¿Qué es Big O?

Big O es una forma de describir qué tan rápido crece el tiempo que tarda un algoritmo cuando la cantidad de datos aumenta.

No te dice cuántos segundos tarda, sino cómo se comporta cuando los datos se hacen más grandes.

Es como decir:

“Si duplico los datos, ¿qué tanto empeora el algoritmo?”

¿Por qué existe Big O?

Sirve para comparar algoritmos sin importar:

- El lenguaje
- La computadora
- Los detalles de la implementación

solo nos importa la idea general de su velocidad a gran escala.

O(1) — Tiempo constante

Da igual cuántos datos haya, siempre tarda lo mismo.

Ejemplo: acceder a un elemento en un arreglo.

```
function obtenerPrimero(lista) {
    return lista[0];
  }
```

No importa si la lista tiene 10 o 10,000 elementos. Siempre hace una sola operación.

O(n) — Tiempo lineal

El tiempo crece según la cantidad de datos.

Si se duplica los datos, el tiempo también.

Ejemplo: recorrer un arreglo con un solo ciclo.

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

Si la lista tiene 100 elementos, hará 100 pasos.

Si tiene 10,000, hará 10,000 pasos.

O(n²) — Tiempo cuadrático

Sucede cuando tienes un ciclo dentro de otro.

El tiempo crece muy rápido.

Ejemplo: comparar cada elemento con todos los demás.

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

Si hay 10 elementos, hace 100 comparaciones.

Si hay 100, hace 10,000.

O(log n) — Logarítmico

Muy eficiente.

Cada paso reduce mucho el problema (por ejemplo, a la mitad).

Ejemplo claro: búsqueda binaria.

```
function busquedaBinaria(lista, objetivo) {
    let inicio = 0;
    let fin = lista.length - 1;

    while (inicio <= fin) {
      let medio = Math.floor((inicio + fin) / 2);

      if (lista[medio] === objetivo) return medio;
      if (lista[medio] < objetivo) inicio = medio + 1;
      else fin = medio - 1;
    }

    return -1;
  }
```

Aunque tengas millones de datos, solo hace muy pocos pasos porque cada vez corta la lista a la mitad.

O(n log n) — Linealítmico

Es más rápido que $O(n^2)$ pero más lento que $O(n)$.

Aparece en algoritmos de ordenamiento buenos.

Ejemplo simple: este no es un código real de un algoritmo, pero así se ve la idea:

```
function ordenamientoBueno(lista) {
    // por dentro usa lógica de "divide y vencerás"
    return lista.sort();
  }
```

El .sort() de JavaScript usa un algoritmo con esa complejidad.

Resumen rápido

O(1): siempre tarda lo mismo.

O(n): tarda más si hay más datos.

$O(n^2)$: crece muy rápido; dos ciclos anidados.

O(log n): muy eficiente; divide en partes.

O(n log n): mezcla de recorrer y dividir; aparece en algoritamientos de ordenamiento.