## Q1. The King’s Feast - Maximum Plate

### *This Java class, `MaxPlate`, finds the maximum element in an array of integers.*

```java
// Find maximum element in an array
public class MaxPlate {
    public static int findMax(int[] arr) {
        int max = Integer.MIN_VALUE;
        for (int v : arr) if (v > max) max = v;
        return max;
    }
    public static void main(String[] args) {
        int[] arr = {2,7,1,9,5};
        System.out.println(findMax(arr)); // expected 9
    }
}
```

## Q2. The Lost Soldier - Missing Number

### *This Java class, `MissingSoldier`, finds the missing number in 0..n using XOR for O(n) time and O(1) space.*

```java
// Find missing number in 0..n
public class MissingSoldier {
    public static int missing(int[] arr, int n) {
        int x = 0;
        for (int i = 0; i <= n; i++) x ^= i;
        for (int v : arr) x ^= v;
        return x;
    }
    public static void main(String[] args) {
        int n = 5;
        int[] arr = {0,1,2,4,5};
        System.out.println(missing(arr, n)); // expected 3
    }
}
```

## Q3. Potion Mixing (Two Sum)

### *This Java class, `TwoSumPotion`, finds indices of two numbers adding to target using a HashMap.*

```java
// Two-sum (returns indices)
import java.util.*;
public class TwoSumPotion {
    public static int[] twoSum(int[] arr, int target) {
        Map<Integer,Integer> map = new HashMap<>();
        for (int i = 0; i < arr.length; i++) {
            int need = target - arr[i];
            if (map.containsKey(need)) return new int[]{map.get(need), i};
            map.put(arr[i], i);
        }
        return new int[]{-1, -1};
    }
    public static void main(String[] args) {
        int[] arr = {3,2,4,7};
        int target = 6;
        int[] res = twoSum(arr, target);
        System.out.println("(" + res[0] + "," + res[1] + ")"); // expected (1,2) (0-based)
    }
}
```

## Q4. The Secret Message - Reverse Array

### *This Java class, `ReverseArray`, reverses an array in-place and returns it.*

```java
// Reverse an array
import java.util.Arrays;
```

```
public class ReverseArray {
    public static void reverse(int[] arr) {
        int l = 0, r = arr.length - 1;
        while (l < r) {
            int t = arr[l]; arr[l] = arr[r]; arr[r] = t;
            l++; r--;
        }
    }
    public static void main(String[] args) {
        int[] arr = {1,2,3,4};
        reverse(arr);
        System.out.println(Arrays.toString(arr)); // expected [4,3,2,1]
    }
}
```

## Q5. The King’s Parade - Check Sorted

### *This Java class, `IsNonDecreasing`, checks if array is sorted in non-decreasing order.*

```
// Check non-decreasing order
public class IsNonDecreasing {
    public static boolean isSorted(int[] arr) {
        for (int i = 1; i < arr.length; i++) if (arr[i] < arr[i-1]) return false;
        return true;
    }
    public static void main(String[] args) {
        System.out.println(isSorted(new int[]{1,3,5,7})); // true
        System.out.println(isSorted(new int[]{3,2,1})); // false
    }
}
```

## Q6. The Treasure Island - Row with Maximum Gold

### *This Java class, `MaxGoldRow`, finds the row index with maximum sum in a 2D matrix.*

```
// Row with maximum sum
public class MaxGoldRow {
    public static int maxSumRow(int[][] mat) {
        int best = -1, bestSum = Integer.MIN_VALUE;
        for (int i = 0; i < mat.length; i++) {
            int s = 0;
            for (int v : mat[i]) s += v;
            if (s > bestSum) { bestSum = s; best = i; }
        }
        return best;
    }
    public static void main(String[] args) {
        int[][] mat = {{1,2,3},{4,5,6},{7,8,9}};
        int idx = maxSumRow(mat);
        System.out.println("Row " + idx + " (sum=" + (idx>=0 ? java.util.Arrays.stream(mat[idx]).sum() : 0) + ")");
    }
}
```

## Q7. The Spiral Library - Spiral Order

### *This Java class, `SpiralOrder`, prints elements of a matrix in spiral order.*

```
// Spiral order traversal
import java.util.*;
public class SpiralOrder {
    public static List<Integer> spiral(int[][] a) {
        List<Integer> res = new ArrayList<>();
        if (a.length==0) return res;
        int top=0, left=0, bottom=a.length-1, right=a[0].length-1;
        while (top<=bottom && left<=right) {
            for (int j=left;j<=right;j++) res.add(a[top][j]);
            top++;
            for (int i=top;i<=bottom;i++) res.add(a[i][right]);
            right--;
            if (top<=bottom) {
                for (int j=right;j>=left;j--) res.add(a[bottom][j]);
                bottom--;
            }
            if (left<=right) {
                for (int i=bottom;i>=top;i--) res.add(a[i][left]);
                left++;
            }
        }
        return res;
    }
    public static void main(String[] args) {
        int[][] mat = {{1,2,3},{4,5,6},{7,8,9}};
        System.out.println(spiral(mat)); // [1,2,3,6,9,8,7,4,5]
    }
}
```

## Q8. The Royal Diagonal - Sum of Diagonals

This Java class, `DiagonalSums`, computes sums of the two main diagonals of a square matrix.

```
// Sum of both diagonals
public class DiagonalSums {
    public static int[] diagonalSums(int[][] a) {
        int n = a.length;
        int d1 = 0, d2 = 0;
        for (int i = 0; i < n; i++) {
            d1 += a[i][i];
            d2 += a[i][n-1-i];
        }
        return new int[]{d1, d2};
    }
    public static void main(String[] args) {
        int[][] m = {{1,2,3},{4,5,6},{7,8,9}};
        int[] s = diagonalSums(m);
        System.out.println(s[0] + " , " + s[1]); // 15 , 15
    }
}
```

## Q9. The Messenger’s Path - Path Existence (0/1 grid)

This Java class, `PathExists`, checks if path exists from (0,0) to (n-1,m-1) avoiding blocked cells (1) using BFS.

```
// Path existence in binary grid using BFS
import java.util.*;
public class PathExists {
    public static boolean canReach(int[][] grid) {
        int n = grid.length, m = grid[0].length;
        if (grid[0][0]==1 || grid[n-1][m-1]==1) return false;
        boolean[][] vis = new boolean[n][m];
        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{0,0}); vis[0][0]=true;
        int[][] dirs = {{1,0},{-1,0},{0,1},{0,-1}};
        while (!q.isEmpty()) {
            int[] p = q.poll();
            if (p[0]==n-1 && p[1]==m-1) return true;
            for (int[] d:dirs) {
                int ni=p[0]+d[0], nj=p[1]+d[1];
                if (ni>=0 && ni<n && nj>=0 && nj<m && !vis[ni][nj] && grid[ni][nj]==0) {
                    vis[ni][nj]=true; q.add(new int[]{ni,nj});
                }
            }
        }
        return false;
    }
    public static void main(String[] args) {
        int[][] grid = {{0,0,0},{0,1,0},{0,0,0}};
        System.out.println(canReach(grid)); // true
    }
}
```

## Q10. The Rainwater Pond - Count Ponds (connected components)

### *This Java class, `CountPonds`, counts connected components of 1s (8-directional) in the grid.*

```
// Count ponds (connected components of 1s)
public class CountPonds {
    static int n,m;
    public static int countPonds(int[][] a) {
        n = a.length; m = a[0].length;
        boolean[][] vis = new boolean[n][m];
        int cnt = 0;
        for (int i=0;i<n;i++) for (int j=0;j<m;j++) if (!vis[i][j] && a[i][j]==1) {
            dfs(a,i,j,vis); cnt++;
        }
        return cnt;
    }
    static int[] dr = {-1,-1,-1,0,0,1,1,1};
    static int[] dc = {-1,0,1,-1,1,-1,0,1};
    static void dfs(int[][] a,int r,int c,boolean[][] vis){
        vis[r][c]=true;
        for (int k=0;k<8;k++){
            int nr=r+dr[k], nc=c+dc[k];
            if (nr>=0 && nr<n && nc>=0 && nc<m && !vis[nr][nc] && a[nr][nc]==1) dfs(a,nr,nc,vis);
        }
    }
    public static void main(String[] args){
        int[][] a = {{1,0,1},{0,1,0},{1,0,1}};
        System.out.println(countPonds(a)); // expected 5
    }
}
```

## Q11. Tower of Temples (Hanoi) - Moves Count

### *This Java class, `TowerOfTemples`, returns number of moves required to solve Tower of Hanoi (2^n - 1) and also prints moves for small n.*

```
// Tower of Hanoi moves count and print (recursive)
public class TowerOfTemples {
    public static long movesCount(int n) {
        return (1L << n) - 1;
    }
    public static void hanoi(int n, char from, char to, char aux) {
        if (n==0) return;
        hanoi(n-1, from, aux, to);
        System.out.println("Move disk " + n + " from " + from + " to " + to);
        hanoi(n-1, aux, to, from);
    }
    public static void main(String[] args) {
        int n = 3;
        System.out.println(movesCount(n)); // expected 7
        // hanoi(n,'A','C','B'); // uncomment to print moves
    }
}
```

## Q12. The Magical Staircase

### *This Java class, `MagicalStaircase`, calculates number of ways to reach step n when child can climb 1 or 2 steps.*

```
// Calculate number of ways to reach step n (1 or 2 steps allowed)
public class MagicalStaircase {
    public static long ways(int n) {
        if (n <= 1) return 1;
        long a = 1, b = 1;
        for (int i = 2; i <= n; i++) {
            long c = a + b;
            a = b;
            b = c;
        }
        return b;
    }
    public static void main(String[] args) {
        int n = 4;
        System.out.println(ways(n)); // expected 5
    }
}
```

## Q13. The Sorcerer’s Spell

### *This Java class, `ReverseStringRecursive`, reverses a string using recursion.*

```
// Reverse a string using recursion
public class ReverseStringRecursive {
    public static String reverse(String s) {
        if (s == null || s.length() <= 1) return s;
        return reverse(s.substring(1)) + s.charAt(0);
    }
    public static void main(String[] args) {
        String input = "abc";
        System.out.println(reverse(input)); // cba
    }
}
```

## Q14. The Dragon’s Roar

### *This Java class, `PrintNumbersRecursive`, prints numbers 1..n using recursion.*

```
// Print numbers from 1 to n using recursion
public class PrintNumbersRecursive {
    public static void printUp(int n) {
        if (n == 0) return;
        printUp(n - 1);
        System.out.print(n + " ");
    }
    public static void main(String[] args) {
        int n = 5;
        printUp(n); // 1 2 3 4 5
    }
}
```

## Q15. The Hidden Chamber

### *This Java class, `SumArrayRecursive`, computes sum of array using recursion.*

```
// Sum of array elements using recursion
public class SumArrayRecursive {
    public static int sum(int[] arr, int idx) {
        if (idx == arr.length) return 0;
        return arr[idx] + sum(arr, idx + 1);
    }
    public static void main(String[] args) {
        int[] arr = {1,2,3,4};
        System.out.println(sum(arr,0)); // 10
    }
}
```

## Q16. The Ancient Scroll

### *This Java class, `SearchArray`, finds index of key in array (linear search).*

```
// Linear search
public class SearchArray {
    public static int indexOf(int[] arr, int key) {
        for (int i=0;i<arr.length;i++) if (arr[i]==key) return i;
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {2,5,7,8};
        System.out.println(indexOf(arr,7)); // 2
    }
}
```

## Q17. The Farmer’s Basket

This Java class, `FindOrNot`, returns index if key exists else -1.

```
// Find element or return -1
public class FindOrNot {
    public static int findIndex(int[] arr, int key) {
        for (int i=0;i<arr.length;i++) if (arr[i]==key) return i;
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {10,20,30};
        System.out.println(findIndex(arr,25)); // -1
    }
}
```

## Q18. The Secret Door

This Java class, `BinarySearch`, performs binary search on a sorted array and returns index or -1.

```
// Binary search iterative
public class BinarySearch {
    public static int binarySearch(int[] arr, int key) {
        int l=0, r=arr.length-1;
        while (l<=r) {
            int mid = l + (r-l)/2;
            if (arr[mid]==key) return mid;
            else if (arr[mid]<key) l = mid+1;
            else r = mid-1;
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {1,3,5,7,9};
        System.out.println(binarySearch(arr,7)); // 3
    }
}
```

## Q19. The Archer’s Range

This Java class, `FirstOccurrence`, finds first index of key using binary search.

```
// First occurrence (lower bound for exact key)
public class FirstOccurrence {
    public static int firstOccurrence(int[] arr, int key) {
        int l=0, r=arr.length-1, ans=-1;
        while (l<=r) {
            int mid = l + (r-l)/2;
            if (arr[mid]==key) { ans=mid; r=mid-1; }
            else if (arr[mid]<key) l = mid+1;
            else r = mid-1;
        }
        return ans;
    }
    public static void main(String[] args) {
        int[] arr = {1,2,2,2,3};
        System.out.println(firstOccurrence(arr,2)); // 1
    }
}
```

## Q20. The Treasure Chest

### This Java class, `LastOccurrence`, finds last index of key using binary search.

```
// Last occurrence (upper bound-1 for exact key)
public class LastOccurrence {
    public static int lastOccurrence(int[] arr, int key) {
        int l=0, r=arr.length-1, ans=-1;
        while (l<=r) {
            int mid = l + (r-l)/2;
            if (arr[mid]==key) { ans=mid; l=mid+1; }
            else if (arr[mid]<key) l = mid+1;
            else r = mid-1;
        }
        return ans;
    }
    public static void main(String[] args) {
        int[] arr = {1,2,2,2,3};
        System.out.println(lastOccurrence(arr,2)); // 3
    }
}
```

## Q21. Lower Bound (first index >= target)

### This Java class, `LowerBound`, returns first index where element >= target or arr.length.

```
// Lower bound (first index >= target)
public class LowerBound {
    public static int lowerBound(int[] arr, int target) {
        int l=0, r=arr.length;
        while (l<r) {
            int mid = l + (r-l)/2;
            if (arr[mid] >= target) r = mid;
            else l = mid+1;
        }
        return l;
    }
    public static void main(String[] args) {
        int[] arr = {1,2,4,6,6,8};
        System.out.println(lowerBound(arr,6)); // 3
    }
}
```

## Q22. Upper Bound (first index > target)

### This Java class, `UpperBound`, returns first index where element > target or arr.length.

```
// Upper bound (first index > target)
public class UpperBound {
    public static int upperBound(int[] arr, int target) {
        int l=0, r=arr.length;
        while (l<r) {
            int mid = l + (r-l)/2;
            if (arr[mid] > target) r = mid;
            else l = mid+1;
        }
        return l;
    }
    public static void main(String[] args) {
        int[] arr = {1,2,4,6,6,8};
        System.out.println(upperBound(arr,6)); // 5
    }
}
```

## Q23. Ceil Value (smallest element >= target)

### This Java class, `CeilValue`, returns the smallest element >= target or -1 if none.

```
// Ceil value using lower bound
public class CeilValue {
    public static int ceilValue(int[] arr, int target) {
        int idx = lowerBound(arr, target);
        if (idx == arr.length) return -1;
        return arr[idx];
    }
    private static int lowerBound(int[] arr, int target) {
        int l=0, r=arr.length;
        while (l<r) {
            int mid = l + (r-l)/2;
            if (arr[mid] >= target) r = mid;
            else l = mid+1;
        }
        return l;
    }
    public static void main(String[] args) {
        int[] arr = {1,2,4,6,6,8};
        System.out.println(ceilValue(arr,5)); // 6
        System.out.println(ceilValue(arr,9)); // -1
    }
}
```

## Q24. Floor Value (largest element <= target)

### *This Java class, `FloorValue`, returns the largest element <= target or -1 if none.*

```
// Floor value using upper bound
public class FloorValue {
    public static int floorValue(int[] arr, int target) {
        int ub = upperBound(arr, target);
        int idx = ub - 1;
        if (idx < 0) return -1;
        return arr[idx];
    }
    private static int upperBound(int[] arr, int target) {
        int l=0, r=arr.length;
        while (l<r) {
            int mid = l + (r-l)/2;
            if (arr[mid] > target) r = mid;
            else l = mid+1;
        }
        return l;
    }
    public static void main(String[] args) {
        int[] arr = {1,2,4,6,6,8};
        System.out.println(floorValue(arr,5)); // 4
        System.out.println(floorValue(arr,0)); // -1
    }
}
```

## Q25. The Treasure Map (Linear Search)

### *This Java class, `TreasureMapSearch`, checks if target exists in a 2D matrix via linear search.*

```
// Linear search in 2D matrix
public class TreasureMapSearch {
    public static boolean exists(int[][] mat, int target) {
        for (int i=0;i<mat.length;i++) for (int j=0;j<mat[i].length;j++) if (mat[i][j]==target) return true;
        return false;
    }
    public static void main(String[] args) {
        int[][] mat = {{1,2,3},{4,5,6},{7,8,9}};
        System.out.println(exists(mat,5) ? "Yes" : "No"); // Yes
    }
}
```

## Q26. The Magical Scrolls (Return Index)

### *This Java class, `Find2DIndex`, returns row and column of target in matrix or (-1,-1).*

```
// Find target in 2D matrix and return indices
```

```java
public class Find2DIndex {
    public static int[] findPos(int[][] mat, int target) {
        for (int i=0;i<mat.length;i++) for (int j=0;j<mat[i].length;j++) if (mat[i][j]==target) return new int[]{i,
        return new int[]{-1,-1};
    }
    public static void main(String[] args) {
        int[][] mat = {{10,20,30},{40,50,60},{70,80,90}};
        int[] p = findPos(mat,60);
        System.out.println("(" + p[0] + "," + p[1] + ")"); // (1,2)
    }
}
```

## Q27. The Battle Formation (Flattened Binary Search)

### *This Java class, `BattleFormation`, searches for target in row-wise sorted matrix where each row's first > previous row's last using flattened binary search.*

```java
// Flattened binary search in 2D matrix
public class BattleFormation {
    public static boolean searchMatrix(int[][] mat, int target) {
        if (mat.length==0) return false;
        int n=mat.length, m=mat[0].length;
        int l=0, r=n*m-1;
        while (l<=r) {
            int mid = l + (r-l)/2;
            int val = mat[mid/m][mid%m];
            if (val==target) return true;
            if (val < target) l = mid+1; else r = mid-1;
        }
        return false;
    }
    public static void main(String[] args) {
        int[][] mat = {{1,3,5},{7,10,11},{16,20,30}};
        System.out.println(searchMatrix(mat,10)); // true
    }
}
```

## Q28. The Queen’s Jewels (Binary Search First Occurrence)

### *This Java class, `FirstOccurrence2D`, finds first occurrence of x in row-major order using binary search per row.*

```java
// Find first occurrence in each row (row-major order)
public class FirstOccurrence2D {
    public static int[] firstPos(int[][] mat, int target) {
        for (int i=0;i<mat.length;i++) {
            int idx = firstInRow(mat[i], target);
            if (idx != -1) return new int[]{i, idx};
        }
        return new int[]{-1,-1};
    }
    private static int firstInRow(int[] row, int target) {
        int l=0,r=row.length-1,ans=-1;
        while (l<=r) {
            int mid = l + (r-l)/2;
            if (row[mid]==target) { ans=mid; r=mid-1; }
            else if (row[mid] < target) l = mid+1;
            else r = mid-1;
        }
        return ans;
    }
    public static void main(String[] args) {
        int[][] mat = {{1,2,2},{3,4,4},{5,6,7}};
        int[] p = firstPos(mat,4);
        System.out.println("(" + p[0] + "," + p[1] + ")"); // (1,1)
    }
}
```

## Q29. The Hidden Scrolls (Staircase Search)

### *This Java class, `StaircaseSearch`, uses O(n+m) top-right search to find target in sorted rows and columns.*

```
// Staircase search (start top-right)
public class StaircaseSearch {
    public static boolean staircaseSearch(int[][] mat, int target) {
        if (mat.length==0) return false;
        int n=mat.length, m=mat[0].length;
        int r=0, c=m-1;
        while (r<n && c>=0) {
            if (mat[r][c]==target) return true;
            if (mat[r][c] > target) c--;
            else r++;
        }
        return false;
    }
    public static void main(String[] args) {
        int[][] mat = {{1,4,7,11},{2,5,8,12},{3,6,9,16},{10,13,14,17}};
        System.out.println(staircaseSearch(mat,6)); // true
    }
}
```

## Q30. The Magic Portal (Binary Search 2D)

### *This Java class, `MagicPortal`, searches each row with binary search and prints Activated/Failed.*

```
// Search each row using binary search and report Activated/Failed
import java.util.Arrays;
public class MagicPortal {
    public static boolean findPortal(int[][] mat, int target) {
        for (int[] row: mat) {
            if (Arrays.binarySearch(row, target) >= 0) return true;
        }
        return false;
    }
    public static void main(String[] args) {
        int[][] mat = {{1,2,8},{3,6,10},{7,9,12}};
        System.out.println(findPortal(mat,9) ? "Activated" : "Failed"); // Activated
    }
}
```