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)

Algunas maneras de medir la Eficiencia Temporal son a través de un Análisis empírico o un 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.

Mientras que 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 número de operaciones que realiza el algoritmo

Existen diferentes tipos de Análisis:

* Peor caso (usualmente): T(n) = Tiempo máximo necesario para un problema de tamaño (n)
* Caso medio (a veces): T(n) = Tiempo esperado para un problema cualquiera de tamaño (n)  
  · (Requiere establecer una distribución estadística)
* Mejor caso (engañoso): T(n) = Tiempo menor para un problema cualquiera de tamaño (n)

**Análisis empírico**

Este método implica implementar el algoritmo y medir cuánto tiempo tarda en

resolver un problema para diferentes tamaños de entrada.

Como realizar un análisis empírico:

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

Ventajas:

* Permite obtener gráficas que muestran el tiempo de ejecución de un

Algoritmo

* Facilita la comparación visual

Desventajas:

* Es necesario implementar el algoritmo, lo que implica una inversión de tiempo y recursos.
* Para que la comparación sea válida, los algoritmos deben ejecutarse bajo las mismas condiciones de hardware y software.
* Los resultados obtenidos pueden no reflejar el comportamiento general del algoritmo, ya que solo se analizan un conjunto limitado de casos de prueba.

**Análisis teórico**

A diferencia del empírico no depende del hardware o software específico ya que no es necesario correr el código. También permite considerar todas las posibles entradas, no solo un conjunto limitado. Es más general y abstracto que el análisis empírico.

Este Análisis se realiza con unas estructuras de control, estas son reglas que definen el “valor” de cada algoritmo y al final se las suma. Por ejemplo:

* Secuencia: Si nuestro algoritmo se compone de varios bloques, su

función T(n), será la suma de las funciones T(n) de cada bloque.

T(n)=3+2+3

Return T(n)

Cada operación (+, -, \*,/ ,%,return) tienen un “valor” de 1. En el ejemplo anterior tenemos dos sumas y un return por lo tanto el “valor” del algoritmo es 3 (O(3))

* Condicionales (if-else): Como solo uno de los bloques se ejecuta o se ejecuta el if o el else se tomara el valor del algoritmo que se encuentre “debajo” de la condicional

If T(n)==5

T(n)=3+5-1

Else:

T(n)=1+5

En este ejemplo si es T==5 se ejecuta T(n)=3+5-1 teniendo un “valor” de 2(una suma y una resta) mientras que el Else un valor de 1 (solo una suma) dependiendo cual se ejecute será el valor que se tomara

* Bucles: Se calculará el número de veces que se ejecuta el bucle. Si dentro de un bucle se encuentra una función con un “valor” de 3 y el bucle se ejecuta 3 veces

Seria 3 x 3(veces)= 9

Tendría un valor de 9 O(9)

For i in range (n) se ejecutará n veces

Suma=2+1-3

La variable suma se ejecutará 2 veces y como la variable suma tiene un “valor” de 2 (una suma y una resta) 2 se ejecutará n veces quedando así O(2n)

* Bucles anidados: Es el producto del número de iteraciones de cada bucle

For i in range (n) se ejecutará n veces

For j in range (n) se ejecutará n veces

Suma=2+1-3

En este caso “Suma” se ejecutará n veces, 2 veces, 1 en bucle J y otra en bucle I. Quedando así n x n = n² y como suma tiene un “valor” de 2, Suma se ejecutará n² veces quedando así 2n²

* Notación Big-O: siempre tomara el peor caso posible de una función.

Se identifica el término de mayor crecimiento en T(n) y luego se eliminan constantes y coeficientes

T(n) = 5n² - 1n + 3 = O(n²) (Se eliminaron las constantes (sumas y retas) y el menor valor de n)

T(n) = 7n - 5 = O(n)

T(n) =9+9-3 = O(2).

**Complejidades Temporales**

**N=N+3-5 O(3) - Constante**

Tiempo independiente del tamaño de entrada.

**O(log n) – Logarítmica**

Crecimiento muy lento. Usada en búsqueda binaria

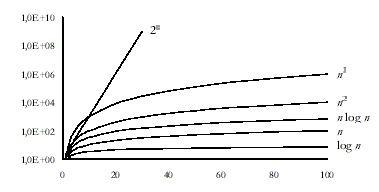
**For i in range (n) O(n) – Lineal**

Tiempo proporcional al tamaño de n

**Doble bucle for O(n²) - Cuadrática**

Crecimiento cuadrático

En este grafico se comparan los principales órdenes de complejidad. Se puede apreciar el excesivo costo que representa una complejidad de orden 2^n, frente a la estabilidad de log n y la constante siendo la menor de todas (cercana al eje x)



**Complejidad en espacio**

La idea utilizada para medir la complejidad temporal de un algoritmo también se aplica al análisis de su complejidad espacial. Cuando se dice que un programa tiene una complejidad de O(n) en espacio, significa que el uso de memoria crece de forma proporcional al tamaño del problema.

Es decir, si el tamaño del problema se duplica, también lo hace la memoria requerida. La complejidad del espacio también se expresa usando la notación O grande, pero se refiere al espacio adicional utilizado por el algoritmo, no al espacio total.

Por ejemplo, O(1) significa que el algoritmo usa una cantidad constante de espacio adicional

Mientras que en el caso de una complejidad de O(n²), el consumo de memoria aumenta con el cuadrado del tamaño del problema, por lo que duplicar el tamaño del problema implicaría utilizar cuatro veces más memoria 2²=4.

# 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.).**

Análisis teórico de la complejidad de cada algoritmo:

Suma iterativa: O(n) en tiempo, O(1) en espacio

Suma por fórmula: O(1) en tiempo y espacio.

Implementación de los algoritmos en Python.

Medición práctica usando la función time.time().

Comparación de resultados de tiempo real.

Documentación del proceso en repositorio GitHub. orativo (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

http://artemisa.unicauca.edu.co/~nediaz/EDDI/cap01.htm#ancla1\_4

**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.**