Q Given an integer array of size N & an integer Q.

You will be asked Q no. of queries on this array.

Query ⇒ l, r ⇒ Print the sum of elements of the array from index l to r (including both)

Eg: A = { -3, 6, 8, 4, 5, 8, -9, 3, 1 }
(indices: 0 1 2 3 4 5 6 7 8)

Q = 5

l	r	Sum
4	8	8
3	7	11
1	3	12
0	4	14
7	7	3

1) Brute force

∀ queries ⇒ Iterate from l to r & calculate sum.

I/P ⇒ N

A[N]

Q

Query[Q][2] → 0 1

— l r

—

Code

```
void querySum (Query[Q][2], A[N]) {
    for (i=0; i<Q; i++) {          // O(Q)
        L = Query[i][0];
        R = Query[i][1];

        int sum = 0;
        for (j=L; j<=R; j++) {     // O(N)
            sum = sum + A[j];
        }
        print (sum);
    }
}
```

T.C. = O(Q×N)

S.C. = O(1)

Const: $1 \le N \le 10^5$; $1 \le Q \le 10^5$

$Q \times N = 10^{10} \Rightarrow TLE$

2) Optimise

Given the score of first 10 overs of a cricket match

1	2	3	4	5	6	7	8	9	10
2,	8,	14,	29,	31,	49,	65,	79,	88,	97

1) 7th Over [7, 7] — Score[7] − Score[6] = 65 − 49 = 16.

2) 6th to 10th Over [6, 10] — Score[10] − Score[5] = 97 − 31 = 66

3) 10th Over [10, 10] — Score[10] − Score[9] = 97 − 88 = 9

4) 3rd to 6th Over [3, 6] — Score[6] − Score[2] = 49 − 8 = 41

⇒ Score is a cumulative data

⇒ Calculating range query sum is very optimal using cumulative array.

⇒ This cumulative array is known as Prefix Sum Array.

⇒ If we create a similar cumulative array for above problem, we will also be able to answer each query in constant time.

Create Prefix Sum Array

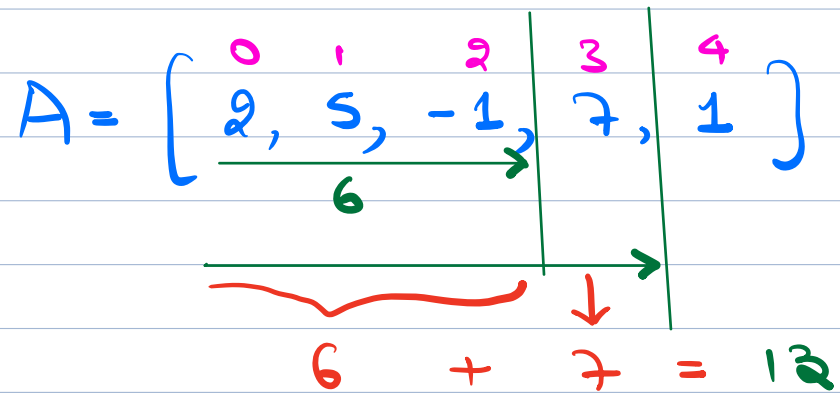
Prefix[i] ⇒ Sum of all elements from index 0 to i (including)

$$A = \begin{bmatrix} \overset{0}{2}, & \overset{1}{5}, & \overset{2}{-1}, & \overset{3}{7}, & \overset{4}{1} \end{bmatrix}$$

$$Prefix = [2, 7, 6, 13, 14]$$

Prefix[0] = Sum[0,0] = A[0]
Prefix[1] = Sum[0,1] = A[0]+A[1]
Prefix[2] = Sum[0,2] = A[0] + A[1] + A[2]

Prefix[3] = Sum[0,3] = A[0] + A[1] + A[2] + A[3]



Prefix[3] = Prefix[2] + A[3]

Prefix[i] = Prefix[i-1] + A[i];

i = 0 → Prefix[-1] ?!

Code

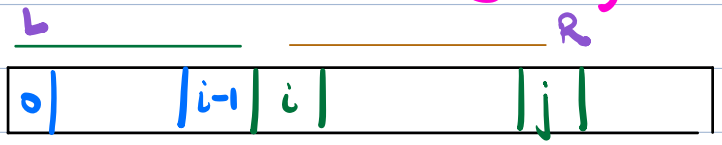
```
Prefix[N];

Prefix[0] = A[0];
for (i = 1; i < N; i++) {

    Prefix[i] = Prefix[i-1] + A[i];

}
```

T.C. = O(N)



Sum[i, j] = Prefix[j] - Prefix[i-1]

if i == 0, Prefix[i-1] = ~~Prefix[-1]~~

Sum[0, j] ⇒ Prefix[j]

```
void querySum (Query[Q][2], A[N]) {

    Prefix[N]; // S.C. = O(N)

    Prefix[0] = A[0];
    for (i = 1; i < N; i++) {          // O(N)

        Prefix[i] = Prefix[i-1] + A[i];
    }

    for (i = 0; i < Q; i++) {          // O(Q)
        L = Query[i][0];
        R = Query[i][1];

        sum;
        if (L == 0) {
            sum = Prefix[R];
        }
        else {
            sum = Prefix[R] - Prefix[L-1];
        }
```

print (sum);

}

}

T.C. = O(N + Q)

S.C. = O(N)

↓

O(1) ???

(Modify I/P)

	0	1	2	3	4
A =	2,	~~5~~ 7,	~~-1~~ 6	~~7~~ 13,	~~1~~ 14.

Code

```
for (i=1; i<N; i++) {
    A[i] = A[i-1] + A[i];
}
```

Given an array of size N & Q queries

Query ⇒ s, e

for every query calculate the sum of even indexed elements in the range from index s to e.

	0	1	2	3	4	5
A =	2	3	1	6	4	5

Q = 4

s	e	Sum of even indexed
1	3	1
2	5	5
0	4	7
3	3	0

Sol^{n} 1) Brute force.

```
void querySum (Query[Q][2], A[N]) {

    for (i=0; i<Q; i++) {
        L = Query[i][0];
        R = Query[i][1];

        Sum = 0;
        for (j=L; j<=R; j++) {

            if (j%2 == 0) {
```

```
            sum = sum + A[j];
        }
    }
    print (sum);
}
```

T.C. = $O(Q \times N)$

	0	1	2	3	4	5
A =	2	3	1	6	4	5
Prefix	2	2	3	3	7	7
Sum =	0	~~2~~	~~2~~	~~3~~	~~3~~	7

2) Optimise

```
void querySum (Query[Q][2], A[N]) {

    Prefix [N];
    Prefix [0] = A[0];
    for (i = 1; i < N; i++) {
        if (i % 2 == 0) { Prefix [i] = Prefix [i-1] + A[i]; }
        else { Prefix [i] = Prefix [i-1] ;
    }
}
```

```
for (i=0; i<Q; i++){
    L = Query[i][0];
    R = Query[i][1];
    sum;
    if (L==0){
        sum = Prefix[R];
    }
    else {
        sum = Prefix[R] - Prefix[L-1];
    }
    print (sum);
}
}
```

T.C. = $O(N+Q)$

Sum of all odd indexed elements in a range ??

```
Prefix[0] = 0;
```

Given an integer array of size N. Count the no. of special index in the array.

1) Understand the problem
   ↳ Ask clarifying questions.

Special Index : That index after removing which the sum of odd indexed elements & the sum of the even indexed elements is equal.

$$A = \underset{0}{[4}, \underset{1}{3}, \underset{2}{2}, \underset{3}{7}, \underset{4}{6}, \underset{5}{-2}]$$

Remove	A	S_e	S_o	
0	[3, 2, 7, 6, -2]	8	8	✓
1	[4, 2, 7, 6, -2]	9	8	✗
2	[4, 3, 7, 6, -2]	9	9	✓
3	[4, 3, 2, 6, -2]	4	9	✗
4	[4, 3, 2, 7, -2]	4	10	✗
5	[4, 3, 2, 7, 6]	12	10	✗

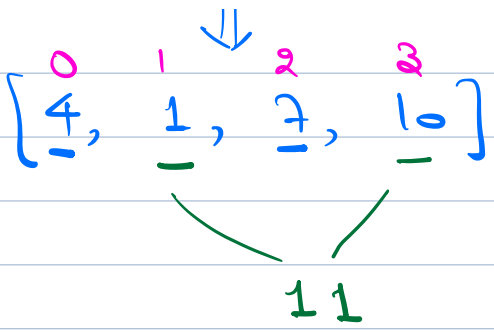
Ans = 2

Sum of Even after removing 2

$$[4, 1, \cancel{3}, 7, 10]$$

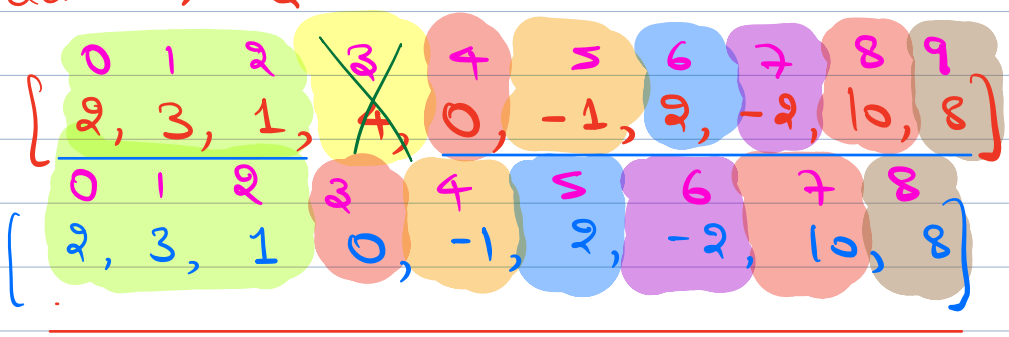
=

11



Q

Odd ⇒ 3



Before Index 3 ⇒ No change in index

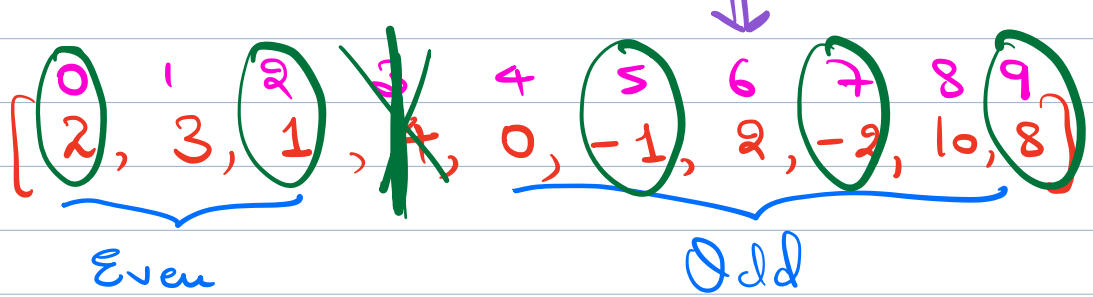
After Index 3 ⇒ Updated Index = Original Index − 1

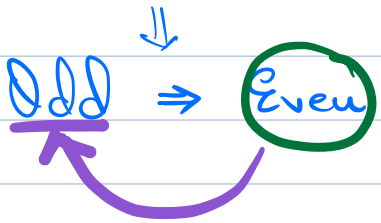
Even ⇒ Odd
Odd ⇒ Even.

Q

Even ⇒ 3

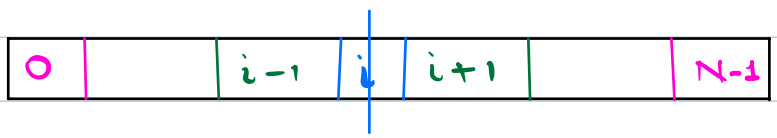
Ans = 8





Generalise

for every index i



Sum of Even indexed Elements after deleting index i = $Sum_{even}[0, i-1]$ + $Sum_{odd}[i+1, N-1]$

$P_{even}[i-1]$

$P_{odd}[N-1] - P_{odd}[i]$

Sum of Odd indexed Elements after deleting index i = $Sum_{odd}[0, i-1]$ + $Sum_{even}[i+1, N-1]$

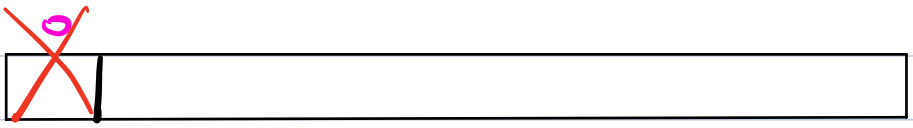
$P_{odd}[i-1]$

$P_{even}[N-1] - P_{even}[i]$

Even == Odd

$P_{even}[i-1] + (P_{odd}[N-1] - P_{odd}[i])$ == $P_{odd}[i-1] + P_{even}[N-1] - P_{even}[i]$

i = 0 i = N-1



Code

1) Create Peven array & Podd array.
O(N) O(N)

```
count = 0;
for (i=0; i<N; i++) {  // O(N)
    int So, Se
    if (i==0) {
        So = Peven[N-1] - Peven[i];
        Se = Podd[N-1] - Podd[i];
    }
    else {
        So = Podd[i-1] + Peven[N-1] - Peven[i];
        Se = Peven[i-1] + Podd[N-1] - Podd[i];
    }
    if (So == Se) {
        count ++;
    }
}
```

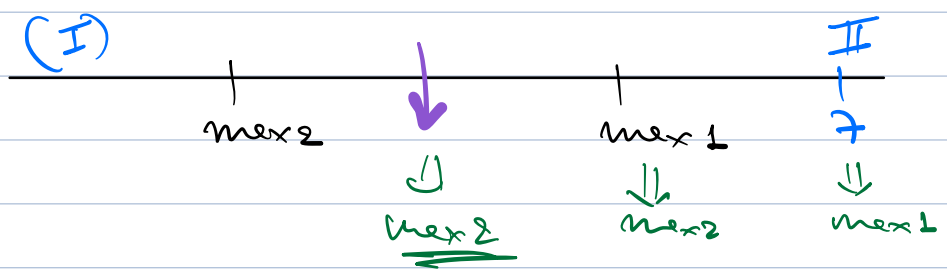
O(1)

T.C. = O(N) + O(N) + O(N)
= O(N)
S.C. = O(N)

Next ⇒ Carry forward.
⇒ ~~XOR~~ Subarrays.

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

max 1 = max = 6 = max (0, 1)
max 2 = 2nd max = 5 = min (0, 1)



A = [8, 5, 6, 2, 4, 1, 9, 1]

#Elements − (count of Max)

Max