Experiment 4

Student Name: Payal Singh **UID:22BET10347**
Branch: BE-IT **Section/Group: IOT-702(A)**
Semester: 6 **Date of Performance:20/02/25**
Subject Name: AP LAB-II **Subject Code: 22ITP-351**

PROBLEM 1:

Aim:

A string s is nice if, for every letter of the alphabet that s contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

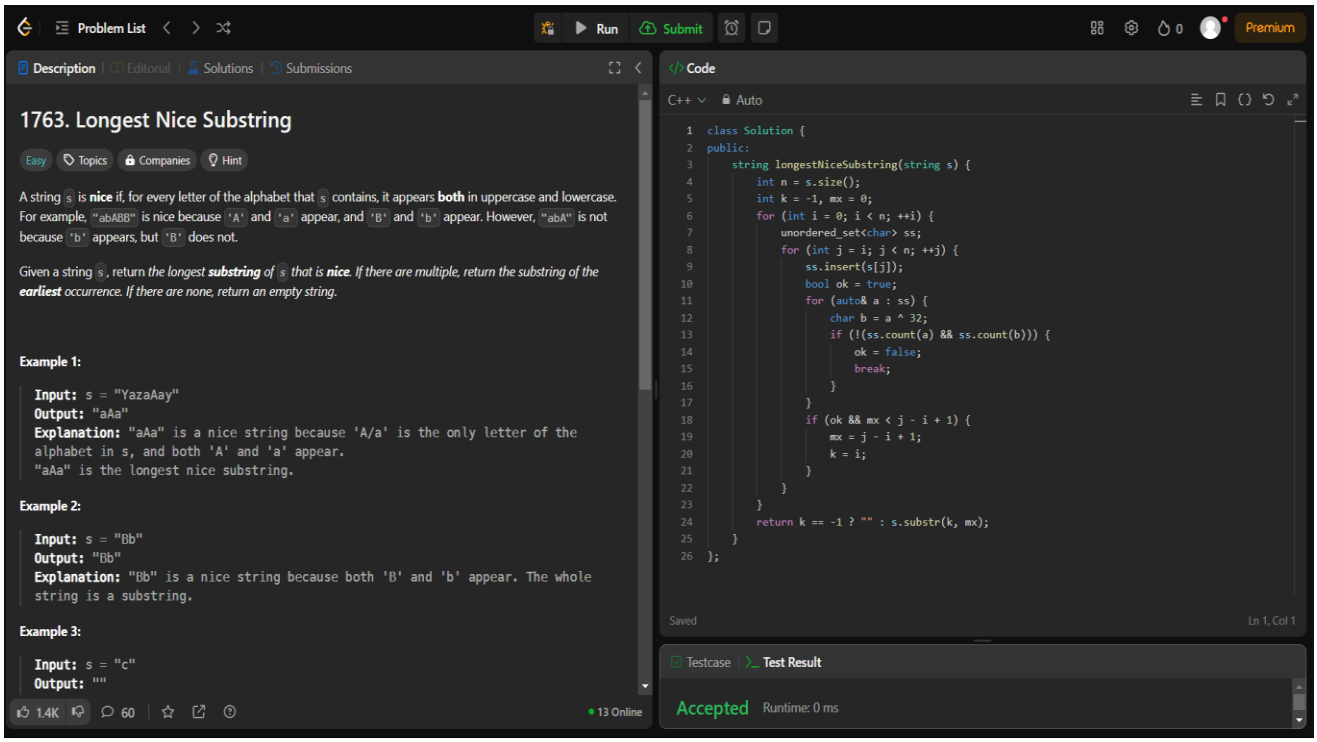
Given a string s, return the longest substring of s that is nice. If there are multiple, return the substring of the earliest occurrence. If there are none, return an empty string.

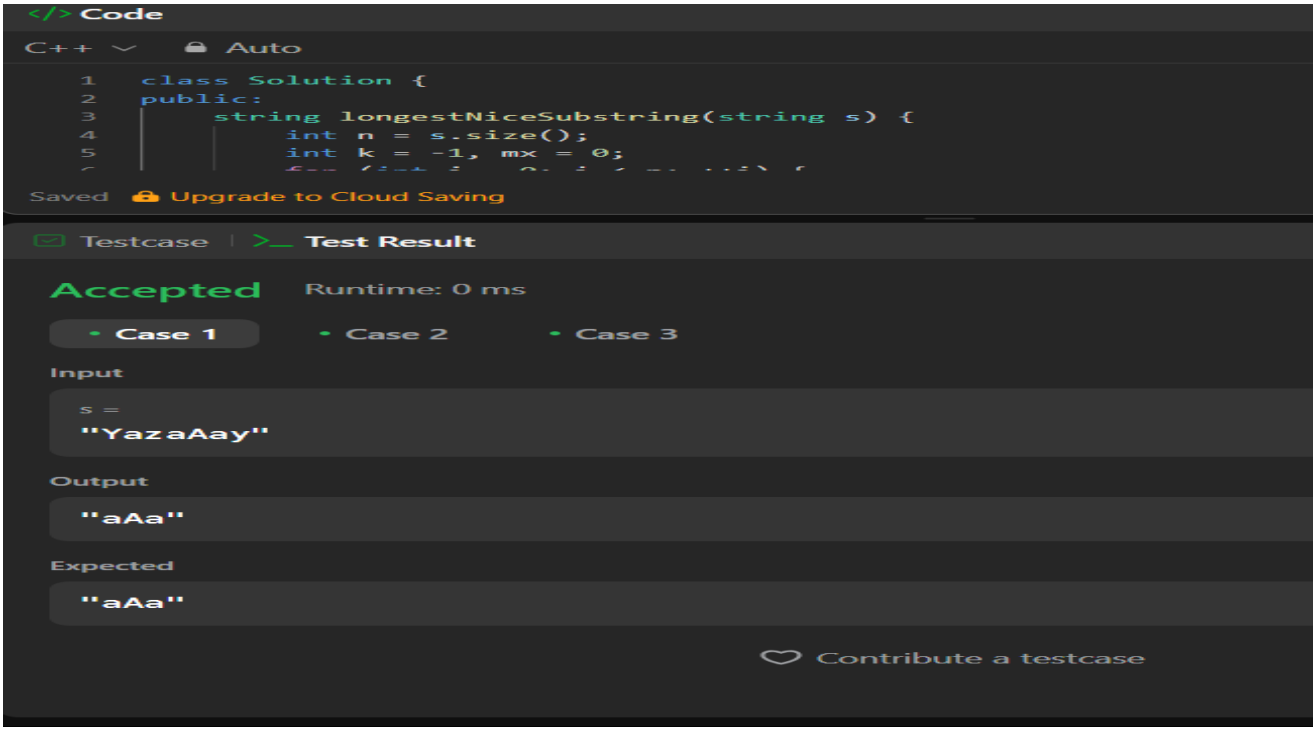
Code:

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        int n = s.size();
        int k = -1, mx = 0;
        for (int i = 0; i < n; ++i) {
            unordered_set<char> ss;
            for (int j = i; j < n; ++j) {
                ss.insert(s[j]);
                bool ok = true;
                for (auto& a : ss) {
                    char b = a ^ 32;
                    if (!(ss.count(a) && ss.count(b))) {
                        ok = false;
                        break;
                    }
                }
                if (ok && mx < j - i + 1) {
                    mx = j - i + 1;
                    k = i;
                }
            }
```

```
        }
        return k == -1 ? "" : s.substr(k, mx);
    }
};
```

Output:





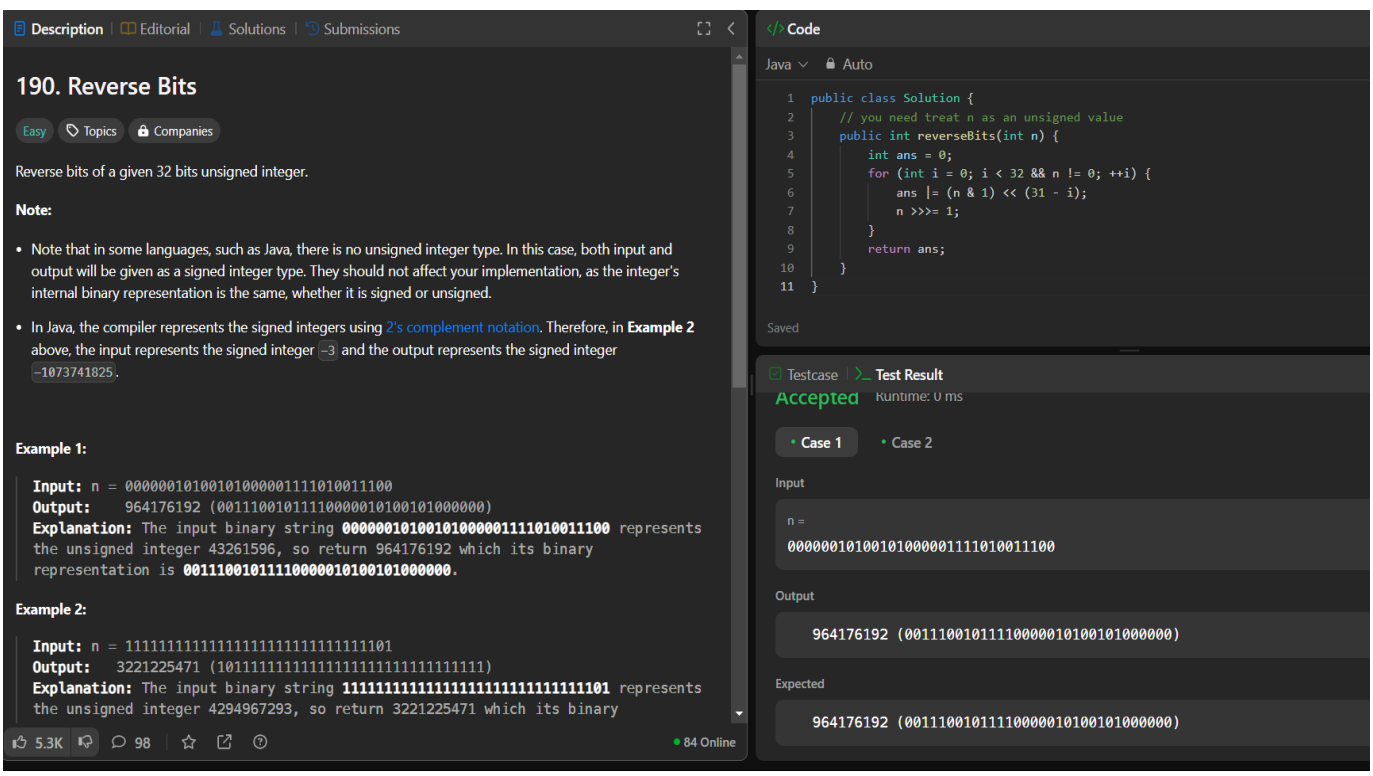
PROBLEM 2:

Aim: Reverse bits of a given 32 bits unsigned integer.

Code:

```
public class Solution {
  public int reverseBits(int n) {
      int ans = 0;
      for (int i = 0; i < 32 && n != 0; ++i) {
          ans |= (n & 1) << (31 - i);
          n >>>= 1;
      }
      return ans;
  }
```

Output:



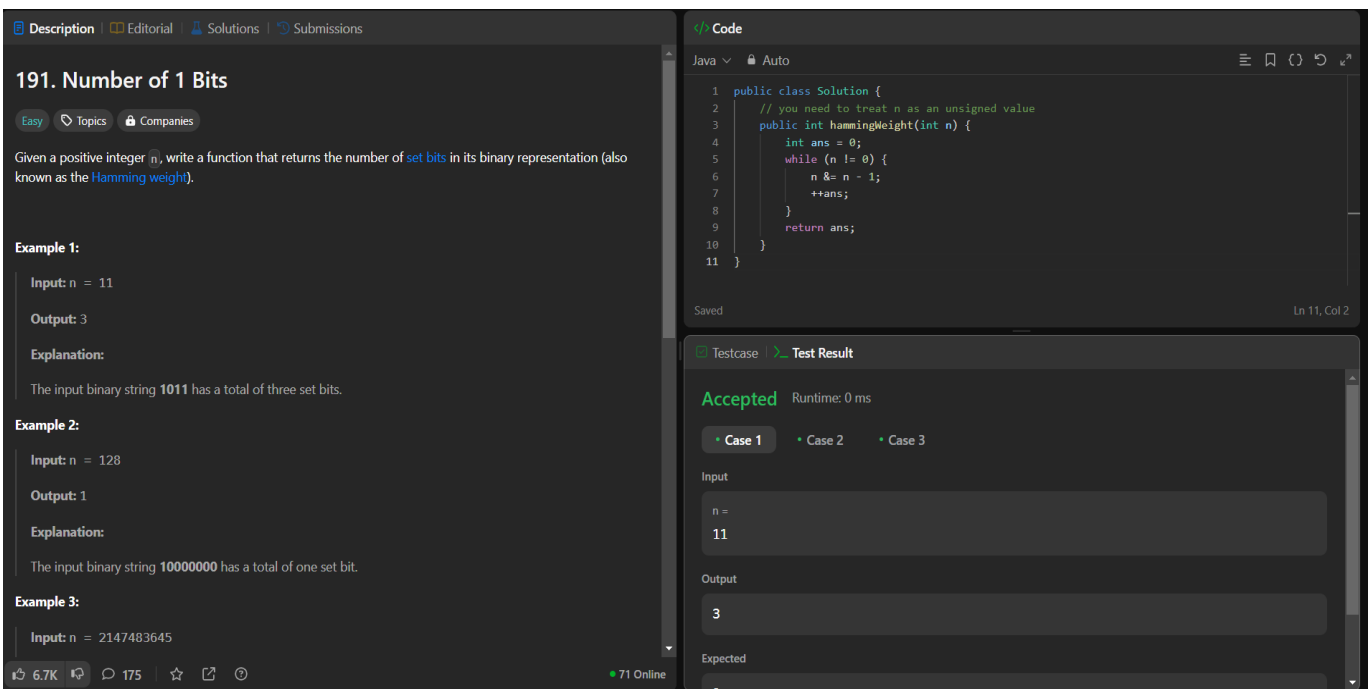
PROBLEM 3:

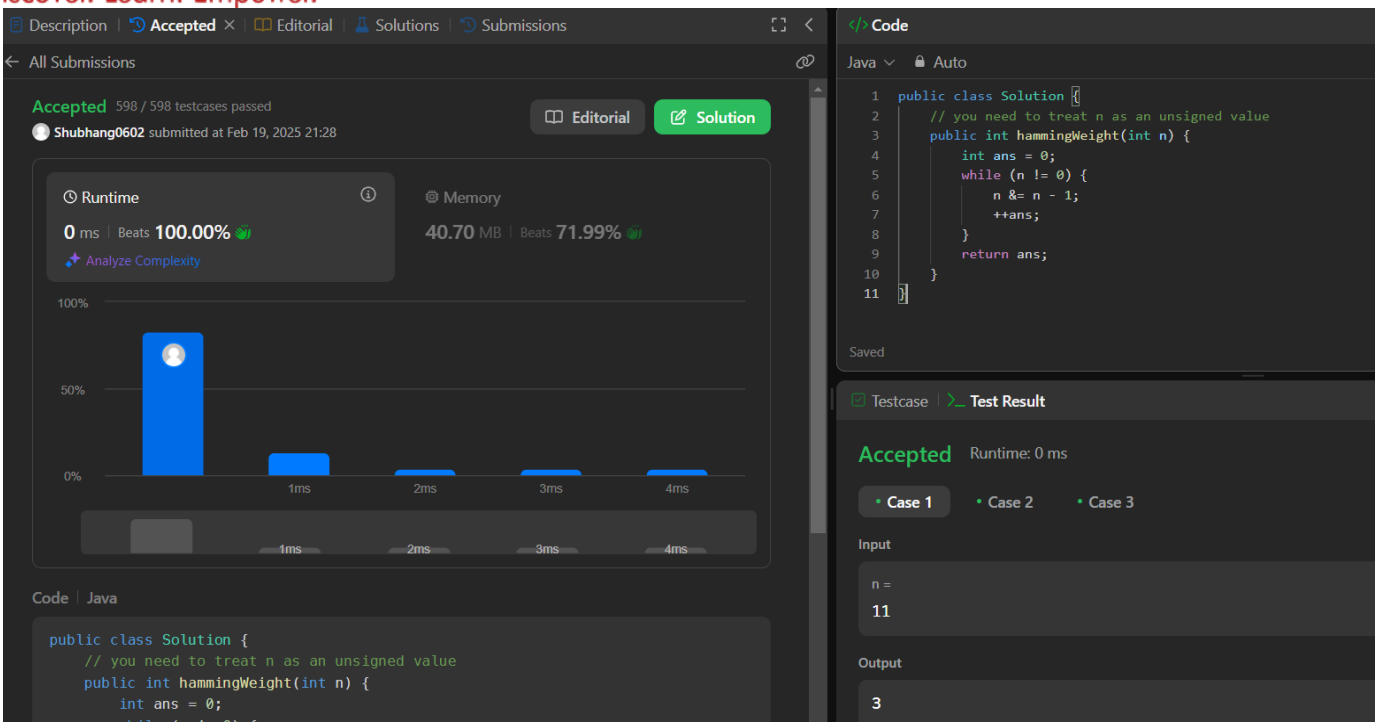
Aim: Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the [Hamming weight]).

Code:

```
public class Solution {
   public int hammingWeight(int n) {
      int ans = 0;
      while (n != 0) {
         n &= n - 1;
         ++ans;
      }
      return ans;
   }
}
```

Output:





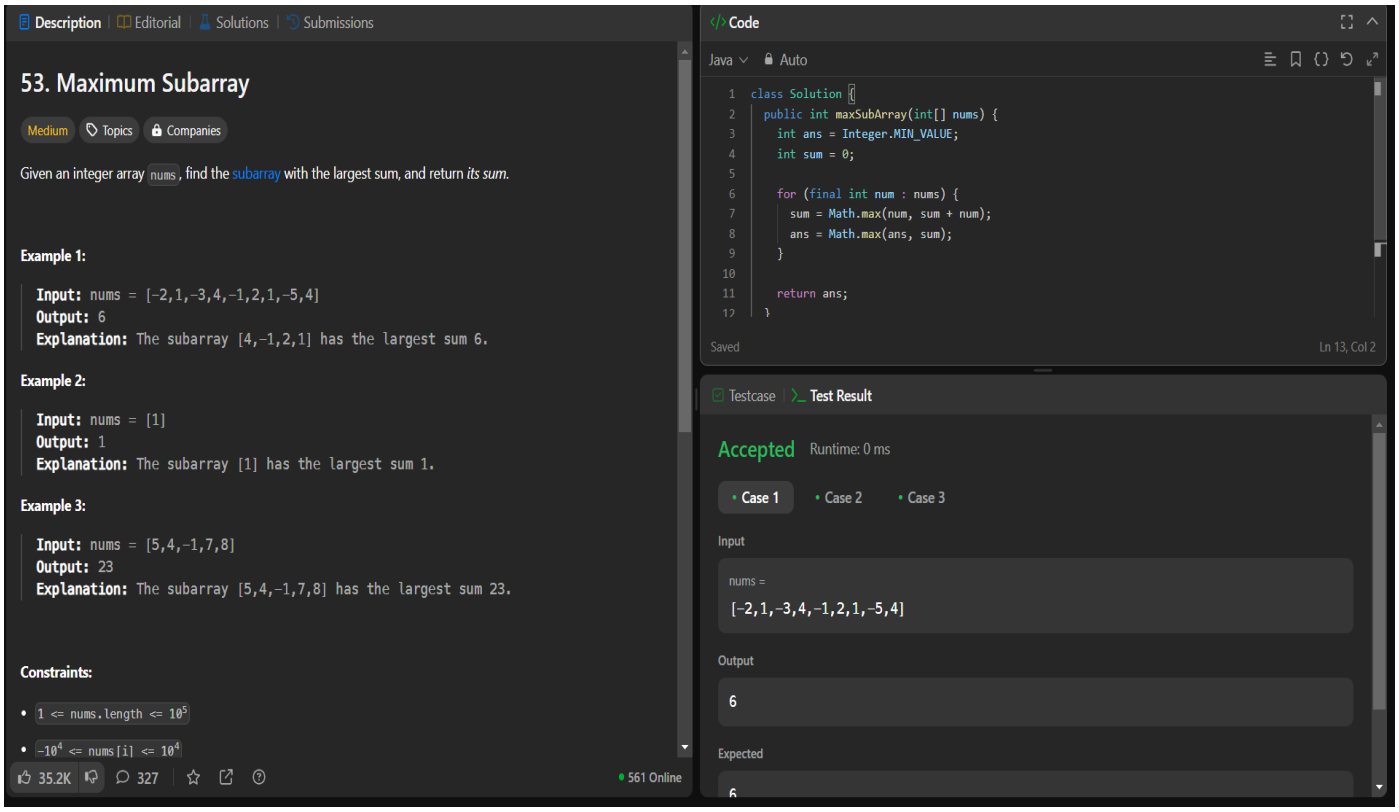
PROBLEM 4:

Aim: Given an integer array nums, find the subarray with the largest sum, and return its sum.

Code:

```
class Solution
{
  public int maxSubArray(int[] nums) {
    int ans = Integer.MIN_VALUE;
    int sum = 0;

    for (final int num : nums) {
      sum = Math.max(num, sum + num);
      ans = Math.max(ans, sum);
    }

    return ans;
  }
}
```

OUTPUT:



PROBLEM 5:

Aim: Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

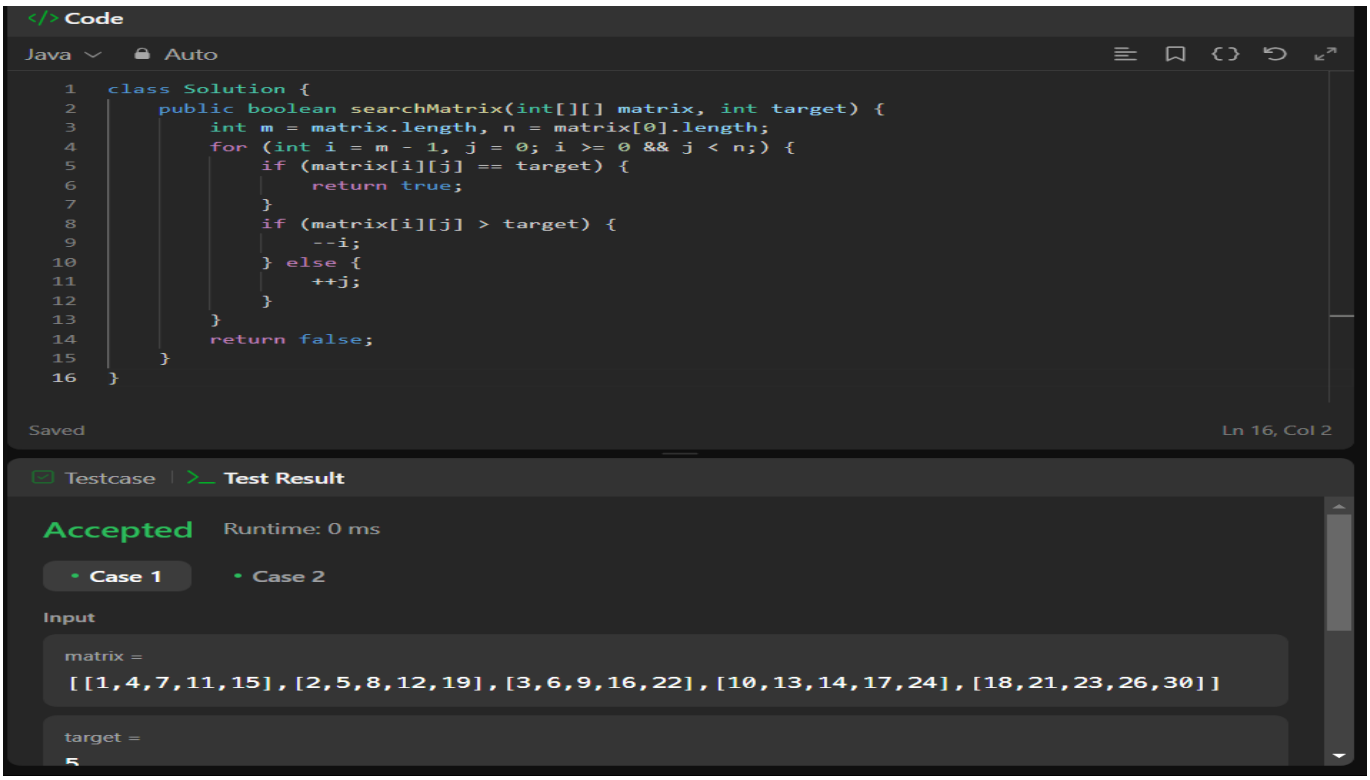
- **Integers in each row are sorted in ascending from left to right.**
- **Integers in each column are sorted in ascending from top to bottom.**

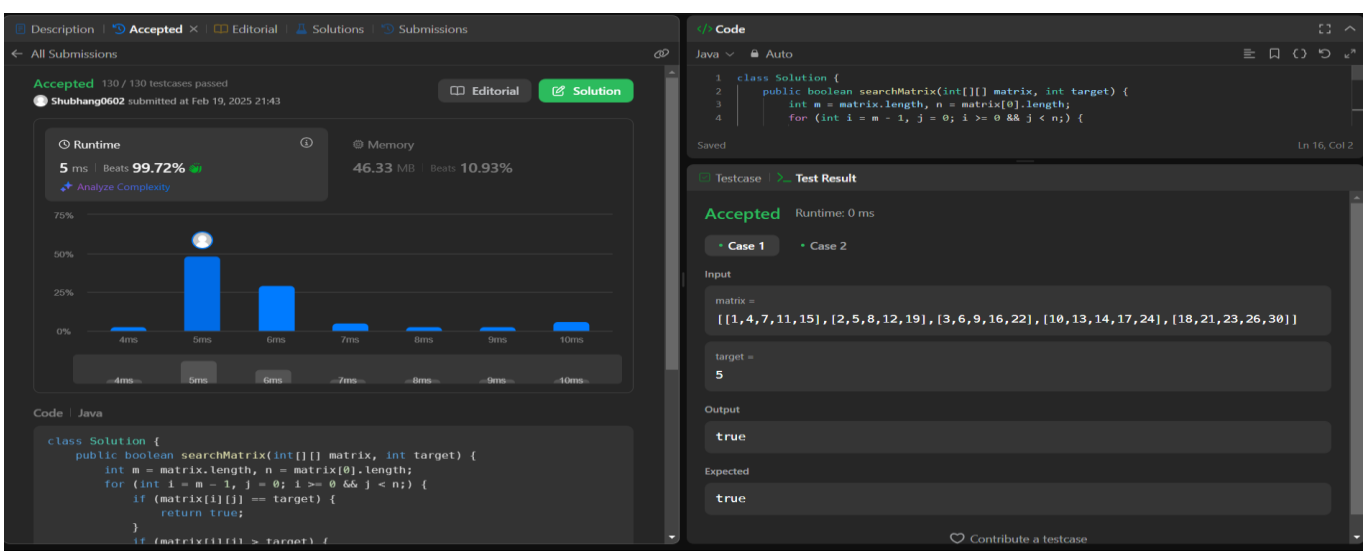
Code:

```
class Solution {
  public boolean searchMatrix(int[][] matrix, int target) {
    int m = matrix.length, n = matrix[0].length;
    int left = 0, right = m * n - 1;
    while (left < right) {
      int mid = (left + right) >> 1;
      int x = mid / n, y = mid % n;
      if (matrix[x][y] >= target) {
        right = mid;
      } else {
```

```
                left = mid + 1;
            }
        }
        return matrix[left / n][left % n] == target;
    }
}
```

Output:





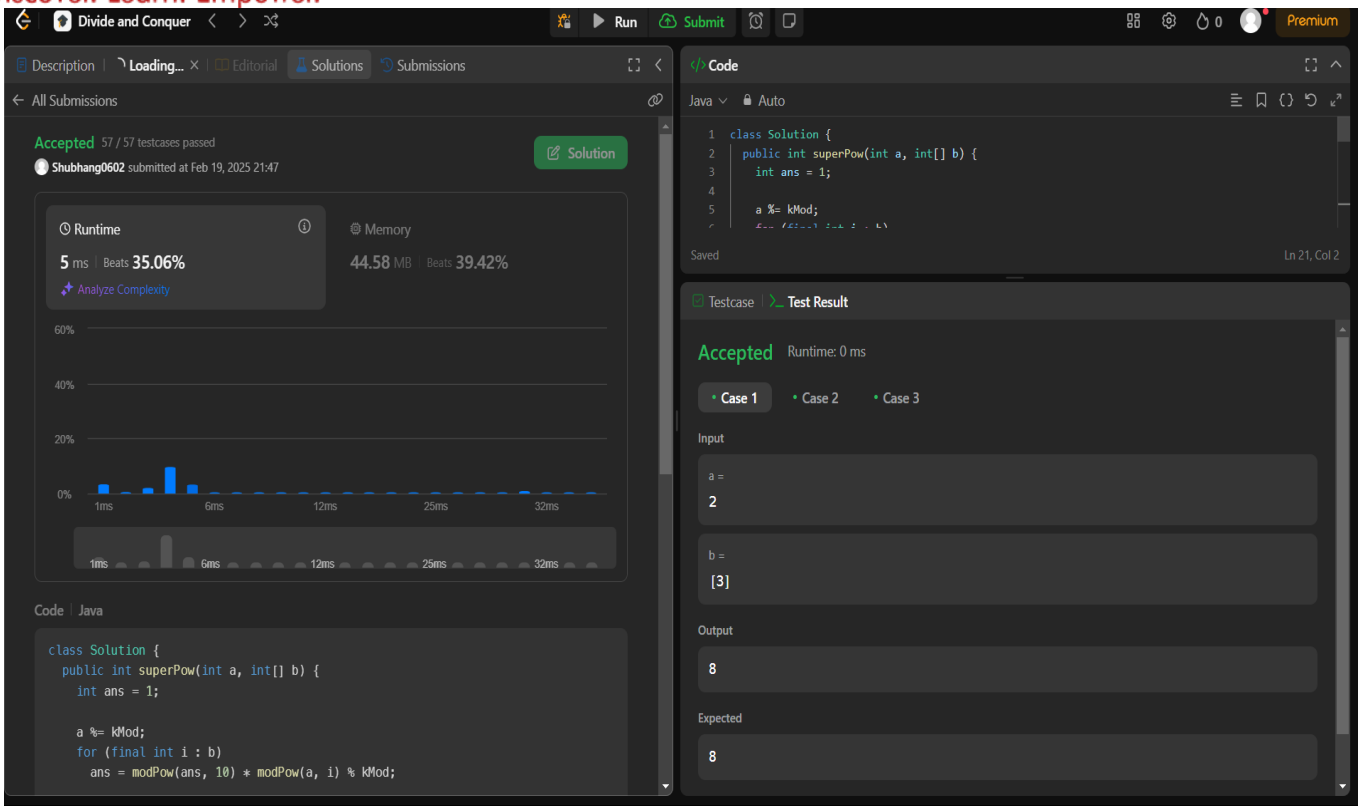
PROBLEM 6:

Aim: Your task is to calculate ab mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

Code:

```
class Solution {
  public int superPow(int a, int[] b) {
    int ans = 1;

    a %= kMod;
    for (final int i : b)
      ans = modPow(ans, 10) * modPow(a, i) % kMod;

    return ans;
  }

  private static final int kMod = 1337;

  private int modPow(int x, int n) {
    if (n == 0)
      return 1;
    if (n % 2 == 1)
      return x * modPow(x % kMod, (n - 1)) % kMod;
    return modPow(x * x % kMod, (n / 2)) % kMod;
  }
}
```

Output:



PROBLEM 7:

Aim: An array nums of length n is beautiful if nums is a permutation of the integers in the range [1, n].
For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums[k] == nums[i] + nums[j].
Given the integer n, return any beautiful array nums of length n. There will be at least one valid answer for the given n.

Code:

```
class Solution {
 public int[] beautifulArray(int n) {
  int[] arr = new int[n];
  for (int i = 0; i < n; ++i)
   arr[i] = i + 1;
  divide(arr, 0, n - 1, 1);
  return arr;
 }

 private void divide(int[] arr, int l, int r, int mask) {
```

```
    if (l >= r)
      return;
    final int m = partition(arr, l, r, mask);
    divide(arr, l, m, mask << 1);
    divide(arr, m + 1, r, mask << 1);
  }

  private int partition(int[] arr, int l, int r, int mask) {
    int nextSwapped = l;
    for (int i = l; i <= r; ++i)
      if ((arr[i] & mask) > 0)
        swap(arr, i, nextSwapped++);
    return nextSwapped - 1;
  }

  private void swap(int[] arr, int i, int j) {
    final int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
  }
}
```

Output: