

Taller 2

Jose David Ramirez Beltran - 506222723

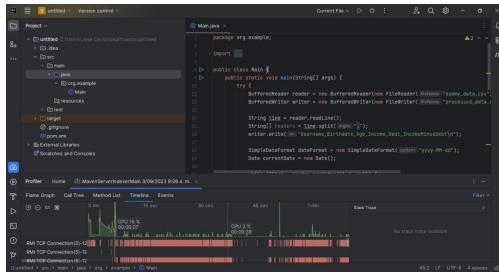
10 de septiembre de 2023

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.

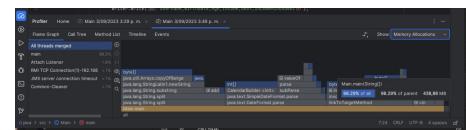
2. Código en Java



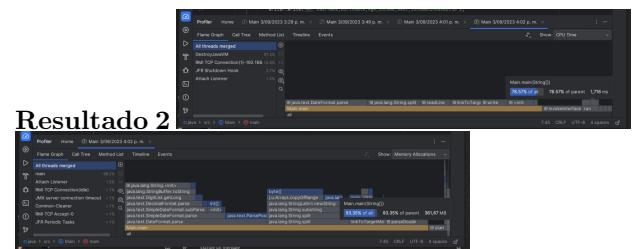
The screenshot shows an IDE interface with a project named "untitled" containing a package "org.example" with a class "Main". The code reads a CSV file "dummy_data.csv" using BufferedReader and StringTokenizer. It processes each line, splits it by commas, and prints the results. A Java Profiler window is overlaid on the IDE, showing a timeline of operations and memory allocations.

```
package org.example;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;
public class Main {
    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(new FileReader("dummy_data.csv"));
        BufferedWriter writer = new BufferedWriter(new FileWriter("processado.csv"));
        String line = reader.readLine();
        while (line != null) {
            String[] values = line.split(",");
            writer.write(values[0] + "," + values[1] + "," + values[2] + "," + values[3] + "\n");
            line = reader.readLine();
        }
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        Date currentDate = new Date();
        writer.write(dateFormat.format(currentDate));
        writer.close();
        reader.close();
    }
}
```

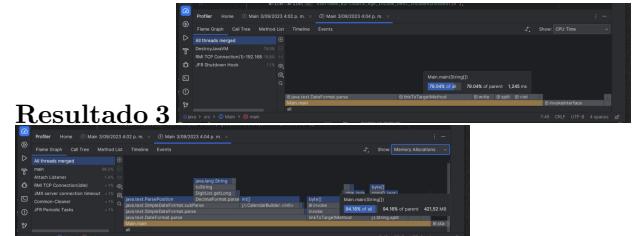
■ Resultado 1



■ Resultado 2



■ Resultado 3



■ 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

```

Project: temporal_spatial-main
  - temporal_spatial-main
    - .idea
    - java
    - python
      - dummy_data.csv
      - main.py
      - requirements.txt
    - resources
    - temporal_spatial-main.egg-info
  - External Libraries
  - Scratches and Consoles

Run | Run main
Current File: main.py
requirements.txt
main.py
-----
1  #!/usr/bin/env python
2  # This script generates 250,000 rows of data for a CSV file.
3  # The data includes columns for name, age, gender, number of children, country, and income.
4  # It uses a loop to generate the data and then writes it to a CSV file.
5
6  import csv
7  import random
8
9  def generate_data():
10    data = []
11    for _ in range(250000):
12      name = f"User_{random.randint(1, 1000000)}-{random.randint(1, 1000000)}"
13      age = random.randint(18, 65)
14      gender = random.choice(['Male', 'Female'])
15      num_children = random.randint(0, 5)
16      country = random.choice(['USA', 'Canada', 'Mexico', 'Brazil', 'Argentina', 'Spain', 'Germany', 'UK', 'Australia', 'New Zealand'])
17      income = random.randint(20000, 100000)
18
19      data.append([name, age, gender, num_children, country, income])
20
21    return data
22
23  if __name__ == "__main__":
24    generated_data = generate_data()
25    save_to_csv(generated_data, filename='dummy_data.csv')
26    print("Data generation and CSV creation complete.")

generate_data() : for _ in range(250000)

Process finished with exit code 0

```

Segunda ejecucion

```

Run | Run main
Current File: main.py
requirements.txt
main.py
-----
1  #!/usr/bin/env python
2  # This script generates 250,000 rows of data for a CSV file.
3  # The data includes columns for name, age, gender, number of children, country, and income.
4  # It uses a loop to generate the data and then writes it to a CSV file.
5
6  import csv
7  import random
8
9  def generate_data():
10    data = []
11    for _ in range(250000):
12      name = f"User_{random.randint(1, 1000000)}-{random.randint(1, 1000000)}"
13      age = random.randint(18, 65)
14      gender = random.choice(['Male', 'Female'])
15      num_children = random.randint(0, 5)
16      country = random.choice(['USA', 'Canada', 'Mexico', 'Brazil', 'Argentina', 'Spain', 'Germany', 'UK', 'Australia', 'New Zealand'])
17      income = random.randint(20000, 100000)
18
19      data.append([name, age, gender, num_children, country, income])
20
21    return data
22
23  if __name__ == "__main__":
24    generated_data = generate_data()
25    save_to_csv(generated_data, filename='dummy_data.csv')
26    print("Data generation and CSV creation complete.")

Process finished with exit code 0

```

Tercera ejecucion

```

Run | Run main
Current File: main.py
requirements.txt
main.py
-----
1  #!/usr/bin/env python
2  # This script generates 250,000 rows of data for a CSV file.
3  # The data includes columns for name, age, gender, number of children, country, and income.
4  # It uses a loop to generate the data and then writes it to a CSV file.
5
6  import csv
7  import random
8
9  def generate_data():
10    data = []
11    for _ in range(250000):
12      name = f"User_{random.randint(1, 1000000)}-{random.randint(1, 1000000)}"
13      age = random.randint(18, 65)
14      gender = random.choice(['Male', 'Female'])
15      num_children = random.randint(0, 5)
16      country = random.choice(['USA', 'Canada', 'Mexico', 'Brazil', 'Argentina', 'Spain', 'Germany', 'UK', 'Australia', 'New Zealand'])
17      income = random.randint(20000, 100000)
18
19      data.append([name, age, gender, num_children, country, income])
20
21    return data
22
23  if __name__ == "__main__":
24    generated_data = generate_data()
25    save_to_csv(generated_data, filename='dummy_data.csv')
26    print("Data generation and CSV creation complete.")

Process finished with exit code 0

```

Complejidad Temporal:

- El bucle `for` que genera los datos se ejecuta 500^2 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

```
Starting profile profiler
Data generation and CSV creation complete.
File total de memoria: 395853737 bytes
Snapshot saved to C:/Users/HP/AppData/Local/Temp/Pycharm2023.2/snapshots/temporal_spatial/main/Temps
Process finished with exit code 0
```

Ejecución 2

```
Starting profile profiler
Data generation and CSV creation complete.
File total de memoria: 395765672 bytes
Snapshot saved to C:/Users/HP/AppData/Local/Temp/Pycharm2023.2/snapshots/temporal_spatial/main/Temps
Process finished with exit code 0
```

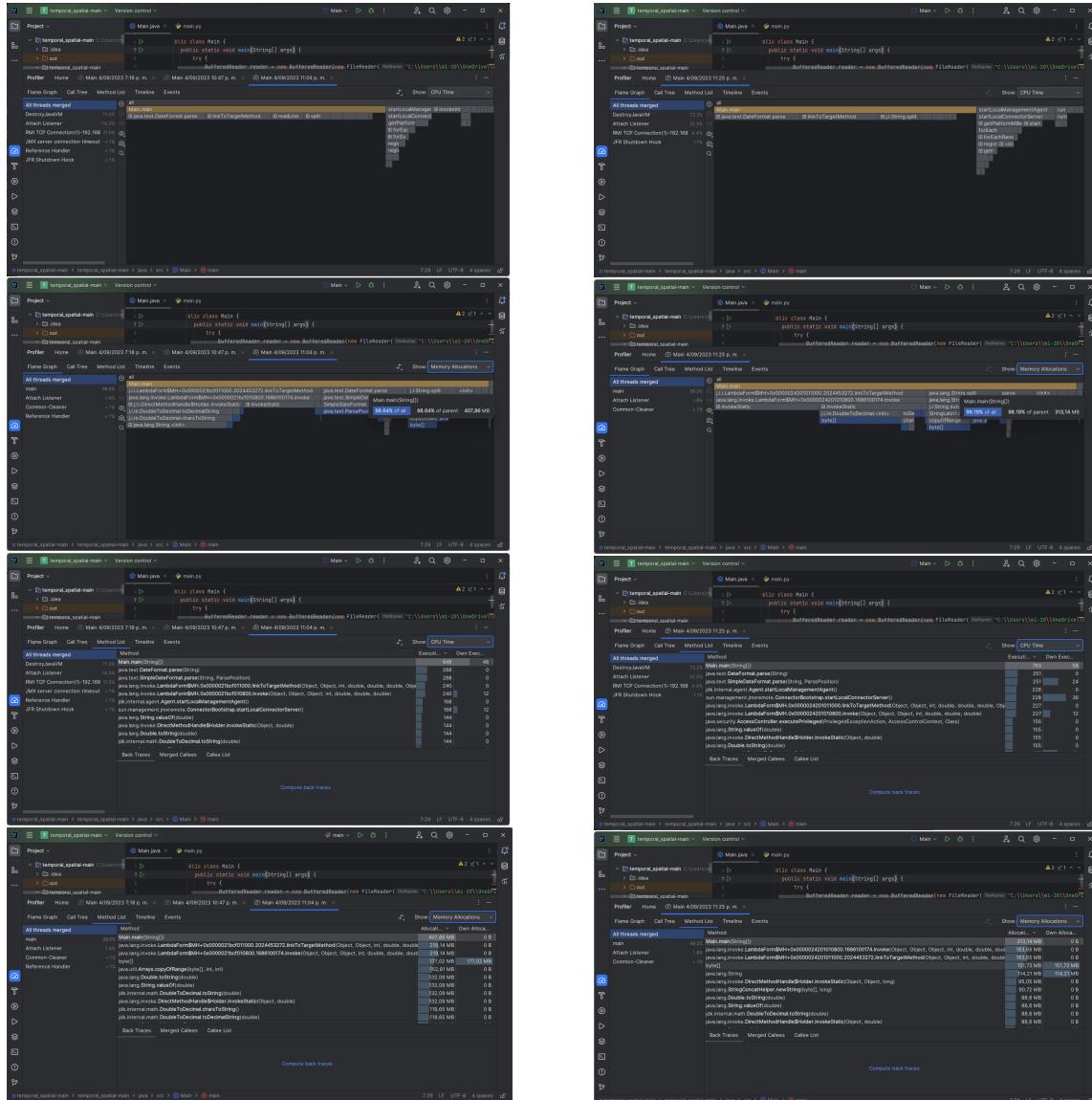
Ejecución 3

```
Starting profile profiler
Data generation and CSV creation complete.
File total de memoria: 395765672 bytes
Snapshot saved to C:/Users/HP/AppData/Local/Temp/Pycharm2023.2/snapshots/temporal_spatial/main/Temps
Process finished with exit code 0
```

JAVA

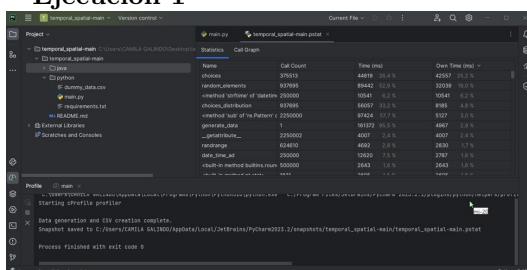
Ejecución 1

Ejecución 2

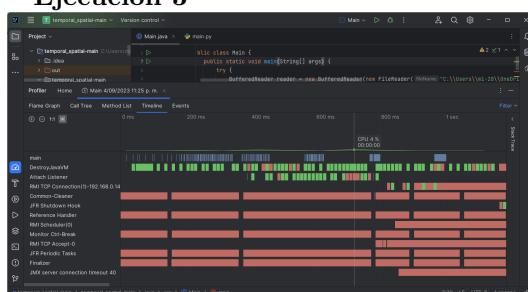


Lenovo AMD 3020e

PYTHON Ejecución 1



Ejecución 3



Ejecución 2

The screenshot shows the Java Profiler interface with the following details:

- Project:** temporal_spatial-main
- Current File:** Main.java
- Statistics:** Call Graph
- Threads:** All threads merged
- Events:** Method
- Timeline:** Events
- Memory Allocations:** Open Allocations
- Code:**

```
try {
    BufferedReader reader = new BufferedReader(new FileReader("filename"));
    BufferedWriter writer = new BufferedWriter(new FileWriter("filename"));
    String line = reader.readLine();
    ...
}
```

Ejecución 3

The screenshot shows the Java Profiler interface with the following details:

- Project:** temporal_spatial-main
- Current File:** Main.java
- Statistics:** Call Graph
- Threads:** All threads merged
- Events:** Method
- Timeline:** Events
- Memory Allocations:** Open Allocations
- Code:**

```
try {
    BufferedReader reader = new BufferedReader(new FileReader("filename"));
    BufferedWriter writer = new BufferedWriter(new FileWriter("filename"));
    String line = reader.readLine();
    ...
}
```

JAVA

Ejecución 1

The screenshot shows the Java Profiler interface with the following details:

- Project:** temporal_spatial-main
- Current File:** Main.java
- Statistics:** Call Graph
- Threads:** All threads merged
- Events:** Method
- Timeline:** Events
- Memory Allocations:** Open Allocations
- Code:**

```
try {
    BufferedReader reader = new BufferedReader(new FileReader("filename"));
    BufferedWriter writer = new BufferedWriter(new FileWriter("filename"));
    String line = reader.readLine();
    ...
}
```

The screenshot shows the Java Profiler interface with the following details:

- Project:** temporal_spatial-main
- Current File:** Main.java
- Statistics:** Call Graph
- Threads:** All threads merged
- Events:** Method
- Timeline:** Events
- Memory Allocations:** Open Allocations
- Code:**

```
try {
    BufferedReader reader = new BufferedReader(new FileReader("filename"));
    BufferedWriter writer = new BufferedWriter(new FileWriter("filename"));
    String line = reader.readLine();
    ...
}
```

The screenshot shows the Java Profiler interface with the following details:

- Project:** temporal_spatial-main
- Current File:** Main.java
- Statistics:** Call Graph
- Threads:** All threads merged
- Events:** Method
- Timeline:** Events
- Memory Allocations:** Open Allocations
- Code:**

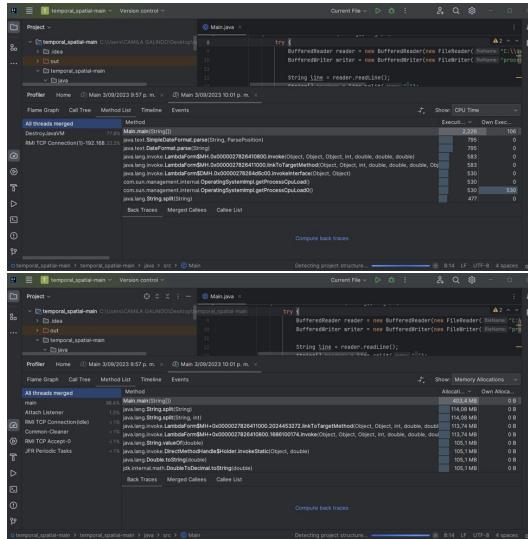
```
try {
    BufferedReader reader = new BufferedReader(new FileReader("filename"));
    BufferedWriter writer = new BufferedWriter(new FileWriter("filename"));
    String line = reader.readLine();
    ...
}
```

Ejecución 2

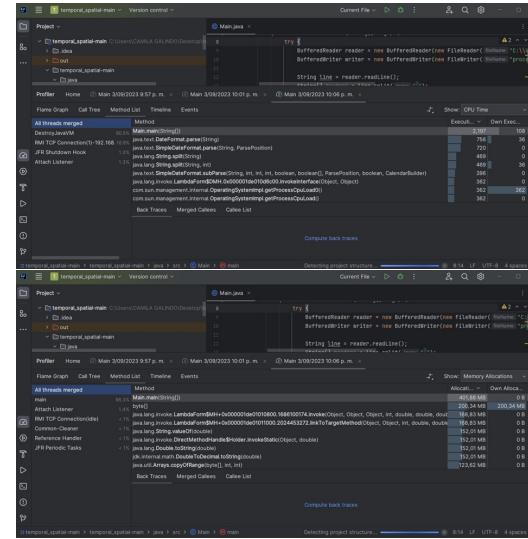
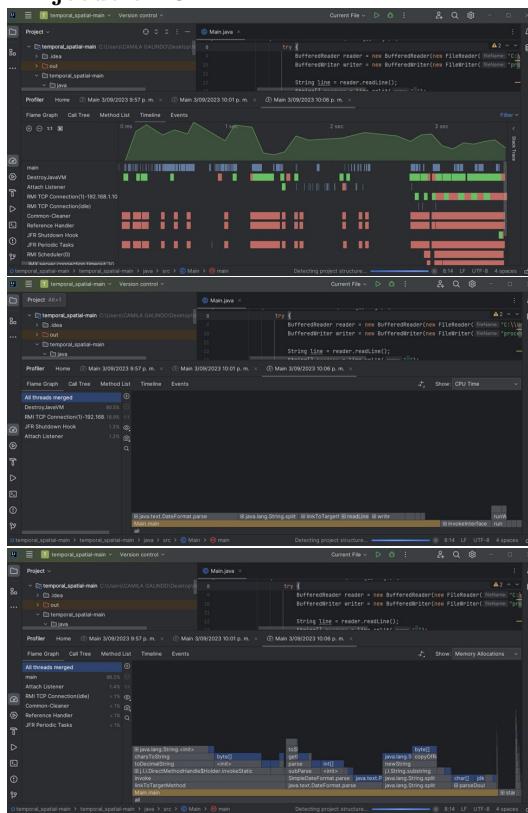
The screenshot shows the Java Profiler interface with the following details:

- Project:** temporal_spatial-main
- Current File:** Main.java
- Statistics:** Call Graph
- Threads:** All threads merged
- Events:** Method
- Timeline:** Events
- Memory Allocations:** Open Allocations
- Code:**

```
try {
    BufferedReader reader = new BufferedReader(new FileReader("filename"));
    BufferedWriter writer = new BufferedWriter(new FileWriter("filename"));
    String line = reader.readLine();
    ...
}
```



Ejecución 3



5. Resumen resultados espacio-temporales

CPU	GPU	Procesador	RAM	Tiempo de Ejecución (ms)	Memos Usados (bytes)
Jose	HP	ASUS TUF Gaming F15 FX506LH	16 GB	16.00 GB (12.8 GB usada)	598.86
Miguel	HP	ASUS ROG Strix G15 G533QR	16 GB	8.00 GB (7.33 GB usada)	530.65
Carola	Lenovo	ASUS ROG Strix G15 G533QR	16 GB	16.00 GB (12.8 GB usada)	417.25
Total				4.00 GB (3.39 GB usada)	1.180.54
<hr/>					
Jose	HP	ASUS TUF Gaming F15 FX506LH	16 GB	16.00 GB (12.8 GB usada)	361.67
Miguel	HP	ASUS ROG Strix G15 G533QR	16 GB	8.00 GB (7.33 GB usada)	307.86
Carola	Lenovo	ASUS ROG Strix G15 G533QR	16 GB	16.00 GB (12.8 GB usada)	240.25
Total				4.00 GB (3.39 GB usada)	910.78
<hr/>					
Jose	HP	ASUS TUF Gaming F15 FX506LH	16 GB	16.00 GB (12.8 GB usada)	421.52
Miguel	HP	ASUS ROG Strix G15 G533QR	16 GB	8.00 GB (7.33 GB usada)	313.14
Carola	Lenovo	ASUS ROG Strix G15 G533QR	16 GB	16.00 GB (12.8 GB usada)	222.00
Total				4.00 GB (3.39 GB usada)	957.66

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. Bibliografía

<https://github.com>