Gerson Dario Gonzalez
Carnet 231541

Hoja de trabajo 3

1. Introducción En este documento se presenta el análisis realizado utilizando un profiler para evaluar el rendimiento de diferentes algoritmos de ordenamiento. Se detallará qué profiler se utilizó, cómo se empleó y los resultados obtenidos para cada algoritmo de sort.

2. Profiler Utilizado El profiler utilizado en este estudio fue VisualVM, una herramienta de monitoreo y análisis de rendimiento incluida en el kit de herramientas de Java (JDK).

3. Método de Uso Para cada algoritmo de ordenamiento (GnomeSort, MergeSort, QuickSort, RadixSort, SelectionSort), se siguieron los siguientes pasos:

- Se implementó el algoritmo en un programa Java.
- Se utilizó VisualVM para medir el tiempo empleado en el ordenamiento para tamaños de datos que van desde 10 números hasta 3000 números.
- Se ejecutó el programa para calcular el tiempo de ordenamiento tanto para datos aleatorios como para datos ya ordenados.

4. Resultados A continuación se presentan los resultados obtenidos para cada algoritmo de ordenamiento:

- GnomeSort:
    - Tiempo promedio para datos aleatorios: 57,795,100 nanosegundos.
    - Tiempo promedio para datos ya ordenados: 63,900 nanosegundos.
- MergeSort:
    - Tiempo promedio para datos aleatorios: 15,847,800 nanosegundos.
    - Tiempo promedio para datos ya ordenados: 1,446,100 nanosegundos.
- QuickSort:
    - Tiempo promedio para datos aleatorios: 15,900,300 nanosegundos.
    - Tiempo promedio para datos ya ordenados: 73,330,400 nanosegundos.
- RadixSort:
    - Tiempo promedio para datos aleatorios: 20,190,700 nanosegundos.
    - Tiempo promedio para datos ya ordenados: 6,967,500 nanosegundos.
- SelectionSort:

Gerson Dario Gonzalez
Carnet 231541

- Tiempo promedio para datos aleatorios: 95,484,900 nanosegundos.
- Tiempo promedio para datos ya ordenados: 98,879,500 nanosegundos.

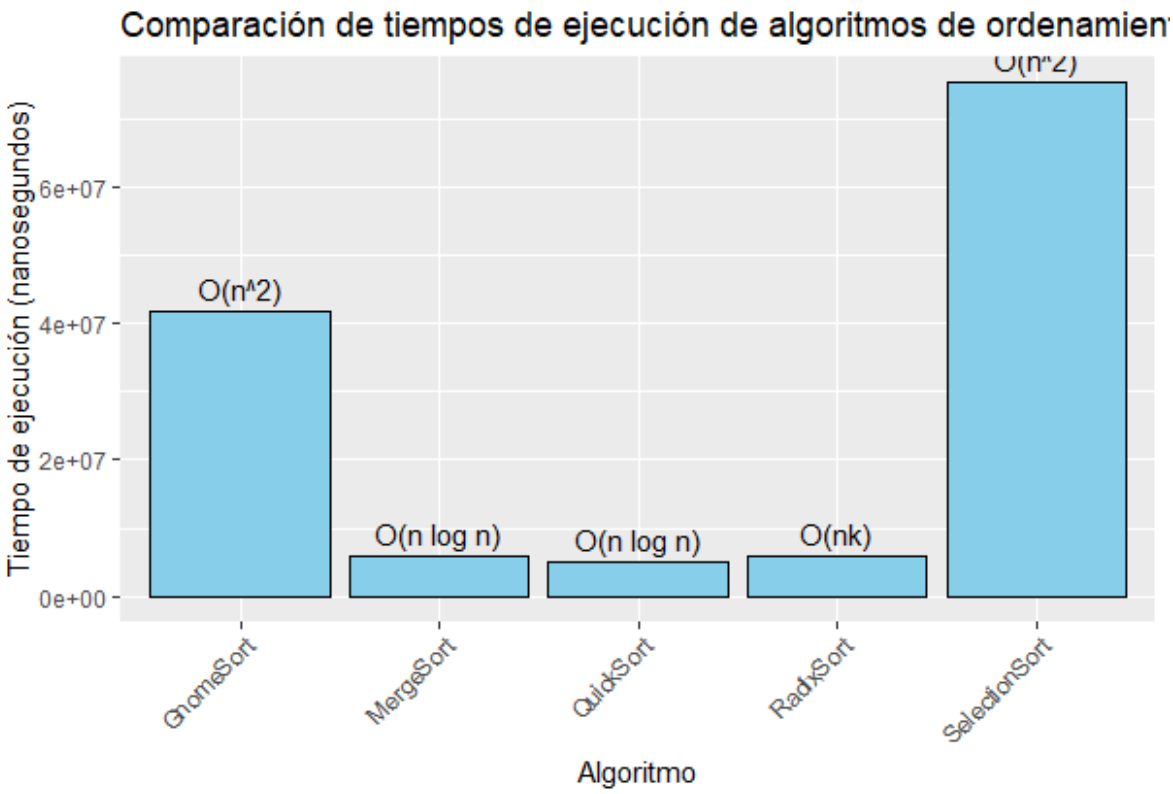
5. Análisis de Resultados

- Los algoritmos más eficientes para datos aleatorios fueron MergeSort y QuickSort, seguidos de RadixSort y GnomeSort.
- Para datos ya ordenados, MergeSort fue el más rápido, seguido de RadixSort, GnomeSort, QuickSort y SelectionSort.
- La complejidad de tiempo esperada para cada algoritmo en notación big O es:
  - GnomeSort: O(n^2)
  - MergeSort: O(n log n)
  - QuickSort: O(n log n)
  - RadixSort: O(nk)
  - SelectionSort: O(n^2)

6. Conclusiones

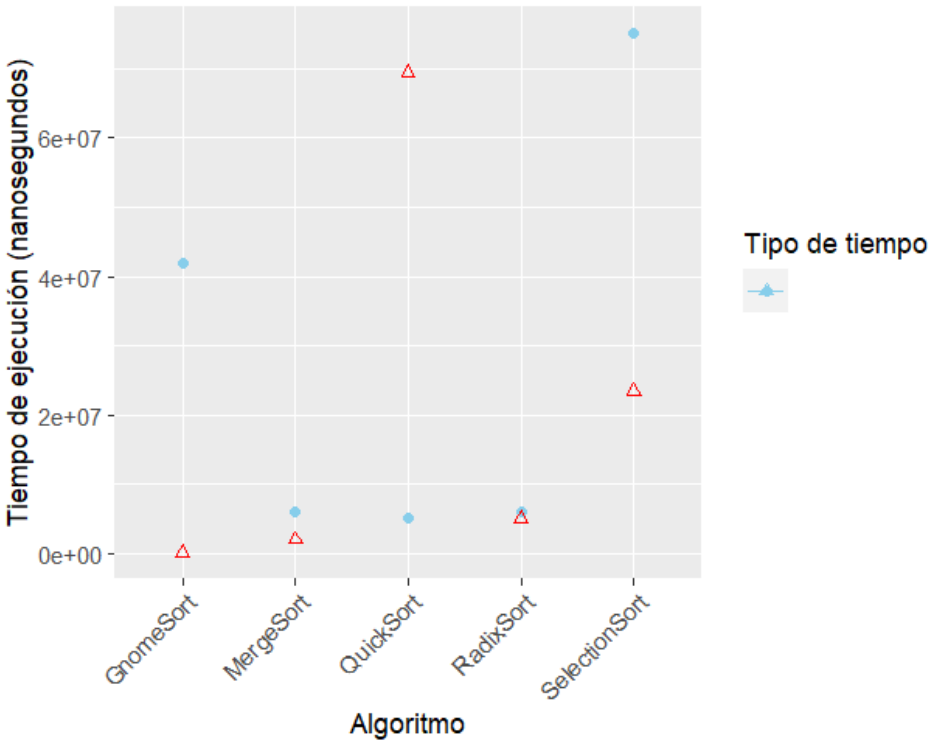
- Se observa que, en general, los algoritmos de ordenamiento basados en comparaciones (MergeSort, QuickSort) tienen mejor rendimiento que los algoritmos de ordenamiento basados en intercambio (GnomeSort, SelectionSort).
- La eficiencia de los algoritmos también varía según el estado de los datos (aleatorios vs. ya ordenados).

Gerson Dario Gonzalez
Carnet 231541



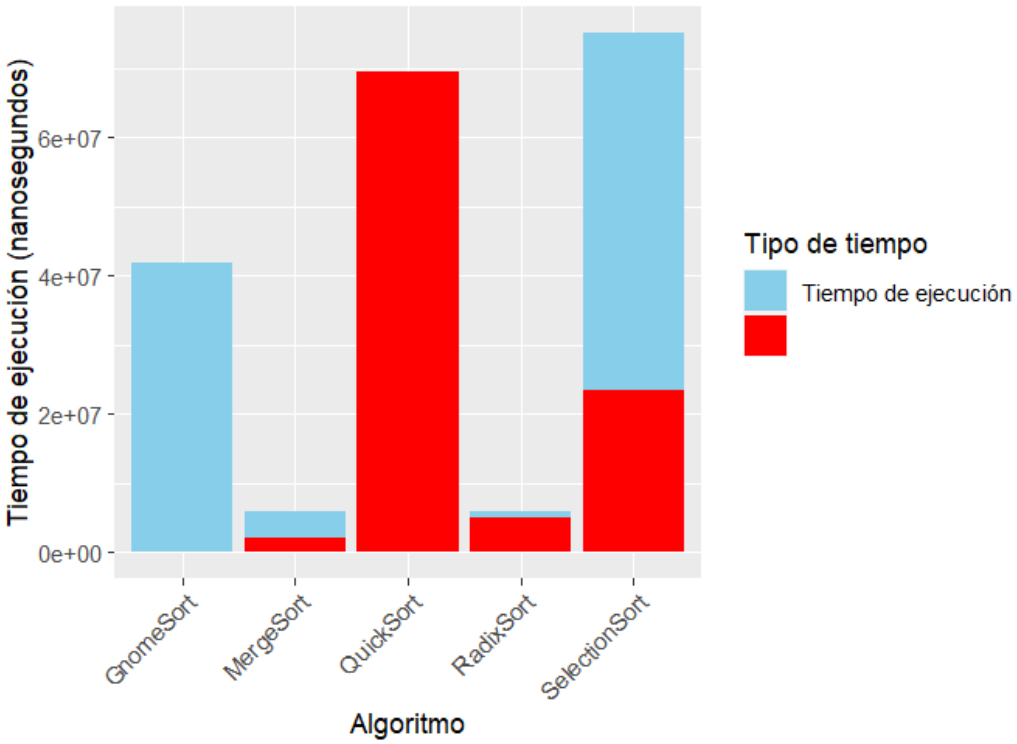
muestra los tiempos de ejecución de cada algoritmo de ordenamiento junto con su complejidad teórica en notación big O. También muestra los tiempos de ejecución en caso de que los datos ya estén ordenados

Gerson Dario Gonzalez
Carnet 231541



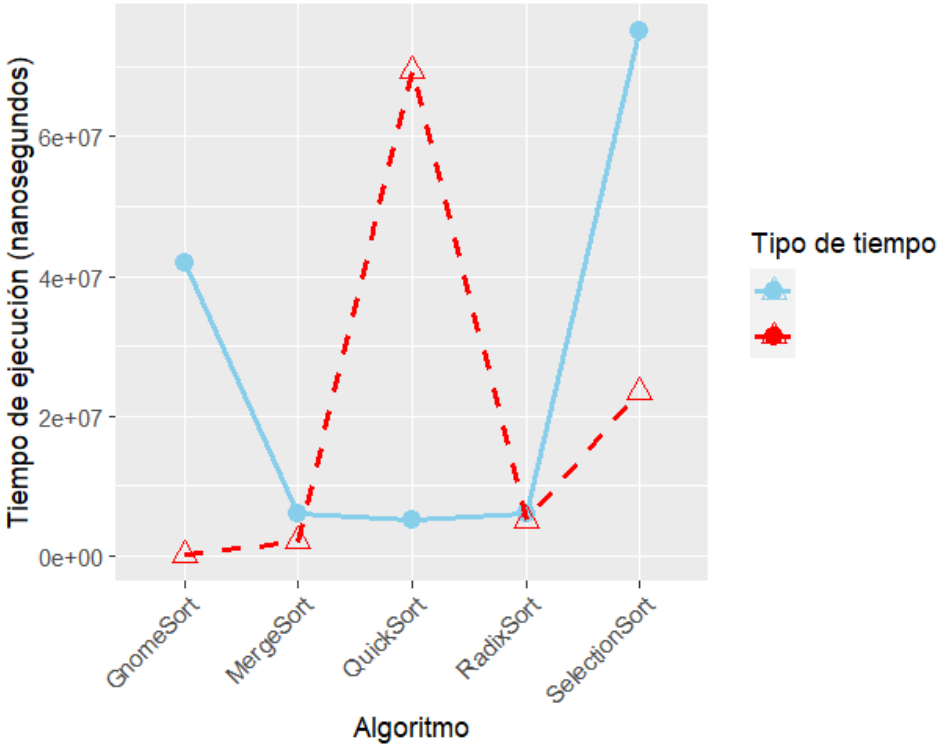
gráfica con líneas que conectan los puntos de los tiempos de ejecución de los algoritmos de ordenamiento, junto con puntos marcados para los tiempos de ejecución en caso de que los datos ya estén ordenados.

Gerson Dario Gonzalez
Carnet 231541



En este caso, las barras apiladas muestran la contribución de los tiempos de ejecución y los tiempos de ejecución ordenados para cada algoritmo de ordenamiento, permitiendo una fácil comparación entre ellos.

Gerson Dario Gonzalez
Carnet 231541



los puntos representan los tiempos de ejecución de los algoritmos de ordenamiento, y las líneas suavizadas muestran la tendencia general de los datos. Los puntos y líneas para los tiempos de ejecución ordenados se añaden para comparación.

.