# Wipro Coding Questions with Java Code Real Company Questions By – Mr. Durgesh StudyHub

## Wipro Coding Questions with Java Code

---

### 1. Reverse an Array

**Question:** Reverse an array in-place.
**Input:** `[1,2,3,4,5]`
**Output:** `[5,4,3,2,1]`

```
public class ReverseArray {
    public static void main(String[] args) {
        int[] arr = {1,2,3,4,5};
        reverseArray(arr);
        for(int num : arr) System.out.print(num + " ");
    }

    static void reverseArray(int[] arr) {
        int left = 0, right = arr.length - 1;
        while(left < right) {
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;
            left++; right--;
        }
    }
}
```

---

## 2. Two Sum Problem

**Question:** Find indices of two numbers that sum to target.
**Input:** `[2,7,11,15], target=9`
**Output:** `[0,1]`

```
import java.util.*;

public class TwoSum {
    public static void main(String[] args) {
        int[] nums = {2,7,11,15};
        int target = 9;
        System.out.println(Arrays.toString(twoSum(nums, target)));
    }

    static int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for(int i=0; i<nums.length; i++) {
            int complement = target - nums[i];
            if(map.containsKey(complement)) return new
int[]{map.get(complement), i};
            map.put(nums[i], i);
        }
        return new int[]{};
    }
}
```

---

## 3. Find Largest Element in Array

```
public class LargestElement {
    public static void main(String[] args) {
        int[] arr = {3, 5, 1, 9, 2};
        System.out.println(largest(arr));
    }

    static int largest(int[] arr) {
        int max = arr[0];
        for(int num : arr) if(num > max) max = num;
        return max;
    }
}
```

---

## 4. Reverse a String

```
public class ReverseString {
    public static void main(String[] args) {
        String str = "HelloWorld";
        System.out.println(reverse(str));
    }

    static String reverse(String str) {
        char[] chars = str.toCharArray();
        int left=0, right=chars.length-1;
        while(left<right) {
            char temp = chars[left];
```

```
            chars[left]=chars[right];
            chars[right]=temp;
            left++; right--;
        }
        return new String(chars);
    }
}
```

---

## 5. Fibonacci Sequence (n terms)

```
public class Fibonacci {
    public static void main(String[] args) {
        int n = 10;
        for(int i=0;i<n;i++) System.out.print(fib(i)+" ");
    }

    static int fib(int n) {
        if(n==0) return 0;
        if(n==1) return 1;
        return fib(n-1)+fib(n-2);
    }
}
```

---

## 6. Check Palindrome String

```
public class Palindrome {
    public static void main(String[] args) {
        String str = "racecar";
        System.out.println(isPalindrome(str));
    }

    static boolean isPalindrome(String str) {
        int left=0, right=str.length()-1;
        while(left<right) {
            if(str.charAt(left) != str.charAt(right)) return false;
            left++; right--;
        }
        return true;
    }
}
```

---

## 7. Linked List: Reverse Linked List

```
class Node {
    int val;
    Node next;
    Node(int val){ this.val = val; }
}

public class ReverseLinkedList {
    public static void main(String[] args) {
        Node head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(3);
        head = reverse(head);
        while(head!=null){
```

```
            System.out.print(head.val + " ");
            head = head.next;
        }
    }

    static Node reverse(Node head){
        Node prev = null, curr = head;
        while(curr!=null){
            Node next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
}
```

---

## 8. Binary Tree: Inorder Traversal

```
class TreeNode {
    int val;
    TreeNode left,right;
    TreeNode(int val){ this.val=val; }
}

public class InorderTraversal {
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        inorder(root);
    }

    static void inorder(TreeNode root){
        if(root==null) return;
        inorder(root.left);
        System.out.print(root.val+" ");
        inorder(root.right);
    }
}
```

---

## 9. Find Maximum Subarray Sum (Kadane's Algorithm)

```
public class MaxSubarraySum {
    public static void main(String[] args) {
        int[] arr = {-2,1,-3,4,-1,2,1,-5,4};
        System.out.println(maxSubArray(arr));
    }

    static int maxSubArray(int[] arr){
        int maxSoFar = arr[0], maxEndingHere = arr[0];
        for(int i=1;i<arr.length;i++){
            maxEndingHere = Math.max(arr[i], arr[i]+maxEndingHere);
            maxSoFar = Math.max(maxSoFar, maxEndingHere);
        }
        return maxSoFar;
    }
```

```
}
```

---

## 10. Count Vowels in a String

```
public class VowelCount {
    public static void main(String[] args) {
        String str = "HelloWorld";
        int count = 0;
        for(char c : str.toLowerCase().toCharArray())
            if("aeiou".indexOf(c) != -1) count++;
        System.out.println(count);
    }
}
```

## 11. Find Second Largest Element in Array

```
public class SecondLargest {
    public static void main(String[] args) {
        int[] arr = {10,5,20,8};
        int first=Integer.MIN_VALUE, second=Integer.MIN_VALUE;
        for(int num : arr){
            if(num>first){
                second=first;
                first=num;
            } else if(num>second && num!=first){
                second=num;
            }
        }
        System.out.println(second);
    }
}
```

---

## 12. Check if Array is Sorted

```
public class IsSorted {
    public static void main(String[] args) {
        int[] arr = {1,2,3,4,5};
        boolean sorted = true;
        for(int i=0;i<arr.length-1;i++){
            if(arr[i]>arr[i+1]){
                sorted=false;
                break;
            }
        }
        System.out.println(sorted);
    }
}
```

---

## 13. Merge Two Sorted Arrays

```
import java.util.*;

public class MergeSorted {
    public static void main(String[] args) {
        int[] a={1,3,5}, b={2,4,6};
        int[] merged = new int[a.length+b.length];
        int i=0,j=0,k=0;
        while(i<a.length && j<b.length){
            if(a[i]<b[j]) merged[k++]=a[i++];
            else merged[k++]=b[j++];
        }
        while(i<a.length) merged[k++]=a[i++];
        while(j<b.length) merged[k++]=b[j++];
        System.out.println(Arrays.toString(merged));
    }
}
```

---

## 14. Remove Duplicates from Sorted Array

```
import java.util.*;

public class RemoveDuplicates {
    public static void main(String[] args) {
        int[] arr={1,1,2,3,3,4};
        int n=arr.length, j=0;
        for(int i=1;i<n;i++){
            if(arr[i]!=arr[j]) arr[++j]=arr[i];
        }
        for(int i=0;i<=j;i++) System.out.print(arr[i]+" ");
    }
}
```

---

## 15. Find Missing Number in 1 to N

```
public class MissingNumber {
    public static void main(String[] args) {
        int[] arr={1,2,4,5};
        int n=arr.length+1;
        int total=n*(n+1)/2, sum=0;
        for(int num : arr) sum+=num;
        System.out.println(total-sum);
    }
}
```

---

## 16. Rotate Array by K Positions

```
import java.util.*;

public class RotateArray {
    public static void main(String[] args) {
        int[] arr={1,2,3,4,5};
        int k=2;
        rotate(arr,k);
        System.out.println(Arrays.toString(arr));
```

```
    }

    static void rotate(int[] arr, int k){
        k=k%arr.length;
        reverse(arr,0,arr.length-1);
        reverse(arr,0,k-1);
        reverse(arr,k,arr.length-1);
    }

    static void reverse(int[] arr,int l,int r){
        while(l<r){
            int temp=arr[l];
            arr[l]=arr[r];
            arr[r]=temp;
            l++; r--;
        }
    }
}
```

---

## 17. Find Intersection of Two Arrays

```
import java.util.*;

public class ArrayIntersection {
    public static void main(String[] args) {
        int[] a={1,2,2,3}, b={2,2,4};
        Map<Integer,Integer> map=new HashMap<>();
        for(int x:a) map.put(x,map.getOrDefault(x,0)+1);
        List<Integer> res=new ArrayList<>();
        for(int x:b){
            if(map.getOrDefault(x,0)>0){
                res.add(x);
                map.put(x,map.get(x)-1);
            }
        }
        System.out.println(res);
    }
}
```

---

## 18. Count Frequency of Each Element

```
import java.util.*;

public class FrequencyCount {
    public static void main(String[] args) {
        int[] arr={1,2,2,3,1};
        Map<Integer,Integer> map=new HashMap<>();
        for(int x:arr) map.put(x,map.getOrDefault(x,0)+1);
        System.out.println(map);
    }
}
```

---

## 19. Longest Increasing Subsequence (LIS)

```
public class LIS {
    public static void main(String[] args) {
```

```
        int[] arr={10,9,2,5,3,7,101,18};
        int n=arr.length;
        int[] dp=new int[n];
        Arrays.fill(dp,1);
        for(int i=1;i<n;i++)
            for(int j=0;j<i;j++)
                if(arr[i]>arr[j]) dp[i]=Math.max(dp[i],dp[j]+1);
        int max=0;
        for(int x:dp) max=Math.max(max,x);
        System.out.println(max);
    }
}
```

---

## 20. Count Number of Set Bits in Integer

```
public class CountBits {
    public static void main(String[] args) {
        int n=13; //1101
        int count=0;
        while(n!=0){
            n=n&(n-1);
            count++;
        }
        System.out.println(count);
    }
}
```

---

## 21. Power of a Number (x^n)

```
public class Power {
    public static void main(String[] args) {
        int x=2,n=10;
        System.out.println(pow(x,n));
    }

    static int pow(int x,int n){
        if(n==0) return 1;
        int half=pow(x,n/2);
        return n%2==0 ? half*half : half*half*x;
    }
}
```

---

## 22. Find All Pairs with Given Sum

```
import java.util.*;

public class PairSum {
    public static void main(String[] args) {
        int[] arr={1,2,3,4,5};
        int sum=5;
        Map<Integer,Integer> map=new HashMap<>();
        for(int x:arr){
            if(map.containsKey(sum-x)) System.out.println((sum-x)+","+x);
            map.put(x,map.getOrDefault(x,0)+1);
        }
    }
```

```
}
```

---

## 23. Factorial of a Number (Recursive)

```
public class Factorial {
    public static void main(String[] args) {
        int n=5;
        System.out.println(fact(n));
    }

    static int fact(int n){
        if(n<=1) return 1;
        return n*fact(n-1);
    }
}
```

---

## 24. Check Prime Number

```
public class PrimeCheck {
    public static void main(String[] args) {
        int n=29;
        boolean prime=true;
        for(int i=2;i*i<=n;i++){
            if(n%i==0){
                prime=false; break;
            }
        }
        System.out.println(prime);
    }
}
```

---

## 25. Swap Two Numbers Without Temp Variable

```
public class SwapNumbers {
    public static void main(String[] args) {
        int a=5,b=10;
        a=a+b; b=a-b; a=a-b;
        System.out.println(a+" "+b);
    }
}
```

## 26. Reverse a Linked List

```
class Node {
    int val;
    Node next;
    Node(int val){ this.val=val; }
}

public class ReverseLinkedList {
    public static void main(String[] args) {
        Node head=new Node(1);
```

```
        head.next=new Node(2);
        head.next.next=new Node(3);
        head=reverse(head);
        while(head!=null){
            System.out.print(head.val+" ");
            head=head.next;
        }
    }

    static Node reverse(Node head){
        Node prev=null, curr=head;
        while(curr!=null){
            Node next=curr.next;
            curr.next=prev;
            prev=curr;
            curr=next;
        }
        return prev;
    }
}
```

---

## 27. Detect Cycle in Linked List

```
class Node {
    int val; Node next;
    Node(int val){ this.val=val; }
}

public class DetectCycle {
    public static void main(String[] args){
        Node head=new Node(1);
        head.next=new Node(2);
        head.next.next=new Node(3);
        head.next.next.next=head.next; // cycle
        System.out.println(hasCycle(head));
    }

    static boolean hasCycle(Node head){
        Node slow=head, fast=head;
        while(fast!=null && fast.next!=null){
            slow=slow.next;
            fast=fast.next.next;
            if(slow==fast) return true;
        }
        return false;
    }
}
```

---

## 28. Binary Search

```
import java.util.*;

public class BinarySearch {
    public static void main(String[] args){
        int[] arr={1,2,3,4,5};
        int target=3;
        System.out.println(search(arr,target));
    }
```

```
    static int search(int[] arr,int target){
        int left=0,right=arr.length-1;
        while(left<=right){
            int mid=left+(right-left)/2;
            if(arr[mid]==target) return mid;
            else if(arr[mid]<target) left=mid+1;
            else right=mid-1;
        }
        return -1;
    }
}
```

---

## 29. Level Order Traversal of Binary Tree

```
import java.util.*;

class TreeNode {
    int val; TreeNode left,right;
    TreeNode(int val){ this.val=val; }
}

public class LevelOrder {
    public static void main(String[] args){
        TreeNode root=new TreeNode(1);
        root.left=new TreeNode(2);
        root.right=new TreeNode(3);
        root.left.left=new TreeNode(4);
        levelOrder(root);
    }

    static void levelOrder(TreeNode root){
        if(root==null) return;
        Queue<TreeNode> q=new LinkedList<>();
        q.add(root);
        while(!q.isEmpty()){
            TreeNode curr=q.poll();
            System.out.print(curr.val+" ");
            if(curr.left!=null) q.add(curr.left);
            if(curr.right!=null) q.add(curr.right);
        }
    }
}
```

---

## 30. Depth of Binary Tree

```
class TreeNode {
    int val; TreeNode left,right;
    TreeNode(int val){ this.val=val; }
}

public class TreeDepth {
    public static void main(String[] args){
        TreeNode root=new TreeNode(1);
        root.left=new TreeNode(2);
        root.right=new TreeNode(3);
        root.left.left=new TreeNode(4);
        System.out.println(depth(root));
```

```
    }

    static int depth(TreeNode root){
        if(root==null) return 0;
        return 1+Math.max(depth(root.left), depth(root.right));
    }
}
```

---

## 31. Count Number of Leaf Nodes in Binary Tree

```
class TreeNode {
    int val; TreeNode left,right;
    TreeNode(int val){ this.val=val; }
}

public class LeafCount {
    public static void main(String[] args){
        TreeNode root=new TreeNode(1);
        root.left=new TreeNode(2);
        root.right=new TreeNode(3);
        root.left.left=new TreeNode(4);
        System.out.println(countLeaves(root));
    }

    static int countLeaves(TreeNode root){
        if(root==null) return 0;
        if(root.left==null && root.right==null) return 1;
        return countLeaves(root.left)+countLeaves(root.right);
    }
}
```

---

## 32. Find Minimum in Rotated Sorted Array

```
public class MinInRotatedArray {
    public static void main(String[] args){
        int[] arr={4,5,6,7,0,1,2};
        System.out.println(findMin(arr));
    }

    static int findMin(int[] arr){
        int left=0,right=arr.length-1;
        while(left<right){
            int mid=left+(right-left)/2;
            if(arr[mid]>arr[right]) left=mid+1;
            else right=mid;
        }
        return arr[left];
    }
}
```

---

## 33. Merge Two Binary Trees

```
class TreeNode {
    int val; TreeNode left,right;
    TreeNode(int val){ this.val=val; }
}
```

```
public class MergeTrees {
    public static void main(String[] args){
        TreeNode t1=new TreeNode(1);
        t1.left=new TreeNode(3); t1.right=new TreeNode(2);
        TreeNode t2=new TreeNode(2);
        t2.left=new TreeNode(1); t2.right=new TreeNode(3);
        TreeNode merged=merge(t1,t2);
        inorder(merged);
    }

    static TreeNode merge(TreeNode t1, TreeNode t2){
        if(t1==null) return t2;
        if(t2==null) return t1;
        t1.val+=t2.val;
        t1.left=merge(t1.left,t2.left);
        t1.right=merge(t1.right,t2.right);
        return t1;
    }

    static void inorder(TreeNode root){
        if(root==null) return;
        inorder(root.left);
        System.out.print(root.val+" ");
        inorder(root.right);
    }
}
```

---

## 34. Implement Stack Using Array

```
class Stack {
    int[] arr; int top=-1;
    Stack(int n){ arr=new int[n]; }

    void push(int x){
        if(top==arr.length-1) throw new RuntimeException("Overflow");
        arr[++top]=x;
    }

    int pop(){
        if(top==-1) throw new RuntimeException("Underflow");
        return arr[top--];
    }

    int peek(){ return arr[top]; }

    boolean isEmpty(){ return top==-1; }
}

public class StackTest {
    public static void main(String[] args){
        Stack s=new Stack(5);
        s.push(1); s.push(2); s.push(3);
        System.out.println(s.pop());
    }
}
```

---

## 35. Implement Queue Using Two Stacks

```
import java.util.*;

class QueueUsingStacks {
    Stack<Integer> s1=new Stack<>();
    Stack<Integer> s2=new Stack<>();

    void enqueue(int x){ s1.push(x); }

    int dequeue(){
        if(s2.isEmpty()){
            while(!s1.isEmpty()) s2.push(s1.pop());
        }
        if(s2.isEmpty()) throw new RuntimeException("Empty");
        return s2.pop();
    }
}

public class QueueTest {
    public static void main(String[] args){
        QueueUsingStacks q=new QueueUsingStacks();
        q.enqueue(1); q.enqueue(2); q.enqueue(3);
        System.out.println(q.dequeue());
    }
}
```

---

## 36. Find Maximum Depth of N-ary Tree

```
import java.util.*;

class Node {
    int val; List<Node> children;
    Node(int val){ this.val=val; children=new ArrayList<>(); }
}

public class NAryTreeDepth {
    public static void main(String[] args){
        Node root=new Node(1);
        root.children.add(new Node(2));
        root.children.add(new Node(3));
        root.children.get(0).children.add(new Node(4));
        System.out.println(maxDepth(root));
    }

    static int maxDepth(Node root){
        if(root==null) return 0;
        int max=0;
        for(Node child: root.children) max=Math.max(max,maxDepth(child));
        return max+1;
    }
}
```

---

## 37. Count Number of Islands in 2D Grid

```
public class NumIslands {
    public static void main(String[] args){
        char[][] grid={
```

```
            {'1','1','0'},
            {'0','1','0'},
            {'1','0','1'}
        };
        System.out.println(numIslands(grid));
    }

    static int numIslands(char[][] grid){
        int count=0;
        for(int i=0;i<grid.length;i++){
            for(int j=0;j<grid[0].length;j++){
                if(grid[i][j]=='1'){
                    dfs(grid,i,j);
                    count++;
                }
            }
        }
        return count;
    }

    static void dfs(char[][] grid,int i,int j){
        if(i<0 || i>=grid.length || j<0 || j>=grid[0].length ||
grid[i][j]=='0') return;
        grid[i][j]='0';
        dfs(grid,i+1,j); dfs(grid,i-1,j);
        dfs(grid,i,j+1); dfs(grid,i,j-1);
    }
}
```

---

## 38. Top K Frequent Elements

```
import java.util.*;

public class TopKFrequent {
    public static void main(String[] args){
        int[] nums={1,1,1,2,2,3};
        int k=2;
        System.out.println(topKFrequent(nums,k));
    }

    static List<Integer> topKFrequent(int[] nums,int k){
        Map<Integer,Integer> map=new HashMap<>();
        for(int x:nums) map.put(x,map.getOrDefault(x,0)+1);
        PriorityQueue<int[]> pq=new PriorityQueue<>((a,b)->a[1]-b[1]);
        for(Map.Entry<Integer,Integer> e: map.entrySet()){
            pq.offer(new int[]{e.getKey(), e.getValue()});
            if(pq.size()>k) pq.poll();
        }
        List<Integer> res=new ArrayList<>();
        while(!pq.isEmpty()) res.add(pq.poll()[0]);
        Collections.reverse(res);
        return res;
    }
}
```

---

## 39. Find Kth Largest Element in Array

```
import java.util.*;
```

```
public class KthLargest {
    public static void main(String[] args){
        int[] arr={3,2,1,5,6,4};
        int k=2;
        System.out.println(findKthLargest(arr,k));
    }

    static int findKthLargest(int[] nums,int k){
        PriorityQueue<Integer> pq=new PriorityQueue<>();
        for(int num:nums){
            pq.offer(num);
            if(pq.size()>k) pq.poll();
        }
        return pq.peek();
    }
}
```

---

## 40. Sort Colors (Dutch National Flag Problem)

```
public class SortColors {
    public static void main(String[] args){
        int[] arr={2,0,2,1,1,0};
        sortColors(arr);
        for(int num: arr) System.out.print(num+" ");
    }

    static void sortColors(int[] arr){
        int low=0, mid=0, high=arr.length-1;
        while(mid<=high){
            if(arr[mid]==0){ int temp=arr[low]; arr[low]=arr[mid];
arr[mid]=temp; low++; mid++; }
            else if(arr[mid]==1) mid++;
            else{ int temp=arr[mid]; arr[mid]=arr[high]; arr[high]=temp;
high--; }
        }
    }
}
```

## 41. Longest Substring Without Repeating Characters

```
import java.util.*;

public class LongestUniqueSubstring {
    public static void main(String[] args){
        String s="abcabcbb";
        System.out.println(lengthOfLongestSubstring(s));
    }

    static int lengthOfLongestSubstring(String s){
```

```
        Map<Character,Integer> map=new HashMap<>();
        int left=0,maxLen=0;
        for(int right=0; right<s.length(); right++){
            char c=s.charAt(right);
            if(map.containsKey(c)) left=Math.max(left,map.get(c)+1);
            map.put(c,right);
            maxLen=Math.max(maxLen,right-left+1);
        }
        return maxLen;
    }
}
```

---

## 42. Median of Two Sorted Arrays

```
public class MedianTwoSorted {
    public static void main(String[] args){
        int[] nums1={1,3};
        int[] nums2={2};
        System.out.println(findMedianSortedArrays(nums1,nums2));
    }

    static double findMedianSortedArrays(int[] a,int[] b){
        if(a.length > b.length) return findMedianSortedArrays(b,a);
        int n=a.length,m=b.length, low=0, high=n;
        while(low<=high){
            int partitionA=(low+high)/2;
            int partitionB=(n+m+1)/2-partitionA;
            int maxLeftA = (partitionA==0)? Integer.MIN_VALUE:
a[partitionA-1];
            int minRightA= (partitionA==n)? Integer.MAX_VALUE:
a[partitionA];
            int maxLeftB= (partitionB==0)? Integer.MIN_VALUE: b[partitionB-
1];
            int minRightB= (partitionB==m)? Integer.MAX_VALUE:
b[partitionB];

            if(maxLeftA<=minRightB && maxLeftB<=minRightA)
                return ((n+m)%2==0)?
(Math.max(maxLeftA,maxLeftB)+Math.min(minRightA,minRightB))/2.0 :
Math.max(maxLeftA,maxLeftB);
            else if(maxLeftA>minRightB) high=partitionA-1;
            else low=partitionA+1;
        }
        return 0.0;
    }
}
```

---

## 43. Longest Palindromic Substring

```
public class LongestPalindrome {
    public static void main(String[] args){
        String s="babad";
        System.out.println(longestPalindrome(s));
    }

    static String longestPalindrome(String s){
        int start=0,end=0;
        for(int i=0;i<s.length();i++){
```

```
            int len1=expandAroundCenter(s,i,i);
            int len2=expandAroundCenter(s,i,i+1);
            int len=Math.max(len1,len2);
            if(len>end-start){
                start=i-(len-1)/2;
                end=i+len/2;
            }
        }
        return s.substring(start,end+1);
    }

    static int expandAroundCenter(String s,int left,int right){
        while(left>=0 && right<s.length() &&
s.charAt(left)==s.charAt(right)){
            left--; right++;
        }
        return right-left-1;
    }
}
```

---

## 44. Trapping Rain Water

```
public class TrappingRainWater {
    public static void main(String[] args){
        int[] height={0,1,0,2,1,0,1,3,2,1,2,1};
        System.out.println(trap(height));
    }

    static int trap(int[] height){
        int n=height.length;
        int leftMax[]=new int[n];
        int rightMax[]=new int[n];
        leftMax[0]=height[0];
        for(int i=1;i<n;i++) leftMax[i]=Math.max(leftMax[i-1],height[i]);
        rightMax[n-1]=height[n-1];
        for(int i=n-2;i>=0;i--)
rightMax[i]=Math.max(rightMax[i+1],height[i]);
        int res=0;
        for(int i=0;i<n;i++) res+=Math.min(leftMax[i],rightMax[i])-
height[i];
        return res;
    }
}
```

---

## 45. Word Ladder (Shortest Transformation)

```
import java.util.*;

public class WordLadder {
    public static void main(String[] args){
        String begin="hit", end="cog";
        List<String>
wordList=Arrays.asList("hot","dot","dog","lot","log","cog");
        System.out.println(ladderLength(begin,end,wordList));
    }

    static int ladderLength(String beginWord,String endWord,List<String>
wordList){
```

```
        Set<String> set=new HashSet<>(wordList);
        if(!set.contains(endWord)) return 0;
        Queue<String> q=new LinkedList<>();
        q.add(beginWord);
        int level=1;
        while(!q.isEmpty()){
            int size=q.size();
            for(int i=0;i<size;i++){
                String word=q.poll();
                char[] arr=word.toCharArray();
                for(int j=0;j<arr.length;j++){
                    char old=arr[j];
                    for(char c='a';c<='z';c++){
                        arr[j]=c;
                        String newWord=new String(arr);
                        if(newWord.equals(endWord)) return level+1;
                        if(set.contains(newWord)){
                            q.add(newWord);
                            set.remove(newWord);
                        }
                    }
                    arr[j]=old;
                }
            }
            level++;
        }
        return 0;
    }
}
```

---

## 46. Maximum Subarray Sum (Kadane's Algorithm)

```
public class MaxSubarraySum {
    public static void main(String[] args){
        int[] arr={-2,1,-3,4,-1,2,1,-5,4};
        int max=arr[0], curr=arr[0];
        for(int i=1;i<arr.length;i++){
            curr=Math.max(arr[i],curr+arr[i]);
            max=Math.max(max,curr);
        }
        System.out.println(max);
    }
}
```

---

## 47. Coin Change (Minimum Coins)

```
import java.util.*;

public class CoinChange {
    public static void main(String[] args){
        int[] coins={1,2,5};
        int amount=11;
        System.out.println(coinChange(coins,amount));
    }

    static int coinChange(int[] coins,int amount){
        int[] dp=new int[amount+1];
        Arrays.fill(dp,amount+1);
```

```
        dp[0]=0;
        for(int i=1;i<=amount;i++){
            for(int coin:coins){
                if(i-coin>=0) dp[i]=Math.min(dp[i],dp[i-coin]+1);
            }
        }
        return dp[amount]>amount?-1:dp[amount];
    }
}
```

---

## 48. N-Queens Problem (Backtracking)

```
import java.util.*;

public class NQueens {
    public static void main(String[] args){
        int n=4;
        List<List<String>> res=new ArrayList<>();
        char[][] board=new char[n][n];
        for(char[] row: board) Arrays.fill(row,'.');
        solve(board,0,res);
        System.out.println(res);
    }

    static void solve(char[][] board,int row,List<List<String>> res){
        if(row==board.length){
            List<String> temp=new ArrayList<>();
            for(char[] r:board) temp.add(new String(r));
            res.add(temp);
            return;
        }
        for(int col=0;col<board.length;col++){
            if(isSafe(board,row,col)){
                board[row][col]='Q';
                solve(board,row+1,res);
                board[row][col]='.';
            }
        }
    }

    static boolean isSafe(char[][] board,int row,int col){
        for(int i=0;i<row;i++) if(board[i][col]=='Q') return false;
        for(int i=row-1,j=col-1;i>=0 && j>=0;i--,j--) if(board[i][j]=='Q')
return false;
        for(int i=row-1,j=col+1;i>=0 && j<board.length;i--,j++)
if(board[i][j]=='Q') return false;
        return true;
    }
}
```

---

## 49. Longest Consecutive Sequence

```
import java.util.*;

public class LongestConsecutive {
    public static void main(String[] args){
        int[] nums={100,4,200,1,3,2};
        Set<Integer> set=new HashSet<>();
```

```
        for(int n:nums) set.add(n);
        int max=0;
        for(int n:nums){
            if(!set.contains(n-1)){
                int curr=n,len=1;
                while(set.contains(curr+1)){
                    curr++; len++;
                }
                max=Math.max(max,len);
            }
        }
        System.out.println(max);
    }
}
```

---

## 50. Sliding Window Maximum

```
import java.util.*;

public class SlidingWindowMax {
    public static void main(String[] args){
        int[] nums={1,3,-1,-3,5,3,6,7};
        int k=3;
        System.out.println(Arrays.toString(maxSlidingWindow(nums,k)));
    }

    static int[] maxSlidingWindow(int[] nums,int k){
        int n=nums.length;
        int[] res=new int[n-k+1];
        Deque<Integer> dq=new LinkedList<>();
        for(int i=0;i<n;i++){
            while(!dq.isEmpty() && dq.peek()<i-k+1) dq.poll();
            while(!dq.isEmpty() && nums[dq.peekLast()]<nums[i])
dq.pollLast();
            dq.offer(i);
            if(i>=k-1) res[i-k+1]=nums[dq.peek()];
        }
        return res;
    }
}
```

## 51. Minimum Window Substring

```
import java.util.*;

public class MinWindowSubstring {
    public static void main(String[] args){
        String s="ADOBECODEBANC", t="ABC";
        System.out.println(minWindow(s,t));
    }

    static String minWindow(String s,String t){
        if(s.length()==0 || t.length()==0) return "";
```

```
        Map<Character,Integer> mapT=new HashMap<>();
        for(char c:t) mapT.put(c,mapT.getOrDefault(c,0)+1);
        Map<Character,Integer> window=new HashMap<>();
        int have=0, need=mapT.size();
        int left=0,resLen=Integer.MAX_VALUE,resStart=0;
        for(int right=0;right<s.length();right++){
            char c=s.charAt(right);
            window.put(c,window.getOrDefault(c,0)+1);
            if(mapT.containsKey(c) &&
window.get(c).intValue()==mapT.get(c).intValue()) have++;
            while(have==need){
                if(right-left+1<resLen){
                    resLen=right-left+1;
                    resStart=left;
                }
                char leftChar=s.charAt(left);
                window.put(leftChar,window.get(leftChar)-1);
                if(mapT.containsKey(leftChar) &&
window.get(leftChar)<mapT.get(leftChar)) have--;
                left++;
            }
        }
        return
resLen==Integer.MAX_VALUE?"":s.substring(resStart,resStart+resLen);
    }
}
```

---

## 52. Decode Ways (DP)

```
public class DecodeWays {
    public static void main(String[] args){
        String s="226";
        System.out.println(numDecodings(s));
    }

    static int numDecodings(String s){
        int n=s.length();
        int[] dp=new int[n+1];
        dp[0]=1;
        for(int i=1;i<=n;i++){
            if(s.charAt(i-1)!='0') dp[i]+=dp[i-1];
            if(i>1 && s.charAt(i-2)!='0' && Integer.parseInt(s.substring(i-
2,i))<=26) dp[i]+=dp[i-2];
        }
        return dp[n];
    }
}
```

---

## 53. Word Break Problem

```
import java.util.*;

public class WordBreak {
    public static void main(String[] args){
        String s="leetcode";
        List<String> wordDict=Arrays.asList("leet","code");
        System.out.println(wordBreak(s,new HashSet<>(wordDict)));
    }
```

```
    static boolean wordBreak(String s, Set<String> dict){
        boolean[] dp=new boolean[s.length()+1];
        dp[0]=true;
        for(int i=1;i<=s.length();i++){
            for(int j=0;j<i;j++){
                if(dp[j] && dict.contains(s.substring(j,i))){
                    dp[i]=true;
                    break;
                }
            }
        }
        return dp[s.length()];
    }
}
```

---

## 54. Serialize and Deserialize Binary Tree

```
import java.util.*;

class TreeNode {
    int val; TreeNode left,right;
    TreeNode(int val){ this.val=val; }
}

public class SerializeDeserialize {
    public static void main(String[] args){
        TreeNode root=new TreeNode(1);
        root.left=new TreeNode(2);
        root.right=new TreeNode(3);
        root.right.left=new TreeNode(4);
        root.right.right=new TreeNode(5);

        String data=serialize(root);
        System.out.println(data);
        TreeNode newRoot=deserialize(data);
        inorder(newRoot);
    }

    static String serialize(TreeNode root){
        if(root==null) return "null,";
        return root.val+","+serialize(root.left)+serialize(root.right);
    }

    static int idx=0;
    static TreeNode deserialize(String data){
        String[] arr=data.split(",");
        return build(arr);
    }

    static TreeNode build(String[] arr){
        if(idx>=arr.length || arr[idx].equals("null")){
            idx++;
            return null;
        }
        TreeNode root=new TreeNode(Integer.parseInt(arr[idx++]));
        root.left=build(arr);
        root.right=build(arr);
        return root;
```

```
    }

    static void inorder(TreeNode root){
        if(root==null) return;
        inorder(root.left);
        System.out.print(root.val+" ");
        inorder(root.right);
    }
}
```

---

## 55. Kth Smallest Element in BST

```
import java.util.*;

class TreeNode {
    int val; TreeNode left,right;
    TreeNode(int val){ this.val=val; }
}

public class KthSmallestBST {
    public static void main(String[] args){
        TreeNode root=new TreeNode(3);
        root.left=new TreeNode(1);
        root.right=new TreeNode(4);
        root.left.right=new TreeNode(2);
        int k=1;
        System.out.println(kthSmallest(root,k));
    }

    static int count=0,res=0;
    static int kthSmallest(TreeNode root,int k){
        inorder(root,k);
        return res;
    }

    static void inorder(TreeNode root,int k){
        if(root==null) return;
        inorder(root.left,k);
        count++;
        if(count==k) res=root.val;
        inorder(root.right,k);
    }
}
```

---

## 56. Find Duplicate Number in Array

```
public class FindDuplicate {
    public static void main(String[] args){
        int[] nums={1,3,4,2,2};
        System.out.println(findDuplicate(nums));
    }

    static int findDuplicate(int[] nums){
        int slow=nums[0], fast=nums[0];
        do{
            slow=nums[slow];
            fast=nums[nums[fast]];
        } while(slow!=fast);
```

```
        fast=nums[0];
        while(slow!=fast){
            slow=nums[slow];
            fast=nums[fast];
        }
        return slow;
    }
}
```

---

## 57. Trapping Rain Water II (2D)

```
import java.util.*;

class Cell implements Comparable<Cell>{
    int x,y,h;
    Cell(int x,int y,int h){ this.x=x; this.y=y; this.h=h; }
    public int compareTo(Cell o){ return this.h-o.h; }
}

public class TrappingWater2D {
    public static void main(String[] args){
        int[][] height={{1,4,3},{3,2,5},{4,3,1}};
        System.out.println(trapRainWater(height));
    }

    static int trapRainWater(int[][] height){
        int m=height.length, n=height[0].length;
        boolean[][] visited=new boolean[m][n];
        PriorityQueue<Cell> pq=new PriorityQueue<>();
        for(int i=0;i<m;i++){ pq.offer(new Cell(i,0,height[i][0]));
pq.offer(new Cell(i,n-1,height[i][n-1])); visited[i][0]=visited[i][n-
1]=true;}
        for(int j=1;j<n-1;j++){ pq.offer(new Cell(0,j,height[0][j]));
pq.offer(new Cell(m-1,j,height[m-1][j])); visited[0][j]=visited[m-
1][j]=true;}
        int[][] dirs={{0,1},{0,-1},{1,0},{-1,0}};
        int res=0;
        while(!pq.isEmpty()){
            Cell c=pq.poll();
            for(int[] d: dirs){
                int nx=c.x+d[0], ny=c.y+d[1];
                if(nx>=0 && nx<m && ny>=0 && ny<n && !visited[nx][ny]){
                    res+=Math.max(0,c.h-height[nx][ny]);
                    pq.offer(new Cell(nx,ny,Math.max(height[nx][ny],c.h)));
                    visited[nx][ny]=true;
                }
            }
        }
        return res;
    }
}
```

---

## 58. Maximum Product Subarray

```
public class MaxProductSubarray {
    public static void main(String[] args){
        int[] nums={2,3,-2,4};
        int maxProd=nums[0], currMax=nums[0], currMin=nums[0];
```

```
        for(int i=1;i<nums.length;i++){
            int temp=currMax;
            currMax=Math.max(nums[i], Math.max(currMax*nums[i],
currMin*nums[i]));
            currMin=Math.min(nums[i], Math.min(temp*nums[i],
currMin*nums[i]));
            maxProd=Math.max(maxProd,currMax);
        }
        System.out.println(maxProd);
    }
}
```

---

## 59. Serialize and Deserialize N-ary Tree

```
import java.util.*;

class Node{
    int val; List<Node> children;
    Node(int val){ this.val=val; children=new ArrayList<>(); }
}

public class SerializeNary {
    public static void main(String[] args){
        Node root=new Node(1);
        root.children.add(new Node(2));
        root.children.add(new Node(3));
        root.children.get(1).children.add(new Node(4));
        String data=serialize(root);
        System.out.println(data);
        Node newRoot=deserialize(data);
        System.out.println(newRoot.val);
    }

    static String serialize(Node root){
        if(root==null) return "";
        StringBuilder sb=new StringBuilder();
        sb.append(root.val).append("[");
        for(Node child:root.children) sb.append(serialize(child));
        sb.append("]");
        return sb.toString();
    }

    static int idx=0;
    static Node deserialize(String s){
        if(s.length()==0) return null;
        idx=0;
        return build(s);
    }

    static Node build(String s){
        int val=0;
        while(idx<s.length() && Character.isDigit(s.charAt(idx))){
            val=val*10+s.charAt(idx)-'0';
            idx++;
        }
        Node node=new Node(val);
        if(idx<s.length() && s.charAt(idx)=='['){
            idx++;
            while(idx<s.length() && s.charAt(idx)!=']'){
```

```
                node.children.add(build(s));
            }
            idx++;
        }
        return node;
    }
}
```

## 60. Sliding Window Maximum Sum of Size K

```
public class MaxSumWindow {
    public static void main(String[] args){
        int[] arr={1,2,3,4,5,6};
        int k=3;
        int n=arr.length, sum=0,max=0;
        for(int i=0;i<k;i++) sum+=arr[i];
        max=sum;
        for(int i=k;i<n;i++){
            sum+=arr[i]-arr[i-k];
            max=Math.max(max,sum);
        }
        System.out.println(max);
    }
}
```

---

## 61. Clone Graph (DFS)

```
import java.util.*;

class Node {
    int val; List<Node> neighbors;
    Node(int val){ this.val=val; neighbors=new ArrayList<>(); }
}

public class CloneGraph {
    public static void main(String[] args){
        Node node=new Node(1);
        Node clone=cloneGraph(node,new HashMap<>());
        System.out.println(clone.val);
    }

    static Node cloneGraph(Node node, Map<Node,Node> map){
        if(node==null) return null;
        if(map.containsKey(node)) return map.get(node);
        Node clone=new Node(node.val);
        map.put(node,clone);
        for(Node n: node.neighbors) clone.neighbors.add(cloneGraph(n,map));
        return clone;
```

```
    }
}
```

---

## 62. Course Schedule (Detect Cycle in Graph)

```
import java.util.*;

public class CourseSchedule {
    public static void main(String[] args){
        int numCourses=2;
        int[][] prereq={{1,0}};
        System.out.println(canFinish(numCourses,prereq));
    }

    static boolean canFinish(int n,int[][] prereq){
        List<List<Integer>> graph=new ArrayList<>();
        for(int i=0;i<n;i++) graph.add(new ArrayList<>());
        int[] indegree=new int[n];
        for(int[] p: prereq){
            graph.get(p[1]).add(p[0]);
            indegree[p[0]]++;
        }
        Queue<Integer> q=new LinkedList<>();
        for(int i=0;i<n;i++) if(indegree[i]==0) q.add(i);
        int count=0;
        while(!q.isEmpty()){
            int cur=q.poll();
            count++;
            for(int nei: graph.get(cur)){
                indegree[nei]--;
                if(indegree[nei]==0) q.add(nei);
            }
        }
        return count==n;
    }
}
```

---

## 63. Maximum Length of Repeated Subarray

```
public class MaxRepeatedSubarray {
    public static void main(String[] args){
        int[] A={1,2,3,2,1};
        int[] B={3,2,1,4,7};
        int n=A.length,m=B.length,max=0;
        int[][] dp=new int[n+1][m+1];
        for(int i=1;i<=n;i++){
            for(int j=1;j<=m;j++){
                if(A[i-1]==B[j-1]){
                    dp[i][j]=dp[i-1][j-1]+1;
                    max=Math.max(max,dp[i][j]);
                }
            }
        }
        System.out.println(max);
    }
}
```

---

## 64. Serialize and Deserialize BST

```
import java.util.*;

class TreeNode {
    int val; TreeNode left,right;
    TreeNode(int val){ this.val=val; }
}

public class SerializeBST {
    public static void main(String[] args){
        TreeNode root=new TreeNode(2);
        root.left=new TreeNode(1);
        root.right=new TreeNode(3);
        String s=serialize(root);
        System.out.println(s);
        TreeNode newRoot=deserialize(s);
        System.out.println(newRoot.val);
    }

    static String serialize(TreeNode root){
        if(root==null) return "";
        return root.val+","+serialize(root.left)+serialize(root.right);
    }

    static int idx=0;
    static TreeNode deserialize(String s){
        if(s.length()==0) return null;
        idx=0;
        String[] arr=s.split(",");
        return build(arr,Integer.MIN_VALUE,Integer.MAX_VALUE);
    }

    static TreeNode build(String[] arr,int min,int max){
        if(idx>=arr.length || arr[idx].equals("")) return null;
        int val=Integer.parseInt(arr[idx]);
        if(val<min || val>max) return null;
        idx++;
        TreeNode node=new TreeNode(val);
        node.left=build(arr,min,val);
        node.right=build(arr,val,max);
        return node;
    }
}
```

---

## 65. Sliding Window Median

```
import java.util.*;

public class SlidingWindowMedian {
    public static void main(String[] args){
        int[] nums={1,3,-1,-3,5,3,6,7};
        int k=3;
        double[] res=medianSlidingWindow(nums,k);
        System.out.println(Arrays.toString(res));
    }

    static double[] medianSlidingWindow(int[] nums,int k){
        double[] res=new double[nums.length-k+1];
```

```
        PriorityQueue<Integer> small=new
PriorityQueue<>(Collections.reverseOrder());
        PriorityQueue<Integer> large=new PriorityQueue<>();
        for(int i=0;i<nums.length;i++){
            if(small.isEmpty() || nums[i]<=small.peek()) small.add(nums[i]);
            else large.add(nums[i]);
            balance(small,large);
            if(i>=k-1){
                if(k%2==0) res[i-k+1]=(small.peek()/2.0+large.peek()/2.0);
                else res[i-
k+1]=small.size()>large.size()?small.peek():large.peek();
                if(small.contains(nums[i-k+1])) small.remove(nums[i-k+1]);
                else large.remove(nums[i-k+1]);
                balance(small,large);
            }
        }
        return res;
    }

    static void balance(PriorityQueue<Integer> small,PriorityQueue<Integer>
large){
        while(small.size()>large.size()+1) large.add(small.poll());
        while(large.size()>small.size()) small.add(large.poll());
    }
}
```

---

## 66. Word Search (Backtracking)

```
public class WordSearch {
    public static void main(String[] args){
        char[][] board={
            {'A','B','C','E'},
            {'S','F','C','S'},
            {'A','D','E','E'}
        };
        String word="ABCCED";
        System.out.println(exist(board,word));
    }

    static boolean exist(char[][] board,String word){
        for(int i=0;i<board.length;i++){
            for(int j=0;j<board[0].length;j++){
                if(dfs(board,word,i,j,0)) return true;
            }
        }
        return false;
    }

    static boolean dfs(char[][] b,String w,int i,int j,int idx){
        if(idx==w.length()) return true;
        if(i<0||j<0||i>=b.length||j>=b[0].length||b[i][j]!=w.charAt(idx))
return false;
        char tmp=b[i][j]; b[i][j]='#';
        boolean found=dfs(b,w,i+1,j,idx+1)||dfs(b,w,i-
1,j,idx+1)||dfs(b,w,i,j+1,idx+1)||dfs(b,w,i,j-1,idx+1);
        b[i][j]=tmp;
        return found;
    }
}
```

## 67. Minimum Path Sum (Grid DP)

```
public class MinPathSum {
    public static void main(String[] args){
        int[][] grid={{1,3,1},{1,5,1},{4,2,1}};
        int m=grid.length,n=grid[0].length;
        int[][] dp=new int[m][n];
        dp[0][0]=grid[0][0];
        for(int i=1;i<m;i++) dp[i][0]=dp[i-1][0]+grid[i][0];
        for(int j=1;j<n;j++) dp[0][j]=dp[0][j-1]+grid[0][j];
        for(int i=1;i<m;i++)
            for(int j=1;j<n;j++)
                dp[i][j]=Math.min(dp[i-1][j],dp[i][j-1])+grid[i][j];
        System.out.println(dp[m-1][n-1]);
    }
}
```

## 68. Largest Rectangle in Histogram

```
import java.util.*;

public class LargestRectangle {
    public static void main(String[] args){
        int[] h={2,1,5,6,2,3};
        System.out.println(largestRectangleArea(h));
    }

    static int largestRectangleArea(int[] h){
        Stack<Integer> st=new Stack<>();
        int max=0;
        for(int i=0;i<=h.length;i++){
            int height=i==h.length?0:h[i];
            while(!st.isEmpty() && height<h[st.peek()]){
                int cur=st.pop();
                int width=st.isEmpty()? i:i-st.peek()-1;
                max=Math.max(max,h[cur]*width);
            }
            st.push(i);
        }
        return max;
    }
}
```

## 69. Maximal Rectangle in Binary Matrix

```
public class MaximalRectangle {
    public static void main(String[] args){
        char[][] matrix={{'1','0','1','0','0'},
                         {'1','0','1','1','1'},
                         {'1','1','1','1','1'},
                         {'1','0','0','1','0'}};
        int m=matrix.length,n=matrix[0].length;
        int[] height=new int[n];
        int max=0;
        for(int i=0;i<m;i++){
```

```
            for(int j=0;j<n;j++){
                height[j]=matrix[i][j]=='0'?0:height[j]+1;
            }
            max=Math.max(max,largestRectangle(height));
        }
        System.out.println(max);
    }

    static int largestRectangle(int[] h){
        Stack<Integer> st=new Stack<>();
        int max=0;
        for(int i=0;i<=h.length;i++){
            int height=i==h.length?0:h[i];
            while(!st.isEmpty() && height<h[st.peek()]){
                int cur=st.pop();
                int width=st.isEmpty()?i:i-st.peek()-1;
                max=Math.max(max,h[cur]*width);
            }
            st.push(i);
        }
        return max;
    }
}
```

## 70. Trapping Rain Water III (Advanced 2D with BFS)

```
import java.util.*;

class Cell implements Comparable<Cell>{
    int x,y,h;
    Cell(int x,int y,int h){ this.x=x; this.y=y; this.h=h; }
    public int compareTo(Cell o){ return this.h-o.h; }
}

public class TrappingWater2DAdvanced {
    public static void main(String[] args){
        int[][] height={{1,4,3},{3,2,5},{4,3,1}};
        System.out.println(trapRainWater(height));
    }

    static int trapRainWater(int[][] height){
        int m=height.length,n=height[0].length;
        boolean[][] visited=new boolean[m][n];
        PriorityQueue<Cell> pq=new PriorityQueue<>();
```

```
        for(int i=0;i<m;i++){
            pq.offer(new Cell(i,0,height[i][0])); pq.offer(new Cell(i,n-
1,height[i][n-1]));
            visited[i][0]=visited[i][n-1]=true;
        }
        for(int j=1;j<n-1;j++){
            pq.offer(new Cell(0,j,height[0][j])); pq.offer(new Cell(m-
1,j,height[m-1][j]));
            visited[0][j]=visited[m-1][j]=true;
        }
        int[][] dirs={{0,1},{0,-1},{1,0},{-1,0}};
        int res=0;
        while(!pq.isEmpty()){
            Cell c=pq.poll();
            for(int[] d:dirs){
                int nx=c.x+d[0],ny=c.y+d[1];
                if(nx>=0 && nx<m && ny>=0 && ny<n && !visited[nx][ny]){
                    res+=Math.max(0,c.h-height[nx][ny]);
                    pq.offer(new Cell(nx,ny,Math.max(height[nx][ny],c.h)));
                    visited[nx][ny]=true;
                }
            }
        }
        return res;
    }
}
```

---

## 71. Dijkstra's Algorithm (Shortest Path in Graph)

```
import java.util.*;

public class Dijkstra {
    public static void main(String[] args){
        int V=5;
        int[][] graph={{0,6,0,1,0},
                       {6,0,5,2,2},
                       {0,5,0,0,5},
                       {1,2,0,0,1},
                       {0,2,5,1,0}};
        int src=0;
        System.out.println(Arrays.toString(dijkstra(graph,src)));
    }

    static int[] dijkstra(int[][] graph,int src){
        int V=graph.length;
        int[] dist=new int[V];
        boolean[] visited=new boolean[V];
        Arrays.fill(dist,Integer.MAX_VALUE);
        dist[src]=0;

        for(int i=0;i<V;i++){
            int u=-1;
            for(int j=0;j<V;j++){
                if(!visited[j] && (u==-1 || dist[j]<dist[u])) u=j;
            }
            visited[u]=true;
            for(int v=0;v<V;v++){
                if(graph[u][v]>0)
dist[v]=Math.min(dist[v],dist[u]+graph[u][v]);
```

```
            }
        }
        return dist;
    }
}
```

---

## 72. Minimum Spanning Tree (Prim's Algorithm)

```
import java.util.*;

public class MSTPrim {
    public static void main(String[] args){
        int[][] graph={{0,2,0,6,0},
                       {2,0,3,8,5},
                       {0,3,0,0,7},
                       {6,8,0,0,9},
                       {0,5,7,9,0}};
        System.out.println(primMST(graph));
    }

    static int primMST(int[][] graph){
        int V=graph.length;
        int[] key=new int[V];
        boolean[] mstSet=new boolean[V];
        Arrays.fill(key,Integer.MAX_VALUE);
        key[0]=0;
        int res=0;

        for(int count=0;count<V;count++){
            int u=-1;
            for(int v=0;v<V;v++){
                if(!mstSet[v] && (u==-1 || key[v]<key[u])) u=v;
            }
            mstSet[u]=true;
            res+=key[u];
            for(int v=0;v<V;v++){
                if(graph[u][v]!=0 && !mstSet[v] && graph[u][v]<key[v])
key[v]=graph[u][v];
            }
        }
        return res;
    }
}
```

---

## 73. Topological Sort (DFS)

```
import java.util.*;

public class TopoSort {
    public static void main(String[] args){
        int V=6;
        List<List<Integer>> graph=new ArrayList<>();
        for(int i=0;i<V;i++) graph.add(new ArrayList<>());
        graph.get(5).add(2); graph.get(5).add(0);
        graph.get(4).add(0); graph.get(4).add(1);
        graph.get(2).add(3); graph.get(3).add(1);

        System.out.println(topologicalSort(graph));
```

```
    }

    static List<Integer> topologicalSort(List<List<Integer>> graph){
        int V=graph.size();
        boolean[] visited=new boolean[V];
        Stack<Integer> st=new Stack<>();
        for(int i=0;i<V;i++) if(!visited[i]) dfs(graph,i,visited,st);
        List<Integer> res=new ArrayList<>();
        while(!st.isEmpty()) res.add(st.pop());
        return res;
    }

    static void dfs(List<List<Integer>> g,int u,boolean[]
visited,Stack<Integer> st){
        visited[u]=true;
        for(int v:g.get(u)) if(!visited[v]) dfs(g,v,visited,st);
        st.push(u);
    }
}
```

---

## 74. Trie (Insert and Search)

```
class TrieNode{
    TrieNode[] children=new TrieNode[26];
    boolean end=false;
}

public class Trie {
    TrieNode root=new TrieNode();

    void insert(String word){
        TrieNode node=root;
        for(char c:word.toCharArray()){
            int i=c-'a';
            if(node.children[i]==null) node.children[i]=new TrieNode();
            node=node.children[i];
        }
        node.end=true;
    }

    boolean search(String word){
        TrieNode node=root;
        for(char c:word.toCharArray()){
            int i=c-'a';
            if(node.children[i]==null) return false;
            node=node.children[i];
        }
        return node.end;
    }

    public static void main(String[] args){
        Trie trie=new Trie();
        trie.insert("apple");
        System.out.println(trie.search("apple"));
        System.out.println(trie.search("app"));
    }
}
```

---

## 75. Maximum Sum Rectangle in 2D Matrix

```
public class MaxSumRectangle {
    public static void main(String[] args){
        int[][] matrix={{1,2,-1,-4,-20},
                        {-8,-3,4,2,1},
                        {3,8,10,1,3},
                        {-4,-1,1,7,-6}};
        System.out.println(maxSumRectangle(matrix));
    }

    static int maxSumRectangle(int[][] m){
        int row=m.length,col=m[0].length,max=Integer.MIN_VALUE;
        for(int left=0;left<col;left++){
            int[] temp=new int[row];
            for(int right=left;right<col;right++){
                for(int i=0;i<row;i++) temp[i]+=m[i][right];
                int sum=kadane(temp);
                max=Math.max(max,sum);
            }
        }
        return max;
    }

    static int kadane(int[] arr){
        int max=arr[0], curr=arr[0];
        for(int i=1;i<arr.length;i++){
            curr=Math.max(arr[i],curr+arr[i]);
            max=Math.max(max,curr);
        }
        return max;
    }
}
```

---

## 76. Longest Increasing Path in Matrix

```
public class LongestIncreasingPath {
    public static void main(String[] args){
        int[][] matrix={{9,9,4},{6,6,8},{2,1,1}};
        System.out.println(longestIncreasingPath(matrix));
    }

    static int[][] dirs={{0,1},{0,-1},{1,0},{-1,0}};
    static int m,n;
    static int longestIncreasingPath(int[][] matrix){
        m=matrix.length;
        n=matrix[0].length;
        int[][] dp=new int[m][n];
        int max=0;
        for(int i=0;i<m;i++)
            for(int j=0;j<n;j++)
                max=Math.max(max,dfs(matrix,i,j,dp));
        return max;
    }

    static int dfs(int[][] mat,int i,int j,int[][] dp){
        if(dp[i][j]!=0) return dp[i][j];
        int max=1;
        for(int[] d: dirs){
```

```
            int x=i+d[0], y=j+d[1];
            if(x>=0 && x<m && y>=0 && y<n && mat[x][y]>mat[i][j])
                max=Math.max(max,1+dfs(mat,x,y,dp));
        }
        dp[i][j]=max;
        return max;
    }
}
```

---

## 77. Edit Distance (DP)

```
public class EditDistance {
    public static void main(String[] args){
        String word1="horse", word2="ros";
        System.out.println(minDistance(word1,word2));
    }

    static int minDistance(String w1,String w2){
        int m=w1.length(), n=w2.length();
        int[][] dp=new int[m+1][n+1];
        for(int i=0;i<=m;i++) dp[i][0]=i;
        for(int j=0;j<=n;j++) dp[0][j]=j;
        for(int i=1;i<=m;i++){
            for(int j=1;j<=n;j++){
                if(w1.charAt(i-1)==w2.charAt(j-1)) dp[i][j]=dp[i-1][j-1];
                else dp[i][j]=1+Math.min(dp[i-1][j-1],Math.min(dp[i-
1][j],dp[i][j-1]));
            }
        }
        return dp[m][n];
    }
}
```

---

## 78. Word Ladder II (All Paths)

```
// This is complex, using BFS + DFS to track all paths.
// Due to space, outline: use BFS to build levels, then DFS backtrack to
form all sequences.
```

---

## 79. Burst Balloons (DP)

```
public class BurstBalloons {
    public static void main(String[] args){
        int[] nums={3,1,5,8};
        System.out.println(maxCoins(nums));
    }

    static int maxCoins(int[] nums){
        int n=nums.length;
        int[] arr=new int[n+2];
        arr[0]=arr[n+1]=1;
        for(int i=0;i<n;i++) arr[i+1]=nums[i];
        int[][] dp=new int[n+2][n+2];
        for(int len=1;len<=n;len++){
            for(int left=1;left<=n-len+1;left++){
                int right=left+len-1;
```

```
                for(int k=left;k<=right;k++){
                    dp[left][right]=Math.max(dp[left][right], arr[left-
1]*arr[k]*arr[right+1]+dp[left][k-1]+dp[k+1][right]);
                }
            }
        }
        return dp[1][n];
    }
}
```