June 28, 2025

CONTRIBUTION TECHNIQUE

QUES #1: For a given array, find the sum total of sum of each possible subarray.

[Subarray can be taken from continous sequence]

Ex: [1, 3, 2]

⇒ Sum (Sum of indv. subarray)

Possible subarrays	Sum
1	1
3	3
2	2
1 3	4
3 2	5
1 3 2	6
Sum Total =	21

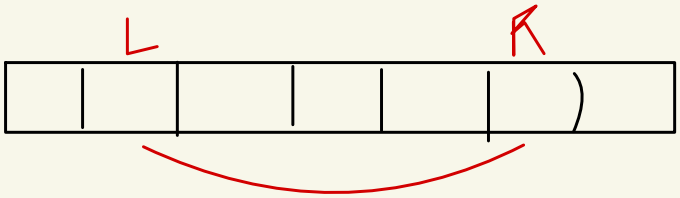
Constraints

① $N \leq 10^3$

② $N \leq 10^5$

Solution:

Brute force:




```
Ans = 0
for(L=0; L<N; L++) {
    for(R=L; R<N; R++) {
```

Here, we are using the outer loops to define a range of L to R. Then we loop from L to R through the third loop, adding all elements in between.

```
{
    sum = 0
    for (i = L; i <= R; i++)        // L → R
    {
        sum += arr[i];
    }
    ans += sum
}
```

→ $O(N^3)$

$O(N^3)$: Not enough to pass the constraints. $N \le 10^3$, $N \le 10^5$

Another approach: Instead of third loop, keep one sum and use it.

Example:

→ [1, 3, 2]

L = 0
R = 0
sum += a[0]
sum = 1

→ L = 0, R = 1 [1, 3, 2]
sum += a[1] = 1 + 3
sum = 4

→ L = 0
R = 2
sum += a[2]
sum = 4 + 2
= 6

and so on...

L = 0, we have all the sum 1 + (1+3) + (1+3+2)

Algo for this approach:

```
Ans = 0
for(L = 0 ; L < N ; L++)
    Sum = 0
    for(R = L ; R < N ; R++)
        Sum += arr[R];
        ans += Sum
```

Complexity : $O(N^2)$

$N \leq 10^3$ will pass here but we will be unable to solve it for (2).

If we go through every subarray, there is no way to do better than $O(N^2)$ because the no. of subarray itself is $O(N^2)$.

⇒ If there is array of size N, How many subarrays does it contain?

Subarray of size = 1	: N
Subarray of size = 2	: N-1
subarray of size = 3	: N-2
subarray of size = N	: 1
Total =	$\frac{N(N+1)}{2}$

→ This tells us that the no. of subarrays is $O(N^2)$ and our current approach includes traving every subarray...

With this approach, we can't code a soln for (2) constraints

Hence, Instead of looking at every subarray contribution, we will now look at another approach where we will look at contribution of every number instead.

New approach to tackle 2) constraint

1		
	3	
		2
1	3	
	3	2
1	3	2

```
ans = 0
for (i → 0 to N) {
    ans += arr[i] * (i+1) * (N-i)
}
```

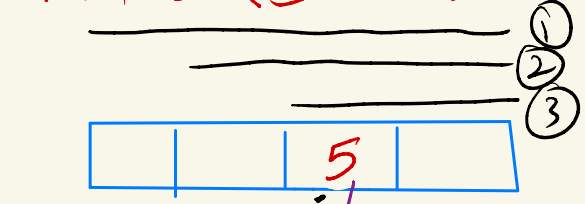
TC: O(N)

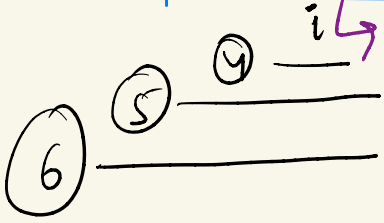
Here, we are calculating the contribution of i-th element

Sum of columns

$3\times1 + 4\times3 + 2\times3 = 21$ (Ans)

Sol^r

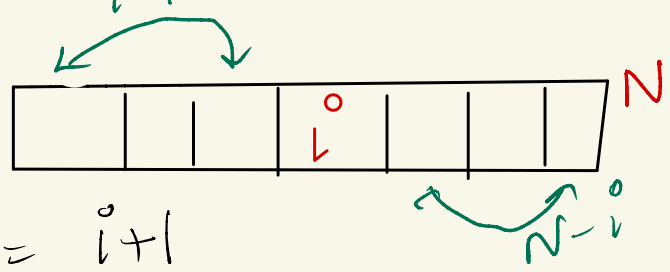




How many subarrays will contain this element?

Start before 5 & end after 5

$\Rightarrow 3 \times 2 = 6$ subarrays will contain 5.

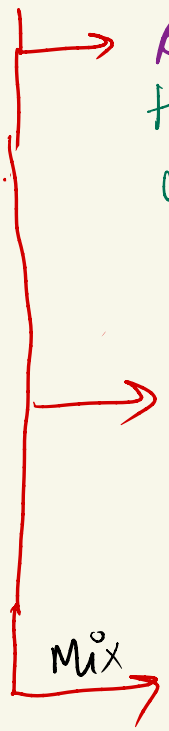


Before = $i+1$

After = $((N+1)-(i+1)) = N-i$

ith element will come = Before × After

★ Contribution



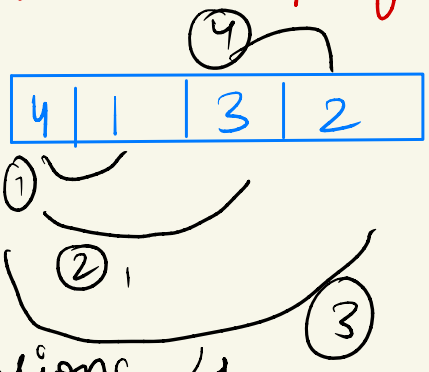
→ Atomic Item contribution
Here, we are looking at individual contribution of an element.
Ex: Previous problem Q1>, Q2>

→ Extended End Contribution Q3>

Mix → Every [Start | End] contribution uses ST2 / Two pointer.

Ques #2 Calculate # of inversions of a given array

Ex:



inversions = 4

→ (i, j) : $i < j$, $a[i] > a[j]$

Σ [# of inversion of each subarray]

$N \leq 10^3$

Working Example :

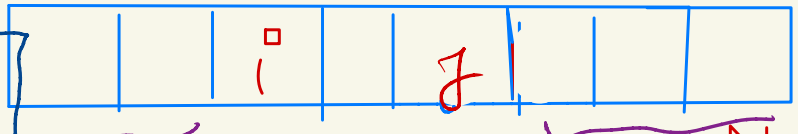
1 3 2

Subarray $(i<j)$: $a[i] > a[j]$

	# of inversion
1 3	→ 0 (1>3 false)
3 2	→ 1 (3>2 true)
1 3 2	→ 1 (3>2)
	Ans = 2

Atomic Item = Inversion

What subarray contains a particular inversion?



How many options to start = i+1

$\Sigma\Sigma (i+1)(N-j)$ $a[i] > a[j]$

```
ans = 0;
for(i -> 0 to N) {
    for(j -> i+1 to N) {
        if(a[i] > a[j]) {
            ans += (i+1) * (n-j)
```

3 3 3 SOLUTION

#3 Find sum of product of every subarray

Ex [1, 3, 2]

Generalizing the example below:

a] b] c] d]

a | +b | +c | +d

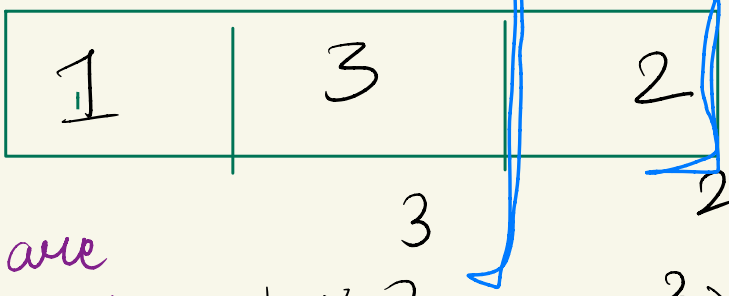
| | +b | +bXc | +cXd

| | | aXbXc | : +bXcXd

| | | | aXbXcXd

1			1
	3		3
		2	2
1	3		1X3 = 3
	3	2	3X2 = 6
1	3	2	1X3X2 = 6

21 ← product Total



3 | 2

1 X 3 | 3 X 2

1 X 3 X 2

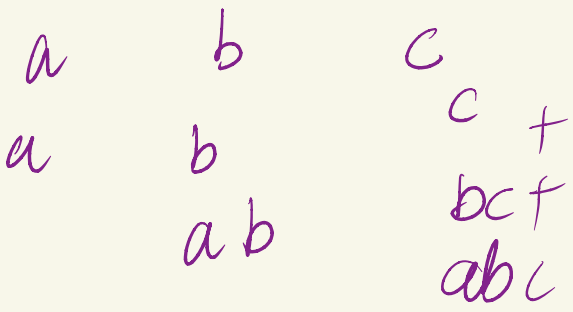
① ② ×2 +2

Here, we are looking at each endpoint and thinking of what is the contribution till that end point.

$$\begin{pmatrix} 3 \\ 1X3 \end{pmatrix} X2 = \begin{pmatrix} +2 \\ 3X2 \\ 1X3X2 \end{pmatrix}$$

Focus: How from ① we can get ② given we know ①.

We can see in this example, that we can just add one 2 and multiply ① by 2 to get ②.

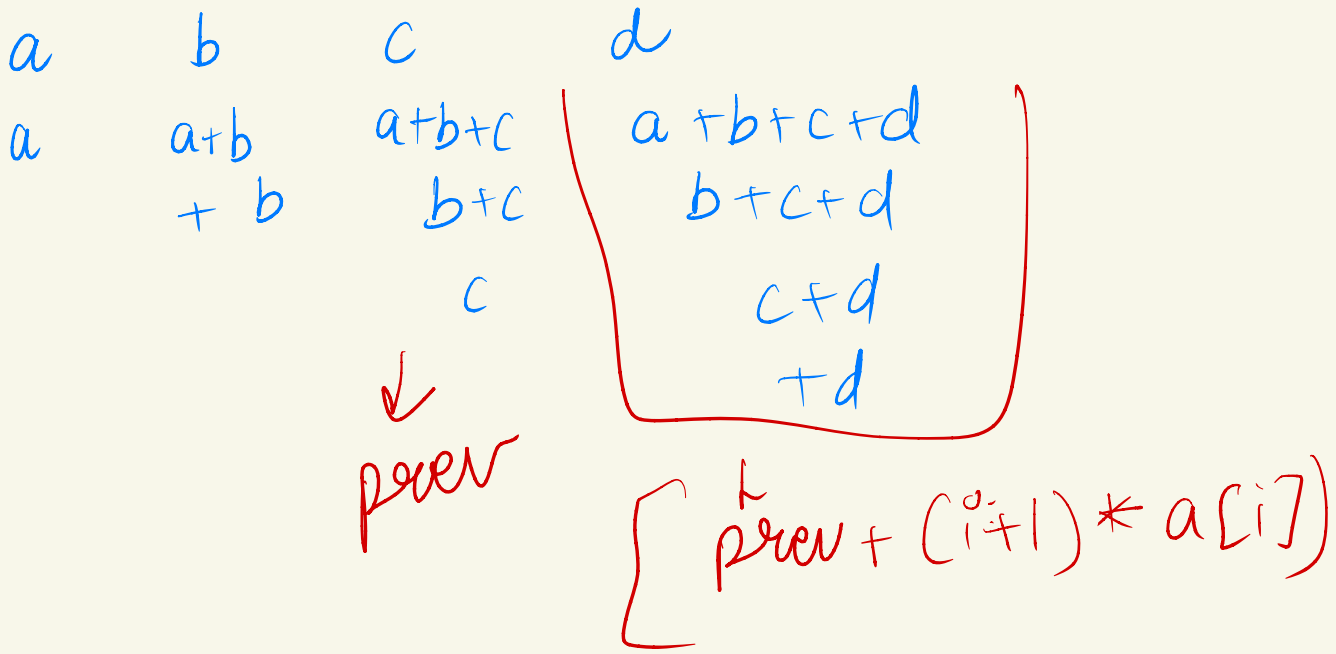


$a + b + ab + c + bc + abc$.

For every end point, we will calculate.

```
ans = 0
lastans = 0
for(0 → N)
{
    lastans = lastans * a[i] + a[i];
    ans += lastans;
}
cout << ans;
```

Let us look at Ques #1 using this technique:



$$prev + (i+1) * a[i]$$

#4 Max Sum Subarray (Kadanes Algo)

```
for (i = 0 to N)
    lastans = max (lastans + a[i], a[i])
    ans = max (ans, lastans).
```

Whenever we encounter subarray problems with count, sum, it can be taken as a hint to use the above discussed contribution Technique.