Experiment-4

Student Name: Garv Kumar **UID:** 22BET10103

Branch: BE-IT **Section/Group:** 22BET_IOT-702/A

Semester: 6th **Date of Performance:** 13/02/2025

Subject Name: Advanced Programming Lab-2 **Subject Code:** 22ITP-351

Problem-1

1. Aim:

Find the longest substring where every letter appears in both uppercase and lowercase. Return the earliest occurrence if multiple exist; return an empty string if none exist.

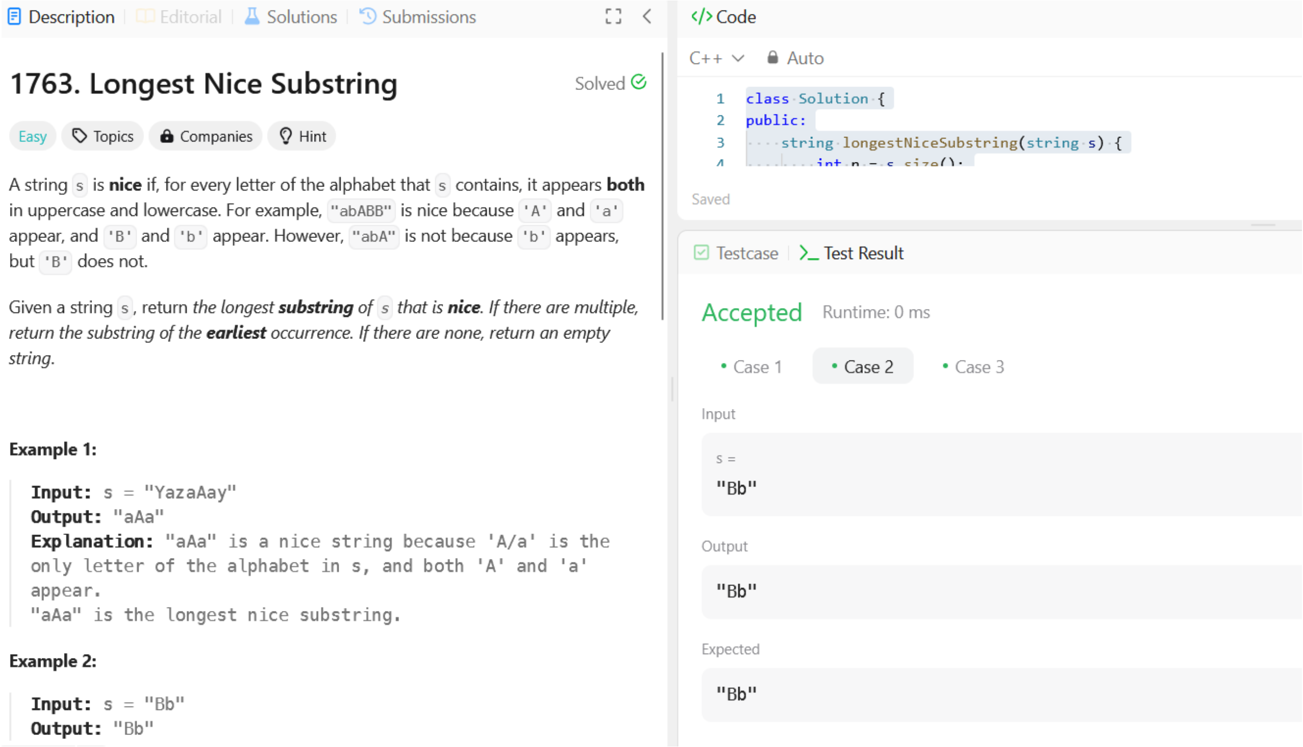
2. Objective:

- Identify the longest contiguous substring where each letter appears in both uppercase and lowercase.
- Return the earliest such substring if multiple exist; otherwise, return an empty string.

3. Implementation:

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        int n = s.size();
        for (int len = n; len > 0; len--) {
            for (int i = 0; i + len <= n; i++) {
                string sub = s.substr(i, len);
                unordered_set<char> st(sub.begin(), sub.end());
                bool nice = true;
                for (char c : sub) {
                    if (!st.count(tolower(c)) || !st.count(toupper(c))) {
                        nice = false;
                        break;
                    }
                }
                if (nice) return sub;
            }
        }
        return "";
    }
};
```

4. Output:



1763. Longest Nice Substring

Solved ✓

Easy | Topics | Companies | Hint

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.
```

Example 2:

```
Input: s = "Bb"
Output: "Bb"
```

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        int n = s.size();
```

Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input

```
s =
"Bb"
```

Output

```
"Bb"
```

Expected

```
"Bb"
```

Fig: Longest Nice Substring.

Problem-2

1. Aim:

Reverse the bits of a given 32-bit unsigned integer and return the resulting value.

2. Objective:

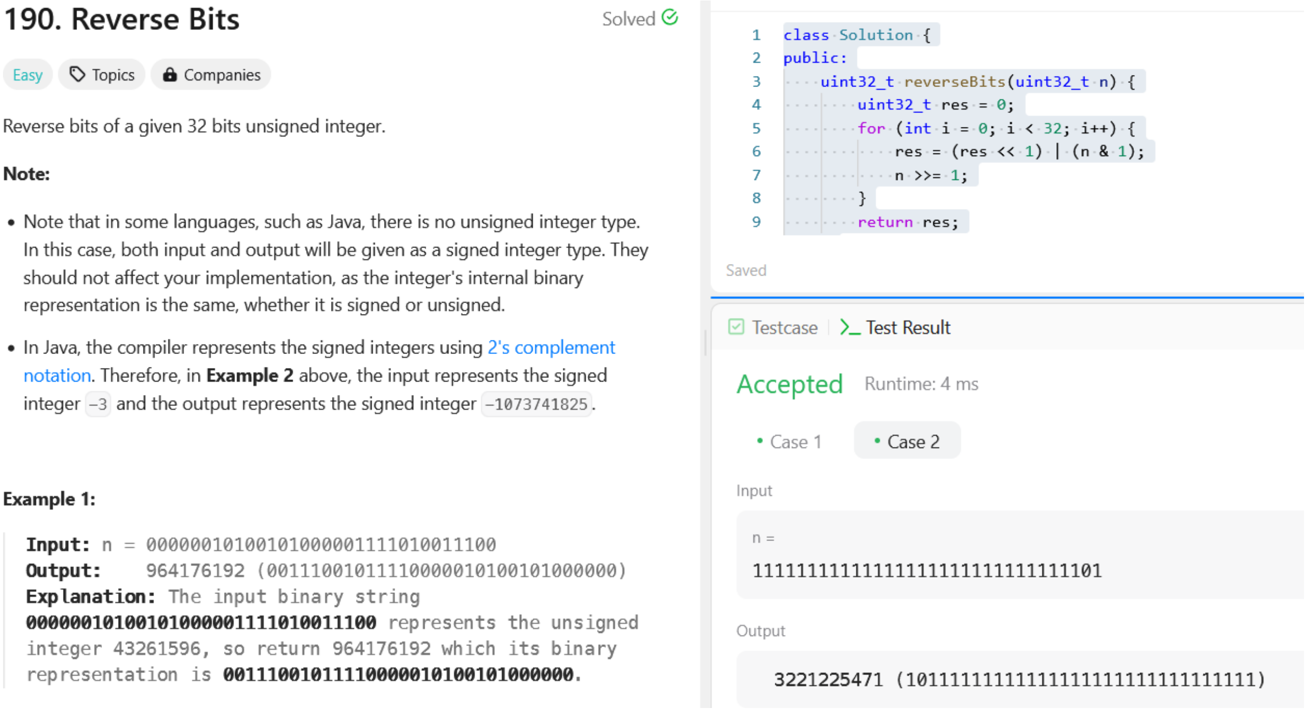
1 Process the 32-bit integer by reversing its binary representation.

2 Return the corresponding integer value of the reversed binary.

3. Implementation:

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

4. Output:



190. Reverse Bits

Solved ✓

Easy | Topics | Companies

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
integer 43261596, so return 964176192 which its binary
representation is 00111001011110000010100101000000.
```

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

Saved

☑ Testcase | >_ Test Result

Accepted Runtime: 4 ms

• Case 1 • Case 2

Input

```
n =
11111111111111111111111111111101
```

Output

```
3221225471 (10111111111111111111111111111111)
```

Fig: Reverse Bits.

Problem-3

1. Aim:

Count the number of set bits (1s) in the binary representation of a given positive integer.

2. Objective:

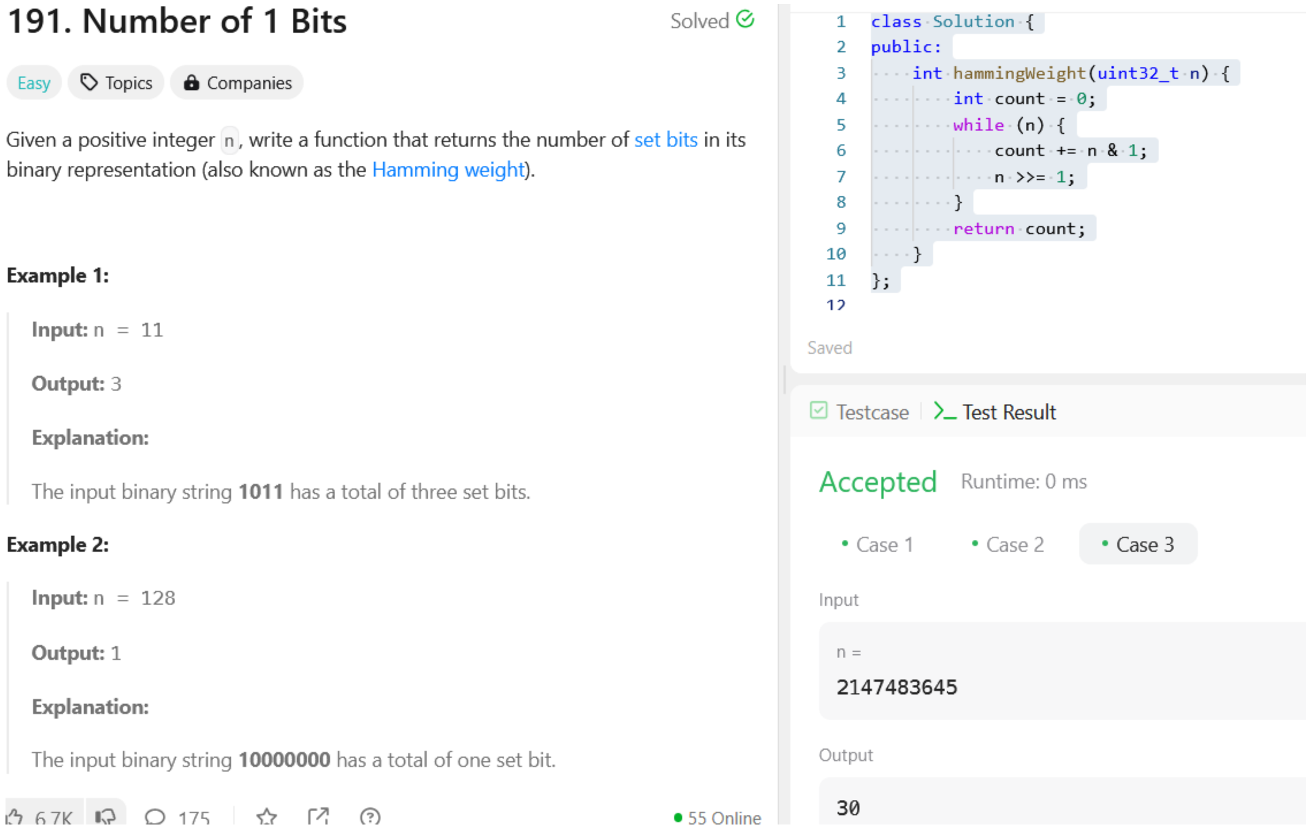
1 Convert the integer to its binary form and count the number of set bits.

2 Return the total count of set bits in the binary representation.

3. Implementation:

```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int count = 0;
        while (n) {
            count += n & 1;
            n >>= 1;
        }
        return count;
    }
 };
```

4. Output:



191. Number of 1 Bits

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

Explanation:

The input binary string **10000000** has a total of one set bit.

```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int count = 0;
        while (n) {
            count += n & 1;
            n >>= 1;
        }
        return count;
    }
};
```

Accepted Runtime: 0 ms

Input

n =

2147483645

Output

30

Fig: Number of 1 Bits.

Problem-4

1. Aim:

Find the contiguous subarray with the largest sum in a given integer array.

2. Objective:

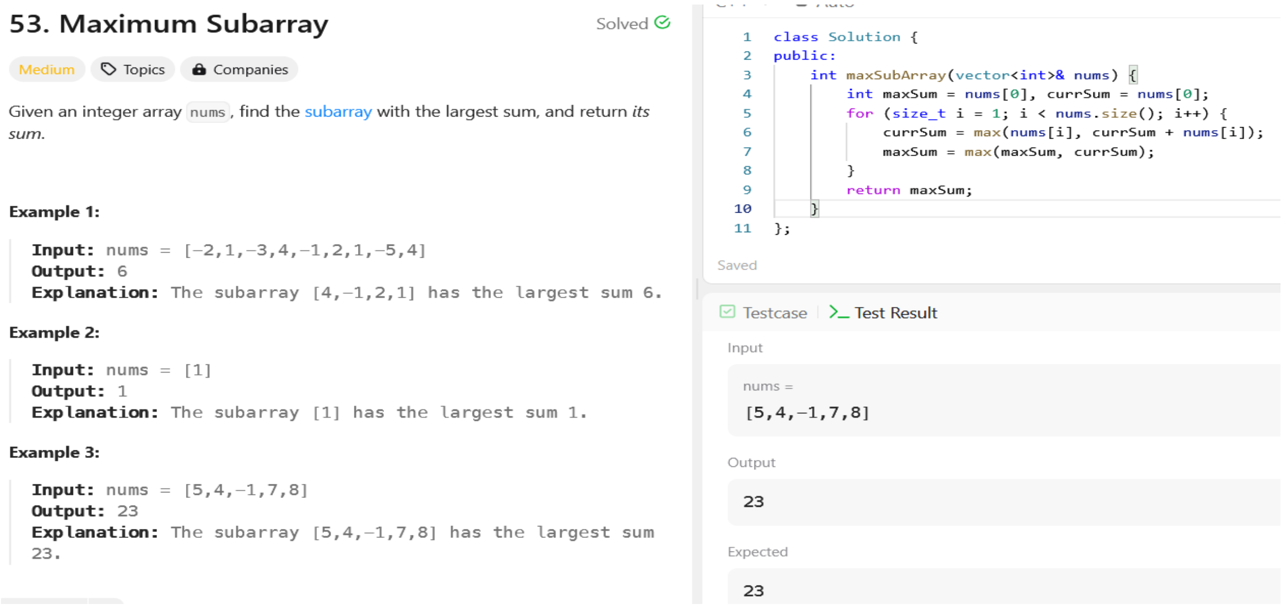
1 Iterate through the array to determine the maximum sum of any contiguous subarray.

2 Return the highest sum found among all possible subarrays.

3. Implementation:

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

4. Output:



53. Maximum Subarray

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]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum
23.
```

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

Input

nums =

```
[5,4,-1,7,8]
```

Output

```
23
```

Expected

```
23
```

Fig: Maximum Subarray.

Problem-5

1. Aim:

Search for a target value in a given m × n matrix where each row and column is sorted in ascending order.

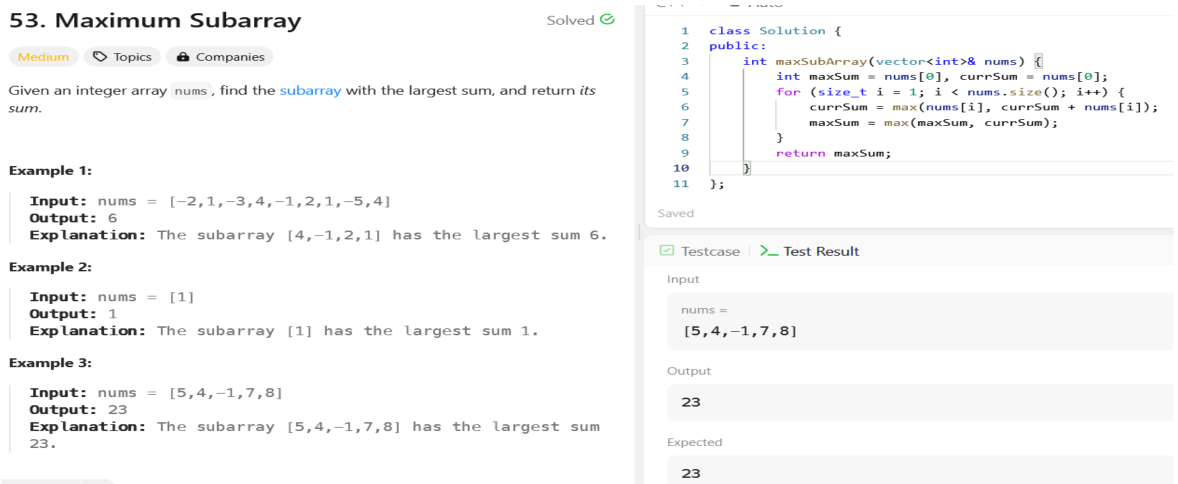
2. Objective:

1 Utilize an efficient search strategy to locate the target within the sorted matrix.
2 Return true if the target is found; otherwise, return false.

3. Implementation:

```
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) j--;
            else i++;
        }
        return false;
    }
 };
```

4. Output:



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]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum
23.
```

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

Input

nums =
[5,4,-1,7,8]

Output

23

Expected

23

Fig: Maximum Subarray.

5. Learning Outcomes:

1 Longest Nice Substring

1. Understand how to identify and extract substrings that satisfy specific character constraints.
2. Develop efficient substring search techniques for solving string manipulation problems.

2 Reverse Bits of a 32-bit Unsigned Integer

1. Learn bitwise operations to manipulate and reverse binary representations.
2. Understand how to convert between binary and integer formats efficiently.

3 Hamming Weight (Count Set Bits)

1. Gain proficiency in counting set bits using bitwise operations.
2. Explore optimized methods like Brian Kernighan's algorithm for bit counting.

4 Maximum Subarray (Kadane's Algorithm)

1. Learn how to apply dynamic programming or greedy approaches to find the largest sum subarray.
2. Understand the significance of maintaining a running sum and updating maximum values efficiently.

5 Search in a Sorted 2D Matrix

1. Develop an understanding of matrix traversal strategies for optimized searching.
2. Implement an efficient search algorithm leveraging the sorted properties of rows and columns.