Today's content.

(i) Subarrays recap.

(ii) Total no. of subarrays.

(iii) Print a given subarray

(iv) Print each subarray.

(v) Print each subarray sum.

↳ 3 Approaches

(vi) Total sum of all subarrays.

↳ 2 Approaches.

Subarray.

-> Continuous part of an array.

-> Single element → Yes

-> Empty element → No.

-> length of subarray [i j] -> [j-i+1]

List all the subarrays.

Ex: ar[4] : $\begin{array}{cccc} 0 & 1 & 2 & 3 \\ 2 & 6 & 3 & 9 \end{array}$ →

subarrays. index	subarray.	
[0 0]	[2]	4 ✓
[0 1]	[2 6]	
[0 2]	[2 6 3]	
[0 3]	[2 6 3 9]	
[1 1]	[6]	3 ✓
[1 2]	[6 3]	
[1 3]	[6 3 9].	
[2 2]	[3]	2 ✓
[2 3]	[3 9]	
[3 3]	[9].	1 ✓

ar[4] : $\begin{array}{cccc} 0 & 1 & 2 & 3 \\ 2 & 6 & 3 & 9 \end{array}$

Total subarrays = 4+3+2+1
= $\underline{\underline{10}}$.

ar[N] : $\begin{array}{ccccccc} 0 & 1 & 2 & 3 & & (N-2) & (N-1). \\ \underline{a_0} & a_1 & a_2 & a_3 & -- & a_{N-1} & a_N. \end{array}$

i	i	i	
[0. 0]	[1 1]	[2 2]	[N-1 N-1].

[0 1]

[0 2]

¦

[0 N-1].   [1 N-1].   [2 N-1]   1.

$\# N + \# N-1. + \# N-2 + \quad 1. = \frac{N(N+1)}{2}.$

If an array of size N exists, then it'll have $\frac{N(N+1)}{2}$ subarrays.

1Q: Given an array, print the subarray from [s e], s≤e.

$$\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ [4 & 5 & 6 & 7 & 8 & 9]. \end{array}$$

```
def printSubarray(arr, s, e)
{
        for i in range(s,e+1)
            print(arr[i])

}
```

2Q: Given an array, print each and every subarray.

$$ar[4] : \begin{array}{cccc} 0 & 1 & 2 & 3 \\ [6 & 8 & -1 & 7] \end{array}$$

Subarrays index	printing every element.
[0 0]	6 .
[0 1]	6 8
[0 2]	6 8 -1
[0 3]	6 8 -1 7
[1 1]	8
[1 2]	8 -1
[1 3]	8 -1 7

Psuedocode.

```
for i in range (0, N).
    for j in range (i, N).
        // print all elements from [i j].
        for k in range(i, j+1)
            print(arr[k]).
        print(newLine)
```

2 { [2 2] -1

[2 3] -1 7

3 [3 3] 7.

TC: O(N^3).
SC: O(1).

3Q: Given n array elements, print each subarray sum.

arr[4] : { 6 8 -1 7 }

0	1	2	3
6	8	-1	7

Subarray	sum.
[0 0]	
[0 1]	
[0 2]	
[0 3]	
[1 1]	
[1 2]	
[1 3]	
[2 2]	
[2 3]	
[3 3]	

Brute force.

```
def printSum (arr, N)
{
    for i in range (0, N)
        for j in range(i, N).
            // sum of subarray from i,j.
            sum = 0 .
            for k in range ( i, j+1)
                sum = sum + arr[k].
            print (sum) // new line .
} .
```

TC: O(N^3).
SC: O(1).

Optimization.

```
def  printSum (arr, N)
{
    // Calculate  pf array (populate it). ——> O(N).
    for  i  in range (0, N)
        for  j  in range(i, N).
            // Sum of subarray from i,j.
            sum = 0 .
            for  k  in range (i, j+1)
                sum = sum + arr[k].
            print (sum) // new line .
} .
```

Re-cap.

$$\text{Sum}[i\ \ j] \rightarrow \begin{cases} i==0, & pf[j]. \\ i!=0, & pf[j]-pf(i-1) \end{cases}$$

Replace it with.

```
if (i==0)
    print(pf [j]
else
    print (pf [j]-pf (i-1)].
```

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

SC: $O(N)$.

Optimize space but do not modify original array.

Q. Given ar[N], print all subarrays sum starting at index 3.

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

Subarray.	sum.
[3 3]	7.
(3 4)	16
(3 5)	20
[3 6]	23
(3 7)	25
[3 8]	35
(3 9)	41.

```
def printSubarraySumStartsAtK(arr, K, N)
{
        sum=0.
        for i in range(K,N)
            sum = sum+arr[i]
            print (sum)
}
```

Optimization of space: Think in lines above question.

```
def printSum (arr, N)
{
    for i in range (0, N)
        // sum of subarrays which starts with index 'i',
        Sum =0
        for j in range (i,N)
            sum = sum+arr[j]
            print(sum)
}
```

TC: O(N^2).

SC: O(1).

Breaks.

10:10:00

10:20:00.

Q: Given an array, return sum of all subarrays sum

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

Subarray	sum
[0 0]	6
[0 1]	14
[0 2]	13
[0 3]	20
[1 1]	8
[1 2]	7
[1 3]	14
[2 2]	-1
[2 3]	6
[3 3]	7.

Sum= 94.

Approaches.

1. 3 Nested loops. $O(N^3)$.

2. Prefix sum $O(N^2)$, $O(N)$.

3. Carry forward $O(N^2)$, $O(1)$.

Code for this.

```
def printSum (arr, N)
{   total = 0,
    for i in range (0, N)
        // sum of subarrays which starts with index 'i',
        Sum = 0
         for j in range (i, N)
             sum = sum + arr[j]
             total = total + sum.
}
```

HW.

version.

simply ignore sum.

total = total + arr[j].

Hint

$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ ar[\,] : & 4 & 3 & 7. \end{array}$$

Subarrays.

[0 0] : {4} +

[0 1] : {4, 3}. +

[0 2] : {4, 3, 7}.+

[1 1] : {3} +

[1 2] : {3, 7} +

[2 2] : {7}.+

sum = 4 * 3 + 3 * 4 + 7 * 3 = 12 + 12 + 21 = 45.

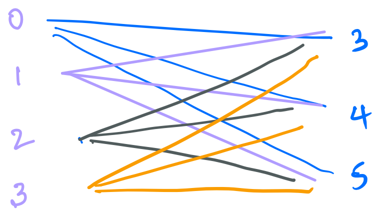
Final soln.

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

" In total sum of all subarrays, how many times element at index 3 is present".

4 3

s (till 3) e (after 3 .include 3).

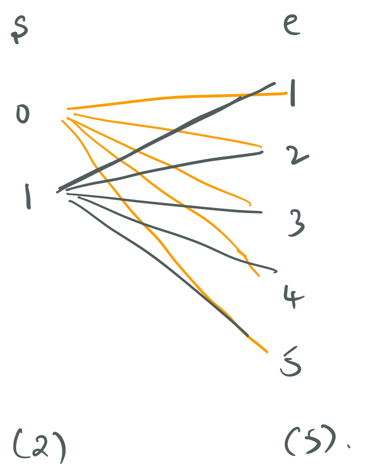


out of $\frac{N(N+1)}{2}$ subarrays,

4 * 3 = 12 subarrays have index element '3'.

	0	1	2	3	4	5
ar[6] :	{ 3	-2	4	-1	2	6 }.

How many times index 1 is present.



2*5 = 10 subarrays which contains element at index 1.

if I take index 'i'.

s.	e
0	i
1	i+1
2	i+2
3	⋮
⋮	⋮
i	[N-1]

[0 i] = i+1.

(a b) = b-a+1

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

= N-i.

For any index 'i', This will come $(i+1)*(N-i)$ times in all subarrays.

N= 4,	0	1	2	3.
	[6	8	-1	7]
i+1	1	2	3	4
N-i	4	3	2	1
.(i+1)(N-i)	4	6	6	4.

Total sum. $= 6*4+8*6+(-1)*6+7*4$

$= 24+48-6+28. = 94.$

Psuedo code:

```
def totalsubarraysum(arr, N)
{
    sum = 0.
    for i in range(0, N)
    |    sum = sum + (i+1)(N-i) * arr[i]
    return sum
}
```

No.of times

TC: O(N)

SC: O(1)