Q1. The King's Feast

The King has n plates of food, each with a certain quantity. He wants to know the **maximum food plate**.

- **Input**: n=5, arr=[2,7,1,9,5]
- **Output**: 9
- **Constraints**: 1 ≤ n ≤ 10^5, -10^9 ≤ arr[i] ≤ 10^9

CODE :-

```java
import java.util.Scanner;

public class The_Kings_Feast {
    @Contract(pure = true)
    static int max( @NotNull int a1[]){  1 usage
        int max1=0;
        for(int i=0; i<a1.length; i++){
            if(max1<a1[i]){
                max1=a1[i];
            }
        }
        return max1;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
//        System.out.println("Enter the value of n : ");
        int n=sc.nextInt();
        int arr[]= new int[n];
        for(int i=0; i<n ;i++){
            arr[i]=sc.nextInt();
        }
        System.out.print(max(arr));
    }
}
```

OUTPUT :-

```
5
2 7 1 9 5
9
Process finished with exit code 0
```

Q2. The Lost Soldier

In the battlefield, soldiers are numbered 0…n. One soldier is missing. Find him.

- **Input**: n=5, arr=[0,1,2,4,5]
- **Output**: 3
- **Constraints**: O(n) or O(log n) solution required.

CODE :-

```java
import java.util.Scanner;

public class The_Lost_Soldier {
    @Contract(pure = true)
    static int search( @NotNull int a1[]){  1 usage
        int n=a1.length;
        int sum=0;
        for(int i=0; i<n; i++){
            sum=sum+a1[i];
        }
        int totalSum=n*(n+1)/2;
        return Math.abs(totalSum-sum);
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
//        System.out.println("Enter the value of n : ");
        int n=sc.nextInt();
        int arr[]=new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        System.out.print(search(arr));
    }
}
```

OUTPUT :-

```
5
0 1 2 4 5
3
Process finished with exit code 0
```

Q3. Potion Mixing (Two Sum)

A wizard wants to mix two potions whose strengths add up to target.

- **Input**: n=4, arr=[3,2,4,7], target=6
- **Output**: Indices (1,2)
- **Constraints**: 1 ≤ n ≤ 10^5, -10^9 ≤ arr[i] ≤ 10^9

CODE :-

```java
import java.util.Scanner;
public class Position_Mixing {
    static void indces( @NotNull int a1[], int target){  1 usage
        boolean flag=false;
        for(int i=0; i< a1.length; i++){
            for(int j=i+1; j<a1.length; j++){
                if(a1[i]+a1[j]==target){
                    System.out.print("Indces ("+i+","+j+")");
                    flag=false;
                    break;
                }else{
                    flag=true;
                }
            }
            if(flag==false){
                break;
            }
        }
        if(flag){
            System.out.print(-1);
        }
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int arr[]= new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        int tar= sc.nextInt();
        indces(arr,tar);
    }
}
```

OUTPUT :-

```
4
3 2 4 7
6
Indces (1,2)
Process finished with exit code 0
```

Q4. The Secret Message

A spy wrote a secret message as numbers. To decode it, reverse the array.

- **Input**: arr=[1,2,3,4]
- **Output**: [4,3,2,1]

CODE :-

```java
import java.util.Scanner;
public class The_Secret_Message {
    @Contract("_ -> param1")
    static int @NotNull [] reverse( @NotNull int a1[]){  1 usage
        int temp=0;
        int j=a1.length-1;
        for(int i=0; i<a1.length/2; i++){
            temp=a1[i];
            a1[i]=a1[j];
            a1[j]=temp;
            j--;
        }
        return a1;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int arr[]= new int[n];
        for( int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        int a2[]=reverse(arr);
        for(int i=0; i<a2.length; i++){
            System.out.print(a2[i]+" ");
        }
    }
}
```

OUTPUT :-

```
4
1 2 3 4
4 3 2 1
Process finished with exit code 0
```

Q5. The King's Parade

Soldiers stand in line. Check if their heights are **sorted in non-decreasing order**.

- **Input**: arr=[1,3,5,7] → Output: true
- **Input**: arr=[3,2,1] → Output: false

CODE :-

```java
import java.util.Scanner;
public class The_Kings_Parade {
    @Contract(pure = true)
    static boolean check( @NotNull int a1[]){  1 usage
        boolean flag=false;
        for(int i=0; i< a1.length-1; i++){
            if(a1[i]<=a1[i+1]){
                flag=true;
            }else{
                flag=false;
                break;
            }
        }
        return flag;
    }
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int n= sc.nextInt();
        int arr[]=new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        System.out.print(check(arr));
    }
}
```

OUTPUT :-

```
4
1 3 5 7
true
Process finished with exit code 0
```

```
3
3 2 1
false
Process finished with exit code 0
```

Q6. The Treasure Island

Each island grid has gold. Find the island row with **maximum gold**.

Input:

3 3
1 2 3
4 5 6
7 8 9

- **Output**: Row 2 (sum=24)

CODE :-

```java
import java.util.Scanner;
public class The_Treasure_Island {
    static void maximumGold( @NotNull int a1[][]){  1 usage
        int max=0;
        int row=-1;
        for(int i=0; i<a1.length; i++){
            int sum=0;
            for(int j=0; j<a1[0].length; j++){
                sum=sum+a1[i][j];
            }
            if(max<sum){
                max=sum;
                row=i;
            }
        }
        System.out.print("Row "+row+" (Sum="+max+")");
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int m=sc.nextInt();
        int arr[][] =new int[n][m];
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                arr[i][j]=sc.nextInt();
            }
        }
        maximumGold(arr);
    }
}
```

OUTPUT :-

```
3 3
1 2 3
4 5 6
7 8 9
Row 2 (Sum=24)
Process finished with exit code 0
```

Q7. The Spiral Library

The King built a library where books are kept in spiral shelves. Print them in **spiral order**.

Input:

3 3
1 2 3
4 5 6
7 8 9

- **Output**: [1,2,3,6,9,8,7,4,5]

CODE :-

```java
import java.util.Scanner;
public class The_Spiral_Library {
    static void spiral( @NotNull int matrix[][]){  1 usage
        int rowStart = 0;
        int rowEnd = matrix.length - 1;
        int colStart = 0;
        int colEnd = matrix[0].length - 1;
        while (rowStart <= rowEnd && colStart <= colEnd) {
            for (int col = colStart; col <= colEnd; col++) {
                System.out.print(matrix[rowStart][col] + " ");
            }
            rowStart++;
            for (int row = rowStart; row <= rowEnd; row++) {
                System.out.print(matrix[row][colEnd] + " ");
            }
            colEnd--;
            if (rowStart <= rowEnd) {
                for (int col = colEnd; col >= colStart; col--) {
                    System.out.print(matrix[rowEnd][col] + " ");
                }
                rowEnd--;
            }
            if (colStart <= colEnd) {
                for (int row = rowEnd; row >= rowStart; row--) {
                    System.out.print(matrix[row][colStart] + " ");
                }
                colStart++;
            }
        }
    }
```

```java
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int m=sc.nextInt();
        int mat[][]= new int[n][m];
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                mat[i][j]=sc.nextInt();
            }
        }
        spiral(mat);
    }
}
```

OUTPUT :-

```
3 3
1 2 3
4 5 6
7 8 9
1 2 3 6 9 8 7 4 5
Process finished with exit code 0
```

Q8. The Royal Diagonal

In a royal hall represented as a square, find **sum of both diagonals**.

Input:

3 3
1 2 3
4 5 6
7 8 9

- **Output**: 1+5+9 =15, 3+5+7 = 15

CODE :-

```java
import java.util.Scanner;
public class The_Royal_Diagonal {
    static void diagonal( @NotNull int matrix[][]){  1 usage
        int d1=0;
        for(int i=0; i< matrix.length; i++){
            d1=d1+matrix[i][i];
        }
        int d2=0;
        int i=0;
        for(int j=matrix.length-1; j>=0; j--){
            d2=d2+matrix[i][j];
            i++;
        }
        System.out.print("d1 = "+d1+" , d2 = "+d2);
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int m=sc.nextInt();
        int mat[][]= new int[n][m];
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                mat[i][j]=sc.nextInt();
            }
        }
        diagonal(mat);
    }
}
```

OUTPUT :-

```
3 3
1 2 3
4 5 6
7 8 9
d1 = 15 , d2 = 15
Process finished with exit code 0
```

Q9. The Messenger's Path

A messenger wants to go from (0,0) to (n-1,m-1). Cells with 1 are blocked. Can he reach?

Input:

3 3
0 0 0
0 1 0
0 0 0

- **Output**: true

CODE :-

```java
import java.util.Scanner;
public class The_Messengers_Path {
        static int n, m;  6 usages
        static int[][] grid;  3 usages
        static boolean[][] visited;  3 usages
        public static boolean dfs(int x, int y) {  5 usages
            if (x < 0 || y < 0 || x >= n || y >= m || grid[x][y] == 1 || visited[x][y]) {
                return false;
            }
            if (x == n - 1 && y == m - 1) {
                return true;
            }
            visited[x][y] = true;
            return dfs( x: x + 1, y) || dfs( x: x - 1, y) || dfs(x,  y: y + 1) || dfs(x,  y: y - 1);
        }
        public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);
            n = sc.nextInt();
            m = sc.nextInt();
            grid = new int[n][m];
            visited = new boolean[n][m];
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < m; j++) {
                    grid[i][j] = sc.nextInt();
                }
            }
            System.out.print(dfs( x: 0,  y: 0));
        }
    }
```

OUTPUT :-

```
3 3
0 0 0
0 1 0
0 0 0
true
Process finished with exit code 0
```

Q10. The Rainwater Pond

Count the number of water ponds in a village (1 = water, 0 = land).

Input:

3 3
1 0 1
0 1 0
1 0 1

- **Output**: 5

CODE :-

```java
import java.util.Scanner;
public class The_Rainwater_Pond {
    @Contract(pure = true)
    static int count( @NotNull int matrix[][]){  1 usage
        int ct=0;
        for(int i=0; i< matrix.length; i++){
            for(int j=0; j< matrix[0].length; j++){
                if(matrix[i][j]>0){
                    ct++;
                }
            }
        }
        return ct;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int m=sc.nextInt();
        int mat[][]= new int[n][m];
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                mat[i][j]=sc.nextInt();
            }
        }
        System.out.print(count(mat));
    }
}
```

OUTPUT :-

```
3 3
1 0 1
0 1 0
1 0 1
5
Process finished with exit code 0
```

Q11. Tower of Temples (Hanoi)

Temples have n golden disks. Move them from source → destination using helper temple. Return moves.

- **Input**: n=3
- **Output**: 7

CODE :-

```java
import java.util.Scanner;
public class Tower_Of_Hanoi {
    static int solve(int n, char S, char A, char D){  3 usages
        if(n==1){
            return 1;
        }
        int num=0;
        num=num+solve( n: n-1,S,A,D);
        num=num+solve( n: n-1,D,A,S);
        return num+1;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        System.out.print(solve(n, S: 'A', A: 'B', D: 'C'));
    }
}
```

OUTPUT :-

```
3
7
Process finished with exit code 0
```

Q12. The Magical Staircase

A child climbs 1 or 2 steps. Find number of ways to reach step n.

- **Input**: n=4
- **Output**: 5

CODE:-

```java
import java.util.Scanner;
public class The_Magical_Staircase {
    @Contract(pure = true)
    static int ways(int n1){  1 usage
        if (n1 <= 1)
            return 1;
        int a = 1, b = 1, c = 0;
        for (int i = 2; i <= n1; i++) {
            c = a + b;
            a = b;
            b = c;
        }
        return b;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        System.out.print(ways(n));
    }
}
```

OUTPUT :-

```
4
5
Process finished with exit code 0
```

Q13. The Sorcerer's Spell

Reverse a string using recursion.

- **Input**: abc
- **Output**: cba

CODE :-

```java
import java.util.Scanner;
public class The_Sorcerers_Spell {
    static void reverseString(String S, int n){  2 usages
        if(n<=0){
            return;
        }
        System.out.print(S.charAt(n-1));
        reverseString(S, n: n-1);
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        String s= sc.next();
        reverseString(s,s.length());
    }
}
```

OUTPUT :-

```
abc
cba
Process finished with exit code 0
```

Q14. The Dragon's Roar

Print numbers 1 to n using recursion.

- **Input**: n=5
- **Output**: 1 2 3 4 5

CODE :-

```java
import java.util.Scanner;

public class The_Dragons_Roar {
    static void print(int n){  2 usages
        if(n==1){
            System.out.print(1+" ");
            return;
        }
        print(n-1);
        System.out.print(n+" ");
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n1=sc.nextInt();
        print(n1);
    }
}
```

OUTPUT :-

```
5
1 2 3 4 5
Process finished with exit code 0
```

Q15. The Hidden Chamber

Find sum of array elements using recursion.

- **Input**: arr=[1,2,3,4]
- **Output**: 10

CODE :-

```java
import java.util.Scanner;

public class The_Hidden_Chamber {
    @Contract(pure = true)
    static int arraySum(int arr[],int n){  2 usages
        if(n==0){
            return arr[0];
        }
        int sum=arr[n]+arraySum(arr, n: n-1);
        return sum;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int m=sc.nextInt();
        int a[]=new int[m];
        for(int i=0; i<m; i++){
            a[i]=sc.nextInt();
        }
        System.out.print(arraySum(a, n: a.length-1));
    }
}
```

OUTPUT :-

```
4
1 2 3 4
10
Process finished with exit code 0
```

Q16. The Ancient Scroll

Search for a scroll ID in the archive.

- **Input**: arr=[2,5,7,8], key=7
- **Output**: 2

CODE :-

```java
import java.util.Scanner;

public class The_Ancient_Scroll {
    @Contract(pure = true)
    static int scrollID( @NotNull int a1[], int target){  1 usage
        int index=-1;
        for(int i=0; i<a1.length; i++){
            if(a1[i]==target){
                index=i;
                break;
            }
        }
        return index;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int arr[]= new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("key : ");
        int key= sc.nextInt();
        System.out.print(scrollID(arr,key));
    }
}
```

OUTPUT :-

```
4
2 5 7 8
key : 7
2
Process finished with exit code 0
```

Q17. The Farmer's Basket

Find if a fruit (number) exists in the basket.

- **Input**: arr=[10,20,30], key=25
- **Output**: -1

CODE :-

```java
import java.util.Scanner;

public class The_Farmers_Basket {
    @Contract(pure = true)
    static int fruit( @NotNull int a1[], int n){  1 usage
        int ans=-1;
        for(int i=0; i<a1.length; i++){
            if(a1[i]==n){
                ans=1;
                break;
            }
        }
        return ans;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int arr[]= new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Key : ");
        int key= sc.nextInt();
        System.out.print(fruit(arr,key));
    }
}
```

OUTPUT :-

```
3
10 20 30
Key : 25
-1
Process finished with exit code 0
```

Q18. The Secret Door

Doors are numbered in increasing order. Find target door using binary search.

- **Input**: arr=[1,3,5,7,9], key=7
- **Output**: 3

CODE :-

```java
import java.util.Scanner;
public class The_Secret_Door {
    @Contract(pure = true)
    static int findingDoor( @NotNull int a1[], int n){  1 usage
        int start=0;
        int end=a1.length-1;
        while(start<=end){
            int mid=start+(end-start)/2;
            if(a1[mid]==n){
                return mid;
            } else if (a1[mid]<n) {
                start=mid+1;
            }else{
                end=mid-1;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n= sc.nextInt();
        int arr[]= new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Key : ");
        int key=sc.nextInt();
        System.out.println("Binary Search ..... ");
        System.out.print(findingDoor(arr,key));
    }
}
```

OUTPUT :-

```
5
1 3 5 7 9
Key : 7
Binary Search .....
3
Process finished with exit code 0
```

Q19. The Archer's Range

Find the **first occurrence** of an arrow's distance.

- **Input**: arr=[1,2,2,2,3], key=2
- **Output**: 1

CODE :-

```java
import java.util.Scanner;

public class The_Archers_Range {
    @Contract(pure = true)
    static int firstOccurance( @NotNull int a1[], int n){  1 usage
        int start=0;
        int end=a1.length-1;
        while(start<=end){
            int mid=start+(end-start)/2;
            if(a1[mid]<n){
                return a1[mid];
            } else if (a1[mid]==n) {
                end=mid-1;
            }else{
                start=mid+1;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int arr[]=new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Key : ");
        int key=sc.nextInt();
        System.out.print(firstOccurance(arr,key));
    }
}
```

OUTPUT :-

```
5
1 2 2 2 3
Key : 2
1
Process finished with exit code 0
```

Q20. The Treasure Chest

Find the **last occurrence** of a key using binary search.

- **Input**: arr=[1,2,2,2,3], key=2
- **Output**: 3

CODE :-

```java
import java.util.Scanner;

public class The_Treasure_Chest {
    @Contract(pure = true)
    static int lastOccuance( @NotNull int a1[], int n){  1 usage
        int start=0;
        int end=a1.length-1;
        while(start<=end){
            int mid=start+(end-start)/2;
            if(a1[mid]>n){
                return a1[mid];
            } else if (a1[mid]==n) {
                start=mid+1;
            }else{
                end=mid-1;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int arr[]=new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Key : ");
        int key=sc.nextInt();
        System.out.print(lastOccuance(arr,key));
    }
}
```

OUTPUT :-

```
5
1 2 2 2 3
Key : 2
3
Process finished with exit code 0
```

Q21. The **first index** where the element is **greater than or equal to** the target.

- If element is found → return its first occurrence.
- If not found → return position where it can be inserted.
- If not possible → return n (array size).

Example

Array = [1, 2, 4, 6, 6, 8], target = 6

- Lower bound = index **3** (first 6).

Array = [1, 2, 4, 6, 6, 8], target = 5

- Lower bound = index **3** (as 6 is the first ≥ 5).

CODE :-

```java
import java.util.Scanner;

public class _21_FirstIndex {
    @Contract(pure = true)
    static int firstIndex( @NotNull int a1[], int n){  1 usage
        int start=0;
        int end=a1.length-1;
        int ans=a1.length;
        while(start<=end){
            int mid=start+(end-start)/2;
            if(a1[mid]>=n){
                ans=mid;
                end=mid-1;
            } else{
                start=mid+1;
            }
        }
        return ans;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int arr[]=new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Key : ");
        int key=sc.nextInt();
        System.out.print("Index : "+firstIndex(arr,key));
    }
}
```

OUTPUT :-

```
6
1 2 4 6 6 8
Key : 6
Index : 3
Process finished with exit code 0
```

```
6
1 2 4 6 6 8
Key : 5
Index : 3
Process finished with exit code 0
```

Q22. The **first index** where the element is **strictly greater** than the target.

- If all elements ≤ target → return n.

Example

Array = [1, 2, 4, 6, 6, 8], target = 6

- Upper bound = index **5** (first element greater than 6 is 8).

Array = [1, 2, 4, 6, 6, 8], target = 7

- Upper bound = index **5** (8 is first > 7).

CODE :-

```java
import java.util.Scanner;

public class _22_First_Index {
    @Contract(pure = true)
    static int firstIndex( @NotNull int a1[], int n){  1 usage
        int start=0;
        int end=a1.length-1;
        int ans=a1.length;
        while(start<=end){
            int mid=start+(end-start)/2;
            if(a1[mid]>n){
                ans=mid;
                end=mid-1;
            } else{
                start=mid+1;
            }
        }
        return ans;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int arr[]=new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Key : ");
        int key=sc.nextInt();
        System.out.print("Index : "+firstIndex(arr,key));
    }
}
```

OUTPUT :-

```
6
1 2 4 6 6 8
Key : 6
Index : 5
Process finished with exit code 0
```

```
6
1 2 4 6 6 8
Key : 7
Index : 5
Process finished with exit code 0
```

Q23. The **smallest element ≥ target** (actual value, not index).

- If no such element exists → return -1.

Example

Array = [1, 2, 4, 6, 6, 8], target = 5

- Ceil = 6.

Array = [1, 2, 4, 6, 6, 8], target = 9

CODE :-

```java
public class _23_CeilingNumber {
    @Contract(pure = true)
    static int ceil( @NotNull int a1[], int n){  1 usage
        int start=0;
        int end=a1.length-1;
        int ans=-1;
        while(start<=end){
            int mid=start+(end-start)/2;
            if(a1[mid]>=n){
                ans=a1[mid];
                end=mid-1;
            } else{
                start=mid+1;
            }
        }
        return ans;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n=sc.nextInt();
        int arr[]=new int[n];
        for(int i=0; i<n; i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Key : ");
        int key=sc.nextInt();
        System.out.print(ceil(arr,key));
    }
}
```

OUTPUT :-

```
6
1 2 4 6 6 8
Key : 5
6
Process finished with exit code 0
```

```
6
1 2 4 6 6 8
Key : 9
-1
Process finished with exit code 0
```

Q24. The **largest element ≤ target**.

- If no such element exists → return -1.

Example

Array = [1, 2, 4, 6, 6, 8], target = 5

- Floor = 4.

Array = [1, 2, 4, 6, 6, 8], target = 0

- Floor = -1 (no element ≤ 0).

CODE :-

```java
import java.util.Scanner;

public class _24_Floor_Number {
        @Contract(pure = true)
        static int floor( @NotNull int a1[], int n){  1 usage
            int start=0;
            int end=a1.length-1;
            int ans=-1;
            while(start<=end){
                int mid=start+(end-start)/2;
                if(a1[mid]<=n){
                    ans=a1[mid];
                    start=mid+1;
                } else{
                    end=mid-1;
                }
            }
            return ans;
        }
        public static void main(String[] args) {
            Scanner sc= new Scanner(System.in);
            int n=sc.nextInt();
            int arr[]=new int[n];
            for(int i=0; i<n; i++){
                arr[i]=sc.nextInt();
            }
            System.out.print("Key : ");
            int key=sc.nextInt();
            System.out.print(floor(arr,key));
        }
    }
```

OUTPUT :-

```
6
1 2 4 6 6 8
Key : 5
4
Process finished with exit code 0
```

Q25. The Treasure Map (Linear Search)

A treasure map is represented as a grid n x m. Each cell contains a number. The King wants to know if the treasure (target) exists on the map.

- **Input**:

n=3, m=3
matrix = [[1,2,3], [4,5,6], [7,8,9]]
target = 5

- **Output**: Yes
- **Constraints**: 1 ≤ n,m ≤ 500, -10^6 ≤ matrix[i][j] ≤ 10^6

CODE :-

```java
import java.util.Scanner;
public class The_Treasure_MapLinearSearch {
    @Contract(pure = true)
    static boolean search( @NotNull int a1[][], int target){  1 usage
        boolean flag=false;
        for(int i=0; i<a1.length; i++){
            for(int j=0; j<a1[0].length; j++){
                if(a1[i][j]==target){
                    flag=true;
                }
            }
            if(flag) break;
        }
        return flag;
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n= sc.nextInt();
        int m= sc.nextInt();
        int arr[][]= new int[n][m];
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                arr[i][j]=sc.nextInt();
            }
        }
        System.out.print("target : ");
        int key=sc.nextInt();
        if(search(arr,key)){
            System.out.print("Yes");
        }else{
            System.out.print("No");
        }
    }
}
```

OUTPUT :-

```
3 3
1 2 3
4 5 6
7 8 9
target : 5
Yes
Process finished with exit code 0
```

Q26. The Magical Scrolls (Linear Search Return Index)

In the royal library, scrolls are arranged in a 2D cabinet of size n x m.
Find the row and column of the scroll with ID = target. If not found, return (-1,-1).

- **Input**:

matrix = [[10,20,30], [40,50,60], [70,80,90]]

target = 60

- **Output**: (1,2)
- **Constraints**: $1 \leq n,m \leq 1000$

CODE :-

```java
import java.util.Scanner;
public class The_Magical_ScrollsLinearSearchReturnIndex {
    static void search( @NotNull int a1[][], int target){  1 usage
        boolean flag=false;
        for(int i=0; i<a1.length; i++){
            for(int j=0; j<a1[0].length; j++){
                if(a1[i][j]==target){
                    System.out.print("("+i+","+j+")");
                    flag=true;
                    break;
                }
            }
            if(flag){
                break;
            }
        }
        if(flag==false){
            System.out.print("("+-1+","+-1+")");
        }
    }
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        int n= sc.nextInt();
        int m= sc.nextInt();
        int arr[][]= new int[n][m];
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                arr[i][j]=sc.nextInt();
            }
        }
        System.out.print("target : ");
        int key=sc.nextInt();
        search(arr,key);
    }
}
```

OUTPUT :-

```
3 3
10 20 30
40 50 60
70 80 90
target : 60
(1,2)
Process finished with exit code 0
```

Q27. The Battle Formation (Binary Search - Flattened)

Soldiers stand in a grid formation. Their strengths are sorted row-wise **and** the first element of each row is greater than the last of the previous row.
The commander wants to know if a soldier with strength x exists.

- **Input**:

matrix = [[1,3,5], [7,10,11], [16,20,30]]
target = 10

- **Output**: True
- **Constraints**: $1 \leq n,m \leq 300$

CODE :-

```java
import java.util.Scanner;

public class The_Battle_Formation {
    @Contract(pure = true)
    static boolean search1( @NotNull int mat[][], int target){  1 usage
        int m= mat.length;
        int n=mat[0].length;
        int start=0;
        int end=m-1;

        while(start<=end){
            int mid1=start+(end-start)/2;
            if(target >= mat[mid1][0] && target <= mat[mid1][n-1]){
                return search2(mat,target,mid1);
            } else if (target >= mat[mid1][n-1]) {
                start=mid1+1;
            }else {
                end=mid1-1;
            }
        }
        return false;
    }
    static boolean search2( @NotNull int mat[][], int target, int row){  1 usage
        int n=mat[0].length;
        int st=0;
        int ed=n-1;

        while(st<=ed){
            int mid2=st+(ed-st)/2;
            if(target == mat[row][mid2]){
                return true;
            } else if (target>mat[row][mid2]) {
                st=mid2+1;
            }else{
                ed=mid2-1;
            }
        }
        return false;
    }
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int n= sc.nextInt();
        int m= sc.nextInt();
        int matrix[][]=new int[n][m];
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                matrix[i][j]=sc.nextInt();
            }
        }
        int tar=sc.nextInt();
        System.out.print(search1(matrix,tar));
    }
}
```

OUTPUT :-

```
3 3
1 3 5
7 10 11
16 20 30
10
true
Process finished with exit code 0
```

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

The Queen's jewels are stored in a 2D sorted grid. She wants to find the **first position** of a jewel type x.

- **Input**:

matrix = [[1,2,2], [3,4,4], [5,6,7]]
target = 4

- **Output**: (1,1)
- **Constraints**: $1 \leq n,m \leq 1000$

CODE :-

```java
import java.util.Scanner;

public class The_Queens_Jewels {
    static void search1( @NotNull int mat[][], int target) {  1 usage
        int m = mat.length;
        int n = mat[0].length;
        int start = 0;
        int end = m - 1;

        while (start <= end) {
            int mid1 = start + (end - start) / 2;
            if (target >= mat[mid1][0] && target <= mat[mid1][n - 1]) {
                search2(mat, target, mid1);
                return;
            } else if (target >= mat[mid1][n - 1]) {
                start = mid1 + 1;
            } else {
                end = mid1 - 1;
            }
        }
    }
```

```java
    static void search2( @NotNull int mat[][], int target, int row) {  1 usage
        int n = mat[0].length;
        int st = 0;
        int ed = n - 1;
        int i= -1;
        int j = -1;
        while (st <= ed) {
            int mid2 = st + (ed - st) / 2;
            if (target == mat[row][mid2]) {
                i = row;
                j = mid2;
                ed = mid2 - 1;
            } else if (target > mat[row][mid2]) {
                st = mid2 + 1;
            } else {
                ed = mid2 - 1;
            }
        }
        System.out.print("(" + i + "," + j + ")");
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int matrix[][] = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }
        int tar = sc.nextInt();
        search1(matrix, tar);
    }
}
```

OUTPUT :-

```
3 3
1 2 2
3 4 4
5 6 7
4
(1,1)
Process finished with exit code 0
```

Q29. The Hidden Scrolls (Staircase Search)

The King hides scrolls in a 2D matrix where rows and columns are **sorted**.
Find if a scroll with ID x exists. Use **O(n+m)** method (start from top-right corner).

- **Input**:

matrix = [[1,4,7,11], [2,5,8,12], [3,6,9,16], [10,13,14,17]]
target = 6

- **Output**: True
- **Constraints**: $1 \leq n,m \leq 1000$

CODE :-

```java
import java.util.Scanner;
public class The_Hidden_ScrollsStaircaseSearch {
    @Contract(pure = true)
    static boolean staircaseSearch( @NotNull int mat[][], int target){  1 usage
        int m=mat.length;
        int n=mat[0].length;
        int i=0;
        int j=n-1;
        while(i<n && j>=0){
            if(mat[i][j]==target){
                return true;
            }else if(mat[i][j]>target){
                j--;
            }else {
                i++;
            }
        }
        return false;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int matrix[][] = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }
        int tar = sc.nextInt();
        System.out.print(staircaseSearch(matrix,tar));
    }
}
```

OUTPUT :-

```
4 4
1 4 7 11
2 5 8 12
3 6 9 16
10 13 14 17
6
true
Process finished with exit code 0
```

Q30. The Magic Portal (Binary Search 2D)

A wizard created portals in a 2D grid sorted in ascending order row-wise and column-wise. To activate a portal, he must find a specific number x.
Return "Activated" if found else "Failed".

- **Input**:

matrix = [[1, 2, 8], [3, 6, 10], [7, 9, 12]]
target = 9

- **Output**: Activated
- **Constraints**: $1 \leq n,m \leq 500$

CODE :-

```java
import java.util.Scanner;
public class The_Magic_Portal {
    @Contract(pure = true)
    static boolean search( @NotNull int mat[][], int target){  1 usage
        int m= mat.length;
        int n=mat[0].length;
        int i=0;
        int j=n-1;
        while(i<m && j>=0){
            if(mat[i][j]==target){
                return true;
            }else if(mat[i][j]>target){
                j--;
            }else{
                i++;
            }
        }
        return false;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int matrix[][] = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }
        int tar = sc.nextInt();
        if (search(matrix, tar)) {
            System.out.print("Activated ");
        } else {
            System.out.print("Failed");
        }
    }
```

OUTPUT :-

```
3 3
1 2 8
3 6 10
7 9 12
9
Activated
Process finished with exit code 0
```