13/11/24

1. Kth Smallest Element

Code:

import java.util.PriorityQueue;

class KthSmallest {

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

PriorityQueue<Integer> maxHeap = new PriorityQueue<>((a, b) -> b - a);

for (int num : arr) {

maxHeap.add(num);

if (maxHeap.size() > k) maxHeap.poll();

}

return maxHeap.peek();

}

}

Time Complexity: O(n logk)

Space Complexity: O(k)

1. Minimize the Heights II

Code:

import java.util.Arrays;

class MinimizeHeightDifference {

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

int n = arr.length;

Arrays.sort(arr);

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

int smallest = arr[0] + k;

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

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

int min = Math.min(smallest, arr[i + 1] - k);

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

if (min >= 0) result = Math.min(result, max - min);

}

return result;

}

}

Time Complexity: O(nlogn)

Space Complexity: O(1)

1. Parenthesis Checker

Code:

import java.util.Stack;

class BalancedBrackets {

public boolean isBalanced(String s) {

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

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

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

stack.push(ch);

} else {

if (stack.isEmpty()) return false;

char top = stack.pop();

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

return false;

}

}

}

return stack.isEmpty();

}

}

Time Complexity: O(n)

Space Complexity: O(n)

1. Equilibrium Point

Code:

class EquilibriumPoint {

public int findEquilibriumPoint(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;

}

}

Time Complexity: O(n)

Space Complexity: O(1)

1. Binary Search

Code:

class BinarySearch {

public int findPosition(int[] arr, int k) {

int left = 0, right = arr.length - 1;

int result = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == k) {

result = mid;

right = mid - 1;

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

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

}

Time Complexity: O(logn)

Space Complexity: O(1)

1. Next Greater Element

Code:

import java.util.Stack;

class NextGreaterElement {

public int[] findNextGreaterElements(int[] arr) {

int n = arr.length;

int[] result = 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();

}

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

stack.push(arr[i]);

}

return result;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

1. Union of Two Arrays with Duplicate Elements

Code:

import java.util.HashSet;

class UnionCount {

public int countUnion(int[] a, int[] b) {

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

for (int num : a) set.add(num);

for (int num : b) set.add(num);

return set.size();

}

}

Time Complexity: O(m+n)

Space Complexity: O(m+n)