23A81A0562

REPORT

on

Master Coding and Competitive Programming - Part-1

(*Course Code: V23CSES01)*

B.TECH V Semester (2023 - 2027 Batch)

Academic Year: 2025-26

SRI VASAVI ENGINEERING COLLEGE (Autonomous)

Pedatadepalli , Tadepalligudem - 534101

Department of Computer Science & Engineering (Accredited by NBA)

SRI VASAVI ENGINEERING COLLEGE (Autonomous)

PEDATADEPALLI, TADEPALLIGUDEM.

Certificate

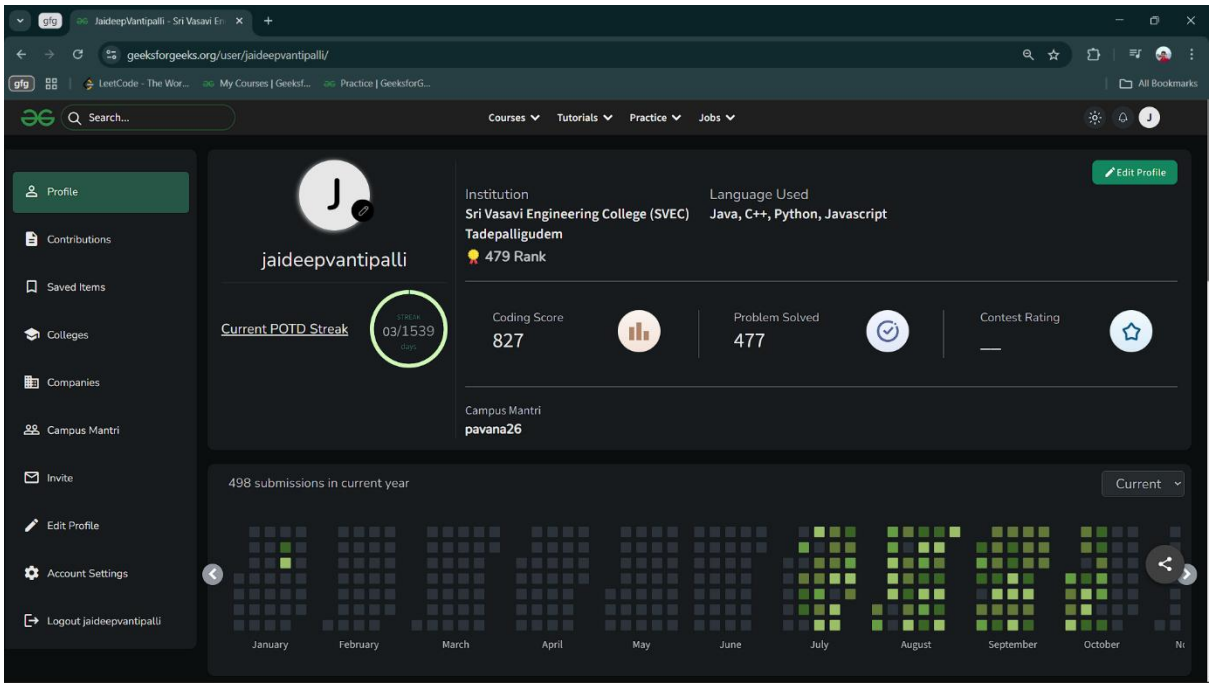
This is to certify Mr. **Jaideep Vantipalli** bearing Roll No. **23A81A0562** of CSE Branch of V Semester submitted the Report on _____ as a part of **Master Coding and Competitive Programming - Part-1** (Course Code:V23CSES01) during the academic year 2025-2026.

Faculty In charge **Head of the Department**

Examiner1 **Examiner2**

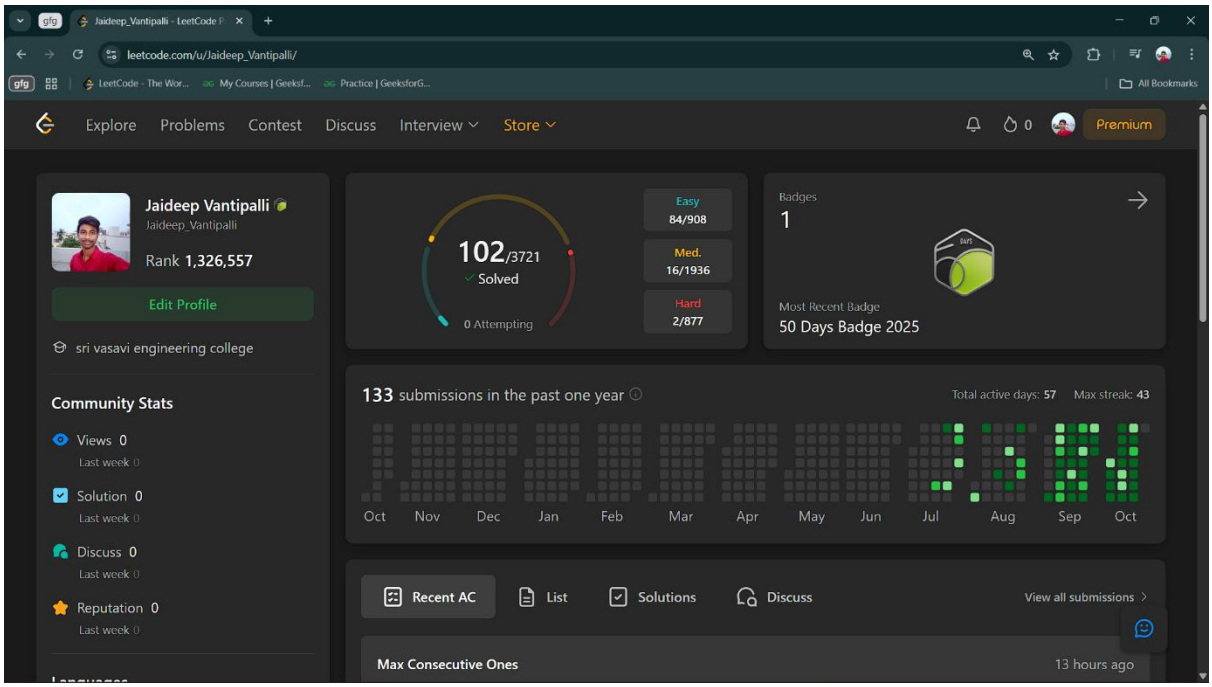
Geeks For Geeks Profile :

https://www.geeksforgeeks.org/user/jaideepvantipalli/



LeetCode Profile :

https://leetcode.com/u/Jaideep_Vantipalli/



INDEX

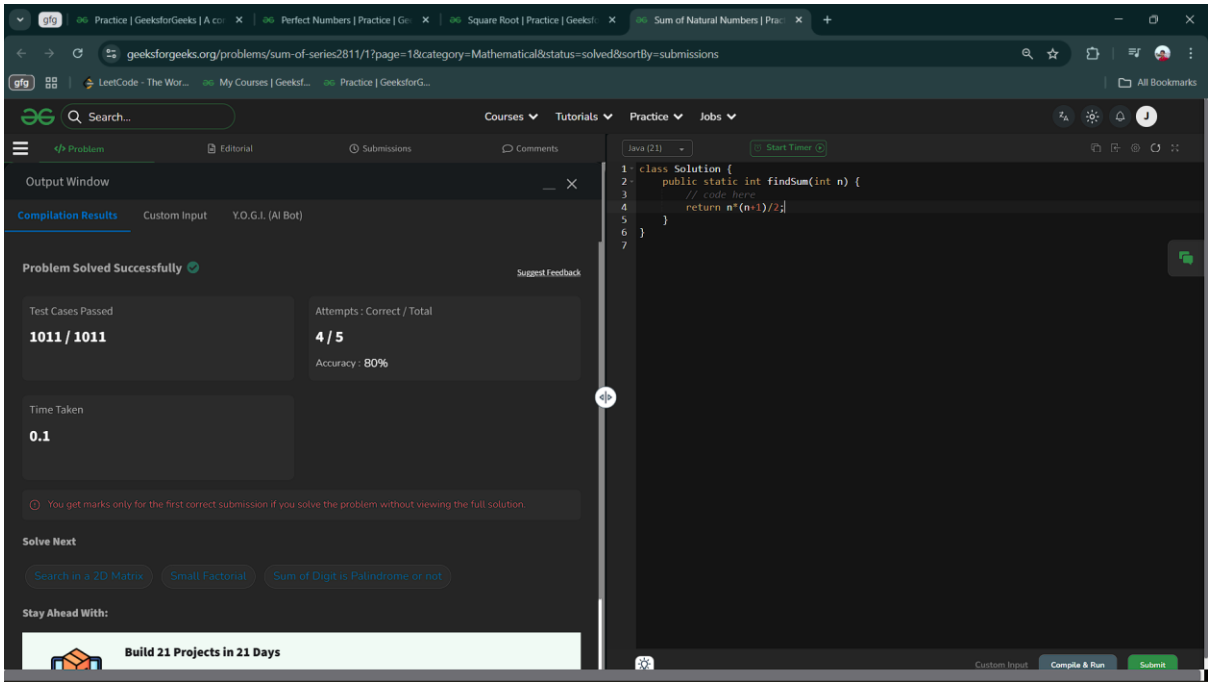
1.Programs on Mathematical reasoning .

1.1 Sum of Natural Numbers

Aim : Given an integer n. Your task is to calculate the sum of all natural numbers from 1 up to n (inclusive). If n is 0, the sum should be 0

```
class Solution {
    public static int findSum(int n) {
        return n*(n+1)/2;
    }
}
```

Output :



1.2 Square Root

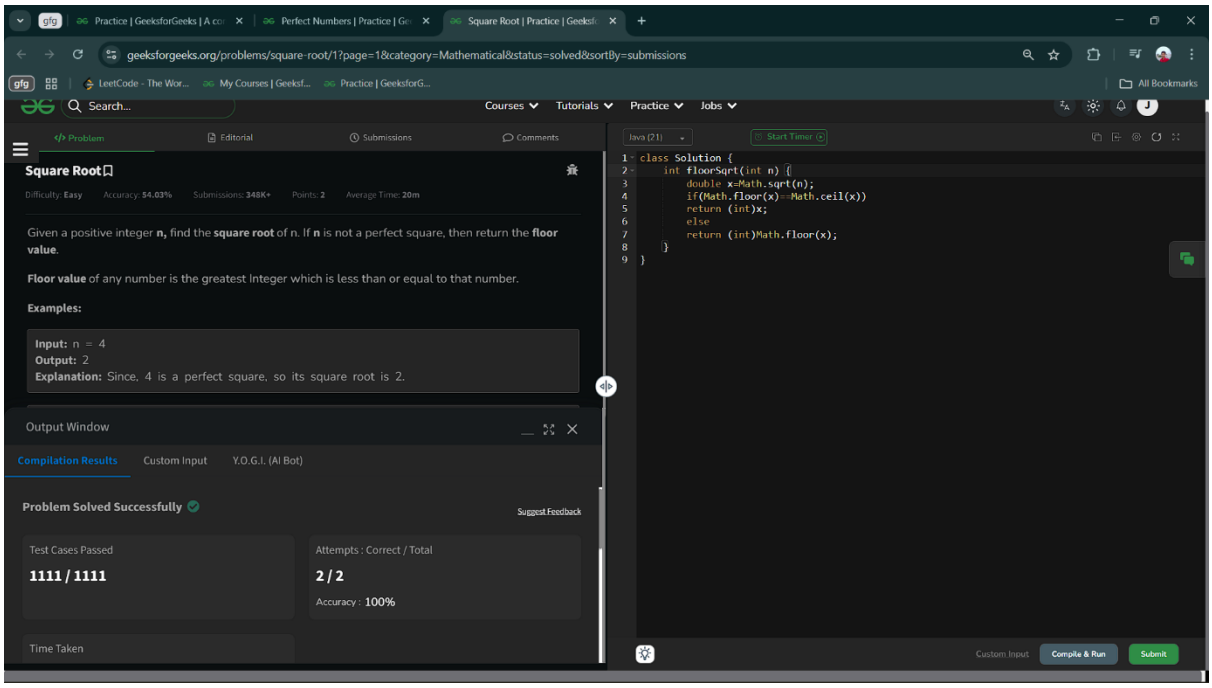
Aim : Given a positive integer n, find the square root of n. If n is not a perfect square, then return the floor value.

Floor value of any number is the greatest Integer which is less than or equal to that number.

Code :

```
class Solution {
    int floorSqrt(int n) {
        double x=Math.sqrt(n);
        if(Math.floor(x)==Math.ceil(x))
        return (int)x;
        else
        return (int)Math.floor(x);
    }
}
```

Output :



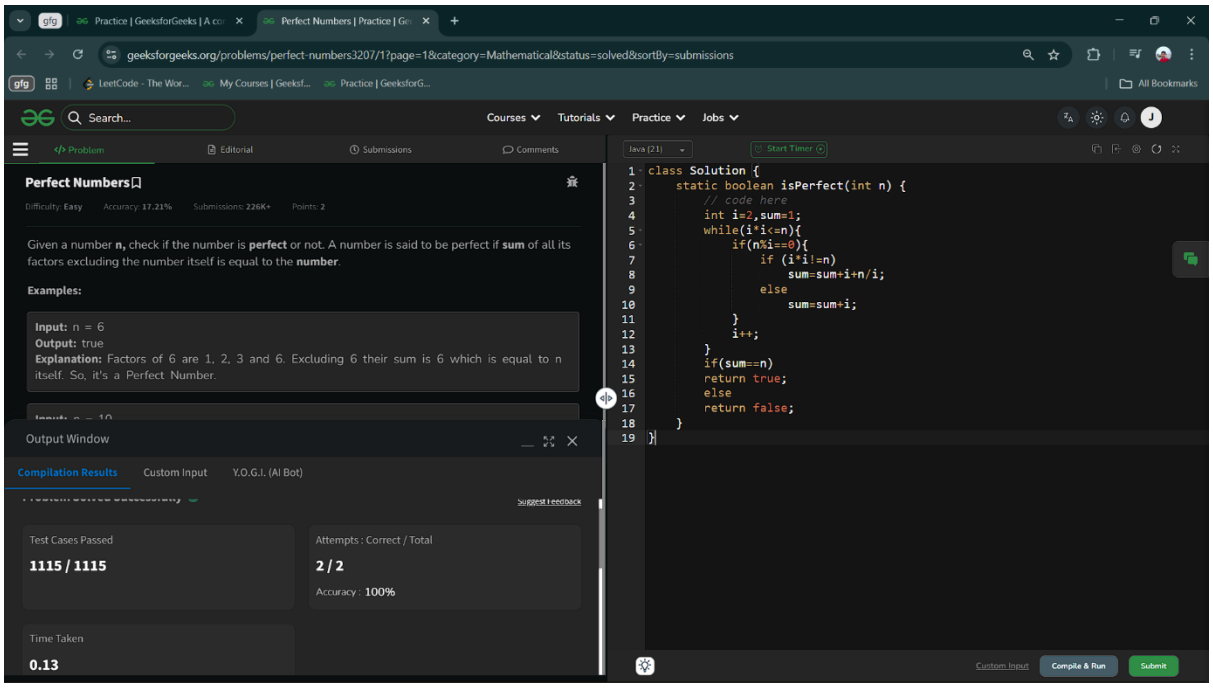
1.3 Perfect Numbers

Aim : Given a number n, check if the number is perfect or not. A number is said to be perfect if sum of all its factors excluding the number itself is equal to the number.

Code :

```
class Solution {
    static boolean isPerfect(int n) {
        // code here
        int i=2,sum=1;
        while(i*i<=n){
            if(n%i==0){
                if (i*i!=n)
                    sum=sum+i+n/i;
                else
                    sum=sum+i;
            }
            i++;
        }
        if(sum==n)
        return true;
        else
        return false;
    }
}
```

Output :



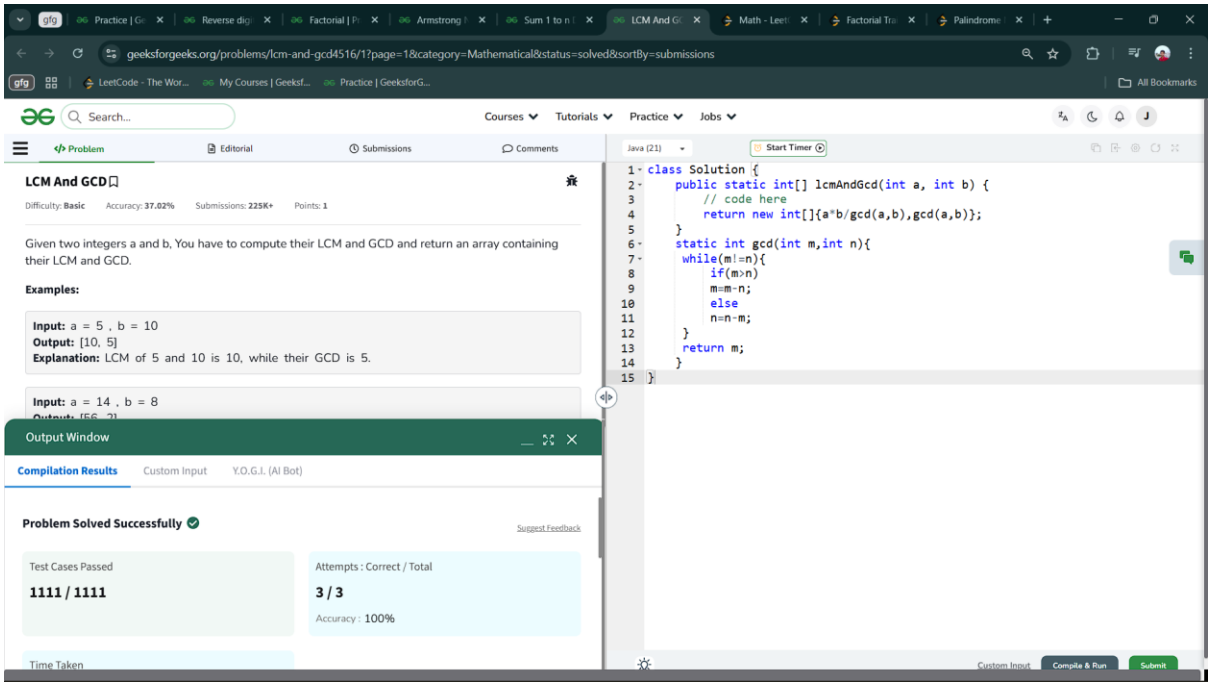
1.4 LCM And GCD

Aim : Given two integers a and b, You have to compute their LCM and GCD and return an array containing their LCM and GCD.

Code :

```
class Solution {
    public static int[] lcmAndGcd(int a, int b) {
        // code here
        return new int[]{a*b/gcd(a,b),gcd(a,b)};
    }
    static int gcd(int m,int n){
     while(m!=n){
         if(m>n)
         m=m-n;
         else
         n=n-m;
     }
     return m;
    }
}
```

Output :



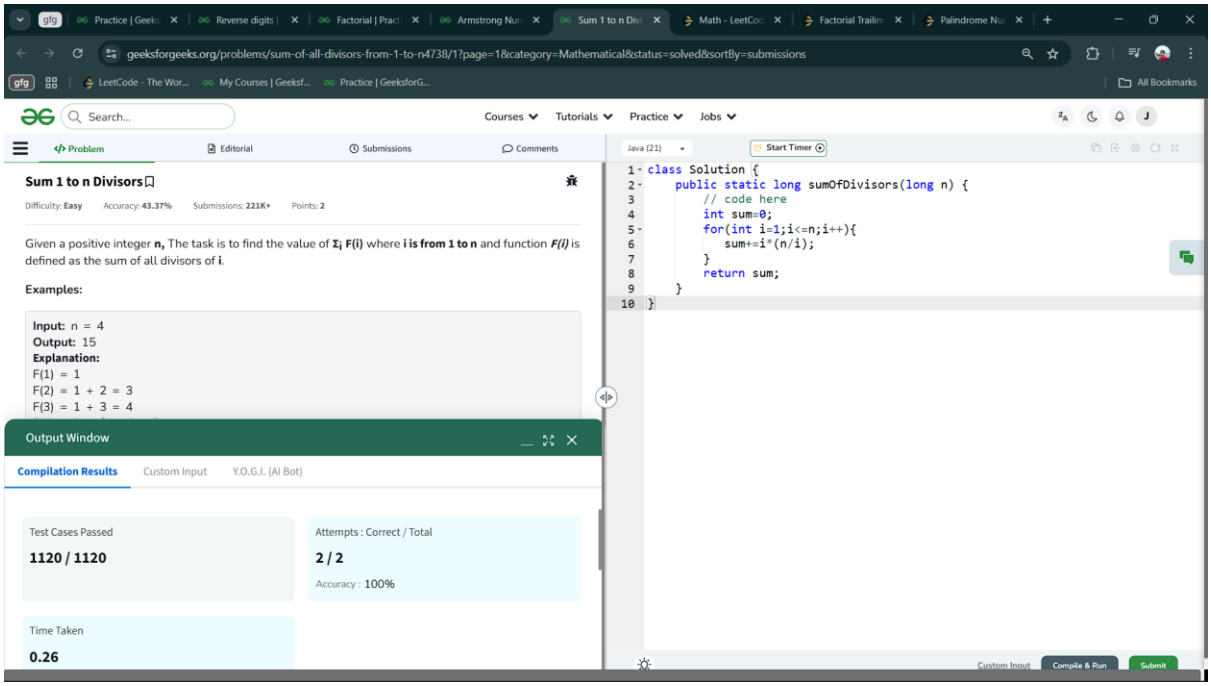
1.5 Sum 1 to n Divisors

Aim : Given a positive integer n, The task is to find the value of Σ_i F(i) where i is from 1 to n and function *F(i)* is defined as the sum of all divisors of i.

Code :

```
class Solution {
    public static long sumOfDivisors(long n) {
        // code here
        int sum=0;
        for(int i=1;i<=n;i++){
            sum+=i*(n/i);
        }
        return sum;
    }
}
```

Output :



1.6 Armstrong Numbers

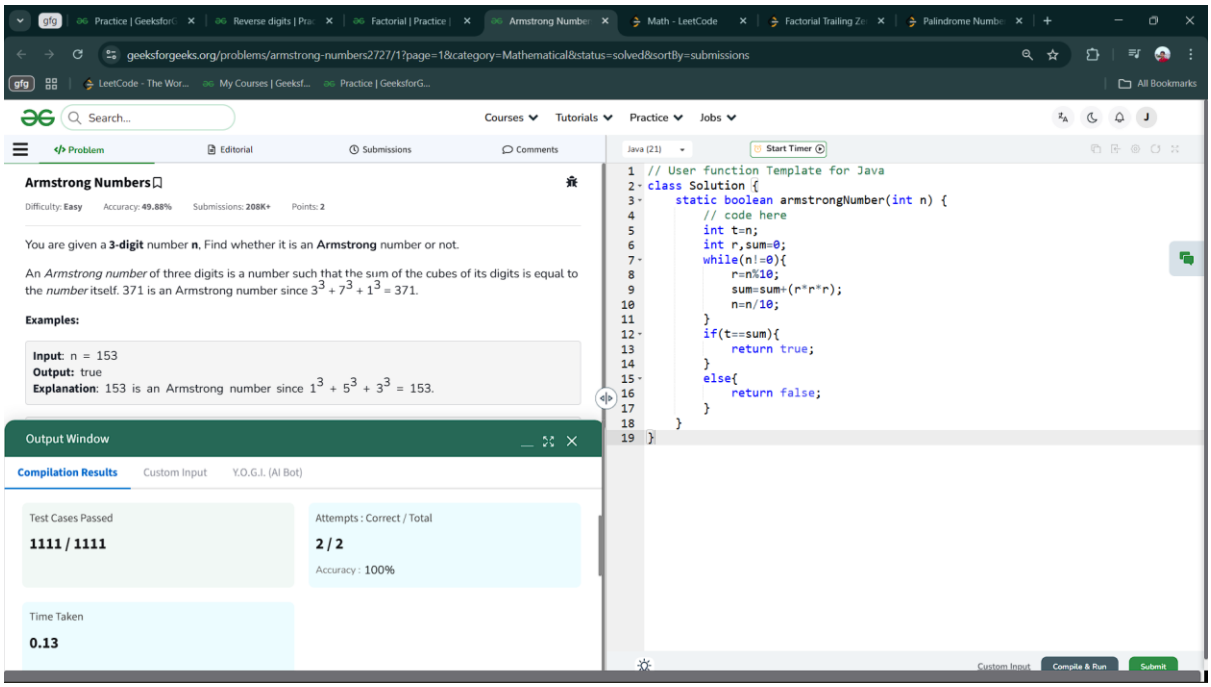
Aim : You are given a 3-digit number n, Find whether it is an Armstrong number or not.

An *Armstrong number* of three digits is a number such that the sum of the cubes of its digits is equal to the *number* itself. 371 is an Armstrong number since $3^3 + 7^3 + 1^3 = 371$

Code :

```
class Solution {
    static boolean armstrongNumber(int n) {
        // code here
        int t=n;
        int r,sum=0;
        while(n!=0){
            r=n%10;
            sum=sum+(r*r*r);
            n=n/10;
        }
        if(t==sum){
            return true;
        }
        else{
            return false;
        }
    }
}
```

Output :



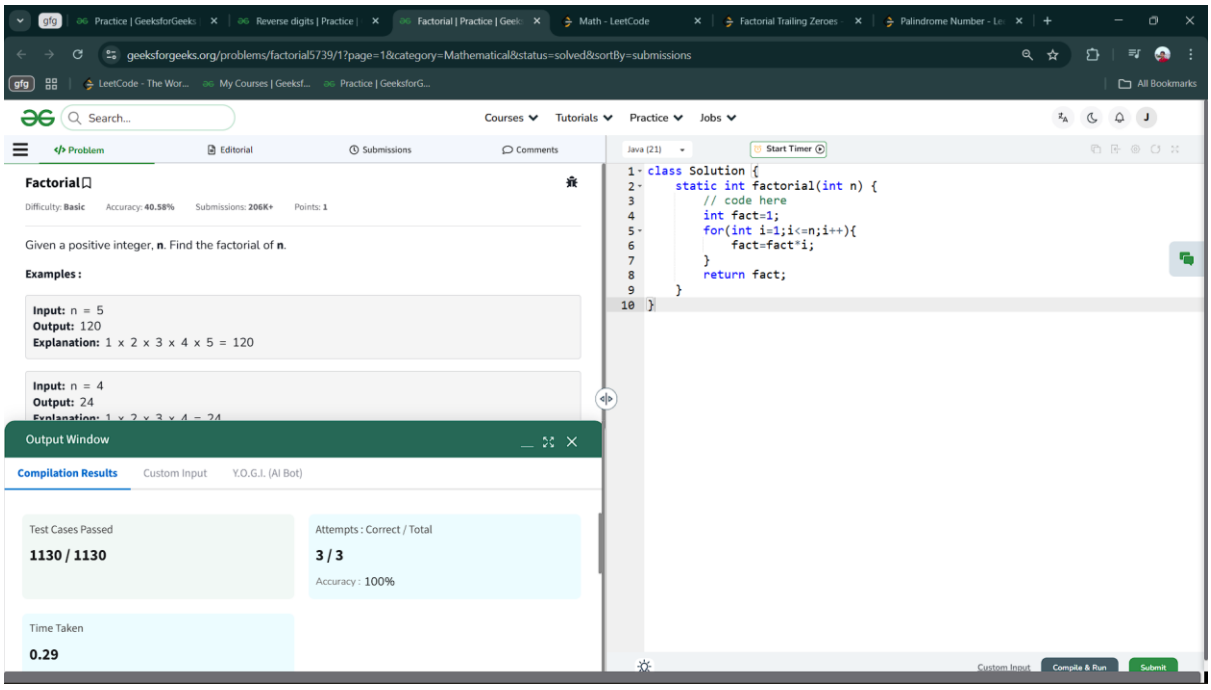
1.7 Factorial

Aim : Given a positive integer, n. Find the factorial of n

Code :

```
class Solution {
    static int factorial(int n) {
        // code here
        int fact=1;
        for(int i=1;i<=n;i++){
            fact=fact*i;
        }
        return fact;
    }
}
```

Output ;



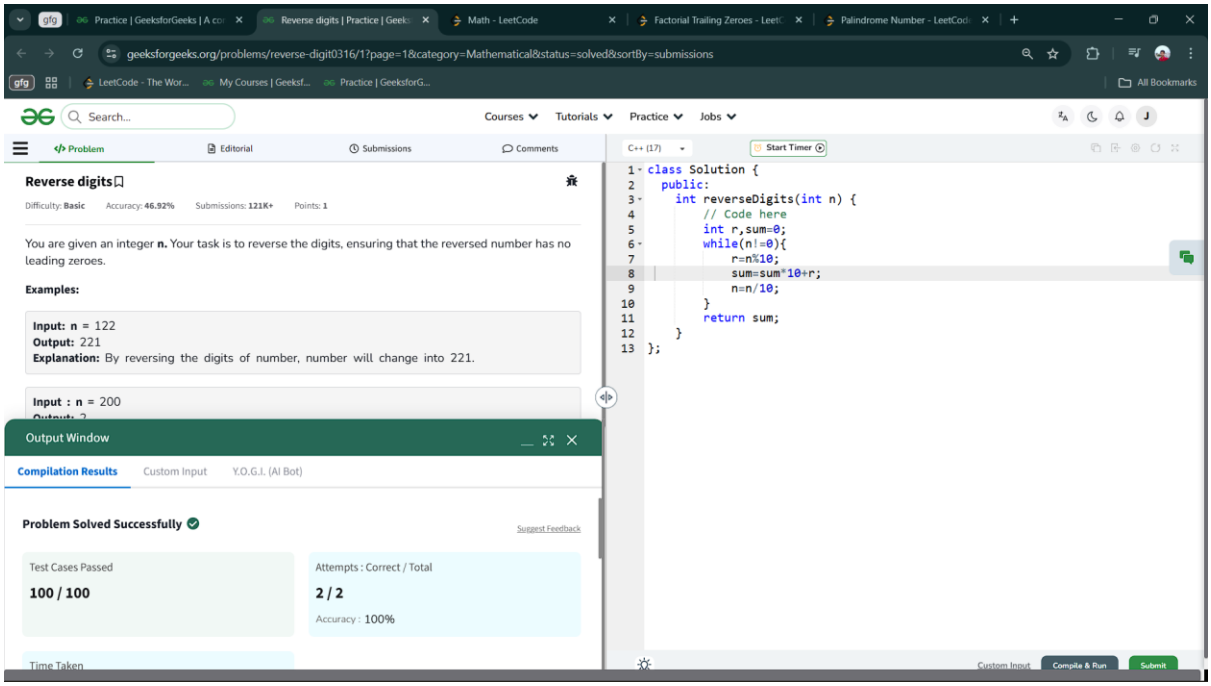
1.8 Reverse digits

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

Code :

```
class Solution {
  public:
    int reverseDigits(int n) {
        // Code here
        int r,sum=0;
        while(n!=0){
            r=n%10;
            sum=sum*10+r;
            n=n/10;
        }
        return sum;
    }
};
```

Output :



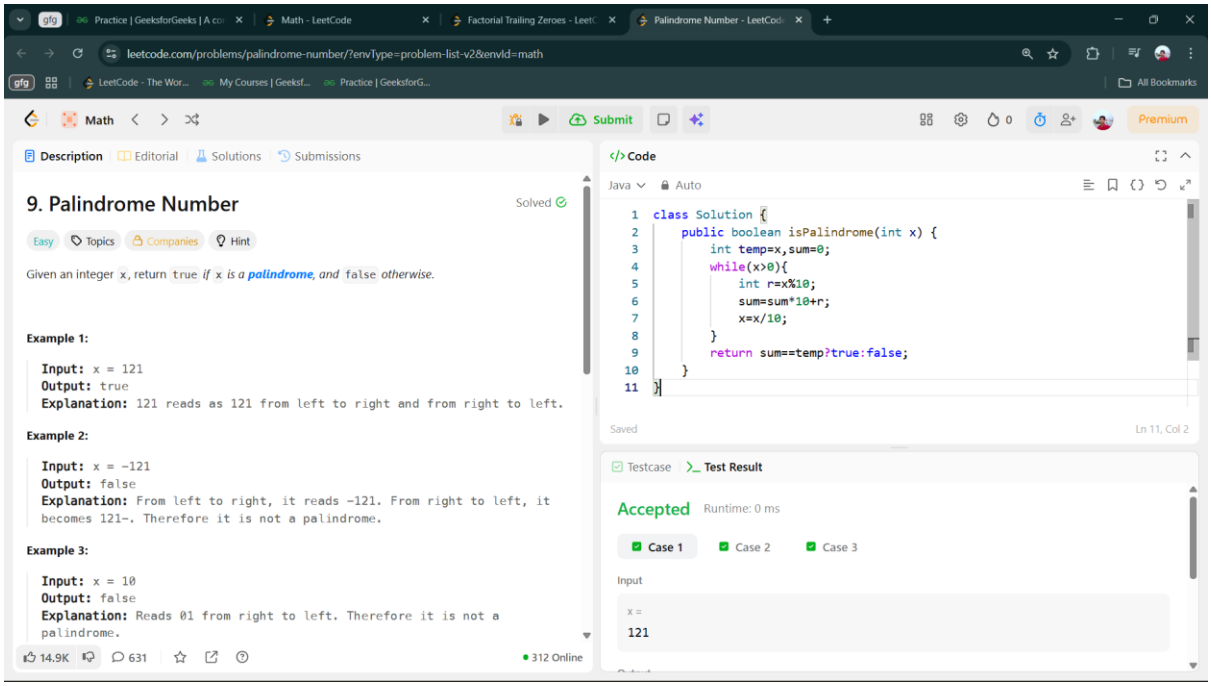
1.9 Palindromic Number

Aim : Given an integer x, return true *if* x *is a palindrome, and* false *otherwise.*

Code :

```
class Solution {
    public boolean isPalindrome(int x) {
        int temp=x,sum=0;
        while(x>0){
            int r=x%10;
            sum=sum*10+r;
            x=x/10;
        }
        return sum==temp?true:false;
    }
}
```

Output :



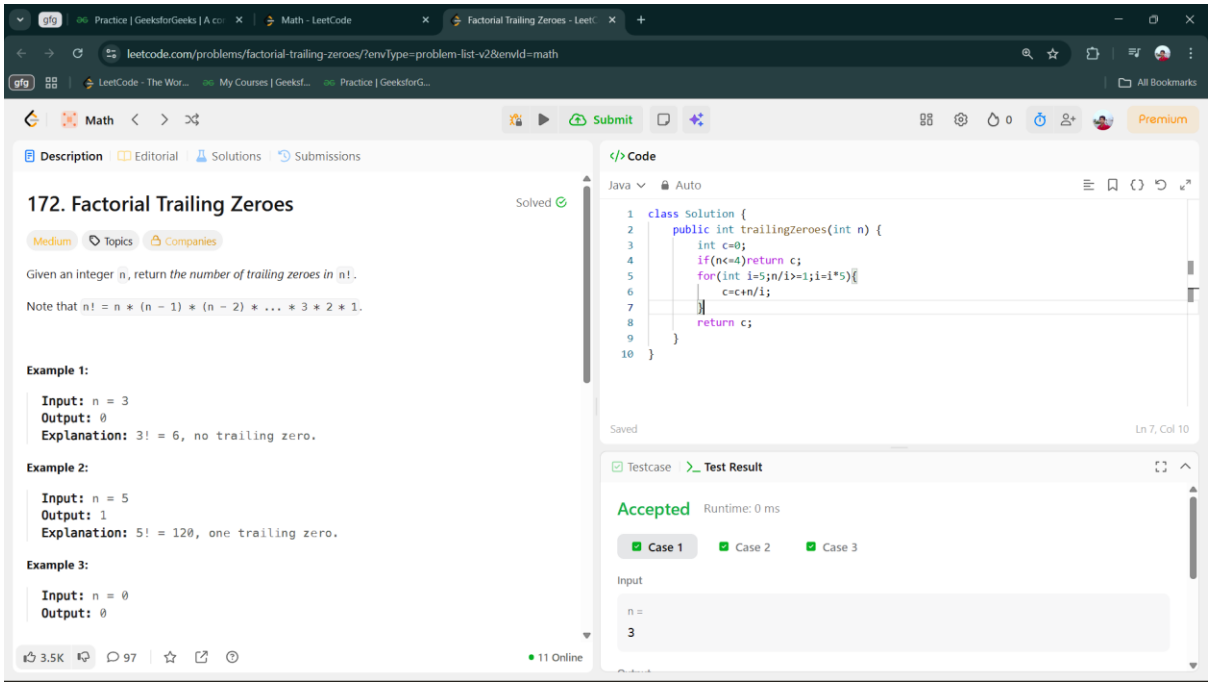
1.10 Factorial Trailing Zeros

Aim : Given an integer n, return *the number of trailing zeroes in* n!.

Code :

```
class Solution {
    public int trailingZeroes(int n) {
        int c=0;
        if(n<=4) return c;
        for(int i=5;n/i>=1;i=i*5){
            c=c+n/i;
        }
        return c;
    }
}
```

Output :



2.Programs on Prime Numbers

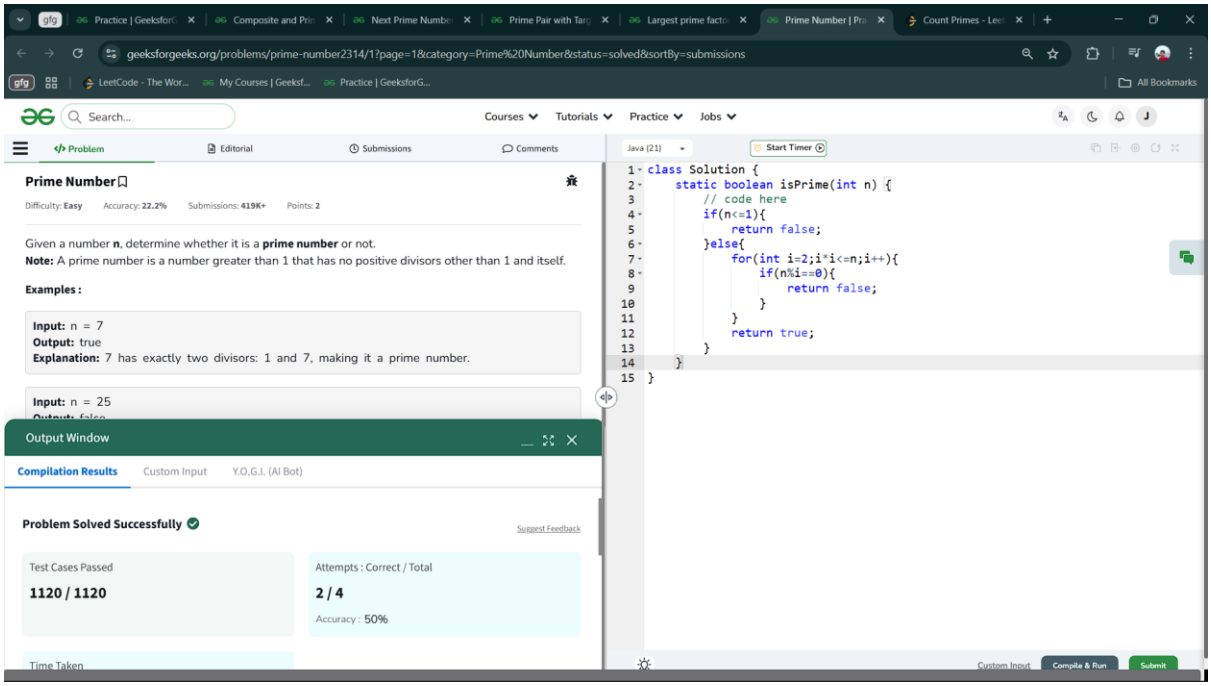
2.1 Prime Number

Aim : 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.

Code :

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

Output :



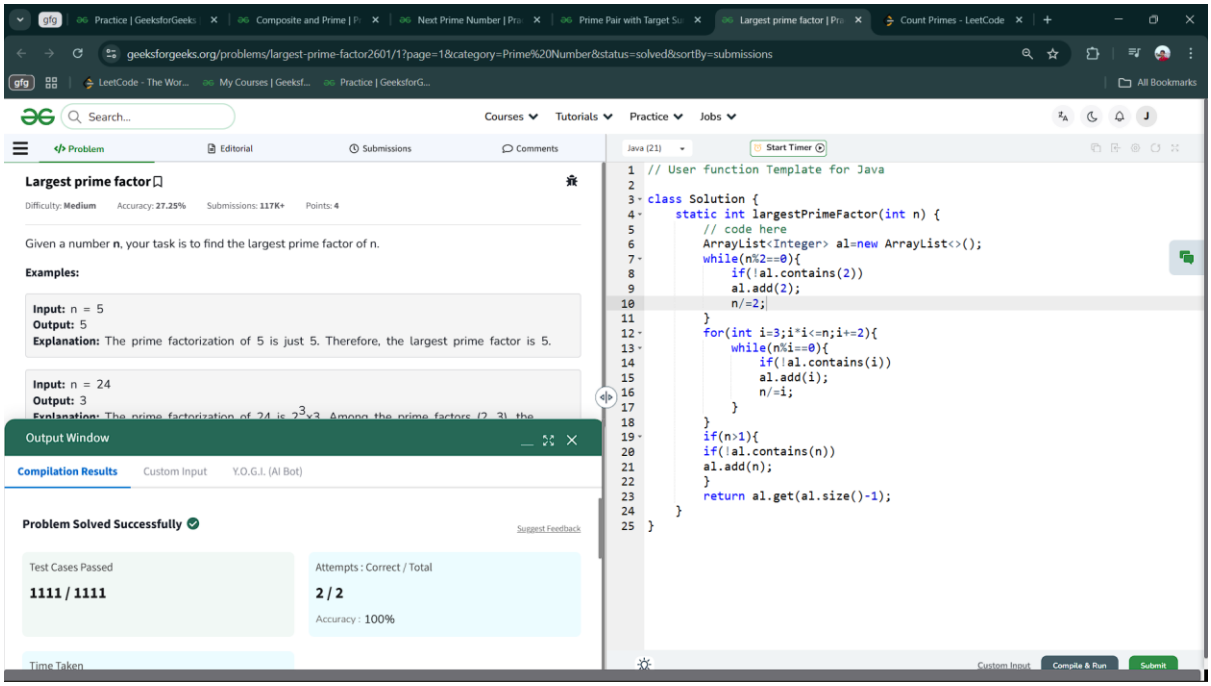
2.2 Largest prime factor

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

Code :

```
class Solution {
    static int largestPrimeFactor(int n) {
        ArrayList<Integer> al=new ArrayList<>();
        while(n%2==0){
            if(!al.contains(2))
            al.add(2);
            n/=2;
        }
        for(int i=3;i*i<=n;i+=2){
            while(n%i==0){
                if(!al.contains(i))
                al.add(i);
                n/=i;
            }
        }
        if(n>1){
        if(!al.contains(n))
        al.add(n);
        }
        return al.get(al.size()-1);
    }
}
```

Output :



2.3 Prime Pair with Target Sum

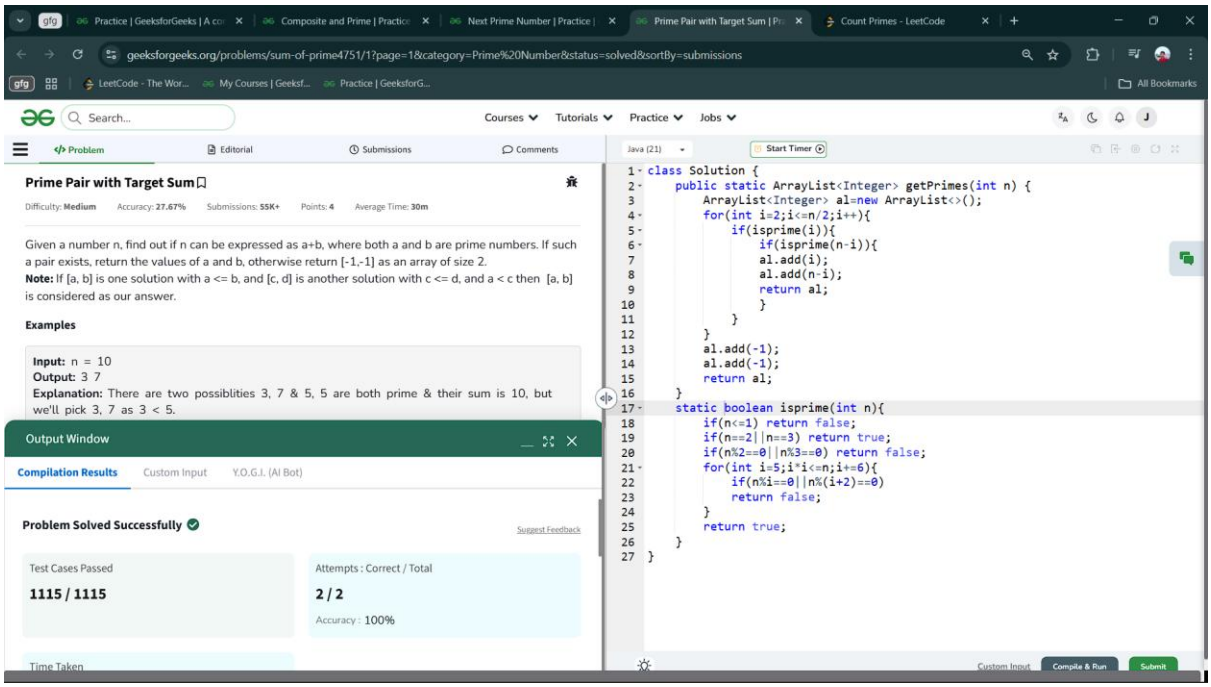
Aim : Given a number n, find out if n can be expressed as a+b, where both a and b are prime numbers. If such a pair exists, return the values of a and b, otherwise return [-1,-1] as an array of size 2.

Code :

```
class Solution {
    public static ArrayList<Integer> getPrimes(int n) {
        ArrayList<Integer> al=new ArrayList<>();
        for(int i=2;i<=n/2;i++){
            if(isprime(i)){
                if(isprime(n-i)){
                al.add(i);
                al.add(n-i);
                return al;
                }
            }
        }
        al.add(-1);
        al.add(-1);
        return al;
    }
```

```
static boolean isprime(int n){
        if(n<=1) return false;
        if(n==2||n==3) return true;
        if(n%2==0||n%3==0) return
false;
        for(int i=5;i*i<=n;i+=6){
            if(n%i==0||n%(i+2)==0)
            return false;
        }
        return true;
    }
`
```

Output :



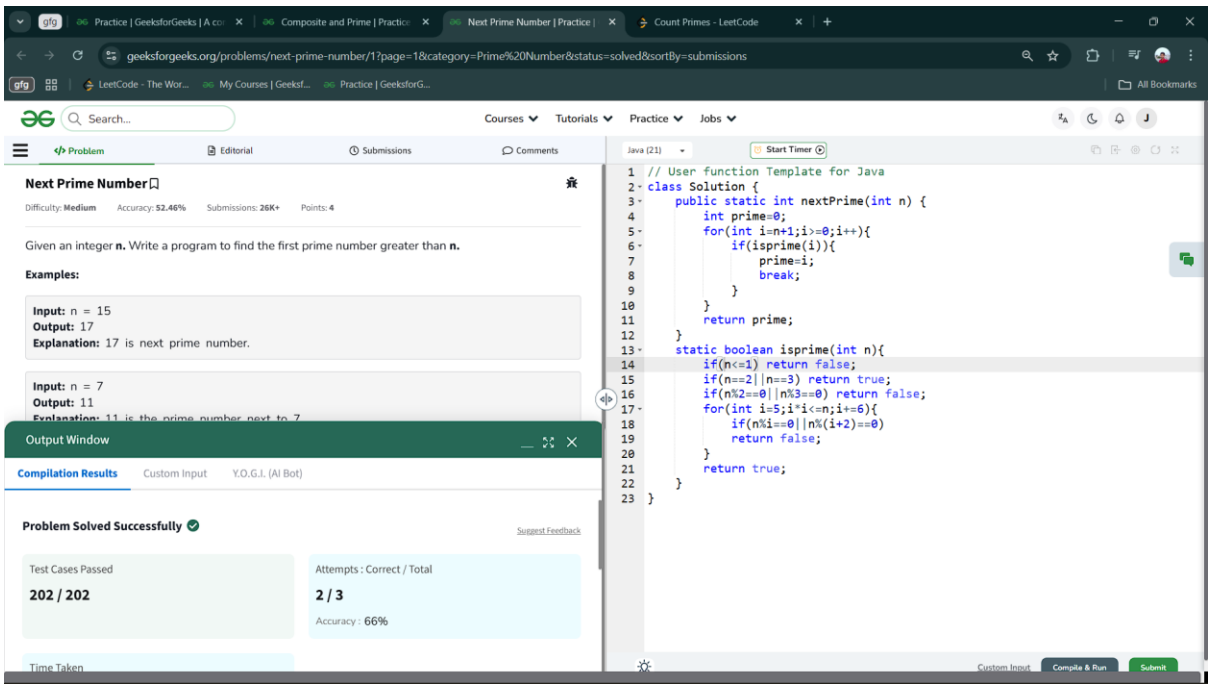
2.4 Next Prime Number

Aim : Given an integer n. Write a program to find the first prime number greater than n.

Code :

```
class Solution {
    public static int nextPrime(int n) {
        int prime=0;
        for(int i=n+1;i>=0;i++){
            if(isprime(i)){
                prime=i;
                break;
            }
        }
        return prime;
    }
    static boolean isprime(int n){
        if(n<=1) return false;
        if(n==2||n==3) return true;
        if(n%2==0||n%3==0) return false;
        for(int i=5;i*i<=n;i+=6){
            if(n%i==0||n%(i+2)==0)
            return false;
        }
        return true;
    }
}
```

Output :

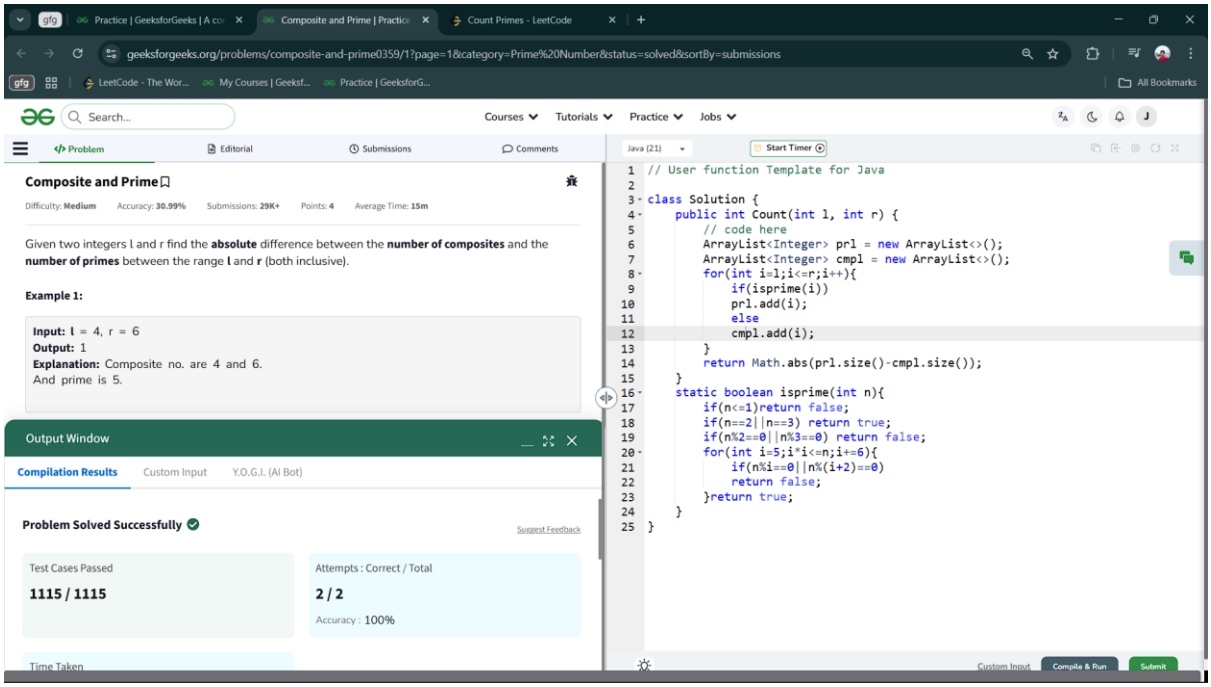


2.5 Composite and Prime

Aim : Given two integers l and r find the absolute difference between the number of composites and the number of primes between the range l and r (both inclusive).

```
class Solution {
    public int Count(int l, int r) {
        ArrayList<Integer> prl = new ArrayList<>();
        ArrayList<Integer> cmpl = new ArrayList<>();
        for(int i=l;i<=r;i++){
            if(isprime(i))
            prl.add(i);
            else
            cmpl.add(i);
        }
        return Math.abs(prl.size()-cmpl.size());
    }
    static boolean isprime(int n){
        if(n<=1)return false;
        if(n==2||n==3) return true;
        if(n%2==0||n%3==0) return false;
        for(int i=5;i*i<=n;i+=6){
            if(n%i==0||n%(i+2)==0)
            return false;
        }return true;
    }
}
```

Output :



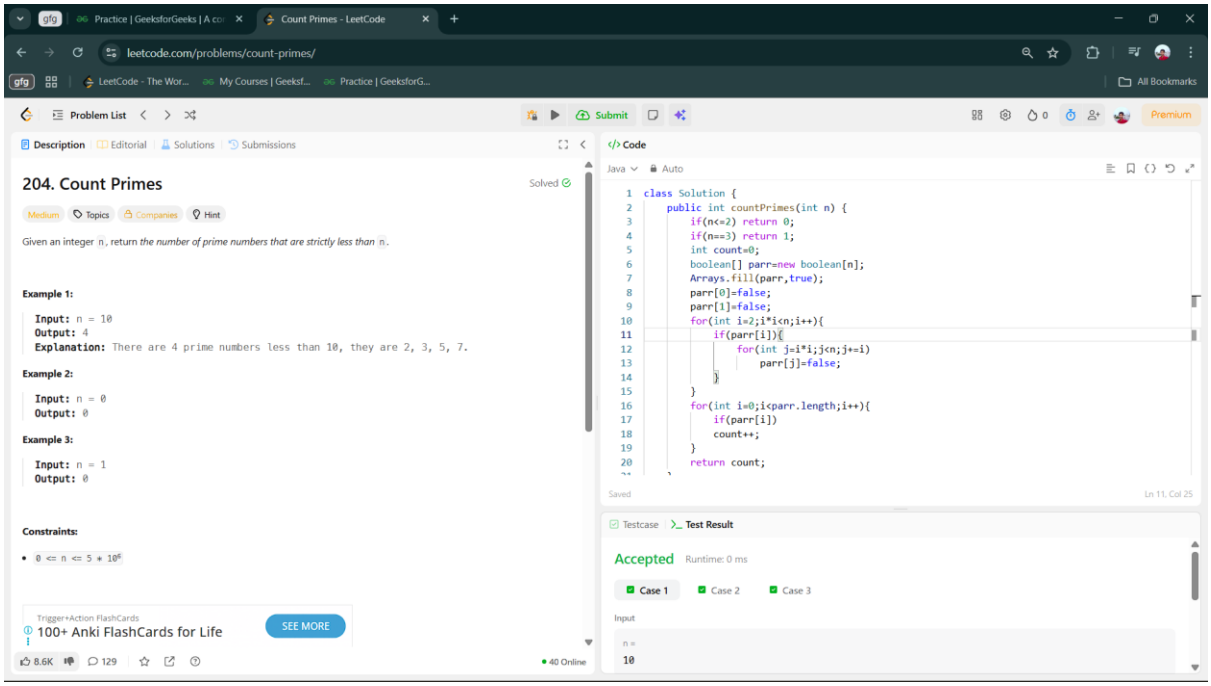
2.6 Count Prime(using Seive)

Aim :Given an integer n, return *the number of prime numbers that are strictly less than* n.

Code :

```
class Solution {
    public int countPrimes(int n) {
        if(n<=2) return 0;
        if(n==3) return 1;
        int count=0;
        boolean[] parr=new boolean[n];
        Arrays.fill(parr,true);
        parr[0]=false;
        parr[1]=false;
        for(int i=2;i*i<n;i++){
            if(parr[i]){
                for(int j=i*i;j<n;j+=i)
                    parr[j]=false;
            }
        }
        for(int i=0;i<parr.length;i++){
            if(parr[i])
            count++;
        }
        return count;
    }
}
```

Output :



3.Problems on Fibonacci Series :

3.1 Fibonacci Number

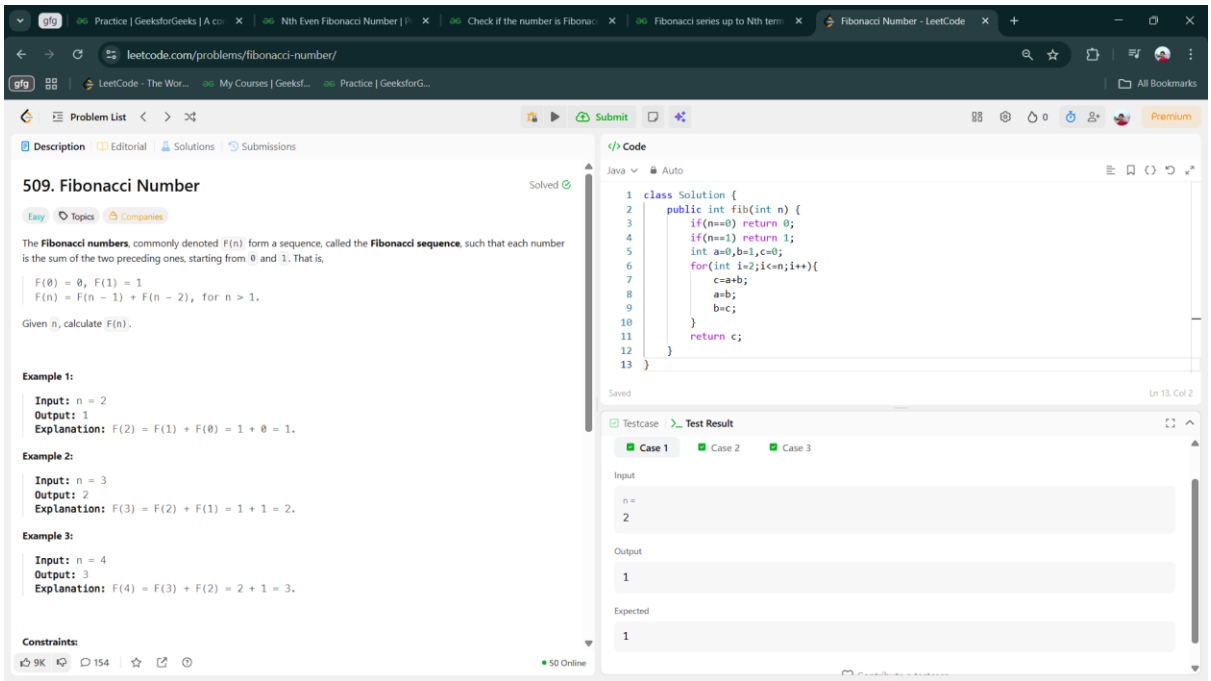
Aim : The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

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

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

```
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 c;
    }
}
```

Output :

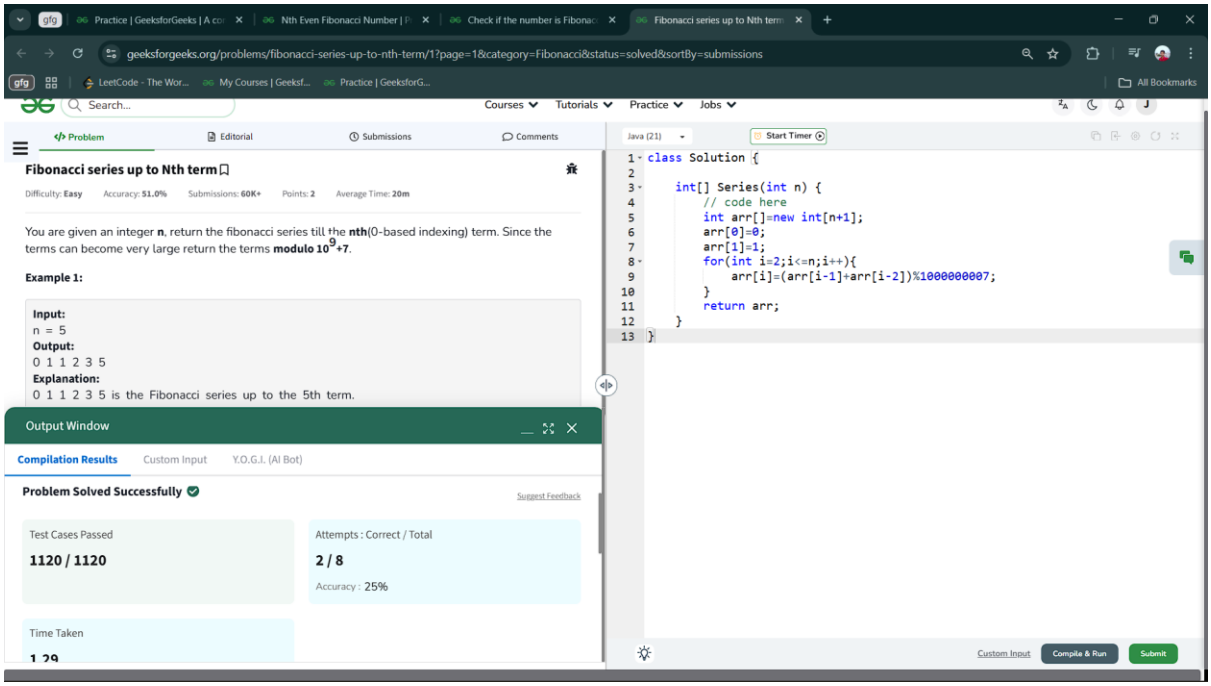


3.2 Fibonacci series up to Nth term

Aim : 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 {
    int[] Series(int n) {
        // code here
        int arr[]=new int[n+1];
        arr[0]=0;
        arr[1]=1;
        for(int i=2;i<=n;i++){
            arr[i]=(arr[i-1]+arr[i-2])%1000000007;
        }
        return arr;
    }
}
```

Output :

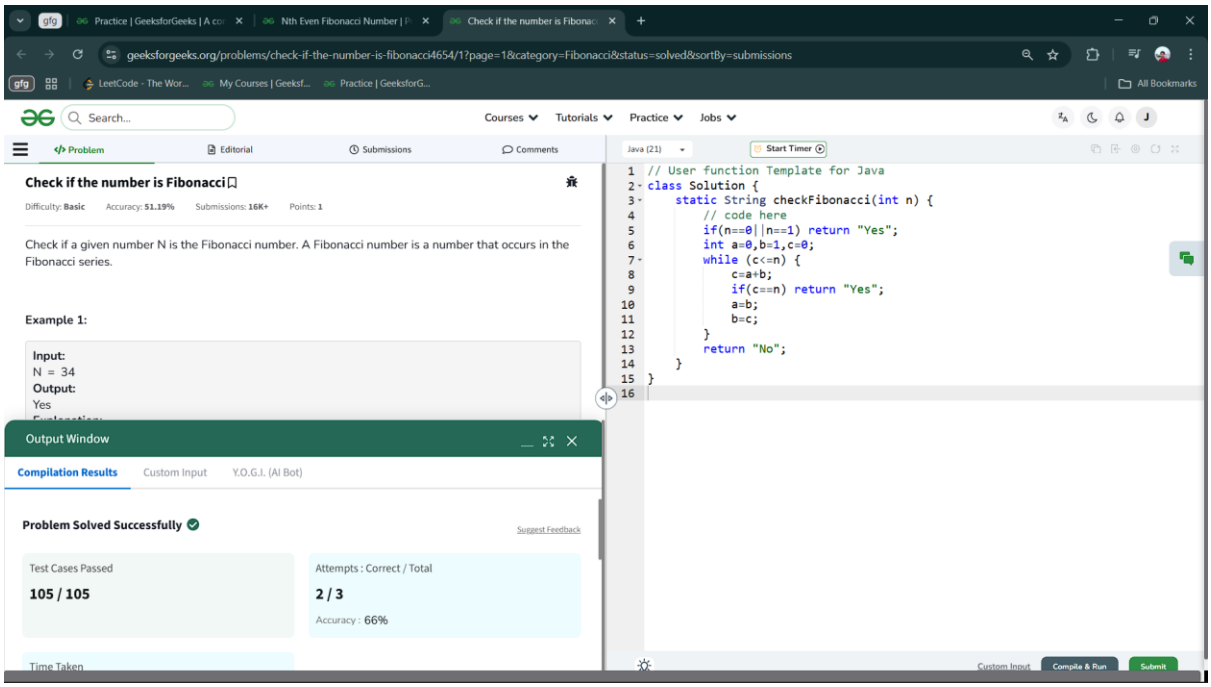


3.3 Check if the number is Fibonacci

Aim : 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) {
        // code here
        if(n==0||n==1) return "Yes";
        int a=0,b=1,c=0;
        while (c<=n) {
            c=a+b;
            if(c==n) return "Yes";
            a=b;
            b=c;
        }
        return "No";
    }
}
```

Output :

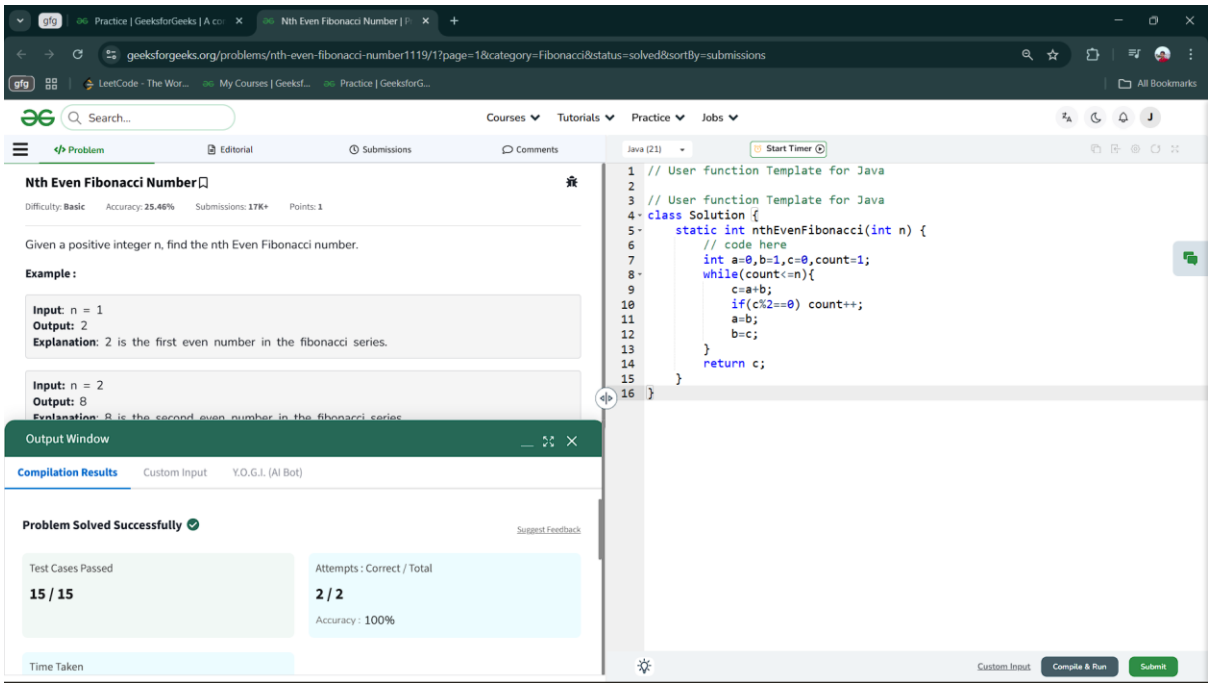


3.4 Nth Even Fibonacci Number

Aim : Given a positive integer n, find the nth Even Fibonacci number.

```
class Solution {
    static int nthEvenFibonacci(int n) {
        // code here
        int a=0,b=1,c=0,count=1;
        while(count<=n){
            c=a+b;
            if(c%2==0) count++;
            a=b;
            b=c;
        }
        return c;
    }
}
```

Output :



4.Programs on Arrays:

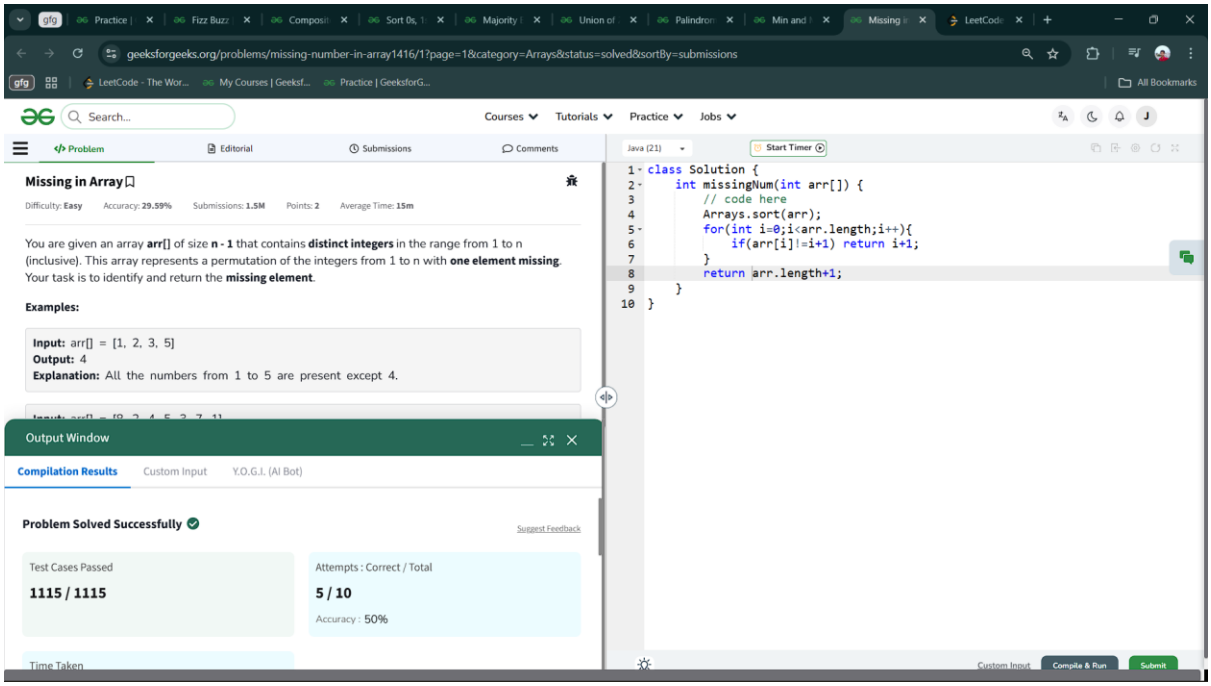
4.1 Missing in Array

Aim : 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.

Code :

```
class Solution {
    int missingNum(int arr[]) {
        // code here
        Arrays.sort(arr);
        for(int i=0;i<arr.length;i++){
            if(arr[i]!=i+1) return i+1;
        }
        return arr.length+1;
    }
}
```

Output :



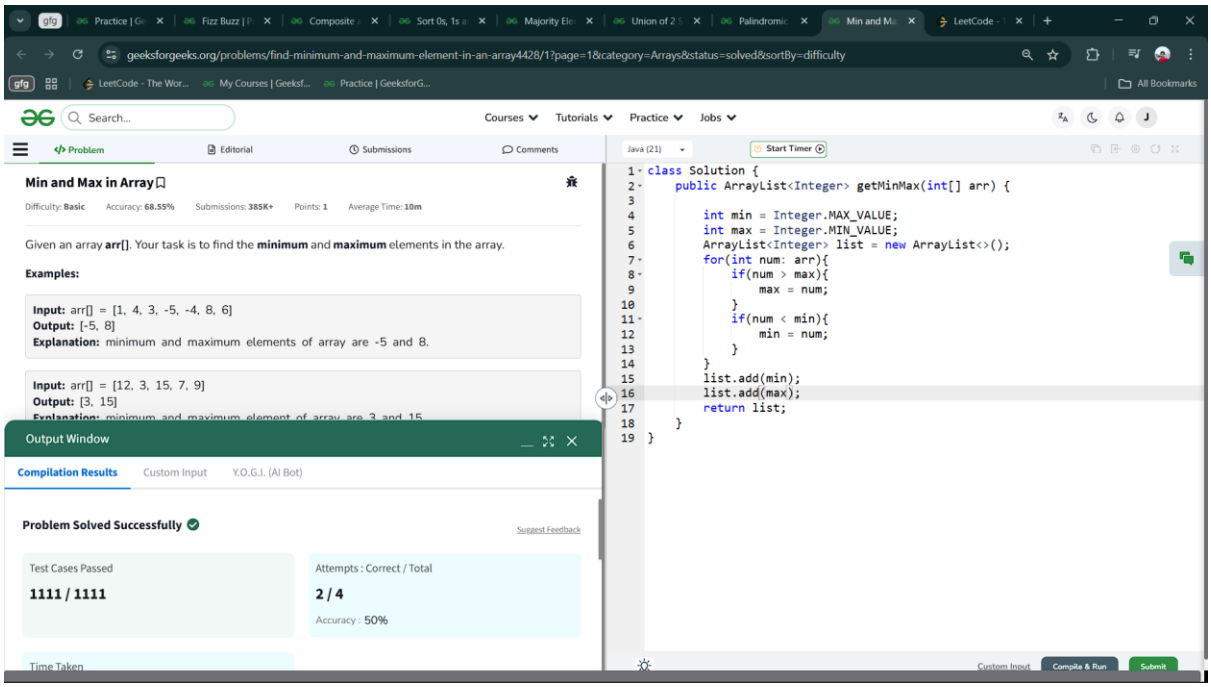
4.2 Min and Max in Array

Aim : Given an array arr[]. Your task is to find the minimum and maximum elements in the array

Code :

```
class Solution {
    public ArrayList<Integer> getMinMax(int[] arr) {
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        ArrayList<Integer> list = new ArrayList<>();
        for(int num: arr){
            if(num > max){
                max = num;
            }
            if(num < min){
                min = num;
            }
        }
        list.add(min);
        list.add(max);
        return list;
    }
}
```

Output ;

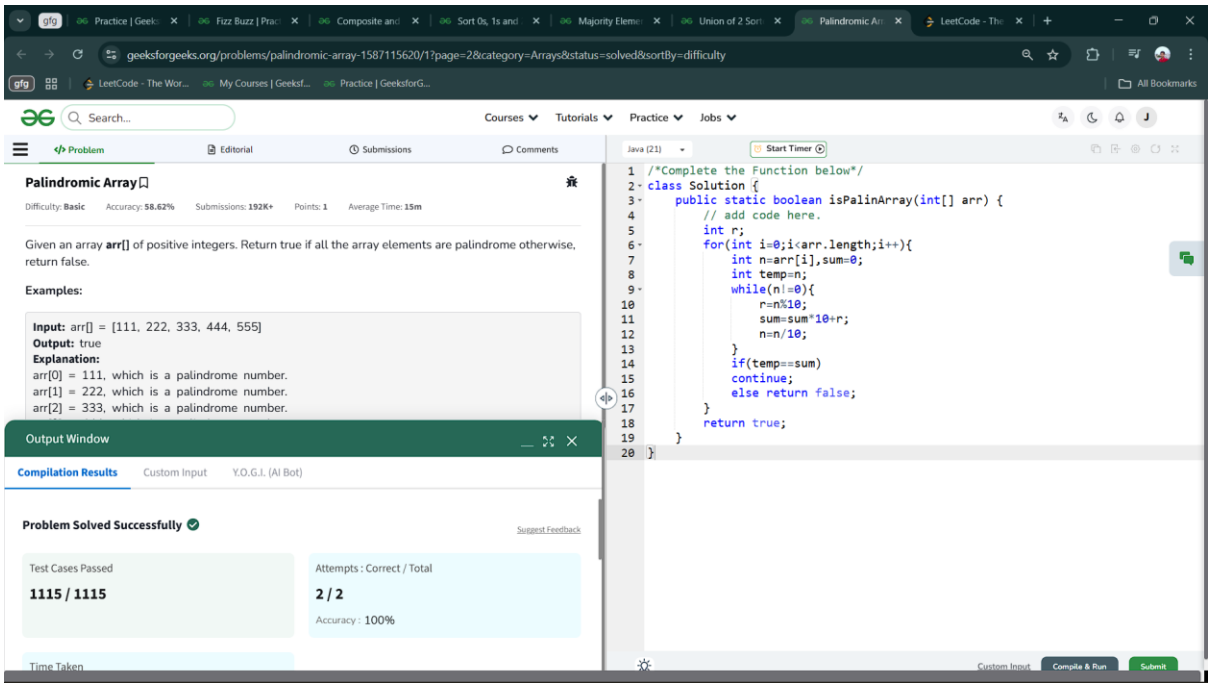


4.3 Palindromic Array

Aim : Given an array arr[] of positive integers. Return true if all the array elements are palindrome otherwise, return false.

```
class Solution {
    public static boolean isPalinArray(int[] arr) {
        // add code here.
        int r;
        for(int i=0;i<arr.length;i++){
            int n=arr[i],sum=0;
            int temp=n;
            while(n!=0){
                r=n%10;
                sum=sum*10+r;
                n=n/10;
            }
            if(temp==sum)
            continue;
            else return false;
        }
        return true;
    }
}
```

Output :

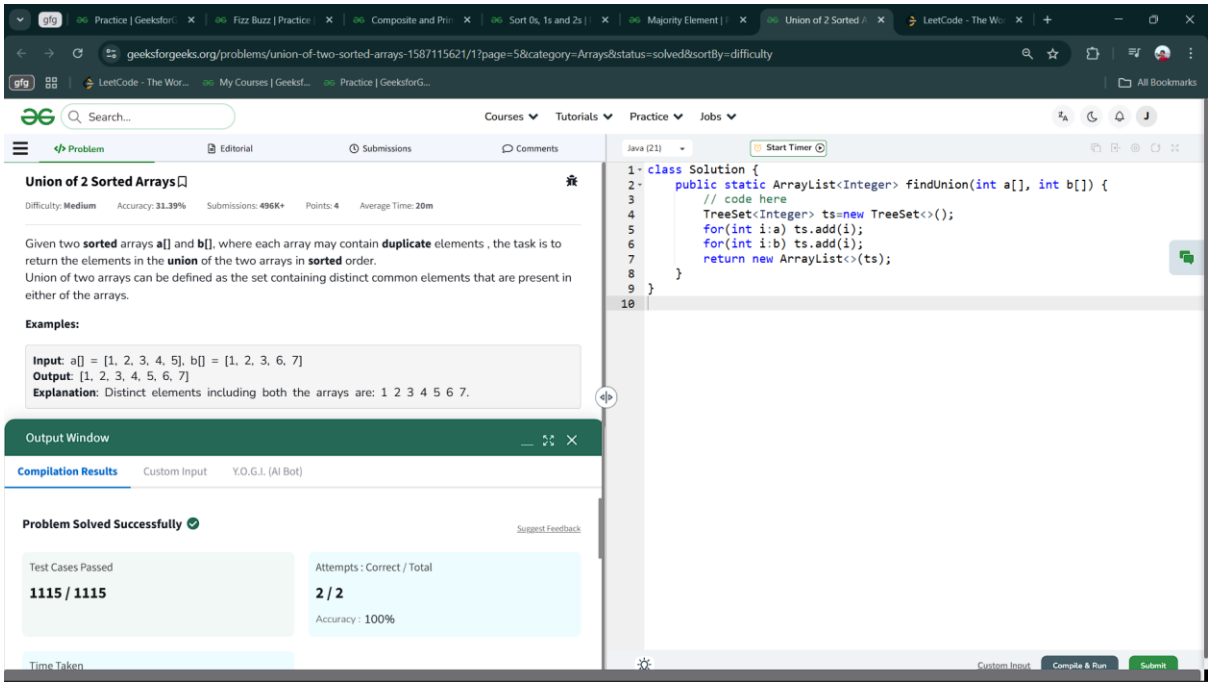


4.4 Union of 2 Sorted Arrays

Aim : Given two sorted arrays a[] and b[], where each array may contain duplicate elements , the task is to return the elements in the union of the two arrays in sorted order.
Union of two arrays can be defined as the set containing distinct common elements that are present in either of the arrays.

```
class Solution {
    public static ArrayList<Integer> findUnion(int a[], int b[]) {
        // code here
        TreeSet<Integer> ts=new TreeSet<>();
        for(int i:a) ts.add(i);
        for(int i:b) ts.add(i);
        return new ArrayList<>(ts);
    }
}
```

Output :

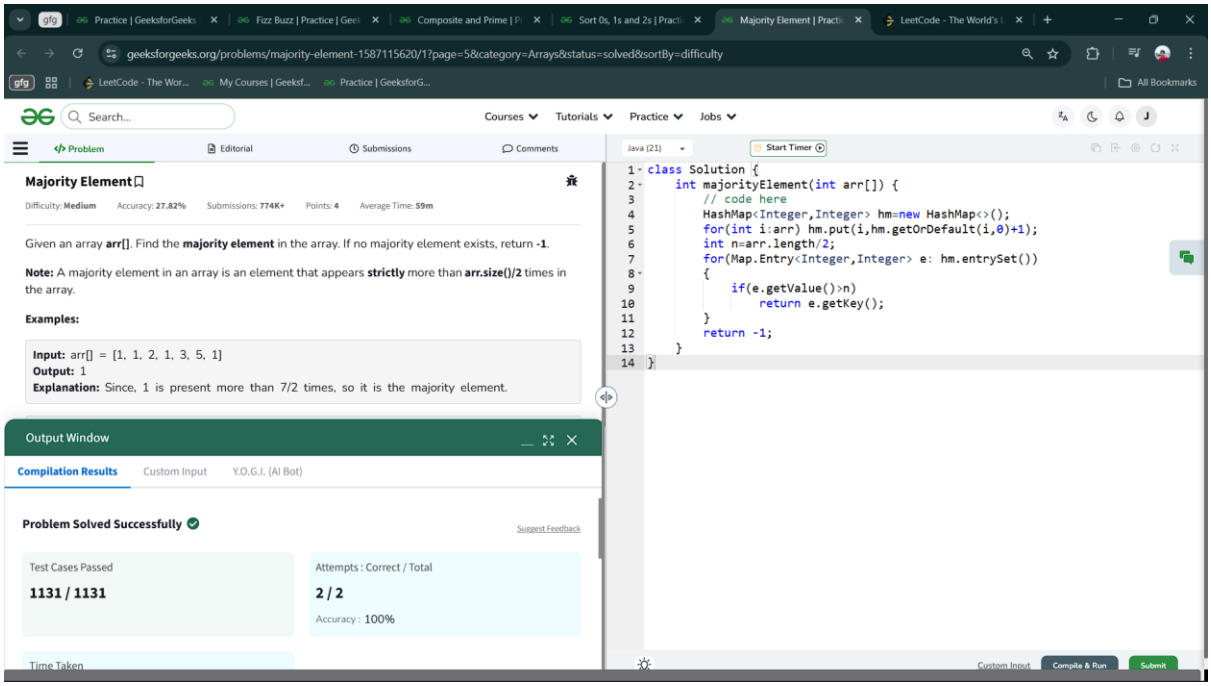


4.5 Majority Element

Aim : Given an array arr[]. Find the majority element in the array. If no majority element exists, return -1.
Note: A majority element in an array is an element that appears strictly more than arr.size()/2 times in the array.

```
class Solution {
    int majorityElement(int arr[]) {
        // code here
        HashMap<Integer,Integer> hm=new HashMap<>();
        for(int i:arr) hm.put(i,hm.getOrDefault(i,0)+1);
        int n=arr.length/2;
        for(Map.Entry<Integer,Integer> e: hm.entrySet())
        {
            if(e.getValue()>n)
                return e.getKey();
        }
        return -1;
    }
}
```

Output :

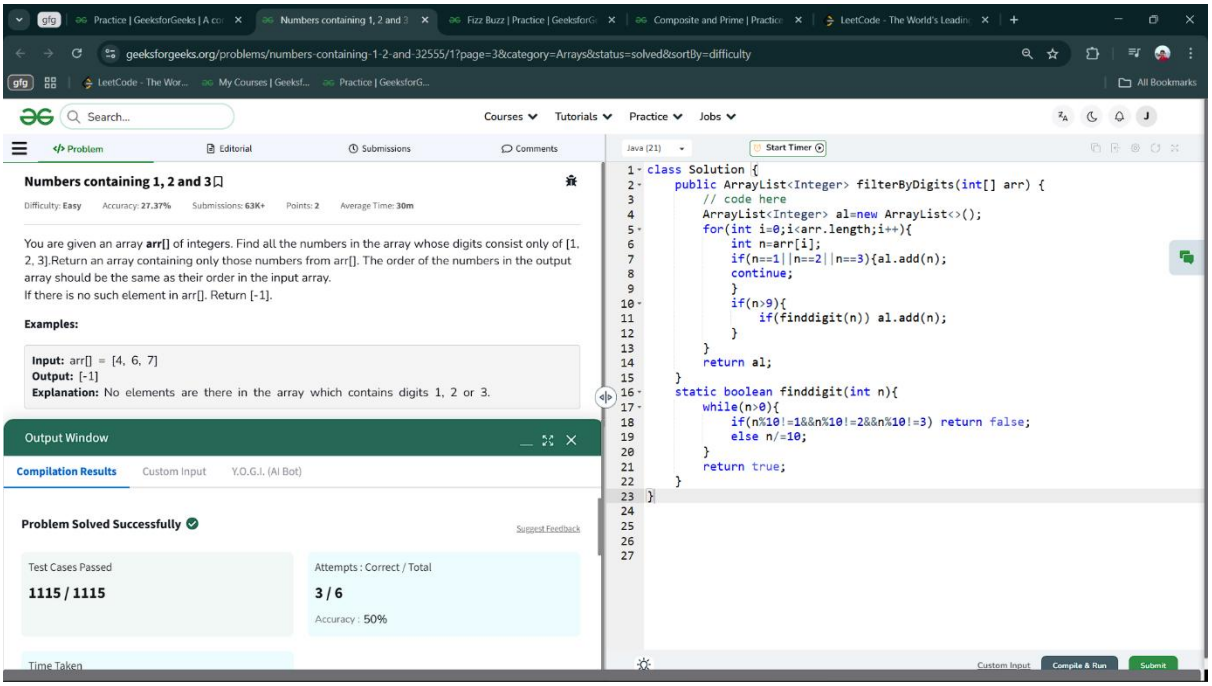


4.6 Numbers containing 1, 2 and 3

Aim : You are given an array arr[] of integers. Find all the numbers in the array whose digits consist only of [1, 2, 3].Return an array containing only those numbers from arr[]. The order of the numbers in the output array should be the same as their order in the input array.If there is no such element in arr[]. Return [-1].

```
class Solution {
    public ArrayList<Integer> filterByDigits(int[] arr) {
        ArrayList<Integer> al=new ArrayList<>();
        for(int i=0;i<arr.length;i++){
            int n=arr[i];
            if(n==1||n==2||n==3){al.add(n);
            continue;
            }
            if(n>9){
                if(finddigit(n)) al.add(n);
            }
        }
        return al;
    }
    static boolean finddigit(int n){
        while(n>0){
            if(n%10!=1&&n%10!=2&&n%10!=3) return false;
            else n/=10;
        }
        return true;
    }
}
```

Output :



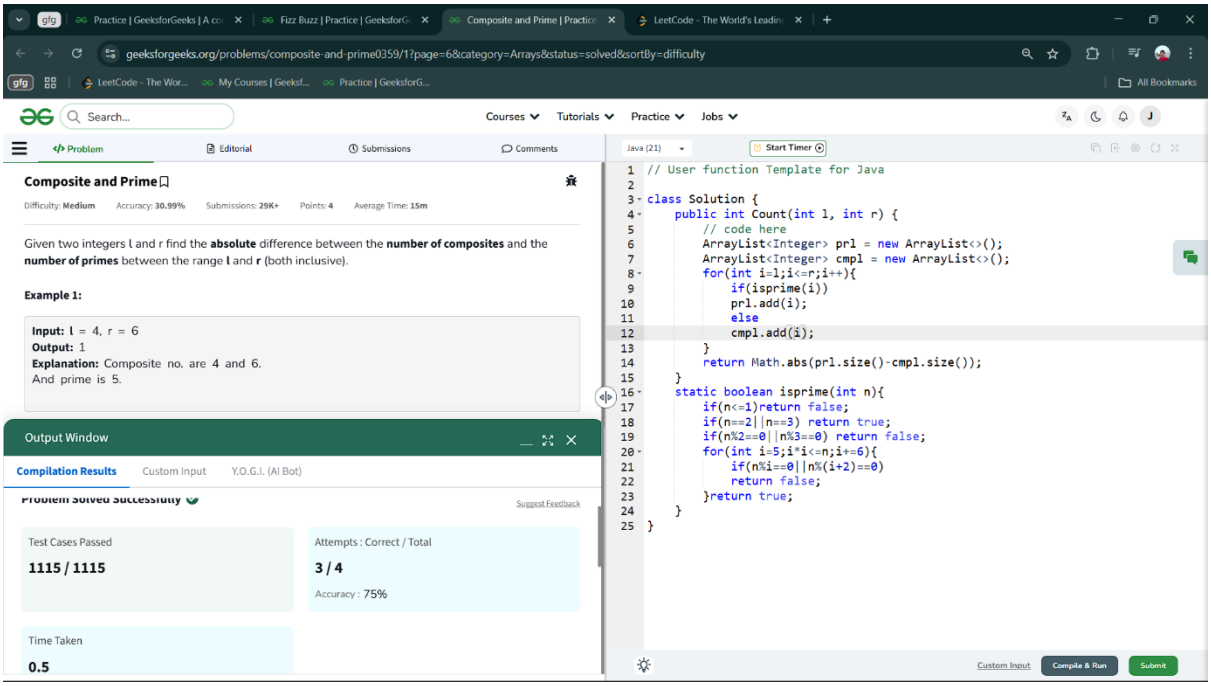
4.7 Composite and Prime

Aim : Given two integers l and r find the absolute difference between the number of composites and the number of primes between the range l and r (both inclusive)

```
class Solution {
    public int Count(int l, int r) {
        ArrayList<Integer> prl = new ArrayList<>();
        ArrayList<Integer> cmpl = new ArrayList<>();
        for(int i=l;i<=r;i++){
            if(isprime(i))
            prl.add(i);
            else
            cmpl.add(i);
        }
        return Math.abs(prl.size()-cmpl.size());
    }
    static boolean isprime(int n){
        if(n<=1)return false;
        if(n==2||n==3) return true;
        if(n%2==0||n%3==0) return false;
        for(int i=5;i*i<=n;i+=6){
            if(n%i==0||n%(i+2)==0)
            return false;
        }return true;
    }
}
```

Output :

4.8 Fizz Buzz

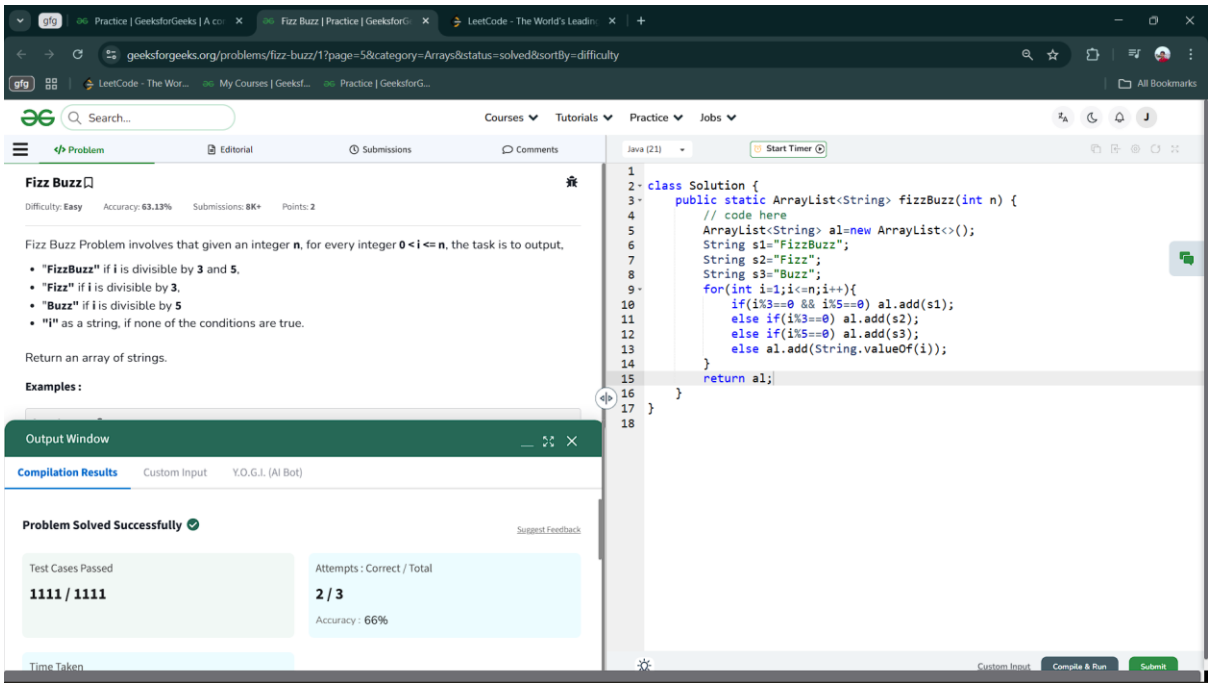


Aim : Fizz Buzz Problem involves that given an integer n, for every integer 0 < i <= n, the task is to output,

- **"FizzBuzz" if i is divisible by 3 and 5,**
- **"Fizz" if i is divisible by 3,**
- **"Buzz" if i is divisible by 5**
- **"i" as a string, if none of the conditions are true.**

```
class Solution {
    public static ArrayList<String> fizzBuzz(int n) {
        // code here
        ArrayList<String> al=new ArrayList<>();
        String s1="FizzBuzz";
        String s2="Fizz";
        String s3="Buzz";
        for(int i=1;i<=n;i++){
            if(i%3==0 && i%5==0) al.add(s1);
            else if(i%3==0) al.add(s2);
            else if(i%5==0) al.add(s3);
            else al.add(String.valueOf(i));
        }
        return al;
    }
}
```

Output :

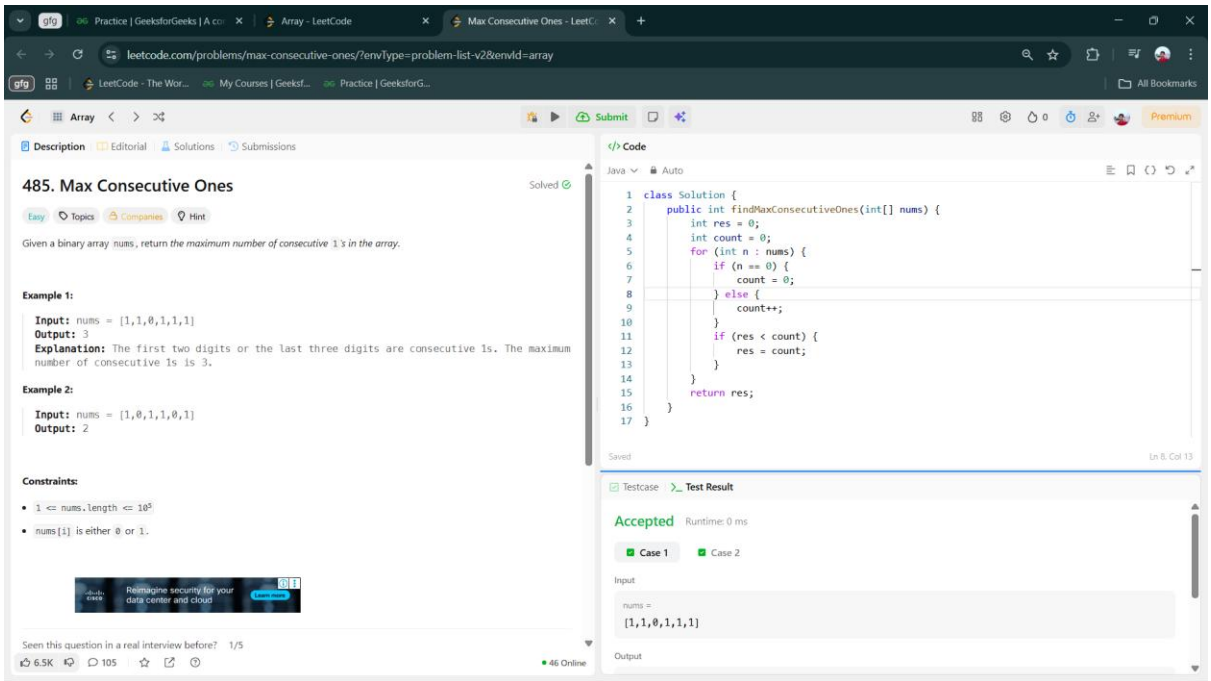


4.9 Max Consecutive ones

Aim : Given a binary array nums, return *the maximum number of consecutive* 1*'s in the array*.

```
class Solution {
    public int findMaxConsecutiveOnes(int[] nums) {
        int res = 0;
        int count = 0;
        for (int n : nums) {
            if (n == 0) {
                count = 0;
            } else {
                count++;
            }
            if (res < count) {
                res = count;
            }
        }
        return res;
    }
}
```

Output :

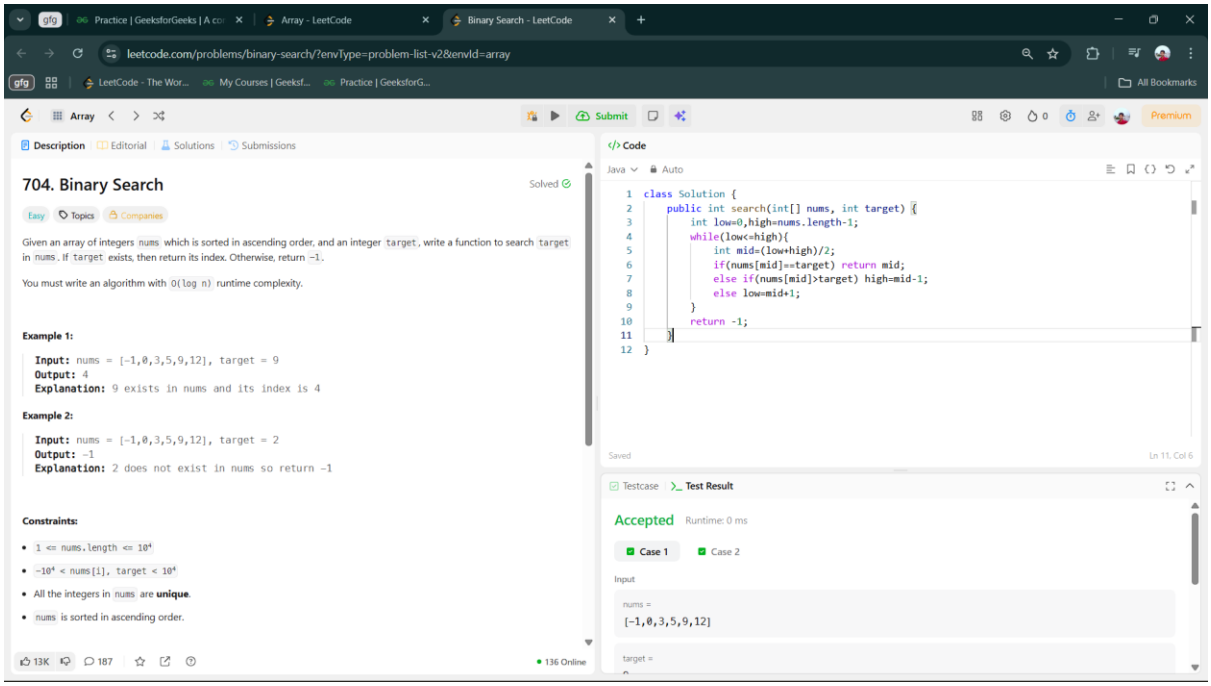


4.10 Binary Search

Aim : Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

```
class Solution {
    public int search(int[] nums, int target) {
        int low=0,high=nums.length-1;
        while(low<=high){
            int mid=(low+high)/2;
            if(nums[mid]==target) return mid;
            else if(nums[mid]>target) high=mid-1;
            else low=mid+1;
        }
        return -1;
    }
}
```

Output :



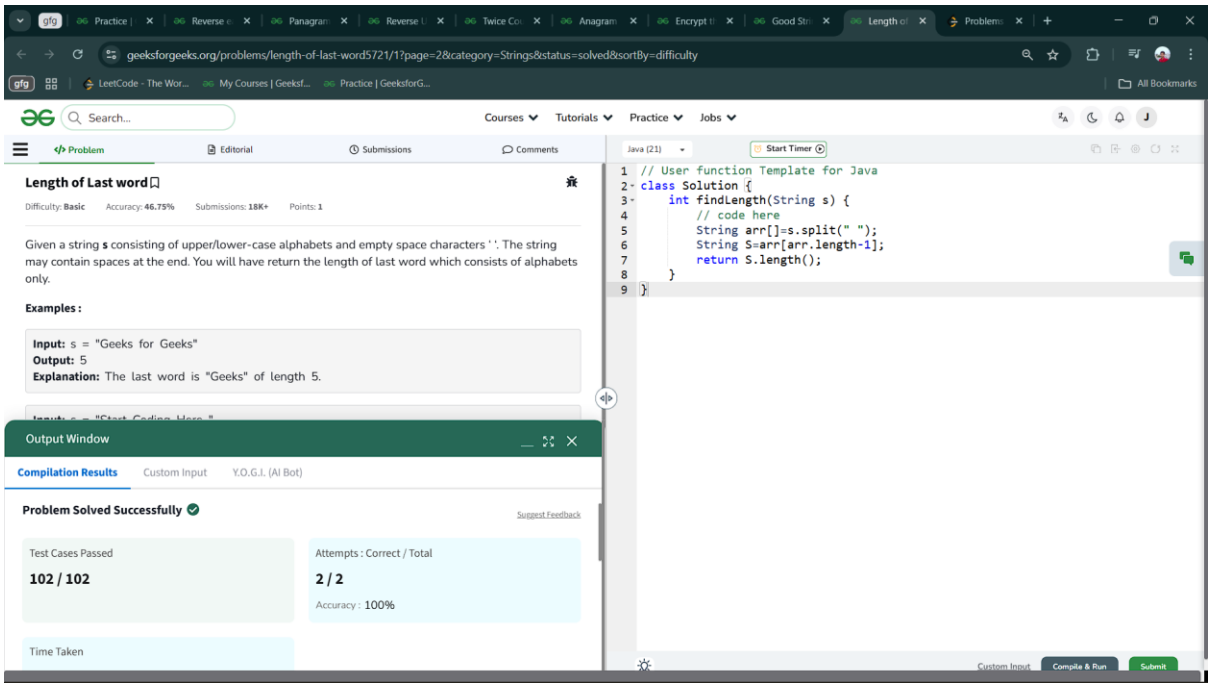
5 . Programs on Strings :

5.1 Length of Last word

Aim : Given a string s consisting of upper/lower-case alphabets and empty space characters ‘ ‘. The string may contain spaces at the end. You will have return the length of last word which consists of alphabets only.

```
class Solution {
    int findLength(String s) {
        // code here
        String arr[]=s.split(" ");
        String S=arr[arr.length-1];
        return S.length();
    }
}
```

Output :



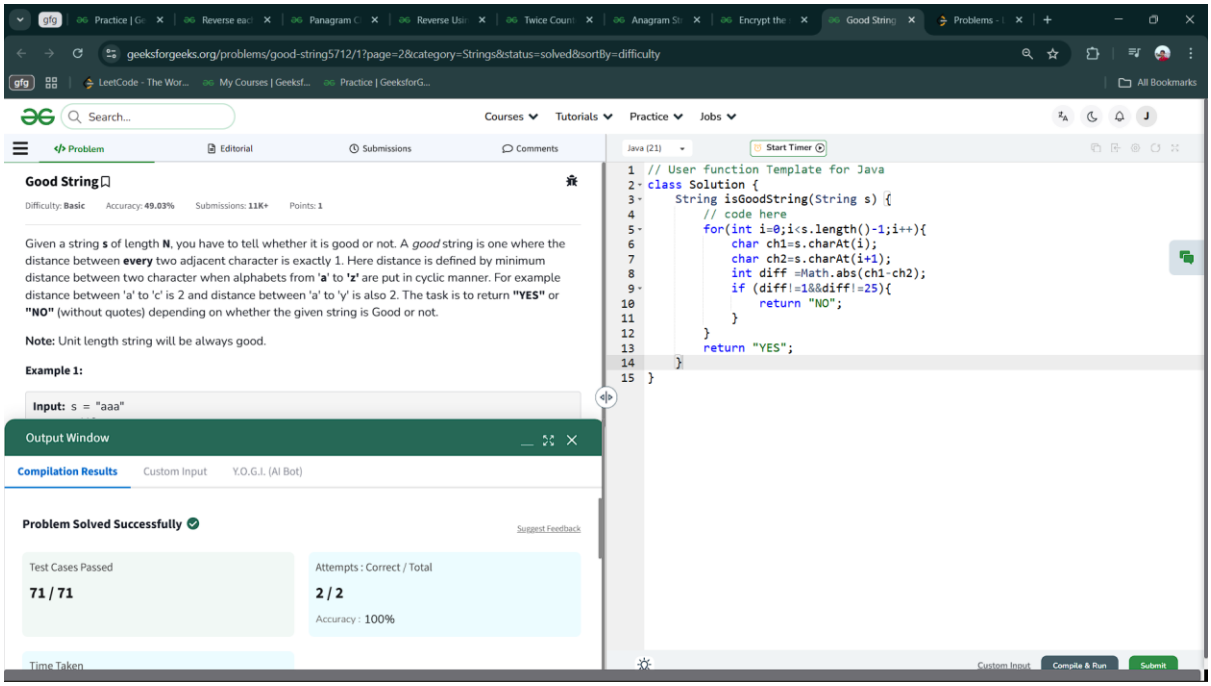
5.2 Good String

Aim : Given a string s of length N, you have to tell whether it is good or not.
A *good* string is one where the distance between every two adjacent character is exactly 1. Here distance is defined by minimum distance between two character when alphabets from 'a' to 'z' are put in cyclic manner. For example distance between 'a' to 'c' is 2 and distance between 'a' to 'y' is also 2. The task is to return "YES" or "NO" (without quotes) depending on whether the given string is Good or not.

Note: Unit length string will be always good.

```
class Solution {
    String isGoodString(String s) {
        // code here
        for(int i=0;i<s.length()-1;i++){
            char ch1=s.charAt(i);
            char ch2=s.charAt(i+1);
            int diff =Math.abs(ch1-ch2);
            if (diff!=1&&diff!=25){
                return "NO";
            }
        }
        return "YES";
    }
}
```

Output :

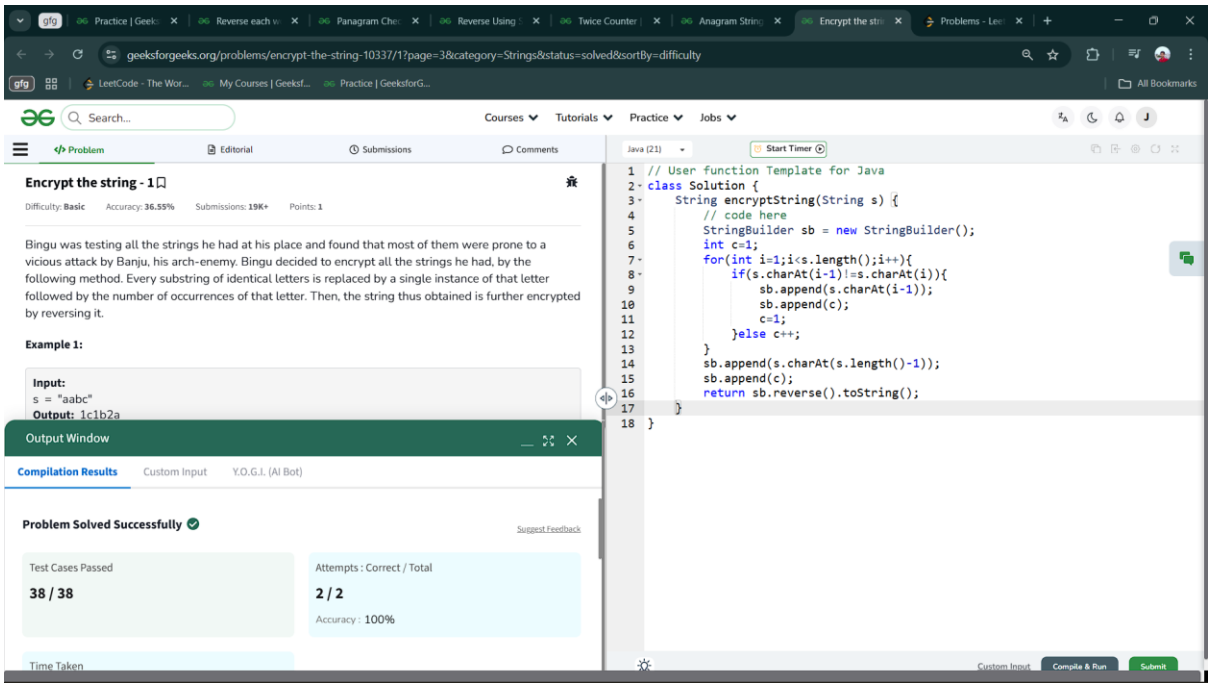


5.3 Encrypt the string - 1

Aim : Bingu was testing all the strings he had at his place and found that most of them were prone to a vicious attack by Banju, his arch-enemy. Bingu decided to encrypt all the strings he had, by the following method. Every substring of identical letters is replaced by a single instance of that letter followed by the number of occurrences of that letter. Then, the string thus obtained is further encrypted by reversing it.

```
class Solution {
    String encryptString(String s) {
        // code here
        StringBuilder sb = new StringBuilder();
        int c=1;
        for(int i=1;i<s.length();i++){
            if(s.charAt(i-1)!=s.charAt(i)){
                sb.append(s.charAt(i-1));
                sb.append(c);
                c=1;
            }else c++;
        }
        sb.append(s.charAt(s.length()-1));
        sb.append(c);
        return sb.reverse().toString();
    }
}
```

Output :



5.4 Anagram Strings

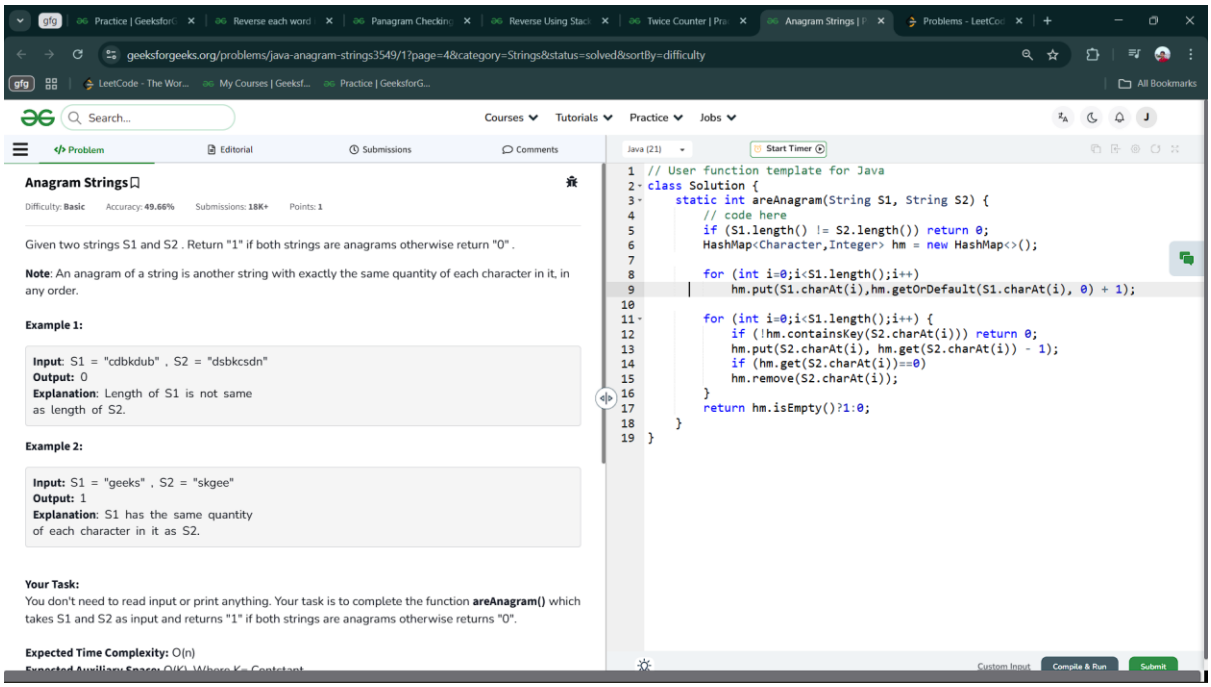
Aim : Given two strings S1 and S2 . Return "1" if both strings are anagrams otherwise return "0" .
Note: An anagram of a string is another string with exactly the same quantity of each character in it, in any order.

```
class Solution {
    static int areAnagram(String S1, String S2) {
        // code here
        if (S1.length() != S2.length()) return 0;
        HashMap<Character,Integer> hm = new HashMap<>();

        for (int i=0;i<S1.length();i++)
            hm.put(S1.charAt(i),hm.getOrDefault(S1.charAt(i), 0) + 1);

        for (int i=0;i<S1.length();i++) {
            if (!hm.containsKey(S2.charAt(i))) return 0;
            hm.put(S2.charAt(i), hm.get(S2.charAt(i)) - 1);
            if (hm.get(S2.charAt(i))==0)
            hm.remove(S2.charAt(i));
        }
        return hm.isEmpty()?1:0;
    }
}
```

Output :

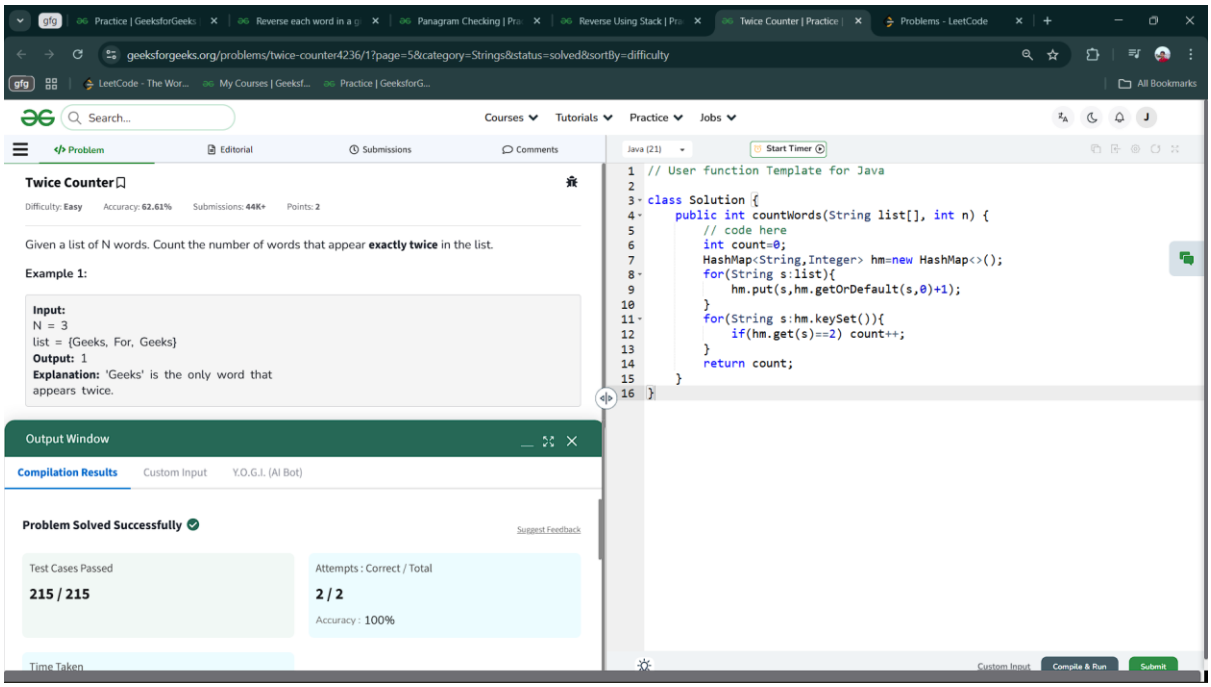


5.5 Twice Counter

Aim : Given a list of N words. Count the number of words that appear exactly twice in the list

```
class Solution {
    public int countWords(String list[], int n) {
        // code here
        int count=0;
        HashMap<String,Integer> hm=new HashMap<>();
        for(String s:list){
            hm.put(s,hm.getOrDefault(s,0)+1);
        }
        for(String s:hm.keySet()){
            if(hm.get(s)==2) count++;
        }
        return count;
    }
}
```

Output :

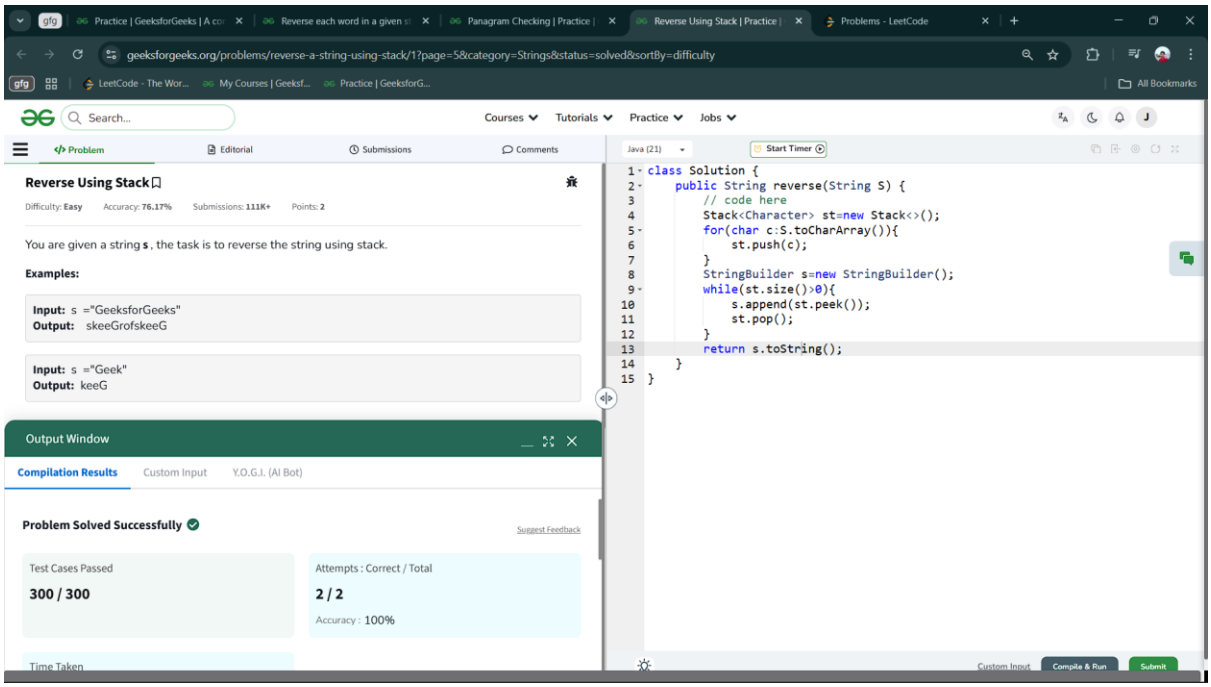


5.6 Reverse Using Stack

Aim : You are given a string s , the task is to reverse the string using stack.

```
class Solution {
    public String reverse(String S) {
        // code here
        Stack<Character> st=new Stack<>();
        for(char c:S.toCharArray()){
            st.push(c);
        }
        StringBuilder s=new StringBuilder();
        while(st.size()>0){
            s.append(st.peek());
            st.pop();
        }
        return s.toString();
    }
}
```

Output :

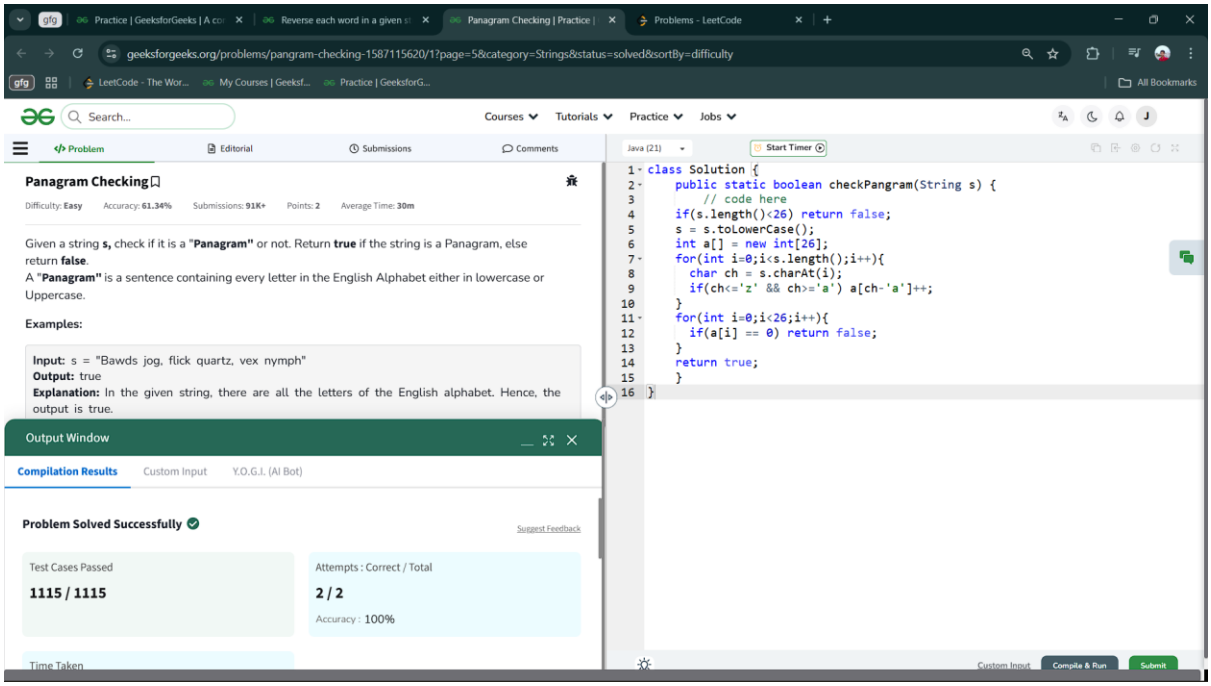


5.7 Panagram Checking

Aim : Given a string s, check if it is a "Panagram" or not. Return true if the string is a Panagram, else return false.
A "Panagram" is a sentence containing every letter in the English Alphabet either in lowercase or Uppercase.

```
class Solution {
    public static boolean checkPangram(String s) {
        // code here
    if(s.length()<26) return false;
    s = s.toLowerCase();
    int a[] = new int[26];
    for(int i=0;i<s.length();i++){
      char ch = s.charAt(i);
      if(ch<='z' && ch>='a') a[ch-'a']++;
    }
    for(int i=0;i<26;i++){
      if(a[i] == 0) return false;
    }
    return true;
    }
}
```

Output :

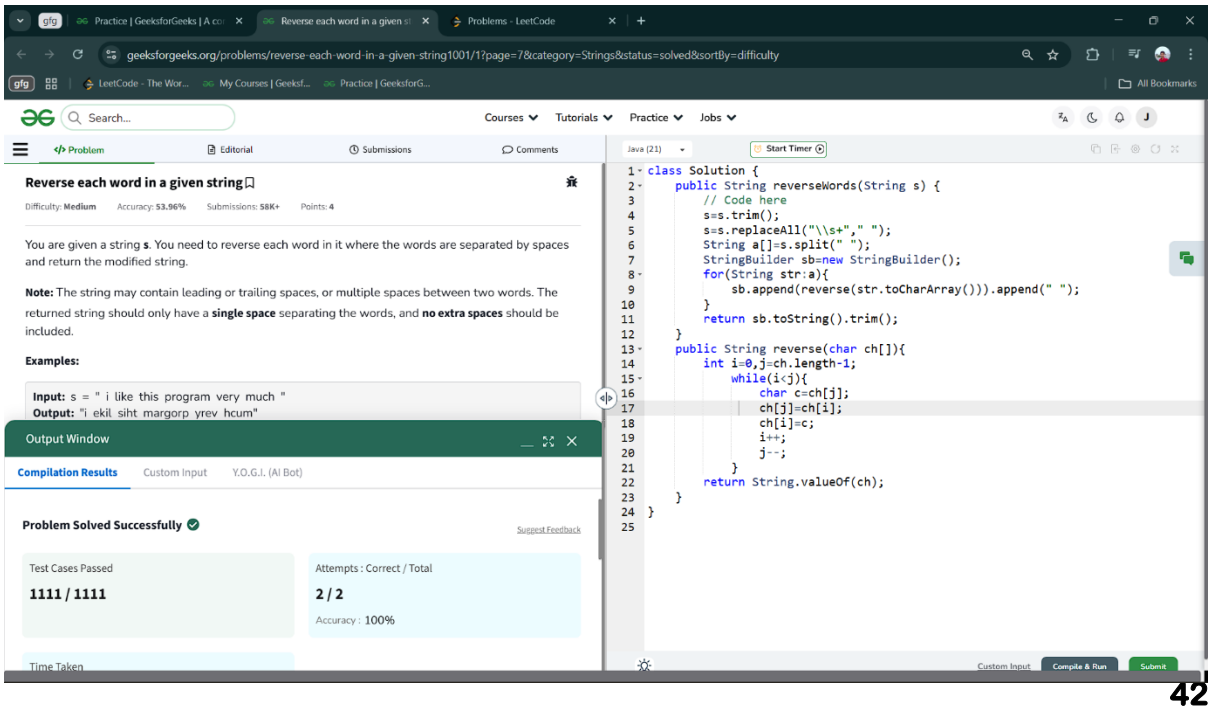


5.8 Reverse each word in a given string

Aim : You are given a string s. You need to reverse each word in it where the words are separated by spaces and return the modified string.
Note: The string may contain leading or trailing spaces, or multiple spaces between two words. The returned string should only have a single space separating the words, and no extra spaces should be included.

```
class Solution {
    public String reverseWords(String s) {
        // Code here
        s=s.trim();
        s=s.replaceAll("\\s+"," ");
        String a[]=s.split(" ");
        StringBuilder sb=new StringBuilder();
        for(String str:a){
            sb.append(reverse(str.toCharArray())).append(" ");
        }
        return sb.toString().trim();
    }
    public String reverse(char ch[]){
        int i=0,j=ch.length-1;
            while(i<j){
                char c=ch[j];
                ch[j]=ch[i];
                ch[i]=c;
                i++;
                j--;
            }
        return String.valueOf(ch);
    }
}
```

Output ;



5.9 Valid Parentheses

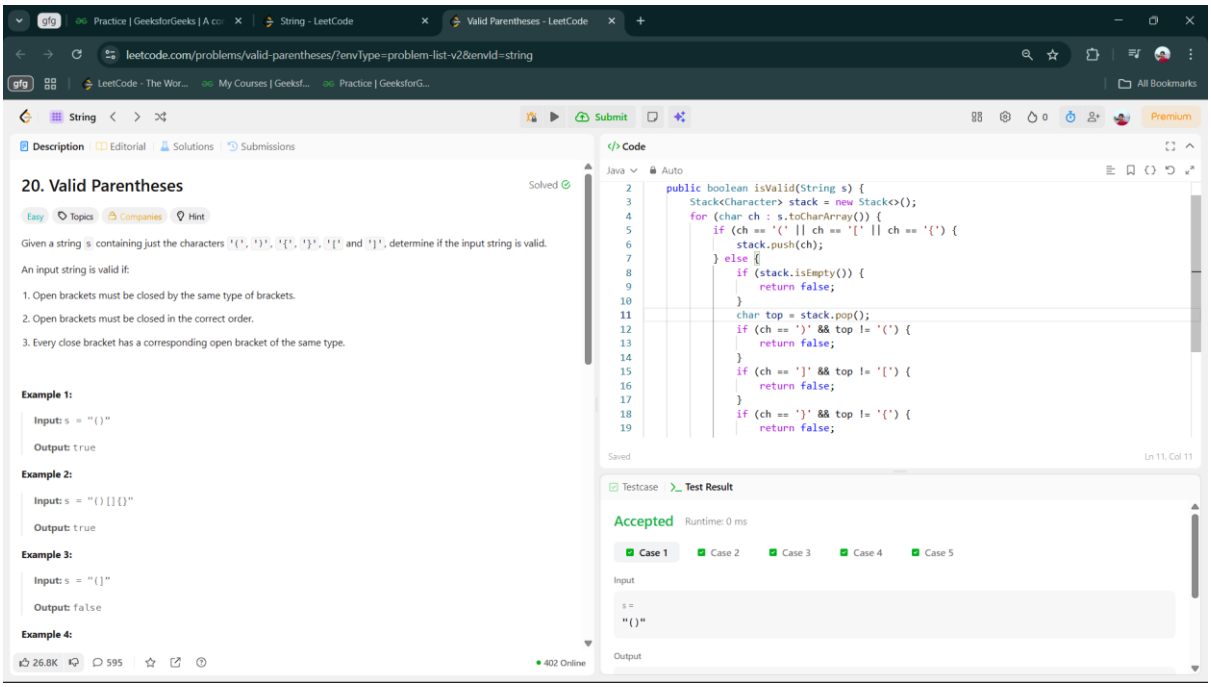
Aim : Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. **Open brackets must be closed by the same type of brackets.**
2. **Open brackets must be closed in the correct order.**
3. **Every close bracket has a corresponding open bracket of the same type**

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
        for (char ch : s.toCharArray()) {
            if (ch == '(' || ch == '[' || ch == '{') {
                stack.push(ch);
            } else {
                if (stack.isEmpty()) {
                    return false;
                }
                char top = stack.pop();
                if (ch == ')' && top != '(') {
                    return false;
                }
                if (ch == ']' && top != '[') {
                    return false;
                }
                if (ch == '}' && top != '{') {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }
}
```

Output :

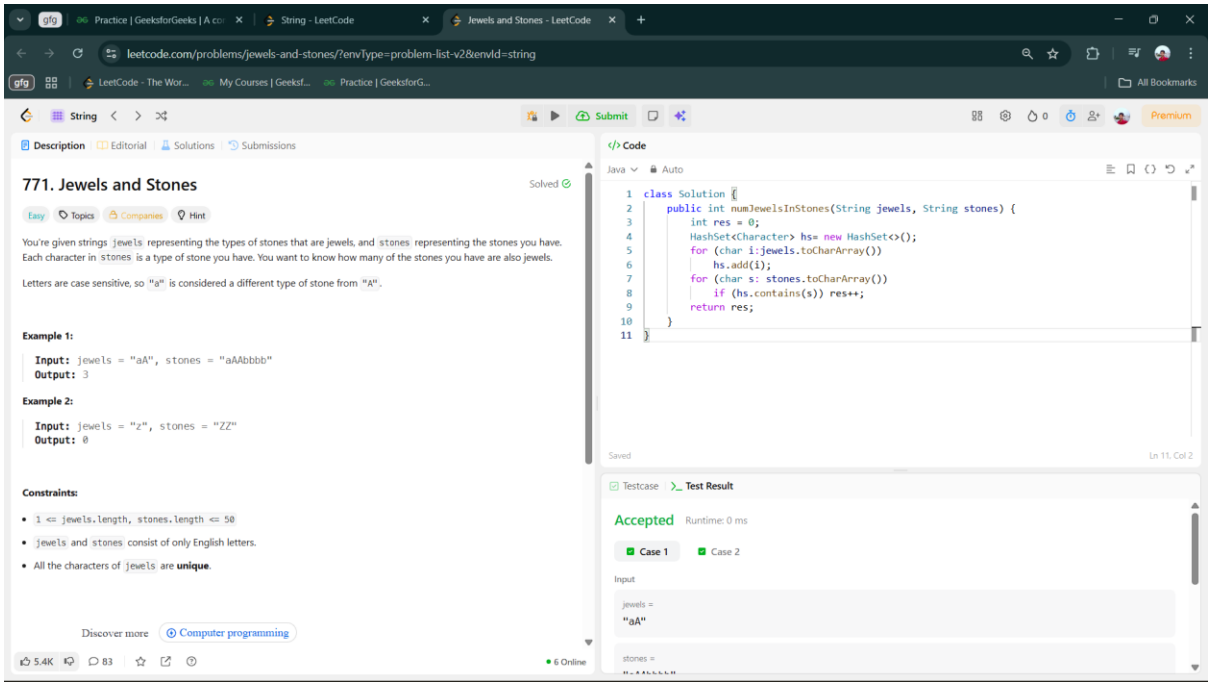


5.10 Jewels and stones

Aim : You're given strings jewels representing the types of stones that are jewels, and stones representing the stones you have. Each character in stones is a type of stone you have. You want to know how many of the stones you have are also jewels. Letters are case sensitive, so "a" is considered a different type of stone from "A".

```
class Solution {
    public int numJewelsInStones(String jewels, String stones) {
        int res = 0;
        HashSet<Character> hs= new HashSet<>();
        for (char i:jewels.toCharArray())
            hs.add(i);
        for (char s: stones.toCharArray())
            if (hs.contains(s)) res++;
        return res;
    }
}
```

Output :



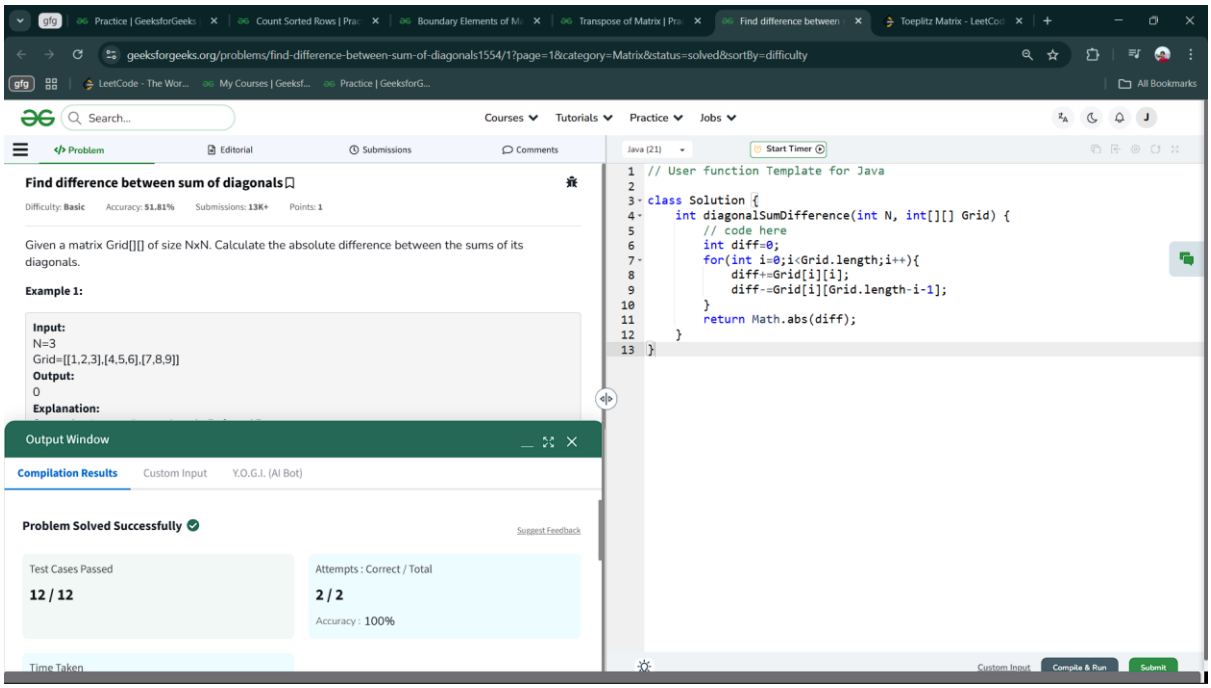
6.Programs on Matrices :

6.1 Find difference between sum of diagonals

Aim : Given a matrix Grid[][][] of size NxN. Calculate the absolute difference between the sums of its diagonals.

```
class Solution {
    int diagonalSumDifference(int N, int[][] Grid) {
        // code here
        int diff=0;
        for(int i=0;i<Grid.length;i++){
            diff+=Grid[i][i];
            diff-=Grid[i][Grid.length-i-1];
        }
        return Math.abs(diff);
    }
}
```

Output :

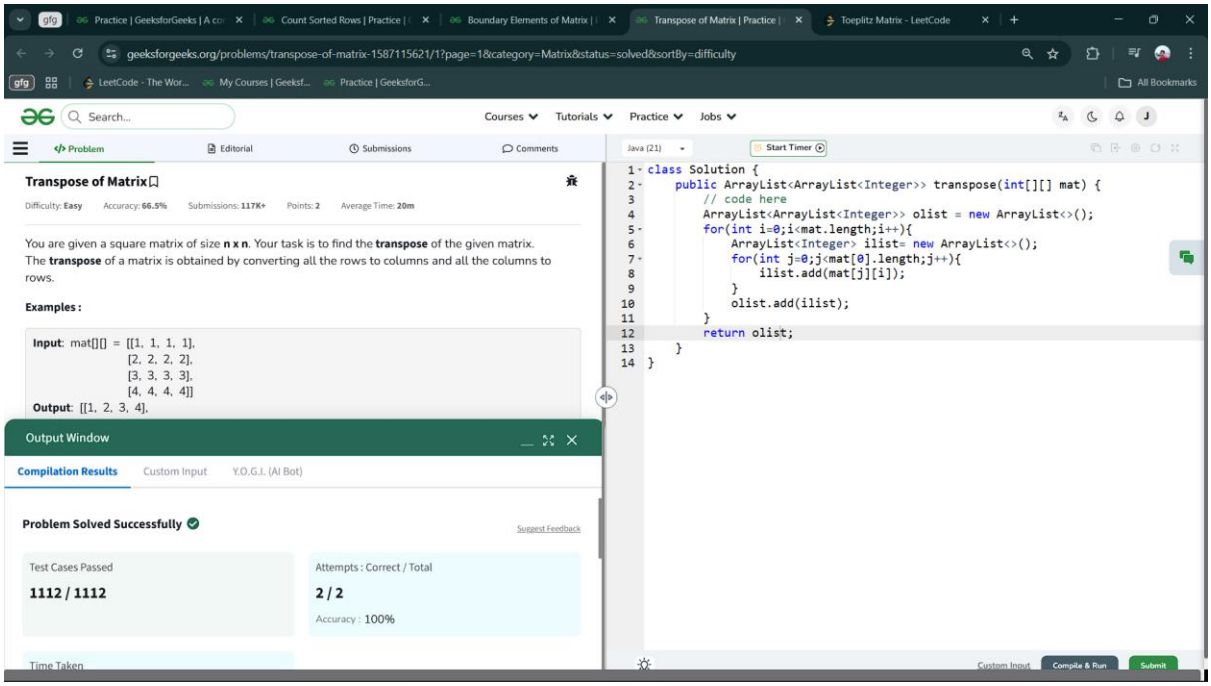


6.2 Transpose of Matrix

Aim : 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) {
        ArrayList<ArrayList<Integer>> olist = new ArrayList<>();
        for(int i=0;i<mat.length;i++){
            ArrayList<Integer> ilist= new ArrayList<>();
            for(int j=0;j<mat[0].length;j++){
                ilist.add(mat[j][i]);
            }
            olist.add(ilist);
        }
        return olist;
    }
}
```

Output :

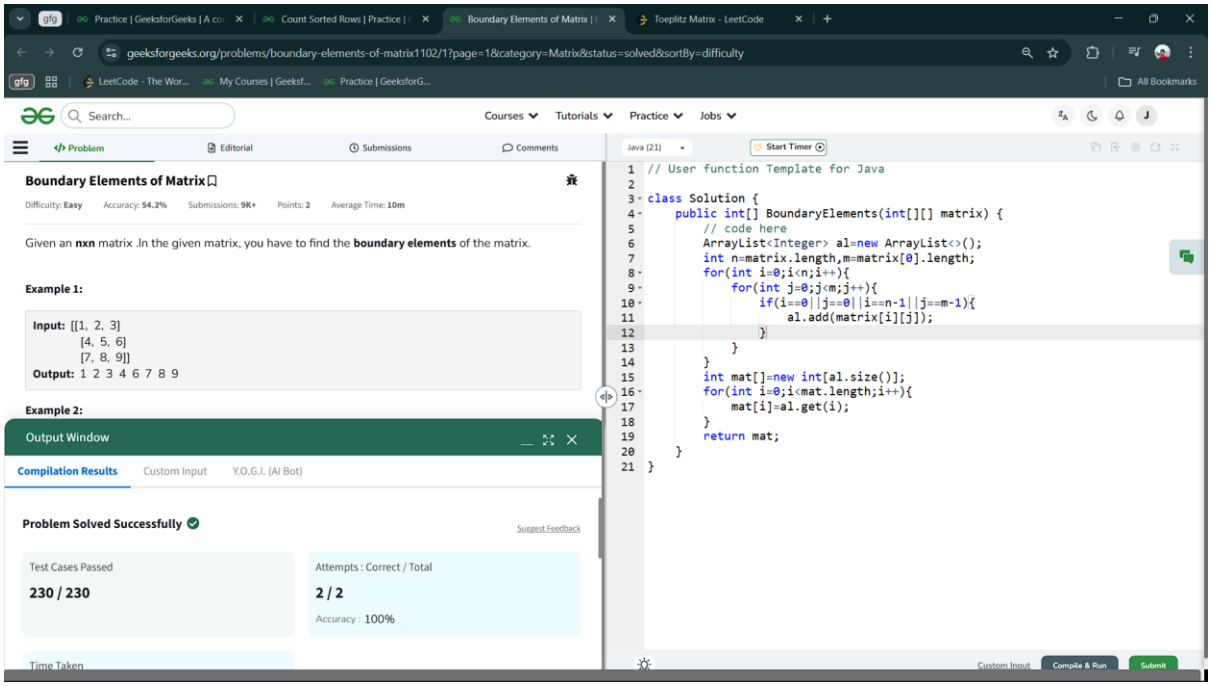


6.3 Boundary Elements of Matrix

Aim : Given an nxn matrix .In the given matrix, you have to find the boundary elements of the matrix.

```
class Solution {
    public int[] BoundaryElements(int[][] matrix) {
        ArrayList<Integer> al=new ArrayList<>();
        int n=matrix.length,m=matrix[0].length;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(i==0||j==0||i==n-1||j==m-1){
                    al.add(matrix[i][j]);
                }
            }
        }
        int mat[]=new int[al.size()];
        for(int i=0;i<mat.length;i++){
            mat[i]=al.get(i);
        }
        return mat;
    }
}
```

Output :

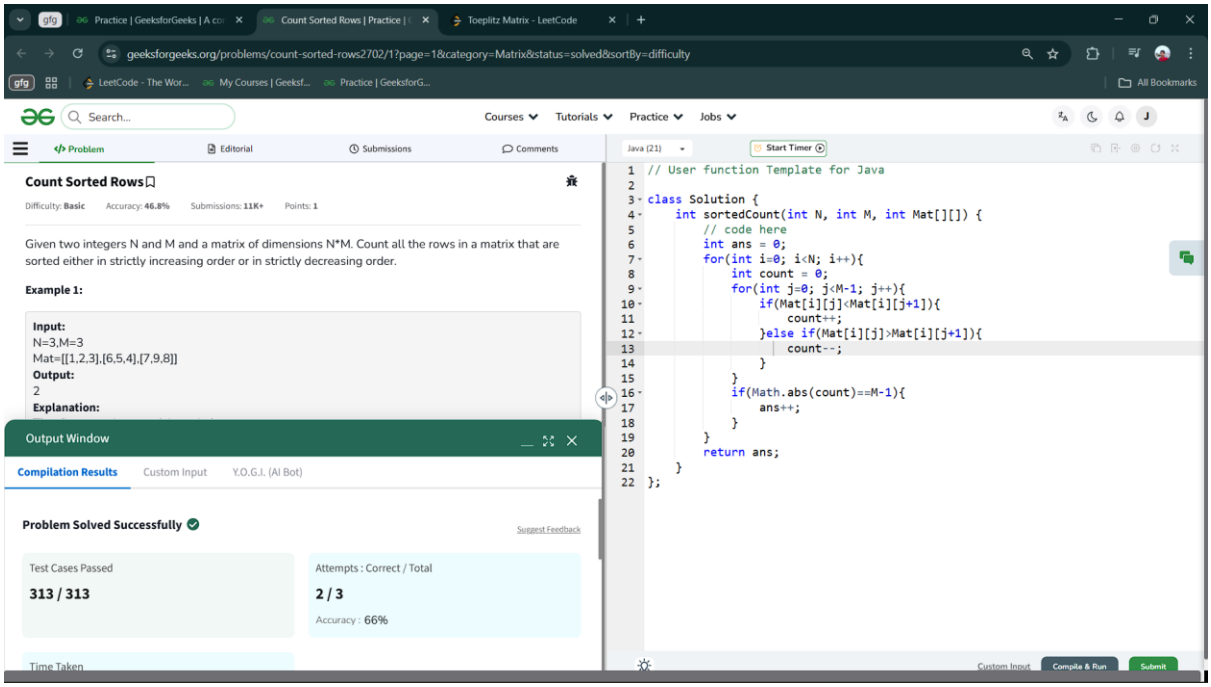


6.4 Count Sorted Rows

Aim : Given two integers N and M and a matrix of dimensions N*M. Count all the rows in a matrix that are sorted either in strictly increasing order or in strictly decreasing order.

```
class Solution {
    int sortedCount(int N, int M, int Mat[][]) {
        // code here
        int ans = 0;
        for(int i=0; i<N; i++){
            int count = 0;
            for(int j=0; j<M-1; j++){
                if(Mat[i][j]<Mat[i][j+1]){
                    count++;
                }else if(Mat[i][j]>Mat[i][j+1]){
                    count--;
                }
            }
            if(Math.abs(count)==M-1){
                ans++;
            }
        }
        return ans;
    }
};
```

Output :



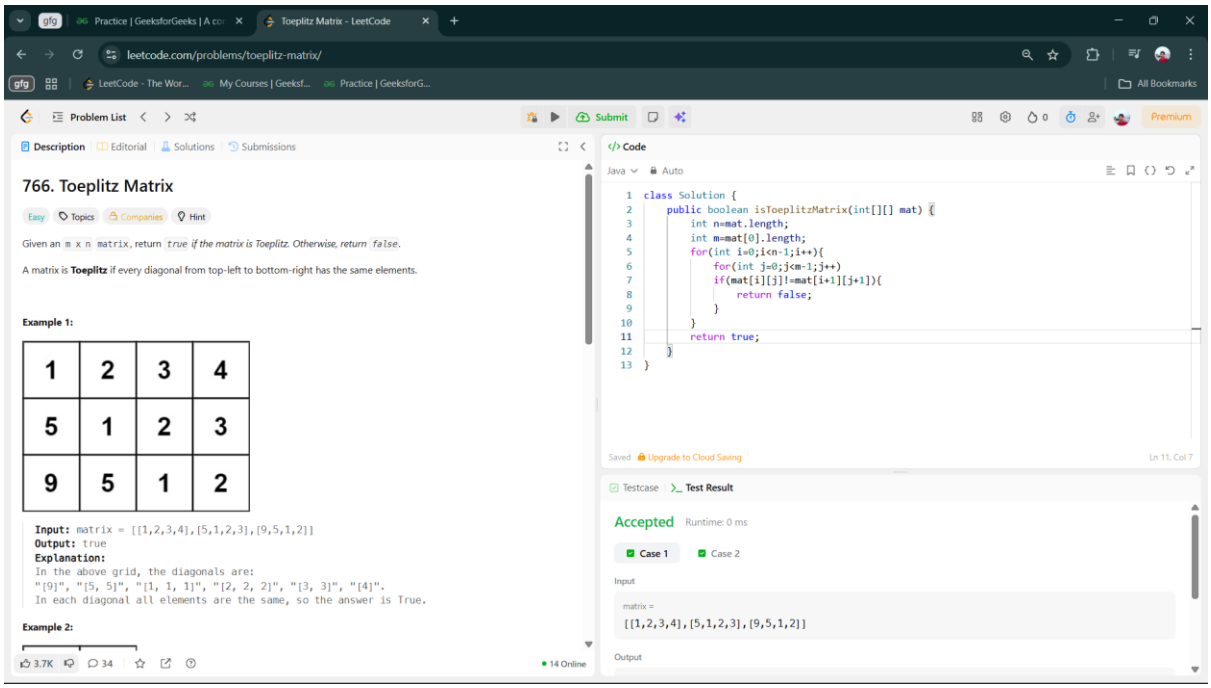
6.5 Toeplitz Matrix

Aim : Given an m x n matrix, return *true if the matrix is Toeplitz. Otherwise, return false.*

A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.

```
class Solution {
    public boolean isToeplitzMatrix(int[][] mat) {
        int n=mat.length;
        int m=mat[0].length;
        for(int i=0;i<n-1;i++){
            for(int j=0;j<m-1;j++)
            if(mat[i][j]!=mat[i+1][j+1]){
                return false;
            }
        }
        return true;
    }
}
```

Output :



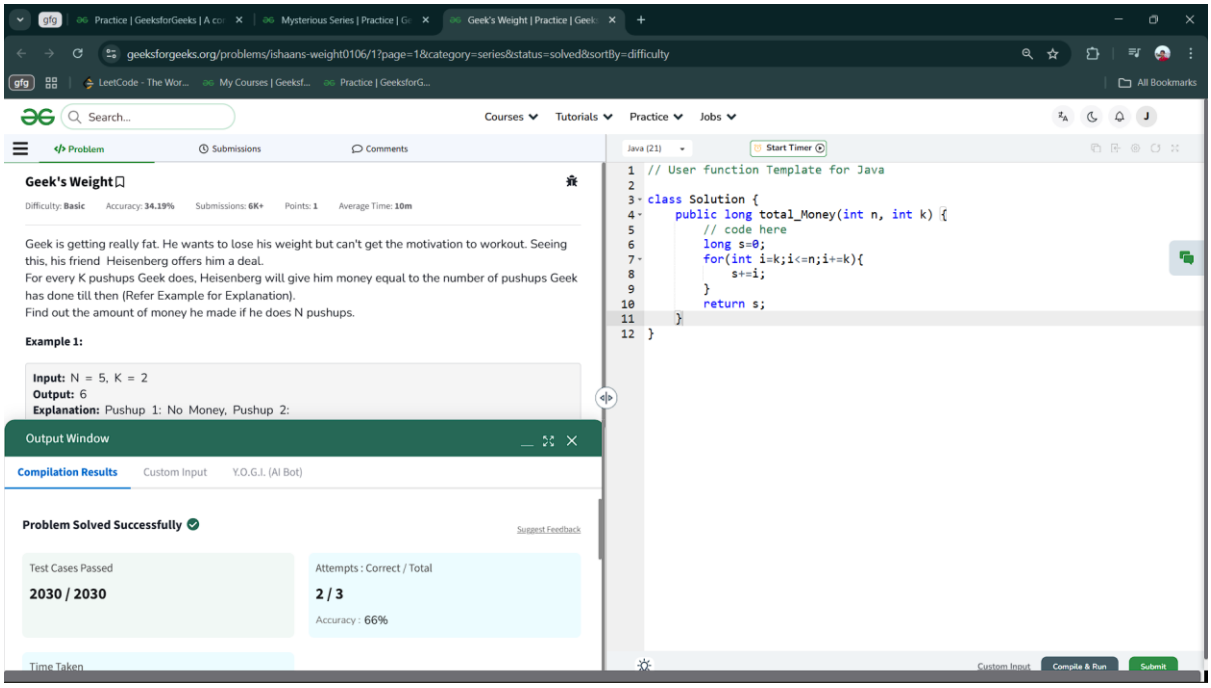
7.Programs on series and patterns :

7.1 Geek's Weight

Aim : Geek is getting really fat. He wants to lose his weight but can't get the motivation to workout. Seeing this, his friend Heisenberg offers him a deal.
For every K pushups Geek does, Heisenberg will give him money equal to the number of pushups Geek has done till then (Refer Example for Explanation).
Find out the amount of money he made if he does N pushups.

```
class Solution {
    public long total_Money(int n, int k) {
        // code here
        long s=0;
        for(int i=k;i<=n;i+=k){
            s+=i;
        }
        return s;
    }
}
```

Output :

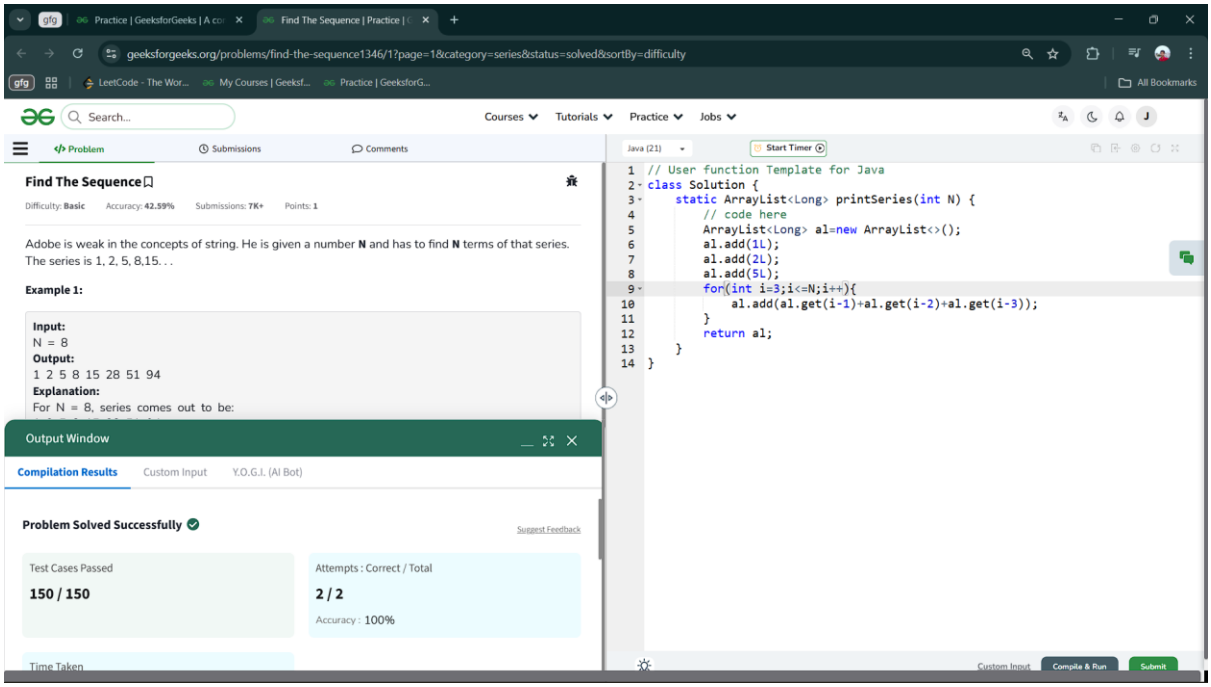


7.2 Find The Sequence

Aim : Adobe is weak in the concepts of string. He is given a number N and has to find N terms of that series. The series is 1, 2, 5, 8,15. .

```
class Solution {
    static ArrayList<Long> printSeries(int N) {
        ArrayList<Long> al=new ArrayList<>();
        al.add(1L);
        al.add(2L);
        al.add(5L);
        for(int i=3;i<=N;i++){
            al.add(al.get(i-1)+al.get(i-2)+al.get(i-3));
        }
        return al;
    }
}
```

Output ;

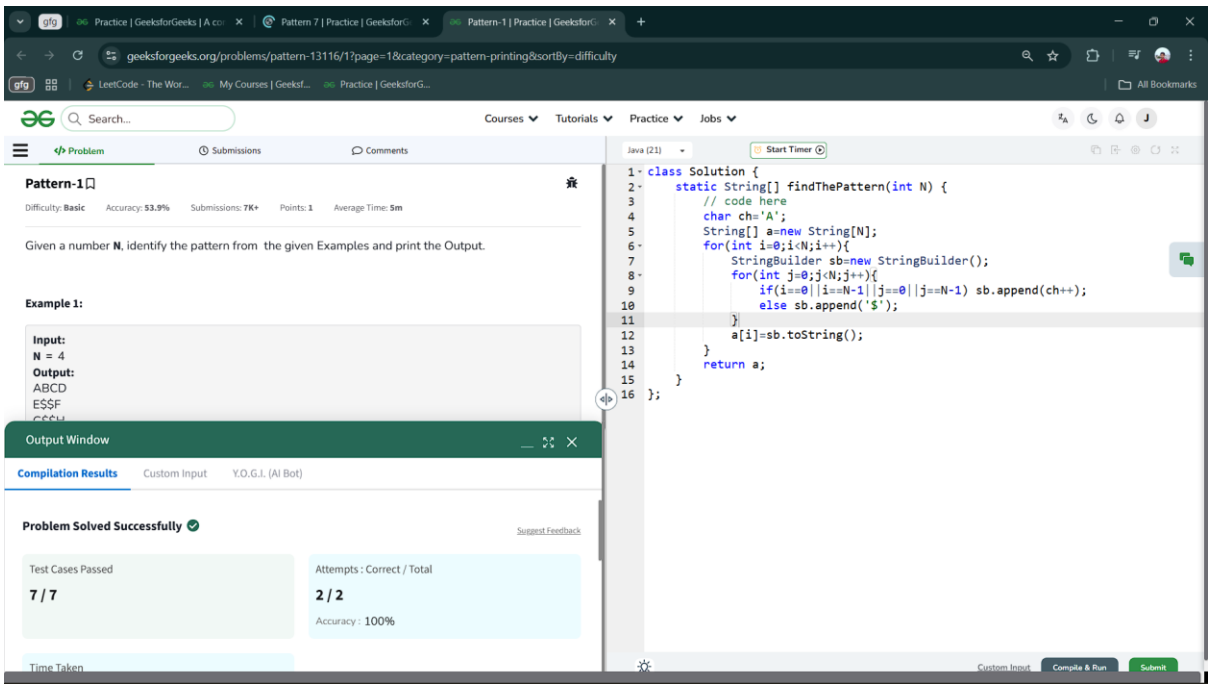


7.3 Pattern-1

Aim : Given a number N, identify the pattern from the given Examples and print the Output.

```
class Solution {
    static String[] findThePattern(int N) {
        char ch='A';
        String[] a=new String[N];
        for(int i=0;i<N;i++){
            StringBuilder sb=new StringBuilder();
            for(int j=0;j<N;j++){
                if(i==0||i==N-1||j==0||j==N-1) sb.append(ch++);
                else sb.append('$');
            }
            a[i]=sb.toString();
        }
        return a;
    }
};
```

Output :

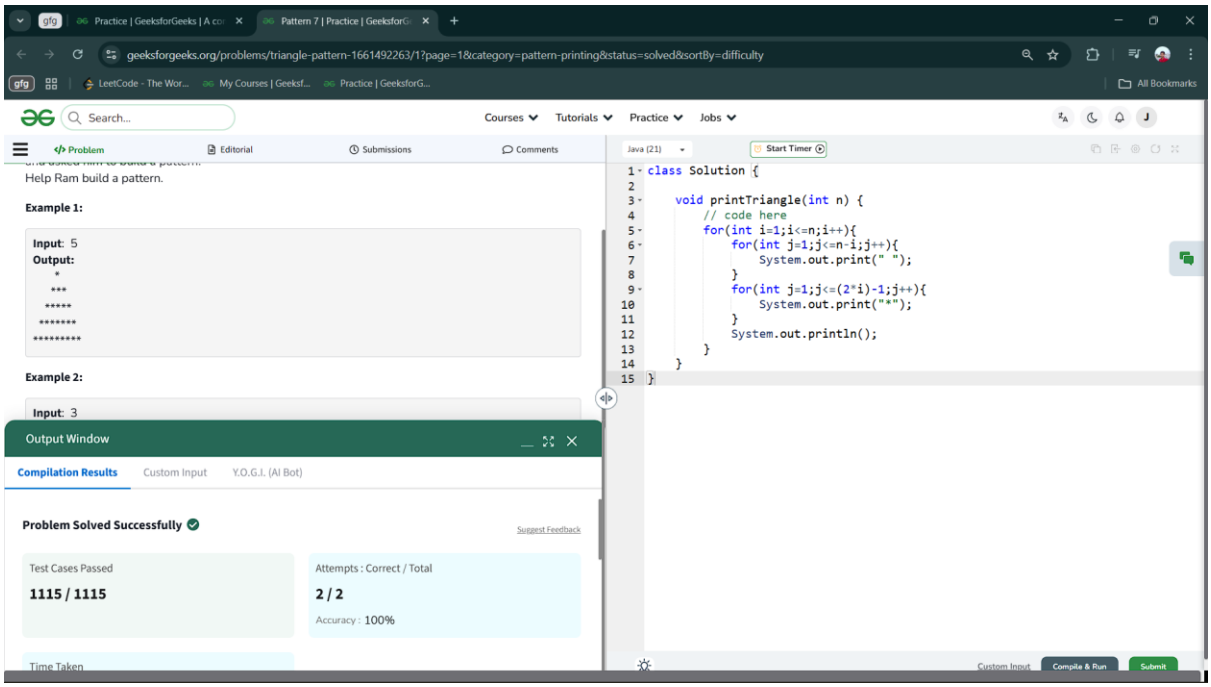


7.4 Pattern 2

Aim : Geek is very fond of patterns. Once, his teacher gave him a pattern to solve. He gave Ram an integer n and asked him to build a pattern.
Help Ram build a pattern.

```
class Solution {
    void printTriangle(int n) {
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n-i;j++){
                System.out.print(" ");
            }
            for(int j=1;j<=(2*i)-1;j++){
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

Output :



7.5 Triangle growing downwards

Aim : Given string str of a constant length, print a triangle out of it. The triangle should start with the first character of the string and keep growing downwards by adding one character from the string.
The spaces on the left side of the triangle should be replaced with a dot character.

```
class Solution {
    static void printTriangleDownwards(String str) {
        int n=str.length();
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n-i;j++){
                System.out.print(".");
            }
            System.out.println(str.substring(0,i));
        }
    }
}
```

Output :