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

In [ ]:
```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q
        if(root == null)
            return null;

        if(root.val > p.val && root.val > q.val)
            return lowestCommonAncestor(root.left, p, q);
        else if(root.val < p.val && root.val < q.val)
            return lowestCommonAncestor(root.right, p, q);
        else{
            return root;
        }
    }
}
```

In [ ]:
```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q
        if(root  p || root  q)
            return root;
        if(p.val<root.val && q.val>root.val)
            return root;
        if(p.val<root.val && q.val<root.val)
            return lowestCommonAncestor(root.left,p,q);
        if(p.val>root.val && q.val>root.val)
            return lowestCommonAncestor(root.right,p,q);
        return root;
    }
}
```

In [ ]:
```
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNo
        temp = root
        while temp:
            if p.val > temp.val and q.val > temp.val:
                temp = temp.right
            elif p.val < temp.val and q.val < temp.val:
                temp = temp.left
            else:
                return temp
```

In [ ]:
```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q

        if(root == null){
            return null;
        }

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

    }
}
```

In [ ]:
```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q

        if(root==null){
            return root;
        }
        int cur = root.val;
        if(cur < p.val && cur < q.val){
            return lowestCommonAncestor (root.right,p,q);
        }
        if(cur > p.val && cur > q.val){
            return lowestCommonAncestor(root.left ,p,q);
        }
        return root;
    }
}
```

In [ ]:
```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode n1, TreeNode
        if(root==null)
        return null;

        if(root==n1 || root==n2)
            return root;

        TreeNode leftLCA = lowestCommonAncestor(root.left, n1, n2);
        TreeNode rightLCA = lowestCommonAncestor(root.right, n1, n2);

        if(leftLCA!=null&&rightLCA!=null){
            return root;
        }
        if(leftLCA != null){
            return leftLCA;
        }
        return rightLCA;
    }
}
```

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 [ ]:
```
/**
 * 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 {
    int count;
    int value = 0;
    public int kthSmallest(TreeNode root, int k) {
        count = k;
          smallestKth(root);
          return value;
    }

    public void smallestKth(TreeNode node){

        if(node == null){
            return;
        }
        smallestKth(node.left);
           count--;
        if(count == 0){
            value = node.val;
        }

        smallestKth(node.right);

    }
}
```

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 {
    int val = 0;
    int ans = -1;
    public int kthSmallest(TreeNode root, int k) {
        if(root.left!=null)
            kthSmallest(root.left, k);
        val++;
        //System.out.println(root.val + ":");
        if(val == k) {
            ans = root.val;
            return root.val;
        }
        if(root.right!=null)
            kthSmallest(root.right, k);
        return ans;
    }
}
```

In [ ]:
```java
class Solution {
    List<Integer> values = new ArrayList<>();

    public int kthSmallest(TreeNode root, int k) {
        inorder(root);
        return values.get(k - 1);
    }

    private void inorder(TreeNode root) {
        if (root == null) {
            return;
        }
        inorder(root.left);
        values.add(root.val);
        inorder(root.right);
    }
}
```

In [ ]:
```java
class Solution {
    private int k;
    private int inOrder(TreeNode node) {
        if (node == null) return -1;

        int leftResult = inOrder(node.left);

        if (leftResult != -1) return leftResult;

        if (--k == 0) return node.val;

        return inOrder(node.right);
    }

    public int kthSmallest(TreeNode root, int k) {
        this.k = k;
        return inOrder(root);
    }
}
```

In [ ]:
```java
class Solution {
    private int k;
    private int res;
    private int inOrder(TreeNode node) {
        if (node == null || k == 0) return;

        inOrder(node.left);
        if (--k == 0) this.res = node.val;
        inOrder(node.right);
    }
    public int kthSmallest(TreeNode root, int k) {
        this.k = k;
        return this.res;
    }
}
```

In [ ]:
```python
class Solution:
    def kthSmallest(self, root: Optional[TreeNode], k: int) -> int:
        values = []
        self.inorder(root, values)
        return values[k - 1]

    def inorder(self, root, values):
        if root is None:
            return
        self.inorder(root.left, values)
        values.append(root.val)
        self.inorder(root.right, values)
```

In [ ]:

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 [ ]:
```
Option-3

/**
 * 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 findTarget(TreeNode* root, int k) {
        return findUtil(root, root, k);
    }

    bool findUtil(TreeNode* root, TreeNode *treeRoot, int k) {
        if (root == NULL) {
            return false;
        }

        if (find(treeRoot, root, k-root->val)) {
            return true;
        }

        return findUtil(root->left, treeRoot, k) || findUtil(root->right, tree
    }

    bool find(TreeNode * root, TreeNode *curr, int val) {
        if (root == NULL) {
            return false;
        }

        if (root->val == val && curr != root) {
            return true;
        }
        return find(root->left, curr, val) || find(root->right,curr, val);
    }
};
```