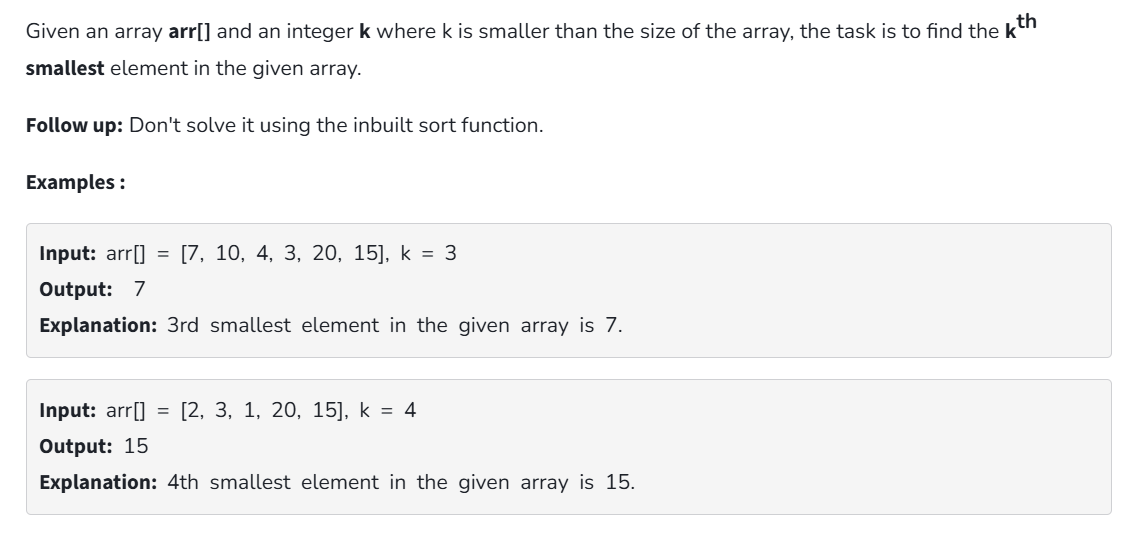
**DSA Practice Test – 4** 13th Nov 2024

**1. Kth Smallest Element**

****

**Code:**

public class Solution {

private static int kthSmallest(int[] arr, int low, int high) {

int pivot = arr[high];

int i = low;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

i++;

}

}

int temp = arr[i];

arr[i] = arr[high];

arr[high] = temp;

return i;

}

private static int quickselect(int[] arr, int low, int high, int k) {

if (low <= high) {

int pivotIndex = kthSmallest(arr, low, high);

if (pivotIndex == k) return arr[pivotIndex];

else if (pivotIndex > k) return quickselect(arr, low, pivotIndex - 1, k);

else return quickselect(arr, pivotIndex + 1, high, k);

}

return Integer.MAX\_VALUE;

}

public static int findKthSmallest(int[] arr, int k) {

if (k > 0 && k <= arr.length) return quickselect(arr, 0, arr.length - 1, k - 1);

return Integer.MAX\_VALUE;

}

public static void main(String[] args) {

int[] arr = {7, 10, 4, 3, 20, 15};

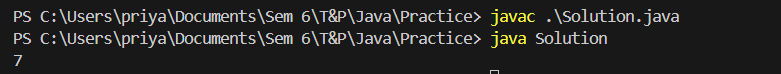
int k = 3;

System.out.println(findKthSmallest(arr, k));

}

}

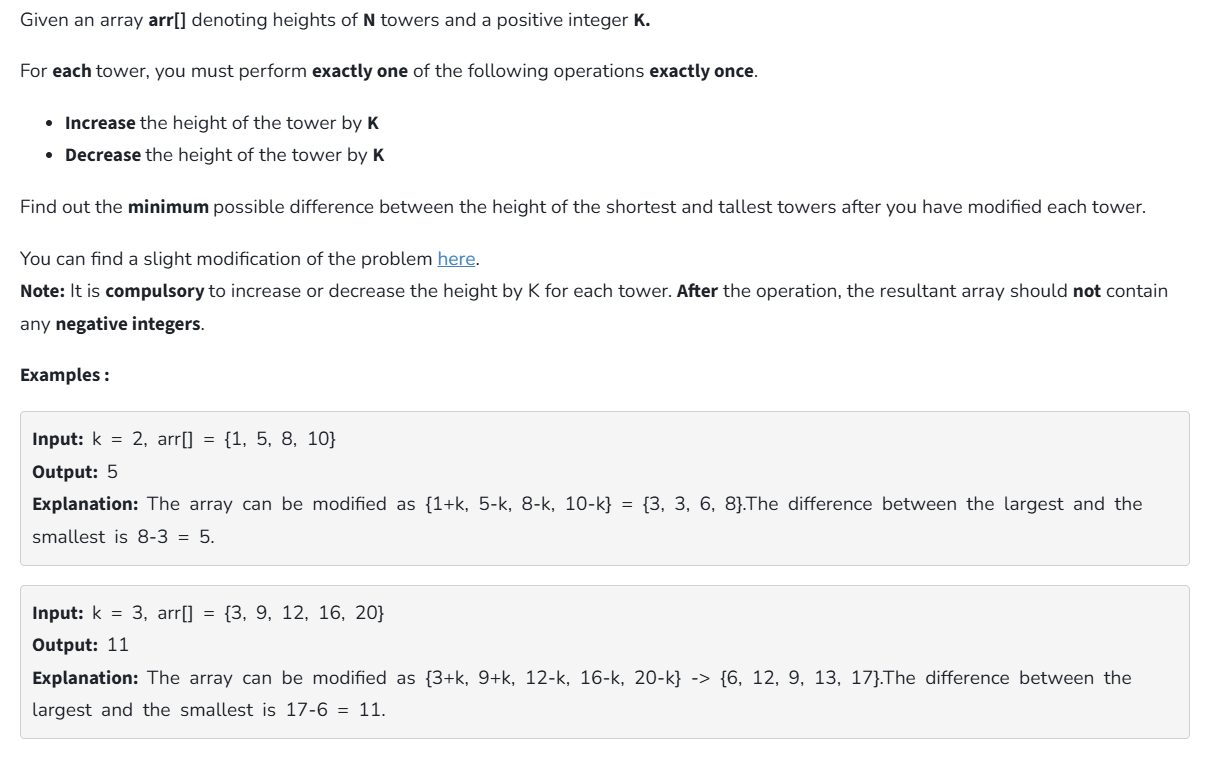
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**2. Minimize the Heights II**



**Code:**

import java.util.Arrays;

public class Solution {

static int getMinDiff(int[] arr, int k) {

int n = arr.length;

if (n == 1) return 0;

Arrays.sort(arr);

int result = arr[n - 1] - arr[0];

for (int i = 1; i < n ; i++) {

if(arr[i]-k<0) continue;

int s = Math.min(arr[0] + k,arr[i]-k);

int l = Math.max(arr[i-1]+k,arr[n - 1] - k);

result = Math.min(result, l-s);

}

return result;

}

public static void main(String[] args) {

int[] arr = {3,9,12,16,20};

int k = 3;

System.out.println(getMinDiff(arr, k));

}

}

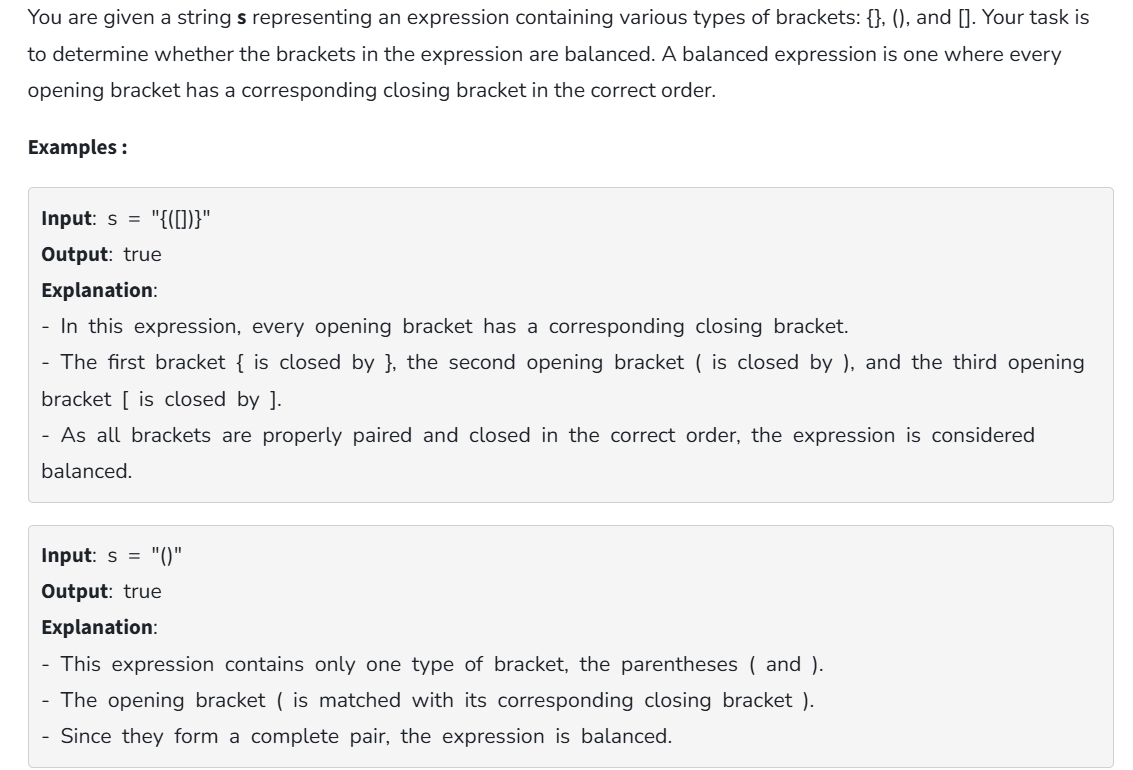
**Output:**



**Time Complexity:** O (n log n)

**Space Complexity:** O (1)

**3. Parentheses Checker:**



**Code:**

import java.util.Stack;

import java.util.Map;

import java.util.HashMap;

public class Solution {

public boolean isValid(String s) {

Stack<Character> stack = new Stack<>();

Map<Character, Character> map = new HashMap<>();

map.put(')', '(');

map.put(']', '[');

map.put('}', '{');

for (char ch : s.toCharArray()) {

if (map.containsValue(ch)) {

stack.push(ch);

} else if (map.containsKey(ch)) {

if (!stack.isEmpty() && stack.peek() == map.get(ch)) {

stack.pop();

} else {

return false;

}

}

}

return stack.isEmpty();

}

public static void main(String[] args) {

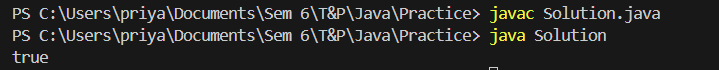
Solution solution = new Solution();

System.out.println(solution.isValid("([{}])"));

}

}

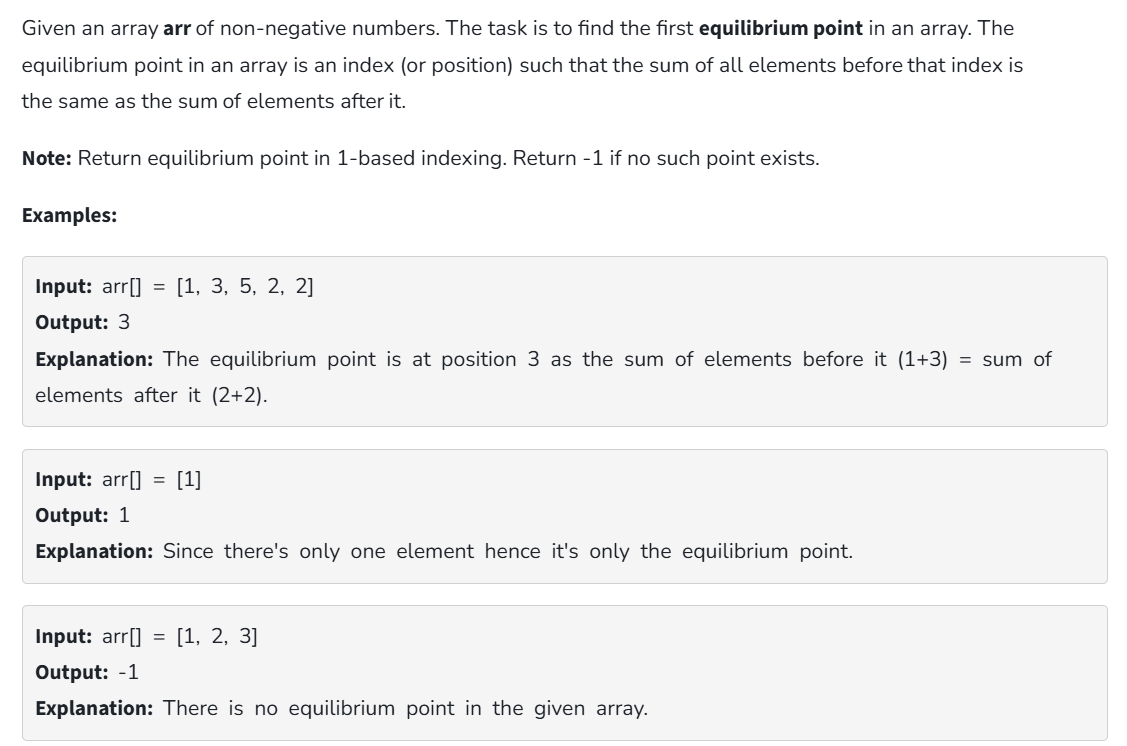
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (n)

**4. Equilibrium point**

****

**Code:**

class Solution {

public static int equilibriumPoint(int arr[]) {

int t = 0;

for (int i = 0; i < arr.length; i++) {

t += arr[i];

}

if (arr.length == 1) return 1;

int l = 0;

for (int i = 0; i < arr.length; i++) {

int r = t - arr[i] - l;

if (l == r) {

return i + 1;

}

l += arr[i];

}

return -1;

}

public static void main(String[] args) {

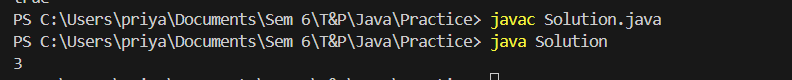
int[] arr = {1, 3, 5, 2, 2};

System.out.println(equilibriumPoint(arr));

}

}

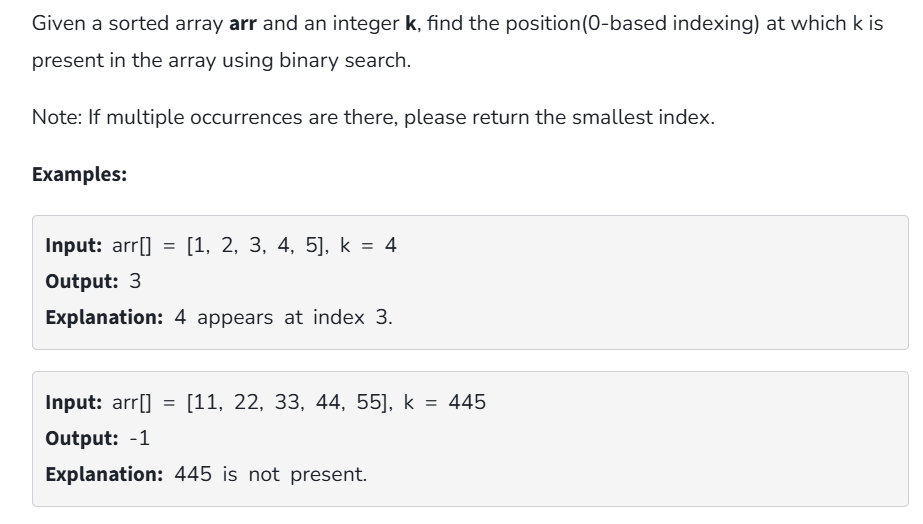
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**5. Binary Search**



**Code:**

class Solution {

public static int binarysearch(int[] arr, int k) {

int n=arr.length;

int l=0;

int r=n-1;

while(l<=r){

int m=(l+r)/2;

if(arr[m]==k) return m;

else if(arr[m]>k) r=m-1;

else l=m+1;

}

return -1;

}

public static void main (String ar[]){

int[] arr={

11,22,33,44,55

};

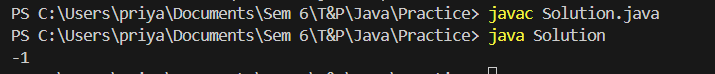
int k=445;

System.out.print(binarysearch(arr, k));

}

}

**Output:**



**Time Complexity:** O (log n)

**Space complexity:** O (1)

**18. 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

Input: arr[] = [ 13 , 7, 6 , 12 ]

Output:

13 -> -1

7 -> 12

6 -> 12

12 -> -1

Explanation: 13 and 12 don‟t have any element greater than them present on the right side

**Code:**

import java.util.Stack;

import java.util.HashMap;

public class NextGreaterElement {

public static void printNextGreaterElement(int[] arr) {

Stack<Integer> stack = new Stack<>();

HashMap<Integer, Integer> ngeMap = new HashMap<>();

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

for (int i = arr.length - 1; i >= 0; i--) {

int current = arr[i];

while (!stack.isEmpty() && stack.peek() <= current) {

stack.pop();

}

if (stack.isEmpty()) {

ngeMap.put(current, -1);

} else {

ngeMap.put(current, stack.peek());

}

stack.push(current);

}

for (int num : arr) {

res.add(ngeMap.get(num));

}

return res;

}

public static void main(String[] args) {

int[] arr1 = {4, 5, 2, 25};

System.out.println("Output for arr1:");

printNextGreaterElement(arr1);

int[] arr2 = {13, 7, 6, 12};

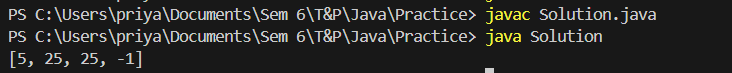
System.out.println("\nOutput for arr2:");

printNextGreaterElement(arr2);

}

}

**Output:**



**Time Complexity: O (n)**