Majority Element(>n/3 times)

BRUTE FORCE :

We need to know how many numbers exist in the array where the freq of an element can be > floor(n/3) where n i the size of the array. A small observation here is consider array of size=8

n/3= 2 so freq must be >2 i.e 3 or more if we consider the min freq needed i.e 3 then 3+3+2 makes up 8 which gives us max 2 elements which can occur in majority. So now we know that our ans vector can only be of size ≤2(using this in our approach).

Traverse element one by one and if the ans vector is empty or it contains 1 element then this means we have to perform search so the inner loop will run only if this condition is satisfied where either the ans vector is empty or it does not contain and element which is already counted in our answer .

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

vector<int> majorityElementII(vector<int> &arr)
{
    // Write your code here.
    vector<int> ans;
    int cnt=1;
    for(int i=0;i<arr.size();i++)
    {
        cnt=1;
        if(ans.empty() || ans.back()!=arr[i])
        {
            for(int j=i+1;j<arr.size();j++)
            {
                if(arr[j]==arr[i])
                    cnt++;
            }
            if(cnt>arr.size()/3)
                ans.push_back(arr[i]);
        }
    }
    return ans;
}
```

- Time Complexity : O(N^2)
- Space Complexity : O(1) as the vector size is ≤ 2 in every case

Approach 2:

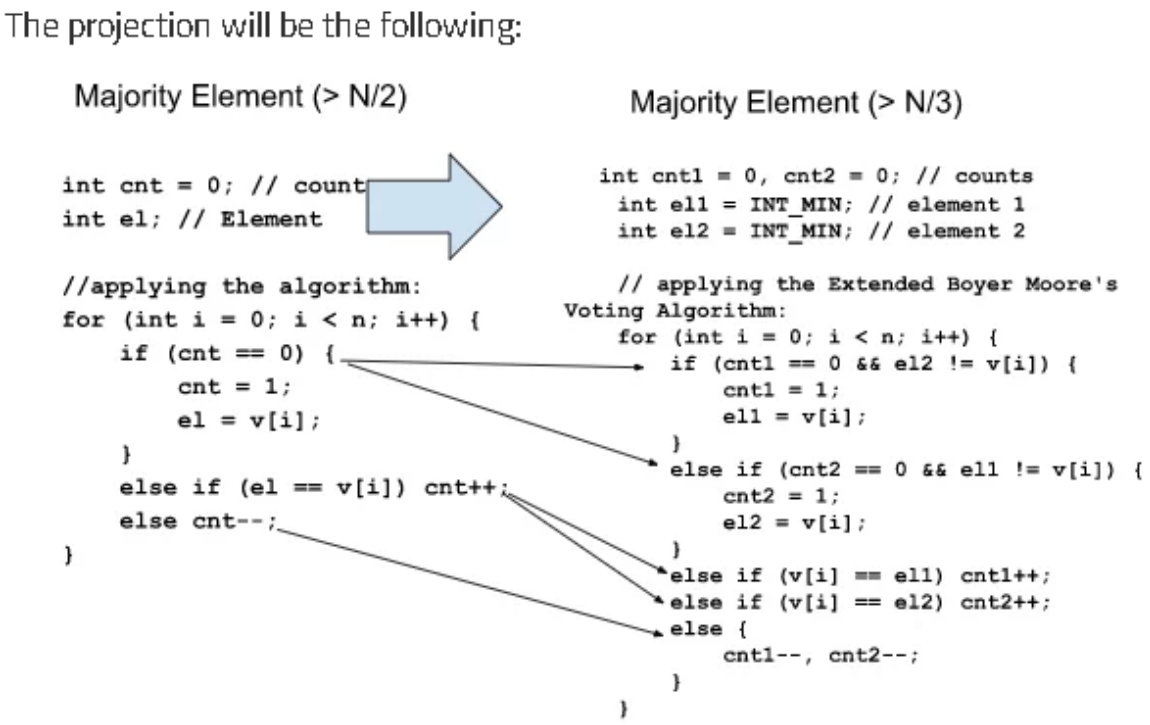
Hashing and finding out where freq ≥n/3 and including to ans.

- Time complexity : O(NlogN) in case of ordered map, if unordered map is used it is O(N*1) but in worst case O(N) is used by unordered map so O(N^2) in worst case.
- Space Complexity : O(N) for map

Approach 3: Extended Boyer Moore's voting algorithm

As we know we will have at max only two elements in our ans we make use of moore's voting algo and keep track using 4 variables cnt1,cnt2, ekem1,elem2.

Projection of majority element 1 problem :



cnt1 counts 1st majority element and cnt2 counts 2nd we use if else if so that only one condition if satisfied every time. We pick up one element and check whether cnt1 is 0 if yes then it will be set to 1 and elem1 will contain this element after this the loop moves for next iteration making sure that only one either of cnt1 or cnt2 is set but there is a problem when one of the cnt is zero and other is not and the current element is the one that is already store in one of the elem var say 1

Ex:

cnt1= 1, elem=1

cnt2=0

Now current elements being processed is 1 so when he cnt2==0 condition is satisfied cnt2 is set to 1 and elem2 now contains 1.Now both elem 1 and elem2 contains 1 and there freq values will be garbled so to avoid this case we also include a cond when either of ct is zero we check that arr[i] should not be equal to elem variables of either contrary cnt variable(cnt1 and elem2 condition is checked) .

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

vector<int> majorityElementII(vector<int> &arr)
{
    // Write your code here.
    int cnt1=0,cnt2=0,n=arr.size(),elem1,elem2;
    vector<int> ans;
    for(int i=0;i<n;i++)
    {
        if(cnt1==0 && arr[i]!=elem2)
        {
            cnt1=1;
            elem1=arr[i];
        }
        else if(cnt2==0 && arr[i]!=elem1)
        {
            cnt2=1;
            elem2=arr[i];
        }
        else if(arr[i]==elem1)
        {
            cnt1++;
        }
        else if(arr[i]==elem2)
        {
            cnt2++;
        }
        else{
```

```
            cnt1--;
            cnt2--;
        }
    }
    cnt1=0,cnt2=0;
    for(int i=0;i<n;i++)
    {
        if(arr[i]==elem1)
            cnt1++;
        else if(arr[i]==elem2)
        {
            cnt2++;
        }
    }
    if(cnt1>n/3)
        ans.push_back(elem1);
    if(cnt2>n/3)
        ans.push_back(elem2);
    return ans;
}
```

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