Todays Agenda

- → Sum of all subarray sums
- → No. of subarrays of length K
- → Print si & ei of subarrays of length K
- -> Maximum subarray sum with length K.

< Question > : Given arr[N]. Find sum of all Subarray sums.

arr[] → [1, 2, 3]

[1] → 1

[1,2] → 3

[1,2,3] → 6

[2] → 2

[2,3] → 5

[3] → 3

ans → 20

BF Idea → Consider all the subarrays. For every subarray, find its sum and keep on adding the sum to get ans.

```
int ans = 0;
for( i = 0; i < N; i++){
    for( j = i; j < N; j++){
        int sum = 0
        for( k = i; k <= j; k++){
            sum += arr[k];
        }
        ans += sum;
    }
}
return ans;
```

T.C → $O(N^3)$
S.C → $O(1)$

Idea -2 use psum[ ]

```
int psum[N];
psum[0] = arr[0];
for(i = 1; i < N; i++){
    psum[i] = psum[i-1] + arr[i];
}

int ans = 0;
for(i = 0; i < N; i++){
    for(j = i; j < N; j++){
        sum = 0;
        if(i == 0){
            sum = psum[j];
        }
        else{
            sum = psum[j] - psum[i-1];
        }
        ans += sum;
    }
}
return ans;
```

T.C → $O(N^2)$

S.C → $O(N)$

modification of given array is not allowed.

O(1)

Idea -3 Carry Forward

```
int ans = 0;
for( i = 0; i < N; i++){
    sum = 0;
    for( j = i; j < N; j++){
        sum += arr[j];
        ans += sum;
    }
}
return ans;
```

sum = 0

i = 0 , j = 0

sum → 0 + arr[0] → arr[0]

sum → arr[0] + arr[1]

sum → arr[0] + arr[1] + arr[2]

sum → arr[0] + arr[1] + arr[2] + arr[3]

|

sum → arr[0] + — — arr[N-1]

T.C → O(N²)

S.C → O(1)

$1 \le N \le 10^6$

Idea -4

arr → [3 -2 1 4]

0 1 2 3

↓ ↓ ↓ ↓

4 6 6 4

ans = 22

[3] [-2] [1] [4]

[3, -2] [-2, 1] [1, 4]

[3, -2, 1] [-2, 1, 4]

[3, -2, 1, 4]

[3]

[3 -2]

[3 -2 1]

[3 -2 1 4]

[-2]

[-2 1]

[-2 1 4]

[1]

[1 4]

[4]

(3*4) + (-2*6) + (1* 6) + (4*4)

= 22

- How many times an element appears in all the subarrays?

arr → [3 -2 4 -1 2 6]
0 1 2 3 4 5

[3] [-2] [4] [-1] [2] [6]
[3, -2] [-2, 4] [4, -1] [-1, 2] [2, 6]
[3, -2, 4] [-2, 4, -1] [4, -1, 2] [-1, 2, 6]
[3, -2, 4, -1] [-2, 4, -1, 2] [4, -1, 2, 6]
[3, -2, 4, -1, 2] [-2, 4, -1, 2, 6]
[3, -2, 4, -1, 2, 6]

ans → 10

- In how many subarrays index-2 will be present?

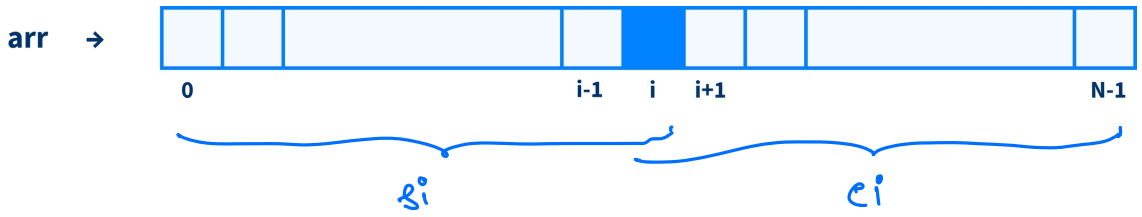
arr → [3 -2 4 -1 2 6]
0 1 2 3 4 5

ans = 12

si	ei
0	2
1	3
2	4
	5

options for si × options for ei

Generalize



' In how many subarray, ith index element will be present? '

↳ $(i+1) \times (N-i)$

options for si (0 to i) : $i+1$ × options for ei (i to N-1) : $N-i$

Contribution of i^{th} element in ans ⇒ $arr[i] \times ((i+1) \times (N-i))$

Contribution technique

< / > Code

```
long ans = 0
for( i = 0; i < N; i++){
{    ans += arr[i] x (i+1) x (N-i)
}
return ans;
```

arr → [	3	-2	1	4]
	0	1	2	3

ans = 0 + 12 + (-12) + 6 + 16

= 22

T.C → O(N)
S.C → O(1)

- **Number of subarrays of length k**

17	3	4	9	12	6
0	1	2	3	4	5

Number of subarrays with length = 1 → 6 [N]

Number of subarrays with length = 2 → 5 [N-1]

Number of subarrays with length = 3 → 4 [N-2]

Number of subarrays with length = 4 → 3 [N-3]

Number of subarrays with length = 5 → 2 [N-4]

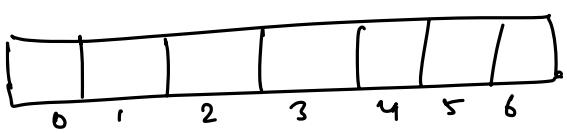
Number of subarrays with length = 6 → 1 [N-5]

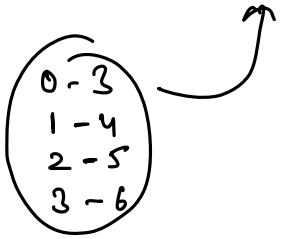
↓

Number of subarrays with length K ⇒ N - K + 1

Quiz→ N=7, K=4

no. of subarrays of length K → 7-4+1 ⇒ 4





< *Question* > : Print si and ei of every subarray of length k.

N = 8, K = 3

5	7	9	8	-2	6	65	19
0	1	2	3	4	5	6	7

si	ei
0	2
1	3
2	4
3	5
4	6
5	7

code →

```
si = 0, ei = k-1;
while( ei < N ){
    print( si + "-" + ei);
    si++;
    ei++;
}
```

Printing si & ei of every subarray of length K.

∴ No. of iterations = no. of subarrays of length K

= N - K + 1.

T.C → O(N-K)

S.C → O(1)

< *Question* > : Given arr[N]. Print maximum subarray sum of subarray with length k.

N = 10

K = 5

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

si	ei		sum
0	4	→	7
1	5	→	8
2	6	→	12
3	7	→	16
4	8	→	10
5	9	→	11

ans = 16

BF Idea → Consider all subarrays of length K. Iterate on each subarray & find its sum.

$1 \le N \le 10^5$

$1 \le K \le N$

</> Code

```
si = 0, ei = k-1;   ans = INT.MIN;
while( ei < N ){
    sum = 0;
    for( k = si; k <= ei; k++){
    {       sum += arr[k]
    }
    ans = Max( ans, sum);

    si++;
    ei++;
}
return ans;
```

total iterations → $(N-K+1) \times K$

- $K=1$ → $(N-1+1) \times 1 \Rightarrow N$
- $K=N$ → $(N-N+1) \times N = N$
- $K=N/2$ → $(N-\frac{N}{2}+1) \times (\frac{N}{2}) \approx \frac{N^2}{4} \approx N^2$

Idea -2 use psum[ ]

```
// Create pSum array → todo

si = 0, ei = k-1;  ans = INT.MIN;

while( ei < N ){
    sum=0;
    if(si==0) { sum = arr[ei] }
    else { sum = arr[ei] - arr[si-1] }

    ans = Max(ans, sum);
    si++;
    ei++;
}

return ans;
```

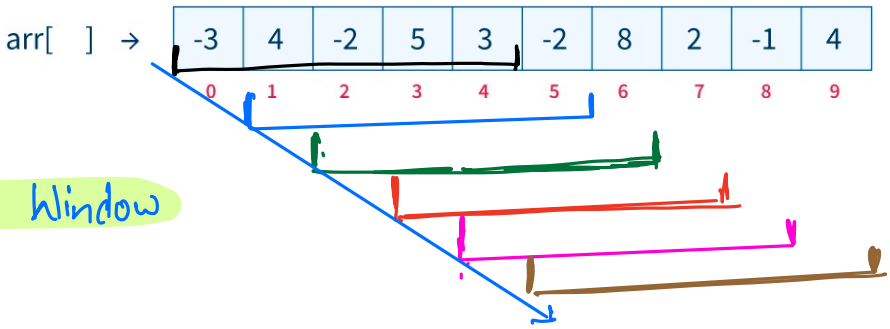
T.C → O(N)
S.C → O(N)

modifying given array is not allowed.

O(1)

N = 10

K = 5



Sliding Window

si	ei	sum
0	4	7
1	5	sum = sum - arr[0] + arr[5] = 7 - (-3) + (-2) = 8
2	6	sum = sum - arr[1] + arr[6] = 8 - 4 + 8 = 12
3	7	sum = sum - arr[2] + arr[7] = 12 - (-2) + 2 = 16
4	8	sum = sum - arr[3] + arr[8] = 16 - 5 + (-1) = 10
5	9	sum = sum - arr[4] + arr[9] = 10 - 3 + 4 = 11

si ei sum = sum - arr[si-1] + arr[ei]

< / > Code

1. Create the window → Calculate sum of first K elements.

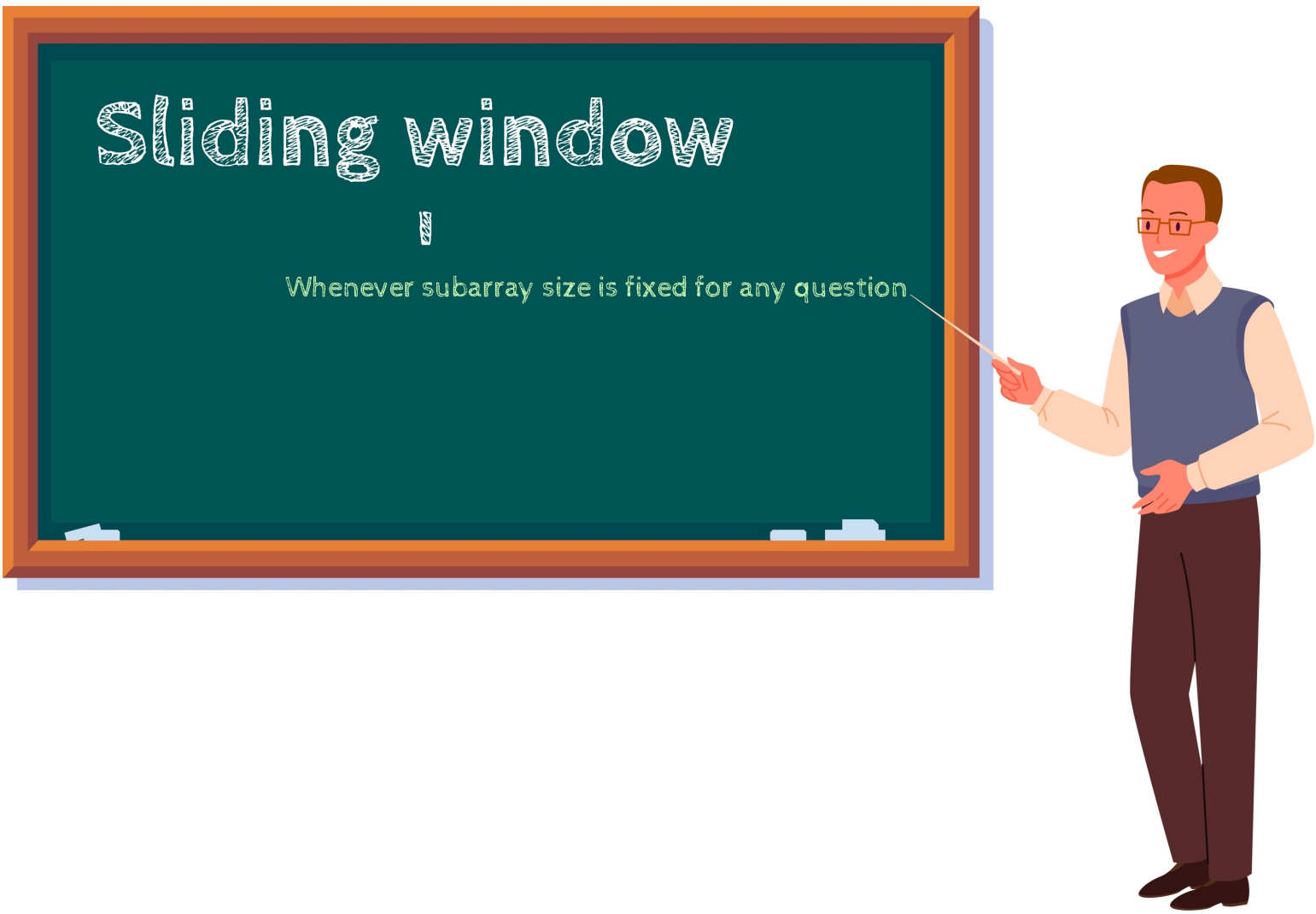
```
sum = 0 ;
for( i = 0; i < K; i++){
        sum += arr[i];
}
ans = sum;
```

2. Consider the remaining subarrays of length K with sliding window

```
si = 1 , ei = K;
while( ei < N){
        sum = sum - arr[si-1] + arr[ei];
        ans = Max( ans, sum);

        si++;
        ei++;
}
return ans;
```

T.C → O(N)
S.C → O(1)


Sliding window
Whenever subarray size is fixed for any question

Contribution technique →

If we need to find contribution of an element in the overall ans

or

how many times an element is present in all sub-arrays.

Sliding Window →

Whenever subarray of fixed size is mentioned in the problem statement, try using sliding window technique.

psum[] → extra space for creating psum()

am-

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

B = 4

	l	r		
✓	4	0	→	14
	3	1	→	12
	2	2	→	19
	1	3	→	17
	0	4	→	18

```
lsum = 0;
for( i=0; i<B; i++){
    lsum += arr[i];
}

rsum = 0;
ans = lsum;

for( k=1; k<=B; k++){
    lsum = lsum - arr[B-k];
    rsum = rsum + arr[N-k];
    am = max(am, lsum + rsum);
}
return am;
```

(T.C → O(B)
S.C → O(1))