Course Name:	**Competitive Programming Laboratory (216U01L401)**	**Semester:**	**IV**
Date of Performance:	**23 / 01 / 2025**	**DIV/ Batch No:**	**E-2**
Student Name:	**Shreyans Tatiya**	**Roll No:**	**16010123325**

Experiment No: 0.1

Title: To interpret the given problem statement and identify test cases for the given problem statement

Aim and Objective of the Experiment:

- Understand the problem statement
- Design Test cases

COs to be achieved:

CO1: Applying various problem-solving paradigms, enabling them to create and implement efficient algorithms for real-world challenges.

Books/ Journals/ Websites referred:

1. https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_majority_vote_algorithm

Theory:

The majority element problem is related to frequency analysis, where we count how often elements appear in an array. It's also tied to voting algorithms, like the Boyer-Moore Voting Algorithm, which efficiently determines the majority by maintaining a candidate and a count. The problem involves basic concepts of divisibility (more than ⌊n / 2⌋ times) and set theory, where the majority element is the most frequent element in a set. This problem showcases counting and optimization techniques in algorithm design. It also emphasizes the concept of probabilistic algorithms, especially when approximating the majority in uncertain data. It serves as a basic example of decision-making algorithms where a final choice is made based on the dominance of one option. Moreover, the majority element problem is often used to introduce concepts of amortized analysis, demonstrating how multiple steps can be combined to optimize performance.

Problem statement

Given an array nums of size n, return the majority element. The majority element is the element that appears more than ⌊n / 2⌋ times.

Code :

```
class Solution {

    public int majorityElement(int[] nums) {
        int el=0;
        int cnt=0;

        for(int i=0;i<nums.length;i++){
            if(cnt==0){
                el=nums[i];
                cnt++;
            }
            else if(nums[i]!=el){
                cnt--;
            }
            else{
                cnt++;
            }
        }
        return el;
    }
}
```

Test cases:

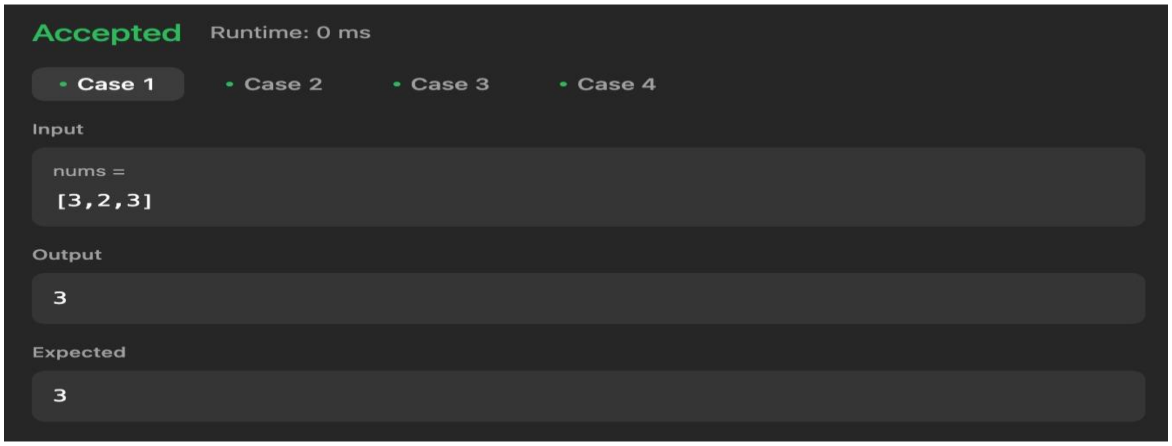
Case 1
[3,2,3]

Case 2
[2,2,1,1,1,2,2]

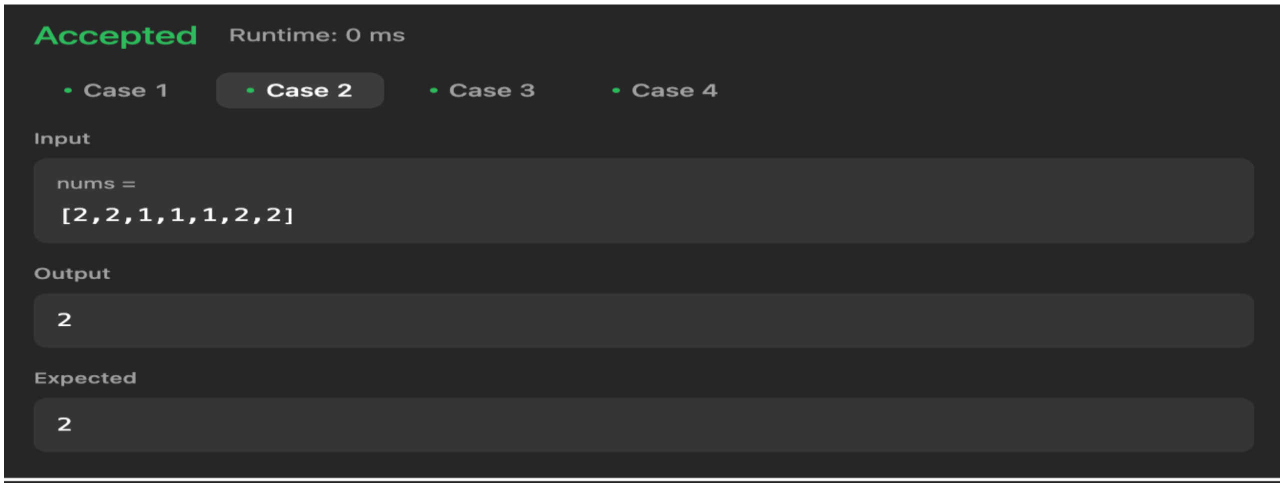
Case 3
[3,3,4]

Case 4
[1]

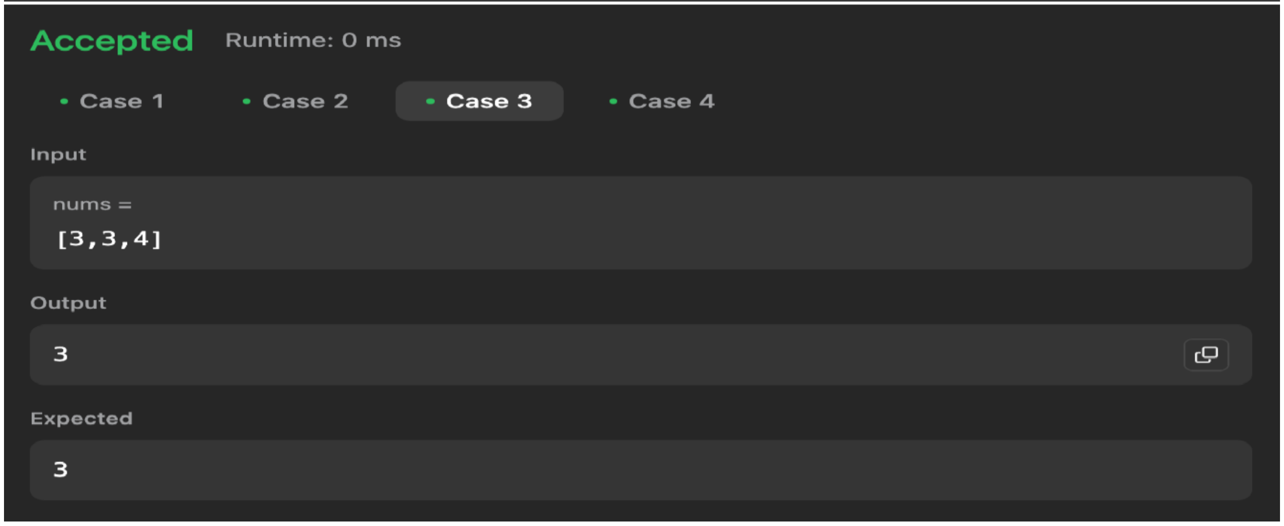
Output:



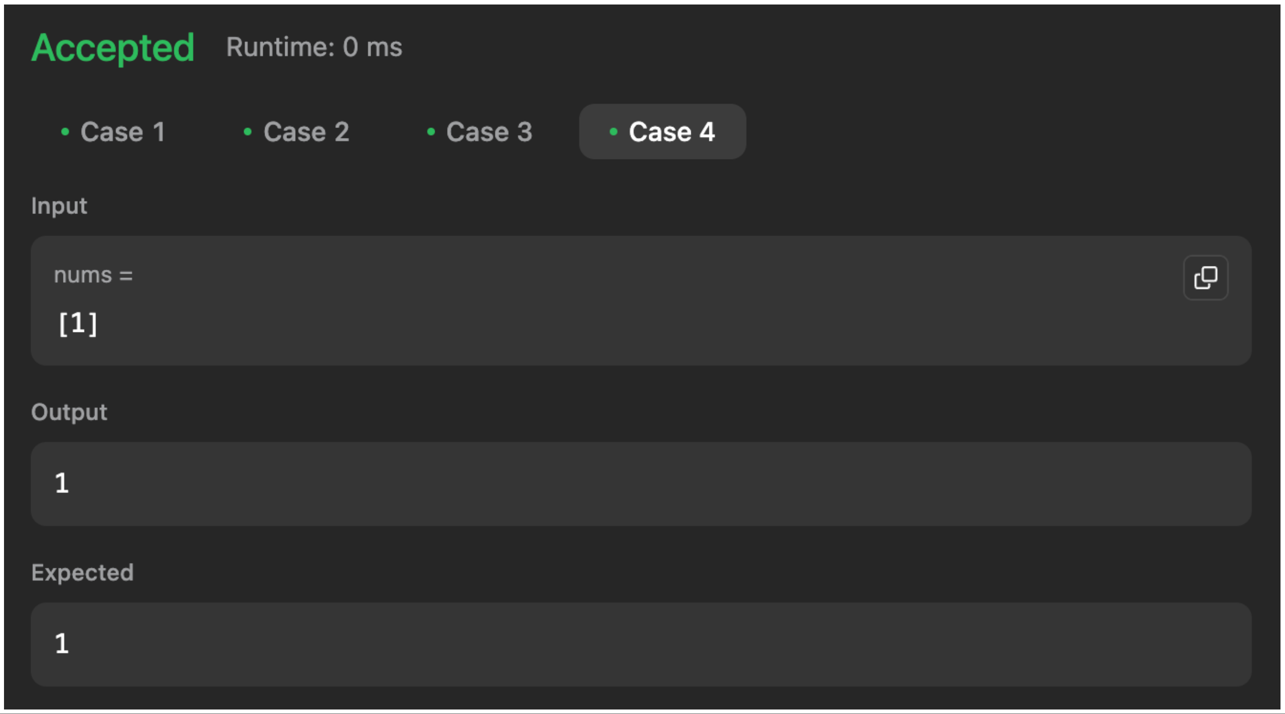
```
Accepted   Runtime: 0 ms
Case 1   Case 2   Case 3   Case 4
Input
nums =
[3,2,3]
Output
3
Expected
3
```



```
Accepted   Runtime: 0 ms
Case 1   Case 2   Case 3   Case 4
Input
nums =
[2,2,1,1,1,2,2]
Output
2
Expected
2
```



```
Accepted   Runtime: 0 ms
Case 1   Case 2   Case 3   Case 4
Input
nums =
[3,3,4]
Output
3
Expected
3
```



Input

```
nums =
[1]
```

Output

```
1
```

Expected

```
1
```

Conclusion:

The above algorithm highlights use Moore’s voting algorithm to find the majority element, since it is the most optimized technique solving the problem in O(n) time complexity and O(1) space complexity.