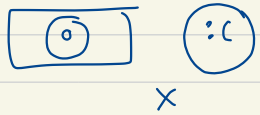
Welcome

- Prefix Sums
  - ↳ Technique

Camera

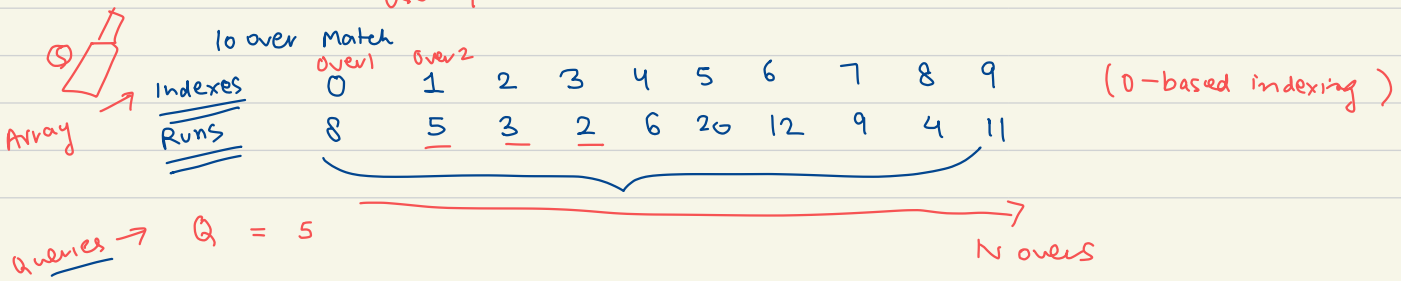


IPL

Motivation Problem

Given a list of N overs, and number of runs scored in each over. So your task is to compute the total runs scored between [L,R]. You will be multiple queries (Q) each query containing two numbers [L,R], you have to find how many runs were scored in in the overs starting from starting from L till R.

User input



L	R	Runs
1	3	5+3+2 = 10 Runs
4	8	6+20+12+9+4 = 51 Runs
0	2	8+5+3 = 16 Runs
6	7	12+9 = 21 Runs
5	5	20 = 20 Runs

L → R

Sum of (arr[L] + ... + arr[R]) for each Query

- Read Q

Q →

- for (int q=1 ; q<=Q ; q++){

Q · N

Read L, R; // for current Query

sum = 0

for (i = L; i<=R, i++){ → worst case N

sum = sum + arr[i];

} finding sum for every query

}

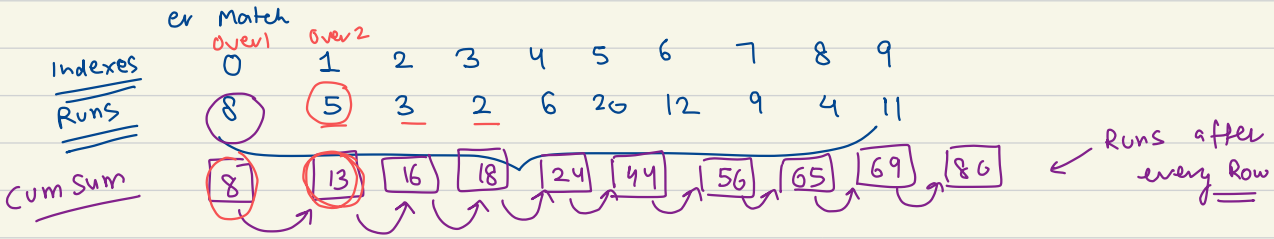
print(sum);

Time Complexity → O(2N)

Space Complexity → O(1)

Optimisation
- Prefix Sums

Given a list of overs, and number of runs scored in each over. So your task is to compute the total runs scored between [L,R]. You will be multiple queries (Q) each query containing two numbers [L,R], you have to find how many runs were scored in in the overs starting from starting from L till R.



Problem -> I want to see the total runs on the leaderboard after each over.

Part-I

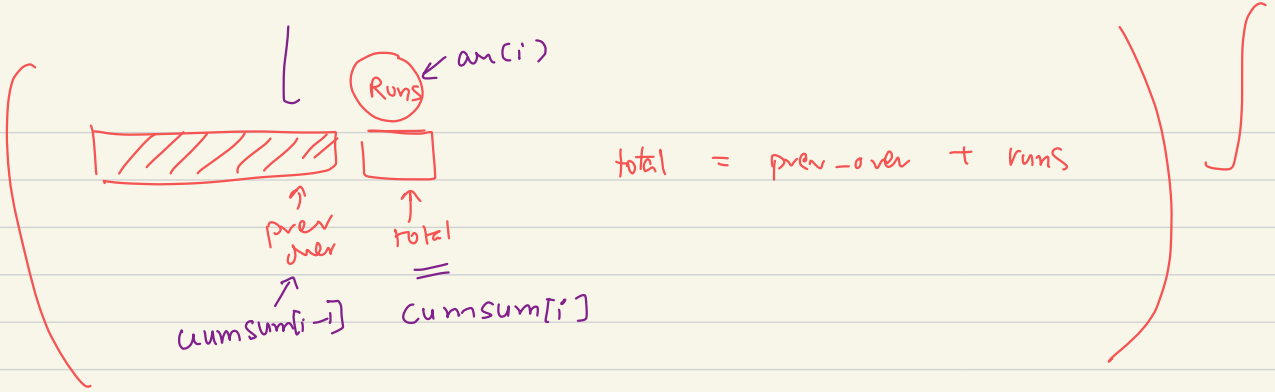
O(N)

Extra Space

```
int cumSum[N]; // int 0
cumSum[0] = arr[0]
for(i=1; i<=N-1; i++) {
    => cumsum[i] = cumsum[i-1] + arr[i];
}
```

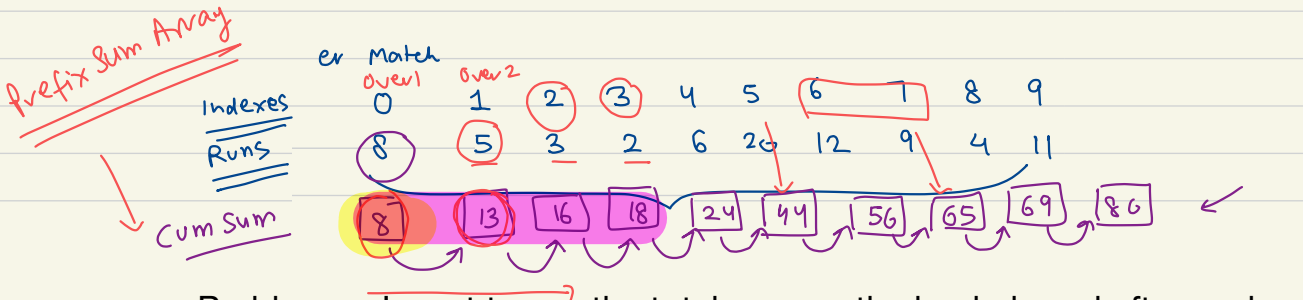
Time => O(N)

Space => O(N)



Part-2 handling Queries

Given a list of overs, and number of runs scored in each over. So your task is to compute the total runs scored between [L,R]. You will be multiple queries (Q) each query containing two numbers [L,R], you have to find how many runs were scored in in the overs starting from starting from L till R.



L R

(1 — 3)

4 — 8

0 — 2

6 — 7

5 — 5

Runs L---R =

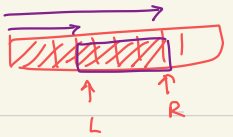
csum[3] − csum[0] = 18 − 8 = 10

64 − 18 = 51

16 = 16

65 − 44 = 21

44 − 24 = 20



Time = O(Q)

Space = O(1)

else (csum[R] − csum[L−1])

if (L == 0){

csum[R]

}

Q ⇒ sum(L---R) = csum[R] − csum[L−1] if L ≠ 0

= csum[R] otherwise

Pre processing Queries

Total time = O(N + Q)

Total space = O(N)

Nested

NQ

Optimised

Two independent loops

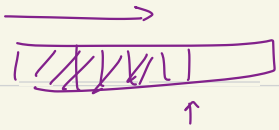
O(Q+N)

for (int i=1; i<=Q, i++){

Read L, R.

Sum = csum[R] − csum[L−1].

}



$ps[i] = ps[i-1] + a(i)$

Prefix sum

$Q >> N$ $Q = 10,000$ $N = 100$

✓ $O(\max(Q, N))$

$O(\max(Q, N))$

$O(\max(Q, N))$ $N = 10,000$ $Q = 100$

$N = 10,000$ $Q = 10,000$

10K → 20K

10K

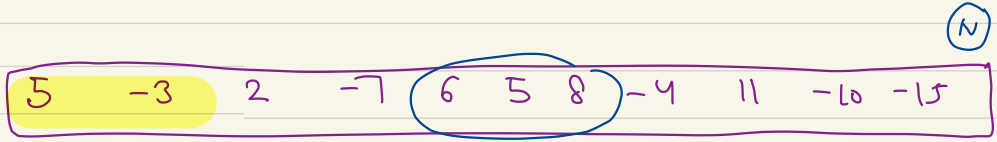
10M

Q Array containing N numbers, find out largest sum of a Subarray.

Subarray ↓

Some continuous part of the array i ---- j

5, -3



-3, 2, -7

6, 5 8

2, -7

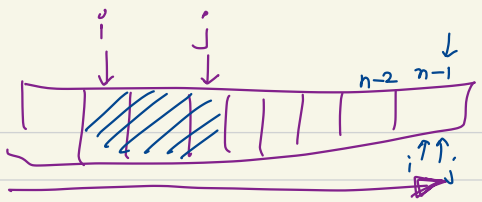
-18

Subarray ? 0

N^1

N^2

2^N



Pick 2 indices out of N indices to create a subarray

$^{N}C_{2}$

$= \frac{N(N+1)}{2} = O(N^2)$

$j \geq i$

Basic

overall

```
for(i=0; i<=n-1; i++){
    for(j=i; j<=n-1; j++){
        print the arr from a(i)---a(j)
    }
}
```

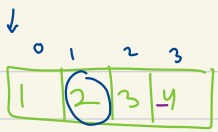
```
for(k=i; k<=j, k++)
    print(a(k));
```

i=0	N
i=1	N-1
i=2	N-2
⋮	⋮
i=N-1	1

$\Sigma n = \frac{n(n+1)}{2}$

$= O(N^2)$ Subarrays.

$O(N^3)$ printing all subarrays



$-2 \times 3 = 6$

$O(N)$ sum

$i=0$, $j=0$ $[1]$ $=1$

$j=1$ $[1,2]$ $=3$

$j=2$ $[1,2,3]$ $=6$

$j=3$ $[1,2,3,4]$ $=2$

$4+3+2+1 = 10$ subs

$\frac{N(N+1)}{2} = \frac{4(5)}{2} = 10$ subs

$i=1$ $j=1$ $[2]$ $=2$

$j=2$ $[2,3]$ $=5$

$j=3$ $[2,3,4]$ $=1$

$i=2$ $j=2$ $[3]$ $=3$

$j=3$ $[3,4]$ $=-1$

$i=3$ $j=3$ $[4]$ $=4$

$O(N^3)$

Largest subarray sum

Basic way

$O(N^3)$

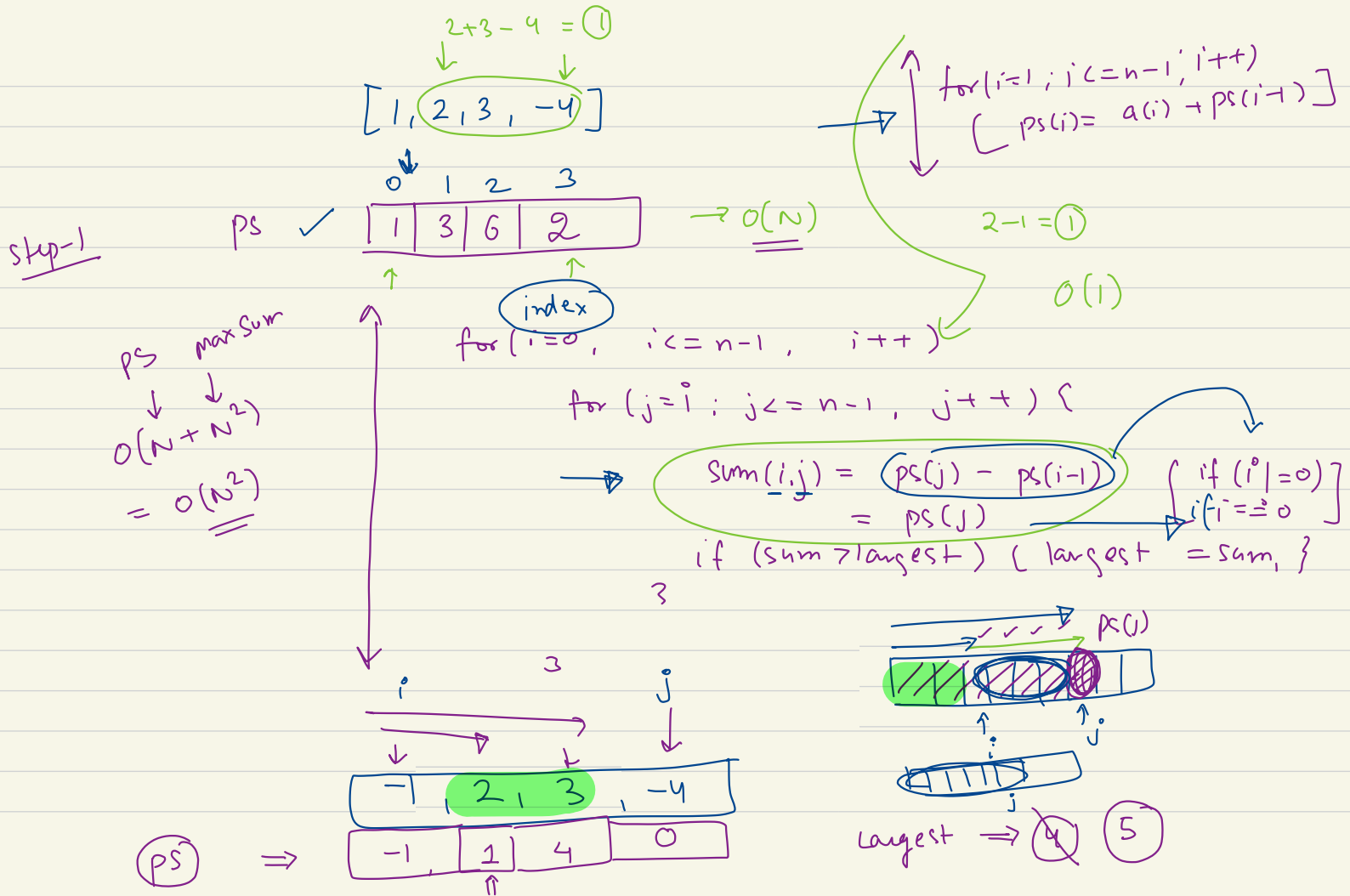
:)

$O(N \cdot N \cdot N) = O(N^3)$

```
int largest = -∞,        // Integer INT-MIN
for(i=0 ; i<= n-1 ; i++){          → N
    for(j=1 ; j<= n-1 ; j++){      → N
        Sum = 0
        for(k=1 ; k<=j ; k++){     → N times
            sum = sum + a(k)
        }
        if (sum > largest) { largest = sum; }
    }
}
print (largest).
```

Sum of a(i---j)

Linear


2+3-4 = 1
[1, 2, 3, -4]
0 1 2 3
ps
1 3 6 2
index
step-1
→ O(N)
for(i=1; i<=n-1; i++)
{ ps(i) = a(i) + ps(i-1) }
2-1 = 1
O(1)
for(i=0; i<=n-1; i++)
for(j=i; j<=n-1; j++) {
sum(i,j) = ps(j) - ps(i-1)
= ps(j)
if(i!=0)
if(i==0
if (sum > largest) { largest = sum; }
ps maxSum
O(N+N^2)
= O(N^2)
3
3
i
j
-1 2 3 -4
ps
-1 1 4 0
ps(j)
j
i
largest
4
5

prev

$-1 + 2 + 3 = 5 - 1 = 4$

find out subarrays and then find their sum, compare largest

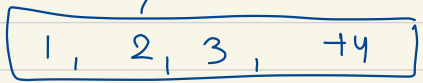
$O(N^3)$ → $O(N^2)$ → $O(N)$ time, $O(1)$ Space

Brute force

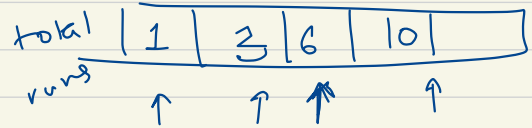
PS sum with extra Space $O(N)$

Runs per over

a

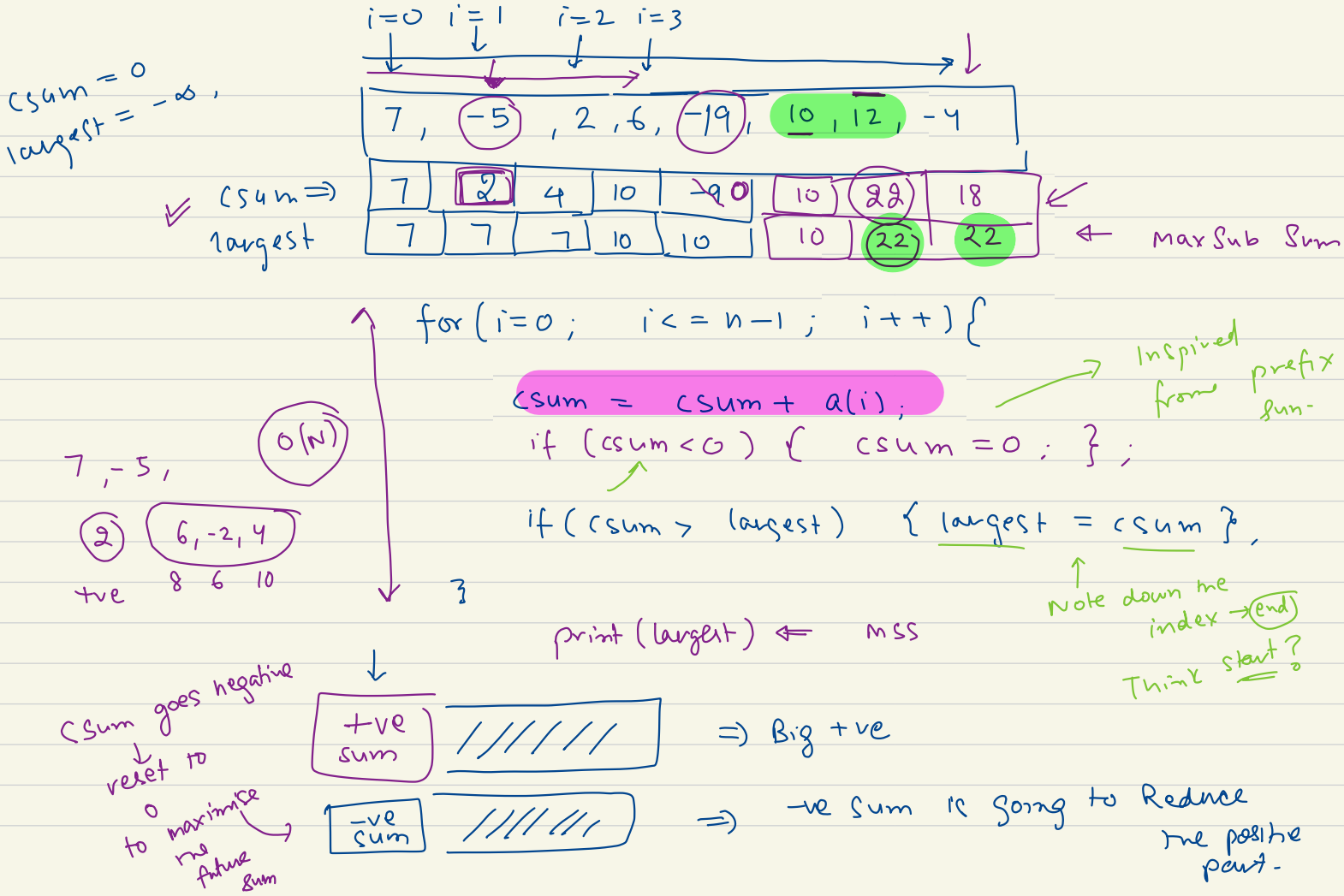


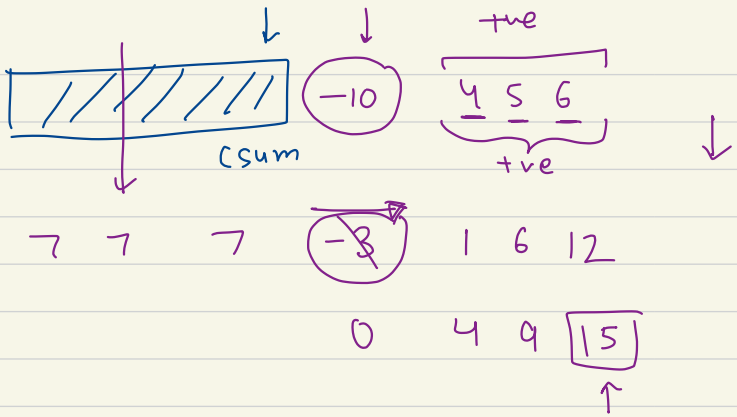
ps

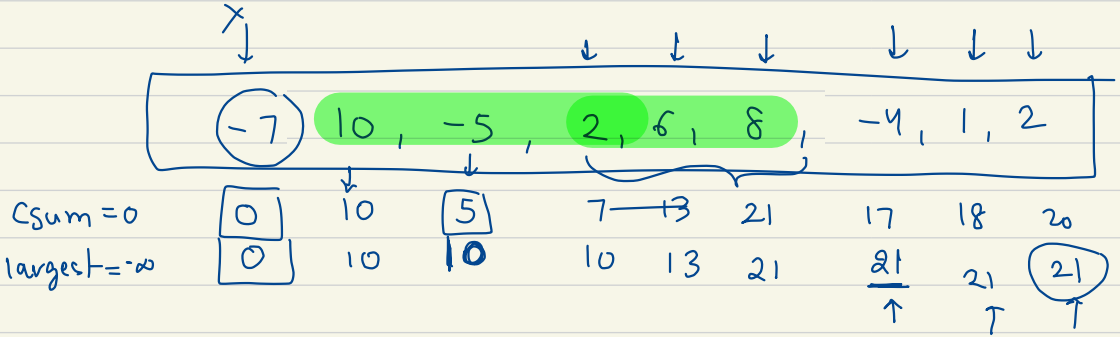


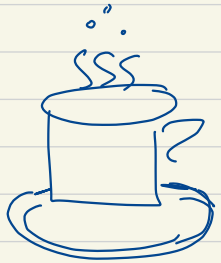
total current over = total prev over + runs.

✓ ps[i] = ps(i-1) + a(i)


csum = 0
largest = -∞
i=0 i=1 i=2 i=3
7, -5, 2, 6, -19, 10, 12, -4
csum ⇒
largest
7 2 4 10 -9 0 10 22 18
7 7 7 10 10 10 22 22
← Max Sub Sum
for (i=0; i<=n-1; i++) {
csum = csum + a[i];
Inspired from prefix sum-
if (csum<0) { csum = 0; };
if (csum > largest) { largest = csum },
Note down the index → end
Think start?
}
print (largest) ⇐ MSS
O(N)
7, -5,
2
6, -2, 4
8 6 10
+ve
csum goes negative
reset to 0 to maximise the future sum
+ve sum
⇒ Big +ve
-ve sum
⇒ -ve sum is going to Reduce the positive part.

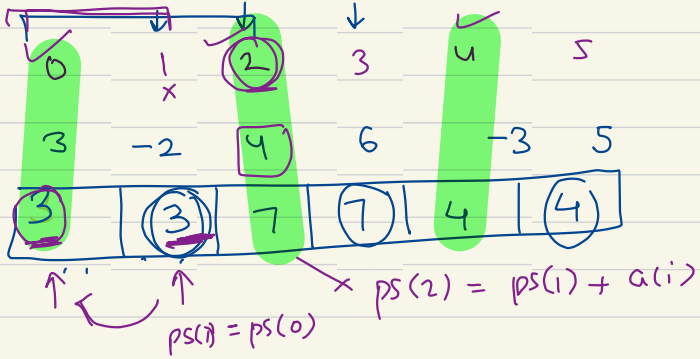

+ve
-10
4 5 6
csum
+ve
7 7 7 -3 1 6 12
0 4 9 15


-7 10 , -5 , 2 , 6 , 8 , -4 , 1 , 2
Csum = 0 0 10 5 7 -3 21 17 18 20
largest = -∞ 0 10 10 10 13 21 21 21 21



Q.

Given an array of size N, construct a prefix array where prefix[I] stores sum of all even indices values from [0,i]



arr[6] =

PSeven[6] =

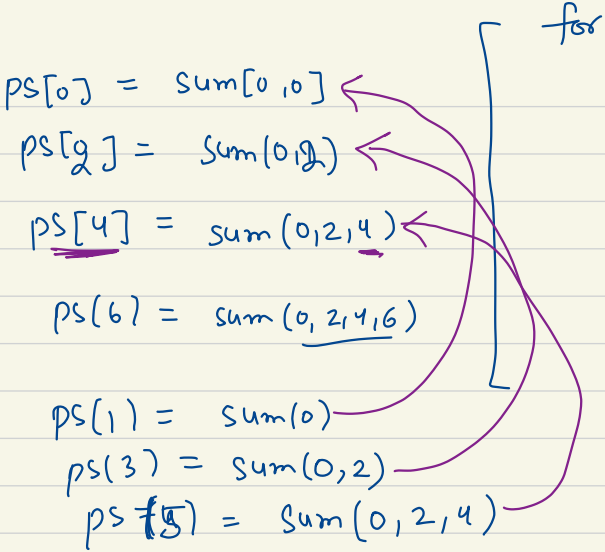
ps(2) = (0,2)

ps(6) = 0,2,4,6

ps(4) = 0,2,4

$\Rightarrow ps[0] = a[0]$

ps(i) = Sum of values of all even idices till i




```
for (i=1 ; i<=n-1 ; i++)
    if (i%2 == 0){
        ps[i] = ps[i-1] + a[i];
    }
    else {
        ps[i] = ps[i-1];
    }
```

prev even position → prev odd pos.

current

prev odd save previous

current[i] as i is even

Q2. Given N Array elements, and Q queries for each query calc the sum. Of all even indices in the given range [L,R]

	0	1	2	3	4	5	6	7
arr[8] =	3	4	-2	8	6	2	1	3
O(N) → pseven[] =	3	3	1	1	7	7	8	8

$sum(L,R) = ps[R] - ps[L-1]$ if $L != 0$

Q

Q = 4

[2, 6] = loop (N) → a(2) + a(4) + a(6)
-2 + 6 + 1 = 5 ✓

faster → O(1)
ps[6] − ps[2−1]
= 8 − 3
= 5 ✓

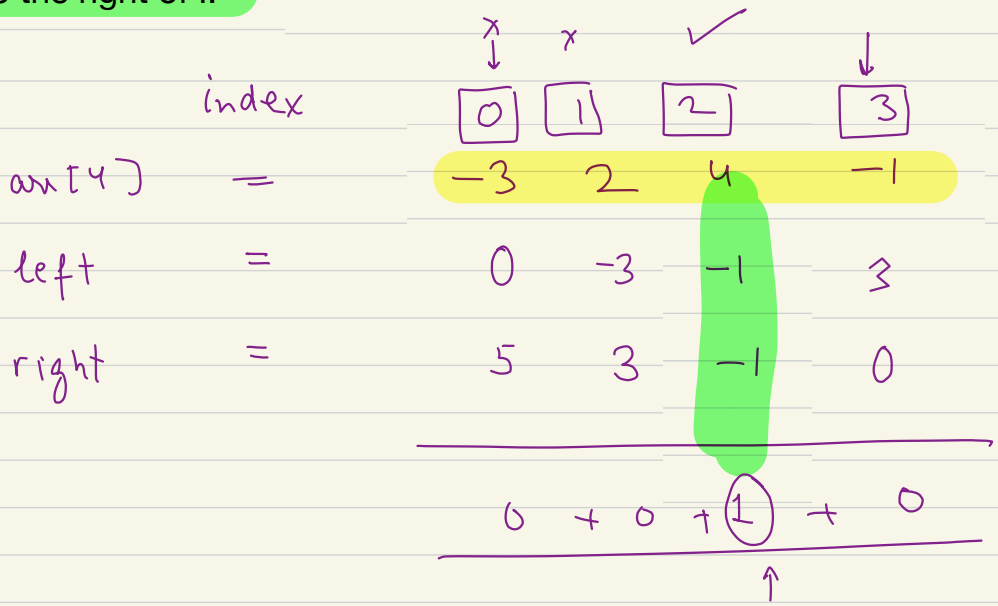
[3, 7] = ps[7] − ps[2]
= 8 − 1
= 7 ✓

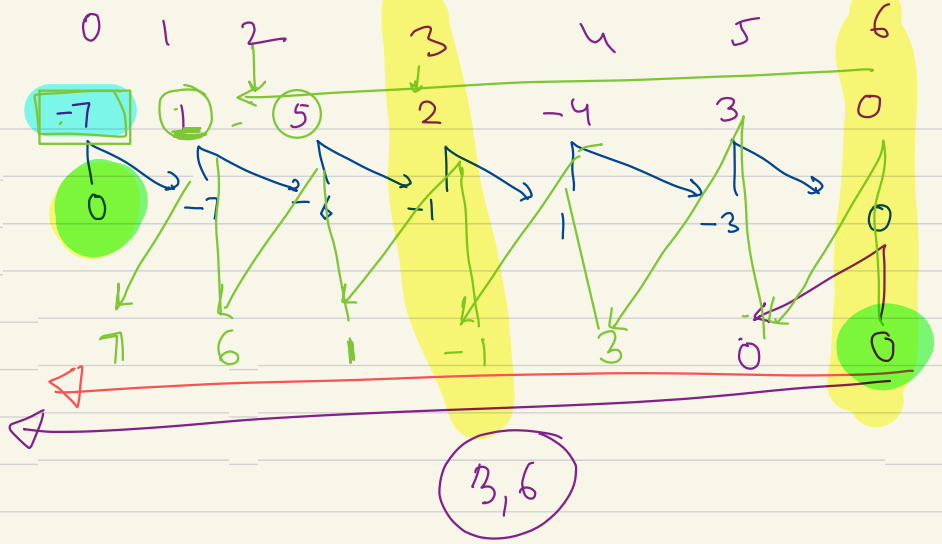
6 + 1
= 7

O(N + Q) with space O(N)

(2) Equilibrium Index

Given an array containing N elements, count no of equilibrium index in the array.
An index I is said to be eq. If sum of all elements of the left to I = sum of all elements to the right of I.





Precompute
- left
- right

~~Aag~~ Again we need ps but not including the current element

Algo-1

N → N

for(every i) ⇒ N(N) = O(N²)

- N ∘ compute left — loop N 0–i-1
- ∘ Compute Right — loop N i+1–N-1
- ∘ Compare

Algo-2

int ps[N] := {0}

-7	1	5	2	-4	3	0

ps

-7	-6	-1	1	-3	0	0

ps(0) = a(0)

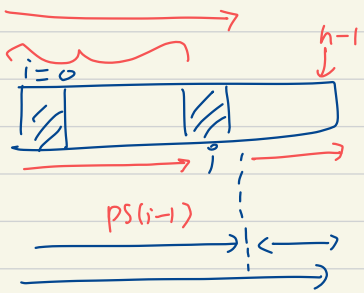
Part-1 →

```
for (i=1, i<N; i++)
    { ps(i) = ps(i-1) + a(i); }
```

Part-2

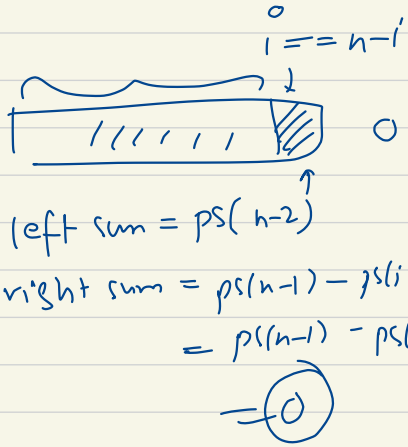
```
cnt = 0
for (i=0, i<= N-1; i++) {
    if (i==0) { left-sum = 0 }
    else { left_sum = ps(i-1); }
    Right-sum = ps(n-1) - ps(i);
```

i≠0



```
if( left_sum == right_sum){
    cnt = cnt + 1 ;
}
```

→ Eg. Index



left sum = ps(n-2)

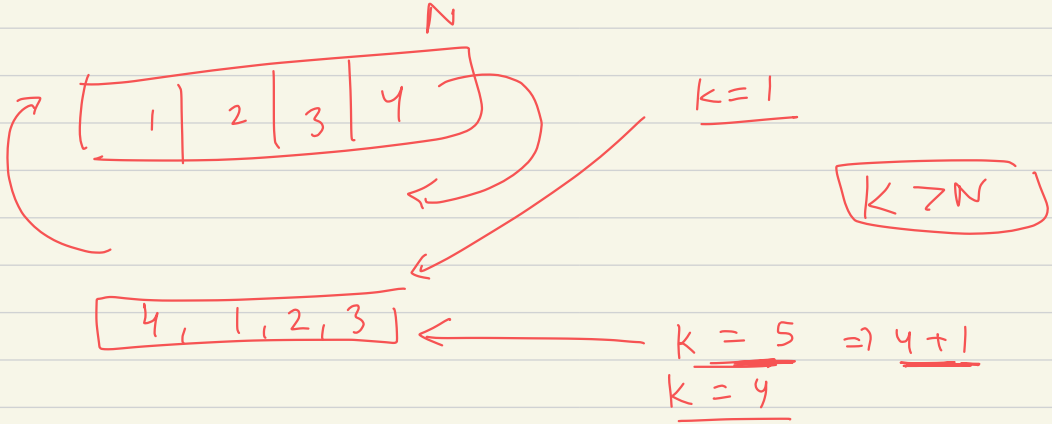
right sum = ps(n-1) - ps(i)

= ps(n-1) - ps(n-1)

= 0

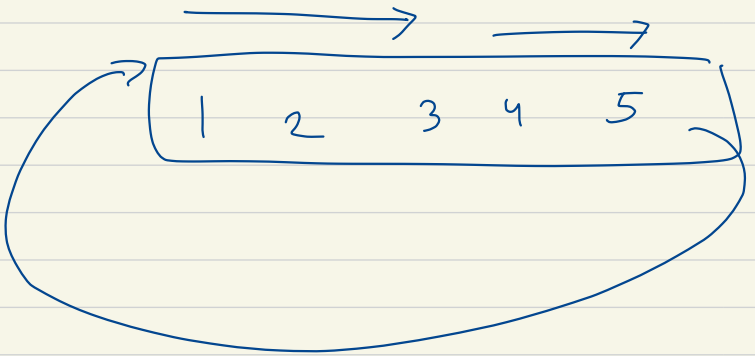
Doubt

Rotate an Array K times



→ effective Rotation = K % N

1) New array



k = 2

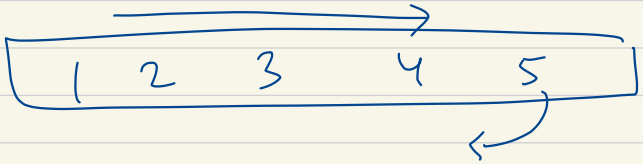
4 5 1 2 3

4 5 | 1 2 3

Copy

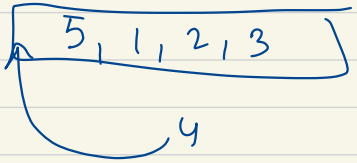
(2) One By One Copy (Time)

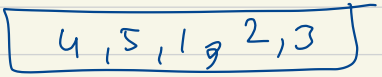
N * K



5

5 1, 2, 3, 4

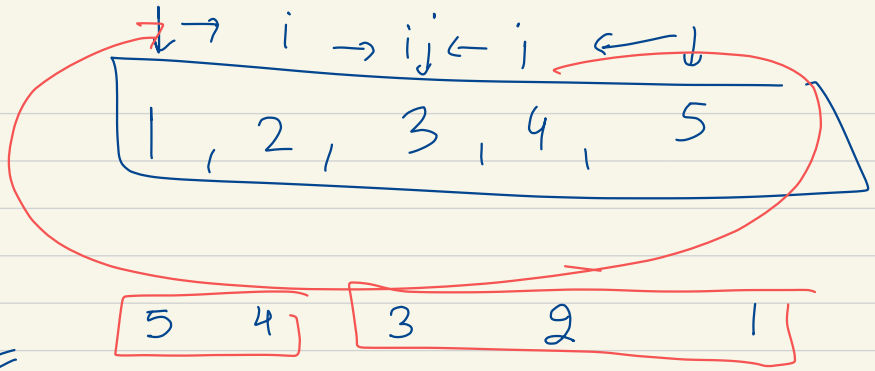




i

j

3



O(N) time
O(1) Space

Same away

Rev all

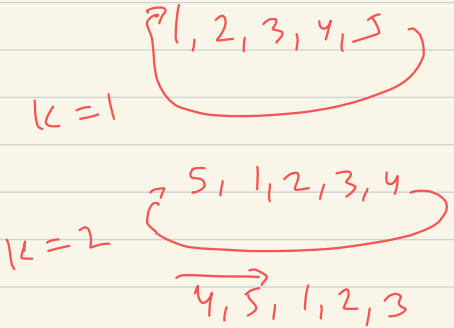
5 4 3 2 1

Rev first K

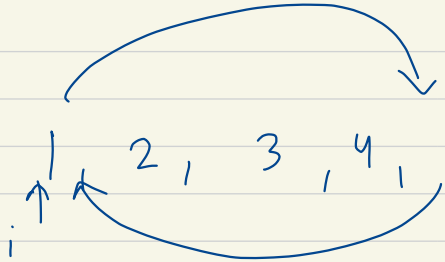
4 5

Rev Next

1 , 2 , 3


1, 2, 3, 4, 5
k = 1
5, 1, 2, 3, 4
k = 2
4, 5, 1, 2, 3

2 pointer approach rev an array in place

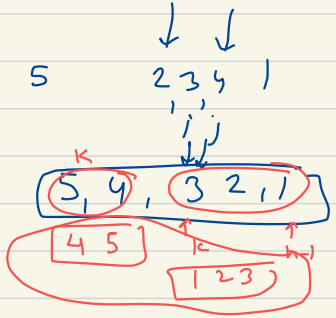


i = 0

j = n - 1

```
while (i < j) {
    swap(a[i], a[j])
    i++ j--
}
```

Step-1

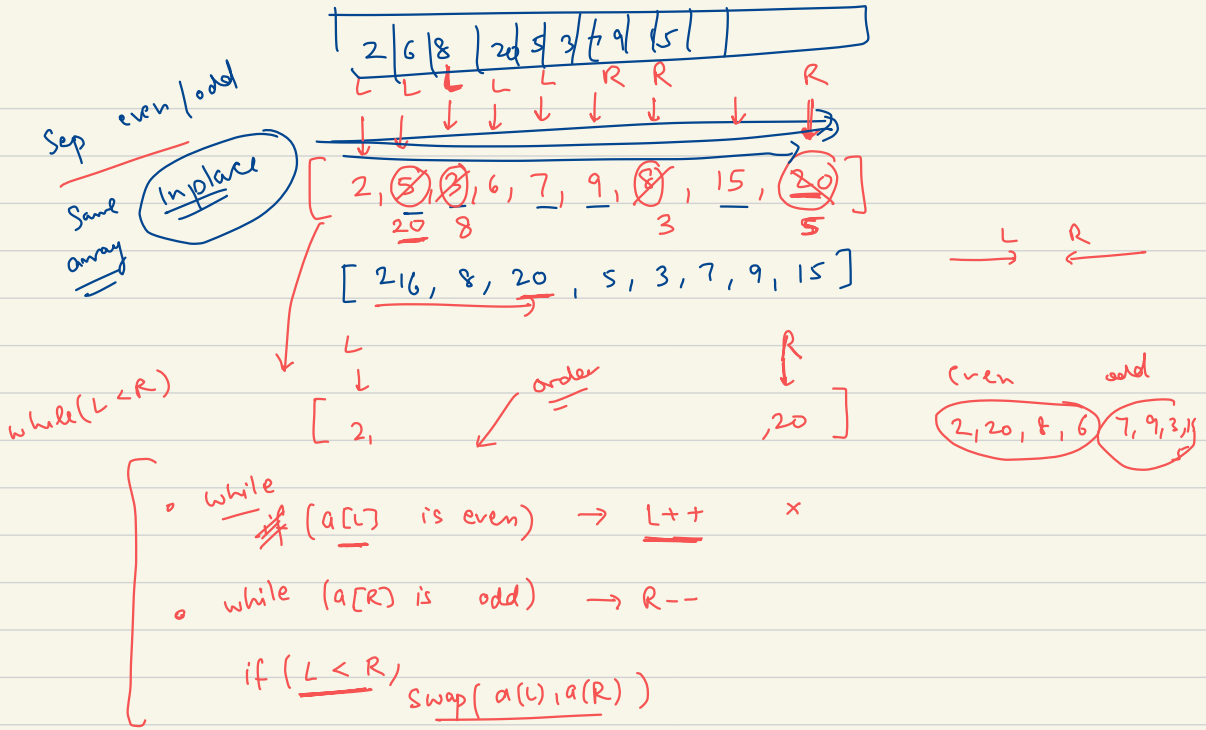


Step-2 Rev [0, k-1]

K value

Stop

- rev(0, n-1) ←
- ~~void rev(arr, i, j)~~
- rev(arr, 0, k-1) ←
- rev(arr, k, n-1) ←


Sep even | odd
Same array
Inplace
2 6 8 20 5 3 7 9 15
L L L L L R R R
[2, 5, 8, 20, 6, 7, 9, 3, 15, 20]
20 8 3 5
[216, 8, 20, 5, 3, 7, 9, 15]
L R
L R
while(L < R)
[2, ,20]
order
even
odd
2, 20, 8, 6
7, 9, 3, 15
• while (a[L] is even) → L++ x
• while (a[R] is odd) → R--
if (L < R) swap(a(L), a(R))