**Coding Practice Day 4**

**1.** **Kth Smallest Element**

**Code**

import java.util.PriorityQueue;

public class Main {

public static void main(String[] args) {

int[] numbers = {10, 5, 4, 3, 48, 6, 2, 33, 53, 10};

int k = 4;

int arrayLength = numbers.length;

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

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

maxHeap.offer(numbers[i]);

if (maxHeap.size() > k) {

maxHeap.poll();

}

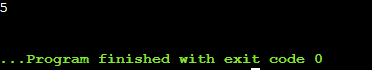
}

System.out.println(maxHeap.peek());

}

}

**Output**



Time Complexity:O(N \* log(k))

Space Complexity: O(k)

**2.** **Minimize the height II**

**Code**

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

int adjustment = 6;

int[] heights = {12, 6, 4, 15, 17, 10};

int arrayLength = heights.length;

Arrays.sort(heights);

int minDifference = heights[arrayLength - 1] - heights[0];

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

if (heights[i] - adjustment < 0)

continue;

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

int maxHeight = Math.max(heights[i - 1] + adjustment, heights[arrayLength - 1] - adjustment);

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

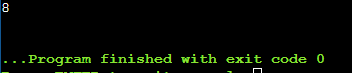
}

System.out.println(minDifference);

}

}

**Output**



Time Complexity:O(n)

Space Complexity: O(n)

**3.** **Parentheses Checker**

**Code**

import java.util.Stack;

public class Main {

public static void main(String[] args) {

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

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

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

char currentChar = expression.charAt(index);

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

bracketStack.push(currentChar);

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

if (bracketStack.isEmpty()) {

System.out.println("false");

return;

}

if ((bracketStack.peek() == '(' && currentChar == ')') ||

(bracketStack.peek() == '{' && currentChar == '}') ||

(bracketStack.peek() == '[' && currentChar == ']')) {

bracketStack.pop();

} else {

System.out.println("false");

return;

}

}

}

if (bracketStack.isEmpty()) {

System.out.println("true");

} else {

System.out.println("false");

}

}

}

**Output  
**

Time Complexity:O(n)

Space Complexity: O(n)

**4.** **Equilibrium point**

**Code**import java.util.\*;

public class Main {

public static void main(String[] args) {

long[] numbers = { -7, 1, 5, 2, -4, 3, 0 };

int arrayLength = numbers.length;

long leftSum, rightSum;

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

leftSum = 0;

for (int j = 0; j < i; j++)

leftSum += numbers[j];

rightSum = 0;

for (int j = i + 1; j < arrayLength; j++)

rightSum += numbers[j];

if (leftSum == rightSum) {

System.out.println(i + 1);

return;

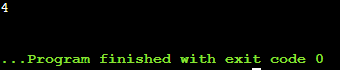
}

}

System.out.println(-1);

}

}

**Output  
**

Time complexity:O(n^2)

Space complexity:O(1)

**5.** **Binary Search**

**Code**

class Main {

public static void main(String args[]) {

int[] numbers = { 2, 3, 4, 10, 40 };

int target = 10;

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

int resultIndex = -1;

while (left <= right) {

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

if (numbers[mid] == target) {

resultIndex = mid;

break;

}

if (numbers[mid] < target)

left = mid + 1;

else

right = mid - 1;

}

if (resultIndex == -1)

System.out.println("Element is not present in array");

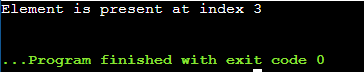
else

System.out.println("Element is present at index " + resultIndex);

}

}

**Output**

****

Time complexity:O(log n)

Space complexity:O(1)

**6.** **Next Greater Element**

**Code**

class Main {

public static void main(String args[]) {

int[] nums = { 11, 13, 21, 3 };

int len = nums.length;

int next, i, j;

for (i = 0; i < len; i++) {

next = -1;

for (j = i + 1; j < len; j++) {

if (nums[i] < nums[j]) {

next = nums[j];

break;

}

}

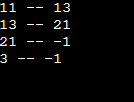
System.out.println(nums[i] + " -- " + next);

}

}

}

**Output**

****

Time complexity:O(n^2)

Space complexity:O(1)

**7. Union of Two Arrays with Duplicate Elements**

**Code**

import java.util.HashSet;

import java.util.Scanner;

class Main {

public 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) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of elements in the first array: ");

int n1 = sc.nextInt();

int[] arr1 = new int[n1];

System.out.println("Enter the elements of the first array:");

for (int i = 0; i < n1; i++) {

arr1[i] = sc.nextInt();

}

System.out.print("Enter the number of elements in the second array: ");

int n2 = sc.nextInt();

int[] arr2 = new int[n2];

System.out.println("Enter the elements of the second array:");

for (int i = 0; i < n2; i++) {

arr2[i] = sc.nextInt();

}

Main sol = new Main();

int unionCount = sol.findUnion(arr1, arr2);

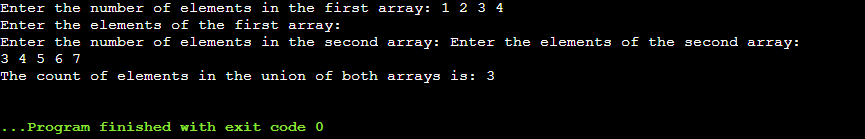
System.out.println("The count of elements in the union of both arrays is: " + unionCount);

sc.close();

}

}

**Output**

****

Time complexity : O(n)

Space complexity : O(n)