**OBSERVACIONES DEL LA PRACTICA**

Ernesto José Duarte Mantilla Cod.202014279

Ana Sofía Padilla Daza Cod.202021748

|  |  |  |
| --- | --- | --- |
|  | Máquina 1 | Máquina 2 |
| Procesadores | AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz | 2,5 GHz Intel Core i5 de dos núcleos |
| Memoria RAM (GB) | 8,00 GB | 16 GB |
| Sistema Operativo | Windows 10 Home Single Language 64-bits | macOS Catalina 10.15.7 |

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

# **Maquina 1**

## **Resultados**

|  |  |  |
| --- | --- | --- |
| ***Carga de Catálogo PROBING*** | | |
| **Factor de Carga (PROBING)** | **Consumo de Datos [kB]** | **Tiempo de Ejecución [ms]** |
| 0.30 | 1045449,188 | 81155,723 |
| 0.50 | 1045434,787 | 73259,383 |
| 0.80 | 1045434,787 | 73014,920 |

Tabla 2. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando PROBING en la Maquina 1.

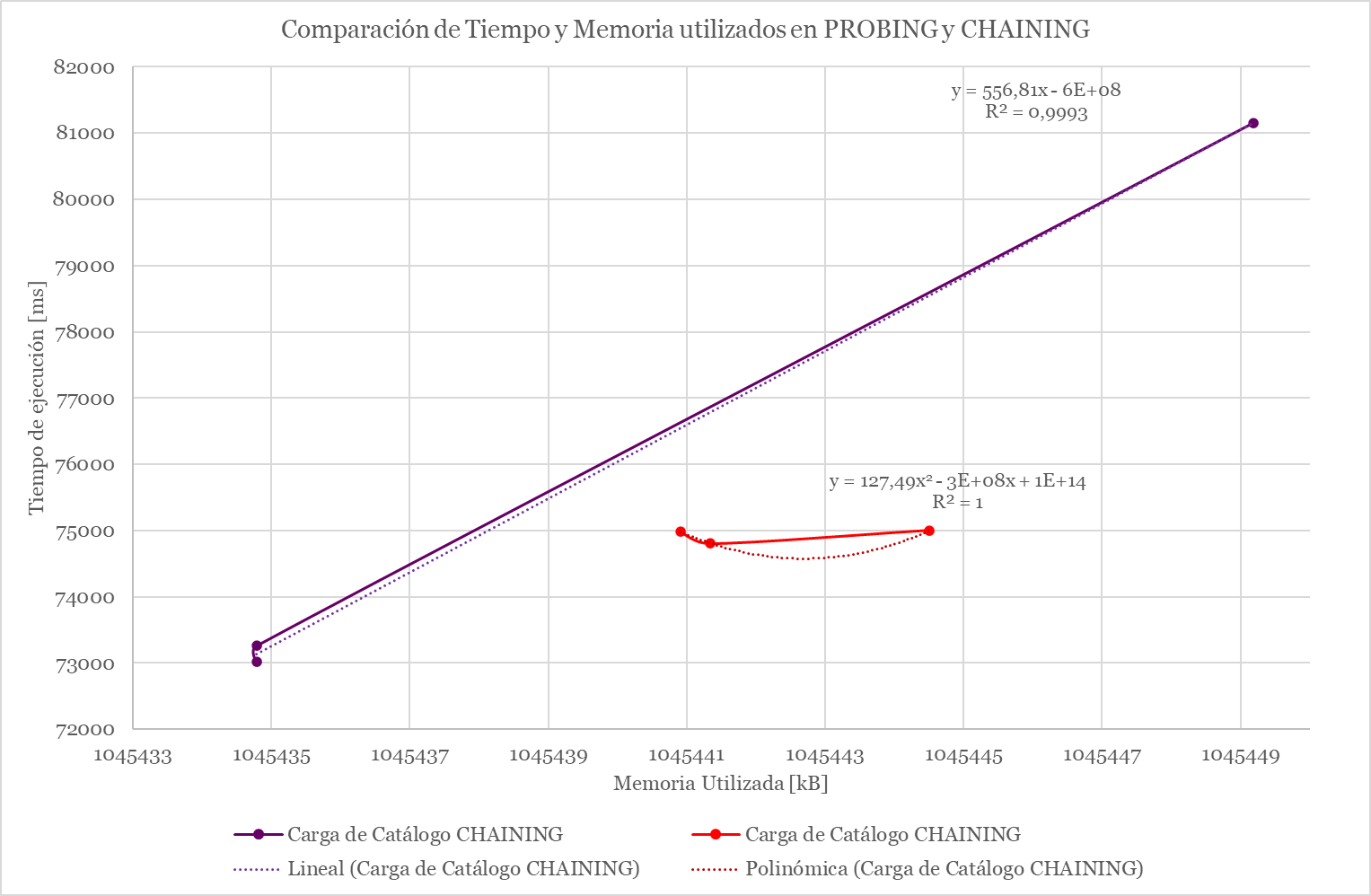
|  |  |  |
| --- | --- | --- |
| ***Carga de Catálogo CHAINING*** | | |
| **Factor de Carga (CHAINING)** | **Consumo de Datos [kB]** | **Tiempo de Ejecución [ms]** |
| 2.00 | 1045444,510 | 74999,037 |
| 4.00 | 1045441,346 | 74807,840 |
| 6.00 | 1045440,912 | 74980,690 |

Tabla 3. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando CHAINING en la Maquina 1.

## **Graficas**

La gráfica generada por los resultados de las pruebas de rendimiento en la **Maquina 1.**

* Comparación de memoria y tiempo de ejecución para PROBING y CHAINING



# **Maquina 2**

## **Resultados**

|  |  |  |
| --- | --- | --- |
| ***Carga de Catálogo PROBING*** | | |
| **Factor de Carga** (PROBING) | **Consumo de Datos [kB]** | **Tiempo de Ejecución [ms]** |
| 0.30 | 1310658,495 | 52450,504 |
| 0.50 | 1310639,305 | 53632,565 |
| 0.80 | 1310639,109 | 50114,944 |

Tabla 4. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando PROBING en la Maquina 2.

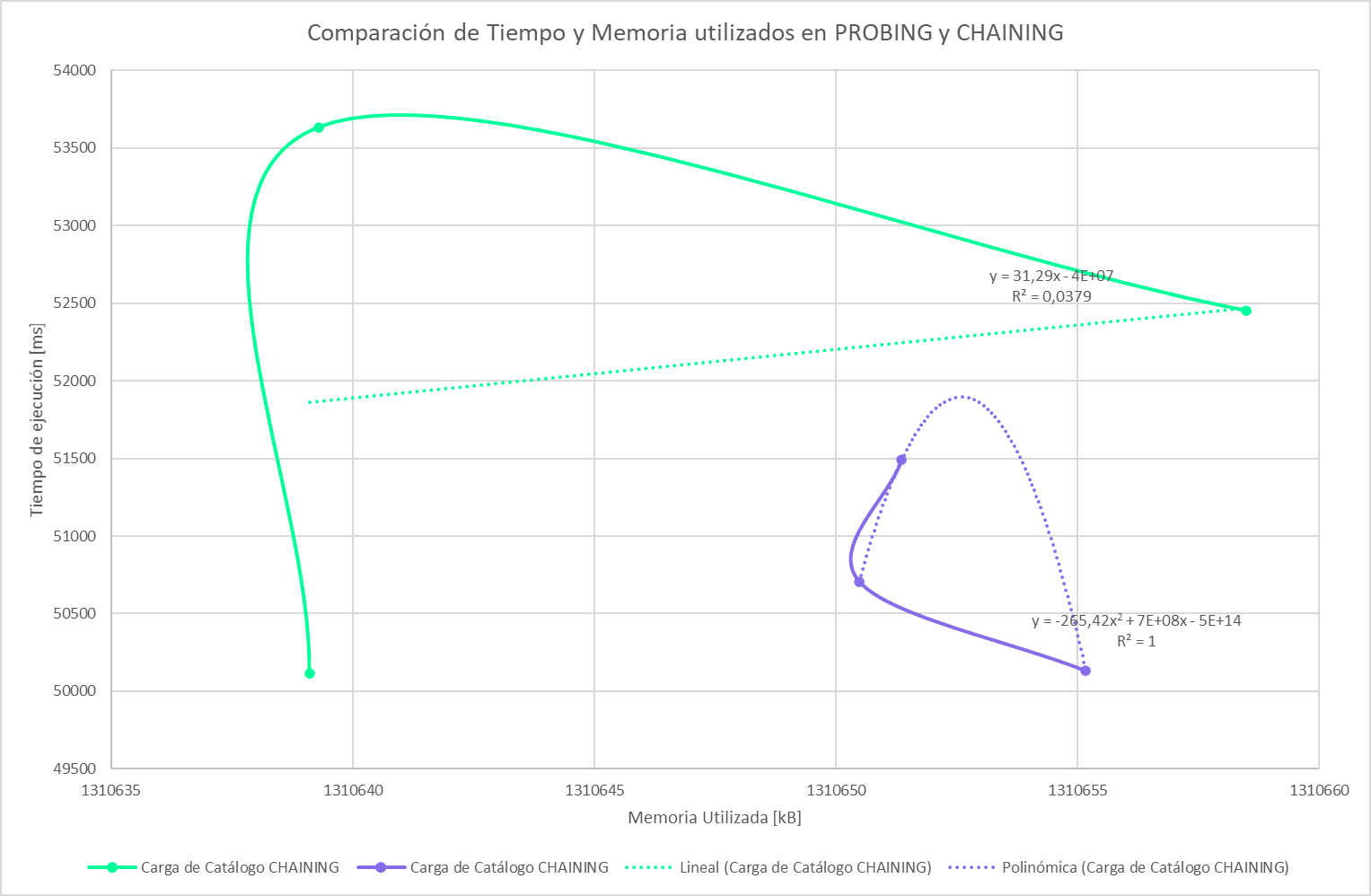
|  |  |  |
| --- | --- | --- |
| ***Carga de Catálogo CHAINING*** | | |
| **Factor de Carga (CHAINING)** | **Consumo de Datos [kB]** | **Tiempo de Ejecución [ms]** |
| 2.00 | 1310655,172 | 50133,423 |
| 4.00 | 1310650,477 | 50707,355 |
| 6.00 | 1310651,359 | 51492,145 |

Tabla 5. Comparación de consumo de datos y tiempo de ejecución para carga de catálogo con el índice por categorías utilizando CHAINING en la Maquina 2.

## **Graficas**

La gráfica generada por los resultados de las pruebas de rendimiento en la **Maquina 2.**

* Comparación de memoria y tiempo de ejecución para PROBING y CHAINING



# **Preguntas de análisis**

1. ¿Por qué en la función **getTime()** se utiliza **time.perf\_counter()** en ves de la previamente conocida **time.process\_time()**?

La función getTime() utiliza **time.perf\_counter()**  en esta ocasión porque se desea tomar el tiempo total que se demora en ejecutar las funciones de loadBooks(catalog), loadTags(catalog) y loadBooksTags(catalog) incluyendo el tiempo que se pueda demorar abriendo los csv. Esto no era necesario en los laboratorios pasados, ya que no se medía este tiempo sino el de los ordenamientos por lo cual no era necesario usar **time.perf\_counter().**

1. ¿Por qué son importantes las funciones **start()** y **stop()** de la librería **tracemalloc**?

El start() hace que la librería empiece a contar la cantidad de memoria que está distribuyendo el programa y el stop() detiene esta cuenta.

1. ¿Qué cambios percibe en el **tiempo de ejecución** al modificar el factor de carga máximo para cargar el catálogo de videos?
2. ¿Qué cambios percibe en el **consumo de memoria** al modificar el factor de carga máximo para cargar el catálogo de videos?
3. ¿Qué cambios percibe en el **tiempo de ejecución** al modificar el esquema de colisiones?, si los percibe, describa las diferencias y argumente su respuesta.
4. ¿Qué cambios percibe en el **consumo de memoria** al modificar el esquema de colisiones?, si los percibe, describa las diferencias y argumente su respuesta.

Respuesta general: Debido a que tenemos muy pocas llaves en el MAP y los hash de Python son muy buenos, es muy poco probable que de presenten colisiones, esto causa que el tiempo y el consumo de memoria no tengan cambios significativos