Sharis Barrios García

Leonel Contreras Quirós

Allan Paniagua Enríquez

**Hoja de Trabajo 3**

**Enlace de repositorio de Git:** [**https://github.com/sharlisbg/HDT-3**](https://github.com/sharlisbg/HDT-3)

En este enlace se deja evidencia de todo el desarrollo realizado.

**PROFILER UTILIZADO**

Para VS Code se instaló GraalVM que permite una integración con VisualVM, el cual es el profiler a utilizar para conocer el rendimiento en tiempo de nuestros algoritmos de ordenamiento. Además, este profiler es la herramienta de solución de problemas y monitoreo de Java (y políglota) todo en uno.

Fuentes:

* <https://marketplace.visualstudio.com/items?itemName=oracle-labs-graalvm.graalvm>
* <https://www.graalvm.org/dev/tools/vscode/graalvm-extension/visualvm-integration/>

Paso a Paso de Instalación para uso:

1. Instalación de GraalVM en VS Code:

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

1. Instalación de VisualVM: Primeramente, se descarga el programa en la computadora y luego se configura en VS Code, en la parte de Paleta de Comandos, el siguiente entorno:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Posteriormente se configuran los elementos expuestos y ya aparecen las siguientes opciones:

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

1. Luego podemos darle Play a JFR y ya estaremos utilizando el profiler para conocer en tiempo real la ejecución de nuestros algoritmos.

Una captura de pantalla de una computadora

Descripción generada automáticamente

**RESULTADOS – COMPARACIÓN**

Los algoritmos de ordenamiento utilizados fueron los siguientes:

* Gnome sort: <https://www.geeksforgeeks.org/java-program-for-gnome-sort/>
* Merge sort- Fuente: <https://www.baeldung.com/java-merge-sort#:~:text=Merge%20sort%20is%20a%20%E2%80%9Cdivide,final%20solution%20to%20the%20problem>.
* Quick sort: Ejercicio creado en clase.
* Selection Sort: Desarrollado por el equipo.
* Bubble Sort: Desarrollado por el equipo.

Este último algoritmo de ordenamiento fue el elegido en el grupo para ser trabajado.

El ejercicio constará en realizar 4 sets de corridas para ordenar, 10, 100, 1000 y 3000 números generados de manera aleatoria. A continuación, se detallan los datos obtenidos y las evidencias de los casos:

|  |  |  |  |
| --- | --- | --- | --- |
| Prueba | Cantidad de datos | Teórico | Real |
| Gnome sort  O(n²) | 10 | 100 | 0.0015 |
| 100 | 10,000 | 0.027 |
| 1000 | 1,000,000 | 0.129 |
| 3000 | 9,000,000 | 0.378 |
| Merge sort  O(n log n) | 10 | 10 | 0.078 |
| 100 | 200 | 0.362 |
| 1000 | 3,000 | 1.17 |
| 3000 | 10,431 | 3.32 |
| Quick sort  O(n log n) | 10 | 10 | 0.073 |
| 100 | 200 | 1.53 |
| 1000 | 3,000 | 8.34 |
| 3000 | 10,431 | 19.5 |
| Selection sort  O(n) | 10 | 10 | 0.052 |
| 100 | 100 | 0.502 |
| 1000 | 1,000 | 13.1 |
| 3000 | 3,000 | 27.9 |
| Bubble sort  O(n²) | 10 | 100 | 0.204 |
| 100 | 10,000 | 0.739 |
| 1000 | 1,000,000 | 23.4 |
| 3000 | 9,000,000 | 98.9 |

Las gráficas que evidencian los tiempos de corrida teóricos vs teóricos de los algoritmos según las diversas escalas son los siguientes

El propósito será generar una gráfica con el eje Y mostrando el tiempo y el eje X mostrando la cantidad de números ordenados.

**Evidencias:**

**Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**Interfaz de usuario gráfica

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

**PRUEBAS UNITARIAS**

Se ejecutaron las siguientes pruebas unitarias para evidenciare que los algoritmos de ordenamiento tienen un correcto funcionamiento.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**VIDEO EXPLICATIVO DE MÉTODO UTILIZADO**

Enlace del video de utilización del funcionamiento del algoritmo de elección Bubble Sort y Selection Sort.

Enlace: