

## ASSIGNMENT -4 (ADVANCED PROGRAMMING)

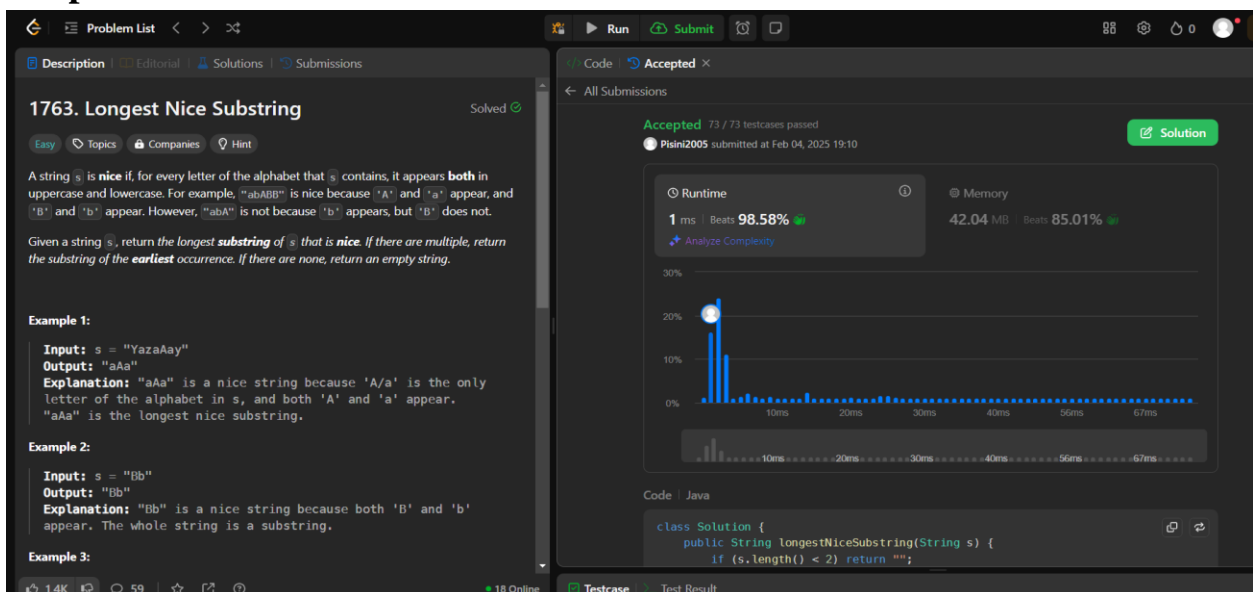
Tapan Kumar– 22BCS10806

### 1. Problem 1: Longest Nice Substring

### 2. Implementation/Code:

```
class Solution {
    public String longestNiceSubstring(String s) {
        if (s.length() < 2) return "";
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (s.contains(Character.toString(Character.toLowerCase(ch))) &&
                s.contains(Character.toString(Character.toUpperCase(ch)))) {
                continue;
            }
            String left = longestNiceSubstring(s.substring(0, i));
            String right = longestNiceSubstring(s.substring(i + 1));
            return left.length() >= right.length() ? left : right;
        }
        return s;
    }
}
```

### 3. Output:

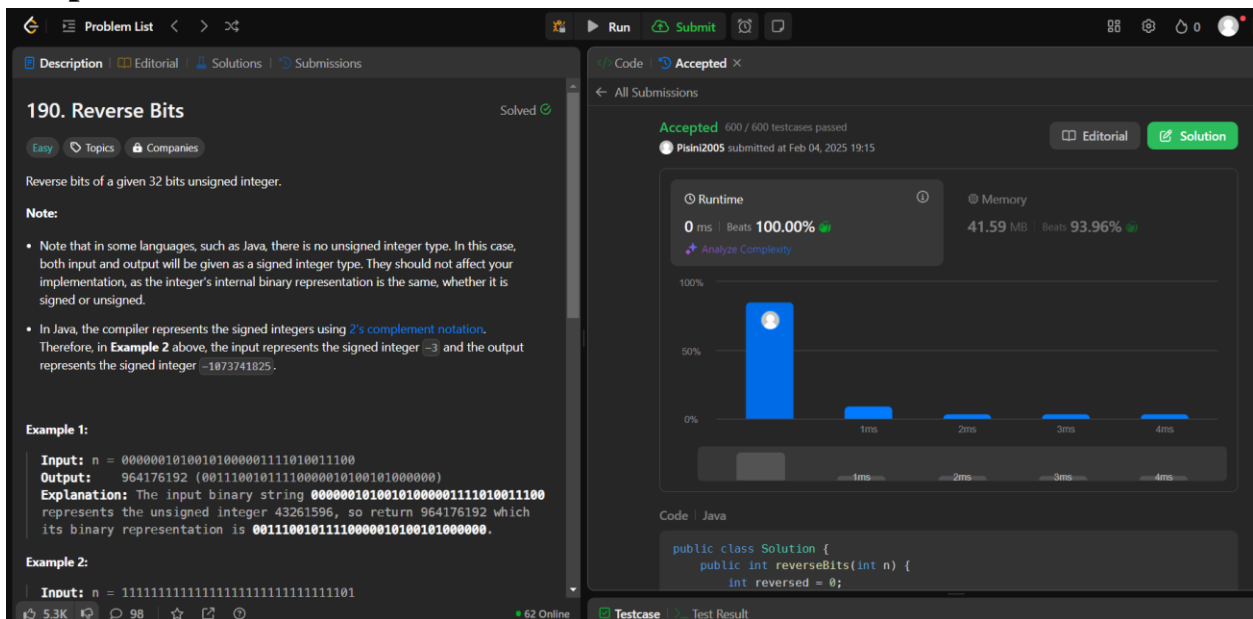


## 1. Problem 2: Reverse Bits

## 2. Implementation/Code:

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

## 3. Output:



The screenshot displays a coding platform interface for the 'Reverse Bits' problem (Problem 190). The left panel shows the problem description, which includes a note about unsigned integers and a Java-specific note about signed integers and 2's complement notation. It also provides two examples: Example 1 with input n = 00000010100101000001111010011100 and output 964176192, and Example 2 with input n = 11111111111111111111111111111101. The right panel shows the submission results, indicating that the solution was accepted, passed 600/600 testcases, and achieved a runtime of 0 ms (Beats 100.00%) and memory usage of 41.59 MB (Beats 93.96%). A bar chart shows the runtime distribution, with a single bar at 0 ms. The code editor shows the Java solution for reversing bits.

**190. Reverse Bits** Solved

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 (0011100101111000010100101000000)  
Explanation: The input binary string 00000010100101000001111010011100 represents the unsigned integer 43261596, so return 964176192 which its binary representation is 0011100101110000010100101000000.

**Example 2:**

Input: n = 11111111111111111111111111111101

Accepted 600 / 600 testcases passed  
Pisini2005 submitted at Feb 04, 2025 19:15

Runtime: 0 ms | Beats 100.00%  
Memory: 41.59 MB | Beats 93.96%

Code: Java

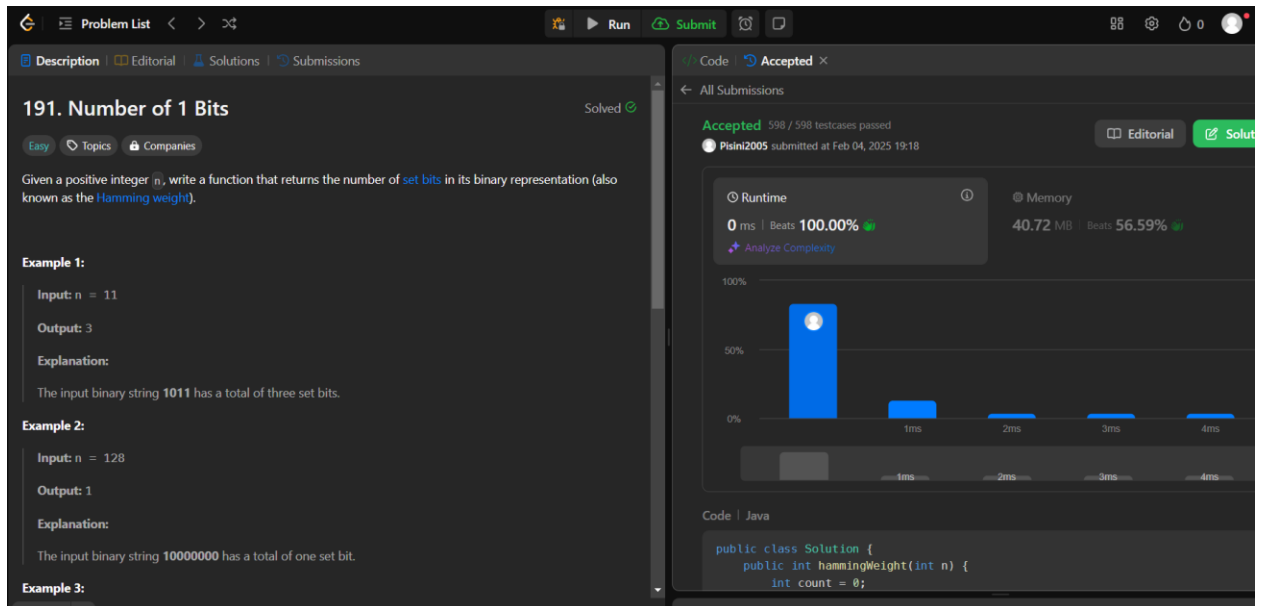
```
public class Solution {
    public int reverseBits(int n) {
        int reversed = 0;
    }
```

## 1. Problem 3: Number of 1 bits

## 2. Implementation/code:

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

## 3. Output:

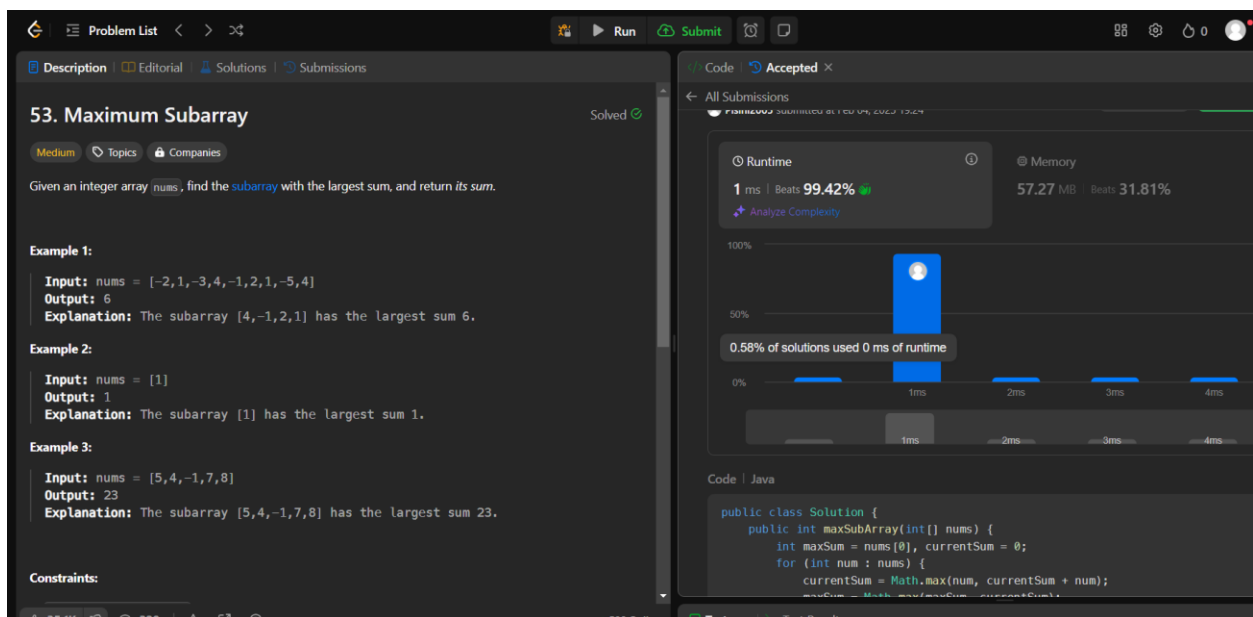


## 1. Problem 4: Maximum Sub array

## 2. Implementation/code:

```
public class Solution {  
    public int maxSubArray(int[] nums) {  
        int maxSum = nums[0], currentSum = 0;  
        for (int num : nums) {  
            currentSum = Math.max(num, currentSum + num);  
            maxSum = Math.max(maxSum, currentSum);  
        }  
        return maxSum;  
    }  
}
```

## 3. Output:

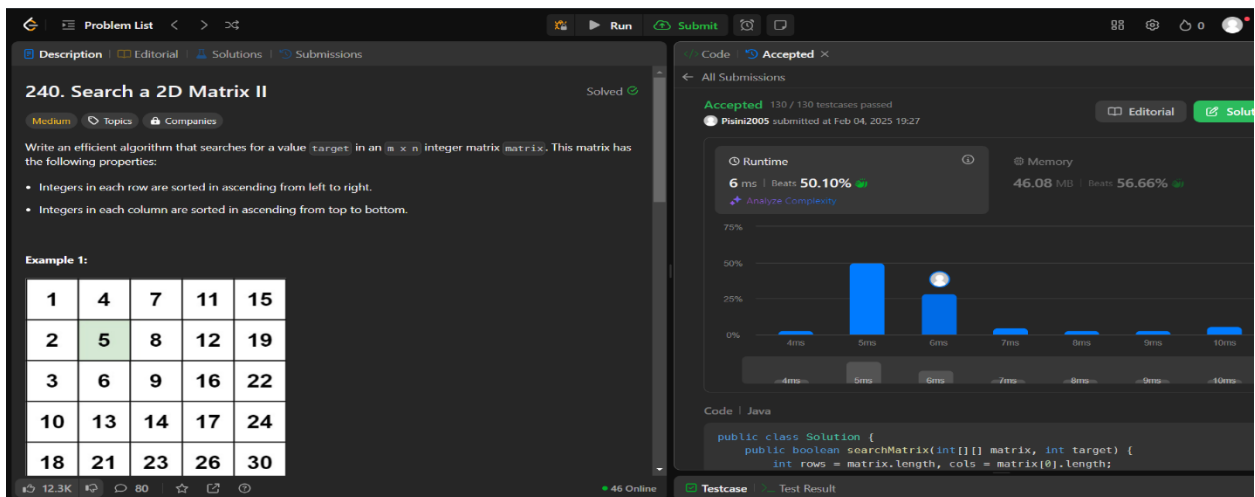


## 1. Problem 5: Search a 2D Matrix II

## 2. Implementation/Code:

```
public class Solution {  
    public boolean searchMatrix(int[][] matrix, int target) {  
        int rows = matrix.length, cols = matrix[0].length;  
        int row = 0, col = cols - 1;  
        while (row < rows && col >= 0) {  
            if (matrix[row][col] == target) {  
                return true;  
            } else if (matrix[row][col] < target) {  
                row++;  
            } else {  
                col--;  
            }  
        }  
        return false;  
    }  
}
```

## 3. Output:



## 1. Problem 6: Super Pow

## 2. Implementation/Code:

```
public class Solution {  
    private static final int MOD = 1337;  
    private int pow(int a, int b) {  
        int res = 1;  
        a %= MOD;  
        for (int i = 0; i < b; i++) {  
            res = (res * a) % MOD;  
        }  
        return res;    }  
    public int superPow(int a, int[] b) {  
        int res = 1;  
        for (int i = b.length - 1; i >= 0; i--) {  
            res = (res * pow(a, b[i])) % MOD;  
            a = pow(a, 10);  
        }  
        return res;    }  
}
```

## 3. Output:

