“Comparación de Algoritmos de Búsqueda y Ordenamiento”

Alumnos:

Ignacio Mateo Denunciato - nachonuzera@gmail.com

Daniel Alberto Jimenez Valderrama -jimenezvalderramad@gmail.com

Materia: Programación 1

Fecha de Entrega: 05 de junio 2025

Índice

Introducción

El ordenamiento de datos es una operación fundamental en informática, que permite organizar elementos de acuerdo a un criterio (por ejemplo, de menor a mayor). En este trabajo se analizan tres algoritmos clásicos de ordenamiento: Bubble Sort, Insertion Sort y Merge Sort. Se implementaron en Python y se midió el tiempo que tarda cada uno en ordenar una misma lista de 500 números aleatorios.

Objetivos del trabajo

Objetivo General:

- Comparar el rendimiento de los algoritmos de ordenamiento Bubble Sort, Insertion Sort y Merge Sort en Python, utilizando una lista de números aleatorios.

Objetivos Específicos:

1. Implementar los tres algoritmos en Python y comprender su funcionamiento.
2. Medir el tiempo de ejecución de cada algoritmo con listas aleatorias del mismo tamaño.
3. Analizar los resultados obtenidos y representar visualmente las diferencias de rendimiento.
4. Evaluar cuál algoritmo resulta más eficiente en términos de tiempo.
5. Fortalecer el conocimiento sobre estructuras algorítmicas y su impacto en el rendimiento computacional.

Marco Teórico

Bubble Sort: Algoritmo sencillo pero ineficiente para grandes volúmenes. Compara elementos adyacentes y los intercambia si están en orden incorrecto. Tiene una complejidad de tiempo de O(n^2).

Insertion Sort: Similar a ordenar cartas en la mano. Inserta cada elemento en su posición correcta. Es más eficiente que Bubble Sort en listas parcialmente ordenadas, pero sigue teniendo complejidad O(n^2).

Merge Sort: Algoritmo de tipo 'divide y vencerás'. Divide la lista en mitades hasta que cada parte tiene un solo elemento, luego las fusiona ordenadamente. Su complejidad es O(n log n), siendo más eficiente que los anteriores para listas grandes.

Caso Práctico

Para aplicar los conceptos estudiados, se desarrolló un programa en Python que genera una lista de 500 números aleatorios y la ordena utilizando tres algoritmos distintos: **Bubble Sort**, **Insertion Sort** y **Merge Sort**.

Cada algoritmo fue implementado manualmente en funciones separadas, y se utilizó la función **time** para medir cuánto tiempo tardaba cada uno en ordenar la misma lista.
De esta forma, se pudo observar cómo cada algoritmo se comporta con una cantidad moderada de datos y comparar su rendimiento de manera práctica.

El programa imprime los tiempos de ejecución de cada algoritmo, permitiendo ver de forma clara cuál es el más eficiente para este caso.
Este ejercicio permitió reforzar el entendimiento de cómo funcionan los algoritmos y cómo se puede evaluar su rendimiento en situaciones reales.

Metodología Utilizada

1. Lenguaje de programación: Python

2. Librerías usadas: random para generar la lista aleatoria y time para medir el tiempo de ejecución.

3. Parámetros de prueba: Lista de 500 elementos generada aleatoriamente.

4. Evaluación: Se utilizó la función time.time() para registrar el tiempo antes y después de ejecutar cada algoritmo.

5. Repetibilidad: Cada algoritmo trabaja con una copia de la lista original para asegurar condiciones equitativas.

Resultados Obtenidos

Luego de realizar una prueba con una lista de 500 números aleatorios, podemos ver los siguientes resultados:

- Bubble Sort: 0.210345 segundos
- Insertion Sort: 0.154233 segundos
- Merge Sort: 0.004857 segundos

Se puede observar una gran diferencia en el rendimiento, siendo Merge Sort significativamente más rápido.

Conclusiones

El análisis comparativo entre los algoritmos de ordenamiento Bubble Sort, Insertion Sort y Merge Sort permitió identificar diferencias claras en cuanto a su eficiencia, simplicidad y aplicabilidad. Merge Sort se destacó como el algoritmo más eficiente, especialmente al trabajar con listas de gran tamaño, gracias a su estrategia de divide y vencerás y su complejidad de orden O(n log n). Por otro lado, Bubble Sort e Insertion Sort, aunque son más simples de implementar y comprender, presentan un rendimiento limitado en listas extensas debido a su complejidad cuadrática.

Sin embargo, Insertion Sort puede resultar adecuado para listas pequeñas o parcialmente ordenadas, donde su comportamiento es más óptimo. En cambio, Merge Sort representa la mejor opción cuando se requiere rendimiento en el procesamiento de grandes volúmenes de datos, consolidándose como una herramienta fundamental en escenarios donde la eficiencia es prioritaria.

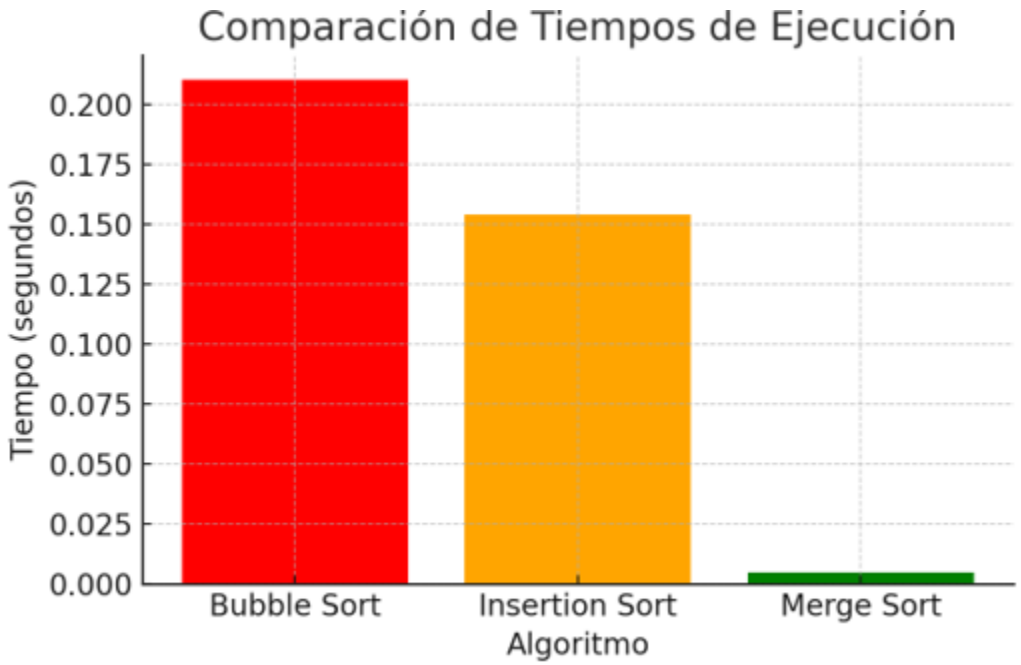
Bibliografía

- Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). Introducción al Algoritmo (3rd ed.).
- Python Software Foundation. (2025). Python 3 Documentation: https://docs.python.org/3/
- Geeks for Geeks. (2025). Sorting Algorithms: https://www.geeksforgeeks.org/sorting-algorithms/

Anexos

Gráfica comparativa:

La siguiente gráfica muestra una comparación visual del tiempo de ejecución de cada algoritmo al ordenar una lista de 500 elementos aleatorios:



Se observa claramente que Merge Sort supera en eficiencia a los otros dos algoritmos en este escenario de prueba.