

Análisis de algoritmos: eficiencia y optimización.

Alumnos: Nidia Samaniego - Ignacio Roveres

Materia: programación I

Profesor: Nicolás Quirós

Fecha de Entrega: 09 de junio de 2025

Introducción:

El análisis de algoritmos nos ayuda a saber qué tan rápido y eficiente es un programa para resolver un problema. Cuando escribimos código en Python, queremos que funcione bien, pero también que sea rápido y no use mucho espacio en la computadora.

Un algoritmo es una serie de pasos para hacer una tarea. Analizar un algoritmo es ver cuánto tiempo tarda en hacer su trabajo y cuánta memoria necesita, sobre todo cuando hay muchos datos.

En Python, podemos crear algoritmos de manera sencilla. Luego, podemos comparar diferentes formas de hacer lo mismo para elegir la mejor solución.

Marco Teórico

Un algoritmo es una serie de pasos que seguimos para resolver un problema o hacer una tarea. Podemos representarlo con dibujos, instrucciones fáciles o código en Python.

El análisis de algoritmos estudia qué tan rápido funciona un algoritmo y cuánta memoria usa, para saber si será eficiente con muchos datos.

La complejidad temporal muestra cómo cambia el tiempo que tarda el algoritmo según la cantidad de datos, y la complejidad espacial muestra cómo cambia la memoria que usa.

Usamos la notación Big O para explicar esto.

Python es un lenguaje sencillo para escribir y probar algoritmos, y tiene herramientas que ayudan a medir su tiempo y memoria.

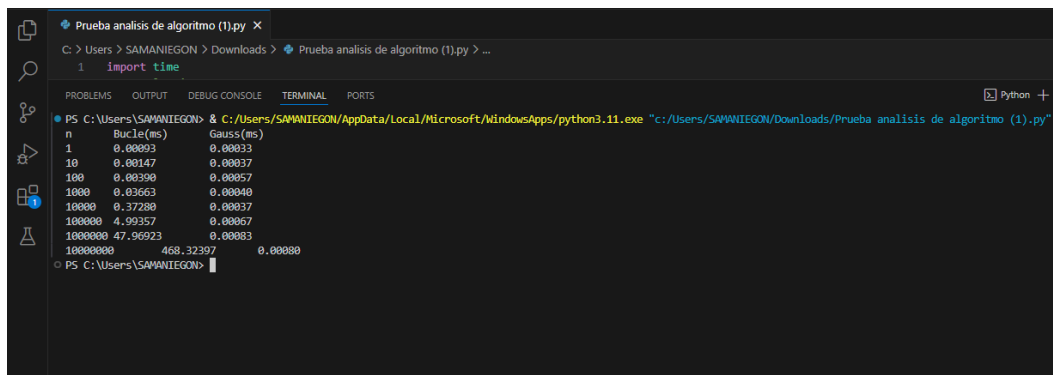
Analizar algoritmos es importante para elegir la mejor solución, hacer programas más rápidos y eficientes, y entender cómo se comportan con grandes cantidades de datos.

Caso Práctico

Para el caso práctico presentamos la importación del módulo time y ejecutamos dos funciones, una con bucle y otra con Gauss para comparar el tiempo de ejecución de ambos algoritmos.

compartimos en el siguiente print el resultado de la ejecución.

(Para el caso práctico utilizamos la librería Time).



```
PS C:\Users\SAMANIEGON> & C:\Users\SAMANIEGON\AppData\Local\Microsoft\WindowsApps\python3.11.exe "c:/Users/SAMANIEGON/Downloads/Prueba analisis de algoritmo (1).py"
n      Bucle(ms)      Gauss(ms)
1      0.00003        0.00033
10     0.00147        0.00037
100    0.00390        0.00057
1000   0.03663        0.00040
10000  0.37280        0.00037
100000 4.99357        0.00067
1000000 47.96923      0.00083
10000000 468.32397    0.00080
PS C:\Users\SAMANIEGON>
```

(invitamos a ver el video en donde detallamos en forma secuencial la ejecución del algoritmo)

Metodología utilizada

Definir la comparación

Primero, sabemos qué queremos hacer. Por ejemplo: comparar dos funciones y su tiempo de ejecución.

Escribir el algoritmo

Creamos el código en Python que resuelve el problema.

Medir el rendimiento

Para saber qué tan rápido funciona nuestro algoritmo, usamos herramientas que miden el tiempo que tarda en ejecutarse.

Probar con diferentes tamaños de datos

Probamos el algoritmo con listas pequeñas y listas muy grandes para ver cómo cambia el tiempo de ejecución cuando hay más datos.

Analizar resultados

Miramos los tiempos que obtuvimos y concluimos cuál algoritmo es mejor para el problema y por qué.

Resultados Obtenidos

Al comparar los dos algoritmos para sumar números o hacer cálculos, encontramos lo siguiente:

1) Suma con bucle

Tarda más tiempo conforme aumenta el número.

Para números pequeños es rápido, pero para números grandes puede ser lento porque suma uno por uno.

2) Suma con fórmula de Gauss

Es muy rápida, sin importar el tamaño de los números.

Usa una fórmula matemática que calcula la suma directamente, sin necesidad

de repetirla.

Conclusiones

La fórmula de Gauss es la mejor opción para sumar números porque es rápida y eficiente.

Los bucles funcionan bien con datos pequeños, pero pueden ser lentos con datos grandes.

Los algoritmos con bucles anidados (cuadráticos) pueden ser muy lentos y no son recomendables para grandes cantidades de datos.

Bibliografía

Big-O Cheat Sheet (www.bigocheatsheet.com)

Muestra de forma visual y resumida las complejidades más comunes.

GeeksforGeeks – Analysis of Algorithms

Excelente para repasar conceptos con ejemplos simples en pseudocódigo y Python.

Python time documentation

Documentación oficial sobre cómo medir tiempos con precisión.

Apuntes de cátedra. Programación I, Tecnicatura Universitaria en Programación, UTN - 2025.