Majority Element(>n/2 times)[Moore’s Voting algorithm]

BRUTE FORCE :

Select an element and count it’s occurence in rest of the array if it is greater than n/2 return that element otherwise keep iterating.

```
int majorityElement(vector<int> v) {

    //size of the given array:
    int n = v.size();

    for (int i = 0; i < n; i++) {
        //selected element is v[i]
        int cnt = 0;
        for (int j = 0; j < n; j++) {
            // counting the frequency of v[i]
            if (v[j] == v[i]) {
                cnt++;
            }
        }

        // check if frquency is greater than n/2:
        if (cnt > (n / 2))
            return v[i];
    }

    return -1;
}
```

- Time Complexity : O(N^2)
- Space Complexity : O(1)

Better Solution : Using Hashing

```
#include <bits/stdc++.h>
```

```
int findMajorityElement(int arr[], int n) {
  // Write your code here.
  map<int,int> m;
  for(int i=0;i<n;i++)
  {
    m[arr[i]]++;
  }
  for(int i=0;i<n;i++)
  {
    if(m[arr[i]]>n/2)
      return arr[i];
  }
  return -1;
}
```

- Time Complexity : O(NlogN)
- Space Complexity : O(N)

Optimal Approach : Moore's Voting Algorithm

We maintain two var count and element .

Initially we initialise count=0 .

If count is 0 then we set count to 1 and initialise elem to curr pointing elem.

We iterate through array and if it is equal to the number stored in elem then we increment count else we decrement it . If count becomes zero we reset elem to curr elem.

At last elem var will store the element that occurs in majority. Thus we again iterate through the array before linear search and count the occurence of this elem if it is greater than n/2 we return it else we return -1

```
#include <bits/stdc++.h>

int findMajorityElement(int arr[], int n) {
  // Write your code here.
  int count=0,elem=arr[0];
  for(int i=0;i<n;i++)
  {
          if (count == 0)
      {
      count=1;
            elem = arr[i];
          }
                if(arr[i]==elem)
      count++;
    else
      count--;

  }
  count=0;
  for(int i=0;i<n;i++)
  {
    if(arr[i]==elem)
      count++;
  }
  if(count>n/2)
    return elem;
  return -1;
}
```

- Time Complexity : O(N) + O(N)
- Space Complexity : O(1)