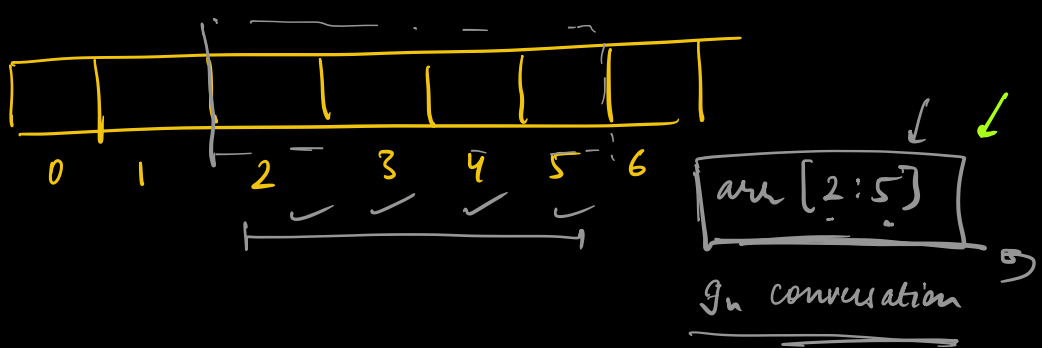
Agenda:

1. Introduction and recap
2. Generate and print all subarrays
3. Find sum of each subarray
4. Find sum of all subarrays
5. Maximum subarray sum

Subarray

Arrays

A continuous non-empty part of an array is called a subarray.



In conversation

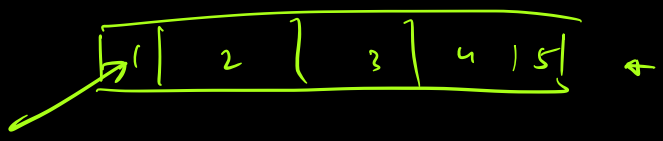
Slices

`l[start:end:step]`

`arr[2:6]`

end index is not taken as a part of slice

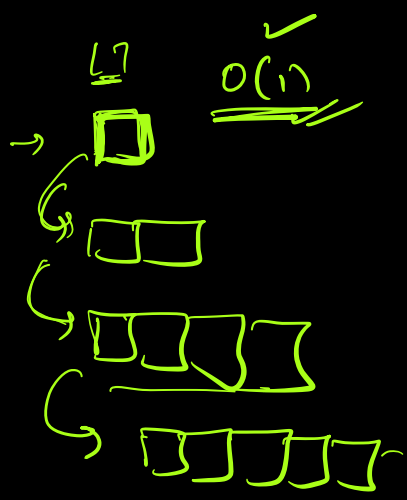
arr[N]



Dynamic arrays ← A beautiful extension of arrays.

Lists

l = [ ] → l.append( )
l.pop( )



5 4 3 1 2

[4 3 1]

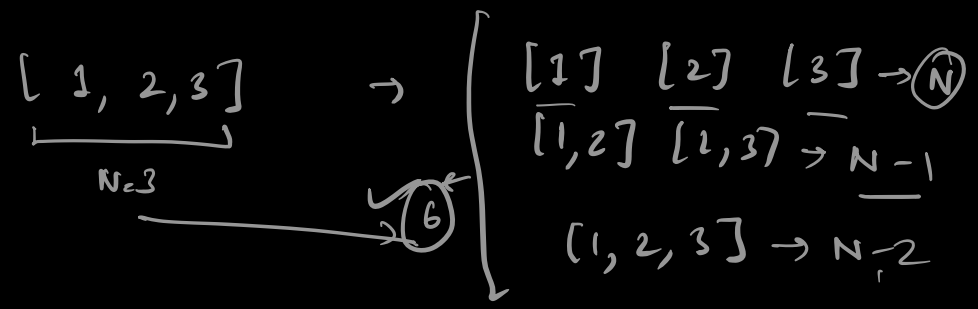
[5 3 1] x

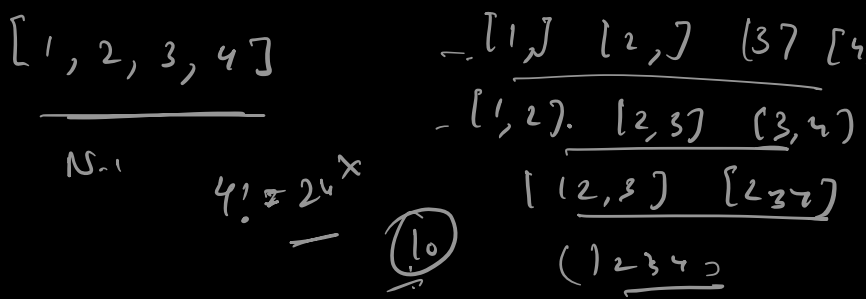
[ ] x

Total no of subarrays

- For an array with n elements.

Q: How many subarrays are there?





No. of subarrays $= N + (N-1) + (N-2) + \dots + 1$

$= \frac{N * (N+1)}{2}$

Every subarray can be described by a start index and an end index.

Start	End	
0	(0 → n-1)	→ N possibilities
1	(1 → n-1)	→ N-1 poss...
2	(2 → n-1)	→ N-2
i		n-i
N-1		1

$A[i:j]$ — both indexes are taken

sum of N natural numbers

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

Q Generate all subarrays

arr: [1, 2, 3]

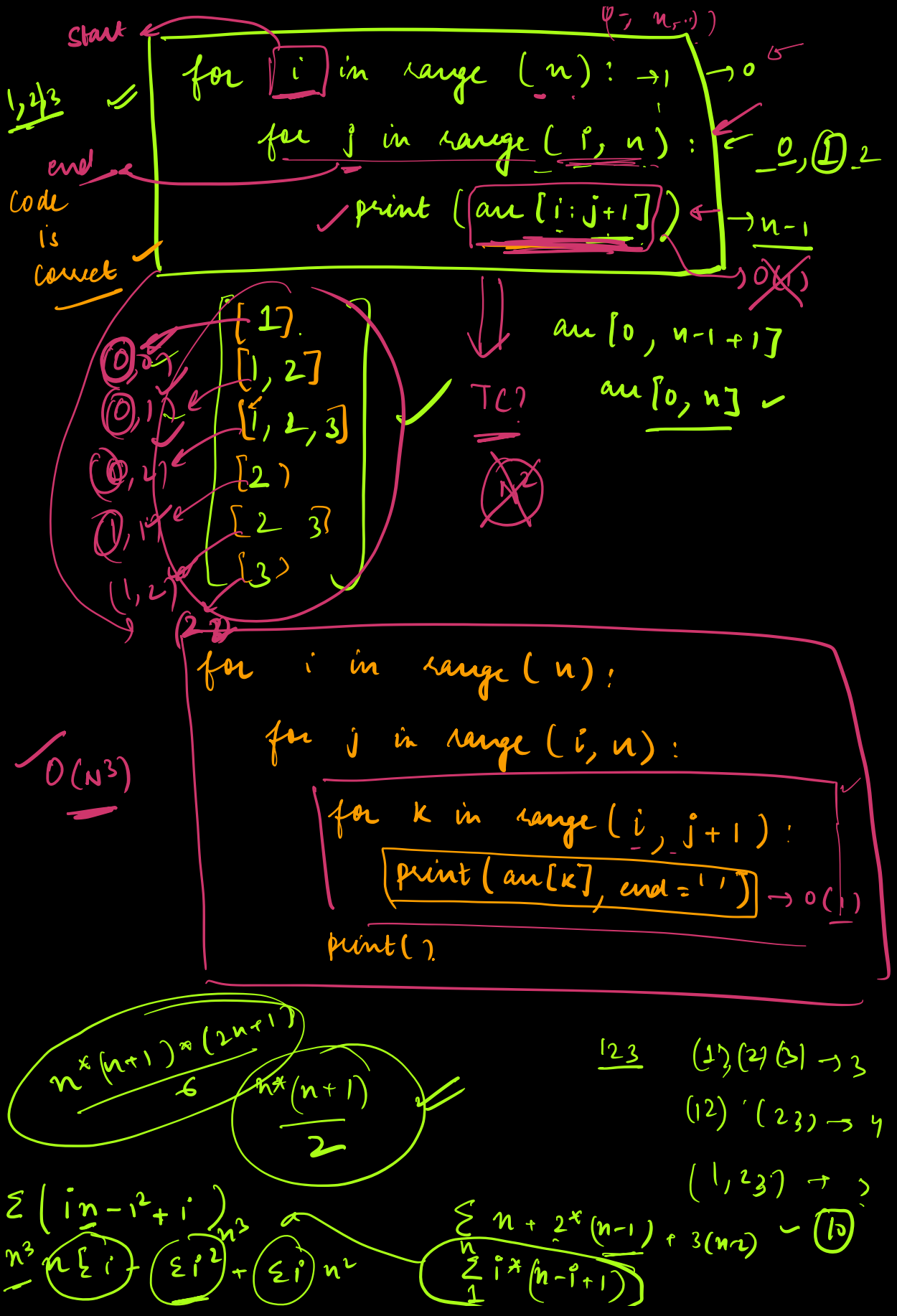
[1 ; 1, 2 ; 1, 2, 3 ; 2 ; 2, 3 ; 3]

Permutations

1, 2, 3
3, 2 1
3, 1, 2
1, 3 2
2, 3, 1
2, 1, 3

→ 6 valid permutations

N! = 1 x 2 x 3 ... N


start
for i in range(n):
for j in range(i, n):
print(arr[i:j+1])
end
code is correct
1,2,3
0, 1, 2
n-1
O(1)
[1]
[1, 2]
[1, 2, 3]
[2]
[2 3]
[3]
arr[0, n-1+1]
arr[0, n]
TC?
O(n^2)
for i in range(n):
for j in range(i, n):
for k in range(i, j+1):
print(arr[k], end=' ')
O(1)
print()
O(n^3)
n*(n+1)*(2n+1)/6
n*(n+1)/2
123
(1),(2),(3) → 3
(12),(23) → 4
(1,23) → 2
10
Σ(i n - i^2 + i)
n^3
n Σ i
Σ i^2
Σ i n^2
Σ n + 2*(n-1) + 3(n-2)
Σ i*(n-i+1)

Q Find sum of each subarray.

[4, 5, 2]

Hint

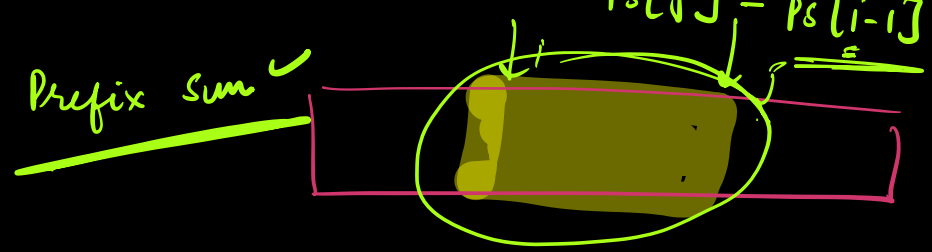
[4] → 4
[4,5] → 9
[4,5,2] → 11
5 → 5
5,2 → 7
2 → 2

$O(n^2)$

① BF (TC) $O(N^3)$ (SC) $O(1)$

```
for i in range(n):
    for j in range(i, n):
        sum=0
        for k in range(i, j+1):
            sum += A[k]
        print(sum)
```

Prefix sum



Approach - 2

O(1)

Prefix Sum Approach

TC → N^2

SC → O(N)

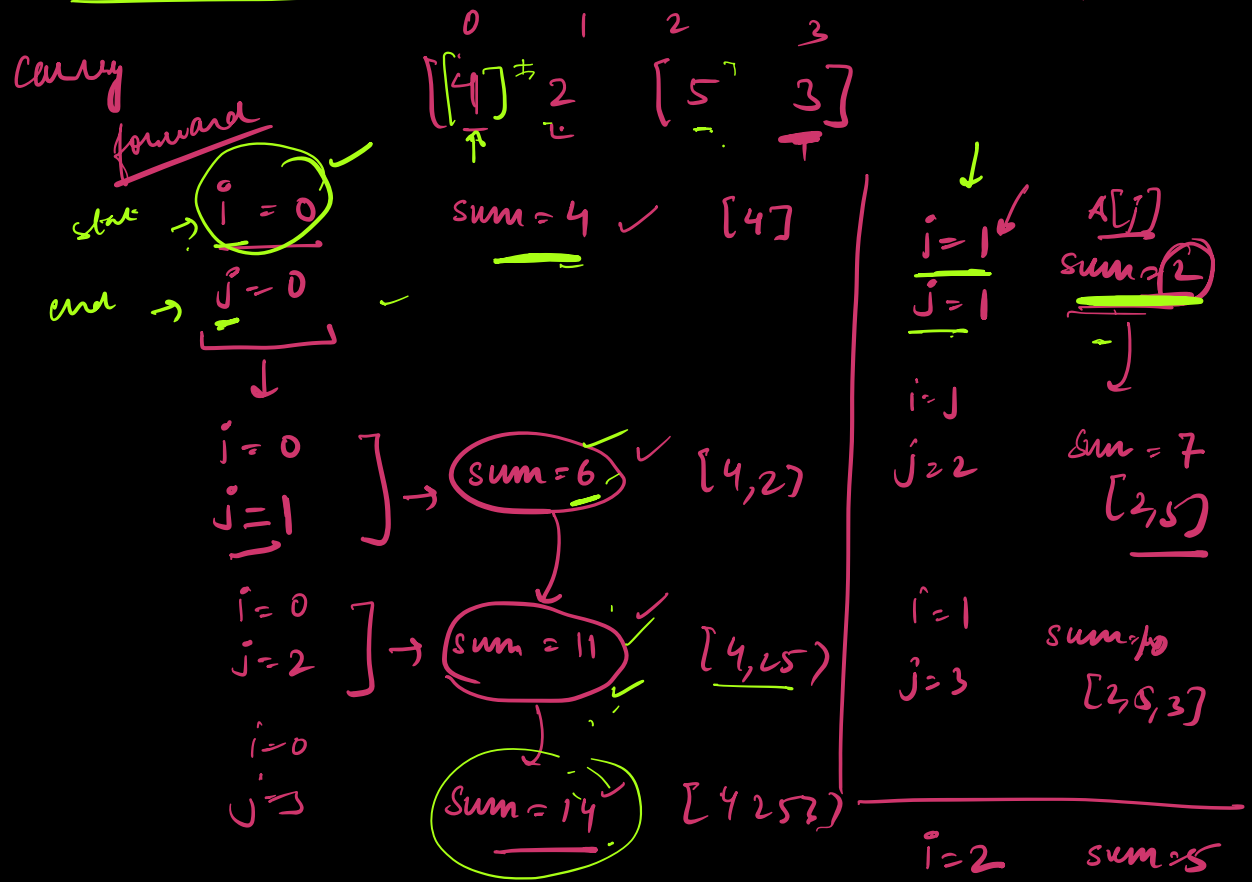
2

①
```
PS = [0] * N
PS[0] = A[0]
for i in range(1, N):
    PS[i] = PS[i-1] + A[i]
```

②
```
for i in range(N):
    for j in range(i, N):
        if i == 0:
            print(PS[j])
        else
            print(PS[j] - PS[i-1])
```

Approach - 3

i = 0 → n-1
j = i → n-1




```
for i in range(n):
    sum = 0
    for j in range(i, N):
        sum += arr[j]
    print(sum)
```

TC $O(N^2)$

SC → $O(1)$

Q3 Print sum of (sum of all subarrays)

5 mins

```
res = 0
for i in range(0, n)
    sum = 0
    for j in range(i, n)
        sum += a[j]
        res += sum
return res
```

$O(n^2)$

$a_0 \quad a_1 \quad a_2 \quad a_3$

TC = $O(N^2)$

SC $O(1)$

$[a_0]$ $[a_0\ a_1]$ $[a_0\ a_1\ a_2]$ $[a_0\ a_1\ a_2\ a_3]$

$[a_1]$ $[a_1\ a_2]$ $[a_1\ a_2\ a_3]$

$[a_2]$ $[a_2\ a_3]$

$[a_3]$

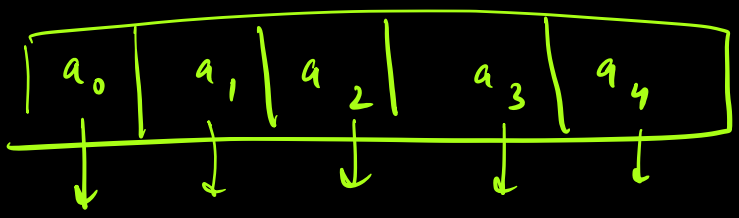
$S = 4 * a_0 + 6 * a_1 + 6 * a_2 + 4 * a_3$

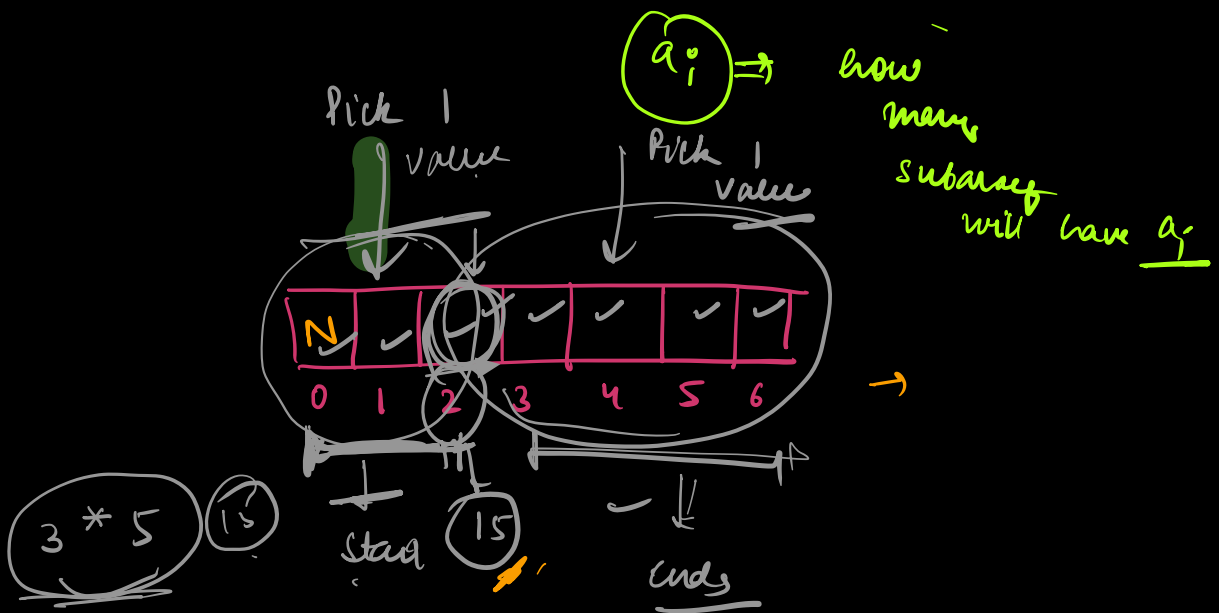
$O(N)$

This coefficient represents no. of times this element occured in all subarrays

OR

No of subarrays which contain this element.





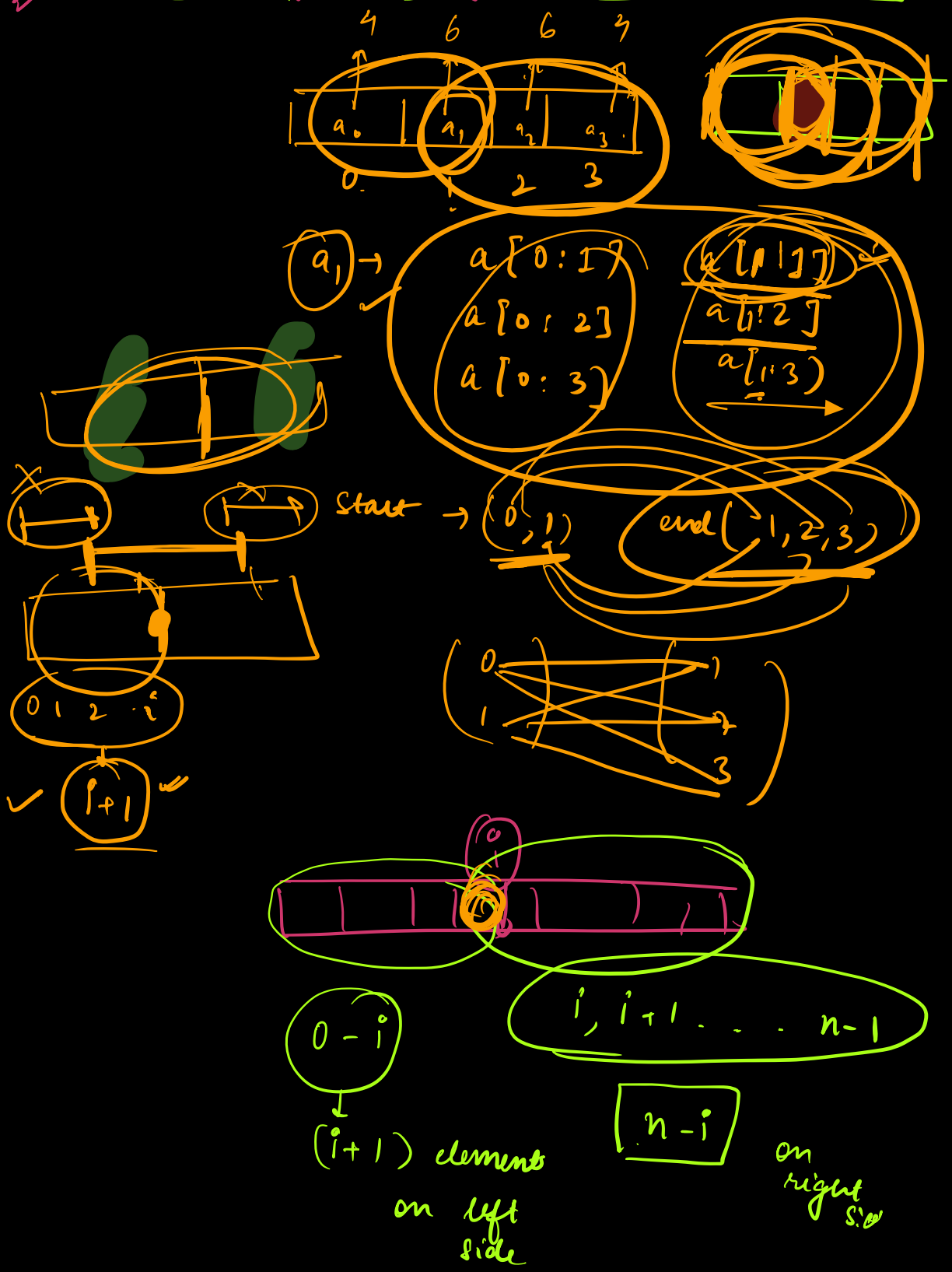
$a[1:5)$ ✓ $a[2)^2$

$a[2:6]$ — $a[2]$

$a[0:2]$ ✓

$a[1:4]$ —

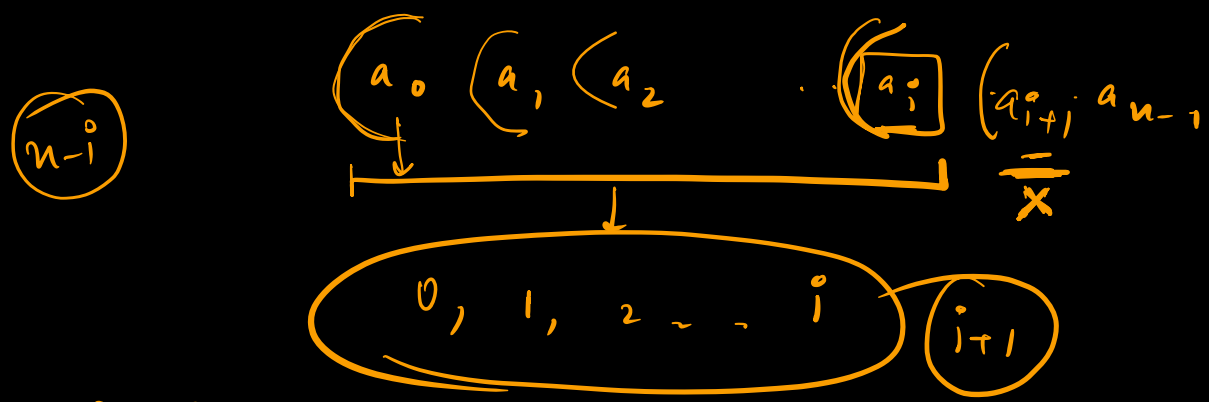
Q Find no. of subarrays which contain a_i



for any i index

no. of subarrays that contain index i will be

$(i+1) * (n-i)$ ✓



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

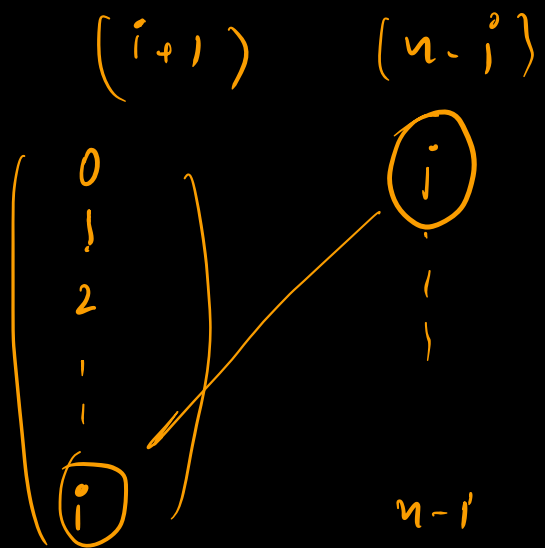
$a_i \quad a_{i+1} \quad \ldots \quad a_{n-1}$

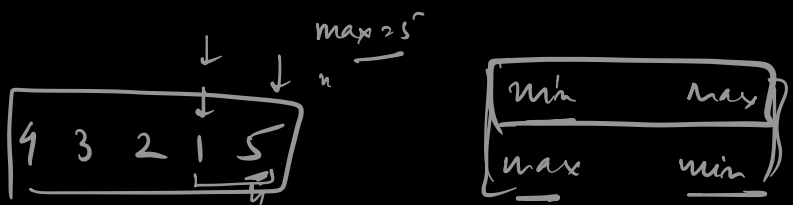
5 6 7 8 9

5 - 10 → 6 ✓

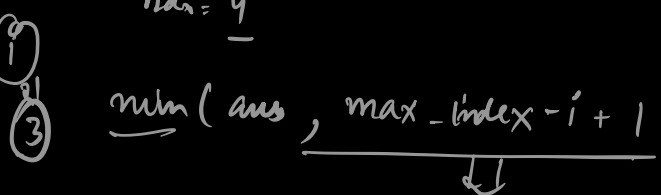
$n-i$ ✓ $= n-i-1+1$ → $10-5+1$

```
ans = 0
for i in range(n):
    ans += (i+1) * (n-i) * A[i]
return ans
```

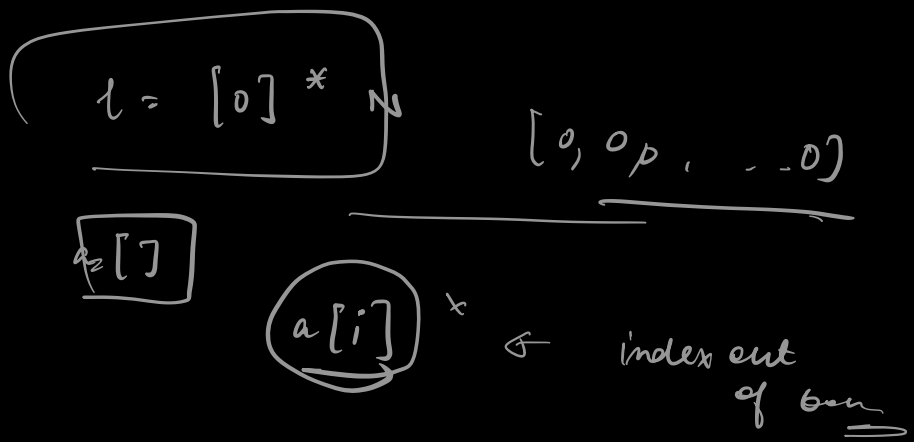




max = 4



ans = 2


t = [0] * N
[0, 0, ... 0]
a[]
a[i]
index out of bounds