**20/11/2024**

**DSA PRACTICE DAY - 8**

**1.3 sum Closest**

Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target.

Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

**Example 1:**

**Input:** nums = [-1,2,1,-4], target = 1

**Output:** 2

**Explanation:** The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

**Example 2:**

**Input:** nums = [0,0,0], target = 1

**Output:** 0

**Explanation:** The sum that is closest to the target is 0. (0 + 0 + 0 = 0).

**Constraints:**

* 3 <= nums.length <= 500
* -1000 <= nums[i] <= 1000
* -104 <= target <= 104

**Code:**

import java.util.Arrays;

class Solution {

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

Arrays.sort(nums);

int closest = nums[0] + nums[1] + nums[2];

for (int i = 0; i < nums.length - 2; i++) {

int left = i + 1, right = nums.length - 1;

while (left < right) {

int currSum = nums[i] + nums[left] + nums[right];

if (currSum == target) {

return target;

}

if (Math.abs(currSum - target) < Math.abs(closest - target)) {

closest = currSum;

}

if (currSum < target) {

left++;

} else {

right--;

}

}

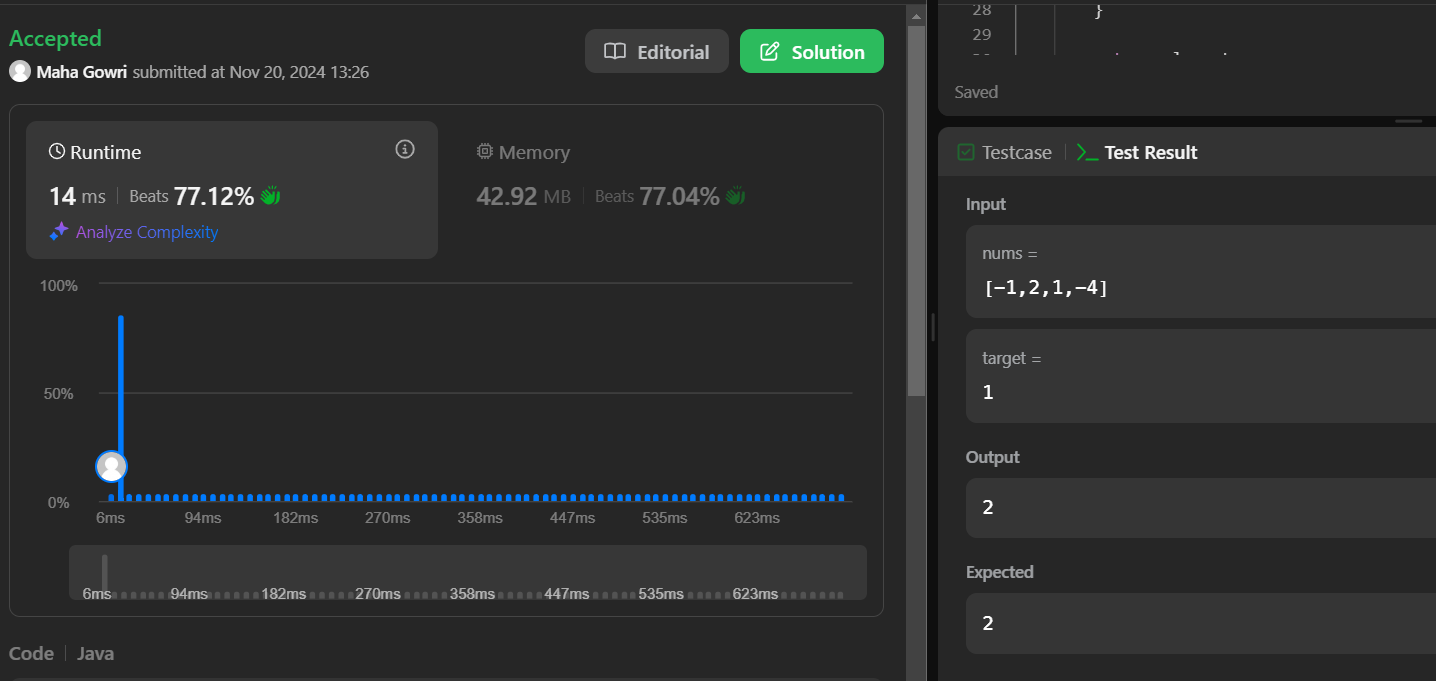
}

return closest;

}

}

**Output:**

****

**2.Jump game II**

You are given a **0-indexed** array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

* 0 <= j <= nums[i] and
* i + j < n

Return *the minimum number of jumps to reach*nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

**Example 1:**

**Input:** nums = [2,3,1,1,4]

**Output:** 2

**Explanation:** The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

**Example 2:**

**Input:** nums = [2,3,0,1,4]

**Output:** 2

**Constraints:**

* 1 <= nums.length <= 104
* 0 <= nums[i] <= 1000
* It's guaranteed that you can reach nums[n - 1].

**Code:**

class Solution {

public int jump(int[] nums) {

int ans = 0;

int end = 0;

int farthest = 0;

for (int i = 0; i < nums.length - 1; ++i) {

farthest = Math.max(farthest, i + nums[i]);

if (farthest >= nums.length - 1) {

++ans;

break;

}

if (i == end) {

++ans;

end = farthest;

}

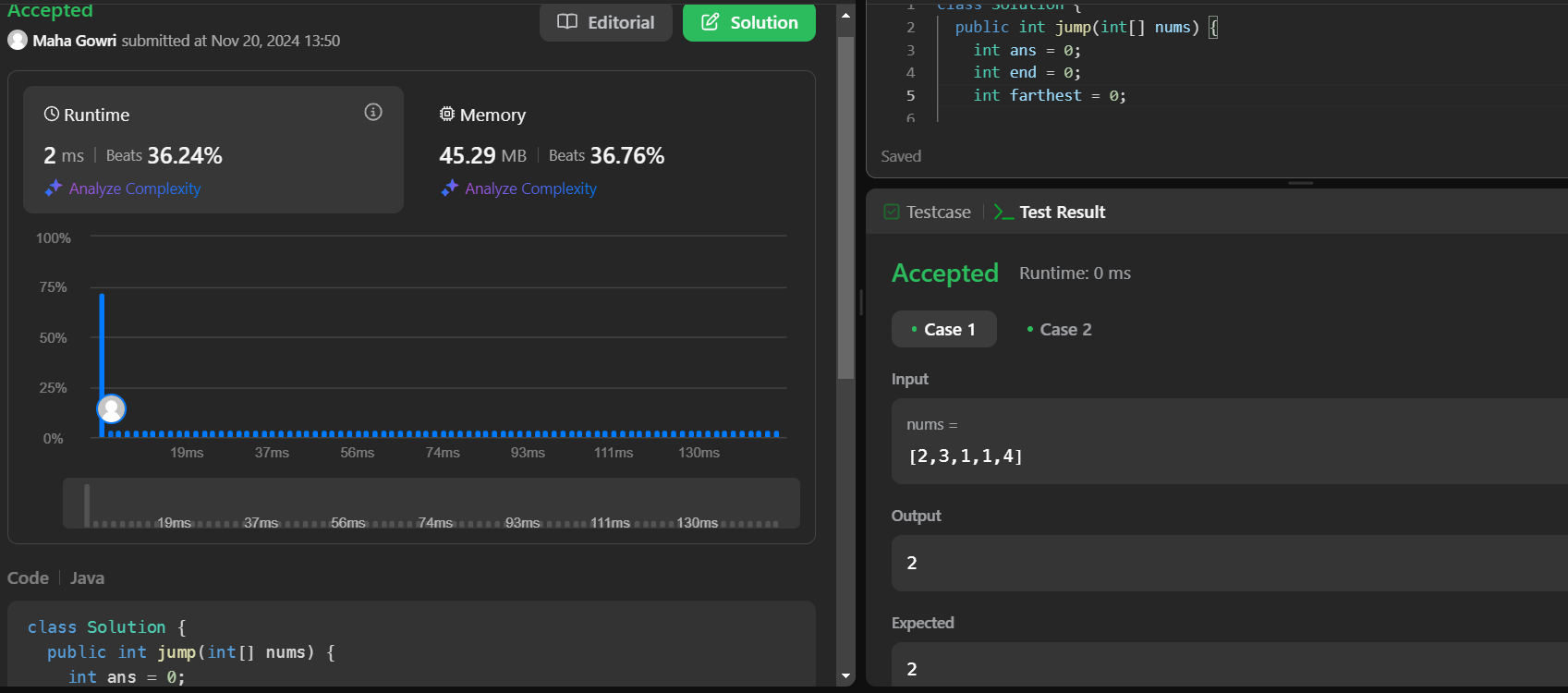
}

return ans;

}

}

**Output:**

****

**3.Group Anagrams**

Given an array of strings strs, group the

anagrams

 together. You can return the answer in **any order**.

**Example 1:**

**Input:** strs = ["eat","tea","tan","ate","nat","bat"]

**Output:** [["bat"],["nat","tan"],["ate","eat","tea"]]

**Explanation:**

* There is no string in strs that can be rearranged to form "bat".
* The strings "nat" and "tan" are anagrams as they can be rearranged to form each other.
* The strings "ate", "eat", and "tea" are anagrams as they can be rearranged to form each other.

**Example 2:**

**Input:** strs = [""]

**Output:** [[""]]

**Example 3:**

**Input:** strs = ["a"]

**Output:** [["a"]]

**Constraints:**

* 1 <= strs.length <= 104
* 0 <= strs[i].length <= 100
* strs[i] consists of lowercase English letters.

**Code:**

class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

Map<String, List<String>> map = new HashMap<>();

for (String word : strs) {

char[] chars = word.toCharArray();

Arrays.sort(chars);

String sortedWord = new String(chars);

if (!map.containsKey(sortedWord)) {

map.put(sortedWord, new ArrayList<>());

}

map.get(sortedWord).add(word);

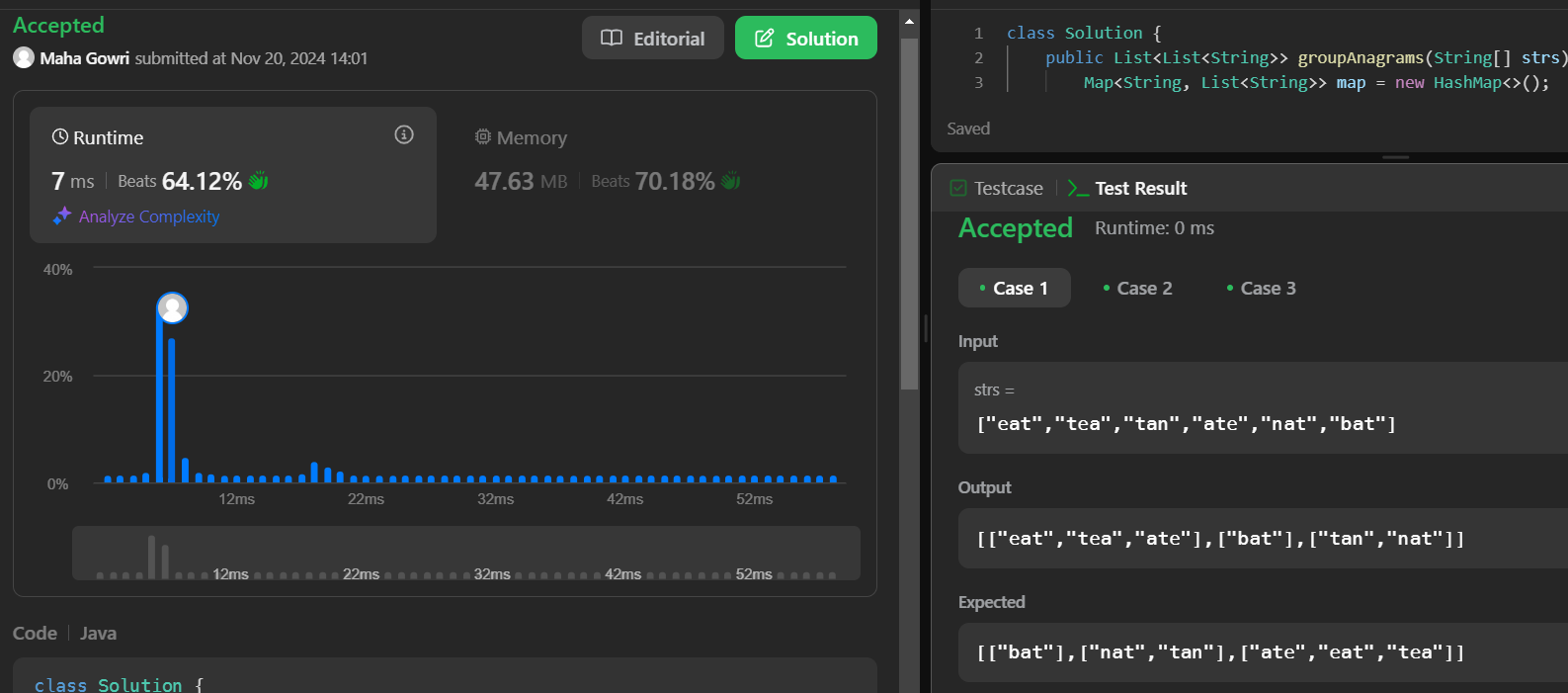
}

return new ArrayList<>(map.values());

}

}

**Output:**

****

**4.Decode Ways**

You have intercepted a secret message encoded as a string of numbers. The message is **decoded** via the following mapping:

"1" -> 'A'  
"2" -> 'B'  
...  
"25" -> 'Y'  
"26" -> 'Z'

However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes ("2" and "5" vs "25").

For example, "11106" can be decoded into:

* "AAJF" with the grouping (1, 1, 10, 6)
* "KJF" with the grouping (11, 10, 6)
* The grouping (1, 11, 06) is invalid because "06" is not a valid code (only "6" is valid).

Note: there may be strings that are impossible to decode.  
  
Given a string s containing only digits, return the **number of ways** to **decode** it. If the entire string cannot be decoded in any valid way, return 0.

The test cases are generated so that the answer fits in a **32-bit** integer.

**Example 1:**

**Input:** s = "12"

**Output:** 2

**Explanation:**

"12" could be decoded as "AB" (1 2) or "L" (12).

**Example 2:**

**Input:** s = "226"

**Output:** 3

**Explanation:**

"226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

**Example 3:**

**Input:** s = "06"

**Output:** 0

**Explanation:**

"06" cannot be mapped to "F" because of the leading zero ("6" is different from "06"). In this case, the string is not a valid encoding, so return 0.

**Constraints:**

* 1 <= s.length <= 100
* s contains only digits and may contain leading zero(s).

**Code:**

class Solution {

public int numDecodings(String s) {

int strLen = s.length();

int[] dp = new int[strLen + 1];

dp[0] = 1;

if (s.charAt(0) != '0') {

dp[1] = 1;

} else {

return 0;

}

for (int i = 2; i <= strLen; ++i) {

if (s.charAt(i - 1) != '0') {

dp[i] += dp[i - 1];

}

if (s.charAt(i - 2) == '1' ||

(s.charAt(i - 2) == '2' && s.charAt(i - 1) <= '6')) {

dp[i] += dp[i - 2];

}

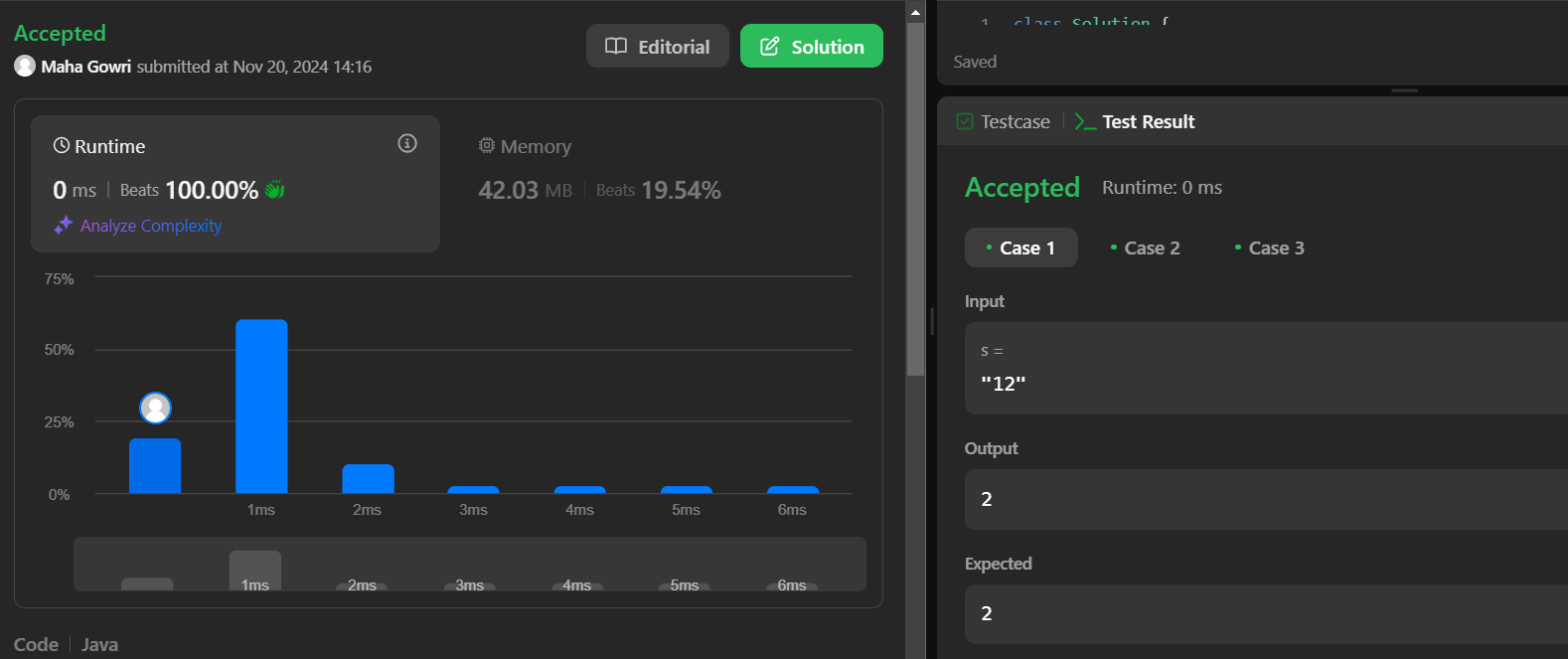
}

return dp[strLen];

}

}

**Output:**

****

**5.Best Time to Buy and Sell Stock II**

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return *the****maximum****profit you can achieve*.

**Example 1:**

**Input:** prices = [7,1,5,3,6,4]

**Output:** 7

**Explanation:** Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Total profit is 4 + 3 = 7.

**Example 2:**

**Input:** prices = [1,2,3,4,5]

**Output:** 4

**Explanation:** Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.

Total profit is 4.

**Example 3:**

**Input:** prices = [7,6,4,3,1]

**Output:** 0

**Explanation:** There is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0.

**Constraints:**

* 1 <= prices.length <= 3 \* 104
* 0 <= prices[i] <= 104

**Code:**

class Solution {

public int maxProfit(int[] prices) {

int max = 0;

int start = prices[0];

int len = prices.length;

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

if(start < prices[i]) max += prices[i] - start;

start = prices[i];

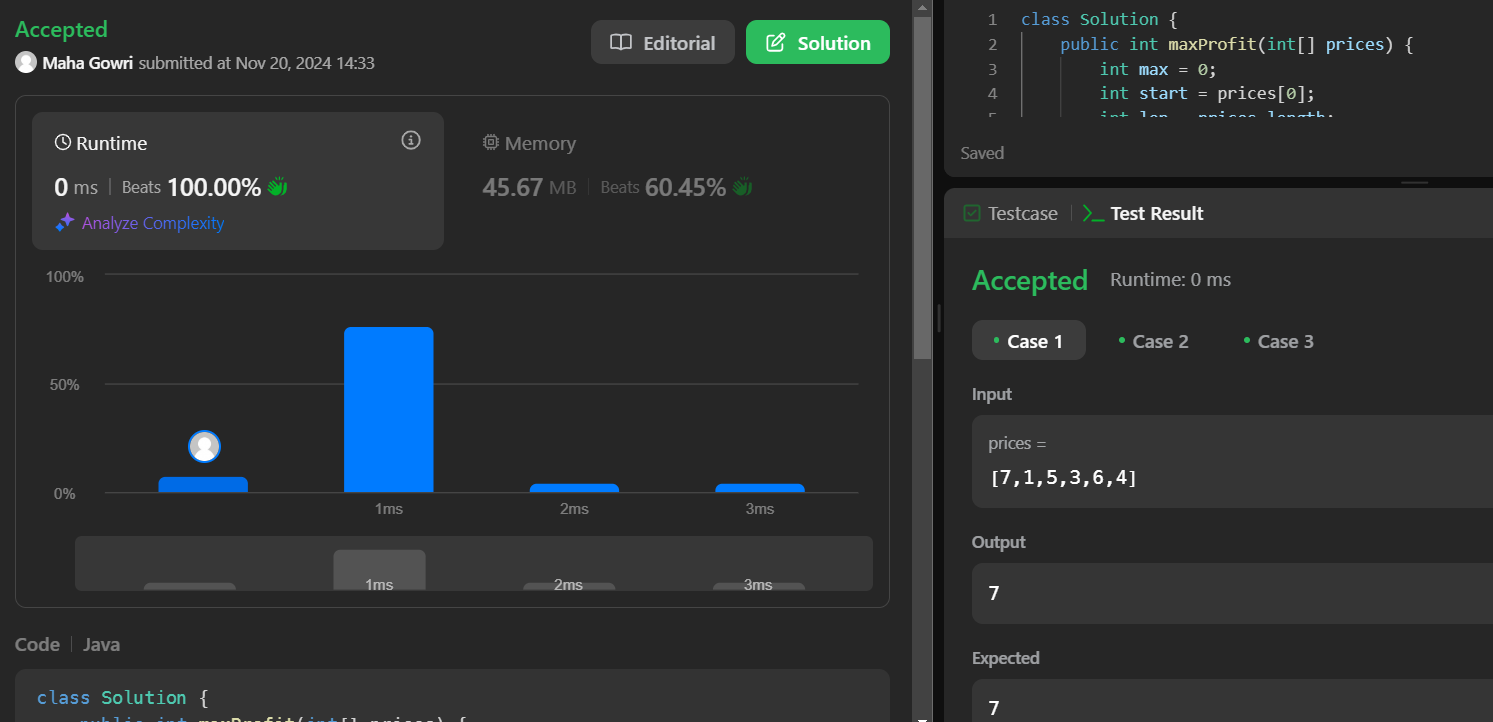
}

return max;

}

}

**Output:**



**6. Number of islands**

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example 1:**

**Input:** grid = [

["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

]

**Output:** 1

**Example 2:**

**Input:** grid = [

["1","1","0","0","0"],

["1","1","0","0","0"],

["0","0","1","0","0"],

["0","0","0","1","1"]

]

**Output:** 3

**Constraints:**

* m == grid.length
* n == grid[i].length
* 1 <= m, n <= 300
* grid[i][j] is '0' or '1'.

**Code:**

class Solution{

public void dfs(char[][]grid,int i,int j){

int m=grid.length,n=grid[0].length;

if(i<0||j<0||i>=m||j>=n||grid[i][j]=='0')return;

grid[i][j]='0';

dfs(grid,i+1,j);

dfs(grid,i-1,j);

dfs(grid,i,j+1);

dfs(grid,i,j-1);

}

public int numIslands(char[][]grid){

int m=grid.length,n=grid[0].length,count=0;

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

for(int j=0;j<n;j++){

if(grid[i][j]=='1'){

count++;

dfs(grid,i,j);

}

}

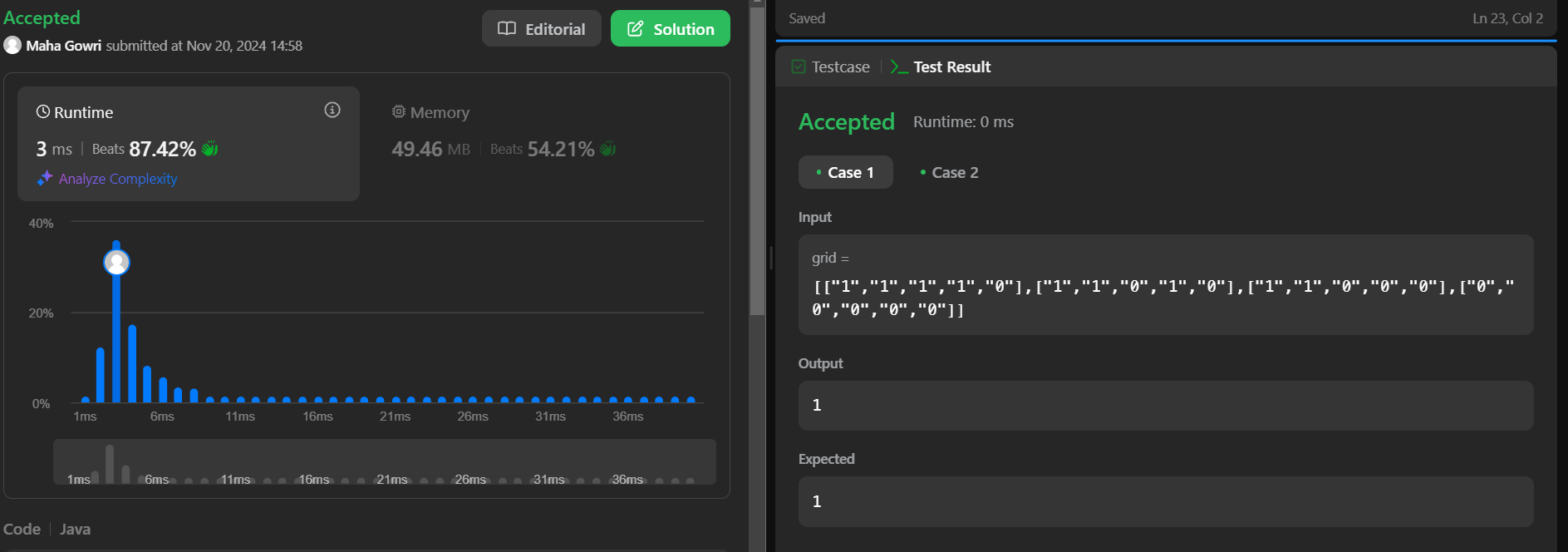
}

return count;

}

}

**Output:**



**7.** **Quick Sort**

**Code:**

import java.util.Arrays;

class Quicksort {

static int partition(int[] arr, int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(arr, i, j);

}

}

swap(arr, i + 1, high);

return i + 1;

}

static void swap(int[] arr, int i, int j) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

static void quickSort(int[] arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

public static void main(String[] args) {

int[] arr = {10, 7, 8, 9, 1, 5};

int n = arr.length;

quickSort(arr, 0, n - 1);

for (int val : arr) {

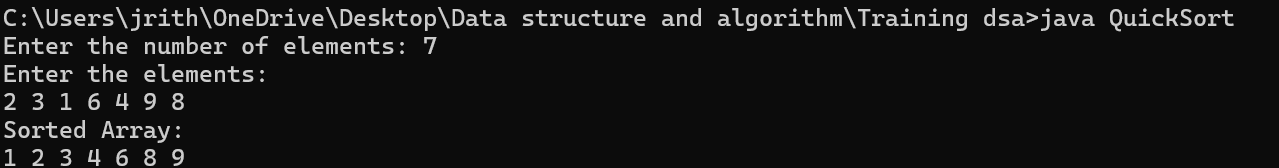
System.out.print(val + " ");

}

}

}

**Output:**



**Time Complexity:O(n log n)**

**8.Merge sort**

**Code:**

import java.util.Scanner;

class MergeSort {

static void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

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

L[i] = arr[l + i];

for (int j = 0; j < n2; ++j)

R[j] = arr[m + 1 + j];

int i = 0, j = 0, k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

static void sort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

sort(arr, l, m);

sort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

static void printArray(int arr[]) {

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

System.out.print(arr[i] + " ");

System.out.println();

}

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int arr[] = new int[n];

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

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

arr[i] = scanner.nextInt();

}

System.out.println("Given array is:");

printArray(arr);

sort(arr, 0, arr.length - 1);

System.out.println("Sorted array is:");

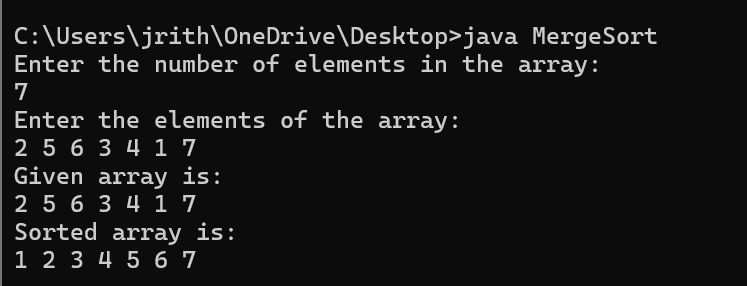
printArray(arr);

scanner.close();

}

}

**Output:**



**9.Ternary Search**

**Code:**

import java.util.Scanner;

import java.util.Arrays;

class TernarySearch {

static int ternarySearch(int l, int r, int key, int ar[]) {

if (r >= l) {

int mid1 = l + (r - l) / 3;

int mid2 = r - (r - l) / 3;

if (ar[mid1] == key) {

return mid1;

}

if (ar[mid2] == key) {

return mid2;

}

if (key < ar[mid1]) {

return ternarySearch(l, mid1 - 1, key, ar);

} else if (key > ar[mid2]) {

return ternarySearch(mid2 + 1, r, key, ar);

} else {

return ternarySearch(mid1 + 1, mid2 - 1, key, ar);

}

}

return -1;

}

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] ar = new int[n];

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

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

ar[i] = scanner.nextInt();

}

Arrays.sort(ar);

System.out.println("Sorted array:");

for (int num : ar) {

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

}

System.out.println();

System.out.println("Enter the key to search:");

int key = scanner.nextInt();

int result = ternarySearch(0, n - 1, key, ar);

if (result != -1) {

System.out.println("Index of " + key + " is " + result);

} else {

System.out.println(key + " is not present in the array.");

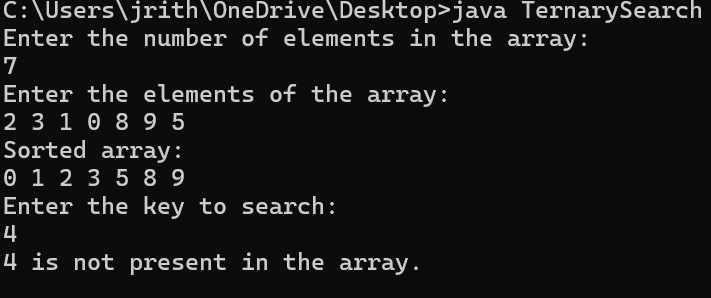
}

scanner.close();

}

}

**Output:**



**10.Interpolation Search**

**Code:**

import java.util.Scanner;

class InterpolationSearch {

public static int interpolationSearch(int arr[], int lo, int hi, int x) {

int pos;

if (lo <= hi && x >= arr[lo] && x <= arr[hi]) {

pos = lo + (((hi - lo) / (arr[hi] - arr[lo])) \* (x - arr[lo]));

if (arr[pos] == x)

return pos;

if (arr[pos] < x)

return interpolationSearch(arr, pos + 1, hi, x);

if (arr[pos] > x)

return interpolationSearch(arr, lo, pos - 1, x);

}

return -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array in sorted order:");

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

arr[i] = scanner.nextInt();

}

System.out.println("Enter the element to search:");

int x = scanner.nextInt();

int index = interpolationSearch(arr, 0, n - 1, x);

if (index != -1)

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

else

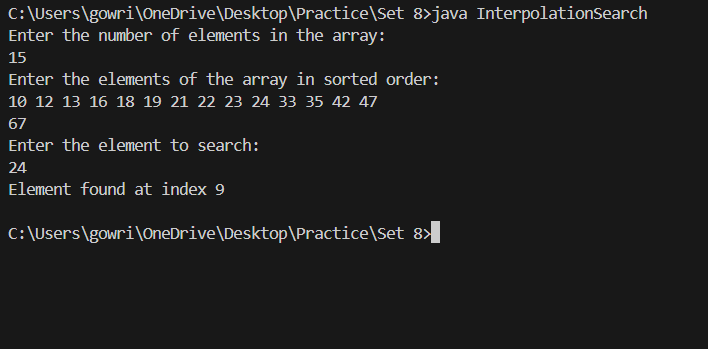
System.out.println("Element not found.");

scanner.close();

}

}

**Output:**

****