TEC | Tecnológico de Costa Rica

Área Académica Ingeniería en Computadores
Ingeniería en Computadores
CE1103 - Algoritmos y Estructuras de Datos I

Tarea Corta

Algoritmos de ordenamiento

Montero Vargas Alexander
2023166058

Ocampo Zamora Nathalia
2021008216

Sibaja Garro Bryan
2022207842

3 de noviembre de 2023

Introducción

Esta investigación tiene como objetivo analizar algunos algoritmos de ordenamiento y su rendimiento. Se busca poder comparar la eficiencia de estos algoritmos de *sorting*, contrastando sus valores teóricos con los valores de ejemplos reales en pruebas con Java 17.

Para este caso concreto se aplicaron a los algoritmos de Selection Sort, Bubble Sort, Insertion Sort, Shell Sort, Merge Sort, Quick Sort y Radix Sort utilizando los códigos de ejemplo de las presentaciones del curso. Se emplearon muestras de diez mil (10 000), cien mil (100 000) y un millón (1 000 000) de elementos de tipo entero, realizando quince (15) pruebas para cada muestra en cada uno de los algoritmos.

Los resultados obtenidos en las quince pruebas correspondientes se anotaron para calcular un promedio de significancia, esto con el fin de realizar la comparación entre el tiempo de la prueba practica y el valor teórico correspondiente. Los datos promedios obtenidos se grafican para cada una de las cargas en los diferentes métodos de ordenamiento para observar el comportamiento de manera visual.

Complejidad Teórica

Selection Sort:

La complejidad temporal de este algorítmo es de la forma $O(n^2)$ ((Joyanes Aguilar, 2006)).

Bubble Sort:

La complejidad temporal de este algorítmo es de la forma $O(n^2)$ ((Joyanes Aguilar, 2006)).

Insertion Sort:

La complejidad temporal de este algorítmo es de la forma $O(n^2)$ ((Joyanes Aguilar, 2006)).

Shell Sort:

La complejidad de este algorítmo es de la forma $O(nlog^2(n))$ ((Rosas Carillo, 2016)).

Merge Sort:

La complejidad de este algorítmo es de la forma $O(nlog(n))$ ((Pankaj, 2022)).

Quick Sort:

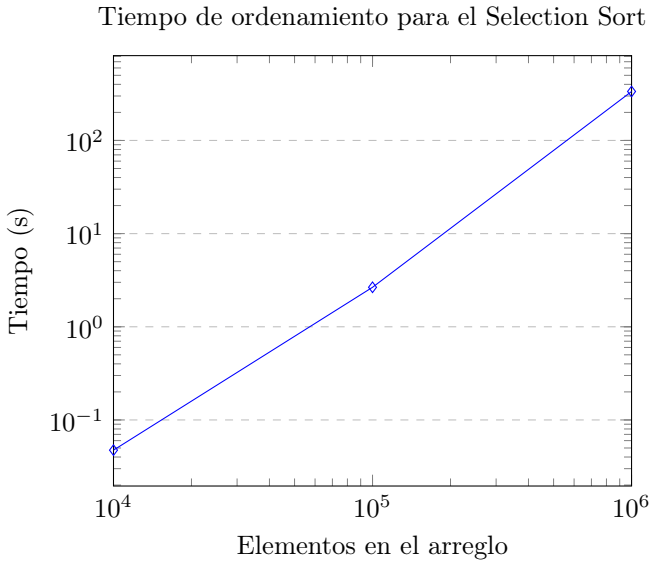
La complejidad de este algorítmo es de la forma $O(nlog(n))$ ((Florez, 2018)).

Radix Sort:

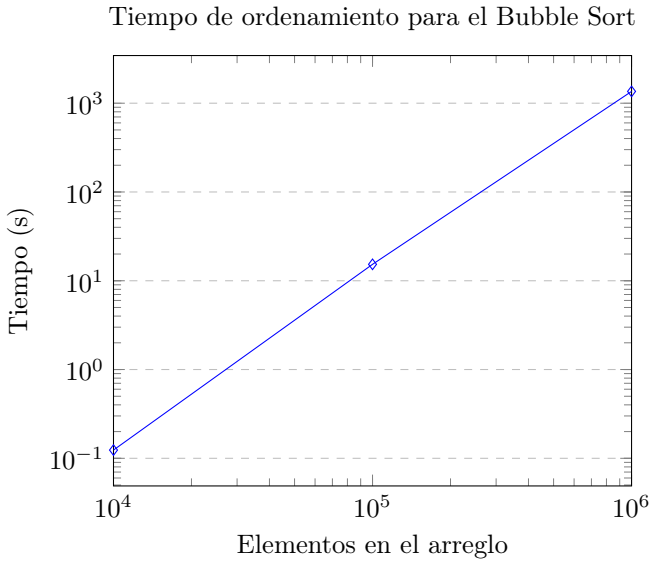
La complejidad de este algorítmo es de la forma $O(d*(n+b)))$, donde d es el número de dígitos de la lista, n es el número de elementos en la lista y b es la base o valde usado. ((*Sorting Algorithms Learning Tool*, s.f.)).

Gráficas de número de elementos vs tiempo de cada algoritmo

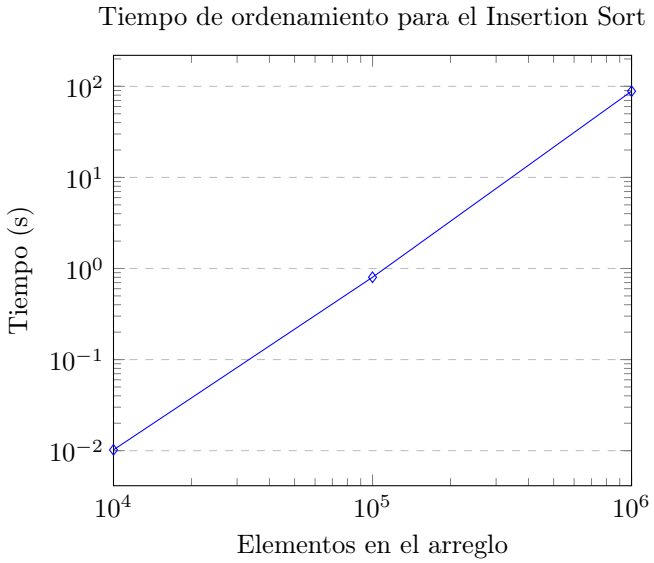
Selection Sort



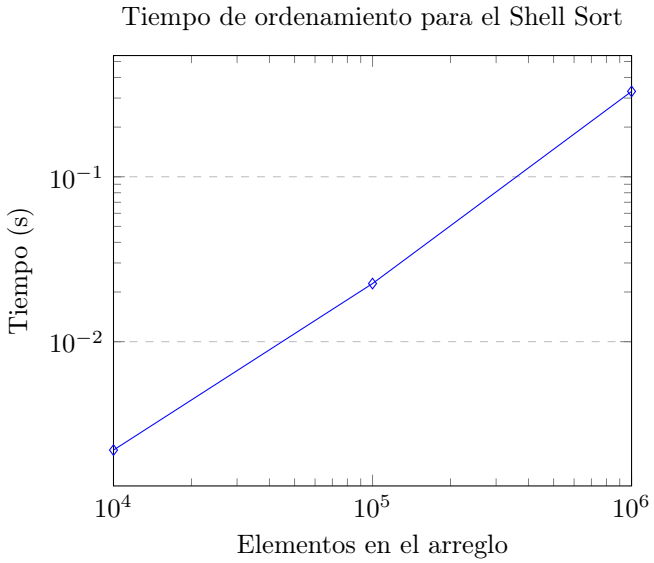
Bubble Sort



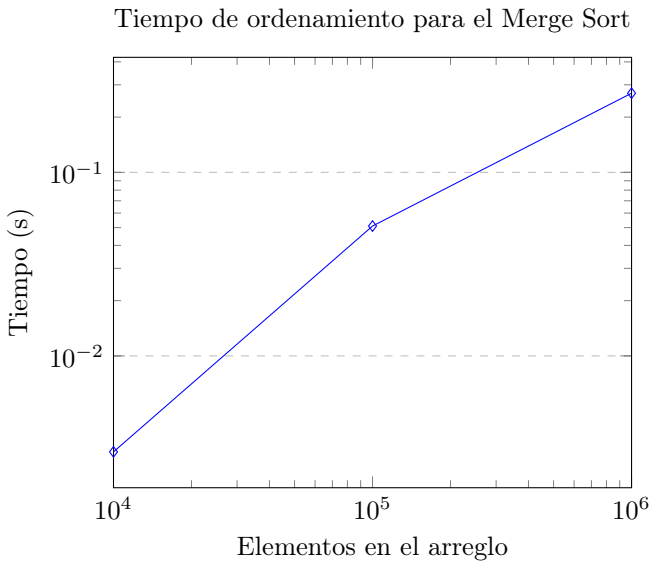
Insertion Sort



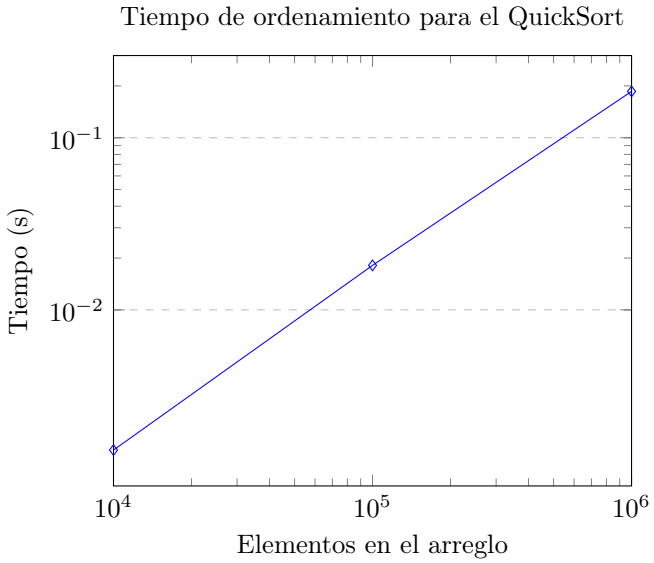
Shell Sort



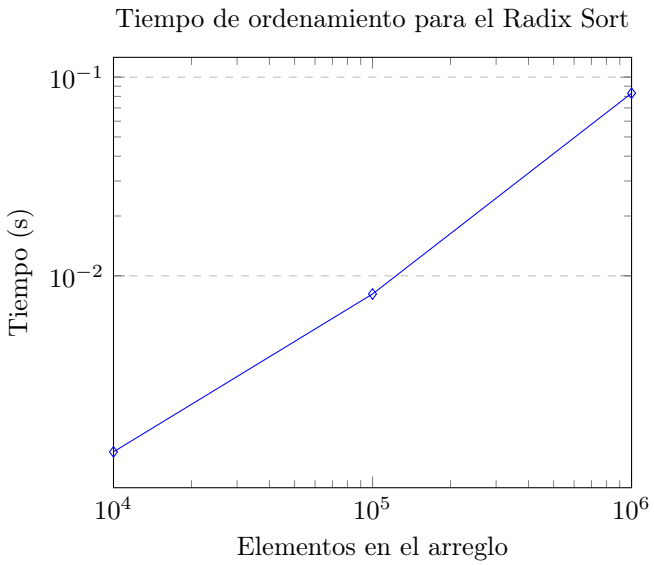
Merge Sort



QuickSort Sort



Radix Sort



Análisis de resultados

Algoritmos de complejidad cuadratica $O(n^2)$

Los algoritmos de complejidad cuadrática, como el Selection Sort, Bubble Sort e Insertion Sort muestra un alto incremento en el tiempo de ejecución al aumentar la carga, en este caso la cantidad de elementos del arreglo que se deseaba ordenar. Entre estos el algoritmo que más tiempo requirió para ordenar la lista dada fue el Bubble Sort, y el de menor tiempo fue el Insertion Sort.

Algoritmos de complejidad logarítmica $O(nlog(n))$

Por otro lado, los algoritmos Shell Sort, Merge Sort y Quicksort que presentan una complejidad logarítmica, da mejor resultados en términos de menor tiempo de ejecución, logrando tiempos de menos de un segundo (1s) en cargas de un millón de elementos. Entre estos tres el que mejor resultado brindó fue el Quicksort y el de peor fue el Shell ya que este es un logaritmo al cuadrado, pero por tiempos muy mínimos.

Algoritmos de complejidad lineal $O(n)$

El Radix Sort sin duda fue con el que se obtuvieron los mejores resultados, reportando tiempos por debajo de 1/10 s para la carga de un millón de elementos, esta complejidad casi lineal hace que se mantenga en tiempos muy estables, su variación se debe a que en su fórmula de complejidad entran en factor la cantidad de dígitos lo cuál hace que varíe de su linealidad.

Conclusiones

La complejidad $O(n^2)$ es significativamente menos eficiente en comparación a la complejidad $O(nlog(n))$

Los tiempos de ejecución de los algorítmos varían dependiendo de la carga de elementos que se entregan para ordenar, esto por la dependencia del elemento n en la fórmulas de complejidad que hace referencia a la cantidad de elementos

Los algoritmos Sort, Merge y Quicksort tienen una complejidad $O(nlog(n))$ mientras que por otro lado los algoritmos Selection, Bubble e Insertion cuentan con complejidad $O(nlog(n))$

Entre algoritmos de la misma complejidad existen algunos que demuestran ser más optimos y brindan mejor rendimiento en comparación con otros de su misma categoría

Los algoritmos que brindaron el mejor rendimiento en cargas muy grandes fueron el Radix Sort con su comlejidad semilineal y el quickSort con complejidad logarítmica

El Radix Sort puede presentar mejores resultados si se trabajan ordenando elementos de menor cantidad de dígitos

Si se necesita ordenar grandes cargas de datos, como para proyectos de *big data* se recomienda emplear algorítmos de ordenamiento que presenten complejidad temporal $O(nlog(n))$ o más optimos para obtener resultados en menor tiempo

Repositorio

Enlace al repositorio github del código fuente utilizado en las pruebas de rendimiento
`https://github.com/alexanderMontV/CE1103_Tarea_Corta`

Referencias

Florez, S. A. (2018). *Análisis comparativo de algoritmos de ordenamiento.* Descargado de https://pereiratechtalks.com/analisis-de-algoritmos-de-ordenamiento/

Joyanes Aguilar, L. (2006). *Programación en C+: algoritmos, estructuras de datos y objetos.* McGraw-Hill.

Pankaj. (2022). *Merge sort algorithm - java, c, and python implementation.* Digital Ocean. Descargado de https://www.digitalocean.com/community/tutorials/merge-sort-algorithm-java-c-python

Rosas Carillo, A. (2016). *Caracterización del algoritmo shellsort en el ámbito del big data* Instituto Poltécnicco Nacional. Descargado de https://tesis.ipn.mx/bitstream/handle/123456789/19973/Caracterizaci%C3%B3n%20del%20algoritmo%20Shellsort%20en%20el%20%C3%A1mbito%20de%20Big%20Data.pdf?sequence=1&isAllowed=y

Sorting algorithms learning tool. (s.f.). Descargado de http://syllabus.cs.manchester.ac.uk/ugt/2019/COMP26120/SortingTool/radix_sort_info.html#:~:text=The%20time%20complexity%20of%20radix,base%2010%20for%20decimal%20representation.