DSA Practice Solutions – 13.11.24

NAME: N J Meenakshi

DEPARTMENT: CSBS

ROLL NO.: 22CB028

1. **K'th Smallest Element in Unsorted 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.
   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:

```java
import java.util.PriorityQueue;
import java.util.Scanner;

public class Kth_Smallest {
    public static int solution(int[] arr,int k){
        PriorityQueue<Integer> heapq=new PriorityQueue<>((a,b)->b-a);
        for(int i=0;i<arr.length;i++){
            heapq.offer(arr[i]);
            if(heapq.size()>k){
                heapq.poll();
            }
        }
        return heapq.peek();
    }
    public static void main(String ags[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        System.out.print("Enter k: ");
        int k=sc.nextInt();
        int[] arr=new int[n];
        for(int i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Kth smallest element: "+solution(arr, k));
        sc.close();
    }
}
```

Output:
Enter n: 6
Enter k: 3
7 10 4 3 20 15
Kth smallest element: 7

Enter n: 5
Enter k: 4
2 3 1 20 15
Kth smallest element: 15

Time Complexity: O(NlogK)

2. **Minimize the maximum difference between the heights**

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.
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.

Code:

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

public class Minimize_max_diff {
    public static int solution(int[] arr, int k){
        int n=arr.length;
        Arrays.sort(arr);
        int res=arr[n-1]-arr[0];
        for(int i=1;i<n;i++){
            if(arr[i]-k<0) continue;
            int minH=Math.min(arr[0]+k,arr[i]-k);
            int maxH=Math.max(arr[n-1]-k,arr[i-1]+k);
            res=Math.min(res,maxH-minH);
        }
        return res;
    }
    public static void main(String ags[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        System.out.print("Enter k: ");
        int k=sc.nextInt();
```

```
        int[] arr=new int[n];
        for(int i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Updated Minimum Difference: "+solution(arr, k));
        sc.close();
    }
}
```

Output:
Enter n: 4
Enter k: 2
1 5 8 10
Updated Minimum Difference: 5

Enter n: 5
Enter k: 3
3 9 12 16 20
Updated Minimum Difference: 11

Time Complexity: O(NlogN)

3. **Valid Parentheses in an Expression**
Given a string s, composed of different combinations of '(' , ')', '{', '}', '[', ']', verify the validity of the arrangement.
An input string is valid if:
    - Open brackets must be closed by the same type of brackets.
    - Open brackets must be closed in the correct order.

Input:
S = ()[]{}
Output: 1
Explanation: The arrangement is valid, as both the conditions are followed here.
Input:
S = ())({}
Output: 0
Explanation: Arrangement is not valid, as for the bold closing bracket in ())({}, there is no opening bracket of similar kind, before it.

Code:

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

public class Valid_Parentheses {
    public static boolean solution(String s){
        Stack<Character> stk=new Stack<>();
        for(int i=0;i<s.length();i++){
            if(s.charAt(i)=='(' || s.charAt(i)=='{' || s.charAt(i)=='[')
stk.push(s.charAt(i));
            else{
                if(!stk.isEmpty() && ((stk.peek()=='(' && s.charAt(i)==')') ||
(stk.peek()=='{' && s.charAt(i)=='}') || (stk.peek()=='[' && s.charAt(i)==']'))){
```

```
                    stk.pop();
                }
                else{
                    return false;
                }
            }
        }
        return stk.isEmpty();
    }
    public static void main(String ags[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter s: ");
        String s=sc.next();
        System.out.print("Valid Parentheses: "+solution(s));
        sc.close();
    }
}
```

Output:
Enter s: ()[]{}
Valid Parentheses: true

Enter s: ())({}
Valid Parentheses: false

Time Complexity: O(N)

4. **Equilibrium index of an array**
   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.
   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.

   Code:

```
import java.util.Scanner;

public class Equilibrium_Index {
    public static int solution(int[] arr){
        int n=arr.length;
        int left=0,right=0,pivot=0;
        for(int i=1;i<n;i++){
            right+=arr[i];
```

```
        }
        while(pivot<n-1 && right!=left){
            pivot++;
            right-=arr[pivot];
            left+=arr[pivot-1];
        }
        if (left==right) return pivot+1;
        return -1;
    }
    public static void main(String ags[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        int[] arr=new int[n];
        for(int i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Equilibrium index: "+solution(arr));
        sc.close();
    }
}
```

Output:

Enter n: 5
1 3 5 2 2
Equilibrium index: 3

Enter n: 1
1
Equilibrium index: 1

Enter n: 3
1 2 3
Equilibrium index: -1

Time Complexity: O(N)

5. **Binary Search**

Code:

```
import java.util.Scanner;

public class Binary_Search {
    public static boolean solution(int[] arr, int val){
        int low=0,high=arr.length;
        while (low<high) {
            int mid=(low+high)/2;
            if(arr[mid]==val) return true;
            if(arr[mid]>val){
                high=mid-1;
            }
            else{
```

```
                low=mid+1;
            }
        }
        return false;
    }
    public static void main(String ags[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        System.out.print("Enter element: ");
        int ele=sc.nextInt();
        int[] arr=new int[n];
        for(int i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Element present: "+solution(arr, ele));
        sc.close();
    }
}
```

Output:
Enter n: 5
Enter element: 15
5 10 15 17 65
Element present: true

Enter n: 5
Enter element: 26
5 10 15 17 65
Element present: false

Time Complexity: O(NlogN)

6. **Next Greater Element (NGE)**
   For every element in given Array Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.
   Input: arr[] = [4 , 5 , 2 , 25]
   Output: 4 –> 5 5 –> 25 2 –> 25 25 –> -1
   Explanation: Except 25 every element has an element greater than them present on the right side

   Code:

```
import java.util.*;
public class Next_Greatest_Element {
    public static int[] solution(int[] nums){
        int[] res=new int[nums.length];
        int n=nums.length;
        Arrays.fill(res,-1);
        Deque<Integer> s=new ArrayDeque<>();
        for(int i=0;i<2*nums.length;i++){
            while(!s.isEmpty() && nums[i%n]>nums[s.peek()]){
                res[s.pop()]=nums[i%n];
```

```
            }
            s.push(i%n);
        }
        return res;
    }
    public static void main(String args[]){
        Scanner sc= new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        int[] nums=new int[n];
        System.out.print("Enter array: ");
        for(int i=0;i<n;i++){
            nums[i]=sc.nextInt();
        }
        int[] res=solution(nums);
        System.out.print("The result is: ");
        for (int i=0;i<n;i++){
            System.out.print(res[i]+" ");
        }
        sc.close();
    }
}
```

Output:
Enter n: 4
Enter array: 4 5 2 25
The result is: 5 25 25 -1

Enter n: 4
Enter array: 13 7 6 12
The result is: -1 12 12 13

Time Complexity: O(N)

7. **Union of two arrays with duplicate elements**
   We are given two sorted arrays a[] and b[] and the task is to return union of both the arrays in sorted order. Union of two arrays is an array having all distinct elements that are present in either array. The input arrays may contain duplicates.

   Input: a[] = {1, 1, 2, 2, 2, 4}, b[] = {2, 2, 4, 4}
   Output: {1, 2, 4}
   Explanation: 1, 2 and 4 are the distinct elements present in either array.
   Input: a[] = {3, 5, 10, 10, 10, 15, 15, 20}, b[] = {5, 10, 10, 15, 30}
   Output: {3, 5, 10, 15, 20, 30}
   Explanation: 3, 5, 10, 15, 20 and 30 are the distinct elements present in either array.

   Code:

```
import java.util.ArrayList;
import java.util.Scanner;

public class Union_of_two_arrays {
    public static ArrayList<Integer> solution(int arr1[], int arr2[], int n, int m){
```

```
        int i=0,j=0;
        ArrayList<Integer> res=new ArrayList<>();
        while(i<n && j<m){
            if(arr1[i]<=arr2[j]){
                if(res.size()==0 || res.get(res.size()-1)!=arr1[i]){
                    res.add(arr1[i]);
                }
                i++;
            }
            else{
                if(res.size()==0 || res.get(res.size()-1)!=arr2[j]){
                    res.add(arr2[j]);
                }
                j++;
            }
        }
            while (i<n) {
                if(res.get(res.size()-1)!=arr1[i]){
                    res.add(arr1[i]);
                }
                i++;
            }
            while (j<m) {
                if(res.get(res.size()-1)!=arr2[j]){
                    res.add(arr2[j]);
                }
                j++;
            }
        return res;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        System.out.print("Enter m: ");
        int m=sc.nextInt();
        int[] arr1=new int[n];
        for(int i=0;i<n;i++){
            arr1[i]=sc.nextInt();
        }
        int[] arr2=new int[m];
        for(int i=0;i<m;i++){
            arr2[i]=sc.nextInt();
        }
        System.out.print("Union of arrays is: "+solution(arr1, arr2, n, m));
        sc.close();
    }
}
```

Output:

Enter n: 6

Enter m: 4

1 1 2 2 2 4

2 2 4 4
Union of arrays is: [1, 2, 4]

Enter n: 8
Enter m: 5
3 5 10 10 10 15 15 20
5 10 10 15 30
Union of arrays is: [3, 5, 10, 15, 20, 30]

Time Complexity: O(M+N)