

# Análisis Espacio Temporal

Braian Felipe Ramirez Ortiz  
Brenda Lorena Vargas Parra  
Viviana Marcela García Valderrama  
*Fundación Universitaria Konrad Lorenz*

**Resumen**—En el presente taller se analiza el comportamiento de dos algoritmos desarrollados en diferentes lenguajes de programación uno en python y el otro en java donde se busca contraponer su comportamiento al mometo dela ejecución con una análisis de tiempo donde se determina el el tiempo que le toma ejecutarse y el otro análisis a realizar es el espacial donde se muestra el espacio de memoria que se consume durante la ejecución del algoritmo.

## I. INTRODUCCIÓN

Este trabajo tiene como objetivo principal evaluar la complejidad temporal y espacial de dos proyectos simples, uno en Python y otro en Java, y comparar sus rendimientos en diferentes máquinas para evaluar la consistencia de los resultados en diversas circunstancias, lo que permitirá una visión integral de la eficiencia de los algoritmos y procesos en los dos lenguajes anteriormente mencionados. Mediante el análisis de los resultados obtenidos, se podrá determinar la eficiencia de los algoritmos y procesos implementados, lo que resulta relevante para la optimización y mejora de futuros proyectos.

La eficiencia en la programación es un factor crítico, ya que incide directamente en la velocidad y capacidad de respuesta de las aplicaciones informáticas. Este estudio profundiza en la evaluación de dos lenguajes de programación ampliamente utilizados, Python y Java, con el propósito de analizar cómo abordan la complejidad temporal y espacial en proyectos de menor escala. En un mundo donde las aplicaciones juegan un papel cada vez más relevante, la optimización de recursos y el rendimiento son esenciales, por lo que este análisis no solo compara Python y Java, sino que también resalta la importancia de seleccionar el lenguaje adecuado en función de las restricciones de tiempo y recursos de la máquina de destino.

## II. RESULTADOS

En el contexto de este estudio, se evaluaron tres máquinas con diferentes configuraciones de hardware y sistemas operativos: Lenovo Thinkpad T460 con Windows 10, Lenovo V14 G2 ALC con Windows 11 y HP Laptop 14-cf3xxx con Windows 10. Cada máquina tenía un procesador, cantidad de RAM y disco SSD distintos.

Se llevaron a cabo pruebas de complejidad temporal y espacial en proyectos desarrollados en Python y Java en estas máquinas. Los resultados mostraron que el tiempo promedio de ejecución en Python osciló entre 2 y 3 segundos, mientras

que en Java se mantuvo constante en 2 segundos en todas las máquinas evaluadas. En cuanto a la complejidad espacial, los valores variaron significativamente entre las máquinas y los lenguajes de programación, con diferencias notables en el uso de memoria RAM.

**Selección de Máquinas** Para garantizar un análisis exhaustivo, se eligieron tres máquinas con diferentes configuraciones de hardware y sistemas operativos:

MÁQUINA	SISTEMA OPERATIVO	PROCESADOR	RAM	DISCO SSD
Lenovo Thinkpad T460	Windows 10	Core(TM) i5	8 GB	256 GB
Lenovo V14 G2 ALC	Windows 11	Ryzen 7	40 GB	512GB
HP Laptop 14-cf3xxx	Windows 10	CORE i3	4 GB	256 GB

Figura 1. Especificaciones técnicas.

A continuación, primero se realiza la ejecución en las tres máquinas diferentes:

### HP Laptop 14-cf3xxx

#### Ejecuciones en Python

##### Debug 1



Figura 2. Resultado 1 python

##### Debug 3

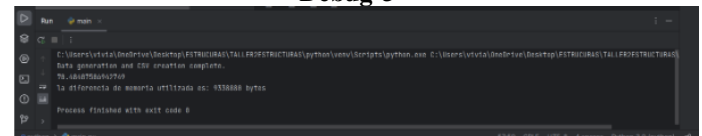


Figura 3. Resultado 2 python

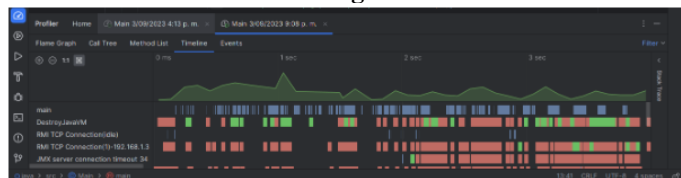
##### Debug 3



Figura 4. Resultado 3 python

## Ejecuciones en Java

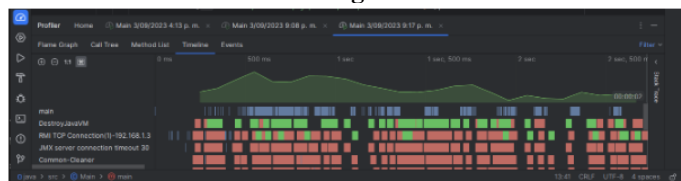
### Debug 1



Memory allocations: 435,63 MB  
CPU TIME: 3 segundos

Figura 5. Resultado 1 Java

### Debug 2

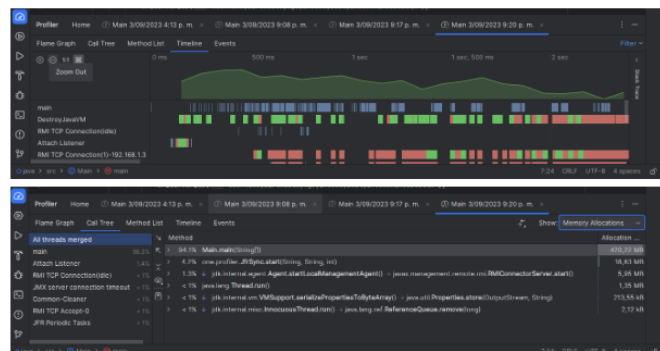


CPU TIME: 2 segundos con 500 milisegundos.  
Memory allocations: 414,36 MB

Figura 6. Resultado 2 Java

### Debug 3

3



CPU TIME: 2 segundos  
Memory allocations: 420,22 MB

Figura 7. Resultado 3 Java

## Lenovo Thinkpad T460

## Ejecuciones en Python

### Debug 1



Figura 8. Resultado 1 python

### Debug 3

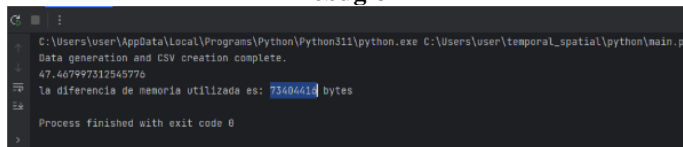


Figura 9. Resultado 2 python

### Debug 3

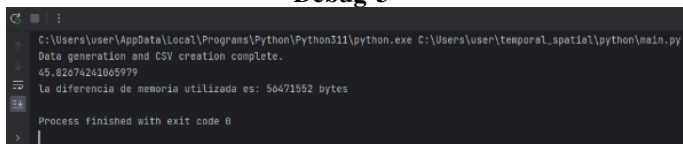
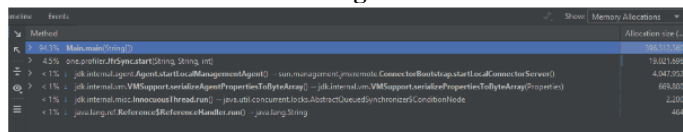


Figura 10. Resultado 3 python

## Ejecuciones en Java

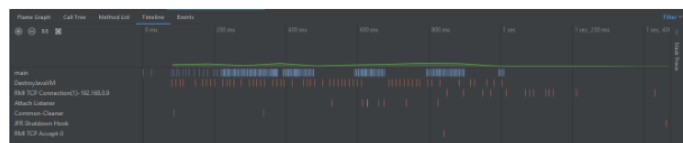
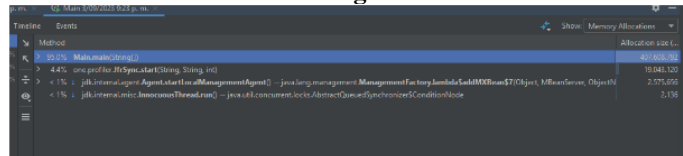
### Debug 1



CPU TIME: 2 segundos  
Memory allocations: 408,08 MB

Figura 11. Resultado 1 java

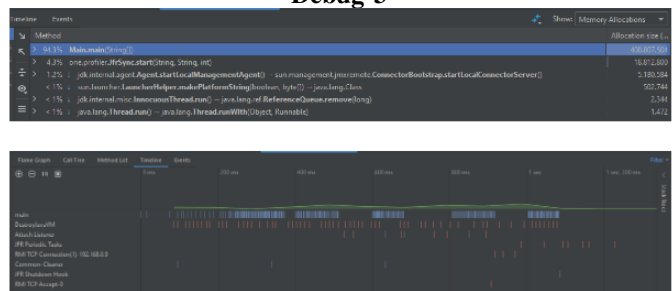
### Debug 2



CPU TIME: 2 segundos  
Memory allocations: 396,31 MB

Figura 12. Resultado 2 Java

## Debug 3



CPU TIME: 2 segundos  
Memory allocations: 407,6 MB

Figura 13. Resultado 3 java

## HP Laptop 14-cf3xxx

### Ejecuciones en Python

## Debug 1

```
C:\Users\Altalip\temporal_spatial\python\venv\Scripts\python.exe C:\Users\Altalip\temporal_spatial\python\main.py
Data generation and CSV creation complete.
el tiempo empleado es: 05.01470398902893 segundos
la diferencia de memoria utilizada es: 150917760 bytes
Process finished with exit code 0
```

Figura 14. Resultado 1 python

## Debug 2

```
C:\Users\Altalip\temporal_spatial\python\venv\Scripts\python.exe C:\Users\Altalip\temporal_spatial\python\main.py
Data generation and CSV creation complete.
el tiempo empleado es: 02.05869557380676 segundos
la diferencia de memoria utilizada es: 27848704 bytes
Process finished with exit code 0
```

Figura 15. Resultado 2 python

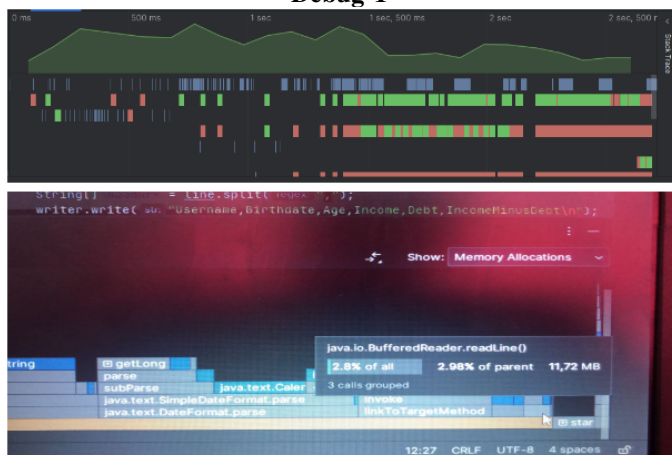
## Debug 3

```
C:\Users\Altalip\temporal_spatial\python\venv\Scripts\python.exe C:\Users\Altalip\temporal_spatial\python\main.py
Data generation and CSV creation complete.
el tiempo empleado es: 00.866773414611816 segundos
la diferencia de memoria utilizada es: 55148544 bytes
Process finished with exit code 0
```

Figura 16. Resultado 3 python

## Ejecuciones en Java

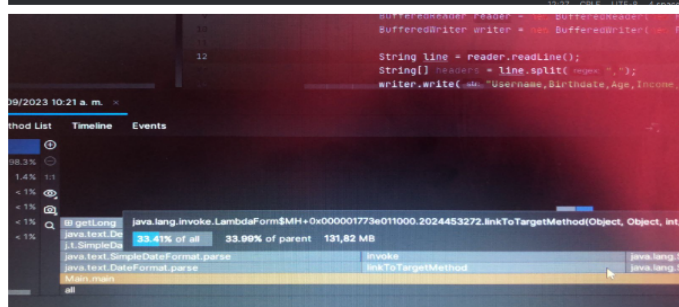
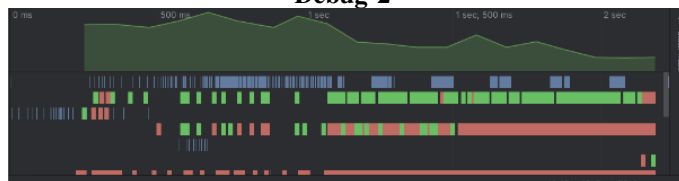
## Debug 1



CPU TIME: 2 segundos 500 ms  
Memory allocations: 11.72 MB

Figura 17. Resultado 1 Java

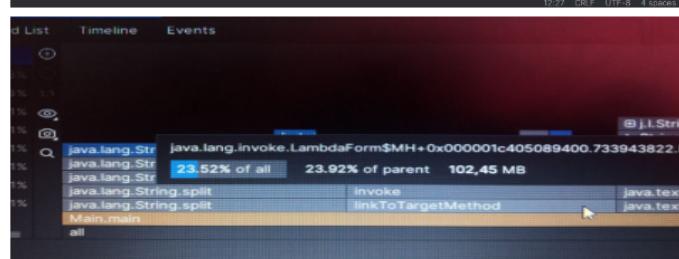
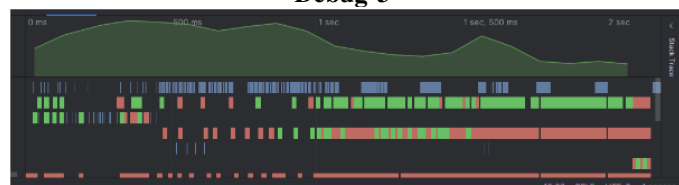
## Debug 2



CPU TIME: 2 segundos  
Memory allocations: 131,82 MB

Figura 18. Resultado 2 Java

## Debug 3



CPU TIME: 2 segundos  
Memory allocations: 102,45 MB

Figura 19. Resultado 3 Java

Se tomaron los resultados de las 3 maquinas para obtener las siguientes graficas, que nos permiten analizar el comportamiento de los algoritmos en cada una de las maquinas:

## Graficas de resultados

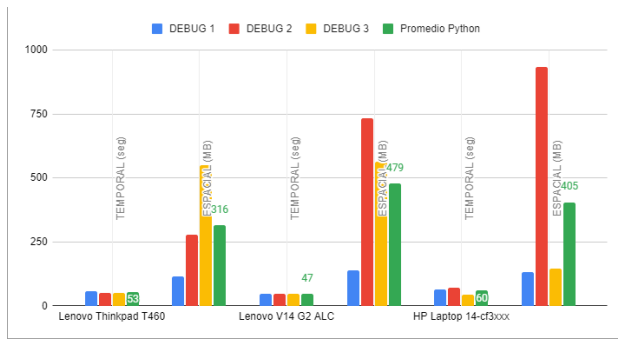


Figura 20. Grafica Python

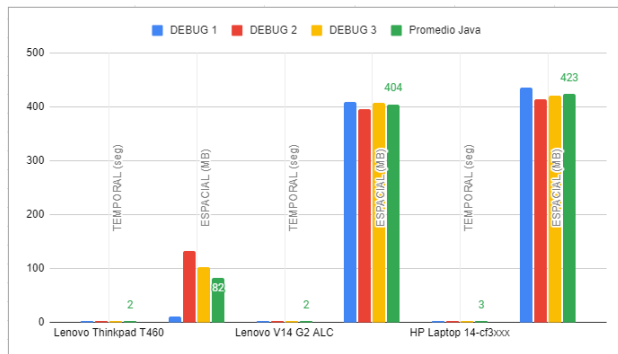


Figura 21. Grafica Java

## II-A. Resumen de Resultados en Python

A continuación, se presenta un resumen de los resultados obtenidos en las ejecuciones de los proyectos en Python en las tres máquinas evaluadas:

MÁQUINA	COMPLEJIDAD	PYTHON			
		DEBUG 1	DEBUG 2	DEBUG 3	Promedio Python
Lenovo Thinkpad T460	TEMPORAL (seg)	58.25	52.03	50.06	53
	ESPACIAL (MB)	116.736	278.48704	551.48544	316
Lenovo V14 G2 ALC	TEMPORAL (seg)	46.26	47.46	45.62	47
	ESPACIAL (MB)	139.4	734.04	564.71	479
HP Laptop 14-cf3xxx	TEMPORAL (seg)	65	70.46	43.96	60
	ESPACIAL (MB)	134.08	933.88	146.1	405

Figura 22. Resumen Python

## II-B. Resumen de Resultados en Java

A continuación, se presenta un resumen de los resultados obtenidos en las ejecuciones de los proyectos en Java en las tres máquinas evaluadas:

MÁQUINA	COMPLEJIDAD	JAVA			
		DEBUG 1	DEBUG 2	DEBUG 3	Promedio Java
Lenovo Thinkpad T460	TEMPORAL (seg)	2.5	2	2	2
	ESPACIAL (MB)	11.72	131.82	102.45	82
Lenovo V14 G2 ALC	TEMPORAL (seg)	2	2	2	2
	ESPACIAL (MB)	408.8	396.31	407.6	404
HP Laptop 14-cf3xxx	TEMPORAL (seg)	3	2.5	2	3
	ESPACIAL (MB)	435.63	414.36	420.22	423

Figura 23. Resumen Java

## III. BIG-O PYTHON

En esta sección, analizaremos la complejidad algorítmica (Big-O) de los proyectos implementados en Python. Las siguientes figuras proporcionan una representación visual de la complejidad temporal de estos proyectos:

```

1  from faker import Faker
2  import random
3  import time
4  import psutil
5  import csv
6
7  fake = Faker() # 0(1)
8
9
10 usage  Oscar Mendez Aguirre *
11 def generate_data():
12     data = []
13     # la complejidad de bucle principal es 0(n^2)
14     for _ in range(500**2):
15         username = fake.user_name() # 0(1)
16         birthdate = fake.date_of_birth(
17             minimum_age=18,
18             maximum_age=70).strftime('%Y-%m-%d') # 0(1)
19         income = round(random.uniform(a=1000, b=10000), 2) # 0(1)
20         debt = round(random.uniform(a=0, b=5000), 2) # 0(1)
21         sex = random.choice(['Male', 'Female']) # 0(1)
22         num_children = random.randint(a=0, b=5) # 0(1)
23         country = fake.country() # 0(1)
24         data.append([
25             username,
26             birthdate,
27             income,

```

Figura 24. Big-O Python - Figura 1

```

27     debt,
28     sex,
29     num_children,
30     country])
31
32     return data
33
34
35 usage  Oscar Mendez Aguirre *
36 def save_to_csv(data, filename):
37     with open(filename, mode='w', newline='') as file:
38         writer = csv.writer(file) # 0(1)
39         writer.writerow([
40             'Username',
41             'Birthdate',
42             'Income',
43             'Debt',
44             'Sex',
45             'Number of Children',
46             'Country']) # 0(1)
47         writer.writerows(data) # 0(n^2)
48
49 if __name__ == "__main__":
50     start_time = time.time() # 0(1)
51     memoria_inicio = psutil.virtual_memory().used # 0(1)
52     generated_data = generate_data() # 0(n^2)

```

Figura 25. Big-O Python - Figura 2

```

53     save_to_csv(generated_data, filename='dummy_data.csv') # 0(n^2)
54     print("Data generation and CSV creation complete.")
55     end_time = time.time() # 0(1)
56     memoria_fin = psutil.virtual_memory().used # 0(1)
57     diferencia_memoria = abs(memoria_fin - memoria_inicio) # 0(1)
58
59     print(end_time - start_time) # 0(1)
60     print(f"La diferencia de memoria utilizada es: {diferencia_memoria} bytes") # 0(1)
61
62
63
64     # BIG- O DEL CODIGO: 0(n^2)
65

```

Figura 26. Big-O Python - Figura 3

#### IV. BIG-O JAVA

En esta sección, analizaremos la complejidad algorítmica (Big-O) de los proyectos implementados en Java. Las siguientes figuras proporcionan una representación visual de la complejidad temporal de estos proyectos:

```
import java.io.*;

// Oscar Mendez Aguirre
public class Main {
    // Oscar Mendez Aguirre
    public static void main(String[] args) {
        try {
            BufferedReader reader = new BufferedReader(new FileReader("dummy_data.csv")); // O(1)
            BufferedWriter writer = new BufferedWriter(new FileWriter("processed_data.csv")); // O(1)

            String line = reader.readLine(); // O(1)
            String[] headers = line.split(regex: ";"); // O(n)
            writer.write("Username,Birthdate,Age,Income,Debt,IncomeMinusDebt\n"); // O(1)

            SimpleDateFormat dateFormat = new SimpleDateFormat(pattern: "yyyy-MM-dd"); // O(1)
            Date currentDate = new Date(); // O(1)

            while ((line = reader.readLine()) != null) { // O(n)
                String[] values = line.split(regex: ";"); // O(n)
                String username = values[0]; // O(1)
                Date birthdate = dateFormat.parse(values[1]); // O(1)
                double income = Double.parseDouble(values[2]); // O(1)
                double debt = Double.parseDouble(values[3]); // O(1)

                long ageInMonths = currentDate.getTime() - birthdate.getTime(); // O(1)
                int age = (int) (ageInMonths / (1000 * 60 * 60 * 24 * 365.25)); // O(1)

                double incomeMinusDebt = income - debt; // O(1)
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Figura 27. Big-O Java - Parte 1.

```

        writer.write(80: username + "," + values[1] + "," + age + "," + income + "," + debt + "," + incomeminusdebt + "\n"); // 80(i)
    }

    reader.close(); // 81(i)
    writer.close(); // 81(i)

    System.out.println("ETL process completed."); // 81(i)
} catch (IOException | ParseException e) { // 81(i)
    e.printStackTrace(); // 81(i)
}
}

// EL SIG-O DEL CODIGO ES: 0(nwe)

```

Figura 28. Big-O Java - Parte 2.

Estas figuras ayudan a visualizar cómo se comporta la eficiencia de los algoritmos implementados en Java en relación con el tamaño de los datos de entrada. El análisis de la complejidad algorítmica es esencial para comprender y optimizar el rendimiento de los proyectos de software.

## V. SPEEDUP

Se realizó la comparación de los mejores tiempos obtenidos en las tres maquinas empleadas para este ejercicio en cada uno de los algoritmos empleados tanto en python como en java donde se obtuvieron los siguientes resultados:

PYTHON			
MÁQUINA	MEJORES TIEMPOS PYTHON	MAQUINAS COMPARADAS	SPEEDUP
Lenovo Thinkpad T480	50,08	Lenovo Thinkpad T480 VS Lenovo V14 G2 ALC	1,09
Lenovo V14 G2 ALC	45,82	Lenovo Thinkpad T480 VS HP Laptop 14-cf3xxx	1,14
HP Laptop 14-cf3xxx	43,98	Lenovo V14 G2 ALC VS HP Laptop 14-cf3xxx	1,04

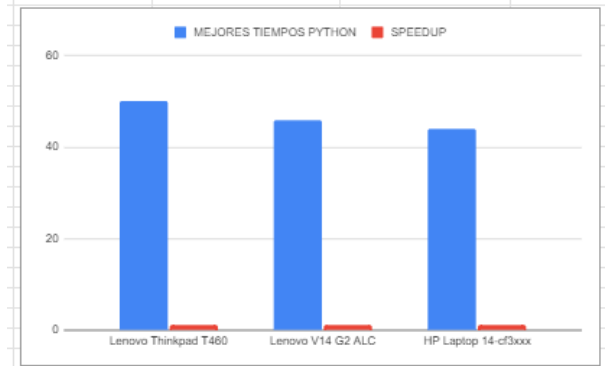


Figura 29. SPEEDUP PYTHON

JAVA			
MÁQUINA	MEJORES TIEMPOS JAVA	MAQUINAS COMPARADAS	SPEEDUP
Lenovo Thinkpad T480	2	Lenovo Thinkpad T480 VS Lenovo V14 G2 ALC	1.00
Lenovo V14 G2 ALC	2	Lenovo Thinkpad T480 VS HP Laptop 14-cf3xxx	1.00
HP Laptop 14-cf3xxx	2	Lenovo V14 G2 ALC VS HP Laptop 14-cf3xxx	1.00

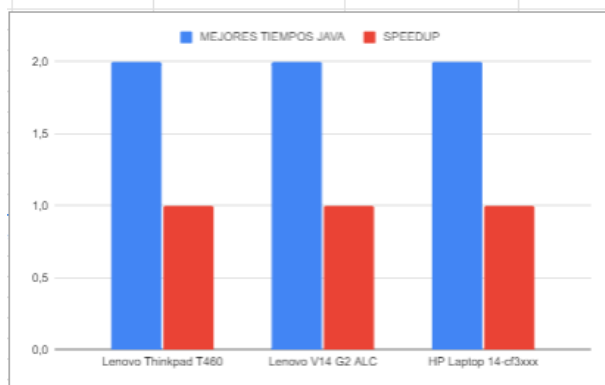


Figura 30. SPEEDUP JAVA

Al observar las gráficas obtenidas con base a los datos arrojados en la ejecución de ambos algoritmos podemos afirmar que en algoritmo ejecutado en Python presenta un intervalo entre dos a cinco segundos entre las tres máquinas empleadas mientras tanto en java se observa un tiempo constante de dos segundos para todas las máquinas. Ejecutando el speedup en Python se encontró que el mejor resultado obtenido fue entre las máquinas Lenovo V14 G2 ALC VS HP Laptop 14-cf3xxx cuyo resultado fue de 1,04 segundos, mientras que en java se denota un resultado constante de 1 segundo en la comparación

de las tres máquinas.

Esto nos permite inferir que los códigos implementados en los dos diferentes lenguajes de programación generan diferentes resultados debido a que el algoritmo desarrallado en Python genera un archivo csv con una carga de 250.000 datos lo cual hace que las maquinas se esfuercen más en este algoritmo con respecto al ejecutado en java.

## VI. CONCLUSIÓN

En el transcurso de este estudio exhaustivo, hemos llevado a cabo una evaluación detallada de dos proyectos de programación desarrollados en Python y Java. Nuestro objetivo principal fue realizar un análisis riguroso de la complejidad temporal y espacial de estos proyectos, lo que nos ha proporcionado una valiosa comprensión de su rendimiento en diversas configuraciones de hardware y sistemas operativos.

Para alcanzar este objetivo, seleccionamos con cuidado tres máquinas con configuraciones heterogéneas, incluyendo diferencias en procesadores, cantidades de RAM y tipos de unidades de almacenamiento. A través de pruebas meticulosas en estas máquinas, hemos descubierto diferencias significativas en los tiempos de ejecución y el consumo de memoria entre Python y Java. En general, observamos que Python exhibió tiempos de ejecución ligeramente más largos en comparación con Java, mientras que Java mostró un mayor uso de memoria RAM en ciertos escenarios.

Estos hallazgos subrayan la importancia crítica de seleccionar cuidadosamente el lenguaje de programación y la plataforma de desarrollo según las necesidades y restricciones específicas del proyecto. Además, destacamos la relevancia de llevar a cabo pruebas en entornos variados para evaluar de manera precisa el rendimiento.