SCALER

Interview Problems

TABLE OF CONTENTS

Contest : 23rd may (7:00 - 8:30 am)

Post contest, contest discussion class from 8:30am on 23rd

Advance module starts : 27th may.

< Question > : Given a binary array [ ]. We can atmost replace a single 0 with 1. Find the maximum consecutive 1's we can get in the array[ ] after the replacement.

$1 \le N \le 10^3$

[1 1 0 1 1 0 1 1 1]

1 1 1 1 1 0 1 1 1 → 5

1 1 0 1 1 1 1 1 1 → 6

ans : 6

Quiz 1 :

[0 1 1 1 0 1 1 0 1 1 0]

1 1 1 1 0 1 1 0 1 1 0 → 4

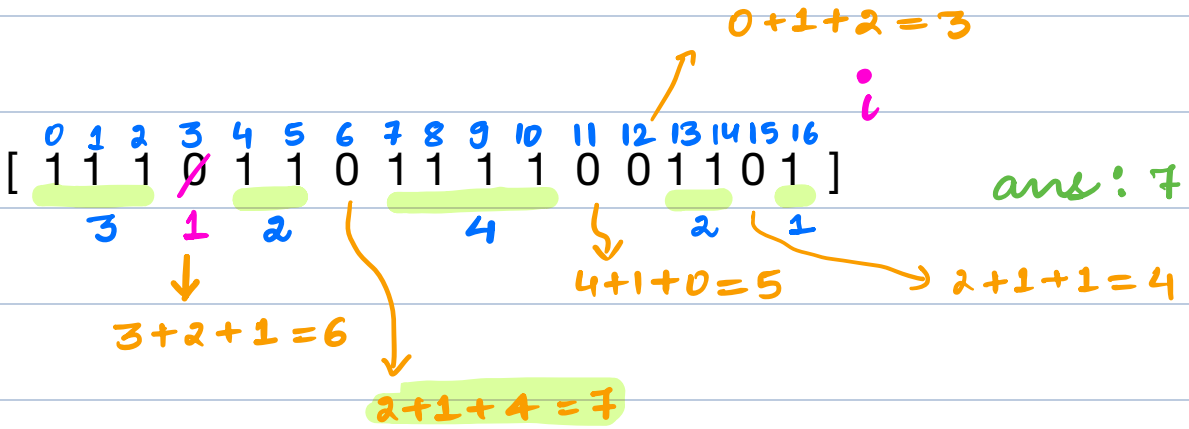
0 1 1 1 1 1 1 0 1 1 0 → 6

0 1 1 1 0 1 1 1 1 1 0 → 5

0 1 1 1 0 1 1 0 1 1 1 → 3

ans : 6

[1 1 1 1 1 1 1] ans : n



Approach :

ans = 0

For every 0 in the array :

count of 1's on left side : l

count of 1's on right side : r

If (l + r + 1) > ans → ans = l + r + 1

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

    int totalOnes = 0, n = arr.length();
    for(i=0; i<n; i++) {                          n iterations
        if (arr[i] == 1)
            totalOnes++;
    }

    if (totalOnes == n)
        return n;

    int ans = 0;
    for(i=0; i<n; i++) {
        if (arr[i] == 0) {
            // count no of consecutive 1's on left
            j = i-1, l = 0;
            while (j >= 0 && arr[j] == 1) {
                l++;
                j--;
            }

            // count no of consecutive 1's on right
            j = i+1, r = 0;
```

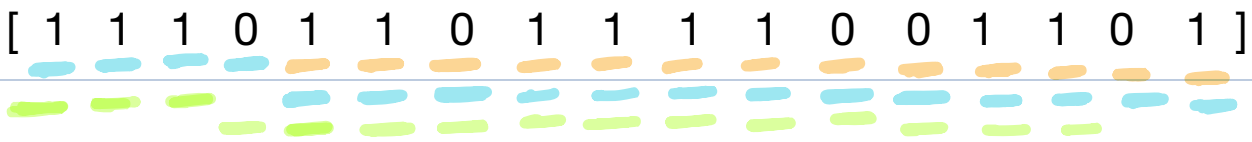
```
        while (j < n && arr[j] == 1) {
            r++;
            j++;
        }

        ans = max(ans, l + r + 1);
    }
}

return ans;
}
```

Quiz 2 :

• loop • Right • Left

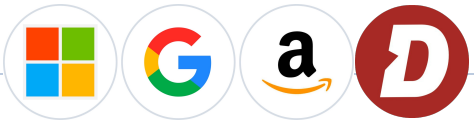


Each element is accessed at max 3 times.

T.C : $O(n + 3n) = O(4n) = O(n)$

S.C : $O(1)$

< Question > : Given a binary array [ ]. We can swap a single 0 with 1. Find the maximum consecutive 1's we can get in the array[ ] after atmost 1 swap.



arr[ ] → [1 1 0 1 1 0 1 1 1] (0 1 2 3 4 5 6 7 8)

<2,8> 1 1 1 1 1 0 1 1 0 → 5 ans: 6

<0,5> 0 1 0 1 1 1 1 1 1 → 6

Quiz 3:

arr[ ] → [1 1 0 1 1 1] (0 1 2 3 4 5)

<2,5> 1 1 1 1 1 0 ans: 5

l = 2

r = 3

l + r = no of 1's in array.

count = l + r

l + r < no of 1's in array

count = l + r + 1

Approach :

count total no of 1's in array : c

ans = 0

For every 0 in the array :

count of 1's on left side : l

count of 1's on right side : r

count = 0 // store length of 1's in current window

if { l + r == c : count = l + r
   { l + r < c : count = l + r + 1

if count > ans

ans = count

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

    int totalOnes = 0, n = arr.length();
    for(i=0; i<n; i++){                          n iterations
        if(arr[i] == 1)
            totalOnes++;
    }

    if(totalOnes == n)
        return n;

    int ans = 0;
    for(i=0; i<n; i++){
        if(arr[i] == 0){
            // count no of consecutive 1's on left
            j = i-1, l = 0;
            while(j >= 0 && arr[j] == 1){
                l++;
                j--;
            }

            // count no of consecutive 1's on right
```

```
        j = i+1 , r = 0 ;
        while (j < n && arr[j] == 1){
            r++;
            j++;
        }

        int count = 0;
        if ( l + r == totalOnes )
            count = l + r;
        else
            count = l + r + 1;    // l+r < totalOnes

        ans = max(ans, count)
    }
}
return ans;
}
```

T.C : O(N)

S.C : O(1)

Break till 8:30

Majority Element

< *Question* > : Given array [N]. Find the majority element

↓

Elements which occurs more than N/2 times.

- *You can assume that majority element always exists.*

eg: [3, 4, 3, 2, 4, 4, 4, 4] (indices 0 1 2 3 4 5 6 7) N = 8 N/2 = 4

freq(3) = 2 freq(4) = 5 freq(2) = 1

ans : 4

[3, 3, 4, 2, 4, 4, 2, 4] (indices 0 1 2 3 4 5 6 7) N=8 N/2=4

freq(3) = 2 freq(2) = 2 no majority element

freq(4) = 4

Quiz

arr[ ] → [3 , 4 , 3 , 6 , 1 , 3 , 2 , 5 , 3 , 3 , 3] (indices 0 1 2 3 4 5 6 7 8 9 10) N = 11 N/2 = 5

ans : 3

Quiz

arr[ ] → [4 , 6 , 5 , 3 , 4 , 5 , 6 , 4 , 4 , 4] (indices 0 1 2 3 4 5 6 7 8 9) N = 10 N/2 = 5

no majority element

Brute Force Solution: Check frequency of each element. If any element has frequency > N/2 i.e. majority element.

```
int majorityElement(int arr[], int n) {
    for(i=0; i<n; i++) {
        int count=0;
        for(j=0; j<n; j++){
            if(arr[j] == arr[i])
                count++;
        }
        if(count > n/2)
            return arr[i];
    }
}
```

T.C : $O(N^2)$

S.C : $O(1)$

Observations :

Quiz :

At max, how many majority elements are possible?

Obs 1 : There will be only 1 majority element in array.

Let's say there are 2 majority elements : m_1 & m_2

$freq(m_1) > N/2$

$freq(m_2) > N/2$

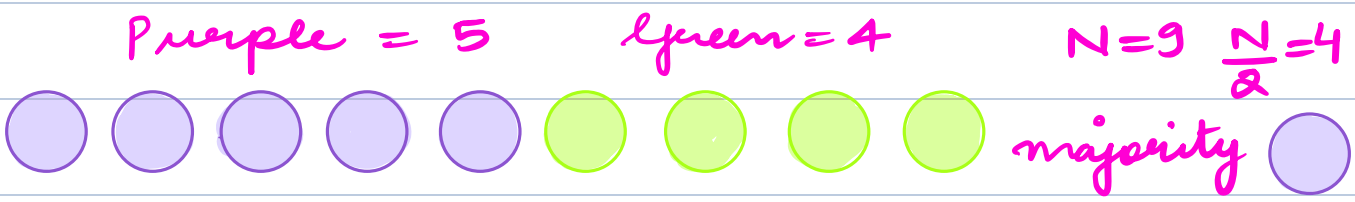
Adding both

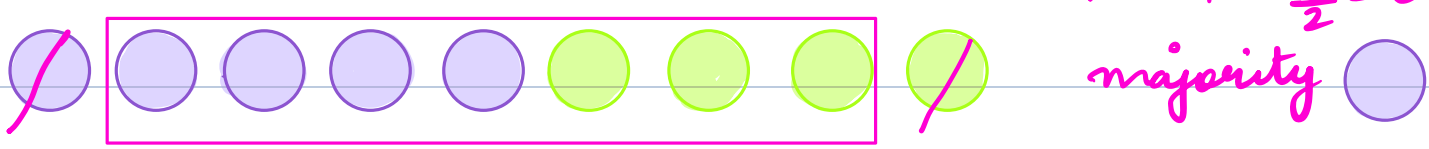
$freq(m_1) + freq(m_2) > \frac{N}{2} + \frac{N}{2}$

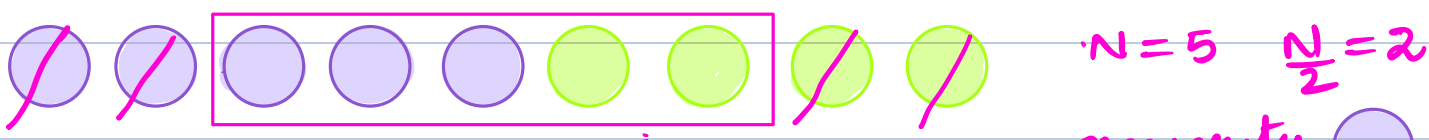
$> N$

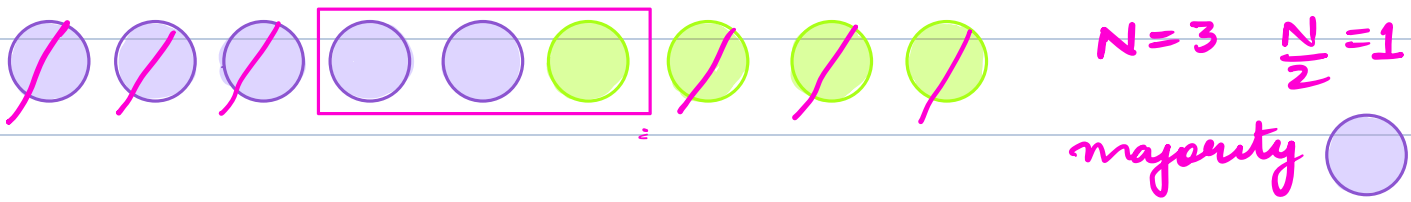
not Possible

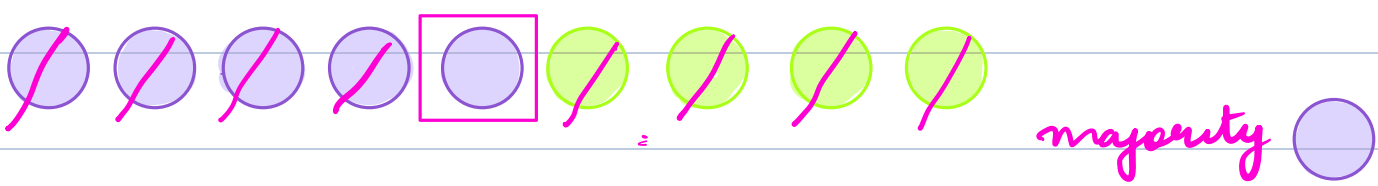
Obs2: If we remove 2 distinct elements, majority remains same.







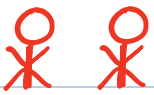




Suppose there are 4 parties participating in election

OP : 

BP : 

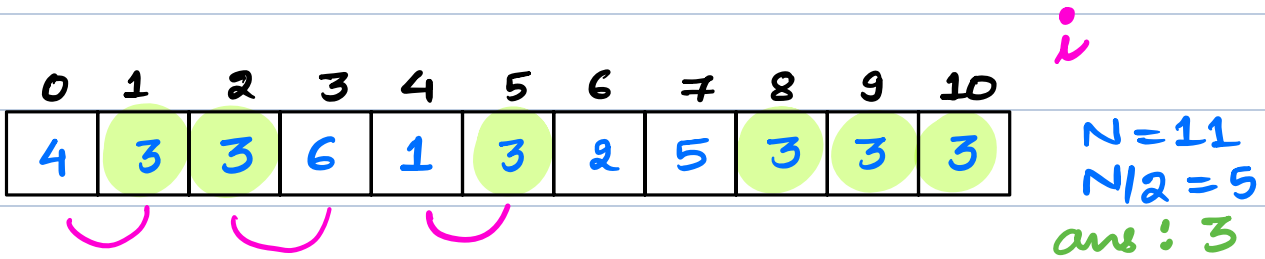
RP : 

GP : 

Remove	Available OP	BP	RP	GP	Winner
	9	3	2	3	OP
1OP & 1BP	8	2	2	3	OP
1OP & 1GP	7	2	2	2	OP
1OP & 1RP	6	2	1	2	OP

After removing 2 distinct party votes, majority remains same.

Moore's Algorithm :



majority element index = ~~0~~ 2 ~~4~~ ~~6~~ 8

count = ~~1~~ 0 ~~1~~ 0 ~~1~~ 0 ~~1~~ 0 ~~1~~ ~~2~~ 3

Returns Possible candidate of majority element.

we have to check if that is actual majority element or not.

If freq > N/2. → MAJORITY ELEMENT

```
int majorityElement(int arr[], int n){
    // Assuming 1st element to be ME.
    int majority_index = 0, count = 1;

    for(i = 1; i < n; i++){

        if(count == 0){
            majority_index = i;
            count = 1;
        }else{

            if(arr[i] == arr[majority_index])
                count++;
            else
                count--;
        }

    }
    // check if majority element has occurence > N/2.

    count = 0;
    for(i = 0; i < n; i++){
```

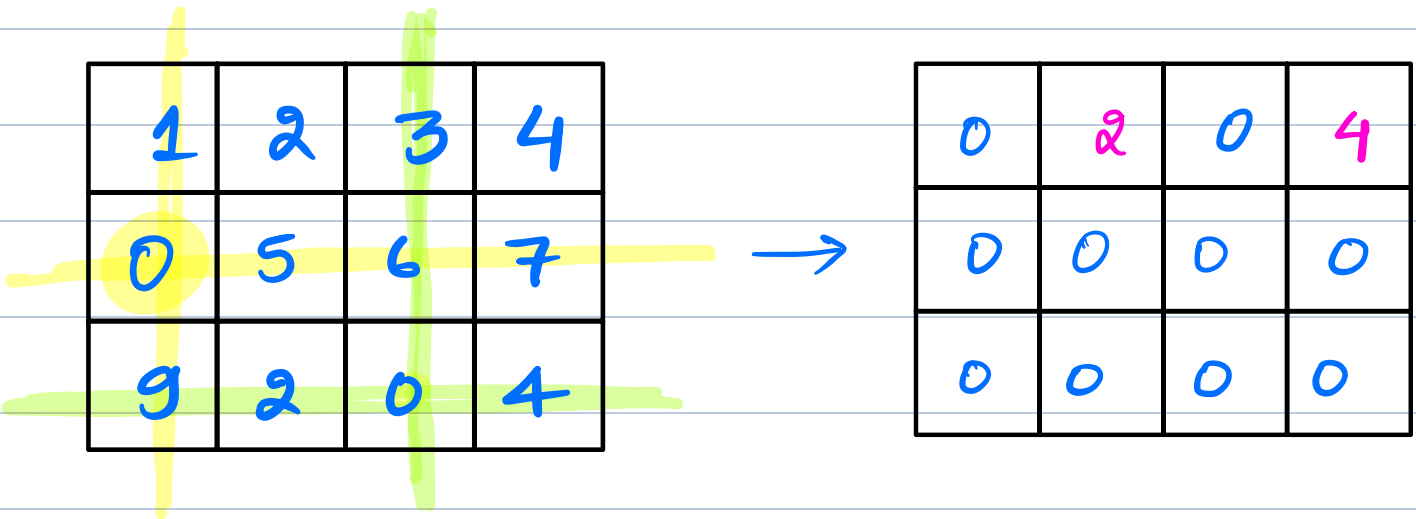
```
    if (arr[i] == arr[majority_index])
        count++;
}

if (count > N/2)
    return arr[majority_index];
else
    return -1; // ME not present
}
```

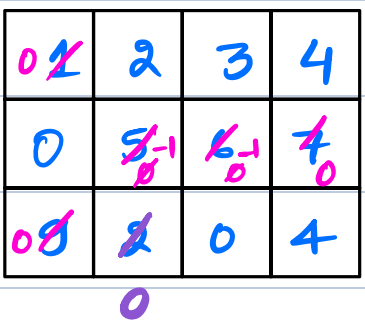
T.C : O(N)
S.C : O(1)

< *Question* > : Given array [N] [M].

Make all elements in a row and column zero if arr[i][j] == 0. Specifically make entire other row and jth column zero. All elements are positive.



Approach :



Issue : If we make elements 0 on the go 0 1 will lose track of original 0.

So instead we can mark elements which needs to be modified first & once we have iterated complete matrix or I can say marked all elements then I can change my marked elements to 0.

< / > Code

```
void transform(int arr[][], int n, int m){

    // Iterate on row & mark col.
    for(i=0; i<n; i++){
        int flag = 0;
        for(j=0; j<m; j++){
            if (arr[i][j]==0)   flag = 1;
        }

        if (flag = 1)
        for(j=0; j<m; j++){
            if([A[i][j] != 0)   A[i][j] = -1;
        }
    }

// Iterate on col & mark rows

    for(j=0; j<m; j++){
        int flag = 0;
        for(i=0; i<n; i++){
```

```
            if (arr[i][j] == 0)   flag = 1;
        }

    if (flag = 1)
    for(j=0; j<m; j++){
        if((A[i][j] != 0)    A[i][j] = -1;
    }
}

// Change all marked elements to 0.
    for(i=0; i<n; i++){
        for(j=0; j<m; j++){
            if (arr[i][j] == -1)
                    arr[i][j] = 0;
        }
    }

}
```

T.C : (3 N*M) = N*M

S.C : O(1)

Doubts:

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

N = 9 N/2 = 4

i

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

N = 11 N/2 = 5

ans : 3

Mi = ~~0~~

~~2~~

~~4~~

~~6~~

~~8~~

10

count = ~~0~~ 0

~~1~~ 0

~~2~~ 0

~~1~~ 0

~~2~~ 0

1