**Bhuvaneswari P AIDS DSA Practice-4 13/11/2024**

**1.Kth Smallest**

class KthSmallest {

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

int maxElement = 0;

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

maxElement = Math.max(maxElement, arr[i]);

}

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

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

freq[arr[i]]++;

}

int count = 0;

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

count += freq[i];

if (count >= k) {

return i;

}

}

return -1;

}

public static void main(String[] args) {

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

int k1 = 3;

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

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

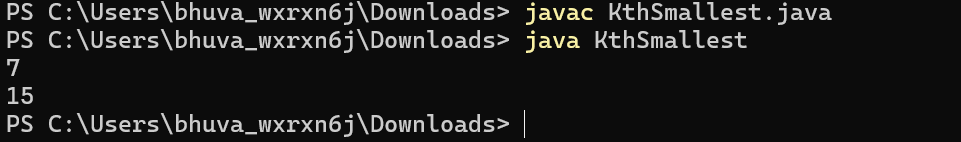
int k2 = 4;

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

}

}

**Output:**



Time Complexity: O(n+maxElement)

Space Complexity: O(maxElement)

**2.Minimize Heights II**

import java.util.Arrays;

class MinimizeHeights {

public static int getMinDifference(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++) {

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

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

result = Math.min(result, maxHeight - minHeight);

}

return result;

}

public static void main(String[] args) {

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

int k1 = 2;

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

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

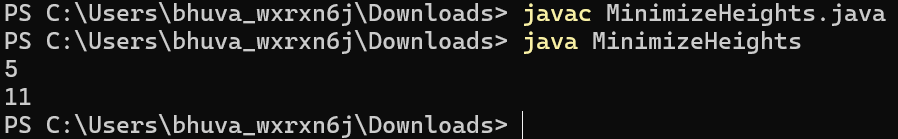
int k2 = 3;

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

}

}

**Output:**



Time complexity : O(n log n)

Space complexity: O(n)

**3.Parenthesis Checker**

import java.util.Stack;

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) {

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

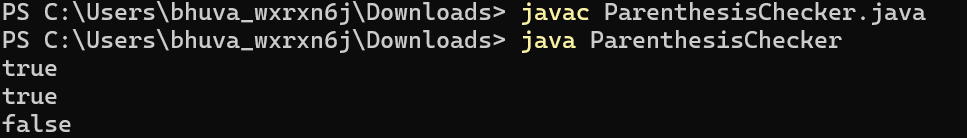
System.out.println(isValid("()"));

System.out.println(isValid("([]"));

}

}

**Output:**



Time complexity :O(n)

Space Complexity: O(n)

**4.Equilibrium Point**

public class EquilibriumPoint {

public static int equilibriumPoint(int[] arr) {

int left = 0;

int right = 0;

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

right += arr[i];

}

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

right -= arr[i];

if (left == right) {

return i + 1;

}

left += arr[i];

}

return -1;

}

public static void main(String[] args) {

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

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

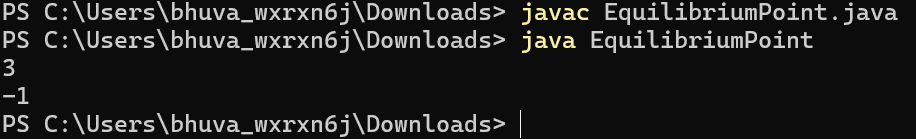
int[] arr2 = {1, 2, 3};

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

}

}

**Output:**



Time Complexity:O(n)

Space Complexity:O(1)

**5.Binary Search**

class BinarySearch {

public int search(int[] nums, int target) {

int left = 0;

int right = nums.length - 1;

while (left <= right) {

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

if (nums[mid] == target) {

return mid;

} else if (nums[mid] > target) {

right = mid - 1;

} else {

left = mid + 1;

}

}

return -1;

}

public static void main(String[] args) {

BinarySearch bs = new BinarySearch();

int[] nums = {-1, 0, 3, 5, 9, 12};

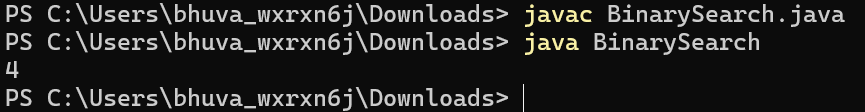
int target = 9;

System.out.println(bs.search(nums, target));

}

}

**Output:**



Time Complexity: O(log n)

Space Complexity: O(1)

**6.Next Greater Element**

import java.util.\*;

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();

}

if (stack.isEmpty()) {

result[i] = -1;

} else {

result[i] = stack.peek();

}

stack.push(arr[i]);

}

return result;

}

public static void main(String[] args) {

NextGreaterElement next = new NextGreaterElement();

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

System.out.println(Arrays.toString(next.findNextGreaterElements(arr)));

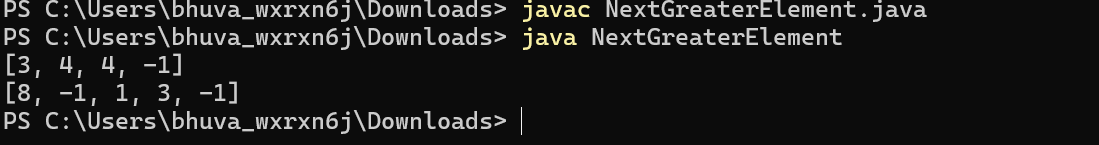
arr = new int[]{6, 8, 0, 1, 3};

System.out.println(Arrays.toString(next.findNextGreaterElements(arr)));

}

}

**Output:**



Time Complexity: O(n)

Space Complexity: O(n)

**7.Union of Arrays**

import java.util.\*;

class UnionOfArrays {

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

Set<Integer> unionSet = new HashSet<>();

for (int num : a) {

unionSet.add(num);

}

for (int num : b) {

unionSet.add(num);

}

return unionSet.size();

}

public static void main(String[] args) {

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

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

System.out.println(findUnionCount(a, b));

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

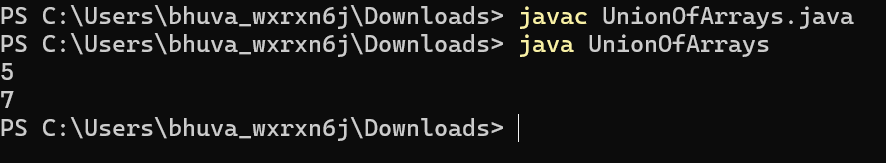
int[] b2 = {85, 2};

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

}

}

**Output:**



Time complexity: O(n+m)

Space Complexity: O(n+m)