

Taller 2

Jose David Ramirez Beltran - 50622273
Camila Andrea Galindo Ruiz - 506222700
Miguel Daniel Ruiz Silva - 506222719

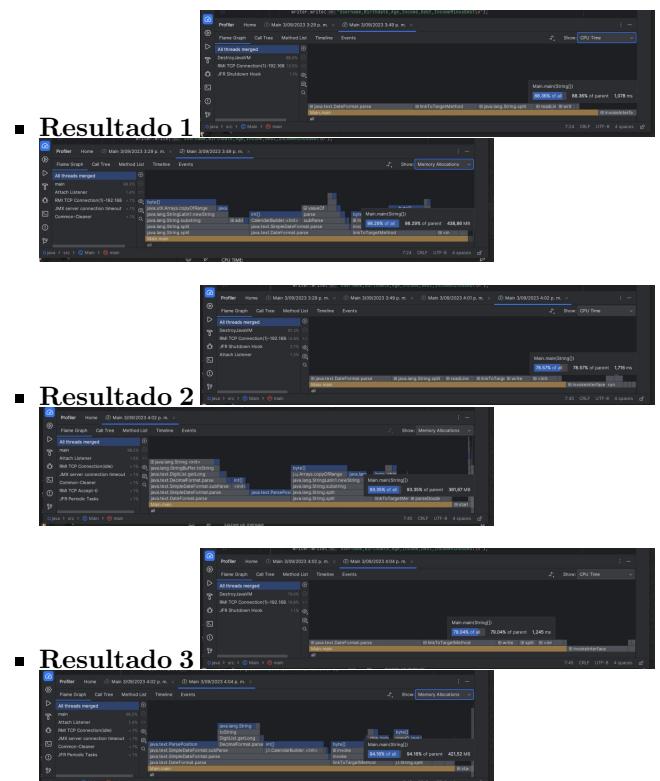
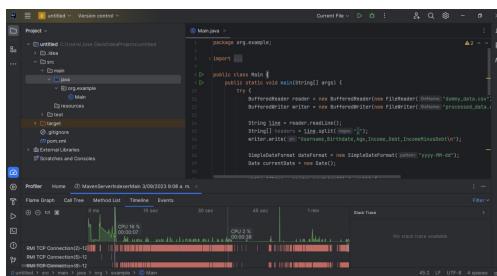
16 de septiembre de 2023

1. Introducción

El término temporal spatial o espacio temporal, en programación se refiere a dos dimensiones cruciales para comprender y optimizar el rendimiento de un programa o algoritmo. Estas especificaciones son esenciales para garantizar que una aplicación funcione eficazmente y se ajuste a las limitaciones de tiempo y espacio disponibles. Mientras que la dimensión espacial se refiere a la cantidad de memoria o espacio de almacenamiento necesario para ejecutar el programa, la dimensión temporal tiene que ver con la velocidad y la eficacia del programa. Estos dos factores deben tenerse en cuenta al desarrollar software en diversos sectores, desde aplicaciones móviles a sistemas, para lograr un equilibrio óptimo entre velocidad de ejecución y eficiencia de recursos.

La gestión de estas dimensiones temporales y espaciales implica tomar decisiones informadas sobre cómo gestionar los recursos informáticos, cómo almacenar y acceder a los datos y cómo diseñar algoritmos para lograr un rendimiento óptimo.

2. Código en Java



- Complejidad Temporal
 - La lectura del archivo de entrada línea por línea tiene una complejidad temporal lineal ($O(n)$), donde n es el número de líneas en el archivo `dummy_data.csv`.
 - La división de cada línea en valores utilizando `split(",")` también tiene una complejidad lineal en relación con el número de caracteres en la línea, por lo que se puede considerar como $O(m)$, donde m es la longitud de la línea más larga.

- La conversión de fechas y números tiene una complejidad constante para cada línea, por lo que no contribuye significativamente al análisis de complejidad.
- El cálculo de la edad y la resta de ingresos y deuda también tienen una complejidad constante por línea.
- Dado que estas operaciones se realizan para cada línea del archivo, podemos decir que la complejidad temporal total depende principalmente del número de líneas en el archivo `dummy_data.csv`, es decir, es lineal en relación con la entrada.
- Complejidad Espacial
- La cantidad de memoria utilizada durante la ejecución está relacionada con la complejidad espacial del programa, en este caso los datos se leen y escriben línea por línea, por lo que el uso de memoria es constante e independiente del tamaño de todo el archivo. Pero, hay que tener en cuenta que las líneas se almacenan en memoria mientras se procesan y se escriben en el archivo de salida.
- Ya que el programa utiliza un máximo del 16 por ciento de la memoria y se ejecuta en un minuto, podemos deducir que el uso de la memoria es racionalmente eficiente dado el tamaño del archivo de trabajo. En este caso, la complejidad espacial no parece ser un problema significativo.

3. Código Python

Primera ejecucion

```

temporal_spatial-main
main.py
#_name__ == "__main__"
import csv
import random
from datetime import date
writer = csv.writer(open("dummy_data.csv", "w"))
for _ in range(500):
    row = [
        str(random.randint(1, 1000)),
        str(date.today()),
        str(random.randint(1, 1000)),
        str(random.randint(1, 1000)),
        str(random.randint(1, 1000))
    ]
    writer.writerow(row)
if __name__ == "__main__":
    generate_data()
    save_to_csv(generate_data())
    print("Data generation and CSV creation complete.")

```

Segunda ejecucion

```

Data generation and CSV creation complete.
Process finished with exit code 0

```

Tercera ejecucion

```

Data generation and CSV creation complete.
Process finished with exit code 0

```

Complejidad Temporal:

- El bucle `for` que genera los datos se ejecuta 500² veces, lo que significa que genera un total de 250,000 registros de datos ficticios.
- Dentro del bucle, se realizan operaciones que son generar un nombre de usuario, fecha de nacimiento, ingreso, deuda, género, número de hijos y país para cada registro, esta generación es constante en relación con el número de registros que se están generando.
- El bucle de escritura a CSV también es lineal en relación con el número de registros (250,000).

Dado que todas estas operaciones son lineales en relación con el número de registros, se puede decir que la complejidad temporal es aproximadamente lineal, es decir, $O(n)$, donde n es el número de registros generados.

Complejidad Espacial:

- El código genera una lista llamada `data` que almacena todos los registros antes de escribirlos en el archivo CSV. La lista `data` contiene 250,000 elementos, uno por cada registro generado.
- Los datos en la lista `data` se escriben en un archivo CSV llamado 'dummy_data.csv' mediante el uso de la función `save_to_csv`.
- Dado que el tamaño de la lista `data` es proporcional al número de registros generados, la complejidad espacial es $O(n)$, donde n es el número de registros.
- Por lo tanto, la complejidad temporal es lineal $O(n)$ y la complejidad espacial también es lineal $O(n)$, esto significa que tanto el tiempo

de ejecución como el uso de memoria aumentan de manera lineal con la cantidad de datos generados. Teniendo en cuenta los datos de la ejecución parece ser razonablemente eficiente para el tamaño de datos generados.

4. Resultados en diferentes pc

HP ryzen 5500U

PYTHON

Ejecución 1

```
temporal_spatial-main | Version control | main.py | temporal_spatial-main11.pysrc |
Project | temporal_spatial-main | C:\Users\luis\OneDrive\Escritorio | Main.java | main.py |
... | temporal_spatial-main | |
| - temporal_spatial-main |
| | - __init__.py |
| | - dummy_data.csv |
| | - requirements.txt |
| | - README.md |
| | - process_data.csv |
| | - temporal_spatial_main |
| | - temporal_spatial_main |
| | - Scratches and Cocktails |
| |
| Services | |
| | - Python |
| | - C_Finished |
| | - man |
|
Starting chprofile profiler
Data generation and CSV creation complete.
use total de memoria: 353513272 bytes
Snapshot saved to C:/Users/luis/OneDrive/Local/databricks/PyCharm2023.2/snapshots/temporal_spatial-main11/temporal_spatial-main11.pysrc
Process finished with exit code 0
```

Ejecución 2

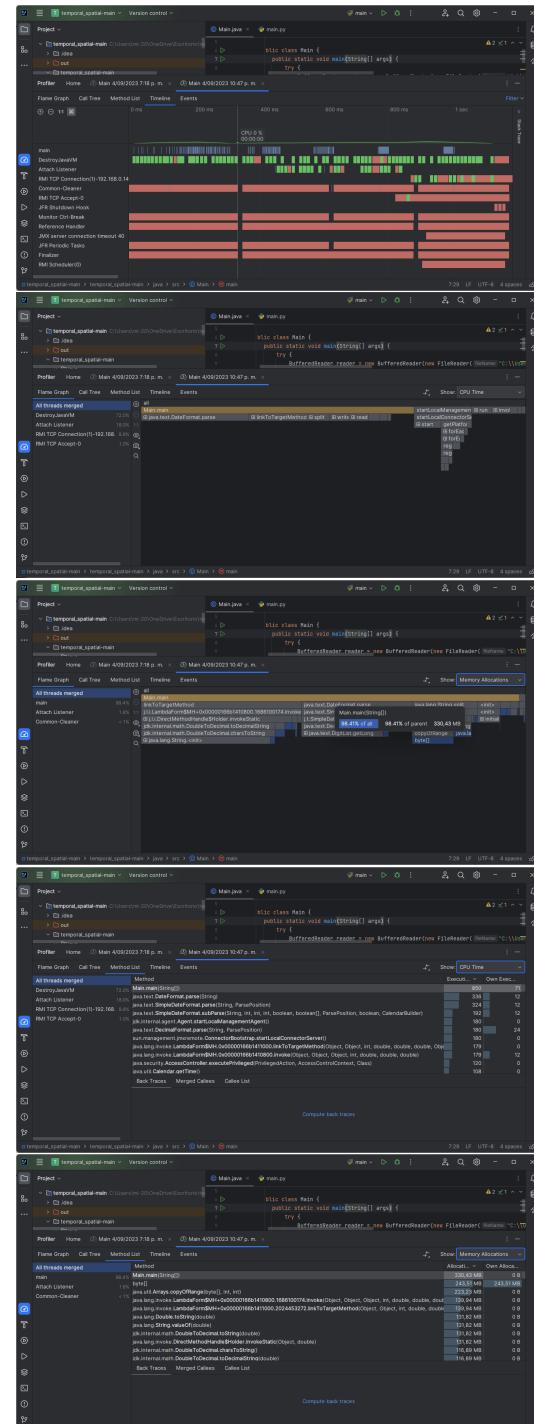
```
temporal_spatial-main | Version control | main.py | temporal_spatial-main12.pysrc |
Project | temporal_spatial-main | C:\Users\luis\OneDrive\Escritorio | Main.java | main.py |
... | temporal_spatial-main | |
| - temporal_spatial-main |
| | - __init__.py |
| | - dummy_data.csv |
| | - main.py |
| | - requirements.txt |
| | - README.md |
| | - process_data.csv |
| | - temporal_spatial_main |
| | - temporal_spatial_main |
| | - Scratches and Cocktails |
| |
| Services | |
| | - Python |
| | - C_Finished |
| | - man |
|
Starting chprofile profiler
Data generation and CSV creation complete.
use total de memoria: 353513272 bytes
Snapshot saved to C:/Users/luis/OneDrive/Local/databricks/PyCharm2023.2/snapshots/temporal_spatial-main12/temporal_spatial-main12.pysrc
Process finished with exit code 0
```

Ejecución 3

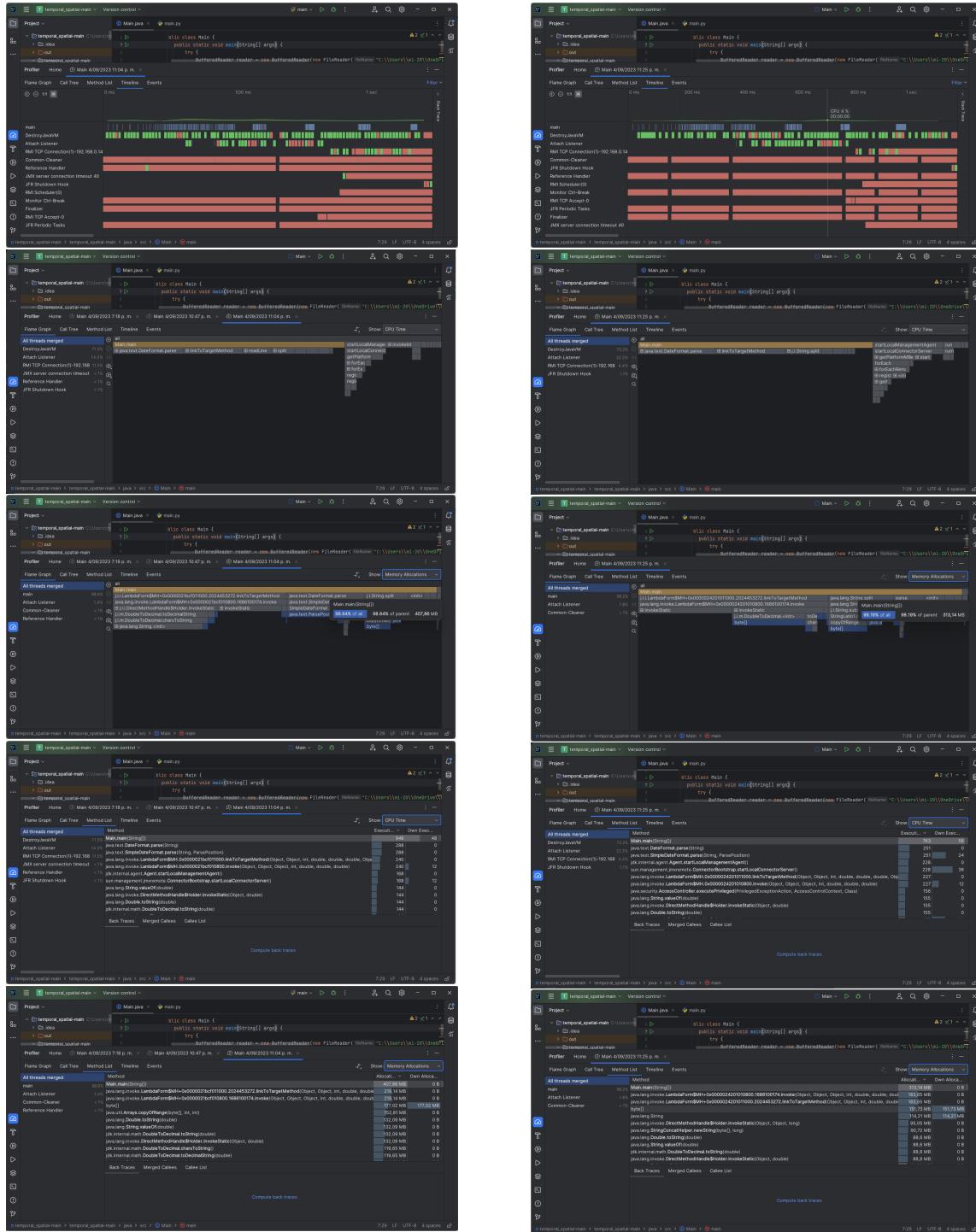
```
temporal_spatial-main | Version control | main.py | temporal_spatial-main13.pysrc |
Project | temporal_spatial-main | C:\Users\luis\OneDrive\Escritorio | Main.java | main.py |
... | temporal_spatial-main | |
| - temporal_spatial-main |
| | - __init__.py |
| | - dummy_data.csv |
| | - main.py |
| | - requirements.txt |
| | - README.md |
| | - process_data.csv |
| | - temporal_spatial_main |
| | - temporal_spatial_main |
| | - Scratches and Cocktails |
| |
| Services | |
| | - Python |
| | - C_Finished |
| | - man |
|
Starting chprofile profiler
Data generation and CSV creation complete.
use total de memoria: 353513272 bytes
Snapshot saved to C:/Users/luis/OneDrive/Local/databricks/PyCharm2023.2/snapshots/temporal_spatial-main13/temporal_spatial-main13.pysrc
Process finished with exit code 0
```

JAVA

Ejecución 1



Ejecución 2

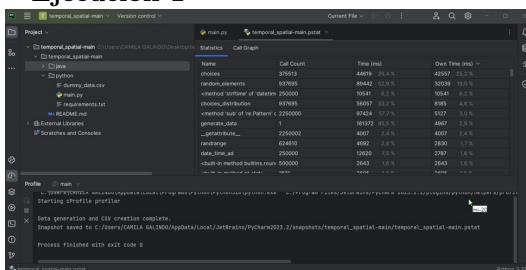


Lenovo AMD 3020e

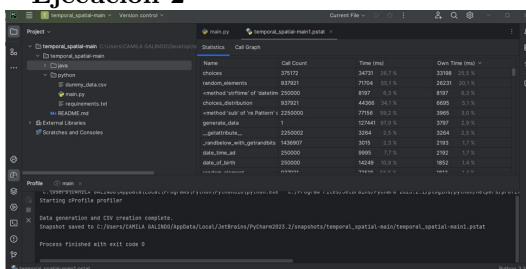
Ejecución 3

PYTHON

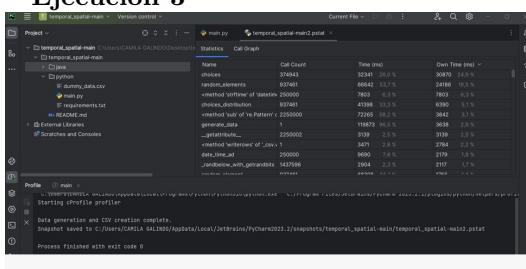
Ejecución 1



Ejecución 2

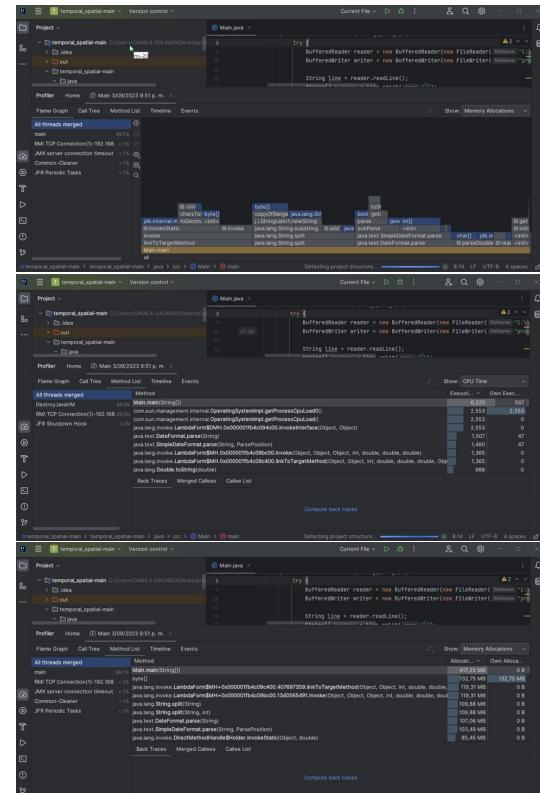
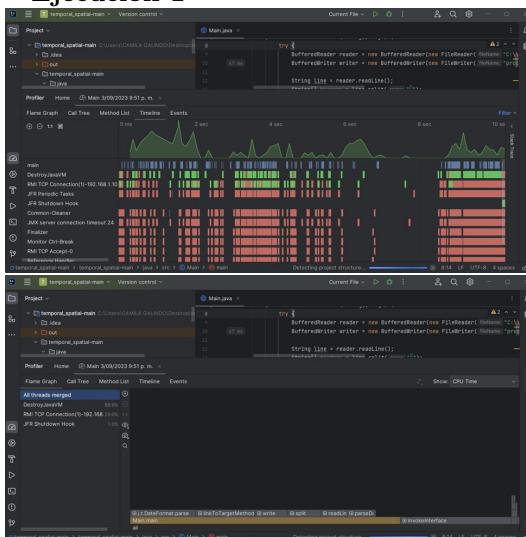


Ejecución 3

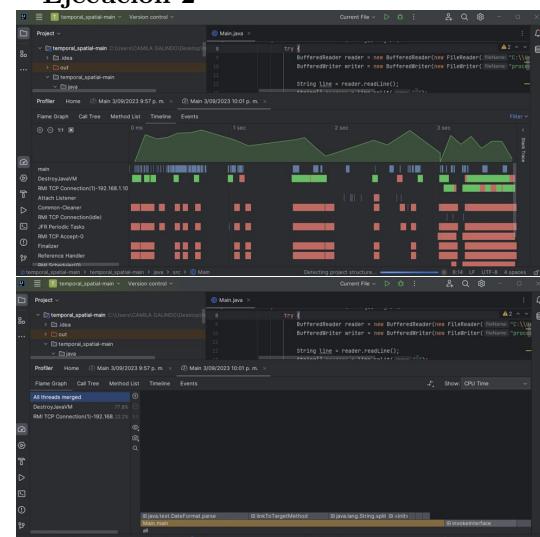


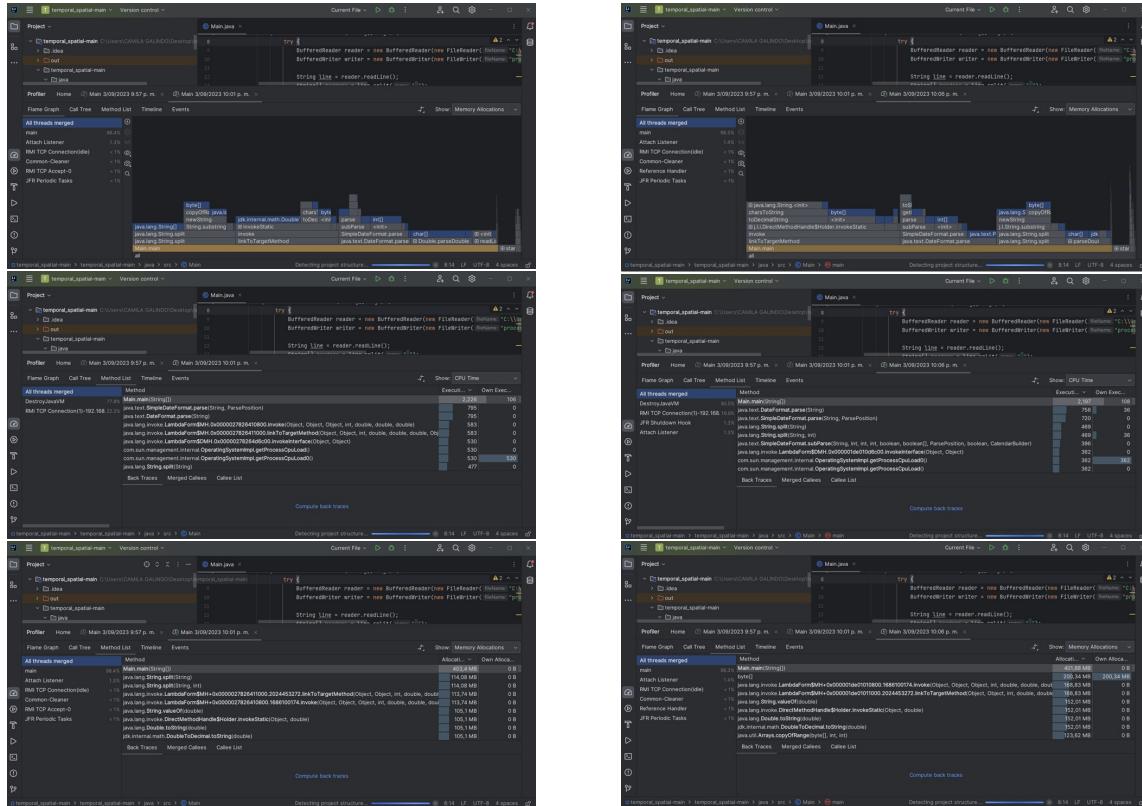
JAVA

Ejecución 1

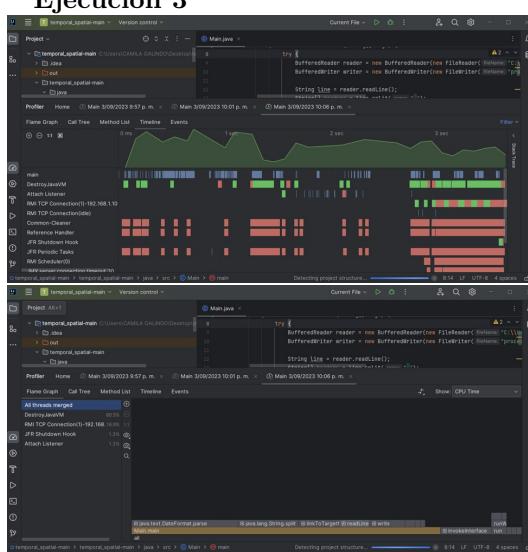


Ejecución 2





Ejecución 3



5. Resumen resultados espacio-temporales

CPU	Nombre	Procesador	RAM	Tiempo de Ejecución (seg)	Memoria Usada (bytes)
Jose	HP	AMD E1 1200 APU with Radeon(tm) HD Graphics 1.40 GHz	8.00 GB (12.6 GB usadas)	1.97	438.86
Miguel	HP	AMD Ryzen 5 3500U with Radeon Graphics 2.10 GHz	8.00 GB (7.33 GB usadas)	0.85	350.43
Carola	Laptops	AMD 3800e with Radeon Graphics 1.20 GHz	8.00 GB (5.39 GB usadas)	0.23	457.55
Total					814.24
<hr/>					
Jose	HP	AMD E1 1200 APU with Radeon(tm) HD Graphics 1.40 GHz	16.00 GB (13.2 GB usadas)	0.017	361.47
Miguel	HP	AMD Ryzen 5 3500U with Radeon Graphics 2.10 GHz	8.00 GB (7.33 GB usadas)	0.94	407.86
Carola	Laptops	AMD 3800e with Radeon Graphics 1.20 GHz	8.00 GB (5.39 GB usadas)	2.405	399.92
Total					1348.84

6. Conclusiones

En conclusión, la comprensión de estas dimensiones y su adecuada aplicación en la programación es esencial para el éxito de un proyecto de desarrollo de software en un mundo cada vez más dependiente de la tecnología, así como para la eficacia de las propias aplicaciones y que podamos ver que tan eficiente es un código comparado con otro, así mismo

como sus diferencias entre equipos y las capacidades y requerimientos que cada programa exige a un dispositivo o equipo.

7. Bibliografia

<https://github.com>