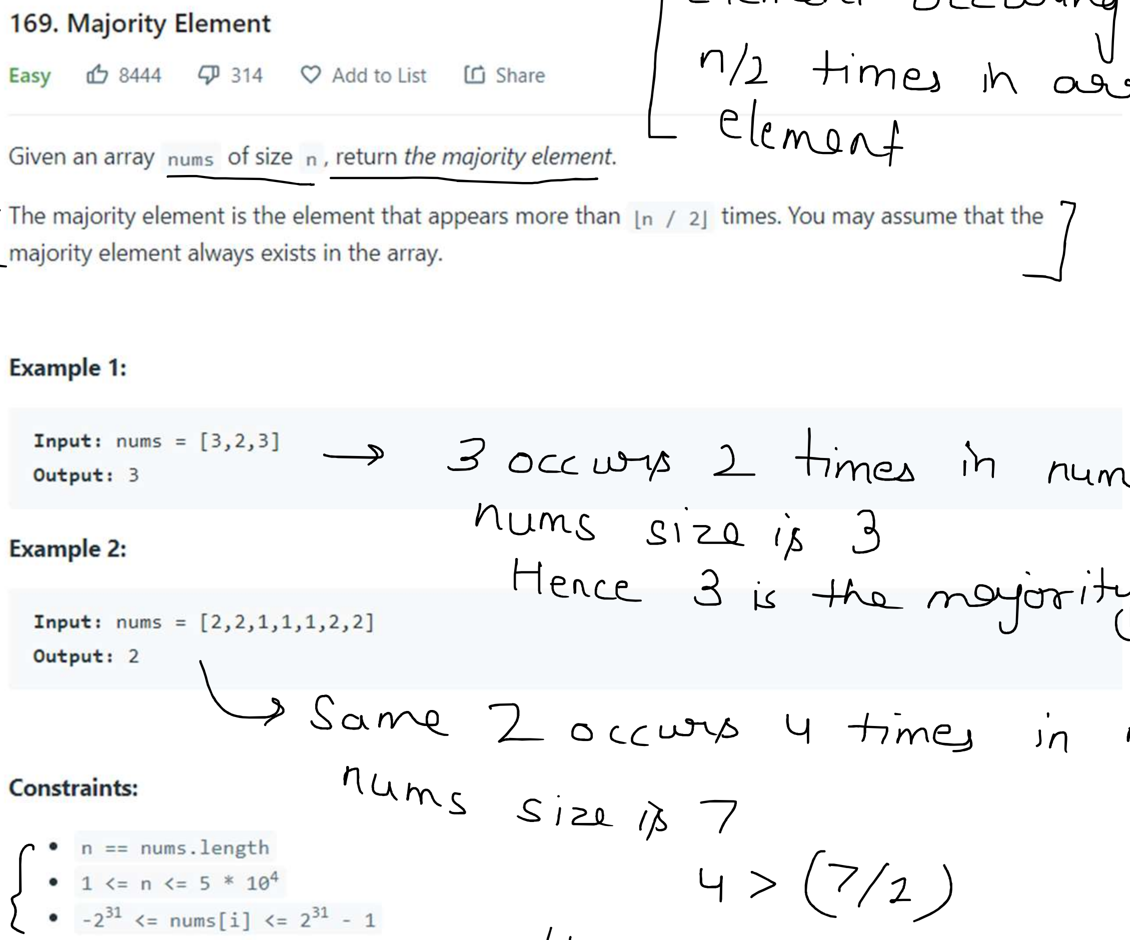
169. Majority Element

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

February 21, 2022

-Priyanshu Arya



169. Majority Element

Easy 8444 314 Add to List Share

Element occuring more than n/2 times in array is 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
```

→ 3 occurs 2 times in nums
nums size is 3
Hence 3 is the majority element

Example 2:

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

→ Same 2 occurs 4 times in nums
nums size is 7
$4 > (7/2)$
Hence 2 is majority element

Constraints:

- `n == nums.length`
- `1 <= n <= 5 * 10^4`
- `-2^31 <= nums[i] <= 2^31 - 1`

If we follow constraints So our solution can be of size $<= O(n^2)$

Approach: 1 Sort the array and return element at n/2 place.

For Ex : [2, 2, 1, 2, 1, 2]

After Sorting [1, 1, 2, 2, 2, 2]

0 1 2 3 4 5

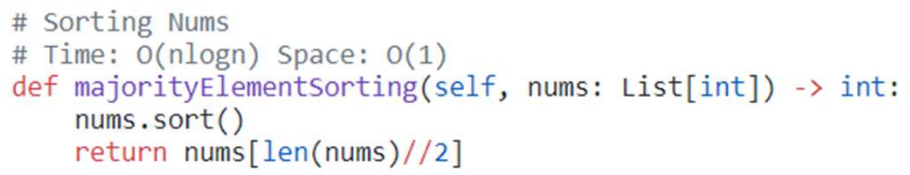
$\uparrow$

(5/2 = 3)

2 is the majority element

Intution: If we sort the array then automatically the majority element occur the middle position in the array

Algorithm:



```python
# Sorting Nums
# Time: O(nlogn) Space: O(1)
def majorityElementSorting(self, nums: List[int]) -> int:
    nums.sort()
    return nums[len(nums)//2]
```

```
majority ele (nums):
    nums.sort()    ← O(n logn)
    n = len(nums)
    return nums[n//2]
```

Time Complexity: $O(n \log n)$

Space Complexity: $O(1)$

Approach 2 : Counting frequency method

Intution: Count frequency of all elements and return the element which have maximum frequency

For Ex: [2, 2, 1, 2, 1, 3, 3]

Ele	Freq
2	3
1	2
3	2

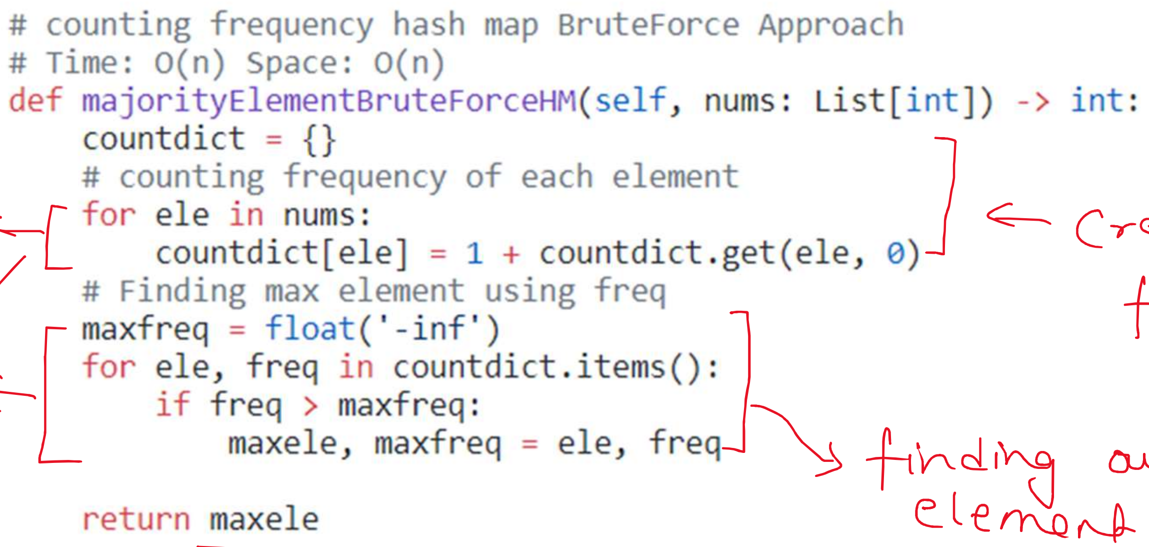
Majority element

Time Complexity : O(n)

for traversing through List

Space Complexity : O(n)

For Storing frequency of all Element



```python
# counting frequency hash map BruteForce Approach
# Time: O(n) Space: O(n)
def majorityElementBruteForceHM(self, nums: List[int]) -> int:
    countdict = {}
    # counting frequency of each element
    for ele in nums:
        countdict[ele] = 1 + countdict.get(ele, 0)
    # Finding max element using freq
    maxfreq = float('-inf')
    for ele, freq in countdict.items():
        if freq > maxfreq:
            maxele, maxfreq = ele, freq

    return maxele
```

Space O(n)

Time O(n)

← Creating Dictionary & Counting frequency

→ finding out maximum frequency element

↳ Returning Element which have maximum frequency

Approach: 3 Moore Voting Algorithm

This algorithm works on the fact that if an element occurs more than N/2 times, it means that the remaining elements other than this would definitely be less than N/2.

For Ex: [2, 2, 2, 1, 1, 1, 2, 3]

$$\boxed{\begin{array}{ll} \text{Time}: & O(n) \\ \text{Space}: & O(1) \end{array}}$$

	Starting								
Item	None	2	2	2	1	1	1	2	3
Frequency	0	1	2	3	2	1	0	1	0
Candidate	None	2	2	2	2	2	2	2	$\boxed{2} \rightarrow$ Answer

For deep understanding ↓

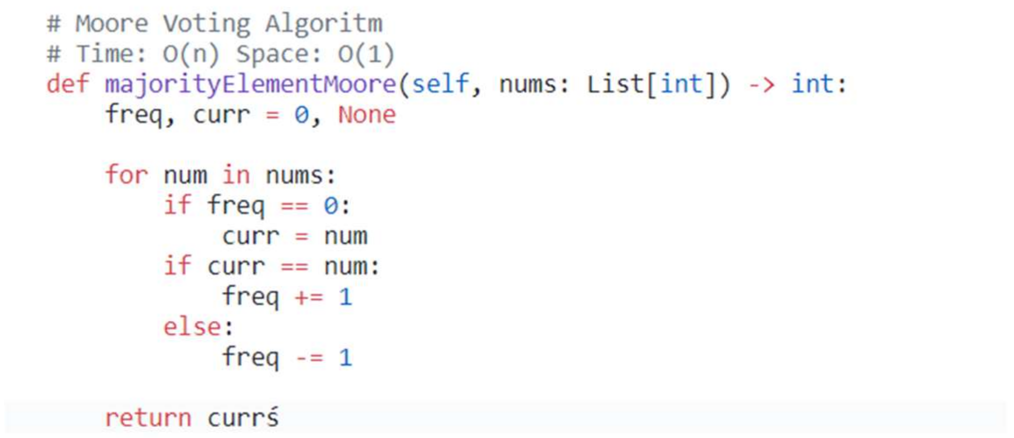
https://www.geeksforgeeks.org/boyer-moore-majority-voting-algorithm/

Algorithm

```
moore voting (nums):
    freq, candi = 0, None

    for num in nums:
        if freq == 0:
            candi = num
        if candi == num:
            freq += 1
        else
            freq -= 1
    return candidate
```

← O(n) Time



```python
# Moore Voting Algoritm
# Time: O(n) Space: O(1)
def majorityElementMoore(self, nums: List[int]) -> int:
    freq, curr = 0, None

    for num in nums:
        if freq == 0:
            curr = num
        if curr == num:
            freq += 1
        else:
            freq -= 1

    return curr
```

This Problem can Solve by other approaches also:

For Ex: — Bit Manipulation

Divide and Conquer

Randomization

Try all approaches If you have any doubt or stuck in any approaches you can ping me.

Thank you

If you like Please share this and feel free to connect for any queries.

Discord: https://discord.gg/qPer56TP

Mail: priyanshuarya2482000@gmail.com