Contribution Technique

Sliding Window

① subarray - contiguous part of array

N=5 < 2, 6, -1, 7, 8 > $cnt = \frac{5 \times 6}{2} = 15$

② < 2, 4, 1, 6, -3, 7 8, 4 >

③ No. of subarrays in array of size N $= \frac{N(N+1)}{2}$

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

```
       0  1  2
A = [3  2  5]        ans : 32
```

		sum
[3]	3	3 +
[3 2]	3 + 2	5 +
[3 2 5]	3 + 2 + 5	10 +
[2]	2	2 +
[2 5]	2 + 5	7 +
[5]	5	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 + ar[i]
        }
        totalsum += sum
        // totalsum = totalsum + sum
    }
}
```

TC : O(N^3)

SC : O(1)

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

```
① pf[N]  → N

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

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

$= O(N^2)$

SC: $O(N)$ → pf[] → $O(1)$ if you modify the array to store pf[]

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

carry forward sum

$$\begin{array}{ccccc} & 0 & 1 & 2 & 3 \\ A[\,] = < & -4, & 1, & 3, & 2 > \end{array}$$

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

s	e	
		0 +A[1]
1	1	1 +A[2]
1	2	4 +A[3]
1	3	6

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

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	-7
	2	0	-7
	3	2	-5
		sum=0	
1	1	1	-4
	2	4	0
	3	6	6
		sum=0	
2	2	3	9
	3	5	14
		sum=0	
3	3	2	16

Approach 4 : Contribution Technique

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

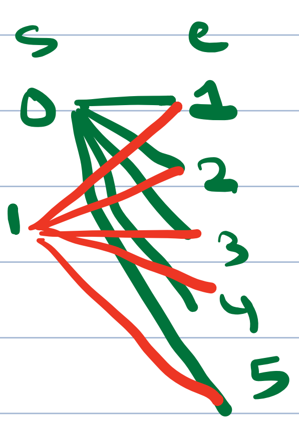
totalSum = contri of 0th ele + contri of 1st ele + ... + contri of (N-1)th ele contribution of ith ele = A[i] × no. of subarrays in which A[i] is present

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

$$A : \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ [3 & -2 & 4 & -1 & 2 & 6] \end{array}$$

s, e

ans = 10



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

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

2 × 5 = 10 subarrays

2. In how many subarrays element at idx 2 is present ? (i+1) × (N-i)

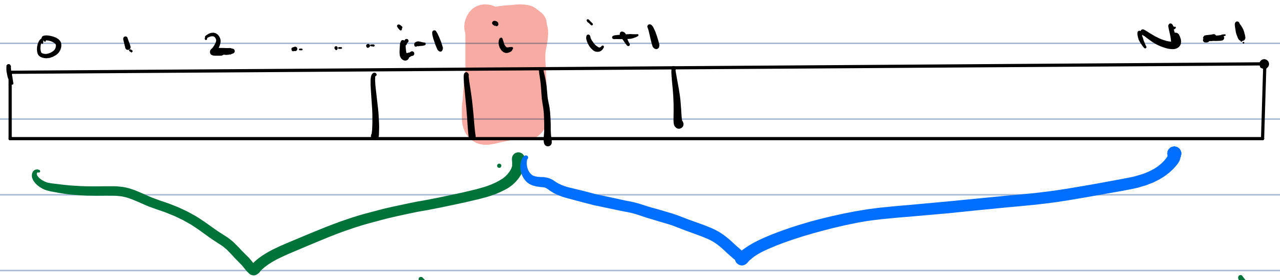
$$A : \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ [3 & -2 & 4 & -1 & 2 & 6] \end{array}$$

ans = 12

s e
0 2
1 3
2 4
5

s e
3 × 4 = 12 subarrays

Generalized calculation   Arr size → N



Starting idx of subarray
[0 → i]
$\boxed{i+1}$

Ending idx of subarray
[i → N-1]
$\boxed{N-i}$

[a b] = b-a+1
$N-\cancel{1}-i+\cancel{1}$

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

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

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

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

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

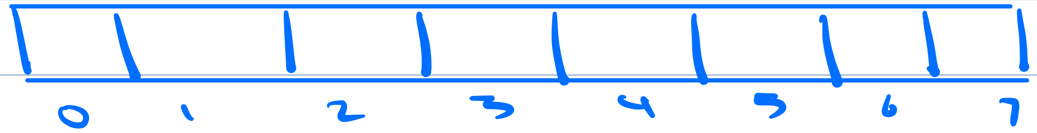
Total no. of subarrays of len K = N-K+1

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.

N = 8 K = 3



s	e
0	2
1	3
2	4
3	5
4	6
5	7
+1	+1
$\boxed{6}$	$\boxed{8}$

Stop

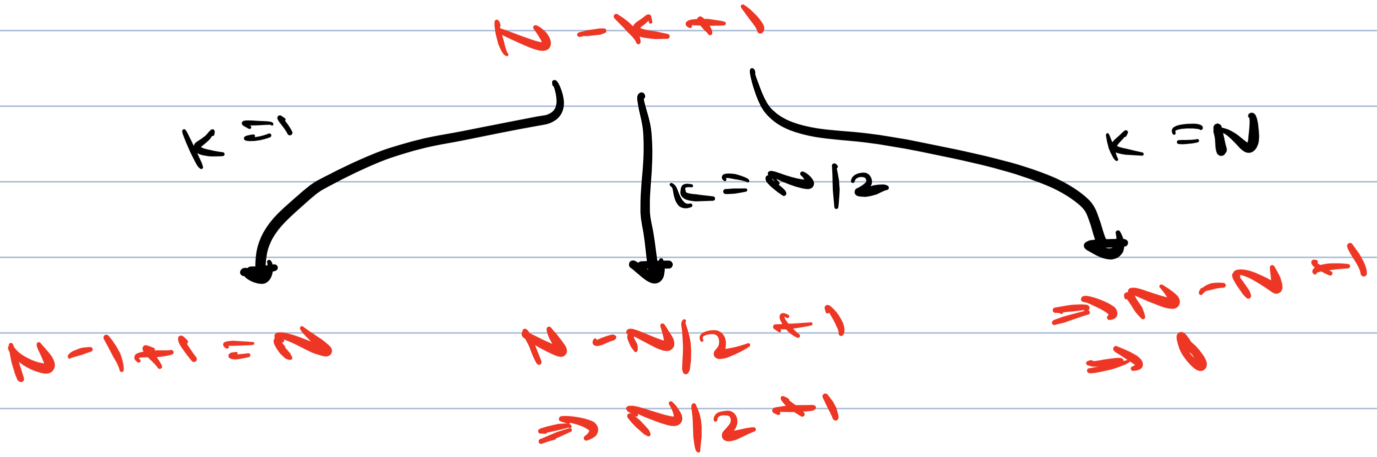
s	e
0	K-1
1	K
2	K+1
3	K + 2
⋮	⋮
__	N-1

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

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

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

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



10:35

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

$$\begin{array}{ccccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ arr = [& -3 & 4 & -2 & 5 & 3 & -2 & 8 & 2 & -1 & 4] \end{array}$$

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 += a[i]

      maxSum = max (maxSum, sum)

       s++  e++

return  maxSum
```

1 subarray → O(k)

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

$k=1$

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

$\Rightarrow N$

$k=N/2$

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

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

$O(N^2)$

$k=N$

$(N-N+1)N$

$\Rightarrow N$

$SC: O(1)$

```
① Build pf [ ]

② int max Sum = INT_MIN (-∞)

   int s=0, e=k-1
   while (e<n){        // [s   e]
       if (s==0)
               sum = pf[e]
       else
               sum = pf[e] - pf[s-1]
       maxSum = max(maxSum, sum)

           s++    e++
   }
```

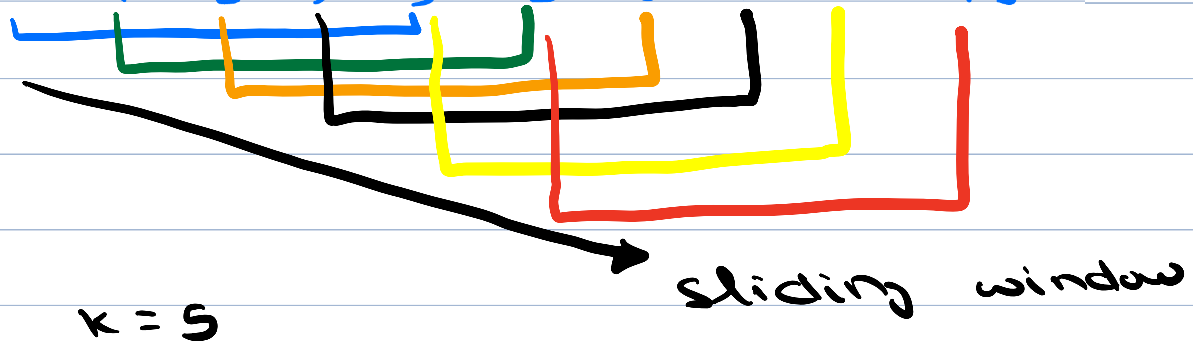
1 subarray → O(1)

TC : O(N-K+1) ≃ O(N)

SC : O(N) → 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]



N = 10   k = 5

0 → 4 = 7

1 → 5 = 7 + (-2) - (-3) = 8

sum + a[5] - a[0]

2 → 6 = 8 + 8 - 4 = 12

= sum + a[6] - a[1]

3 → 7 = 12 + 2 - (-2) = 16

sum + a[7] - a[2]

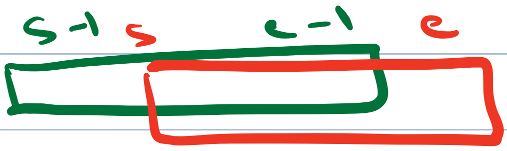
4 → 8 = 16 + (-1) - 5 = 10

sum + a[8] - a[3]

5 → 9 = 10 + 4 - 3 = 11

sum + a[9] - a[4]

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



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

maxSum = max (maxSum, sum)


s++   e++
while(e<N){
N-K     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)

Adding an item at end → O(1)

Amortised TC → O(1)

Java

```
Arraylist <String>  a = new  ArrayList<>()
a.add ("50")        // 50 is inserted at end
a.clear()
for (int i=0 ; i<a.size(); i++) {
     System.out.print (a.get(i) +" ");
},
```

Python

```
a = [ ]
a.append ("Orange")     // Added orange at end
a.clear ()
for i in range(len(a)):
        print ( a[i])
```

C++

```
vector <int>  a
a.push_back (60)
a.clear()
for (int i=0 ; i<a.size() ; i++)
          cout << a[i]
```

A = 1, 4, 5, 2, 4