else
                return root;
        }
        return nullptr;
    }
};
```

```java
public TreeNode searchBST(TreeNode root, int val) {

        if(root==null)
        return null;

        if(root.val == val)
        return root;

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

        if(val>root.val)
        return searchBST(root.right,val);

        return null;
    }
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

Check if tree is BST
https://leetcode.com/problems/validate-binary-search-tree/ (https://leetcode.com/problems/validate-binary-search-tree/)

Inorder Traversal

```
    Integer prev;
    public boolean sol (TreeNode root){
        if(root == null) return true;

        Boolean left = sol(root.left);
        if(prev == null){
            prev = root.val;
        } else if(prev >= root.val)
            return false;

        prev = root.val;
        Boolean right = sol(root.right);

        return left && right;

    }
```

Recursive Inorder(JAVA)

```
    public boolean isValidBST(TreeNode root) {
            isValidBST(root, Integer.MIN_VALUE, Integer.MAX_VALUE);
        }

         public boolean isValidBST(TreeNode root, long minVal, long maxVa
    l) {
             if(root == null) {
                 return true;
             }

             if(root.val <= minVal || root.val >= maxVal) {
                 return false;
             }

             return isValidBST(root.left, minVal, root.val) && isValidBST
    (root.right, root.val, maxVal);
         }
```

Inorrder Traversal(C++)

```
class Solution {
public:
    vector<int> node_vals;

    void inorder(TreeNode* root) {
        if (root->left)
            inorder(root->left);

        node_vals.push_back(root->val);

        if (root->right)
            inorder(root->right);
    }

    bool isValidBST(TreeNode* root) {
        inorder(root);

        for (int i=0; i<node_vals.size()-1; i++) {
            if (node vals[i] >= node vals[i+1])
```

Inorder traversal and BST

Kth smallest element
https://leetcode.com/problems/kth-smallest-element-in-a-bst/ (https://leetcode.com/problems/kth-smallest-element-in-a-bst/)

```python
def kthSmallest(self, root: Optional[TreeNode], k: int) -> int:
        res= [ ]
        def helper(root):
            if root == None:return
            helper(root.left)
            res.append(root.val)
            helper(root.right)
        helper(root)
        return res[k-1]

def kthSmallest(self, root: Optionaml[TreeNode], k: int) -> int:
        stack = [ ]
        cur  = root
        while cur or stack:
            while cur:
                stack.append(cur)
                cur = cur.left
            cur = stack.pop()
            k -= 1
            if k == 0:
                return cur.val
            cur = cur.right
```

```java
    int curr = 0;
    int ans = 0;
    public void sol(TreeNode root , int k ){
        if(root == null) return;

        sol(root.left,k);
        curr++;
        if(curr==k) {
            ans = root.val;
            return;
        }

        sol(root.right,k);

    }
```

```
class Solution {
public:

    vector<int> node_vals;
    void inorder(TreeNode* root) {
        if (root->left)
            inorder(root->left);

        node_vals.push_back(root->val);b

        if (root->right)
            inorder(root->right);
    }

    int kthSmallest(TreeNode* root, int k) {
        inorder(root);
        return node_vals[k-1];
    }
};
```

In [ ]:
```
1
```

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 [ ]:
```
1
```

DIY
LCA: Lowest common ancestor
https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/ (https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/)

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeN
ode* q) {
        if (root == NULL) return NULL;

        if (root == p || root == q) {
            return root;
        }

        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);

        if (left && right) return root;
        if (left) return left;
        if (right) return right;

        return NULL;
    }
};
```

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeN
ode* q) {
        if (root == NULL) return NULL;

        if (root == p || root == q) {
            return root;
        }

        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);

        if (left && right) return root;
        return left || right;
    }
};
```

```
public TreeNode sol(TreeNode root , TreeNode p , TreeNode q){
        if(root == null) return null;
        if(root.val == p.val || root.val == q.val) return root;

        TreeNode left = sol(root.left , p, q);
        TreeNode right = sol( root.right , p , q);

        if(left != null && right != null) return root;
        if(left != null) return left;
        return right;
    }
```

In [ ]:
```
1
```

LCA in BST

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/ (https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/)

```
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode',
q: 'TreeNode') -> 'TreeNode':

        if root == None:
            return

        if (root.val > p.val  and root.val <= q.val) or (root.val <=
p.val  and root.val > q.val) or (root.val == p.val or root.val == q.v
al):
            return root

        left = self.lowestCommonAncestor(root.left, p, q)
        right = self.lowestCommonAncestor(root.right, p, q)

        return left or right
```

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeN
ode* q) {
        if (root == NULL) return NULL;

        if (root == p || root == q) {
            return root;
        }

        if (root->val > p->val && root->val > q->val)
            return lowestCommonAncestor(root->left, p, q);
        if (root->val < p->val && root->val < q->val)
            return lowestCommonAncestor(root->right, p, q);
        else
            return root;

        return NULL;
    }
};
```

In [ ]:
```
1
```

1. Find largest value less than or equal to k
2. Check if tree is balanced https://leetcode.com/problems/balanced-binary-tree/ (https://leetcode.com/problems/balanced-binary-tree/)

In [ ]:
```
1
```