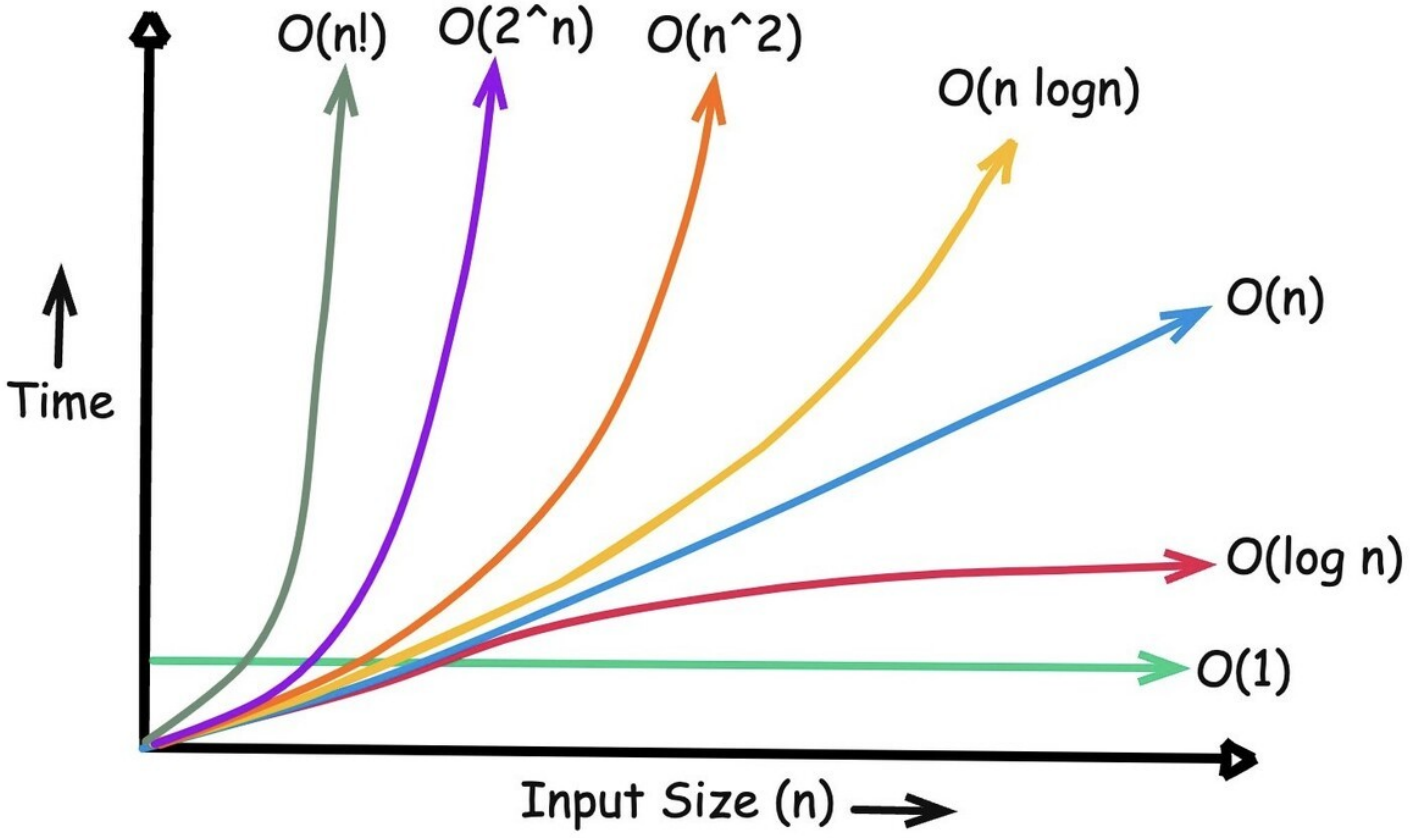
Complejidad Algorítmica: Big O Notation



1. ¿Qué es Big O?

Definición Practica:

Big O describe **como crece el tiempo de ejecución** cuando **aumenta el tamaño de entrada.**

```
No te dice: "Este algoritmo tarda 5 segundos"

Sí te dice: "Si duplicas los datos, el tiempo se multiplica por
4"
```

¿Por qué es importante?

Imagina que tienes dos algoritmos para ordenar una lista:

- Algoritmo A: Tarda 1 segundo con 100 elementos
- Algoritmo B: Tarda 2 segundos con 100 elementos

¿Cuál es mejor? No lo sabemos aún.
Si duplicamos los datos a 200 elementos:

- Algoritmo A: Tarda 10 segundos (10x más lento)
- Algoritmo B: Tarda 4 segundo (2x más lento)

Ahora Algoritmo B es mejor, aunque empezó más lento. Esto es lo que Big O predice.

Ejemplo Intuitivo: Buscar en una Lista

```python
# ¿Cuánto tarda buscar un nombre en una lista?

lista = ["Alice", "Bob", "Carol", ... , "Zoe"]  # n nombres

def buscar_nombre(lista, objetivo):
    for nombre in lista:
        if nombre == objetivo:
            return True
    return False
```

¿Cuántas comparaciones necesita?
R: Depende de DONDE está el nombre:

Escenario 1: "Alice" (primero)
→ 1 comparación

Escenario 2: "Carol" (posición 3)
→ 3 comparaciones

Escenario 3: "Zoe" (último)
→ n comparaciones

Escenario 4: "Xander" (no existe)
→ n comparaciones (revisa toda la lista)

EN PROMEDIO: n/2 comparaciones
Pero Big O ignora constantes → O(n)