Comparación de la Rapidez de Algoritmos de Ordenamiento

Camila Rodríguez Águila - C36624
CI-0116 Análisis de Algoritmos y Estructuras de Datos - I-2025
Correo Electrónico: camila.rodriguezaguila@ucr.ac.cr

***Abstract*—Este informe presenta una comparativa del tiempo de ejecución de distintos algoritmos de ordenamiento, la comparación se basa en tiempos de ejecución para diferentes tamaños de entrada. Se analizaron seis algoritmos: selección, inserción, mezcla, montículos, rápido y residuos. A través de 3 ejecuciones se midieron los tiempos, de los cuales se obtuvo un promedio para cada uno, demostrando las diferencias de eficiencia entre algoritmos cuadráticos y algoritmos que cuentan con un mejor comportamiento asintótico.**

***Palabras clave*—algoritmos, análisis, computación, estructuras de datos, rapidez**

I. Introducción

En el mundo de la computación, los algoritmos de ordenamiento son muy importantes porque nos ayudan a ordenar grandes cantidades de datos de forma rápida y ordenada. Aunque aprender la teoría detrás de estos algoritmos es útil para entender el trasfondo de cómo funcionan, también es necesario probarlos en la práctica para ver realmente cómo se comportan. Por eso, en este trabajo se comparan seis algoritmos de ordenamiento diferentes: selección, inserción, mezcla, montículos, rápido y residuos, analizando cuánto tardan en ordenar datos de distintos tamaños.

La idea de ejecutar esta comparación es poder ver con nuestros propios ojos qué tan rápidos son los algoritmos cuando se usan con diferentes tamaños de listas. A veces, dependiendo de la cantidad de datos o de cómo están organizados, algunos algoritmos funcionan mejor que otros. Para sacar conclusiones más confiables, cada prueba se repitió 3 veces y se calculó un promedio del tiempo que tardó cada algoritmo en ordenarlos.

Este informe busca ayudar a entender mejor cómo funcionan estos métodos de ordenamiento en la práctica, mostrando las diferencias reales entre algoritmos más lentos y los que están diseñados para ser más rápidos. Así, se puede aprender cuál conviene usar según el caso, y tomar decisiones más acertadas al momento de programar o resolver problemas con grandes cantidades de datos.

II. Metodología

Cada algoritmo fue ejecutado tres veces sobre listas de cuatro tamaños: 1000, 10000, 100000, y 1000000 elementos. Se registró el tiempo de ejecución en microsegundos para cada corrida, y se calculó el promedio. El entorno de ejecución fue el mismo para todos los algoritmos, evitando otras tareas que pudieran alterar los resultados, además, es importante mencionar que la computadora estaba dedicada exclusivamente a esta tarea en todo momento.

III. Resultados

Los tiempos promedio obtenidos se muestran en la Tabla I. Los algoritmos de mezcla, montículos y rápido presentan un rendimiento notablemente superior en comparación con selección e inserción. En cambio, el algoritmo de residuos muestra resultados variables según el caso, sin embargo, el algoritmo de residuos presenta resultados mixtos.

Tabla I: Tiempos Promedio de Ejecución (en microsegundos)

Algoritmo	**1,000**	**10,000**	**100,000**	**1,000,000**
Selección	1,526	128,949	16,237,079	1,254,679,998
Inserción	726	64,341	6,377,704	639,481,776
Mezcla	194	1,472	15,405	181,881
Montículos	206	2,266	28,640	335,299
Rápido	120	2,326	92,905	12,451,941
Residuos	1,075	757	6,705	68,818

IV. Discusión

Los resultados obtenidos son en gran parte consistentes con lo esperado desde la teoría algorítmica. En el caso de selección e inserción, ambos algoritmos presentan un comportamiento cuadrático, lo que significa que su tiempo de ejecución crece rápidamente conforme aumenta la cantidad de datos. Esto se refleja claramente en la Tabla I, donde para un millón de elementos, los tiempos alcanzan valores muy altos, volviéndose imprácticos para aplicaciones con grandes volúmenes de datos.

Por otro lado, los algoritmos de mezcla y montículos, con complejidad $O(n \log n)$, demostraron una mayor estabilidad y eficiencia ante el crecimiento del tamaño de la entrada. A pesar de que el tiempo también aumenta con más elementos, lo hace de manera mucho más moderada. Quicksort, con su eficiencia promedio también de $O(n \log n)$, ofreció un rendimiento competitivo, aunque con una excepción notable en la prueba de 100,000 elementos, donde su tiempo fue inesperadamente alto, posiblemente debido a un caso desfavorable que activó su peor caso de $O(n^2)$.

El algoritmo de residuos, o Radix Sort, presentó un comportamiento interesante. Sus tiempos fueron especialmente buenos en listas grandes, lo cual es coherente con su naturaleza no comparativa y su posible complejidad lineal. Sin embargo, en listas pequeñas no fue tan eficiente, lo que indica que su rendimiento está altamente influenciado por el tamaño y tipo de entrada. Esto sugiere que no siempre es la mejor opción por defecto, sino que debe seleccionarse considerando el contexto específico del problema.

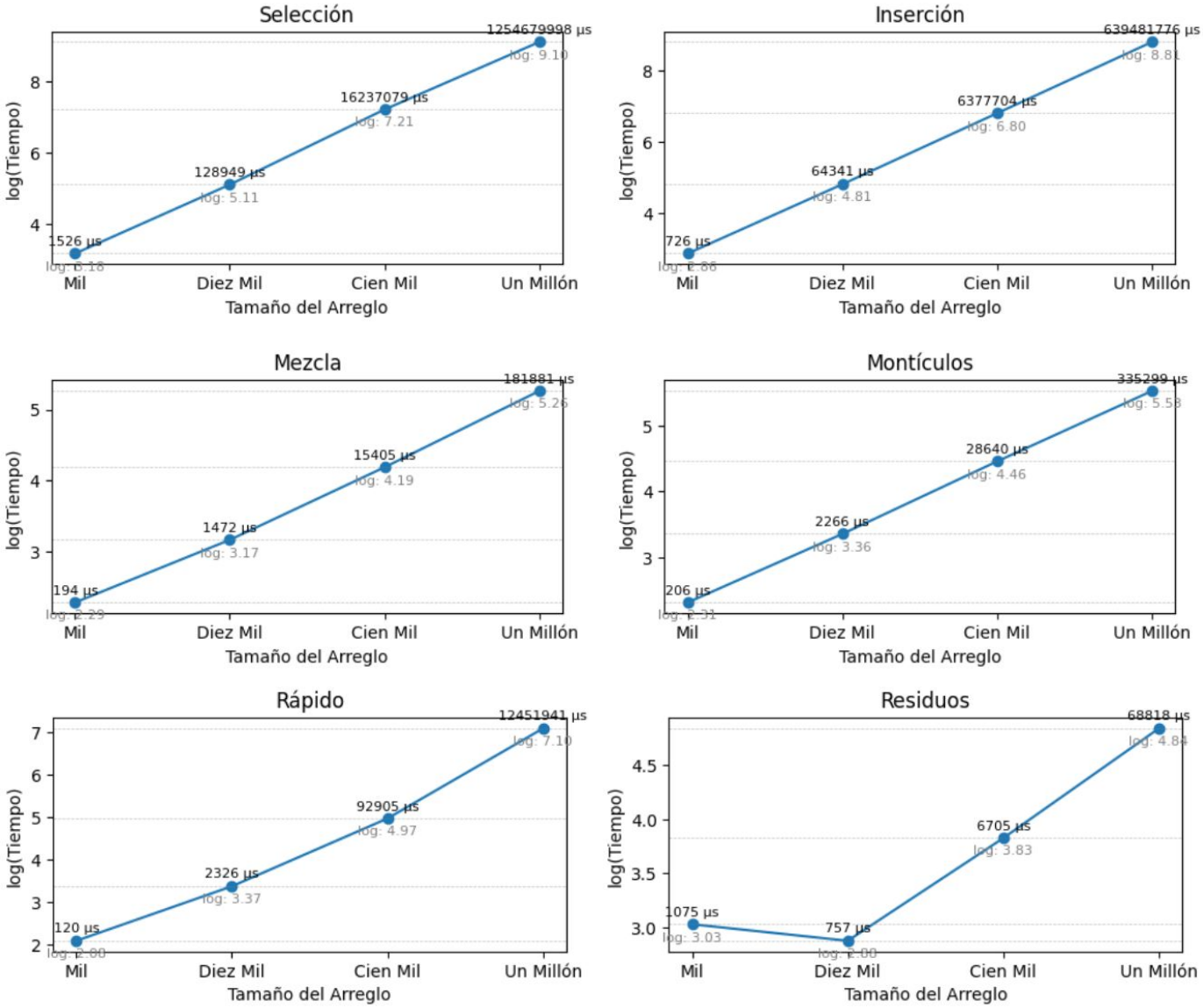

Fig. 1: Tiempos promedio de ejecución de los algoritmos de ordenamiento por selección, inserción, mezcla, montículos, rápido y residuos.

V. Conclusiones

Este estudio permitió analizar de manera práctica las complejidades de diversos algoritmos de ordenamiento de acuerdo a la notación asintótica, validando cómo estas se manifiestan al aplicar los algoritmos en listas de distinto tamaño. Se confirmó que los algoritmos con complejidad cuadrática deben evitarse en aplicaciones reales con grandes volúmenes de datos. A pesar de que son útiles para enseñar los diferentes tipos de algoritmos que existen, estos en realidad no son eficientes

Los algoritmos con comportamiento $O(n \log n)$ como mezcla, montículos y quicksort se destacaron por su eficiencia, viéndose como soluciones efectivas para diferentes casos. En particular, mezcla y montículos mostraron consistencia y rendimiento estable incluso con entradas grandes, mientras que quicksort, aunque generalmente rápido, puede ser sensible a ciertos casos desfavorables dependiendo del estado inicial de los datos.

Finalmente, Radix Sort demostró que es un algoritmo con grandes ventajas en escenarios con grandes cantidades de datos, al alcanzar tiempos notablemente bajos. Sin embargo, su rendimiento con listas pequeñas no fue tan bueno, lo cual indica que no hay un algoritmo completamente superior. Por esto, es importante que la elección del algoritmo de ordenamiento se base no solo en su análisis teórico, sino también en el tipo, tamaño y distribución de los datos a procesar.

Referencias

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

VI. Anexo: Gráfico Comparativo

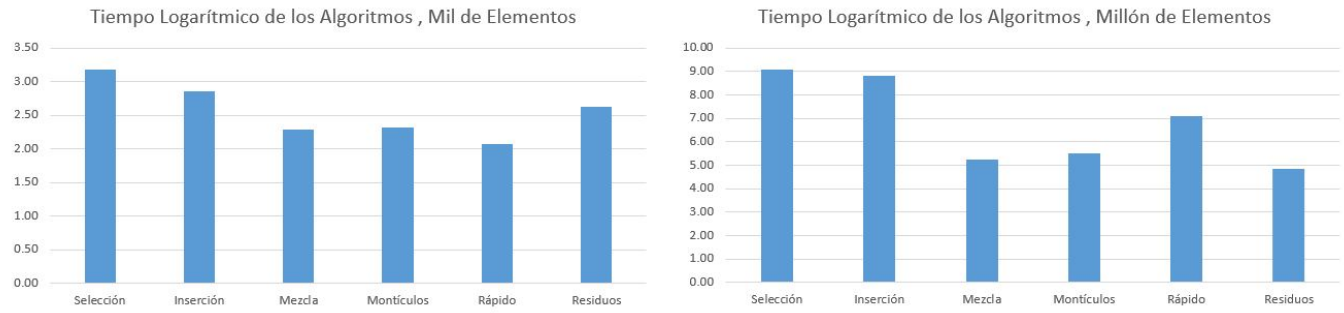

Fig. 2: Tiempos promedio de ejecución de los algoritmos.

Los gráficos muestran el rendimiento de distintos algoritmos de ordenamiento en escala logarítmica, comparando su eficiencia con un millón y mil elementos. Para el caso del millón, los algoritmos simples como selección e inserción resultan ser los más lentos debido a su complejidad $O(n^2)$, mientras que los algoritmos más avanzados como mezcla, montículos y rápido, con complejidad $O(n \log n)$, ofrecen mejor desempeño. El algoritmo de residuos destaca por ser el más eficiente, probablemente por ser de tipo lineal como Radix o Counting Sort.

En cambio, con mil elementos, las diferencias se reducen. Aunque selección e inserción siguen siendo los más lentos, la ventaja de los algoritmos complejos disminuye debido a la menor cantidad de datos, y el algoritmo de residuos ya no es el más rápido, lo que sugiere que su eficiencia depende de trabajar con grandes volúmenes.

Esto es solo una diferente forma de visualizar la ya acordada conclusión, los algoritmos eficientes como Quicksort o Radix son preferibles para grandes conjuntos de datos, mientras que para tamaños pequeños, algoritmos simples pueden ser suficientes.