Análisis de Algoritmos

* Alumnos:
  + Nahuel Urciuoli Zabala – [Nahuel\_zabala@live.com](mailto:Nahuel_zabala@live.com)
  + Xxxxx [x-xxx@xxx.com](mailto:x-xxx@xxx.com)
* Profesor:
  + Bruselario, Sebastián
* Tutora:
  + Gubiotti, Flor
* Fecha de entrega:
* 09/06/2025

**CONTENIDO**

[Introducción 2](#_Toc199189527)

[Marco Teórico 3](#_Toc199189528)

[Caso Práctico 4](#_Toc199189529)

[Metodología Utilizada 4](#_Toc199189530)

[Resultados Obtenidos 4](#_Toc199189531)

[Conclusiones 4](#_Toc199189532)

[Bibliografía 4](#_Toc199189533)

[Anexos 4](#_Toc199189534)

# Introducción

Ya que los algoritmos son la base de la programación pensamos que lograr algoritmos eficientes aportan también mucho peso al código escrito, pero. ¿Qué es un algoritmo? Un algoritmo es un conjunto de instrucciones bien definidas y ordenadas diseñadas para resolver un problema específico. El análisis de algoritmos evalúa su rendimiento, permitiendo comprender cómo se comportan frente a grandes volúmenes de datos. Para ello, se utilizan dos métricas fundamentales:

Eficiencia temporal: Tiempo de ejecución del algoritmo en función del tamaño de entrada (n).

Eficiencia espacial: Cantidad de memoria utilizada durante su ejecución.

Estas métricas son herramientas indispensables para desarrollar software escalable y optimizado, especialmente en sistemas que manejan datos masivos o requieren alta velocidad de procesamiento.

Este trabajo tiene como objetivo comparar algoritmos de distinta complejidad, evidenciando mediante métricas cuantitativas (tiempo de ejecución y uso de memoria) por qué algunos enfoques son más eficientes que otros

# Marco Teórico

**Este apartado contiene la fundamentación conceptual del tema tratado. Debe incluir**

**definiciones, clasificaciones, jerarquías, estructuras y sintaxis si corresponde.**

**• Incluir gráficos, esquemas o tablas si son útiles para la comprensión.**

**• Mencionar autores o documentación oficial si se toman definiciones o explicaciones.**

**• Para temas técnicos, se recomienda describir cómo se implementa el concepto en el**

**lenguaje de programación trabajado (ej. Python, Java, etc.).**

Los algoritmos son esenciales en las ciencias de la computación porque proporcionan métodos sistemáticos para resolver problemas complejos de manera eficiente y reproducible.

Hay ciertas cualidades importante que un algoritmo debe cumplir:

* Tener un número finito de pasos claramente especificado
* Aceptar cero o más entradas
* Producir al menos una salida

Y estos en lo posible deberían ser:

* Correctos
  + Resolver el problema de manera precisa.
* Robusto
  + Manejar situaciones inesperadas.
* Eficiente
  + Utilizar los recursos de manera óptima, especialmente en

términos de tiempo de ejecución y uso de memoria.

La elección de un algoritmo eficiente puede marcar la diferencia en

aplicaciones con grandes volúmenes de datos o alta demanda de

procesamiento.

¿Y cómo los identificamos?

Con el análisis de algoritmos. Este análisis es el estudio formal del rendimiento de los algoritmos midiendo sus métricas de:

* Eficiencia Temporal (Tiempo de ejecución)
* Eficiencia Espacial (Uso de memoria)

A través de un Análisis empírico o Análisis teórico (notación Big-O)

El Análisis empírico mide la eficiencia de un algoritmo mediante la observación del tiempo de ejecución de un algoritmo y cuanto tarda en resolverlo mediante diferentes inputs.

El Análisis teórico

Es un enfoque matemático sin necesidad de ejecutar el algoritmo basándose en pseudocódigo y permite calcular una función temporal que representa el numero de operaciones que realiza el algoritmo

Como realizar un análisis empírico:

* Implementar el algoritmo
  + Escribir el código del algoritmo
* Instrumentación
  + Incluir instrucciones para medir el tiempo de ejecución
* Diferente Inputs
  + Ejecutar el algoritmo con datos de entrada de diferentes tamaños
* Resultados
  + Comparar y medir resultados obtenidos
  + Pueden ser graficos como tablas

**Objetivos del Análisis de Algoritmos**

Evalúa cuánto tiempo tarda el algoritmo en ejecutarse según el tamaño de los datos de entrada

Determina cuánta memoria adicional necesita el algoritmo para completar su ejecución

Analiza cómo se comporta el algoritmo al aumentar significativamente el tamaño de los datos

**Enfoques de Análisis**

**Análisis Teórico (Asintótico)**

Utiliza la notación Big-O para expresar el crecimiento del algoritmo en función del tamaño de entrada. Proporciona una comprensión matemática del comportamiento algorítmico.

Este enfoque permite comparar algoritmos independientemente del hardware o lenguaje de programación utilizado.

**Análisis Empírico (Experimental)**

Mide el tiempo de ejecución real en diferentes condiciones y tamaños de datos. Ofrece resultados prácticos y tangibles del rendimiento.

Es especialmente útil para validar las predicciones teóricas y optimizar implementaciones específicas.

**Tipos de Análisis de Casos**

**Mejor Caso**

Evalúa el escenario más favorable para el algoritmo

**Caso Promedio**

Considera el tiempo esperado para todas las entradas posibles

**Peor Caso**

Analiza el escenario más desfavorable para el algoritmo

**Introducción a la Notación Big-O**

La notación Big-O describe cómo escala el tiempo de ejecución de un algoritmo en función del tamaño de entrada n. Es la herramienta fundamental para expresar la complejidad algorítmica de manera matemática y comparable.

Esta notación nos permite abstraer los detalles de implementación y hardware para centrarnos en el comportamiento fundamental del algoritmo cuando los datos crecen hacia el infinito.

**Complejidades Temporales Fundamentales**

**O(1) - Constante**

Tiempo independiente del tamaño de entrada. Ejemplo: acceso directo a un índice de array.

**O(log n) – Logarítmica**

Crecimiento muy lento. Ejemplo: búsqueda binaria en estructuras ordenadas.

**O(n) – Lineal**

Tiempo proporcional al tamaño. Ejemplo: recorrido completo de una lista.

**O(n²) - Cuadrática**

Crecimiento cuadrático. Ejemplo: algoritmos de ordenamiento básicos como bubble sort.

¿Qué es el Análisis de Algoritmos? Es el estudio formal del rendimiento de los algoritmos,

buscando optimizar:

• Tiempo de ejecución (Eficiencia Temporal)

• Uso de memoria (Eficiencia Espacial)

Conceptos Clave:

• Notación Big-O: Describe el peor caso de crecimiento del algoritmo (por ejemplo:

O(n), O(log n), O(n²)).

• Tiempo de ejecución real: Se puede medir usando módulos como time en Python.

• Complejidad espacial: Se refiere a la cantidad de memoria adicional que utiliza el

algoritmo.

# Caso Práctico

**Aquí se debe presentar un problema o situación concreta que haya sido desarrollada o**

**simulada para aplicar el contenido del trabajo.**

**Incluye:**

**• Breve descripción del problema a resolver.**

**• Código fuente comentado.**

**• Capturas de pantalla si corresponde.**

**• Explicación de decisiones de diseño (por ejemplo: ¿por qué se eligió un método de**

**ordenamiento sobre otro?).**

**• Validación del funcionamiento.**

**El código debe estar identado correctamente y con comentarios que expliquen los pasos**

**importantes.**

Ejemplo:Se analiza y compara dos algoritmos para calcular la suma de los primeros n números

naturales:

# Metodología Utilizada

**Metodología Utilizada**

**Describe los pasos seguidos durante el desarrollo del trabajo.**

**Se pueden incluir:**

**• Investigación previa (fuentes utilizadas).**

**• Etapas de diseño y prueba del código.**

**• Herramientas y recursos utilizados (IDE, librerías, control de versiones, etc.).**

**• Trabajo colaborativo (reparto de tareas si se realizó en grupo).**

# Resultados Obtenidos

**Detalla qué se logró con el caso práctico, qué aspectos funcionaron correctamente y qué**

**dificultades se presentaron.**

**Se pueden incluir:**

**• Casos de prueba realizados.**

**• Errores corregidos.**

**• Evaluación de rendimiento (si aplica, por ejemplo, comparar el tiempo de ejecución**

**entre algoritmos).**

**• Enlace a repositorio si el trabajo está subido a GitHub u otra plataforma.**

Ejemplo: Resultados Obtenidos

• Ambos algoritmos devuelven el mismo resultado correcto.

• La fórmula matemática es mucho más rápida que la iterativa para valores grandes

de n.

• El tiempo de ejecución para n = 10.000.000 en la suma iterativa fue cientos de veces

mayor que usando la fórmula.

# Conclusiones

**Reflexión final del grupo de trabajo. Aquí se recomienda incluir:**

**• Qué se aprendió al hacer el trabajo.**

**• Qué utilidad tiene el tema trabajado para la programación o para otros proyectos.**

**• Posibles mejoras o extensiones futuras.**

**• Dificultades que surgieron y cómo se resolvieron.**

Ejemplo: El análisis de algoritmos permite elegir mejores soluciones en función de la cantidad de datos. Aunque dos algoritmos resuelvan el mismo problema, su eficiencia puede ser muy

diferente. Python, aunque es un lenguaje interpretado, permite aplicar técnicas de análisis

de algoritmos que son esenciales para cualquier tipo de desarrollo serio.

Recomendación: Siempre analizar la complejidad de los algoritmos antes de

implementarlos en proyectos que manejarán grandes volúmenes de datos.

# Bibliografía

**Listado de fuentes consultadas, utilizando normas básicas APA u otro formato consistente.**

**Se sugiere incluir:**

**• Libros.**

**• Documentación oficial.**

**• Sitios web con fecha de acceso.**

**• Artículos o materiales recomendados por el docente.**

**Ejemplo:**

**• Python Software Foundation. (2024). Python 3 Documentation.**

**https://docs.python.org/3/**

**• Sweigart, A. (2019). Automate the Boring Stuff with Python. No Starch Press.**

# Anexos

• **Captura de resultados de la ejecución: (insertar imagen de terminal mostrando**

**tiempos de ejecución)**

**Repositorio en GitHub: https://github.com/grupo-analisis-algoritmos/analisis-python**

**Video explicativo**

**(enlace a YouTube o Drive mostrando el código y la medición de tiempos)**

**Notas de entrega**

**El repositorio debe tener:**

**• Código de los algoritmos.**

**• Archivo de README explicativo.**

**El video debe incluir:**

**• Introducción breve sobre análisis de algoritmos.**

**• Demostración práctica de los tiempos de ejecución.**

**• Reflexión grupal sobre el aprendizaje del análisis de eficiencia.**