Majority Element

0 1 2 3 4 5 6

nums = [2, 2, 1, 1, 1, 2, 2]

Output = 2

n = 7

Majority Element

frequency > n/2

Problem statement is that we need to find the element that is in majority based on the below condition: frequency > n/2

2 : 4 > 7/2 = (3) → Taking the floor value for the division 7/2

Frequency of 2 is 4 that is greater than n/2 or 7/2. Therefore, 2 will be treated as the majority element.

Approach 1 : Sort the array — nlogn

↳ Θ(nlogn) (1, 1, 1, 2, 2, 2, 2)

↳ 2

Sorting followed by calculating the frequency of unique elements which in best case scenario will take around nlogn Time complexity

Approach 2 : Hash Data Structure (Dictionary)

(key, value)

In Hash data structure we store the element in the Key and the Value manner. In python DICTIONARY is the data structure that can be used to replicate the same.

Output = 2

2 : 4 *

1 : 3

Here we as storing element of the list in the form of KEY whereas, frequency of that element in the form of VALUE

TC = O(n)

Here we will be traversing through the list once so that's why we TC is O(n).

SC = O(n)

Since, here we are using Dictionary on top of available list of elements. So, that's the reason why space complexity is O(n).

Hashtable

Internally dictionary maintains below kind of HASH TABLE

key	value
2	4
1	3

2

See the ipynb implementation notebook for the Pythonic implementation

Approach 3: Boyer-Moore Voting Algorithm

This is the most optimal approach to solve this problem with TC=O(n) and SC=O(1)

{ TC → O(n)
{ SC → O(1)

2 ⟷ 4 > 7/2 (3)

↳ Yes

Example 1

[2, 2, 1, 1, 1, 2, 2]

Take two variable Candidate representing the element in the list and count representing the frequency of the respective Candidate/element. Initially, Candidate will be assigned with None whereas, count will be assigned with 0.

candidate = ~~None~~ ~~2~~ ~~1~~ 2

count = ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~0~~ 1

Flow control

This will always run for the each element

1. count == 0
   ↳ candidate = num

2. num == candidate
   ↳ count += 1

One of these will always run for the each element

3. else:

Flow control see example 2 below for better understanding

count -= 1

Example 2

n = 5

n/2 = 5/2 = 2

0 1 2 3 4

[2, 3, 4, 3, 3]

output = 3 ⟷ 3 > 2 (n/2)

At the end the element that is captured inside Candidate variable after the complete traversal through the list will be the one representing the Majority element if it's respective frequency is greater than the number obtained by dividing the number of elements in list by 2.
i.e. frequency > n/2

candidate = ~~None~~ ~~2~~ 3 ~~3~~ ~~4~~

count = ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ 2 ~~0~~ ~~1~~

3 ⟷ 3 > n/2

↳ Yes

Example 3

0 1 2 3 4

(2, 3, 7, 3, 4)

$n = 5$

$n/2 = 5/2 = 2$

$3 \leftrightarrow 2 \not> n/2$

↳ No majority element

~~2~~ ~~3~~ ~~7~~ ~~3~~ 4

candidate = ~~None~~

count = ~~0~~ ~~1~~ ~~0~~ ~~1~~

~~0~~ ~~1~~ ~~0~~

1 ~~0~~ ~~1~~

$4 \leftrightarrow 1 \not> n/2$

↳

No majority element

Please note that here we have 2 loops which are not nested. That why TC is n+n=2n or order of O(n).

In case we were using nested loops then in that case TC would be n*n that is order of O(n^2)