EL LORO HUASTECO
Órgano de Divulgación Científica y Tecnológica
Tecnológico Nacional de México, Campus Pánuco
Memorias del Congreso Multidisciplinario Internacional 2022

Implementación de algoritmos de ordenamiento para comparar su complejidad algorítmica

Ángel Mario Lerma Sánchez[1], Felipe Anastasio González González[1], Joaquín Torres Mata[1], Yosshio Del Ángel Zapata[1]*
[1]Universidad Autónoma de Tamaulipas, Unidad Académica Multidisciplinaria Mante
*amlerma@docentes.uat.edu.mx

RESUMEN

El presente trabajo tiene como objetivo describir el proceso de la implementación de algoritmos de ordenamiento en el lenguaje de programación Java como un ejercicio didáctico para comparar su complejidad algorítmica y que los estudiantes tengan una mejor comprensión de los fundamentos teóricos de dicha complejidad al experimentar el impacto que tiene en algoritmos de ordenamiento seleccionados. Para el desarrollo del presente trabajo se investigaron las diversas complejidades, así como algunos algoritmos de ordenamiento, se realizó su implementación y se ejecutaron pruebas con datos de entrada de *n* grandes como se sugiere la teoría. Como resultado se obtuvo un programa funcional y la mejor comprensión de los temas por parte de los alumnos al comparar la complejidad algorítmica.

Palabras clave: complejidad algorítmica, algoritmos de ordenamiento, Java

ABSTRACT:

The objective of this paper is to describe the process of implementing sorting algorithms in the Java programming language as a didactic exercise to compare their algorithmic complexity and for students to have a better understanding of the theoretical foundations of such complexity by experiencing the impact it has on selected sorting algorithms. For the development of the present work, the various complexities as well as some sorting algorithms were investigated, their implementation was performed, and tests were executed with large n input data as suggested by the theory. As a result, a functional

program and better understanding of the topics by the students were obtained by comparing the algorithmic complexity.

INTRODUCCIÓN

En ciencias de la computación la complejidad algorítmica es una medida de cuánto tiempo tardaría en completarse un algoritmo dada una entrada de tamaño *n.* Si un algoritmo tiene que escalar, debe calcular el resultado dentro de un límite de tiempo finito y práctico incluso para valores grandes de *n*. Por esta razón, la complejidad se calcula asintóticamente cuando *n* tiende a infinito. Si bien la complejidad generalmente se expresa en términos de tiempo, a veces la complejidad también se analiza en términos de espacio, lo que se traduce en los requisitos de memoria del algoritmo (Zenil, 2020).

El análisis de la complejidad algorítmica es útil cuando se comparan algoritmos y se buscan mejoras. La complejidad algorítmica cae dentro de una rama de la informática teórica llamada teoría de la complejidad computacional. Es importante tener en cuenta que se está interesado en el orden de complejidad del algoritmo, no en el tiempo de ejecución real en milisegundos. La complejidad del algoritmo también se conoce como complejidad o tiempo de ejecución (Deshko et al., 2021)

La ordenación es otra área muy importante de los algoritmos. Las computadoras a menudo tienen que ordenar grandes cantidades de datos en función de algún atributo de los mismos, como ordenar una lista de archivos por su nombre o tamaño, o los correos electrónicos por la fecha en que se recibieron, o una lista de clientes según los nombres de las personas. La mayoría de las veces se hace para facilitar la búsqueda. Por ejemplo, se puede tener una gran cantidad de datos y cada uno de ellos puede ser el nombre de alguien y su número de teléfono. Si se desea buscar a alguien por su nombre, sería útil tener primero los datos ordenados alfabéticamente según los nombres de cada uno, pero si luego se quiere buscar un número de teléfono, sería más útil tener los datos ordenados según los números de teléfono de las personas (Alsuwaiyel, 2021).

La ordenación es un paso adicional que se lleva a cabo para aumentar la eficacia de la operación que se está realizando. Por ejemplo, la ordenación previa de una estructura de datos como un array puede acelerar una operación de búsqueda. Además, es necesario para que

algunos algoritmos funcionen, por ejemplo, la búsqueda binaria sólo funciona con datos ordenados (Karunanithi, 2014).

Se tiene una gran variedad de algoritmos de ordenación que van desde la ordenación por burbuja, la ordenación por selección, la ordenación por fusión, la ordenación rápida, la ordenación por recuento, la ordenación radix, etc. Cada uno de ellos tiene su propio conjunto de puntos débiles y fuertes, en función de los cuales pueden ser empleados por los desarrolladores (Prajapati et. al, 2017)

Una de las dificultades que se presentan cuando se tocan los temas de complejidad algorítmica es que muchas se veces se presenta el fundamento teórico muy abstracto y una gran mayoría de estudiantes comentan que no se entiende porque no se aterriza dicho tema en algo tangible. Una forma de entender la teoría es experimentar como impacta dicha complejidad en algunos algoritmos en este caso de ordenamiento.

METODOLOGIA

Como primer paso se procede a seleccionar y estudiar de manera teórica los fundamentos de los algoritmos de ordenamiento más populares, así como su complejidad. Para posteriormente implementar los siguientes algoritmos de ordenamiento: burbuja, selección, inserción, mezcla y rápido, en el lenguaje de programación Java, lo anterior para tomar el tiempo de ejecución y con dicho tiempo graficarlo, todo ello para comparar los fundamentos teóricos con la práctica.

Como se ha mencionado para cada algoritmo se procedió a su implementación en el lenguaje de programación Java con las librerías JFreeChart que es una librería que sirve para realizar de manera simple gráficos en el lenguaje Java.

Por cada algoritmo seleccionado se procedió a internamente realizar diez vectores, en dichos arreglos se almacenaron números pseudoaleatorios generados con la instrucción *Random* propia del lenguaje de programación Java. Posterior a eso el programa desarrollado tiene la funcionalidad de poder capturar desde teclado la cantidad de números que se deseen ordenar en el vector inicial (el programa sugiere el numero 10,000) pero se puede modificar con cualquier otro número, para la experimentación realizada se colocó un valor de 100,000 y el mismo programa va incrementando en 100,000 más hasta llegar a 1,000,000 ya que como se comentó se realizaron 10 vectores entonces en el arreglo 1 quedaría de 100,000 en el arreglo

2 de 200,000 y así sucesivamente hasta el arreglo 10 que quedaría de 1,000,000 de posiciones y lleno de números aleatorios.

El siguiente paso es poner a ordenar cada uno de los métodos de ordenamiento con sus respectivos arreglos todos llenos de números pseudoaleatorios y tomar solo el tiempo en milisegundos que tarda en ordenar cada uno de los vectores.

Una vez que se han ordenado todos los arreglos con todos los métodos de ordenamiento se procede a mostrarlos en pantalla y el programa automáticamente realiza un gráfico del tiempo empleado con cada algoritmo, así como sus elementos procesados. Con lo anterior los alumnos pueden comprobar las diversas ordenes de complejidad vistos en un inicio de manera teórica, su comprobación en programas funcionales en este caso de ordenamiento.

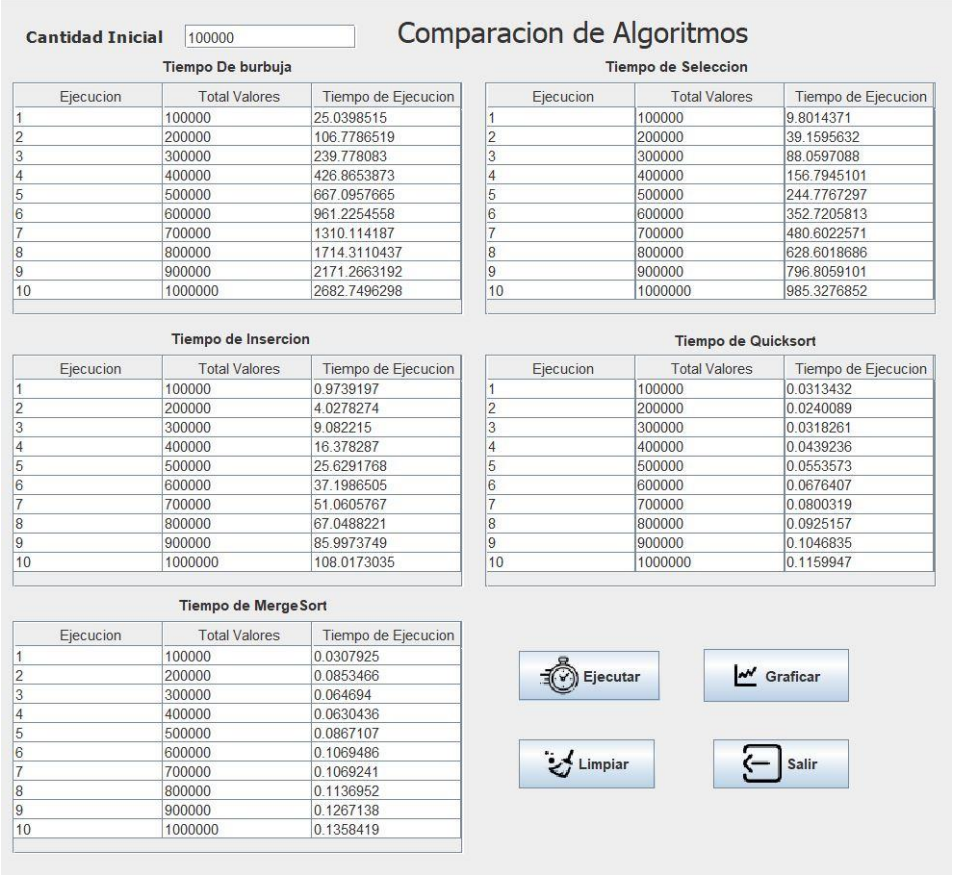
Cabe comentar que dicha experimentación que llamaremos experimento 1 se realizó en un equipo de cómputo con las siguientes características: procesador Intel Core i5-7200U CPU 2.50 Ghz, sistema operativo Windows 10 64 bits, memoria RAM de 16 GBytes, Disco duro primario SSD de 256 GBytes y Disco duro secundario de 1 Terabyte SATA III.

Para fines de ampliar la perspectiva se puso a prueba la misma implementación de algoritmos con otro equipo de cómputo (experimento 2) de diferentes características las cuales son: procesador Intel Pentium G620 2.60 Ghz, sistema operativo Windows 10 64 bits, memoria RAM de 4 GBytes, Disco duro primario de 465 GBytes SATA

RESULTADOS

Los resultados obtenidos del tiempo de ejecución (en milisegundos) por parte de cada uno de los algoritmos de ordenamiento se pueden ver en la figura 1. Donde se pueden apreciar tanto los algoritmos de ordenamiento, el número de ejecuciones realizadas, el total de valores ordenados y el tiempo de ejecución (ordenamiento) en milisegundos para cada ejecución por cada algoritmo de ordenamiento.

Dentro de los resultados obtenidos también es posible ver la gráfica en la figura 2 de los diversos algoritmos de ordenamiento, una vez que ha terminado el ordenamiento para todos los métodos. Donde se puede apreciar el orden de complejidad de manera gráfica y con ellos comprobándose los fundamentos teóricos de las diversas ordenes de complejidad.



Tiempo De burbuja

Ejecucion	Total Valores	Tiempo de Ejecucion
1	100000	25.0398515
2	200000	106.7786519
3	300000	239.778083
4	400000	426.8653873
5	500000	667.0957665
6	600000	961.2254558
7	700000	1310.114187
8	800000	1714.3110437
9	900000	2171.2663192
10	1000000	2682.7496298

Tiempo de Seleccion

Ejecucion	Total Valores	Tiempo de Ejecucion
1	100000	9.8014371
2	200000	39.1595632
3	300000	88.0597088
4	400000	156.7945101
5	500000	244.7767297
6	600000	352.7205813
7	700000	480.6022571
8	800000	628.6018686
9	900000	796.8059101
10	1000000	985.3276852

Tiempo de Insercion

Ejecucion	Total Valores	Tiempo de Ejecucion
1	100000	0.9739197
2	200000	4.0278274
3	300000	9.082215
4	400000	16.378287
5	500000	25.6291768
6	600000	37.1986505
7	700000	51.0605767
8	800000	67.0488221
9	900000	85.9973749
10	1000000	108.0173035

Tiempo de Quicksort

Ejecucion	Total Valores	Tiempo de Ejecucion
1	100000	0.0313432
2	200000	0.0240089
3	300000	0.0318261
4	400000	0.0439236
5	500000	0.0553573
6	600000	0.0676407
7	700000	0.0800319
8	800000	0.0925157
9	900000	0.1046835
10	1000000	0.1159947

Tiempo de MergeSort

Ejecucion	Total Valores	Tiempo de Ejecucion
1	100000	0.0307925
2	200000	0.0853466
3	300000	0.064694
4	400000	0.0630436
5	500000	0.0867107
6	600000	0.1069486
7	700000	0.1069241
8	800000	0.1136952
9	900000	0.1267138
10	1000000	0.1358419

Figura 1. Algoritmos de ordenamiento con su tiempo de ejecución, experimento 1. Fuente: Ejecución en programa Java de elaboración propia.

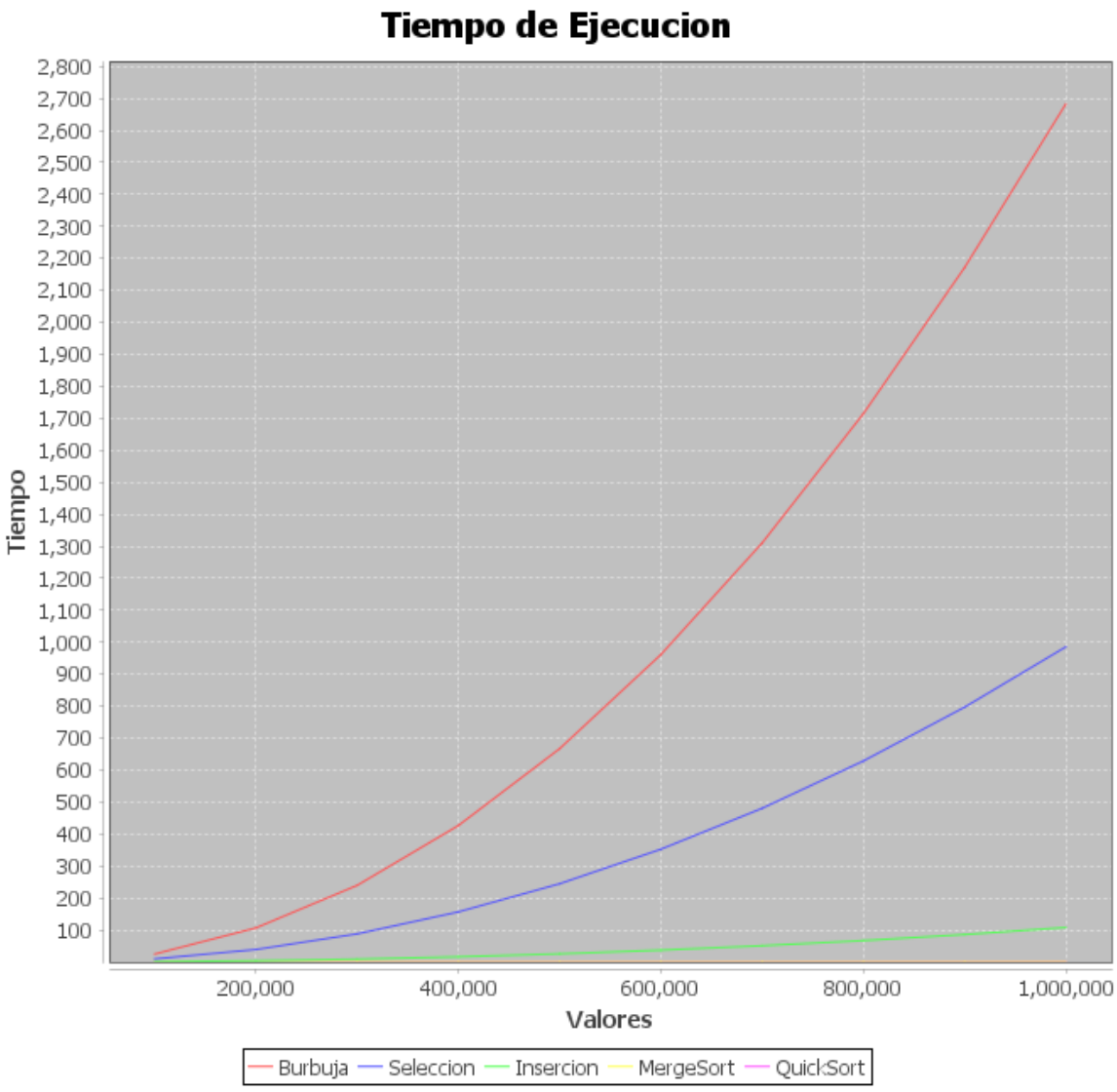


Figura 2. Grafica del tiempo de ejecución de los algoritmos de ordenamiento, experimento 1. Fuente: Ejecución en programa Java de elaboración propia.

DISCUSIÓN

Se investigo dentro de la literatura existente (Chauhan et al.,2020) y (Roy et al., 2019) que complejidad temporal debieran tener los algunos de los algoritmos más populares de ordenación (Burbuja, Selección, inserción, Mezcla y Rápido) con la finalidad de realizar una comparación con los resultados de la implementación realizada por este trabajo, encontrándose que a nivel de grafica de tiempo de ejecución son muy similares dentro de los algoritmos mencionados.

De igual manera para tener una perspectiva más amplia se procedió a poner a prueba la misma implementación, pero en otro equipo de cómputo con diferentes prestaciones (experimento 2) para comprobar los resultados entre un equipo y otro. Los resultados a nivel tiempo de ejecución variaron un poco siendo más tiempo de ejecución (en milisegundos) en el experimento 2 que en el experimento 1 pero la complejidad temporal manifestada a nivel grafica tanto del experimento 1 y del 2 son bastante similares.

CONCLUSIONES

Con relación al objetivo planteado y en virtud de los resultados obtenidos se puede concluir lo siguiente: la implementación de los algoritmos de ordenamiento se ha efectuado de manera satisfactoria y será de gran utilidad para la comprensión de la complejidad algorítmica en los estudiantes. Lo anterior da pie a en un trabajo futuro poder implementar otros tipos de algoritmos de estructuras de datos populares para comprobar su complejidad.

LITERATURA CITADA

Alsuwaiyel, M. H. (2021). Algorithms: design techniques and analysis (Vol. 15). World Scientific.

Chauhan, Yash & Duggal, Anuj. (2020). *Different Sorting Algorithms comparison based upon the Time Complexity*. 7. 10.1729/Journal.24472.

Deshko, I.P., Tsvetkov, V.Y. (2021). *Physical Complexity of Algorithms*. In: Silhavy, R. (eds) Software Engineering and Algorithms. CSOC 2021. Lecture Notes in Networks and Systems, vol 230. Springer, Cham. https://doi.org/10.1007/978-3-030-77442-4_32

Karunanithi, A. K. (2014). A survey, discussion and comparison of sorting algorithms. *Department of Computing Science, Umea University*.

Prajapati, P., Bhatt, N., & Bhatt, N. (2017). Performance comparison of different sorting algorithms. *vol. VI, no. Vi*, 39-41.

Roy, Hridoy & Shafiuzzaman, Md & Samsuddoha, Md. (2019). *SRCS: A New Proposed Counting Sort Algorithm based on Square Root Method*. 1-6. 10.1109/ICCIT48885.2019.9038601.

Zenil, H. (2020). *A Review of Methods for Estimating Algorithmic Complexity: Options, Challenges, and New Directions. Entropy, 22(6), 612.* doi:10.3390/e22060612.