Lab 9

Bst

1)700

class Solution {

    public TreeNode searchBST(TreeNode root, int val) {

        if(root == null){

            return null;

        }

        else if(root.val == val){

            TreeNode ans = root;

            return ans;

        }

        else if(root.val > val){

           return searchBST(root.left, val);

        }

        return searchBST(root.right,val);

    }

}

2)701

class Solution {

    public TreeNode insertIntoBST(TreeNode root, int val) {

        if(root == null) return new TreeNode(val);

        if(val > root.val) root.right = insertIntoBST(root.right, val);

        if(val < root.val) root.left = insertIntoBST(root.left, val);

        return root;

    }

}

3)450

class Solution {

    public TreeNode deleteNode(TreeNode root, int key) {

        if(root==null) return root;

        if(key<root.val){

            root.left=deleteNode(root.left,key);

        }

        else if(key>root.val){

            root.right=deleteNode(root.right,key);

        }

        else{

            if(root.left==null) return root.right;

            if(root.right==null) return root.left;

            root.val=minvalue(root.right);//to find inorder succesor

            root.right=deleteNode(root.right,root.val);

        }

        return root;

    }

    public int minvalue(TreeNode root){

        int min=root.val;

        while(root.left!=null){

            min=root.left.val;

            root=root.left;

        }

        return min;

    }

}

4)230

class Solution {

    ArrayList<Integer> ls = new ArrayList<>();

    private void Inorder(TreeNode root, ArrayList ls){

        if(root == null){

            return;

        }

        Inorder(root.left,ls);

        ls.add(root.val);

        Inorder(root.right,ls);

        return;

    }

    public int kthSmallest(TreeNode root, int k) {

        Inorder(root,ls);

        return ls.get(k-1);

    }

}

5)98

class Solution {

    private long minVal = Long.MIN\_VALUE;

    public boolean isValidBST(TreeNode root) {

        if (root == null) return true;

        if (!isValidBST(root.left)) return false;

        if (minVal >= root.val) return false;

        minVal = root.val;

        if (!isValidBST(root.right)) return false;

        return true;

    }

}

6)235

class Solution {

    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

        if(root == null || root == p || root == q){

            return root;

        }

        TreeNode left = lowestCommonAncestor(root.left, p, q);

        TreeNode right = lowestCommonAncestor(root.right, p, q);

        if(left == null){

            return right;

        }else if(right == null){

            return left;

        }

        return root;

    }

}

7)1008

class Solution {

    public TreeNode bstFromPreorder(int[] preorder) {

     return helper(preorder, 0, preorder.length - 1);

    }

    private TreeNode helper(int[] preorder, int start, int end) {

        if(start > end) return null;

        TreeNode node = new TreeNode(preorder[start]);

        int i;

        for(i=start;i<=end;i++) {

        if(preorder[i] > node.val)

            break;

        }

        node.left = helper(preorder, start+1, i-1);

        node.right = helper(preorder, i, end);

        return node;

    }

}

8)173

public class BSTIterator {

    private Stack<TreeNode> stack = new Stack<TreeNode>();

    public BSTIterator(TreeNode root) {

        pushAll(root);

    }

    /\*\* @return whether we have a next smallest number \*/

    public boolean hasNext() {

        return !stack.isEmpty();

    }

    /\*\* @return the next smallest number \*/

    public int next() {

        TreeNode tmpNode = stack.pop();

        pushAll(tmpNode.right);

        return tmpNode.val;

    }

    private void pushAll(TreeNode node) {

        for (; node != null; stack.push(node), node = node.left);

    }

}

9)653

class Solution {

    public boolean findTarget(TreeNode root, int k) {

        List<Integer> nums = new ArrayList<>();

        inorder(root, nums);

        for(int i = 0, j = nums.size()-1; i<j;){

            if(nums.get(i) + nums.get(j) == k)return true;

            if(nums.get(i) + nums.get(j) < k)i++;

            else j--;

        }

        return false;

    }

    public void inorder(TreeNode root, List<Integer> nums){

        if(root == null)return;

        inorder(root.left, nums);

        nums.add(root.val);

        inorder(root.right, nums);

    }

}

10)99

class Solution {

    private TreeNode first = null;

    private TreeNode second = null;

    private TreeNode pre = null;

    public void recoverTree(TreeNode root) {

        if(root==null) return;

        inorder(root);

        int temp = first.val;

        first.val = second.val;

        second.val = temp;

    }

    private void inorder(TreeNode root){

        if(root==null) return;

        inorder(root.left);

        if(first==null && (pre==null ||pre.val>=root.val)){

            first = pre;

        }

        if(first!=null && pre.val>=root.val){

            second = root;

        }

        pre = root;

        inorder(root.right);

    }

}