



## Experiment 2

**Student Name:** Nitesh Bamber

**Branch:** BE-CSE

**Semester:** 6<sup>th</sup>

**Subject Name:** Advanced Programming - II

**UID:** 22BCS13791

**Section/Group:** 602/A

**Date of Performance:** 4-02-25

**Subject Code:** 22CSP-351

### 1. Aim:

To solve the following problems on Leetcode, with the goal of optimizing solutions in terms of time complexity and space efficiency:

#### (I) No. of 1 Bits –

*Aim:* Given a positive integer  $n$ , the objective is to return the number of set bits in its binary representation (also known as the Hamming weight). This helps in understanding bit manipulation and the efficient counting of 1-bits.

#### (II) Median of Two Sorted Arrays –

*Aim:* Given two sorted arrays `nums1` and `nums2` of size  $m$  and  $n$  respectively, the objective is to find and return the median of these two sorted arrays with an optimal time complexity of  $O(\log(m+n))$ , which involves leveraging binary search techniques.

#### (III) Sort Colors –

*Aim:* Given an array `nums` with  $n$  objects colored red, white, or blue, the goal is to sort them in-place such that objects of the same color are adjacent, in the order red, white, and blue. The problem focuses on sorting without using library sort functions and utilizing counting or the Dutch National Flag algorithm.

#### (IV) Reverse Bits –

*Aim:* Given a 32-bit unsigned integer, the task is to reverse its bits. This problem emphasizes bitwise operations and understanding the concept of bit reversal.

#### (V) Maximum Subarray –

*Aim:* Given an integer array `nums`, the objective is to find the contiguous subarray that has the largest sum, using the Kadane's algorithm to achieve an optimal  $O(n)$  time complexity.

## (VI) Search a 2D Matrix II –

*Aim:* Given an  $m \times n$  matrix where each row and column is sorted, the goal is to search for a target value in the matrix. This problem requires efficient traversal techniques to minimize time complexity, using the "top-right" search approach to find the target in  $O(m + n)$  time.

## (VII) First Bad Version –

*Aim:* Given a versioning system, the goal is to find the first bad version using a binary search approach, ensuring optimal performance with  $O(\log n)$  time complexity.

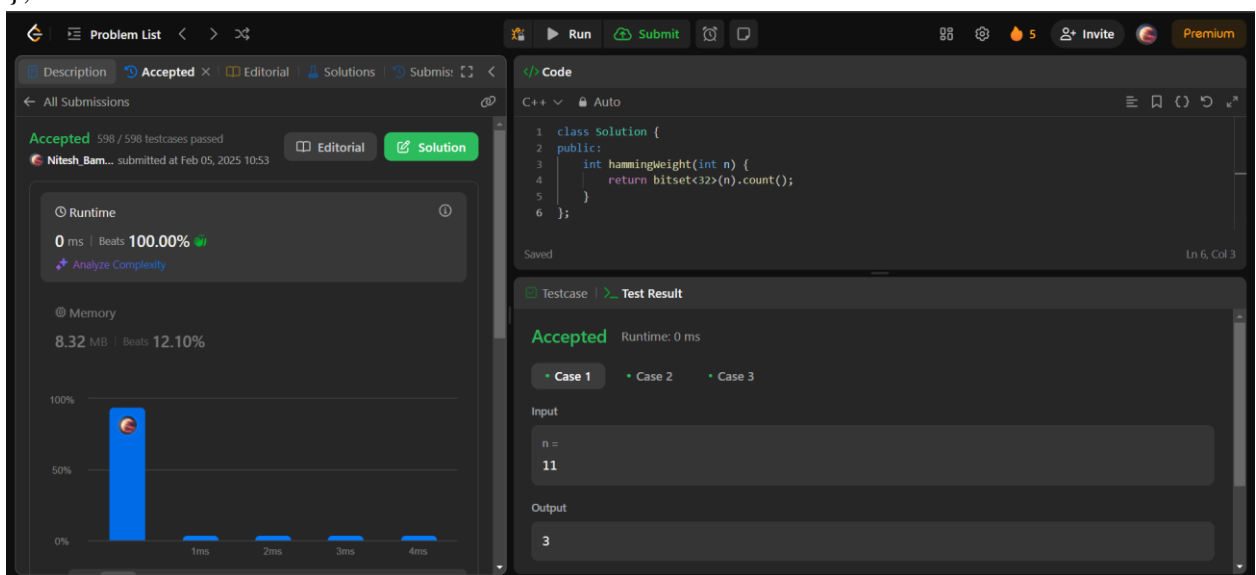
**2. Objective:** To efficiently solve a set of algorithmic problems on Leetcode, optimizing for time and space complexity, by implementing advanced techniques such as bit manipulation, binary search, sorting algorithms, and dynamic programming, with the goal of enhancing problem-solving skills and achieving optimal solutions for real-world applications.

## 3. Implementation/Code:

### *Problem No. 191 : Number of 1 Bits*

#### *Code:*

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



***Problem No. 4 : Median of Two Sorted Arrays***

***Code:***

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int m = nums1.size();
        int n = nums2.size();
        vector<int> ans;
        int i = 0, j = 0;

        while (i < m && j < n) {
            if (nums1[i] < nums2[j]) {
                ans.push_back(nums1[i++]);
            } else {
                ans.push_back(nums2[j++]);
            }
        }

        while (i < m) {
            ans.push_back(nums1[i++]);
        }

        while (j < n) {
            ans.push_back(nums2[j++]);
        }

        int totalSize = m + n;
        if (totalSize % 2 == 1) {
            return ans[totalSize / 2];
        } else {
            return (ans[totalSize / 2 - 1] + ans[totalSize / 2]) / 2.0;
        }
    }
};
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
1 class Solution {
2 public:
3     double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
4         int m = nums1.size();
5         int n = nums2.size();
6         vector<int> ans;
7         int i = 0, j = 0;
8
9         while (i < m && j < n) {
10             if (nums1[i] < nums2[j]) {
11                 ans.push_back(nums1[i++]);
12             } else {
13                 ans.push_back(nums2[j++]);
14             }
15         }
16
17         while (i < m) {
18             ans.push_back(nums1[i++]);
19         }
20
21         while (j < n) {
22             ans.push_back(nums2[j++]);
23         }
24     }
25 }
```

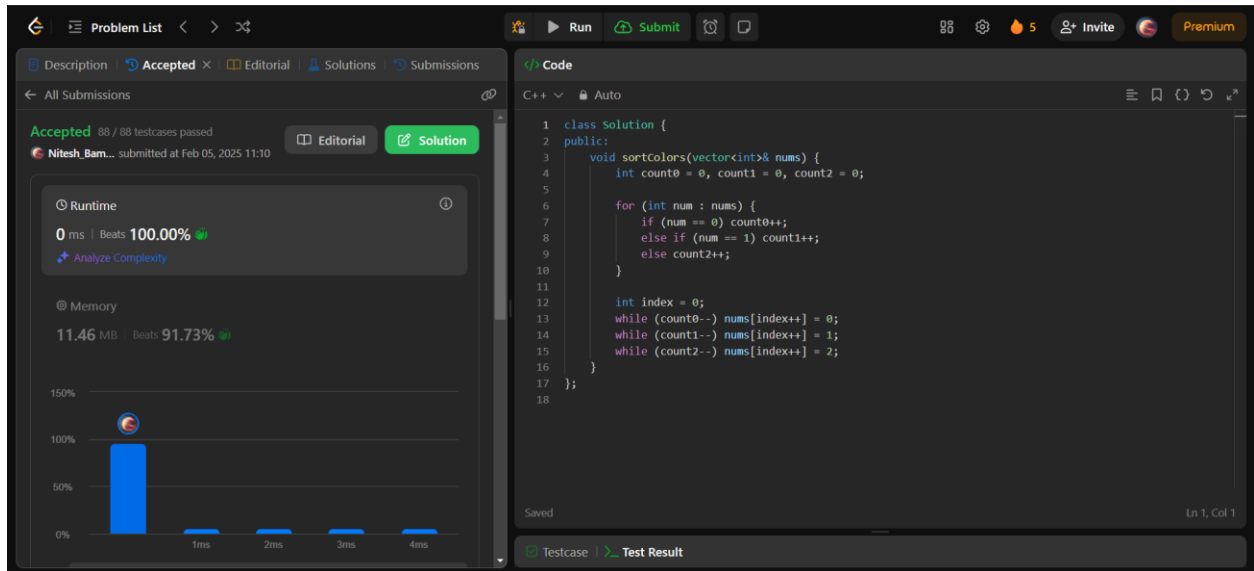
## *Problem No. 75 : Sort Colors*

**Code:**

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int count0 = 0, count1 = 0, count2 = 0;

        for (int num : nums) {
            if (num == 0) count0++;
            else if (num == 1) count1++;
            else count2++;
        }

        int index = 0;
        while (count0-- > 0) nums[index++] = 0;
        while (count1-- > 0) nums[index++] = 1;
        while (count2-- > 0) nums[index++] = 2;
    }
};
```



```

1 class Solution {
2 public:
3     void sortColors(vector<int>& nums) {
4         int count0 = 0, count1 = 0, count2 = 0;
5
6         for (int num : nums) {
7             if (num == 0) count0++;
8             else if (num == 1) count1++;
9             else count2++;
10        }
11
12        int index = 0;
13        while (count0--) nums[index++] = 0;
14        while (count1--) nums[index++] = 1;
15        while (count2--) nums[index++] = 2;
16    }
17 }
18

```

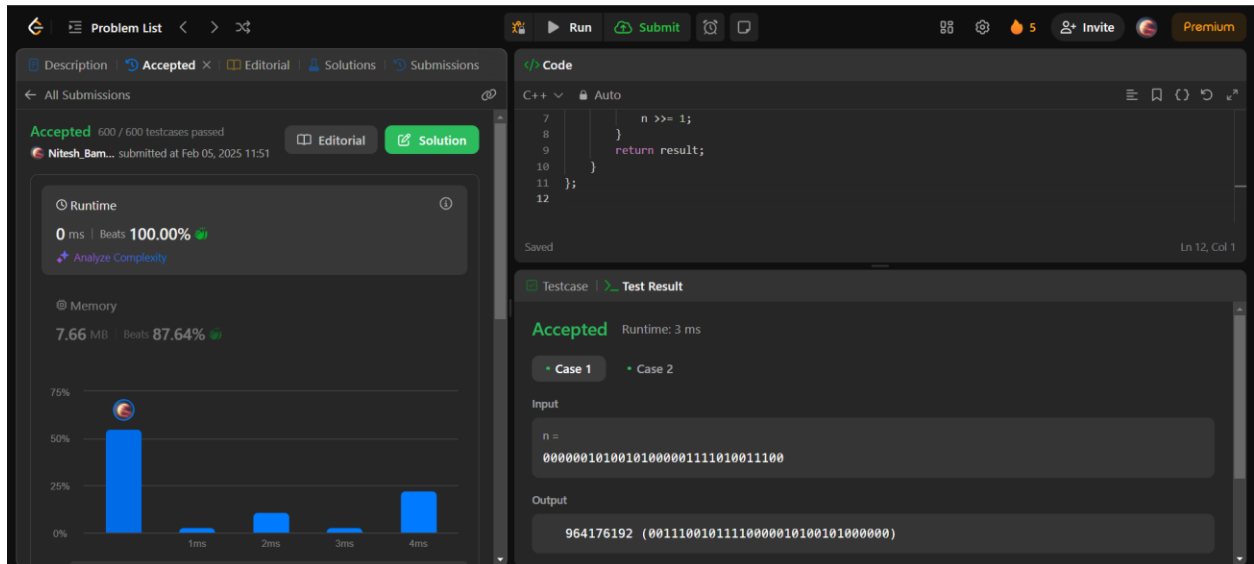
## ***Problem No. 190 : Reverse Bits***

**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;
    }
};

```



**Problem No. 53 : Maximum Subarray -**

**Code:**

```

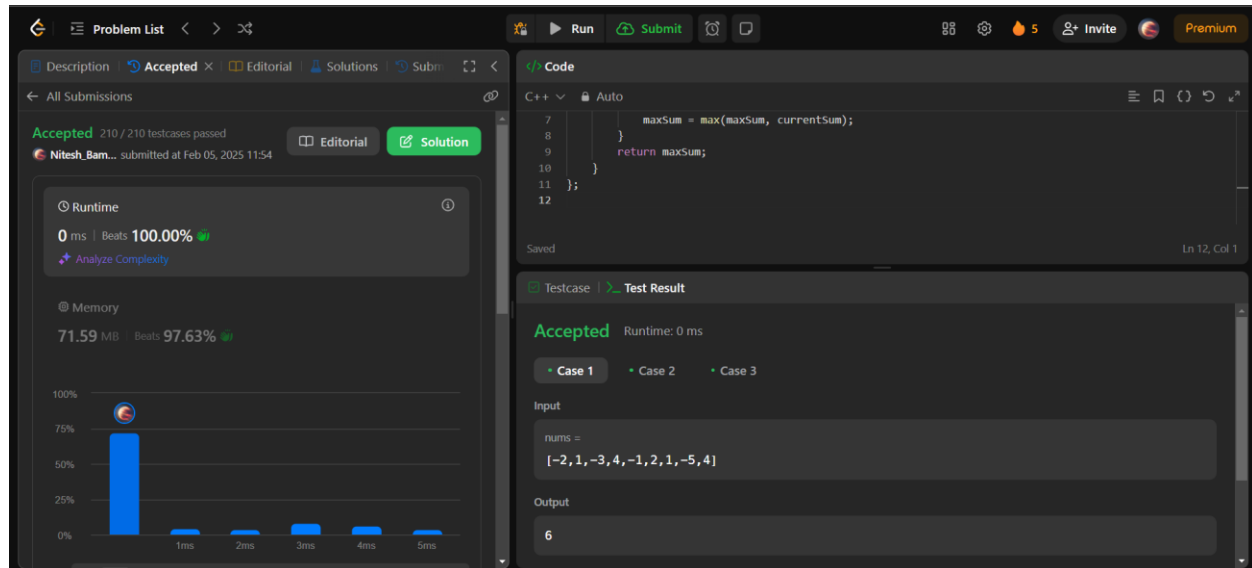
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = nums[0], currentSum = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }
        return maxSum;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

```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

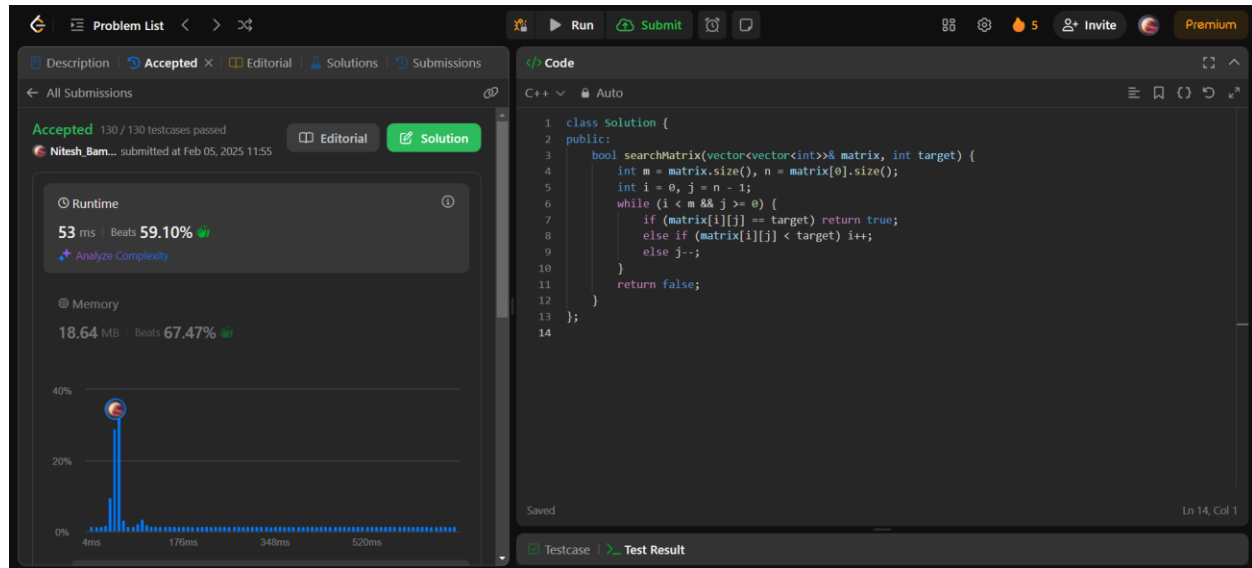
Discover. Learn. Empower.



## *Problem No. 240 : Search a 2D Matrix –*

### *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;
    }
};
```



## *Problem No. 278 First Bad Version*

**Code:**

```

class Solution {
public:
    int firstBadVersion(int n) {
        int left = 1, right = n;
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (isBadVersion(mid)) right = mid;
            else left = mid + 1;
        }
        return left;
    }
};

```





# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Problem List

Accepted

Editorial

Solutions

Submis...

All Submissions

Accepted 24 / 24 testcases passed

Nitesh\_Bam... submitted at Feb 05, 2025 11:58

Editorial

Solution

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

7.77 MB | Beats 93.11%

Time Interval	Performance (%)
1ms	~45%
2ms	~35%
3ms	~15%
4ms	~5%

Code

C++

Auto

```
1 class Solution {
2 public:
3     int firstBadVersion(int n) {
4         int left = 1, right = n;
5         while (left < right) {
6             int mid = left + (right - left) / 2;
7             if (isBadVersion(mid)) right = mid;
8             else left = mid + 1;
9         }
10        return left;
11    }
12 };
13
```

Saved

Ln 9, Col 10

Testcase

Test Result