Pruebas de rendimiento de varios algorítmos de ordenamiento de datos

Samuel Marín Soto

2023073212

Instituto Tecnológico de Costa Rica

CE1103: Algorítmos y estructuras de datos I

Leonardo Andres Araya Martinez

2 de noviembre del 2023

Índice

Introducción

En el ámbito de la ciencia de la computación, la eficiencia y el rendimiento son aspectos críticos a considerar al abordar problemas de procesamiento de datos. Los algoritmos de ordenamiento son pilares fundamentales en la optimización y manipulación de datos, desempeñando un papel esencial en la optimización del tiempo y recursos computacionales.

El presente informe tiene como objetivo realizar un exhaustivo benchmarking de una serie de algoritmos de ordenamiento implementados en el contexto de este estudio. Se explorarán y compararán varios enfoques, desde los más simples hasta los más sofisticados, con el fin de analizar su rendimiento y eficiencia en la tarea de ordenar conjuntos de datos de diferentes tamaños.

Este análisis exhaustivo no solo proporcionará una comprensión más profunda del rendimiento relativo de estos algoritmos, sino que también ofrecerá una visión clara de sus aplicaciones prácticas. La evaluación de su comportamiento ante diversas cantidades de datos nos permitirá no solo identificar las ventajas y limitaciones de cada algoritmo, sino también entender su idoneidad en situaciones de procesamiento de datos en la vida real.

Con este fin, se ha implementado una selección de algoritmos de ordenamiento clásicos, y se han ejecutado pruebas rigurosas utilizando conjuntos de datos de distintos tamaños. Los resultados obtenidos se analizarán y se presentarán en este informe, brindando una perspectiva clara y fundamentada sobre el rendimiento comparativo de estos algoritmos. Estos hallazgos pueden resultar valiosos para la toma de decisiones en el diseño e implementación de sistemas que involucren la manipulación y organización eficiente de grandes volúmenes de datos.

Complejidad teórica de algoritmos

Selection sort

El algoritmo Selection Sort es un método sencillo, aunque poco eficiente en el procesamiento de grandes conjuntos de datos. Este mecanismo opera localizando el elemento más pequeño y desplazándolo a la primera posición. Sin embargo, independientemente del estado de la lista, siempre realiza el mismo número de comparaciones, lo que genera una complejidad cuadrática, siendo $O(n^2)$.

Bubble sort

Bubble Sort, otro algoritmo elemental, compara pares de elementos adyacentes y los intercambia si se encuentran en un orden erróneo. En el mejor escenario, con una lista ya ordenada, su complejidad es lineal ($O(n)$), pero en la mayoría de los casos, la cantidad de comparaciones repetitivas lo lleva a una complejidad cuadrática ($O(n^2)$).

Insertion sort

Insertion Sort, al igual que Selection Sort, opera con una lista dividida en dos secciones: una ordenada y la otra desordenada. Inserta elementos en la porción ordenada de la lista, lo que le otorga una complejidad lineal ($O(n)$) en el mejor caso, cuando la lista ya está ordenada. No obstante, en promedio y en el peor escenario, su complejidad es cuadrática ($O(n^2)$).

Shell sort

Shell Sort, una mejora de Insertion Sort, utiliza secuencias de brechas para reducir la cantidad de comparaciones y movimientos de elementos. Su desempeño, no obstante, depende de la secuencia utilizada, logrando una complejidad de $O(n \log^2 n)$ con ciertas secuencias.

Merge sort

Por otro lado, Merge Sort, basado en la estrategia "dividir y conquistar", divide la lista repetidamente en mitades, las ordena y luego las fusiona. Su complejidad es siempre

O(n log n) debido a su eficaz enfoque recursivo y su habilidad para dividir y combinar las listas eficientemente.

Quick sort

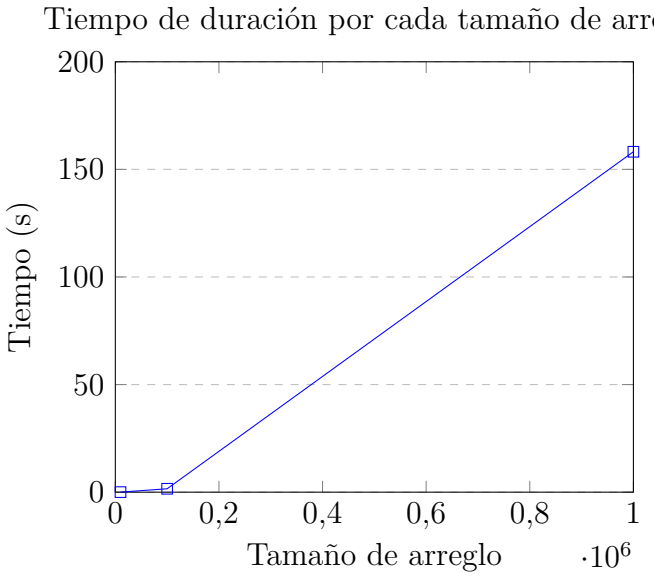
El conocido Quick Sort también se basa en la estrategia de "dividir y conquistar". En promedio, su complejidad es O(n log n) gracias a su partición eficiente, aunque en el peor caso podría llegar a una complejidad cuadrática ($O(n^2)$) si la elección del pivote no es óptima.

Radix sort

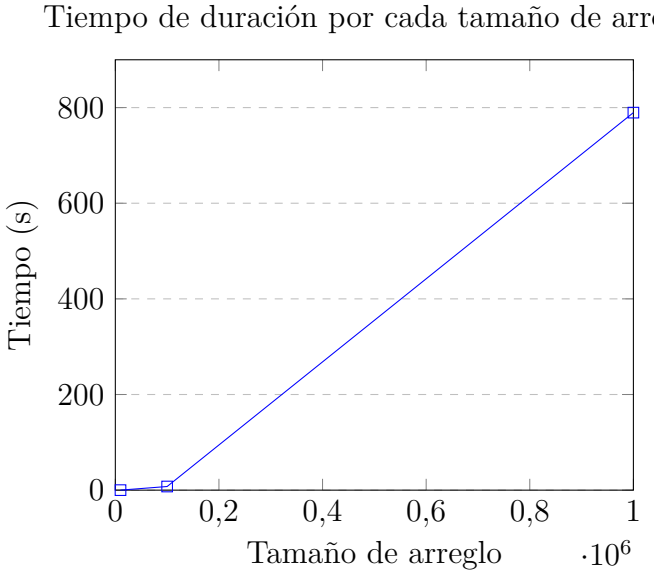
Finalmente, Radix Sort clasifica los elementos basándose en sus dígitos individuales. Su complejidad depende de la longitud de los números y el número de dígitos a considerar, manteniendo una complejidad lineal (O(n * k)) y mostrando eficiencia al ordenar números enteros.

Gráficos del tiempo que tomó ordenar arreglos de varios tamaños utilizando distintos algorítmos de ordenamiento

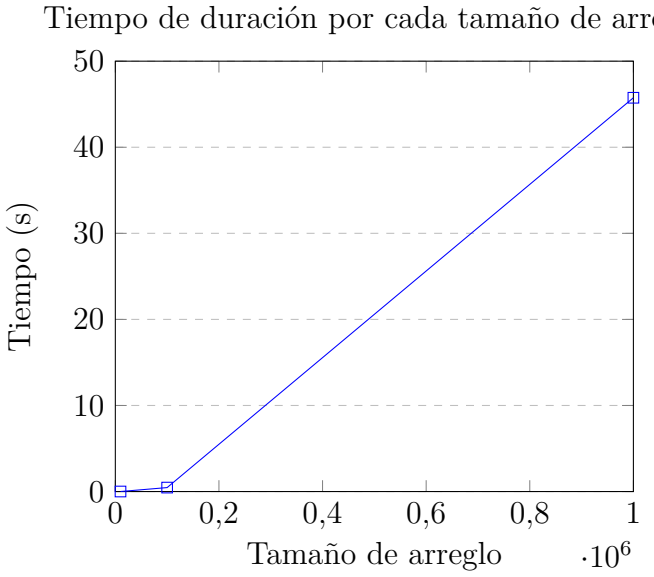
Selection sort



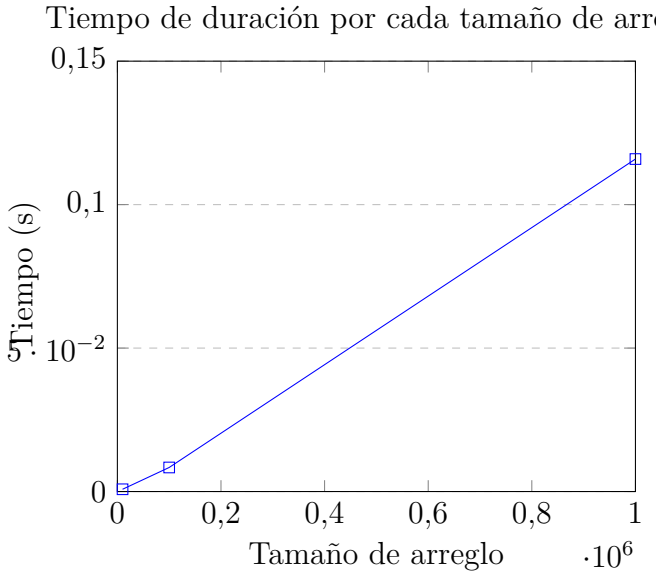
Bubble sort



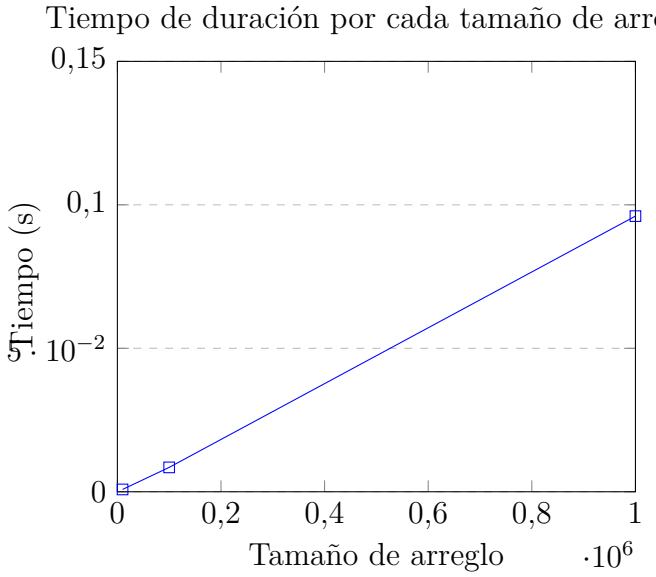
Insertion sort



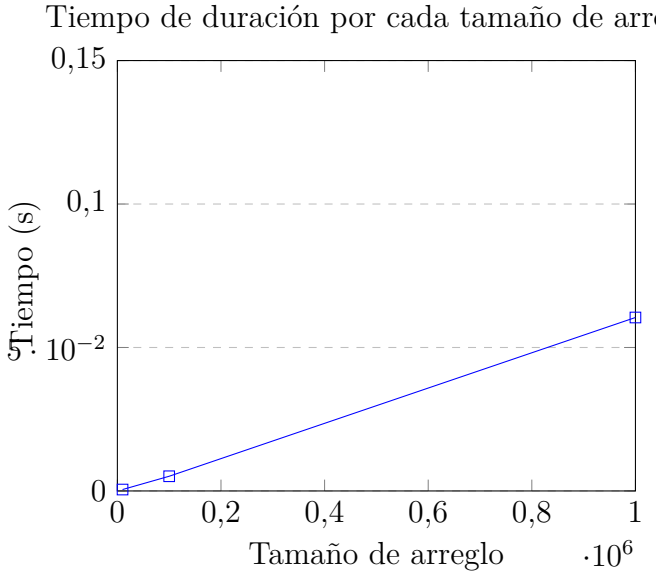
Shell sort



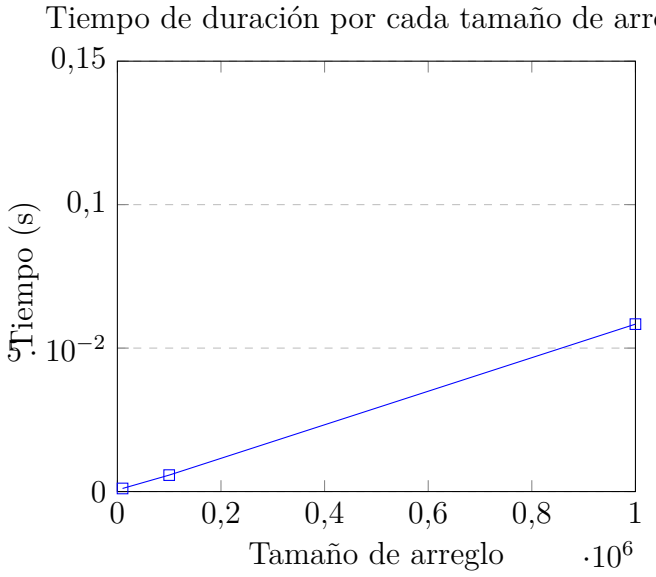
Merge sort



Quick sort



Radix sort


Tiempo de duración por cada tamaño de arreglo
0,15
0,1
$\cdot 10^{-2}$
Tiempo (s)
0
0
0,2
0,4
0,6
0,8
1
Tamaño de arreglo
$\cdot 10^{6}$

Análisis de resultados

Los datos recopilados a partir del benchmarking de los diferentes algoritmos de ordenamiento aportan una visión clara de su rendimiento en conjuntos de datos de tamaño 10,000 y 100,000. En términos generales, se puede observar una variabilidad considerable en los tiempos de ejecución de cada algoritmo en las distintas iteraciones.

En el caso de tamaños de conjuntos de datos más pequeños (10,000 elementos), se puede identificar que los algoritmos Merge Sort, Quick Sort y Shell Sort presentan tiempos de ejecución más bajos y consistentes en la mayoría de las iteraciones. Aunque Radix Sort ofrece tiempos razonables, en ciertas ocasiones, tiende a ser menos constante en sus resultados, especialmente en comparación con los otros algoritmos de ordenamiento.

Por otro lado, al aumentar el tamaño del conjunto de datos a 100,000 elementos, se observa un incremento generalizado en los tiempos de ejecución de todos los algoritmos. No obstante, al igual que en el conjunto de datos más pequeño, Merge Sort, Quick Sort y Shell Sort siguen siendo consistentes en sus tiempos de ejecución, aunque con un aumento en la brecha de tiempo.

En general, tanto para conjuntos de datos de tamaño 10,000 como 100,000, los algoritmos de ordenamiento más eficientes y consistentes resultan ser Merge Sort, Quick Sort y Shell Sort. Estos algoritmos tienden a ser preferibles para conjuntos de datos de mayor tamaño, ya que su rendimiento se mantiene razonablemente constante y son capaces de manejar conjuntos de datos más grandes de manera eficiente en términos de tiempo de ejecución.

Sin embargo, es importante considerar las características específicas de los conjuntos de datos y los requisitos del sistema antes de seleccionar un algoritmo de ordenamiento. En algunas situaciones, como cuando se ordenan números enteros o datos con ciertas propiedades, Radix Sort podría ofrecer un rendimiento competitivo.

Conclusiones

El benchmark realizado sobre una variedad de algoritmos de ordenamiento revela una clara diferenciación en sus rendimientos, especialmente al manipular conjuntos de datos de diferentes tamaños. Los resultados indican que, para conjuntos de datos más pequeños, aproximadamente de 10,000 elementos, los algoritmos de Merge Sort, Quick Sort y Shell Sort sobresalen en términos de eficiencia y consistencia en sus tiempos de ejecución. Mientras que Radix Sort, a pesar de ofrecer tiempos razonables, muestra cierta variabilidad en sus resultados.

A medida que el tamaño del conjunto de datos se amplía a 100,000 elementos, se evidencia un aumento generalizado en los tiempos de ejecución de todos los algoritmos, siendo nuevamente Merge Sort, Quick Sort y Shell Sort los que mantienen su eficiencia relativa. Esta consistencia, incluso con conjuntos de datos más grandes, subraya su idoneidad para aplicaciones que requieren la ordenación eficiente de grandes volúmenes de información.

En última instancia, la elección del algoritmo de ordenamiento más adecuado dependerá de las características específicas del conjunto de datos y los requisitos del sistema. A pesar de la superioridad relativa de algunos algoritmos en este benchmark, es crucial considerar las particularidades del escenario de uso para tomar una decisión fundamentada y eficiente en aplicaciones del mundo real.

Código

https://github.com/samuelmarinsoto/tareacortadatosI