Problem Link:

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

Problem Description:

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.

Problem Approach:

We can count the frequency of elements in the array, and then, return the element that appears more than n/2 times. But that's not the optimal approach, and may fail when the input is too big. So, we follow a famous algorithm called: "Boyer-Moore Voting Algorithm". The algorithm maintains a count of a potential majority candidate and adjusts that count based on how frequently that candidate appears in relation to other elements in the array.

Sample Test Case:

Input: nums = [3,2,3]

Output: 3

Solution:

We initialize two variables namely candidate and count with None and 0 respectively, since before iteration, there's no candidate and hence, the 0 count. Then, we iterate through the array, and check if the count is 0. If that's the case, then the current element will become the element, and if not, then, we'll check if the current element is the same as the candidate element. If yes, then we increase the count by 1, and if not, then we decrease the count by 1. By the end of iteration of the entire array, we would have the majority element as the candidate. If an element appears more than half the time, it will survive these cancellations because, eventually, it will dominate the count.

The time complexity of this algorithm is O(n), and the space complexity is O(1).

Another approach to this problem can be to sort the array and return the middle element. But that'll take O(n log(n)) time complexity, hence, isn't as efficient as the Boyer-Moore Voting Algorithm.

Code (Python):

```
def majorityElement(self, nums: List[int]) -> int:
      candidate = None
      count = 0
      for num in nums:
       if count == 0:
           candidate = num
       else:
           if num == candidate:
               count+=1
           else: count -=1
       return candidate
```