Informe Final Análisis de Algoritmos de Ordenamiento

José Antonio Mora M C15114

Resumen- En este trabajo se estudiaron y compararon los tiempos, cotas y curvas que forman los principales algoritmos de ordenamiento, para verificar su eficiencia y la relación de la teoría con la práctica. Para esto, se implementaron los 6 algoritmos más conocidos de ordenamiento, siendo estos ordenamiento por selección, inserción, mezcla, montículos, rápido y residuos, en el lenguaje C++, y se promediaron los tiempos que cada uno duró para arreglos de distintos tamaños, para luego graficar estos tiempos de cada algoritmo. El resultado fue igual a la teoría, en donde los algoritmos iterativos con cota ajustada Θ(n^2), como selección o inserción, duraron más tiempo, y formaron una curva parabólica, por su naturaleza cuadrática, mientras que los algoritmos recursivos acotados por Θ($n \log n$), como Montículos, Mezcla y Rápido, duraron mucho menos, y sus tiempos formaron una curva casi lineal, por su naturaleza logarítmica, pero el algoritmo con menor duración de todos, fue el de Residuos al utilizar la base óptima de $2^{\lfloor \log n \rfloor}$, que alcanza una cota de Θ(n), y formaron una curva lineal, debido a su naturaleza lineal. Al comparar sus tiempos en escala logarítmica, se puede apreciar la enorme diferencia que existe entre los algoritmos con cota Θ(n^2) y los demás, y al mismo tiempo, se nota una pequeña diferencia entre algoritmos de la misma naturaleza, debido a diferencias más técnicas como manejo de memoria, cantidad de comparaciones y ejecuciones que difieren entre algoritmos. Se concluye que la práctica cumple con lo establecido en la teoría, por lo que en la mayoría de los casos cuando se necesita implementar un algoritmo de ordenamiento, es más eficiente un algoritmo Θ($n \log n$) o Θ(n), debido a que, aunque existe un algoritmo claramente más rápido, las diferencias temporales entre estos es tan mínima, que se vuelve prácticamente imperceptible, y se vuelven más importantes factores secundarios como eficiencia espacial, manejo de memoria y caché, etc. a la hora de determinar qué algoritmo implementar, ya que depende de las condiciones y recursos disponibles en cada caso específico.

I. Introducción

En este trabajo se estudiaron los tiempos que toma cada uno de los algoritmos de ordenamiento más famosos en existencia, para verificar cuáles son los más eficientes, y lograr identificar las principales diferencias entre ellos. Por esto mismo, para este Informe Final se tomaron en cuenta los algoritmos de: Selección, Inserción, Mezcla, Montículos, Rápido y Residuos. La finalidad del trabajo es comparar los tiempos que toma cada algoritmo, entre cada corrida y entre ellos, y graficar sus tiempos para comparar sus curvas, cotas ajustadas, y comportamientos dependiendo del tamaño del arreglo, y así lograr concluir si la teoría de estos es verídica o se aproxima en la práctica real, y de este

mismo modo, conseguir encontrar el algoritmo más eficiente dependiendo del tamaño de elementos a ordenar.

II. Metodología

Para lograr lo propuesto, se implementaron cada uno de los algoritmos utilizando el lenguaje de programación C++, para así automatizar el proceso, debido a que se utilizaron arreglos o conjuntos de elementos muy grandes, de 50 000, 100 000, 150 000 y 200 000 números, por lo que hacer un análisis manual tomaría mucho más tiempo. El código se encuentra en el archivo Ordenador.h adjunto al documento, y está basado en el pseudocódigo del libro de Cormen y colaboradores [1]. Los números utilizados para los arreglos fueron generados aleatoriamente, negativos y positivos, entre el rango de -2147483648 y 2147483647, que equivalen al límite inferior y superior de la variable computacional entero “int”, generando así una muestra importante con la que se pueden calcular resultados significativos para el estudio. Cada uno de los algoritmos fue ejecutado un total de 3 veces en el mismo equipo, para así poder calcular el tiempo promedio que toma cada algoritmo, y minimizar variaciones significativas externas ocurridas durante la ejecución de estos. El equipo utilizado cuenta con 16 GB de RAM, un i7 de 8 núcleos, y la ejecución de los algoritmos se dio en el Windows Subsystem For Linux emulando la distribución Ubuntu, dentro del sistema operativo Windows 11. La herramienta utilizada para recolectar la duración de los algoritmos fue el comando time de Linux, que tiene como funcionalidad medir en minutos, segundos y milisegundos la duración de una tarea específica. Después de esto se promediaron los tiempos de ejecución de los algoritmos y se graficaron para comparar sus diferencias.

III. Resultados

Los resultados iniciales pueden ser resumidos en el siguiente cuadro:

Cuadro I

Tiempo de ejecución de los algoritmos

(Números representan la corrida. Tiempo dado en milisegundos)

Algoritmo	Tamaño (k)	1	2	3	Promedio
Selección	50	2251	2267	2233	2250.33
	100	8871	8868	8959	8899.33
	150	20126	19790	19799	19905
	200	35512	35457	35813	35594
Inserción	50	1282	1273	1269	1274.66
	100	5113	5084	5082	5093
	150	11328	11262	11438	11342.66
	200	20127	20303	20152	20194
Mezcla	50	27	26	25	26
	100	49	46	47	47.33
	150	67	68	69	68
	200	90	88	86	88
Montículos	50	28	27	26	27
	100	52	49	53	51.33
	150	75	77	76	76
	200	99	101	100	100
Rápido	50	22	21	23	22
	100	39	41	40	40
	150	59	63	60	60.66
	200	83	78	79	80
Residuos	50	23	21	22	22
	100	39	38	41	39.33
	150	55	56	57	56
	200	75	72	73	73.3

Los tiempos de ejecución de las 3 corridas de los algoritmos se muestran en el Cuadro I, en donde se puede observar cómo los algoritmos con $\Theta(n^2)$ son los más lentos de todos, siendo los únicos en demorar segundos. Entrando en detalle, se puede observar como el algoritmo por selección, que encuentra el menor elemento en cada llamado y lo ordena, es el más ineficiente con respecto al tiempo, con un tiempo promedio en el rango de 2.25 segundos hasta 35.59 segundos dependiendo del tamaño del arreglo. El penúltimo algoritmo en posicionarse en términos de eficiencia es el de inserción, que comprueba si el siguiente elemento es mayor que sí mismo y lo coloca en su posición correspondiente, y que a pesar de que en teoría tiene una naturaleza similar al de selección, demostró ser más eficiente, con un tiempo promedio en el rango de 1.27 segundos a 20.19 segundos, dependiendo nuevamente del tamaño del arreglo, debido a que como respalda la teoría, el algoritmo de Inserción necesita hacer menos comparaciones entre elementos para ordenar el arreglo, otorgándole una ventaja de tiempo.

Las siguientes posiciones en términos de eficiencia temporal corresponden a los algoritmos $\Theta(n \log n)$, algoritmos que se diferencian por ser recursivos y por lo tanto duran

menos de un segundo sin importar el tamaño del arreglo utilizado. Clasificados individualmente, se observa como en la cuarta posición se encuentra el algoritmo por montículos, que utiliza montículos para encontrar el elemento mayor en cada pasada y lo coloca en su posición correspondiente, con un rango de duración entre 0.027 y 0.100 segundos, y que a pesar de tener la misma naturaleza que los demás algoritmos con su misma cota ajustada, debido a un tecnicismo computacional, siendo la carga ineficiente y en mayores cantidades de memoria al caché del procesador, suele tener tiempos ligeramente mayores, tal como indica la teoría. En tercera posición se encuentra el algoritmo de mezcla, que utiliza la técnica de ”Divide y vencerás” para dividir y ordenar un arreglo en subarreglos, con un tiempo promedio de 0.026 segundos a 0.088 segundos. En segunda posición se encuentra el algoritmo de ordenamiento rápido, con un rango de duración de 0.022 segundos a 0.080 segundos, dependiendo del tamaño del arreglo. El algoritmo de ordenamiento rápido presenta los mejores tiempos, aunque en su caso promedio sea de la misma naturaleza que los demás con cota de $\Theta(n \log n)$, y en su peor caso siendo peor debido a su cota de $\Theta(n^2)$, gracias a que, como dice la teoría, tiende a leer datos de manera secuencial, por lo que su carga de memoria a la caché del procesador es menor y más eficiente, dándole una leve ventaja con respecto al tiempo tanto en la teoría como la práctica.

Finalmente, en la primera posición, se encuentra el ordenamiento por residuos, que ordena a los elementos haciendo varias pasadas, y en cada una ordena al dígito menos significativo, o grupos de bits específicos, hasta terminar. Este algoritmo, al utilizar la base óptima de $2^{\lfloor \log n \rfloor}$, consigue una cota ajustada teórica de $\Theta(n)$, convirtiéndolo en el más rápido tanto en la teoría como en la práctica, debido a sus tiempos promedio de 0.022 a 0.073 segundos, aunque al necesitar memoria extra para ejecutar estas pasadas, suele ser bastante más ineficiente espacialmente que algoritmos con cota $E(1)$, como ordenamiento rápido.

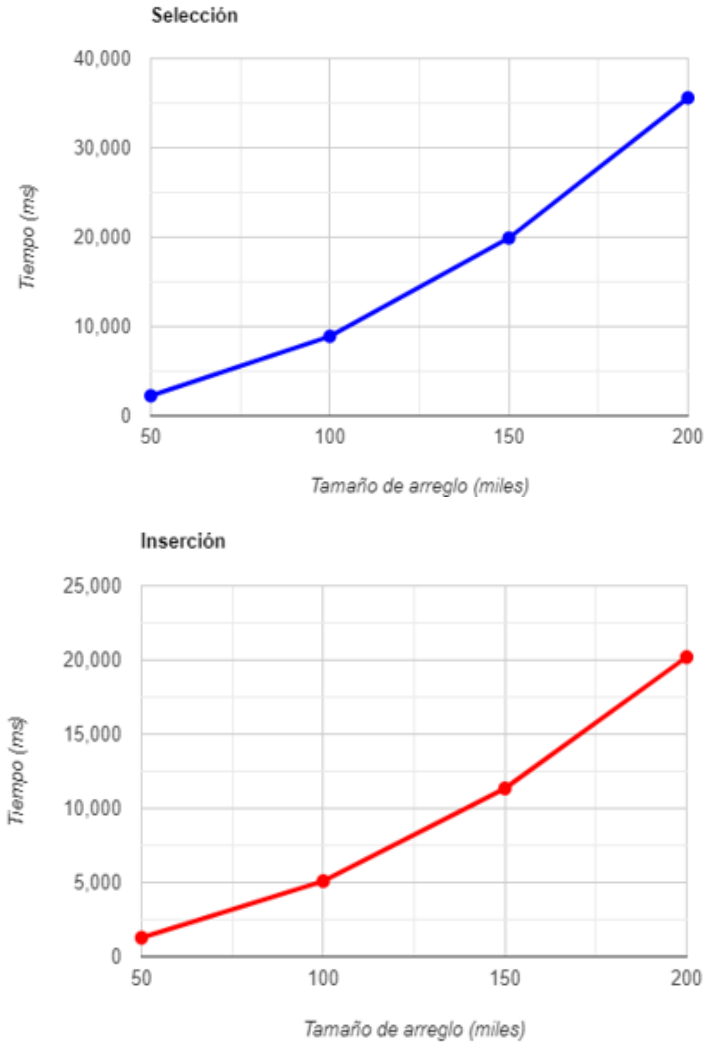
Como se puede comprobar nuevamente en el Cuadro I, los tiempos entre cada corrida no son tan diferentes entre ellos, pero tampoco son iguales, lo que indica que existe cierta variación entre cada corrida, ya sea por variantes externas ya mencionadas, o por diferencias existentes entre cada arreglo que ordena el algoritmo. De todos modos, la diferencia entre ellos no es lo suficientemente significativa para causar una diferencia importante, y esto se puede apreciar en los promedios, en donde las diferencias son de pocos milisegundos, por lo que se vuelven casi imperceptibles.

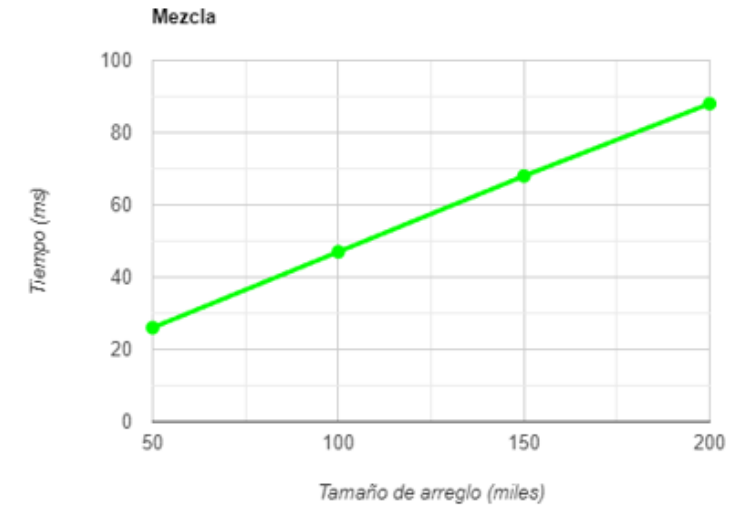
Los tiempos promedio se muestran gráficamente en la figura 1, en donde se puede comprobar cómo los algoritmos siguen las formas de curva esperadas en la teoría, debido a que los algoritmos de Selección e Inserción forman una curva parabólica, no lineal, debido a su naturaleza cuadrática, mientras que los algoritmos con cota ajustada de $\Theta(n \log n)$ como mezcla, montículos y rápido forman una curva prácticamente lineal, gracias a su naturaleza logarítmica. Finalmente se puede apreciar como el algoritmo de Residuos, al ser $\Theta(n)$, su curva es lineal, debido a su naturaleza de algoritmo lineal,

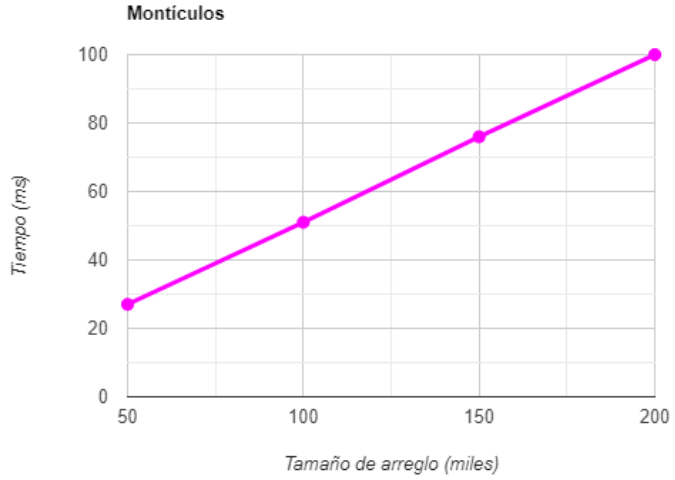
aunque la diferencia es mínima y casi imperceptible en comparación con los algoritmos $\Theta(n \log n)$.

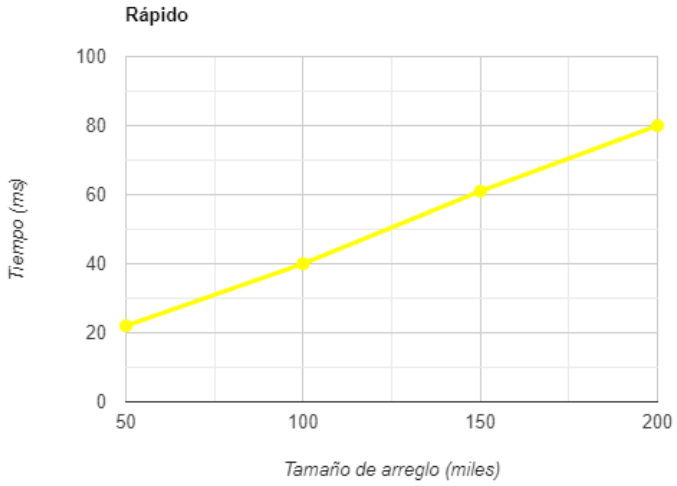
Las curvas se muestran de forma conjunta en la figura 2, en donde se puede apreciar ya con un gráfico en escala de tiempo logarítmico, las grandes diferencias que existen entre los algoritmos de Selección e Inserción en comparación con los demás, como lo demuestra la teoría, ya que se pasa de ordenar arreglos grandes en segundos, a apenas unos milisegundos. A menor medida, igualmente se puede llegar a apreciar algunas diferencias de tiempo entre algoritmos de la misma naturaleza, como Inserción y Selección, o Residuos con Montículos, donde sus diferencias no son tan grandes, pero sí apreciables, influenciados por factores ya mencionados como un mejor uso de la memoria caché, cantidad de comparaciones realizadas, y carga de memoria del sistema.

Figura 1. Tiempos promedio de ejecución de los algoritmos de ordenamiento por Selección, Inserción, Mezcla, Montículos, Rápido y Residuos.




Mezcla
100
80
60
40
20
0
Tiempo (ms)
50
100
150
200
Tamaño de arreglo (miles)


Montículos
100
80
60
40
20
0
Tiempo (ms)
50
100
150
200
Tamaño de arreglo (miles)


Rápido
100
80
60
40
20
0
Tiempo (ms)
50
100
150
200
Tamaño de arreglo (miles)

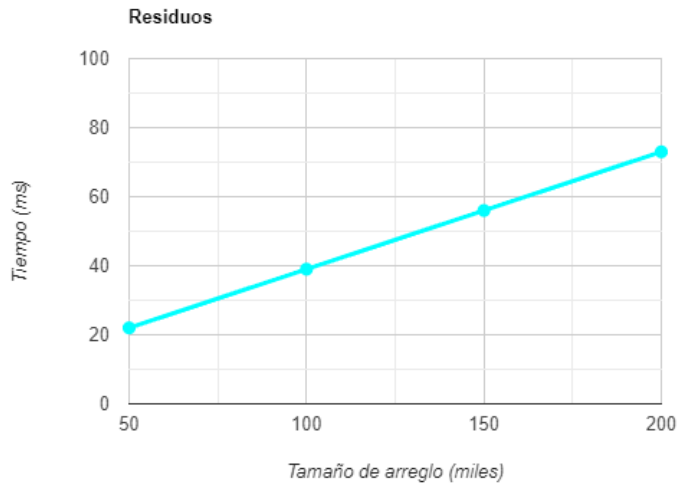
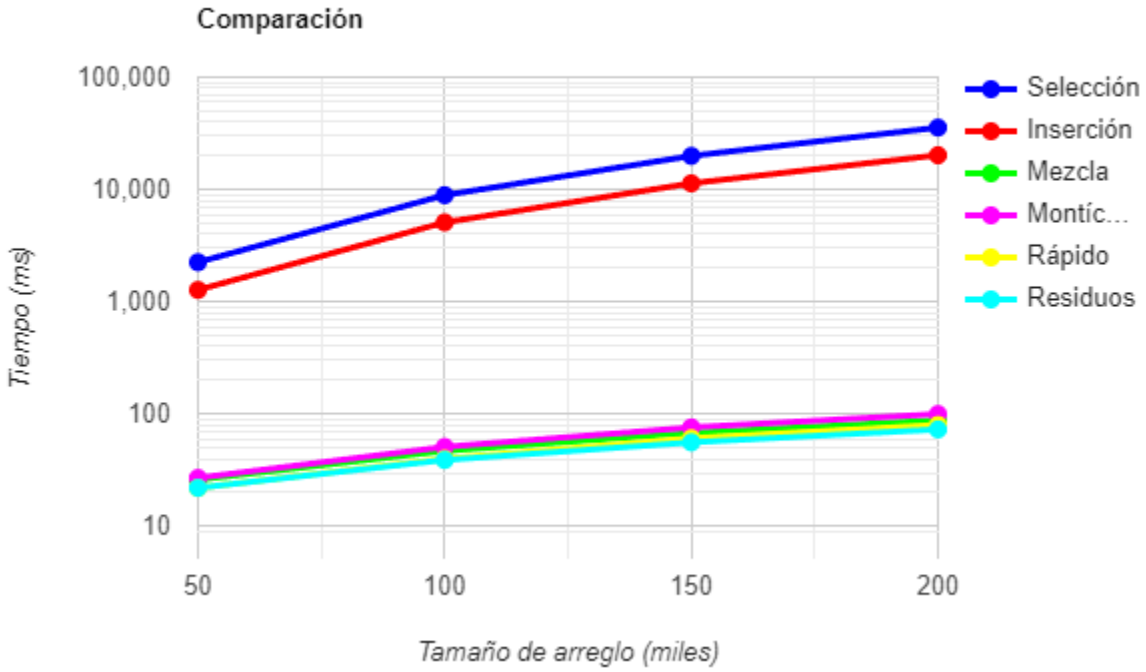


Figura 2. Gráfico comparativo de los tiempos promedio de ejecución de algoritmos de ordenamiento.



IV. Conclusiones

A partir de los resultados se concluye que, tanto en la teoría como en la práctica, los algoritmos iterativos de ordenamiento $\Theta(n^2)$, como Selección e Inserción, son considerablemente menos eficientes, al menos con respecto al tiempo, en arreglos con grandes cantidades de elementos, debido a su comportamiento parabólico mientras aumenta la cantidad de elementos a ordenar, a diferencia de los algoritmos de ordenamiento recursivos $\Theta(n \log n)$ como Montículos, Mezcla y Rápido, que tienen un comportamiento casi lineal, debido a su naturaleza logarítmica, o el algoritmo de ordenamiento por residuos, que al utilizar la base óptima de $2^{\lfloor \log n \rfloor}$, mantiene una cota de $\Theta(n)$, convirtiéndolo en el único con naturaleza lineal, y por lo tanto, en el más rápido. Eso sí, cabe recalcar que la diferencia entre los algoritmos con cota $\Theta(n \log n)$ y $\Theta(n)$ es mínima, debido a que al $\Theta(n \log n)$ no ser polinomialmente más grande que $\Theta(n)$, las diferencias son casi insignificantes, de apenas pocos milisegundos, e igualmente ocurre lo mismo al comparar algoritmos de la misma naturaleza, por lo que la diferencia no es notoria en la mayoría de los casos, y es mejor en esta situación tomar en cuenta otros factores como el uso del caché del procesador o la eficiencia espacial para escoger cuál implementar dependiendo de los recursos y condiciones específicas en cada caso. Si se desea tomar en cuenta solamente la eficiencia temporal, si se va a trabajar con grandes cantidades de datos o no se conoce el tamaño de datos con el que se trabajará, en la mayoría de las ocasiones va a resultar mejor la implementación de un algoritmo logarítmico, como Montículos, Mezcla o Rápido, siendo Rápido el mejor de estos, y si se desea tomar en cuenta diferencias temporales sin importar qué tan pequeñas sean, el algoritmo de ordenamiento por Residuos, debido a su naturaleza lineal, resulta el más rápido de todos, como lo demuestra tanto la teoría como la práctica.

Referencias

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