**// GEEKSFORGEEKS PRACTISE QUESTION**

**Q1. Count number of elements between two given elements in array**

**class Solution {**

**// Function to get the count of elements between num1 and num2 in array arr.**

**public int getCount(ArrayList<Integer> arr, int num1, int num2) {**

**// Your Code goes here.**

**int leftIndex = arr.indexOf(num1);**

**int rightIndex = arr.lastIndexOf(num2);**

**// Check if the indices are valid and leftIndex is before rightIndex**

**if (leftIndex != -1 && rightIndex != -1 && leftIndex < rightIndex) {**

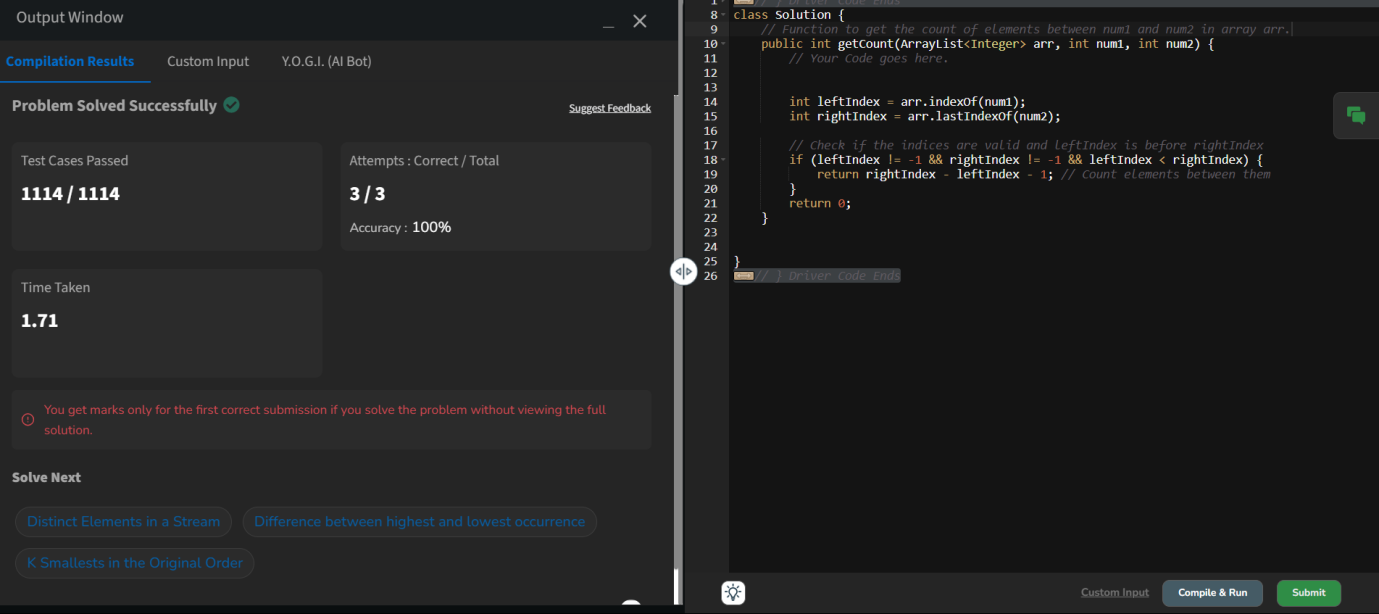
**return rightIndex - leftIndex - 1; // Count elements between them**

**}**

**return 0;**

**}**

}



**Q2.Count Odd Even**

**class Solution {**

**public static in[] countOddEven( int [] arr){**

**int oddCount = 0;**

**int evenCount = 0;**

**for ( int num : arr){**

**if ( num % 2 == 0){**

**evenCount++;**

**}**

**else{**

**oddCount++;**

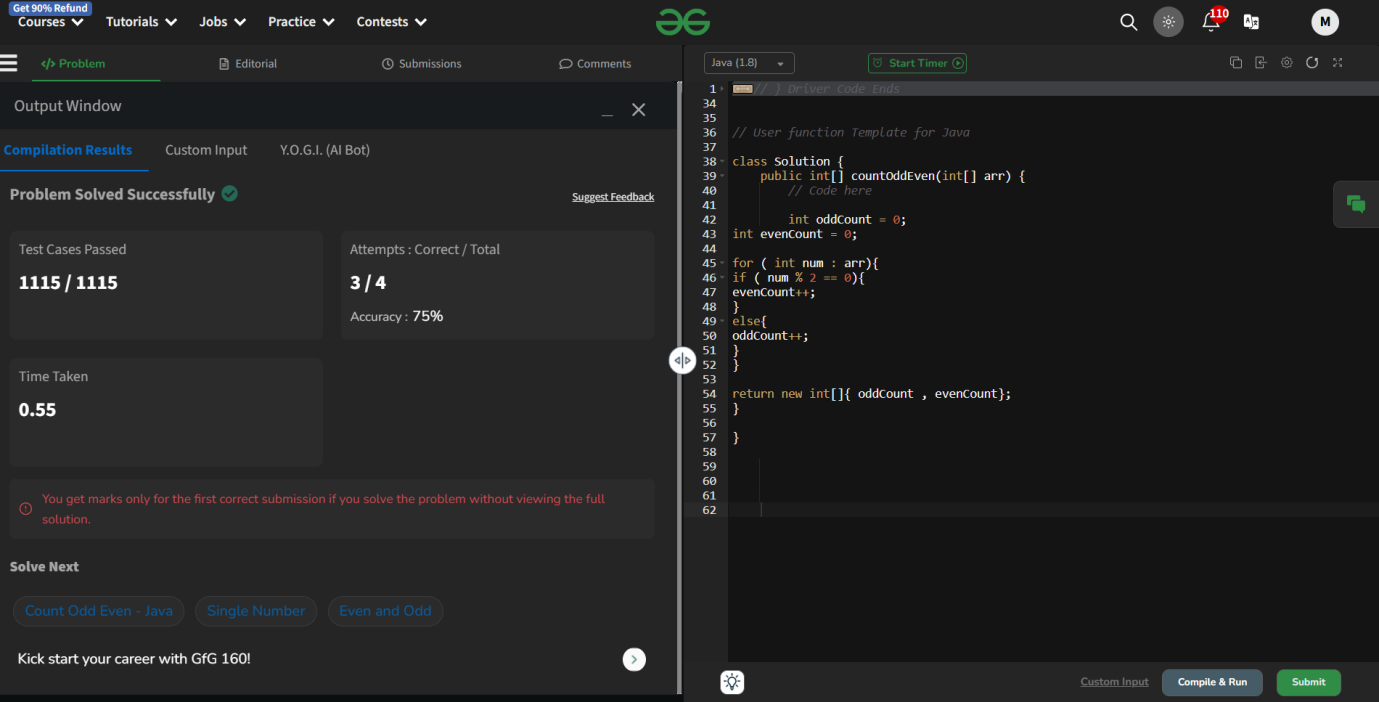
**}**

**}**

**return new int[]{ oddCount , evenCount};**

**}**

**}**

****

Q3. Implement stack using array

**class MyStack {**

**private int[] arr;**

**private int top;**

**// Constructor to initialize stack**

**public MyStack() {**

**arr = new int[1000]; // Initialize array with size 1000**

**top = -1; // Initialize top to -1**

**}**

**// Push operation**

**public void push(int x) {**

**if (top == arr.length - 1) {**

**System.out.println("Stack Overflow"); // Handle overflow if required**

**} else {**

**arr[++top] = x; // Increment top and add element**

**}**

**}**

**// Pop operation**

**public int pop() {**

**if (top == -1) {**

**return -1; // Return -1 if the stack is empty**

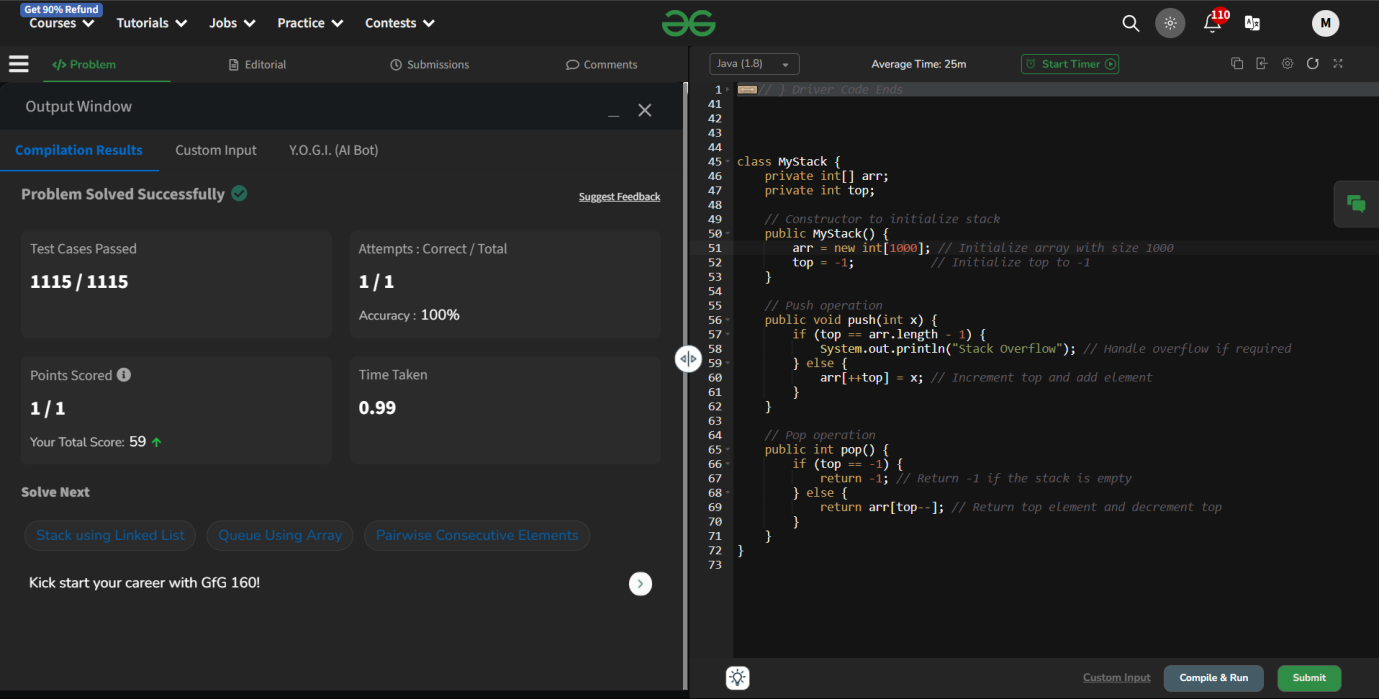
**} else {**

**return arr[top--]; // Return top element and decrement top**

**}**

**}**

**}**

****

Q4. Maximum weight difference

class Solution {

public long MaxWeightDiff(int[] arr, int k) {

int n = arr.length;

// Sort the array

Arrays.sort(arr);

// Case 1: Sum of k smallest elements

int sumKSmallest = 0;

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

sumKSmallest += arr[i];

}

// Case 2: Sum of k largest elements

int sumKLargest = 0;

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

sumKLargest += arr[i];

}

// Total sum of array

int totalSum = 0;

for (int num : arr) {

totalSum += num;

}

// Calculate the differences

int diff1 = Math.abs(sumKSmallest - (totalSum - sumKSmallest));

int diff2 = Math.abs(sumKLargest - (totalSum - sumKLargest));

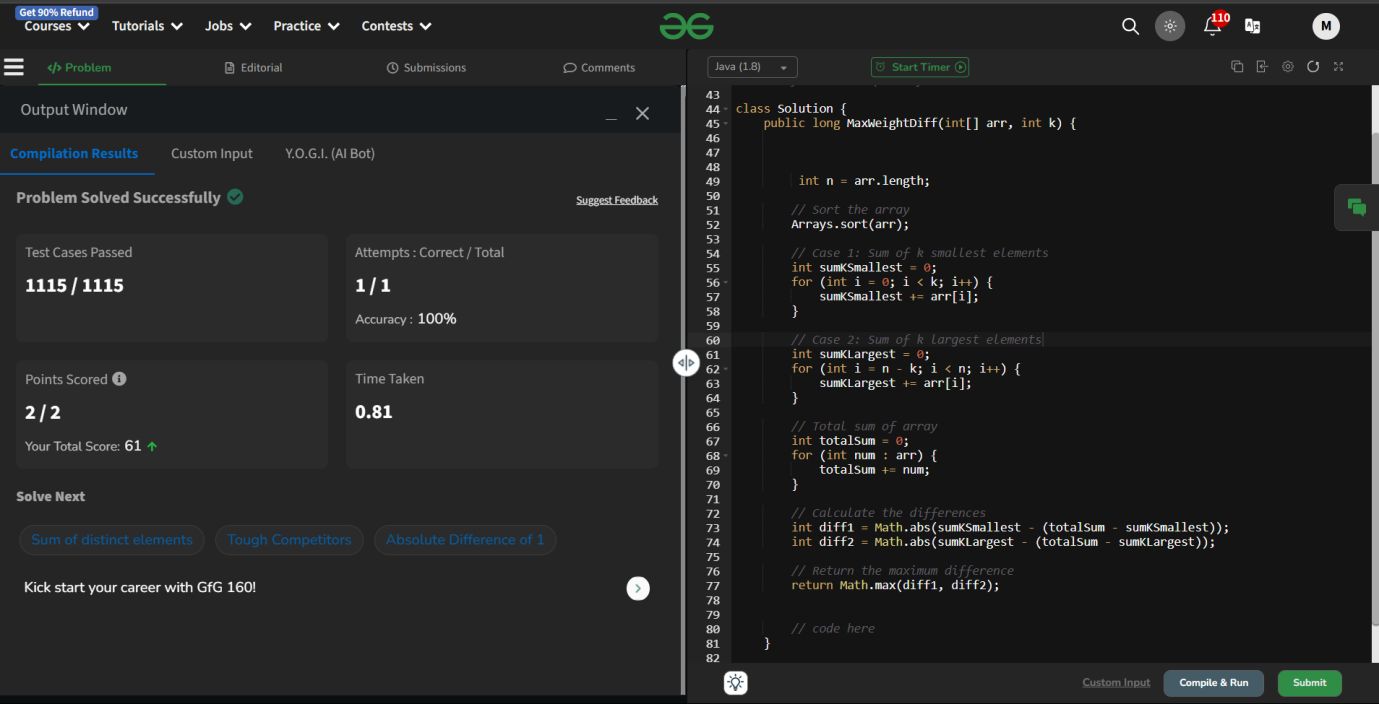
// Return the maximum difference

return Math.max(diff1, diff2);

// code here

}

}



**Q5. Missing number in shuffled array**

**class Solution {**

**public int findMissing(int[] arr1, int[] arr2) {**

**// code here**

**int sum1 = 0;**

**for ( int num : arr1){**

**sum1 += num;**

**}**

**int sum2 = 0;**

**for ( int num : arr2){**

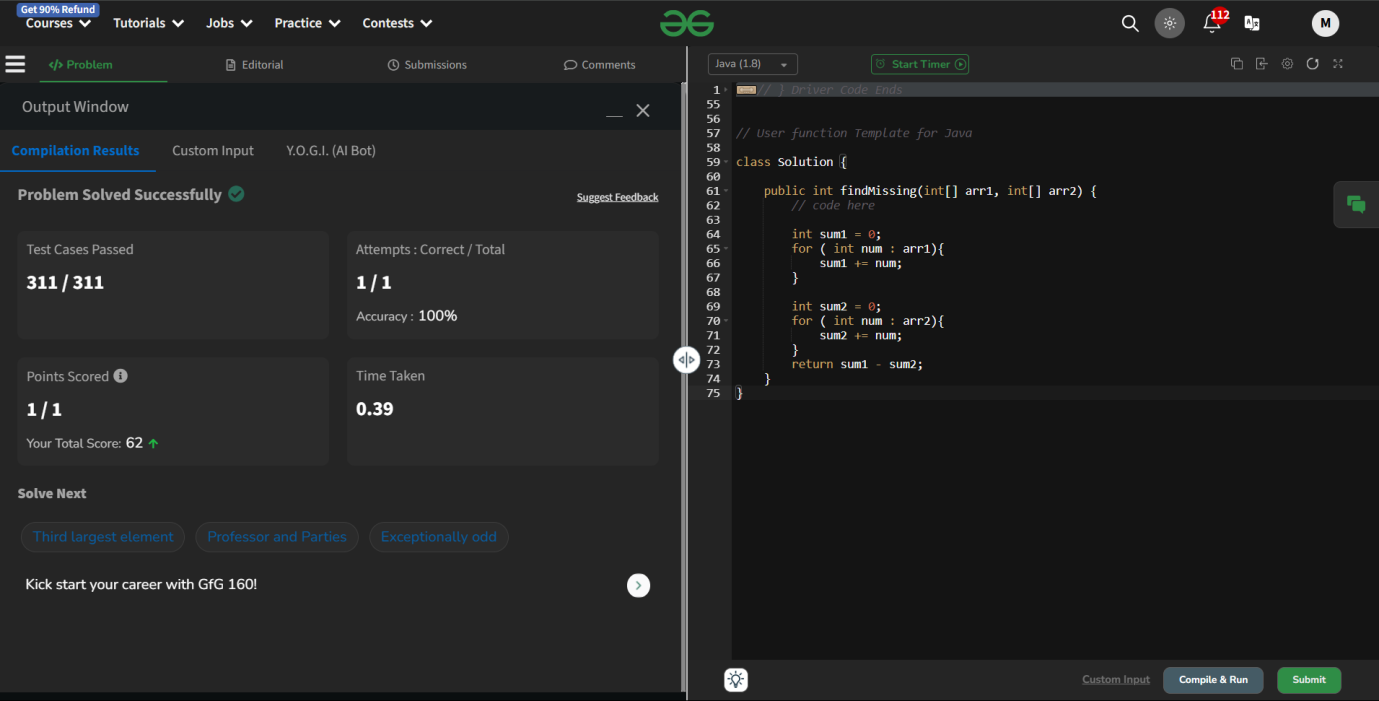
**sum2 += num;**

**}**

**return sum1 - sum2;**

**}**

**}**

****

**Q6. Perfect Array**

**class Solution {**

**public boolean isPerfect(int[] arr) {**

**// code here**

**int n = arr.length;**

**if (n < 2) return true; // An array with less than 2 elements is always perfect.**

**int i = 0;**

**// Check for strictly increasing part**

**while (i < n - 1 && arr[i] < arr[i + 1]) {**

**i++;**

**}**

**// Check for constant part**

**while (i < n - 1 && arr[i] == arr[i + 1]) {**

**i++;**

**}**

**// Check for strictly decreasing part**

**while (i < n - 1 && arr[i] > arr[i + 1]) {**

**i++;**

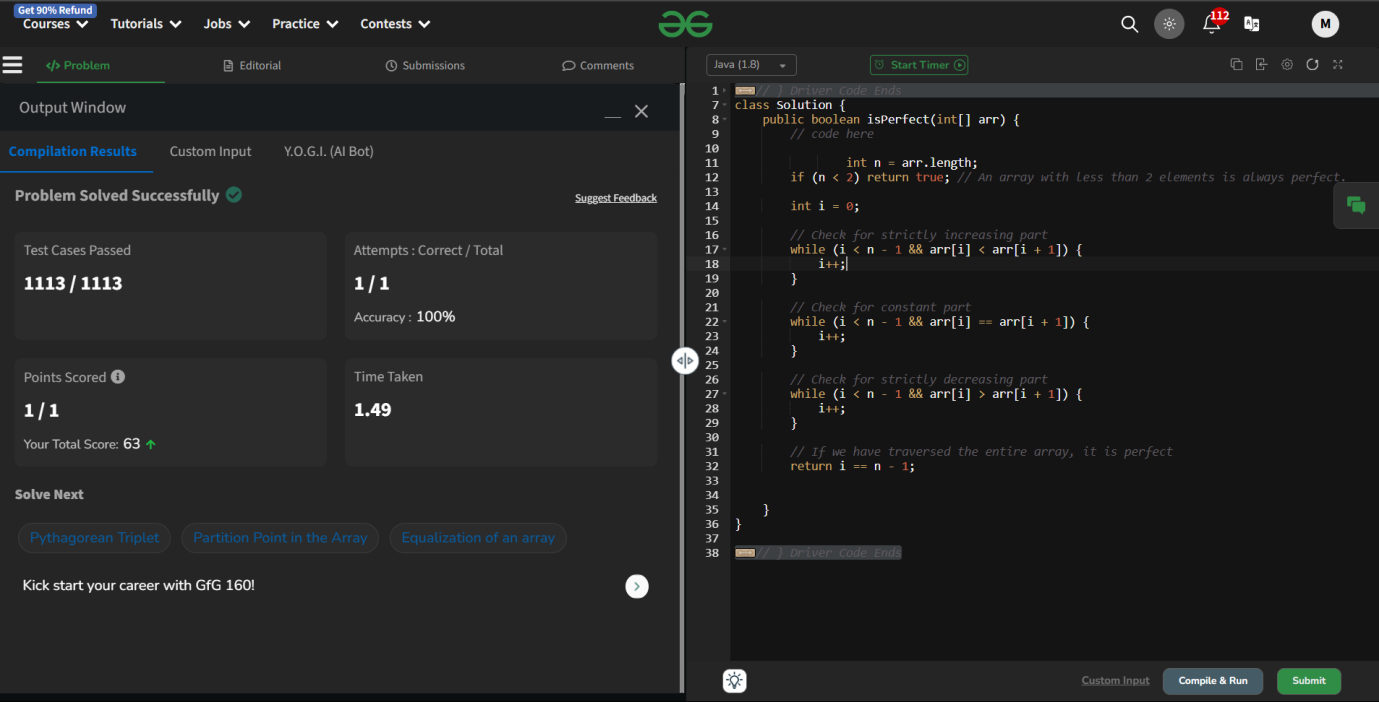
**}**

**// If we have traversed the entire array, it is perfect**

**return i == n - 1;**

**}**

**}**

****

**Q7. Find unique element**

**class Solution {**

**public int findUnique(int k, int[] arr) {**

**// code here**

**int [] bitCount = new int[32];**

**for ( int num : arr){**

**for( int i = 0; i < 32 ; i++){**

**bitCount[i] += (num >> i) & 1;**

**}**

**}**

**int result = 0;**

**for ( int i = 0; i < 32 ; i++){**

**if ( bitCount[i] % k != 0){**

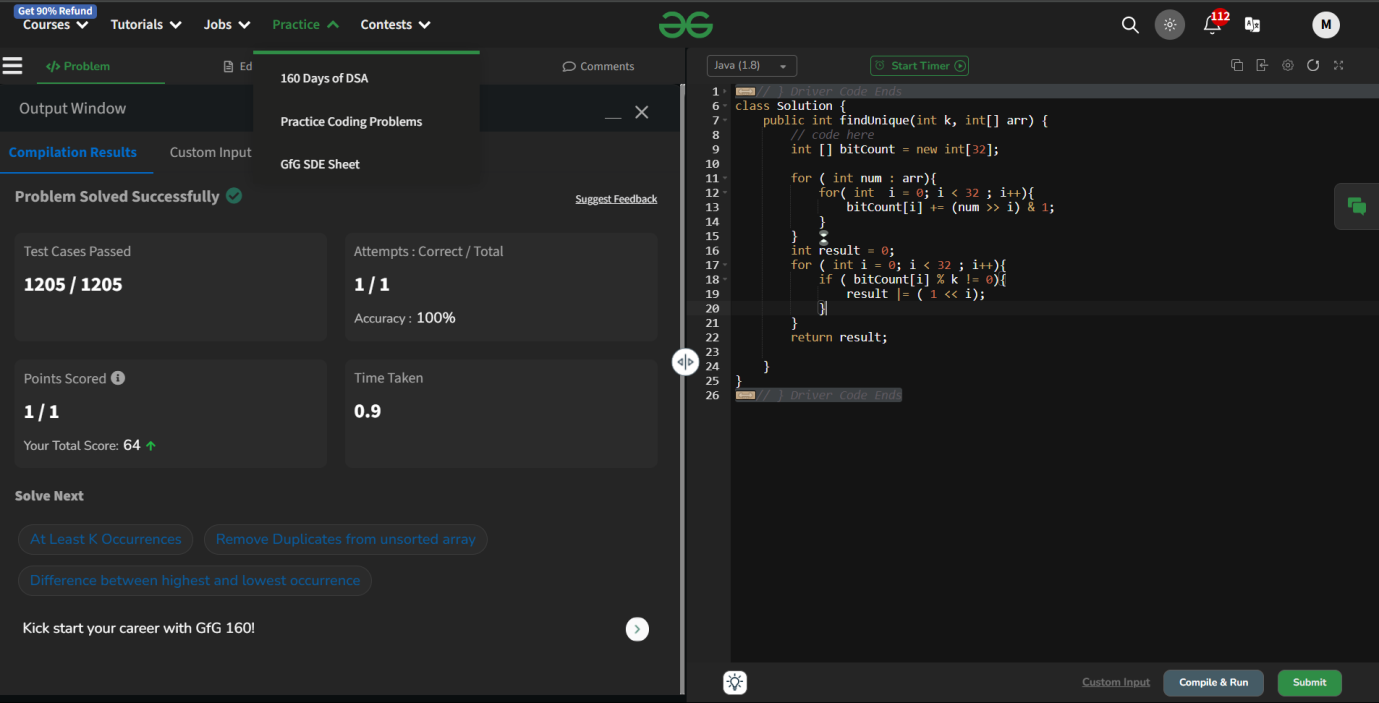
**result |= ( 1 << i);**

**}**

**}**

**return result;**

**}**

**}**