SRI VASAVI ENGINEERING COLLEGE (Autonomous)

PEDATADEPALLI, TADEPALLIGUDEM.

Certificate

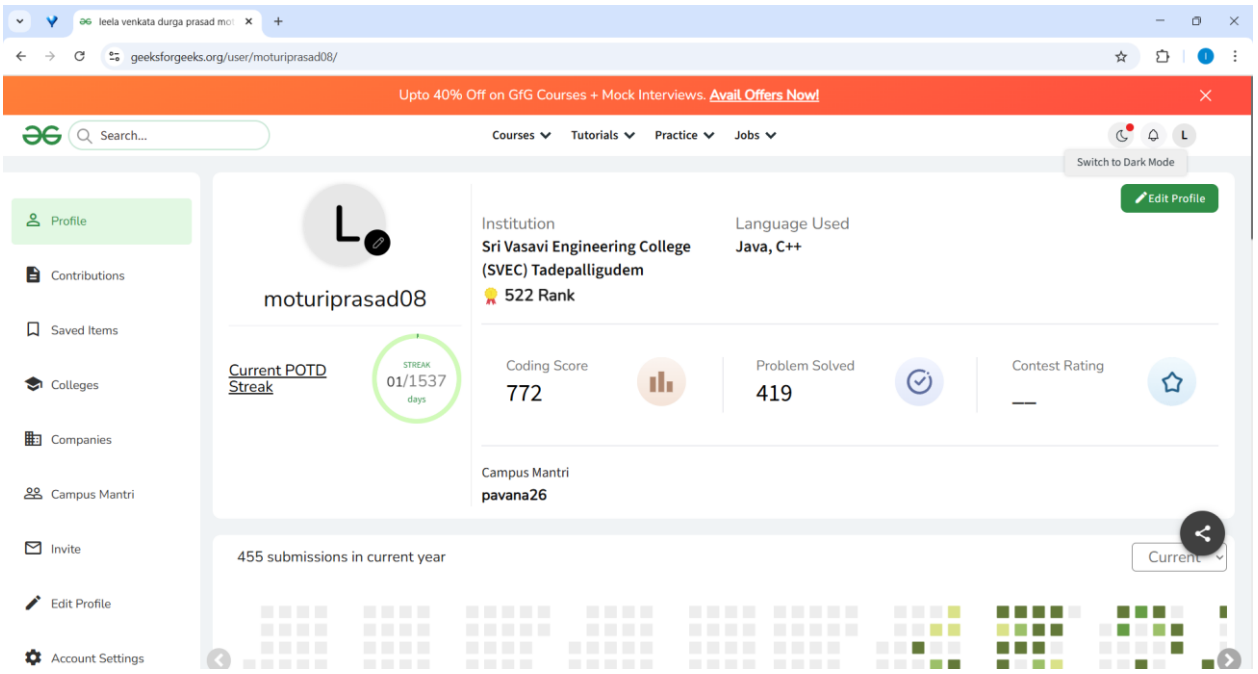
This is to certify that this is a bonafide record of Practical Work done in ***Master Coding in Competitive Programming-I Lab*** *by Mr / Ms* _____ *bearing Roll No.* _____ *of* _____ *Branch of V Semester during the academic year 2025-26.*

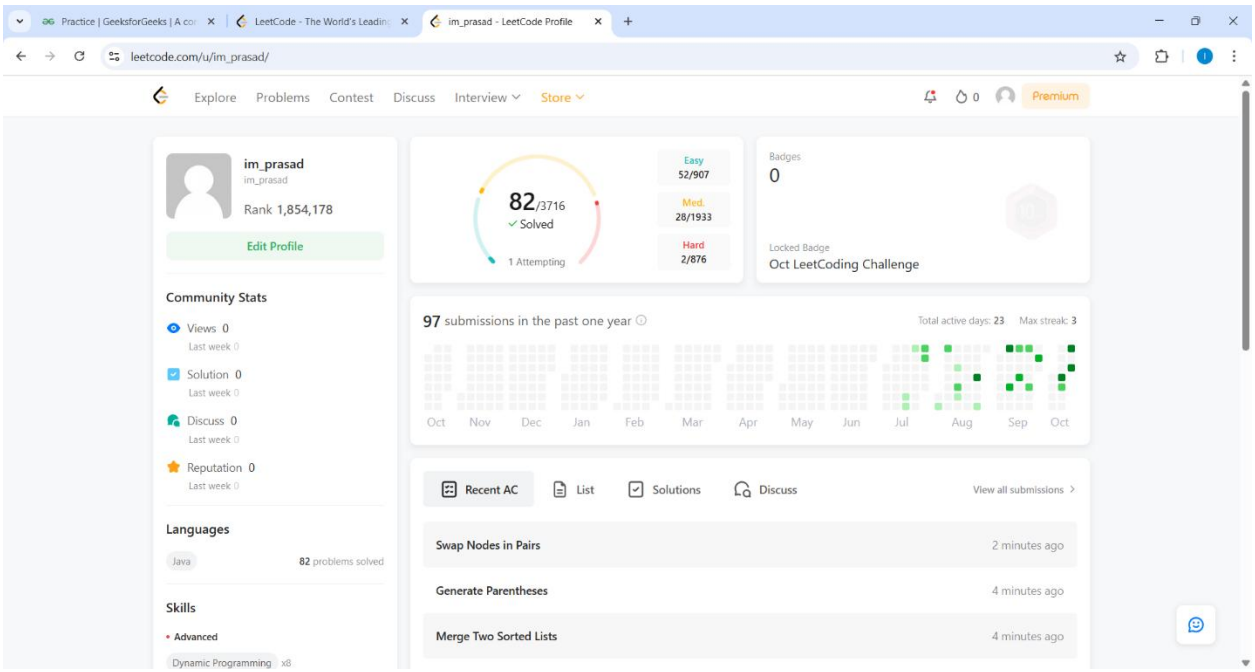
No. of Experiments Done:

Faculty Incharge of the Laboratory

Head of the Department

EXTERNAL EXAMINER


leela venkata durga prasad mot
geeksforgeeks.org/user/moturiprasad08/
Upto 40% Off on GfG Courses + Mock Interviews. Avail Offers Now!
Search...
Courses
Tutorials
Practice
Jobs
Switch to Dark Mode
Profile
Contributions
Saved Items
Colleges
Companies
Campus Mantri
Invite
Edit Profile
Account Settings
Edit Profile
moturiprasad08
Institution
Sri Vasavi Engineering College (SVEC) Tadepalligudem
522 Rank
Language Used
Java, C++
Current POTD Streak
STREAK
01/1537
days
Coding Score
772
Problem Solved
419
Contest Rating
—
Campus Mantri
pavana26
455 submissions in current year
Current


Practice | GeeksforGeeks | A co
LeetCode - The World's Leadin
im_prasad - LeetCode Profile
leetcode.com/u/im_prasad/
Explore
Problems
Contest
Discuss
Interview
Store
Premium
im_prasad
im_prasad
Rank 1,854,178
Edit Profile
Community Stats
Views 0
Last week 0
Solution 0
Last week 0
Discuss 0
Last week 0
Reputation 0
Last week 0
Languages
Java
82 problems solved
Skills
Advanced
Dynamic Programming x8
82/3716
Solved
1 Attempting
Easy
52/907
Med.
28/1933
Hard
2/876
Badges
0
Locked Badge
Oct LeetCoding Challenge
97 submissions in the past one year
Total active days: 23
Max streak: 3
Oct
Nov
Dec
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Recent AC
List
Solutions
Discuss
View all submissions
Swap Nodes in Pairs
2 minutes ago
Generate Parentheses
4 minutes ago
Merge Two Sorted Lists
4 minutes ago

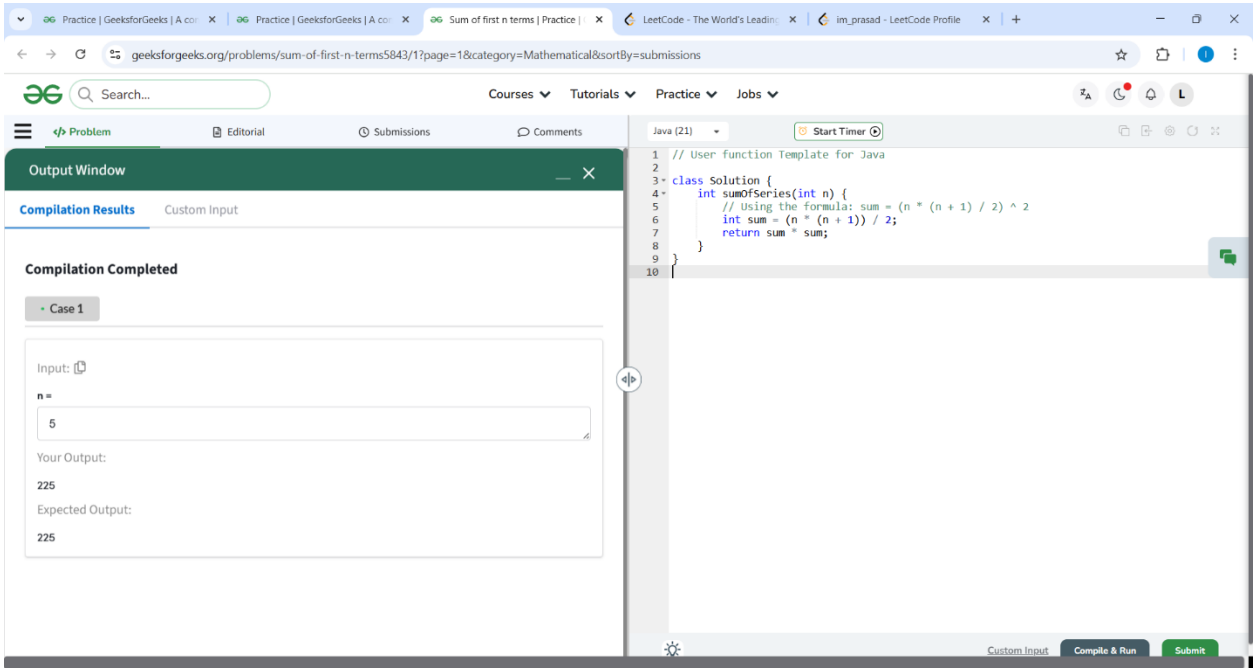
Programs on Matrices Problems			
47	Row-wise sorted matrices	61-62	
48	Transpose of the Matrix	63	
49	2D Dimensions	64	
50	Square Matrix	65	

Mathematical Problems

1. Given an integer **n,** calculate the sum of series $1^3 + 2^3 + 3^3 + 4^3 + \ldots$ till n-th term.

```
// User function Template for Java

class Solution {
    int sumOfSeries(int n) {
        // Using the formula: sum = (n * (n + 1) / 2) ^ 2
        int sum = (n * (n + 1)) / 2;
        return sum * sum;
    }
}
```

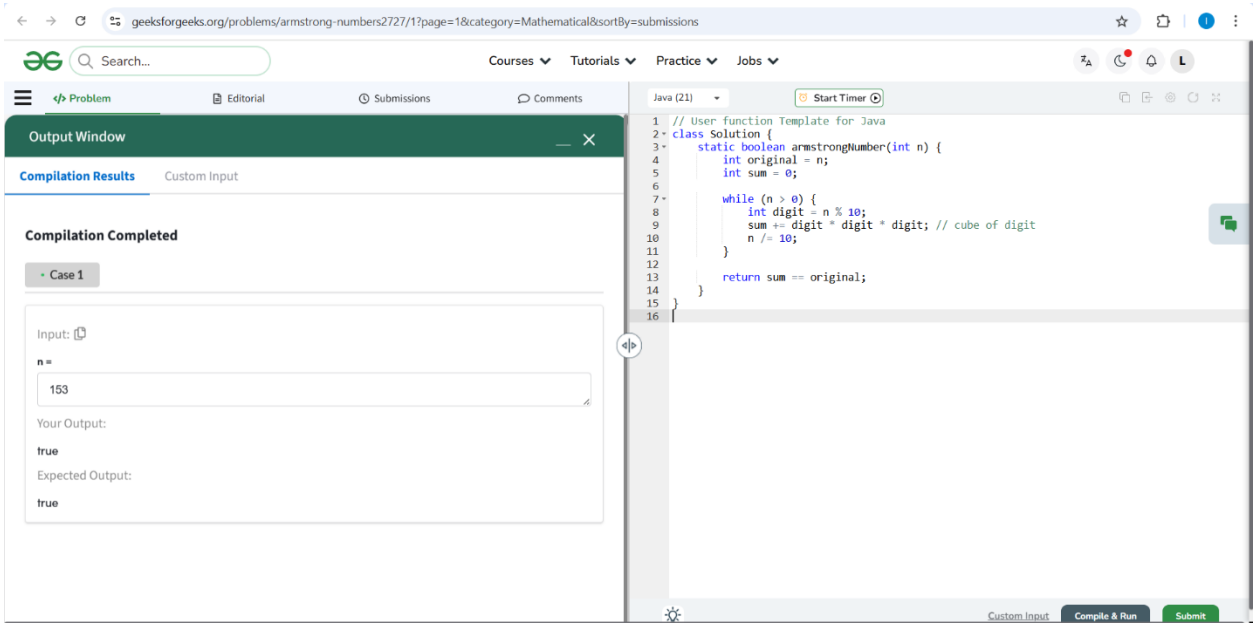
OUTPUT:



2. You are given a **3-digit** number **n**, Find whether it is an **Armstrong** number or not.

```
// User function Template for Java
class Solution {
    static boolean armstrongNumber(int n) {
        int original = n;
        int sum = 0;

        while (n > 0) {
            int digit = n % 10;
            sum += digit * digit * digit; // cube of digit
            n /= 10;
        }

        return sum == original;
    }
}
```

OUTPUT:

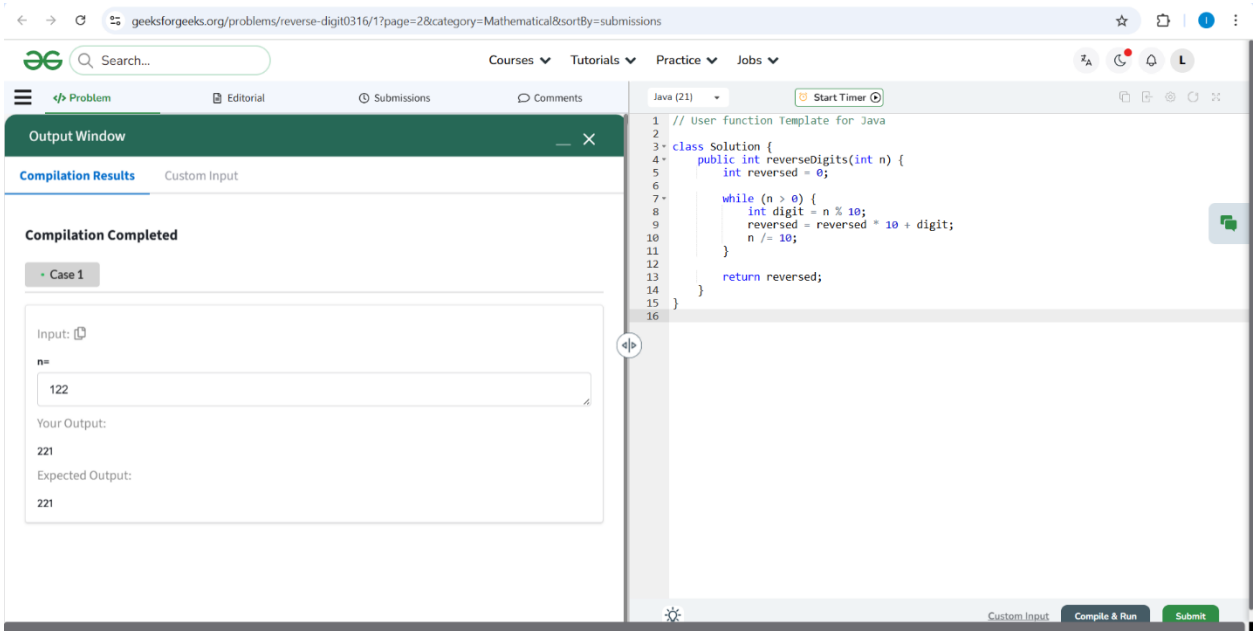


3. You are given an integer **n.** Your task is to reverse the digits, ensuring that the reversed number has no leading zeroes.

// User function Template for Java

```
class Solution {
    public int reverseDigits(int n) {
        int reversed = 0;

        while (n > 0) {
            int digit = n % 10;
            reversed = reversed * 10 + digit;
            n /= 10;
        }

        return reversed;
    }
}
```

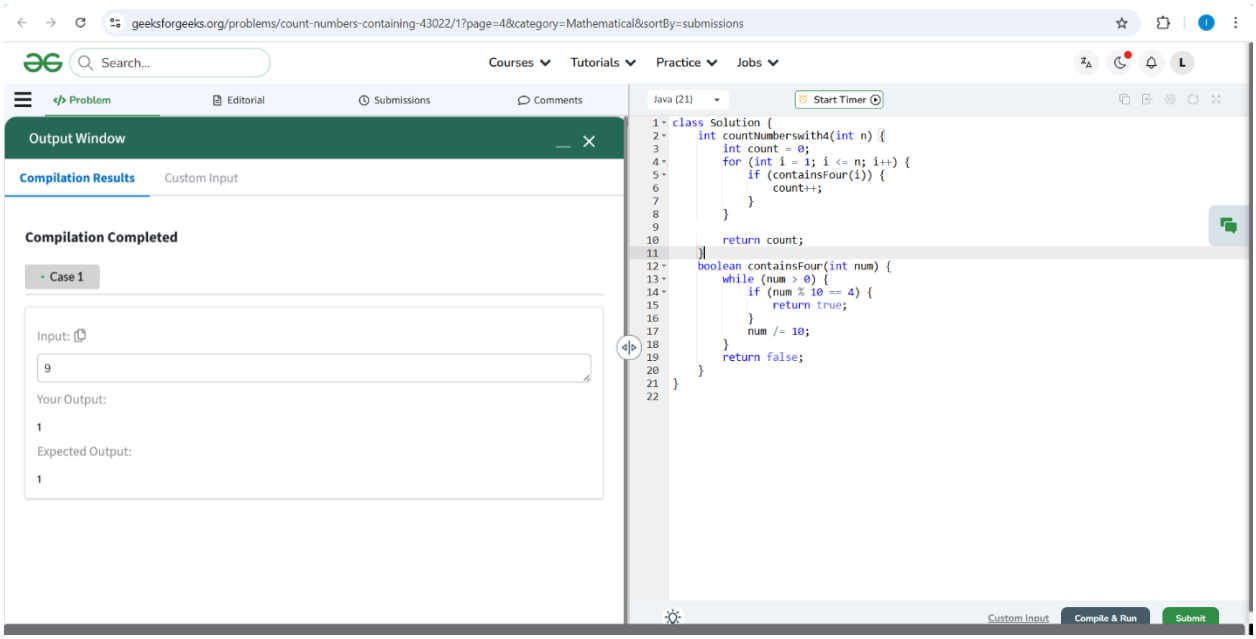
OUTPUT:



4.You are given a number **n**, Return the count of total numbers from **1** to **n** containing **4** as a digit.

```
class Solution {
    int countNumberswith4(int n) {
        int count = 0;
        for (int i = 1; i <= n; i++) {
            if (containsFour(i)) {
                count++;
            }
         return count;
    }
    boolean containsFour(int num) {
        while (num > 0) {
            if (num % 10 == 4) {
                return true;
            }
            num /= 10;
        }
        return false;
    }
}
```

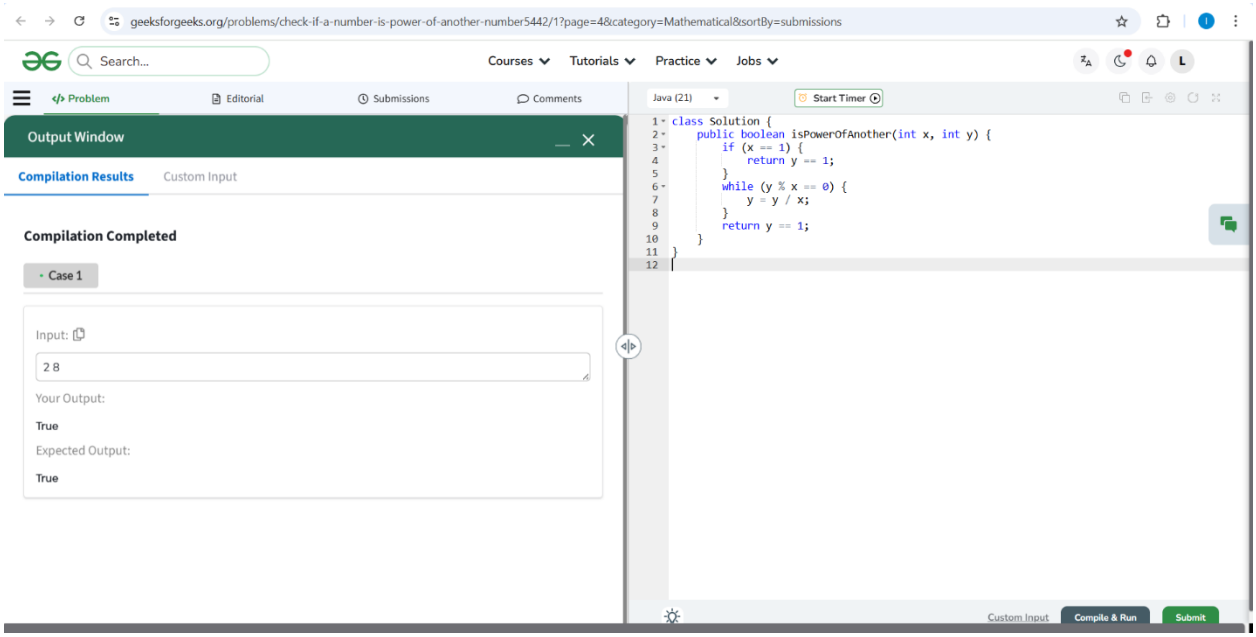
OUTPUT:


geeksforgeeks.org/problems/count-numbers-containing-43022/1?page=4&category=Mathematical&sortBy=submissions
Search...
Courses
Tutorials
Practice
Jobs
Problem
Editorial
Submissions
Comments
Output Window
Compilation Results
Custom Input
Compilation Completed
Case 1
Input:
9
Your Output:
1
Expected Output:
1
Java (21)
Start Timer
class Solution {
int countNumberswith4(int n) {
int count = 0;
for (int i = 1; i <= n; i++) {
if (containsFour(i)) {
count++;
}
}
return count;
}
boolean containsFour(int num) {
while (num > 0) {
if (num % 10 == 4) {
return true;
}
num /= 10;
}
return false;
}
}
Custom Input
Compile & Run
Submit

5. Given two positive integers **x** and **y**, determine if y is a power of x. If y is a power of x, return **True**. Otherwise, return **False**.

```
class Solution {
  public boolean isPowerOfAnother(int x, int y) {
    if (x == 1) {
      return y == 1;
    }
    while (y % x == 0) {
      y = y / x;
    }
    return y == 1;
  }
}
```

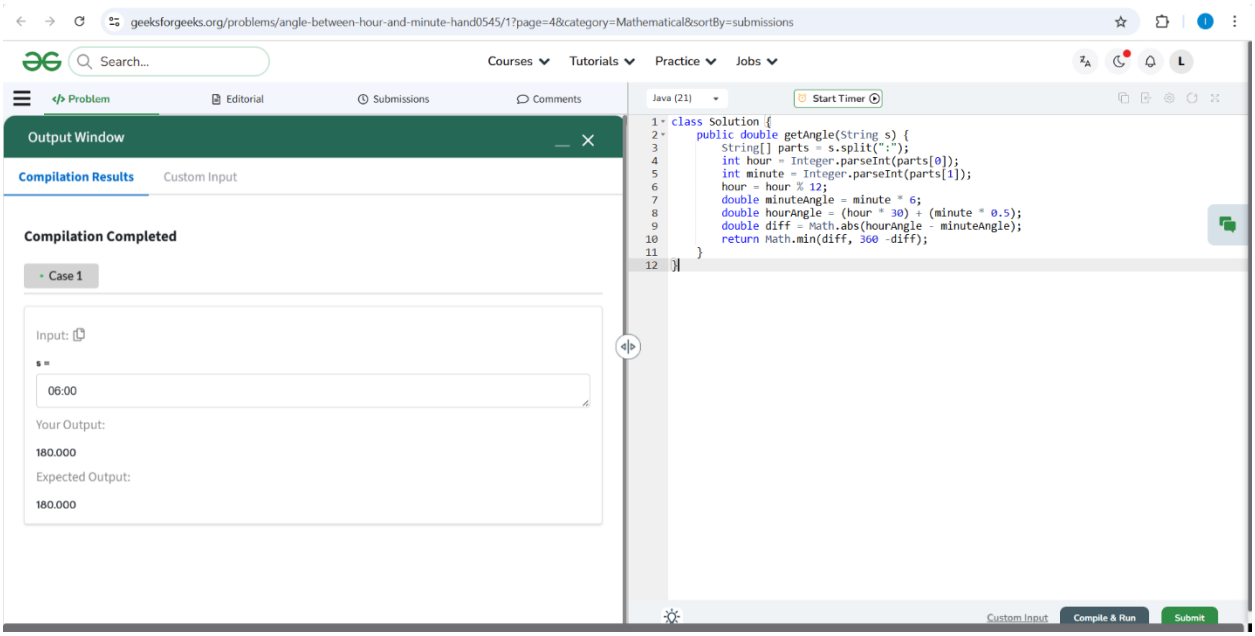
OUTPUT:



6. Given a string **s** representing time in 24-hour format **"HH:MM"**, compute the **smallest** angle in **degrees** between the hour and minute hands of an analog clock.

```
class Solution {
    public double getAngle(String s) {
        String[] parts = s.split(":");
        int hour = Integer.parseInt(parts[0]);
        int minute = Integer.parseInt(parts[1]);
        hour = hour % 12;
        double minuteAngle = minute * 6;
        double hourAngle = (hour * 30) + (minute * 0.5);
        double diff = Math.abs(hourAngle - minuteAngle);
        return Math.min(diff, 360 -diff);
    }
}
```

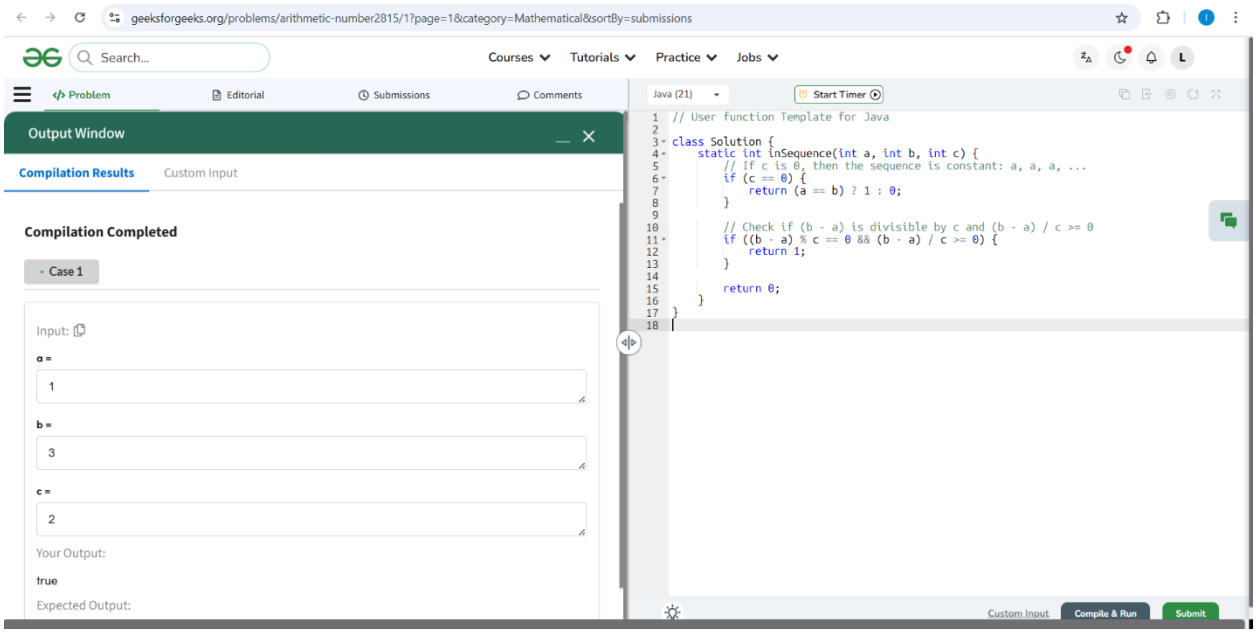
OUTPUT:



7. Given three integers **'a'** denotes the first term of an arithmetic sequence, **'c'** denotes the common difference of an arithmetic sequence and an integer **'b'**. you need to tell whether 'b' exists in the arithmetic sequence or not. Return 1 if b is present in the sequence. Otherwise, returns 0.

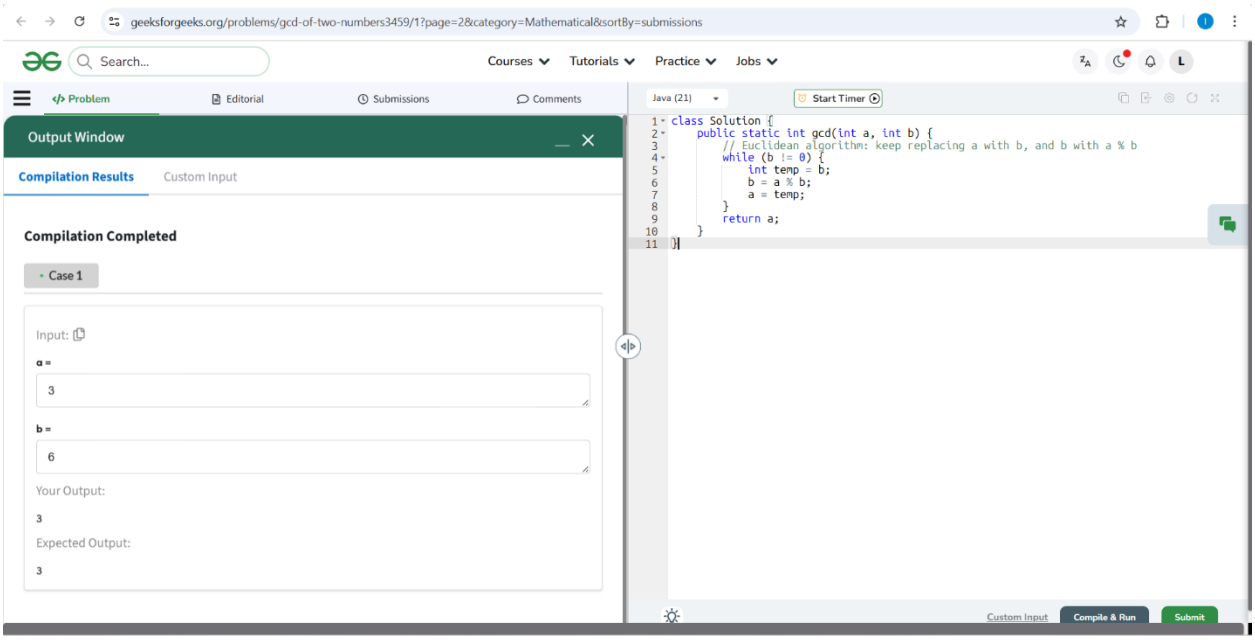
```
// User function Template for Java

class Solution {
    static int inSequence(int a, int b, int c) {
        // If c is 0, then the sequence is constant: a, a, a, ...
        if (c == 0) {
            return (a == b) ? 1 : 0;
        }

        // Check if (b - a) is divisible by c and (b - a) / c >= 0
        if ((b - a) % c == 0 && (b - a) / c >= 0) {
            return 1;
        }

        return 0;
    }
}
```

OUTPUT:



8. Given two positive integers **a** and **b**, find GCD of a and b.

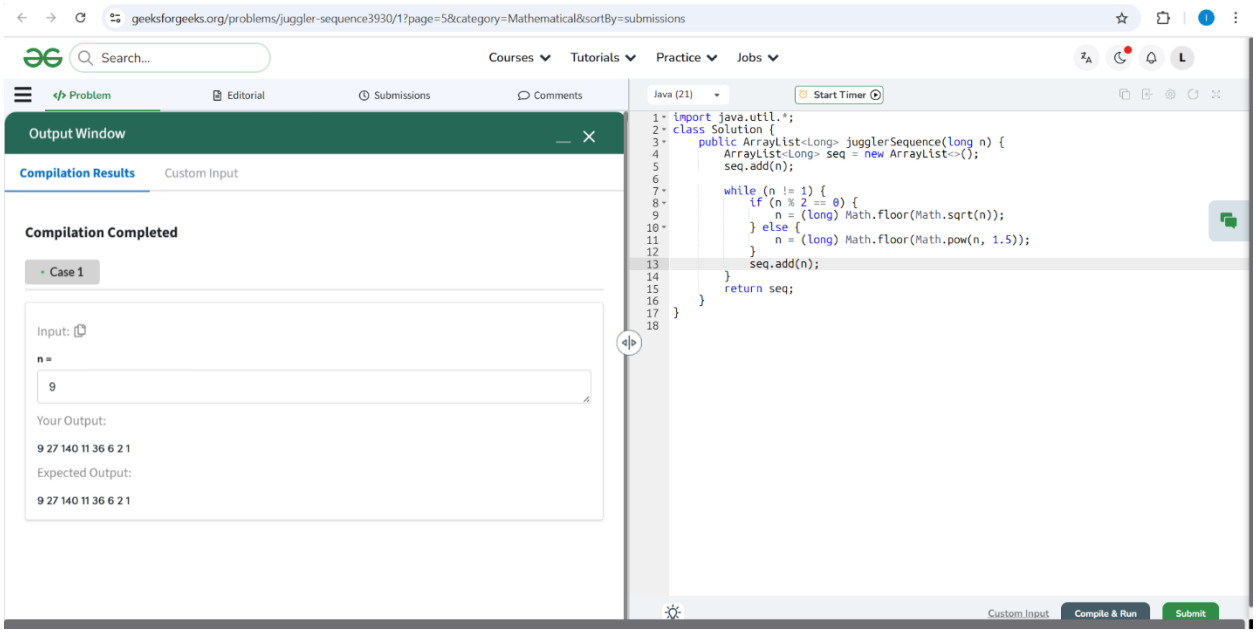
```
class Solution {
    public static int gcd(int a, int b) {
        // Euclidean algorithm: keep replacing a with b, and b with a % b
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }
    }OUTPUT:
```



9. Juggler Sequence is a series of integers in which the first term starts with a positive integer number *a* and the remaining terms are generated from the immediate previous term using the below recurrence relation:

```
import java.util.*;
class Solution {
    public ArrayList<Long> jugglerSequence(long n) {
        ArrayList<Long> seq = new ArrayList<>();
        seq.add(n);

        while (n != 1) {
            if (n % 2 == 0) {
                n = (long) Math.floor(Math.sqrt(n));
            } else {
                n = (long) Math.floor(Math.pow(n, 1.5));
            }
            seq.add(n);
        }
        return seq;
    }
}
```

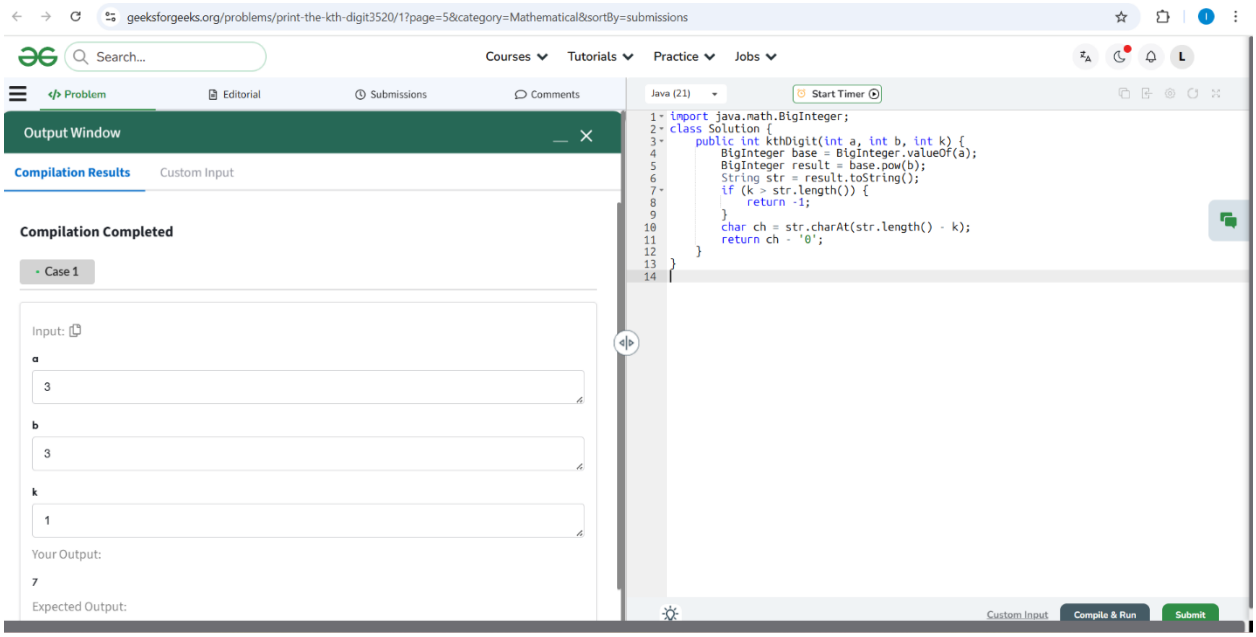
OUTPUT:



10. Given two numbers **a** and **b**, find **kth** digit from right of **a^{b}.**

```
import java.math.BigInteger;
class Solution {
   public int kthDigit(int a, int b, int k) {
      BigInteger base = BigInteger.valueOf(a);
      BigInteger result = base.pow(b);
      String str = result.toString();
      if (k > str.length()) {
         return -1;
      }
      char ch = str.charAt(str.length() - k);
      return ch - '0';
   }
}
```

OUTPUT:

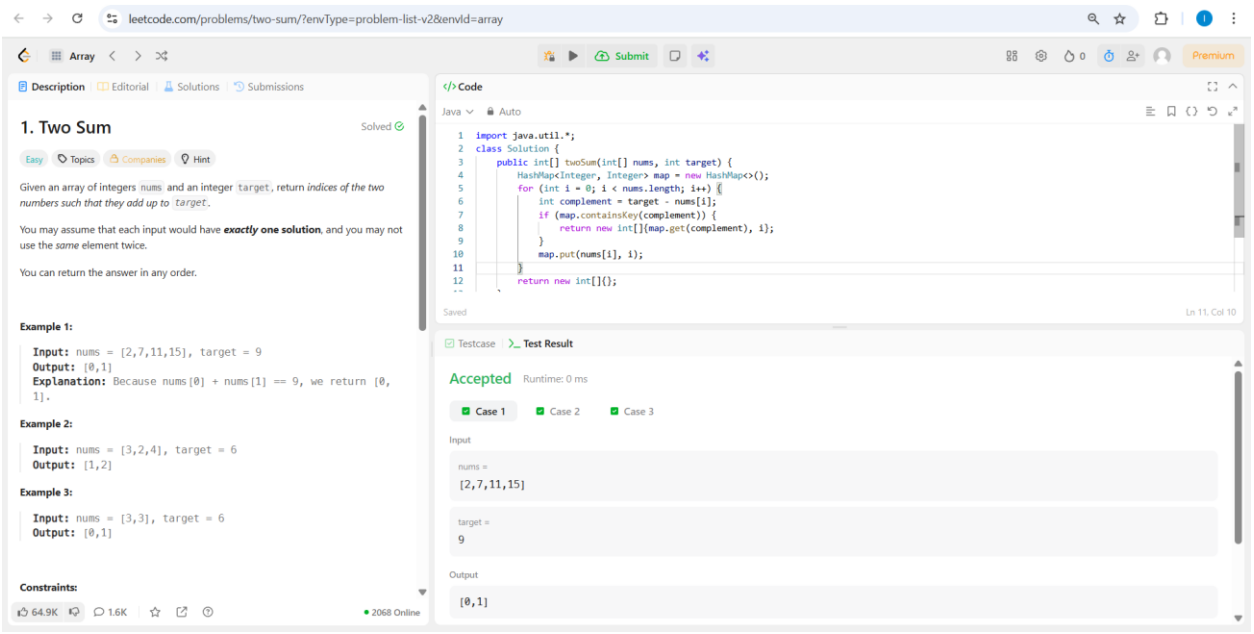


Programs on Array problems

11. Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

```
import java.util.*;
class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[]{map.get(complement), i};
            }
            map.put(nums[i], i);
        }
        return new int[]{};
    }
}
```

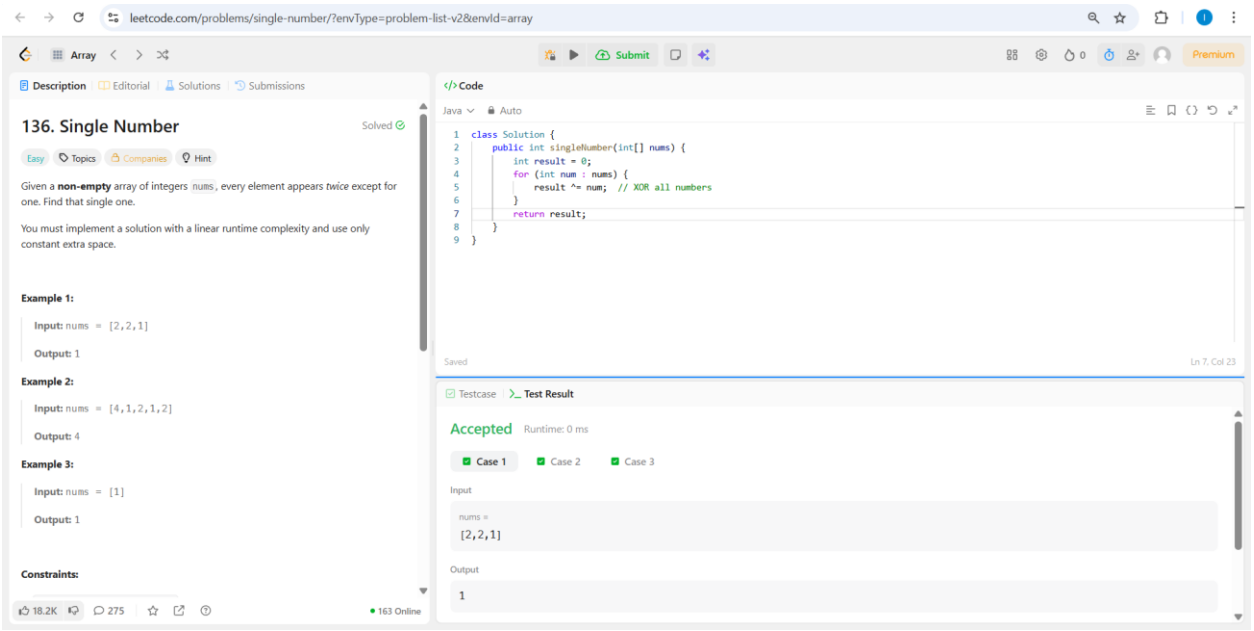
OUTPUT:



12. Given a **non-empty** array of integers nums, every element appears *twice* except for one. Find that single one.
You must implement a solution with a linear runtime complexity and use only constant extra space.

```
class Solution {
    public int singleNumber(int[] nums) {
        int result = 0;
        for (int num : nums) {
            result ^= num;  // XOR all numbers
        }
        return result;
    }
}
```

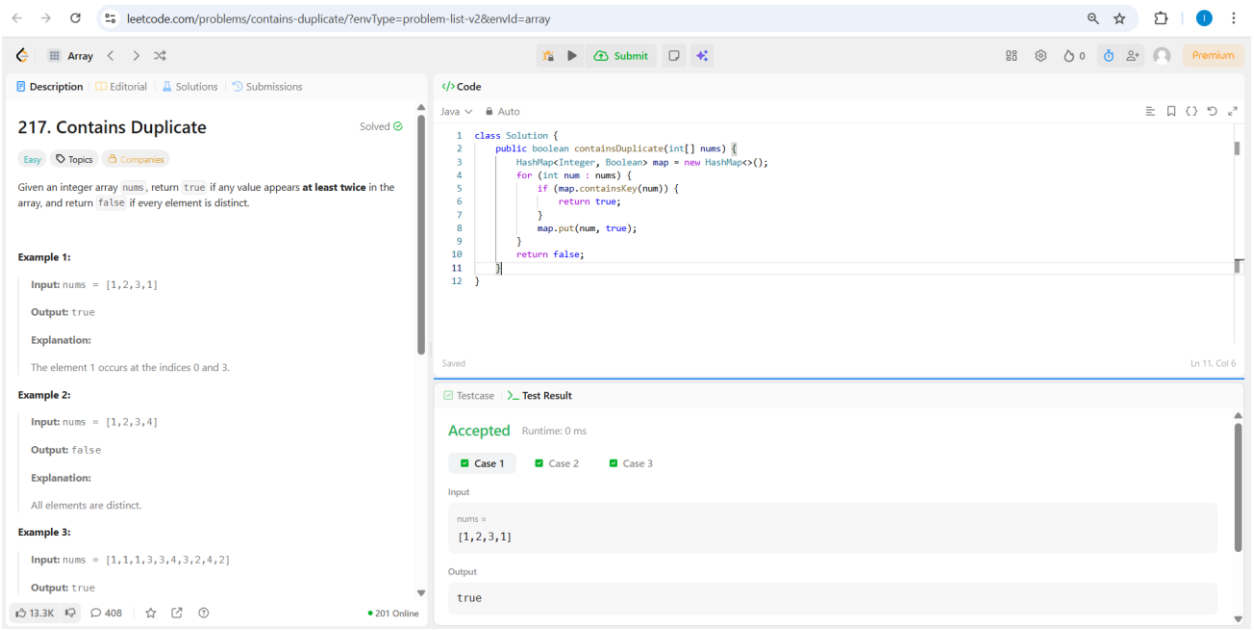
OUTPUT:



13. Given an integer array nums, return true if any value appears **at least twice** in the array, and return false if every element is distinct.

```
class Solution {
    public boolean containsDuplicate(int[] nums) {
        HashMap<Integer, Boolean> map = new HashMap<>();
        for (int num : nums) {
            if (map.containsKey(num)) {
                return true;
            }
            map.put(num, true);
        }
        return false;
    }
}
```

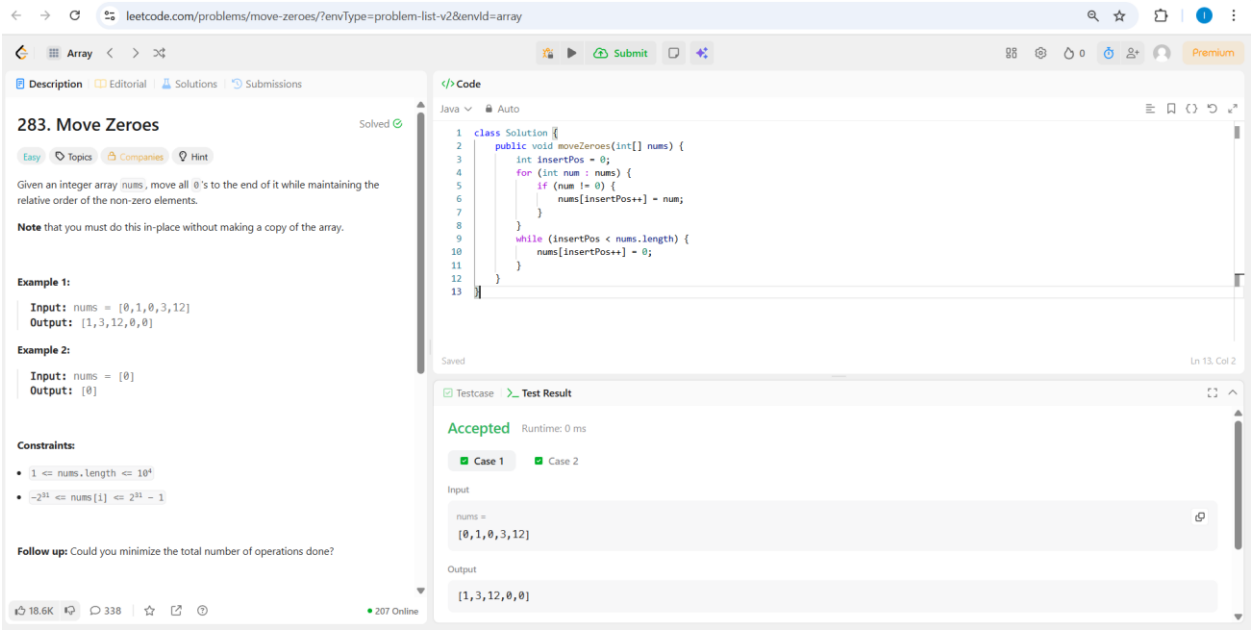
OUTPUT:



14. Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

```
class Solution {
    public void moveZeroes(int[] nums) {
        int insertPos = 0;
        for (int num : nums) {
            if (num != 0) {
                nums[insertPos++] = num;
            }
        }
        while (insertPos < nums.length) {
            nums[insertPos++] = 0;
        }
    }
}
```

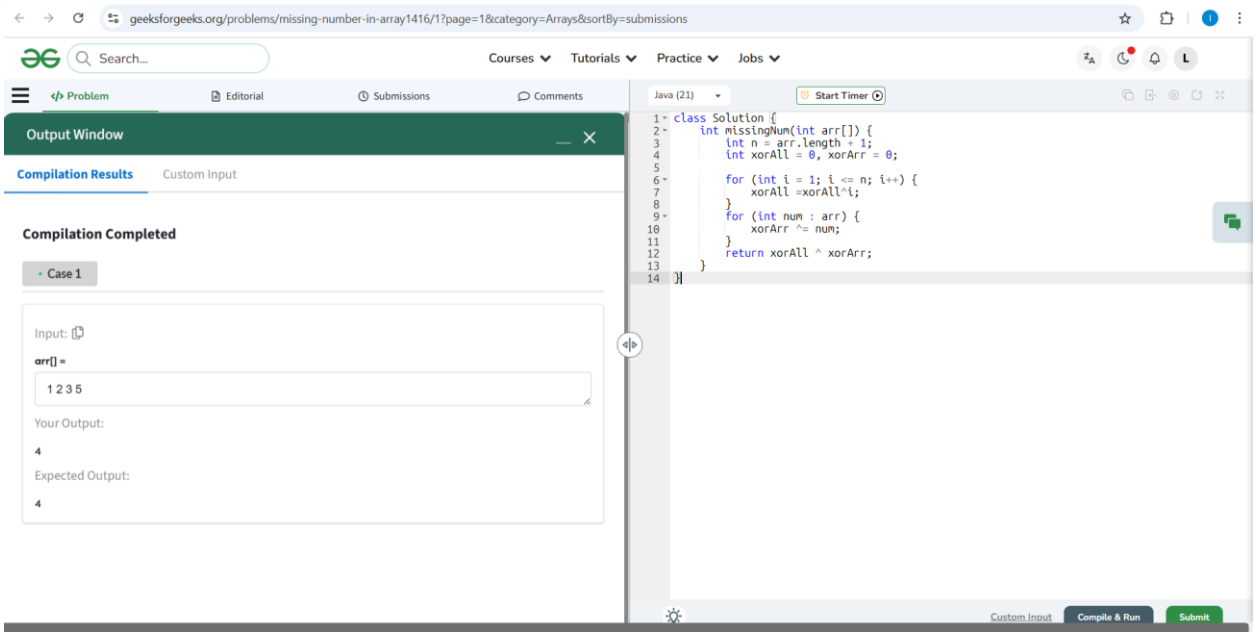
OUTPUT :



15. You are given an array **arr[]** of size **n - 1** that contains **distinct integers** in the range from 1 to n (inclusive). This array represents a permutation of the integers from 1 to n with **one element missing**. Your task is to identify and return the **missing element**.

```
class Solution {
    int missingNum(int arr[]) {
        int n = arr.length + 1;
        int xorAll = 0, xorArr = 0;

        for (int i = 1; i <= n; i++) {
            xorAll =xorAll^i;
        }
        for (int num : arr) {
            xorArr ^= num;
        }
        return xorAll ^ xorArr;
    }
}
```

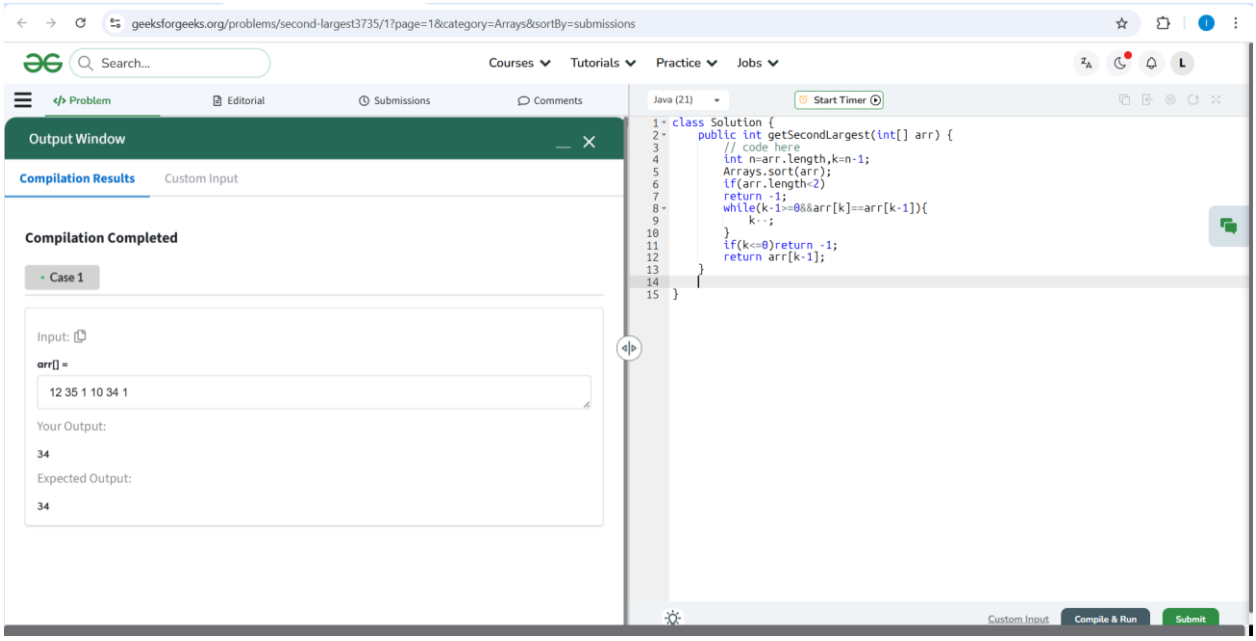
OUTPUT:



16. Given an array of **positive** integers **arr[]**, return the **second largest** element from the array. If the second largest element doesn't exist then return **-1.**

```
class Solution {
    public int getSecondLargest(int[] arr) {
        // code here
        int n=arr.length,k=n-1;
        Arrays.sort(arr);
        if(arr.length<2)
        return -1;
        while(k-1>=0&&arr[k]==arr[k-1]){
            k--;
        }
        if(k<=0)return -1;
        return arr[k-1];
    }
}
```

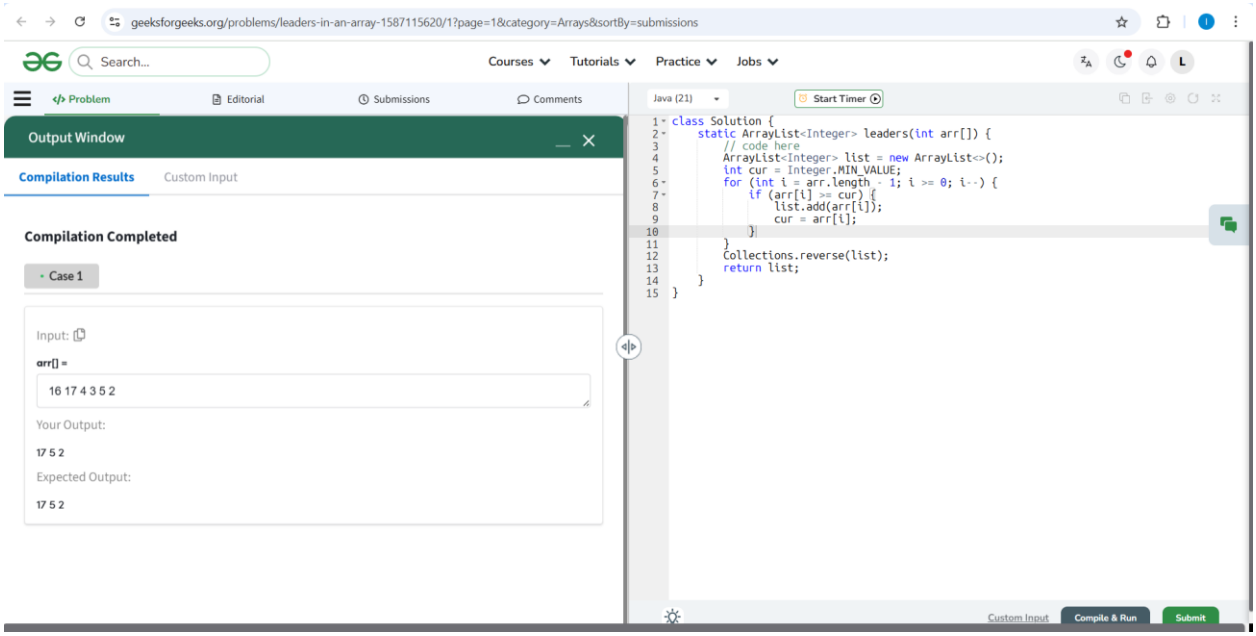
OUTPUT :



17. You are given an array **arr** of positive integers. Your task is to find all the leaders in the array. An element is considered a leader if it is greater than or equal to all elements to its right. The rightmost element is always a leader.

```
class Solution {
    static ArrayList<Integer> leaders(int arr[]) {
        // code here
        ArrayList<Integer> list = new ArrayList<>();
        int cur = Integer.MIN_VALUE;
        for (int i = arr.length - 1; i >= 0; i--) {
            if (arr[i] >= cur) {
                list.add(arr[i]);
                cur = arr[i];
            }
        }
        Collections.reverse(list);
        return list;
    }
}
```

OUTPUT:

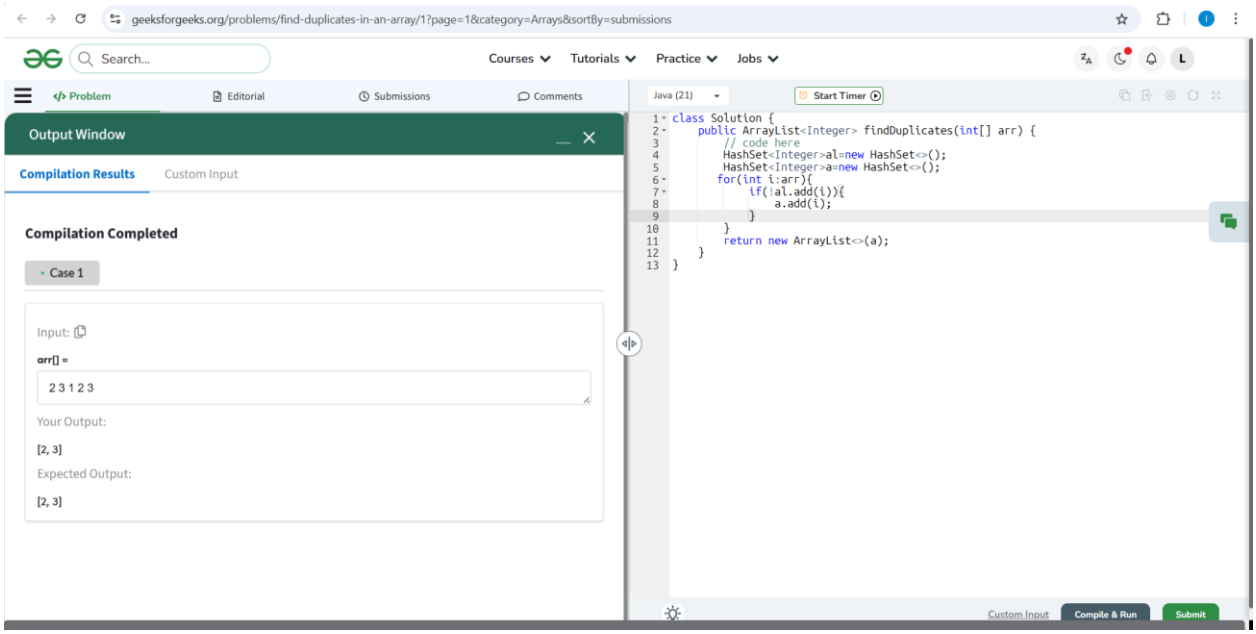


18. Given an array **arr[]** of size **n**, containing elements from the range **1** to **n**, and each element appears at most **twice**, return an array of all the integers that appears twice.

You can return the elements in any order but the driver code will print them in sorted order.

```
class Solution {
    public ArrayList<Integer> findDuplicates(int[] arr) {
        // code here
        HashSet<Integer>al=new HashSet<>();
        HashSet<Integer>a=new HashSet<>();
       for(int i:arr){
            if(!al.add(i)){
                a.add(i);
            }
        }
        return new ArrayList<>(a);
    }
}
```

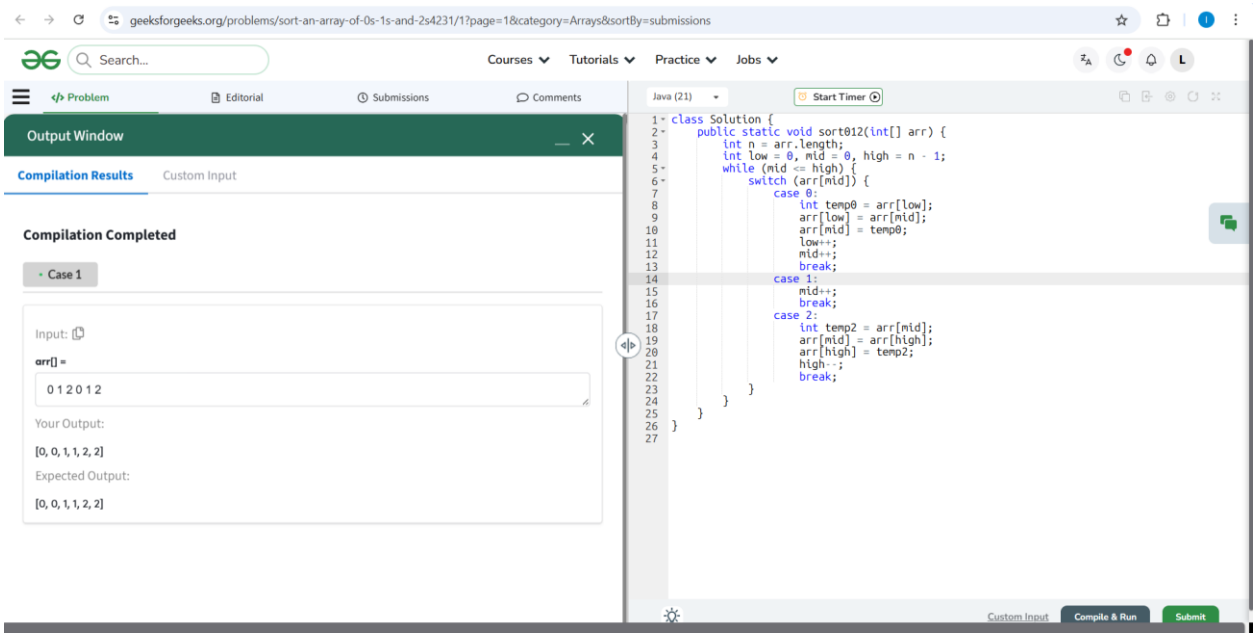
OUTPUT :



19. Given an array **arr[]** containing only **0s, 1s, and 2s.** Sort the array in ascending order.
Note: You need to solve this problem without utilizing the built-in sort function.

```
class Solution {
    public static void sort012(int[] arr) {
        int n = arr.length;
        int low = 0, mid = 0, high = n - 1;
        while (mid <= high) {
            switch (arr[mid]) {
                case 0:
                    int temp0 = arr[low];
                    arr[low] = arr[mid];
                    arr[mid] = temp0;
                    low++;
                    mid++;
                    break;
                case 1:
                    mid++;
                    break;
                case 2:
                    int temp2 = arr[mid];
                    arr[mid] = arr[high];
                    arr[high] = temp2;
                    high--;
                    break;
            }
        }
    }
}
```

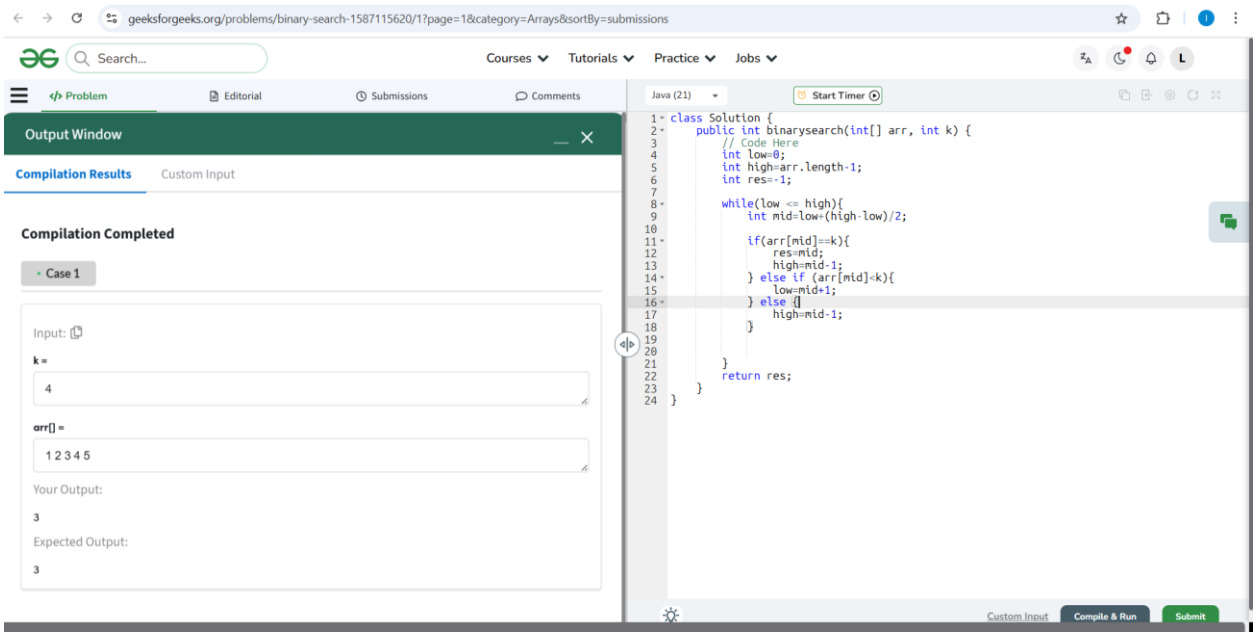
OUTPUT:



20. Given a sorted array **arr[]** and an integer **k**, find the position(0-based indexing) at which k is present in the array using binary search. If k doesn't exist in arr[] return **-1**.
Note: If multiple occurrences are there, please return the smallest index.

```
class Solution {
    public int binarysearch(int[] arr, int k) {
        // Code Here
        int low=0;
        int high=arr.length-1;
        int res=-1;

        while(low <= high){
            int mid=low+(high-low)/2;

            if(arr[mid]==k){
                res=mid;
                high=mid-1;
            } else if (arr[mid]<k){
                low=mid+1;
            } else {
                high=mid-1;
            }


        }
        return res;
    }
}
```

OUTPUT:

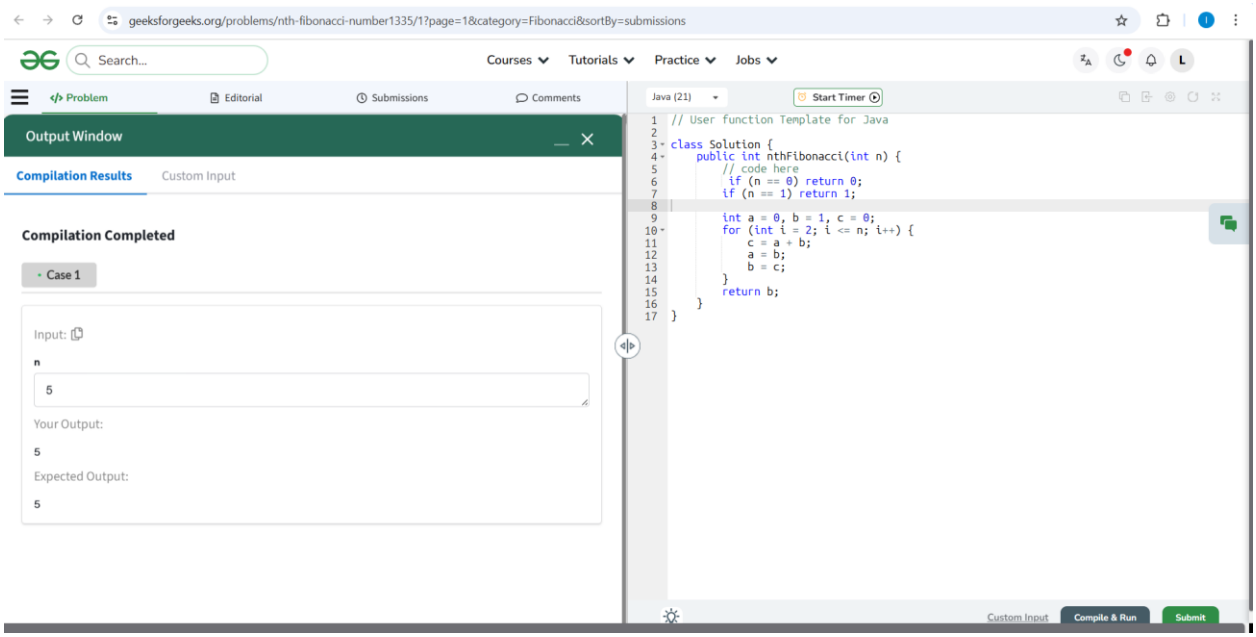


Programs on Fibonacci Problems

21. Given a non-negative integer **n**, your task is to find the **nth Fibonacci number**.
The Fibonacci sequence is a sequence where the next term is the sum of the previous two terms. The first two terms of the Fibonacci sequence are 0 followed by 1. The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21

```
// User function Template for Java

class Solution {
   public int nthFibonacci(int n) {
       // code here
        if (n == 0) return 0;
       if (n == 1) return 1;

       int a = 0, b = 1, c = 0;
       for (int i = 2; i <= n; i++) {
          c = a + b;
          a = b;
          b = c;
       }
       return b;
   }
}
```

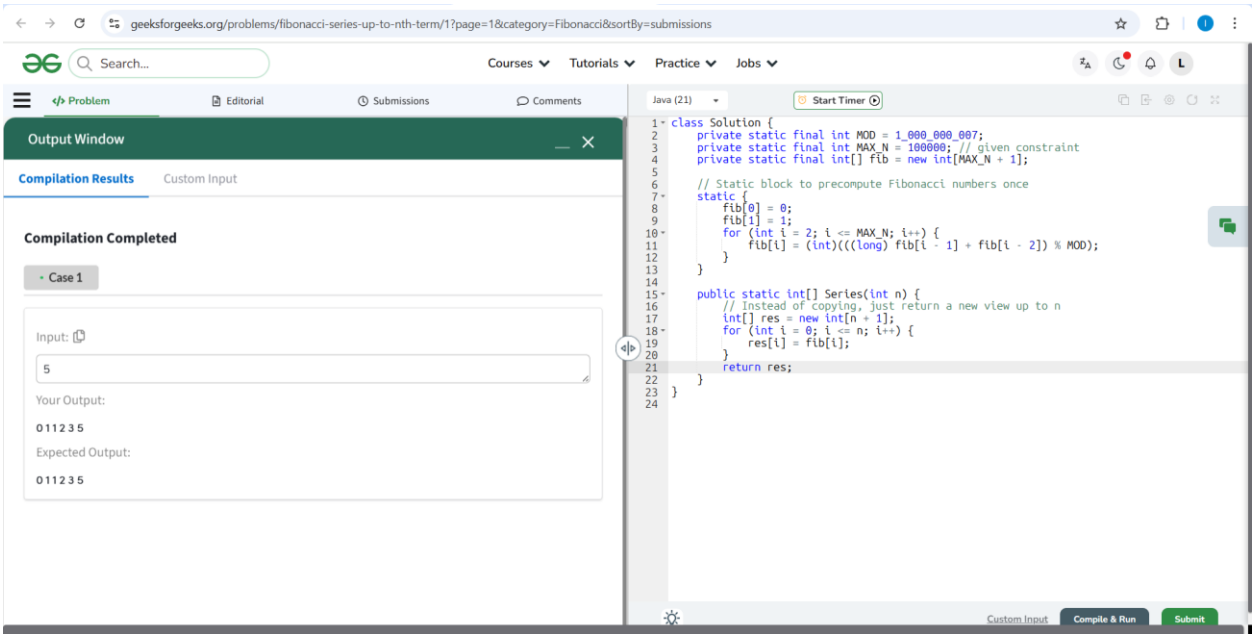
OUTPUT :



22. You are given an integer **n**, return the fibonacci series till the **nth**(0-based indexing) term. Since the terms can become very large return the terms **modulo 10^9+7**.

```
class Solution {
    private static final int MOD = 1_000_000_007;
    private static final int MAX_N = 100000; // given constraint
    private static final int[] fib = new int[MAX_N + 1];
    // Static block to precompute Fibonacci numbers once
    static {
        fib[0] = 0;
        fib[1] = 1;
        for (int i = 2; i <= MAX_N; i++) {
            fib[i] = (int)(((long) fib[i - 1] + fib[i - 2]) % MOD);
        }
    }
    public static int[] Series(int n) {
        // Instead of copying, just return a new view up to n
        int[] res = new int[n + 1];
        for (int i = 0; i <= n; i++) {
            res[i] = fib[i];
        }
        return res;
    }
}
```

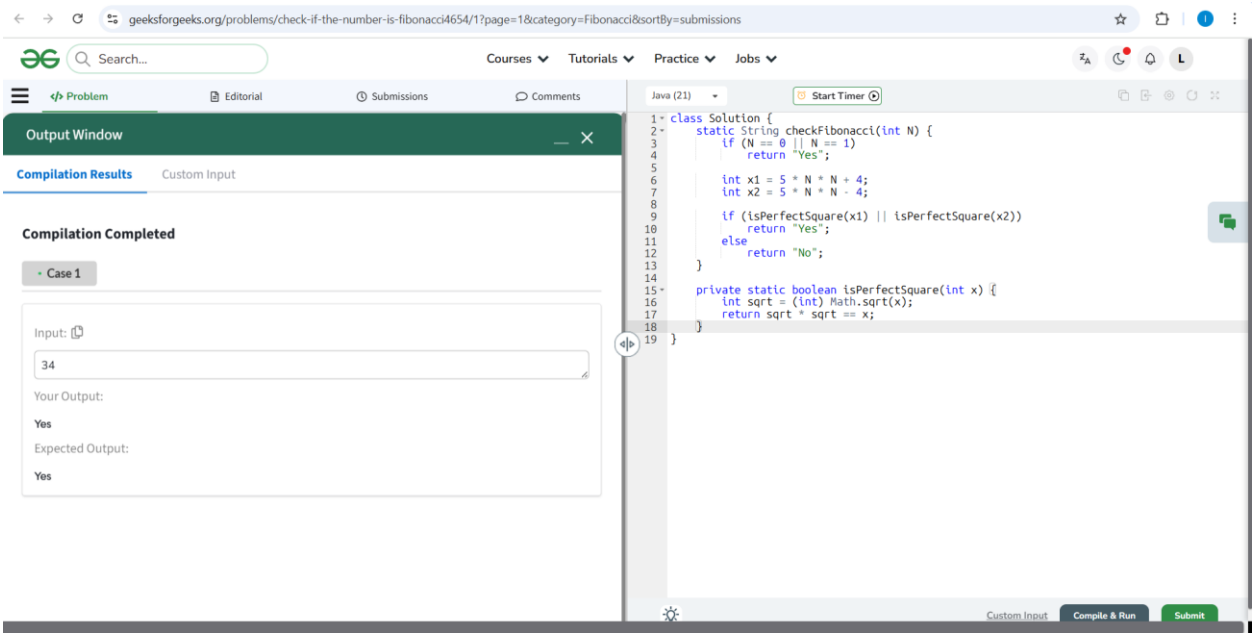
OUTPUT:


geeksforgeeks.org/problems/fibonacci-series-up-to-nth-term/1?page=1&category=Fibonacci&sortBy=submissions
Search...
Courses
Tutorials
Practice
Jobs
Problem
Editorial
Submissions
Comments
Output Window
Compilation Results
Custom Input
Compilation Completed
Case 1
Input:
5
Your Output:
0 1 1 2 3 5
Expected Output:
0 1 1 2 3 5
Java (21)
Start Timer
class Solution {
private static final int MOD = 1_000_000_007;
private static final int MAX_N = 100000; // given constraint
private static final int[] fib = new int[MAX_N + 1];
// Static block to precompute Fibonacci numbers once
static {
fib[0] = 0;
fib[1] = 1;
for (int i = 2; i <= MAX_N; i++) {
fib[i] = (int)(((long) fib[i - 1] + fib[i - 2]) % MOD);
}
}
public static int[] Series(int n) {
// Instead of copying, just return a new view up to n
int[] res = new int[n + 1];
for (int i = 0; i <= n; i++) {
res[i] = fib[i];
}
return res;
}
}
Custom Input
Compile & Run
Submit

23. Check if a given number N is the Fibonacci number. A Fibonacci number is a number that occurs in the Fibonacci series.

```
class Solution {
    static String checkFibonacci(int N) {
        if (N == 0 || N == 1)
            return "Yes";

        int x1 = 5 * N * N + 4;
        int x2 = 5 * N * N - 4;

        if (isPerfectSquare(x1) || isPerfectSquare(x2))
            return "Yes";
        else
            return "No";
    }

    private static boolean isPerfectSquare(int x) {
        int sqrt = (int) Math.sqrt(x);
        return sqrt * sqrt == x;
    }
}
```

OUTPUT:



24. The **Fibonacci numbers**, commonly denoted F(n) form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

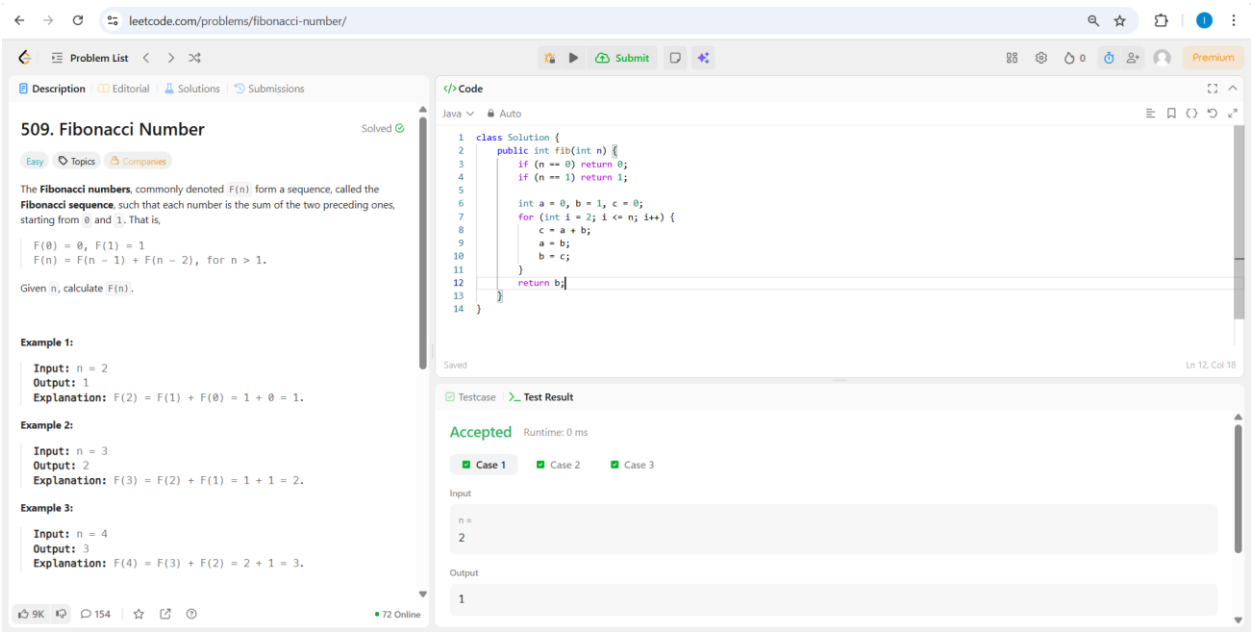
F(0) = 0, F(1) = 1

F(n) = F(n - 1) + F(n - 2), for n > 1.

Given n, calculate F(n).

```
class Solution {
    public int fib(int n) {
        if (n == 0) return 0;
        if (n == 1) return 1;

        int a = 0, b = 1, c = 0;
        for (int i = 2; i <= n; i++) {
            c = a + b;
            a = b;
            b = c;
        }
        return b;
    }
}
```

OUTPUT:

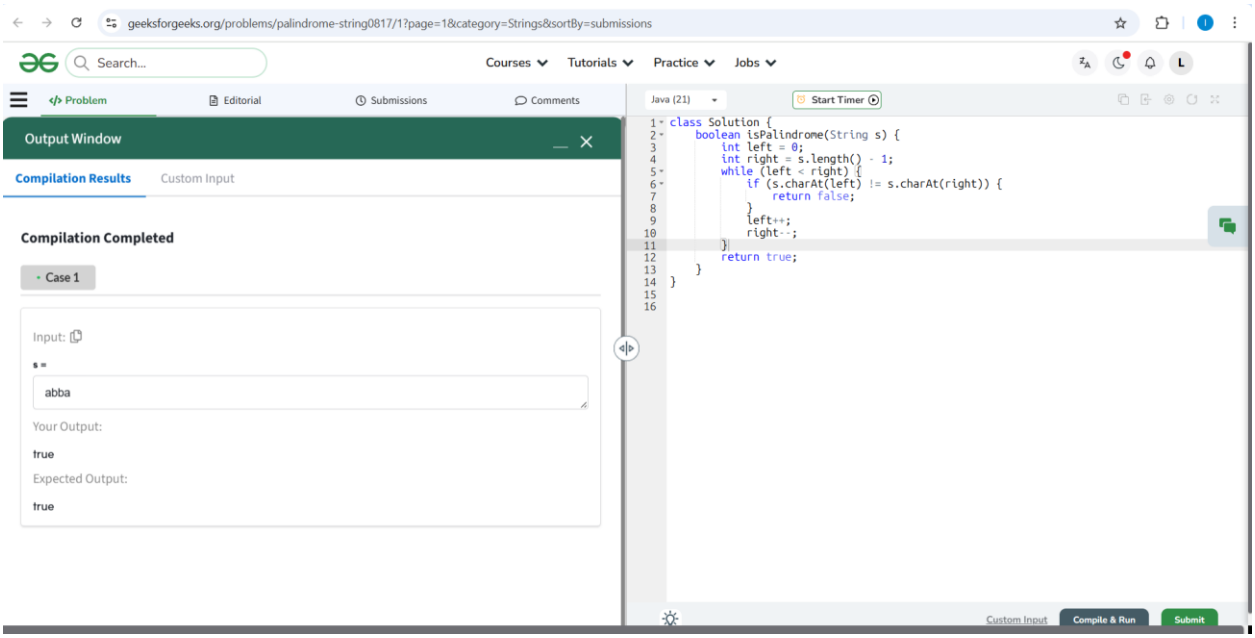


Programs on String Problems

25. You are given a string **s**. Your task is to determine if the string is a **palindrome**. A string is considered a palindrome if it reads the same forwards and backwards.

```
class Solution {
    boolean isPalindrome(String s) {
        int left = 0;
        int right = s.length() - 1;
        while (left < right) {
            if (s.charAt(left) != s.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }
}
```

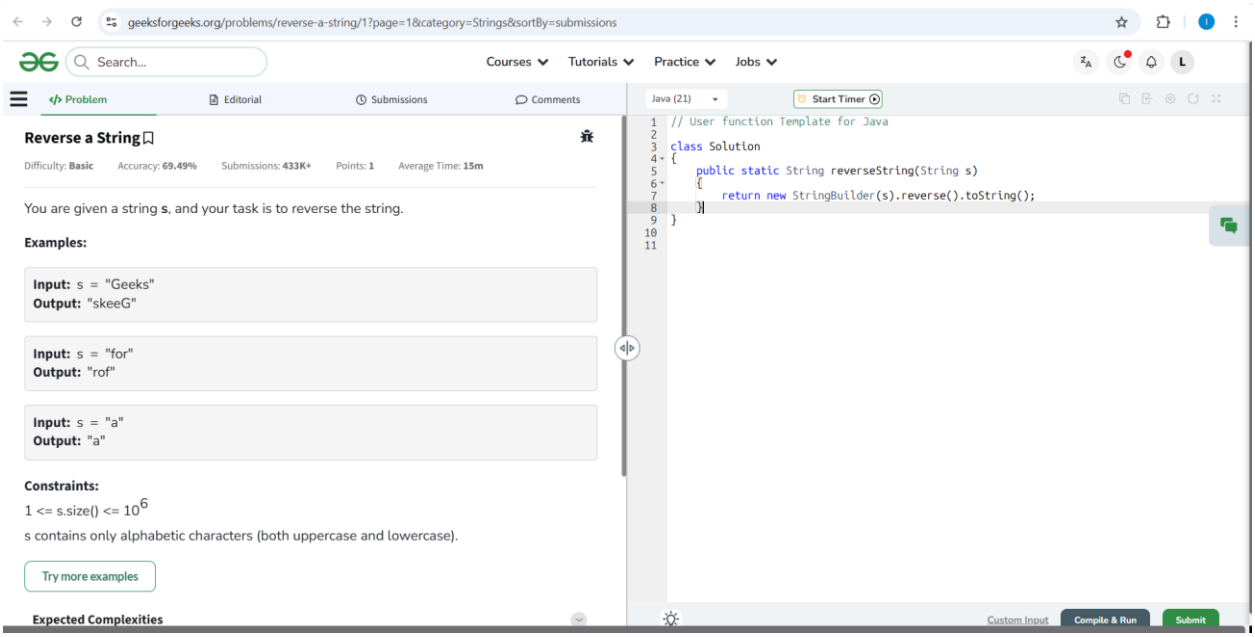
OUTPUT:



26. You are given a string **s**, and your task is to reverse the string.

```
// User function Template for Java

class Solution
{
   public static String reverseString(String s)
   {
      return new StringBuilder(s).reverse().toString();
   }
}
```

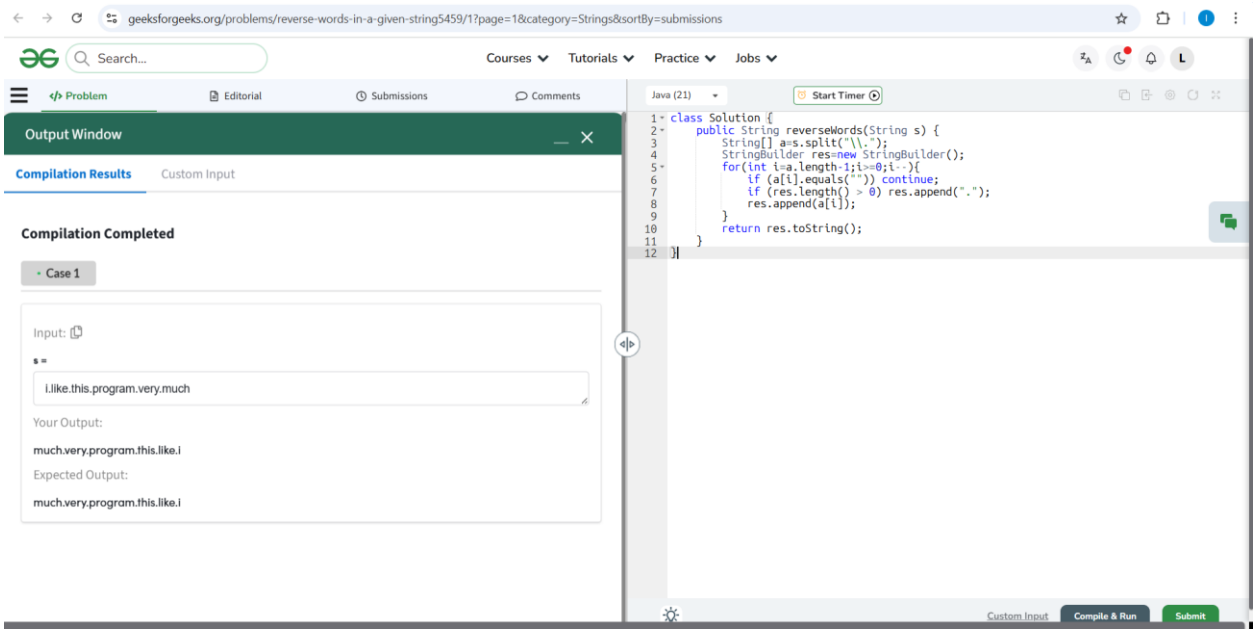
OUTPUT:



27. Given a string **s**, **reverse** the string without reversing its **individual words**. Words are separated by **dots(.)**.
Note: The string may contain leading or trailing dots(.) or multiple dots(.) between two words. The returned string should only have a single dot(.) separating the words, and **no extra dots** should be included.

```
class Solution {
    public String reverseWords(String s) {
        String[] a=s.split("\\.");
        StringBuilder res=new StringBuilder();
        for(int i=a.length-1;i>=0;i--){
            if (a[i].equals("")) continue;
            if (res.length() > 0) res.append(".");
            res.append(a[i]);
        }
        return res.toString();
    }
}
```

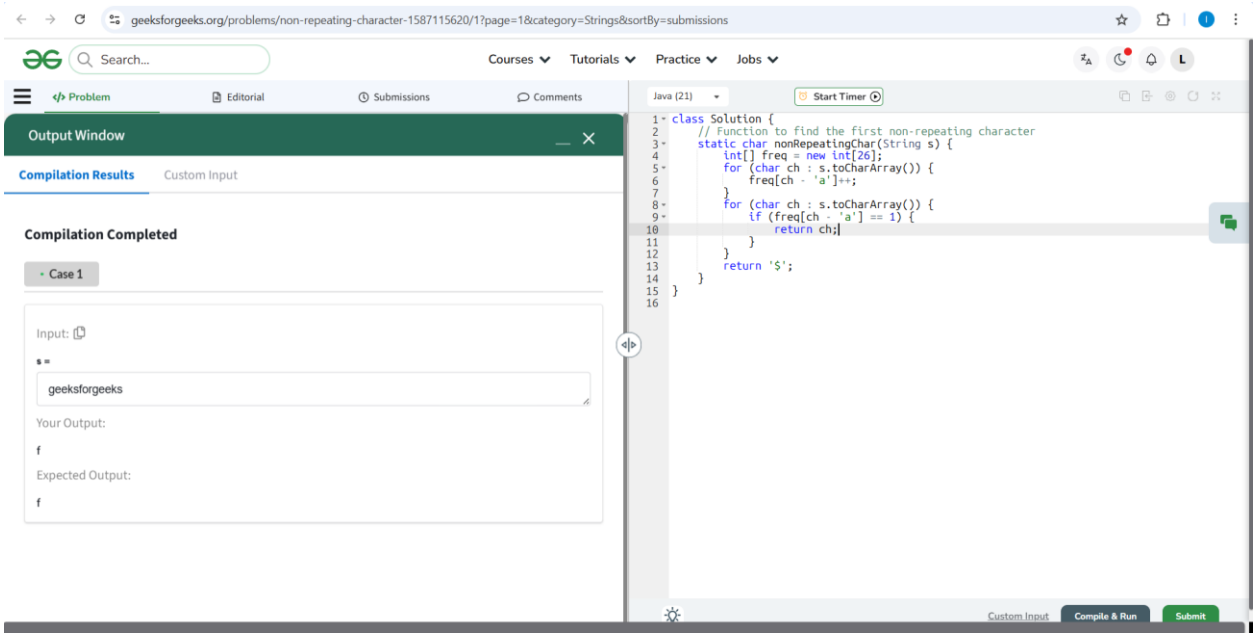
OUTPUT:



28. Given a string **s** consisting of **lowercase** English Letters. return the first non-repeating character in **s**. If there is no non-repeating character, return **'$'**.

```
class Solution {
    // Function to find the first non-repeating character
    static char nonRepeatingChar(String s) {
        int[] freq = new int[26];
        for (char ch : s.toCharArray()) {
            freq[ch - 'a']++;
        }
        for (char ch : s.toCharArray()) {
            if (freq[ch - 'a'] == 1) {
                return ch;
            }
        }
        return '$';
    }
}
```

OUTPUT:

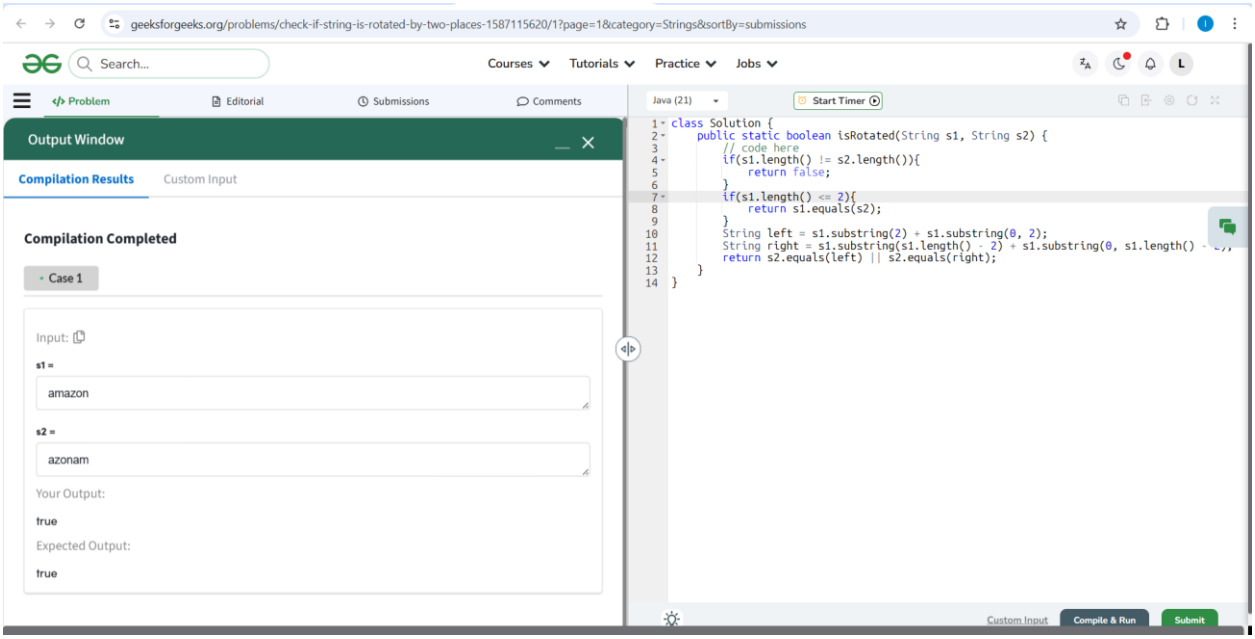


29. Given two strings **s1** and **s2**. Return true if the string **s2** can be obtained by rotating (**in any direction**) string **s1** by **exactly 2** places, otherwise, false.

Note: Both rotations should be performed in same direction chosen initially.

```
class Solution {
    public static boolean isRotated(String s1, String s2) {
        // code here
        if(s1.length() != s2.length()){
            return false;
        }
        if(s1.length() <= 2){
            return s1.equals(s2);
        }
        String left = s1.substring(2) + s1.substring(0, 2);
        String right = s1.substring(s1.length() - 2) + s1.substring(0, s1.length() - 2);
        return s2.equals(left) || s2.equals(right);
    }
}
```

OUTPUT:

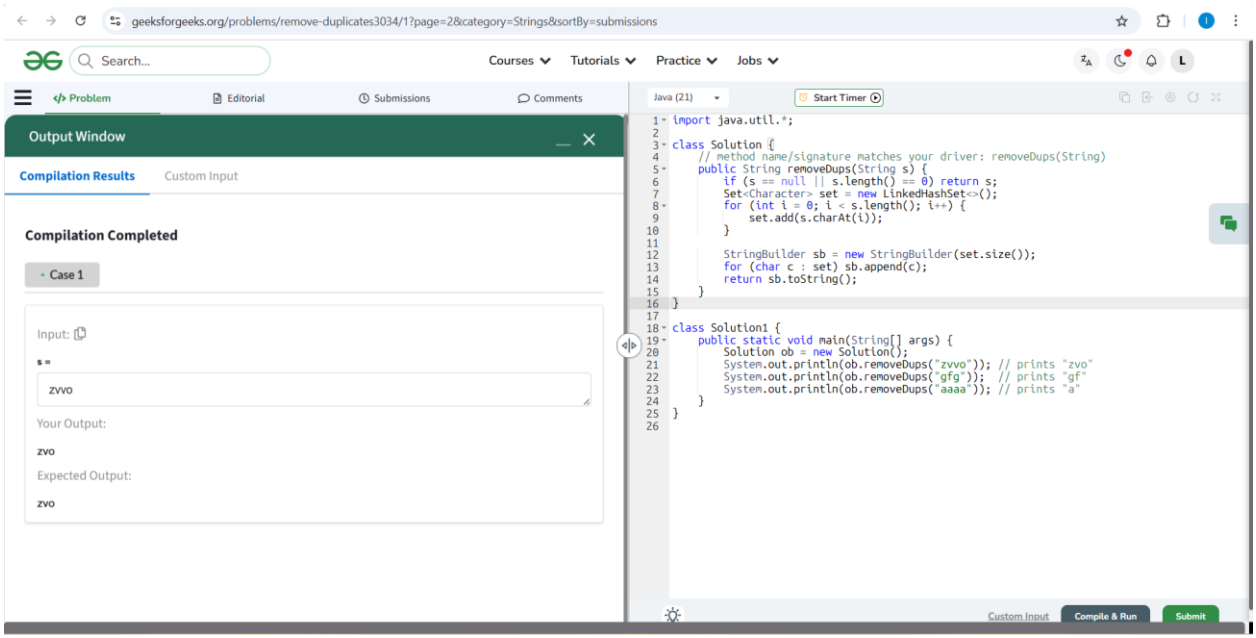


30. Given a string **s** without spaces, the task is to remove all duplicate characters from it, keeping only the first occurrence.

Note: The original order of characters must be kept the same.

```
import java.util.*;
class Solution {
   // method name/signature matches your driver: removeDups(String)
   public String removeDups(String s) {
      if (s == null || s.length() == 0) return s;
      Set<Character> set = new LinkedHashSet<>();
      for (int i = 0; i < s.length(); i++) {
         set.add(s.charAt(i));
      }
      StringBuilder sb = new StringBuilder(set.size());
      for (char c : set) sb.append(c);
      return sb.toString();
   }
}
class Solution1 {
   public static void main(String[] args) {
      Solution ob = new Solution();
      System.out.println(ob.removeDups("zvvo")); // prints "zvo"
      System.out.println(ob.removeDups("gfg"));  // prints "gf"
      System.out.println(ob.removeDups("aaaa")); // prints "a"
   }
}
```

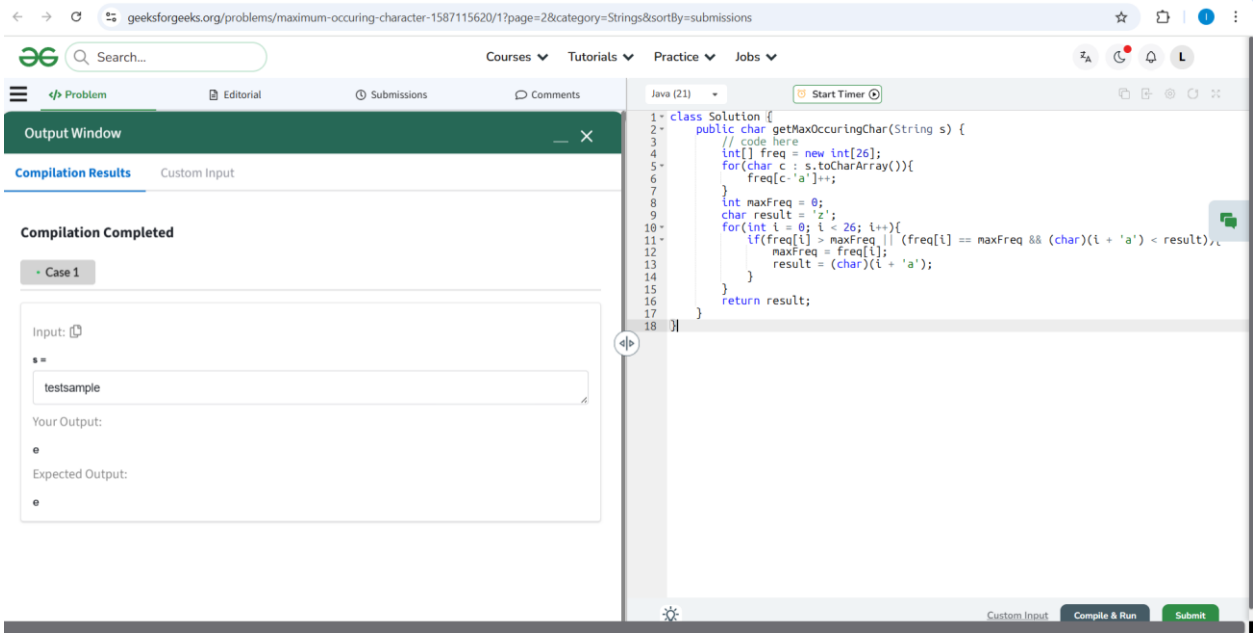
OUTPUT:


geeksforgeeks.org/problems/remove-duplicates3034/1?page=2&category=Strings&sortBy=submissions
Search...
Courses
Tutorials
Practice
Jobs
Problem
Editorial
Submissions
Comments
Output Window
Compilation Results
Custom Input
Compilation Completed
Case 1
Input:
s =
zvvo
Your Output:
zvo
Expected Output:
zvo
Java (21)
Start Timer
import java.util.*;
class Solution {
// method name/signature matches your driver: removeDups(String)
public String removeDups(String s) {
if (s == null || s.length() == 0) return s;
Set<Character> set = new LinkedHashSet<>();
for (int i = 0; i < s.length(); i++) {
set.add(s.charAt(i));
}
StringBuilder sb = new StringBuilder(set.size());
for (char c : set) sb.append(c);
return sb.toString();
}
}
class Solution1 {
public static void main(String[] args) {
Solution ob = new Solution();
System.out.println(ob.removeDups("zvvo")); // prints "zvo"
System.out.println(ob.removeDups("gfg")); // prints "gf"
System.out.println(ob.removeDups("aaaa")); // prints "a"
}
}
Custom Input
Compile & Run
Submit

31. Given a string **s** of lowercase alphabets. The task is to find the maximum occurring character in the string **s**. If more than one character occurs the maximum number of times then print the lexicographically smaller character.

```
class Solution {
    public char getMaxOccuringChar(String s) {
        // code here
        int[] freq = new int[26];
        for(char c : s.toCharArray()){
            freq[c-'a']++;
        }
        int maxFreq = 0;
        char result = 'z';
        for(int i = 0; i < 26; i++){
            if(freq[i] > maxFreq || (freq[i] == maxFreq && (char)(i + 'a') < result)){
                maxFreq = freq[i];
                result = (char)(i + 'a');
            }
        }
        return result;
    }
}
```

OUTPUT:

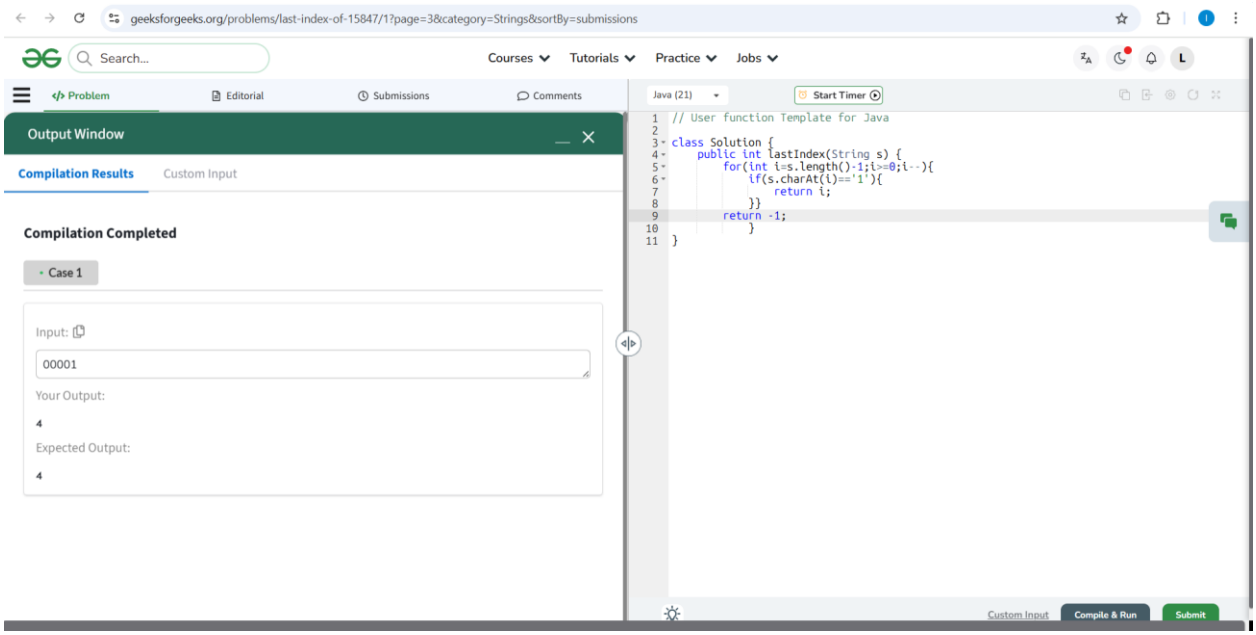


32. Given a string **s** consisting of only '**0**'s and '**1**'s, find the last index of the '1' present.

Note: If '1' is not present, return "-1"

```
// User function Template for Java

class Solution {
    public int lastIndex(String s) {
        for(int i=s.length()-1;i>=0;i--){
            if(s.charAt(i)=='1'){
                return i;
            }}
        return -1;
            }
}
```

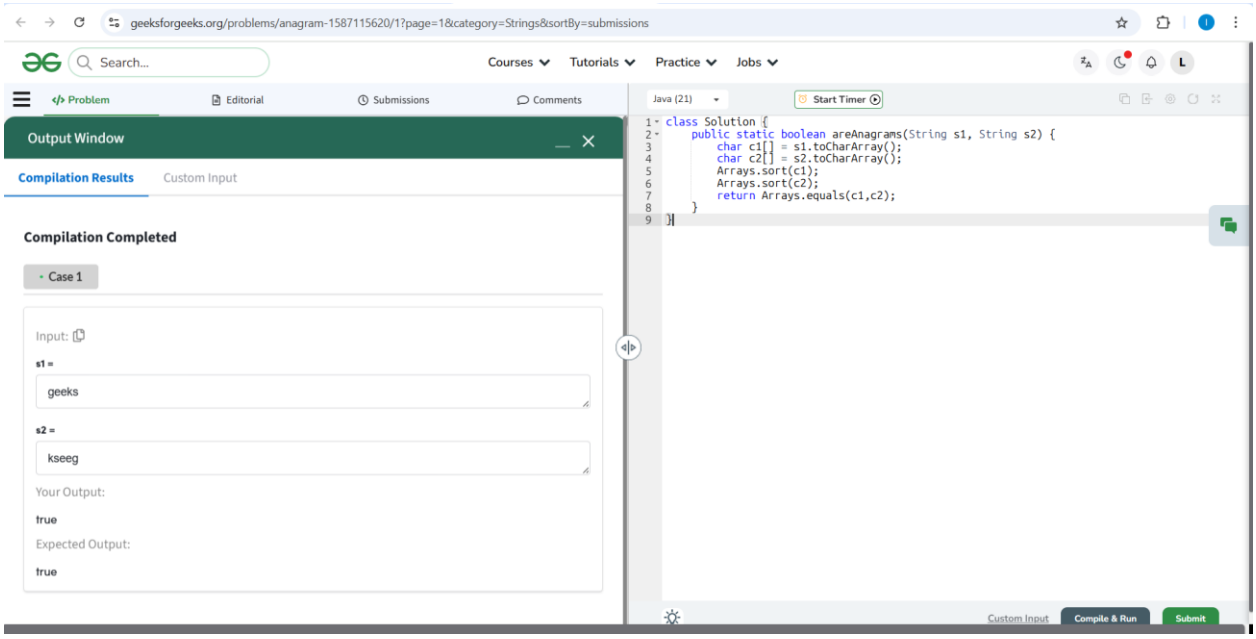
OUTPUT:



33. Given two non-empty strings **s1** and **s2**, consisting only of lowercase English letters, determine whether they are **anagrams** of each other or not.
Two strings are considered **anagrams** if they contain the **same characters** with exactly the **same frequencies**, regardless of their order.

```
class Solution {
    public static boolean areAnagrams(String s1, String s2) {
        char c1[] = s1.toCharArray();
        char c2[] = s2.toCharArray();
        Arrays.sort(c1);
        Arrays.sort(c2);
        return Arrays.equals(c1,c2);
    }
}
```

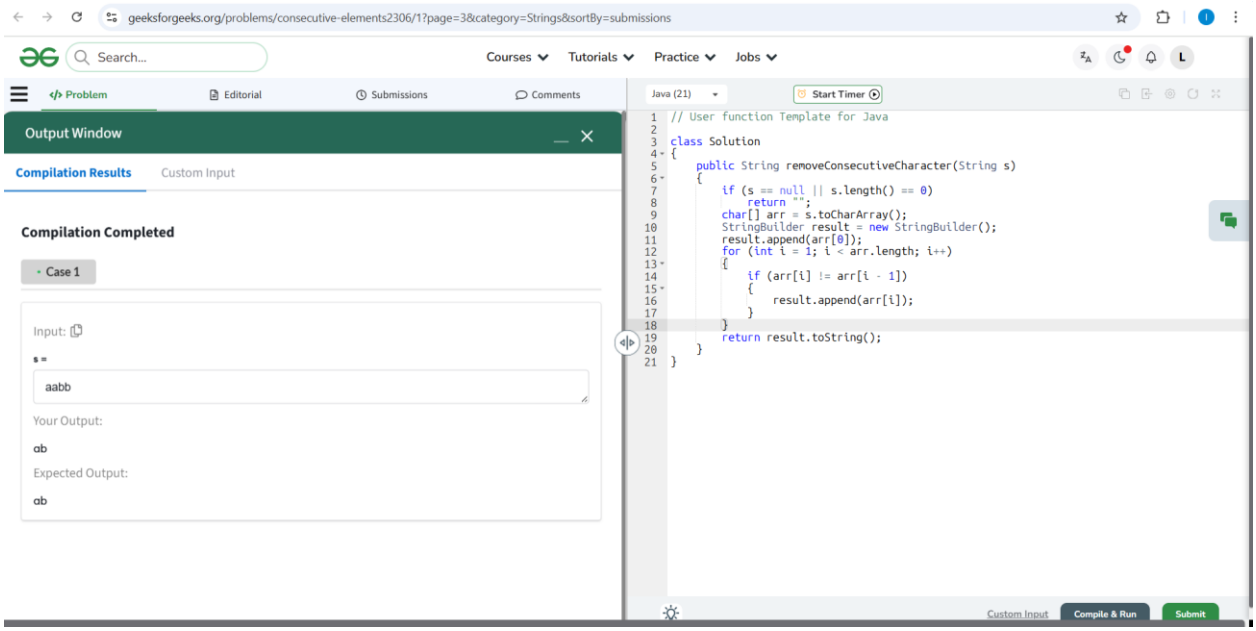
OUTPUT:



34. You are given a string **s**, consisting of lowercase alphabets. Your task is to remove consecutive duplicate characters from the string.

```
// User function Template for Java

class Solution
{
    public String removeConsecutiveCharacter(String s)
    {
        if (s == null || s.length() == 0)
            return "";
        char[] arr = s.toCharArray();
        StringBuilder result = new StringBuilder();
        result.append(arr[0]);
        for (int i = 1; i < arr.length; i++)
        {
            if (arr[i] != arr[i - 1])
            {
                result.append(arr[i]);
            }
        }
        return result.toString();
    }
}
```

OUTPUT:



Programs on Series Problems

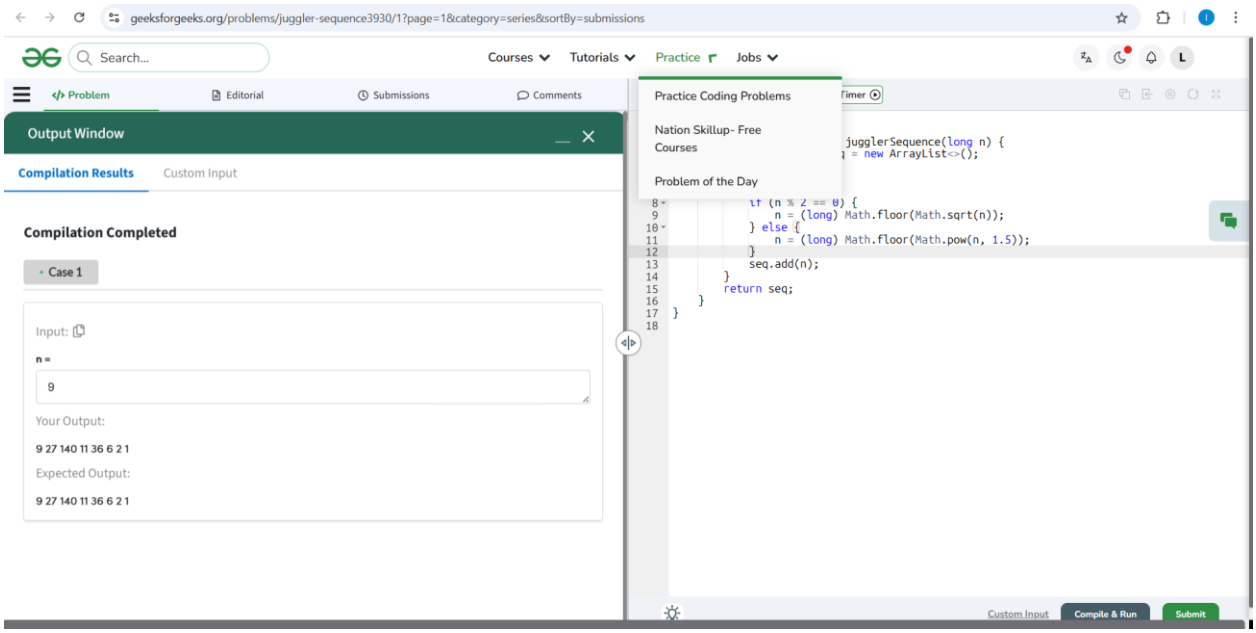
35. Juggler Sequence is a series of integers in which the first term starts with a positive integer number *a* and the remaining terms are generated from the immediate previous term using the below recurrence relation:

$$a_{k+1} = \begin{cases} \lfloor a_k^{1/2} \rfloor & \text{for even } a_k \\ \lfloor a_k^{3/2} \rfloor & \text{for odd } a_k, \end{cases}$$

Given a number n, find the Juggler Sequence for this number as the first term of the sequence until it becomes 1.

```
import java.util.*;
class Solution {
   public ArrayList<Long> jugglerSequence(long n) {
      ArrayList<Long> seq = new ArrayList<>();
      seq.add(n);
      while (n != 1) {
         if (n % 2 == 0) {
            n = (long) Math.floor(Math.sqrt(n));
         } else {
            n = (long) Math.floor(Math.pow(n, 1.5));
         }
         seq.add(n);
      }
      return seq;
   }
}
```

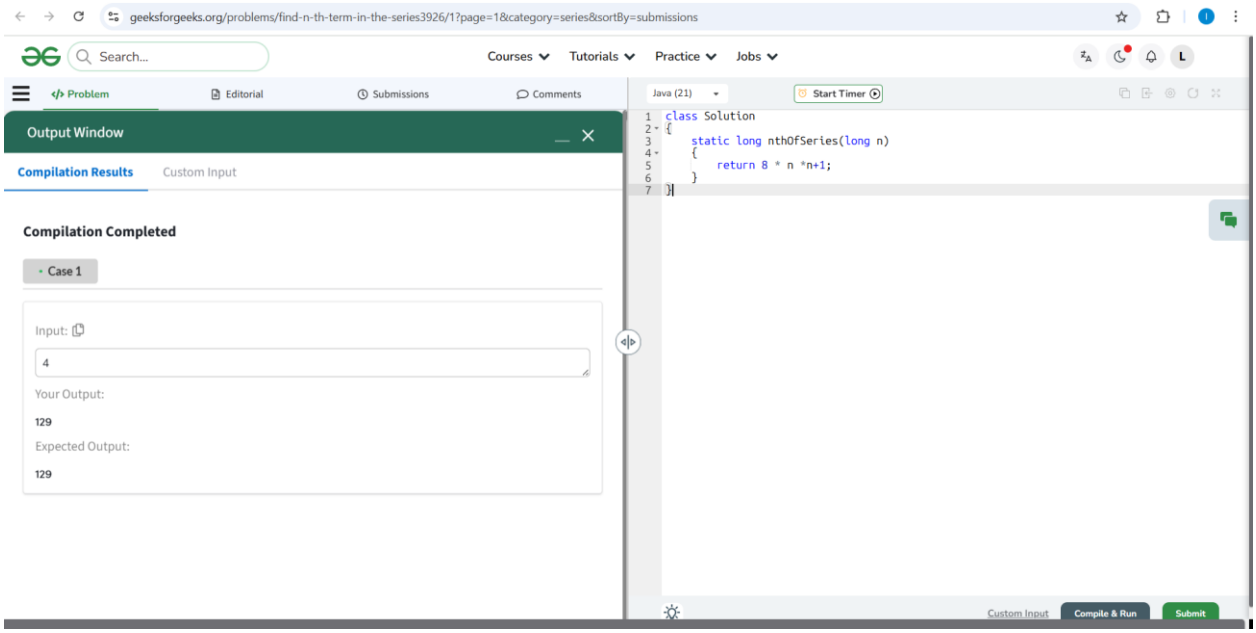
OUTPUT:


geeksforgeeks.org/problems/juggler-sequence3930/1?page=1&category=series&sortBy=submissions
Search...
Courses
Tutorials
Practice
Jobs
Problem
Editorial
Submissions
Comments
Practice Coding Problems
Nation Skillup- Free Courses
Problem of the Day
Output Window
Compilation Results
Custom Input
Compilation Completed
Case 1
Input:
n =
9
Your Output:
9 27 140 11 36 6 2 1
Expected Output:
9 27 140 11 36 6 2 1
jugglerSequence(long n) {
= new ArrayList<>();
if (n % 2 == 0) {
n = (long) Math.floor(Math.sqrt(n));
} else {
n = (long) Math.floor(Math.pow(n, 1.5));
}
seq.add(n);
return seq;
Custom Input
Compile & Run
Submit

36. Given a series 9, 33, 73, 129... Find the n-th term of the series.

```
class Solution
{
   static long nthOfSeries(long n)
   {
      return 8 * n *n+1;
   }
}
```

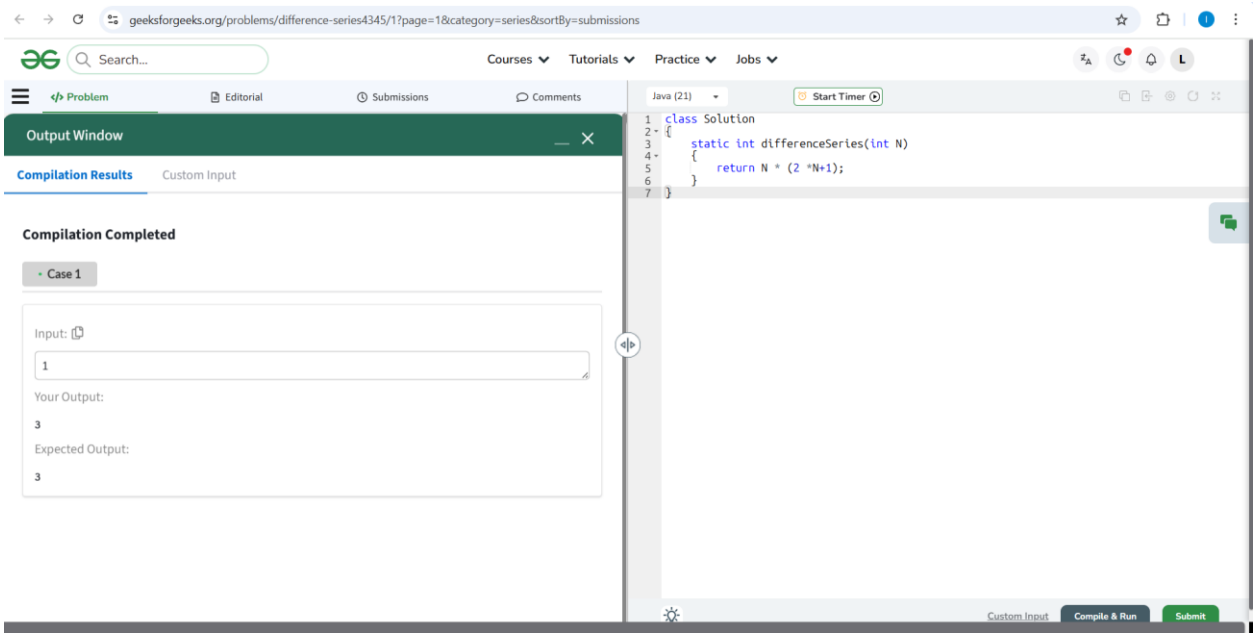
OUTPUT:



37. Given a series of numbers 3,10,21,36 …., and series starting from N = 1, find the pattern and output the N'th value of the above series.

```
class Solution
{
   static int differenceSeries(int N)
   {
      return N * (2 *N+1);
   }
}
```

OUTPUT:

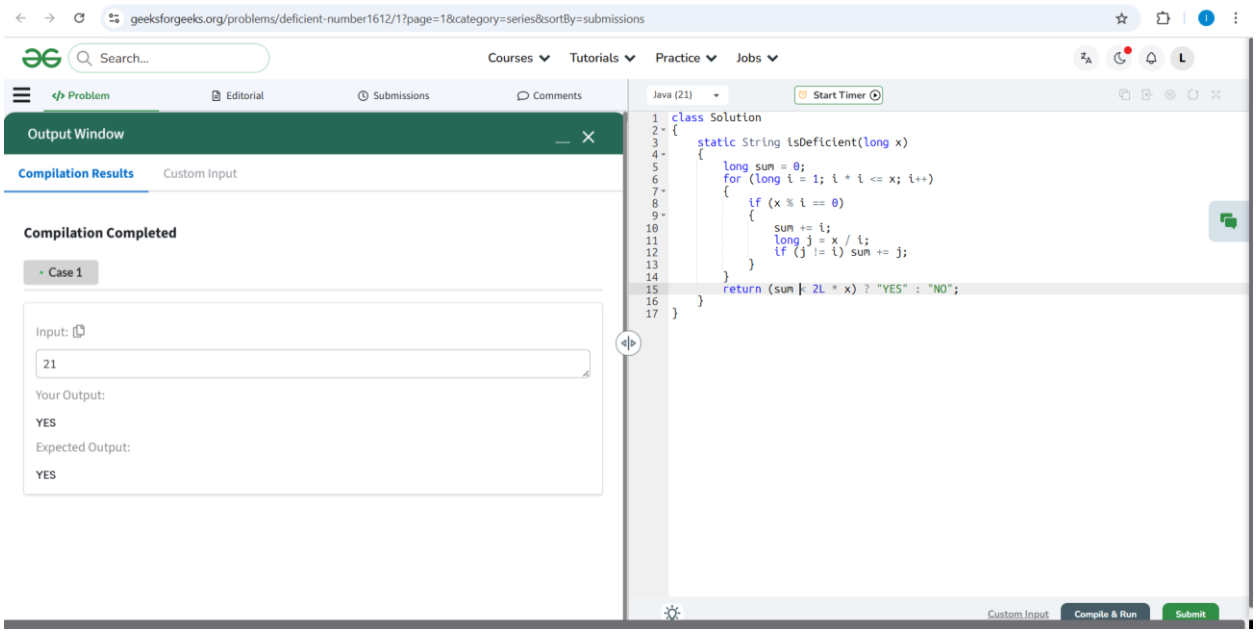


38. â€‹Given a number x, your task is to find if this number is Deficient number or not. A number x is said to be Deficient Number if sum of all the divisors of the number denoted by *divisorsSum(x)* is less than twice the value of the number x. And the difference between these two values is called the **deficiency**.

Mathematically, if below condition holds the number is said to be Deficient:
divisorsSum(x) < 2*x
deficiency = (2*x) - divisorsSum(x)

```
class Solution
{
   static String isDeficient(long x)
   {
      long sum = 0;
      for (long i = 1; i * i <= x; i++)
      {
         if (x % i == 0)
         {
            sum += i;
            long j = x / i;
            if (j != i) sum += j;
         }
      }
      return (sum < 2L * x) ? "YES" : "NO";
   }
}
```

OUTPUT:

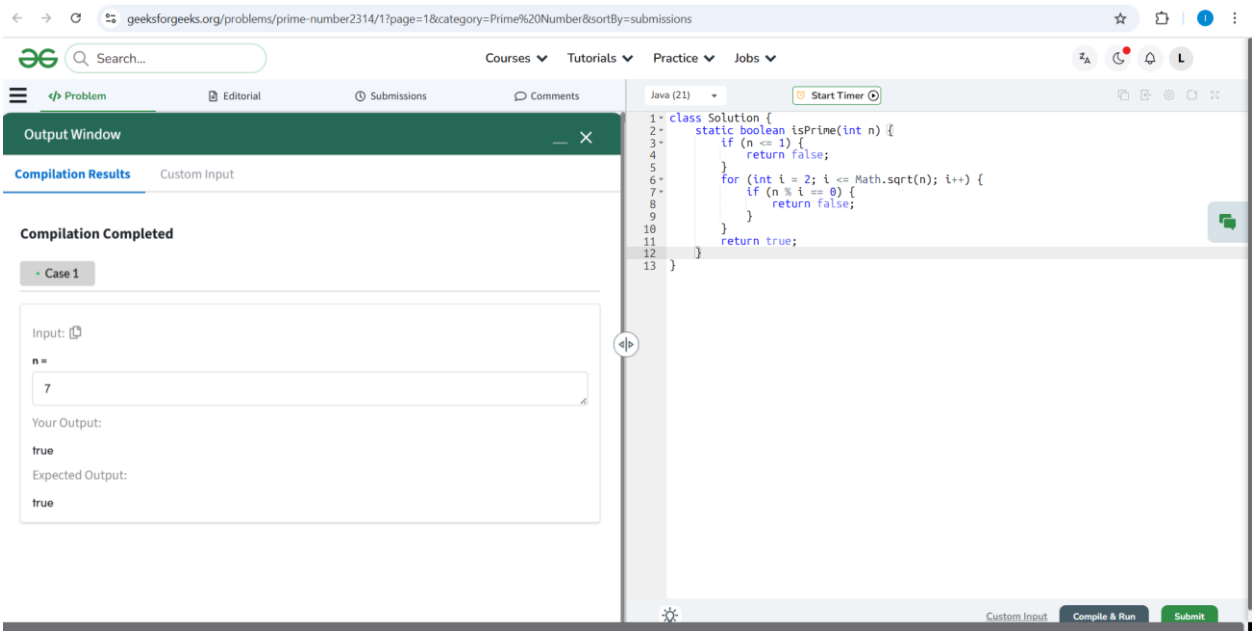

geeksforgeeks.org/problems/deficient-number1612/1?page=1&category=series&sortBy=submissions
Search...
Courses
Tutorials
Practice
Jobs
Problem
Editorial
Submissions
Comments
Output Window
Compilation Results
Custom Input
Compilation Completed
Case 1
Input:
21
Your Output:
YES
Expected Output:
YES
Java (21)
Start Timer
class Solution
{
static String isDeficient(long x)
{
long sum = 0;
for (long i = 1; i * i <= x; i++)
{
if (x % i == 0)
{
sum += i;
long j = x / i;
if (j != i) sum += j;
}
}
return (sum < 2L * x) ? "YES" : "NO";
}
}
Custom Input
Compile & Run
Submit

Programs on Prime Numbers Problems

39. Given a number **n**, determine whether it is a **prime number** or not.
Note: A prime number is a number greater than 1 that has no positive divisors other than 1 and itself.

```
class Solution {
    static boolean isPrime(int n) {
        if (n <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

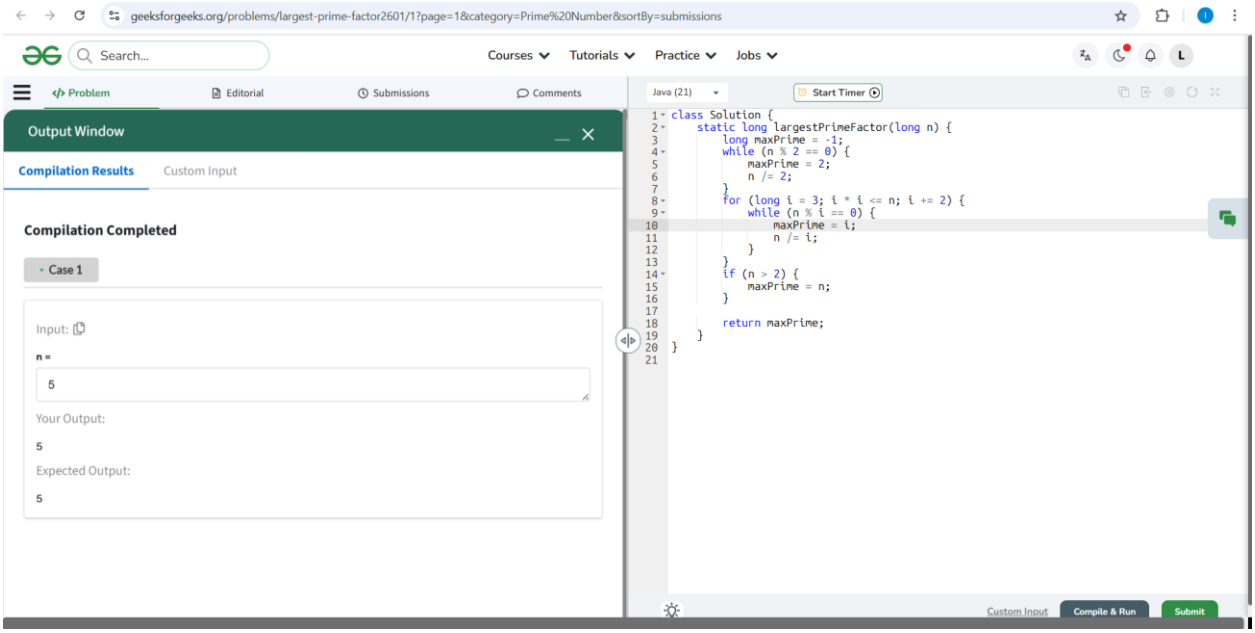
OUTPUT:



40. Given a number **n**, your task is to find the largest prime factor of n.

```
class Solution {
   static long largestPrimeFactor(long n) {
      long maxPrime = -1;
      while (n % 2 == 0) {
         maxPrime = 2;
         n /= 2;
      }
      for (long i = 3; i * i <= n; i += 2) {
         while (n % i == 0) {
            maxPrime = i;
            n /= i;
         }
      }
      if (n > 2) {
         maxPrime = n;
      }

      return maxPrime;
   }
}
```

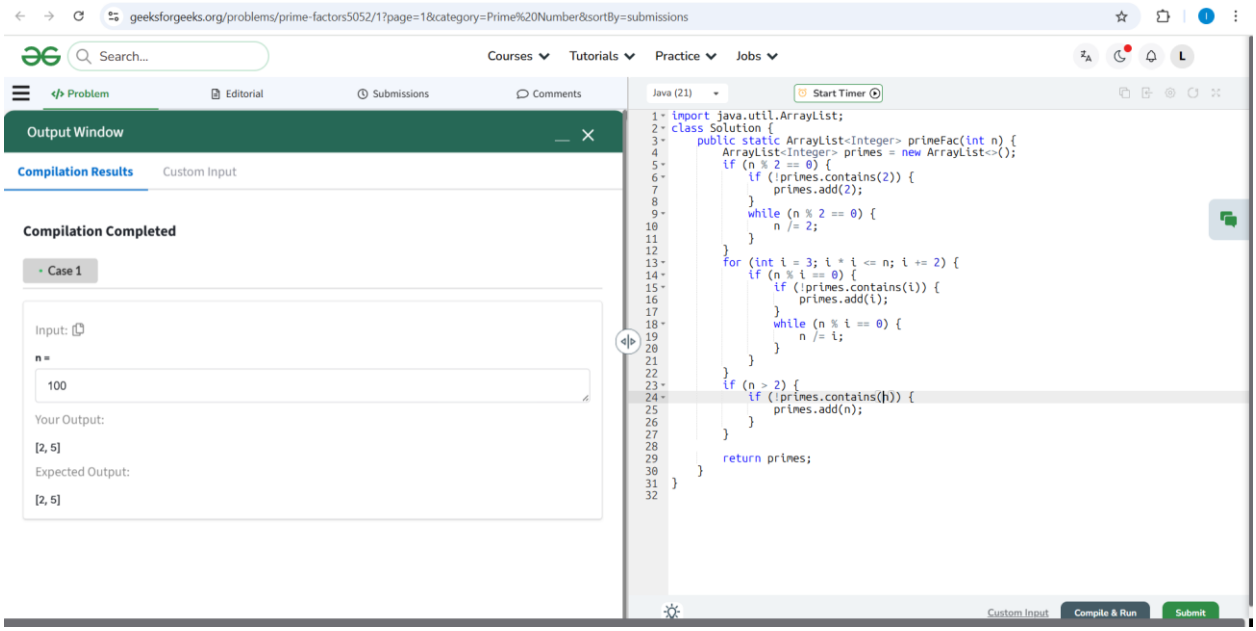
OUTPUT:


geeksforgeeks.org/problems/largest-prime-factor2601/1?page=1&category=Prime%20Number&sortBy=submissions
Search...
Courses
Tutorials
Practice
Jobs
Problem
Editorial
Submissions
Comments
Output Window
Compilation Results
Custom Input
Compilation Completed
Case 1
Input:
n =
5
Your Output:
5
Expected Output:
5
Java (21)
Start Timer
class Solution {
static long largestPrimeFactor(long n) {
long maxPrime = -1;
while (n % 2 == 0) {
maxPrime = 2;
n /= 2;
}
for (long i = 3; i * i <= n; i += 2) {
while (n % i == 0) {
maxPrime = i;
n /= i;
}
}
if (n > 2) {
maxPrime = n;
}
return maxPrime;
}
}
Custom Input
Compile & Run
Submit

41. Given a number **n**. Find its **unique** prime factors in **increasing order**.

```
import java.util.ArrayList;
class Solution {
    public static ArrayList<Integer> primeFac(int n) {
        ArrayList<Integer> primes = new ArrayList<>();
        if (n % 2 == 0) {
            if (!primes.contains(2)) {
                primes.add(2);
            }
            while (n % 2 == 0) {
                n /= 2;
            }
        }
        for (int i = 3; i * i <= n; i += 2) {
            if (n % i == 0) {
                if (!primes.contains(i)) {
                    primes.add(i);
                }
                while (n % i == 0) {
                    n /= i;
                }
            }
        }
        if (n > 2) {
            if (!primes.contains(n)) {
                primes.add(n);
            }
        }

        return primes;
    }
}
```

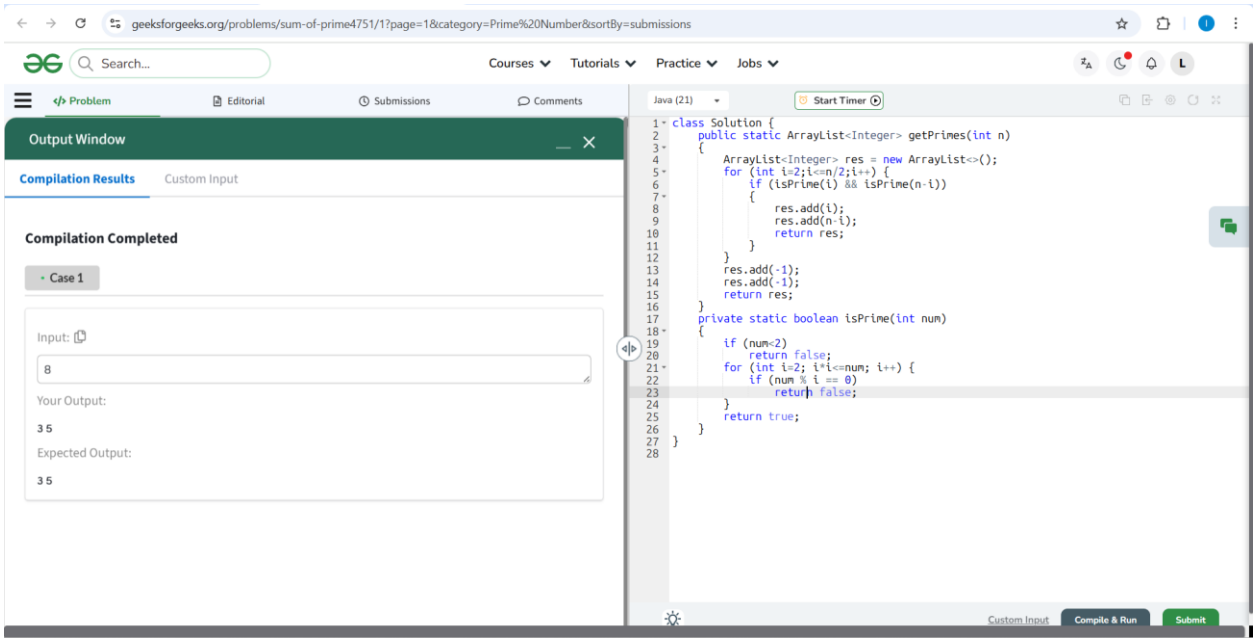
OUTPUT:


geeksforgeeks.org/problems/prime-factors5052/1?page=1&category=Prime%20Number&sortBy=submissions
Search...
Courses
Tutorials
Practice
Jobs
Problem
Editorial
Submissions
Comments
Output Window
Compilation Results
Custom Input
Compilation Completed
Case 1
Input:
n =
100
Your Output:
[2, 5]
Expected Output:
[2, 5]
Java (21)
Start Timer
import java.util.ArrayList;
class Solution {
public static ArrayList<Integer> primeFac(int n) {
ArrayList<Integer> primes = new ArrayList<>();
if (n % 2 == 0) {
if (!primes.contains(2)) {
primes.add(2);
}
while (n % 2 == 0) {
n /= 2;
}
}
for (int i = 3; i * i <= n; i += 2) {
if (n % i == 0) {
if (!primes.contains(i)) {
primes.add(i);
}
while (n % i == 0) {
n /= i;
}
}
}
if (n > 2) {
if (!primes.contains(n)) {
primes.add(n);
}
}
return primes;
}
}
Custom Input
Compile & Run
Submit

42. Given a number n, find out if n can be expressed as a+b, where both a and b are prime numbers. If such a pair exists, return the values of a and b, otherwise return [-1,-1] as an array of size 2.
Note: If [a, b] is one solution with a <= b, and [c, d] is another solution with c <= d, and a < c then [a, b] is considered as our answer.

```
class Solution {
   public static ArrayList<Integer> getPrimes(int n)
   {
      ArrayList<Integer> res = new ArrayList<>();
      for (int i=2;i<=n/2;i++) {
         if (isPrime(i) && isPrime(n-i))
         {
            res.add(i);
            res.add(n-i);
            return res;
         }
      }
      res.add(-1);
      res.add(-1);
      return res;
   }
   private static boolean isPrime(int num)
   {
      if (num<2)
         return false;
      for (int i=2; i*i<=num; i++) {
         if (num % i == 0)
            return false;
      }
      return true;
   }
}
```

OUTPUT:

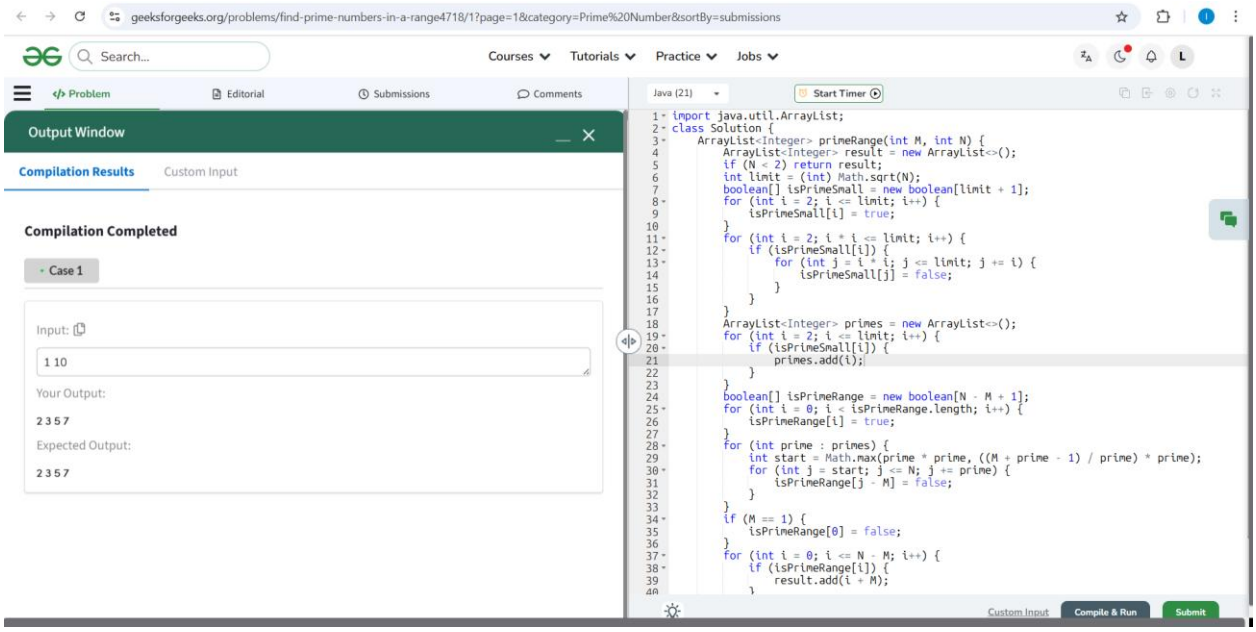


43. Given two integers M and N, generate all primes between M and N including M and N.

```
import java.util.ArrayList;
class Solution {
   ArrayList<Integer> primeRange(int M, int N) {
      ArrayList<Integer> result = new ArrayList<>();
      if (N < 2) return result;
      int limit = (int) Math.sqrt(N);
      boolean[] isPrimeSmall = new boolean[limit + 1];
      for (int i = 2; i <= limit; i++) {
         isPrimeSmall[i] = true;
      }
      for (int i = 2; i * i <= limit; i++) {
         if (isPrimeSmall[i]) {
            for (int j = i * i; j <= limit; j += i) {
               isPrimeSmall[j] = false;
            }
         }
      }
      ArrayList<Integer> primes = new ArrayList<>();
      for (int i = 2; i <= limit; i++) {
         if (isPrimeSmall[i]) {
            primes.add(i);
         }
      }
      boolean[] isPrimeRange = new boolean[N - M + 1];
      for (int i = 0; i < isPrimeRange.length; i++) {
         isPrimeRange[i] = true;
      }
      for (int prime : primes) {
         int start = Math.max(prime * prime, ((M + prime - 1) / prime) * prime);
         for (int j = start; j <= N; j += prime) {
            isPrimeRange[j - M] = false;
         }
      }
```

```
        if (M == 1) {
            isPrimeRange[0] = false;
        }
        for (int i = 0; i <= N - M; i++) {
            if (isPrimeRange[i]) {
                result.add(i + M);
            }
        }

        return result;
    }
}
```

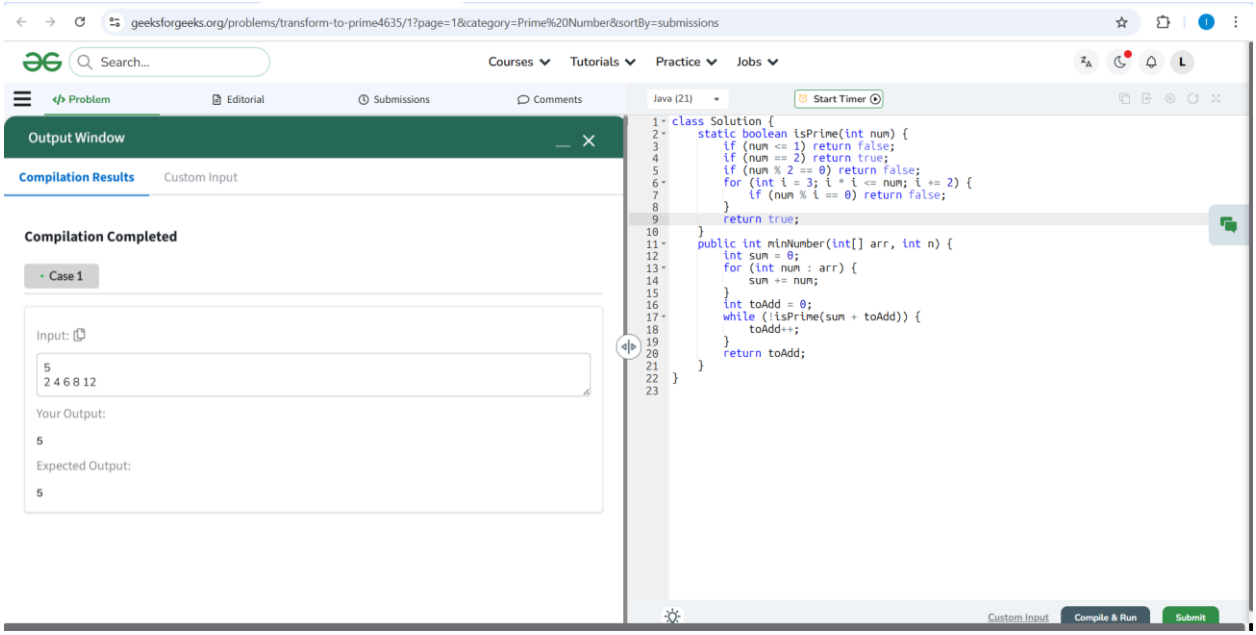
OUTPUT:



44. Given an array of **n** integers. Find the **minimum** non-negative number to be inserted in array, so that sum of all elements of array becomes **prime**.

```
class Solution {
    static boolean isPrime(int num) {
        if (num <= 1) return false;
        if (num == 2) return true;
        if (num % 2 == 0) return false;
        for (int i = 3; i * i <= num; i += 2) {
            if (num % i == 0) return false;
        }
        return true;
    }
    public int minNumber(int[] arr, int n) {
        int sum = 0;
        for (int num : arr) {
            sum += num;
        }
        int toAdd = 0;
        while (!isPrime(sum + toAdd)) {
            toAdd++;
        }
        return toAdd;
    }
}
```

OUTPUT:



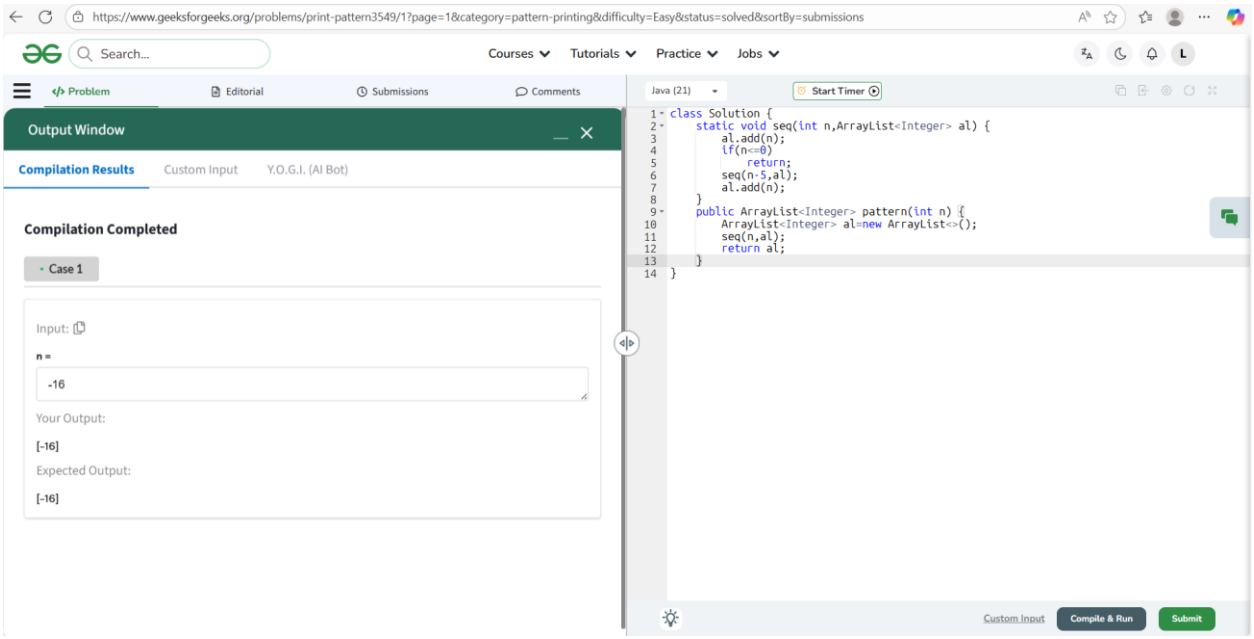
Programs on Pattern Problems

45. Given a number **n**, print a **sequence of numbers** starting from **n**. Each next number in the sequence is **n - 5**, and this continues recursively until the number becomes **less than** or **equal to 0**. After that, print the sequence in reverse order, **adding 5** each time, until it reaches back to the original number **n**.

Note: You must not use loops.class Solution

```
class Solution {
    static void seq(int n,ArrayList<Integer> al) {
        al.add(n);
        if(n<=0)
            return;
        seq(n-5,al);
        al.add(n);
    }
    public ArrayList<Integer> pattern(int n) {
        ArrayList<Integer> al=new ArrayList<>();
        seq(n,al);
        return al;
    }
}
```

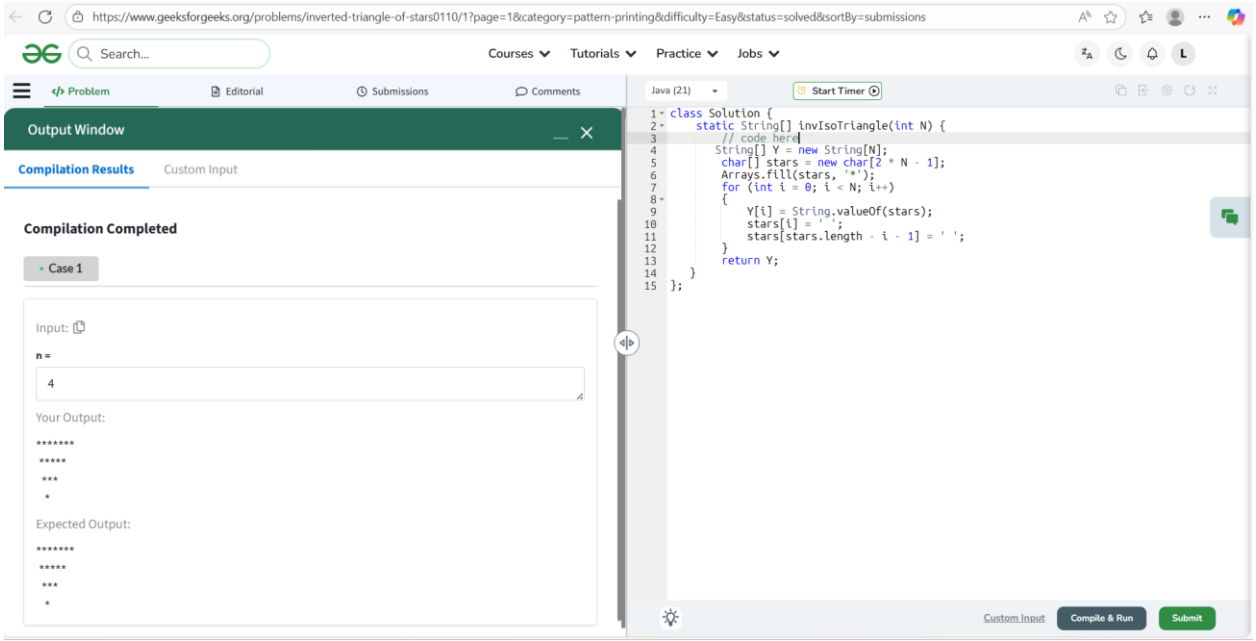
OUTPUT:



46. Given an integer **n**, print an inverted isosceles triangle of stars such that the height of the triangle is **n**.

```
class Solution {
   static String[] invIsoTriangle(int N) {
       // code here
      String[] Y = new String[N];
       char[] stars = new char[2 * N - 1];
       Arrays.fill(stars, '*');
       for (int i = 0; i < N; i++)
       {
          Y[i] = String.valueOf(stars);
          stars[i] = ' ';
          stars[stars.length - i - 1] = ' ';
       }
       return Y;
   }
};
```

OUTPUT:



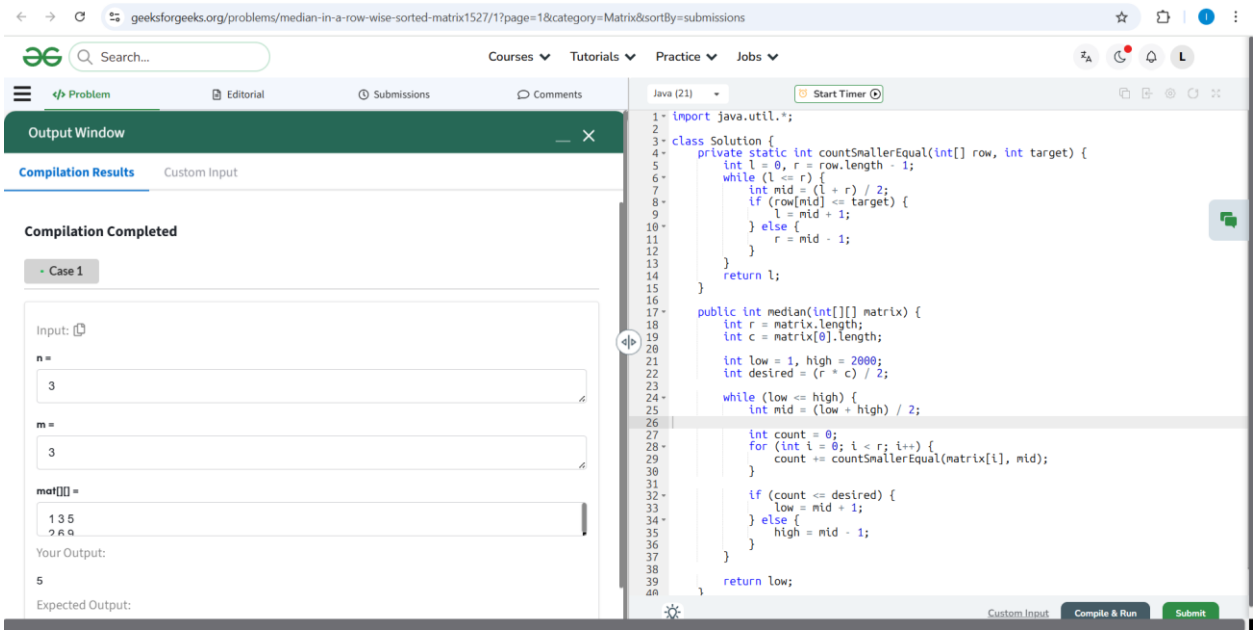
Programs on Matrices Problems

47. Given a **row-wise sorted** matrix **mat[][]** of size n*m, where the number of rows and columns is always **odd**. Return the **median** of the matrix.

```
import java.util.*;

class Solution {

    // Function to count elements <= target in a sorted row using binary search
    private static int countSmallerEqual(int[] row, int target) {
        int l = 0, r = row.length - 1;
        while (l <= r) {
            int mid = (l + r) / 2;
            if (row[mid] <= target) {
                l = mid + 1;  // move right
            } else {
                r = mid - 1;  // move left
            }
        }
        return l; // number of elements <= target
    }

    public int median(int[][] matrix) {
        int r = matrix.length;
        int c = matrix[0].length;

        int low = 1, high = 2000; // given constraints
        int desired = (r * c) / 2;

        while (low <= high) {
            int mid = (low + high) / 2;

            // count how many elements <= mid
            int count = 0;
            for (int i = 0; i < r; i++) {
                count += countSmallerEqual(matrix[i], mid);
            }

            if (count <= desired) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        return low;
```

```
    }
}
```

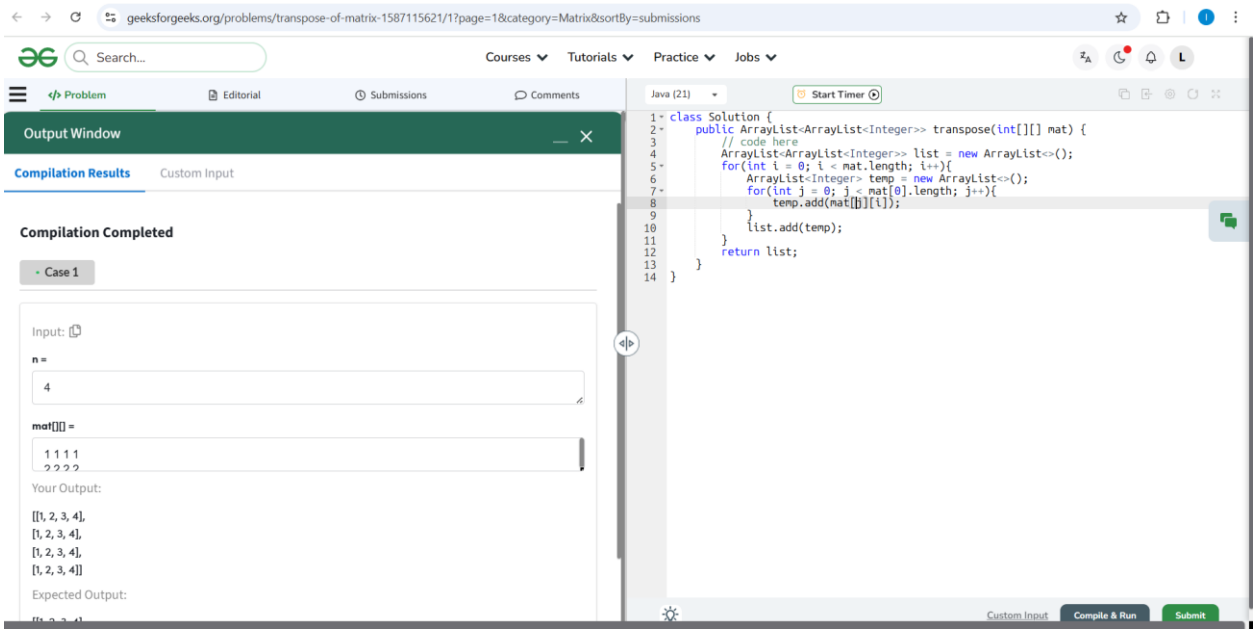
OUTPUT:


geeksforgeeks.org/problems/median-in-a-row-wise-sorted-matrix1527/1?page=1&category=Matrix&sortBy=submissions
Search...
Courses
Tutorials
Practice
Jobs
Problem
Editorial
Submissions
Comments
Output Window
Compilation Results
Custom Input
Compilation Completed
Case 1
Input:
n =
3
m =
3
mat[][] =
1 3 5
2 6 9
Your Output:
5
Expected Output:
Java (21)
Start Timer
import java.util.*;
class Solution {
private static int countSmallerEqual(int[] row, int target) {
int l = 0, r = row.length - 1;
while (l <= r) {
int mid = (l + r) / 2;
if (row[mid] <= target) {
l = mid + 1;
} else {
r = mid - 1;
}
}
return l;
}
public int median(int[][] matrix) {
int r = matrix.length;
int c = matrix[0].length;
int low = 1, high = 2000;
int desired = (r * c) / 2;
while (low <= high) {
int mid = (low + high) / 2;
int count = 0;
for (int i = 0; i < r; i++) {
count += countSmallerEqual(matrix[i], mid);
}
if (count <= desired) {
low = mid + 1;
} else {
high = mid - 1;
}
}
return low;
}
Custom Input
Compile & Run
Submit

48. You are given a square matrix of size **n x n**. Your task is to find the **transpose** of the given matrix.
The **transpose** of a matrix is obtained by converting all the rows to columns and all the columns to rows.

```
class Solution {
    public ArrayList<ArrayList<Integer>> transpose(int[][] mat) {
        // code here
        ArrayList<ArrayList<Integer>> list = new ArrayList<>();
        for(int i = 0; i < mat.length; i++){
            ArrayList<Integer> temp = new ArrayList<>();
            for(int j = 0; j < mat[0].length; j++){
                temp.add(mat[j][i]);
            }
            list.add(temp);
        }
        return list;
    }
}
```

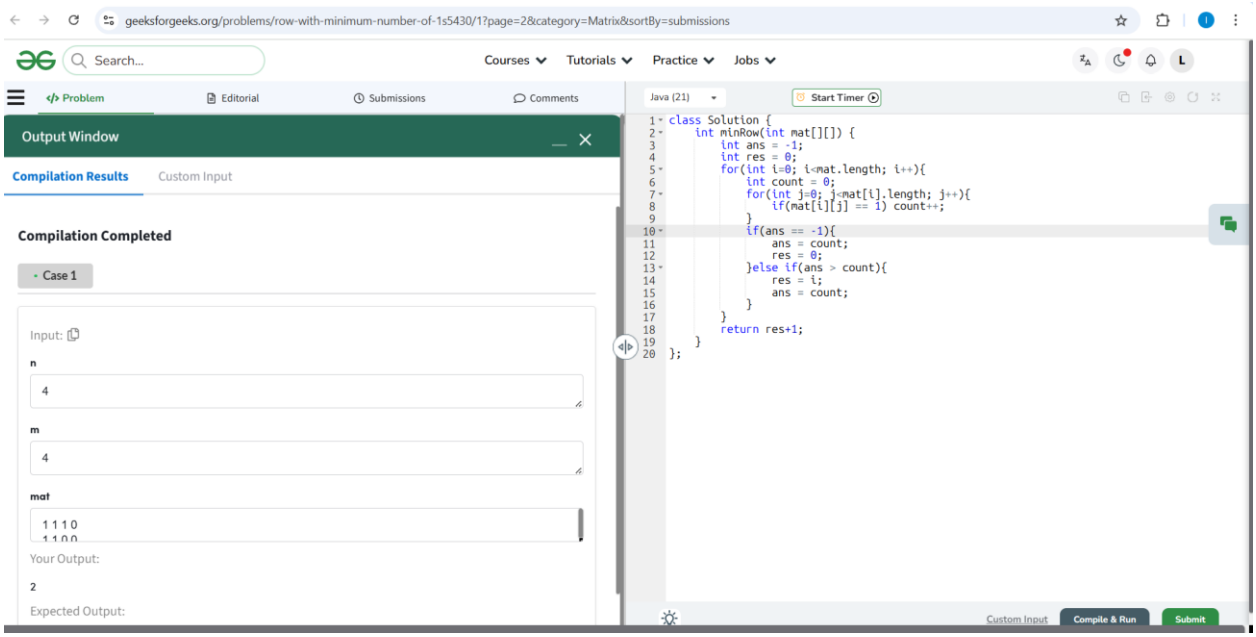
OUTPUT:



49. Given a 2D **binary matrix**(1-based indexed) **mat** of dimensions **nxm** , determine the **row** that contains the **minimum number** of **1's**.
Note: The matrix contains only **1's** and **0's**. Also, if two or more rows contain the **minimum number** of **1's**, the answer is the **lowest** of those **indices**.

```
class Solution {
    int minRow(int mat[][]) {
        int ans = -1;
        int res = 0;
        for(int i=0; i<mat.length; i++){
            int count = 0;
            for(int j=0; j<mat[i].length; j++){
                if(mat[i][j] == 1) count++;
            }
            if(ans == -1){
                ans = count;
                res = 0;
            }else if(ans > count){
                res = i;
                ans = count;
            }
        }
        return res+1;
    }
};
```

OUTPUT:



50. Given a square matrix of size **n*n**, print the **sum of upper and lower triangular elements**. Upper Triangle consists of elements on the diagonal and above it. The lower triangle consists of elements on the diagonal and below it.

```
class Solution {
    public static ArrayList<Integer> sumTriangles(int[][] mat) {
        int n = mat.length;
        int upperSum = 0;
        int lowerSum = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (j >= i)
                    upperSum += mat[i][j];
                if (j <= i)
                    lowerSum += mat[i][j];
            }
        }
        ArrayList<Integer> ans = new ArrayList<>();
        ans.add(upperSum);
        ans.add(lowerSum);
        return ans;
    }
}
```

OUTPUT: