Interview Problems.

"Learning is never done without errors and defeat." - Vladimir Lenin

Agenda:

Max occurrence's 1 by →

a) Almost 1 replace

b) Almost 1 swap.

Majority Element

Analysing Constraints

Why??

It helps us determine which time comp. & data structure to use for a given problem.

Note: In interviews don't ask constraints directly. First, tell your approach & ask if you need to optimize further.

General Guidelines:

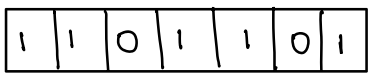
Constraint	Possible time complexities
$n <= 10^6$	$O(n)$, $O(n \log n)$
$n <= 20$	$O(n^2)$, $O(n^3)$, $O(n!)$
$n <= 10^{10}$	$O(\log n)$, $O(\text{sqrt}(n))$

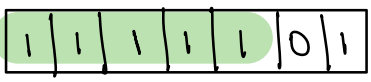
Amazon, Microsoft, facebook.

Q1. Given a binary arr[], we can atmost replace a single 0 with 1. Find the maximum consecutive 1's we can get in arr[].

0, 1

eg1.





ans = 5

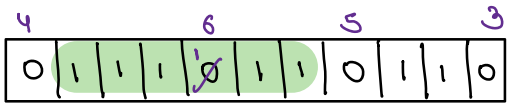
ans = 4.

ans = 5

eg2.

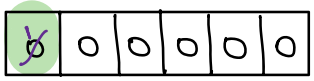
ans = 6

eg3.



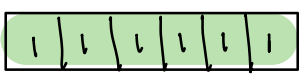
ans = 6

eg4.



ans = 1

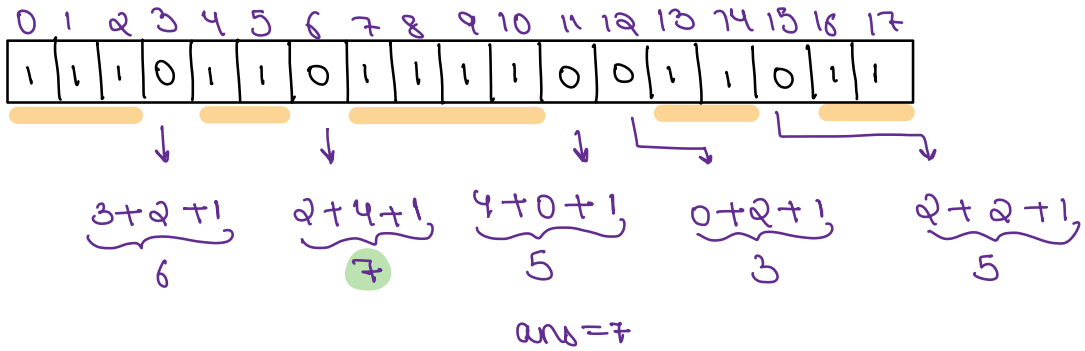
eg5.



ans = 6

i
i-1 i

eg:



ans = 7

Idea:

For every 0's

1. Calculate no. of consecutive ones on LHS → l
2. Calculate no. of consecutive ones on RHS → r

if(l + r + 1 > ans) → ans = l + r + 1

* Edge cases.

1. if all elements are 0 . ans = 1
2. if all elements are 1, ans = ar. length;

Code:

```
int maxones(int[] ar){
    int n = ar.length, ans=0, count=0;
    for(int i=0; i<n; i++){
        if(ar[i]==1){count++;}   // count → no. of 1's
    }
    // Edge case [0's and 1's]
    if(count==0){return 1;}
    if(count==n){return n;}
    for(int i=0; i<n; i++){
        if(ar[i]==0){   // center point
            // count consecutive 1's on left
            int j=i-1;
            int l=0;   // count of left
            while(j>=0 && ar[j]==1){
                l++;
                j--;
            }
            // count consecutive 1's on right
            j = i+1, int r=0;
            while(j<n && ar[j]==1){
                r++; j++;
            }
            int max=l+r+1;
            ans= Math.max(ans, max);
        }
    }
    return ans;
}
```

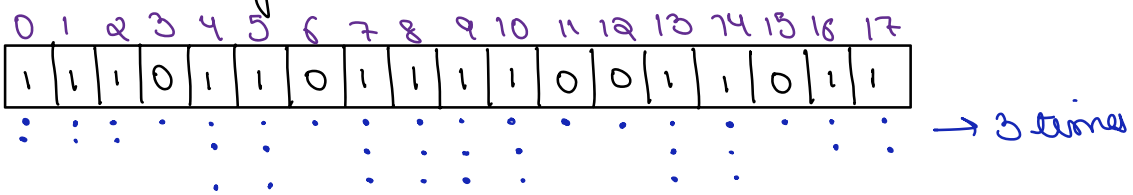
i	0	1	2	3	4	5
	1	1	1	0	1	1

i=3 → 0
j=2, l=1
j=1, l=2
j=0, l=3
j=-1 → stop

i	0	1	2	3	4
	1	1	0	1	1

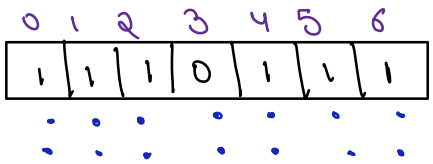
i=3, r=1
j++
j=4, r=2
j++
j=5 → stop

Time complexity



i = 0 to N
↳ 3 times
] total iteration
3N
O(N)

eg.



⇒ 2N ⇒ TC: O(N)

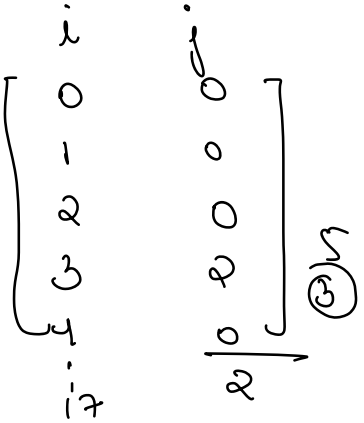
eg: 1, 0, 0, 0, 1, 1, 1, 0, 0, 0

3N
⇒ TC: O(N)

eg: 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1

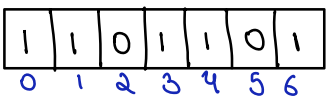
⇒ 2N
⇒ TC: O(N)

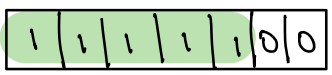
TC: O(N)
SC: O(1)



Q2. Given a binary arr[], we can atmost swap a single 0 with 1. Find the max consecutive 1's we can get in an arr[].

Eg:





ans = 5

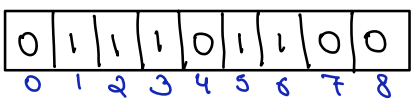
ans = 4

} Ans = 5

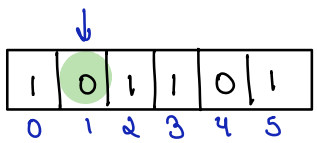
Eg2.

1, 1, 1, 1, 1, 0 ans = 5

Eg3.



eg:



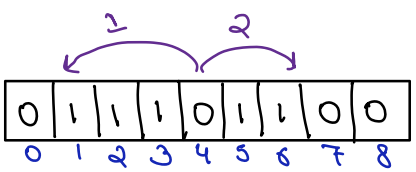
$i = 1, \quad l = 1, \quad r = 2$

countOnes = 4 // total no. of ones

$l + r < countOnes$.

↳ we have answer 1 in case to swap arr[i] = 0 with

maxCount = $l + r + 1$



1st

0 1 0 1 1 1 1 1 1 0 1 0

and

0 1 1 1 1 1 1 1 1 0 1 0

$l = 3, \quad r = 2$

countOnes = 5

$l + r == countOnes$

↳ no extra one to swap this 0.

maxCount = $l + r$

Idea:

For every 0's

1. Calculate no. of consecutive ones on LHS → l
2. Calculate no. of consecutive ones on RHS → r

```
if( l + r == total no. of ones)
    max = l + r
}else{
    max = l + r + 1
}
```

↳ TODO

TC: O(N)

SC: O(1)

After while →

```
if( l + r == count){
    max = l + r;
} else{
    max = l + r + 1;
}
ans = Math.max(max, ans);
```

↳ Break → 10:40

Q3. Given N arr elements, find majority element without modifying array.

ele with freq $> \frac{N}{2}$

arr[] = {2, 1, 4} ⇒ No majority ele

0 1 2 3 4 5

arr[] = {1, 2, 1, 6, 1, 1}, N = 6

cf 1 = 4 > 6/2 → M.E = 1

cf 2 = 1

cf 6 = 1

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

cf 3 = 6 > 11/2 M.E = 3

arr[] = {4, 6, 5, 3, 4, 5, 6, 4, 4, 4}

cf 4 = 5, 10/2 ×

[No majority element]

Quiz. At max how many majority elements can be there in an array.

Lets say we have 2 majority elements → M_1, M_2

$$\begin{array}{l} freq(M_1) > N/2 \\ + \; freq(M_2) > N/2 \\ \hline freq(M_1) + freq(M_2) > N \end{array}$$

X → not possible

$$M.E > \frac{N}{2}$$

Brute force

find freq of every element.
& check freq $> \frac{N}{2}$

code:

```
int majorityElement(int[] arr){

    for(int i=0; i<n; i++){
        int ele = arr[i];
        //find freq of ele.
        int freq=0;
        for(int j=0; j<n; j++){
            if(arr[j]==ele)
                freq++;
        }
        if (freq > N/2) {
            return ele;
        }
    }
    return -1;
```

→ TC: $O(N^2)$

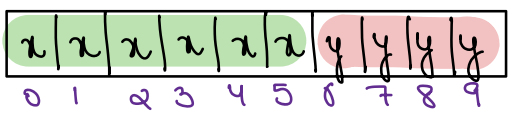
SC: $O(1)$

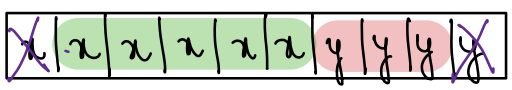
Observations.

1. There is only one majority element in the array

$M.E > \frac{N}{2}$

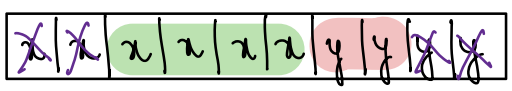
2. If we remove any two distinct elements. the majority element remains same.



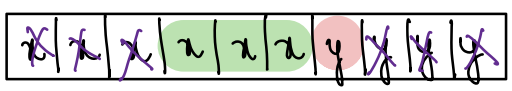


$n = 8$

$5 > \frac{8}{2}$



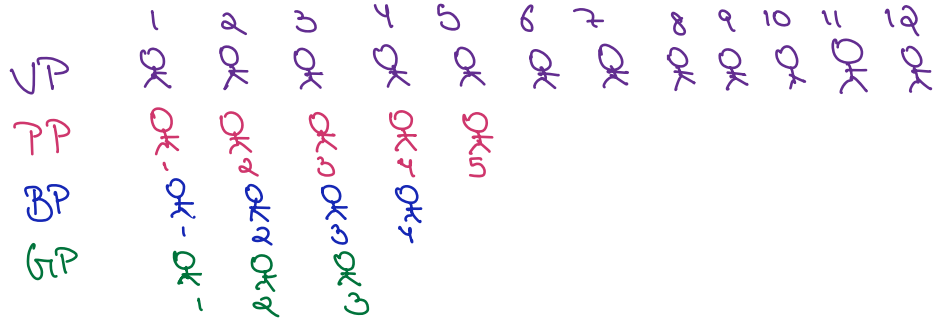
$M.E = x$



$M.E = x$

Moore's voting Algorithm

Explanation:



Remove	VP	BP	PP	GP	winner
Remove 1 VP & 1 GP	11	4	5	2	VP
Remove 1 VP & 1 BP	10	3	5	2	VP
Remove 1 VP & 1 PP	9	3	4	2	VP
Remove 1 VP & 1 BP	8	2	4	2	VP

A	B	C
1	0	0
✗	✗	0
1	0	0
✗	0	✗
1		

✗ ✗ ✗ ✗ ✗

A ⇒ 2 , B = 5

B has 3 more votes than A

* if freq of M.E > $\frac{N}{2}$.

then some occurences of M.E will be together.

Idea:

1. Iterate through the array, keeping track of M.E candidate & its freq.
2. if the next element is the same as candidate, inc. the count otherwise, dec the count.
3. if $C = 0$, update the candidate for M.E & reset count = 1.
4. Continue iterating & repeat step 2.

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

$i = 0$, candidate $= 3$.
$c = 1$.

$i = 1$, candidate $==$ arr[i]
$3 == 4 \rightarrow$ false $\rightarrow$ count $--$, $c = 0$.

candidate $= 4$, $c = 1$

$i = 2$, $3 == 4$
arr[i] $==$ candidate $\rightarrow$ false, $c--$, $c = 0$

candidate $= 3$
$c = 1$

$i = 3$, arr[i] $==$ candidate
$6 == 3 \rightarrow$ false, $c--$, $c = 0$

candidate $= 6$
$c = 1$.

$i = 4$, $1 == 6 \rightarrow$ false, $c--$, $c = 0$

candidate $= 1$.
$c = 1$

$i=5$, $3 == 1 \rightarrow$ false, $c--$, $c=0$

Candidate $= 3$

$c = 1$.

$i=6$, $2 == 3 \rightarrow$ false, $c--$, $c=0$

Candidate $= 2$

$c = 1$.

$i=7$, $5 == 2 \rightarrow$ false, $c--$, $c=0$.

Candidate $= 5$

$c = 1$.

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

$i=8$, $3 == 5 \rightarrow$ false, $c--$, $c=0$

Candidate $= 3$

$c = 1$

$i=9$, $3 == 3 \rightarrow$ true, $c++$, $c=2$

$i=10$, $3 == 3 \rightarrow$ true, $c++$, $c=3$

Best candidate for M.E = 3.

Iterate over array & find freq of best candidate

freq of 3 = 6

$6 > \frac{10}{2}$ ⇒ Element is M.E

Code:

```
int ME(int[] ar){
    int me_index = 0;
    int count = 1, n = ar.length;
    for(int i=1; i<n; i++){
        if(count==0){
            me_index = i;
            count = 1;
        }
        else{
            if(ar[me_index] == ar[i]){
                count++;
            }
            else{
                count--;
            }
        }
    }
```

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

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

{3, 2, 3, 9, 3, 4, 3, 4, 3, 2, 3} N = 11

6, $\frac{11}{2}$ ⇒ 5

A	B
1	0
0	0
0	1