Asymptotic Analysis

```
function printNumber (n) {
    console.log(n); — (1) unit Time
}
```

O(1)

```
function printNumbers (N) {
    for (let i=1; i<=N; i++) {
        console.log(i); - N
    }
}
```

O(N) — n times

N - 5 → 5 times

N - 7 → 7 times

⋮

N - 1,00,000 → 1,00,000 times

TC of my program ≈ input size

TC → O(N)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

input size → n * n

→ m * n

[13 14 15] 4×4 → matrix

```
function printMatrix ( mat ){
    for ( let i = 0; i < mat.length; i++){
        for (let j=0; j<mat[0].length; j++) {
            console.log (mat[i][j]);
        }
    }
}
```

n*n

n times

n*n → times

n times

O(n*n) → input size

O(1), O(n), O(n*n) → O(n²)

↓ input size

Naive Product Except Self.

f(n) = n*(n-1)

= n² - n

$f(n) = O(n^2)$

Efficient product except self.

$f(n) = 1 + 1 + (n + n) + 1 + n + n + 1$

$= 4n + 4$ $\quad n = n^1 \Rightarrow n$

$f(n) = O(n)$

$cn + x.$

$O(n)$

---

for (i = 1 to n) { → n+1

for (j = 1 to n) { → n*(n+1)

```
for (i = 1 to n) {
    ---
    ---
    ---
}
}
}
```

```
n * n * (n+1)
```

$\Rightarrow n^3 + n^2$

$f(n) = O(n^3)$

$f(n) = (n+1) + n*(n+1) + n*n*(n+1)$

$= n + 1 + n^2 + n + n^3 + n^2$

$= n^3 + 2n^2 + 2n + 1$

$f(n) = O(n^3)$

1	2	3	4

```
for (i=0; i<4; i++) {
```

$0 < 4 \rightarrow 1$ ④

$1 < 4 \rightarrow 2$

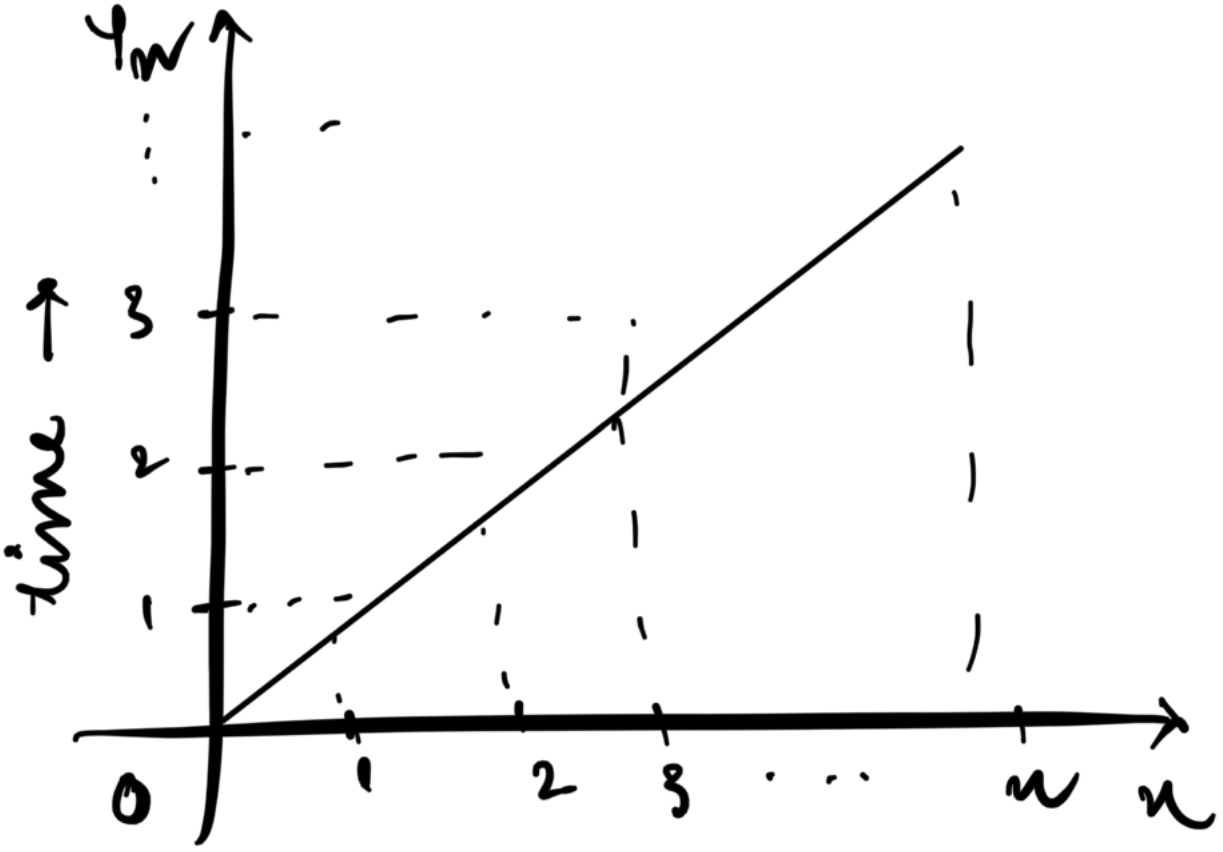
$2 < 4 \rightarrow 3$

$3 < 4 \rightarrow 4$

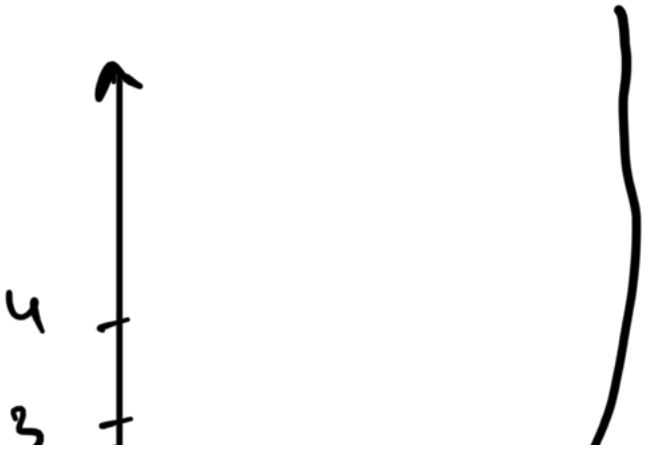
$4 < 4 \rightarrow 5$ $(n+1)$

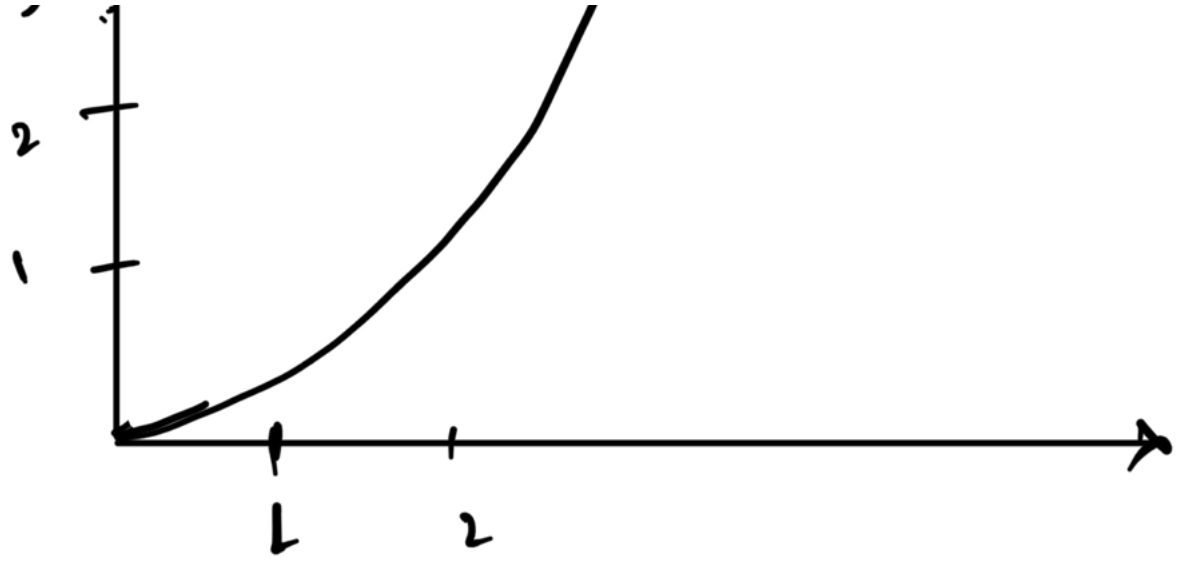
↑ n-times

```
for (i=1 to n) {        → n+1 times
    for (j=1 to n) {
        - - -
        - - -
    }                   n*(n+1)
```



input → $O(n)$.





$O(n+m) \rightarrow O(m)$ or $O(n)$

or $O(n+m)$

relationship between m & n.

$n = O(m^2)$

$O(n^2+n) \rightarrow O(n)$

$m = 3 \quad n = 9$

$m = 100, \quad n = 10000$

$n \geq O(m^2)$

$O(m) \times$

$O(m + m^2) \approx O(m^2)$

$O(m+n)$

Time complexities in increasing order

$1, \log N, \sqrt{N}, N, N \log\log N, N \log N, N\sqrt{N}, N^2, N^3, 2^N, \ldots$

$N \log N$, [illegible]

$n!$, n^n

$f(n) = n * f(n-1)$

$f(n-1) = (n-1) * f(n-2)$

$\vdots$

$f(2) = 2 * f(1)$

$f(n) * f(n-1) * \cdots f(1) =$

$n * f(n-1) * (n-1) * f(n-2)$

$\cdots * 2 * f(1)$;

$\Rightarrow f(n) = n * (n-1) * \ldots * 2 * 1$

$\therefore f(n) = n!$

$f(n) = \underbrace{n * f(n-1)}_{①}$ $\Gamma(n) = \Gamma(n+1)$

$f(n-1) = (n-1) * f(n-2)$

$f(n-3) = (n-2) * f(n-3)$

$f(n) * f(n-1) * f(n-2) * f(n-3) \ldots 1$

$1 = n * f(n-1) * (n-1) * f(n-2) * (n-2) * f(n-3) \ldots$

L

$f(n) = \begin{cases} n * (n-1) * (n-2) \cdots \\ \quad L \end{cases}$

$f(n) = n!$ $\quad f(n) = O(n!)$

$n(n-1)*(n-2)\cdots$

$n^n - n^{n-1} - n^{n-2} - \cdots$

$n^n > n!$

$n * f(n-2)$

```
fact (n) {                                    -(1)
    if (n == 0 || n == 1) {
        return n;
    }
    return n * fact (n-1)
}
```

(1) $T(n-1)$

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

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

$$\vdots$$

$$T(2) = 2 + T(1)$$

(-1) (-1)

$$T(n) = n * T(0) + 1$$

$\rightarrow O(n)$

$n! = O(n)$

TYPES OF NOTATIONS.

1. Big O notation → upper bound (O)
2. omega notation → lower bound (Ω)
3. Theta notation → average bound (θ)

O, Ω, θ. →

↙ upper ↓ lower → average.

Big O notation :-

The function $f(n) = O(g(n))$

if and only if for some

positive constant c and no.

$f(n) \leq c * g(n) \quad \forall \; n \geq n_0$

$n \geq n_0$

$f(n) = O(g(n))$

n_0 ....

```
function addNumbers (n) {
    let sum = 0;          — (1).
    for (let i = 1; i <= n; i++) {      → n+1
        sum += i;         — (n)
    }
    return sum;           → 1
}.
```

$f(n) = 1 + (n+1) + n + 1$

$= 2n + 3$

$f(n) \leq c * g(n)$

$f(n) = O(g(n))$

$2n+3 \leq c * g(n)$

$n_0 \geq 1$

$2n+3 \leq 10 * n$

$5 \leq 10$

$7 \leq 20$

$\vdots$

$2n+3 \leq 7 * n$ $\quad n=1$

$2+3 \leq 7$

$7 \leq 4$ $\quad n=2$

$\vdots$

$n > 0$

$2n+3 \leq 2n+3n$

$3n \geq 3$

$\Rightarrow 2n+3 \leq 5n$

$f(n) \leq c * g(n) \Rightarrow g(n) = n$

$f(n) = O(n)$

$2n+3 \leq 2n^2+3n^2$

$2n+3 \leq 5n^2$

$f(n) = O(n^2)$

upper bound

1, $\log n$, $\sqrt{n}$, n, $n\log n$, n^2 ..... n^n

$f(n) = n^3 + 2n^2 + 2n + 1$

$f(n) \leq c * g(n)$

$n \to \infty$

$n^3 + 2n^2 + 2n + 1 \leq n^3 + 2n^3 + 2n^3 + n^3$

$\Rightarrow n^3 + 2n^2 + 2n + 1 \leq 6n^3$

$\Rightarrow f(n) \leq c * g(n)$

$$f(n) = O(n^3)$$

$n^3 + 2n^2 + 2n + 1 \leq n^3 + 2n^3 + 2n^3 + n^3$

$n^3 + 2n^2 + 2n + 1 \leq 6n^3$

$g(n)$

$$f(n) = O(n^3)$$

Ω notation → lower bound

$f(n) \geq c * g(n)$ where $c > 0$ & $n > n_0$

$f(n) = 2n+3$

$2n+3 \geq 1 * n$

$f(n) \geq c * g(n)$

$f(n) = \Omega(n)$

$f(n) = n^3 + 2n^2 + 2n + 1 \geq 1 * n^3$ $n > 0$

$\Rightarrow f(n) \geq c * g(n)$

$f(n) = \Omega(n^3)$

Theta notation

$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

$O(g(n))$

$O(g(n))$

$f(n) = \Omega(g(n))$

$f(n) = O(g(n))$

$f(n) = \Theta(g(n))$

$c_1 * g(n) \le 2n+3 \le c_2 * g(n)$

$1*n \le 2n+3 \le 2n+3n$

$f(n) = \Theta(n)$

$c_1 * g(n) \le f(n) \le c_2 * g(n)$

$f(n) = n!$

$n*(n-1)*(n-2)\dots 1$

$f(n) \le c * g(n)$

$n*(n-1)*(n-2)\dots 1 \le n*n*n*\dots n$

f(n) ≤ n^n

f(n) = O(n^n)

Lower Bound

1 * 2 * 3 * ... * n ≥ n * (n-1) * (n-2) * ... 1

① ② ≥ f(n)

f(n) = Ω(1)

Ω(1) → O(n^n) O(n!) O(n^n)

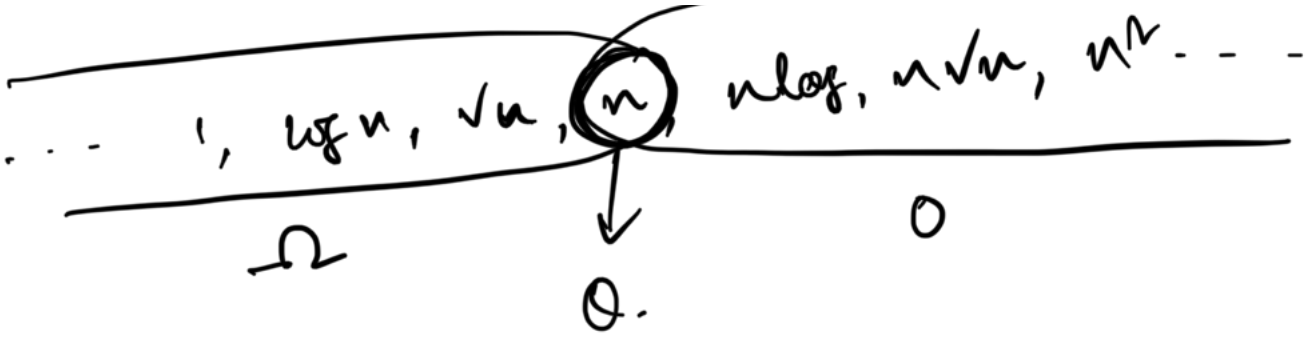
log n! ≥ log (n * n * n * n)

log (n * (n-1) * ...)

log (1 * 2 * 3 * ... n) ≥ f(n) ≥ n log n

Ω(1) ≥ f(n) ≥ O(n log n) → g(n)

f(n) ≠ Θ(g(n))



---

function calculateTime () {

let p = 0;

for (let i = 1; p < n; i++) {

p = p + i;

}

}

i	p
1	0+1
2	0+1+2
3	0+1+2+3
⋮	
K	0+1+2+3+...+K

when i = K.

p = n

1+2+3+....+K = n

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

$\Rightarrow K^2 \simeq n$

$\Rightarrow K \simeq \sqrt{n}$

$K = O(\sqrt{n})$

→ AP

$\frac{n}{2}\{2a + (n-1)*d\}$

$\Rightarrow \frac{K}{2}\{2*1 + (K-1)\}$

$\Rightarrow \frac{K(K+1)}{2}$

```
int sum=20
for (i=1; i<n; i=(i*2)){
        sum+=i
}
```

1, 2, 4, ... when the loop has run K times $i == n$?

what is the value of i in terms of n.

$$\frac{1, 2, 4, 8, 16, 32 \cdots \quad (n) =}{1, 2, 2^2, 2^3, 2^4, 2^5 \cdots \quad (2^k)}$$

$(((((i*2)*2)*2) \cdots *2) = (2^k)$

condition breaks when

$i = n$

$2^k = n$

$\log_2 2^k = \log_2 n$

$\Rightarrow \log_2 2^k = \log_2 n$

$\Rightarrow k * \log_2 2 = \log_2 n$

$\Rightarrow k * 1 = \log_2 n$

$\Rightarrow k = \log_2 n$

$\Rightarrow k = O(\log_2 n)$

$n = k$

$n = 2^k$

$\log_2 2^k$

$\Rightarrow k$

$\Rightarrow n$