⚡ Problem Summary Leetcode: 169

We are given an array `nums` of size `n`.

- **Majority element** = The element that appears **more than ⌊n/2⌋ times**.
- We are guaranteed that such an element always exists.

Example:

- `nums = [3,2,3]` → Majority = 3
- `nums = [2,2,1,1,1,2,2]` → Majority = 2

🔍 Code You Wrote

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        return nums[((nums.size())/2)];
    }
};
```

🧠 Logic of This Code

1. **Sorting Step**:
    - First, we sort the array.
    - Example: `[2,2,1,1,1,2,2]` → After sorting → `[1,1,1,2,2,2,2]`.
2. **Middle Element Trick**:
    - Since the majority element appears more than `n/2` times, after sorting, **it will always occupy the middle index**.

    - Example: `n = 7`, so `n/2 = 3`. The element at index `3` (0-based indexing) is guaranteed to be the majority.
    - Sorted array: `[1,1,1,2,2,2,2]` → element at index `3` = `2`.

3. **Return It**:
    - Just return `nums[n/2]`.

✅ That's why this works perfectly.

⏱️ Time & Space Complexity

- Sorting takes **O(n log n)** time.
- Space complexity depends on sorting algorithm (in C++ STL sort = O(log n) stack space).
- Not the best for the **follow-up** (which asks O(n) time and O(1) space).

⚡ Follow-up: Optimal Solution (Moore's Voting Algorithm)

There is a better approach called **Moore's Voting Algorithm**:

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int candidate = 0, count = 0;
        for(int num : nums){
            if(count == 0) candidate = num;
            count += (num == candidate) ? 1 : -1;
        }
        return candidate;
    }
};
```

🧠 Logic of Moore's Algorithm

- If we take pairs of **different elements** and cancel them out, the **majority element** will always remain in the end.
- Works in **O(n) time** and **O(1) space**.

📝 Explanatory Notes (for revision)

- **Majority element** = appears more than `n/2` times.
- **Sorting Trick**: Majority always at middle index after sorting → `nums[n/2]`. (O(n log n))
- **Optimal Method (Moore's Voting):** Cancel out different pairs → candidate left is majority. (O(n), O(1))
- **Guarantee**: Majority element always exists in given array (so no need for extra check).