GATE TECHNICAL TRAINING – DSA CODING PRACTICE PROBLEMS 2026

DATE: 13-11-24 **NAME: JASHVARTHINI R – CSBS**

1. K'TH SMALLEST ELEMENT IN UNSORTED ARRAY

Given an array arr[] of N distinct elements and a number K, where K is smaller than the size of the array. Find the K'th smallest element in the given array.

Examples:

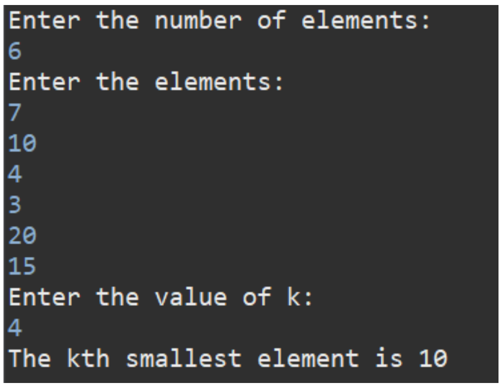
Input: arr[] = {7, 10, 4, 3, 20, 15}, K = 3
Output: 7

Input: arr[] = {7, 10, 4, 3, 20, 15}, K = 4
Output: 10

CODE:

```java
package JavaDSA;
import java.util.*;

public class KthSmallestElement {

        public static int kthSmallest(int arr[], int k) {
            Arrays.sort(arr);
            return arr[k-1];
        }

        public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the number of elements:");
            int n = sc.nextInt();

            int arr[]= new int[n];
            System.out.println("Enter the elements:");
            for(int i=0; i<n; i++) {
                arr[i] = sc.nextInt();
            }

            System.out.println("Enter the value of k:");
            int k = sc.nextInt();

            int output = kthSmallest(arr, k);
            System.out.println("The kth smallest element is " + output);

            sc.close();
        }
    }
```

OUTPUT:



```
Enter the number of elements:
6
Enter the elements:
7
10
4
3
20
15
Enter the value of k:
4
The kth smallest element is 10
```

TIME COMPLEXITY: O(n log n)

2. MINIMIZE THE HEIGHTS II

Given an array arr[] denoting heights of N towers and a positive integer K.

For each tower, you must perform exactly one of the following operations exactly once.

- Increase the height of the tower by K
- Decrease the height of the tower by K

Find out the minimum possible difference between the height of the shortest and tallest towers after you have modified each tower.

You can find a slight modification of the problem here.
Note: It is compulsory to increase or decrease the height by K for each tower. After the operation, the resultant array should not contain any negative integers.

Examples :

Input: k = 2, arr[] = {1, 5, 8, 10}

Output: 5

Explanation: The array can be modified as {1+k, 5-k, 8-k, 10-k} = {3, 3, 6, 8}.The difference between the largest and the smallest is 8-3 = 5.

Input: k = 3, arr[] = {3, 9, 12, 16, 20}

Output: 11

Explanation: The array can be modified as {3+k, 9+k, 12-k, 16-k, 20-k} -> {6, 12, 9, 13, 17}.The difference between the largest and the smallest is 17-6 = 11.

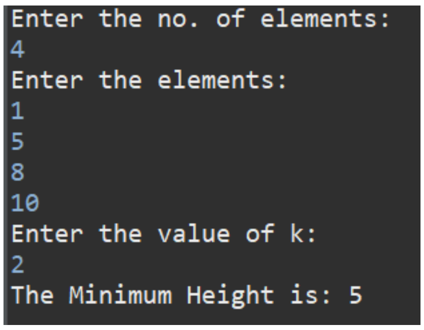
Expected Time Complexity: O(n*logn)
Expected Auxiliary Space: O(n)

Constraints
1 ≤ k ≤ 107
1 ≤ n ≤ 105
1 ≤ arr[i] ≤ 107

CODE:

```java
package JavaDSA;

import java.util.Arrays;
import java.util.Scanner;

public class MinimumHeights2 {

    public static int minimumHeights(int arr[], int k, int n) {
        Arrays.sort(arr);
        int mini= arr[n-1] - arr[0];

        for(int i=0; i<n; i++) {
            if(arr[i]-k<0) {
                continue;
            }
            mini=Math.min(mini, Math.max(arr[n-1]-k, arr[i-1]+k)-
Math.min(arr[0]+k, arr[i]+k ));
        }

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

        System.out.println("Enter the no. of elements: ");
        int n = sc.nextInt();

        int arr[] = new int[n];
        System.out.println("Enter the elements: ");
        for(int i=0; i<n; i++) {
            arr[i]= sc.nextInt();
        }

        System.out.println("Enter the value of k: ");
        int k= sc.nextInt();

        int output = minimumHeights(arr, k, n);
        System.out.println("The Minimum Height is: " + output);

        sc.close();
    }

}
```

OUTPUT:



```
Enter the no. of elements:
4
Enter the elements:
1
5
8
10
Enter the value of k:
2
The Minimum Height is: 5
```

TIME COMPLEXITY: O(n*logn)

3. PARENTHESIS CHECKER

You are given a string s representing an expression containing various types of brackets: {}, (), and []. Your task is to determine whether the brackets in the expression are balanced. A balanced expression is one where every opening bracket has a corresponding closing bracket in the correct order.

Examples :

Input: s = "{([])}"

Output: true

Explanation:
- In this expression, every opening bracket has a corresponding closing bracket.
- The first bracket { is closed by }, the second opening bracket (is closed by), and the third opening bracket [is closed by].
- As all brackets are properly paired and closed in the correct order, the expression is considered balanced.

Input: s = "()"

Output: true

Explanation:
- This expression contains only one type of bracket, the parentheses (and).
- The opening bracket (is matched with its corresponding closing bracket).
- Since they form a complete pair, the expression is balanced.

Input: s = "([]"

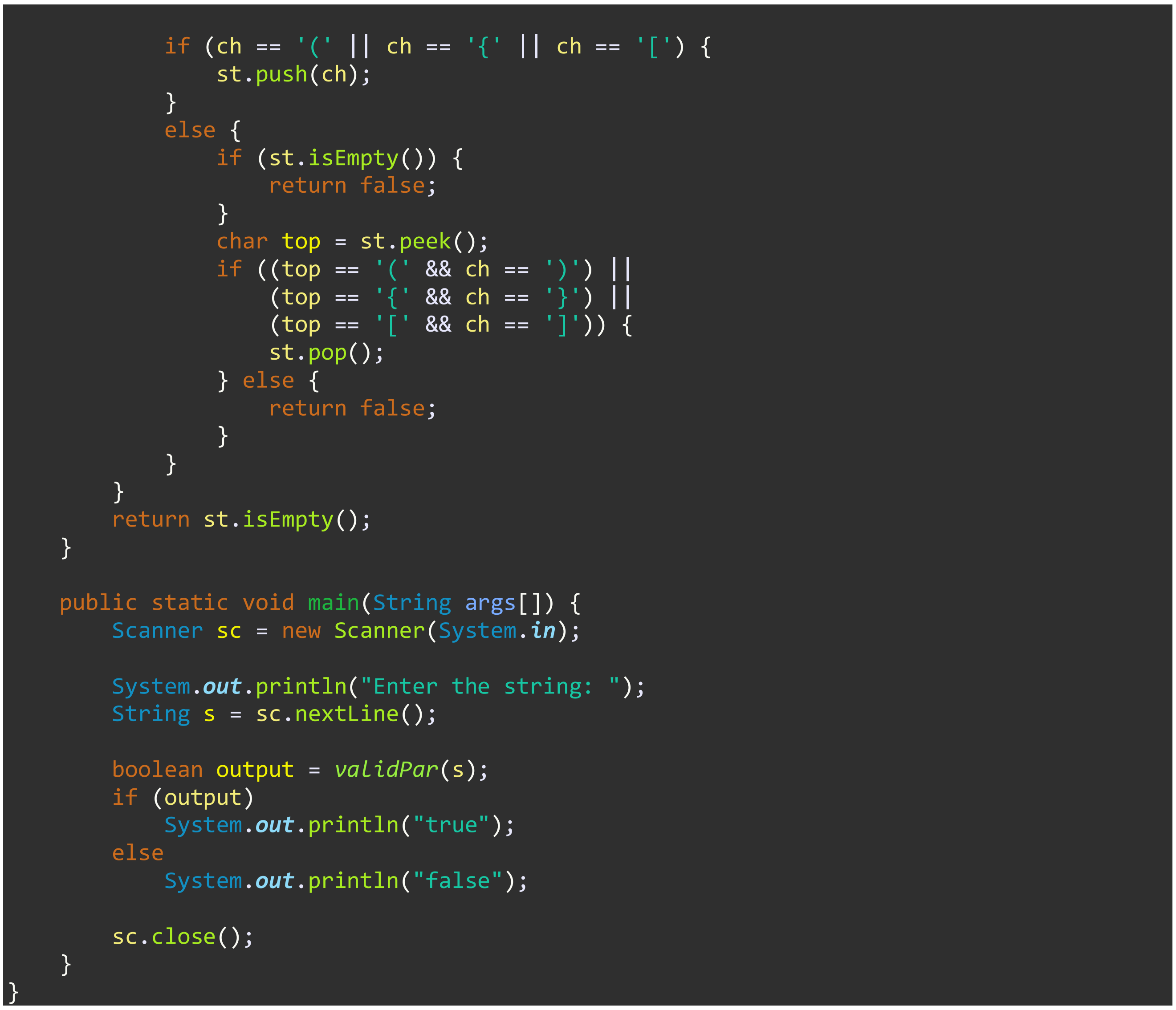
Output: false

Explanation:
- This expression contains only one type of bracket, the parentheses (and).
- The opening bracket (is matched with its corresponding closing bracket).
- Since they form a complete pair, the expression is balanced.

Constraints:
1 ≤ s.size() ≤ 106
s[i] ∈ {'{', '}', '(', ')', '[', ']'}

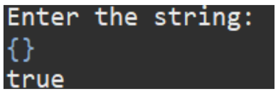
CODE:

```java
package JavaDSA;

import java.util.Scanner;
import java.util.Stack;

public class ParenthesesChecker {

    public static boolean validPar(String s) {
        Stack<Character> st = new Stack<>();

        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
```

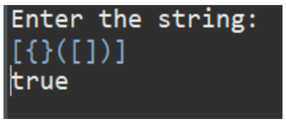


```java
            if (ch == '(' || ch == '{' || ch == '[') {
                st.push(ch);
            }
            else {
                if (st.isEmpty()) {
                    return false;
                }
                char top = st.peek();
                if ((top == '(' && ch == ')') ||
                    (top == '{' && ch == '}') ||
                    (top == '[' && ch == ']')) {
                    st.pop();
                } else {
                    return false;
                }
            }
        }
        return st.isEmpty();
    }

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

        System.out.println("Enter the string: ");
        String s = sc.nextLine();

        boolean output = validPar(s);
        if (output)
            System.out.println("true");
        else
            System.out.println("false");

        sc.close();
    }
}
```

OUTPUT:



```
Enter the string:
{}
true
```



```
Enter the string:
[{}([])]
true
```

TIME COMPLEXITY: O(n)

4. EQUILIBRIUM POINT

Given an array arr of non-negative numbers. The task is to find the first equilibrium point in an array. The equilibrium point in an array is an index (or position) such that the sum of all elements before that index is the same as the sum of elements after it.

Note: Return equilibrium point in 1-based indexing. Return -1 if no such point exists.

Examples:

Input: arr[] = [1, 3, 5, 2, 2]
Output: 3

Explanation: The equilibrium point is at position 3 as the sum of elements before it (1+3) = sum of elements after it (2+2).

Input: arr[] = [1]
Output: 1

Explanation: Since there's only one element hence it's only the equilibrium point.

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

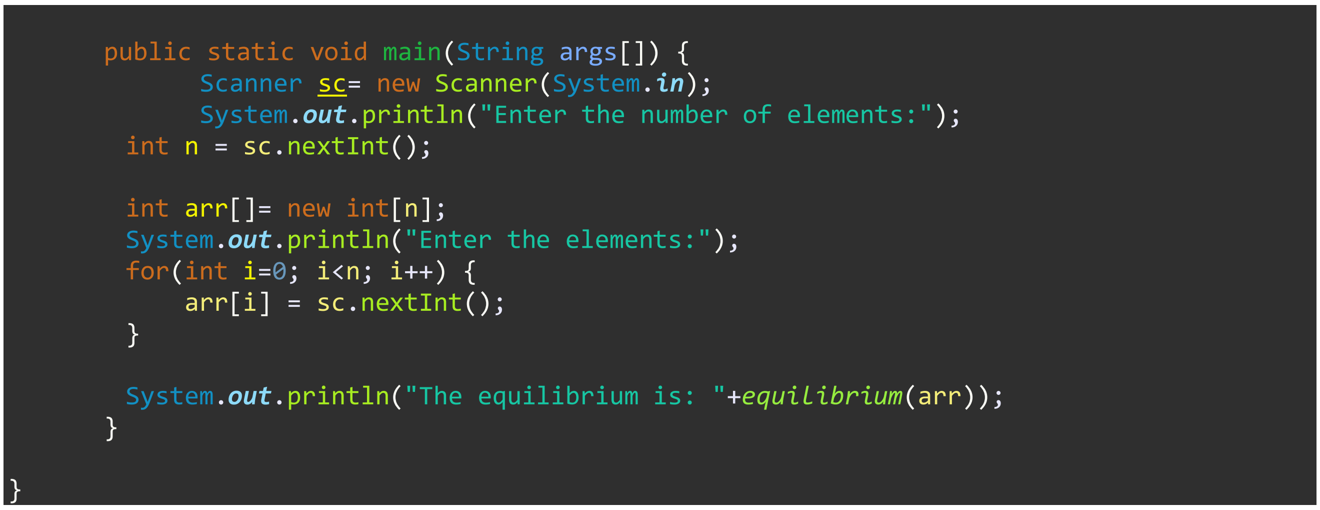
Explanation: There is no equilibrium point in the given array.

Expected Time Complexity: O(n)
Expected Auxiliary Space: O(1)

Constraints:
1 <= arr.size <= 106
0 <= arr[i] <= 109

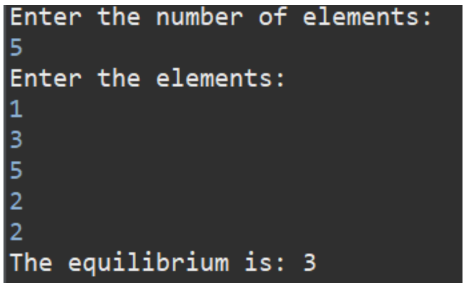
CODE:

```java
package JavaDSA;
import java.util.*;

public class EquilibriumPoint {
      public static int equilibrium(int[] arr) {
            int n=arr.length;

        if(n==1){
            return 1;
        }

        long prefix[] = new long[n];
        long suffix[] = new long[n];

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

        suffix[n-1]=arr[n-1];
        for(int i= n-2; i>=0; i--){
            suffix[i]=suffix[i+1]+arr[i];
        }

        for(int i=0; i<n; i++){
            if(prefix[i]==suffix[i]){
                return i+1;
            }

        }
        return -1;
      }
```



```
      public static void main(String args[]) {
            Scanner sc= new Scanner(System.in);
            System.out.println("Enter the number of elements:");
        int n = sc.nextInt();

        int arr[]= new int[n];
        System.out.println("Enter the elements:");
        for(int i=0; i<n; i++) {
            arr[i] = sc.nextInt();
        }

        System.out.println("The equilibrium is: "+equilibrium(arr));
      }

}
```

OUTPUT:



```
Enter the number of elements:
5
Enter the elements:
1
3
5
2
2
The equilibrium is: 3
```

TIME COMPLEXITY: O(n)

5. BINARY SEARCH

Given a sorted array arr and an integer k, find the position(0-based indexing) at which k is present in the array using binary search.

Note: If multiple occurrences are there, please return the smallest index.

Examples:

Input: arr[] = [1, 2, 3, 4, 5], k = 4

Output: 3

Explanation: 4 appears at index 3.

Input: arr[] = [11, 22, 33, 44, 55], k = 445

Output: -1

Explanation: 445 is not present.

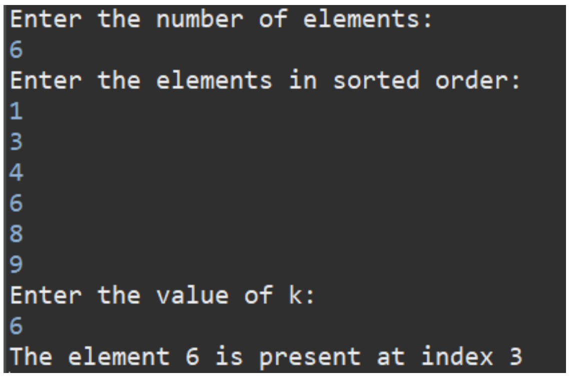
Note: Try to solve this problem in constant space i.e O(1)

Constraints:
1 <= arr.size() <= 105
1 <= arr[i] <= 106
1 <= k <= 106

CODE:

```java
package JavaDSA;

import java.util.Scanner;

public class BinarySearch {
    public static int binarySearch(int[] arr, int k) {
        int low = 0;
        int high = arr.length - 1;

        while (low <= high) {
            int mid = (low + high) / 2;

            if (arr[mid] == k) {
                return mid;
            }
            else if (arr[mid] < k) {
                low = mid + 1;
            }
            else {
                high = mid - 1;
            }
        }

        return -1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements:");
        int n = sc.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the elements in sorted order:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        System.out.println("Enter the value of k:");
        int k = sc.nextInt();

        int output = binarySearch(arr, k);
        if (output == -1)
            System.out.println("Element is not present in the array");
        else
            System.out.println("The element " + k + " is present at index " + output);

        sc.close();
    }
}
```

OUTPUT:



```
Enter the number of elements:
6
Enter the elements in sorted order:
1
3
4
6
8
9
Enter the value of k:
6
The element 6 is present at index 3
```

TIME COMPLEXITY: O(log n)

6. NEXT GREATER ELEMENT

Given an array arr[ ] of integers, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.
If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

Examples

Input: arr[] = [1, 3, 2, 4]

Output: [3, 4, 4, -1]

Explanation: The next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4, since it doesn't exist, it is -1.

Input: arr[] = [6, 8, 0, 1, 3]

Output: [8, -1, 1, 3, -1]

Explanation: The next larger element to 6 is 8, for 8 there is no larger elements hence it is -1, for 0 it is 1 , for 1 it is 3 and then for 3 there is no larger element on right and hence -1.

Input: arr[] = [10, 20, 30, 50]

Output: [20, 30, 50, -1]

Explanation: For a sorted array, the next element is next greater element also exxept for the last element.

Input: arr[] = [50, 40, 30, 10]

Output: [-1, -1, -1, -1]

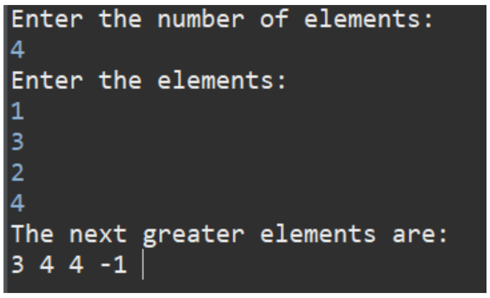
Explanation: There is no greater element for any of the elements in the array, so all are -1.

Constraints:
1 ≤ arr.size() ≤ 106
0 ≤ arr[i] ≤ 109

CODE:

```java
package JavaDSA;
import java.util.Scanner;
import java.util.Stack;

public class NextGreaterElement {

    public static int[] findNextGreaterElements(int[] arr) {
        int n = arr.length;
        int[] result = new int[n];
        Stack<Integer> stack = new Stack<>();

        for (int i = n - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }

            if (!stack.isEmpty()) {
                result[i] = stack.peek();
            } else {
                result[i] = -1;
            }

            stack.push(arr[i]);
        }

        return result;
    }

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

        System.out.println("Enter the number of elements:");
        int n = sc.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        int[] result = findNextGreaterElements(arr);

        System.out.println("The next greater elements are:");
        for (int res : result) {
            System.out.print(res + " ");
        }

        sc.close();
    }
}
```

OUTPUT:



```
Enter the number of elements:
4
Enter the elements:
1
3
2
4
The next greater elements are:
3 4 4 -1
```

TIME COMPLEXITY: O(n)

7. UNION OF 2 ARRAYS WITH DUPLICATE ELEMENTS

Given two arrays a[] and b[], the task is to find the number of elements in the union between these two arrays.

The Union of the two arrays can be defined as the set containing distinct elements from both arrays. If there are repetitions, then only one element occurrence should be there in the union.

Note: Elements are not necessarily distinct.

Examples

Input: a[] = [1, 2, 3, 4, 5], b[] = [1, 2, 3]

Output: 5

Explanation: 1, 2, 3, 4 and 5 are the elements which comes in the union setof both arrays. So count is 5.

Input: a[] = [85, 25, 1, 32, 54, 6], b[] = [85, 2]
Output: 7

Explanation: 85, 25, 1, 32, 54, 6, and 2 are the elements which comes in the union set of both arrays. So count is 7.

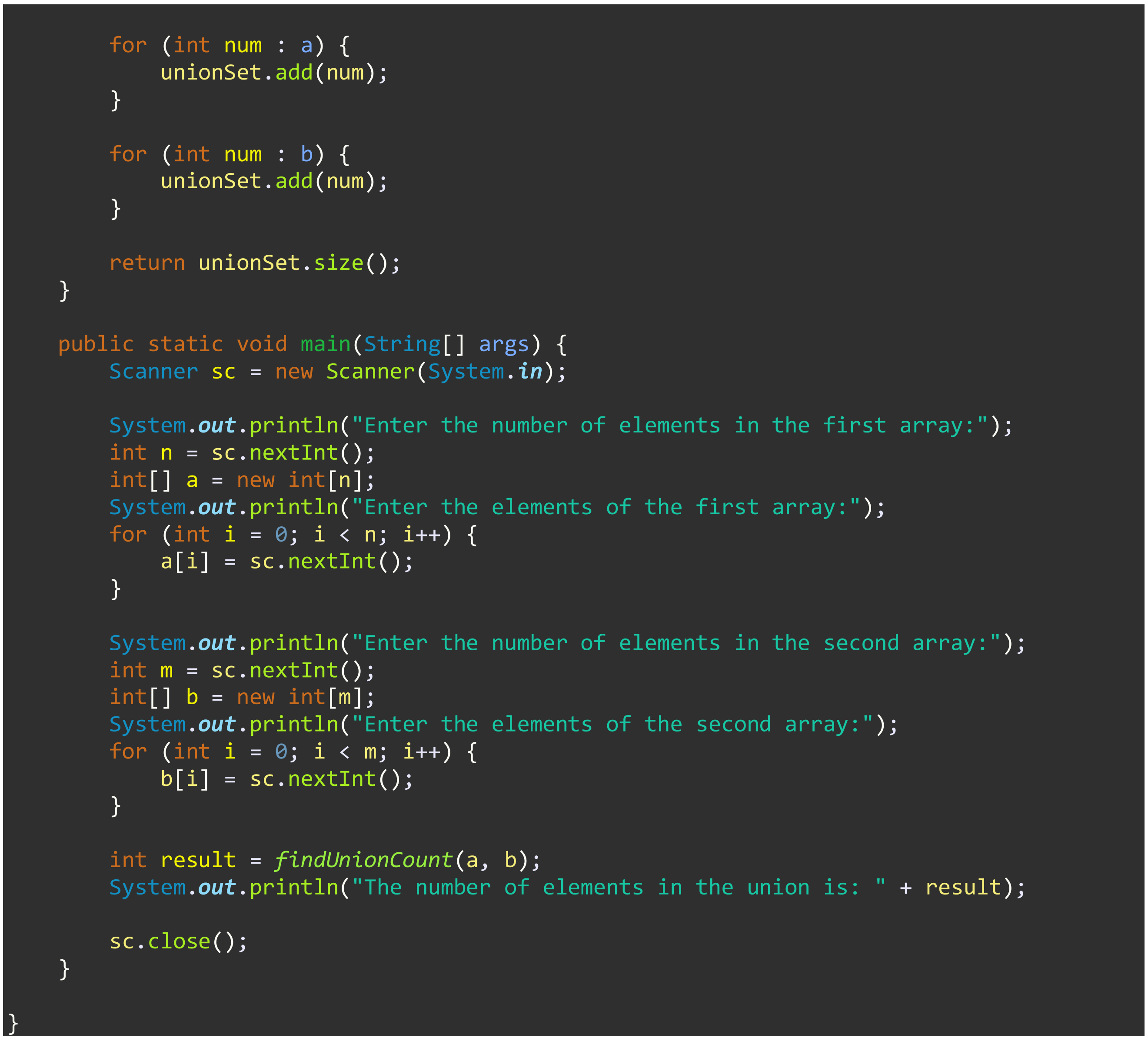
Input: a[] = [1, 2, 1, 1, 2], b[] = [2, 2, 1, 2, 1]
Output: 2

Explanation: We need to consider only distinct. So count is 2.

Constraints:
1 ≤ a.size(), b.size() ≤ 106
0 ≤ a[i], b[i] < 105

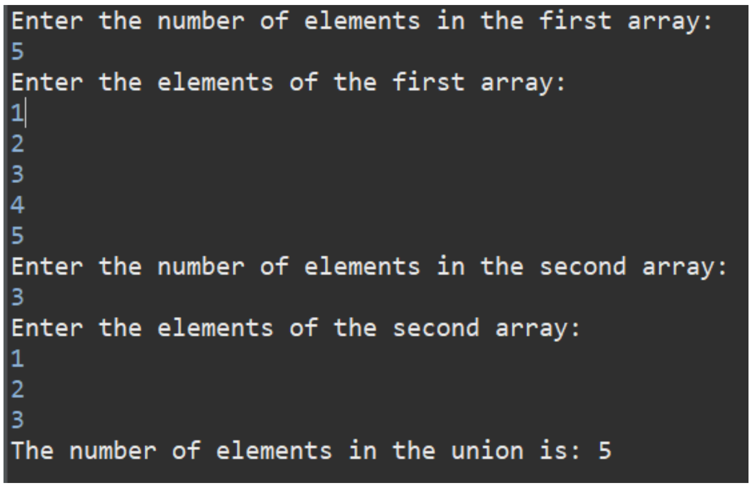
CODE:

```
package JavaDSA;
import java.util.HashSet;
import java.util.Scanner;

public class UnionOfArrays {

    public static int findUnionCount(int[] a, int[] b) {
        HashSet<Integer> unionSet = new HashSet<>();
```



```java
        for (int num : a) {
            unionSet.add(num);
        }

        for (int num : b) {
            unionSet.add(num);
        }

        return unionSet.size();
    }

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

        System.out.println("Enter the number of elements in the first array:");
        int n = sc.nextInt();
        int[] a = new int[n];
        System.out.println("Enter the elements of the first array:");
        for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }

        System.out.println("Enter the number of elements in the second array:");
        int m = sc.nextInt();
        int[] b = new int[m];
        System.out.println("Enter the elements of the second array:");
        for (int i = 0; i < m; i++) {
            b[i] = sc.nextInt();
        }

        int result = findUnionCount(a, b);
        System.out.println("The number of elements in the union is: " + result);

        sc.close();
    }

}
```

OUTPUT:



```
Enter the number of elements in the first array:
5
Enter the elements of the first array:
1
2
3
4
5
Enter the number of elements in the second array:
3
Enter the elements of the second array:
1
2
3
The number of elements in the union is: 5
```

TIME COMPLEXITY: O(n+m)