Content

1) Prefix Sum
2) Problems Based on Prefix Sum
3) Carry forward Problems
4) Why TLE Occurs?

1Q)

	0	1	2	3	4	5	6	7	8	9
Given ar[10]:	-3	6	2	4	5	2	8	-9	3	1

Q queries

For each query {L-R}, find sum of all elements from L-R

Q = 5 Sum

4-8 : 9
2-7 : 12
1-3 : 12
0-4 : 14
7-7 : -9

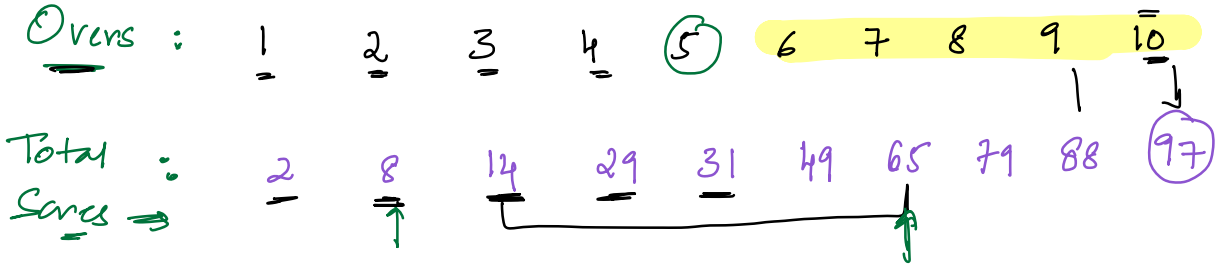
Idea: For every query iterate & get the sum

```
while(Q--){
    // sum = 0
    for(i = L; i <= R; i++){
        sum = sum + ar[i]
    }
    print(sum)
}
```

O(N)

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

// Given India Team Score for first 10 Overs of Batting



Q_1: Runs Scored in last Over : 1 : 97 - 88

[10, 10] : S[10] - S[9]

Q_2: Runs Scored in last 5 Overs : 97 - 31 = 66

[6 10] : S[10] - S[5]

Q_3: Runs Scored in 7th Over : 65 - 49 = 16

[7 7] : S[7] - S[6]

Q_4: Runs Scored from 3rd to 7th Over

both included.

[3 7] : S[7] - S[2]

	0	1	2	3	4	5	6	7	8	9
Given ar[10] :	-3	6	2	4	5	2	8	-9	3	1
prefix[] :	-3	3	5	9	14	16	24	15	18	19

// cummulative sum

prefix[i] : Sum of all array elements [0 i]

prefix[3] : Sum of all elements from [0 3]

Query:

4-8 : Pf[8] - Pf[3]

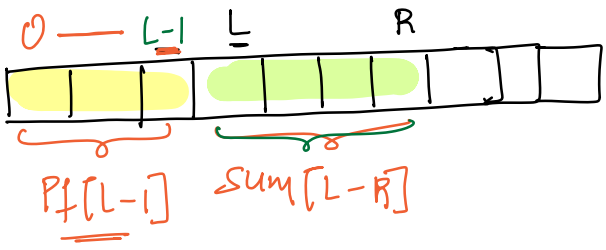
{0, 1, 2, 3} {4, 5, 6, 7, 8}

Pf[3]

3-5 : Pf[5] - Pf[2]

{0, 1, 2} {3, 4, 5}

Query : L<=R



L R

0 - 4 : Pf[4] - Pf[-1]

error

0-4 : Pf[4]

Sum[L-R] =

```
if(L==0) { Pf[R] }
else
  Pf[R] - Pf[L-1]
```

Calculate pf[]

Pf[i-1] = Sum of all [0, i-1]

```
pf[0] = ar[0]

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

N Iterations

TC: O(N)

SC: O(N)

pf[i] = 0, 1, 2, .. i-1, i

pf[i] = pf[i-1] + ar[i]

```
// Answer Q Queries

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

Q Iterations

Total = (N+Q)

TC: O(N+Q)

SC: O(Q+N)

// TODO: use given ar[] to store pf[] & answer queries.

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

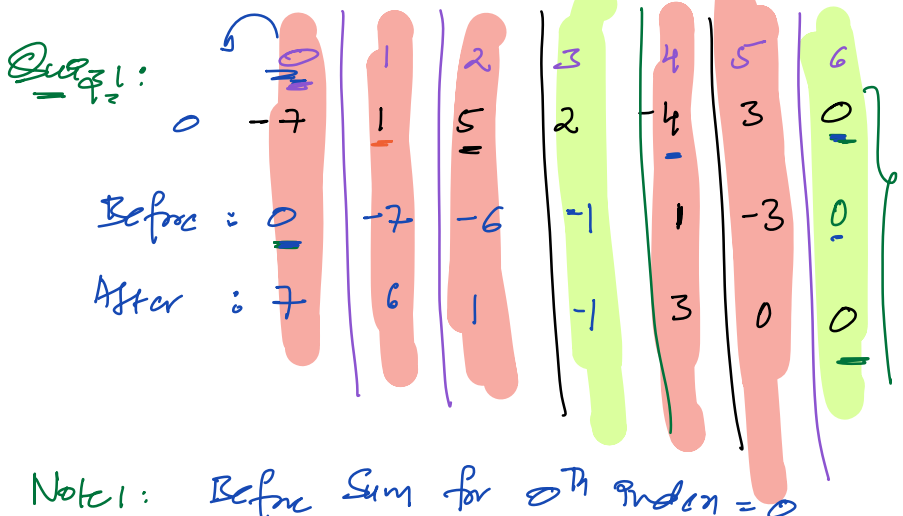
// 2nd

Given N array elements, count no: of **Equilibrium Index**

An index **i** is said to be equilibrium index if

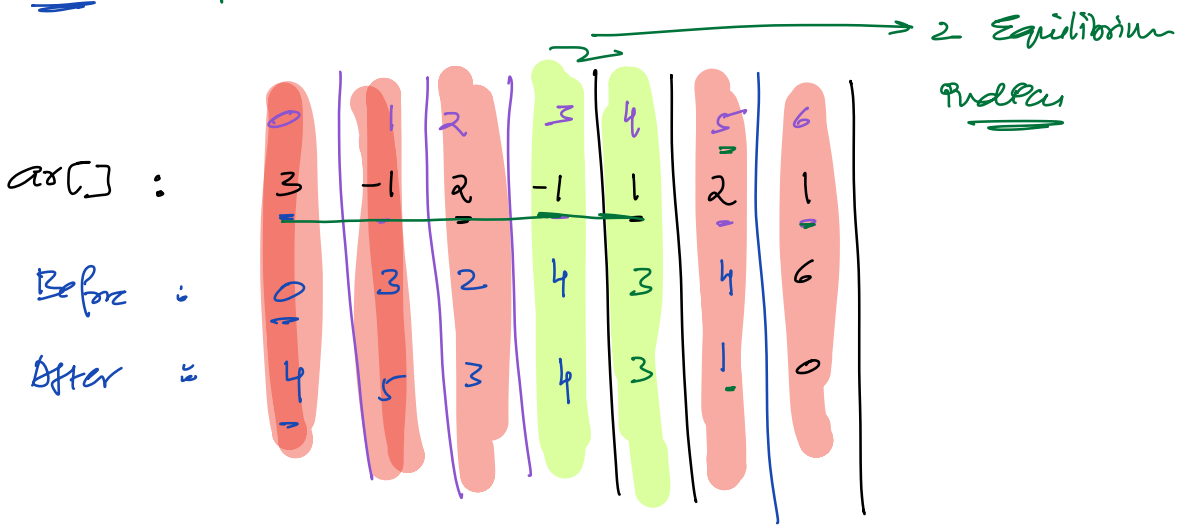
Sum of all Elements Before i^{Th} Index = Sum of all Elements after i^{Th} Index

$[0, i-1]$ = $[i+1, N-1]$



Note1: Before Sum for 0^{Th} Index = 0

Note2: After Sum for $N-1^{Th}$ Index = 0



// Idea: For every index

Check equilibrium or Not

// Build pf[] → TC: O(N), SC: O(N)

// For q == 0, check equilibrium or Not

```
for(q = 1; q < N; q++){
    // For index q
    left = sum of all [0, q-1]
    left = pf[q-1]
    Right = sum of all [q+1, N-1]
    Right = pf[N-1] - pf[q]
    if(left == Right){
        cout << q;
    }
}
```

TC: O(N)

if q == 0 corner

L R

if q == N-1
Right = pf[N-1] - pf[N-1]
// working

How to check if index 0 Equilibrium or Not?

left = 0

Right = [1, N-1] : pf[N-1] - pf[0]

if(pf[N-1] - pf[0] == 0) { cout << 0 }

return cut;

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

TODO: Do it without extra space: 10:35 PM

29) → Advanced Session

	0	1	2	3	4	5	6	7	8
ar[] :	3	2	4	1	2	-1	3	0	1
leftM[] :	3	3	4	4	4	4	4	4	4

leftM[i] = Max of all array elements [0 - i]

leftM[i] = Max of all { 0, 1, 2 .. i-1, i }

leftM[i] = Max(leftM[i-1], ar[i]) } [0, 1, .. i]

leftM[i] = max(leftM[i-1], ar[i])

leftM[0] = ar[0]

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

  leftM[i] = max(leftM[i-1], ar[i])

}

i-1 → i

i==0 Error

TC: O(N)

SC: O(N)

// Max of (2 - 5) : leftM[5] - leftM[1] ✗

Max of [2 - 5] : Max(leftM[5], leftM[2]) ✗

Max of [2 - 5] : leftM[5] ✓

Ex:

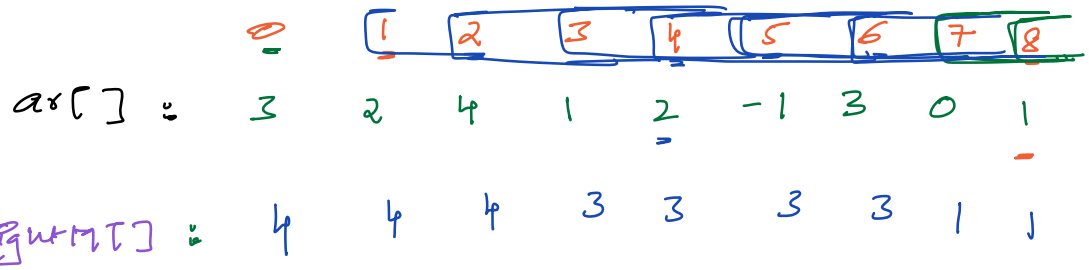
	0	1	2	3	4	5
ar[6] :	8	8	4	3	2	9
left[6] :	8	8	8	8	8	9

Max value from [2-4] = left[4] ~~X~~

// max(a, b) : It will return max a, b
// min (a, b) : It will return min a, b
} both predefined functions, you can use.

// Can use sorting as well

// Only basic maths & sorting



RightM[i] = Max of all Elements from [i, N-1]

RightM[i] = Max of all Elem [i, i+1, i+2, -- N-1]

RightM[i] = Max(ar[i], RightM[i+1])

i → i+1 (R to L)

RightM[N-1] = ar[N-1]

i = N-2 ; i >= 0 ; i--) {

RightM[i] = Max(ar[i], RightM[i+1])

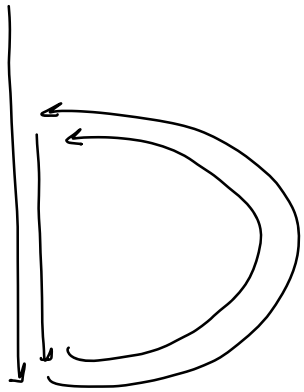
}

if i == N-1, i+1 > N

RightM[N] // Array of Zeroes

Why TLE? (Time limit Exceeded)

1) Question
2) logic in your mind
   Pseudocode
3) Code
4) Time limit Exceed



Online / Flat-forms /

Your Code 1 sec

We Execute our code on Their Servers → Execute 1 sec → 1 GHz

{ 1 GHz = 10^9 instructions / sec }

// our code should <= 10^9 instructions

```
for(i=0; i<100; i++){
    {print(i)}
    {print(i+1)}
}
```

Iterations: 100

In Each iteration ⇒ 4 instructions

Total ⇒ 400 instructions

// Lower Assumption

1 Iteration = 10 instructions

// our code should <= 10^9 instructions

// our code should $x = 10^8 * 10$ instructions

// our code should $x = 10^8$ iterations

// Higher

1 Iteration = 100 instructions

// our code should <= 10^9 instructions

// our code should <= $10^7 * 10^2$ instructions

// our code should $x = 10^7$ iterations

// our code should ≈ $[10^7 - 10^8]$ iterations

Importance of Constraints

N - array size

$1 \le N \le 10^5$

Nested loop logic

$O(N^2) \Rightarrow N = 10^5$

$O(N^2) \Rightarrow$ Iteration $\approx 10^{10}$ } TLE

We have to optimize

$O(N)$

$N = 10^5$

Iterate $= 10^5$ ✓

$O(N \log N) \Rightarrow N = 10^5$

// When we use sorting library

$10^5 * \log_2 10^5 \approx 10^5 * 17 \approx 1.5 * 10^6$ ✓

Sorting

// Sum of array Elements

Constraints :

$\begin{cases} 1 \le N \le 10^5 \\ 1 \le ar[i] \le 10^6 \end{cases}$ $\left.\begin{matrix} N=1 \\ ar[0]=1 \end{matrix}\right\}$ 1

$\therefore$ Sum $= \begin{cases} N = 10^5 \\ \text{Each Ele} \\ 10^6. \end{cases}$

$\Rightarrow 10^{11}$

Put Sum = 0; long Sum = 0

Sum $\Rightarrow [1, 10^{11}]$ ✗

```
for(i=0; i<N; i++){
    Sum = Sum + ar[i]   => overflow
}
return sum;
```

1) During constraints you can also get datatype of variables

2) Use constraints TLE or not.