Arrays : Subarrays

Apr 4, 2022

AGENDA:

- Subarray
  - total no. of subarrays
  - generate / print all subarrays
  - sum of each subarray
  - find sum of all subarrays

- Maximum subarray sum ** (Interview).
  - Optimised approach

- Finding no. of subarrays of length K.

{ Prefix sum.. Carry forward. }

Subarrays.

Array. → a contigous block of memory. (elements)

Subarrays → a slice of an array.

arr := [5, 3, 2, 6, 7]
(0, 1, 2, 3, 4)

arr[2:4] → [2, 6].

arr[start : end] → include all values from start to end (but excluding end)

Slice operator → used to get any subarray out of an array.

* Can subarray have a single element?
  → Yes!
  arr[start : start+1]

Is this a subarray? → arr[i] // arr[start].
→ Element Not a subarray

* Can subarray contain the whole array?

arr = [. . . .].

arr [:] → subarray.

* Can subarray be empty ?

// Subarrays cannot be empty.'

* arr = [3, 4, 5, 6, 7, 8]

[6, 5, 7] is a subarray ? ✗

subset.

- Element should come in the same order!

* sub-arr = [3, 5, 6, 7]. → Subsequence.

✗ non-contigous.

Order should be maintained, but it might not be contigous.

Subset < Subsequence. < Subarray.

↓ ✗ | ↓ order. | ↓ contigous + order.

Subarrays

[3, 4, 5, 6].

Start from 3.

→ [3], [3,4], [3,4,5], [3,4,5,6]

→ [4], [4,5], [4,5,6]

→ [5], [5,6]

→ [6].

4! N!

4! = 24.

Approach?

Count no. of subarrays of arr size N.

arr = [. . . . .].

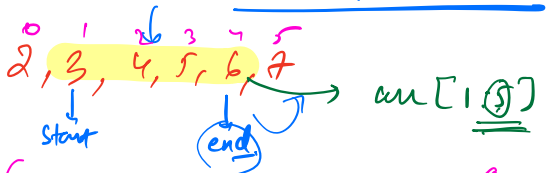
arr [start : end]

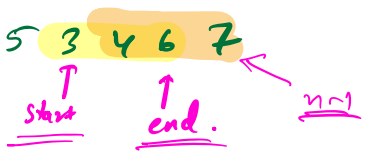
Count how many combinations of (start, end) is present?

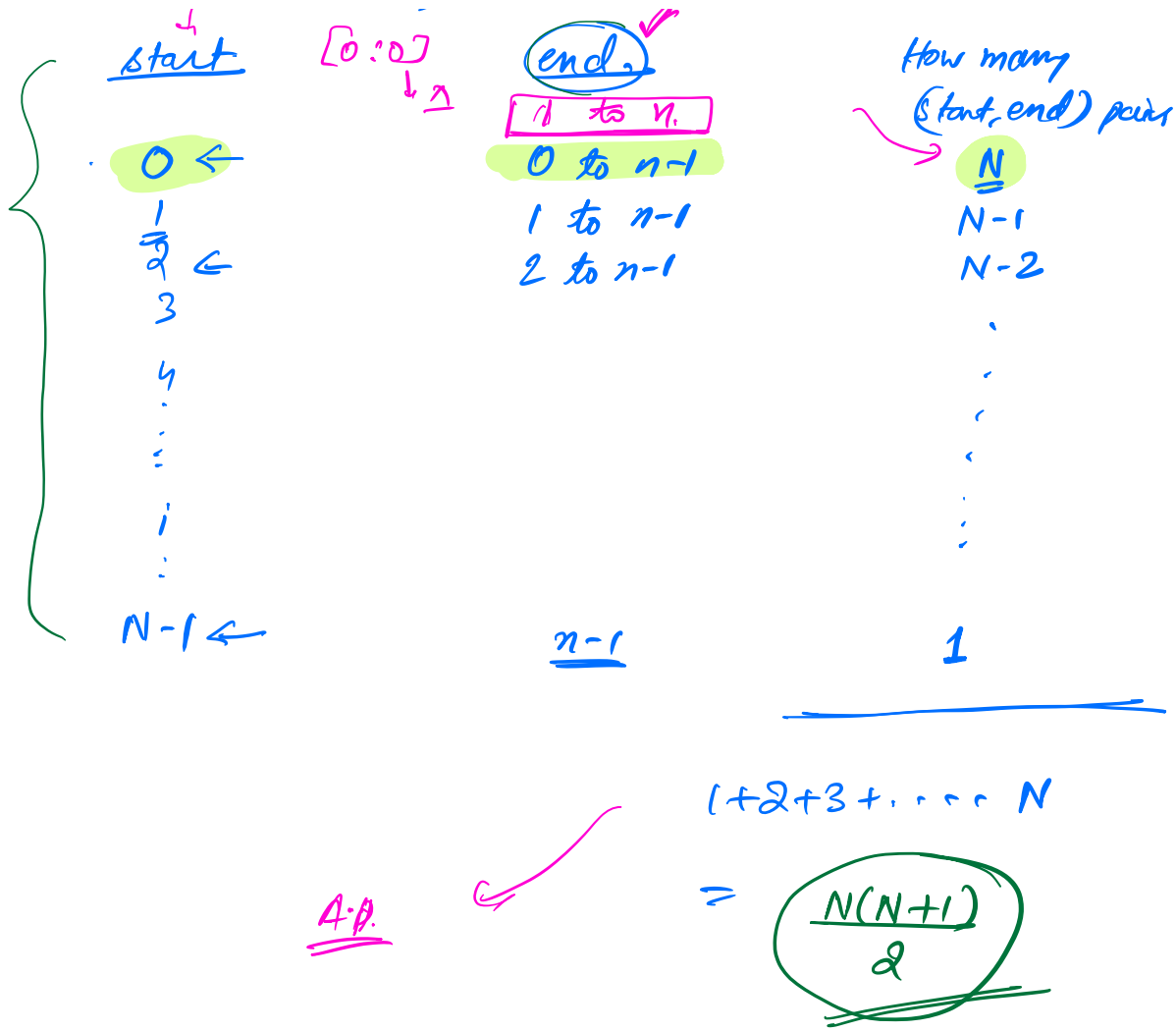
start → [0, N-1]

end →

end ≥ start.







* How many subarray for n size array.

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

Generate / print all subarrays.

arr = [3, 4, 5, 6, 7]

3 → 3 4, 5 → 15

$\frac{5(5+1)}{2}$

$O(N^3)$

* Print all (start, end) pairs.

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

$O(N^2)$

print(5)
print(6)

Brute force / Most optimum.

O(N) → for start in range(0, n):
  O(N) → for end in range(start, n):
    # subarray defined by [start : end+1]
    O(N) → for i in range(start, end+1):
      print(arr[i], end = " ")

print(arr[start : end+1])

print()

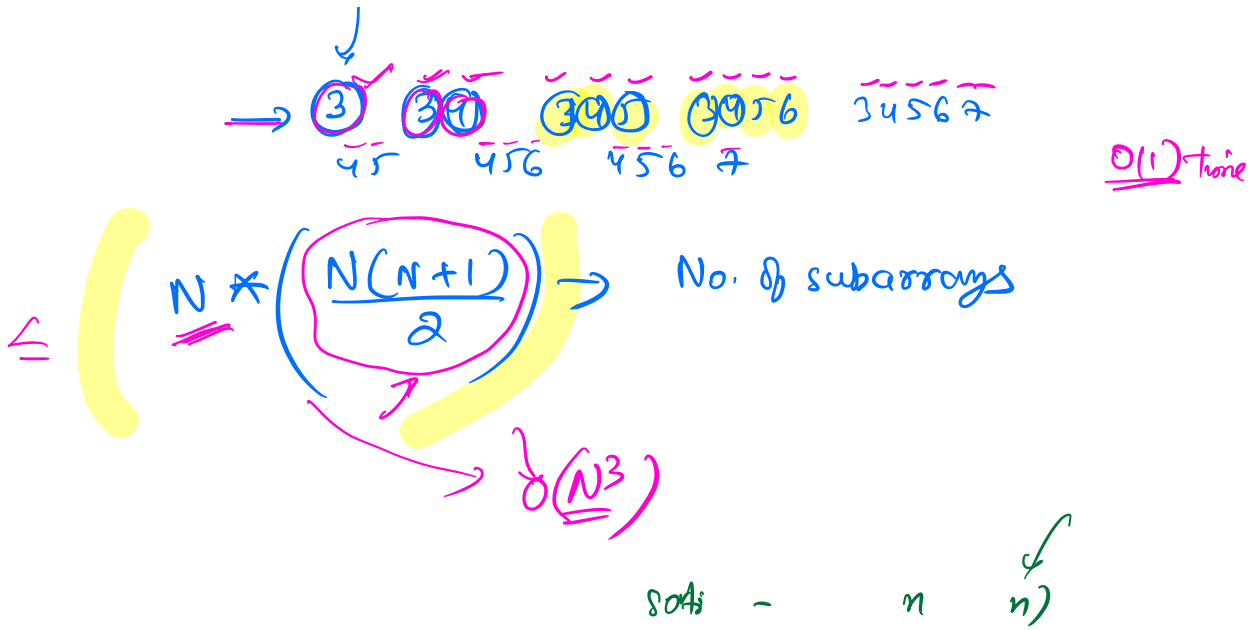
O(N)

$O(N^3)$

same thing.

** arr = [3, 4, 5, 6, 7]

print(string) → O(1)



* Find the sum of each subarray.

$\frac{N(N+1)}{2}$ subarrays → Print the sum of each of those.

Brute force.

→ $O(N^3)$

```
for start in range(0,n):
    :
    for end in range(start, n)
        # subarray defined by [start:end+1]
        sum=0
        for i in range(start, end+1):
            sum += arr[i]
        print(sum)
```

→ $O(N^3)$

Can we optimise?

[start, end].

Prefix sum → Range sum Queries.

* Build up a prefix sum array.

* for start in range(0, n):
  for end in range(start, n):
    # subarray → [start, end+1]
    print(preSum[end] - preSum[start-1])

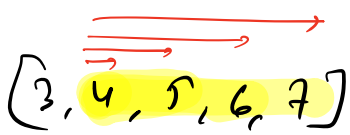
start = 0.
Handle this separately.

TC: O(N²).
SC: O(N) → Prefix sum array

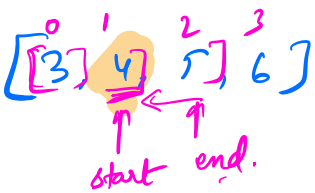
* Can we further optimise?

→ Space can be optimised.

** Carry-forward.



```
for start in range(0,n):
    sum = 0
    for end in range(start, n):
        # subarray defined by [start:end+1]
        -> sum += arr[end]
        -> print(sum)
```



[3]
[3,4]
[3,4,5]

[4]
[4,5]

[5]
[5,6]

[6]

sum = 0.
sum = 3 → 4.
sum = 7
sum = 12.
sum = 18.

sum = 4
sum = 9.
sum = 15

sum = 5
sum = 11

sum = 6

TC: O(N²)

SC: O(1)

Break till 10:20.

Q. Find the total sum of all subarrays.

```
total_sum = 0
for start in range(0, n):
    sum = 0
    for end in range(start, n):
        # subarray defined by [start : end+1]
        sum += arr[end]
        total_sum += sum
print(total_sum)
```

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

Single variable:

```
sum = 0
for start in range(0, n):
    for en in range(start, n):
        # subarray defined by [start : end+1]
        sum += arr[end]
```

[3 4 5 6]
(0 1 2 3)

start
end

sum = 0
3
3 + 4
7 + 5
12 + 6
(18) + 4

22 +5
27 +6
33 +5
38 +6
44 +6
(50)

* Maximum Subarray Sum (Interview problem)

Print the sum of the subarray which has maximum sum.

[3, 4, 5, 6, 7] →

[3, -1, -2, 0, 3] → (3) ✓

Brute force.

3 loops. max_ans

start —— 0 to n.

end. —— start to n.

```
sum=0
i from start to end.
    sum+=arr[i]
max_ans=max(max_ans, sum)
```

↓ O(N^3)

```
max_ans = -float("inf")
for start in range(0, n):
    sum = 0
    for end in range(start, n):
        # subarray defined by [start:end+1]
        sum += arr[end]
        max_ans = max(max_ans, sum)
print(max_ans)
```

$O(N^2)$

0: identity element

For +, identity = 0 — $5+0=5$, $6+0=6$, $-100+0=-100$

For *, identity = 1

For max, identity = -float("inf") — $\max(N, ?) = N$

$\max(N, N+5) \neq N$

For min, identity = float("inf") — $\max(N, 0) \neq N$

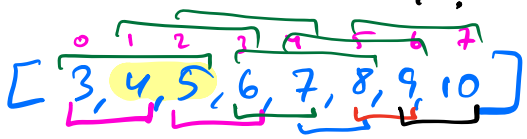
$\max(N, -\infty) = N$

** This can further be optimised.

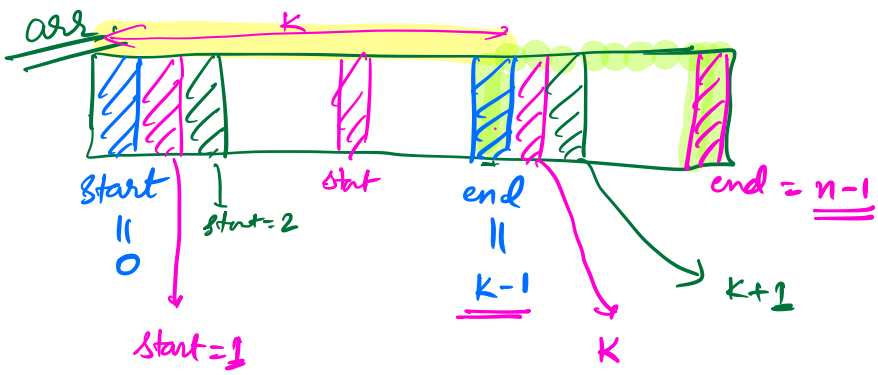
↳ TC: O(N)

Kadane's algorithm.

** How many subarrays of length k are present in n size subarray?



K=1	ans = 8	N
K=2	ans = 7	N-1
K=3	ans = 6	N-2
K = k	ans =	
K=N		1



end range

[k-1, k, k+1, k+2 ..... n-1]

↳ (n-1)-(k-1)+1

= n-k+1

$= n-k+1$

Total no. of subarrays $= n-k+1$