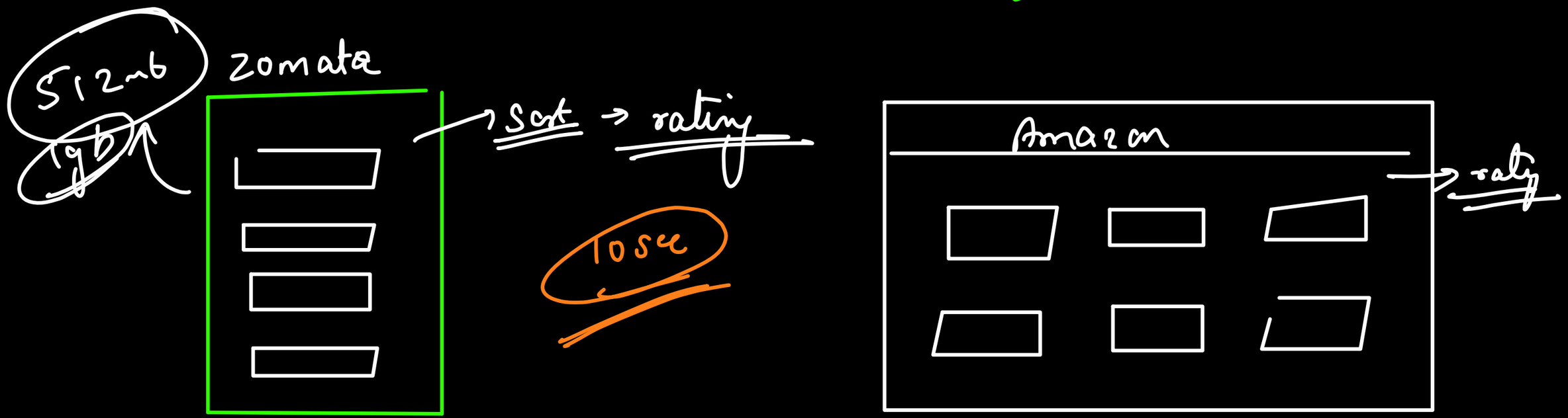
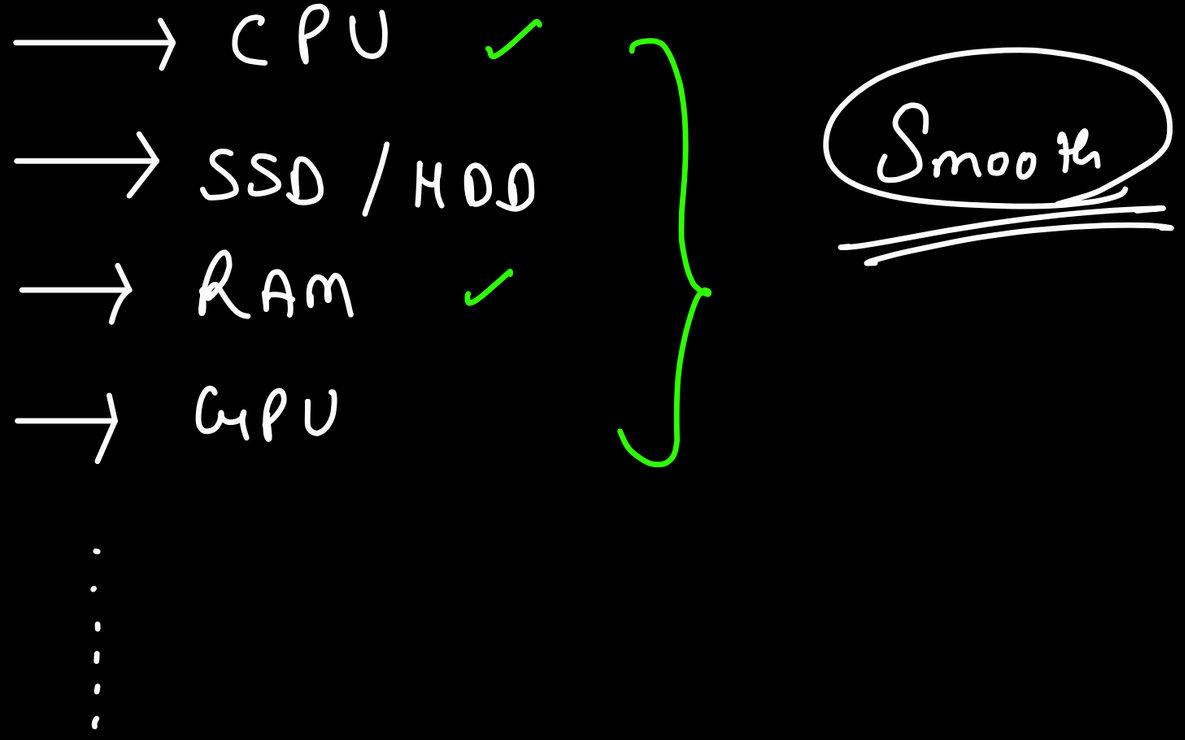
Time And Space Complexity Analysis

Algorithm Complexity Analysis



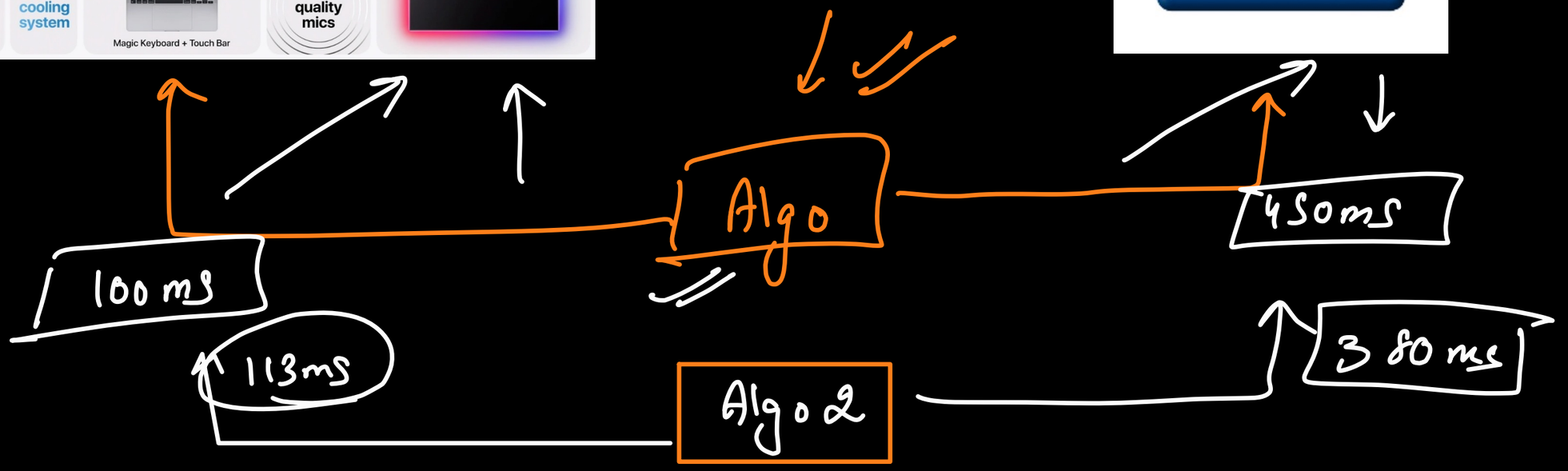
1) Efficiency of an algorithm in terms of time taken is imp.
2) Efficiency of an algorithm in terms of space/memory taken is imp



Experimental Analysis

User 1

User 2



Multitasking

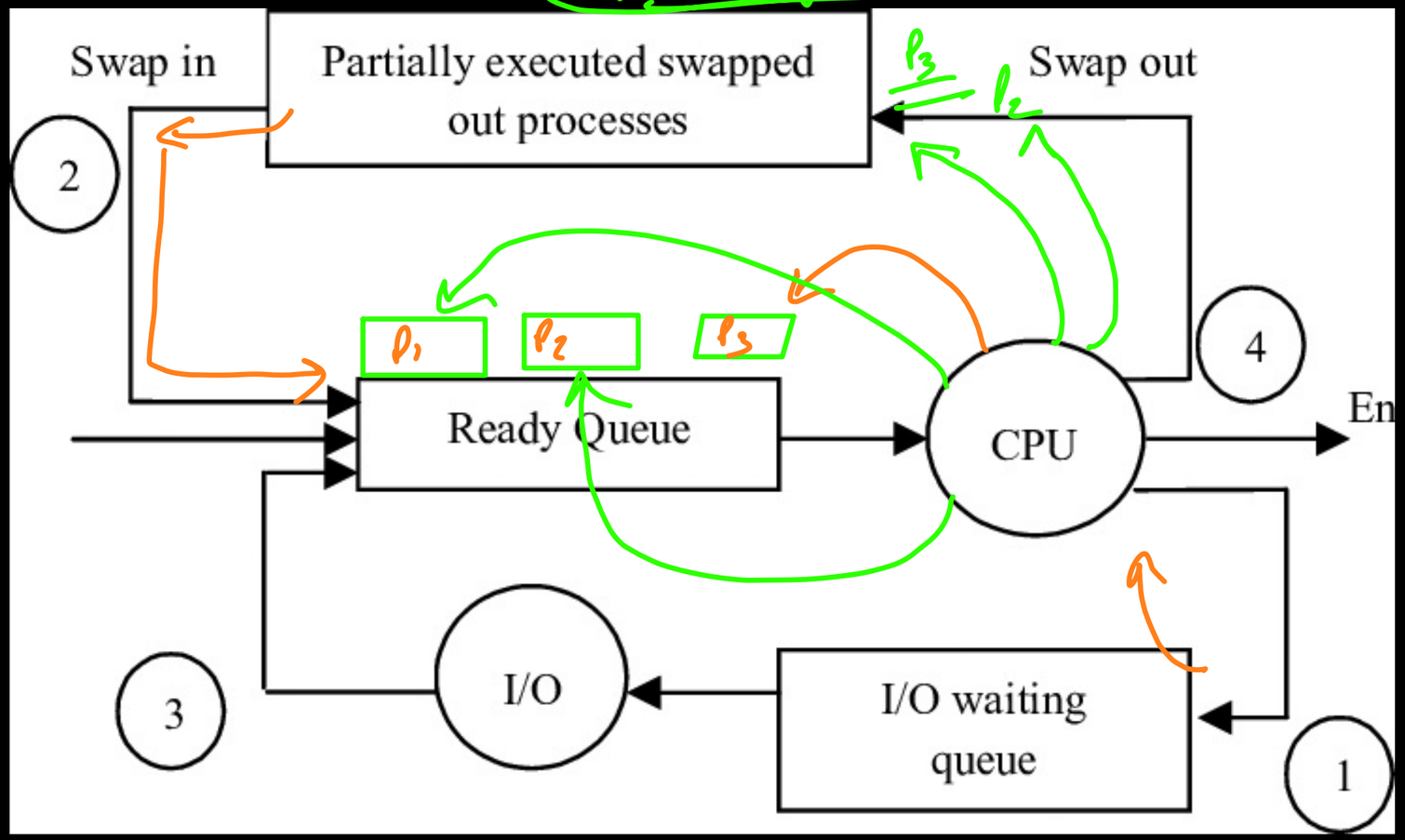
Single core

10.48.35

2 waiting area

1 sec

10^8 instructions



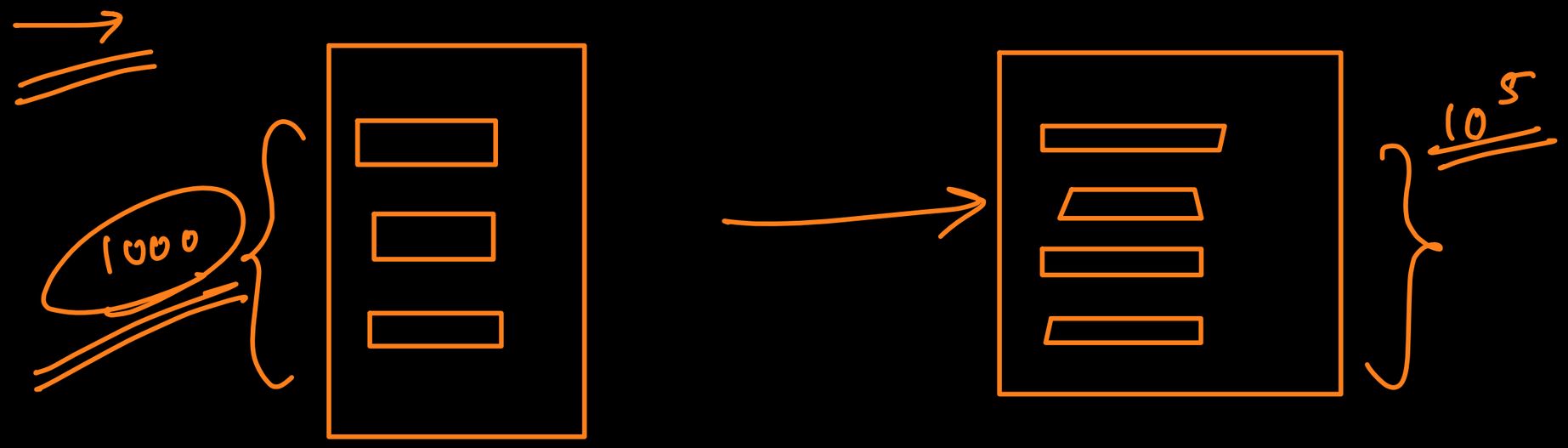
P_3 → 1000 ins

310

P_2 → 5000 ins

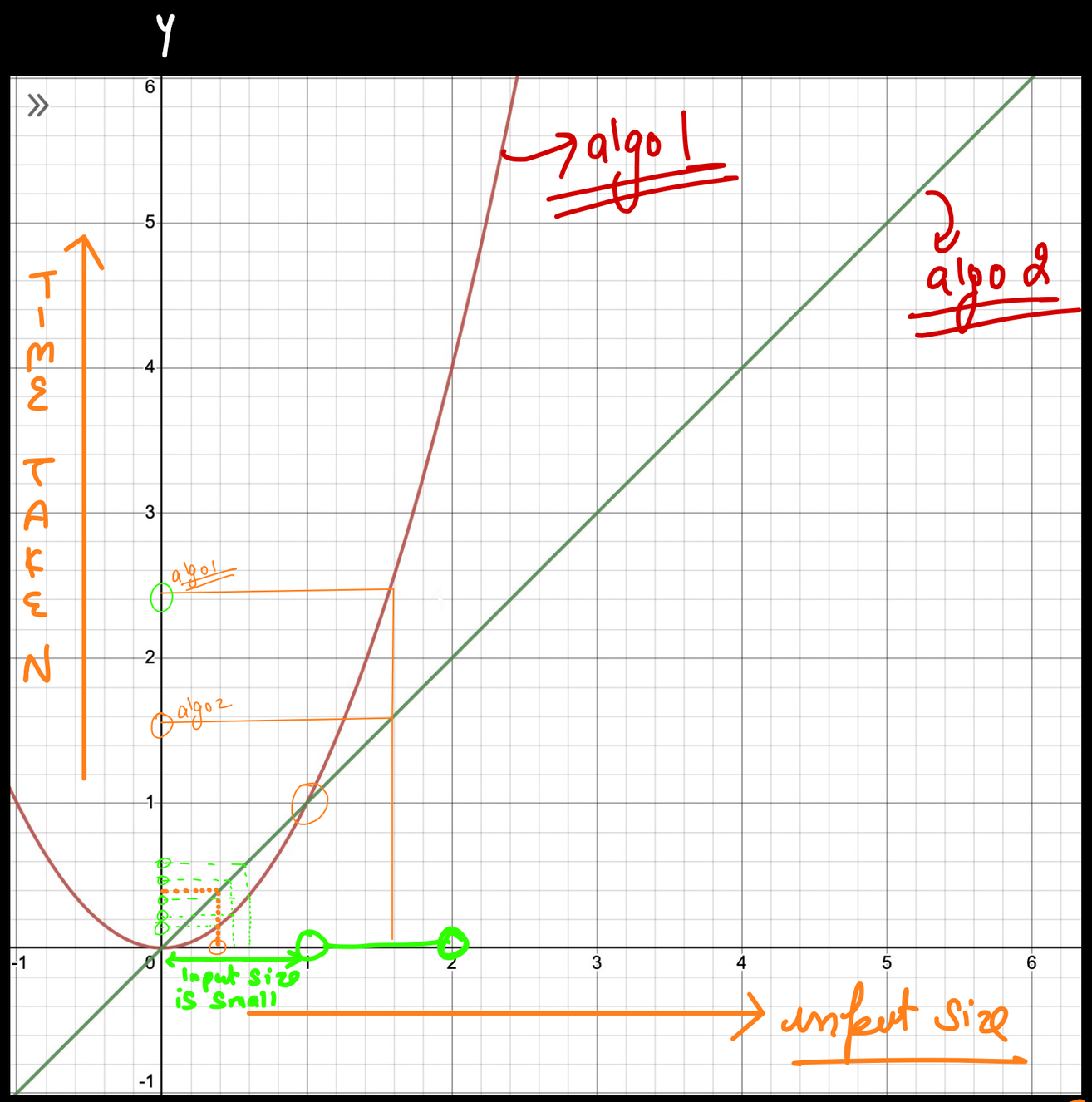
800

1000



→ Input (what, size, how input is given) directly affects the course of algo execution.

Rate of Growth



always we care about how the algorithms are performing for large input size.

if a function changes extremely high with a small change in input then growth is high.

	C_1	C_2
1st	100	20
2nd	105	80
3rd	110	140

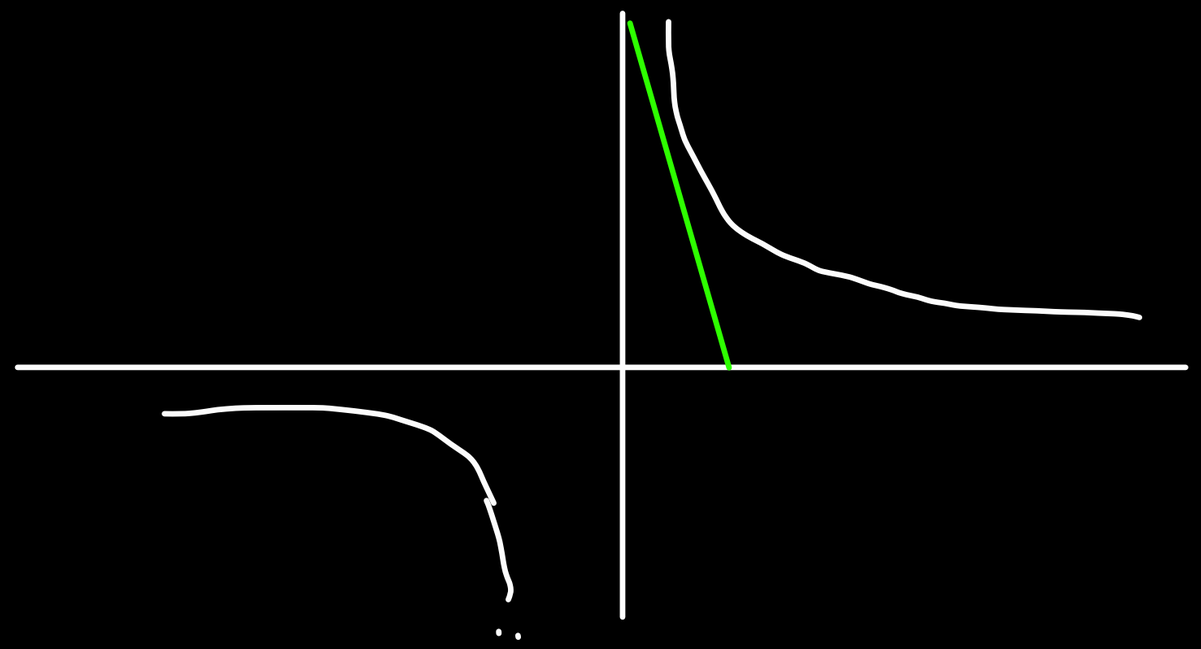
C_2

$\frac{\Delta t}{\Delta i}$ ↑

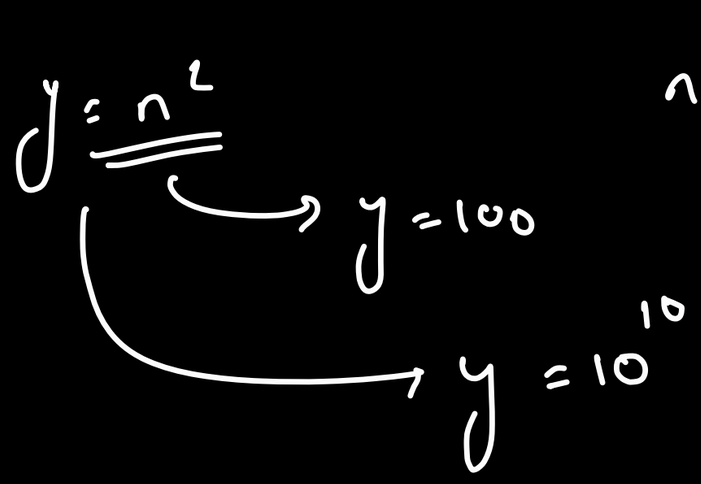
Rate at which running time increases as a func^n of input is called Rate of growth.

Asymptotic Analysis

Asymptote → It is a st. line that constantly approaches a given curve but doesn't meet at any ∞ dist.



→ Rate of growth of algo (running time w.r.t input size)
→ Behaviour of the rate at very large input values.



$n \to 10$

$n \to 10^5$

$y = n \times \sqrt{n}$

$n \to 100$

$n \to 10^4$

$100 \times 10 \to 1000$

$10^6 \times \sqrt{10^6} \to 10^9$

$y = n \log n$

$n \to 10^3$

$y = 10^3 \times 10$

$\to 10^4$

$n \to 10^6$

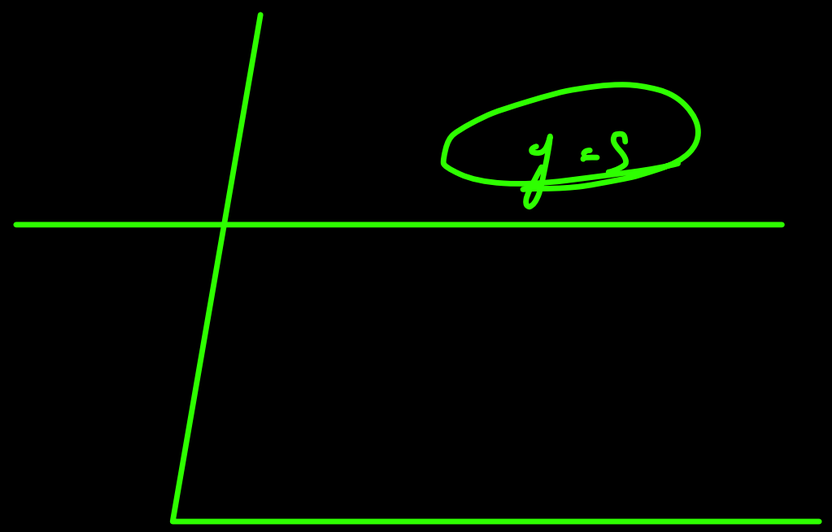
$y = 10^6 \times 20$

$\to 2 \times 10^7$

$y = n$

$y = 10^3$

$y = 10^6$



high rate of grout

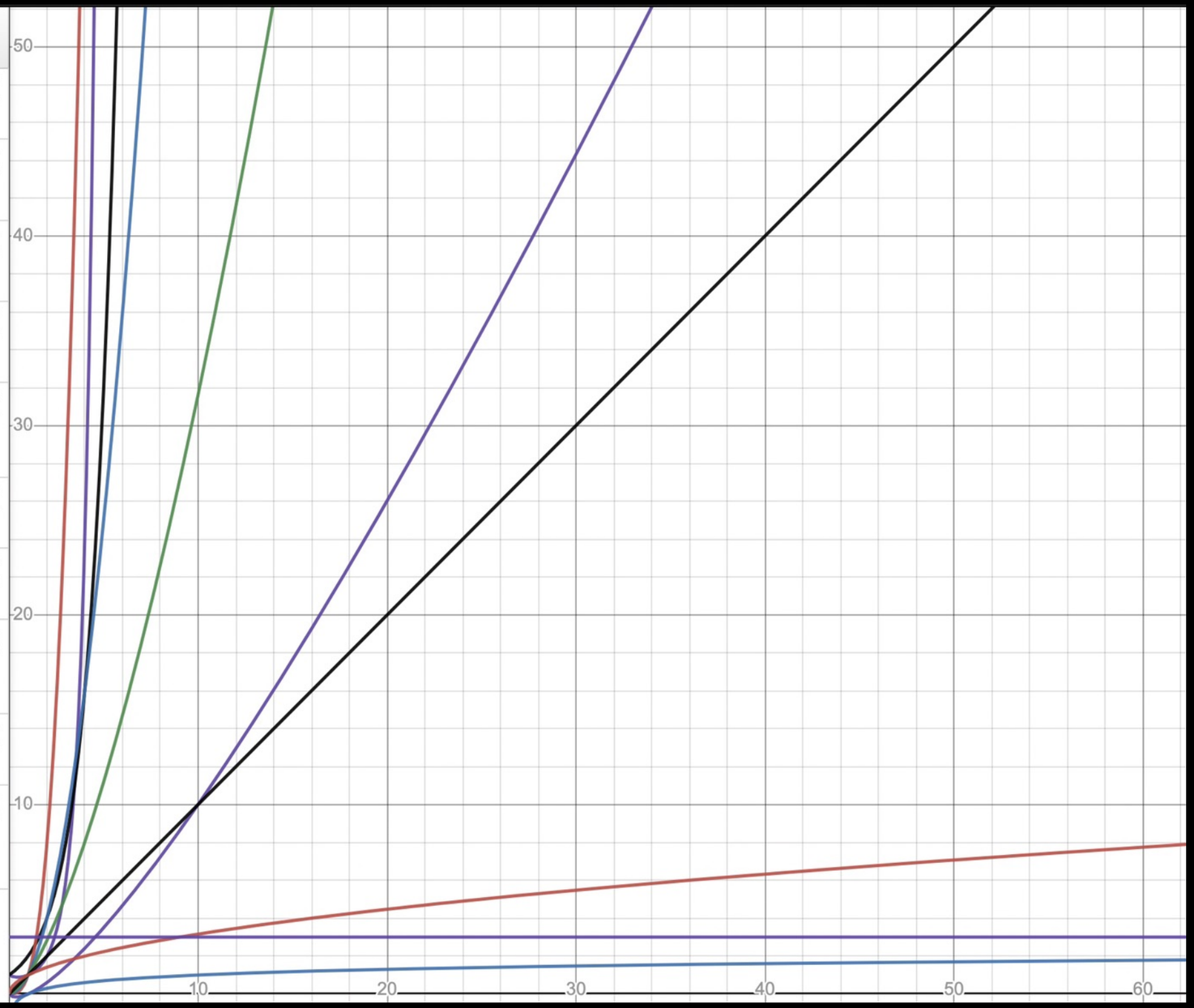
worst

1. $y = n!$
2. $y = 2^n$
3. $y = n^3$
4. $y = n^2$
5. $y = n\, n^{\frac{1}{2}}$
6. $y = n \log n$
7. $y = n$
8. $y = n^{\frac{1}{2}}$
9. $y = \log n$
10. $y = 3$

-10 10

11

powered by desmos



best

slow rate of growth

Algo 1

$y = 3x^2 + 8$

$x \to 10^6$

$y = 3 \times (10^6)^2 + 8$

↓

$3 \times 10^{12} + 8$

avoidable

↓

negligible

Algo 2

$y = 10x^5 + 3x^3 + 2x + 10$

$y = 10 \times (10^6)^5 + 3 \times (10^6)^3 + 2 \times 10^6 + 10$

const

↑

negligible

negligible

lower degree terms

lower degree ter

negligible

→ we avoid all constants and lower degree terms.

→ we only concern ourselves with the highest degree term.

algo 1

$y \approx x^2$

algo 2

$y \approx x^5$

```
1 → let x = 20;
2 → let y = 3000;
3 → let count = 3;
4 → for(let i = x; i ≤ y; i++) {
5       console.log(i*2);
6       count += i;
7   }
8
9   console.log("END");
```

→ O(n)

Let (y-x) be n

1 + 1 + 1 + 1 + 4n + 1

4n + 5 → f(n)

n → 10^7

$4 \times 10^7 + 5$ → negligible

→ approx → n

y = n → linear eqn

O(n)

	n=1	n=2	n=3	n=4	n=5	n=6
→ 4n+5	9	13	17	21	25	29
5n	5	10	15	20	25	30

$\forall\ n \geq 6$

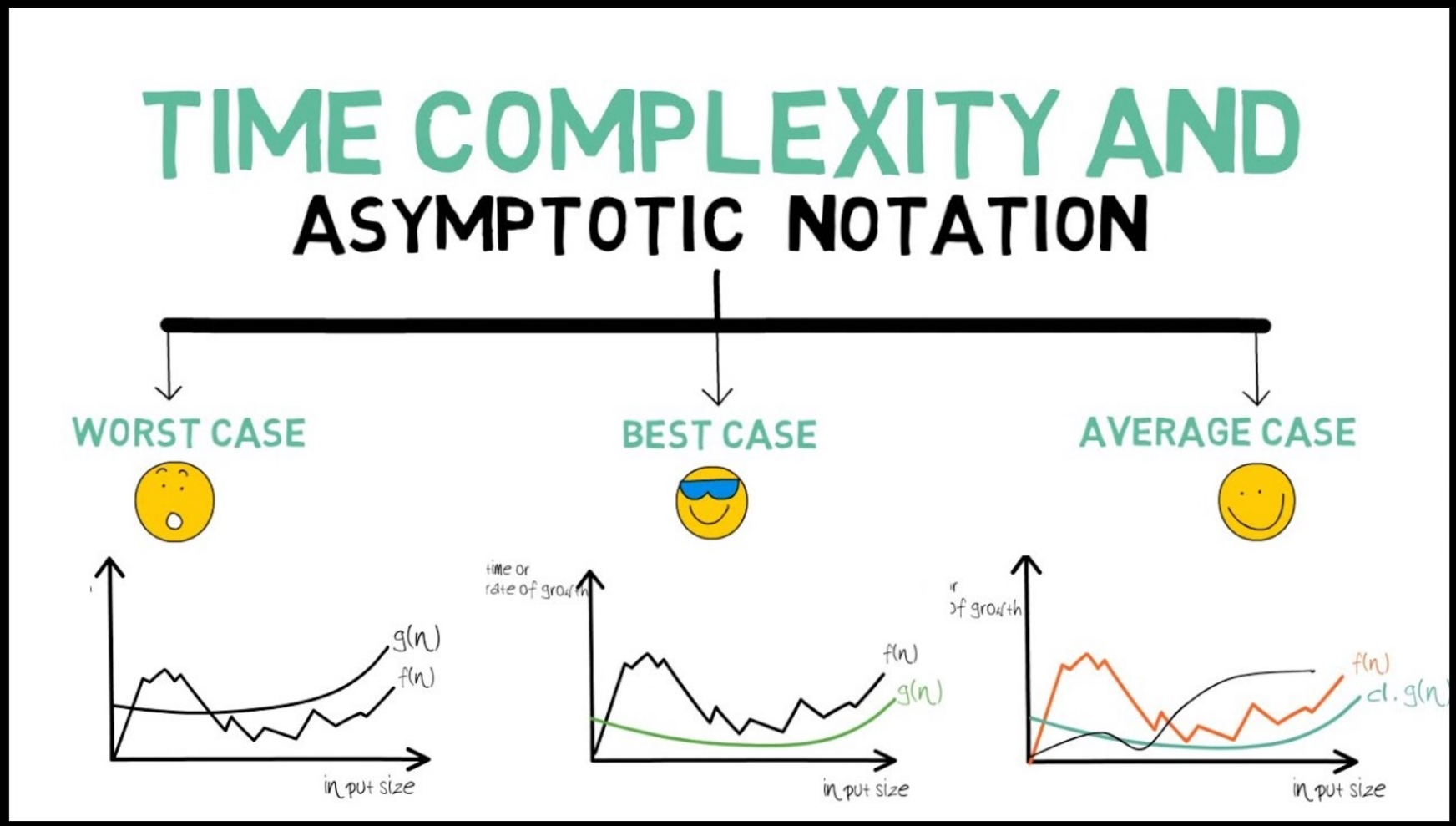
$f(n) \leq cg(n)$

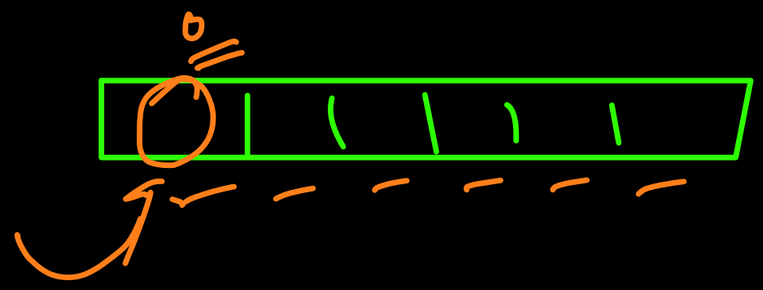
$4n + 5 \leq 5n$

c → 5

g(n) = n

Notations





For representing the rate of growth in terms of highest degree terms, we use these notations

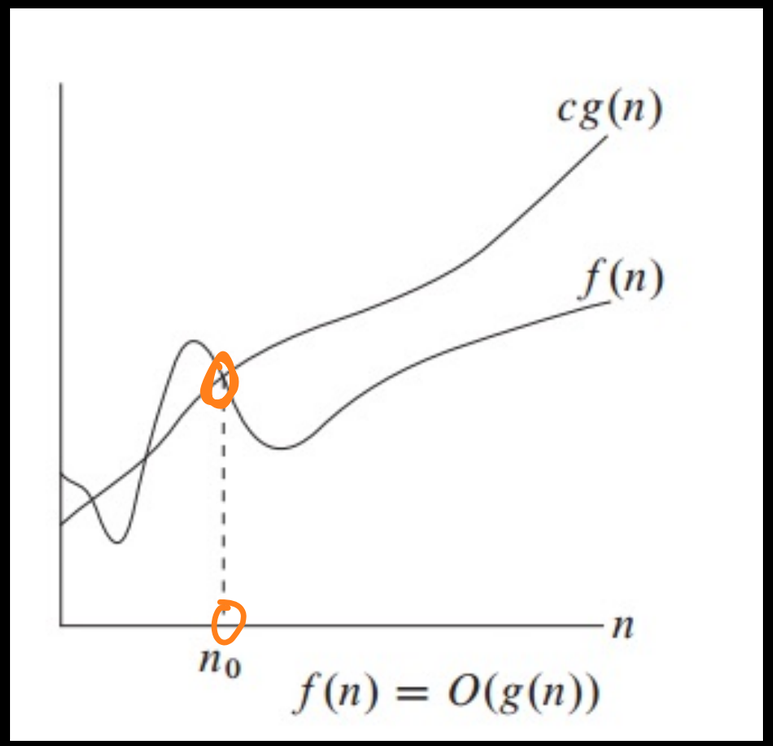
Best Case →	Ω → Big omega
Avg Case →	Θ → Big theta
Worst Case →	O → Big O.

Imp

```
for(i=0; i<n; i++)
{
   ...
}
```

→ $O(n)$

Worst ← Big O Notation



This notation gives Tight Upper Bound of the given function.

Ex ⇒ $f(n) = 2n^2 + 3$

n → 1 → f(1) → 5
n → 2 f(2) → 11
!
!

Big O of the func^n f(n) means, → $O(n^2)$

there is some function g(n) such that

$\forall\ n > n_0 \qquad 0 \le f(n) \le C \times g(n)$

C → constant

$0 \le f(n) \le 5n^2$

C = 3

		$n=1$	$n=10$	$n=10^4$
$2n^2+2$	$\longrightarrow$	5	203	200003
$3n^2$	$\longrightarrow$	3	300	300000

$C=3$

Ex → $f(n) = 3n+8$

$g(n) = n$

$C \rightarrow 4$

$3n+8$

$4n$

$n_0 = 8$

$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$
11	14	17	20	23	26	29	32
4	8	12	16	20	24	28	32

→

$0 \leq f(n) \leq cg(n) \quad \forall\, n > n_0$

$3n+8 \leq 4n$

$O(n)$

Ex → $f(n) = 2n^3 - 2n^2$

	$n=1$	$n=2$
$2n^3-2n^2$	0	$16-8=8$
n^3	1	8

$g(n) \to n^3$

$C \to 2$

$f(n) \leq cg(n)$ $\quad n \geq 1$

$O(n^3)$

```
for(i=0; i<n ; i++)
    for(j=0 ; j<n ; j++)
```

		j
i=0	→	n
i=1	→	n
i=2	→	n
⋮		⋮
i=n-1	→	n

n^2

algo

$f(n) \Rightarrow 3n^2 - 2\sqrt{n} + 6$

$f(n) \leq C\, n^2$

$O(n^2)$

algo 2

$f(n) = 10n^2 - 100n$

$f(n) \leq C_2 n^2$

```
function f0(n) {
    let ans = 1;
    for(let i = 0; i < n; i++) {
        console.log(i);
        ans += i;
    }
    return ans;
}
```

input

C + C + C + 4nC → 4nC + 3C

const.

i=0 → i=1 → i=2 ..... i=n-1

4 4 4 4

O(n)

total n iterations

what is the metric to measure time ??

No. of instructions executed -

Now we can assume every single instruction taken C unit of time

So if we have total K instruction, we will spend K*C unit of time.

```
function f1(n, m) {
    let ans = 0;
    for(let i = 0; i < n; i++) {
        ans += 1;
    }
    for(let j = 0; j < m; j++) {
        ans += 2;
    }
    console.log(ans);
}
```

input

3 instruction per iteration

→ 2nd for loop starts if the first one ends.

(assumptions → 1 instruction takes C unit time, C → const)

$C + C + C + 3nC + C + 3mC$

$i = 0 \to i = 1 \to i = 2 \quad \cdots\cdots \quad i = n-1$

$3 \quad 3 \quad 3 \quad 3$

$j = 0 \to j = 1 \to j = 2 \quad \cdots\cdots \quad j = m-1 \to$

$3 \quad 3 \quad 3 \quad 3$

Total no. of instruction

$3nC + 3mC + 4C$

$(n+m) \times 3C \longrightarrow O(n+m)$

```
function f2(n) {
    let ans = 9;
    for(let i=1;i<=Math.log(n);i++) {
        console.log(i);
    }
    return 0;
}
```

$C + C + 3 \log n\, C + C$

3 ins per iteration

$i = 1 \to i = 2 \to i = 3 \dots i = \log n \to$

3 3 3 3

$f_{inv} \to \underbrace{3 \log n\, C}_{\text{cost}} + \underbrace{3C}_{\text{lower degree}}$

$O(\log n)$

```
function f1(n, m) {
    let ans = 0;
    for(let i = 0; i < n; i++) {
        ans += 1;
    }
    for(let j = 0; j < m; j++) {
        ans += 2;
    }
    console.log(ans);
}
```

$O(n+m)$

$n \leq 10^6$

$m \leq 10^6$

$n \leq 10^6$

$m \leq 10^2$

$m << n$

$O(n + \cancel{m}) = O(n)$

10 wen dgree

```
function f3(n) {
    let ans = 0;
    for(let i = 0; i < n; i++) {
        for(let j = 0; j < n; j++) {
            ans += 1
        }
    }
    return ans;
}
```

input

$C + C + n \times n \times C + C$

$i=0 \to i=1 \to i=2 \cdots\cdots i=n-1 \to x$

$1 \quad n \quad n \quad n$

$1 \to$ comparison

$n \to$ for loop inst

$1 \to$ increment

time $\to n^2 C + 3C$ (const)

↳ 10 min

↳ $O(n^2)$

$n \to 1$	$\approx$	1
$n \to 2$	$\approx$	4
$n \to 3$	$\approx$	9
$n \to 4$	$\approx$	16

```
function f4(n) {
    let ans = 0;
    for(let i = 0; i < n; i++) {
        for(let j = 0; j < i; j++) {
            ans += 1
        }
    }
    return ans;
}
```

i=0 → i=1 → i=2 → i=3 → i=4 ..... i=n-1

↓ ↓ ↓ ↓ ↓ ↓

0 3 6 9 12 ..... 3x(n-1)

i=1 , j=~~0~~ → 1

3

i=2 → j=~~0~~ → ~~1~~ → 2

↳ 2x3 → ~~6~~

0 + 3 + 6 + 9 + 12 ..... 3(n-1)

3(1 + 2 + 3 + 4 ------ n-1)

3 × n x (n-1) / 2

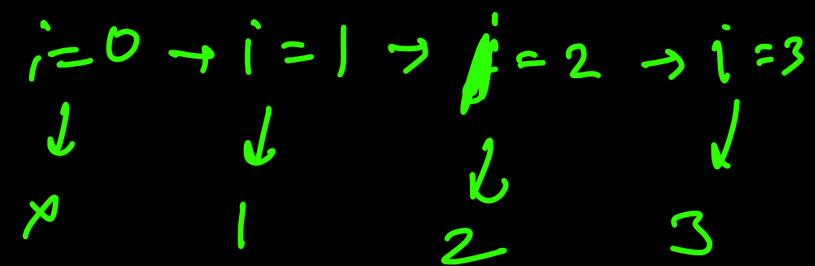
3 → const

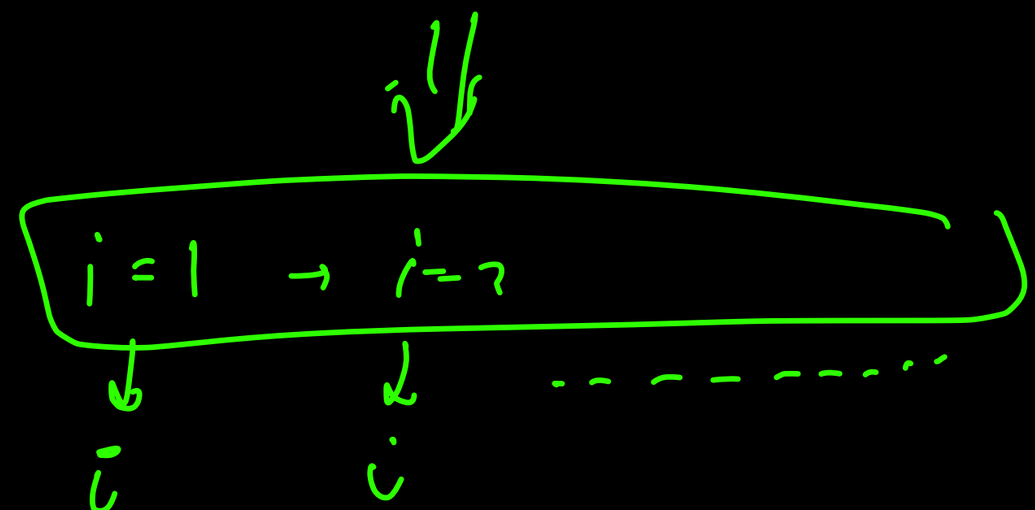
(n-1) → const

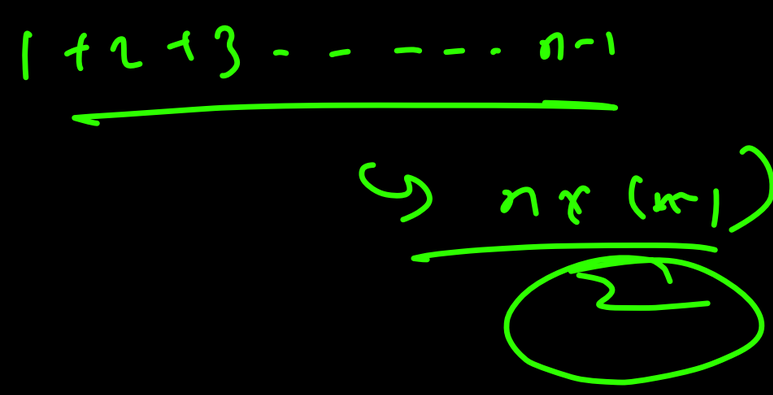
↳ n x (n-1) → $n^2 - n$ → lower degree

$O(n^2)$

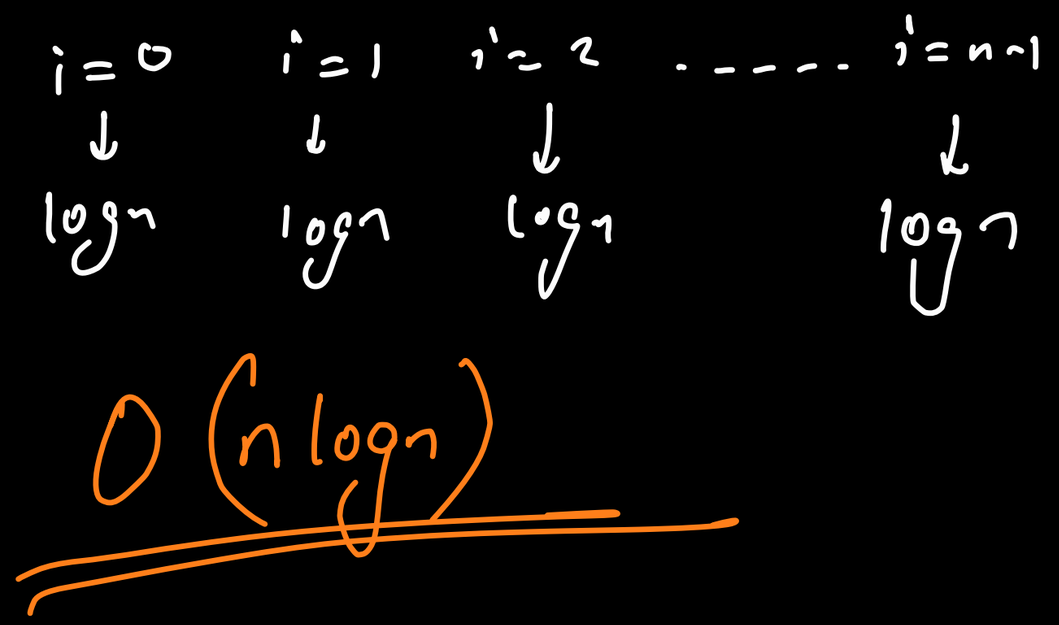
```
function f4(n) {
    let ans = 0;
    for(let i = 0; i < n; i++) {
        for(let j = 0; j < i; j++) {
            ans += 1
        }
    }
    return ans;
}
```



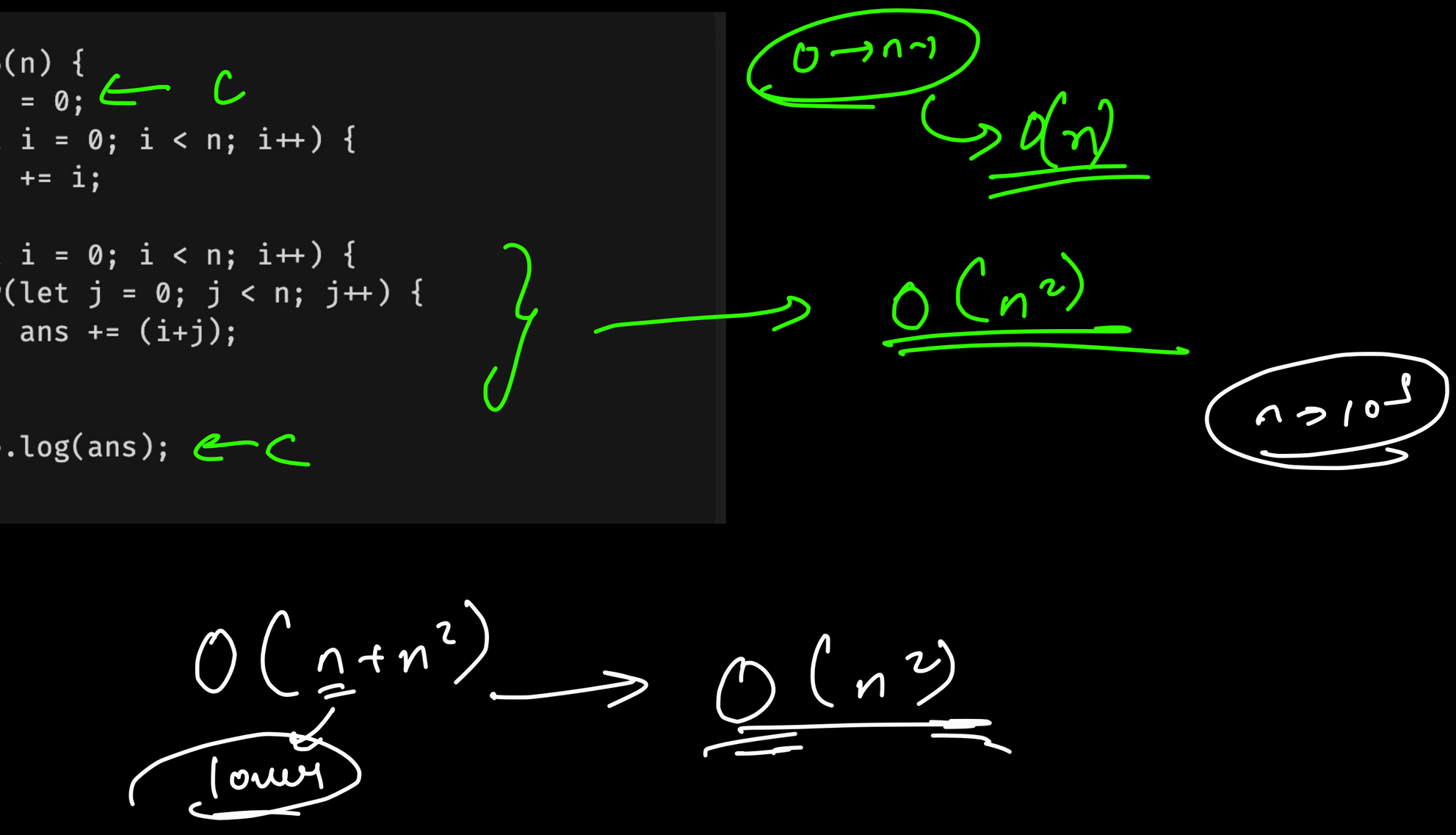




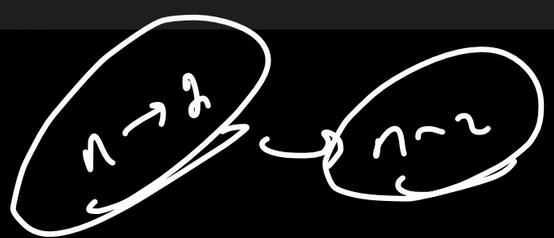
```
function f5(n) {
    let ans = 0;
    for(let i = 0; i < n; i++) {
        for(let j = 0;j<Math.log(n);j++) {
            ans += 1
        }
    }
    return ans;
}
```

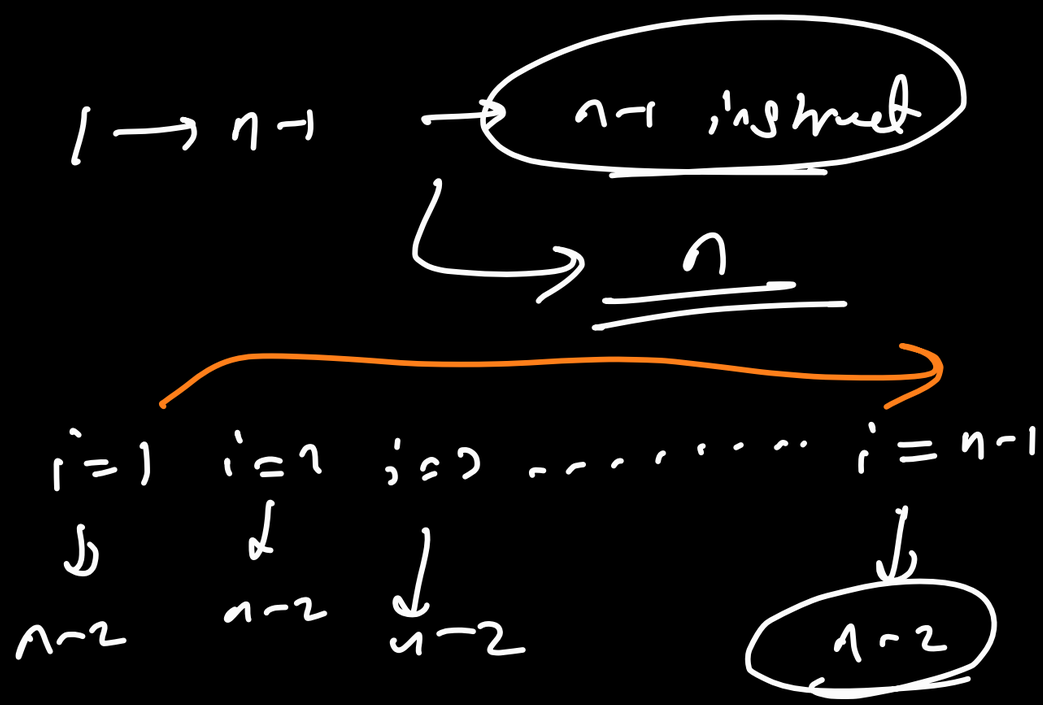


```
function f6(n) {
    let ans = 0;
    for(let i = 0; i < n; i++) {
        ans += i;
    }
    for(let i = 0; i < n; i++) {
        for(let j = 0; j < n; j++) {
            ans += (i+j);
        }
    }
    console.log(ans);
}
```



```
function f7(n) {
    let ans = 0;
    for(let i = 1; i < n; i++) {
        for(let j = n; j > 1; j--) {
            ans += i;
        }
    }
    return ans;
}
```





$(n-1) \times (n-2) \rightarrow n^2 - n - 2n + 3$

$O(n^2)$

```
function f8(n) {
    let ans = 0;
    for(let i = 1; i < n; i *= 2) {
        ans += i;
    }
    return ans;
}
```

i=1 → i=2 → i=4 → i=8 .....

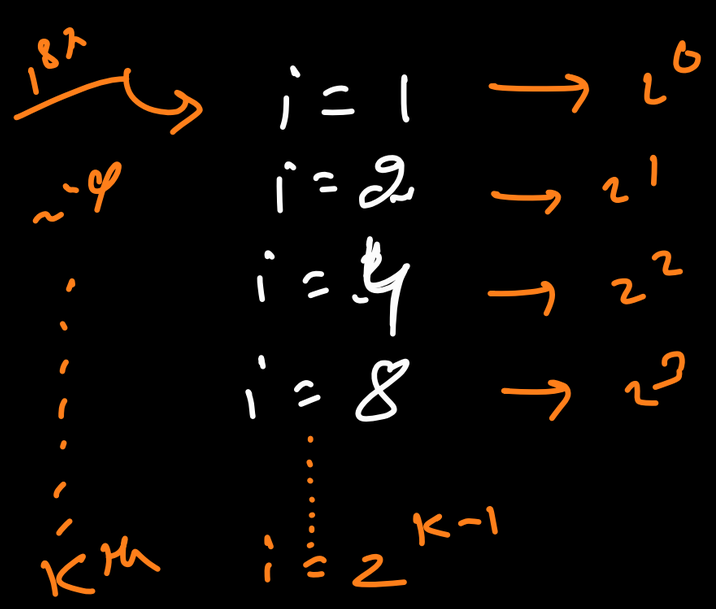
↓ ↓ ↓ ↓

3 3 3 3

O(log n)

Total iterations of i is K.

then our loop total 3K instructions



$2^{K-1} < n$

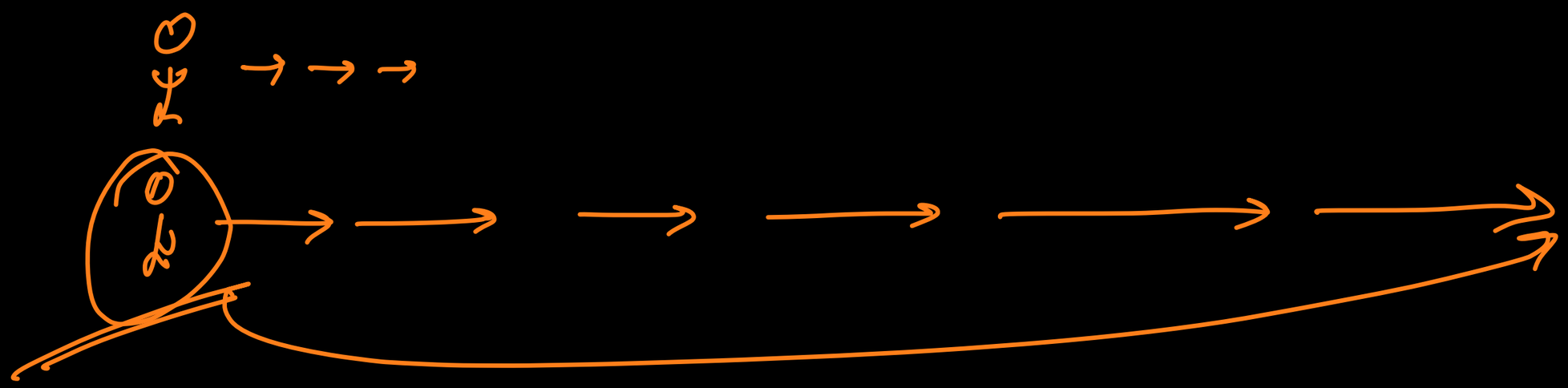
Taking log both sides

$\log_2 2^{K-1} < \log_2 n$

$(K-1)\log_2 2 < \log n \rightarrow (K-1) < \log n$

$K < \log_2 n + 1$

O(log n)



```
function f9(n) {
    let ans = 0;
    while(n > 0) {
        ans += n;
        n /= 2;
    }
    return ans;
}
```

→ C

→ C

→ O(log n)

In every iteration our execute 3 instructions

if there are (K) iteration

3K → total instructions

Total Run → 3KC → $3\log_2 n\, C$ → O(log n)

1st → $n \to \frac{n}{2} \to \frac{n}{4} \to \frac{n}{8}$ 1

2nd → $\frac{n}{2^0} \to \frac{n}{2^1} \to \frac{n}{2^2} \to \frac{n}{2^3}$ ..... $\frac{n}{2^{K-1}}$

3rd → - - - -

K → 1

$$\frac{n}{2^{k-1}} \approx 1$$

$$n \approx 2^{k-1}$$ taking log both sides

$$\log_2 n \approx \log_2 2^{k-1}$$

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

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

$$\boxed{k \approx \log_2 n + 1}$$

```
function f10(n) {
    for(let  i = n; i > 0; i/=2) {
        for(j = 0; j < i; j++) {
            console.log(i, j);
        }
    }
}
```

$i = n \longrightarrow$ j will go n iterations

$i = \frac{n}{2} \longrightarrow$ j will go $\frac{n}{2}$ iterations

$i = \frac{n}{4} \longrightarrow$ j will go $\frac{n}{4}$ iterations

$\vdots$

$i = \frac{n}{2^k} \longrightarrow$ j will go $\frac{n}{2^k}$ iterations

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} \cdots\cdots \frac{n}{2^k}$$

$$n\left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} \cdots\cdots \frac{1}{2^k}\right) \rightarrow \log n$$

$$n\left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} \cdots\cdots \frac{1}{2^{\log_2 n}}\right)$$

GP

Sum of a gp → $a\frac{(1-r^n)}{1-r}$

$a \rightarrow$ first term

$n \rightarrow$ total terms

$r \rightarrow$ ratio

$r \rightarrow 1/2 \quad a \rightarrow 1$

$n \rightarrow \log_2 n + 1$

$$\text{Sum} \rightarrow \frac{1 \times \left(1 - \left(\frac{1}{2}\right)^{\log_2 n + 1}\right)}{1 - \frac{1}{2}}$$

$$\rightarrow \frac{\left(1 - \left(\frac{1}{2}\right)^{\log_2 n + 1}\right)}{1/2} \rightarrow 2\left(1 - \frac{1}{2^{\log_2 n + 1}}\right)$$

$$\rightarrow \left(2 - \frac{2}{2^{\log_2 n + 1}}\right) \rightarrow 2 - \frac{2}{2^{\log_2 n} \times 2}$$

$$\rightarrow \left(2 - \frac{1}{n}\right) \rightarrow \text{negligible}$$

$$\simeq 2 \rightarrow \text{Const}$$

Time → $n \times c$

$O(n)$

2, 4, 8, ----

$\frac{1}{3}$ $\frac{1}{9}$ $\frac{1}{27}$ $\frac{1}{81}$ $\rightarrow$ $\frac{1}{2}$

```
function f11(n) {
    for(let j = 1; j ≤ n; j++) {
        for(let i = 0; i < n; i = i + j) {
            console.log(i, j);
        }
    }
}
```

→ O(n log n)

j = 1 i → [0, n-1] → inc → 1 → total iterations → n.

j = 2 i → [0, n-1] → inc → 2 → total iterations → n/2

j = 3 i → [0, n-1] → inc → 3 → total iterations → n/3

⋮

j = n i → [0, n-1] → inc → n → total iterations → 1

$$\left(n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \frac{n}{5} \cdots\cdots + 1\right)$$

$$n\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} \cdots\cdots \frac{1}{n}\right)$$

↳ H.P

$n \log n$

$$\frac{1}{a}, \frac{1}{a+d}, \frac{1}{a+2d}, \frac{1}{a+3d} \cdots\cdots \frac{1}{a+(n-)d}$$

$$S_n = \frac{1}{d} \log_e \left(\frac{2a + (2n-1)d}{2a - d}\right)$$

$$\Rightarrow \frac{1}{1} \times \log_e \left(\frac{2 + 2n - 1}{2 - 1}\right)$$

$$\Rightarrow \log_e \left(\frac{2n+1}{1}\right) \Rightarrow \approx \log n$$

```
function f12(n) {
    let ans = 0;
    for(let i = 2; i ≤ n; i *= i) {
        ans++;
    }
}
```

K → does it denote no. of iterations

Some final value of i is less than or equal to n

2 → 4 → 16 ..... 2^{2^K}

2 2 2 2

$2^1 \to 2^2 \to 2^4 \to 2^8$ ..... 2^K

1 → 2 → 3 → 4 ..... T → O(log(log n))

Total count

$2^K \leq n$

$K \leq \log_2 n$

1st → $1 \to 2^0$

2nd → $2 \to 2^1$

3rd → 2^2

4th → 2^3

⋮

$T^{th} \to 2^{T-1}$

$2^{T-1} = K$

$(T-1)\log_2 2 = \log_2 K$

$T = \log_2 K + 1$

$T = \log(\log n) + 1$

Bubble Sort → $\Omega(n)$

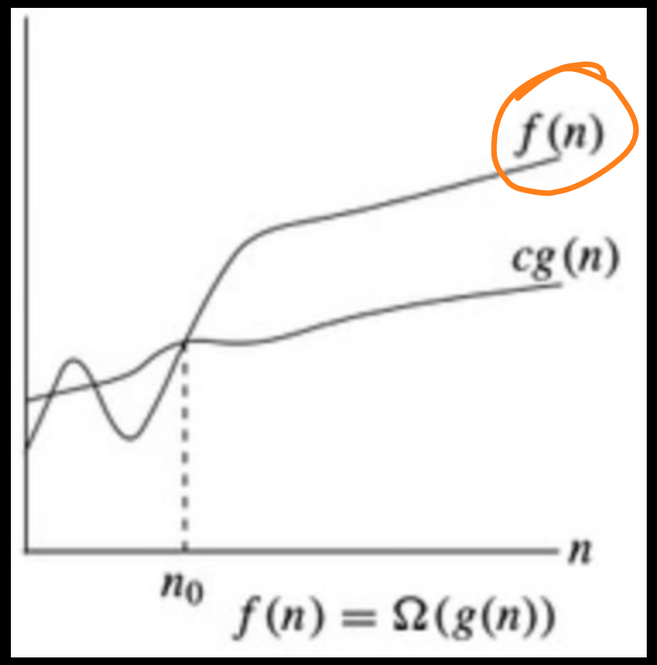
Selection Sort → $\Omega(n^2)$

$\Big] \rightarrow O(n^2)$

C

Best Case Analysis (Big Ω)

→



The big Ω notation gives the tight Lower Bound of the given algorithm.

$n > n_0 \quad \{0 \le cg(n) \le f(n)\} \rightarrow \Omega(g(n))$

Best Case of $f(n)$

Ex → $f(n) = 5n^2$

$c = 5$ | $g(n) = n^2$

$\Omega(n^2)$ $\boxed{n_0 = 1}$

Big Theta ~ Θ → Average Case