**[104. Maximum Depth of Binary Tree](https://leetcode.com/problems/maximum-depth-of-binary-tree/)**

class Solution {

    public int maxDepth(TreeNode root) {

        return md(root);

    }

    private int md(TreeNode root){

        if(root == null) return 0;

        int ld = md(root.left);

        int rd = md(root.right);

        return Math.max(ld, rd)+1;

    }

}

[**112. Path Sum**](https://leetcode.com/problems/path-sum/)

class Solution {

    public boolean rootToLeafPathSum(TreeNode root, int targetSum, int sum){

        if(root == null)

            return false;

        if(root.left == null && root.right == null){

            sum = sum + root.val;

            if(sum == targetSum)

                return true;

        }

        return rootToLeafPathSum(root.left, targetSum, sum + root.val) || rootToLeafPathSum(root.right, targetSum, sum + root.val);

    }

    public boolean hasPathSum(TreeNode root, int targetSum) {

        int sum = 0;

        return rootToLeafPathSum(root, targetSum, sum);

    }

}

[**105. Construct Binary Tree from Preorder and Inorder Traversal**](https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/)

class Solution {

    public TreeNode buildTree(int[] preorder, int[] inorder) {

        Map<Integer , Integer> inMap = new HashMap<Integer , Integer>() ;

        for(int i = 0 ; i < inorder.length ; i++){

            inMap.put(inorder[i] , i);

        }

        TreeNode root = buildBinaryTree(preorder , 0 , preorder.length - 1 , inorder , 0 , inorder.length - 1 , inMap) ;

        return root ;

    }

    public TreeNode buildBinaryTree(int[] preorder , int preStart , int preEnd , int[] inorder , int inStart , int inEnd , Map<Integer , Integer> inMap){

        if(preStart > preEnd || inStart > inEnd) return null ;

        TreeNode root = new TreeNode(preorder[preStart]);

        int inRoot = inMap.get(root.val) ;

        int numsLeft = inRoot - inStart ;

        root.left = buildBinaryTree(preorder , preStart + 1 , preStart + numsLeft , inorder , inStart , inRoot - 1 , inMap);

        root.right = buildBinaryTree(preorder , preStart + numsLeft + 1 , preEnd , inorder , inRoot + 1 , inEnd , inMap);

        return root ;

    }

}

[**114. Flatten Binary Tree to Linked List**](https://leetcode.com/problems/flatten-binary-tree-to-linked-list/)

class Solution {

    public void flatten(TreeNode root) {

        if(null == root) return;

        flatten(root.left);

        flatten(root.right);

        TreeNode right = root.right;

        TreeNode left = root.left;

        if(null == left) return;

        while(left.right != null) left = left.right;

        root.right = root.left;

        left.right = right;

        root.left = null;

    }

}

[**102. Binary Tree Level Order Traversal**](https://leetcode.com/problems/binary-tree-level-order-traversal/)

class Solution {

    public List<List<Integer>> levelOrder(TreeNode root) {

        Queue<TreeNode> q = new LinkedList<>();

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

        if(root==null){

            return finalAns;

        }

        q.add(root);

        while(!q.isEmpty()){

            int levels = q.size();

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

            for(int i=0;i<levels;i++){

                if(q.peek().left!=null){

                    q.add(q.peek().left);

                }

                if(q.peek().right!=null){

                    q.add(q.peek().right);

                }

                subLevels.add(q.remove().val);

            }

            finalAns.add(subLevels);

        }

        return finalAns;

    }

}