

DSA PRACTICE PROBLEMS – SET 4

Madhulekha R – AIML - [13/11/2024]

1. **Question:** K'th smallest element in unsorted array

Code:

```
import java.util.Arrays;

public class Main3{
    public static void main(String[] args){
        int[] arr = { 19, 3, 5, 30, 60 };
        int n = arr.length;
        int k = 2;
        System.out.println(ksmallest(arr, n, k));
    }

    static int ksmallest(int[] arr, int n, int k){
        int maxelement = arr[0];
        for (int i = 1; i < n; i++) {
            if (arr[i] > maxelement) {
                maxelement = arr[i];
            }
        }

        int[] freq = new int[maxelement + 1];
        Arrays.fill(freq, 0);
        for (int i = 0; i < n; i++) {
            freq[arr[i]]++;
        }

        int count = 0;
        for (int i = 0; i <= maxelement; i++) {
            if (freq[i] != 0) {
                count += freq[i];
                if (count >= k) {
                    return i;
                }
            }
        }
        return -1;
    }
}
```

Output:

```
C:\Users\HP\Documents>javac Main3.java

C:\Users\HP\Documents>java Main3
5
```

Time Complexity: $O(n + \text{maxelement})$

Space Complexity: $O(\text{maxelement})$

2. Question: Minimize the heights

Given a positive integer **k** and an array **arr[]** denoting heights of towers, you have to modify the height of each tower either by increasing or decreasing them by **k** only **once**. Find out what could be the **minimum difference** of the height of **shortest** and **longest** towers after you have modified each tower.

Code:

```
import java.util.*;

public class Main3{
    public static void main(String[] args){
        int[] arr = {1, 5, 8, 10};
        int n=4;
        int k=2;
        System.out.println(func(arr,n,k));
    }
    public static int func(int[] arr, int n, int k) {
        if (arr == null || n<=0){
            return -1;
        }

        Arrays.sort(arr);
        int min = 0,max = 0,res = 0;
        res = arr[n-1] - arr[0];
        for (int i = 1;i<n;++i){
            if (arr[i]>=k){
                max = Math.max(arr[i-1]+k,arr[n-1]-k);
                min =Math.min(arr[i]-k,arr[0]+k);
                res = Math.min(res,max-min);
            }
            else {
                continue;
            }
        }
    }
}
```

```

    }
    return res;
}
}

```

Output:

```

C:\Users\HP\Documents>javac Main3.java

C:\Users\HP\Documents>java Main3
5

```

Time Complexity: $O(n \log n)$

Space Complexity: $O(1)$

3. **Question:** 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.

Code:

```

import java.util.Stack;
import java.util.*;

class Main3{
    public static void main(String[] args){
        String s = "()[]{}";
        System.out.println(isValid(s));
    }

    static boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
        for (int i = 0; i < s.length(); i++) {
            char cur = s.charAt(i);
            if (!stack.isEmpty()) {
                char last = stack.peek();
                if (isPair(last, cur)) {
                    stack.pop();
                    continue;
                }
            }
        }
    }
}

```

```

        stack.push(cur);
    }
    return stack.isEmpty();
}

private static boolean isPair(char last, char cur) {
    return (last == '(' && cur == ')') || (last == '{' && cur == '}') || (last == '[' && cur == ']');
}
}

```

Output:

```

C:\Users\HP\Documents>javac Main3.java

C:\Users\HP\Documents>java Main3
true

```

Time Complexity: $O(n)$

Space Complexity: $O(n)$

4. **Question:** 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.

Code:

```

import java.util.*;

public class Main3{
    public static void main(String[] args){
        long[] arr = { -7, 1, 5, 2, -4, 3, 0 };
        System.out.println(equilibriumPoint(arr));
    }

    public static int equilibriumPoint(long[] arr){
        int n = arr.length;
        long leftsum, rightsum;
        for (int i = 0; i < n; ++i) {
            leftsum = 0;
            for (int j = 0; j < i; j++){
                leftsum += arr[j];
            }

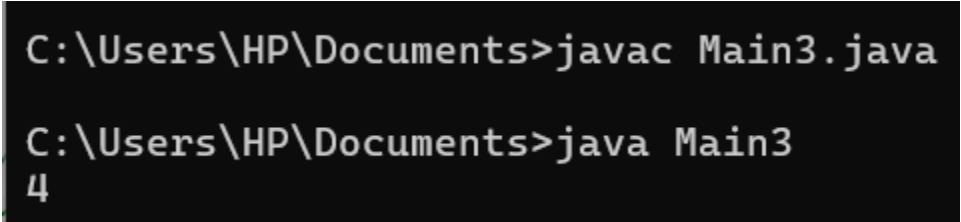
```

```

        rightsum = 0;
        for (int j = i + 1; j < n; j++){
            rightsum += arr[j];
        }
        if (leftsum == rightsum)
            return i + 1;
    }
    return -1;
}
}

```

Output:



```

C:\Users\HP\Documents>javac Main3.java

C:\Users\HP\Documents>java Main3
4

```

Time Complexity: $O(n^2)$

Space Complexity: $O(1)$

5. Question: Binary Search

Code:

```

import java.util.*;

public class Main {
    public static void main(String[] args) {
        int[] arr = {1,2,3,4,5,6,7};
        int target=1;
        System.out.println(binarysearch(arr,target));
    }

    static int binarysearch(int[] arr, int target){
        int start=0;
        int end=arr.length-1;

        while (start<=end){
            int mid = start+(end-start)/2;
            if (target<arr[mid]){
                end=mid-1;
            }
        }
    }
}

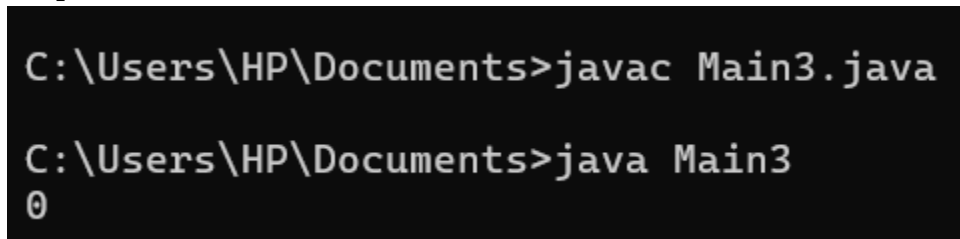
```

```

        else if (target>arr[mid]){
            start=mid+1;
        }
        else{
            return mid;
        }
    }
    return -1;
}
}

```

Output:



```

C:\Users\HP\Documents>javac Main3.java

C:\Users\HP\Documents>java Main3
0

```

Time Complexity: $O(\log n)$

Space Complexity: $O(1)$

6. **Question:** Next Greater Element (NGE) for every element in given Array

Code:

```

import java.util.Stack;

public class Main{
    public static void main(String[] args) {
        int[] arr1 = {4, 5, 2, 25};
        int[] arr2 = {13, 7, 6, 12};
        printNGE(arr1);
        System.out.println();
        printNGE(arr2);
    }
    public static void printNGE(int[] arr) {
        Stack<Integer> stack = new Stack<>();
        for (int i = 0; i < arr.length; i++) {
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }
            if (!stack.isEmpty()) {
                System.out.println(arr[i] + " --> " + stack.peek());
            }
        }
    }
}

```

```

    } else {
        System.out.println(arr[i] + " --> -1");
    }
    stack.push(arr[i]);
}
}
}

```

Output:

```

C:\Users\HP\Documents>javac Main.java

C:\Users\HP\Documents>java Main
4 --> -1
5 --> -1
2 --> 5
25 --> -1

13 --> -1
7 --> 13
6 --> 7
12 --> 13

```

Time Complexity: $O(n)$

Space Complexity: $O(n)$

7. Question: Union of two arrays with duplicates

Code:

```

import java.util.*;

public class Main3{
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 2, 3, 4};
        int[] arr2 = {2, 3, 4, 4, 5};
        int[] result = union(arr1, arr2);
        System.out.println(Arrays.toString(result));
    }

    static int[] union(int[] arr1, int[] arr2) {
        List<Integer> unionList = new ArrayList<>();
        for (int num : arr1) {
            unionList.add(num);
        }
        for (int num : arr2) {
            unionList.add(num);
        }
        int[] result = new int[unionList.size()];
    }
}

```

```
        for (int i = 0; i < unionList.size(); i++) {  
            result[i] = unionList.get(i);  
        }  
        return result;  
    }  
}
```

Output:

```
C:\Users\HP\Documents>javac Main3.java  
  
C:\Users\HP\Documents>java Main3  
[1, 2, 2, 3, 4, 2, 3, 4, 4, 5]
```

Time Complexity: $O(m+n)$

Space Complexity: $O(m+n)$