Yunyeong Kim

Question

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`

Definition

1. Majority element always exist in the array.
2. Appears more than **⌊n / 2⌋** times
    1. If size = 5 / it should appear more than 5/2 times → 2
    2. **⌊ ⌋** represents **바닥 함수(Floor function)**
3. Only one element should be returned.

First Code : 10 min

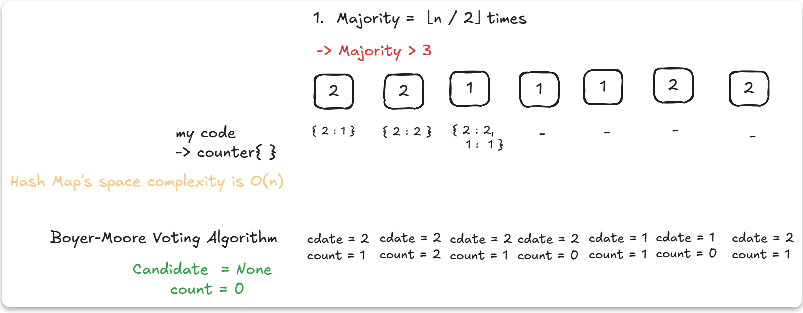
```
class Solution(object):
    def majorityElement(self, nums):
        flag = int(len(nums)/2)
        counter = {}
        for num in nums:
            if num in counter:
                counter[num] += 1
            else:
                counter[num] = 1
            if counter[num] > flag:
                return num
        return None
# O(n) / O(n)
```

Solution

```
# Boyer-Moore Voting Algorithm
class Solution(object):
    def majorityElement(self, nums):
        count=0
        number=None
        for num in nums:
            if count==0:
                number=num
            if num==number:
                count+=1
            else:
                count-=1
        return number
# O(n) /  O(1) space complextiсy
```

Solution Explain

- Since the "majority element" is guaranteed to exist,
  there is no need to check when an element surpasses the majority threshold during the array traversal.
  For example, the Boyer-Moore Voting Algorithm also returns the candidate without verifying if it actually exceeds the majority count.


1. Majority = ⌊n / 2⌋ times
-> Majority > 3
2
2
1
1
1
2
2
my code
-> counter{ }
{ 2 : 1 }
{ 2 : 2 }
{ 2 : 2,
1 : 1 }
-
-
-
-
Hash Map's space complexity is O(n)
Boyer-Moore Voting Algorithm
Candidate = None
count = 0
cdate = 2
count = 1
cdate = 2
count = 2
cdate = 2
count = 1
cdate = 2
count = 0
cdate = 1
count = 1
cdate = 1
count = 0
cdate = 2
count = 1