Análisis de los Algoritmos de Ordenamiento por Selección, Inserción, Mezcla, Montículos, Rápido y por Residuos

Josué Torres Sibaja, C37853, josue.torressibaja@ucr.ac.cr

Resumen—El presente trabajo analiza el comportamiento de los algoritmos de ordenamiento por Selección, Inserción, Mezcla, Montículos, Rápido y Residuos, comparando sus complejidades temporales teóricas con resultados prácticos obtenidos a través de pruebas experimentales. Los algoritmos fueron implementados en C++, y las pruebas se realizaron en un entorno controlado usando el sistema operativo Windows 11 con WSL 2, y un procesador AMD Ryzen 7. Para la medición de tiempos, se empleó la biblioteca *chrono*, realizando tres corridas con arreglos de diferentes tamaños (50 000, 100 000, 150 000 y 200 000 elementos) para cada algoritmo. Los resultados demostraron que, si bien los tiempos de ejecución concuerdan generalmente con las cotas teóricas, ciertos factores, como la selección de pivotes o el estado inicial del arreglo, pueden afectar el rendimiento en mayor o menor medida. Se concluyó que estos hallazgos resaltan la importancia de realizar pruebas empíricas para obtener una visión más completa del comportamiento de los algoritmos en contextos prácticos.

Palabras clave—algoritmo, ordenamiento, complejidad, temporal, pruebas.

I. INTRODUCCIÓN

El análisis de algoritmos tiene una gran importancia en las ciencias de la computación y la informática, ya que nos permite comprender y predecir el comportamiento de un algoritmo con relación al tiempo, permitiéndonos establecer cotas teóricas para los diferentes casos en los que se utilicen. Estas cotas nos permiten realizar estimaciones sobre el tiempo de ejecución de cada algoritmo respecto al tamaño de los datos que procesan, sin embargo, la realización de pruebas y experimentos nos permiten comparar los resultados prácticos de los tiempos de ejecución con sus estimaciones teóricas (Cormen et al., 2022).

Asimismo, es importante mencionar que existen diferentes tipos de algoritmos de ordenamiento, como el Algoritmo de Ordenamiento por Selección, el Algoritmo de Ordenamiento por Inserción, el Algoritmo de Ordenamiento por Mezcla, el Algoritmo de Ordenamiento por Montículos, el Algoritmo de Ordenamiento Rápido, y el Algoritmo de Ordenamiento por Residuos, entre otros (Cormen et al., 2022).

El Algoritmo de Ordenamiento por Selección, busca el menor valor en un arreglo y lo coloca en la primera posición de la parte desordenada del arreglo. Posee una complejidad temporal de $\Theta(n^2)$ en todos sus casos (peor caso, caso promedio y mejor caso) (Cormen et al., 2022; W3Schools, n.d.).

El Algoritmo de Ordenamiento por Inserción, toma el primer valor en la parte desordenada de un arreglo y lo compara con el siguiente, intercambiándolos si el valor actual es mayor al siguiente. Posee una complejidad temporal de $\Theta(n^2)$ en su peor caso y en sus casos promedio, y una complejidad temporal de $\Theta(n)$ en su mejor caso (Cormen et al., 2022; W3Schools, n.d.).

El Algoritmo de Ordenamiento por Mezcla, divide un arreglo de valores a la mitad, y divide recursivamente los subarreglos hasta que contengan solamente un elemento. Luego, une los arreglos, dejando el valor menor siempre de primero, y continúa hasta unir todos los arreglos. Posee una complejidad temporal de $\Theta(n \log n)$ en todos sus casos (peor caso, caso promedio y mejor caso) (Cormen et al., 2022; W3Schools, n.d.).

El Algoritmo de Ordenamiento por Montículos, toma un arreglo de valores y lo reorganiza en una estructura de datos llamada montículo máximo, dejando el mayor elemento como su raíz. Luego, intercambia la raíz por el último elemento, dejando al valor que estaba en la raíz en su posición correcta. Posee una complejidad temporal de $\Theta(n \log n)$ en todos sus casos (peor caso, caso promedio y mejor caso) (Cormen et al., 2022; W3Schools, n.d.).

El Algoritmo de Ordenamiento Rápido selecciona un valor en el arreglo como pivote, y ordena el arreglo de forma

que los valores menores al pivote queden a su izquierda, y los valores mayores queden a su derecha. Luego, realiza el mismo proceso recursivamente para los subarreglos menores y mayores. Posee una complejidad temporal de $\Theta(n^2)$ en su peor caso, y una complejidad temporal de $\Theta(n \log n)$ en sus casos promedio y en su mejor caso (Cormen et al., 2022; W3Schools, n.d.).

El Algoritmo de Ordenamiento por Residuos ordena un arreglo por dígitos individuales, de derecha a izquierda, ordenando respecto al dígito que se está tomando en cuenta, y luego respecto a los siguientes dígitos. Posee una complejidad temporal de $\Theta(d(n + k))$ en todos sus casos (peor caso, caso promedio y mejor caso) (Cormen et al., 2022; W3Schools, n.d.).

En este trabajo, se llevará a cabo una comparación entre las cotas teóricas de complejidad temporal y los tiempos de ejecución prácticos de los algoritmos de ordenamiento mencionados anteriormente. El propósito de este análisis es estudiar si, y hasta qué punto, las predicciones y el comportamiento teórico de los algoritmos se alinea con sus resultados prácticos. De esta forma, se podrá obtener una comprensión más completa del comportamiento real de los algoritmos de ordenamiento estudiados.

II. METODOLOGÍA

Para lograr lo propuesto, se realizó la implementación de los algoritmos en el lenguaje de programación C++, utilizando el sistema operativo Windows 11, en el cual se instaló WSL 2 (Windows Subsystem for Linux) para facilitar la compilación. Para garantizar condiciones consistentes para todas las pruebas, se deshabilitó cualquier proceso de fondo que pudiera generar interferencias. Para la medición en milisegundos de los tiempos de ejecución se utilizó la biblioteca *chrono* (específicamente la función *chrono::high_resolution_clock*) para obtener mediciones de alta precisión, capturando el tiempo de inicio y finalización de cada ejecución.

Las pruebas se corrieron por medio de un código de prueba que realizaba la ejecución de cada corrida, la medición de tiempo y la impresión de los resultados. Para esto, se utilizó el IDE Visual Studio Code, el compilador g++ (Ubuntu 11.4.0), y un equipo de 16 GB de RAM con procesador AMD Ryzen 7. El código de los algoritmos de ordenamiento está basado en el pseudocódigo del libro de Cormen y colaboradores.

Se realizaron tres pruebas con cuatro diferentes tamaños de arreglos de valores (50 000, 100 000, 150 000 y 200 000) para cada algoritmo de ordenamiento, con el objetivo de obtener el tiempo de ejecución de cada prueba y el promedio de las tres pruebas. Los números en los arreglos eran de tipo uint32_t. El arreglo de valores aleatorios utilizado fue el mismo en cada ejecución de cada algoritmo de ordenamiento (solamente se modificó su tamaño), y contenía números en el rango de $[0, (2^{32}) - 1]$.

Tras realizar las pruebas y obtener el tiempo de cada ejecución, estos se registraron junto con su promedio en el Cuadro I. A partir de los tiempos promedio, se generaron gráficos individuales y comparativos para visualizar el comportamiento de cada algoritmo de ordenamiento según incrementa el tamaño del arreglo.

III. RESULTADOS

Algoritmo	Tam.	Tiempo (ms) Corrida 1	2	3	Prom.
Selección	50 000	2528.65	2526.57	2527.16	2527.46
	100 000	10140.20	10122.50	10153.80	10138.90
	150 000	22890.50	22810.10	22838.70	22846.40
	200 000	40667.40	40472.10	40454.30	40531.30
Inserción	50 000	1164.54	1164.68	1169.12	1166.11
	100 000	4715.16	4845.85	6998.94	5519.98
	150 000	14237.70	13579.70	13180.50	13666.00
	200 000	20792.30	20344.20	21477.60	20871.40
Mezcla	50 000	19.76	19.95	20.40	20.04
	100 000	41.13	40.77	42.57	41.49
	150 000	62.61	63.02	62.32	62.65
	200 000	84.66	84.51	85.16	84.78
Montículos	50 000	34.27	33.61	32.89	33.59
	100 000	70.64	66.73	67.02	68.13
	150 000	106.28	109.38	114.11	109.92
	200 000	150.62	150.78	152.16	151.19
Rápido	50 000	15.12	16.31	17.89	16.44
	100 000	33.91	33.31	33.13	33.45
	150 000	51.92	49.52	51.47	50.97
	200 000	69.52	67.66	66.99	68.06
Residuos	50 000	5.45	7.26	5.62	6.11
	100 000	10.86	10.83	10.85	10.84
	150 000	18.18	14.18	14.69	15.69
	200 000	20.39	19.46	19.78	19.88

Figura 1

Gráfico de los tiempos de ejecución promedio del Algoritmo de Ordenamiento por Selección

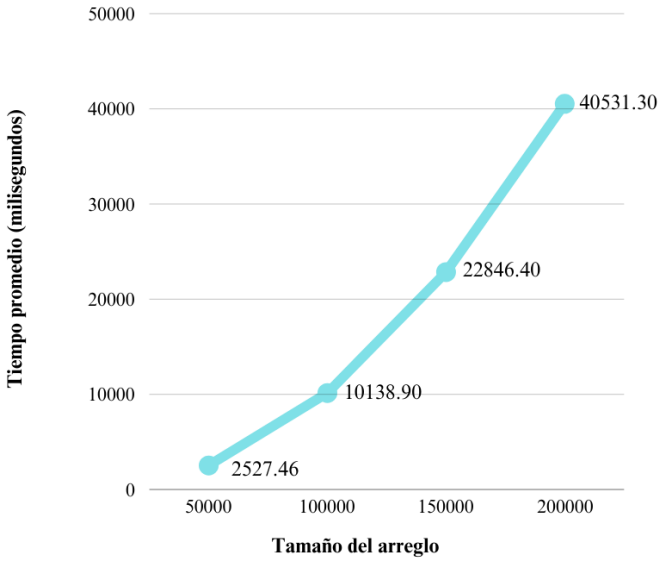


Figura 2

Gráfico de los tiempos de ejecución promedio del Algoritmo de Ordenamiento por Inserción

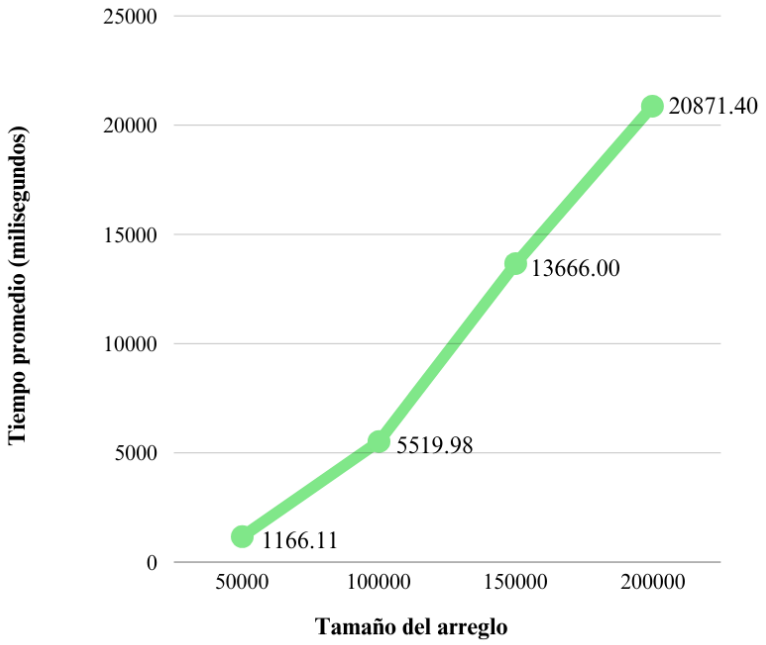


Figura 3

Gráfico de los tiempos de ejecución promedio del Algoritmo de Ordenamiento por Mezcla

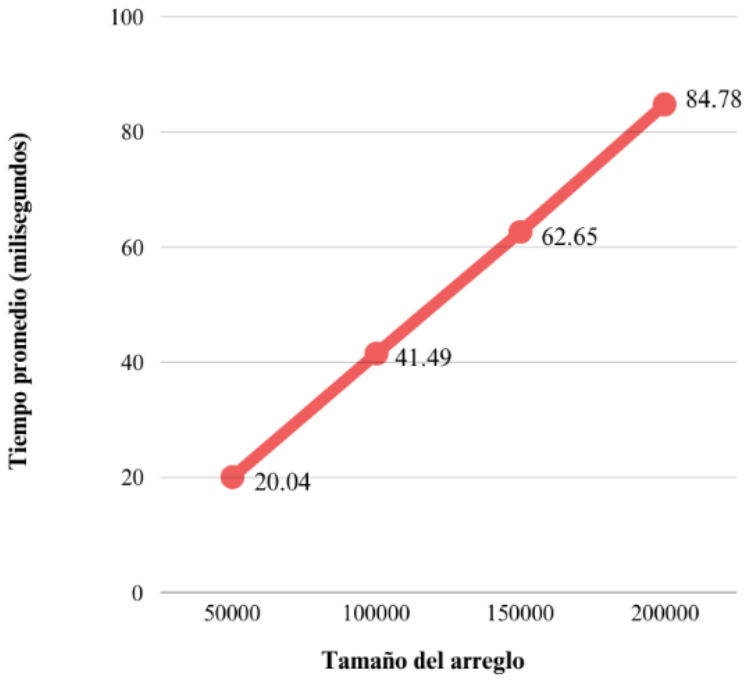


Figura 4

Gráfico de los tiempos de ejecución promedio del Algoritmo de Ordenamiento por Montículos

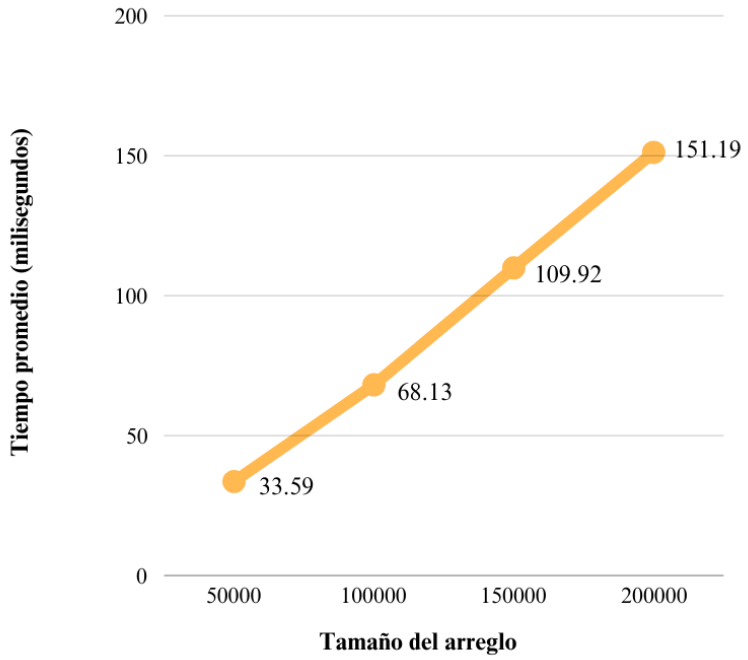


Figura 5

Gráfico de los tiempos de ejecución promedio del Algoritmo de Ordenamiento Rápido

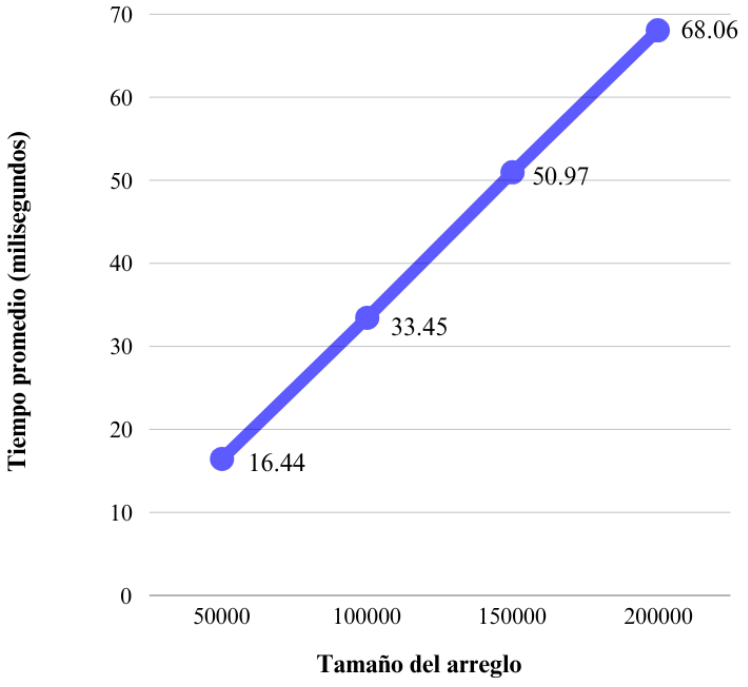


Figura 6

Gráfico de los tiempos de ejecución promedio del Algoritmo de Ordenamiento por Residuos

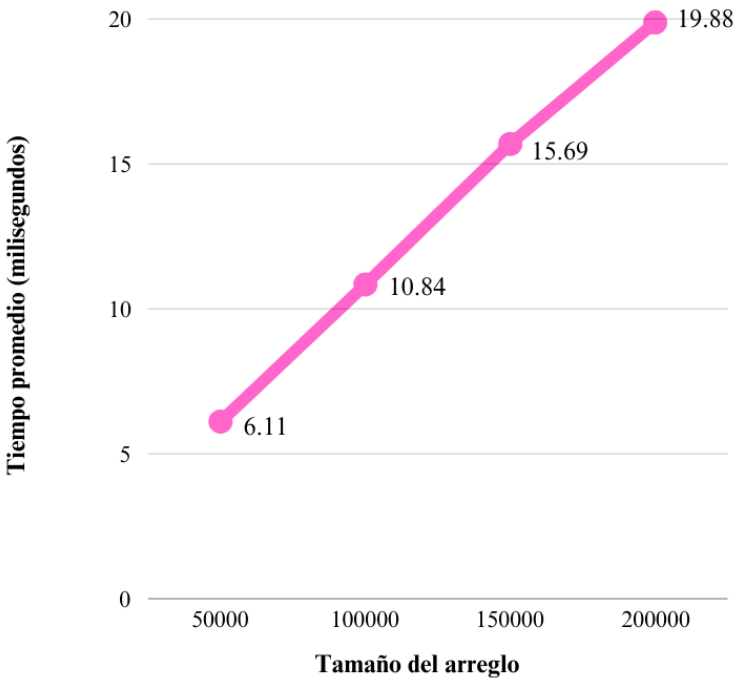


Figura 7

Gráfico comparativo de los tiempos de ejecución promedio del Alg. de Ord. por Selección y el Alg. de Ord. por Inserción

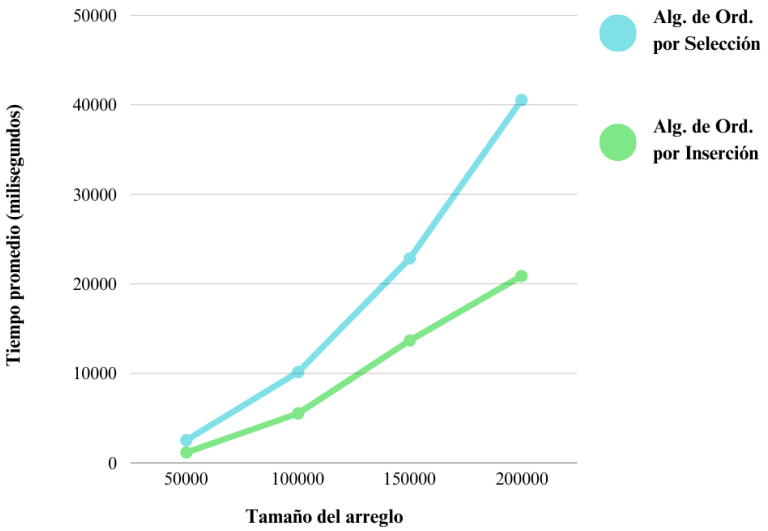
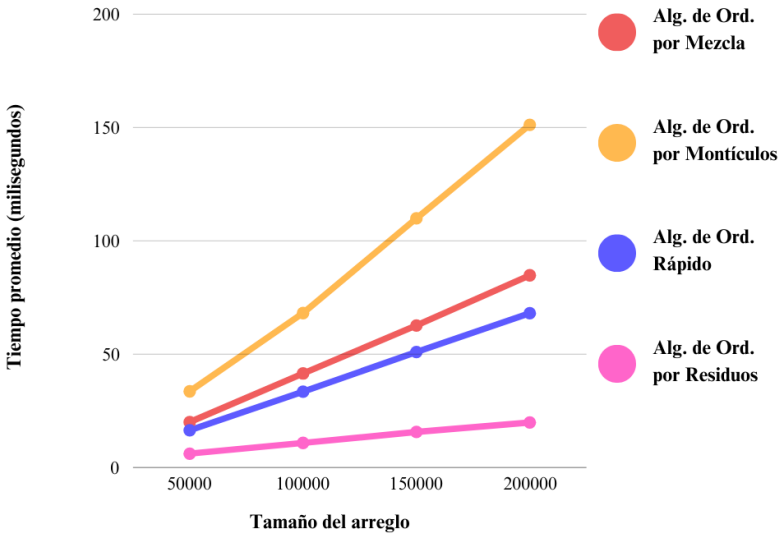


Figura 8

Gráfico comparativo de los tiempos de ejecución promedio del Alg. de Ord. por Mezcla, el Alg. de Ord. por Montículos, el Alg. de Ord. Rápido y el Alg. de Ord. por Residuos



Los resultados de los tiempos de ejecución se obtuvieron por medio de la toma de tiempo antes y después de ejecutar cada corrida de cada algoritmo de ordenamiento, imprimiendo en la consola el tiempo que duró cada corrida, y luego calculando e imprimiendo el promedio de las tres corridas para cada tamaño de arreglo. Posteriormente, todos resultados se registraron en el Cuadro I.

En el Cuadro I, se muestra el nombre de cada algoritmo de ordenamiento, el tamaño del arreglo que se utilizó para cada corrida, el número de corrida y el tiempo promedio de cada conjunto de corridas. Este debe leerse de izquierda a derecha, identificando cada uno de los factores mencionados, y así obtener una comprensión correcta de los resultados.

Como se muestra en los resultados del Cuadro I, el Algoritmo de Ordenamiento por Selección obtuvo tiempos de ejecución prácticamente iguales en las tres corridas de cada tamaño, siendo la mayor diferencia 213.1 ms entre la primera y la última corrida con el arreglo de 200 000 números.

Para el Algoritmo de Ordenamiento por Inserción, los tiempos tienden a variar un poco entre la primera y la última corrida con cada tamaño. Para la mayoría de los arreglos, la primera corrida es más rápida que la última, siendo la diferencia más grande 2283.78 ms con el arreglo de 100 000. Para el

arreglo de 150 000 la última corrida fue más rápida que la primera.

Para el Algoritmo de Ordenamiento por Mezcla, para los arreglos de 100 000 y 150 000, la corrida más diferente es la segunda. En el caso del arreglo de 50 000 la primera corrida es la más rápida y la última la más lenta, y para el arreglo de 200 000 la segunda corrida es la más rápida y la tercera la más lenta.

Para el Algoritmo de Ordenamiento por Montículos, las corridas son bastante consistentes, con diferencias pequeñas entre ellas. La mayor variación ocurre con el arreglo de 150 000 elementos, donde la diferencia más grande entre la primera y la última corrida es de 7.83 ms. Sin embargo, no se muestra ninguna otra diferencia tan significativa con los demás tamaños de arreglos.

En el caso del Algoritmo de Ordenamiento Rápido, los tiempos se mantienen bastante parecidos, con algunas variaciones en los arreglos de 200 000 y 50 000 elementos. Para el primero, la mayor diferencia es de 2.53 ms; y para el segundo, la mayor diferencia es de 2.77 ms, siendo el tamaño de arreglo con la variación más significativa en las corridas.

Para el Algoritmo de Ordenamiento por Residuos, los tiempos de ejecución son bastante parecidos, teniendo una pequeña tendencia a que una de las tres corridas sea más lenta que las otras dos. La mayor variación se da con el arreglo de 150 000 elementos, con una diferencia de 4 ms entre la primera y la segunda corrida.

Las Figuras (1-8) muestran el comportamiento de los algoritmos de forma gráfica, a través de gráficos de líneas, en los que el eje X representa los diferentes tamaños de arreglos, y el eje Y corresponde al tiempo promedio en milisegundos que duró el algoritmo en ordenar el arreglo de valores. Los gráficos (Figuras 1–8) se deben leer de izquierda a derecha, observando la relación entre el tamaño del arreglo y el tiempo promedio que tardó el algoritmo en ordenarlo.

Las Figuras 1-6 son gráficos individuales para cada algoritmo de ordenamiento, lo que permite observar su comportamiento de forma clara y detallada, indicando el tiempo promedio que representa cada punto. Las Figuras 7-8 muestran los gráficos de forma conjunta y comparativa, permitiendo observar las diferencias entre ellos de forma más clara.

IV. DISCUSIÓN

Basándonos en los resultados obtenidos de los experimentos, podemos discutir ciertos puntos clave, relacionando las tendencias de crecimiento prácticas observadas en cada algoritmo de ordenamiento y su comportamiento teórico.

Para el Algoritmo de Ordenamiento por Selección, el comportamiento observado refleja adecuadamente su complejidad temporal cuadrática de $\Theta(n^2)$. Esto es evidenciado por el crecimiento acelerado en el tiempo de ejecución con arreglos más grandes. Los tiempos registrados siguen de manera predecible la estructura cuadrática de su complejidad, ya que cada vez que el algoritmo selecciona el valor más pequeño, no puede evitar volver a comparar todos los demás elementos del subarreglo restante. Debido a que el algoritmo no tiene información previa sobre el orden del arreglo, siempre debe realizar esta búsqueda exhaustiva (Roberts, 2015; OpenGenus IQ, n.d.).

El Algoritmo de Ordenamiento por Inserción mostró un comportamiento cuadrático en la mayoría de los casos, siendo coherente con la teoría de sus casos promedio (complejidad $\Theta(n^2)$). Las variaciones respecto a su comportamiento, las cuales se observan en la forma de su gráfica, pueden explicarse en parte por la naturaleza del algoritmo, que depende del estado inicial del arreglo. De esta forma, si el arreglo está parcialmente ordenado, su comportamiento tiende a ser lineal, siendo coherente con la teoría de su mejor caso (complejidad $\Theta(n)$). En general, su comportamiento es coherente con la teoría (Cormen et al., 2022; Roberts, 2015).

El Algoritmo de Ordenamiento por Mezcla mostró estabilidad en sus tiempos de ejecución, con incrementos suaves y lineales. Esto es coherente con su complejidad $\Theta(n \log n)$, ya que, a diferencia de algoritmos con una complejidad cuadrática, que muestran incrementos más drásticos en tiempo, el Algoritmo de Ordenamiento por Mezcla distribuye la mayor parte de su carga computacional entre las fases de división y mezcla. De esta forma, conforme el tamaño de los arreglos crece, el número de divisiones crece lentamente (logarítmicamente), mientras que el trabajo de mezclar en cada nivel se mantiene proporcional al número total de elementos, resultando en un crecimiento del tiempo más moderado y lineal en comparación con otros algoritmos (Cormen et al., 2022; Roberts, 2015).

El Algoritmo de Ordenamiento por Montículos, al igual que el Algoritmo de Ordenamiento por Mezcla, mostró un comportamiento consistente, con tiempos de ejecución que aumentaron de forma lineal conforme crecía el tamaño del arreglo. Este algoritmo también tiene una complejidad temporal de $\Theta(n \log n)$, y los resultados experimentales fueron coherentes con esta predicción. A pesar de que los montículos pueden tener un overhead adicional debido a las operaciones de

reorganización del heap, los tiempos observados se mantuvieron dentro de los márgenes esperados (Cormen et al., 2022; Roberts, 2015).

Los tiempos de ejecución del Algoritmo de Ordenamiento Rápido reflejan su comportamiento para casos promedio (complejidad $\Theta(n \log n)$), y las pequeñas variaciones entre corridas pueden atribuirse a la sensibilidad en la elección del pivote, que, aunque influye en la eficiencia, no causó grandes divergencias en el comportamiento general. El peor caso de este algoritmo de ordenamiento ocurre cuando el pivote seleccionado es siempre el menor o el mayor elemento del arreglo, lo que resulta en una complejidad cuadrática de $\Theta(n^2)$. No obstante, no parece que esto ocurriera en los experimentos, ya que los tiempos se mantuvieron estables incluso en los arreglos más grandes, lo cual indica que las particiones estuvieron lo suficientemente balanceadas para evitar el peor caso (Cormen et al., 2022; OpenGenus IQ, n.d.).

El Algoritmo de Ordenamiento por Residuos, con su complejidad $\Theta(d(n + k))$, mostró un comportamiento bastante consistente con su teoría. Este algoritmo trabaja particularmente bien con números bien distribuidos, siendo esto lo que se trató de hacer en los arreglos a lo largo de este trabajo. Las variaciones mínimas observadas entre las corridas no parecen afectar su comportamiento, manteniéndose dentro de los límites esperados de acuerdo con su teoría (Cormen et al., 2022; WsCube Tech, n.d.).

En términos generales, todos los algoritmos mostraron una alineación razonable con sus cotas teóricas de complejidad temporal, en especial con las referentes a los casos promedio. Los algoritmos con complejidad cuadrática, como el Algoritmo de Ordenamiento por Selección y el Algoritmo de Ordenamiento por Inserción, mostraron tiempos de ejecución que crecían de manera pronunciada conforme aumentaba el tamaño del arreglo, lo cual concuerda con su complejidad temporal de $\Theta(n^2)$. Por otro lado, los algoritmos con complejidad $\Theta(n \log n)$ y $\Theta(d(n + k))$ tuvieron comportamientos lineales más controlados que no presentaban grandes incrementos de tiempo conforme aumentaba el tamaño del arreglo.

De la misma forma, en los gráficos de las Figuras 1-6, se observa que el Algoritmo de Ordenamiento por Selección y el Algoritmo de Ordenamiento por Inserción muestran una curva similar a una función cuadrática. Por otro lado, el Alg. de Ord. por Mezcla, el Alg. de Ord. por Montículos, el Alg. de Ord. Rápido y el Alg. de Ord. por Residuos muestran un comportamiento similar a una función lineal. Estos comportamientos concuerdan con las estimaciones teóricas de cada algoritmo de ordenamiento.

V. CONCLUSIONES

El análisis de los algoritmos de ordenamiento vistos en este estudio mostró una correlación coherente entre los tiempos de ejecución observados y sus complejidades temporales teóricas. Los algoritmos con complejidad cuadrática, como el Algoritmo de Ordenamiento por Selección y el Algoritmo de Ordenamiento por Inserción, mostraron tiempos de ejecución que aumentaban de manera pronunciada conforme crecía el tamaño del arreglo, validando sus respectivas complejidades $\Theta(n^2)$.

Por otro lado, el Algoritmo de Ordenamiento por Mezcla y el Algoritmo de Ordenamiento por Montículos, con complejidad $\Theta(n \log n)$, presentaron un crecimiento más moderado y predecible en tiempo de ejecución, incluso para arreglos más grandes. En el caso del Ordenamiento Rápido, aunque es sensible a la elección del pivote, su desempeño en promedio también fue consistente con su complejidad $\Theta(n \log n)$. Igualmente, el Ordenamiento por Residuos mostró una ejecución estable, confirmando su facilidad para trabajar con datos distribuidos uniformemente, y siendo consistente con su complejidad $\Theta(d(n + k))$.

Este estudio resalta tanto la importancia de las cotas teóricas, como la relevancia de realizar pruebas empíricas para analizar el comportamiento real de los algoritmos en contextos prácticos. Las cotas teóricas proporcionan una valiosa herramienta para predecir el comportamiento de un algoritmo bajo diferentes escenarios o casos. Sin embargo, en la práctica, los resultados pueden verse afectados por factores adicionales, como el estado inicial de los datos, las características particulares del hardware, y la implementación específica del algoritmo. Por ello, las pruebas empíricas permiten medir cómo estos factores pueden influir en el rendimiento y ayudar a validar las estimaciones teóricas.

Estas pruebas proporcionan una visión más detallada de la capacidad de los algoritmos para manejar datos de diferentes tamaños y distribuciones. Así, la combinación de análisis teórico y pruebas experimentales garantiza una evaluación más completa y precisa de la eficiencia y robustez de los algoritmos en escenarios prácticos.

REFERENCIAS

Cormen, T. et al. (2022). *Introduction to algorithms (4th ed.)*. MIT Press.

Roberts, E. (2015). *Sorting and Efficiency*. https://cs.stanford.edu/people/eroberts/courses/cs106b/handouts/20-SortingAndEfficiency.pdf

OpenGenus IQ. (n.d.). *Time & Space Complexity of Selection Sort*. https://iq.opengenus.org/time-complexity-of-selection-sort/

WsCube Tech. (n.d.). *Radix Sort: Algorithm, Time Complexity, Code, Example*. https://www.wscubetech.com/resources/dsa/radix-sort

W3Schools. (n.d.a). *DSA Selection Sort*. https://www.w3schools.com/dsa/dsa_algo_selectionsort.php

W3Schools. (n.d.b). *DSA Insertion Sort*. https://www.w3schools.com/dsa/dsa_algo_insertionsort.php

W3Schools. (n.d.c). *DSA Merge Sort*. https://www.w3schools.com/dsa/dsa_algo_mergesort.php

W3Schools. (n.d.d). *DSA Quicksort*. https://www.w3schools.com/dsa/dsa_algo_quicksort.php

W3Schools. (n.d.e). *DSA Radix Sort*. https://www.w3schools.com/dsa/dsa_algo_radixsort.php