Agenda

1) Basics of Subarrays
2) Problems on Subarrays

Subarray → continuous sequence of an array

- Full Array → Yes.
- Single Element → Yes.
- Is there an empty subarray? ⇒ Yes but we don't generally encounter them

$$A[] = \{ \underset{0}{-3}, \underset{1}{4}, \underset{2}{5}, \underset{3}{-6}, \underset{4}{8}, \underset{5}{9}, \underset{6}{10}, \underset{7}{-10}, \underset{8}{8} \}$$

$\{-3, 4, 5\}$ ⇒ YES

$\{-6, 8, -10, 8\}$ ⇒ NO

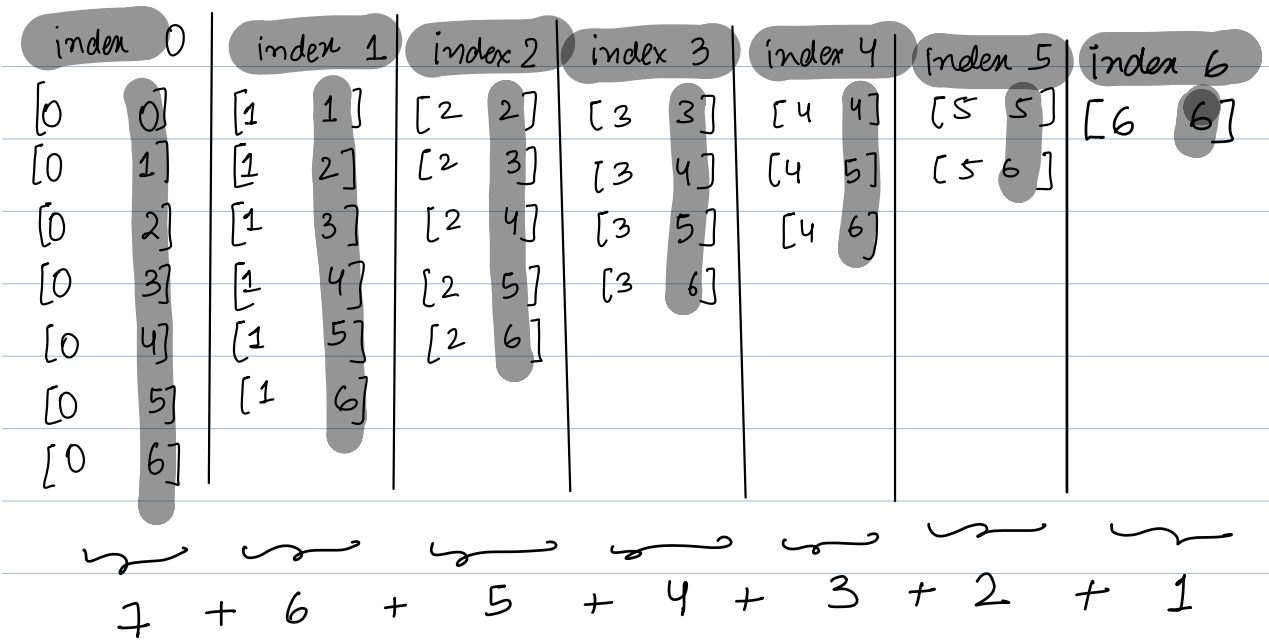
$\{5\}$ ⇒ YES

Total No. of Subarrays

$$A[] = \{ \underset{0}{4}\ \underset{1}{2}\ \underset{2}{10}\ \underset{3}{3}\ \underset{4}{12}\ \underset{5}{-2}\ \underset{6}{15} \}$$

$N = 7$

Subarrays starting at



$$\underbrace{1 + 2 + 3 + 4 + \cdots N}$$

$$\frac{N(N+1)}{2}$$

Generate all the subarrays

start of a subarray can be

0, 1, 2, 3, 4 ..... N-1

i	j
0	[0 N-1]
1	[1 N-1]
2	[2 N-1]
3	[3 N-1]
4	[4 N-1]
⋮	
N-1	[N-1 N-1]

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

TC → $O(N^2)$

$\Downarrow$

$\frac{N(N+1)}{2}$

Max sum of each subarray

```
max_sum = -∞
for (i=0; i<N; i++){
    for (j=i; j<N; j++){
        int sum = 0
        for (K=i; K<=j; K++){
            sum += A[K]
        }
        max_sum = max(max_sum, sum)
    }
}
return max_sum
```

// Brute Force

$O(N^3)$

Optimization — Using Prefix Sum Array

max_sum = $-\infty$ / A[0]

C++ : INT_MIN
Java : Integer.MIN_VALUE
Python : -inf

① Create PF[ ]

② for (i=0; i<N; i++){

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

    [i j]

    sum = PF[j] - PF[i-1]

    Handle that edge case.

    max_sum = max (max_sum, sum)

  }

}

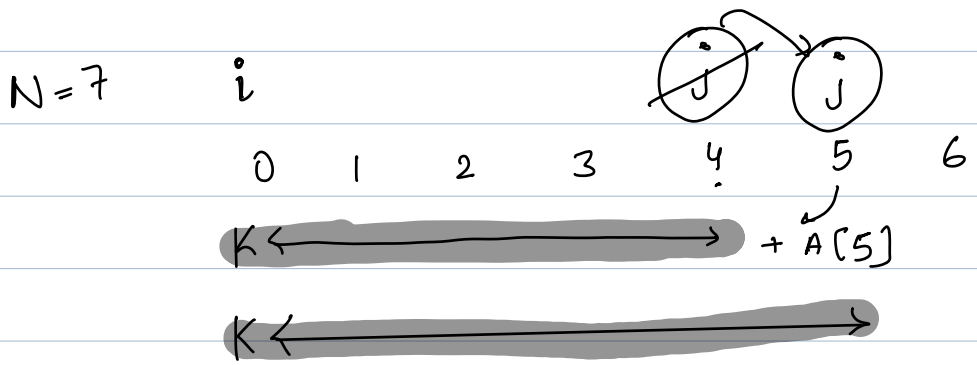
TC → O(N) + O(N²)

= O(N²)

SC → O(N)

Optimized Approach [using carry forward]

Observations :



i	j	[i j] ⇒ K
0	0	A[0]
0	1	A[0] + A[1]
0	2	A[0] + A[1] + A[2]
0	3	A[0] + A[1] + A[2] + A[3]
	4	+ A[4]

[0 4]

i	j	S = 0
0	0	S = S + A[j]
0	1	S = S + A[j]
0	2	S = S + A[j]
0	3	S = S + A[j]

```
max_sum = -∞
for(i=0; i<N; i++){
    s = 0
    for (j=i; j<N; j++) {
        s = s + A[j]
        if (max_sum < s) {
            max_sum = s
        }
    }
}
```

TC = $O(N^2)$

SC = $O(1)$

$\{ \overset{0}{2}, \overset{1}{-1}, \overset{2}{3}, \overset{3}{5} \}$

max_sum
~~$-\infty$~~
~~2~~
~~4~~
9

s = ~~0~~ ~~2~~ ~~1~~ ~~4~~ ~~9~~

~~0~~ ~~-1~~ ~~2~~ ~~7~~

~~0~~ ~~3~~ ~~8~~

~~0~~ 5

Further Reduction

↓

$O(N)$ → Kadane's

Advance Session.

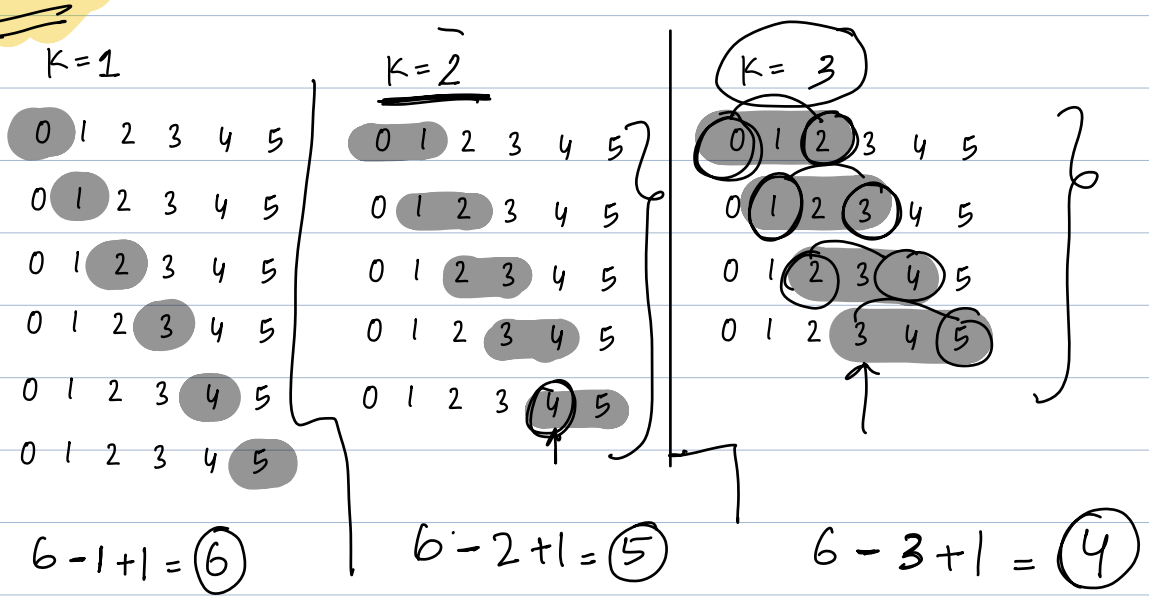
Break 6 min

9:20

Subarrays of length K

N=6 0 1 2 3 4 5

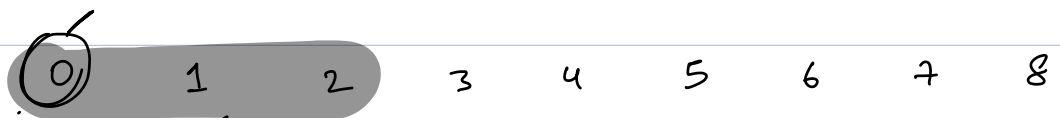
N-K+1

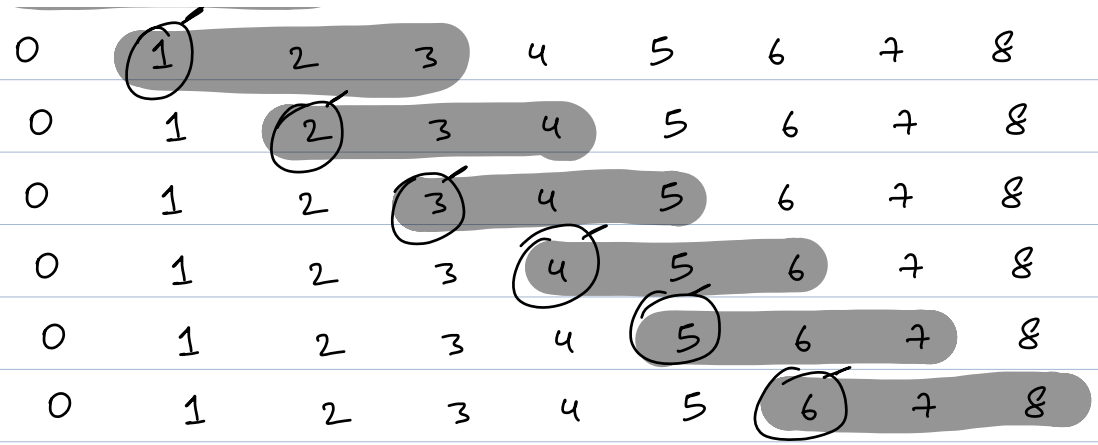


Given N & len=K, generalize total no. of subarrays of length K.

To a specific starting pt., there'll be a specific ending pt.

N=9
K=3





All the start pos → [0 6] → 6-0+1 = 7

Total no. of starting pts. of a subarray of len k = total no. of subarrays of len k

k	starting index of first window	starting index of last window
1	0	N-1
2	0	N-2
3	0	N-3
⋮		
k	[0	N-K]

[a b] → b-a+1

$[0 \quad N-K]$

$N-K-0+1 = N-K+1$

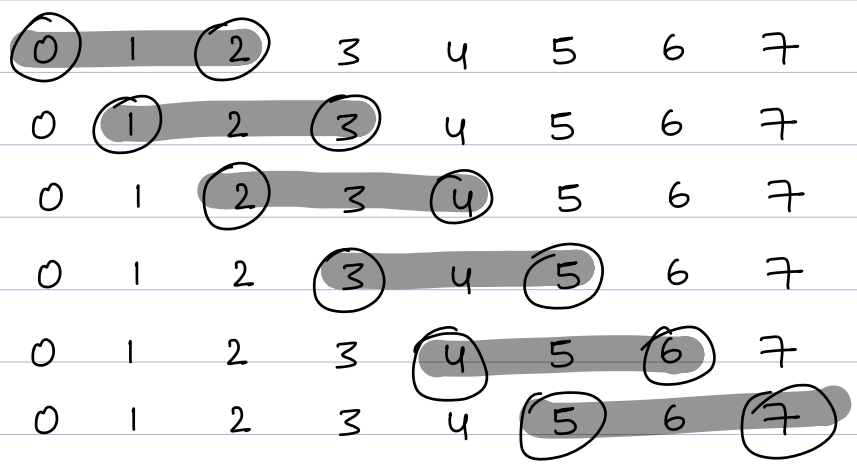
→ Total no. of subarrays len K

$N = 10$
$K = 4$

$10-4+1$
$= 7$

Que. Given N, len = K, print all the start & end indices of array of length K

N=8 K=3



start indices

K → [0 N-K]

K=3

i: 0 1 2 3 . . . N-K

0+3-1 = 2 | 1+3-1 = 3 | 2+3-1 = 4

$[i \quad e] = K$

$e - i + 1 = K$

$\boxed{e = i + K - 1}$

$[a \quad b] = b - a + 1$

```
for (i = 0; i <= N-K; i++) {
        j = i+K-1
        print (i & j)
}
```

// How to iterate over the subarrays of length K.

Qu. Max subarray sum with K.

A[] = { 4, -1, 3, 9, 2, 7 }

K = 3

4, -1, 3, 9, 2, 7	6
4, -1, 3, 9, 2, 7	11
4, -1, 3, 9, 2, 7	14
4, -1, 3, 9, 2, 7	(18) = Return

```
max_sum = -∞
for (i = 0; i <= N-K; i++) {          // N-K+1
    j = i+K-1
    sum = 0
    for (k = i; k <= j; k++) {        // K
        sum += A[k]
    }
    if (sum > max_sum) {
        max_sum = sum
    }
}
```

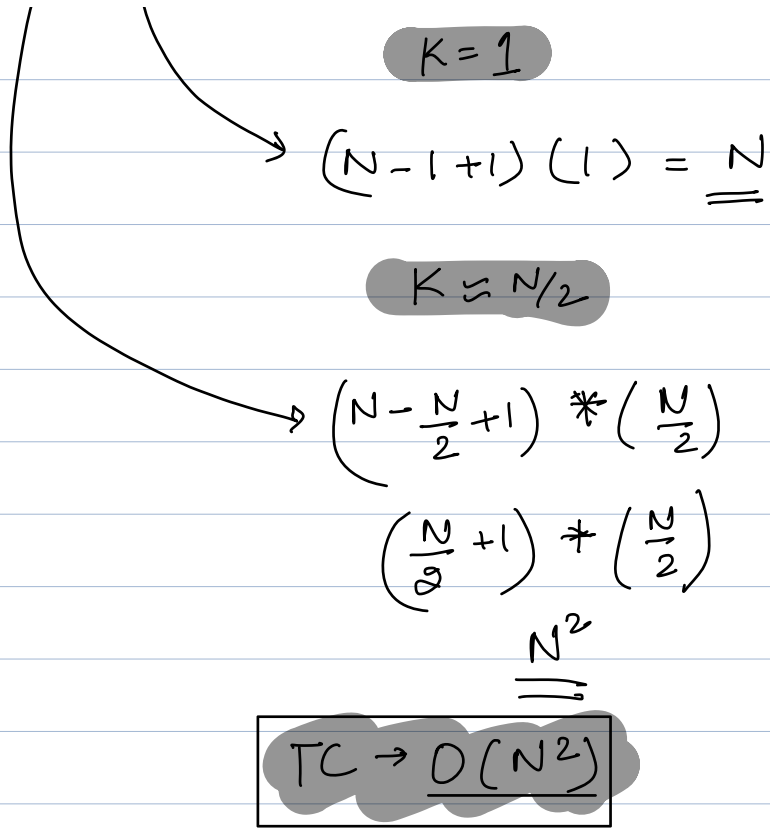
$NK - K^2 + 1$

$O(NK)$

Total: $(N-K+1) * (K)$

$K = N$

$(N - N + 1) * (N) = N$.



① Construct PF[ ]

② max_sum = −∞

for (i = 0 ; i <= N-K ; i++) {

    j = i+K-1

    sum = PF[j] - PF[i-1]

    if (max_sum < sum) {

        max_sum = sum

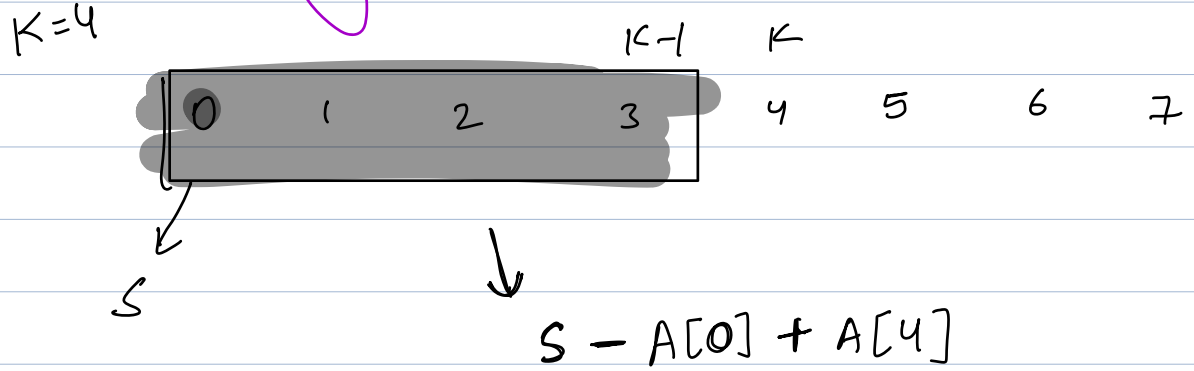
    }

}

TC → O(N) + O(N) = O(N)

SC = O(N)

Sliding Window Technique



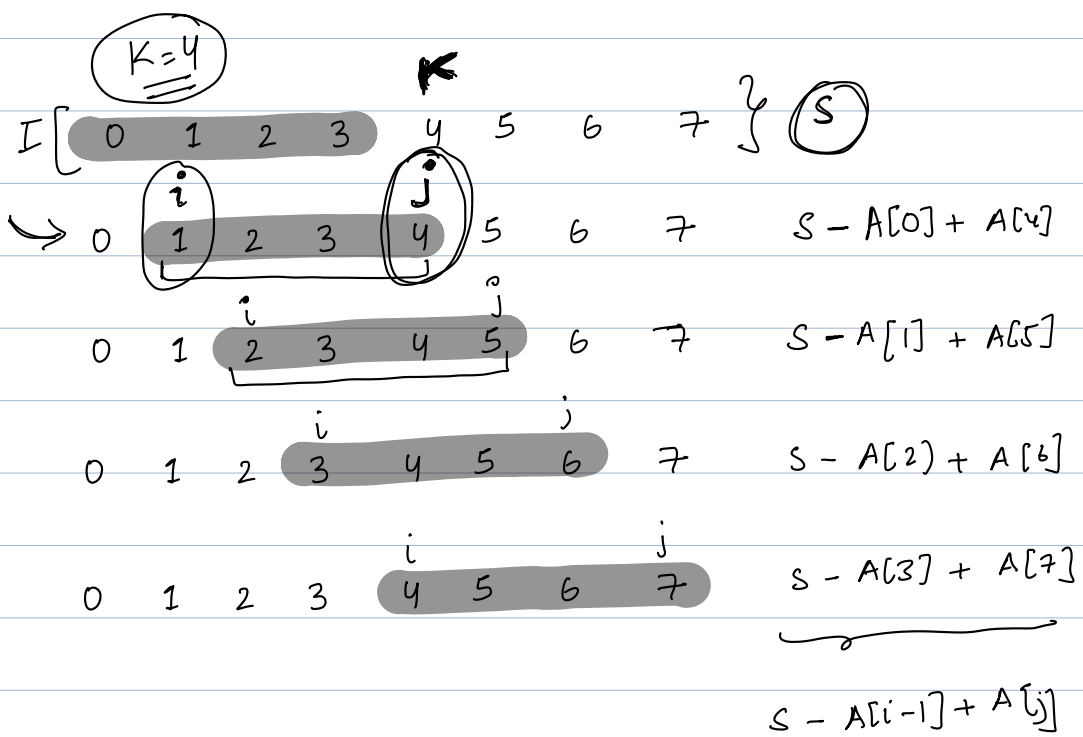
$S = A[0] + A[1] + A[2] + A[3]$ // [0 3]

$S = S - A[0] + A[4]$ // [1 4]

$S = S - A[1] + A[5]$ // [2 5]

$S = S - A[2] + A[6]$ // [3 6]

$S = S - A[3] + A[7]$ // [4 7]



Code

```
s = 0
for ( i = 0 ; i < K ; i++ ) {
    s += A[K]
}
max_sum = s

i = 1
j = i + K - 1 / K

while ( j < N ) {
    s = s - A[i-1] + A[j]
    if (max_sum < s) {
        max_sum = s
    }
    i++
    j++
}
```

K ← for loop

calc sum of first window

N-K ← while loop

calc answer of next windows

j → (K) N-1 ; N-1-K+1 = N-K

Total Iterations

$K + (N-K)$

N

TC $O(N)$

SC

$O(1)$

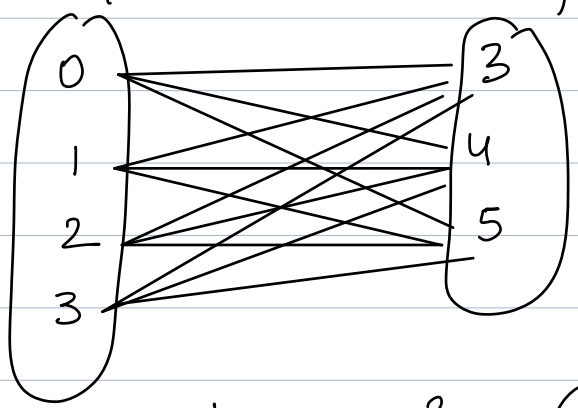
Q.11 Given an array, in how many subarray index 3 is present

	0	1	2	3	4	5
A[]: {	3,	-2,	4,	-1,	2,	6 }

$s <= e$

start pos

end pos



$4 * 3 = 12$

	0	1	2	3	4	5
{	3,	-2	4,	-1,	2,	6]

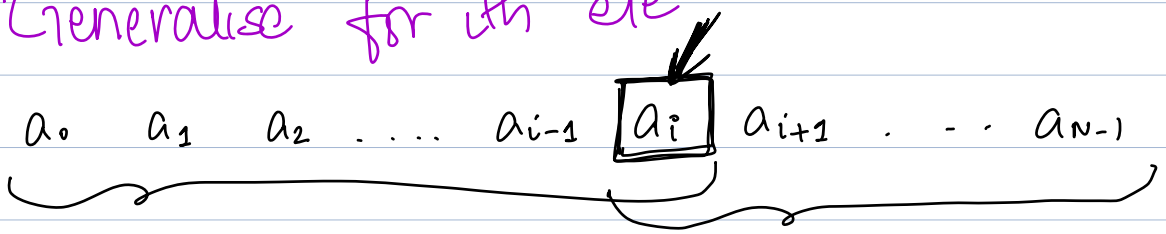
start: 0, 1

end: 1, 2, 3, 4, 5

$2 * 5$

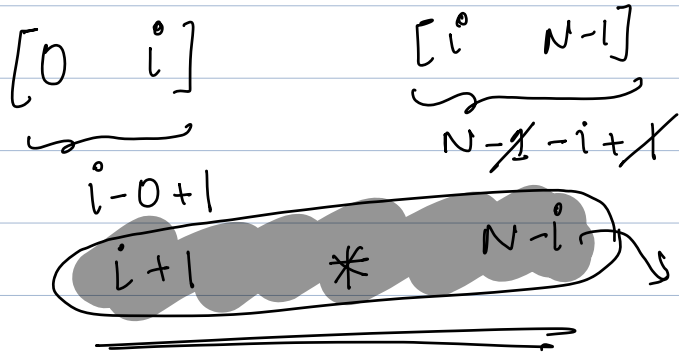
$= 10$

Generalise for ith ele



start ind

end ind



A[i] will be present in these many subarrays.

Contribution Technique

Google | FB

Que. A[] = {3, 2, 9}

Find sum of all the subarray sums.

[3] → 3

[3,2] → 5

[3,2,9] → 14

[2] → 2

[2,9] → 11

[9] → 9

44 ans

Brute Force

Notes:

Create all the subarrays & find their sum. Keep the total_sum variable & keep adding sum of individual sum of every subarray.

Observation

A = {3, 2, 9}

[3] → 3

[3,2] → 5

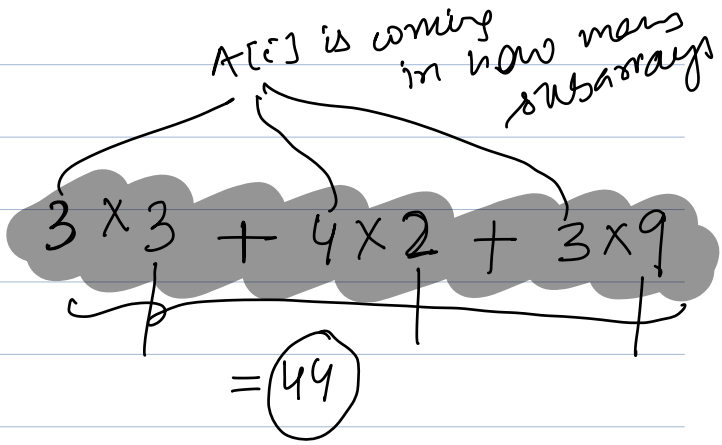
[3,2,9] → 14

[2] → 2

[2,9] → 11

[9] → 9

44




```
total_sum = 0
for ( i = 0 ; i < N ; i++ ) {
    count = // In how many subarrays A[i] will be present
    contribution = count X A[i]
    total_sum += contribution
}
return sum
```

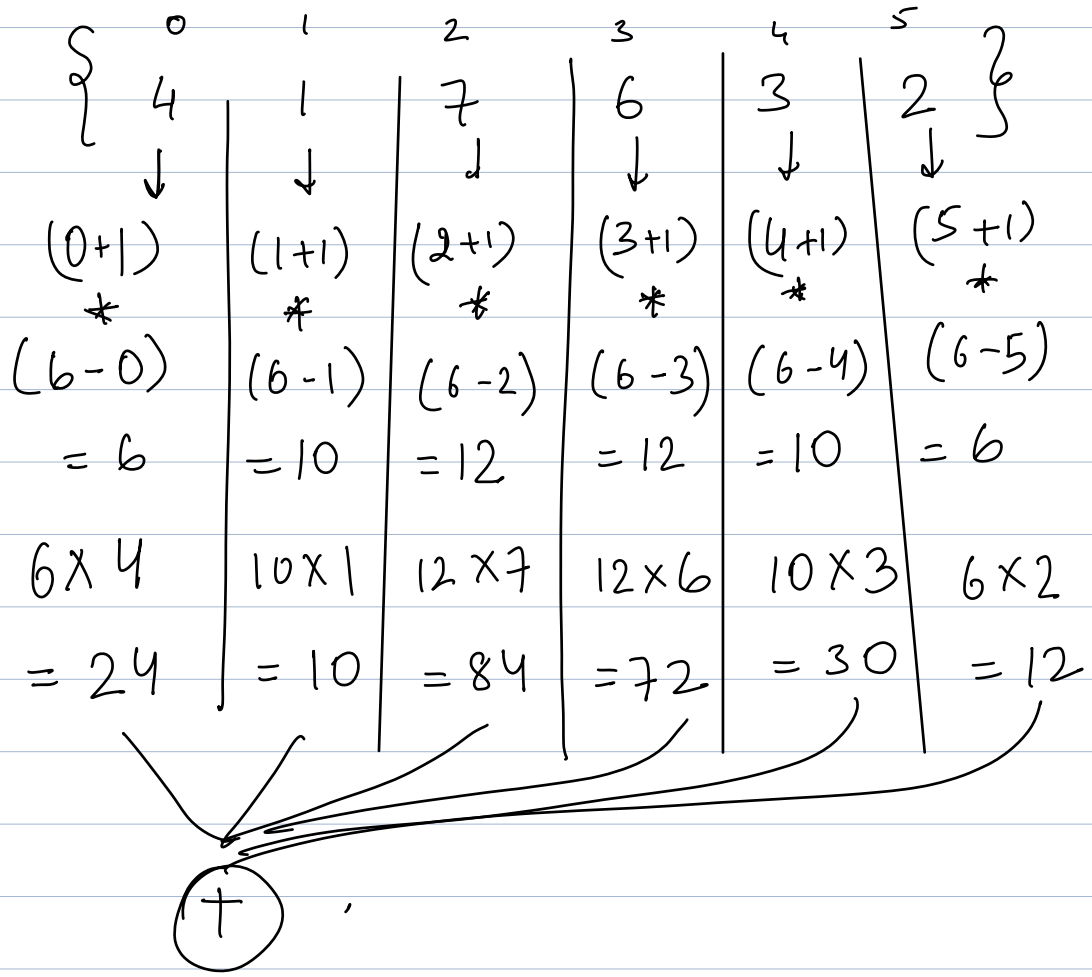
count = (i+1) X (N-i)

TC → O(N)

SC → O(1)

DRY RUN

$(i+1) * (N-i)$


0 1 2 3 4 5
4 1 7 6 3 2
(0+1) (1+1) (2+1) (3+1) (4+1) (5+1)
(6-0) (6-1) (6-2) (6-3) (6-4) (6-5)
= 6 = 10 = 12 = 12 = 10 = 6
6X4 10X1 12X7 12X6 10X3 6X2
= 24 = 10 = 84 = 72 = 30 = 12
+