25/7/25

DSA

Sliding Window & Contribution Technique

✯ Introduction to Subarray:

A subarray is a contiguous part of an array, defined by a starting index and ending index. It represents part of an array.

- single element is also considered to be a subarray.
- complete array is also a subarray
- Subarray is considered from left to right.

	0	1	2	3	4	5	6	7	8
arr:	4	1	2	3	-1	6	9	8	12

2	3	-1	6	✓	8	✓	4	2	✗

8	9	6	✗

1) $\begin{cases} s = 2 \\ s = 2 \end{cases}$ $\begin{matrix} e = 5 \text{ (ending index)} \\ l = 4 \text{ (length)} \end{matrix}$ 2 3 -1 6 / 2 3 -1 6

starting point

2) How many subarrays of the following array?

⇒

4	2	10	3
0	1	2	3

n = 4

all subarrays starting from index 0

s	e	subarray
0	0	4
0	1	4 2
0	2	4 2 10
0	3	4 2 10 3

Total subarrays = 4

all subarrays starting from index 1

s	e	subarray
1	1	2
1	2	2 10
1	3	2 10 3

Total sub. = 3

from index 2		
s	e	sub.
2	2	10
2	3	10 13

Total = 2

from index 3		
s	e	subarray
3	3	3

Total = 1

Total subarrays = 1 + 2 + 3 + 4 = 10

n → length of list
i → starting index

So, for 'i' total possible sublists = (n - i)

Total subarrays = (n - 0) + (n - 1) + (n - 2) + ... 2 + 1
= n * (n + 1) / 2

Print all possible sublists of the list

[1 2 3] (n = 3)

$\Rightarrow \frac{n * (n+1)}{2}$

$\Rightarrow \frac{3 * (3+1)}{2}$

$\Rightarrow \frac{3 * 4}{2}$

$\Rightarrow 6$

Brute force : To print all sublists of array A

```
def print_sublist(A):
    N = len(A)
    for s in range(N):
        for e in range(s, N):
            for k in range(s, e+1):
                print(A[k], end = " ")
            print()
```

TC = $O(N^3)$
SC = $O(1)$

★ Return sum of all subarrays Sum :

Given an array of N elements, you need to return the sum of all the subarray sums.

4	3	7	6
0	1	2	3

n=4

all possible subarrays :

[4] → 4
[4, 3] → 7
[4, 3, 7] → 14
[4, 3, 7, 6] → 20

[3] → 3
[3, 7] → 10
[3, 7, 6] → 16

[7] → 7
[7, 6] → 13

[6] → 6

Total sum ⇒ 4 + 7 + 14 + 20 + 3 + 10 + 16 + 7 + 13 + 6
⇒ (100)

Brute force approach:

```
def sum_sublists(A, N):
    total_sum = 0
    for s in range(N):
        for e in range(s, N):
            sum = 0
            for k in range(s, e+1):
                sum += A[k]
            total_sum += sum
    return total_sum
```

TC = $O(N^3)$
SC = $O(1)$

☆ Optimization using Prefix Sum

1) Create psum list
2) Iterate in range & calculate sum with the help of psum
3) Add to the total sum

```
def sum_psum_lists(li, N):
    # Create psum
    psum = [0] * N
    psum[0] = li[0]
    for i in range(1, N):
        psum[i] = psum[i-1] + li[i]
```

```
# Use query to get sum in range
total_sum = 0
for s in range (0, N):
    for e in range (s, N):
        if s == 0:
            sum = psum[e]
        else:
            sum = psum[e] - psum[s-1]
        total_sum += sum

return total_sum
```

TC : O(N²)
SC : O(N)

Can we do this in constant space?

→ Yes, carryforward technique

```
def sum_sublists (A, N):
    total_sum = 0
    for s in range (0, N):
        cur_sum = 0
        for e in range (s, N):
            cur_sum += A[e]
            total_sum += cur_sum

    return total_sum
```

TC : O(N²)
SC : O(1)

✱ Optimization :

arr: | 3 | 4 | 2 | n=3, count of subarray = $\frac{n(n+1)}{2}$

$= \frac{3(3+1)}{2}$

$= \frac{3 \times 4}{2} = 6$

1 → [3]
2 → [3 4]
3 → [3 4 2]
4 → [4]
5 → [4 2]
6 → [2]

Total sum = $\underbrace{3+3+3}_{9} + \underbrace{4+4+4+4}_{16} + \underbrace{2+2+2}_{6} = 31$

Element	Occurence	Values of freq. x element
3	3	9
4	4	16
2	3	6

arr: | 2 | 8 | -1 | 4 | (0 1 2 3) all subarrays:

Ele	Occur. (freq)	Val. x freq	S	E	sub.array	sum
2 →	4	8	0	0	2	2
8 →	6	48	0	1	2 8	10
-1 →	6	-6	0	2	2 8 -1	9
4 →	4	16	0	3	2 8 -1 4	13
			1	1	8	8
			1	2	8 -1	7
	Total = 66		1	3	8 -1 4	11
			2	2	-1	-1
			2	3	-1 4	3
			3	3	4	4

Intution :

0	1	2	3	4	5
3	-2	4	-1	2	6

0	1	2
3	7	6

Sum of all subarray sum = $3*x + 7*y + 6*z$

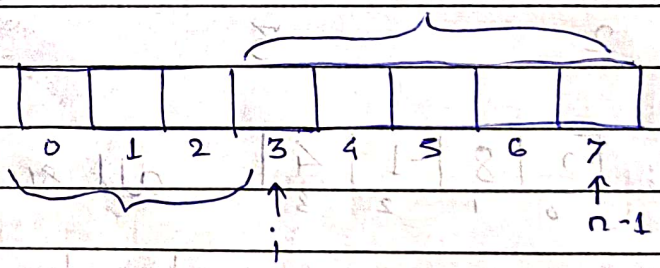
where x, y, z is the freq. of each element in all subarray.

↘ In how many subarrays, does element at index 3 occur?

0-3	1-3	2-3	3-3
0-4	1-4	2-4	3-4
0-5	1-5	2-5	3-5
3	3	3	3

Total sublists = 12

General Solution:



start points : count of elements from 0 to i

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

ending point : count of elements from i to n-1

$[i \quad n-1] \rightarrow n-1-i+1 = n-i$

for index i → start point (s) = i+1

ending point (e) = n-i

count of occur. of i = s * e

freq = $(i+1)*(n-i)$

arr: | 2 | 8 | -1 | 4 |
0 1 2 3

n = 4

i	arr[i]	s ⇒ i+1	e ⇒ n-i	count = s*e	contri count * arr[i]
0	2	1	4	1*4 = 4	4*2 = 8
1	8	2	3	2*3 = 6	6*8 = 48
2	-1	3	2	3*2 = 6	6*-1 = -6
3	4	4	1	4*1 = 4	4*4 = 16
					sum = 66

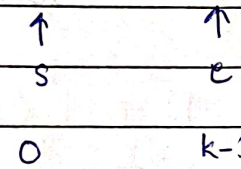
```
def sum_sublists(A, N):
    total = 0
    for i in range(N):
        s = i+1
        e = N-i
        count = s*e
        total += count * A[i]

    return total
```

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

★ Contribution Technique -

Total no. of subarrays of length k :

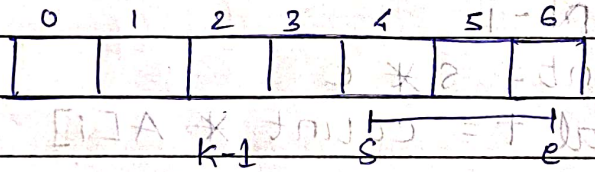
Given that N elements first subarray of len k is from



first sublist of len k

[0 k-1]

Given N elements, no. of subarrays of len = k ?



n = 7 , k = 3

⇒ n - k + 1

⇒ 7 - 3 + 1

⇒ 5

All possible endings of subarray of len k = [k-1 n-1]

count of ending point = no. of subarrays with len k

= [k-1 n-1]

= n - 1 - (k-1) + 1

= n - 1 - k + 1 + 1

= n - k + 1

★ Max subarray sum of len = k :

Given arr [N] elements, return Max subarray sum of len = k

2	3	9	-1	7	1	0

k = 4 max subarray sum having length = 4

13

18

16

7

Brute force approach :

2	3	9	-1	7	1	0

s	e	sum
0	3	13
1	4	18
2	5	16
3	6	17
4	7	⟶ exit

★ Bruteforce approach :

```
def max_subarray_sum (A, N, k):
        maxsum = 0
        for i in range (N-k+1):
            cursum = 0
            for j in range (i, i+k):
                cursum += A[j]
            maxsum = max (maxsum, cursum)
        return maxsum
```

TC : O(N²) SC : O(1)

★ Optimization : (Sliding Window)

0	1	2	3	4	5	6
2	3	9	-1	7	1	0

S	e	sum
0	3	a[0]+a[1]+a[2]+a[3] = 13
1	4	a[0] + a[1]+a[2]+a[3] - a[0]+a[4] = 18
2	5	a[1]+ a[2] + a[3] + a[4] - a[1] + a[5] = 16
3	6	a[2]+ a[3] + a[4] + a[5] - a[2] + a[6] = 7

1> For the first window we iterate & get the sum

2> Then slide window forward :

Going out : a[s-1]

Going in : a[e]

★ Code :

```
def max_subarray_sum (A, N, k):
    sum = 0
    #Calculate the sum of 1st window
    for i in range (0, k):
        sum += A[i]

    ans = sum
    # next window
    s = 1
    e = k
```

```
# slide the window
while (e<N):
    sum = sum - a[s-1] + a[e]
    ans = max(ans, sum)
    s += 1
    e += 1

return ans
```

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