# Análisis y Comparación de Algoritmos de Ordenamiento

## 1. Objetivo General

Analizar, comparar y discutir el comportamiento de los algoritmos de ordenamiento Bubble Sort, Insertion Sort, QuickSort y HeapSort, considerando tanto su desempeño teórico (complejidad temporal y estructura del algoritmo) como empírico (tiempo de ejecución y cantidad de operaciones) a través de la implementación en Python y la experimentación con distintos tamaños de entrada.

## 2. Implementación

Cada algoritmo fue programado utilizando el lenguaje Python, asegurando la corrección de las implementaciones y la equidad en las condiciones de prueba. Se desarrolló un script que permitió ejecutar cada método con cinco conjuntos de datos aleatorios y tamaños de entrada crecientes (1.000, 5.000, 10.000, 20.000 y 50.000 elementos). Durante la ejecución se midió el tiempo total de procesamiento y el número total de operaciones básicas (comparaciones e intercambios). Estos resultados fueron luego promediados y graficados para realizar un análisis comparativo del rendimiento.

## 3. Análisis Teórico

A continuación, se detalla la complejidad teórica de cada algoritmo:

|  |  |  |
| --- | --- | --- |
| Algoritmo | Complejidad Temporal (Peor Caso) | Ecuación de Tiempo / Explicación |
| Bubble Sort | O(n²) | Comparaciones anidadas: T(n) ≈ n(n−1)/2. |
| Insertion Sort | O(n²) | Inserciones secuenciales: T(n) ≈ n(n−1)/2. |
| QuickSort | O(n²) (promedio O(n log n)) | División recursiva: T(n) = 2T(n/2) + cn. |
| HeapSort | O(n log n) | Construcción del heap O(n) + n extracciones O(log n). |

## 4. Experimentos Empíricos

Se realizaron 5 experiencias empíricas con conjuntos de datos aleatorios de distintos tamaños: 1000, 5000, 10000, 20000 y 50000 elementos. Para cada tamaño se midió el tiempo de ejecución y la cantidad total de operaciones (comparaciones + asignaciones) de cada algoritmo.

Las gráficas a continuación muestran las tendencias observadas en el tiempo de ejecución y el número de operaciones según el tamaño de la entrada.

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

## Gráfico, Gráfico de líneas El contenido generado por IA puede ser incorrecto.

## 5. Discusión de Resultados

Los resultados empíricos coinciden con las predicciones teóricas. El Bubble Sort mostró el peor rendimiento debido a su necesidad de múltiples pasadas y comparaciones redundantes. El Insertion Sort resultó algo más eficiente para listas pequeñas, pero su desempeño disminuyó rápidamente con el tamaño de la entrada. Por otro lado, QuickSort destacó por su rapidez en la mayoría de los casos, gracias a su estrategia de dividir y conquistar, aunque puede degradarse al peor caso si el pivote no se selecciona adecuadamente. Finalmente, HeapSort mantuvo un rendimiento predecible y eficiente, demostrando su utilidad en escenarios donde la estabilidad del tiempo de ejecución es importante. Estos resultados concuerdan con la literatura clásica sobre algoritmos de ordenamiento (Sedgewick & Wayne, 2011), donde se demuestra que QuickSort suele superar a HeapSort en promedio, aunque este último posee una mejor garantía de rendimiento en el peor caso.

## 6. Conclusiones

El estudio permitió confirmar la relación entre la complejidad teórica y el comportamiento empírico de los algoritmos de ordenamiento. Los algoritmos con complejidad cuadrática (Bubble e Insertion) son inviables para volúmenes de datos grandes, mientras que QuickSort y HeapSort resultan más escalables y eficientes.

En términos prácticos:

1. QuickSort es ideal para grandes conjuntos de datos aleatorios.
2. HeapSort ofrece estabilidad y previsibilidad en su rendimiento.
3. Insertion Sort es útil solo para listas pequeñas o casi ordenadas.
4. Bubble Sort es principalmente pedagógico.

## 7. Respuestas y Observaciones

Durante el desarrollo del experimento, se comprobó de manera práctica la diferencia de rendimiento entre los algoritmos de ordenamiento estudiados. En particular, se observó que los algoritmos de complejidad cuadrática, Bubble Sort e Insertion Sort, presentan un aumento exponencial en el tiempo de ejecución conforme crece el tamaño de los datos. Esto se debe a que realizan un número de comparaciones y desplazamientos proporcional, lo que los vuelve poco eficientes en contextos con grandes volúmenes de información.

En contraste, los algoritmos QuickSort y HeapSort demostraron un rendimiento mucho más eficiente, manteniendo un crecimiento cercano. El QuickSort resultó ser el más veloz en promedio, gracias a su técnica de divide y vencerás, que permite reducir rápidamente el tamaño de los subproblemas. Sin embargo, también se observó que su desempeño puede verse afectado cuando los pivotes seleccionados no dividen bien el conjunto de datos, acercándose así a su peor caso. El HeapSort, aunque ligeramente más lento que QuickSort en los casos promedio, ofreció una ejecución más estable y consistente, sin grandes variaciones entre pruebas. Este comportamiento lo convierte en una opción sólida cuando se requiere garantizar tiempos de respuesta uniformes, independientemente del tipo de datos procesados. A nivel empírico, los gráficos obtenidos permiten visualizar claramente las diferencias en las pendientes de las curvas de tiempo y operaciones. Finalmente, se pudo comprobar la importancia de medir experimentalmente el comportamiento de los algoritmos, ya que la complejidad asintótica, aunque útil para comparar eficiencia a gran escala, no siempre refleja las diferencias prácticas en contextos reales o con tamaños de datos moderados. El análisis empírico permitió corroborar que QuickSort y HeapSort son los algoritmos más adecuados para implementaciones productivas o sistemas que requieren rapidez en el procesamiento de información.