Q1 Given N array elements, print start and end index of all subarrays of size k.

Index :-	0	1	2	3	4	5	6	7	8	9	10	11
arr[] :-	3	4	2	-1	6	7	8	9	3	2	-1	4

K = 3

[0, 2]
[1, 3]
[2, 4]
[3, 5]
...
[9, 11]
↓
[12-3, 11]

K = 6

[0, 5]
[1, 6]
[2, 7]
[3, 8]
...
[6, 11]
[12-6, 11]

In General

K

s e

[0, K-1]
[1, K]
[2, K+1]
[3, K+2]
...
[N-K, N-1]

start end

[N-K+1, N] (Invalid)

How many subarrays are there with length K.?

s = 0, s = N-K

[0, N-K]

⇒ N-K-0+1

⇒ N-K+1

# of iterations = N-K+1

TC: O(N)

```
s = 0, e = k-1
while(end < N.) {   // s <= N-K
    print(s, e);
    s++;
    e++;
}
```

Q2) Given N array elements, find max subarray sum of **len = k**.

index :-	0	1	2	3	4	5	6	7	8	9
arr[] :-	-3	4	-2	5	3	-2	8	2	-1	4

K = 5

S	e	sum
0	4	7
1	5	8
2	6	12
3	7	16
4	8	10
5	9	11

ans = 16

Range Sum

K

(N-K+1)

```
S = 0 , e = K-1
ans = INT_MIN          // S <= N-K
while ( end < N ) {
    sum = 0
    for(i = S; i <= e; i++){        [S, e]
        sum = sum + arr[i];
    }
    ans = max (ans, sum);
    S++;
    e++;
}
return ans;
```

Idea : For every subarray of size K, get its sum and get overall max.

(N-K) * K => N * K

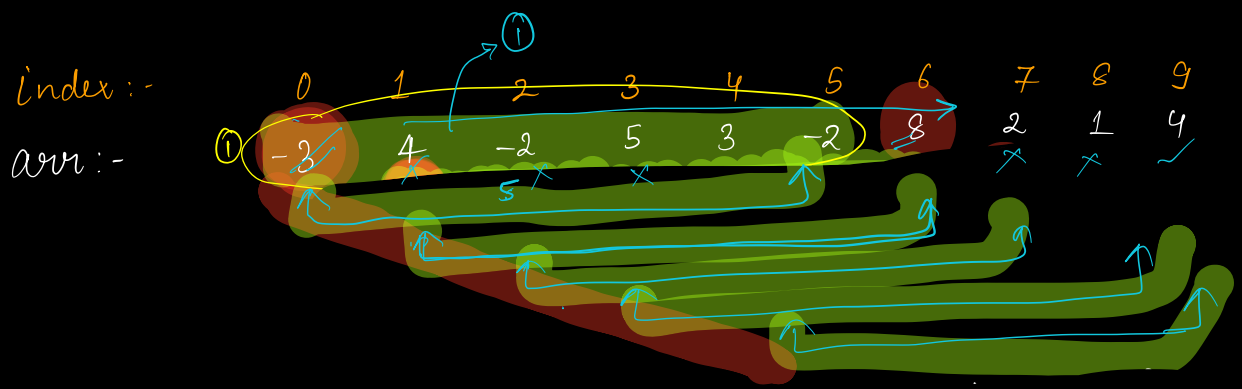
# of iterations = (N-K+1) * K

TC : O(N^2)

K = 1	K = N	K = N/2
↓	↓	$(N - \frac{N}{2} + 1) * \frac{N}{2}$
N	N	$\Rightarrow (\frac{N}{2} + 1) * \frac{N}{2}$
		$\Rightarrow (N) * (N) \Rightarrow N^2$

Optimization 1: Build PS array & answer each query.

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



K = 6

sum

$\Rightarrow [0, 5] = 5 \Rightarrow sum$

$\Rightarrow [1, 6] = 5 - \underbrace{arr[0]}_{s-1} + \underbrace{arr[6]}_{e} = 16 \Rightarrow sum$

$[\underbrace{2}_{s}, \underbrace{7}_{e}] = \underbrace{16 - arr[1]}_{s-1} + \underbrace{arr[7]}_{e} = 14 \Rightarrow sum$

$[\underbrace{3}_{s}, \underbrace{8}_{e}] = 14 - \underbrace{arr[2]}_{s-1} + \underbrace{arr[8]}_{e} = 17 \Rightarrow sum$

$[4, 9] \Rightarrow 17 - \underbrace{arr[3]}_{s-1} + \underbrace{arr[9]}_{e} = 16$

ans = 17

```
sum = 0
for(i = 0 ; i < k ; i++){ || sum for first subarray.
    sum = sum + arr[i];
}
ans = sum
s = 1
e = k || second subarray.
while(e < N){
{ => sum = sum - arr[s-1] + arr[e]; }
    ans = max(ans, sum);
    s++;
    e++;
}
```

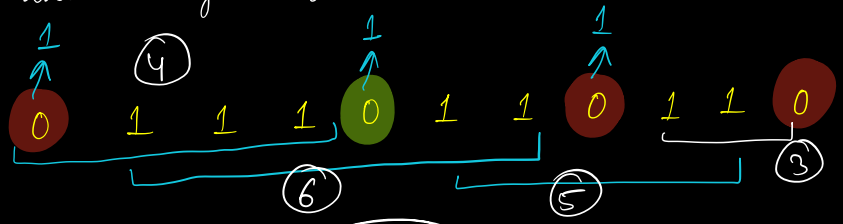
// sum is the sum for the last subarray

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

Q3) Given a binary array. We are allowed to replace atmost one 0 with a 1.

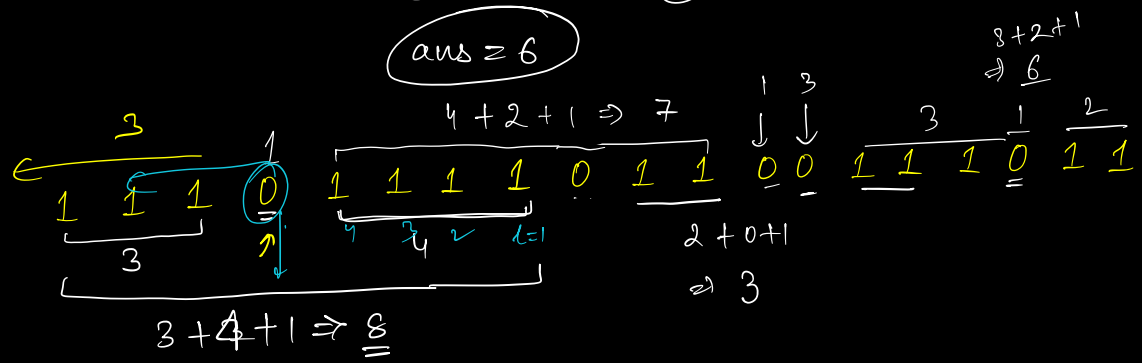
Return the length of max consecutive 1s.

Ex1



ans = 6

Ex2



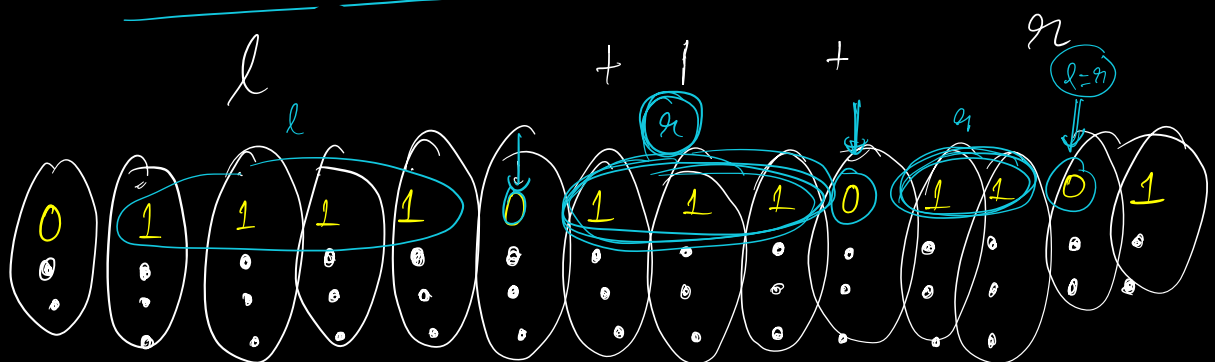
arr: [1 , 1 , 1 , 1 , 1 , 1] ⇒ ans = 6

6

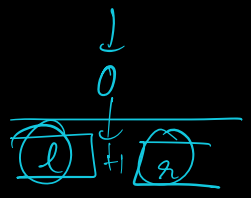
// Observation:

For every 0, length of consecutive 1's after removing that 0 ⇒

No of consecutive 1's on left side + 1 + No of consecutive 1's on right side.



```
//Pseudo Code.
ans = 0
for(i = 0; i < N; i++){
    if(arr[i] == 0){
        //l = length of consecutive 1's on left side
        l = 0
        for(j = i-1; j >= 0; j--){ //left side
            if(arr[j] == 1) l++;
            else break;
        }
        //r = count of 1's on right side.
        r = 0
        for(j = i+1; j < N; j++){ //right side
            if(arr[j] == 1) r++;
            else break;
        }
        ans = max(ans, l + r + 1);
    }
}
if(ans == 0) ans = N;
return ans;
```



# of iterations = 3N

TC : O(N)

6 Min

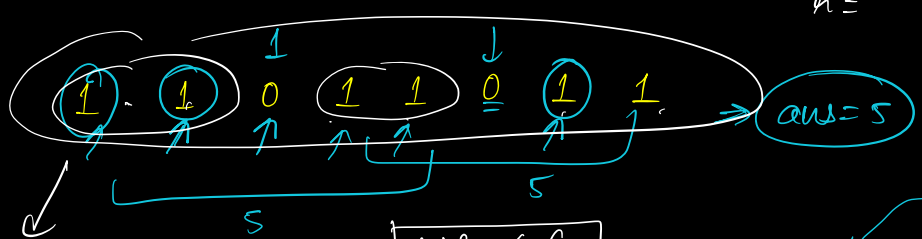
Q 4) Given a binary array. We are allowed to swap atmost one 0 with a 1.

Return the length of max consecutive 1s.

l = l+r+1 (if l+r < C)

r =

Ex



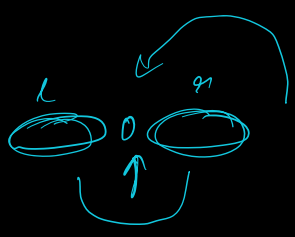
⇒

C = 6

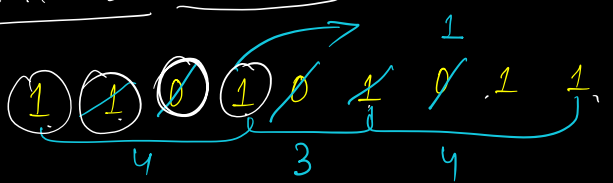
l = 2, r = 2

l + r = 4

l + r < C



Ex



⇒ ans = 4

l = 2

r = 1

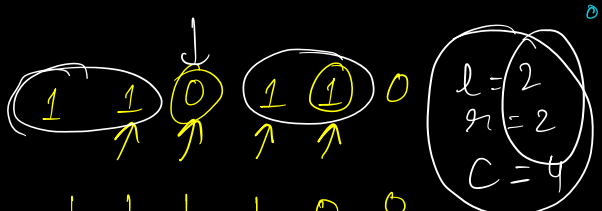
C = 6

l + r < C

if you are at 0,

① Try to swap that with a 1 other than its left & right 1's

Ex



l = 2
r = 2
C = 4

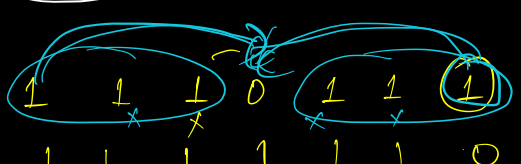
l + r + 1

⇒ ans = 4

② l + r

l + r = 4
C = 4

Ex



⇒ ans = 6 , l + r

l = 3
r = 3
C = 6

l + r = C

l + r < C

ans = l + r + 1

l + r == C

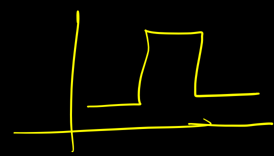
ans = l + r

→ C = total count of 1's in entire array (N)

```
ans = 0
for(i = 0; i < N; i++) {
    if(arr[i] == 0) {
        // l = length of consecutive 1's on left side
        l = 0
        for(j = i-1; j >= 0; j--) { // left side
            if(arr[j] == 1) l++;
            else break;
        }
        // r = count of 1's on right side.
        r = 0
        for(j = i+1; j < N; j++) { // right side
            if(arr[j] == 1) r++;
            else break;
        }
        if(l+r < C)   len = l+r+1;
        else len = l+r
        ans = max(ans, len);
    }
}
if (ans == 0) ans = N;
return ans;
```

6 Min

Code → Servers → 10^9 cc/seconds



1 cc : 1 instruction

10^9 instruction/second

In general, Time Limit :→ 1 second

:→ Our code can have atmax : 10^9 instructions

```
for(i=0; i<N; i++){
    if(i%2==0)
        print(even)
    else
        print(odd)
}
```

# of iterations = N iterations

In 1 iterations ⇒ 4 instructions

In N iterations ⇒ 4N instructions

4N

Max $4 * 10^5$

Constraints :-

$1 \le N \le 10^5$

Atleast 1

Atmax 10^5

Assumption1 :- 1 Iteration = 10 Instruction

At max, we can have 10^9 instructions,

$\Rightarrow 10^8 \times 10$ instructions

$\Rightarrow$ At max, we can have 10^8 iterations

1 Iteration = 100 Instructions

Assumptions2 :-

At max, we can have 10^9 instructions,

$\Rightarrow 10^7 \times 100$ instructions

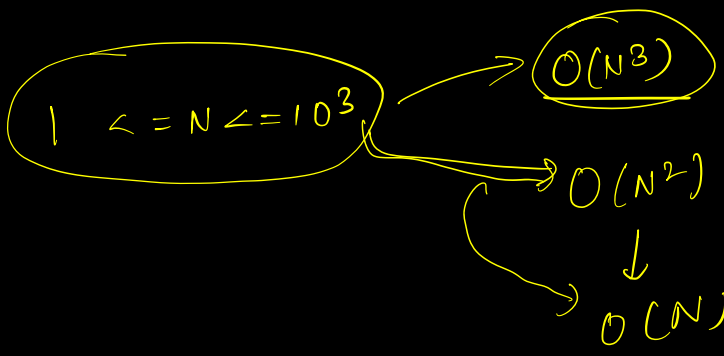
$\Rightarrow$ At max, we can have 10^7 iterations

In general, our code should have $10^7 - 10^8$ iterations

Max Subarray Sum :-

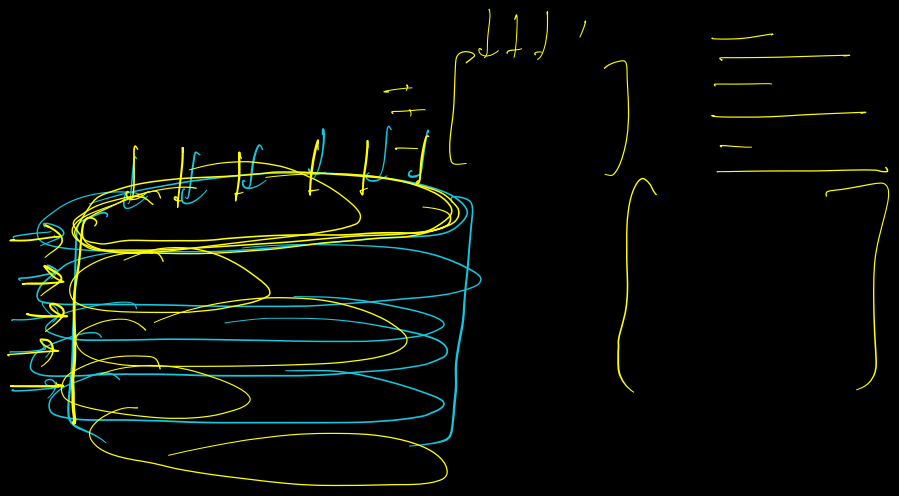
$1 <= N <= 10^6$ → $O(N^2)$ $\Rightarrow 10^{12}$

$O(N)$



// Doubts:-

arr



rows = arr.length

col = arr[0].length

N*M

leader N [ ]

leader[i] = 1

= 0

[0 1]

[1 1]

Not a leader

leader

(3) → Bit Manipulations