

DSA PRACTICE – 4

Name : Tharani Kumar K

DEPT : IT

Date : 13/11/2024

1. Kth smallest Element In The Array.

Given an array `arr[]` and an integer `k` where `k` is smaller than the size of the array, the task is to find the `k`th smallest element in the given array.

Follow up: Don't solve it using the inbuilt sort function.

Examples :

Input: `arr[] = [7, 10, 4, 3, 20, 15]`, `k = 3`

Output: 7

Explanation: 3rd smallest element in the given array is 7.

Input: `arr[] = [2, 3, 1, 20, 15]`, `k = 4`

Output: 15

Explanation: 4th smallest element in the given array is 15.

Code:

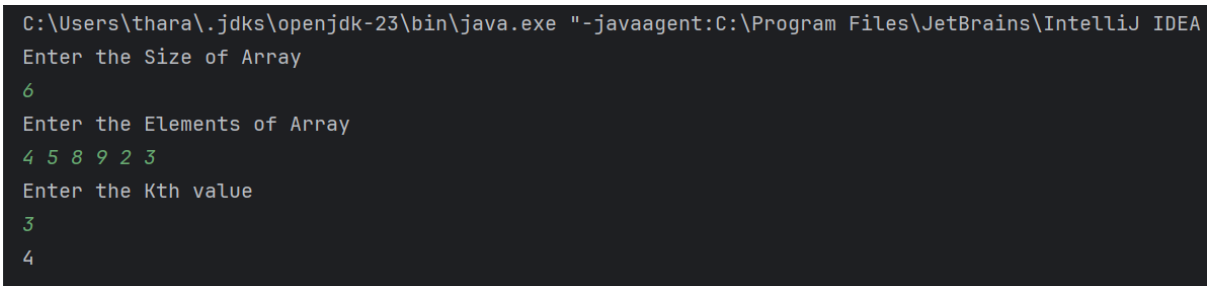
```
import java.util.*;
class KthSmallest{
    public static int helper(int[] arr, int k){
        PriorityQueue<Integer> pq=new PriorityQueue<>();
        int ans=0;
        for(int i:arr){
            pq.add(i);
        }
        while(k>0){
            ans=pq.poll();
            k--;
        }
        return ans;
    }
    public static void main(String[] ar){
```

```

Scanner sc = new Scanner(System.in);
System.out.println("Enter the Size of Array");
int n = sc.nextInt();
System.out.println("Enter the Elements of Array");
int[] arr= new int[n];
for(int i=0;i<n;i++) {
    arr[i] = sc.nextInt();
}
System.out.println("Enter the Kth value");
int k = sc.nextInt();
System.out.println(helper(arr,k));
}
}

```

Output:



```

C:\Users\thara\.jdk\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the Size of Array
6
Enter the Elements of Array
4 5 8 9 2 3
Enter the Kth value
3
4

```

Time Complexity: $O(n)$

Space Complexity: $O(n)$

2. Valid Parentheses in an Expression

Given an expression string s , write a program to examine whether the pairs and the orders of “{”, “}”, “(”, “)”, “[”, “]” are correct in the given expression.

Example:

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

Output: true

Explanation: All the brackets are well-formed

Input: $s = "[()]"$

Output: false

Explanation: 1 and 4 brackets are not balanced because there is a closing ‘]’ before the closing ‘(’

Code:

```
import java.util.*;
class ValidParentheses {
    public static boolean helper(String s) {
        Stack<Character> stack = new Stack<>();
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (c == '}' || c == ']' || c == ')') {
                if (stack.isEmpty())
                    return false;
                char peek = stack.pop();
                if (c == '}' && peek != '{' || c == ']' && peek != '[' || c == ')' && peek !=
'(')
                    return false;
            } else {
                stack.push(c);
            }
        }
        return stack.isEmpty();
    }

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

        System.out.print("Enter a string of parentheses: ");
        String input = sc.nextLine();

        if (helper(input)) {
            System.out.println("The string is valid.");
        } else {
            System.out.println("The string is invalid.");
        }

        sc.close();
    }
}
```

OUTPUT:

```
C:\Users\thara\.jdk\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter a string of parentheses: [{})]
The string is invalid.

Process finished with exit code 0
```

Time Complexity: $O(n)$

Space Complexity: $O(n)$

3) 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:

$1 \leq \text{arr.size} \leq 106$

$0 \leq \text{arr}[i] \leq 109$

Code:

```

import java.util.*;
class Solution {

    public static int helper(int arr[]) {
        // code here
        int n=arr.length;
        int[] pref=new int[n];
        pref[0]=arr[0];
        int[] suf=new int[n];
        suf[n-1]=arr[n-1];
        for(int i=1;i<n;i++){
            pref[i]=pref[i-1]+arr[i];
        }
        for(int i=n-2;i>=0;i--){
            suf[i]=suf[i+1]+arr[i];
        }
        for(int i=0;i<n;i++){
            if(pref[i]==suf[i]) return i+1;
        }
        return -1;
    }

    public static void main(String[] ar){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of Array");
        int n = sc.nextInt();
        System.out.println("Enter the Elements of Array");
        int[] arr = new int[n];
        for(int i=0;i<n;i++){
            arr[i] = sc.nextInt();
        }
        System.out.println(helper(arr));
    }

}

```

Output:

```
C:\Users\thara\.jdk\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the size of Array
5
Enter the Elements of Array
1 3 5 2 2
3
```

4. 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.

Code:

```
import java.util.*;
class BinarySearch {
    public static int binarysearch(int[] arr, int k) {
        // Code Here
        int l=0;
        int r=arr.length-1;
        int mid=0;
        while(l<=r){
            mid=l+(r-l)/2;
            if(arr[mid]==k){
                return mid;
            }
            else if(k>arr[mid]){
                l=mid+1;
            }
        }
    }
}
```

```

        }
        else{
            r=mid-1;
        }
    }
    return -1;
}

public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the size of Array");
    int n = sc.nextInt();
    System.out.println("Enter the Elements of Array");
    int[] arr = new int[n];
    for(int i=0;i<n;i++){
        arr[i] = sc.nextInt();
    }
    System.out.println("Enter the Search Element");
    int k = sc.nextInt();
    System.out.println(binarysearch(arr,k));
}
}

```

Output:

```

C:\Users\thara\.jdk\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the size of Array
6
Enter the Elements of Array
1 2 3 4 5 6
Enter the Search Element
5
4

```

Time Complexity: $O(\log n)$

Space Complexity: $O(1)$

5.Union of Two 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 set of 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.

Code:

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

```
class Intersection{  
    public static int findUnion(int a[], int b[]) {  
  
        HashSet<Integer> hs=new HashSet<>();  
        for(int i:a){  
            hs.add(i);  
        }  
        for(int i:b){  
            hs.add(i);  
        }  
        return hs.size();  
    }  
    public static void main(String[] args){  
        Scanner sc = new Scanner(System.in);
```

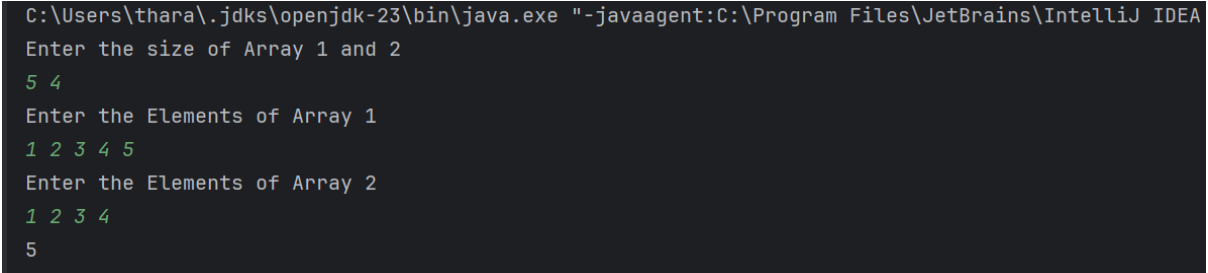


```

        System.out.println("Enter the size of Array 1 and 2");
        int[] s = new int[2];
        for(int i=0;i<2;i++){
            s[i] = sc.nextInt();
        }
        int[] arr1 = new int[s[0]];
        System.out.println("Enter the Elements of Array 1");
        for(int i=0;i<s[0];i++){
            arr1[i] = sc.nextInt();
        }
        int[] arr2 = new int[s[1]];
        System.out.println("Enter the Elements of Array 2");
        for(int i=0;i<s[1];i++){
            arr2[i] = sc.nextInt();
        }
        System.out.println(findUnion(arr1,arr2));
    }
}

```

Output:



```

C:\Users\thara\.jdk\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the size of Array 1 and 2
5 4
Enter the Elements of Array 1
1 2 3 4 5
Enter the Elements of Array 2
1 2 3 4
5

```

Time Complexity: $O(n+m)$

Space Complexity: $O(n+m)$

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.

Code:

```
import java.util.*;
class nextLargest{
    public static ArrayList<Integer> NLE(int[] arr) {
        // code here
        ArrayList<Integer> ls=new ArrayList<>();

        for(int i=0;i<arr.length;i++) {
            int max = -1;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[i] < arr[j]) {
                    max = arr[j];
                    break;
                }
            }
            ls.add(max);
        }
        return ls;
    }
    public static void main(String[] ar){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of Array");
        int n = sc.nextInt();
        int[] arr1 = new int[n];
        System.out.println("Enter the Elements of Array");
```

```
        for(int i=0;i<n;i++){
            arr1[i] = sc.nextInt();
        }

        System.out.println(NLE(arr1));

    }
}
```

Output:

```
C:\Users\thara\.jdk\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the size of Array
5
Enter the Elements of Array
6 8 4 9 11
[8, 9, 9, 11, -1]
```

Time Complexity: $O(n^2)$

Space Complexity: $O(n)$