

TALLER I PROFILING

Estructura De Datos.
Gamboa Erika Cod.506221053

El concepto de mutabilidad e inmutabilidad es consistente en todos los lenguajes de programación, aunque la implementación y aplicación de estos conceptos pueden variar según el lenguaje específico.

1. Tabla Tipo Datos Python

| Nombre | Dato | Mutable | Inmutable |
|-----------------------|------------|---------|-----------|
| Enteros | int | | ✓ |
| Flotantes | float | | ✓ |
| Complejos | complex | | ✓ |
| Cadenas | str | | ✓ |
| Booleanos | bool | | ✓ |
| Listas | list | ✓ | |
| Tuplas | tuple | | ✓ |
| Diccionarios | dict | ✓ | |
| Conjuntos | set | ✓ | |
| Frozensets | frozenset | | ✓ |
| Bytes | bytes | | ✓ |
| Arreglos de bytes | bytearray | ✓ | |
| Rangos | range | | ✓ |
| Números decimales | decimal | | ✓ |
| Números fraccionarios | fractions | | ✓ |
| Mapeos de memoria | memoryview | ✓ | |

```

prueba.py X
1 x = 10
2 print("Identidad: " , id(x))
3 print("Tipo: " , type(x))
4 print("Value:" , x)
5 x=20
6 print("Identidad: " , id(x))
7 print("Tipo: " , type(x))
8 print("Value:" , x)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORT
PS C:\Users\ERIKA\Desktop\Estructura> & C:/Users/ERIKA/AppData/Local/Programs/Python/Python38-64/Python.exe -i
Identidad: 140727544007384
Tipo: <class 'int'>
Value: 10
Identidad: 140727544007704
Tipo: <class 'int'>
Value: 20
PS C:\Users\ERIKA\Desktop\Estructura>

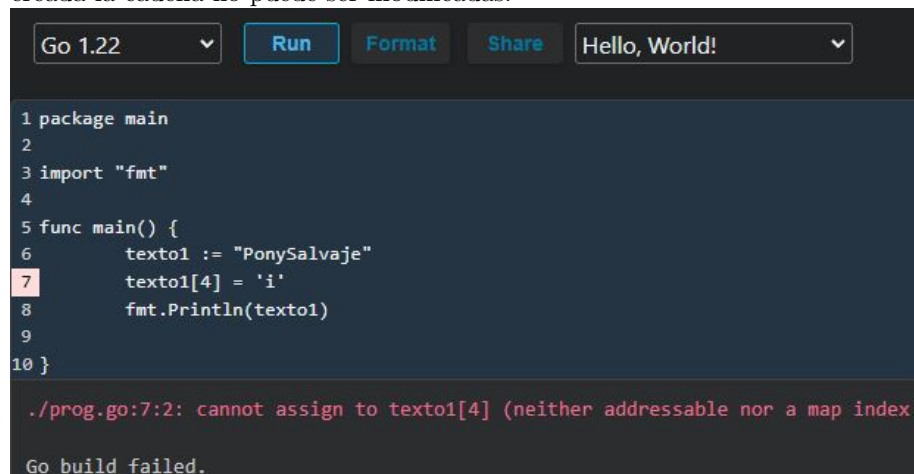
```

Los enteros (**int**) son **inmutables**, lo que significa que no pueden modificarse directamente. Cuando asignamos un nuevo valor a una variable int, Python crea un nuevo objeto en lugar de modificar el existente. Esto se confirma al observar que las identidades (dirección en memoria) de los objetos antes y después de la reasignación son diferentes, indicando que se ha creado un nuevo objeto con el nuevo valor.

2. Tabla Tipo Datos Golang

| Nombre | Dato | Mutable | Inmutable |
|------------------|---------|---------|-----------|
| Enteros | int | | ✓ |
| Flotantes | float64 | | ✓ |
| Booleanos | bool | | ✓ |
| Cadenas de texto | string | | ✓ |
| Arrays | [n]T | ✓ | |
| Slices | []T | ✓ | |
| Mapas | map[K]V | ✓ | |
| Structs | struct | ✓ | |
| Punteros | *T | ✓ | |
| Canal (Channel) | chan T | ✓ | |

Las **cadenas de textos son inmutables** en Go, aquí al intentar modificar el carácter de la cadena, se obtendrá un error de compilación, ya que una vez creada la cadena no puede ser modificadas.



The screenshot shows the Go Playground interface. At the top, there's a version selector set to 'Go 1.22', and buttons for 'Run', 'Format', and 'Share'. A dropdown menu shows 'Hello, World!'. Below this is a code editor with the following Go code:

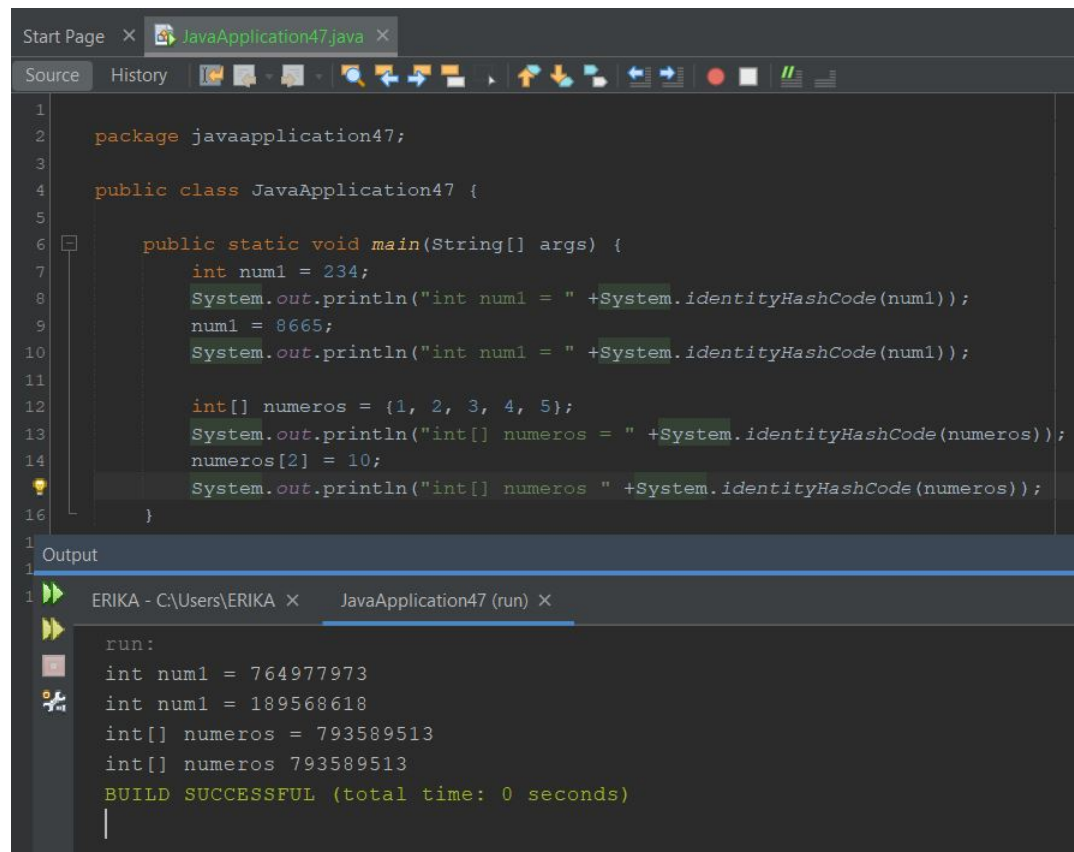
```
1 package main
2
3 import "fmt"
4
5 func main() {
6     texto1 := "PonySalvaje"
7     texto1[4] = 'i'
8     fmt.Println(texto1)
9 }
10 }
```

Line 7 is highlighted with a red background. Below the code editor, a red error message is displayed: `./prog.go:7:2: cannot assign to texto1[4] (neither addressable nor a map index)`. At the bottom, it says 'Go build failed.'

3. Tabla Tipo Datos Java

| Nombre | Dato | Mutable | Immutable |
|---------------------------------|---|---------|-----------|
| Enteros | byte | | ✓ |
| Enteros | short | | ✓ |
| Enteros | int | | ✓ |
| Enteros | long | | ✓ |
| Flotantes | float | | ✓ |
| Flotantes | double | | ✓ |
| Caracteres | char | | ✓ |
| Texto | String | | ✓ |
| Booleanos | boolean | | ✓ |
| Clases | Clases de usuario | ✓ | |
| Interfaces | Interfaces | ✓ | |
| Arreglos | Arrays | ✓ | |
| Enumeraciones | Enums | ✓ | |
| Clases Wrappers | Byte, Short, Integer, Long, Float, Double, Character, Boolean | ✓ | |
| Implementaciones de colecciones | Colecciones (ArrayList, LinkedList, HashSet, etc.) | ✓ | |

Java no permite ver la dirección de memoria de los objetos, es por eso que para este ejemplo utilizamos el método `System.identityHashCode(Object obj)` para obtener la identidad del objeto. Para el dato inmutable `int`, su hash cambia al asignarle un nuevo valor ya que realmente lo que hizo fue crear un nuevo espacio en memoria. Para el tipo de dato `Array`, aún después de la modificación en uno de sus índices, su hash continúa igual ya que este es mutable.



The screenshot shows an IDE window with a tab titled "JavaApplication47.java". The editor displays the following Java code:

```
1 package javaapplication47;
2
3
4 public class JavaApplication47 {
5
6     public static void main(String[] args) {
7         int num1 = 234;
8         System.out.println("int num1 = " + System.identityHashCode(num1));
9         num1 = 8665;
10        System.out.println("int num1 = " + System.identityHashCode(num1));
11
12        int[] numeros = {1, 2, 3, 4, 5};
13        System.out.println("int[] numeros = " + System.identityHashCode(numeros));
14        numeros[2] = 10;
15        System.out.println("int[] numeros " + System.identityHashCode(numeros));
16    }
```

Below the code editor is the "Output" window, which shows the execution results:

```
1 run:
2 int num1 = 764977973
3 int num1 = 189568618
4 int[] numeros = 793589513
5 int[] numeros 793589513
6 BUILD SUCCESSFUL (total time: 0 seconds)
```

4. Tabla Tipo Datos C++

En C++, no hay tipos de datos inmutables proporcionados por el lenguaje de forma predeterminada. Sin embargo, se pueden crear tipos de datos inmutables en C++ utilizando el modificador `const`

| Nombre | Dato | Mutable |
|-------------|-----------------|---------|
| Enteros | int | ✓ |
| Enteros | long | ✓ |
| Flotantes | float | ✓ |
| Flotantes | double | ✓ |
| Caracteres | char | ✓ |
| Booleanos | boolean | ✓ |
| Arreglos | int[], double[] | ✓ |
| Punteros | int*, double* | ✓ |
| Estructuras | struct | ✓ |
| Uniones | union | ✓ |
| Clases | class | ✓ |

```
#include <iostream>

void numero() {
    int num = 10;
    std::cout << "Dirección Memoria Antes: " << &num << std::endl << num ;
    std::cout.write(reinterpret_cast<const char*>(&num), sizeof(num));
    num = 90;
    std::cout << "Dirección Memoria Despues: " << &num << std::endl << num ;
    std::cout.write(reinterpret_cast<const char*>(&num), sizeof(num));
}

int main() {
    numero();
    return 0;
}
```

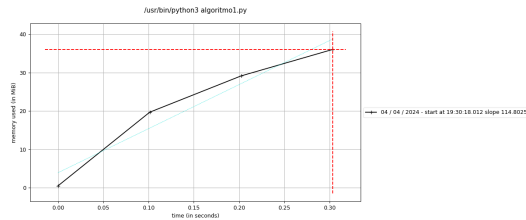
Run ☆ Share ⌵ Fullscreen ✓ Done

```
Dirección Memoria Antes: 0x7ffcb62a0ea4
10
Dirección Memoria Despues: 0x7ffcb62a0ea4
90Z
```

En este ejemplo, se demuestra que para C++, el tipo de dato `int` es mutable. Esto se ve al asignar un nuevo valor a la variable `num` dentro de la función `numero()`. A pesar de que el valor de `num` cambia de 10 a 90, la variable sigue siendo el mismo objeto de memoria.

5. Profiler Python

Algoritmo 1



Filename: algoritmo1.py

| Line # | Mem usage | Increment | Occurrences | Line Contents |
|--------|-----------|-----------|-------------|----------------------|
| 4 | 36.9 MiB | 36.9 MiB | 1 | @profile |
| 5 | | | | def imprimir(lista): |
| 6 | 36.9 MiB | 0.0 MiB | 1 | n = lista |
| 7 | 36.9 MiB | 0.0 MiB | 1 | print(lista) |

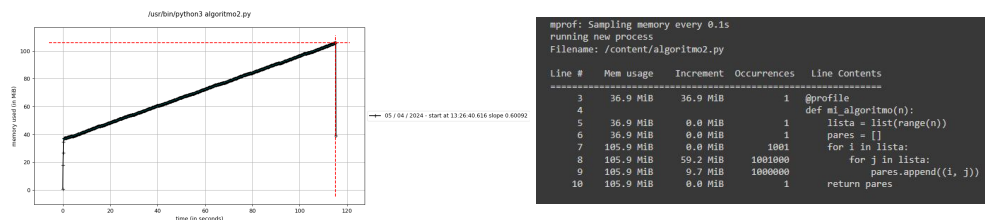
El algoritmo cuenta con un BigO $O(1)$ contante el cual se puede ver reflejado en su Mem Usage donde la cantidad de memoria empleado es constante.

```
from memory_profiler import profile

@profile
def imprimir(lista): # O(1)
    n = lista # O(1)
    print(lista) # O(1)

if "__main__" == __name__: # O(1)
    lista = 2 # O(1)
    imprimir(lista) # O(1)
```

Algoritmo 2



El algoritmo cuenta con un BigO $O(n^2)$ el cual se puede ver en la anidación de los dos ciclos for, también se puede ver este crecimiento en Mem Usage donde la cantidad de memoria empleada es lineal.

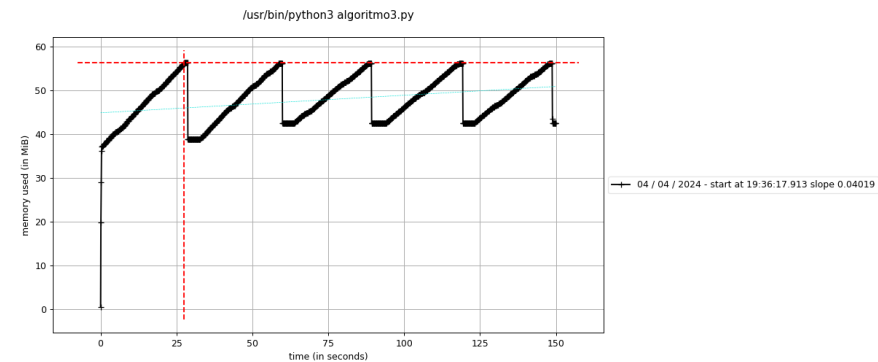
```

from memory_profiler import profile

@profile
def mi_algoritmo(n): # O(1)
    lista = list(range(n)) # o(1)
    pares = [] # O(1)
    for i in lista:
        for j in lista: # O(n^2)
            pares.append((i, j))
    return pares

if __name__ == "__main__":
    mi_algoritmo(1000) # O(1)
  
```


Algoritmo 3



Pico1

```
Pico 1: Operación intensiva en memoria
Filename: algoritmo3.py

Line #   Mem usage   Increment   Occurrences   Line Contents
=====
5        36.8 MiB     36.8 MiB      1      @profile
6
7
8        56.8 MiB     20.0 MiB    500003      def operacion_intensiva_memoria(n):
9        56.8 MiB      0.0 MiB      1          """Operación que genera un gran uso de memoria temporalmente."""
10       56.8 MiB      0.0 MiB      1          gran_lista = [random.random() for _ in range(n)]
11                                     time.sleep(1) # Simulamos un procesamiento
12                                     return sum(gran_lista)

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

Line #   Mem usage   Increment   Occurrences   Line Contents
=====
12       39.4 MiB     39.4 MiB      1      @profile
13
14
15       39.4 MiB      0.0 MiB      1      def operacion_intensiva_cpu(n):
16       39.4 MiB      0.0 MiB    101          """Operación que consume tiempo de CPU."""
17       39.4 MiB      0.0 MiB    100          contador = 0
18       39.4 MiB      0.0 MiB    100          for _ in range(n):
19                                     contador += random.random()
20                                     time.sleep(0.01) # Añade un pequeño retardo para simular procesamiento
```

Pico2

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

Line #   Mem usage   Increment   Occurrences   Line Contents
=====
5        39.4 MiB     39.4 MiB      1      @profile
6
7
8        56.1 MiB     16.8 MiB    500003      def operacion_intensiva_memoria(n):
9        56.1 MiB      0.0 MiB      1          """Operación que genera un gran uso de memoria temporalmente."""
10       56.1 MiB      0.0 MiB      1          gran_lista = [random.random() for _ in range(n)]
11                                     time.sleep(1) # Simulamos un procesamiento
12                                     return sum(gran_lista)

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

Line #   Mem usage   Increment   Occurrences   Line Contents
=====
12       42.4 MiB     42.4 MiB      1      @profile
13
14
15       42.4 MiB      0.0 MiB      1      def operacion_intensiva_cpu(n):
16       42.4 MiB      0.0 MiB    101          """Operación que consume tiempo de CPU."""
17       42.4 MiB      0.0 MiB    100          contador = 0
18       42.4 MiB      0.0 MiB    100          for _ in range(n):
19                                     contador += random.random()
20                                     time.sleep(0.01) # Añade un pequeño retardo para simular procesamiento
```

Pico3

| Pico 3: Operación intensiva en memoria Filename: algoritmo3.py | | | | |
|---|-----------|-----------|-------------|--|
| Line # | Mem usage | Increment | Occurrences | Line Contents |
| ===== | | | | |
| 5 | 42.4 MiB | 42.4 MiB | 1 | @profile |
| 6 | | | | def operacion_intensiva_memoria(n): |
| 7 | | | | """Operación que genera un gran uso de memoria temporalmente.""" |
| 8 | 56.1 MiB | 13.7 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 3: Operación intensiva en CPU Filename: algoritmo3.py | | | | |
| Line # | Mem usage | Increment | Occurrences | Line Contents |
| ===== | | | | |
| 12 | 42.4 MiB | 42.4 MiB | 1 | @profile |
| 13 | | | | def operacion_intensiva_cpu(n): |
| 14 | | | | """Operación que consume tiempo de CPU.""" |
| 15 | 42.4 MiB | 0.0 MiB | 1 | contador = 0 |
| 16 | 42.4 MiB | 0.0 MiB | 101 | for _ in range(n): |
| 17 | 42.4 MiB | 0.0 MiB | 100 | contador += random.random() |
| 18 | 42.4 MiB | 0.0 MiB | 100 | time.sleep(0.01) # Añade un pequeño retardo para simular procesamiento |

Pico4

| Pico 4: Operación intensiva en memoria Filename: algoritmo3.py | | | | |
|---|-----------|-----------|-------------|--|
| Line # | Mem usage | Increment | Occurrences | Line Contents |
| ===== | | | | |
| 5 | 42.4 MiB | 42.4 MiB | 1 | @profile |
| 6 | | | | def operacion_intensiva_memoria(n): |
| 7 | | | | """Operación que genera un gran uso de memoria temporalmente.""" |
| 8 | 56.1 MiB | 13.7 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 4: Operación intensiva en CPU Filename: algoritmo3.py | | | | |
| Line # | Mem usage | Increment | Occurrences | Line Contents |
| ===== | | | | |
| 12 | 42.4 MiB | 42.4 MiB | 1 | @profile |
| 13 | | | | def operacion_intensiva_cpu(n): |
| 14 | | | | """Operación que consume tiempo de CPU.""" |
| 15 | 42.4 MiB | 0.0 MiB | 1 | contador = 0 |
| 16 | 42.4 MiB | 0.0 MiB | 101 | for _ in range(n): |
| 17 | 42.4 MiB | 0.0 MiB | 100 | contador += random.random() |
| 18 | 42.4 MiB | 0.0 MiB | 100 | time.sleep(0.01) # Añade un pequeño retardo para simular procesamiento |

Pico5

| Pico 5: Operación intensiva en memoria Filename: algoritmo3.py | | | | |
|---|-----------|-----------|-------------|--|
| Line # | Mem usage | Increment | Occurrences | Line Contents |
| ===== | | | | |
| 5 | 42.4 MiB | 42.4 MiB | 1 | @profile |
| 6 | | | | def operacion_intensiva_memoria(n): |
| 7 | | | | """Operación que genera un gran uso de memoria temporalmente.""" |
| 8 | 56.1 MiB | 13.7 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 5: Operación intensiva en CPU Filename: algoritmo3.py | | | | |
| Line # | Mem usage | Increment | Occurrences | Line Contents |
| ===== | | | | |
| 12 | 42.4 MiB | 42.4 MiB | 1 | @profile |
| 13 | | | | def operacion_intensiva_cpu(n): |
| 14 | | | | """Operación que consume tiempo de CPU.""" |
| 15 | 42.4 MiB | 0.0 MiB | 1 | contador = 0 |
| 16 | 42.4 MiB | 0.0 MiB | 101 | for _ in range(n): |
| 17 | 42.4 MiB | 0.0 MiB | 100 | contador += random.random() |
| 18 | 42.4 MiB | 0.0 MiB | 100 | time.sleep(0.01) # Añade un pequeño retardo para simular procesamiento |

El algoritmo cuenta con un BigO $O(n^3)$, el cual se puede ver en el `main` del algoritmo. Los cinco picos en la grafica son causados por el `"for i in range(5)"`. Cada pico en la grafica es el conjunto `OperacionIntensivaMemoria` y `OperacionIntensivaCpu`.

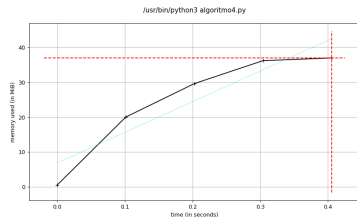
```
@profile
def operacion_intensiva_memoria(n): # O(1)
    """Operación que genera un gran uso de memoria temporalmente."""
    gran_lista = [random.random() for _ in range(n)] # O(n)
    time.sleep(1) # Simulamos un procesamiento
    return sum(gran_lista) # O(1)

@profile
def operacion_intensiva_cpu(n): # O(1)
    """Operación que consume tiempo de CPU."""
    contador = 0
    for _ in range(n): # O(n)
        contador += random.random()
        time.sleep(0.01) # Añade un pequeño retardo para simular procesam

def main():
    n = 500000
    for i in range(5): # O(n^3)
        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



Filename: algoritmo4.py

| Line # | Mem usage | Increment | Occurrences | Line Contents |
|--------|-----------|-----------|-------------|--------------------------------------|
| 5 | 36.9 MiB | 36.9 MiB | 1 | @profile |
| 6 | | | | def busqueda(arr, elemento_buscado): |
| 7 | 36.9 MiB | 0.0 MiB | 1 | izquierda, derecha = 0, len(arr) - 1 |
| 8 | 36.9 MiB | 0.0 MiB | 2 | while izquierda <= derecha: |
| 9 | 36.9 MiB | 0.0 MiB | 2 | medio = (izquierda + derecha) // 2 |
| 10 | 36.9 MiB | 0.0 MiB | 2 | medio_valor = arr[medio] |
| 11 | | | | |
| 12 | 36.9 MiB | 0.0 MiB | 2 | if medio_valor == elemento_buscado: |
| 13 | 36.9 MiB | 0.0 MiB | 1 | return medio |
| 14 | 36.9 MiB | 0.0 MiB | 1 | elif elemento_buscado < medio_valor: |
| 15 | | | | derecha = medio - 1 |
| 16 | | | | else: |
| 17 | 36.9 MiB | 0.0 MiB | 1 | izquierda = medio + 1 |
| 18 | | | | return -1 |

El algoritmo cuenta con un BigO $O(\log n)$, el cual se puede ver en la búsqueda binaria realizada. en Mem usage se muestra como el uso de memoria es contante.

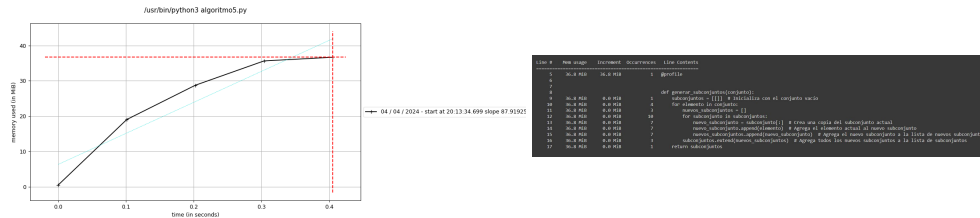
```
import random
import time
from memory_profiler import profile

@profile
def busqueda(arr, elemento_buscado):
    izquierda, derecha = 0, len(arr) - 1 #O(1)
    while izquierda <= derecha: #O(n)
        medio = (izquierda + derecha) // 2 #O(log n)
        medio_valor = arr[medio]

        if medio_valor == elemento_buscado: #O(1)
            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)#O(1)
```

Algoritmo 5



El algoritmo cuenta con una complejidad espacial de $O(2^n)$, y una complejidad de tiempo $O(1)$. En Mem usage se muestra como el uso de memoria es constante.

```

import random
import time
from memory_profiler import profile

def generar_subconjuntos(conjunto):
    subconjuntos = [[]] # Inicializa con el conjunto vacío
    for elemento in conjunto:
        nuevos_subconjuntos = []
        for subconjunto in subconjuntos: # O(2^n)
            nuevo_subconjunto = subconjunto[:] # Crea una copia
            nuevo_subconjunto.append(elemento) # Agrega el elemento
            nuevos_subconjuntos.append(nuevo_subconjunto)
        subconjuntos.extend(nuevos_subconjuntos) # Agrega los nuevos subconjuntos
    return subconjuntos

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

6. Tabla de comparación de algoritmos

Java fue mas eficiente en los tiempos de ejecución pero Python tendió a usar menos memoria.

| Algoritmo | JAVA | | PYTHON | |
|-------------|-----------|-----------------------------|--------------------------|-----------|
| | Tiempo | Memoria | Tiempo | Memoria |
| Algoritmo 1 | 143ms | Max used Heap: 9.9001.208 | 0.0625009536743164 seg | 36.9 MiB |
| Algoritmo 1 | 178,434ms | Max used Heap: 122.171.656B | 180.32537317276 seg | 105.9 MiB |
| Algoritmo 1 | 13.549ms | Max used Heap: 30.872B | 181.442373752594 seg | 56.1 MiB |
| Algoritmo 1 | 127ms | Max used Heap: 9.901B | 0.046872615814208984 seg | 36.9 MiB |
| Algoritmo 1 | 115ms | Max used Heap: 9.901B | 0.07812333106994629 seg | 36.8 MiB |