Subarrays → Continuous part of the array.
Single Element ✓
Full Array ✓

Q→ For a given integer array A & integers L & R.
Print all the elements from index L to R.

A = [4 6 10 2 3 -8 6 -5 0]
0 1 2 3 4 5 6 7 8

L = 2 R = 5
o/p → 10 2 3 -8

```
for i → L to R
    print(A[i])
```

TC = O(N)
SC = O(1)

Q→ What is count of subarrays of an array of size N?

A = [8 6 5]
0 1 2

subarrays = 6

[8 5] → not continuous

[8]
[8 6]
[8 6 5]
[6]
[6 5]
[5]

A = [8 6 5 7 2]
0 1 2 3 4

0 ⟼ {0,1,2,3,4} 5
1 ⟼ {1,2,3,4} 4
2 ⟼ {2,3,4} 3
3 ⟼ {3,4} 2
4 ⟼ {4} 1

5+4+3+2+1
= 15

subarrays = N + (N-1) + (N-2) ... + 1
= N*(N+1)/2 → O(N²) ✓

Q→ Print all subarrays of a given array.

A = [8 6 5]
0 1 2

[8]
[8 6]
[8 6 5]
[6]
[6 5]
[5]

```
for i → 0 to (N-1)    // start
    for j → i to (N-1)    // end
        for k → i to j
            print(A[k])
        print("\n")
```

$TC = O(N^3)$

$SC = O(1)$

A = [8 6 5]
0 1 2

i	j	k	o/p
0	0	{0}	8
	1	{0,1}	8 6
	2	{0,1,2}	8 6 5
1	1	{1}	6
	2	{1,2}	6 5
2	2	{2}	5

$\rightarrow O(N^3)$

Total # subarrays $= O(N^2)$

Print 1 subarray $= O(N)$ > Print all subarrays $= O(N^3)$

0 → Print all subarray sum

A = [8 6 5]
0 1 2

i	j
0	0
	1
	2
1	1
	2
2	2

o/p
[8] → 8
[8 6] → 14
[8 6 5] → 19
[6] → 6
[6 5] → 11
[5] → 5

```
for i → 0 to (N-1)   // start
    for j → i to (N-1)   // end
        sum = 0
        for k → i to j
            sum += A[k]
        print (sum)
```

$TC = O(N^3)$

$SC = O(1)$

Range sum → P[j] - P[i-1]

Steps 1) calculate prefix sum → $TC = O(N)$ $SC = O(N)$

2)

```
for i → 0 to (N-1)   // start
    for j → i to (N-1)   // end
        if (i == 0)
            print (P[j])
        else
            print (P[j] - P[i-1])
```

$TC = O(N^2)$ $SC = O(1)$

Total

$TC = O(N + N^2) = O(N^2)$ ✓

$SC = O(N+1) = O(N)$

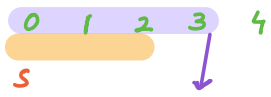
N elements → # subarrays $= O(N^2)$ → # elements to print $= O(N^2)$ → Best $TC = O(N^2)$

How to optimize space → 1) Modify input array to prefix sum. ✓

2) carry forward

```
for i → 0 to (N-1)   // start
    sum = 0
    for j → i to (N-1)   // end
        sum += A[j]
        print (sum)
```

Index



(A[0]+A[1]+A[2])

TC = O(N²)

SC = O(1)

A = [8 6 5]
0 1 2

o/p

[8] → 8
[8 6] → 14
[8 6 5] → 19
[6] → 6
[6 5] → 11
[5] → 5

i	j	sum
0		0
	0	8 ✓
	1	14 ✓
	2	19 ✓
1		0
	1	6 ✓
	2	11 ✓
2		0
	2	5 ✓

10:25 PM

Q → Given an integer array A & an integer K, check in how many subarrays index K is present.

A = [8 8 8 8 8 8]
0 1 2 3 4 5

K = 2

i	j
0	{2, 3, 4, 5} → 4
1	{2, 3, 4, 5} → 4
2	{2, 3, 4, 5} → 4
3	—
4	—
5	—

12 (Ans)

K = 4

i	j
0	{4, 5} → 2
1	{4, 5} → 2
2	{4, 5} → 2
3	{4, 5} → 2
4	{4, 5} → 2
5	—

10 (Ans)

K = 0

i	j
0	{0, 1, 2, 3, 4, 5} → 6 (Ans)

start (i) → {0, 1, ... K}

end (j) → {K, K+1, K+2, ... (N-1)}

$[L \quad R] \rightarrow R-L+1$

start $\rightarrow [0 \quad K] \rightarrow K-0+1 = K+1$

end $\rightarrow [K \quad (N-1)] \rightarrow (N-1)-K+1 = N-K$

∵ for all start we can select any end $\rightarrow$ Ans $= (K+1)*(N-K)$ ✓

Q → Find the sum of all subarray sums of the array.

A = [8 6 5]
0 1 2

[8] → 8
[8 6] → 14
[8 6 5] → 19
[6] → 6
[6 5] → 11
[5] → 5

63 (Ans)

```
ans = 0
for i → 0 to (N-1)      // start
    sum = 0
    for j → i to (N-1)  // end
        sum += A[j]
        ans += sum
print(ans)
```

TC = O(N²)
SC = O(1)

```
ans = 0
for i → 0 to (N-1)      // start
    sum = 0
    for j → i to (N-1)  // end
        sum += A[j]
        ans += A[j]
print(ans)
```

X incorrect

Contribution Technique

If any element of array/matrix etc is contributing multiple times in the ans, then ↓

ans = Σ contribution of all elements

A = [8 6 5]
0 1 2

8 → 8
8+6 → 14
8+6+5 → 19
6 → 6
6+5 → 11
5 → 5

63 (Ans)

→ 8*3 + 6*4 + 5*3

A[i]*(# subarray in which index i is present)

8*(# subarrays in which 8 is present)

$Ans = \sum_{i=0}^{N-1} A[i] * (\#\text{ subarray in which index } i \text{ is present})$

```
ans = 0
for i → 0 to (N-1)
    ans += A[i]*(i+1)*(N-i)
print(ans)
```

TC = O(N) SC = O(1)

$Ans = \sum_{i=0}^{N-1} A[i] * (i+1)*(N-i)$

Q→ Find max subarray sum with length = K.

```
   0  1  2  3  4  5
A = [8  6  5  7  10  2]   K = 3
```

19 18 22 19

(Ans)

Bruteforce → ∀ subarray of length K, travel to calculate sum & keep track of max.

```
   0  1  2  3  4  5
A = [8  6  5  7  10  2]   K = 3
```

start → [0 3] ⟶ 3 − 0 + 1 = 4

end → [2 5] ⟶ 5 − 2 + 1 = 4

start → 0 ⇒ min end index

end
- max = N-1
- min = K-1

A[0], A[1], A[2] ... A[K-1] (K-1)

→ first K element

A[N-K], A[N-K+1], A[N-K+2] ... A[N-1]

→ last K elements

start
- min = 0
- max = N-K

subarrays of length = K

(N-1) − (K-1) + 1 = N − K + 1

(N-K) − 0 + 1 = N-K+1

```
ans = INT_MIN // -2^31
for i → 0 to (N-K)  → (N-K+1)
    sum = 0
    for j → i to (i+K-1)  → K
        sum += A[j]
    ans = max(ans, sum)
print(ans)
```

TC = O((N-K+1) * K) = O(N*K) ✓

K = N (N-N+1) * N = N

```
   0  1  2  3  4  5
A = [-8 -6 -5 -7 -10 -2]   K = 3
```

-19 -18 -22 -19

(Ans)

SC = O(1)

Sliding Window

(subarray of fixed length)

Prefix Sum → H.W

```
      i          i+(3-1)
   0  1  2  3  4  5
A = [8  6  5  7  10  2]   K = 3
```

19

19 + 7 − 8 = 18

18 + 10 − 6 = 22

start = i

end = i+K-1

old sum + A[i+K-1] − A[i-1]

```
sum = 0
for i → 0 to (K-1) → K
    sum += A[i]
ans = sum
for i → 1 to (N-K) → N-K
    sum = sum + A[i+K-1] - A[i-1]
    ans = max(ans, sum)
print(ans)
```

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

SC = O(1)
