Sum of every subarray

Contribution Technique

Sliding Window (Max subarray sum of size k)

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

1. Given an array of integers, find total sum of all possible subarrays.

Google, Facebook

0 1 2

A = [3 2 5]   ans : 32

[3]   3

[3 2]   3+2

[3 2 5]   3+2+5

[2]   2

[2 5]   2+5

[5]   5

sum

3 +
5 +
10 +
2 +
7 +
5

32

$3(3) + 2(4) + 5(3)$

$9 + 8 + 15 = 32$

BF : Go to all subarrays and calculate their sum (iterate from s to e), add it to total sum.

```
int totalsum =0
for (s=0; s<n; s++){
    for (e=s; e<n; e++){
        //(s e)
        int sum =0
        for (int i=s; i ≤e ; i++){
            sum = sum + A[i]
        totalsum = totalsum +sum
return totalsum
```

TC : $O(N^3)$

SC : $O(1)$

Approach 2: Go to all subarrays and calculate sum (using pf[]), add to totalsum

① Build pf[] → N

②
```
int totalsum =0
for (s=0; s<n; s++){                    N^2
    for (e=s; e<n; e++){
        //(s e)
        int sum =0
        if (s==0){
            sum = pf[e]
        else {
            sum = pf[e] - pf[s-1]

        totalsum = totalsum + sum
    }
}
return totalsum
```

0 — R pf[R]
L — R

TC: O(N + N^2) = O(N^2)

SC: O(N) → O(1)

O(N) ↓ pf[ ]

O(1) ↓ Modify same array to store pf[ ]

Approach 3: Go to all subarrays and calculate sum (without pf[ ])

carry forward sum

$A[\,] = \langle \overset{0}{-4}, \overset{1}{1}, \overset{2}{3}, \overset{3}{2} \rangle$

s	e	sum	Total
0	0	A[0]	-4
0	1	A[0]+A[1]	-4+1=-3
0	2	A[0]+A[1]+A[2]	-3+3=0
0	3		0+2=2

s	e	sum=0
1	1	0+1=1
1	2	1+3=4
1	3	4+2=6

```
int totalsum = 0
for (s=0; s<n; s++) {
    int sum=0
    for (e=s; e<n; e++) {
        sum = sum + A[e]
        totalsum = totalsum + sum
    }
}
```

1 subarray → O(1)

N^2 subarray → $O(N^2)$

TC: $O(N^2)$

SC: O(1)

$A = < \overset{0}{-4}, \overset{1}{1}, \overset{2}{3}, \overset{3}{2} >$

s	e	sum=0	totalsum = 0
0	0	-4	-4
	1	-3 +3	-7
	2	0	-7
	3	2	-5

s	e	sum = 0	total = -5
1	1	1 +3	-4
	2	4	0
	3	6	6

s	e	sum = 0	total = 6
2	2	3	9
	3	5	14

Approach 4 : Contribution Technique

We are going to every element, get their contribution and add it to total sum.

totalSum = contri of 0th ele + contri of 1st ele + ... + contri of (N-1)th ele contribution of ith ele = A[i] × no. of subarrays in which A[i] is present

1. In how many subarrays element at idx 1 is present?

	0	1	2	3	4	5
A :	[3	−2	4	−1	2	6]

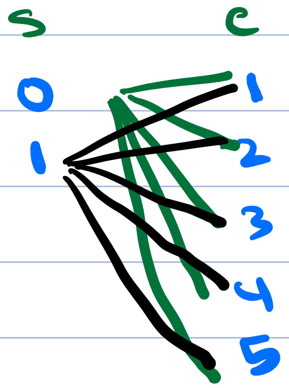
ans = 10

(i+1) × (N−i)

(1+1) × (6 −1)

2 × 5

= 10



0, 1	1, 1
0, 2	1, 2
0, 3	1, 3
0, 4	1, 4
0, 5	1, 5

2 × 5 = 10 subarrays

2. In how many subarrays element at idx 2 is present?

	0	1	2	3	4	5
A :	[3	−2	4	−1	2	6]

ans = 12

(i+1) × (N−i)

(2+1) × (6−2)

3 × 4

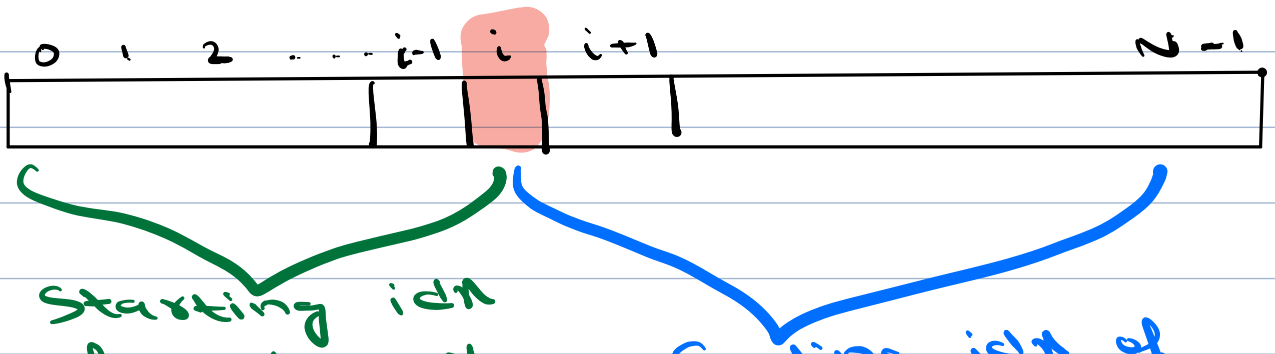
= 12

s	e
0	2
1	3
2	4
	5

$\overset{s}{3} \times \overset{e}{4}$ = 12 subarrays

Generalized calculation

Arr size → N



Starting idx of subarray [0 → i)

(i+1)

Ending idx of subarray [i → N-1]

(N-i)

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

$[0 \quad i] =$
$i - 0 + 1$

$[i \quad N\text{-}1] =$
$N - \cancel{1} - i + \cancel{1}$

Total subarrays which contain ith element = (i+1) × (N-i)

Contribution of ith element = A[i] × (i+1) × (N-i)

```
int totalsum = 0
for (i=0 ; i<n ; i++){
    int contri = A[i] × (i+1) × (N-i)
    totalsum = totalsum + contri
}
return totalsum
```

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

10:34

Sum of all subarrays

Sum of all
Product of all
all of all

Think about contribution technique

Q. Find total no. of subarrays of length K

Subarray of fixed length is called window.

Length	Start of 1st window	Start of last window	# windows
1	0	N-1	N
2	0	N-2	N-1
3	0	N-3	N-2
⋮			
K	0	N-K	N-K+1

Total no. of subarrays of len K = N-K+1

N=7 K=4, total no. of subarrays of len K

N-K+1 = 7-4+1 = 4

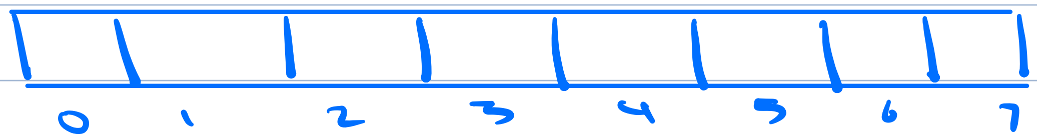
Q. Given an array of size N, print start and end indices of subarrays of length k.

$[s \rightarrow e]$

$e - s + 1$

$\boxed{k-1-0+1} = k$

N=8 k=3



// N, k

s	e
0	2
1	3
2	4
3	5
4	6
5	7
6	8

+1 +1 STOP

s	e
0	k-1
1	k
2	k+1
3	k+2
⋮	
N-k	N-1

+1 +1

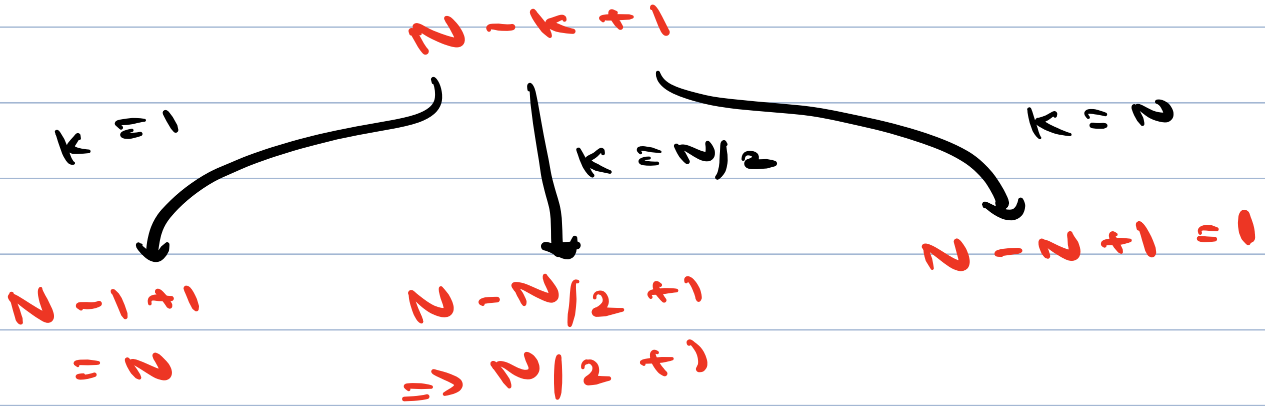
```
// N, k
int s=0, e=k-1
while( e < N ){
        print(s,e)
        s++  e++
}
```

TC : O(N-K+1)

SC : O(1)

$\boxed{1 \leq K \leq N}$

TC ≃ O(N)



Given an array of N elements, print maximum subarray sum with length = k.

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

N = 10 , k = 5   ans = 16

s	e		sum
0	4	(-3) + 4 + (-2) + 5 + 3	7
1	5	4 + (-2) + 5 + 3 + (-2)	8
2	6	(-2) + 5 + 3 + (-2) + 8	12
3	7	5 + 3 + (-2) + 8 + 2	16
4	8	3 + (-2) + 8 + 2 + (-1)	10
5	9	(-2) + 8 + 2 + (-1) + 4	11

Approach 1 : Iterate on all subarrays of len k, get sum and get max
BF

```
// N, k                                              -∞
int  s=0 , e=k-1 , maxSum = INT_MIN
while( e < N ){
        // (s ,e)
        int  sum = 0
        for (i=s ; i <= e ; i++){
            sum = sum + A[i]
        }

        maxSum = max (maxSum , sum)
        s++  e++
}
```

1 subarray → $O(k)$

$N-k+1$ subarrays → $(N-k+1)k$

TC : $O((N-K+1)K) \simeq O(N^2)$

$1 \leq K \leq N$

$K=1$

$(N-1+1)\times 1$

$= N$

$O(N)$

$K=N/2$

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

$\Rightarrow (N/2+1)(N/2)$

$\Rightarrow \frac{N^2}{4} + N/2$

$\simeq O(N^2)$

$K=N$

$(N-N+1)N$

$\Rightarrow N$

① Build pf[] → N

```
// N, k                                            -∞
int s=0, e=k-1, maxSum=INT_MIN
while( e < N ){
    // (s,e)
    int sum=0

    // formula

    maxSum = max (maxSum, sum)
    s++ e++
}
```

TC : $O(N + N-K+1)$

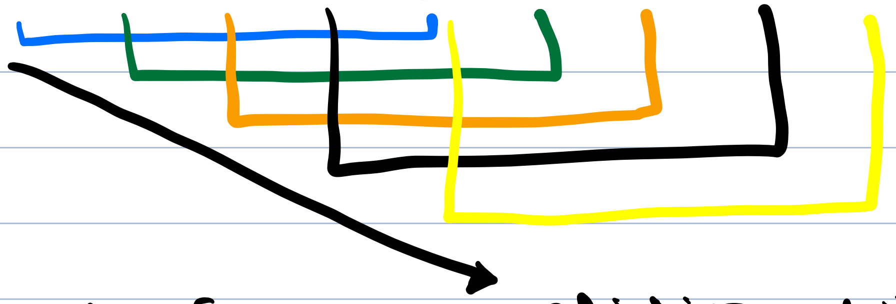
$= O(2N-K+1)$

$\simeq O(N)$

SC : $O(N) \longrightarrow$ pf[]

Approach 3 : Reduce SC ?

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



ans = 16

N = 10 k = 5 sliding window

0 → 4 = 7

1 → 5 = 7 + (-2) - (-3) = 8

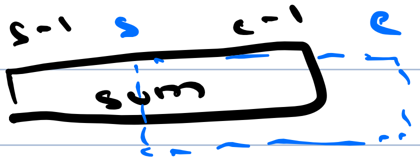
= sum + A[e] - A[s-1]

2 → 6 = 8 + 8 - 4 = 12

3 → 7 = 12 + 2 - (-2) = 16

4 → 8 = 16 + (-1) - 5 = 10

5 → 9 = 10 + 4 - 3 = 11



s-1	e-1	sum
s	e	sum + A[e] - A[s-1]

```
// A[], N, K
                                               -∞
int s=0, e=k-1, maxSum=INT_MIN
int sum =0
for (int i=s; i<=e; i++){          } K
    sum =sum + A[i]

maxSum = max(maxSum, sum)
s++  e++

while( e<N ){    → N-K
    // (s,e)
    sum = sum + A[e] - A[s-1]
    maxSum = max (maxSum, sum)
    s++  e++

return maxSum
```

TC : O(K + N-K)
= O(N)

SC : O(1)