Course Name:	**Competitive Programming Laboratory (216U01L401)**	**Semester:**	**IV**
Date of Performance:	**13 / 03 / 2025**	**DIV/ Batch No:**	A1
Student Name:	Aaryan Sharma	**Roll No:**	16010123012

Experiment No: 3

Title: To use bit manipulation to solve competitive programming problems.

Aim and Objective of the Experiment:

1. **Understand** the concepts of Bit manipulation
2. **Apply** the concepts to solve the problem
3. **Implement** the solution to given problem statement
4. **Create** test cases for testing the solution
5. **Analyze** the result for efficiency

COs to be achieved:

CO3: Solve intricate problems involving graphs, tree structures, and algorithms.

Books/ Journals/ Websites referred:

1. https://www.hackerearth.com/practice/basic-programming/bit-manipulation/basics-of-bit-manipulation/tutorial/

Theory:

Bit manipulation is a fundamental technique in computer science that involves directly operating on individual bits of binary representations. It is widely used in competitive programming, cryptography, embedded systems, and low-level optimization. Bitwise operations such as AND (&), OR (|), XOR (^), NOT (~), left shift (<<), and right shift (>>) allow efficient computations by leveraging the binary properties of numbers.

Problem statement

Given a **non-empty** array of integers nums, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:

Input: nums = [2,2,1]

Output: 1

Example 2:

Input: nums = [4,1,2,1,2]

Output: 4

Example 3:

Input: nums = [1]

Output: 1

Code:

136. Single Number

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int result = 0;
        for(int num : nums){
            result ^= num;
        }
        return result;
    }
};
```

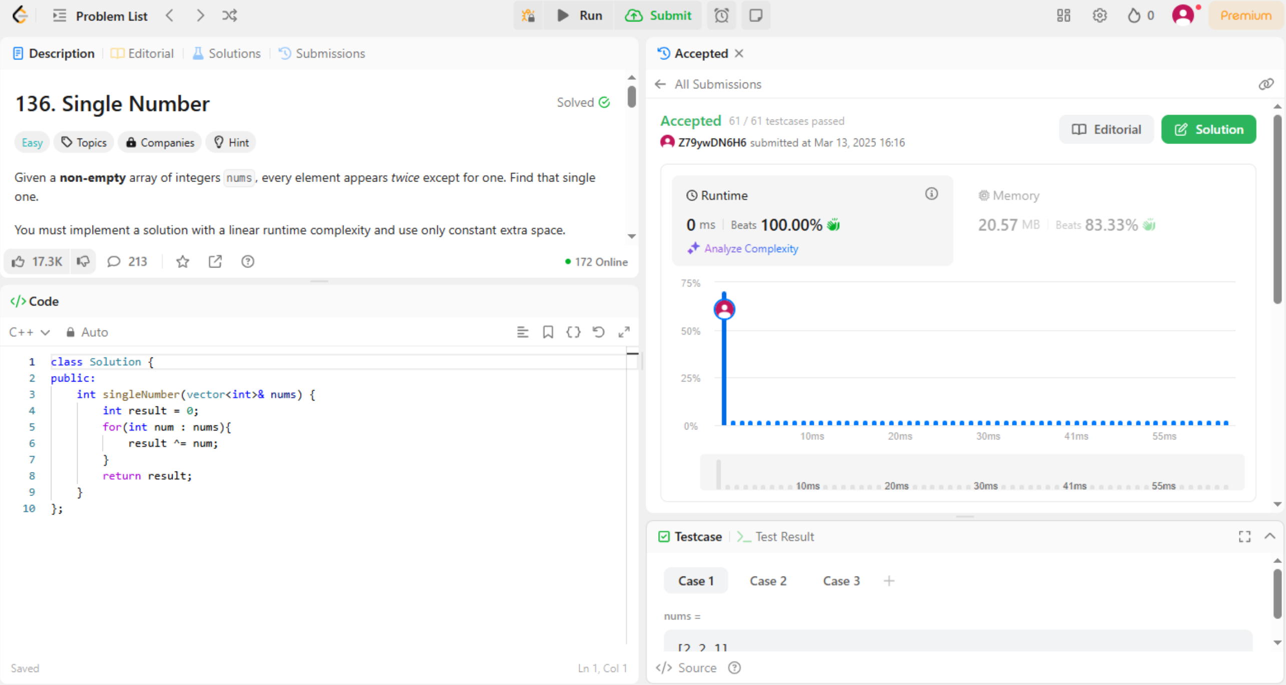
137. Single Number II

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        unordered_map<int, int> f;
        for(int num : nums){
            f[num]++;
        }
        for (auto i : f) {
            if (i.second == 1) {
                return i.first;
                break;
            }
        }
        return 0;
    }
};
```

260. Single Number III

```
class Solution {
public:
    vector<int> singleNumber(vector<int>& nums) {
        unordered_map<int, int> f;
        for(int num : nums){
            f[num]++;
        }
        vector<int> result;
        for (auto i : f) {
            if (i.second == 1) {
                result.push_back(i.first);
            }
        }
        return result;
    }
};
```

Output:



136. Single Number

Solved

Easy · Topics · Companies · Hint

Given a **non-empty** array of integers `nums`, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

17.3K · 213 · 172 Online

Code

C++ · Auto

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int result = 0;
        for(int num : nums){
            result ^= num;
        }
        return result;
    }
};
```

Saved · Ln 1, Col 1

Accepted 61 / 61 testcases passed

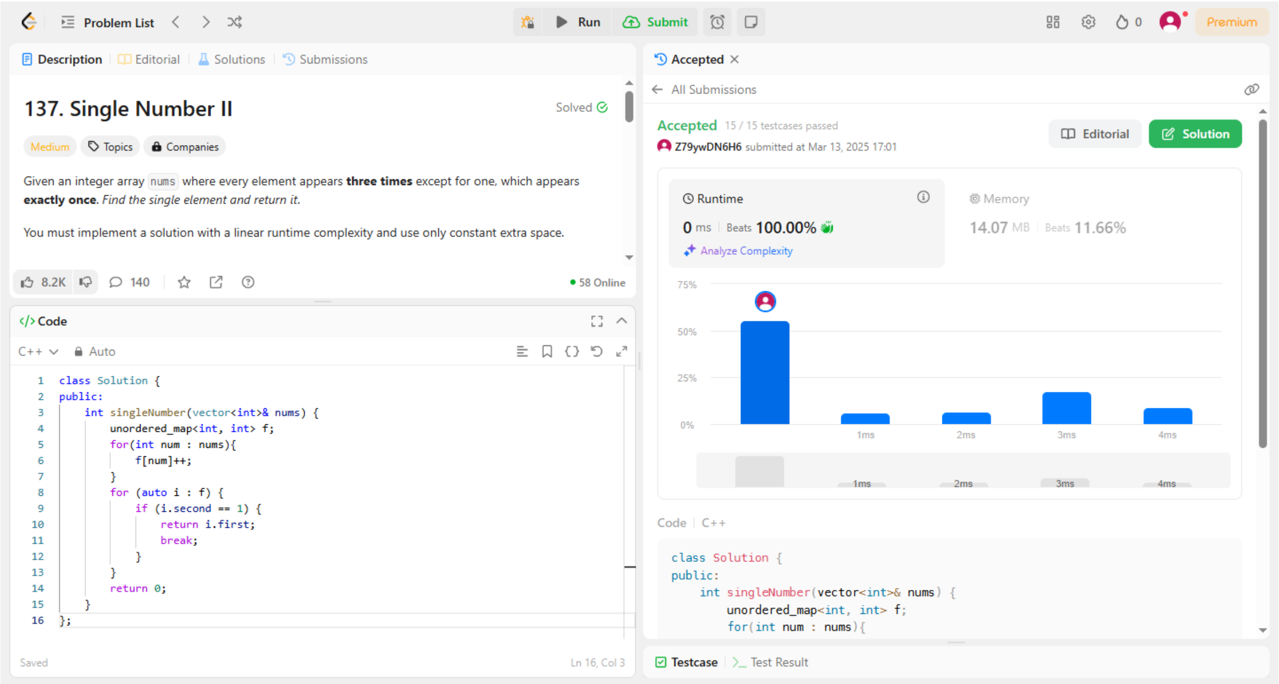
Z79ywDN6H6 submitted at Mar 13, 2025 16:16

Runtime: 0 ms | Beats 100.00%

Memory: 20.57 MB | Beats 83.33%

nums =

[2, 2, 1]



137. Single Number II

Solved

Medium · Topics · Companies

Given an integer array `nums` where every element appears **three times** except for one, which appears **exactly once**. *Find the single element and return it.*

You must implement a solution with a linear runtime complexity and use only constant extra space.

8.2K · 140 · 58 Online

Code

C++ · Auto

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        unordered_map<int, int> f;
        for(int num : nums){
            f[num]++;
        }
        for (auto i : f) {
            if (i.second == 1) {
                return i.first;
                break;
            }
        }
        return 0;
    }
};
```

Saved · Ln 16, Col 3

Accepted 15 / 15 testcases passed

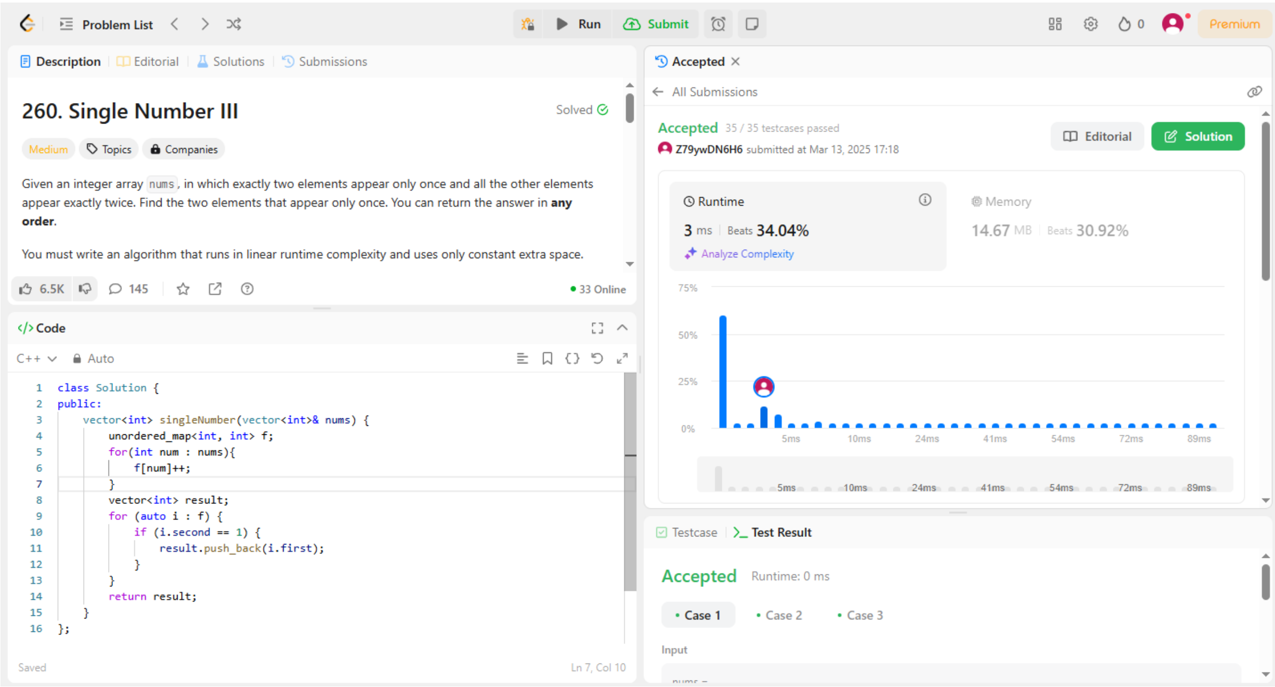
Z79ywDN6H6 submitted at Mar 13, 2025 17:01

Runtime: 0 ms | Beats 100.00%

Memory: 14.07 MB | Beats 11.66%

Code | C++

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        unordered_map<int, int> f;
        for(int num : nums){
```



260. Single Number III

Solved ✓

Medium · Topics · Companies

Given an integer array `nums`, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once. You can return the answer in **any order**.

You must write an algorithm that runs in linear runtime complexity and uses only constant extra space.

6.5K · 145 · 33 Online

Code

C++ · Auto

```cpp
class Solution {
public:
    vector<int> singleNumber(vector<int>& nums) {
        unordered_map<int, int> f;
        for(int num : nums){
            f[num]++;
        }
        vector<int> result;
        for (auto i : f) {
            if (i.second == 1) {
                result.push_back(i.first);
            }
        }
        return result;
    }
};
```

Saved · Ln 7, Col 10

Accepted 35 / 35 testcases passed

Z79ywDN6H6 submitted at Mar 13, 2025 17:18

Runtime: 3 ms | Beats 34.04%

Memory: 14.67 MB | Beats 30.92%

Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

Input

nums =

Post Lab Subjective/Objective type Questions:

1. Explore and describe any 5 scenarios where bit manipulation is extremely useful to reduce time and space complexity of an algorithm.
    - Finding a Unique Element in an Array (XOR Technique): In an array where every number appears twice except one, XOR can be used to find the unique element in O(n) time with O(1) space. Since x ^ x = 0 and x ^ 0 = x, XORing all elements cancels out duplicates, leaving only the unique number.
    - Checking if a Number is a Power of Two: A number is a power of two if it has exactly one 1 bit in its binary representation. This can be checked in O(1) time using n & (n - 1), which clears the lowest set bit.
    - Swapping Two Numbers Without Using Extra Space: Using XOR, two numbers can be swapped without a temporary variable, saving memory and improving efficiency.
    - Counting the Number of Set Bits (Hamming Weight): Instead of iterating through all bits, we can use n & (n - 1) to turn off the lowest set bit in O(k) time, where k is the number of set bits.
    - Generating All Subsets of a Set (Bitmasking): A set of n elements has 2^n subsets. Using bit manipulation, each subset can be represented by an n-bit number where 1 at position i means the i-th element is included. This approach efficiently generates all subsets in O(2^n * n) time.

Conclusion:

I have implemented bit manipulation techniques to solve problems efficiently with minimal memory usage. The use of bitwise XOR to find unique numbers demonstrated its power in reducing both time and space complexity.