**Experiment 4**

**Student Name:** Keshav Mangla **UID:** 22BET10218

**Branch:** BE -IT **Section/Group:**22BET\_IOT-702/A

**Semester:** 6th **Date of Performance:**14/2/2025 **Subject Name:** Advanced Programming Lab-2 **Subject Code:** 22ITP-351

# Longest Nice Substringlist

**Aim :**

To identify the longest substring of a given string that satisfies the "nice" condition. A string is considered nice if, for every letter of the alphabet that it contains, both its uppercase and lowercase forms appear in the string.

# Objectives :

* + Implement a solution to find the longest "nice" substring.
  + Understand string manipulation techniques and how to check for the presence of uppercase and lowercase versions of characters.
  + Apply dynamic programming or sliding window approach for efficient substring extraction.
  + Learn how to handle edge cases like strings with no nice substrings or multiple nice substrings.

**Implementation/Code :**

import java.util.HashSet;

public class Solution {

public String longestNiceSubstring(String s) {

if (s.length() < 2) return ""; // A single character cannot be nice

HashSet<Character> set = new HashSet<>(); for (char c : s.toCharArray()) {

set.add(c);

}

for (int i = 0; i < s.length(); i++) { char c = s.charAt(i);

if (set.contains(Character.toUpperCase(c)) && set.contains(Character.toLowerCase(c))) { continue; // This character satisfies the "nice" condition

}

// Split at the invalid character and solve recursively

String left = longestNiceSubstring(s.substring(0, i)); String right = longestNiceSubstring(s.substring(i + 1));

return left.length() >= right.length() ? left : right;

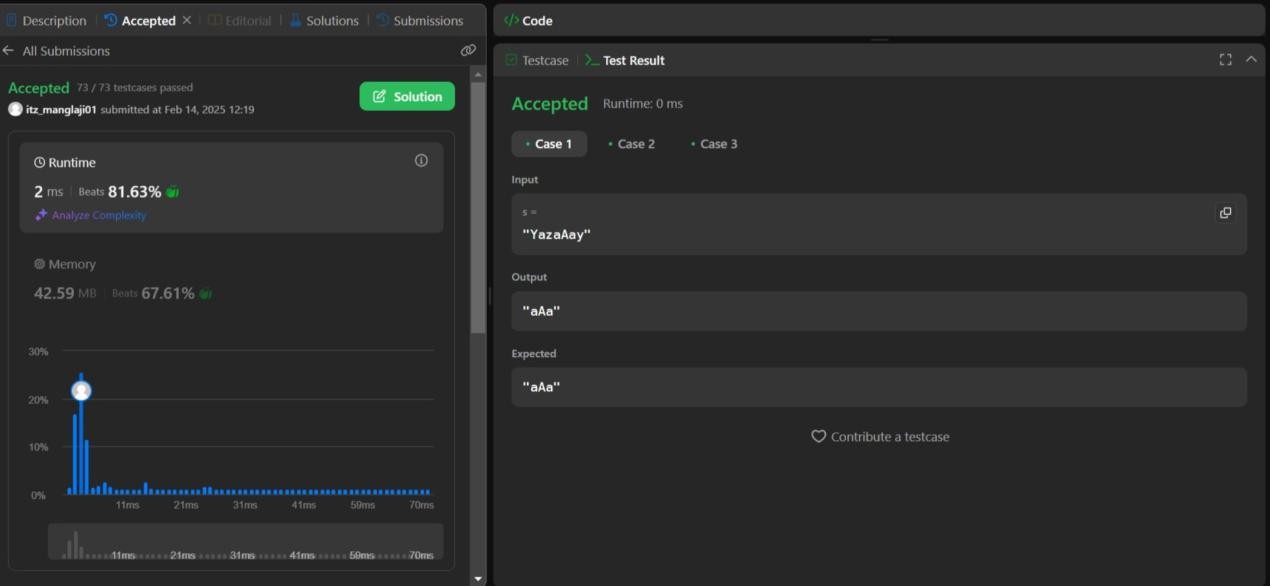
}

return s; // The whole string is nice

}

}

**Output :**

****

# Reverse Bits Aim :

To reverse the bits of a given 32-bit unsigned integer and return the reversed number.

# Objective :

* + Gain a deeper understanding of bitwise operations.
  + Learn to manipulate individual bits of an integer using logical operators such as AND, OR, XOR, and bit shifts.
  + Implement an efficient algorithm to reverse the bits of a 32-bit integer.
  + Understand how signed and unsigned integers are represented in binary, especially in languages like Java where the signed nature of integers may be a consideration.

# Implementation/Code :

public class Solution {

public int reverseBits(int n) { int result = 0;

for (int i = 0; i < 32; i++) {

result <<= 1; // Shift result left to make space

result |= (n & 1); // Append the least significant bit of n

n >>>= 1; // Shift n right (unsigned shift to avoid sign extension)

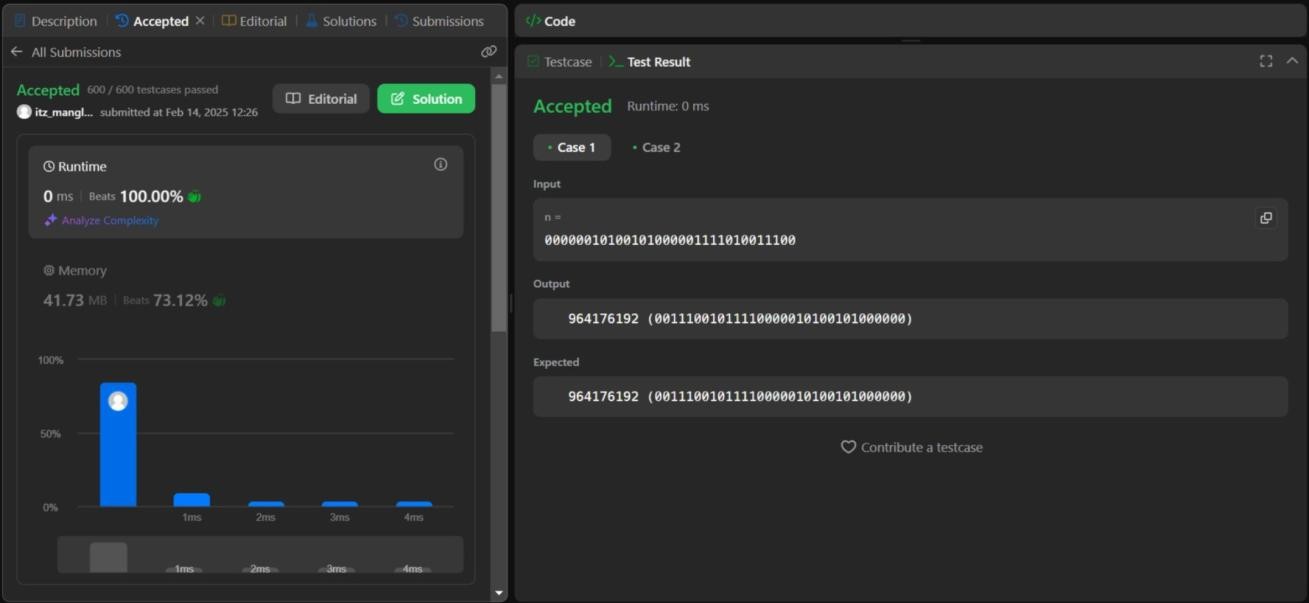
}

return result;

}

}

# Output :

****

1. **Number of 1 Bits Aim :**

To calculate the number of "1" bits (also known as Hamming weight or population count) in the binary representation of a given integer.

# Objective :

* + Learn how to efficiently count the number of 1-bits using bitwise operators.
  + Understand different techniques to count set bits, such as using the "Brian Kernighan’s Algorithm" or simpler iterative approaches.
  + Practice optimizing the bit counting process for large integers.
  + Understand the significance of binary operations in improving algorithm efficiency for various computational problems.

# Implementation/Code :

public class Solution {

public int hammingWeight(int n) { int count = 0;

while (n != 0) { count++;

n &= (n - 1); // Removes the rightmost set bit

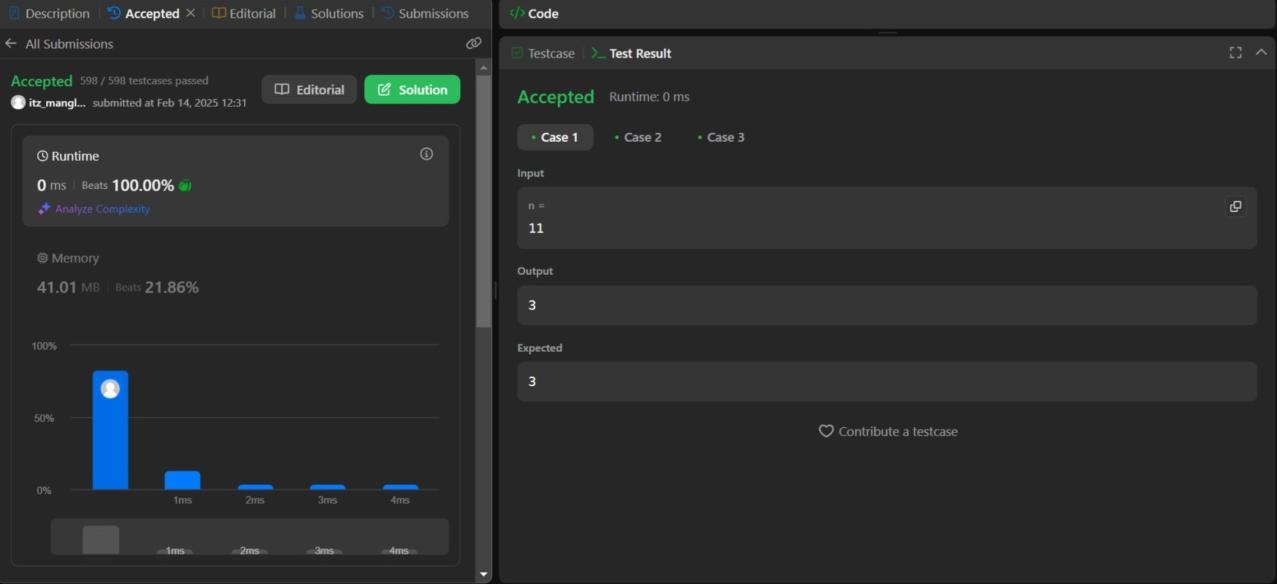
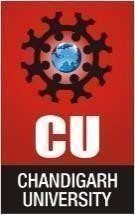
}

return count;

}

}

# Output :



& ENGINEERING



COMPUTER SCIENCE

1. **Max Subarray Aim :**

To find the subarray with the largest sum in a given integer array.

# Objective :

* + Implement Kadane's Algorithm, which efficiently solves the Maximum Subarray Problem in linear time.
  + Learn about dynamic programming and how to apply it in solving subarray sum problems.
  + Analyze the time and space complexity of the algorithm.
  + Understand how to handle both positive and negative integers in the array, and ensure that the algorithm works for arrays of varying sizes.

# Implementation/Code :

public class Solution {

public int maxSubArray(int[] nums) {

int maxSum = nums[0]; // Initialize max sum with first element int currentSum = nums[0]; // Initialize current sum

for (int i = 1; i < nums.length; i++) {

currentSum = Math.max(nums[i], currentSum + nums[i]); // Choose between starting new subarray or extending existing

maxSum = Math.max(maxSum, currentSum); // Update max sum

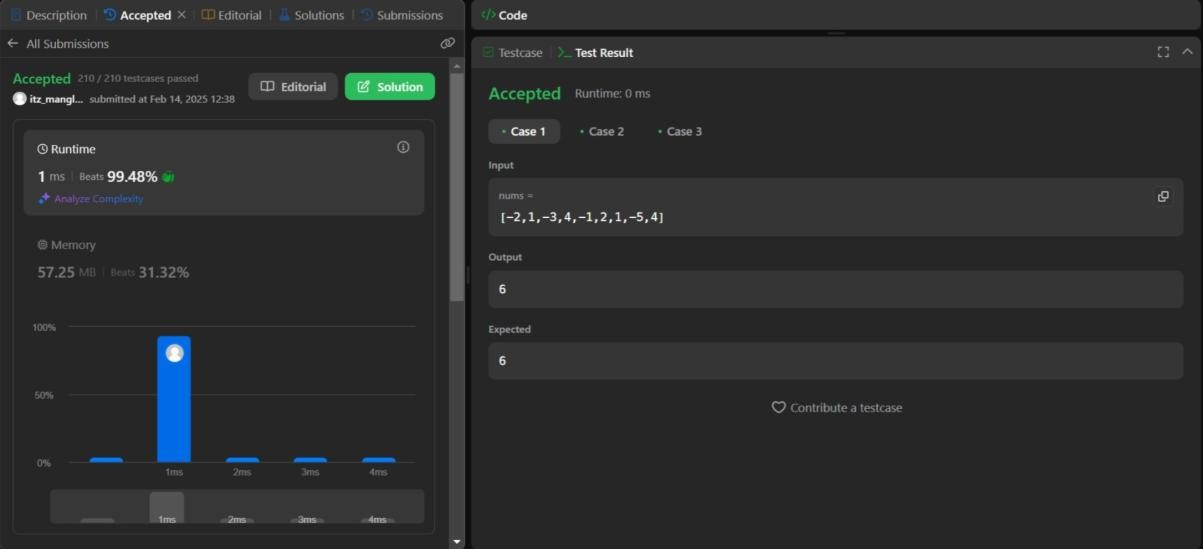
}

return maxSum;

}

}

# Output :

****

1. **Search in 2D Matrix Aim :**

To implement an efficient search algorithm to find a target value in a 2D matrix where the rows and columns are sorted in ascending order.

# Objectives :

* + Explore matrix search strategies such as binary search and staircase search.
  + Learn how to exploit the sorted nature of the matrix for faster search than brute-force approaches.
  + Apply the staircase search technique to reduce the time complexity to O(m + n), where m and n are the number of rows and columns in the matrix.
  + Analyze the constraints and apply optimal space and time complexities for large matrices.

# Implementation/Code :

public class Solution {

public boolean searchMatrix(int[][] matrix, int target) { int m = matrix.length; // Number of rows

int n = matrix[0].length; // Number of columns

int row = 0, col = n - 1; // Start at top-right corner

while (row < m && col >= 0) {

if (matrix[row][col] == target) { return true; // Found target

} else if (matrix[row][col] > target) { col--; // Move left

} else {

row++; // Move down

}

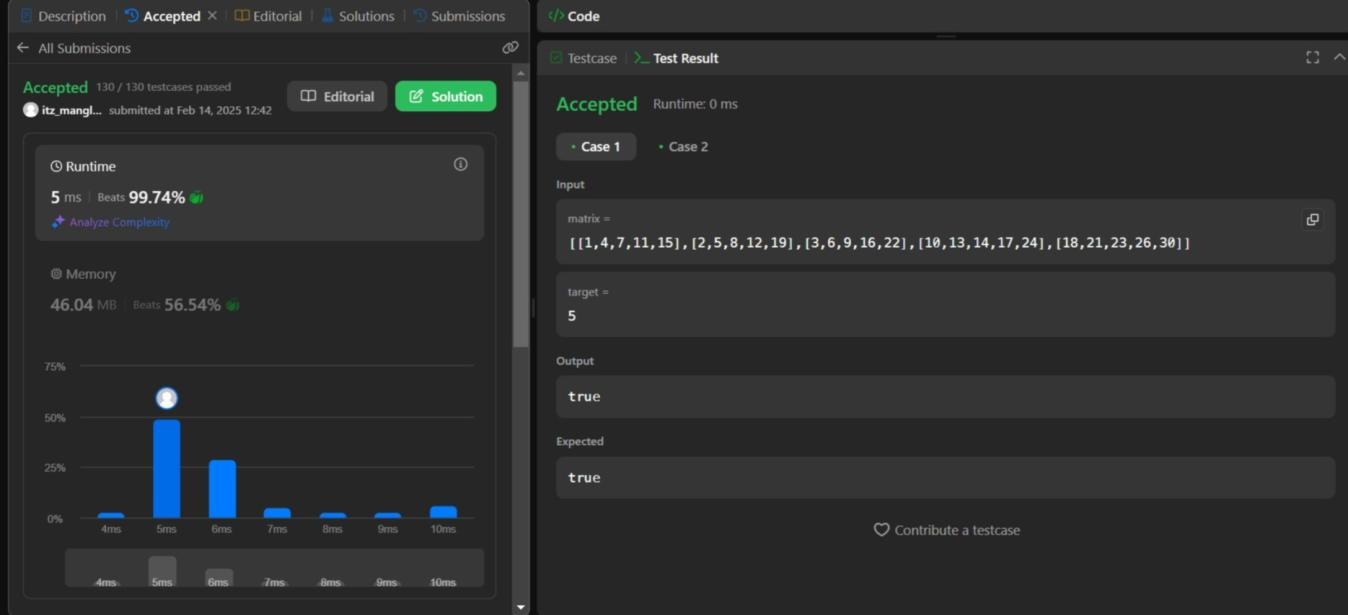
}

return false; // Target not found

}

}

# Output :

****

**Learning Outcomes :**

* + Students will develop a solid understanding of algorithmic approaches for solving diverse problems, such as substring extraction, bit manipulation, and matrix searching.
  + Gain hands-on experience in implementing efficient algorithms, particularly those involving dynamic programming, bitwise operations, and greedy techniques.
  + They will understand how to use data structures and algorithms effectively to exploit properties such as sorted arrays or matrices, reducing the computational cost of operations (e.g., searching, subarray calculations).
  + Gain experience with bitwise operations and dynamic programming, which are key to solving complex problems with optimal time complexity.
  + They will learn various techniques to manipulate bits, including the use of shifts and logical operators, and apply these in real-world contexts, especially when handling large integers or binary representations.