Reporte de Tarea 1

Liqing Yosery Zheng Lu, C38680, liqing.zheng@ucr.ac.cr

***Resumen*—El presente trabajo aborda la comparación entre los tiempos de ejecución teóricos y reales de diversos algoritmos de ordenamiento. Con el objetivo de evaluar cómo se ajustan las predicciones teóricas a mediciones prácticas, se implementaron en C++ seis algoritmos: selección, inserción, mezcla, montículos, rápido (quicksort) y residuos (radix sort). Para ello, se generaron arreglos aleatorios de** 1 000**,** 10 000**,** 100 000 **y** 1 000 000 **y se midió el tiempo de ejecución promedio en tres corridas mediante la biblioteca `chrono`. Los resultados muestran que los métodos cuadráticos (selección e inserción) crecen según lo esperado con complejidad $O(n^2)$, mientras que los algoritmos $O(n \log n)$ (mezcla, montículos y rápido) requieren tamaños superiores a 10^6 para evidenciar plenamente su ventaja. El ordenamiento por residuos, con complejidad $O(d\,(n+k))$, demostró ser el más rápido en los rangos analizados. En conclusión, se confirma la validez de la teoría de complejidad presentada en el libro *Introducción a los algoritmos*[1].**

***Palabras clave*—ordenamiento, selección, inserción, mezcla, montículos, rápido, residuos**

I. Introducción

El objetivo principal de este trabajo es comparar la duración de distintos algoritmos de ordenamiento con sus complejidades teóricas, utilizando arreglos de diferentes tamaños representativos. Como parte del análisis, se busca observar cómo escalan en la práctica algoritmos con diferentes órdenes de complejidad, en especial aquellos con crecimiento cuadrático frente a los que presentan crecimiento log-lineal. Esto permite evaluar hasta qué punto las predicciones teóricas coinciden con los tiempos de ejecución medidos experimentalmente y comprender mejor el impacto práctico de la complejidad algorítmica en diferentes contextos.

Se separan los algoritmos en dos categorías: los considerados lentos, como el ordenamiento por selección e inserción[1], cuyo comportamiento general es de $O(n^2)$, una complejidad que limita su escalabilidad y restringe su uso en colecciones de datos grandes. En contraste, los considerados aceptables, como el ordenamiento por mezcla, montículos y rápido (quicksort)[2], presentan una complejidad teórica de $O(n \log n)$, lo cual permite una mejor eficiencia en volúmenes de datos mayores. Analizar ambos grupos resulta clave para entender las diferencias prácticas entre algoritmos de orden $O(n^2)$ y $O(n \log n)$.

Adicionalmente, se considerará el algoritmo de ordenamiento por residuos (radix sort), cuya complejidad es de $O(d \cdot (n + k))$, donde d representa la cantidad de dígitos del mayor número a ordenar y k el rango de los dígitos posibles. Este método resulta particularmente interesante, dado que su desempeño depende más de las características de los datos que de su cantidad.

[1]La complejidad temporal del ordenamiento por inserción es teóricamente $O(n)$ en el mejor caso, sin embargo, su peor caso y su promedio son $O(n^2)$.

[2]El peor caso de quicksort es $O(n^2)$, y ocurre cuando las particiones no se escogen adecuadamente.

II. Metodología

Con el propósito de comparar la duración de distintos algoritmos de ordenamiento con sus complejidades teóricas expresadas mediante notación asintótica, se crean cuatro arreglos de prueba con tamaños representativos: 1 000, 10 000, 100 000 y 1 000 000 elementos. Cada uno contiene números generados aleatoriamente de 32 bits, es decir, valores en el rango de 0 a 2^{32} inclusive.

Los algoritmos seleccionados se implementan en C++ respetando las estrategias tradicionales descritas en la literatura. Los métodos de selección, inserción y residuos se programan de forma iterativa, siguiendo las directrices del libro *Introducción a los algoritmos*[1]. Por su parte, los algoritmos de mezcla, montículos y rápido (quicksort) se desarrollan como funciones recursivas, siguiendo el mismo marco teórico.

El algoritmo de ordenamiento rápido (quicksort) incorpora la técnica de la mediana de tres para elegir el pivote, comparando el primer, último y elemento central del arreglo, y seleccionando el valor intermedio. El algoritmo de ordenamiento por residuos (radix sort) se implementa usando *counting sort* como método estable para ordenar cada dígito.

Para asegurar una comparación equitativa, cada algoritmo ordena una copia idéntica de los arreglos generados. Las pruebas de rendimiento se realizan midiendo el tiempo de ejecución mediante la biblioteca `chrono`, registrando las duraciones en microsegundos.

Los tiempos obtenidos se convierten a milisegundos para facilitar la presentación del dato. Los resultados se documentan en una tabla y se representan gráficamente mediante líneas que muestran la relación entre el tamaño del arreglo y el tiempo de ejecución para cada algoritmo. También se genera un gráfico comparativo con todas las curvas superpuestas, permitiendo visualizar las diferencias de escalabilidad y contrastarlas con las predicciones teóricas.

Finalmente, los datos recolectados se analizan en la discusión para identificar cómo se comportan los algoritmos en relación con sus complejidades esperadas, y qué factores pueden explicar las posibles discrepancias entre teoría y práctica.

III. Resultados

Los tiempos de ejecución de las tres corridas de los algoritmos se muestran en el cuadro I.

- **Selección:** Para los arreglos de 1 000 y 10 000 elementos, la primera y la segunda corrida mostraron duraciones prácticamente idénticas. En el caso de 100 000 elementos, la segunda corrida fue notablemente más rápida que la primera. La tercera corrida tuvo un ligero aumento

en los tiempos de todos los tamaños, pero sin alterar significativamente el valor promedio, que quedó muy parecido al de la primera ejecución.

- **Inserción:** Las tres corridas dieron resultados muy similares en todos los tamaños. Solo en el arreglo de 1 000 000 elementos se observó un incremento de aproximadamente dos segundos en la tercera corrida. Debido a la homogeneidad de los tiempos, el promedio se parece a los datos.
- **Mezcla:** En los tamaños de 1 000 y 10 000 elementos, las tres ejecuciones arrojan exactamente el mismo tiempo. Para 1 000 000 elementos, cada corrida mostró un ligero aumento progresivo, aunque el promedio se mantiene muy cercano al tiempo registrado en la segunda corrida.
- **Montículos:** En el arreglo de 1 000 elementos, la primera corrida fue algo más lenta que las dos siguientes. Para 10 000 elementos, la segunda corrida resultó un milisegundo más rápida que las demás. En el caso de 1 000 000 elementos, la segunda ejecución fue notablemente más veloz, por lo que el promedio final se ve influenciado y no parecido a ninguno de los datos.
- **Rápido:** Los tiempos para 1 000 y 10 000 elementos fueron consistentes entre las tres corridas. En 1 000 000 elementos, la primera corrida tuvo la mayor duración y la segunda la menor, dejando el promedio muy cercano al resultado de la tercera ejecución.
- **Radix Sort (residuos):** Este algoritmo fue el más rápido en todos los tamaños analizados. Las tres corridas presentaron variaciones mínimas, de modo que el promedio coincide prácticamente con cada medición individual.

Cuadro I
TIEMPO DE EJECUCIÓN DE LOS ALGORITMOS.

		Tiempo (ms)			
		Corrida			
Algoritmo	Tam. (k)	1	2	3	Prom.
Selección	1000	0,8	0,8	1	0,9
	10000	86	84	90	86,7
	100000	11029	8255	11206	10163,3
	1000000	890898	883981	918631	Ne
Inserción	1000	0,4	0,4	0,5	0,43
	10000	47	45	44	45,3
	100000	4405	4310	4436	4383,6
	1000000	487428	474615	507112	489718,3
Mezcla	1000	0,1	0,1	0,1	0,1
	10000	1	1	1	1
	100000	16	17	15	16
	1000000	165	174	181	173,3
Montículos	1000	0,2	0,1	0,1	0,13
	10000	1	2	2	1,6
	100000	30	28	32	30
	1000000	378	354	375	369
Rápido	1000	0,1	0,1	0,2	0,13
	10000	2	2	2	2
	100000	30	29	28	29
	1000000	360	301	334	331,7
Residuos	1000	0,1	0,09	0,09	0,093
	10000	1	0,9	0,9	0,93
	100000	12	12	10	11,3
	1000000	109	106	109	108

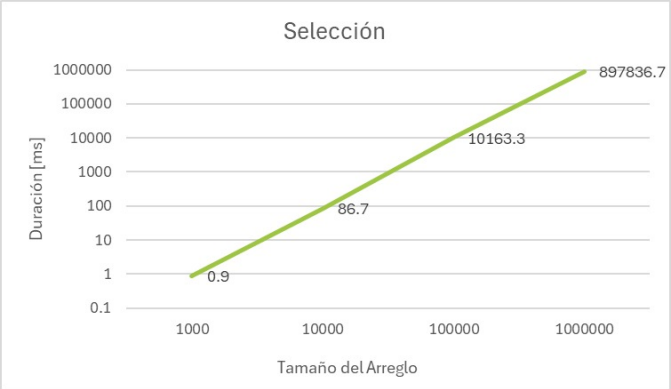

Figura 1. Gráfico del tiempo promedio de ejecución del algoritmo por selección.

Figura 2. Gráfico del tiempo promedio de ejecución del algoritmo de ordenamiento por inserción.

Los tiempos promedio de todos los algoritmos de ordenamiento estudiados en este proyecto se muestran gráficamente en la figura 7.

En la figura 1 y figura 2, que son las curvas del ordenamiento de selección e inserción respectivamente, la curva se comporta de una manera exponencial, lo cual está de acuerdo con su complejidad $O(n^2)$.

En las figuras 3 y 4, que muestran los algoritmos de mezcla y montículos, las curvas inicialmente aparentan un comportamiento cercano al cuadrático. Sin embargo, a partir de $n = 100\,000$ elementos se nota un crecimiento más lento, en línea con su complejidad $O(n \log n)$.

En la figura 5 se puede observar que el algoritmo de ordenamiento rápido sí crece más lento que la función exponencial, por lo tanto sí cumple con lo esperado porque es un algoritmo de complejidad $O(n \log n)$.

Por su parte, la curva del algoritmo por residuos que se observa en la figura 6 parece comportarse como una función exponencial, pero es de complejidad $O(d \cdot (n + k))$. Por lo tanto, se esperaba una curva con un crecimiento menor.

Al comparar todas las curvas en la figura 7, es más notable la diferencia en crecimiento de los algoritmos $O(n^2)$ (selec-

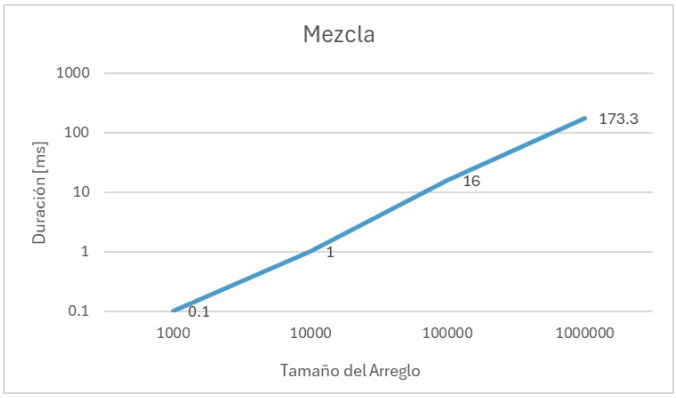

Figura 3. Gráfico del tiempo promedio de ejecución del algoritmo de ordenamiento por mezcla.

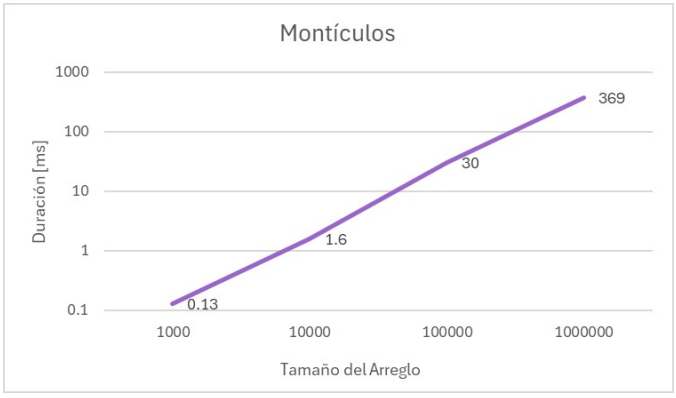

Figura 4. Gráfico del tiempo promedio de ejecución del algoritmo por montículos.

ción, inserción) con los de $O(n \log n)$ (mezcla, montículos, rápido) y el de $O(d \cdot (n+k))$ (residuos). El orden de crecimiento observado es del más rapido al más lento: residuos, mezcla, rápido, montículos, inserción y selección.

IV. Discusión

Como primer punto, con base en los resultados presentados, se puede observar que los algoritmos de ordenamiento por

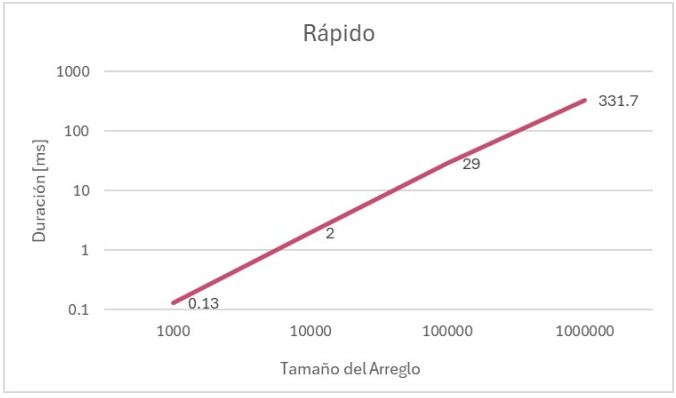

Figura 5. Gráfico del tiempo promedio de ejecución del algoritmo de ordenamiento rápido.

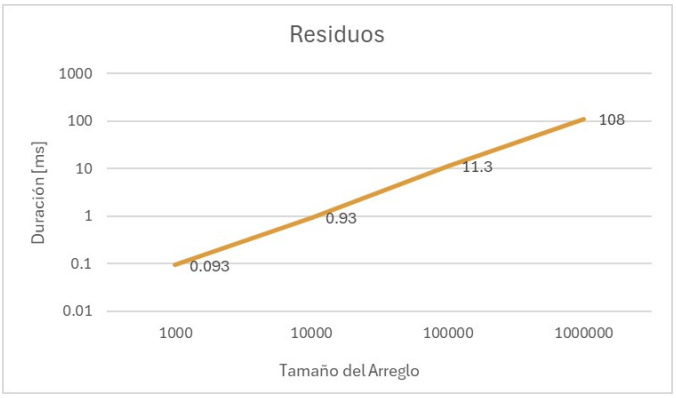

Figura 6. Gráfico del tiempo promedio de ejecución del algoritmo de ordenamiento por residuos.

mezcla, montículos y residuos no mostraron el comportamiento óptimo esperado de un algoritmo de complejidad inferior a $O(n^2)$. Lo anterior se podría deber a que los arreglos utilizados no fueron suficientemente grandes para reflejar la diferencia entre un algoritmo de complejidad $O(n^2)$ con uno de $O(n \log n)$ o menor. Sin embargo, sí se aprecia en la figura 7 que estas tres curvas permanecen consistentemente por debajo de las de selección e inserción, confirmando que su orden de crecimiento es efectivamente menor al cuadrático.

Adicionalmente, los experimentos realizados revelan una variabilidad significativa en las mediciones individuales, especialmente para el ordenamiento rápido y montículos. Esta dispersión puede atribuirse a la sobrecarga de llamadas recursivas, la gestión de memoria dinámica y al propio comportamiento del sistema operativo al asignar recursos. Por ejemplo, la técnica de mediana de tres en ordenamiento rápido redujo la frecuencia del peor caso, pero no eliminó por completo las fluctuaciones de tiempo.

Por otra parte, el análisis de la tabla I muestra que, en entornos de datos muy grandes ($n \geq 10^5$), radix sort supera a los algoritmos $O(n \log n)$ siempre que el rango de valores (k) y la cantidad de dígitos (d) se mantengan moderados. Sin embargo, cuando k y d aumentan significativamente, su coste adicional puede aproximarse al de un algoritmo logarítmico, reduciendo su ventaja práctica. De este modo, la elección del algoritmo más adecuado debe basarse no solo en la complejidad asintótica, sino también en las características específicas de los datos y el entorno de ejecución.

V. Conclusiones

Retomando los objetivos de este trabajo, se puede concluir que:

- Los algoritmos de ordenamiento con complejidad $O(n^2)$ (selección e inserción) resultan ineficientes para arreglos grandes, confirmando del libro *Introducción a los algoritmos*[1] que solo deben ser usados cuando la simplicidad de implementación sea prioritaria.
- Los algoritmos de ordenamiento con complejidad $O(n \log n)$ (mezcla, montículos y rápido) ofrecen un rendimiento sustancialmente mejor que los $O(n^2)$ a partir de tamaños intermedios ($n \geq 10^4$), aunque para reflejar

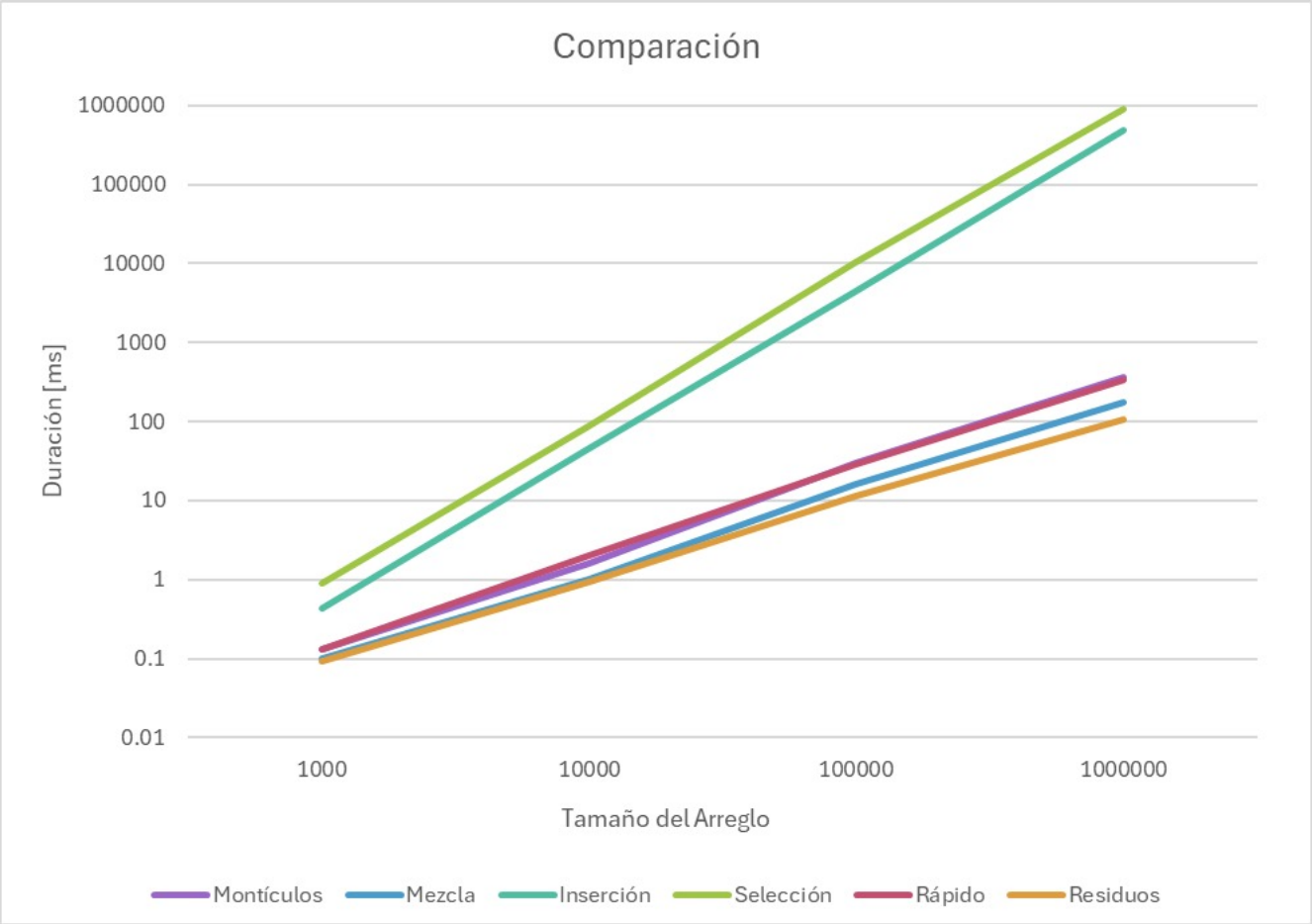

Figura 7. Gráfico de tiempos promedio de ejecución de los algoritmos de ordenamiento por selección, inserción, mezcla, montículos, rápido y residuos.

plenamente su ventaja se requieren volúmenes de datos mayores a 10^6 elementos. Para futuras investigaciones, sería interesante ampliar el rango de prueba a tamaños aún mayores y/o incorporar variantes que aprovechen la arquitectura de múltiples núcleos.

- Las mediciones experimentales presentan variabilidad atribuible a factores de sistema y a la gestión de memoria. Lo que reafirma la necesidad de realizar múltiples corridas y reportar promedios junto con desvíos estándar para comparaciones rigurosas en futuras investigaciones.

En conjunto, este proyecto confirma la validez de la teoría de complejidad algorítmica explicado en el libro *Introducción a los algoritmos*[1].

Referencias

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

Liqing Yosery Zheng Lu Estudiante de computación en la Universidad de Costa Rica. Lleva el curso de Algoritmos y Estructuras de Datos durante el primer ciclo del año 2025 en la sede Rodrigo Facio.