Majority element ($> \lfloor \frac{n}{3} \rfloor$)

8

2	1	1	3	1	4	5	6
0	1	2	3	4	5	6	7

$n = 8$

$\lfloor \frac{n}{3} \rfloor = 2$

- **Brute Force :—**

We iterate over the array and count the frequency of each element. As soon as count is greater than $\frac{n}{3}$, we store it. And then again we find the frequency of each elements (except previously stored element) and as soon as we get count $> \frac{n}{3}$, we store it.

Here, we are finding two elements because we can have maximum 2 elements whose count $> \frac{n}{3}$.

So, $T = O(n^2)$

$S = O(1)$

- **Better Approach :—**

Same as the approach we used for majority element ($> \lfloor \frac{n}{2} \rfloor$)

— Refer to majority element ($> \lfloor \frac{n}{2} \rfloor$)

- **Best Approach** (Moore's Voting Algorithm)

Cancel Out Algorithm

Example 1 :—

i i i i i i i i

2	1	1	3	1	4	5	6
0	1	2	3	4	5	6	7

$n = 8$

$\lfloor \frac{n}{3} \rfloor = 2$

ele1 = ~~2~~ ~~4~~ ~~6~~

count1 = ~~1~~ ~~0~~ ~~1~~ ~~0~~ 1

ele2 = 1

count2 = ~~1~~ ~~0~~ ~~1~~ ~~0~~ 1

```
int ele1 = ∞ ; // Anything which is not in the array.
int ele2 = ∞-1 ;// Anything which is not in the array && ≠ ele1
for(int i=0; i< arr.length ; i++){

    if(count1 ==0 && arr[i] != ele2){
        ele1 = arr[i];
        count1 = 1;
    } else if(count2 ==0 && arr[i] != ele1){
        ele2 = arr[i];
        count2 = 1;
    } else if( ele1 == arr[i]) {
        count1++;
    } else if (ele2 == arr[i]) {
        count2 ++;
    } else {
        count1--;
        count2--;
    }
}
```

After, this step, we have ele1 and ele2 with some value in it.

Now we will find count of ele1 and ele2 in $O(n)$ time. And if count $> \frac{n}{3}$ then we will store that element (or elements)

$T = O(n) + O(n) = O(2n) = O(n)$	$S = O(1)$

Example 2: (Just for more clarity)

i	i	i	i	i	i	i	i
4	8	6	6	8	6	8	5
0	1	2	3	4	5	6	7

$n = 8$

$\left\lfloor \frac{n}{3} \right\rfloor = \frac{8}{3} = 2$

ele1 = ~~4~~ 6 ele2 = ~~8~~ 8

count1 = ~~1~~ ~~0~~ ~~1~~ 1 count2 = ~~1~~ ~~0~~ ~~1~~ 1