Index - 5

1123. Lowest Common Ancestor of Deepest Leaves ====> Do it

Binary Search Tree

1. Insert a key into BST
2. Search in a Binary Search Tree
3. Delete a node from BST
4. Construct Binary Search Tree from Preorder Traversal
5. Check for BST
6. Connect Nodes at Same Level
7. LCA of BST
8. Inorder Predecessor and Successor in BST
9. BST from Sorted Array
10. Find Ceil of a number in BST
11. Find Floor of a number in BST
12. Find the Closest Element in BST
13. K-th smallest element in BST
14. Kth largest element in BST
15. Binary Tree to DLL
16. Find a pair with given target in BST
17. Largest BST in BT

1) Insert a key into BST

No Duplicates should exist in it.

```
Node insert(Node root, int Key)
   {
      if(root==null){
         return new Node(Key);
      }

      if(root.data>Key){
         root.left = insert(root.left,Key);
      }
      else if(root.data<Key){
         root.right = insert(root.right,Key);
      }
      return root;
   }
```

T.C = O(h) = O(logn)
S.C = 1

2) Search in a Binary Search Tree

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

```
        if(root==null)
            return root;
        if(root.val==val)
            return root;
        else if(root.val>val)
            return searchBST(root.left, val);
        else
            return searchBST(root.right, val);
    }
```

T.C = O(h) = O(logn)
S.C = 1

3) Delete a node from BST

```
 public TreeNode deleteNode(TreeNode root, int key) {
        if(root==null)
            return root;

        if(root.val>key)
            root.left = deleteNode(root.left, key);
        else if(root.val<key)
            root.right = deleteNode(root.right, key);
        else{
            if(root.left==null)
                return root.right;
            else if(root.right==null)
                return root.left;

            root.val = inorderSuccessor(root.right);
            root.right = deleteNode(root.right, root.val);
        }
        return root;
    }
    int inorderSuccessor(TreeNode root){
        TreeNode cur = root;
        while(cur.left!=null)
            cur = cur.left;
        return cur.val;
    }
```

T.C = O(h) = O(logn)
S.C = 1

4) Construct Binary Search Tree from Preorder Traversal

```
 int pIndex;
    public TreeNode bstFromPreorder(int[] preorder) {
        pIndex = 0;

        return buildTree(preorder, preorder[pIndex],Integer.MIN_VALUE,Integer.MAX_VALUE);
```

```
    }
    public TreeNode buildTree(int pre[],int key, int min, int max){
        if(pIndex>=pre.length)
            return null;

        TreeNode root = null;
        if(min<key && key<max){
            root = new TreeNode(key);
            pIndex++;
            if(pIndex<pre.length){
                root.left = buildTree(pre, pre[pIndex],min, key);
            }
            if(pIndex<pre.length)
                root.right = buildTree(pre, pre[pIndex],key, max);
        }
        return root;
    }
```

T.C = O(n)
S.C = 1

5) Check for BST

i. Using max and min functions

```
int isBST(struct node* node)
{
  if (node == NULL)
    return 1;

  /* false if the max of the left is > than us */
  if (node->left!=NULL && maxValue(node->left) > node->data)
    return 0;

  /* false if the min of the right is <= than us */
  if (node->right!=NULL && minValue(node->right) < node->data)
    return 0;

  /* false if, recursively, the left or right is not a BST */
  if (!isBST(node->left) || !isBST(node->right))
    return 0;

  /* passing all that, it's a BST */
  return 1;
}
```

Time Complexity is more because we are calculating min and max for every node. Min and max return min and max values in the trees

It is inefficient

ii. Passing the min and max values possible for each node.

We can pass Integer.MAX AND MIN_VALUES but it will fail in corner cases. So send the nodes it self.

```
boolean isBST(Node root)
   {
      return checkBst(root, null, null);
   }
   boolean checkBst(Node root,Node l,Node r){
      if(root==null)
         return true;

      if(l!=null && root.data<=l.data)
         return false;
      if(r!=null && root.data>=r.data)
         return false;


      return checkBst(root.left ,l, root) && checkBst(root. right, root, r);
   }
```

T.C = O(n)
S.C = 1

6) Connect Nodes at Same Level

```
public void connect(Node root)
   {
      if(root==null)
         return;
      Queue<Node> q = new LinkedList<>();
      q.add(root);

      while(!q.isEmpty()){
         int size = q.size();
         Node temp = q.peek();
         for(int i=0;i<size ;i++){
            temp = q.remove();

            if(temp.left!=null)
               q.add(temp.left);
            if(temp.right!=null)
               q.add(temp.right);
            if(!q.isEmpty());
               temp.nextRight = q.peek();
         }
         temp.nextRight = null;
      }

   }
```

T.C = O(n)
S.C = 1

7) LCA of BST

```
Node LCA(Node root, int n1, int n2)
{
        if(root==null)
            return root;

        if(root.data>n1 && root.data>n2)
            return LCA(root.left, n1,n2);

        if(root.data<n1 && root.data<n2)
            return LCA(root.right, n1,n2);

      return root;
}
```

T.C = O(logn) = O(h)
S.C = 1

8) Inorder Predecessor and Successor in BST

Algorithm:

- If root is null then return
- If root.data> k then k is in Left subtree, also root can be the successor of k ,if k is leaf node. So update succ
- If root.data<k then k is in right subtree, also root can be the predecessor ok, if k is leaf node. So update pred.
- If root.data==k, if(root.left!=null) then predecessor is rightmost node of left subtree and if(root.right!=null) then successor is leftmost node of right subtree.

// Recursion

```
Node pre = null, succ = null;
//pre and succ is declared in main function;

void findPreSuc(Node root, Node pre, Node succ, int key)
{
        if(root==null)
            return;

        //key in left subtree
        if(root.data>key)
        {
            succ = root;
            findPreSuc(root.left, pre, succ, key);
        }

        else if(root.data<key)
        {
            pre = root;
            findPreSuc(root.right, pre, succ, key);
        }

        else
```

```
        {
                if(root.left!=null)
                {
                        Node temp = root.left;
                        while(temp.right!=null)
                                temp = temp.right;
                        pre = temp;
                }
                if(root.right!=null)
                {
                        Node temp = root.right;
                        while(temp.left!=null)
                                temp = temp.left;
                        succ = temp;
                }
        }
}
```

// Iterative

```
void findPreSuc(Node root, Node pre, Node suc, int k)
{
        if(root==null)
                return ;

        Node node = root;

        while(node!=null)
        {
                if(node.data>k)
                {
                        suc = node;
                        node = node.left;
                }
                else if(root.data<k)
                {
                        pre = node;
                        node = node.right;
                }
                else
                {
                        if(node.left!=null)
                        {
                                Node temp = node.left;
                                while(temp.right!=null)
                                        temp = temp.right;

                                pre = temp;
                        }
                        if(node.right!=null)
                        {
```

```
                                Node temp = node.right;
                                while(temp.left!=null)
                                        temp = temp.left;

                                suc = temp;
                        }
                }
        }
}
```

T.C = O(logn) = O(h)
S.C = 1

9) BST from Sorted Array

```
 int i;
    public int[] sortedArrayToBST(int[] nums)
    {
        Node root = buildTree(nums, 0, nums.length-1);
        i = 0;
        int answer[] = new int[nums.length];
        preorder(root ,answer);
        return answer;
    }
    Node buildTree(int a[],int start ,int end){
        if(start>end)
            return null;

        int mid = start + (end-start)/2;
        int cur = a[mid];
        Node root = new Node(cur);

        if(start==end)
            return root;

        root.left = buildTree(a, start, mid-1);
        root.right = buildTree(a, mid+1, end);

        return root;
    }
    void preorder(Node root, int ans[]){
        if(root==null)
            return;

        ans[i++] = root.data;
        preorder(root.left, ans);
        preorder(root.right, ans);
    }
```

T.C = O(n)
S.C = O(n)

10) Find Ceil of a number in BST

```
int findCeil(Node root, int k)
{
        if(root==null)
                return Integer.MIN_VALUE;
        if(root.data==k)
                return root.data;

        if(root.data<k)
                return findCeil(root.right, k);

        int ceil = findCeil(root.left, k);

        return (ceil>=k)?ceil : root.data;
}
```

T.C = O(h)
S.C = O(1)

11) Find Floor of a number in BST

```
int findFloor(Node root, int k)
{
        if(root==null)
                return Integer.MAX_VALUE;

        if(root.data==k)
                return k;

        if(root.data>k)
                return findFloor(root.left, k);

        int floor = findFloor(root.right, k)

        return (floor<=k)?floor:root.data;
}
```

T.C = O(h)
S.C = O(1)

12) Find the Closest Element in BST

Return the absolute difference given key and the closest element of it in Bst

```
static int maxDiff(Node  root, int k)
   {
      int c = findCeil(root, k);        // Get Ceil from above function
      int f = findFloor(root, k);       // Get Floor from above function
```

```
        if(c==Integer.MIN_VALUE)
            return k-f;
        else if(f==Integer.MAX_VALUE)
            return c-k;

        return Math.min(c-k, k-f);
    }
```

T.C = O(h)
S.C = O(1)

13) K-th smallest element in BST

i. Do inorder traversal and return kth element. Here we need extra space of O(n)

ii. Morris Traversal

```
 public int KthSmallestElement(Node root, int k)
   {
        Node cur = root;
        int kthSmallest = -1;
        int count = 0;

        while(cur!=null){
            if(cur.left==null){
                count++;
                if(count==k){
                    kthSmallest = cur.data;
                    break;
                }
                cur = cur.right;
            }
            else{
                Node predecessor = cur.left;
                while(predecessor.right!=null && predecessor.right!=cur)
                    predecessor = predecessor.right;

                if(predecessor.right!=cur){
                    predecessor.right = cur;
                    cur = cur.left;
                }
                else{
                    predecessor.right = null;
                    count++;
                    if(count==k){
                        kthSmallest = cur.data;
                        break;
                    }
                    cur = cur.right;
                }
            }
```

```
        }
        return kthSmallest;
    }
```

14) Kth largest element in BST

```
public int kthLargest(Node root, int k)
    {
        int answer = -1;
        Node cur = root;
        int count = 0;

        while(cur!=null){
            if(cur.right==null){
                count++;
                if(count==k){
                    answer = cur.data;
                    break;
                }
                cur = cur.left;
            }
            else{
                Node successor = cur.right;

                while(successor.left!=null && successor.left!=cur)
                    successor = successor.left;

                if(successor.left==null){
                    successor.left = cur;
                    cur = cur.right;
                }
                else{
                    count++;
                    successor.left = null;
                    if(count==k){
                        answer = cur.data;
                        break;
                    }
                    cur = cur.left;
                }
            }
        }
        return answer;
    }
```

T.C = O(n) =====> Every edge is atmost traversed twice.
S.C = O(1)

15) Binary Tree to DLL

i. **Do Inorder traversal and create a new double Linked List. It takes extra space complexity of O(n)**

ii.

```
Node prev ;
Node head;

Node bToDLL(Node root)
{
        prev = null;
        head = null;

        convertToDoubleList(root);
        return head;
}
void convertToDoubleList(Node root)
{
        if(root==null)
                return;

        convertToDoubleList(root.left);

        if(pre==null)
        {
                prev = root;
                head = root;
        }
        else
        {
                root.left = prev;
                prev.right = root;
                prev = root;
        }

        convertToDoubleList(root.right);
}
```

T.C = O(n)
S.C = O(1)

16) Find a pair with given target in BST

i. **Using Hash Set recursively solve it(Since all values are unique we get answer)**

```
HashSet<Integer> s = new HashSet<>();

boolean checkPair(Node root, int k)
{
        if(root==null)
                return false;
```

```
        if(s.contains(k-root.data))
                return true;

        s.add(root.data);

        return checkPair(root.left, k) || checkPair(root.right, k);
}
```

ii. Use inorder traversal and then two pointer approach

```
 public int isPairPresent(Node root, int target)
   {
      ArrayList<Integer> a = new ArrayList<>();
      inorder(root, a);

      int i=0, j = a.size()-1;

      while(i<j){
         int x = a.get(i);
         int y = a.get(j);
         if(x+y==target)
            return 1;
         else if(x+y<target)
            i++;
         else
            j--;
      }
      return 0;
   }
```

For Above both approaches

T.C = O(n)
S.C = O(n)

iii. Also it can be solved by converting tree into DLL and use two pointer approach But Tree Modification is needed

T.C = O(n)
S.C = O(1)

iv. Using Two Stacks and use two pointer approach

```
 public boolean findTarget(TreeNode root, int k) {
      if(root==null)
         return false;
      Stack<TreeNode> s1 = new Stack<>();
      Stack<TreeNode> s2 = new Stack<>();
      pushLeft(root, s1);
      pushRight(root, s2);
```

```
        while(!s1.isEmpty() && !s2.isEmpty() && s1.peek().val<s2.peek().val){
            int x = s1.peek().val;
            int y = s2.peek().val;

            if(x+y==k)
                return true;
            else if(x+y<k){
                TreeNode temp = s1.pop();
                pushLeft(temp.right, s1);
            }
            else{
                TreeNode temp = s2.pop();
                pushRight(temp.left, s2);
            }
        }
        return false;
    }
    void pushLeft(TreeNode root,Stack<TreeNode> s){
        while(root!=null){
            s.push(root);
            root = root.left;
        }
    }
    void pushRight(TreeNode root,Stack<TreeNode> s){
        while(root!=null){
            s.push(root);
            root = root.right;
        }
    }
```

T.C = O(n)
S.C = O(h)

17) Largest BST in BT

i. Inorder traversal approach

For every Node find the inorder traversal and check if the traversal is sorted if yes then max size of all sorted inorder traversal is answer

T.C = O(n^2)
S.C = O(n)

ii. Having the knowledge of the left and right subtree status. If left or right sub tree is not BST then no other subtree above root is also a BST.

```
class Info{
        int size, int min, int max, int ans;
        boolean isBst;
```

```
Info()
{
}
Info(int s, int i, int a, int answer, boolean bst)
{
        size = s;
        min = i;
        max = a;
        ans = answer;
        isBst = bst;
}

int largestBst(Node root)
{
        return findLargestBst(root).ans;
}

Info findLargestBst(Node root)
{
        if(root==null)
                return new Info(0, Integer.MAX_VALUE, Integer.MIN_VALUE, 0, true);

        if(root.left==null && root.right==null)
                return new Info(1, root.data, root.data, 1, true);

        Info leftValue = findLargestBst(root.left);
        Info rightValue = findLargestBst(root.right);

        Info cur = new Info();

        cur.size = 1 + leftValue.size + rightValue.size;

        if(leftValue.isBst && rightValue.isBst && leftValue.max<root.data &&
                                                        root.data<rightValue.min)
        {
             cur.min = Math.min(root.data, Math.min(leftValue.min, rightValue.min));
            cur.max = Math.max(root.data, Math.max(leftValue.max, rightValue.max));

            cur.ans = cur.size;
            cur.isBst = true;
            return cur;
        }

        cur.ans = Math.max(leftValue.ans, rightValue.ans);

        return cur;
}
```

To calculate the maximum sum BST in this, keep a global variable max, and in info make size as sum and ans as summation of all values in bst. And in every recursive call let maximum holds the max of maximum and cur.ans. This will give the result.

T.C = O(n)
S.C = O(1)