Todays Content

1. Majority Element
2. Min swaps $< =$ B elements

#Majority Element

Given ar[N] elements, return majority elements.

An element is said to be majority, if it's frequency > N/2

#If no majority return -1.

Ex1:

```
           0  1  2
ar[3] = { 2  1  4 }
```

#No majority return -1;

Ex2:

```
          0  1  2  3  4  5  6
ar[7] = { 3  4  3  2  4  4  4 }
```

freq(4) > 7/2

4 > 3 ✓

Ex3:

```
          0  1  2  3  4  5  6  7
ar[8] = { 3  3  4  2  4  4  2  4 }
```

freq(4) > 8/2

4 > 4 ✗

#No majority return -1;

Ex4:

```
           0  1  2  3  4  5  6  7  8  9  10
ar[11] = { 3  4  3  6  1  3  2  5  3  3  3 }
```

freq(3) > 11/2

6 > 5

Ex5:

```
           0  1  2  3  4  5  6  7  8  9
ar[10] = { 4  6  5  3  4  5  6  4  4  4 }
```

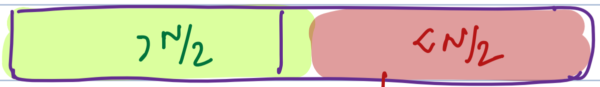
freq(4) > 10/2

5 > 5

#No majority return -1;

Q At max how many diff majority elements we can have? 1.

ar[N] :

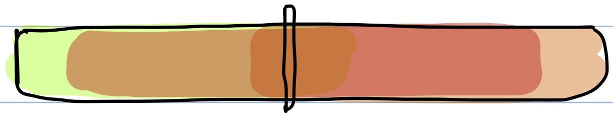


↳ Here, we cannot have another majority.

Idea1:
```
For every element arr[i]:
|   Iterate on arr[] & calculate frequency of arr[i] = c.
|   if(c > N/2){ return arr[i]}
}
return -1;
```

TC: $O(N*N) = O(N^2)$ SC: $O(1)$

Idea2: Sort arr[] & Take centre ele = arr[N/2]



```
Iterate on arr[] & get frequency of arr[N/2] = c.
if( c > N/2){
|    return arr[N/2]
}
else{
|    return -1;
}
```

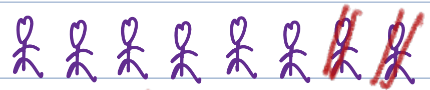
TC: $O(N\log N + N)$
↳ Sort ↳ Check middle element

Idea3: Insert all elements in hashmap
```
For every element arr[i]:
|   Get frequency of arr[i] from hashmap = c.
|   if(c > N/2){ return arr[i]}
}
return -1;
```

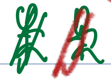
TC: $O(N + N*1) = O(N)$ SC: $O(N)$ ↗ hashmap

#Idea3 : Election GOA 13 Seats

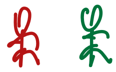
Ex1:

Manash : 

Nazneen :

Munaf : 

N = 13

# Seats (Manash) > 13/2		N = N-2	
# Seats (Manash) > 11/2		N = N-2	
# Seats (Manash) > 9/2		N = N-2	
# Seats (Manash) > 7/2			

#obs: By deleting 2 diff items, majority will not change.

Ex2: Election Andaman 7 Seats

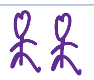
Manash :

Nazneen :

Munaf :

N = 7

#obs: By deleting 2 same items, we might loose majority

Seats (Manash) > 7/2  N = N-2

Seats (Manash) > 5/2 #majority lost

#Idea4:

1. Keep deleting 2 distinct elements, untill we have a 1 unique element
2. Iterate & check if 1 unique ele is majority.

Ex1: ar[11] = {

0	1	2	3	4	5	6	7	8	9	10
~~3~~	~~4~~	~~3~~	~~6~~	~~1~~	~~3~~	~~2~~	~~5~~	3	3	3

}

Ex2: ar[11] = {

0	1	2	3	4	5	6	7	8	9	10
~~4~~	~~6~~	~~5~~	~~3~~	4	~~5~~	~~6~~	4	4	~~4~~	~~10~~

}

Moore's Voting Algo:

0 1 2 3 4 5 6 7 8 9 10

arr[11] = {3 3 4 6 1 ~~3~~ 2 ~~5~~ 3 3 3}

#ele = ~~0~~ ~~3~~ ~~1~~ ~~2~~ 3

freq = ~~0~~ 1 2 1 ~~0~~ ~~1~~ 0 ~~1~~ ~~0~~ ~~1~~ ~~2~~ 3

Iterate in arr[]:

if freq of 3 = 6 > 11/2 : return 3.

0 1 2 3 4 5 6 7 8 9 10

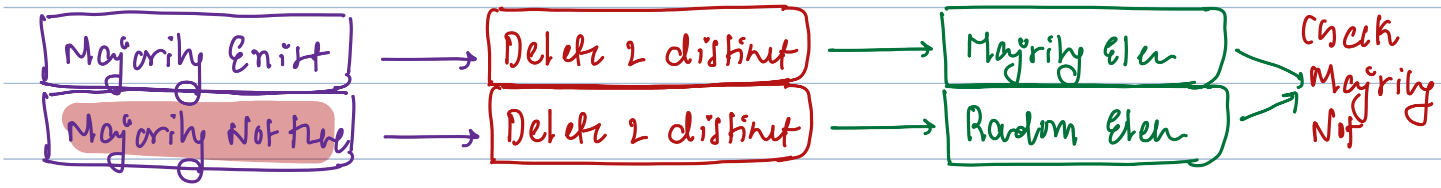
arr[11] = {4 6 5 3 4 5 6 4 4 4 10}

#ele = ~~0~~ ~~4~~ ~~5~~ ~~4~~ ~~6~~ 4

freq = ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~2~~ 1

Iterate in arr[]:

if freq of 4 = 5 > 11/2 : return -1; # No majority.



```
int majorityElement(vector<int> &ar){

    int ele=∞, f=0;
    int N= ar.size();
    for(int i=0; i<N; i++){
        #Compare 2 ele #{ele, ar[i]}
        if(f==0){
            ele=ar[i]; f=1;
        }
        else if( ele == ar[i]){
            f++;
        }
        else{ # ele != ar[i]
            f--;
        }
    }

    int c=0;
    for(int i=0; i<N; i++){
        if(ar[i]==ele){
            c++;
        }
    }
    if( c > N/2){return ele;}
    else{return -1;}
}
```

Variation: It's majority if it's > N/3

Hint: Delete 3 distinct elements majority won't change

$\{ ele_1 \quad ele_2, ar[i] \}$ $\boxed{\text{TODO: 5 conditions}}$

$f_1 \quad f_2$

29 Min swaps to bring all elements <= B together.

↳ continous / Subarray

0 1 2 3 4 5 6

Ex1: arr[] = {1 12 10 5 14 10 3} B = 8 #swaps = 2

1 5 3

Ex2: arr[] = {3 7 6 13 2 15} B = 7 #swaps = 1

3 7 6 2

Ex3: arr[] = {25 30 2 ~~18~~ 7 6 9 50 3} B = 10 #swap = 1

2 3 7 6 9

#Idea

Hint1: Subarray of fix length.

0 1 2 3 4 5 6 7 8

arr[] = {19 11 3 9 7 25 6 20 4} B = 10

#subarray k = 5 <= B > B

s ..	e	#good ele	#bad ele	#swaps
0	4	3	2	2 swaps
1	5	3	2	2 swaps
2	6	4	1	1 swap
3	7	3	2	2 swaps
4	8	3	2	2 swaps

ans = min swaps = 1

#obs For every subarray of len = k TC: O(N-K+1) * O(K) = O(N²) SC: O(1)

Iterate & calculate no: of bad element = No: of swaps

ans = min(ans, swaps)

return ans;

Idea2 Optimize with sliding window

We slide no: of bad elements forward.

	0	1	2	3	4	5	6	7	8	
arr[] = {	~~19~~	~~11~~	~~3~~	~~9~~	~~7~~	25	6	20	4	} B = 10

subarray

s ..	e	remove	add	#bad ele	#swaps
0	4			2	2
1	5	19	25	2	2
2	6	11	6	1	1
3	7	3	20	2	2
4	8	9	4	2	2

Note:

If we remove bad element decrease count by 1

If we add bad element Increase count by 1.

```
int minSwaps(vector<int> &arr, int B){
```

TC: O(N + N) = O(N) SC: O(1)

↳ # Sliding window of len = k

↳ # Count ele <= B, say = k