Introducción al Análisis de Algoritmos

En esta presentación, exploraremos conceptos esenciales del análisis de algoritmos, centrándonos en el análisis empírico como herramienta para evaluar la eficiencia de diferentes soluciones.

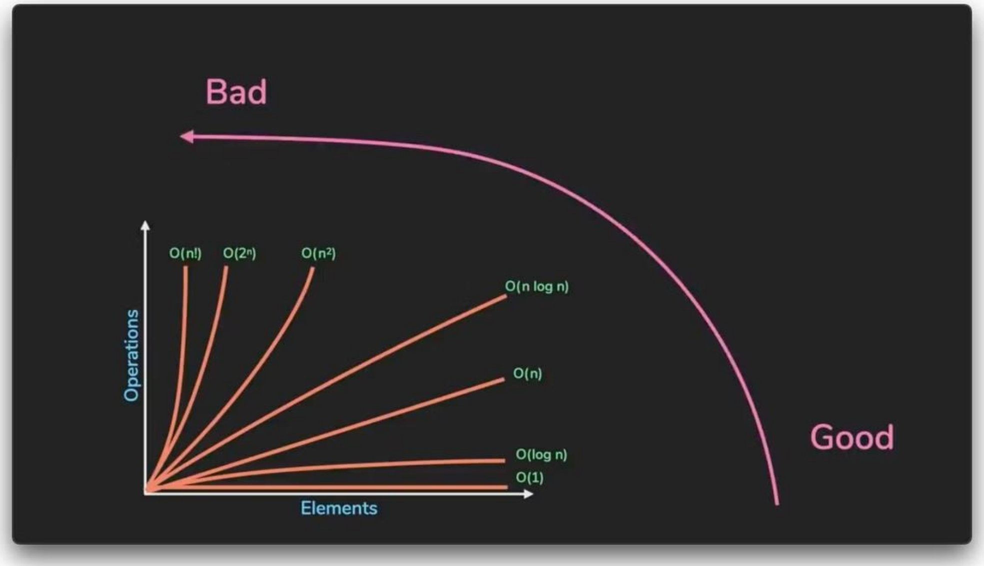
por Ariel Enferrel

¿Qué es un algoritmo?

Un algoritmo es un conjunto de instrucciones precisas y ordenadas que resuelven un problema específico.

Se caracteriza por ser correcto, robusto y eficiente, es decir, debe producir el resultado deseado, funcionar correctamente en diferentes condiciones y usar recursos de forma razonable.

Los algoritmos son la base de la programación y se encuentran en todas partes, desde la búsqueda en internet hasta la gestión de datos financieros.

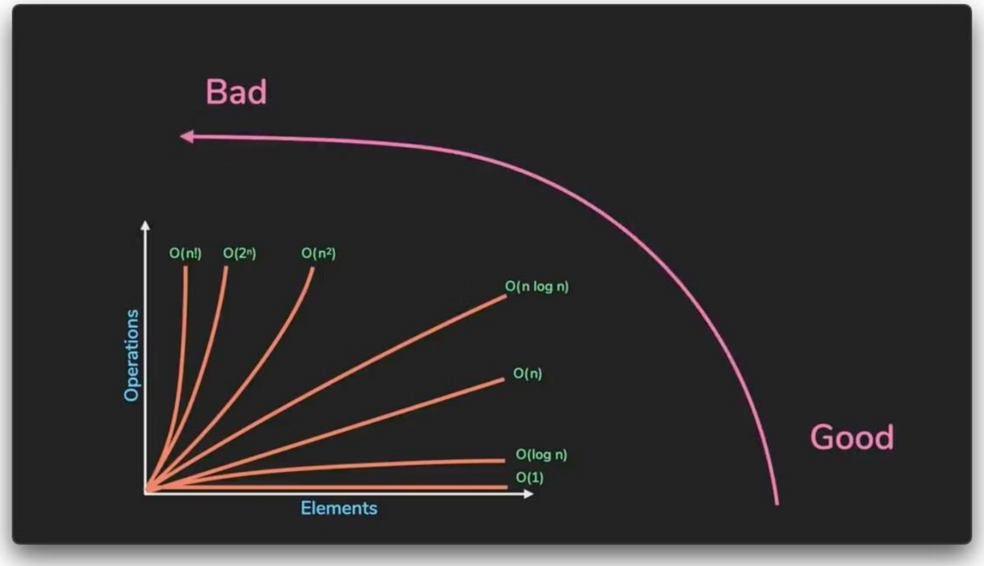


¿Por qué analizar algoritmos?

1. Analizar algoritmos nos permite comparar la eficiencia de diferentes soluciones para un mismo problema.

2. La optimización del uso de memoria y el tiempo de ejecución es fundamental para el rendimiento de las aplicaciones, especialmente en sistemas de gran escala.

3. Un análisis adecuado nos ayuda a elegir el algoritmo más adecuado para cada situación, garantizando la eficiencia y escalabilidad de nuestras soluciones.



Tipos de Análisis

1 Análisis Empírico

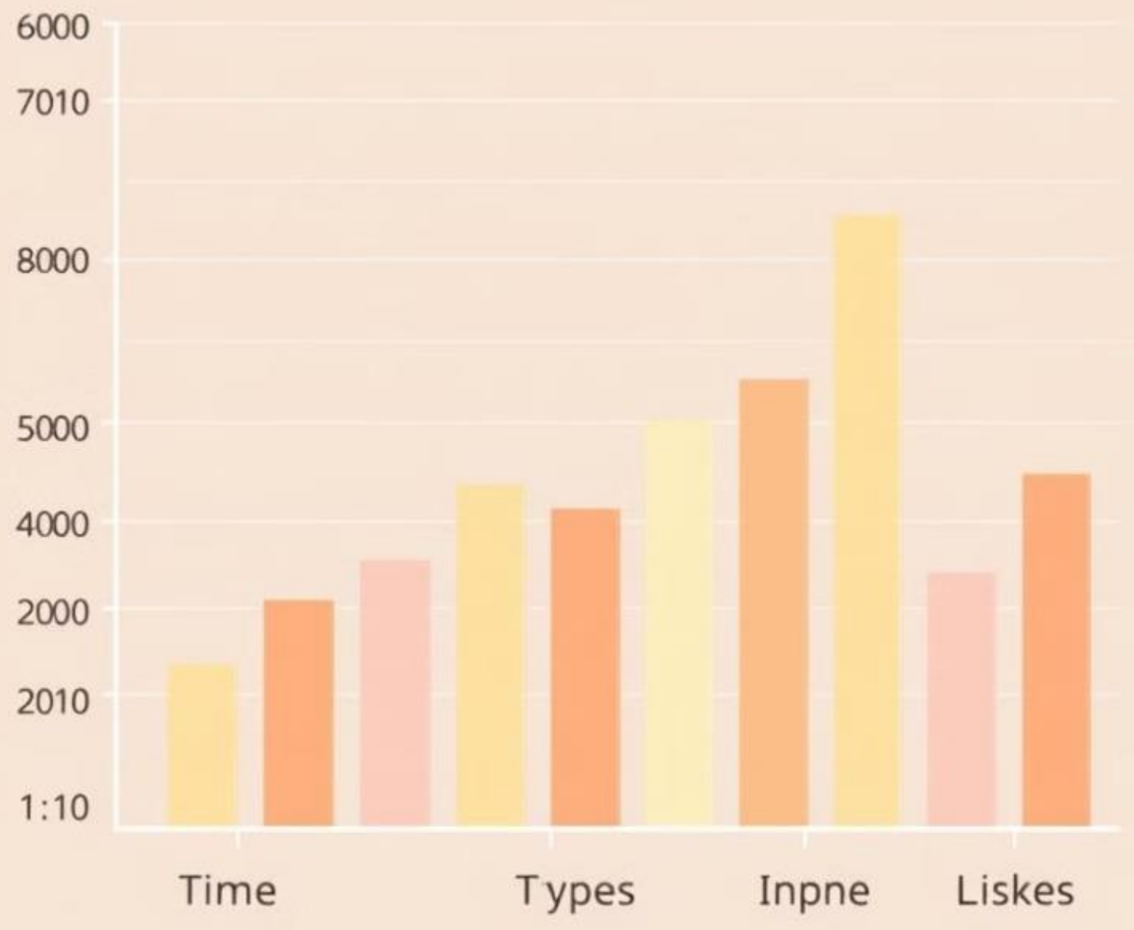
2 Análisis Teórico

Análisis Empírico

El análisis empírico consiste en medir el tiempo de ejecución de un algoritmo en la práctica.

Se implementa el algoritmo en código y se ejecutan pruebas con diferentes tamaños de entrada, registrando el tiempo que toma completar cada ejecución.

Los datos recopilados se representan en gráficos que muestran la relación entre el tiempo de ejecución y el tamaño de la entrada, permitiendo visualizar tendencias y patrones.



Ejemplo: Suma de los primeros n números (bucle)

```
def sumN(n):
    resultado = 0
    for i in range(1,n+1):
        resultado+=i
    return resultado
```

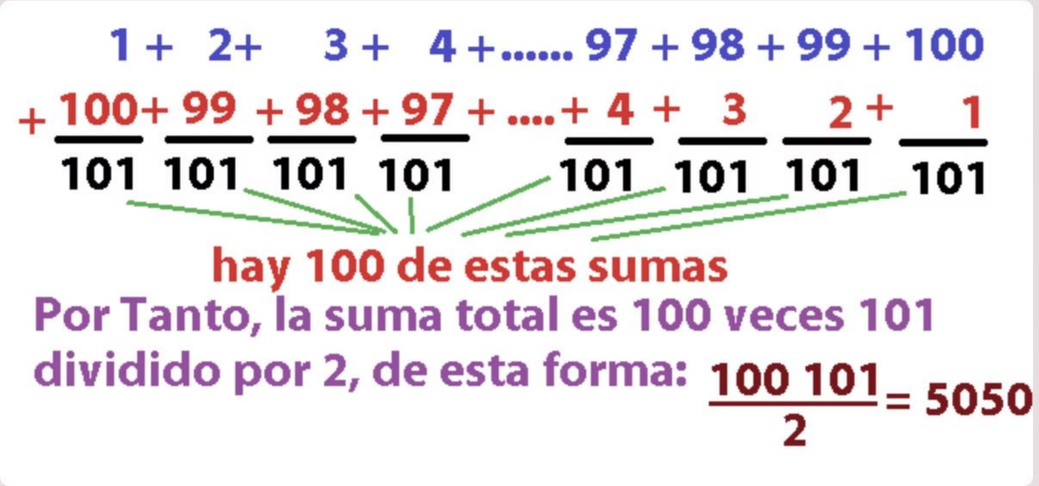
Este código en Python utiliza un bucle `for` para sumar los primeros `n` números naturales. Se inicializa `resultado` a 0 y se recorre un rango de 1 hasta `n+1`. En cada iteración, se suma el valor actual de `i` a `resultado`.

La formula de Gaus

La fórmula de Gauss proporciona un método mucho más eficiente para calcular la suma de los primeros `n` números naturales.

Suma de Gauss:

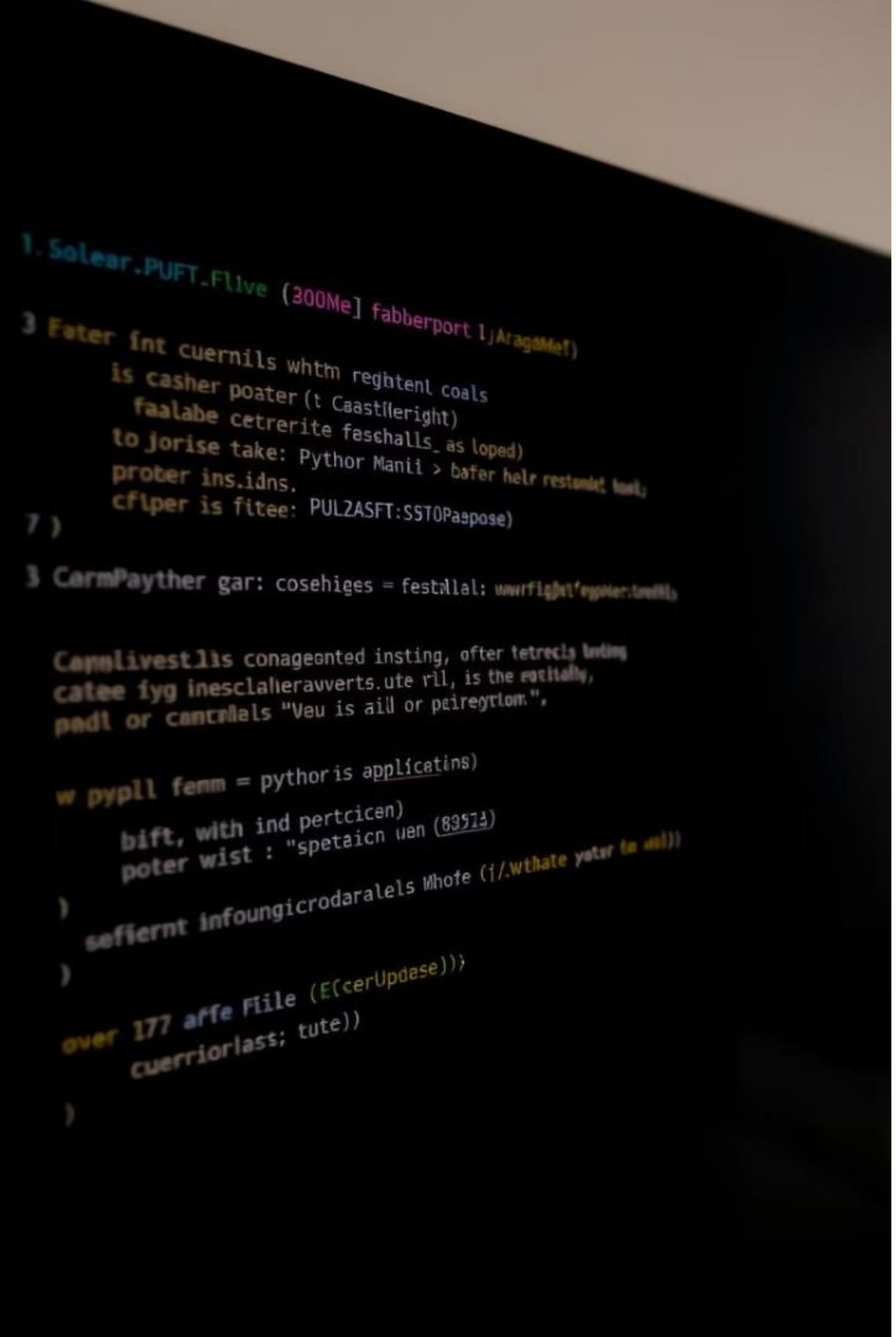
$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$



Razonamiento de Gaus

Si sumamos los números del 1 al 100, podemos agruparlos en pares: el 1 con el 100, el 2 con el 99, el 3 con el 98, y así sucesivamente. En cada caso, la suma es n+1, es decir, 101.

Como hay n/2 pares (en este caso, 50), el resultado total es n×(n+1)/2. Esta es la fórmula de Gauss, que nos permite calcular rápidamente la suma de los primeros n números naturales.



Ejemplo: Suma con fórmula de Gauss

```
def sumGausN(n):
    return n * (n + 1) / 2
```

El código en Python para la suma de Gauss es mucho más conciso y se traduce a una operación matemática sencilla.

Comparación de Resultados

1. Al ejecutar ambos algoritmos con diferentes tamaños de entrada, observamos que la solución de Gauss es significativamente más eficiente.

2. El tiempo de ejecución del bucle crece linealmente con `n`, mientras que la fórmula de Gauss se ejecuta en tiempo constante, sin importar el tamaño de la entrada.

Ventajas y Desventajas del Análisis Empírico

Ventajas:

- Permite obtener gráficas del tiempo de ejecución
- Comparación visual de algoritmos

Desventajas:

- Requiere implementación
- Depende del hardware y software donde se ejecuta
- No evalúa todas las posibles entradas