12. Majority Element

169. Majority Element

Easy 14K 433

Companies

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
```

Old Video

7 7 5 7 5 1 5 7 5 5 7 7 5 5 5 5

Brute force

→ Pick one element & count the occurence in the arr and if it maximum than floor of the array.

T.C ⇒ $O(n^2)$ S.C ⇒ $O(1)$

Optimised Approach Using HashMap

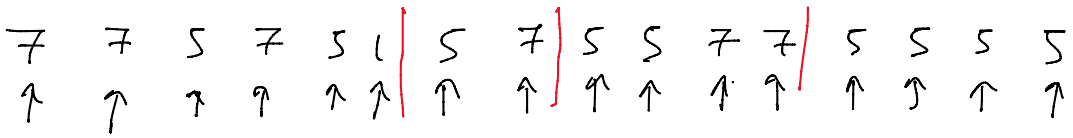
7 7 5 7 5 1 5 7 5 5 7 7 5 5 5 5

T.C ⇒ $O(N)$ or $O(N \log N)$

(1,
(5,
(7,

(element, times)

Moore Voting Algorithm



→ Initialise takes cnt = 0, ele = 0

→ Linear iterate

```
if (cnt == 0)
{
    ele = a[i];
}
if (ele == a[i])
    cnt += 1
else
    cnt -= 1
```

cnt = 0 x 1 + 1 x 0
+ 0 + 1 x 0 + 1 x 2 3 4

ele = 0 7 5 5 (5) → o/p

7 7 5 7 5 1 | 5 7 | 5 5 7 7 | 5 5 5 5

Intitution

Moore's Voting Algorithm is a simple algorithm used to find the majority element in an array or sequence. The majority element is the element that appears more than ⌊n/2⌋ times in the array, where n is the length of the array.

The intuition behind Moore's Voting Algorithm is to use a counter variable and a candidate variable. The algorithm iterates through the array, incrementing the counter if the current element matches the candidate, and decrementing the counter if the current element does not match the candidate. If the counter reaches zero, the algorithm chooses the next element in the array as the new candidate. By the end of the iteration, the candidate variable will contain the majority element.

The algorithm works because the majority element will always have a count that is greater than all other elements combined. Therefore, even if the algorithm encounters some non-majority elements that decrement the counter, the majority element will still be left with a positive count at the end of the iteration.

The algorithm has a time complexity of O(n) and a space complexity of O(1), making it a very efficient solution for finding the majority element in an array or sequence.

```cpp
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int count = 0;
        int candidate = 0;
        for(int num: nums){
            if(count == 0){ // Check if 'count' is 0
                candidate = num; // If 'count' is 0, set the 'candidate' variable to the current 'num'
            }
            if(num == candidate) count += 1; // Check if the current 'num' is equal to 'candidate', if so, increment 'count'
            else count -= 1; // If the current 'num' is not equal to 'candidate', decrement 'count'
        }
        return candidate;
    }
};
```

New Video

Majority Elements ($> n/2$ times)

arr[] = [2 2 3 3 1 2 2]

N = 7

7/2 = 3

N/2

2 ⇒ 4 > 3

output

Brute force

→ Pick up an element and scan through entire array and cnt ++.

→ if cnt $\geq n/2$ return.

```
for(i=0 ⟶ i++)
{ cnt=0
  for(i=0 ⟶ j++)
  { if(arr[j] == arr[i])
        cnt++
  }
  if(cnt > n/2) return arr[i];
}
```

T.C ⇒ $O(n^2)$

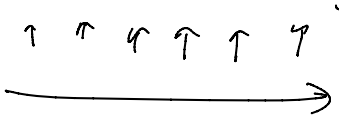
S.C ⇒ $O(1)$

Better Approach Using HashMap :

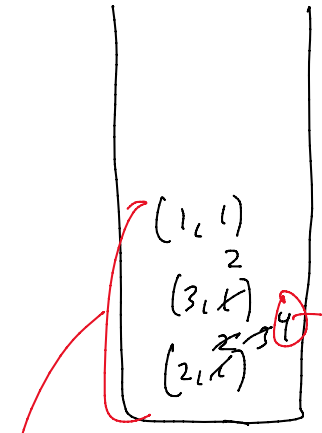
arr[] = [2 2 3 3 1 2 2]

↑ ↑ ↑ ↑ ↑ ↑ ↑

arr[] = [2 2 3 3 1 2 2]



* Create a hashmap with element and value
* Iterate over the array and increase the cnt value
* Now iterate over the map, you will see '2' has maximum cnt > n/2.




```cpp
#include <bits/stdc++.h>
int majorityElement(vector<int> v) {
    map<int, int>mpp;
    for(int i=0; i<v.size();i++){
        mpp[v[i]]++;
    }
    // iterate the map
    for(auto it: mpp){
        // second because that is the value (first is key)
        if(it.second > (v.size() /2)){
            return it.first;
        }
    }
    return -1;
}
```

T.C ⇒ O(N log N) + O(N)

S.C ⇒ O(N)

Moore's Voting Algorithm :

arr[] = { 7 7 5 7 5 1 5 7 5 5 7 7 5 5 5 5 }

el = 7

cnt = 0 1 2 1 2 1 0

You reach '0' so '7' is not majority element for now.

So count '0' so new majority element is 5

el = 5

cnt = 1 0

Now again its '0' so '5' is not majority element

el = 5

cnt = 1 2 1 0

Now again its '0' so '5' is not majority element

el = 5

cnt = 0 1 2 3 4

Since the iteration is over, but the cnt has no significant and majority element = 5 as this didn't get cancel out.

as 5 appear ⇒ 'q' times in the array.
which is greater than n/2.

Approach

1. Apply Moore's Voting Algorithm
2. Verify if element appear ≥ n/2.

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int count = 0;
        int candidate = 0;
        for(int num: nums){
            if(count == 0){ // Check if 'count' is 0
                candidate = num; // If 'count' is 0, set the 'candidate' variable to the current 'num'
            }
            if(num == candidate) count += 1; // Check if the current 'num' is equal to 'candidate', if so, increment 'count'
            else count -= 1; // If the current 'num' is not equal to 'candidate', decrement 'count'
        }
        return candidate;
    }
};
```