Pragy
pragy@scaler.com
7351769231

Agenda

- Subarray
  - total no of subarrays
  - generate / print all subarrays
  - find sum of each subarray
  - find sum of all subarrays — (1)
- Maximum Subarray sum
- optimized approach for (1)

Arrays

↳ contiguous chunks of homogenous (same datatype) memory

	0	1	2	3	4	5	6	7
Ar = [	3	-6	0	1	7	8	9	2]
	↑							↑ end index

|Ar| = len(Ar) = 8

ar = [1, -3, "hello", object(), 2+7j, True]

lists ≠ arrays

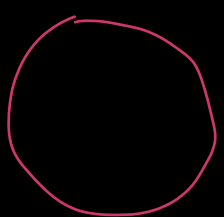
import array

dynamic (resize)
stack / --- many other operations.

Subarray is a "slice" of an array.

contiguous → one piece

→ X not a subarray ∵ not one piece.

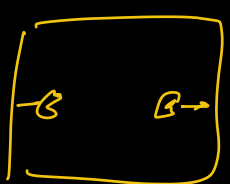


→ entire cake is also a slice of the cake. ✓

ar = [3 (0) | -6 (1) | 0 (2) | 1 (3)]

|ar| = 4

[2,0] not contiguous



[1,3] is not a subarray. L → R

All the subarrays of ar

a subarray must have at least 1 element

↳ empty array is NOT a subarray.

[3] (0,0) [3 -6] (0,1) [3 -6 0] (0,2) [3 -6 0 1] (0,3) ← entire array

[-6] (1,1) [-6 0] (1,2) [-6 0 1] (1,3)

[0] (2,2) [0 1] (2,3)

[1] (3,3)

[0] [0 1]

[1]

Subarray are contiguous subsequence can be broken.

Total no of subarrays

$|ar| = 4$ # of subarrays $= 10$

$|ar| = 3$ " " " $= 6$.

end $\geq$ start

a subarray $\rightarrow$ (start index, end index)

n

$\{0 \ldots n-1\}$

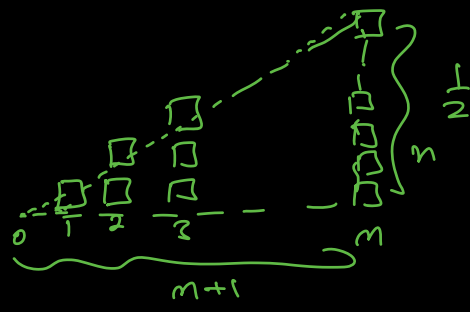
Start	end	# of end values.
0	$0 \ldots n-1$	$\Rightarrow n$
1	$1 \ldots n-1$	$\Rightarrow n-1$
2	$2 \ldots n-1$	$\Rightarrow n-2$
⋮		
i	$i \ldots n-1$	$\Rightarrow n-i$
⋮		
$n-1$	$n-1 \ldots n-1$	$\Rightarrow 1$

$n + (n-1) + \ldots 2 + 1$

$= 1 + 2 + 3 + \ldots n$

= Sum of first N natural nos.

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



$\frac{1}{2} \cdot base \cdot height = \frac{n \cdot (n+1)}{2}$

$$\frac{n(n+1)}{2} = {}^{n+1}C_2 = O(n^2)$$

${}^{n}C_{r}$ ⇒ n choose r $= \frac{n!}{(n-r)!\, r!}$

(combination / choose) ⇒ # of ways of choosing r items out of n items

---

Generate / Print all subarrays

Subarray → start .. end

(0 .. n-1) (start ... n-1)

range(l, r) — l included, r excluded

Or → given to → [0 .. n-1]

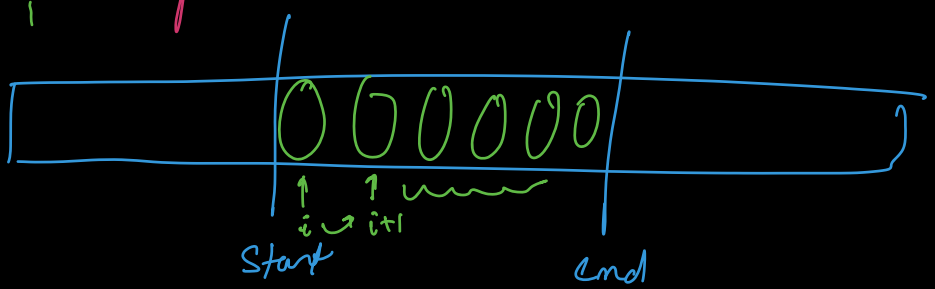
```
for start in range(n):          → O(n)
    for end in range(start, n)  → O(n)
        # subarray is defined by start, end
```

print(ar[start : end+1]) — ignore

for loop hidden slice notation. inside this has O(n) complexity → O(n)

```
for i in range (start, end+1):
    print(ar[i], end = ' ')
print()
```

↳ for printing.




```
print (1)
print (2)
```
⇒ 1 2

```
print (1, end = '')
print (2)
```
⇒ 12

$O(n \cdot n \cdot n) = O(n^3)$ TC

$O(1)$ SC

} print it ourselves

print(ar[start : end+1]) → TC → $O(n^3)$

SC → $O(n)$

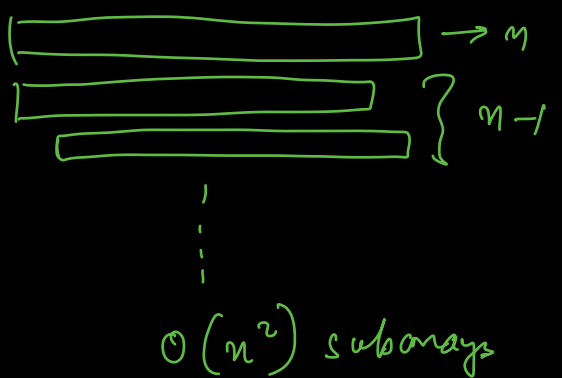
↳ class list:

--getitem-- (self, start, end):

for - - -

Can we do better than n^3 to "print" all subarrays?

↳ No!



$O(n^2)$ subarrays

↳ each of size $O(n)$

↳ print $O(n^3)$ items

Find the sum of each subarray & print

```
for start in range(n):                O(n)
    for end in range(start, n):       O(n)
        sum = 0
        for i in range(start, end+1): O(n)
            sum += ar[i]
        print(f"The sum of subarray [{start}, {end}] =",
              sum)
```

TC = $O(n^3)$

SC = $O(1)$

arr = [2 1 0 -3 6]

sum

The sum of subarray [0, 0] = 2

" " " [0, 1] = 3

[0, 2] = 3

[0, 3] = 0

[0, 4] = 6

[1, 1] = 1

[1, 2] = 1

[1, 3] = -2

⋮

[1, 4]

[2, 2]

⋮

[2, 4]

[3, 3]

[3, 4]

[4, 4] =

① Prefix sum array.

prefix-sum-arr

sum [start, end+1] ⟶ O(1) operation using the prefix sum array.

```
prefix_sum = [ar[0]]
for i in range (1, n):                    O(n)
    prefix_sum.append(prefix_sum[i-1] + ar[i])

for start in range (n):                   O(n)
    for end in range (start, n):          O(n)
        if start == 0:
            sum = prefix_sum[end]
        else
            sum = prefix_sum[end] - prefixsu[start-1]
        print(f"---- -- =", sum)          O(1) space
```

TC $O(n+n^2) = O(n^2)$

SC $O(n)$ → because storing the prefix sum array.

Count the # of subarrays → nested for loops → $O(n^2)$

→ formula $= \frac{n(n+1)}{2}$ → $O(1)$

tt

To print the sum of each subarray will take at-least $O(n^2)$ time → ∵ we have $O(n^2)$ subarrays & we have to print the sum of each.

arr = [1 3 0 -6 8]

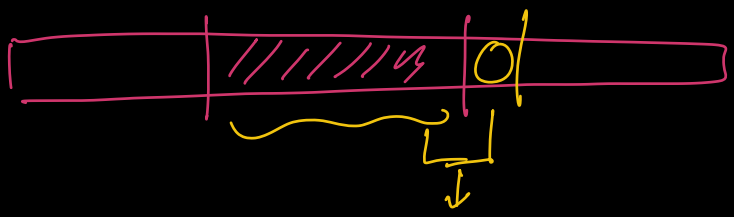
start, end, end

sum = 0

sum += 1
sum += 3
sum += 0
sum += -6
= -3 → print

sum = 0

sum += 3
sum += 0
sum += -6
sum += 8
= 5 → print



Calculate the prefix-sum "on-the-fly"

```
for start in range(n):                    O(n)
    sum = 0
    for end in range(start, n):           O(n)
        sum += arr[end]
        print(f"sum of arr[{start}, {end}] = ", sum)
```

	0	1	2	3	4
arr	-3	0	6	8	1

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

Dry-Run → pretend to be the computer.

start	sum	end	printed
0	0 / -3 / -3 / 3	0 / 1 / 2	arr[0:0] = -3 arr[0:1] = -3 arr[0:2] = 3
⋮			

1

for stat in range(n): O(n) prints all subarrays.

s = ""

for end in range(stat, n): O(n)

s += str(arr[end])] O(n)

hidden loop O(n) ← s += ...

print(s) → O(n)

TC	SC
$O(n^3)$	$O(n)$

---

s = "this is a string"

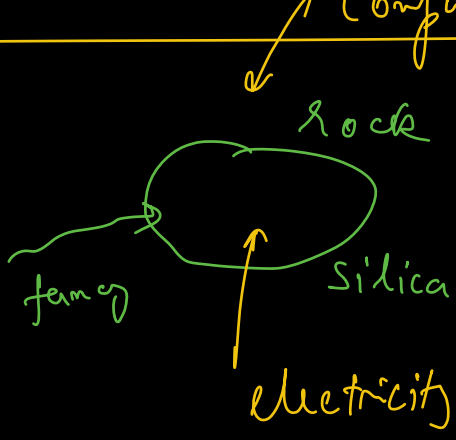
print(s)

① detect the type of s

② s is a string

③ s.__str__()

④ for i in range(n):
    print(char)



10 + = 3 → 13

"x" + = "x" → "x" ... "x"

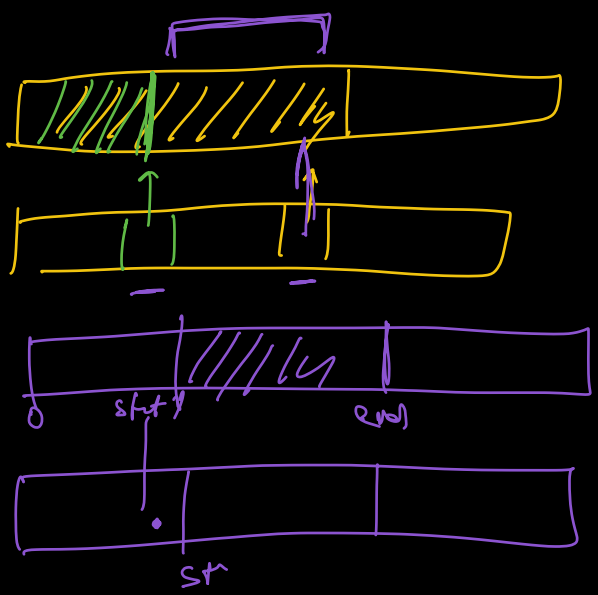
10:30 → 15 min break

10:45 → doubts

---

arr = [1 3 -6 8 4] → sum on [start, end]

prefix sum = [1 4 -2 6 10]



prefix[end] O(1)

⊖ prefix[start-1] O(1)

O(1)

---

print(var) → O(1) time, O(1) space if variable is int/bool

→ O(n) time, O(1) } for array/string

s = "_____"

print(s) ← O(1) space

print(s[2:-5]) ← O(n) space

copy

---

most modern languages do "automatic memory mangent"

↳ unused memory is freed automatically thanks to the garbage collector

Python, java, Ruby, PHP, javascript, go

some languages require you to manage the memory manually.

↳ C++, Cobol, pascal

some langages don't have a garbage collector but still free the memory for you

↳ ownership & memory

↳ Rust

Subarray

the no of all subarrays

print all subarrays

find sum of each subarray

- brute force
- prefix sum
- carry forward

Find the (total) sum of all subarrays.

arr = [1 3 0 -6]

[1]$_1$ [1 3]$_4$ [1 3 0]$_4$ [1 3 0 -6]$_{-2}$

[3]$_3$ [3 0]$_3$ [3 0 -6]$_{-3}$

[0]$_0$ [0 -6]$_{-6}$

[-6]$_{-6}$

$1 + 4 + 4 + -2 + 3 + 3 + -3 + 0 + -6 + -6$

$= -2$

$a_1 = [1 \quad 2]$

$[1]_1 \quad [1 \quad 2]_3 \quad [2]_2 \qquad = 1 + 3 + 2 = 6$

total_sum = 0 ✓ integer.

```
for start in range(n):                O(n)
    sum = 0
    for end in range(start, n):       O(n)
        sum += arr[end]
        total_sum += sum
```

✓

```
print(total_sum)
```

TC = $O(n^2)$

SC = $O(1)$

```
sum = 0
for s in range(n):
    for e in range(s, n):
        sum += arr[e]
```

print(sum) X incorrect.

Can this be done better?

[1 2 3 4 5]

(1) + (1+2) + (1+2+3) + (1+2+3+4) + (1+2+3+4+5)

+ 2 + 2+3 — —

(1+2+3+4+5)

+ (2+3+4+5) X

+ (3+4+5)

+ (4+5)

+ 5

Dry Run

good

```
total = 0
for s ← 0..n
    sum = 0
    for e ← s..n
        sum += arr[e]
        total += sum
print(total)
```

bad

```
sum = 0
for s ← 0..n
    for e ← s..n
        sum += arr[e]
print(sum)
```

arr = [1, 2, 3]

total	s	e	sum
0 1 3 6 10 12 17 20	0 1 2	0 1 2 3	0 1 3 6 3 5 2

20

sum	s	e
0 1 3 6 4	0 1 2	0 1 2 3 2

14

14 x in comect:

Can you calculate the sum of any subarray in O(1) time?

Can we read entire array in O(1) time ✗ No.

is each elmt important? Yes → each elm must be read.

Cannot calculate sum of any subarray in O(1) time.

O(n) time? → yes –

Maximum Subarray Sum

print the sum of the subarray with the max sum.

Morgan Stanley, Microsoft, Amazon, VMWare, DE Shaw.

① Brute force!

```
max = -∞
start ——
    end ——
        sum = 0
        i ——
            sum += arr[i]
        if max < sum
            max = sum
```

$O(n^3)$ TC

$O(1)$ SC

② $O(n^2)$ Carry forward technique

```
max = -float('inf')
for start in range(n):
    sum = 0
    for end in range(start, n):
        sum += arr[end]
        if max < sum:        ← can't comp NoneT w. INT
            max = sum
print(max)
```

TC = $O(n^2)$

SC = $O(1)$

- 999999, None, int-min, -inf, big negative no, -sys.maxsize - 1
ok but avoid

float('inf'), 0, an [0]

sum = 0 ⟶ ∵ identity$_+$ is 0

for i in range(n):
    sum += a[i] ⟶ operator is +

print("sum of array is " + sum)

identity$_+$ is the value that has no effect when + is applied.

$\underbrace{0}_{\text{id}} \underbrace{+}_{\text{operator}} x = x$

$\boxed{1} \underbrace{X}_{\text{operator}} x = x$

identity for min = ∞

min(∞, x) = x

max(−∞, x) = x

In most languages integers don't represent $-\infty$ $+\infty$

↳ INT_MAX (99999)

INT_MIN (-99999)

In every language float can represent inf -inf

float('inf') ⇒ infinity

-float('inf') ⇒ -infinity

Typing

↳ strong: C++ java Python Ruby Rust

↳ weak: C++ java javascript

↳ Static: C++ java Rust

↳ Dynamic: Python Ruby javascript.

0 < None → error in python

0 + [] → error

0 + '' → error

---

find the max sum subarray in O(N) time O(1) space

Kadane's algorithm.

How many subarrays of length K?

arr = [1, 3, 5, 6, 7]

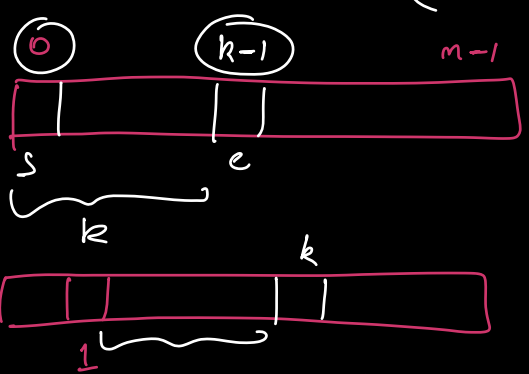
How many subarrays of length

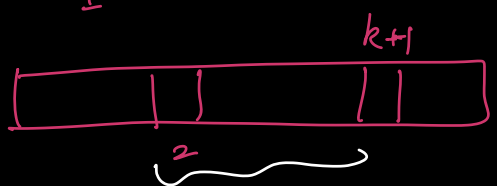
0 → 0
1 → 5
2 → 4
3 → 3
4 → 2
5 → 1
6 → 0

How many subarrays of length K?

(0, k-1) (1, k) (2, k+1) -- (n-k, n-1)

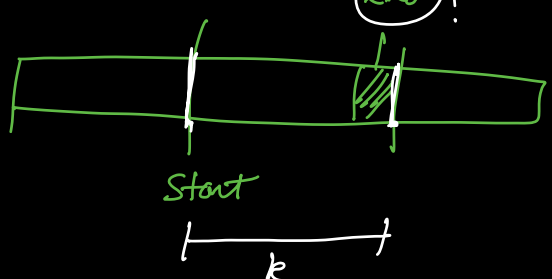
(n-k+1)

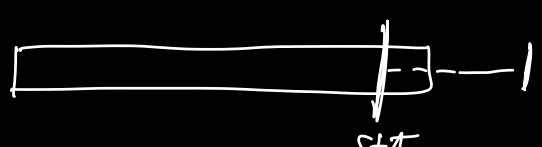


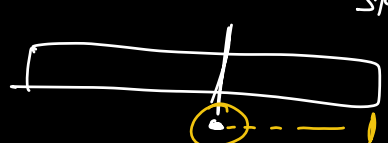


(n-k)

(start+k-1)

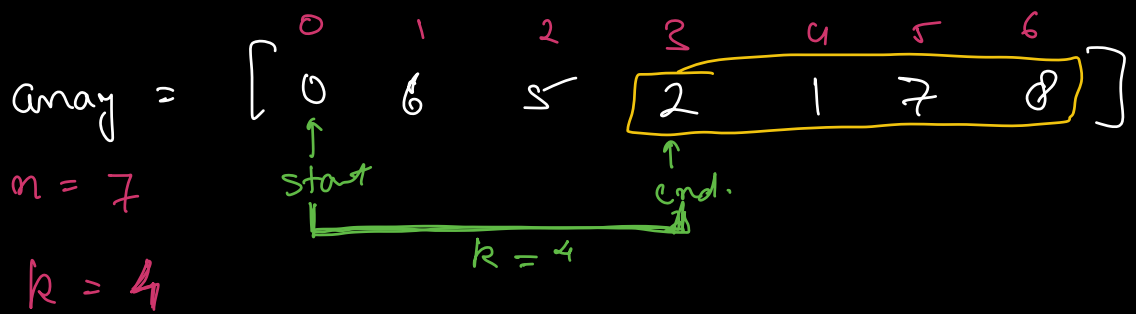






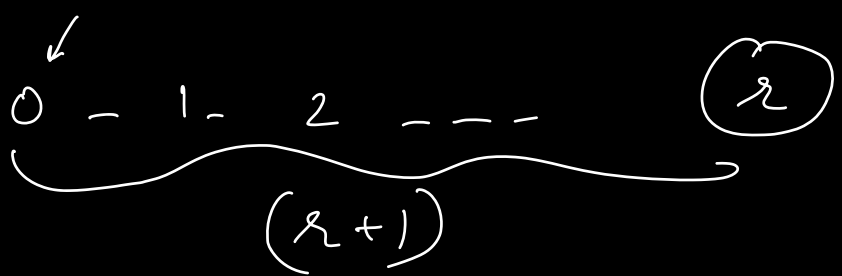
s+k-1 = n-1

$$s = n - k - (k-1)$$
$$= n - k$$



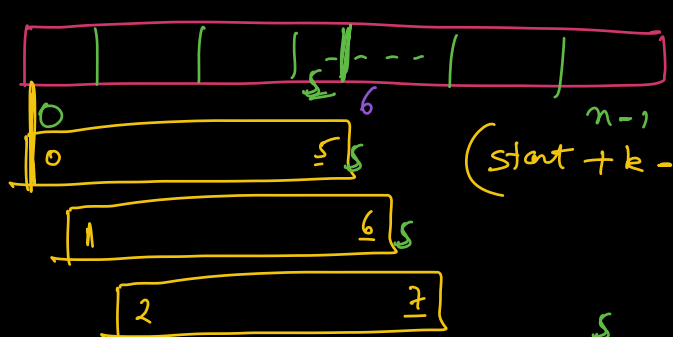
[0 6 5 2] [6 5 2 1] [5 2 1 7] [2 1 7 8]

(end − start + 1) = k



Summary

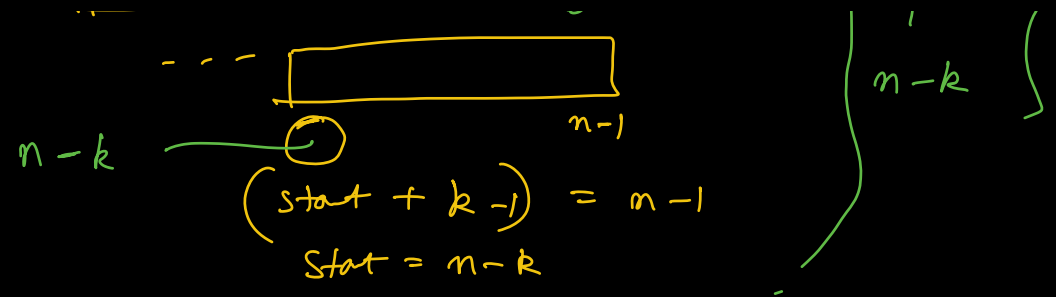
Count the no of subarrays of length k. = 6



(start + k − 1) = end

|arr| = n

n − k + 1



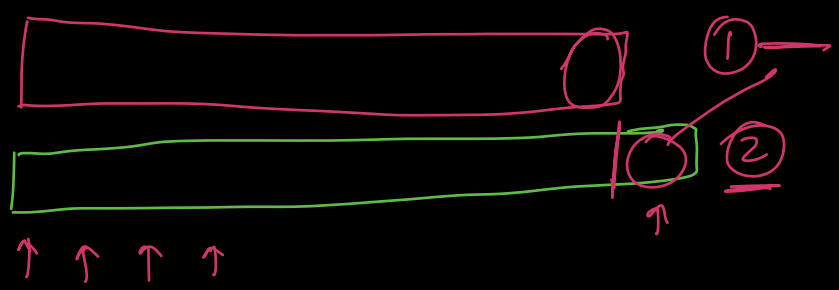
---

arr = 100

the no of subarrays of length 17

= 100 - 17 + 1 = 84

Carry-forward / on-the-fly computation.



print the sum of each sub that starts at 0

start = 0

```
for end in range(start, n):
    (start, end)
    sum = 0
    for i ← start - end:
        sum += arr[i]
    print(sum)
```

n or n-1

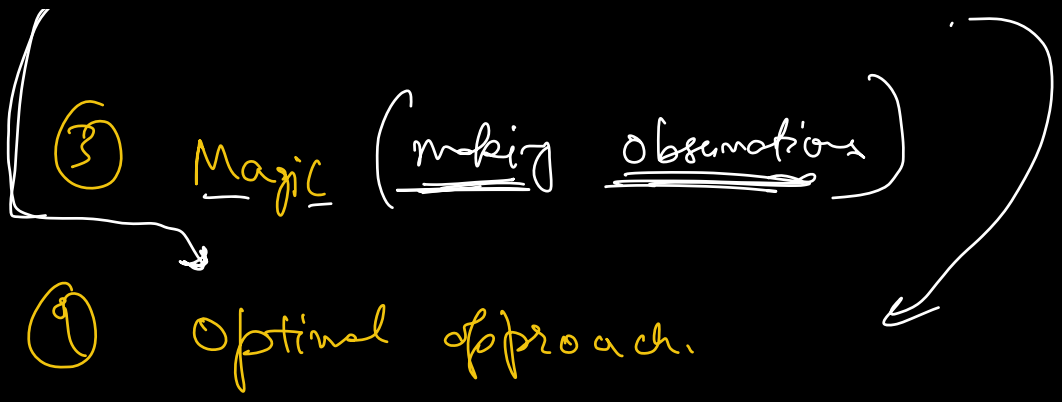
```
start = 0
sum = 0
for end in range(start, n):
    sum += arr[end]
    print(sum)
```

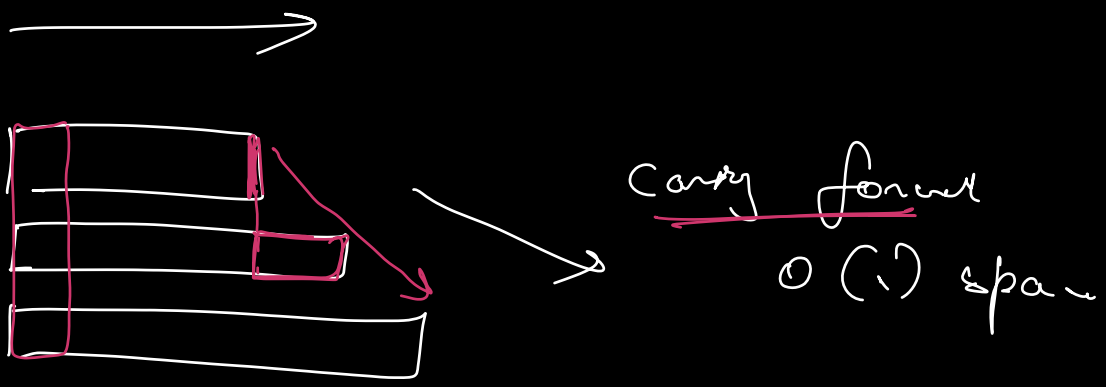
→ accumulate the sum on the fly.

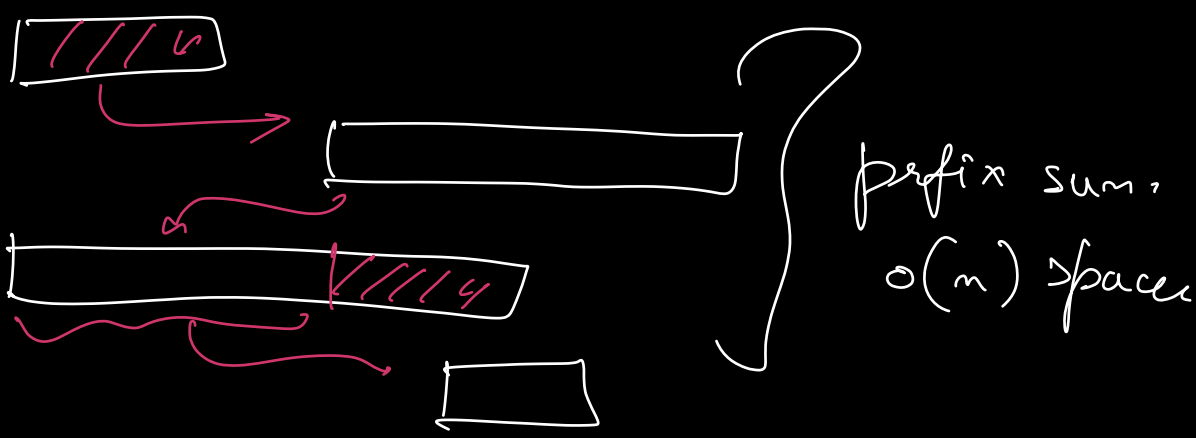
calculate the prefix sum on the fly.

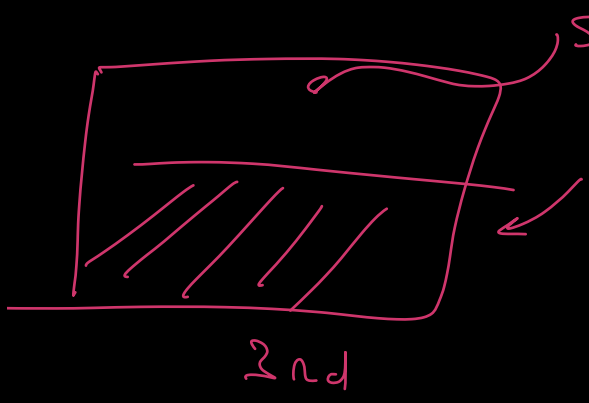
---

1. Clarify the problem
   - ↳ ask questions
   - ↳ draw examples
2. Brute force approach
   - ↳ better late than never

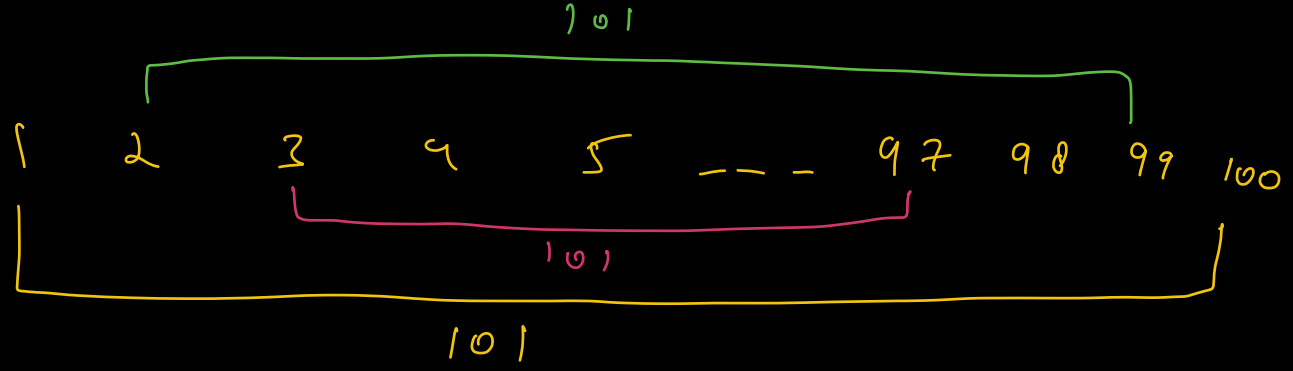

③ Magic (making observations)
① optimal approach.


Carry forward
O(1) space


prefix sum.
O(n) space



Guass → V.V. famous Mathematician.



1 + 100	= 101
2 + 99	= 101
3 + 98	= 101
4 + 97	= 101
⋮	
98 + 3	= 101
99 + 2	= 101
100 + 1	= 101

100 times.

1 -- 100 $\quad$ 101×100

100 -- 1

$S * 2$

$S = \dfrac{101 \times 100}{2}$

1 $\quad$ n

$1 + n = n+1$

$2 + (n-1) = n+1$

$3 + (n-2) = n+1$ $\quad$ } n times

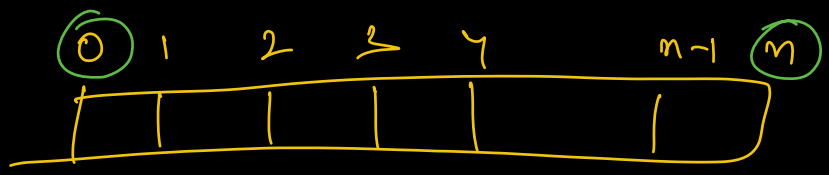
$(n-2) + 2 = n+1$

$n + 1 = n+1$

$2 * S = n * (n+1)$

$S = \dfrac{n(n+1)}{2}$ } gauss trick.

$\dfrac{n(n+1)}{2} = {}^{n}C_{2} =$



Start, end

$$(n+1)C_2 = \frac{n(n+1)}{2}$$

DSML → Data Scien & ML

DSA is imp of

7 month of DSA

Basics first

DS

DSA important

Academy

DSA → start

LLD

HLD

Project

---

① Tabij Notes

Kadane's algo → Don't carry bad baggage.

support@scaler.com.