# Experiment-4

**Student Name: Ayushi Gupta UID:** 22BET10133

**Branch:** BE-IT **Section/Group:** 22BET/IOT-702/B

**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.

1. **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.
2. **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;

}

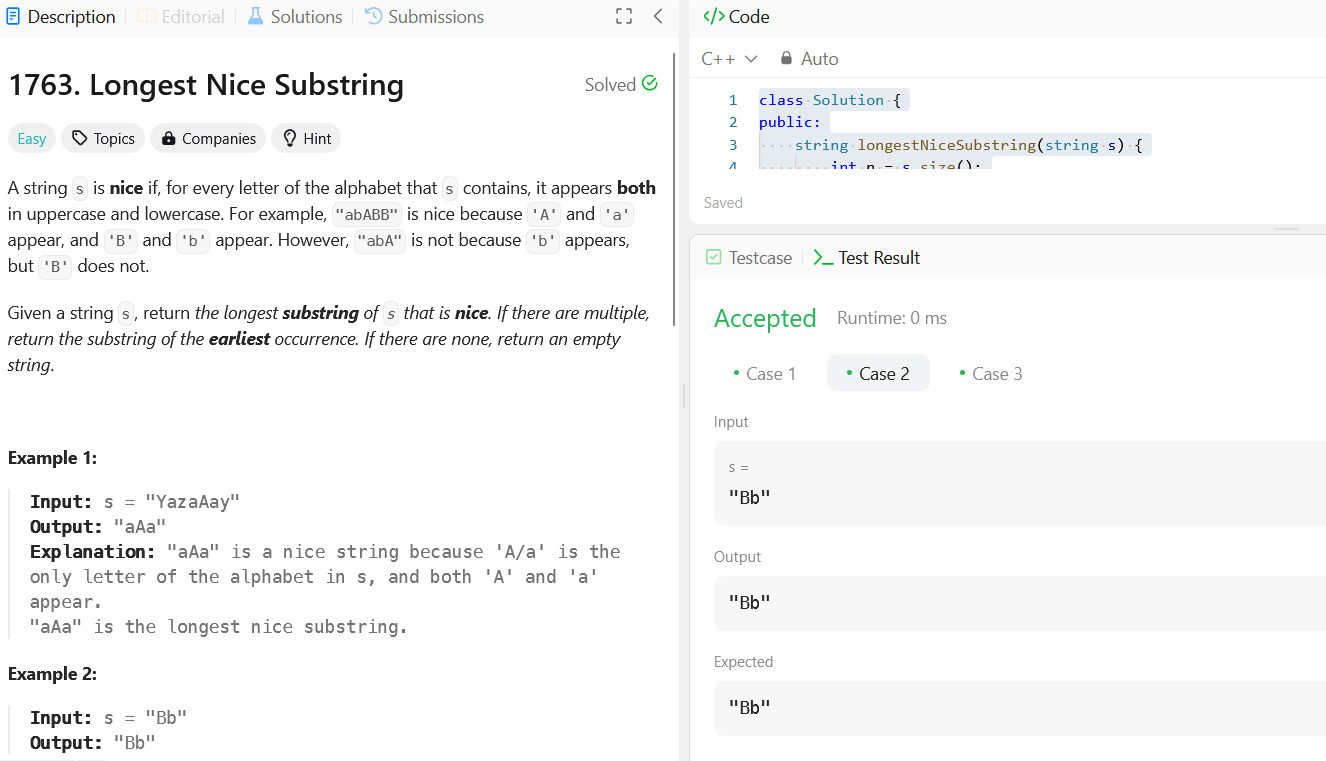
}

return "";

}

};

1. **Output:**

****

***Fig: Longest Nice Substring.***

**Problem-2**

1. **Aim:**

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

1. **Objective:**
2. Process the 32-bit integer by reversing its binary representation.
3. Return the corresponding integer value of the reversed binary.
4. **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;

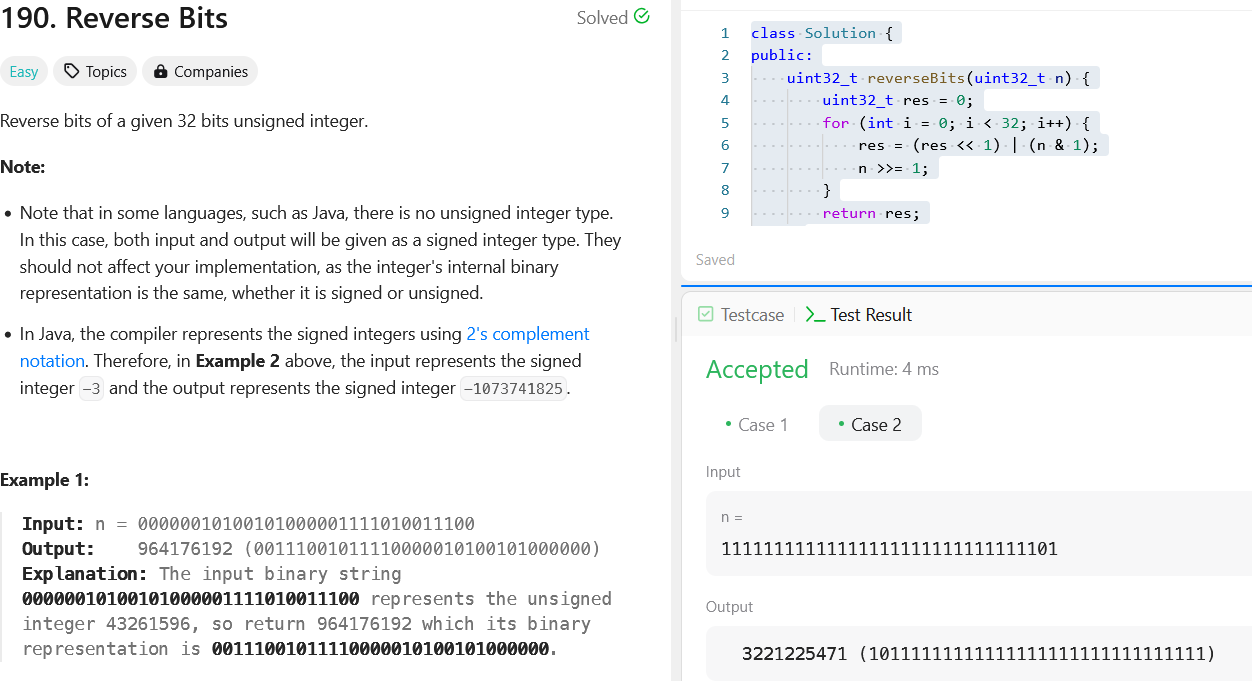
}

return res;

}

};

1. **Output:**

****

***Fig: Reverse Bits.***

**Problem-3**

1. **Aim:**

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

1. **Objective:**
2. Convert the integer to its binary form and count the number of set bits.
3. Return the total count of set bits in the binary representation.
4. **Implementation:**

class Solution {

public:

int hammingWeight(uint32\_t n) {

int count = 0;

while (n) {

count += n & 1;

n >>= 1;

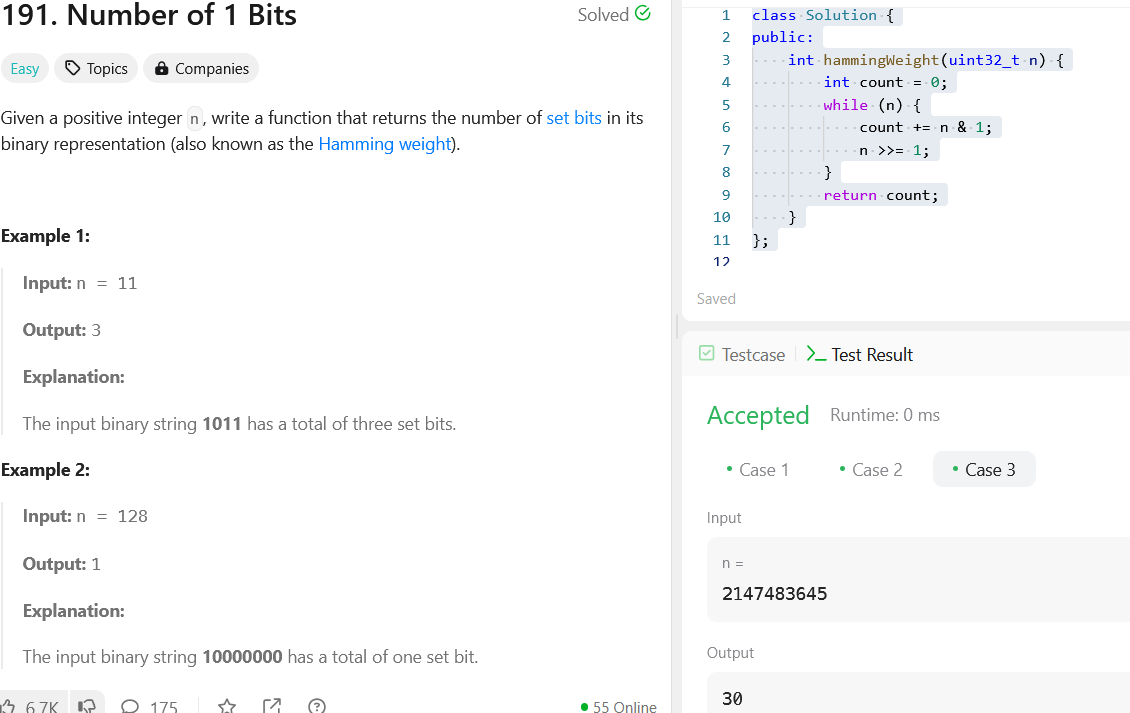
}

return count;

}

};

1. **Output:**

****

***Fig: Number of 1 Bits.***

**Problem-4**

1. **Aim:**

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

1. **Objective:**
2. Iterate through the array to determine the maximum sum of any contiguous subarray.
3. Return the highest sum found among all possible subarrays.
4. **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);

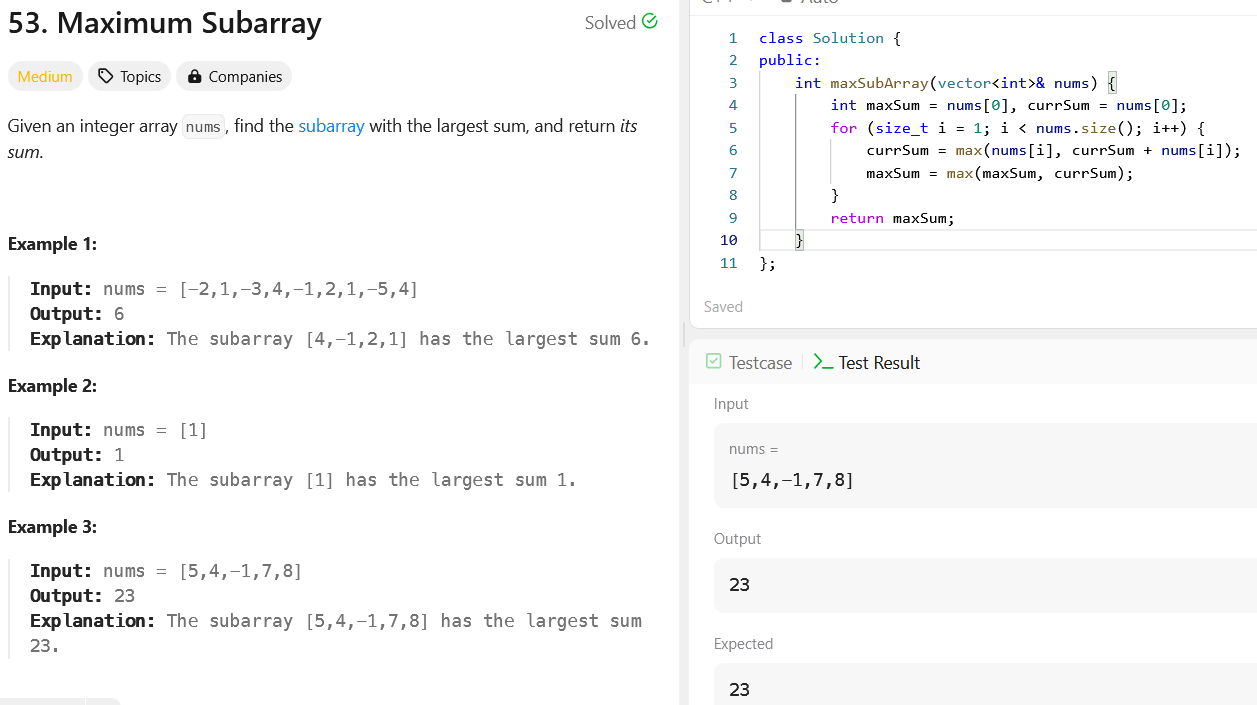
}

return maxSum;

}

};

1. **Output:**

****

***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.

1. **Objective:**
2. Utilize an efficient search strategy to locate the target within the sorted matrix.
3. Return true if the target is found; otherwise, return false.
4. **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++;

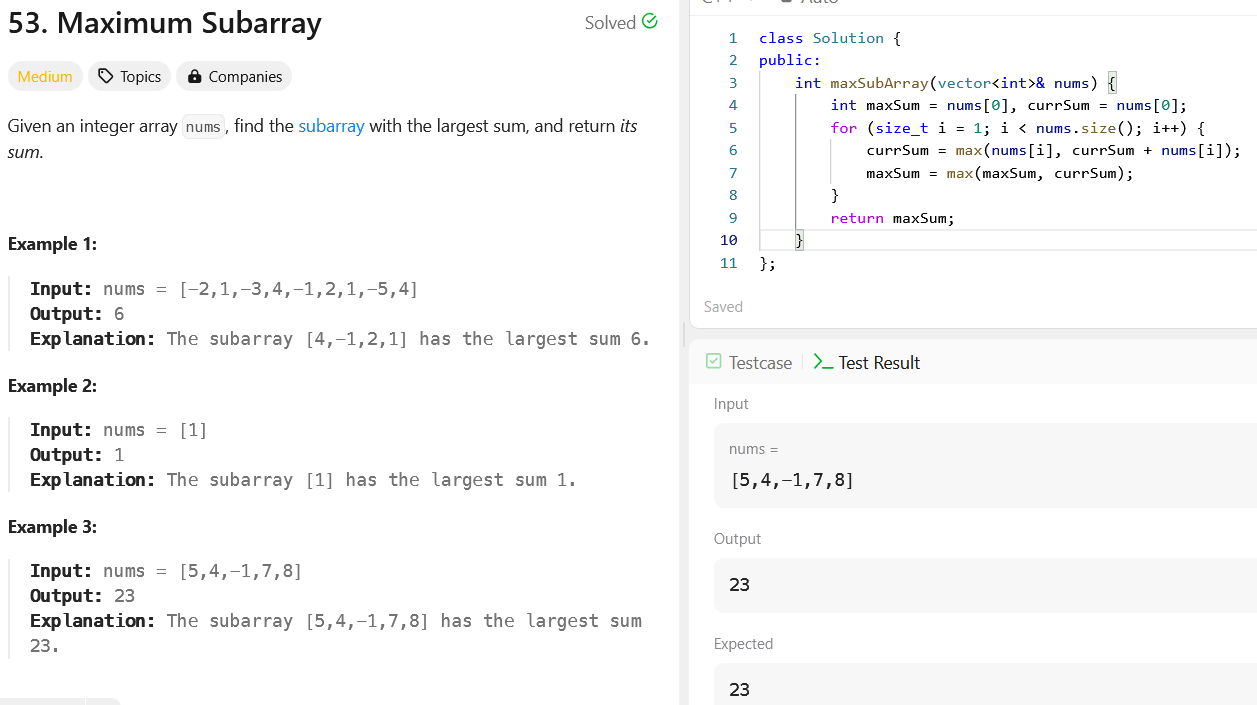
}

return false;

}

};

1. **Output:**

****

***Fig: Maximum Subarray.***

1. **Learning Outcomes:**

### 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.
3. **Reverse Bits of a 32-bit Unsigned Integer**
4. Learn bitwise operations to manipulate and reverse binary representations.
5. Understand how to convert between binary and integer formats efficiently.
6. **Hamming Weight (Count Set Bits)**
7. Gain proficiency in counting set bits using bitwise operations.
8. Explore optimized methods like Brian Kernighan’s algorithm for bit counting.
9. **Maximum Subarray (Kadane’s Algorithm)**
10. Learn how to apply dynamic programming or greedy approaches to find the largest sum subarray.
11. Understand the significance of maintaining a running sum and updating maximum values efficiently.
12. **Search in a Sorted 2D Matrix**
13. Develop an understanding of matrix traversal strategies for optimized searching.
14. Implement an efficient search algorithm leveraging the sorted properties of rows and columns.