**PRACTICE SET – 4**

Meena Rhoshini C

Nov 13, 24

1. **Kth Smallest**

import java.util.\*;

class KthSmallest {

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

int maxElement = Arrays.stream(arr).max().getAsInt();

int[] count = new int[maxElement + 1];

for (int num : arr) {

count[num]++;

}

int countNum = 0;

for (int i = 0; i <= maxElement; i++) {

countNum += count[i];

if (countNum >= k) {

return i;

}

}

return -1;

}

public static void main(String[] args) {

int[] arr1 = {7, 10, 4, 3, 20, 15};

int[] arr2 = {2, 3, 1, 20, 15};

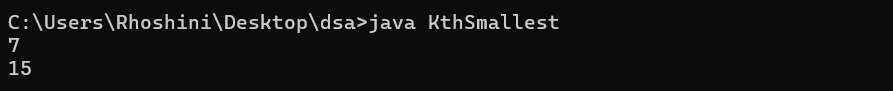
System.out.println(findKthSmallest(arr1, 3));

System.out.println(findKthSmallest(arr2, 4));

}

}

OUTPUT:



Time complexity: O(n + max\_element)

Space complexity: O(max\_element)

1. **Minimize the Heights II**

import java.util.\*;

public class MinimizeTheHeightsII {

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

Arrays.sort(arr);

int n = arr.length;

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

int small = arr[0] + k;

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

int result = diff;

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

int min = Math.min(small, arr[i] - k);

int max = Math.max(big, arr[i] + k);

if (min >= 0) {

result = Math.min(result, max - min);

}

}

return result;

}

public static void main(String[] args) {

int[] arr1 = {1, 5, 8, 10};

int k1 = 2;

System.out.println(getMinDifference(k1, arr1));

int[] arr2 = {3, 9, 12, 16, 20};

int k2 = 3;

System.out.println(getMinDifference(k2, arr2));

int[] arr3 = {1, 3, 6, 9, 12};

int k3 = 4;

System.out.println(getMinDifference(k3, arr3));

int[] arr4 = {2, 8, 10, 12, 16};

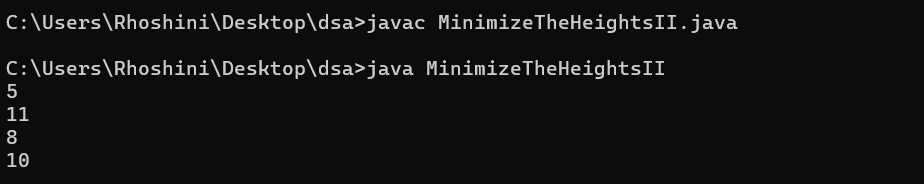
int k4 = 5;

System.out.println(getMinDifference(k4, arr4));

}

}

OUTPUT:



Time Complexity: O(n log n)

Space Complexity: O(n)

1. **Parenthesis Checker**

import java.util.\*;

public class ParenthesisChecker {

public static boolean isValid(String s) {

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

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

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

stack.push(c);

} else if (c == '}' && !stack.isEmpty() && stack.peek() == '{') {

stack.pop();

} else if (c == ')' && !stack.isEmpty() && stack.peek() == '(') {

stack.pop();

} else if (c == ']' && !stack.isEmpty() && stack.peek() == '[') {

stack.pop();

} else {

return false;

}

}

return stack.isEmpty();

}

public static void main(String[] args) {

String s1 = "{([])}";

System.out.println(isValid(s1));

String s2 = "()";

System.out.println(isValid(s2));

String s3 = "([]";

System.out.println(isValid(s3));

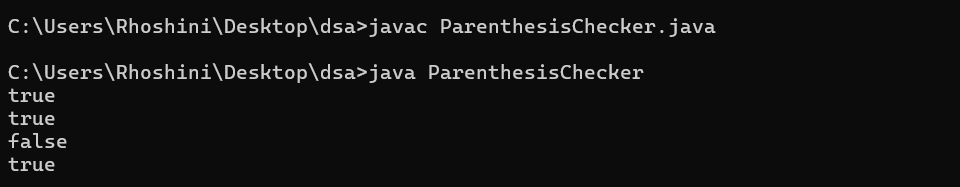
String s4 = "([{}])";

System.out.println(isValid(s4));

}

}

OUTPUT:



Time Complexity: O(n)  
Space Complexity: O(n)

1. **Equilibrium Point**

public class EquilibriumPoint {

public static int equilibriumPoint(int[] arr) {

int totalSum = 0, 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[] arr1 = {1, 3, 5, 2, 2};

System.out.println(equilibriumPoint(arr1));

int[] arr2 = {1};

System.out.println(equilibriumPoint(arr2));

int[] arr3 = {1, 2, 3};

System.out.println(equilibriumPoint(arr3));

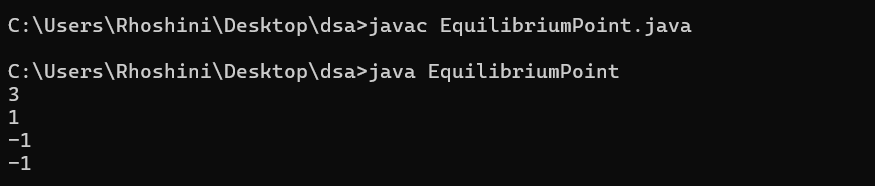
int[] arr4 = {10, 5, 10, 5};

System.out.println(equilibriumPoint(arr4));

}

}

OUTPUT:



Time Complexity: O(n)  
Space Complexity: O(1)

1. **Binary Search**

public class BinarySearch {

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

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

while (low <= high) {

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

if (arr[mid] == k) {

while (mid > 0 && arr[mid - 1] == k) {

mid--;

}

return mid;

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

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

int[] arr1 = {1, 2, 3, 4, 5};

System.out.println(binarySearch(arr1, 4));

int[] arr2 = {11, 22, 33, 44, 55};

System.out.println(binarySearch(arr2, 445));

int[] arr3 = {1, 2, 2, 2, 3};

System.out.println(binarySearch(arr3, 2));

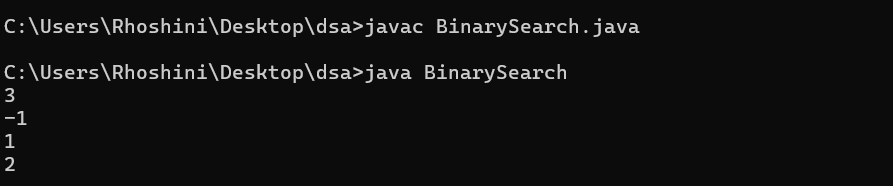
int[] arr4 = {10, 20, 30, 40, 50};

System.out.println(binarySearch(arr4, 30));

}

}

OUTPUT:



Time Complexity: O(log n)  
Space Complexity: O(1)

1. **Next Greater Element**

import java.util.Stack;

public class NextGreaterElement {

public static int[] findNextGreater(int[] arr) {

int n = arr.length;

int[] res = new int[n];

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

for (int i = n - 1; i >= 0; i--) {

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

stack.pop();

}

res[i] = stack.isEmpty() ? -1 : stack.peek();

stack.push(arr[i]);

}

return res;

}

public static void main(String[] args) {

int[] arr1 = {1, 3, 2, 4};

for (int val : findNextGreater(arr1)) {

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

}

System.out.println();

int[] arr2 = {6, 8, 0, 1, 3};

for (int val : findNextGreater(arr2)) {

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

}

System.out.println();

int[] arr3 = {10, 20, 30, 50};

for (int val : findNextGreater(arr3)) {

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

}

System.out.println();

int[] arr4 = {50, 40, 30, 10};

for (int val : findNextGreater(arr4)) {

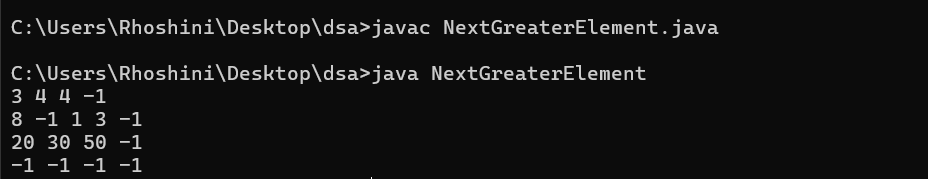
System.out.print(val + " ");

}

}

}

OUTPUT:



Time Complexity: O(n)  
Space Complexity: O(n)

1. **Union of Arrays with Duplicates**

import java.util.HashSet;

public class UnionOfArrays {

public static int findUnionCount(int[] a, int[] b) {

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

for (int val : a) {

set.add(val);

}

for (int val : b) {

set.add(val);

}

return set.size();

}

public static void main(String[] args) {

int[] a1 = {1, 2, 3, 4, 5};

int[] b1 = {1, 2, 3};

System.out.println(findUnionCount(a1, b1));

int[] a2 = {85, 25, 1, 32, 54, 6};

int[] b2 = {85, 2};

System.out.println(findUnionCount(a2, b2));

int[] a3 = {1, 2, 1, 1, 2};

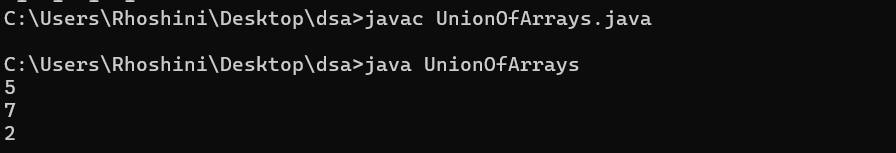
int[] b3 = {2, 2, 1, 2, 1};

System.out.println(findUnionCount(a3, b3));

}

}

OUTPUT:



Time Complexity: O(n + m)  
Space Complexity: O(n + m)