1. Notación para el "orden de".

La notación Big O, escrita como $O(\mathrm{f}(n))$ O(f(n)), describe el comportamiento del tiempo de ejecución o uso de memoria de un algoritmo cuando el tamaño de entrada n crece mucho (tendiendo a infinito). Es una forma de clasificar algoritmos por su eficiencia.

1.1 ¿Para qué sirve?

- Comparar algoritmos de forma independiente al hardware o implementación.
- Saber qué tan bien escalará un algoritmo cuando aumente el tamaño del problema.
- Identificar cuellos de botella en el rendimiento.

1.1.2 Clasificación común de órdenes

Orden	Nombre	Ejemplo/Comentario
O(1)	Constante	Acceso directo a un array
O(log n)	Logarítmica	Búsqueda binaria
O(n)	Lineal	Recorrido de lista
O(n \log n)	Lineal-logarítmica	Merge sort, heapsort
O(n^2)	Cuadrática	Burbujas, doble bucle anidado
O(n^3)	Cúbica	Multiplicación de matrices
O(2^n)	Exponencial	Problemas de decisión (backtracking, etc.)
O(n!)	Factorial	Permutaciones, problemas NP-completos

1.1.3 Ejemplo

```
for (int i = 0; i < n; i++) {
   System.out.println(i);
}
```

Este bucle imprime del 0 a n−1. Se ejecuta n veces, por lo que su orden es: O(n)

1.1.4 Importancia

Cuando el tamaño de los datos crece, una diferencia de orden es más importante que cualquier optimización menor.

Ejemplo:

- Un algoritmo O(n log n) será más eficiente que uno O(n^2) cuando n sea muy grande, sin importar que el primero tenga más líneas de código o use más memoria.