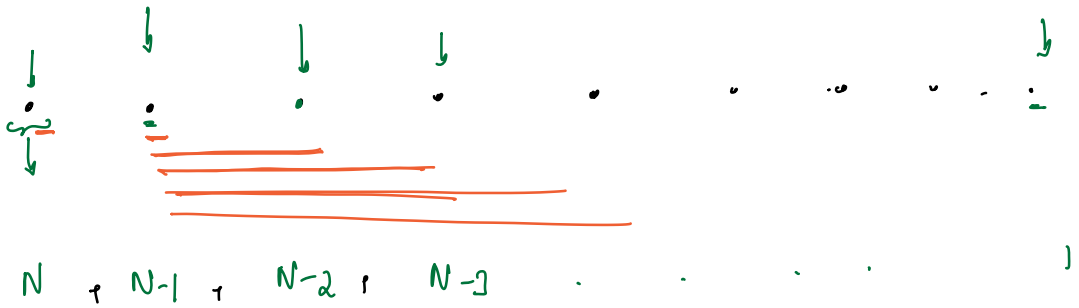
Subarrays : Continous segment of an array

→ Single Element

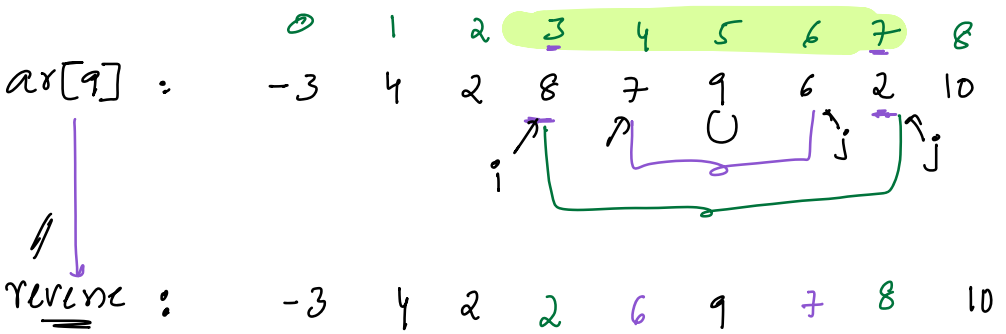
→ Full array

→ {si, ei} si <= ei



Total Subarrays = $\frac{(N)(N+1)}{2}$

Q) Given an array & si & ci, reverse entire subarray [si - ci]

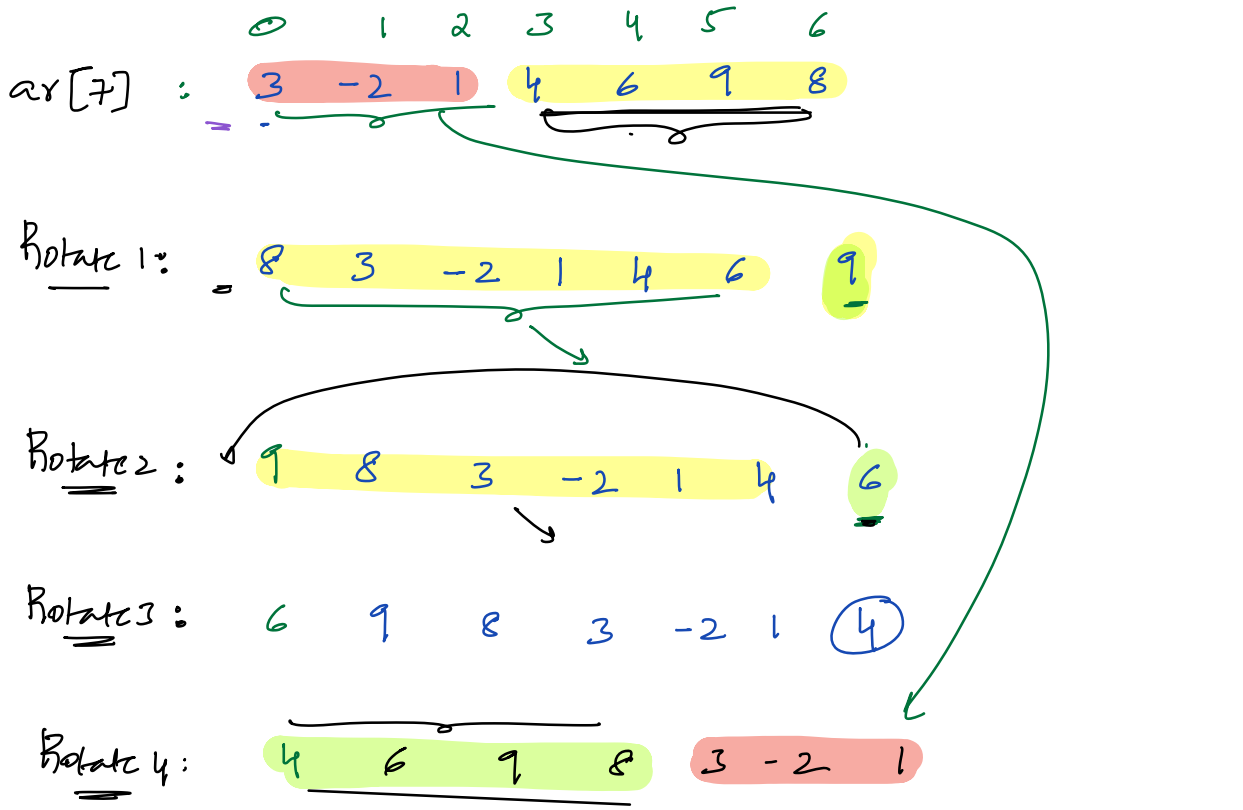



```
void reverse(int ar[], int s, int e){
    i = s, j = e;
    while(i <= j){
        swap ar[i] & ar[j]
        i++; j--
    }
}
```

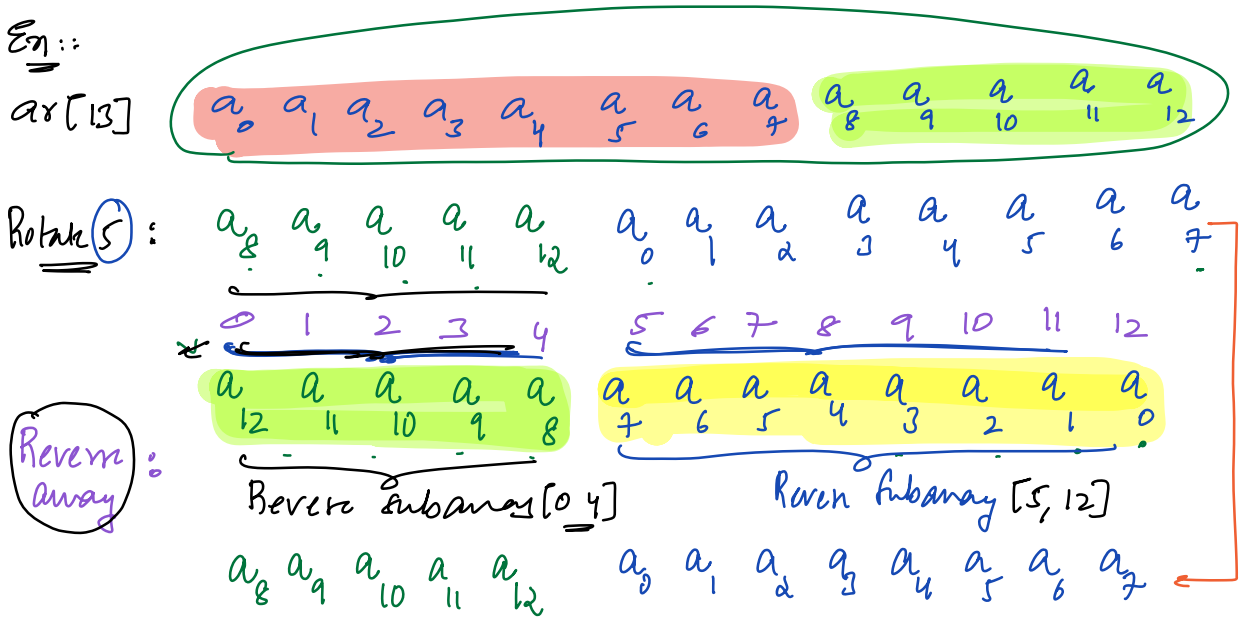
s = 0, e = N-1

O(N)

28) Rotate Array — Rotate array from R-L



Eg::



→ Given N array Elements Rotate by (k) Times

Iterations

Step1: Reverse Entire array. [N] — N/2

Step2: Reverse 1st k Elements : [0, k-1] — k/2

Step2: Reverse remaining N-k Elements : [k, N-1] — (N-k)/2

[N, N-k]

TC : O(N) SC : O(1) — 1

N/2 + k/2 + (N/2 - k/2)

→ N

if (k ≥ N) we will error, Array out of bounds / Segmentation

{ k = k % N }

→ ar[6] : a_0 a_1 a_2 a_3 a_4 a_5

k = k % 6

		Rotate		
Rotate by 0 :	a_0 a_1 a_2 a_3 a_4 a_5	6 = 0	12 % 6 = 0	
Rotate by 1 :	a_5 a_0 a_1 a_2 a_3 a_4	7 % 6 = 1	13 % 6 = 1	
Rotate by 2 :	a_4 a_5 a_0 a_1 a_2 a_3	8 % 6 = 2	14 % 6 = 2	
Rotate by 3 :	a_3 a_4 a_5 a_0 a_1 a_2	9 = 3	15 = 3	21
Rotate by 4 :	a_2 a_3 a_4 a_5 a_0 a_1	10 = 4	16 = 4	
Rotate by 5 :	a_1 a_2 a_3 a_4 a_5 a_0	11 = 5	17 = 5	

Given N, Rotate by k : k = k % N } k > N, k = k % N,

→ TODO: Build from Rotating left & Right

3@) Print start & end index of all subarrays (len = k)

	0	1	2	3	4	5	6	7	8	9	10	11
ar[12] :	3	4	2	-1	6	7	8	9	3	2	-1	4

[a b] = b-a+1

k = 6

[0, k-1]

// Generalize N, k.

→ k-x-0 ex

→ k

Subarray	string	endind
1	0	5
2	1	6
3	2	7
4	3	8
5	4	9
6	5	10
7	6	11

Subar	st	end
1st	0	k-1
2nd	1	k
3rd	2	k+1
4th	3	k+2
⋮		
[last] =	[x	N-1] k Elements

k-1

add k-1

add k-1

add k-1

Pseudocode:

```
for(i = 0; j = N-k; i++) {
    j = i + k-1
    print (i, j)
}
```

N-y-x+y = k

N-x = k, k = N-x

last = [N-k N-1]

[i, x] = k } x-i+1 = k

x = i+k-1

4&) Find Max subarray sum with len = k.

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

k = 5

Sums

0-4 : 7

1-5 : 8

2-6 : 12

3-7 : 16

4-8 : 10

5-9 : 11

Max : 16

output : 16

TC : (N-k+1)x(k)

For what k we can get max prod.

k = N → (N-N+1)(N) = N

k = 1 → (N-1+1)(1) = N

k ≈ N/2 →

(N-N/2+1) * (N/2)

≈ (N/2+1)(N/2) → O(N²)

Idea:

Sum of all subarray of len = k

& get max of them

Pseudocode:

```
Max = 0/ar[0]/INT_MIN
// Subarrays of k
for(i=0; i<N-k; i++){ -> [N-k+1]
    j = i+k-1
    [i j] len = k
    S=0;
    for(l=i; l<=j; l++){ // len = k
        S = S + ar[l]
    }
    // S is subarray sum from [i,j]
    if(S>Max){ Max = S }
}
return Max
```

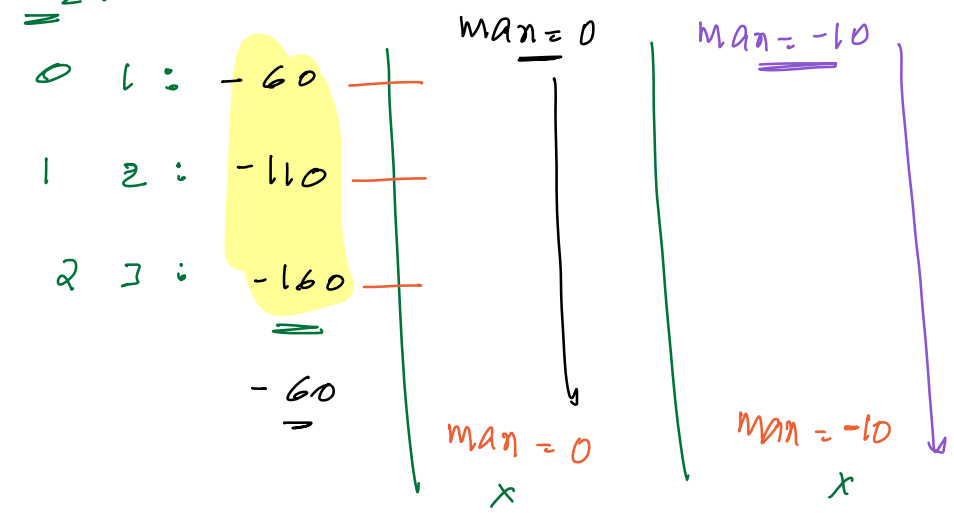
we are getting sum

10:45 break

Ex:

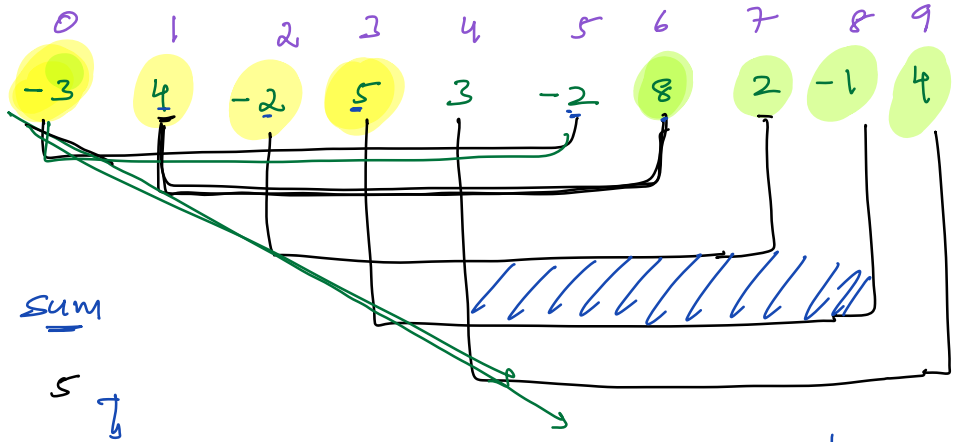
0	1	2	3
-10	-50	-60	-100

k = 2 :



arr[10] :

k = 6



sum

[0 - 5] : 5

[1 - 6] : 5 - arr[0] + arr[6] = 5 - (-3) + 8 = 16

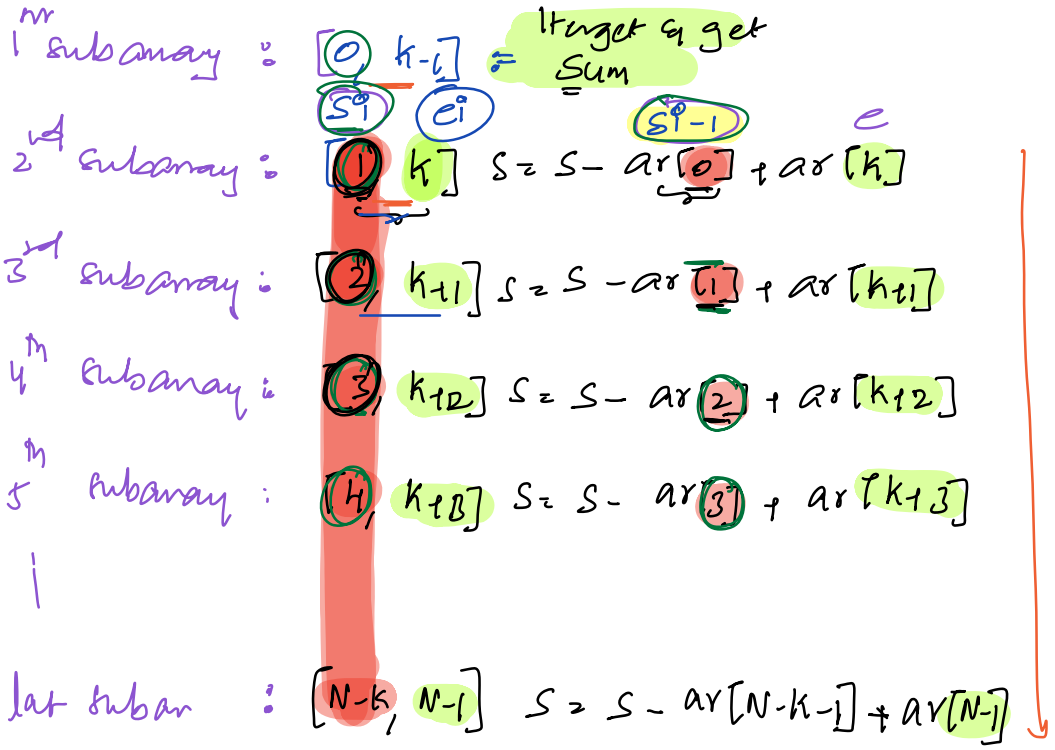
[2 - 7] : 16 - arr[1] + arr[7] = 16 - (4) + 2 = 14

[3 - 8] : 14 - arr[2] + arr[8] = 14 - (-2) + -1 = 15

[4 9] = 15 - arr[3] + arr[9] = 15 - 5 + 4 = 14

//

Generalize N Array Sliding window.




```
S = 0;
for(i = 0; i < k; i++){
    S = S + ar[i]
}                                   [0, k-1] : k
                                    times
max = S // S is first subarray sum
for(i = 1; i <= N-k; i++){          [1, N-k] : N-k
    j = [i+k-1]                                 N
    S = S - ar[i-1] + ar[j]
    if(S > max) { max = S }
}
return max
```

TC : O(N)

SC : O(1)

//

Given N array elements, print all subarray sums

	0	1	2	3
ar[4] :	3	-1	0	2

[0 0] : 3

[0 1] : 2

[0 2] : 2

[0 3] : 4

[1 1] : -1

[1 2] : -1

[1 3] : 1

[2 2] : 0

[2 3] : 2

[3 3] : 2

// Sum of all subarray sums = 14

Sol 1: For every subarray calc & print sum.

```
i = 0; i < N; i++){
    j = i; j < N; j++){
        print [i  j]
        s = 0;
        k = i; k <= j; k++){
            s = s + ar[k]
        }
        print(s)
    }
}
```

TC : O(N3)

ar[6]

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

5 + -1 = 4 + 6

10 + 4

14 + 8

22

[0 0] : 3

[0 1] : 5

[0 2] : 4

[0 3] : 10

[0 4] = [0 3] + ar[4] = 14

[0 5] = [0 4] + ar[5] = 22

[0 i] = s

[0, i+1] = s + ar[i+1]

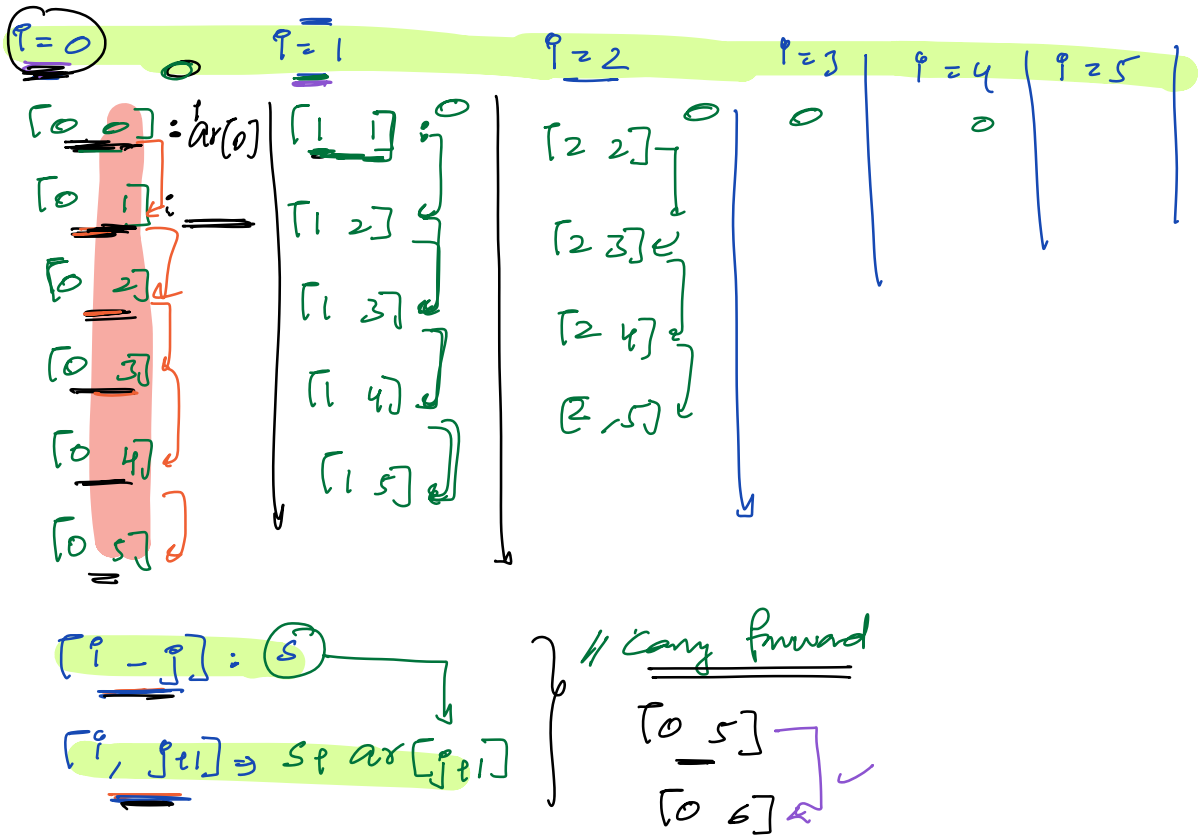
```
for(i=0; i<N; i++){
    S=0
    for(j=i; j<N; j++){
        S = S + ar[j]
        Print(S)
    }
}
```

TC: $O(N^2)$

SC: $O(1)$

ar[6]

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



```
int ans = 0
for(i = 0; i < N; i++){
    for(j = i; j < N; j++){
        -> [i  j]
        s = 0;
        for(k = i; k <= j; k++){
            s = s + arr[k]
        }
        ans = ans + s // sum of all subarrays
    }
}
return ans;   TC: O(N^3)
```

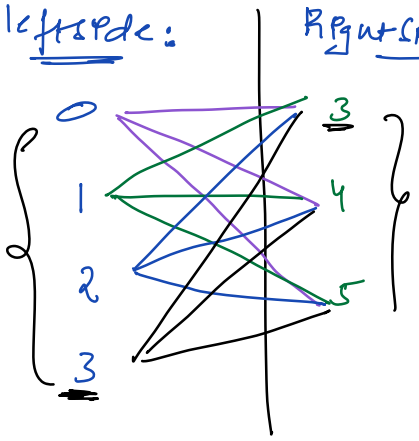
```
ans = 0
for(i = 0; j < N; i++){
    s = 0
    for(j = i; j < N; j++){
        s = s + arr[j]
        ans = ans + s // -> sum of all subarray sums
    }
}
return ans; -> TC: O(N^2)
```

// Sum of all subarray sums → O(N^3) ✓ → O(N^2) ✓ → ...

arr[] :

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

// In how many subarrays index 3 is present



→ Total pairs = 12 = left * right =

4 * 3 = 12

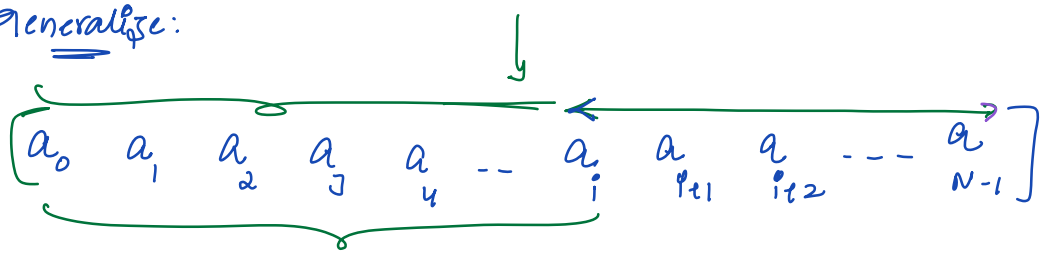
arr[] :

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

leftend	Right
0	1
1	2
	3
	4
	5

// subarrays = 5*2 = 10

// Generalize:



Leftside : $[0, i]$ ⇒ $i+1$ ⇒ $i+1$

Rightside : $[i, N-1]$ ⇒ $N-1-i+1$ ⇒ $N-i$

// Number of subarrays in which index i is present = $(i+1)(N-i)$

	0	1	2	3	4	5
ar[6]	3	2	1	6	-1	4
$(i+1)(N-i)$	6	10	12	12	10	6
#cont :	18	20	12	12	-10	24

Sum of all subarrays =

// ↦ // Adding contribution of individual Elements

// Pseudocode :

```
S=0;
for(i=0; i<N; i++){
    left = (i+1), right = (N-i)
    S = S + ar[i] * left * right
}
return S
```

left * right : Number of subarrays in which ar[i] present

TC : O(N)

SC : O(1)