Experiment 4

Student Name: Khushmn Sangha **UID:** 22BCS14585
Branch: BE-CSE **Section/Group:** 602/A
Semester: 6th **Date of Performance:**21-02-25
Subject Name: Advanced Programming - II **Subject Code:** 22CSP-351

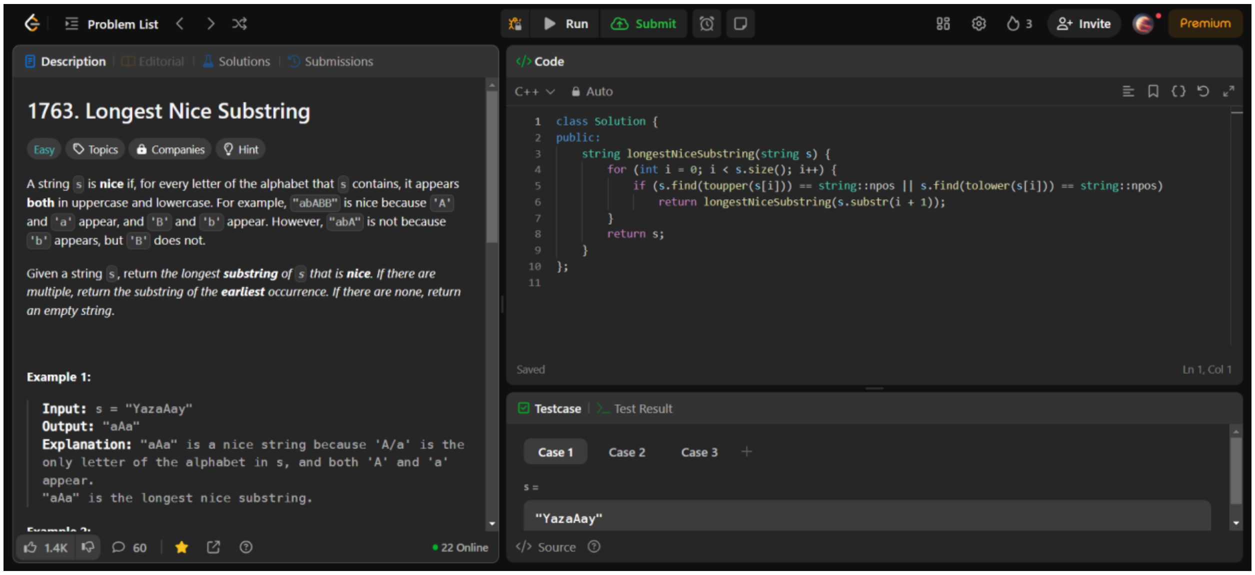
1763. Longest Nice Substring

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) {
        for (int i = 0; i < s.size(); i++) {
            if (s.find(toupper(s[i])) == string::npos || s.find(tolower(s[i])) == string::npos)
                return longestNiceSubstring(s.substr(i + 1));
        }
        return s;
    }
};
```



1763. Longest Nice Substring

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.*

Example 1:

```
Input: s = "YazaAay"
Output: "aAa"
Explanation: "aAa" is a nice string because 'A/a' is the
only letter of the alphabet in s, and both 'A' and 'a'
appear.
"aAa" is the longest nice substring.
```

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        for (int i = 0; i < s.size(); i++) {
            if (s.find(toupper(s[i])) == string::npos || s.find(tolower(s[i])) == string::npos)
                return longestNiceSubstring(s.substr(i + 1));
        }
        return s;
    }
};
```

s =

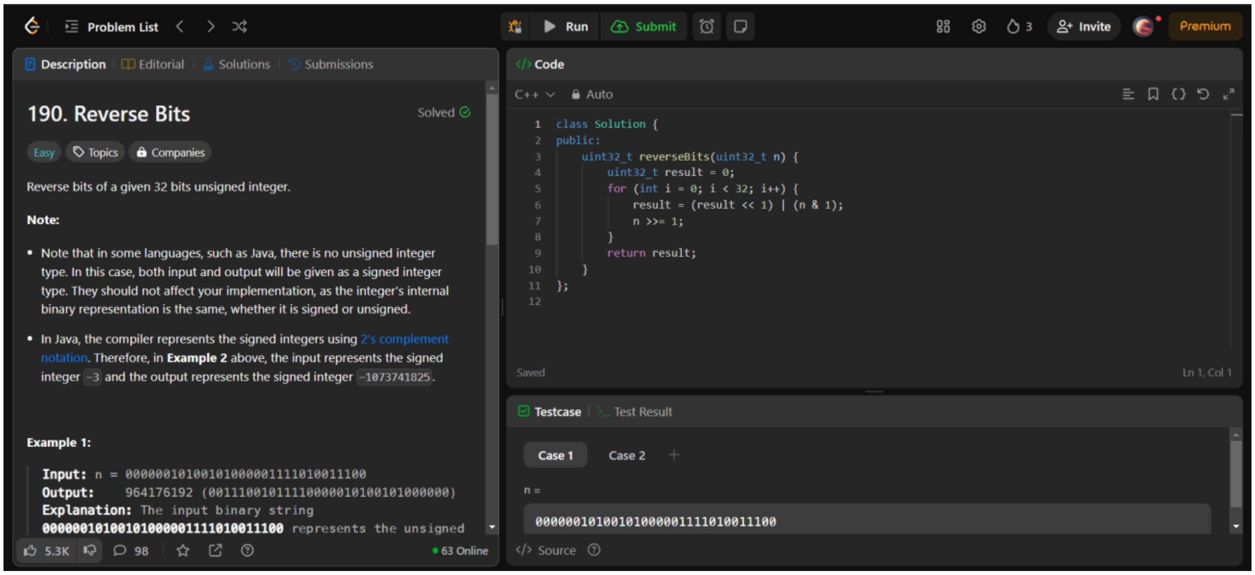
"YazaAay"

190. Reverse Bits

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

CODE:-

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t result = 0;
        for (int i = 0; i < 32; i++) {
            result = (result << 1) | (n & 1);
            n >>= 1;
        }
        return result;
    }
};
```



190. Reverse Bits

Reverse bits of a given 32 bits unsigned integer.

Note:

- Note that in some languages, such as Java, there is no unsigned integer type. In this case, both input and output will be given as a signed integer type. They should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 2** above, the input represents the signed integer `-3` and the output represents the signed integer `-1073741825`.

Example 1:

```
Input: n = 00000010100101000001111010011100
Output:    964176192 (00111001011110000010100101000000)
Explanation: The input binary string
00000010100101000001111010011100 represents the unsigned
```

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t result = 0;
        for (int i = 0; i < 32; i++) {
            result = (result << 1) | (n & 1);
            n >>= 1;
        }
        return result;
    }
};
```

n =

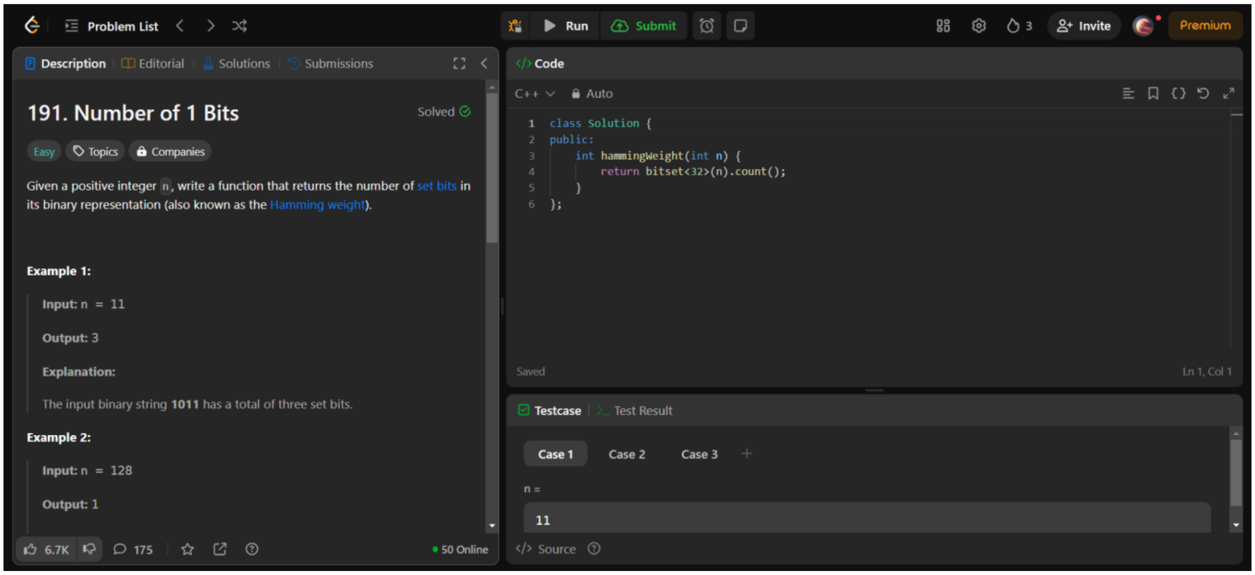
```
00000010100101000001111010011100
```

191. Number of 1 Bits

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:-

```
class Solution {
public:
    int hammingWeight(int n) {
        return bitset<32>(n).count();
    }
};
```



191. Number of 1 Bits

Solved

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).

Example 1:

Input: n = 11

Output: 3

Explanation:

The input binary string **1011** has a total of three set bits.

Example 2:

Input: n = 128

Output: 1

```
class Solution {
public:
    int hammingWeight(int n) {
        return bitset<32>(n).count();
    }
};
```

Saved

Ln 1, Col 1

n =

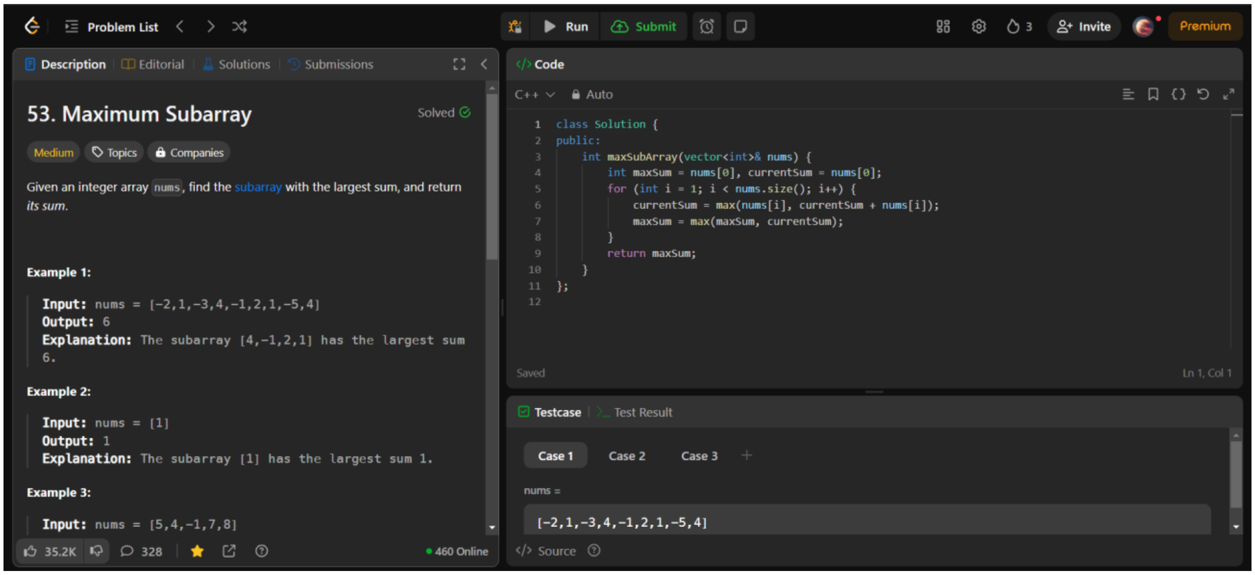
11

53. Maximum Subarray

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

CODE:-

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = nums[0], currentSum = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }
        return maxSum;
    }
};
```



53. Maximum Subarray

Solved

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

Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.
```

Example 3:

```
Input: nums = [5,4,-1,7,8]
```

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = nums[0], currentSum = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }
        return maxSum;
    }
};
```

nums =

```
[-2,1,-3,4,-1,2,1,-5,4]
```

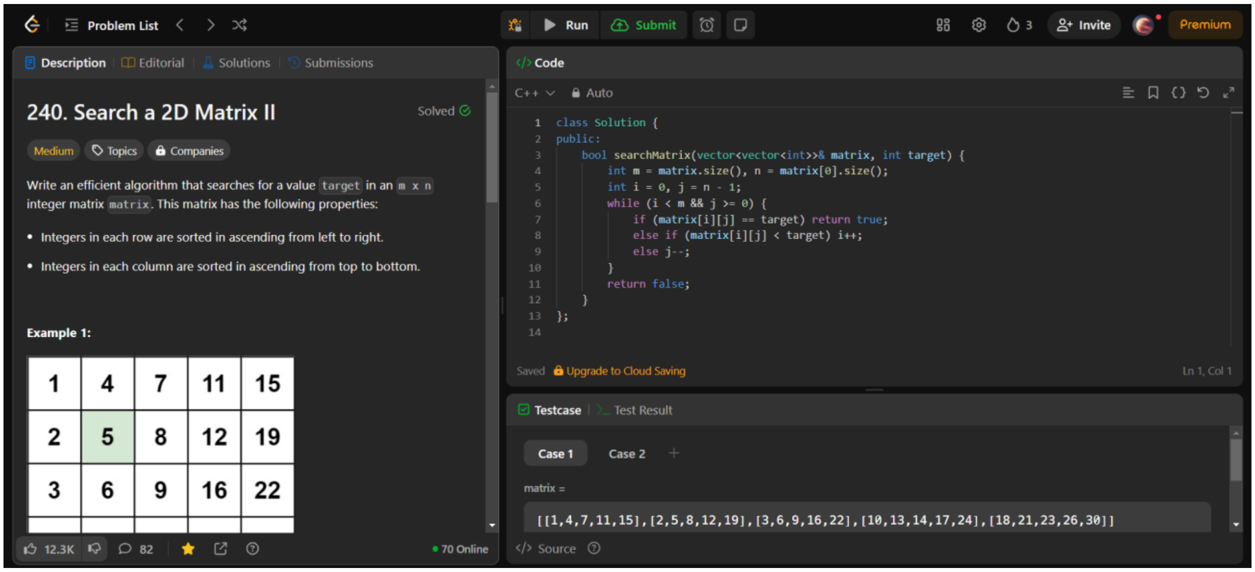
240. Search a 2D Matrix

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:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size(), n = matrix[0].size();
        int i = 0, j = n - 1;
        while (i < m && j >= 0) {
            if (matrix[i][j] == target) return true;
            else if (matrix[i][j] < target) i++;
            else j--;
        }
        return false;
    }
};
```



240. Search a 2D Matrix II

Solved

Medium · Topics · Companies

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.

Example 1:

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size(), n = matrix[0].size();
        int i = 0, j = n - 1;
        while (i < m && j >= 0) {
            if (matrix[i][j] == target) return true;
            else if (matrix[i][j] < target) i++;
            else j--;
        }
        return false;
    }
};
```

matrix =

```
[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]
```

372. Super Pow

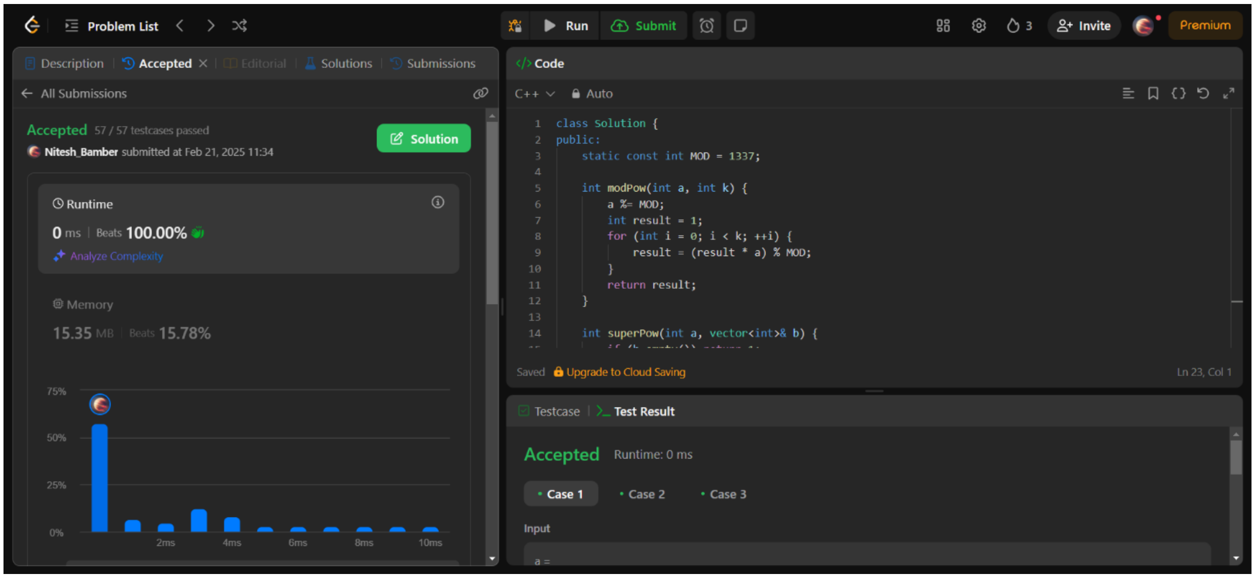
Aim- Your task is to calculate a^b mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

CODE:-

```
class Solution {
public:
    static const int MOD = 1337;

    int modPow(int a, int k) {
        a %= MOD;
        int result = 1;
        for (int i = 0; i < k; ++i) {
            result = (result * a) % MOD;
        }
        return result;
    }

    int superPow(int a, vector<int>& b) {
        if (b.empty()) return 1;
        int lastDigit = b.back();
        b.pop_back();
        int part1 = modPow(a, lastDigit);
        int part2 = modPow(superPow(a, b), 10);
        return (part1 * part2) % MOD;
    }
};
```



Accepted 57 / 57 testcases passed

Nitesh_Bamber submitted at Feb 21, 2025 11:34

Runtime

0 ms | Beats 100.00%

Memory

15.35 MB | Beats 15.78%

```
class Solution {
public:
    static const int MOD = 1337;

    int modPow(int a, int k) {
        a %= MOD;
        int result = 1;
        for (int i = 0; i < k; ++i) {
            result = (result * a) % MOD;
        }
        return result;
    }

    int superPow(int a, vector<int>& b) {
```

Saved Upgrade to Cloud Saving

Ln 23, Col 1

Accepted Runtime: 0 ms

Input

a =