Abril Duarte

Fecha de entrega: 09 de Junio del 2025

**Trabajo Integrador - Programación**

Investigación aplicada en Python - Algoritmos de búsqueda y ordenamiento, análisis de algoritmos y estructuras de datos avanzadas.

Integrantes:

Yamila Fernández (fernandezyamila075@gmail.com)  
Abril Duarte (abril.duarte18@gmail.com)  
  
Profesor:  
  
AUS Bruselario, Sebastián

# Índice:

1. Introducción ……………………………………………………………………..Pág. 2

2. Marco Teórico …………………………………………………………………..Pág. 2

3. Caso Práctico ………………………………………………………………….. Pág. 3

4. Metodología Utilizada …………………………………………………………. Pág. 8

5. Resultados Obtenidos ………………………………………………………….Pág. 9

6. Conclusiones ………………………………………………………………...… Pág. 9

7. Bibliografía ……………………………………………………………………… Pág. 10

8. Anexos…………………………………………………………………………… Pág. 10

# 1.Introducción

Un algoritmo es un conjunto de pasos o instrucciones bien definidas, diseñadas para resolver un problema específico. En programación, los algoritmos son la base de cualquier solución, y su eficiencia es crucial para el rendimiento de las aplicaciones, especialmente cuando se trabaja con grandes volúmenes de datos.

En este trabajo integrador se analiza y compara la eficiencia de distintos algoritmos de búsqueda y ordenamiento, tanto desde el punto de vista teórico como empírico, utilizando el lenguaje de programación Python.

A través del análisis de la complejidad temporal (con notación Big-O) y de pruebas prácticas cronometradas, se busca comprender cómo diferentes algoritmos se comportan frente a distintas cantidades de datos, y qué implicancias tiene su elección para el desarrollo de software eficiente.

# 2. Marco Teórico

Un algoritmo es un conjunto de pasos ordenados que permite resolver un problema determinado. En programación, los algoritmos son esenciales, ya que definen cómo se procesan los datos y cómo se llega a una solución de manera eficiente.

Existen diferentes enfoques para analizar la eficiencia de un algoritmo, principalmente en cuanto al tiempo de ejecución y el uso de memoria. Entre ellos se destacan:

🔹 Análisis empírico: consiste en ejecutar el algoritmo con diferentes entradas y medir el tiempo de ejecución real. Aunque es útil para obtener resultados concretos, depende del entorno (hardware, lenguaje, etc.), por lo que no siempre es generalizable.

🔹 Análisis teórico: se basa en estudiar el código del algoritmo y contar las operaciones que realiza en función del tamaño de entrada n. Esto permite estimar cuántos pasos realizará, sin necesidad de ejecutarlo.

🔹 Notación Big-O: es una forma de expresar la eficiencia de un algoritmo en el peor caso. Mide cómo crece el tiempo de ejecución cuando aumenta el tamaño de los datos (n). Algunas de las complejidades más comunes son:

* O(1): tiempo constante.
* O(log n): logarítmico, como en la búsqueda binaria.
* O(n): lineal, como recorrer una lista.
* O(n log n): log-lineal, típico de algoritmos como mergesort o quicksort.
* O(n²): cuadrática, como bubble sort.
* O(2ⁿ) y O(n!): exponencial y factorial, muy costosas para entradas grandes.

En este trabajo se abordan dos tipos de algoritmos fundamentales:

* Algoritmos de búsqueda, que permiten localizar un dato en una estructura. Los más comunes son:
  + *Búsqueda lineal:* recorre uno por uno los elementos.
  + *Búsqueda binaria:* divide la lista en mitades (requiere datos ordenados).
* Algoritmos de ordenamiento, que organizan los datos según un criterio. Ejemplos:
  + *Bubble Sort (burbuja):* método simple pero ineficiente en listas grandes.
  + *Quick Sort:* divide y conquista, mucho más rápido en la práctica.
  + *Insertion Sort:* eficiente para listas pequeñas o casi ordenadas.

# 3. Caso Práctico

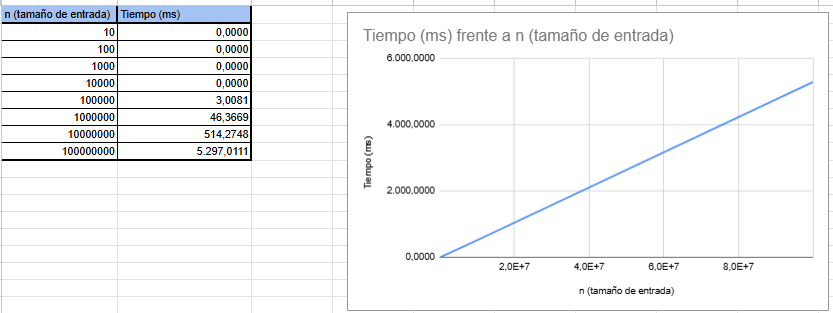
# 3.1 Análisis Empírico

Suma de los primeros n números:

Para comenzar el análisis empírico, se desarrollaron dos algoritmos diferentes que resuelven el mismo problema: sumar los primeros n números naturales.

Texto

El contenido generado por IA puede ser incorrecto.

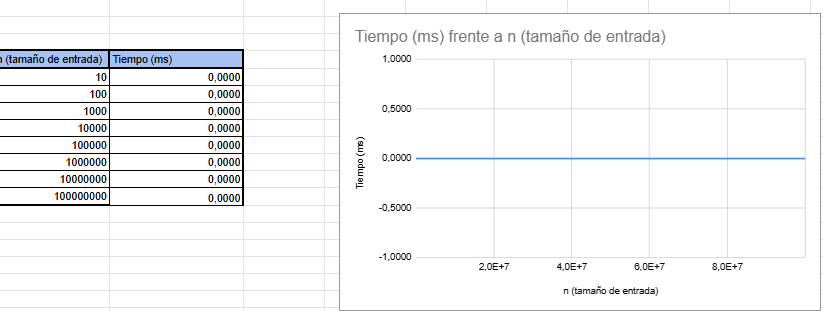


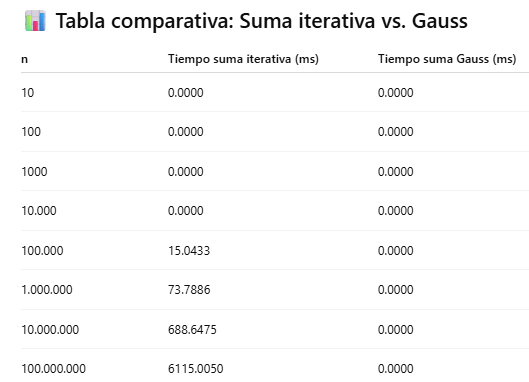
Diagrama, Texto

El contenido generado por IA puede ser incorrecto.El primer algoritmo realiza la suma mediante una estructura repetitiva, sumando uno por uno dentro de un bucle. El segundo, en cambio, utiliza la fórmula matemática de Gauss que nos permite calcular rápidamente la suma de los primeros n número naturales.

Texto

El contenido generado por IA puede ser incorrecto.





Como puede observarse, el algoritmo iterativo comienza a registrar tiempos significativos a partir de n = 100.000, y crece de forma proporcional al tamaño de entrada, alcanzando más de 5 segundos con n = 100.000.000. En contraste, el algoritmo con la fórmula de Gauss mantiene un tiempo de ejecución constante de 0.0000 ms en todos los casos, demostrando su eficiencia empírica y teórica, correspondiente a una complejidad O(1).

# 3.2 Anális Teórico

Suma n primeros números

Texto

El contenido generado por IA puede ser incorrecto.

def sumar\_numeros(n):

resultado = 0 #T(n)=1

for i in range(1, n+1): #T(n)=2\*n

resultado += i #T(n)=2

return resultado T(n)=1

T(n) = 1 + n \* 2 + 1 = 2n + 2.

Suma con Fórmula de Gauss

Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.

def sumar\_numeros(n):

return n\*(n+1)/2 #T(n)=1+1+1+1

T(n) = 1+3=4

Vamos a comparar ambos algoritmos:

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

La línea azul crece a medida que n aumenta → representa el algoritmo iterativo que realiza más operaciones cuanto mayor es la entrada.

La línea roja es plana → representa el algoritmo de Gauss, que no cambia sin importar el tamaño de n.

# 3.3. Introducción a la Notación Big-O

La notación Big-O es una forma de describir el comportamiento asintótico de una función, es decir, cómo crece cuando el tamaño de la entrada tiende a infinito. Se utiliza para simplificar la comparación de algoritmos eliminando constantes y términos de menor orden.

Pasos para calcular Big-O:

1. Identificar el término de mayor crecimiento en T(n).

2. Eliminar constantes y coeficientes.

Ejemplos:

• T(n) = 3n² + 2n + 1 → O(n²).

• T(n) = 5n + 10 → O(n).

• T(n) = 7 → O(1).

Suma de los primeros n números (bucle)

Texto

El contenido generado por IA puede ser incorrecto.

T(n) = 2n + 2 → O(n)

Esto significa que el tiempo de ejecución crece linealmente con respecto a n, por eso es una complejidad O(n).

Suma de Gauss

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

• T(n) = 4 → O(1)

Esto significa que la cantidad de operaciones no cambia aunque aumente n, por eso tiene complejidad constante O(1).

# 4. Metodología

Para llevar a cabo este trabajo se utilizó el lenguaje de programación Python, debido a su simplicidad y versatilidad para desarrollar y analizar algoritmos. Las pruebas empíricas se realizaron midiendo el tiempo de ejecución de diferentes algoritmos utilizando el módulo time de Python, que permite calcular el tiempo transcurrido entre el inicio y el final de una función en milisegundos.

Se compararon dos algoritmos para calcular la suma de los primeros *n* números naturales:

* Uno que realiza la suma mediante un bucle (estrategia iterativa).
* Otro que utiliza la fórmula matemática de Gauss, mucho más eficiente.

Para cada caso, se probaron diferentes tamaños de entrada (*n = 10, 100, 1000, ..., 100000000*) y se registraron los tiempos correspondientes. Luego, se realizó un análisis teórico complementario utilizando la notación Big O, que permite expresar el comportamiento de cada algoritmo a medida que crece el tamaño del problema.

# 5. Resultados Obtenidos

En esta sección se presentan los resultados obtenidos a partir de la ejecución de los algoritmos desarrollados para resolver el caso práctico propuesto. Se compararon dos métodos para calcular la suma de los primeros *n* números naturales:

* Método Iterativo: suma los números uno por uno en un bucle.
* Fórmula de Gauss: utiliza la fórmula matemática
* S=n(n+1)2S = \frac{n(n + 1)}{2}S=2n(n+1)​.
* Tabla

  El contenido generado por IA puede ser incorrecto.

# 6. Conclusiones

A través del análisis práctico y teórico realizado, se pudo comprobar la diferencia significativa en eficiencia entre un algoritmo iterativo y uno basado en una fórmula matemática (Gauss). Mientras que el algoritmo que recorre uno por uno los números naturales muestra un crecimiento en el tiempo de ejecución proporcional al tamaño de entrada (*O(n)*), el algoritmo de Gauss mantiene un rendimiento constante (*O(1)*) incluso con entradas extremadamente grandes.

Los resultados muestran que, si bien ambos algoritmos devuelven el mismo resultado, su eficiencia es muy distinta. En el entorno práctico, esto se traduce en tiempos de ejecución considerablemente menores al utilizar la fórmula de Gauss, lo que valida la importancia de considerar la eficiencia algorítmica al momento de programar.

Este trabajo permitió comprender y visualizar cómo los conceptos teóricos de análisis de algoritmos y notación Big O se reflejan en la práctica. Además, destaca la importancia de elegir soluciones adecuadas en función del problema, especialmente cuando se trabaja con grandes volúmenes de datos o se busca optimización.

# 7. Bibliografía

• Material de la cátedra  
• Big O Cheat Sheet (https://www.bigocheatsheet.com)  
• Documentación oficial de Python  
• Grokking Algorithms - Aditya Bhargava

# 8. Anexos

Capturas de pantalla del código, resultados, y link al repositorio de GitHub y al video explicativo.