1. The King’s Feast

```
import java.util.*;

class Solution {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      int n = sc.nextInt();     // number of plates
      int[] arr = new int[n];

      for (int i = 0; i < n; i++) {
         arr[i] = sc.nextInt();
      }

      int maxFood = arr[0];
      for (int i = 1; i < n; i++) {
         if (arr[i] > maxFood) {
            maxFood = arr[i];
         }
      }

      System.out.println(maxFood);
      sc.close();
   }
}
```

2. The Lost Soldier

```
import java.util.*;

class Solution {
   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();
      }

      // Sum of numbers from 0 to n
      int totalSum = n * (n + 1) / 2;

      // Sum of given numbers
      int arrSum = 0;
      for (int num : arr) {
         arrSum += num;
      }

      // Missing number is the difference
      int missing = totalSum - arrSum;
      System.out.println(missing);
      sc.close();
   }
}
```

3. Potion Mixing (Two Sum)

```
import java.util.*;

class Main {
  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 target = sc.nextInt();

    for(int i = 0; i < n; i++){
      for(int j = i + 1; j < n; j++){
        if(arr[i] + arr[j] == target){
          System.out.println("Indices (" + (i+1) + "," + (j+1) + ")");
          return;
        }
      }
    }
    System.out.println("No pair found");
  }
}
```

4. The Secret Message

```
import java.util.*;

class Main {
  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();
    }

    // Reverse the array
    for(int i = n - 1; i >= 0; i--){
      System.out.print(arr[i] + " ");
    }
  }
}
```

5. The King’s Parade

```
import java.util.*;

class Main {
  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();
    }

    boolean sorted = true;
    for(int i = 1; i < n; i++){
      if(arr[i] < arr[i - 1]){
        sorted = false;
        break;
      }
    }

    System.out.println(sorted);
  }
}
```

6. The Treasure Island

```
import java.util.*;

class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int rows = sc.nextInt();
    int cols = sc.nextInt();

    int[][] gold = new int[rows][cols];

    for(int i = 0; i < rows; i++) {
      for(int j = 0; j < cols; j++) {
        gold[i][j] = sc.nextInt();
      }
    }

    int maxSum = Integer.MIN_VALUE;
    int maxRow = -1;

    for(int i = 0; i < rows; i++) {
      int sum = 0;
      for(int j = 0; j < cols; j++) {
        sum += gold[i][j];
      }
      if(sum > maxSum) {
        maxSum = sum;
        maxRow = i;
      }
    }

    System.out.println("Row " + maxRow + " (sum=" + maxSum + ")");
  }
}
```

7. The Spiral Library

```
import java.util.*;

class Main {
  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();
      }
    }

    int top = 0, bottom = n - 1;
    int left = 0, right = m - 1;

    while (top <= bottom && left <= right) {
      // left to right
      for (int j = left; j <= right; j++)
        System.out.print(arr[top][j] + " ");
      top++;

      // top to bottom
      for (int i = top; i <= bottom; i++)
        System.out.print(arr[i][right] + " ");
      right--;

      // right to left
      if (top <= bottom) {
        for (int j = right; j >= left; j--)
          System.out.print(arr[bottom][j] + " ");
        bottom--;
      }

      // bottom to top
      if (left <= right) {
        for (int i = bottom; i >= top; i--)
          System.out.print(arr[i][left] + " ");
        left++;
      }
    }
  }
}
```

8. The Royal Diagonal

```
import java.util.*;

class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();  // number of rows
    int m = sc.nextInt();  // number of columns (same as n for square)

    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();
      }
    }

    int primarySum = 0;  // main diagonal (top-left to bottom-right)
    int secondarySum = 0; // secondary diagonal (top-right to bottom-left)

    for (int i = 0; i < n; i++) {
      primarySum += arr[i][i];
      secondarySum += arr[i][n - i - 1];
    }

    System.out.println("Primary Diagonal Sum = " + primarySum);
    System.out.println("Secondary Diagonal Sum = " + secondarySum);
  }
}
```

9. The Messenger's Path

```
import java.util.*;

class Main {
  static int n, m;

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    n = sc.nextInt();
    m = sc.nextInt();
    int[][] grid = new int[n][m];

    for(int i = 0; i < n; i++)
      for(int j = 0; j < m; j++)
        grid[i][j] = sc.nextInt();

    boolean reachable = dfs(grid, 0, 0);
    System.out.println(reachable);
  }

  static boolean dfs(int[][] grid, int x, int y) {
    // Out of bounds or blocked
    if(x < 0 || y < 0 || x >= n || y >= m || grid[x][y] == 1) return false;

    // Reached destination
    if(x == n-1 && y == m-1) return true;

    grid[x][y] = 1; // mark visited

    // Explore 4 directions
    return dfs(grid, x+1, y) || dfs(grid, x-1, y) ||
         dfs(grid, x, y+1) || dfs(grid, x, y-1);
  }
}
```

10. The Rainwater Pond

```
import java.util.*;
class Main {
   static int n, m;
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);


      n = sc.nextInt();
      m = sc.nextInt();
      int[][] grid = new int[n][m];


      for(int i = 0; i < n; i++)
         for(int j = 0; j < m; j++)
            grid[i][j] = sc.nextInt();
      int count = 0;
      for(int i = 0; i < n; i++) {
         for(int j = 0; j < m; j++) {
            if(grid[i][j] == 1) {
               count++;
               dfs(grid, i, j);
            }
         }
      }
       System.out.println(count);
   }
   static void dfs(int[][] grid, int x, int y) {
      if(x < 0 || y < 0 || x >= n || y >= m || grid[x][y] == 0) return;
       grid[x][y] = 0; // mark visited
      dfs(grid, x+1, y);
      dfs(grid, x-1, y);
      dfs(grid, x, y+1);
      dfs(grid, x, y-1);
      dfs(grid, x+1, y+1);
      dfs(grid, x+1, y-1);
      dfs(grid, x-1, y+1);
      dfs(grid, x-1, y-1);
   }
}
```

11. Tower of Temples (Hanoi)

```
import java.util.*;

class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int moves = hanoi(n, 'A', 'C', 'B'); // move from A → C using B
    System.out.println(moves);
  }

  static int hanoi(int n, char source, char dest, char helper) {
    if (n == 0) return 0; // base case: no disk

    int count = 0;

    // Move n-1 disks from source to helper
    count += hanoi(n - 1, source, helper, dest);

    // Move nth disk from source to destination
    // System.out.println("Move disk " + n + " from " + source + " to " + dest);
    count++;

    // Move n-1 disks from helper to destination
    count += hanoi(n - 1, helper, dest, source);

    return count;
  }
}
```

12. . The Magical Staircase

```
import java.util.*;

class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    System.out.println(climbStairs(n));
  }

  static int climbStairs(int n) {
    if(n == 0 || n == 1) return 1;

    int a = 1, b = 1;
    for(int i = 2; i <= n; i++) {
      int temp = a + b;
      a = b;
      b = temp;
    }
    return b;
  }
}
```

13. The Sorcerer's Spell

```
import java.util.*;

class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String str = sc.nextLine();
    String reversed = reverse(str);
    System.out.println(reversed);
  }

  static String reverse(String s) {
    if (s.length() <= 1) return s;  // base case
    return reverse(s.substring(1)) + s.charAt(0); // recursive call
  }
}
```

14. . The Dragon's Roar

```
import java.util.*;

class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    printNumbers(1, n);
  }

  static void printNumbers(int current, int n) {
    if(current > n) return; // base case
    System.out.print(current + " ");
    printNumbers(current + 1, n); // recursive call
  }
}
```

15. The Hidden Chamber

```
import java.util.*;

class Main {
  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 sum = sumArray(arr, n);
    System.out.println(sum);
  }

  static int sumArray(int[] arr, int n) {
    if(n == 0) return 0; // base case
    return sumArray(arr, n - 1) + arr[n - 1]; // recursive call
  }
}
```

16. The Ancient Scroll

```
import java.util.*;

class Main {
  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 key = sc.nextInt();
    int index = search(arr, key);
    System.out.println(index);
  }

  static int search(int[] arr, int key) {
    for(int i = 0; i < arr.length; i++) {
      if(arr[i] == key) return i; // found
    }
    return -1; // not found
  }
}
```

17. The Farmer's Basket

```
import java.util.*;

class Main {
  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 key = sc.nextInt();
    System.out.println(search(arr, key));
  }

  static int search(int[] arr, int key) {
    for(int i = 0; i < arr.length; i++) {
      if(arr[i] == key) return i;
    }
    return -1;
  }
}
```

18. The Secret Door

```
import java.util.*;

class Main {
  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 key = sc.nextInt();
    int index = binarySearch(arr, key);
    System.out.println(index);
  }

  static int binarySearch(int[] arr, int key) {
    int left = 0, right = arr.length - 1;

    while(left <= right) {
      int mid = left + (right - left) / 2;
```

```
            if(arr[mid] == key) return mid;
            else if(arr[mid] < key) left = mid + 1;
            else right = mid - 1;
        }

        return -1; // not found
    }
}
```

19. The Archer's Range

```
import java.util.*;

class Main {
    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 key = sc.nextInt();
        int index = firstOccurrence(arr, key);
        System.out.println(index);
    }

    static int firstOccurrence(int[] arr, int key) {
        int left = 0, right = arr.length - 1;
        int result = -1;

        while(left <= right) {
            int mid = left + (right - left) / 2;

            if(arr[mid] == key) {
                result = mid;     // potential first occurrence
                right = mid - 1;  // move left to find earlier occurrence
            }
            else if(arr[mid] < key) left = mid + 1;
            else right = mid - 1;
        }

        return result;
    }
}
```

20. The Treasure Chest

```
import java.util.*;

class Main {
    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 key = sc.nextInt();
        int index = lastOccurrence(arr, key);
        System.out.println(index);
    }

    static int lastOccurrence(int[] arr, int key) {
        int left = 0, right = arr.length - 1;
        int result = -1;

        while(left <= right) {
            int mid = left + (right - left) / 2;
```

```
            if(arr[mid] == key) {
                result = mid;     // potential last occurrence
                left = mid + 1;   // move right to find later occurrence
            }
            else if(arr[mid] < key) left = mid + 1;
            else right = mid - 1;
        }

        return result;
    }
}
```

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

```
import java.util.*;

class Main {
    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 target = sc.nextInt();
        int lb = lowerBound(arr, target);
        System.out.println(lb);
    }

    static int lowerBound(int[] arr, int target) {
        int left = 0, right = arr.length - 1;
        int result = arr.length; // default if target is larger than all elements

        while(left <= right) {
            int mid = left + (right - left) / 2;

            if(arr[mid] >= target) {
                result = mid;   // possible lower bound
                right = mid - 1; // search left for first occurrence
            } else {
                left = mid + 1;
            }
        }

        return result;
    }
}
```

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

```
import java.util.*;

class Main {
    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 target = sc.nextInt();
        int ub = upperBound(arr, target);
        System.out.println(ub);
    }

    static int upperBound(int[] arr, int target) {
        int left = 0, right = arr.length - 1;
```

```
    int result = arr.length; // default if target >= all elements

    while(left <= right) {
      int mid = left + (right - left) / 2;

      if(arr[mid] > target) {
        result = mid;      // potential upper bound
        right = mid - 1;   // search left for first greater element
      } else {
        left = mid + 1;    // move right if arr[mid] <= target
      }
    }

    return result;
  }
}
```

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

```
import java.util.*;

class Main {
  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 target = sc.nextInt();
    int ceil = findCeil(arr, target);
    System.out.println(ceil);
  }

  static int findCeil(int[] arr, int target) {
    int left = 0, right = arr.length - 1;
    int result = -1; // default if no element >= target

    while(left <= right) {
      int mid = left + (right - left) / 2;

      if(arr[mid] >= target) {
        result = arr[mid];  // potential ceil
        right = mid - 1;    // search left for smaller eligible element
      } else {
        left = mid + 1;     // move right if arr[mid] < target
      }
    }

    return result;
  }
}
```

24. The largest element ≤ target

```
import java.util.*;

class Main {
  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 target = sc.nextInt();
    int floor = findFloor(arr, target);
    System.out.println(floor);
  }

  static int findFloor(int[] arr, int target) {
    int left = 0, right = arr.length - 1;
    int result = -1; // default if no element <= target

    while(left <= right) {
      int mid = left + (right - left) / 2;

      if(arr[mid] <= target) {
        result = arr[mid];   // potential floor
        left = mid + 1;      // search right for larger eligible element
      } else {
        right = mid - 1;     // move left if arr[mid] > target
      }
    }

    return result;
  }
}
```

25.The Treasure Map (Linear Search)

```
import java.util.*;

class Main {
  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 target = sc.nextInt();

    System.out.println(findTreasure(matrix, target) ? "Yes" : "No");
  }

  static boolean findTreasure(int[][] matrix, int target) {
    for(int i = 0; i < matrix.length; i++)
      for(int j = 0; j < matrix[0].length; j++)
        if(matrix[i][j] == target)
          return true;
    return false;
  }
}
```

26. The Magical Scrolls (Linear Search Return Index)

```
import java.util.*;

class Main {
  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 target = sc.nextInt();

    int[] pos = findScroll(matrix, target);
    System.out.println("(" + pos[0] + "," + pos[1] + ")");
  }

  static int[] findScroll(int[][] matrix, int target) {
    for(int i = 0; i < matrix.length; i++) {
      for(int j = 0; j < matrix[0].length; j++) {
        if(matrix[i][j] == target) {
          return new int[]{i, j};  // row, column
        }
      }
    }
    return new int[]{-1, -1}; // not found
  }
}
```

27. The Battle Formation (Binary Search - Flattened)

```
import java.util.*;
class Main {
  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 target = sc.nextInt();
    System.out.println(searchSoldier(matrix, target) ? "True" : "False");
  }

  static boolean searchSoldier(int[][] matrix, int target) {
    int n = matrix.length;
    int m = matrix[0].length;
    int left = 0, right = n * m - 1;

    while(left <= right) {
      int mid = left + (right - left) / 2;
      int midValue = matrix[mid / m][mid % m]; // map 1D index to 2D

      if(midValue == target) return true;
      else if(midValue < target) left = mid + 1;
      else right = mid - 1;
    }

    return false;
  }
}
```

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

```
import java.util.*;
class Main {
  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 target = sc.nextInt();
    int[] pos = findFirstJewel(matrix, target);
    System.out.println("(" + pos[0] + "," + pos[1] + ")");
  }
   static int[] findFirstJewel(int[][] matrix, int target) {
    for(int i = 0; i < matrix.length; i++) {
      for(int j = 0; j < matrix[0].length; j++) {
        if(matrix[i][j] == target) {
          return new int[]{i, j};
        }
      }
    }
    return new int[]{-1, -1}; // not found
  }
}
```

29. The Hidden Scrolls (Staircase Search)

```
import java.util.*;

class Main {
  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 target = sc.nextInt();
    System.out.println(searchScroll(matrix, target) ? "True" : "False");
  }

  static boolean searchScroll(int[][] matrix, int target) {
    int row = 0;
    int col = matrix[0].length - 1;

    while(row < matrix.length && col >= 0) {
      if(matrix[row][col] == target) {
        return true;     // found
      } else if(matrix[row][col] > target) {
        col--;           // move left
      } else {
        row++;           // move down
      }
    }

    return false; // not found
  }
}
```

30. The Magic Portal (Binary Search 2D)

```
import java.util.*;

class Main {
  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 target = sc.nextInt();
    System.out.println(activatePortal(matrix, target));
  }

  static String activatePortal(int[][] matrix, int target) {
    int row = 0;
    int col = matrix[0].length - 1;

    while(row < matrix.length && col >= 0) {
      if(matrix[row][col] == target) {
        return "Activated";  // portal found
      } else if(matrix[row][col] > target) {
        col--;             // move left
      } else {
        row++;             // move down
      }
    }

    return "Failed";         // portal not found
  }
}
```

31.