Analysing constraints

→ It helps to narrow down which time complexity and data structure or algorithm to use for a given problem

→ It is important to look at constraints when any problem.

→ Asking constraints directly from interviewer is some times considered as hint.

→ Instead of directly asking, given your approach, it's TC & SC and check if it works.

- → Does this make sence to you?
- → Will this TC works?
- → Will this SC works?
- → Should I focus on optimising more?
- → Should I foucus on time or space.

Constraint	Possible TC
$n <= 10^6$	O(N), O(N log N)
$n <= 20$	$O(2^N)$, O(N!)
$n <= 10^{10}$	O(log N), O(1), O(sqrt(N))

These are just general guidelines.
The actual TC can vary based on the specific problem and implementation.

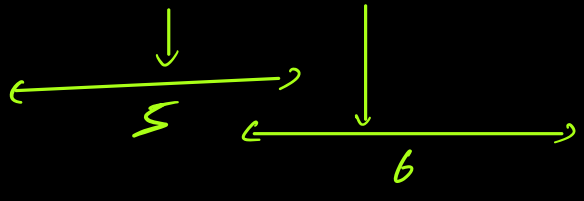
Q1: Given an array of 1's and 0's, you are allowed to replace at max one 0 with 1.
Find the maximum no. of consecutive 1's that can be obtain after making replacement.

Amazon
Microsoft
Adobe etc.

A = [1 1 0 1 1 0 1 1]

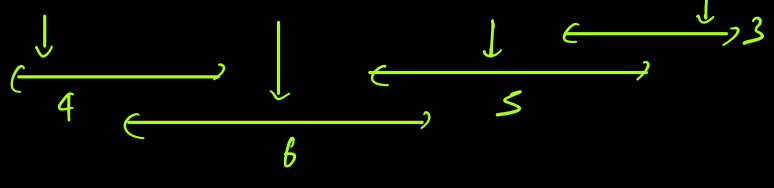
Ans: 5 replace 2nd or 5th index.

Quiz 1: A = [1 1 0 1 1 0 1 1 1]



Ans: 6

Quiz 2: A = [0 1 1 1 0 1 1 0 1 1 0]



ans: 6

Solution Approach:

1) Maintain a 'ans' variable to track max len.
2) Initialise it with 0
3) Iterate through array, whenever we encounter a '0'.
   - → Count no. of consecutive 1's on left: l
   - → Count no. of consecutive 1's on right: r
   - → if (l+r+1 > ans) ans = l+r+1

Edge Case: A: [1 1 1 1 1] ans: 5

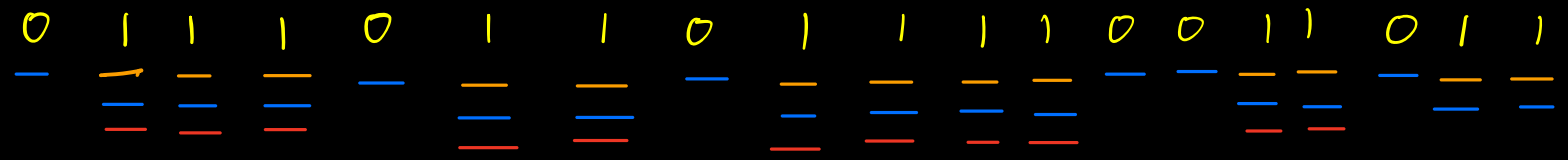
```
int findMaxConsecutiveOnes( int nums[]){
    int n = nums.size()
    int ans = 0
    int totalOnes = 0;

    for(int i=0; i<N; i++){
        if(nums[i] == 1)
            totalOnes++;
    }
    if(totalOnes == n)
        return n;

    for(i=0; i<n; i++){
        if(nums[i] == 0){
```

```
int l=0, r=0;
int j= i+1
while(j<n && nums[j]==1){
    r++;
    j++;
}
j = i-1
while(j>=0 && nums[j]==1){
    l++
    j--
}
if( l+r+1 > ans)
    ans = l+r+1
}
}
return ans
}
```

Time Complexity

Let's count number of access to each element

— => i — => j going right — => j going left



r=2, l=2
ans = 4 & 7

Every element is getting accessed at max 3 times

∴ # iterations = 2N

TC: O(N)

Q2: Given an array of 1's and 0's, you are allowed to SWAP at max one 1 with a 0. Find the maximum no. of consecutive 1's that can be obtain after making replacement.

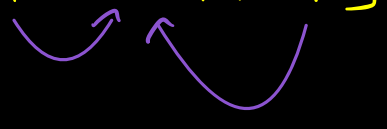
Directi
Microsoft
Amazon
Adobe

A = [1 0 1 1 0 1]

ans = 4

swap A[1] with A[5]
swap A[4] with A[0]

Quiz: A = [1 1 0 1 1 1]



ans = 5

```
if ( l+r == totalOnes)
        ans = max(ans, l+r)
else
        ans = max(ans, l+r+1)
```

```
int findMaxConsecutiveOnes(int nums[]) {
    int n = nums.length()
    int ans = 0
    int totalOnes = 0;

    for (int i = 0; i < N; i++) {
        if (nums[i] == 1)
            totalOnes++;
    }
    if (totalOnes == n)
        return n;

    for (i = 0; i < n; i++) {
        if (nums[i] == 0) {

            int l = 0, r = 0;
            int j = i + 1
            while (j < n && nums[j] == 1) {
                r++;
                j++;
            }
            j = i - 1
            while (j >= 0 && nums[j] == 1) {
                l++
                j--
            }
            curAns = l + r + 1
            if (l + r == totalOnes)
                curAns = l + r

            if (curAns > ans)
                ans = curAns
```

```
    }
    return ans
}
```

Majority Element

Given N elements, find the majority element
The majority element is the element that occurs more than N/2 times where N is size of the array. Google, Facebook etc.

A = [2 1 4]
ans: -1 (No majority element)

A = [3 4 3 2 4 4 4]
N = 7, N/2 = 3.5 > at least 4 times

ans: 4

A = [3 3 4 2 4 4 2 4]
N = 8, N/2 = 4 < at least 5 times

ans: -1 (No majority element)

Quiz: 3 4 3 6 1 3 2 5 3 3 3

$N = 11 \Rightarrow N/2 = 5.5$ atleast 6

and: 3

Quiz: 4 6 5 3 4 5 6 4 4 4

$N = 10 \Rightarrow N/2 = 5$ atleast 6

ans: -1 (NO ME)

At max how many majority element can be there in an array?

Ans: 1

Proof by contradiction:

Let's say there are two ME $\Rightarrow x, y$

$$freq(x) > \frac{N}{2} \Rightarrow freq(x) = \frac{N}{2} + 1$$

$$freq(y) > \frac{N}{2} \Rightarrow freq(y) = \frac{N}{2} + 1$$

$$freq(x) + freq(y) = \frac{N}{2} + \frac{N}{2} + 1 + 1$$

$$= N + 2$$

It's not possible, thus assumption was clear

Brute Force: Put all elements in hashmap and return the element with freq more than $N/2$.

TC: $O(N)$, SC: $O(N)$

The requirement is $O(1)$ SC.

Brute Force: Check the freq of each element by iterating in the array

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

Slight optimized: Sort using $O(1)$ SC sorting algo.
Compare neighbours to count the freq and return ME.

TC: $O(N \log N + N)$
$\approx O(N \log N)$ SC: $O(1)$

Moore's Voting Algorithm

→ There can be only single ME

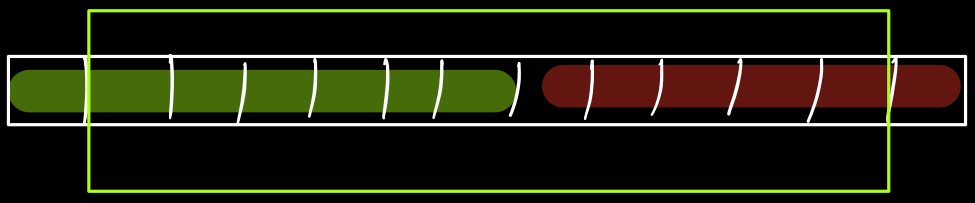
Proof above.

→ If you remove 2 distinct elements then the majority element will remain the same

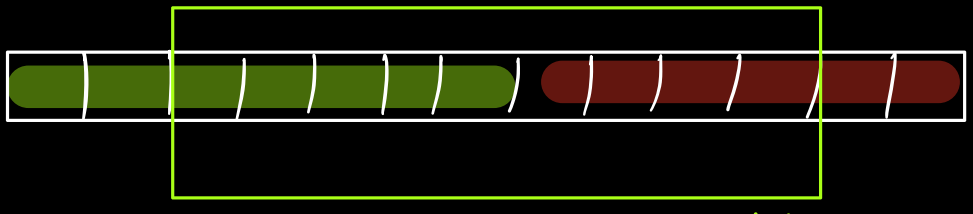
N = 13

N/2 = 6.5

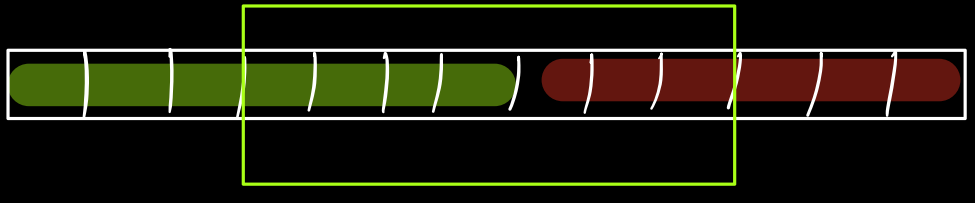
To be ME we need atleast 7



N = 11, req = 6, green is still ME



N = 9, req = 5, green is still ME

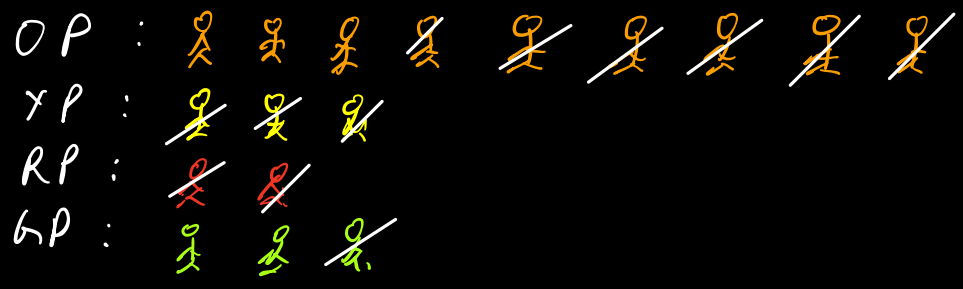


N = 7, req = 4, green is still ME

Voting Example :

OP : 9 candidates
YP : 3 candidates
RP : 2 candidates
GP : 3 candidates

Total candidats = 17 / To be ME min req = 9



Remove	OP	YP	RP	GP	Winner
OP & RP	8	3	1	3	OP
OP & YP	7	2	1	3	OP
OP & RP	6	2	0	3	OP
OP & YP	5	1	0	3	OP
OP & YP	4	0	0	3	OP
OP & GP	3	0	0	2	OP

If we remove 2 different element from the array the ME does not change.

Algorithm:

→ Iterate through each element in the array, keep tracking ME and it's frequency

→ If next element is same as our ME the increase the count, elict decrease the count.

→ If count becomes zero, update the ME to the current element and reset the count to 1.

→ After the iteration, go through the array and check if freq(ME) > N/2 or not.
If it's not the ME does not exist.

3 4 3 6 1 3 2 5 3 3 3
↑

ME = 3 3 x x 3

count = x 0 x 0 x 0 x 0 x 1 3

ans = 3

```
int findME(int a[], int size){
    int me = A[0], count = 1
    for(i=1; i<size; i++){
        if(count==0){
            me = A[i]
            count = 1
        }
        else{
            if(me == A[i])
                count++
            else
                count--
        }
    }

    // Need to check even if ME exists ?
    int count = 0;
    for(i=0; i<size; i++){
        if(A[i]==me) count++
    }
    if(count > size/2){
        return me;
    }
    return -1;
}
```

1 2 3 4 4 4

TC: O(N)

SC: O(1)

me = 4

count = 0 1 0 1 2

There is no ME

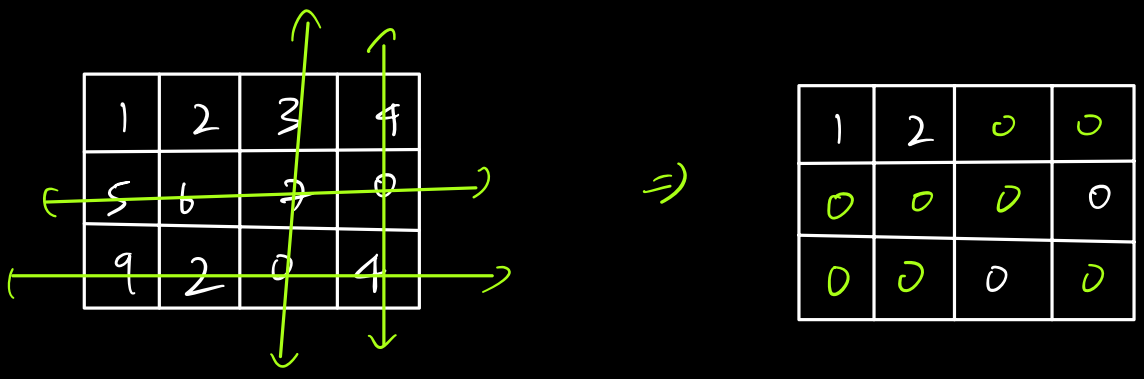
1 3 3 3 3 1 1

me = 3

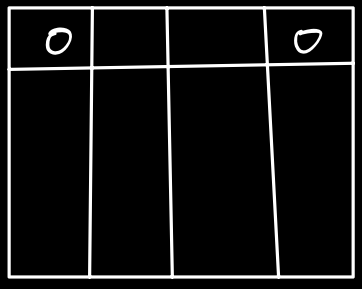
count = 1 2 3 4

count = 4 > 7/2 ∴ ME = 3

Q Given 2D +ve integer matrix. make all the elements in a row or column zero if A[i][j]==0 Specifically make i^{th} row and j^{th} column zero.



ans.



Sol: → For every A[i][j]==0 make all elements in i^{th} row and j^{th} column *= -1

→ Iterate again and make element 0 which are -ve.

TC: O((N+M)(N+M)) ≈ O(N3)

optimize: Also if there are -ve.

→ Create two sets. row_to_make_zero
col_to_make_zero

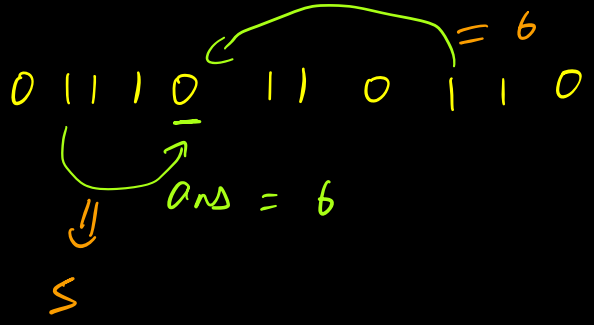
→ if A[i][j] == 0, the row_to_make_zero.insert(i)
col_to_make_zero.insert(j)

→ Iterate in sets and make zero.

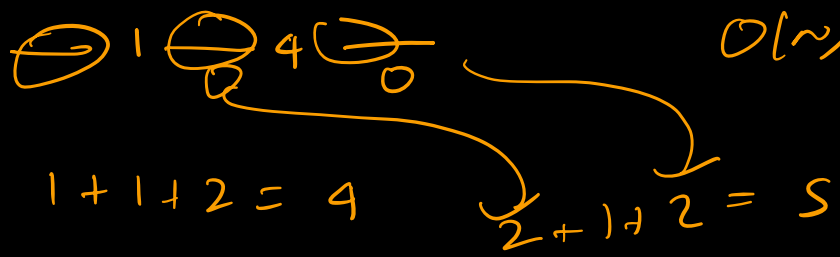
TC: O((N+M) + 2*(N+M))

= O(N+M)

Doubts



1 0 1 1 0 1 1



O(N) SC O(N)