Given an array nums of size n, return *the majority element*.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Example 1:

Input: nums = [3,2,3]
Output: 3

Example 2:

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

Constraints:

- **n == nums.length**
- **1 <= n <= $5 * 10^4$**
- **-10^9 <= nums[i] <= 10^9**

Follow-up: Could you solve the problem in linear time and in O(1) space?

Solution:

```
class Solution {
    public int majorityElement(int[] nums) {
        HashMap<Integer,Integer> n=new HashMap<Integer,Integer>();
        for(int i=0;i<nums.length;i++){
            if(n.containsKey(nums[i])){
                int count=n.get(nums[i]);
                if((count+1)>(nums.length/2)){
                    return nums[i];
                }
                n.put(nums[i],count+1);
            }else{
                n.put(nums[i],1);
            }
        }
        return nums[0];
    }
}
```