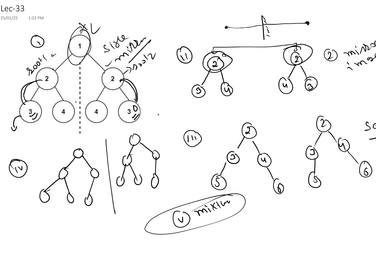
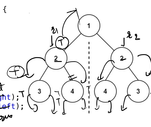
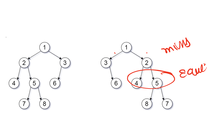
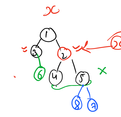
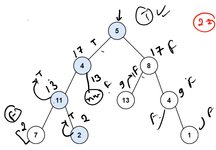


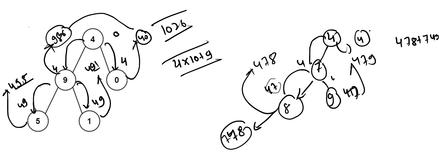

```
public boolean Symmetric(TreeNode root1, TreeNode root2) {
        if (root1 == null && root2 == null) {
            return true;
        }
        if (root1 == null || root2 == null) {
            return false;
        }
        if (root1.val != root2.val) {
            return false;
        }
        boolean left = Symmetric(root1.left, root2.right);
        boolean right = Symmetric(root1.right, root2.left);
        return left && right;
    }
```



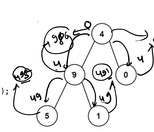


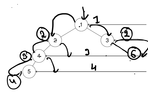


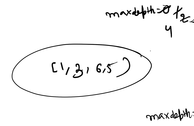




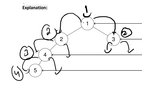
```
public int Numbers(TreeNode root, int sum) {
        if (root == null) {
            return 0;
        }
        if (root.left == null && root.right == null) {
            return sum * 10 + root.val;
        }
        int left = Numbers(root.left, sum * 10 + root.val);
        int right = Numbers(root.right, sum * 10 + root.val);
        return left + right;
    }
```

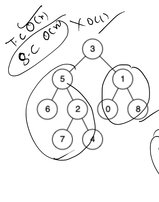


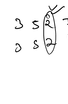





```
public void rightView(TreeNode root, int curr_level) {
        if (root == null) {
            return;
        }
        if (curr_level > max_Depth) {
            // list me add
            max_Depth = curr_level;
        }
        rightView(root.right, curr_level + 1);
        rightView(root.left, curr_level + 1);
    }
```

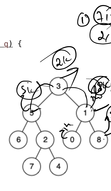


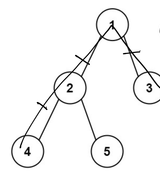


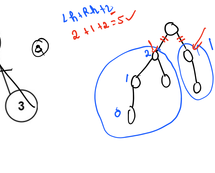


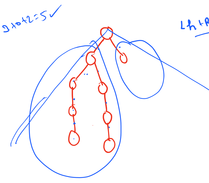

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if (root == null) {
            return null;
        }
        if (root == p || root == q) {
            return root;
        }
        TreeNode left = lowestCommonAncestor(root.left, p, q);
        TreeNode right = lowestCommonAncestor(root.right, p, q);
        if(left!=null && right!=null) {
            return root;
        }
        else if(left==null) {
            return right;
        }
        else {
            return left;
        }

    }
```









```
public int diameterOfBinaryTree(TreeNode root) {
        if (root == null) {
            return 0;
        }

        int ld = diameterOfBinaryTree(root.left);
        int rd = diameterOfBinaryTree(root.right);
        int td = ht(root.left) + ht(root.right) + 2;
        return Math.max(td, Math.max(ld, rd));

    }

    public int ht(TreeNode root) {
        if (root == null) {
            return -1;
        }
        int lh = ht(root.left);
        int rh = ht(root.right);
        return Math.max(lh, rh) + 1;
    }
```