Agenda

- AP and GP
- Time Complexity
  - Big O notation
  - Comparison of Orders of TC
- Space Complexity

Arithmetic Progression

1 2 3 4 . - - - - 10

$n \times (n+1)/2$

$$\frac{\cancel{10}^{5} \times (10+1)}{\cancel{2}}$$

$$\boxed{\frac{n^2+n}{2}}$$

$$\underbrace{4 \qquad 7}_{3} \; \underbrace{\qquad 10}_{3} \; \underbrace{\qquad 13}_{3} \quad - - - - - -$$

$- \; . \; - \; . \; -$

$a \qquad a+d \qquad a+2d \qquad a+3d \; - - -$

$a = 4$
$d = 3$

A.P

$$\frac{N}{2}\left[2a + (N-1)d\right]$$

Ex-2

$5 \qquad 10 \qquad 20 \qquad 40 \quad - - - \quad N$

$\times 2 \qquad \times 2 \qquad \times 2$

$a \qquad ar \qquad ar^2 \qquad ar^3 \; - - -$

$5 \qquad 5\times 2 \qquad 5\times 2^2$

$a = 5$

$r = 2$

$$\left.\right\} \quad \boxed{\frac{a\times(r^n - 1)}{r-1}}$$

Concept - 1

$N \Rightarrow N/2 \Rightarrow N/4 \text{ ------ } \boxed{1}$

① $\log_2 N \Leftarrow$ No of Steps

$$\underbrace{N \quad \frac{N}{2} \quad \frac{N}{2^2} \quad \frac{N}{2^3} \text{ ---- } \frac{N}{2^K}}_{\text{no of Elements}}$$

$$\frac{N}{2^K} = 1$$

$$N = 2^K$$

$$\log_2 N = \log_2 2^K$$

$$K \rightarrow \log_2 N$$

2^3

$\log_2 (2^3)$

$3 \log_2 2$

3×1

$\log_a a \rightarrow 1$

$n \rightarrow 1$

but at Every Step, reduce the number to its half ✓

$\Rightarrow \log_2 N \longrightarrow$ Binary Search

Binary Tree

MergeSort

Heaps

Q

$1 \longrightarrow N$

but Every time we multiply cur_num with 2

Total number of Step to reach to N ?

$$\underset{a}{[3,}\ \underset{b}{10]} \Rightarrow (b-a+1)$$

$$\Rightarrow (10-3)+1 = 8$$

Time – Complexity

① measure of Efficiency

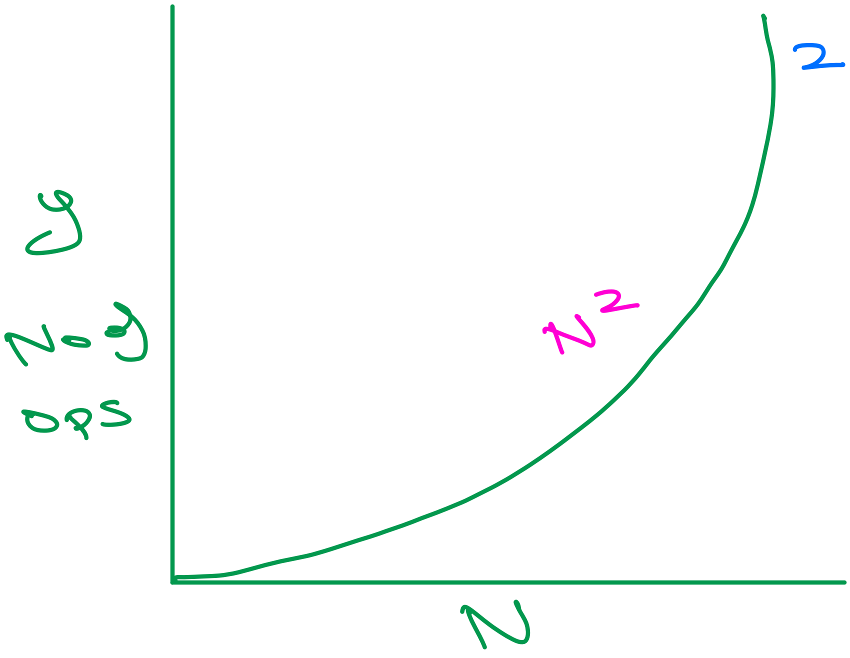
② Relationship between number of Ops vs Size of input

How

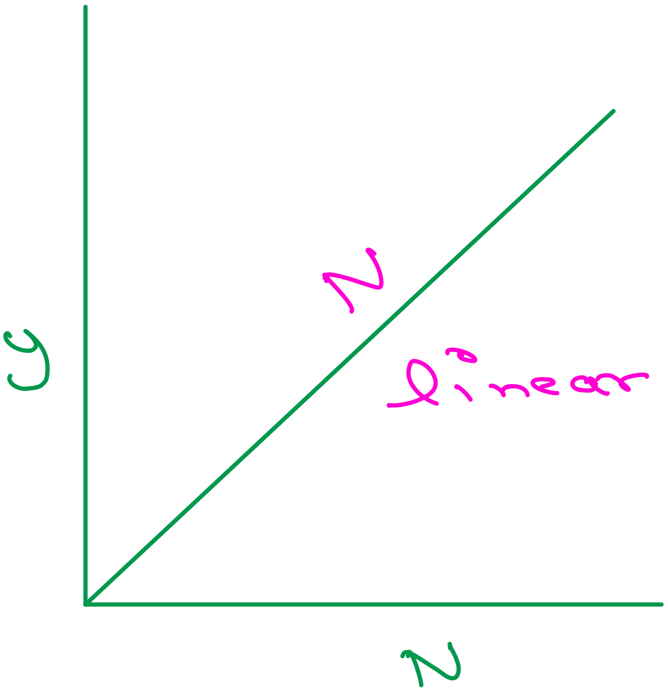
Vaibhav ← → Amit

Vaibhav	Amit
2 secs	3 secs
3 GHz	1 GHz

we cannot say ?



Vaibhav



Amity

- 10 → 100
- 1000 → 1000,000

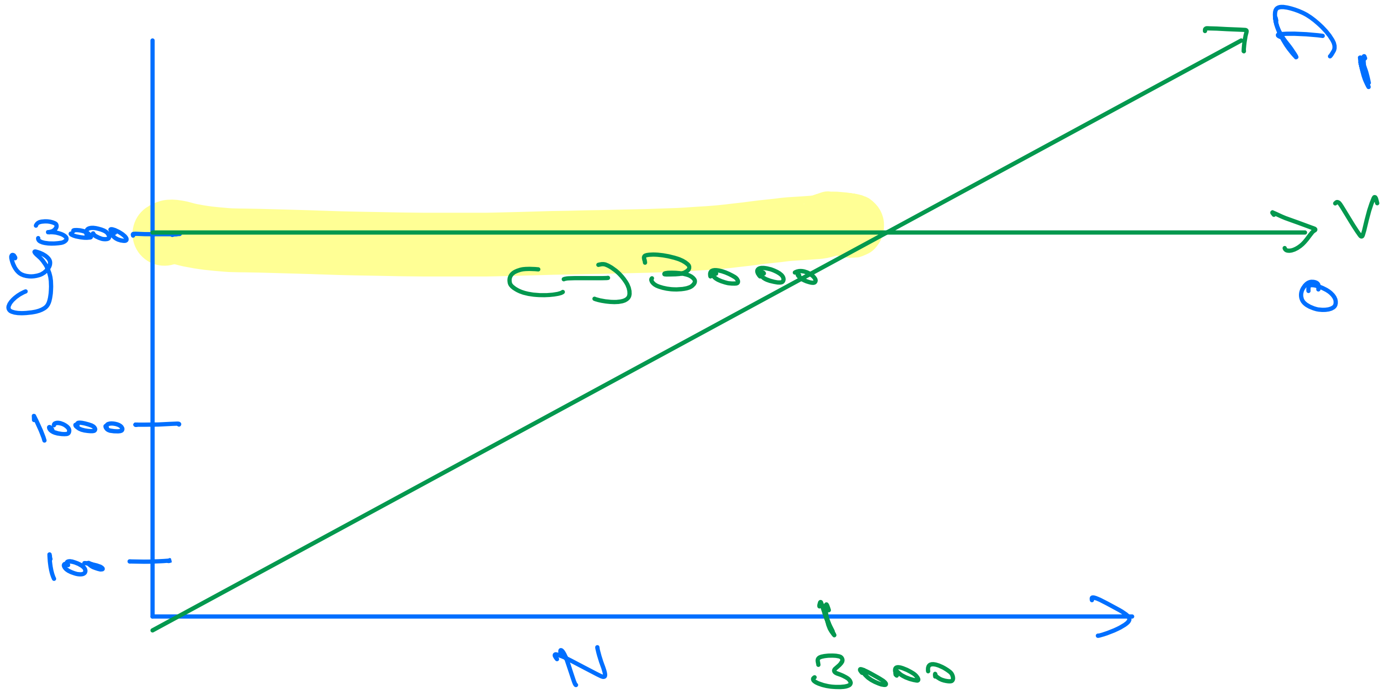
10 → 10

1000 → 1000

1 million order

1 million

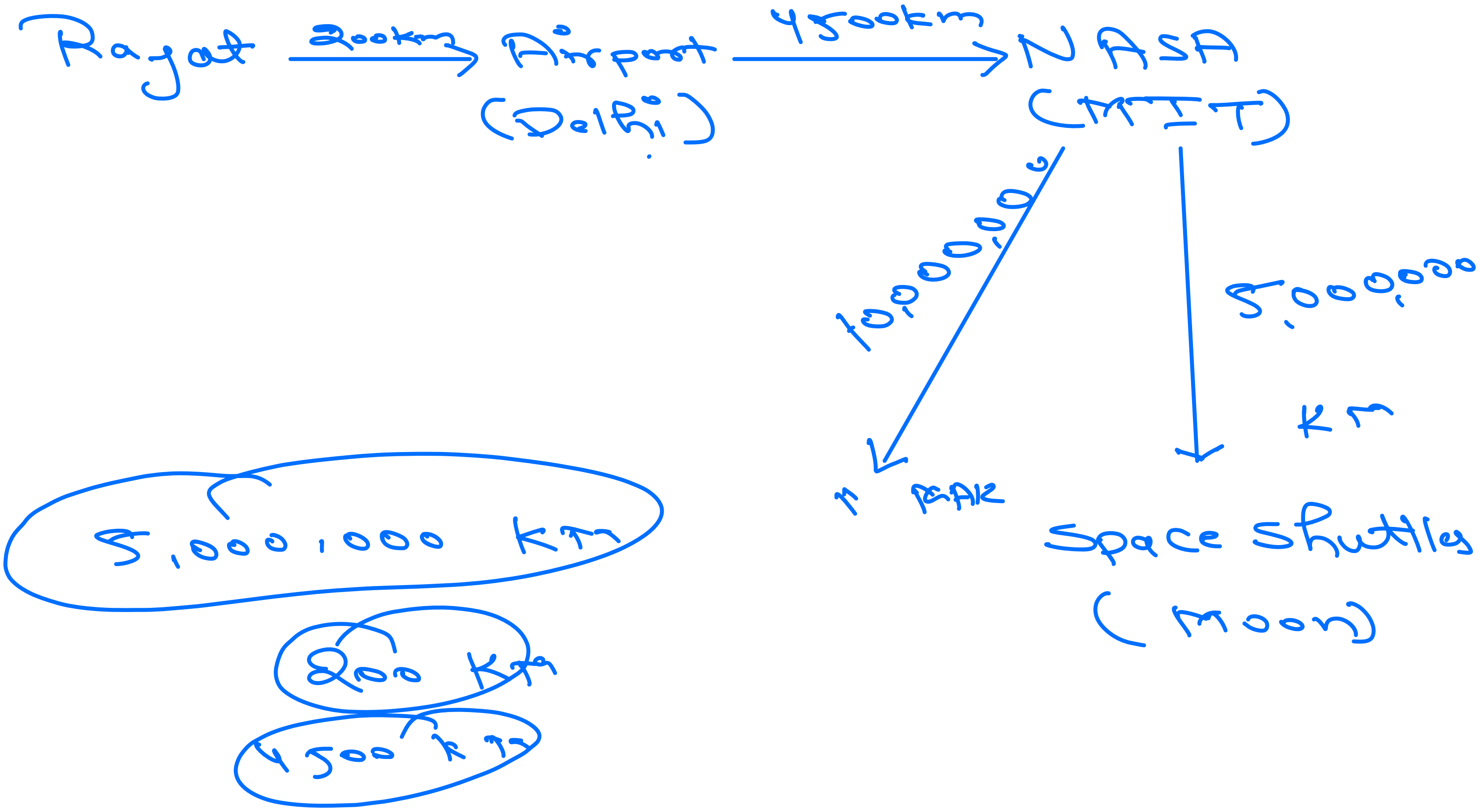
100 million



Long Term ⇒ V

① TC ⇒ measures the trend
ignoring constant factors
and Co-efficient

Big O notation ⇒ Only the
highest Poly
degree



* Big O Notation only focuses on the highest Polynomial Degree i.e. Trend of the Equation

$$\boxed{TC \Rightarrow 10n^2 + 3n + 50}$$

$\downarrow$

$O(n) \Rightarrow \cancel{10}\, n^2 \Rightarrow n^2$

$N \Rightarrow 100$

$$\begin{tikzcd}
100 \arrow[d, "N/2"] \\
50 \arrow[d, "N/2"] \\
25 \arrow[d] \\
12 \arrow[d] \\
7 \arrow[d] \\
4 \arrow[d] \\
2 \arrow[d] \\
1
\end{tikzcd}$$

① ② ③ ④ ⑤ ⑥ 7

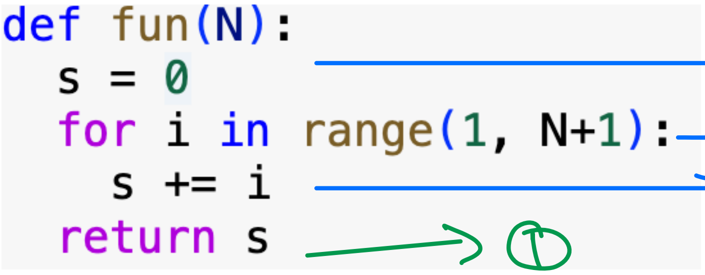
$\log_2 100$

$1 \Rightarrow 100$

N×2

N×2

1 — ?



```python
def fun(N):
  s = 0
  for i in range(1, N+1):
    s += i
  return s
```

1

1

N - 1 + 1

b - a + 1

1

$N - \cancel{1} \cancel{+1} + 1 + 1 + 1$

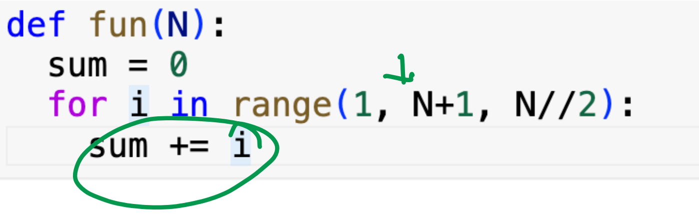
$TC \Rightarrow N + \cancel{3}$

$O \Rightarrow N$

$[a \rightarrow b]$

$\rightarrow$

$b - a + 1$



```python
def fun(N):
  sum = 0
  for i in range(1, N+1, N//2):
    sum += i
```

$\Rightarrow \quad C \Rightarrow O(1)$

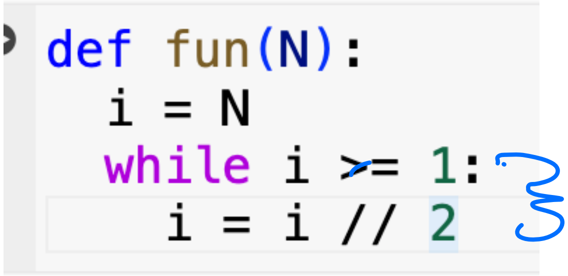
for i in range(1, 100, 50)

2

N

1, 1000, 500

2



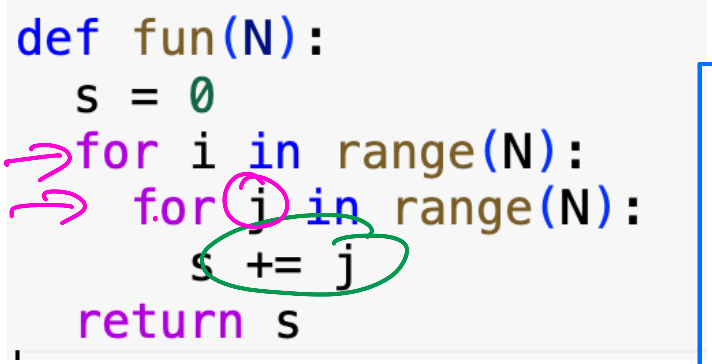
```
def fun(N):
    i = N
    while i >= 1:
        i = i // 2
```

	Before i	After i
1	N	$N//2$
2	$N//2$	$N//2^2$
3	$N/2^2$	$N/2^3$
k		$N/2^k$

↓ 1

$O(\log_2 N)$

$O(\infty)$



```
def fun(N):
  s = 0
  for i in range(N):
    for j in range(N):
      s += j
  return s
```

i	j	iterations
1	1 → N	N
2	1 → N	N
3	1 → N	N
N	1 → N	N

$$\underbrace{N + N \ldots N}$$

$n \times N \Rightarrow n^{~~**~~2}$

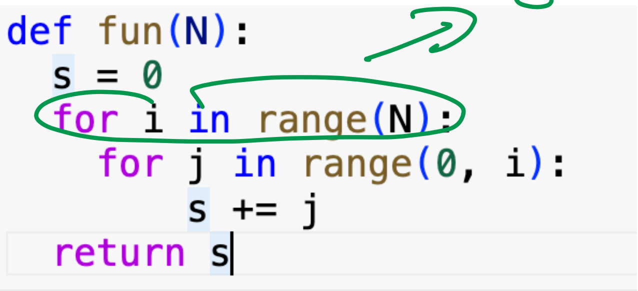
i → 1, 2

j → 1, 2

i	j
1	1 → 2
2	1 → 2

2 + 2 = 4

$2 + 2 + 2 \ldots N \Rightarrow N \times 2$



```python
def fun(N):
  s = 0
  for i in range(N):
    for j in range(0, i):
        s += j
  return s
```

0, n-1

i	j	iteration
0	0	0
1	0 → 1	1
2	0 → 2	2
	0 → 3	3
N-1	0 → N-1	N-1

0 1 2 3 - - - N-1

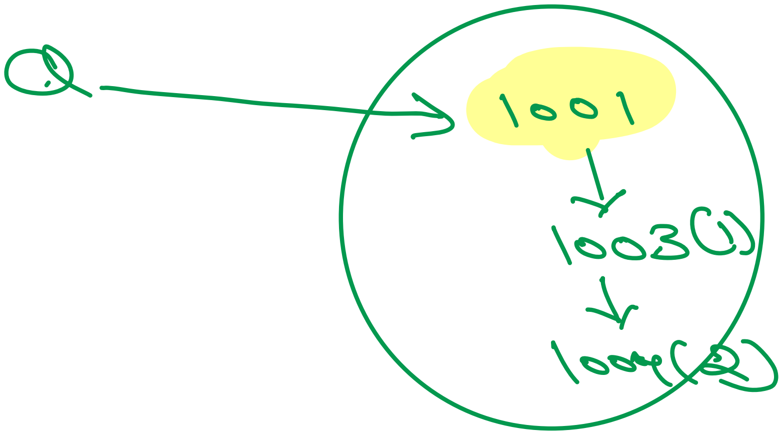
$$\frac{n \times (n+1)}{2} \Rightarrow \frac{n^2}{\cancel{2}} + \cancel{\frac{n}{2}}$$

$O(n^2)$

Space Complexity

- Measure Extra Memory a program/Algo needs to complete Execution

- Its Analysis of memory usage w.r.t Input Size



Q = [ ]

Q.append(1)

Q.append(2)

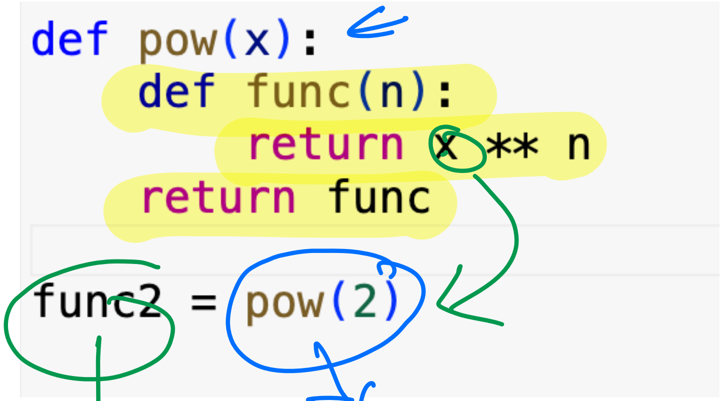
T.C

0	Constant	C
1	logarithmic	$\log_2 n$
2	Linear	n
3	Linear Logarithmic	$n\log_2 n$
4	Quadratic	n^2
5	Exponential	2^n or n^n

Higher Order Function

HOF

0 A function that returns a function



```
def pow(x):
    def func(n):
        return x ** n
    return func

func2 = pow(2)
```

Pow(2) ⟹ def func(n)
return 2**n

retur func

func2 ⟹ func2(10) ⟹ 2^{**10}

anotherFunc ⟹ Pow(100)

anotherfunc(50) ⟹ 100^{**50}