

13-11-2024

Java DSA Problems (SET - 4)

1.K'th smallest element in unsorted array

Solution:

```
import java.util.*;

class KthSmall {

    static int partition(int[] arr, int l, int r) {
        int pivot = arr[r];
        int i = l;
        for (int j = l; j <= r - 1; j++) {
            if (arr[j] <= pivot) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
                i++;
            }
        }
        int temp = arr[i];
        arr[i] = arr[r];
        arr[r] = temp;
        return i;
    }

    static int kthSmallestElement(int[] arr, int l, int r, int k) {
        if (k > 0 && k <= r - l + 1) {
            int index = partition(arr, l, r);
            if (index - l == k - 1) {
                return arr[index];
            }
            if (index - l > k - 1) {
                return kthSmallestElement(arr, l, index - 1, k);
            }
            return kthSmallestElement(arr, index + 1, r, k - index + l - 1);
        }
    }
}
```

```

        return Integer.MAX_VALUE;
    }

    public static void main(String[] args) {
        int arr[] = {12, 3, 5, 7, 4, 19, 26};
        int k = 1;

        System.out.println("Kth Smallest element is " + kthSmallestElement(arr, 0, arr.length - 1, k));
    }
}

```

Output:

```

D:\OOPS\13-11-2024 practice.java>javac KthSmall.java

D:\OOPS\13-11-2024 practice.java>java KthSmall
Kth Smallest element is 3

```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

2. 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.

Solution:

```

import java.util.*;

public class MinimizeHeight {
    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, max, res;
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);
    }
}
return res;
}
}

```

Output:

```

D:\00PS\13-11-2024 practice.java>javac MinimizeHeight.java
D:\00PS\13-11-2024 practice.java>java MinimizeHeight
5

```

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

Space Complexity: $O(1)$

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.

Solution:

```

import java.util.*;
class BalancedParanthesis {
    public static boolean isValid(String s) {
        Stack<Character> st = new Stack<Character>();
        for (char it : s.toCharArray()) {

```

```

        if (it == '(' || it == '[' || it == '{')
            st.push(it);
        else {
            if (st.isEmpty()) return false;
            char ch = st.pop();
            if ((it == ')' && ch == '(') || (it == ']' && ch == '[') || (it == '}' && ch == '{')) continue;
            else return false;
        }
    }
    return st.isEmpty();
}

public static void main(String[] args) {
    String str1 = "((()))()";
    if (isValid(str1)) {
        System.out.println("Balanced");
    } else {
        System.out.println("Not Balanced");
    }
    String str2 = "()()()";
    if (isValid(str2)) {
        System.out.println("Balanced");
    } else {
        System.out.println("Not Balanced");
    }
}
}

```

Output:

```

D:\00PS\09-11-2024 practice java>javac BalancedParanthesis.java
D:\00PS\09-11-2024 practice java>java BalancedParanthesis
Balanced
Not Balanced

```

Time Complexity: $O(n)$

Space 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.

Solution:

```
import java.util.*;

class EquilibriumPoint {

    public static int equilibriumPoint(long arr[])
    {
        int n = arr.length;
        int left = 0, pivot = 0, right = 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];
        }
        return (left == right) ? pivot + 1 : -1;
    }

    public static void main(String[] args)
    {
        long[] arr = { 1, 7, 3, 6, 5, 6 };
        int result = equilibriumPoint(arr);
        System.out.println(result);
    }
}
```

Output:

```
D:\OOPS\13-11-2024 practice.java>javac EquilibriumPoint.java
D:\OOPS\13-11-2024 practice.java>java EquilibriumPoint
4
```

Time Complexity: $O(n)$

Space Complexity: $O(1)$

5. Binary Search

Solution:

```
import java.util.*;

public class BinarySearch {

    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:

```
D:\OOPS\13-11-2024 practice.java>javac BinarySearch.java

D:\OOPS\13-11-2024 practice.java>java BinarySearch

0
```

Time Complexity: $O(\log n)$

Space Complexity: $O(1)$

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

Solution:

```
import java.util.Stack;

public class NextGreaterElement {

    public static void main(String[] args) {

        int[] arr1 = {10, 3, 8, 6};

        int[] arr2 = {2, 7, 1, 9};

        System.out.println("Next Greater Element for arr1:");

        printNGE(arr1);

        System.out.println("Next Greater Element for arr2:");

        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");

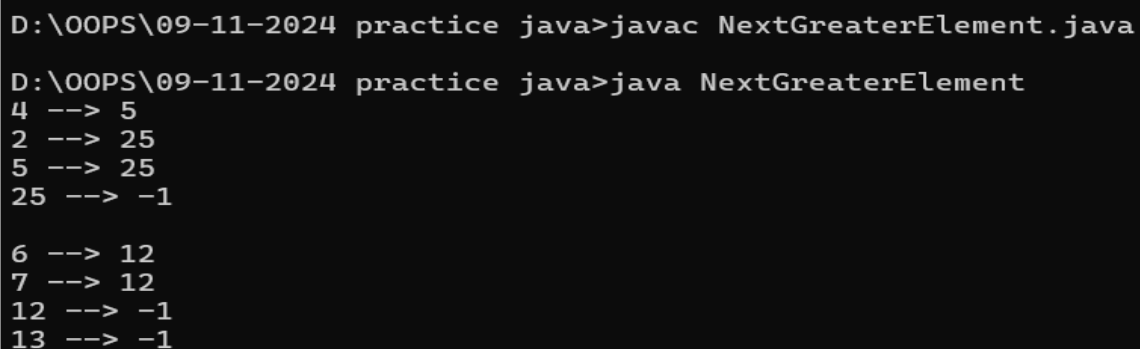
            stack.push(arr[i]);

        }

    }

}
```

Output:



```
D:\OOPS\09-11-2024 practice java>javac NextGreaterElement.java
D:\OOPS\09-11-2024 practice java>java NextGreaterElement
4 --> 5
2 --> 25
5 --> 25
25 --> -1

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

Time Complexity: $O(n)$

Space Complexity: $O(n)$

7. Union of two arrays with duplicates

Solution:

```
import java.util.*;

class UnionOfTwoArray {

    static ArrayList<Integer> findUnion(int[] a, int[] b) {

        HashSet<Integer> st = new HashSet<>();

        for (int num : a)

            st.add(num);

        for (int num : b)

            st.add(num);

        ArrayList<Integer> res = new ArrayList<> ();

        for(int it: st)

            res.add(it);

        return res;

    }

    public static void main(String[] args) {

        int[] a = {1, 2, 3, 2, 1};

        int[] b = {3, 2, 2, 3, 3, 2};

        ArrayList<Integer> res = findUnion(a, b);

        for (int num : res)

            System.out.print(num + " ");

    }

}
```

Output:

```
D:\00PS\13-11-2024 practice.java>javac UnionOfTwoArray.java
D:\00PS\13-11-2024 practice.java>java UnionOfTwoArray
1 2 3
```

Time Complexity: $O(n+m)$

Space Complexity: $O(n+m)$