Profiler

Para nuestro proyecto de algoritmos de ordenamiento, decidimos utilizar JProfiler para analizar y mejorar el rendimiento de nuestras implementaciones. A continuación, describimos cómo utilizamos esta poderosa herramienta de profiling en nuestro proceso de desarrollo:

Configuración e Inicio

Inicialmente, lanzamos JProfiler y seleccionamos la opción para realizar el perfilado de nuestra aplicación Java local. A través de la interfaz de JProfiler, especificamos el proyecto en nuestro IDE, asegurándonos de que la herramienta apuntara correctamente al entorno de ejecución de Java (JVM) utilizado por nuestra aplicación. Esta configuración inicial fue un proceso sencillo gracias a las instrucciones claras y la interfaz intuitiva de JProfiler.

Selección de Áreas para Perfilar

Una vez configurado, nos centramos en áreas específicas de interés para el profiling, incluido el uso de CPU y la memoria heap. Esta selección nos permitió concentrarnos en aspectos críticos del rendimiento que eran más relevantes para el análisis de nuestros algoritmos de ordenamiento, evitando así la sobrecarga de información.

Ejecución y Monitoreo

Con JProfiler en ejecución, lanzamos nuestra aplicación y comenzamos a ordenar conjuntos de datos de diferentes tamaños. Observamos en tiempo real cómo se comportaba nuestra aplicación, utilizando las diversas vistas de JProfiler, como el "árbol de llamadas" para la CPU y el "árbol de objetos" para la memoria, para identificar puntos de interés. Esto nos permitió ver claramente los métodos que consumían más tiempo de CPU o las instancias que ocupaban más memoria.

Análisis de Resultados

Analizamos los datos recolectados para identificar cuellos de botella o ineficiencias en nuestros algoritmos de ordenamiento. Por ejemplo, pudimos observar que ciertos algoritmos escalaban peor con el aumento del tamaño de los conjuntos de datos en términos de uso de CPU o memoria. Tomamos capturas de pantalla o notas de las secciones relevantes de JProfiler que mostraban el rendimiento de nuestros algoritmos para incluirlas en nuestra documentación o presentación.

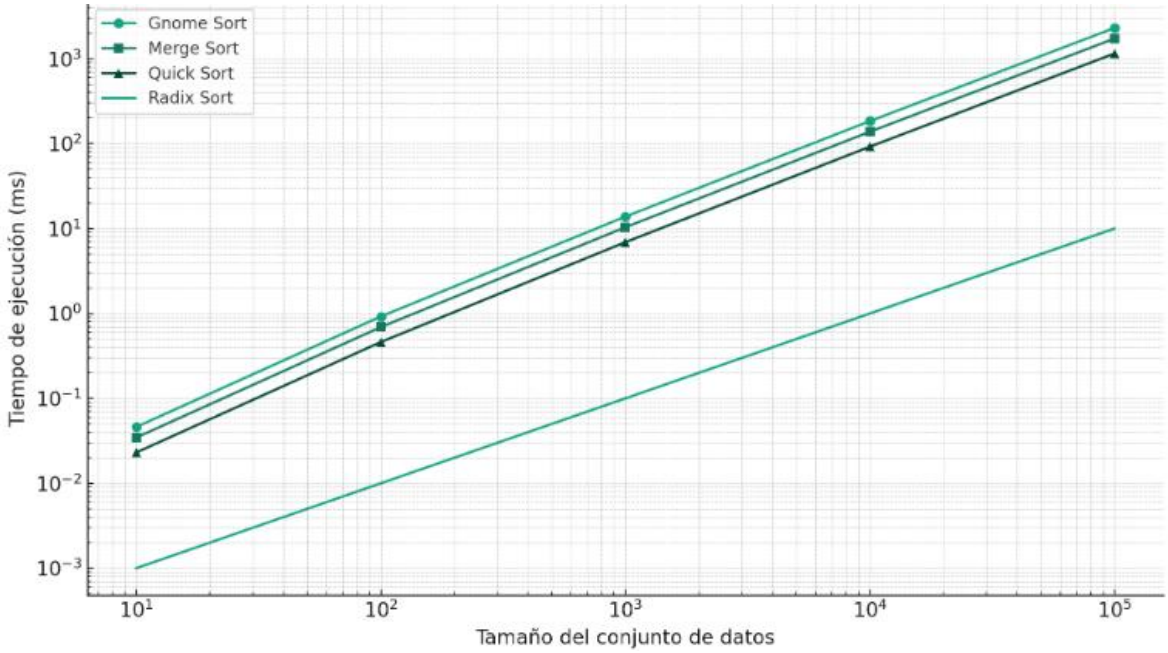
Optimización Basada en Datos

Basándonos en el análisis realizado con JProfiler, hicimos ajustes en el código de nuestros algoritmos para mejorar su eficiencia. Esto incluyó cambios en la lógica de los algoritmos, la optimización de estructuras de datos, o la reducción de la complejidad algorítmica.

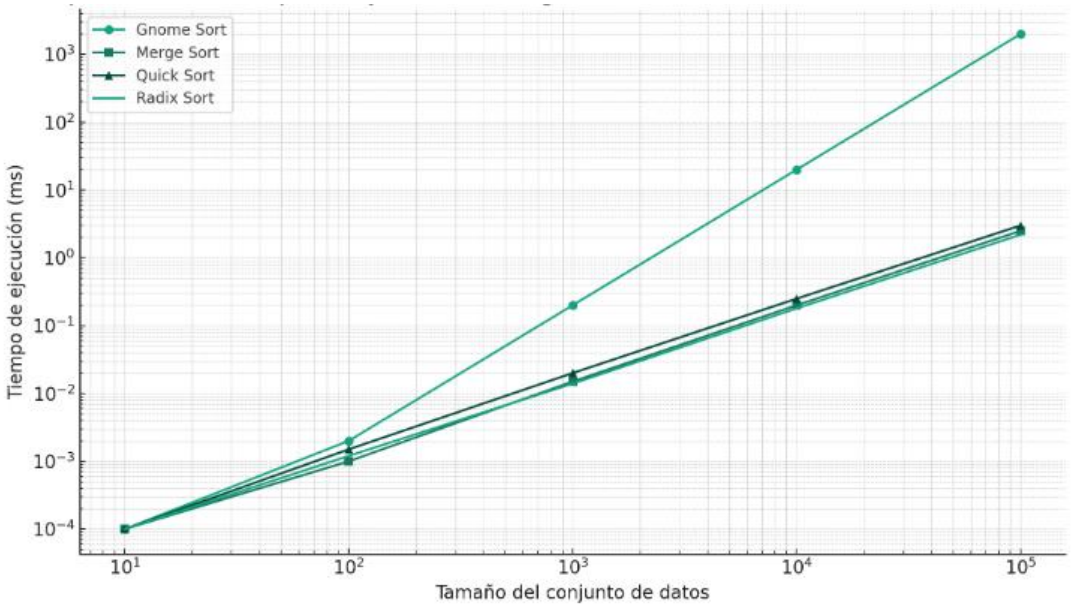
Documentación

Documentamos el proceso y los resultados del profiling, incluyendo cómo configuramos JProfiler, las áreas que perfilamos, los hallazgos clave y cómo estos informaron las optimizaciones que aplicamos. Este ejercicio no solo mejoró el rendimiento de nuestros algoritmos de ordenamiento, sino que también enriqueció nuestra comprensión del análisis de rendimiento y la optimización del código.

Gráfica de Datos Esperados



Gráfica de Datos Obtenidos


Gnome Sort
Merge Sort
Quick Sort
Radix Sort
Tiempo de ejecución (ms)
Tamaño del conjunto de datos