Vasanthan_T_DSA_Practice-4

Date: 13.11.24

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 kth 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 Small(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){
            int[] arr= {7, 10, 4, 3, 20, 15};
            int  k = 3;
```

```
                System.out.println(Small(arr,k));
        }
}
```

Output:

```
D:\code\JavaCodes>javac KthSmallest.java

D:\code\JavaCodes>java KthSmallest.java
7

D:\code\JavaCodes>
```

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*:*

```
D:\code\JavaCodes>javac ParenthesisChecker.java

D:\code\JavaCodes>java ParenthesisChecker.java
true

D:\code\JavaCodes>
```

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 <= arr.size <= 106
0 <= arr[i] <= 109

Code:

```
import java.util.*;
class Solution {

    public static int equilibriumPoint(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){
                int[] arr = {1, 3, 5, 2, 2};
                System.out.println(equilibriumPoint(arr));
        }

}
```

Output:

```
D:\code\JavaCodes>javac equilibriumPoint.java

D:\code\JavaCodes>java equilibriumPoint.java
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){
                  int[] arr = {1, 2, 3, 4, 5};
                 int k = 4;
                 System.out.println(binarysearch(arr,k));
         }
}
```

Output:

```
D:\code\JavaCodes>javac BinarySearch.java

D:\code\JavaCodes>java BinarySearch.java
3
```

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 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.

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){
            int[] a = {1, 2, 3, 4, 5};
            int[] b= {1, 2, 3};
            System.out.println(findUnion(a,b));
    }
}
```

Output:

```
D:\code\JavaCodes>javac Intersection.java

D:\code\JavaCodes>java Intersection.java
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.*;
public class nextLargest{
      public static ArrayList<Integer> nextLargerElement(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){
                    int[] arr = {1, 3, 2, 4};
                    System.out.println(nextLargerElement(arr));
          }
}
```

Output:

```
D:\code\JavaCodes>javac nextLargest.java

D:\code\JavaCodes>java nextLargest.java
[3, 4, 4, -1]
```

Time Complexity:O(n^2)
Space Complexity:O(n)