Reporte Profiling

Javier Eduardo Barreto Rojas

I. Introduccion

Mediante el presente documento se muestra un analisis computacional, en donde se analizan distintos tipos de datos en varios lenguajes.

Luego se realiza un analisis mucho mas especifico sobre el tipo de dato array de mutabilidad e inmutabilidad. A continuacion se realizo un analisis de la complejidad computacional de distintos algoritmos, en lenguaje de programacion python.

Ademas, se muestra un analisis profiling y graficacion de estos algoritmos, final y nuevamente se presento un analisis profiling mediante el lenguaje de programacion java, esto junto con su tiempo de ejecucion.

II. Tipos de datos mutables e inmutables

Lenguaje	Mutables	Inmutables
Python	Listas, diccionarios, conjuntos	Tuplas, cadenas de texto, números inmutables (int, float)
Golang	Slices, mapas	Arrays, strings
Java	ArrayList, HashMap	Arrays, Strings, objetos inmutables (como Integer)
C++	Vectores, mapas (std::map o std::unordered_map)	Arrays, strings

Tabla I
Datos mutables e inmutables en varios lenguajes de programación.

III. Arrays: Mutables o inmutables

III-A. Python (Mutable)

En python, los arrays normalmente son usados para crear listas, lo que significa que pueden cambiar su contenido despues de haber sido creadas, esto se debe a que se pueden agregar o eliminar elementos de dichas listas.

```python
# Ejemplo de un array mutable en Python
array_mutable = [1, 2, 3, 4]

# Modificando un elemento del array
array_mutable[0] = 10

# Agregando un nuevo elemento al array
array_mutable.append(5)

# Eliminando un elemento del array
array_mutable.remove(2)

print(array_mutable)  # Output: [10, 3, 4, 5]
```

III-B. Golang (Mutable)

En Golang, los arrays son mutables. Esto significa que el tamaño y los elementos de un array pueden modificarse después de su creación.

```go
package main

import "fmt"

func main() {
    // Ejemplo de un array mutable en Golang
    arrayMutable := [4]int{1, 2, 3, 4}

    // Modificando un elemento del array
    arrayMutable[0] = 10

    // Imprimiendo el array modificado
    fmt.Println(arrayMutable) // Output: [10 2 3 4]
}
```

III-C. Java (Inmutable)

En Java, los arrays son inmutables en el sentido de que su tamaño no puede cambiar después de su creación. Sin embargo, los elementos individuales del array pueden modificarse.

```java
public class Main {
    public static void main(String[] args) {
        // Ejemplo de un array inmutable en Java
        int[] arrayInmutable = {1, 2, 3, 4};

        // Modificando un elemento del array
        arrayInmutable[0] = 10;

        // Imprimiendo el array modificado
        for (int i : arrayInmutable) {
            System.out.print(i + " "); // Output: 10 2 3 4
        }
    }
}
```

*III-C1. **C++(Mutable):*** **En C++**, los arrays son mutables. Puedes modificar tanto el tamaño del array como sus elementos después de la creación.

```cpp
#include <iostream>
using namespace std;

int main() {
    // Ejemplo de un array mutable en C++
    int arrayMutable[4] = {1, 2, 3, 4};

    // Modificando un elemento del array
    arrayMutable[0] = 10;

    // Imprimiendo el array modificado
    for (int i = 0; i < 4; ++i) {
        cout << arrayMutable[i] << " "; // Output: 10 2 3 4
    }
    return 0;
}
```

IV. COMPLEJIDAD COMPUTACIONAL

IV-A. Analisis de Algoritmos:

Algoritmo 1 : Segun el analisis realizado al primer algoritmo, se pudo evidenciar de que la complejidad temporal (Tiempo de ejecucion) de este algoritmo es de tipo constante

Algoritmo 1

```python
from memory_profiler import profile


@profile
def imprimir(lista):
    n = lista
    print(lista)


if "__main__" == __name__:
  lista = 2
  imprimir(lista)
```

Algoritmo 2 : Se pudo evidenciar de que la complejidad temporal (Tiempo de ejecucion) es de tipo cuadratica, esto debido a que mediante una estructura ciclica repetitiva tipo ”for”se recorre una lista de rango ”n”, para numero de filas ïż columnas ”j”, agregados a una variable diccionario tipo pares

Algoritmo 2

```python
from memory_profiler import profile

@profile
def mi_algoritmo(n):
    lista = list(range(n))
    pares = []
    for i in lista:
        for j in lista:
            pares.append((i, j))
    return pares

if __name__ == "__main__":
    mi_algoritmo(1000)
```

Algoritmo 3 : Para este algoritmo se determino que hay una complejidad de tipo logaritrmica, debido a que a travez de una estructura ciclica ”for”se estan realizando varias operaciones en memoria de ordenamiento de datos en un diccionario

Algoritmo 3

```python
import random
import time
from memory_profiler import profile

@profile
def operacion_intensiva_memoria(n):
    """Operación que genera un gran uso de memoria temporalmente."""
    gran_lista = [random.random() for _ in range(n)]
    time.sleep(1)  # Simulamos un procesamiento
    return sum(gran_lista)

@profile
def operacion_intensiva_cpu(n):
    """Operación que consume tiempo de CPU."""
    contador = 0
    for _ in range(n):
        contador += random.random()
        time.sleep(0.01)  # Añade un pequeño retardo para simular procesamiento

def main():
    n = 500000
    for i in range(5):
        print(f"Pico {i+1}: Operación intensiva en memoria")
        operacion_intensiva_memoria(n)
        print(f"Pico {i+1}: Operación intensiva en CPU")
        operacion_intensiva_cpu(100)

if __name__ == "__main__":
    main()
```

Algoritmo 4 : Para este algoritmo se determino una complejidad de tipo constante ya que se realiza una busqueda con algun par de posiciones en el array de ”busqueda”de la variable indice

Algoritmo 4

```
def busqueda(arr, elemento_buscado):
    izquierda, derecha = 0, len(arr) - 1
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        medio_valor = arr[medio]

        if medio_valor == elemento_buscado:
            return medio
        elif elemento_buscado < medio_valor:
            derecha = medio - 1
        else:
            izquierda = medio + 1
    return -1

indice = busqueda([1, 2, 3, 4, 5, 6, 7, 8, 9], 7)
```

Algoritmo 5 : Aqui, en este algoritmo se determino una complejidad de tipo cuadratica, esto porque en la sintaxis del mismo, primero, se realizo una anidacion de dos estructuras ciclicas repetitivas tipo ”for”, luego se inicializa un diccionario vacio, para asi en la segunda estructura, copiar y modificar este diccionario

Algoritmo 5

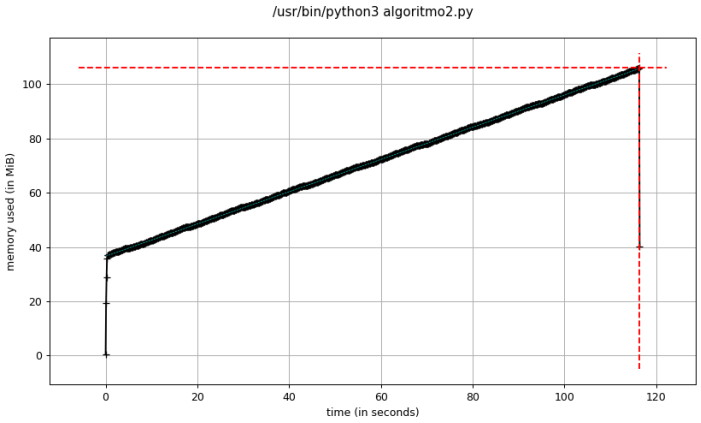
```
def generar_subconjuntos(conjunto):
    subconjuntos = [[]]  # Inicializa con el conjunto vacío
    for elemento in conjunto:
        nuevos_subconjuntos = []
        for subconjunto in subconjuntos:
            nuevo_subconjunto = subconjunto[:]  # Crea una copia del subconjunto actual
            nuevo_subconjunto.append(elemento)  # Agrega el elemento actual al nuevo subconjunto
            nuevos_subconjuntos.append(nuevo_subconjunto)  # Agrega el nuevo subconjunto a la lista de n
        subconjuntos.extend(nuevos_subconjuntos)  # Agrega todos los nuevos subconjuntos a la lista de s
    return subconjuntos

# Ejemplo de uso
conjunto = [1, 2, 3]
subconjuntos = generar_subconjuntos(conjunto)
print("Subconjuntos:", subconjuntos)
```

V. Analisis profiling y graficacion

A continuacion, se presenta mediante un analisis profiling y diversos graficos el tiempo y coste de ejecucion de algunos algoritmos

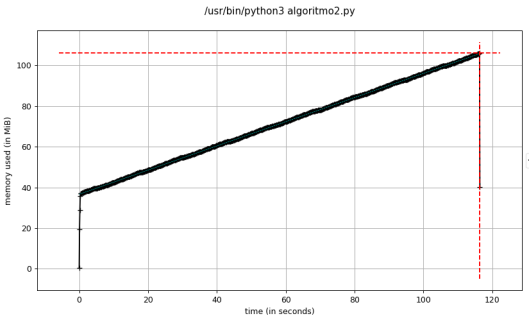
Algoritmo 1 Para este algoritmo se determino una complejidad computacional en tiempo constante, esto debido a que el analisis realizado al grafico y la sintaxis del codigo, nos dice que el costo de memoria con respecto al tiempo en segundos, escala de forma constante dependiendo de la entrada en la lista

```
mprof: Sampling memory every 0.1s
running new process
2
Filename: /content/algoritmo1.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     4     37.0 MiB     37.0 MiB           1   @profile
     5                                         def imprimir(lista):
     6     37.0 MiB      0.0 MiB           1       n = lista
     7     37.0 MiB      0.0 MiB           1       print(lista)
```



Algoritmo 2 Para este algoritmo se determino una complejidad computacional en tiempo cuadratico, esto debido a que alli, en la sintaxis del codigo se demuestra de que la lista .el arrayçrece con respecto a la entrada

```
mprof: Sampling memory every 0.1s
running new process
Filename: /content/algoritmo2.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     3     37.1 MiB     37.1 MiB           1   @profile
     4                                         def mi_algoritmo(n):
     5     37.1 MiB      0.0 MiB           1       lista = list(range(n))
     6     37.1 MiB      0.0 MiB           1       pares = []
     7    106.0 MiB      0.0 MiB        1001       for i in lista:
     8    106.0 MiB     57.4 MiB     1001000           for j in lista:
     9    106.0 MiB     11.5 MiB     1000000               pares.append((i, j))
    10    106.0 MiB      0.0 MiB           1       return pares
```



Algoritmo 3 En este algoritmo se determino una complejidad computacional en tiempo lineal. En primer lugar en la sintaxis el algoritmo se usan tipos de datos mutables, en este caso, la lista, ya que esta es dinamica, luego se tiene en cuenta de que n es igual a 500000 datos, la cual recibe la lista y por este hecho de que la lista va crecer en memoria con respecto a la entrada, nos indica una complejidad lineal, por otro lado, teniendo en cuenta la CPU, aqui a travez de rango se realiza un computo de una variable inmutable, sin embargo, se contrasta de que al final se utiliza un main, en donde, nuevamente mediante una estructura ciclica de tipo ”for”se define una entrada o parametro constante, lo que nos muestra otra complejidad en esta sintaxis.

V-A. Pico 1

```
mprof: Sampling memory every 0.1s
running new process
Pico 1: Operación intensiva en memoria
Filename: /content/algoritmo3.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     5     37.1 MiB     37.1 MiB           1   @profile
     6                                         def operacion_intensiva_memoria(n):
     7                                             """Operación que genera un gran uso de memoria tem
     8     57.0 MiB     19.9 MiB      500003       gran_lista = [random.random() for _ in range(n)]
     9     57.0 MiB      0.0 MiB           1       time.sleep(1)  # Simulamos un procesamiento
    10     57.0 MiB      0.0 MiB           1       return sum(gran_lista)


Pico 1: Operación intensiva en CPU
Filename: /content/algoritmo3.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    12     40.4 MiB     40.4 MiB           1   @profile
    13                                         def operacion_intensiva_cpu(n):
    14                                             """Operación que consume tiempo de CPU."""
    15     40.4 MiB      0.0 MiB           1       contador = 0
    16     40.4 MiB      0.0 MiB         101       for _ in range(n):
    17     40.4 MiB      0.0 MiB         100           contador += random.random()
    18     40.4 MiB      0.0 MiB         100           time.sleep(0.01)  # Añade un pequeño retardo p
```

V-B. Pico 2

```
Pico 2: Operación intensiva en memoria
Filename: /content/algoritmo3.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     5     40.4 MiB     40.4 MiB           1   @profile
     6                                         def operacion_intensiva_memoria(n):
     7                                             """Operación que genera un gran uso de memoria tem
     8     56.1 MiB     15.7 MiB      500003       gran_lista = [random.random() for _ in range(n)]
     9     56.1 MiB      0.0 MiB           1       time.sleep(1)  # Simulamos un procesamiento
    10     56.1 MiB      0.0 MiB           1       return sum(gran_lista)


Pico 2: Operación intensiva en CPU
Filename: /content/algoritmo3.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    12     43.4 MiB     43.4 MiB           1   @profile
    13                                         def operacion_intensiva_cpu(n):
    14                                             """Operación que consume tiempo de CPU."""
    15     43.4 MiB      0.0 MiB           1       contador = 0
    16     43.4 MiB      0.0 MiB         101       for _ in range(n):
    17     43.4 MiB      0.0 MiB         100           contador += random.random()
    18     43.4 MiB      0.0 MiB         100           time.sleep(0.01)  # Añade un pequeño retardo p
```

V-C. Pico 3

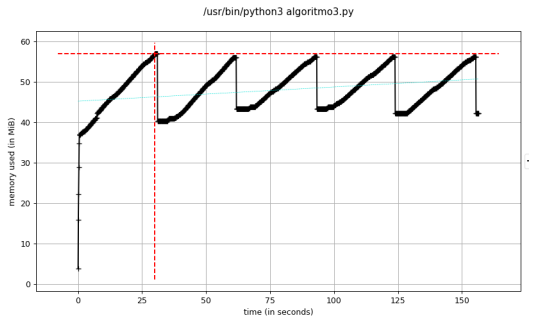
```
Pico 3: Operación intensiva en memoria
Filename: /content/algoritmo3.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     5     43.4 MiB     43.4 MiB           1   @profile
     6                                         def operacion_intensiva_memoria(n):
     7                                             """Operación que genera un gran uso de memoria tem
     8     56.3 MiB     12.9 MiB      500003       gran_lista = [random.random() for _ in range(n)]
     9     56.3 MiB      0.0 MiB           1       time.sleep(1)  # Simulamos un procesamiento
    10     56.3 MiB      0.0 MiB           1       return sum(gran_lista)


Pico 3: Operación intensiva en CPU
Filename: /content/algoritmo3.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    12     43.3 MiB     43.3 MiB           1   @profile
    13                                         def operacion_intensiva_cpu(n):
    14                                             """Operación que consume tiempo de CPU."""
    15     43.3 MiB      0.0 MiB           1       contador = 0
    16     43.3 MiB      0.0 MiB         101       for _ in range(n):
    17     43.3 MiB      0.0 MiB         100           contador += random.random()
    18     43.3 MiB      0.0 MiB         100           time.sleep(0.01)  # Añade un pequeño retardo p
```

V-D. Pico 4

```
Pico 4: Operación intensiva en memoria
Filename: /content/algoritmo3.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     5     43.3 MiB     43.3 MiB           1   @profile
     6                                         def operacion_intensiva_memoria(n):
     7                                             """Operación que genera un gran uso de memoria te
     8     56.2 MiB     12.9 MiB      500003       gran_lista = [random.random() for _ in range(n)]
     9     56.2 MiB      0.0 MiB           1       time.sleep(1)  # Simulamos un procesamiento
    10     56.2 MiB      0.0 MiB           1       return sum(gran_lista)


Pico 4: Operación intensiva en CPU
Filename: /content/algoritmo3.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    12     42.3 MiB     42.3 MiB           1   @profile
    13                                         def operacion_intensiva_cpu(n):
    14                                             """Operación que consume tiempo de CPU."""
    15     42.3 MiB      0.0 MiB           1       contador = 0
    16     42.3 MiB      0.0 MiB         101       for _ in range(n):
    17     42.3 MiB      0.0 MiB         100           contador += random.random()
    18     42.3 MiB      0.0 MiB         100           time.sleep(0.01)  # Añade un pequeño retardo
```

V-E. Pico 5

```
Pico 5: Operación intensiva en memoria
Filename: /content/algoritmo3.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     5     42.3 MiB     42.3 MiB           1   @profile
     6                                         def operacion_intensiva_memoria(n):
     7                                             """Operación que genera un gran uso de memoria tem
     8     56.3 MiB     13.9 MiB      500003       gran_lista = [random.random() for _ in range(n)]
     9     56.3 MiB      0.0 MiB           1       time.sleep(1)  # Simulamos un procesamiento
    10     56.3 MiB      0.0 MiB           1       return sum(gran_lista)


Pico 5: Operación intensiva en CPU
Filename: /content/algoritmo3.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    12     42.3 MiB     42.3 MiB           1   @profile
    13                                         def operacion_intensiva_cpu(n):
    14                                             """Operación que consume tiempo de CPU."""
    15     42.3 MiB      0.0 MiB           1       contador = 0
    16     42.3 MiB      0.0 MiB         101       for _ in range(n):
    17     42.3 MiB      0.0 MiB         100           contador += random.random()
    18     42.3 MiB      0.0 MiB         100           time.sleep(0.01)  # Añade un pequeño retardo p
```




```java
package tallerestructuras;

public class Algoritmo1 {

    public static void imprimir(int n) {
        System.out.println(x: n);
    }

    public static void main(String[] args) {
        int lista = 2;
        imprimir(n: lista);
    }
}
```

Algoritmo 4 A este algoritmo no se le fue posible efectuar una operacion de analisis profiling y graficacion

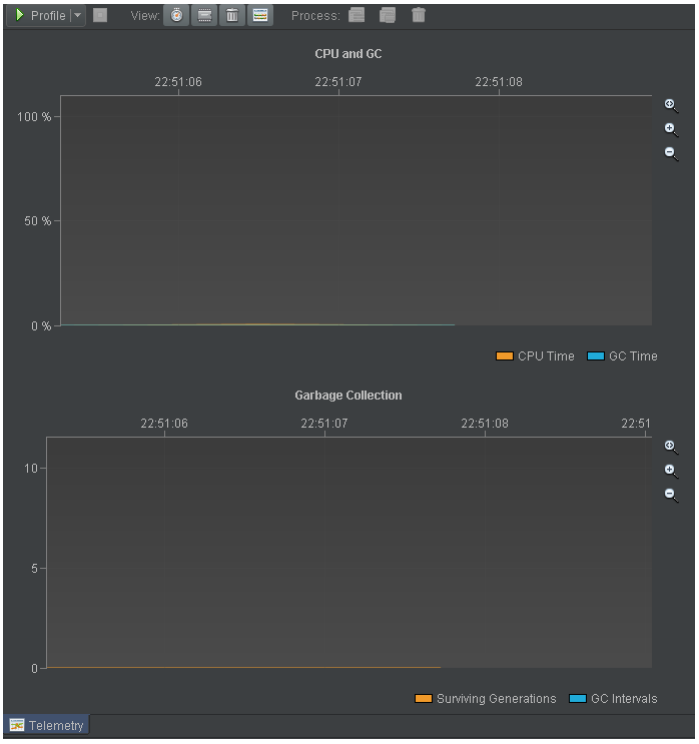
Algoritmo 5 A este algoritmo no se le fue posible efectuar una operacion de analisis profiling y graficacion

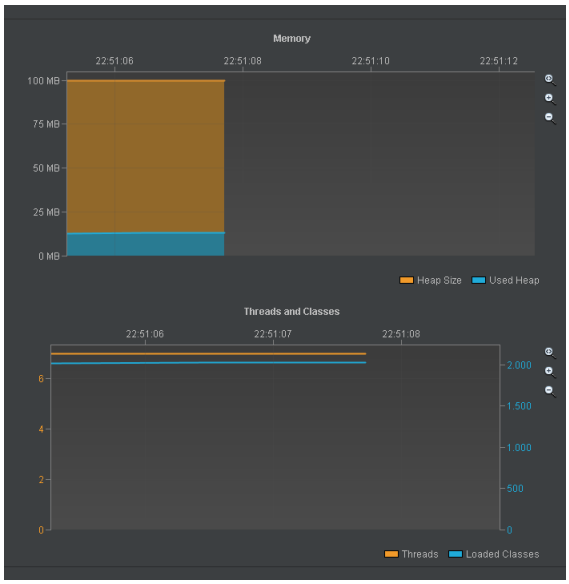
VI. Algoritmos en Java

Ahora, bajo el mismo enfoque que en el punto anterior, se presenta, pero desde la perspectiva del lenguaje de programacion Java

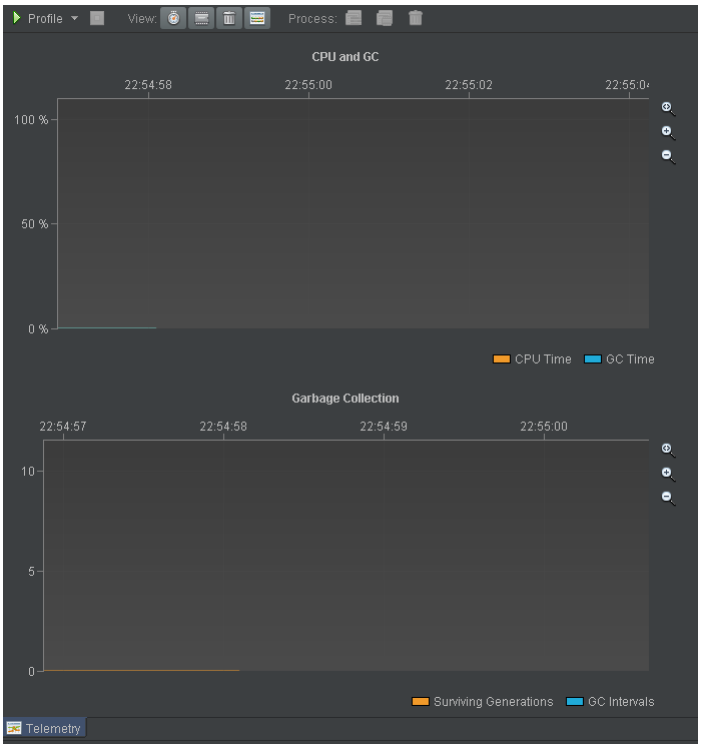
Algoritmo 1

El algoritmo, en tiempo de ejecucion, tiene un coste maximo y constante de 100Mb en memoria durante su tiempo de ejecucion, esto queriendo dar a entender que su coste en memoria no aumentara sin importar su tiempo de ejecucion, ya que su complejidad temporal y espacial es de tipo constante, su coste temporal es constante entre los 189 y 200 (ms)



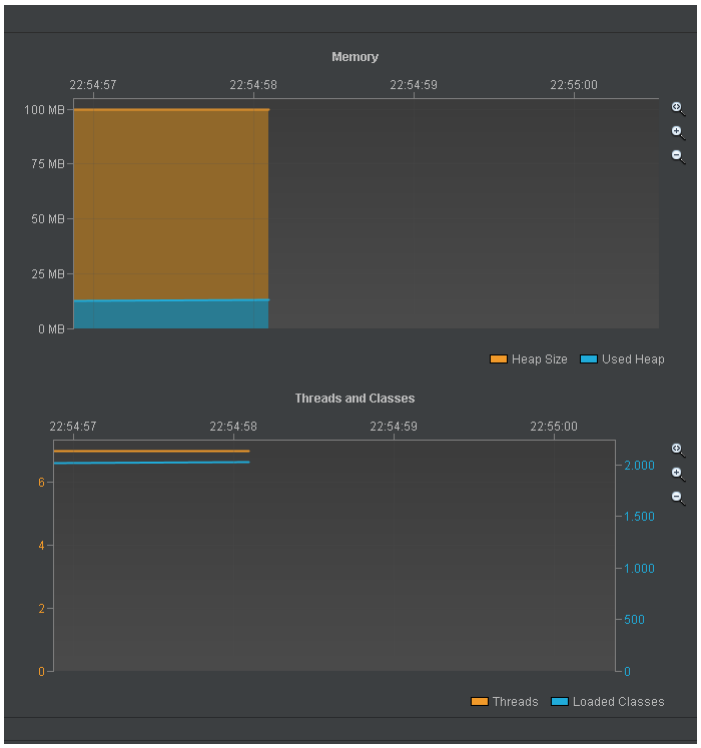



```java
package tallerestructuras;

import java.util.ArrayList;
import java.util.List;


public class Algoritmo2 {

    @SuppressWarnings("unused")
    public static List<Pair> miAlgoritmo(int n) {
        List<Pair> pares = new ArrayList<>();
        List<Integer> lista = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            lista.add(e: i);
        }

        for (int i : lista) {
            for (int j : lista) {
                pares.add(new Pair(first: i, second: j));
            }
        }

        return pares;
    }

    public static void main(String[] args) {
        miAlgoritmo(n: 1000);
    }

    public static class Pair {
        int first;
        int second;

        public Pair(int first, int second) {
            this.first = first;
            this.second = second;
        }
    }
}
```

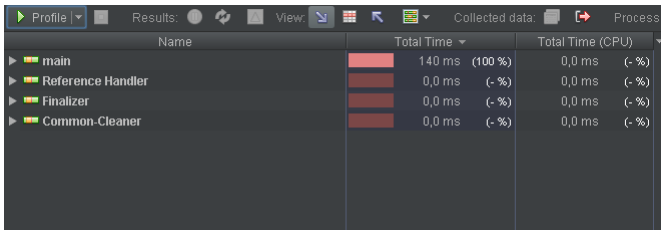


Algoritmo 2

El algoritmo, en tiempo de ejecucion e igualmente que en el anterior algoritmo tiene un coste de 100Mb de memoria, por el contrario al tratarse de una conplejidad temporal cuadratica, esto por la anidacion de dos estructuras ciclicas de tipo ”for”

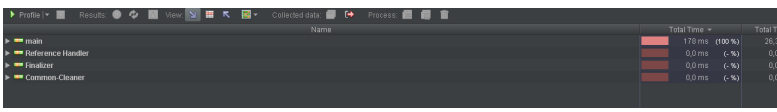



```java
package tallerestructuras;

import java.util.Random;

public class Algoritmo3 {
    public static double operacionIntensivaMemoria(int n) throws InterruptedException {
        /* Operación que genera un gran uso de memoria temporalmente. */
        double[] granArray = new double[n];
        Random random = new Random();
        for (int i = 0; i < n; i++) {
            granArray[i] = random.nextDouble();
        }
        Thread.sleep(1000); // Simulamos un procesamiento
        double sum = 0;
        for (double num : granArray) {
            sum += num;
        }
        return sum;
    }

    public static void operacionIntensivaCPU(int n) throws InterruptedException {
        /* Operación que consume tiempo de CPU. */
        double contador = 0;
        Random random = new Random();
        for (int i = 0; i < n; i++) {
            contador += random.nextDouble();
            Thread.sleep(10); // Añade un pequeño retardo para simular procesamiento
        }
    }

    public static void main(String[] args) throws InterruptedException {
        int n = 500000;
        for (int i = 0; i < 5; i++) {
            System.out.println("Pico " + (i + 1) + ": Operación intensiva en memoria");
            operacionIntensivaMemoria(n);
            System.out.println("Pico " + (i + 1) + ": Operación intensiva en CPU");
            operacionIntensivaCPU(n, 100);
        }
    }
}
```



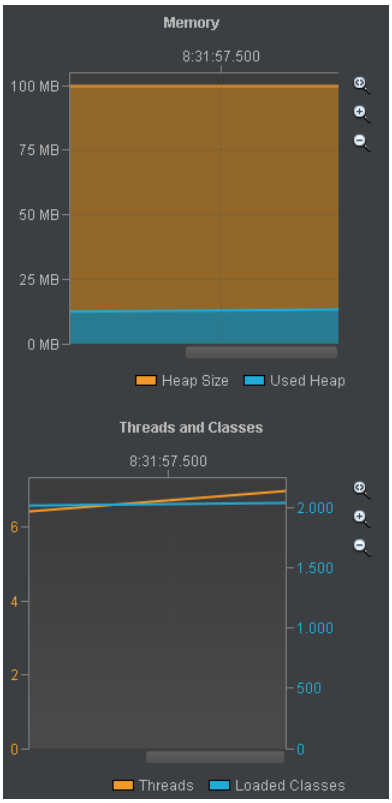
Algoritmo 3

El algoritmo, en tiempo de ejecucion tiene un coste de CPU minimo, sin embargo el tiempo total de ejecucion es en promedio de 178 (ms), en terminos de complejidad computacional, se determino una complejidad temporal en tiempo lineal, hecho que en el grafico en python se reprepp

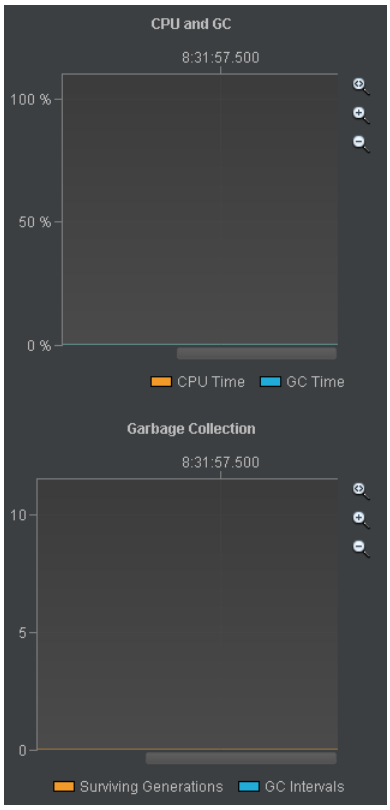


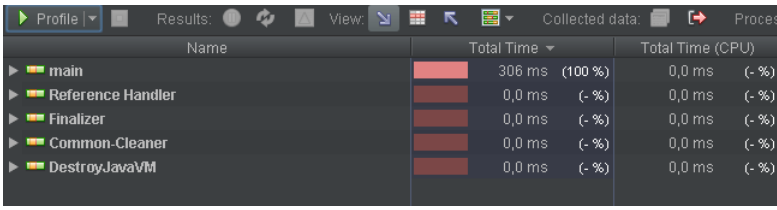
Algoritmo 4

El algoritmo, segun el analisis profiling realizado, nos muestra un tiempo de ejecucion cercano a 306 (ms), lo cual para una complejidad logaritmica como la que tiene este algoritmo es eficiente, sin embargo a travez del grafico retornado por memoria, nos indica que su complejidad espacial es de tiempo constante, esto ya que en medio de la sintaxis del algoritmo se hacen uso de algunas variables locales las cuales ocupan un espacio adicional en memoria




```java
package tallerestructuras;

public class Algoritmo4 {
        public static int busqueda(int[] arr, int elementoBuscado) {
        int izquierda = 0;
        int derecha = arr.length - 1;

        while (izquierda <= derecha) {
            int medio = (izquierda + derecha) / 2;
            int valorMedio = arr[medio];

            if (valorMedio == elementoBuscado) {
                return medio;
            } else if (elementoBuscado < valorMedio) {
                derecha = medio - 1;
            } else {
                izquierda = medio + 1;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        int indice = busqueda(new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9}, elementoBuscado: 7);
        System.out.println("Indice encontrado: " + indice);
    }
}
```

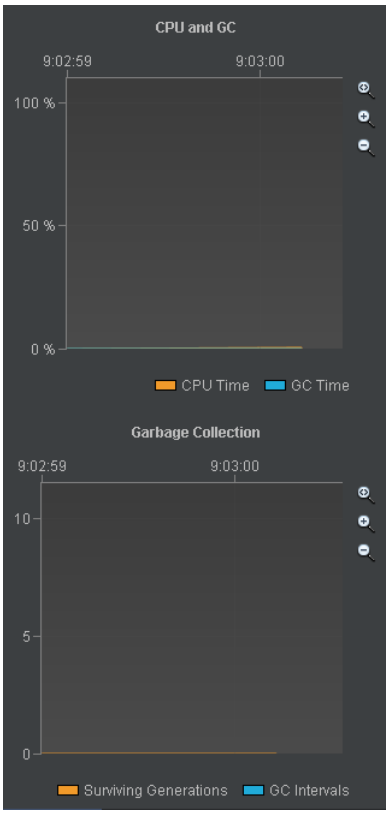


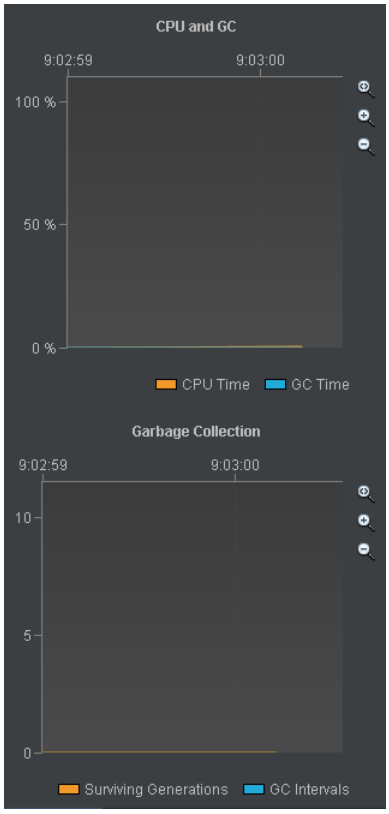


Algoritmo 5

Finalmente, para este algoritmo se pudo evidenciar mediante un analisis profiling de metodos, un tiempo de ejecucion cercano a los 200 (ms), luego teniendo en cuenta la sintaxis del codigo, se pudo determinar una complejidad espacial en tiempo exponencial

```java
package tallerestructuras;

import java.util.ArrayList;
import java.util.List;

public class Algoritmo5 {
    public static List<List<Integer>> generarSubconjuntos(List<Integer> conjunto) {
        List<List<Integer>> subconjuntos = new ArrayList<>();
        // Inicializa con el conjunto vacío
        subconjuntos.add(new ArrayList<>());

        for (int elemento : conjunto) {
            List<List<Integer>> nuevosSubconjuntos = new ArrayList<>();
            for (List<Integer> subconjunto : subconjuntos) {
                // Crea una copia del subconjunto actual
                List<Integer> nuevoSubconjunto = new ArrayList<>(c: subconjunto);
                // Agrega el elemento actual al nuevo subconjunto
                nuevoSubconjunto.add(e: elemento);
                // Agrega el nuevo subconjunto a la lista de nuevos subconjuntos
                nuevosSubconjuntos.add(e: nuevoSubconjunto);
            }
            // Agrega todos los nuevos subconjuntos a la lista de subconjuntos
            subconjuntos.addAll(c: nuevosSubconjuntos);
        }

        return subconjuntos;
    }

    public static void main(String[] args) {
        List<Integer> conjunto = List.of(e1: 1, e2: 2, e3: 3);
        List<List<Integer>> subconjuntos = generarSubconjuntos(conjunto);
        System.out.println("Subconjuntos: " + subconjuntos);
    }
}
```





Name	Total Time		Total Time (CPU)	
main	191 ms	(100 %)	18,9 ms	(100 %)
Reference Handler	0,0 ms	(- %)	0,0 ms	(- %)
Finalizer	0,0 ms	(- %)	0,0 ms	(- %)
Common-Cleaner	0,0 ms	(- %)	0,0 ms	(- %)
DestroyJavaVM	0,0 ms	(- %)	0,0 ms	(- %)

Referencias

Google colaboratory. (s/f). Google.com. Recuperado el 5 de abril de 2024, de https://colab.research.google.com/drive/1GdcDMWsbEPllTEVb93_mXnsPm07eHyu-?usp=sharing

Calvo, J. (s/f). La complejidad de los algoritmos. Europeanvalley.es. Recuperado el 5 de abril de 2024, de https://www.europeanvalley.es/noticias/la-complejidad-de-los-algoritmos/

Análisis de la complejidad de los algoritmos. (s/f). Cs.us.es. Recuperado el 5 de abril de 2024, de https://www.cs.us.es/~jalonso/cursos/i1m-19/temas/tema-28.html

(S/f). Upv.es. Recuperado el 5 de abril de 2024, de https://www.prhlt.upv.es/~evidal/students/prg/tema3/t3prg.pdf