**DSA PRACTICE DAY 4 DHANUSHSHRUTHI S T AI&DS**

1. **Kth Smallest Number:**

**CODE:**

import java.util.Arrays;

public class KthSmallestNumber {

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

Arrays.sort(arr);

return arr[k - 1];

}

public static void main(String[] args) {

int[] arr = {12, 3, 5, 7, 19};

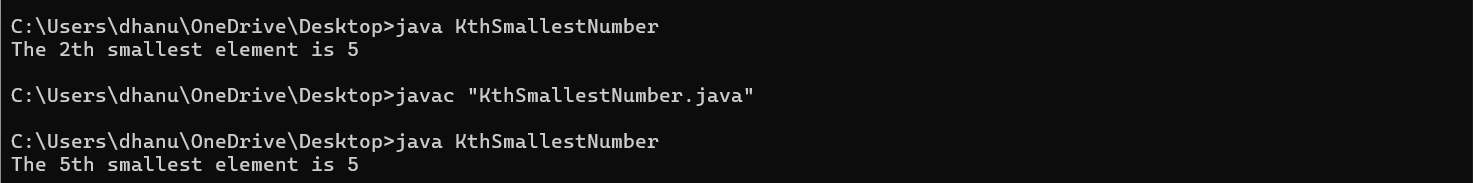
int k = 2;

System.out.println("The " + k + "th smallest element is " + kthSmallest(arr, k));

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(nlogn)**

1. **Minimize the Height 2’s**

**CODE:**

import java.util.Arrays;

public class MinimizeHeight {

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

int n = arr.length;

if (n == 1) return 0;

Arrays.sort(arr);

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

int small = arr[0] + k;

int large = arr[n - 1] - k;

if (small > large) {

int temp = small;

small = large;

large = temp;

}

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

int subtract = arr[i] - k;

int add = arr[i] + k;

if (subtract >= small || add <= large) continue;

if (large - subtract <= add - small) {

small = subtract;

} else {

large = add;

}

}

return Math.min(result, large - small);

}

public static void main(String[] args) {

int[] arr = {1, 15, 10};

int k = 6;

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

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(n log n).**

1. **Parenthesis Checker:**

**CODE:**

import java.util.Stack;

public class ParenthesisChecker {

public static boolean isParenthesisBalanced(String expression) {

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

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

char ch = expression.charAt(i);

if (ch == '(' || ch == '[' || ch == '{') {

stack.push(ch);

} else if (ch == ')' || ch == ']' || ch == '}') {

if (stack.isEmpty()) return false;

char top = stack.pop();

if ((ch == ')' && top != '(') || (ch == ']' && top != '[') || (ch == '}' && top != '{')) {

return false;

}

}

}

return stack.isEmpty();

}

public static void main(String[] args) {

String expression = "{[]}";

if (isParenthesisBalanced(expression)) {

System.out.println("The parentheses are balanced.");

} else {

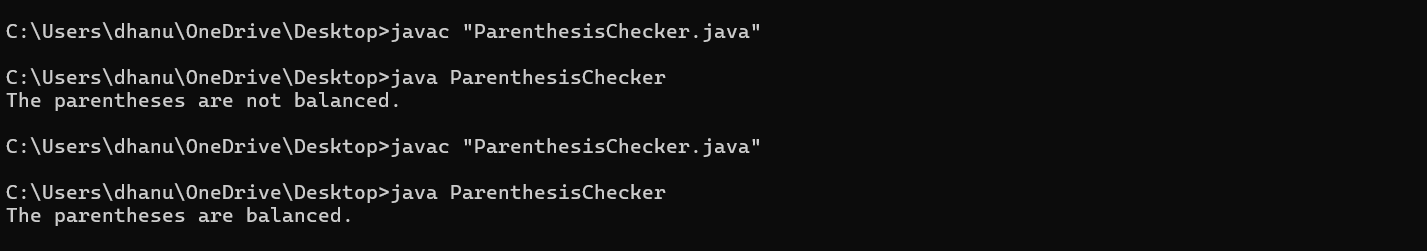
System.out.println("The parentheses are not balanced.");

}

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(n)**

1. **Equilibrium Point:**

**CODE:**

public class EquilibriumPoint {

public static int findEquilibrium(int[] arr) {

int totalSum = 0;

int leftSum = 0;

for (int num : arr) {

totalSum += num;

}

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

totalSum -= arr[i];

if (leftSum == totalSum) {

return i + 1;

}

leftSum += arr[i];

}

return -1;

}

public static void main(String[] args) {

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

int result = findEquilibrium(arr);

if (result == -1) {

System.out.println("No equilibrium point");

} else {

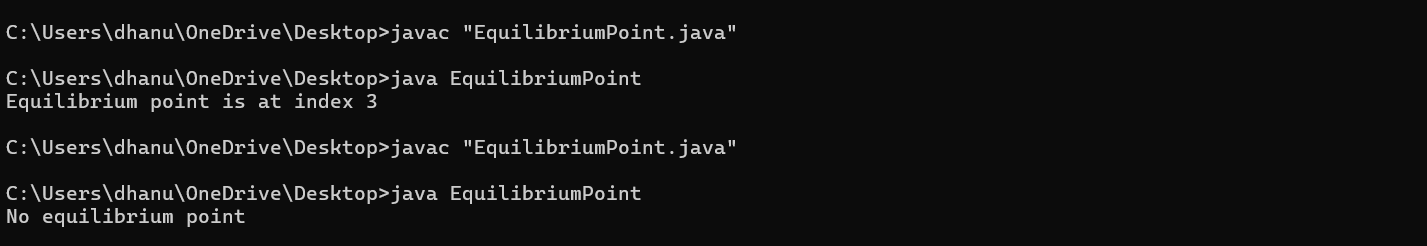
System.out.println("Equilibrium point is at index " + result);

}

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(n)**

1. **Binary Search:**

**CODE:**

public class BinarySearch {

public static int binarySearch(int[] arr, int target) {

int low = 0, high = arr.length - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == target) {

return mid;

} else if (arr[mid] < target) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

int[] arr = {1, 3, 5, 7, 9, 11};

int target = 7;

int result = binarySearch(arr, target);

if (result == -1) {

System.out.println("Element not found.");

} else {

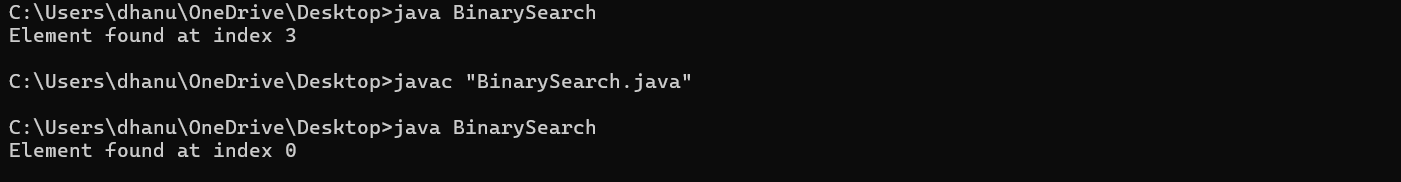
System.out.println("Element found at index " + result);

}

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(log n)**

1. **Next Greater Element:**

**CODE:**

import java.util.Stack;

public class NextGreaterElement {

public static void nextGreaterElement(int[] arr) {

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

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

while (!stack.isEmpty() && arr[stack.peek()] < arr[i]) {

int index = stack.pop();

System.out.println("Next Greater Element for " + arr[index] + " is " + arr[i]);

}

stack.push(i);

}

while (!stack.isEmpty()) {

int index = stack.pop();

System.out.println("Next Greater Element for " + arr[index] + " is -1");

}

}

public static void main(String[] args) {

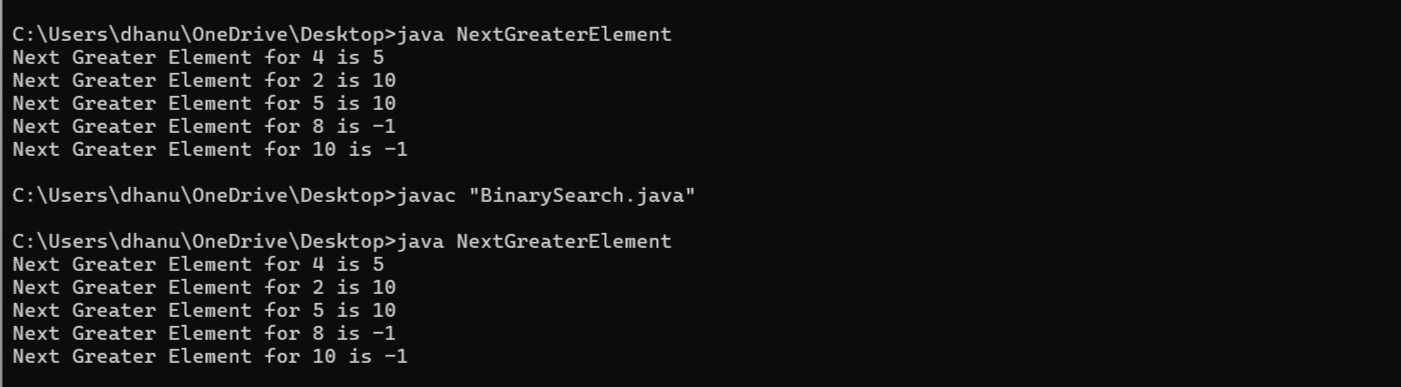
int[] arr = {4, 5, 2, 10, 8};

nextGreaterElement(arr);

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(n)**

1. **Union Of Two Arrays:**

**CODE:**

import java.util.HashSet;

public class UnionOfTwoArrays {

public static int findUnion(int[] arr1, int[] arr2) {

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

for (int num : arr1) {

set.add(num);

}

for (int num : arr2) {

set.add(num);

}

return set.size();

}

public static void main(String[] args) {

int[] arr1 = {1, 2, 2, 1};

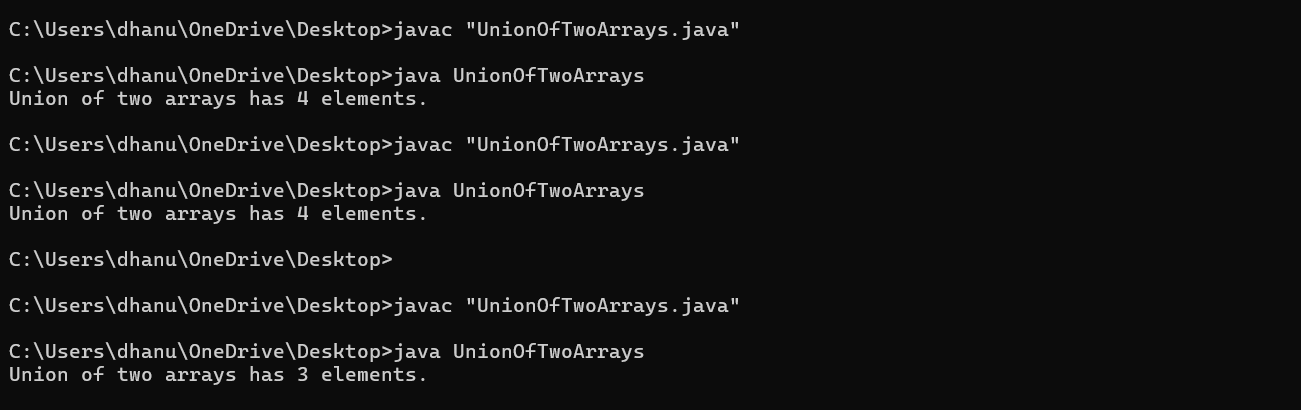
int[] arr2 = {2, 3, 3, 4};

System.out.println("Union of two arrays has " + findUnion(arr1, arr2) + " elements.");

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(n+m)**