Complexity Analysis

Why we need to understand complexity analysis ?

→ It helps to measure performance of the algorithms.

→ It's very imp for interviews.

photo editor
off

lightroom

→ we should get the desired output as early as possible.

Q. How to measure performance of an algo??

experimental analysis

note the time stamp of start.

algorithm

note the time stam of end

this is not accurate. why ??

RAM

HDD/SSD

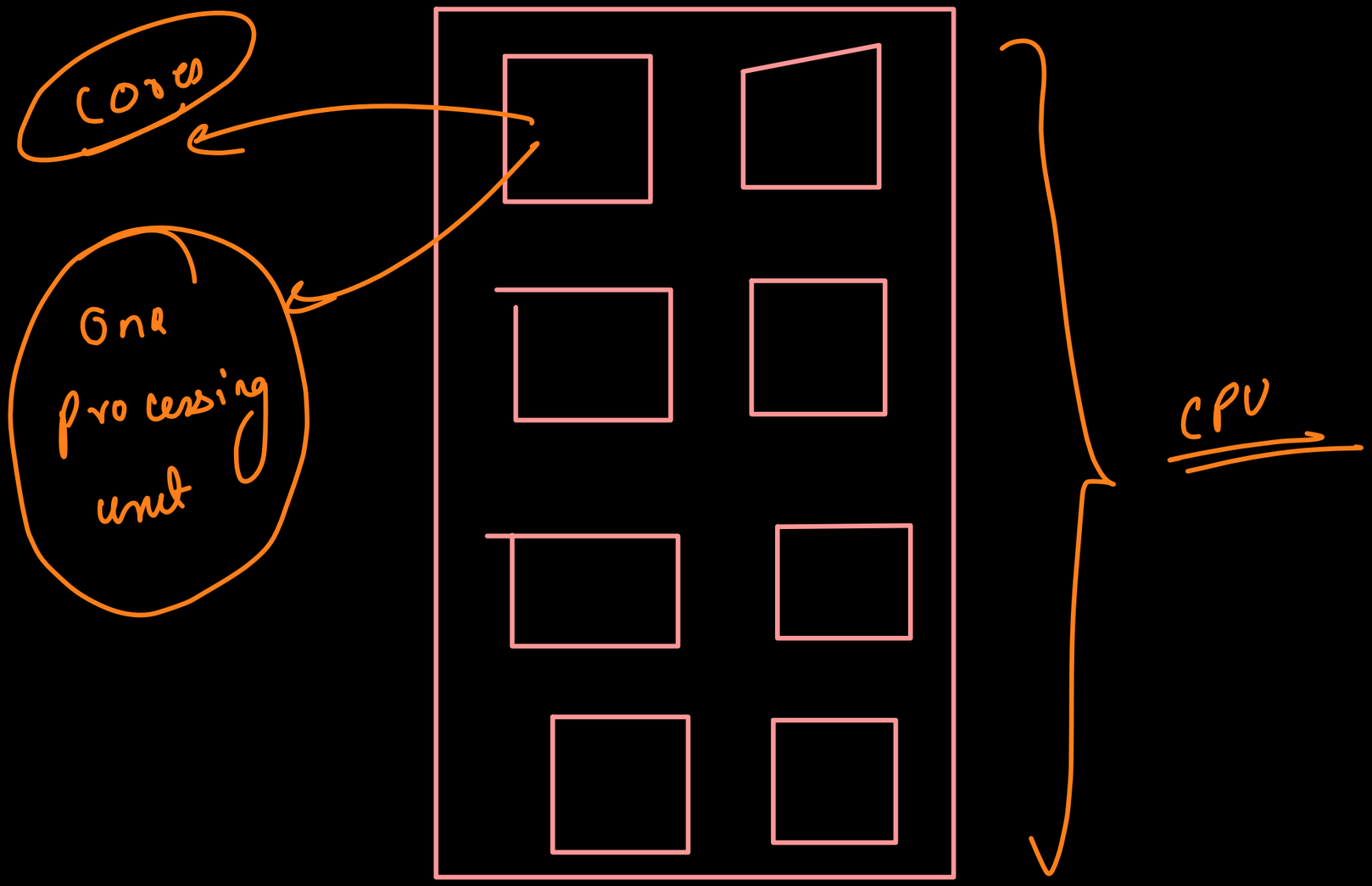
CPU

CPU → cored

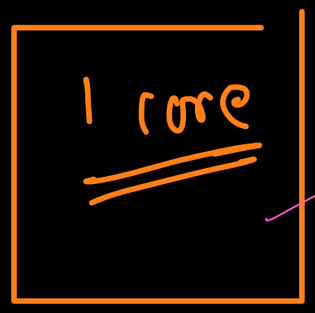
dual core → 2

quad core → 4

octa core → 8


Cores
One
processing
unit
CPU

for simplicity let's consider a system with a single core.



In a single core, 2 or more processes cannot run at the same point of time.

↳ then why we donot feel a lag ??

Content Switching → 1 sec → $\approx 10^8$ instructions

≡

Process → Program under execution

Asymptotic Analysis

Asymptote

← a line that curve approaches at infinity

→ in asymptotic analysis, we try to analyse algorithms for very large input.

→ why we are dealing with very large inputs ??

→ because as a general trend we can see that for large input, algorithms take more instructions to execute.

```
for (i=0 ; i<n ; i++)
    console.log (i)
```

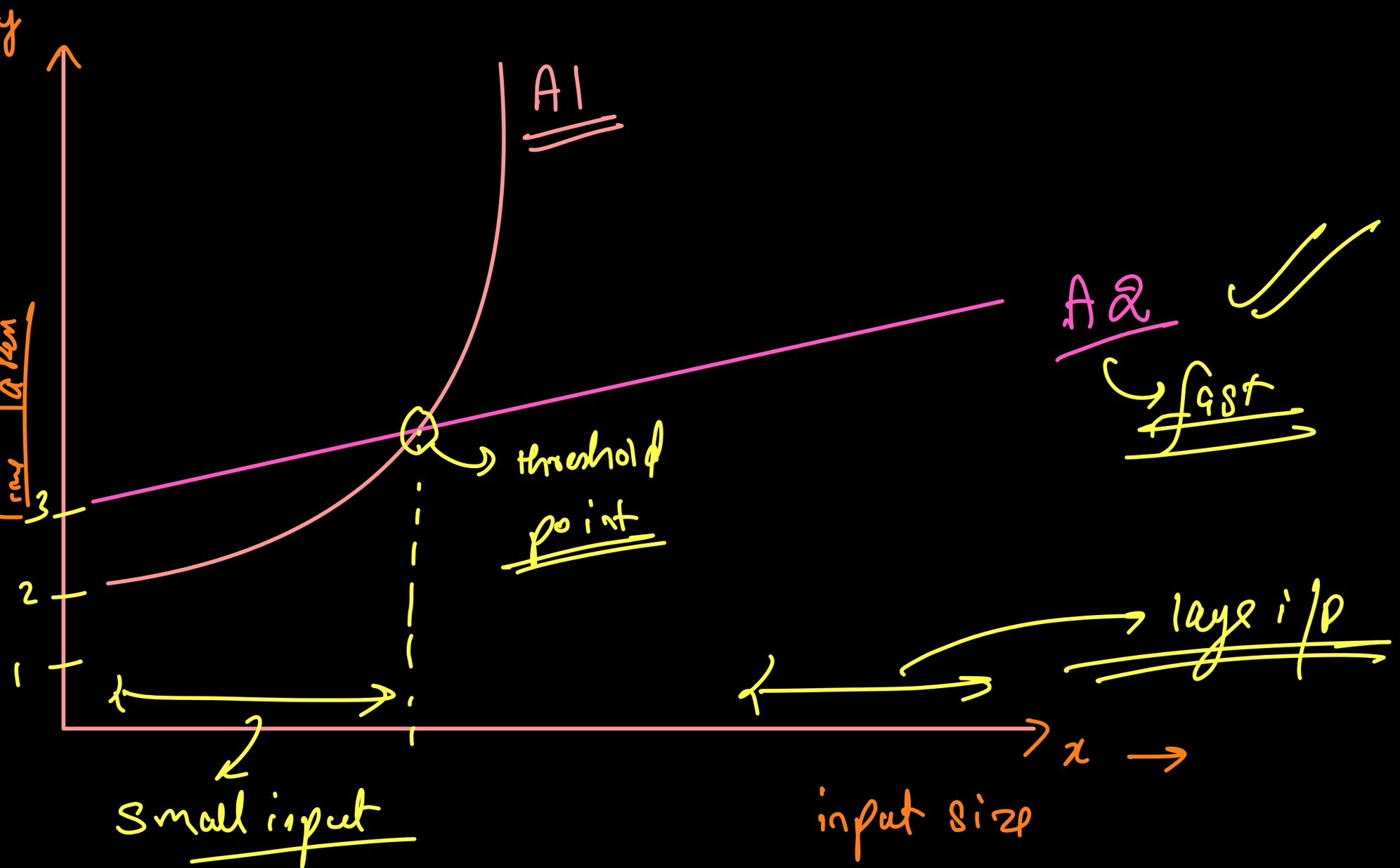
$n \rightarrow 10$

$n \rightarrow 10^6$

```
if (n%2==0)
    console.log ("even")
```

$n \rightarrow 10$

$n \rightarrow 10^9$



Before threshold pt. A1 is faster than A2

But before threshold the input size is Small. & for small i/p size we donot

Case who is Fast:

After threshold pt, input size increases. & then A1 is taking too much time. So after threshold A2 is bts.

Asymptotic Analysis

notations

Big omega Ω ↖

Best ↑

case when you Consider algo will run

Avg ↗ Big Theta Θ

Avg 2 avg

Worst ↗ O Big O

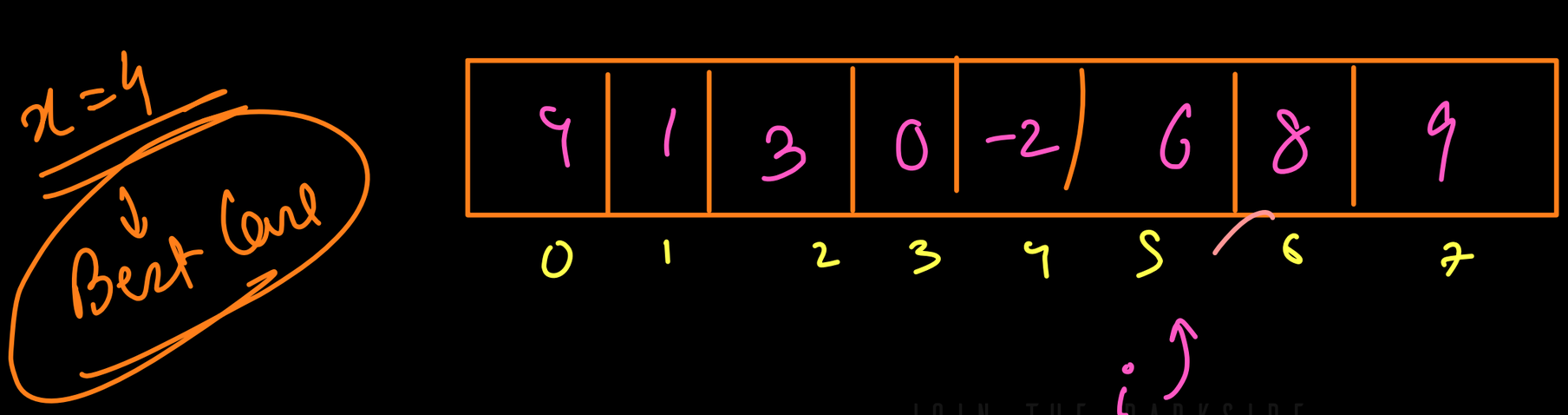
2

if your algo runs, how slow it can run

as fast as possible

Q: Given an array of elements (integers). Apart from the array you've an element x, Return the index at which x is present in the array. If not present return -1.

worst $x = 100, 200, 101 \ldots$



how about, we one by one go to every index & check if x is present.

Time complexity

```
n = arr.length ──→ 1
for (i=0; i<n; i++) {
    if (arr[i] == x) {
        return i;
    }
}
return -1;
```

linear search

worst
↓
you will go to every idx & still not find any key

1 + 1 + 3n + 1
- 1 ← level of arr
- i=0
- for loop
- return

n → large value
↓
n → 10^7

$1 + 1 + 3\times10^7 + 1 \approx 10^9$

const const const

$\longrightarrow 3\times10^7 \approx 10^9$

1 → avoid constant terms

VVVI

Worst Case → $O(n)$ → Big O of n

Best Case → Ω (constant) → $\Omega(1)$

↳ Omission

↳ for linear search is independent of input.

→ constant time we are taking for every n.

Avg → $\Theta(n)$

Fastest

1

$\log(\log(n))$

$\log(n)$

$\sqrt{n}$

n

$n \log n$

$n \sqrt{n}$

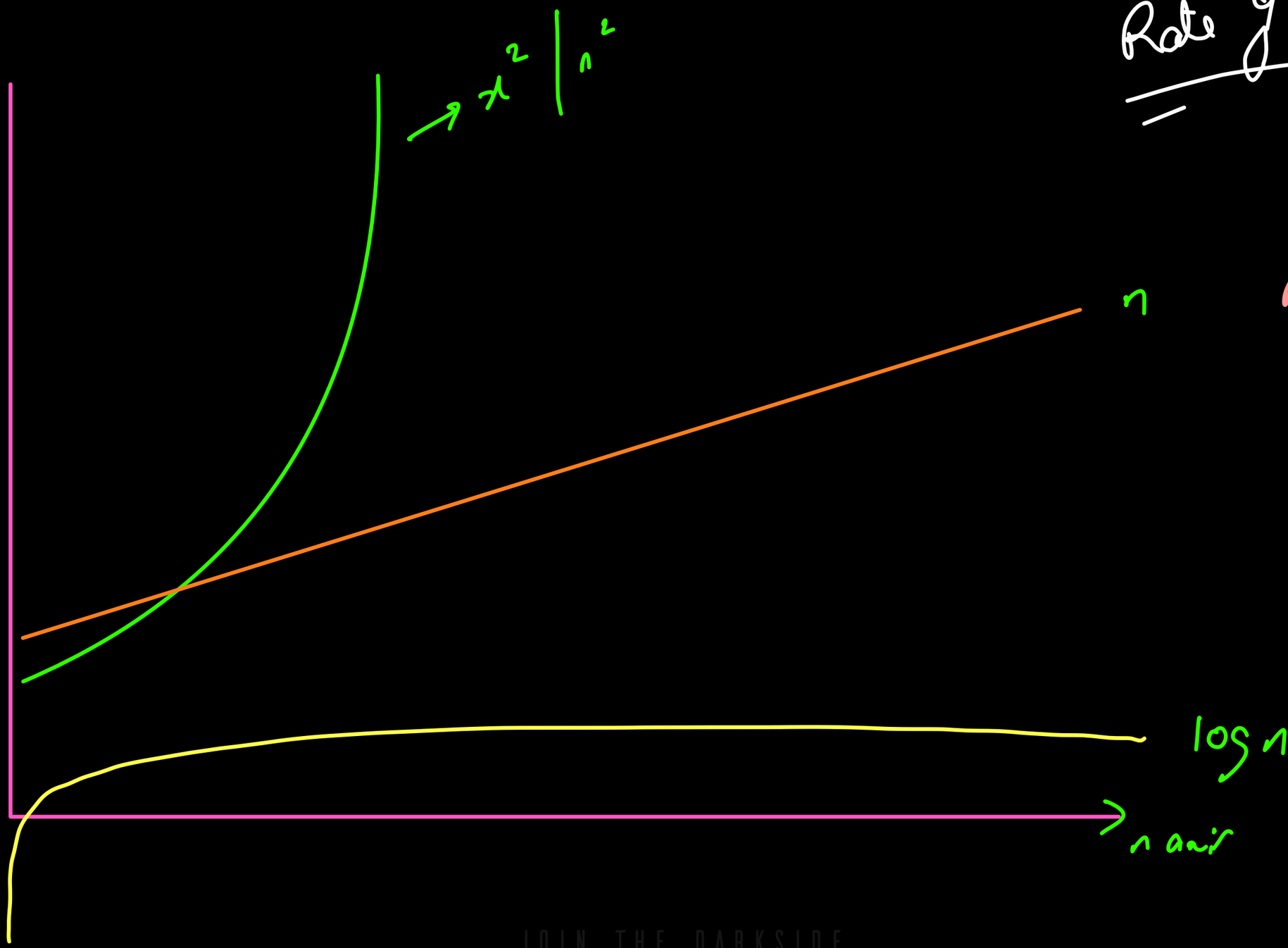
n^2

n^3

$n!$

$2^n, 3^n \ldots$

Slowest



Rate of growth

⇒ in algo we need slow rate of growth.

Worst Case

```
for (i=0 ; i<n ; i++) {
    for (j=0 ; j<n ; j++) {
        console.log (i, j);
    }
}
```

$O(n^2)$

HW

↓

2dsa classes

$i = 0 \longrightarrow 3n$

$i = 1 \longrightarrow 3n$

$i = 2 \longrightarrow 3n$

$\vdots$

$i = n-1 \longrightarrow 3n$

$3n + 3n + 3n \cdots\cdots 3n$

$3(n + n + n \cdots\cdots n)$

↳ n times

$3n^2 \rightarrow \approx n^2$

```
for ( i=1 ; i<n ; i*=2)
    console.log (i)
```

i = 1 ⟶ first iteration

i = 2 ⟶ 2nd ''

i = 4 ⟶ 3rd ''

i = 8 ⟶ 4th ''

⋮ ⋮

i = $2^{(k-1)}$ ⟶ k^{th} iteration

$1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \ldots\ldots\ldots$

$\hookrightarrow \quad 2^{k-1} < n$

$\log_2$ on both sides

$$\log_2(2^{k-1}) < \log_2 n$$

$$(k-1)\log_2 2 < \log_2 n$$

$$(k-1) < \log_2 n$$

$K < \frac{\log_2 n + 1}{}$

the highest value of K

$3K \rightarrow 3(\log_2 n + 1)$

$3\log_2 n \rightarrow \log_2 n$

$\hookrightarrow O(\log n)$