Majority Element

Question:

https://leetcode.com/problems/majority-element/

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
```

My approach:

Since we just need to count the occurrences of a number, I used a hashmap to map the elements with the number of occurrences of that element.

My Solution:

Code: https://gist.github.com/vermaayush680/f8edcd4d1c7a767b38561e34e40ae38a

```
d={}
for i in nums:
    d[i]=d.get(i,0)+1
    if d[i]>len(nums)//2:
        return i
```

Time Complexity: O(n)
Space Complexity: O(n)

Looking at the discussions forum, I found 2 more unique solutions.

Solution 1:

Basically, Sort the array/list and the element at (length of array/2) will be the majority element. The reason for that is that the majority element will be at least (length of array/2)+1 times in the array hence post-sorting, the middle element will always be the majority element.

Code: https://gist.github.com/vermaayush680/9cfdc07592b1df3892574038892ac605

```
nums.sort()
return nums[len(nums)//2]
```

We can use Merge Sort for faster sorting.

Time Complexity: O(nlogn)
Space Complexity: O(1)

Solution 2:

This was the best thing I found through the discussions forum.
Boyer-Moore Majority Voting Algorithm is used to find the majority element having more than n/2 occurrences.

Explanation: https://www.geeksforgeeks.org/boyer-moore-majority-voting-algorithm/

The reason this algorithm caught my eye was that it solves the same question using constant space and linear time.

Code: https://gist.github.com/vermaayush680/edd7d34c5dcacd259376f793f99f7ce1

```
c=0
can=0
for i in nums:
    if c==0:
        can = i
    if i==can:
        c+=1
    else:
        c-=1
```

return can

Time Complexity: O(n)
Space Complexity: O(1)