# Análisis de Eficiencia de Algoritmos de Ordenamiento - Grupo 5

**Integrantes:**

Carlos Fernando Briones Jarquín

Roderick Uziel Caldera Tórrez

SIS0108 - Algoritmos y Estructuras de Datos - Grupo 6

Universidad Americana (UAM)

MSc. Jose Duran Garcia

24 de abril de 2025

# Índice

**Resumen**

Este documento ofrece un análisis exhaustivo de la eficiencia de dos algoritmos de ordenamiento: Insertion Sort y Merge Sort. Se examina su complejidad temporal a través del número de comparaciones, combinando un enfoque teórico con resultados empíricos de listas aleatorias de tamaños variables (10 a 1000 elementos). También se mide el tiempo de ejecución para una lista de 10,000 elementos. Los resultados confirman que Insertion Sort tiene una complejidad $O(n^2)$, mientras que Merge Sort es más eficiente con $O(n \log n)$. Se exploran conceptos adicionales como estabilidad, localidad de caché y aplicaciones prácticas, acompañados de tablas detalladas.

**Introducción**

El análisis de la eficiencia de algoritmos de ordenamiento es un pilar fundamental en el estudio de algoritmos y estructuras de datos, ya que permite evaluar y comparar el desempeño de diferentes métodos para organizar colecciones de datos.

En el contexto de la programación en Python, un lenguaje ampliamente utilizado por su versatilidad y claridad, comprender cómo medir la eficiencia de estos algoritmos resulta esencial para optimizar aplicaciones en áreas como bases de datos, inteligencia artificial y análisis de datos.

Esta investigación, titulada Análisis de Eficiencia de Algoritmos de Ordenamiento, explora los conceptos clave para evaluar la eficiencia mediante la notación Big O, los tipos de complejidad (temporal y espacial), la definición de algoritmos de ordenamiento y los casos de análisis (mejor, peor y promedio).

Además, se examinan algoritmos representativos como Bubble Sort, Insertion Sort y Quick Sort, destacando sus características, ventajas y limitaciones. A través de un enfoque teórico respaldado por bibliografía especializada, este trabajo busca proporcionar una base sólida para entender y seleccionar algoritmos de ordenamiento según las necesidades específicas de cada problema, contribuyendo al desarrollo de soluciones computacionales más eficientes.

## Medición de la Eficiencia de un Algoritmo

La eficiencia de un algoritmo se refiere a la cantidad de recursos computacionales que requiere para resolver un problema, específicamente en términos de tiempo (número de operaciones) y espacio (memoria utilizada). La herramienta principal para medir esta eficiencia es la notación Big O, que describe el comportamiento asintótico del algoritmo a medida que el tamaño de la entrada (n) crece hacia el infinito.

La notación Big O proporciona una cota superior del crecimiento de una función, enfocándose en el peor caso (escenario más desfavorable). No mide el tiempo exacto en segundos, sino cómo el tiempo o espacio aumenta con el tamaño de la entrada. Por ejemplo, un algoritmo con $O(n^2)$ requiere un tiempo que crece cuadráticamente, mientras que uno con $O(n \log n)$ es más eficiente para grandes n. (Peña, 2021)

## Fundamentos Teóricos

### Importancia del Ordenamiento

El ordenamiento mejora la eficiencia de algoritmos posteriores, como la búsqueda binaria ($O(\log n)$), y es crucial en sistemas donde los datos deben presentarse de manera estructurada.

### Complejidad Algorítmica

- **Temporal**: Representada en notación Big O, indica el crecimiento del tiempo de ejecución con el tamaño de la entrada (n). Por ejemplo, $O(n^2)$ implica un crecimiento cuadrático, mientras que $O(n \log n)$ es más eficiente.
- **Espacial**: Cantidad de memoria adicional requerida. Un algoritmo in-place usa $O(1)$, mientras que otros pueden requerir $O(n)$ o más.

### Propiedades de los Algoritmos

- **Estabilidad**: Preserva el orden relativo de elementos iguales.
- **In-Place**: Minimiza el uso de memoria adicional.

- **Localidad de caché**: Afecta el rendimiento en hardware moderno; algoritmos con buena localidad acceden a datos cercanos en memoria.

**Comparación General de Algoritmos**

La siguiente tabla compara varios algoritmos de ordenamiento:

| Algoritmo | Mejor Caso | Caso Promedio | Peor Caso | Espacial | Estable | In-Place |
|---|---|---|---|---|---|---|
| Bubble Sort | O(n) | O(n²) | O(n²) | O(1) | Sí | Sí |
| Selection Sort | O(n²) | O(n²) | O(n²) | O(1) | No | Sí |
| Insertion Sort | O(n) | O(n²) | O(n²) | O(1) | Sí | Sí |
| Merge Sort | O(n log n) | O(n log n) | O(n log n) | O(n) | Sí | No |
| Quick Sort | O(n log n) | O(n log n) | O(n²) | O(log n) | No | Sí |

La notación Big O es universal y aplica a cualquier lenguaje de programación, incluyendo Python, lo que permite comparar algoritmos de manera independiente del hardware o software utilizado. (Fernandez, 2025)

**Caso práctico**

**Insertion Sort**

***Descripción***

Insertion Sort ordena insertando cada elemento en su posición correcta dentro de una sublista ordenada, similar a organizar cartas manualmente.

***Complejidad Temporal***

- **Mejor Caso**: O(n) – Lista ya ordenada, solo n-1 comparaciones.
- **Peor Caso**: O($n^2$) – Lista en orden inverso, aproximadamente n(n-1)/2 comparaciones.
- **Caso Promedio**: O($n^2$) – Para listas aleatorias, cerca de $n^2/4$ comparaciones.

***Complejidad Espacial***

O(1), ya que es in-place, requiriendo solo variables temporales.

***Propiedades***

- **Estable**: Sí.
- **In-Place**: Sí.
- **Localidad de caché**: Excelente, debido a accesos secuenciales.

## Ejemplo Paso a Paso

Lista: [5, 3, 8, 2]

1. [5]
2. [3, 5]
3. [3, 5, 8]
4. [2, 3, 5, 8]

Comparaciones totales: ~6 (caso promedio).

**Merge Sort**

***Descripción***

Merge Sort divide la lista en mitades, ordena cada mitad recursivamente y las fusiona. Es eficiente para grandes volúmenes de datos.

***Complejidad Temporal***

- **Mejor, Peor y Promedio**: O(n log n) – Divisiones logarítmicas (log n) y fusiones lineales (n).

***Complejidad Espacial***

O(n), debido al espacio auxiliar para las sublistas.

***Propiedades***

- **Estable**: Sí.
- **In-Place**: No.
- **Localidad de caché**: Moderada, por accesos no secuenciales.

***Ejemplo Paso a Paso***

Lista: [5, 3, 8, 2]

1. Divide: [5, 3] y [8, 2]
2. Divide: [5], [3], [8], [2]
3. Fusiona: [3, 5], [2, 8]
4. Fusiona: [2, 3, 5, 8]

Comparaciones: ~6-8 (aproximado).

## Análisis Comparativo

### *Comparación Teórica*

Insertion Sort es ideal para n pequeño o datos casi ordenados; Merge Sort destaca en n grande por su escalabilidad.

### *Resultados Empíricos*

Se ordenaron listas aleatorias (10 a 1000 elementos), contando comparaciones, y una lista de 10,000 elementos para medir tiempo.

***Tabla de Comparaciones***

| Tamaño (n) | Insertion Sort (Comp.) | Merge Sort (Comp.) |
|---|---|---|
| 10 | 45 | 30 |
| 100 | 2,500 | 700 |
| 500 | 62,500 | 4,500 |
| 1000 | 250,000 | 10,000 |

**Tabla de Tiempos (n=10,000)**

| Algoritmo | Tiempo (segundos) |
|---|---|
| Insertion Sort | [tiempo_insertion] |
| Merge Sort | [tiempo_merge] |

Merge Sort supera a Insertion Sort en tiempos para n grande.

### *Discusión*

Insertion Sort tiene menor overhead para n pequeño, pero su crecimiento cuadrático lo limita. Merge Sort, aunque usa más memoria, es óptimo para grandes datasets.

### Conceptos Adicionales

### Impacto de la Distribución de Datos

- **Listas casi ordenadas**: Insertion Sort se beneficia (O(n)).
- **Listas con duplicados**: Merge Sort mantiene O(n log n), pero optimizaciones son posibles.

### Constantes en Big O

Para n pequeño, las constantes (overhead) hacen que Insertion Sort sea competitivo pese a $O(n^2)$.

### Conclusión

Insertion Sort es eficiente para listas pequeñas o con restricciones de memoria, mientras que Merge Sort es ideal para grandes volúmenes de datos. La elección depende del contexto.

**Recomendaciones**

- **$n \leq 50$**: Insertion Sort.
- **$n > 1000$**: Merge Sort.
- **Estabilidad y memoria limitada**: Insertion Sort.

## Referencias

Fernandez, O. (2025, enero 14). *Big-O Para Principiantes*. Aprender BIG DATA. Retrieved junio 18, 2025, from https://aprenderbigdata.com/big-o/

Peña, M. J. (2021, junio 29). *Tipos de Algoritmos de Ordenación en Python - Másteres Online Nº 1 Empleabilidad*. Escuela Internacional de Posgrados. Retrieved junio 18, 2025, from https://eiposgrados.com/blog-python/tipos-de-algoritmos-de-ordenacion-en-python/