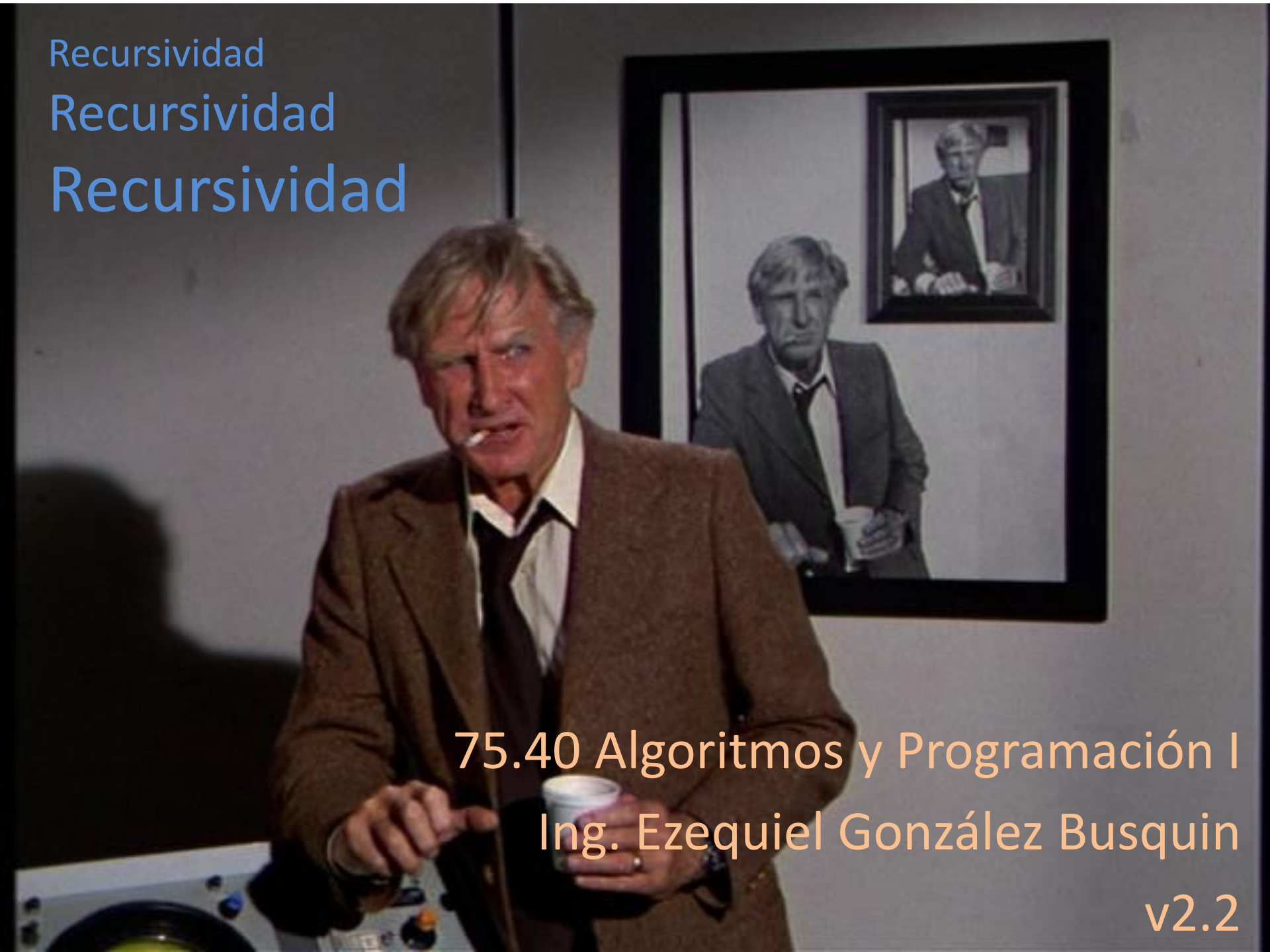
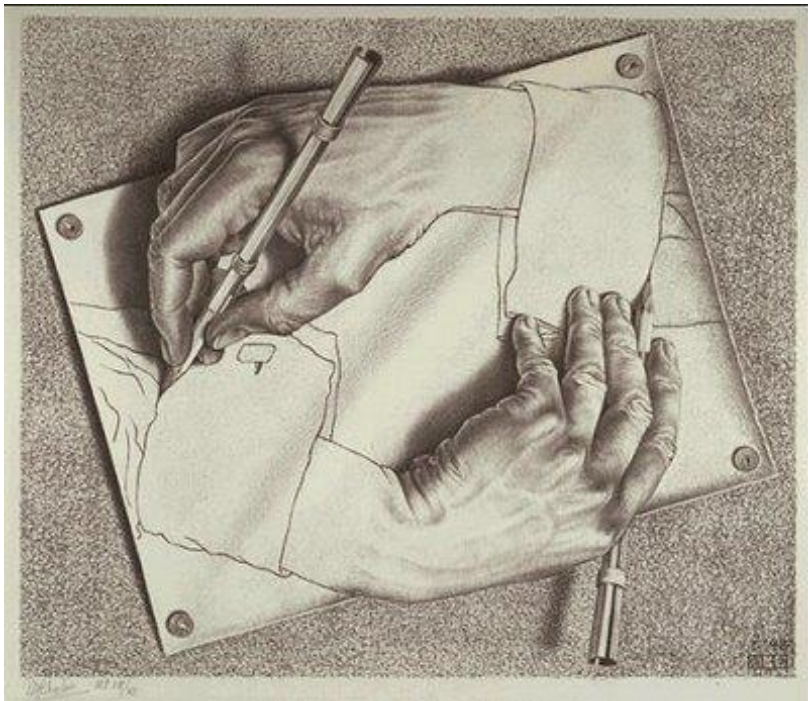


Recursividad
Recursividad
Recursividad

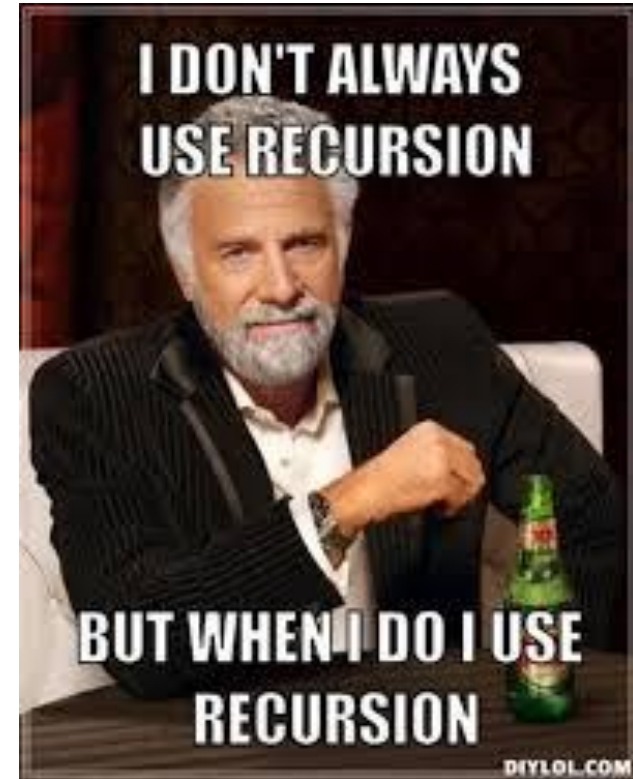
A man with light brown hair, wearing a brown jacket over a white shirt and dark tie, is shown from the chest up. He is holding a cigarette in his mouth and a white cup in his left hand. He is pointing with his right hand towards the camera. In the background, there is a large framed picture of the same man, also holding a cup, which is itself framed within a larger frame.

75.40 Algoritmos y Programación I
Ing. Ezequiel González Busquin
v2.2



Recursividad

- Para entender la recursividad, primero hay que entender la recursividad.



Recursividad

- Es cuando una funcionalidad o una estructura de datos hace **referencia a sí misma** para **definirse**.
- Recursividad = Recursión = Recurrencia
- En inglés: *Recursion*



recursion

Search

About 1,570,000 results (0.20 seconds)

Everything

Images

Maps

Videos

News

More

Pune, Maharashtra
Change location

Did you mean: [recursion](#)

[Recursion](#) - Wikipedia, the free encyclopedia

en.wikipedia.org/wiki/Recursion

Recursion is the process of repeating items in a self-similar way
the surfaces of two mirrors are exactly parallel with each other th

[Formal definitions of recursion](#) - [Recursion in language](#) - [Recursio](#)

[Recursion \(computer science\)](#) - Wikipedia, the free e

[en.wikipedia.org/wiki/Recursion_\(computer_science\)](http://en.wikipedia.org/wiki/Recursion_(computer_science))

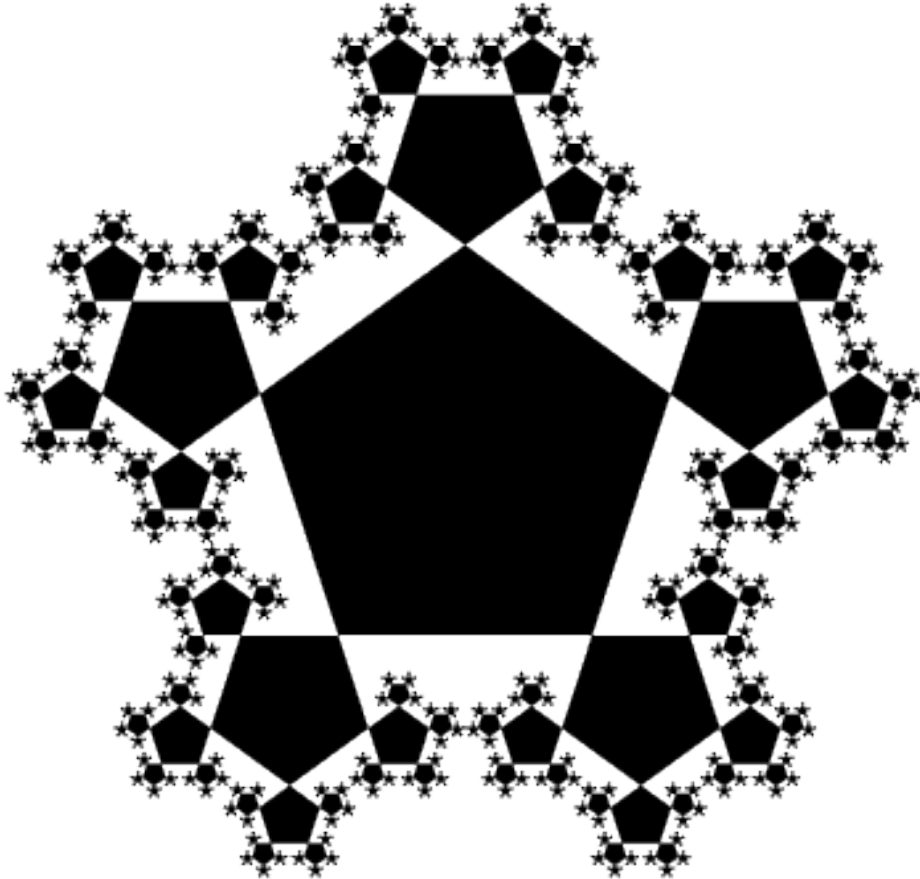
Recursion in computer science is a method where the solution t

[Recursive data types](#) - [Recursive algorithms](#) - [Structural versus g](#)

Recursividad en la naturaleza



Recursividad en la matemática



$$0 \in \mathbb{N}$$

$$\forall n: n \in \mathbb{N} \Rightarrow n + 1 \in \mathbb{N}$$

Recursividad

- Tener en cuenta que para hacer uso de la recursividad se requiere
 - Uno o más **casos base**
 - Un **juego de reglas** que **reduce** todos los demás casos hasta llegar a los **casos base**



Factorial

- Función **factorial** de un natural positivo se define como:

$$n = 0 \Rightarrow n! = 1$$

$$\forall n: n > 0 \Rightarrow n! = n \cdot (n - 1)!$$

Diagram illustrating the calculation of factorials:

- $0! = 1$
- $1! = 1$
- $2! = 2 \cdot 1 = 2$
- $3! = 3 \cdot 2 \cdot 1 = 6$
- $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$
- $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$

Factorial

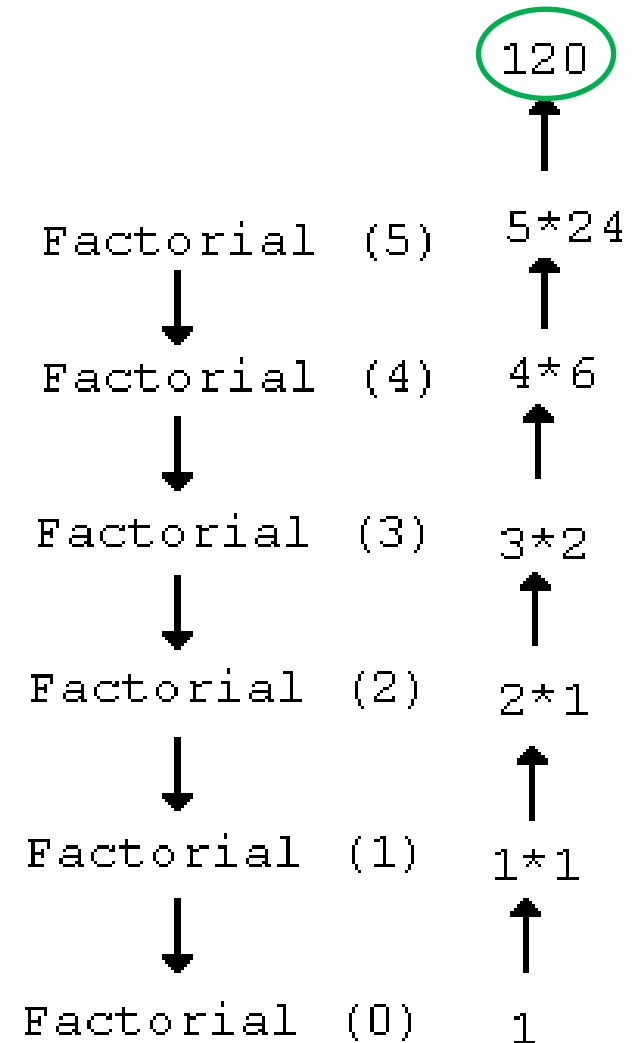
- Una implementación en Python del algoritmo puede ser:

```
def factorial(n):  
    if n == 0: return 1  
    return n * factorial(n-1)
```

Factorial

```
def factorial(n):  
    if n == 0: return 1  
    return n * factorial(n-1)
```

```
print factorial(5)
```

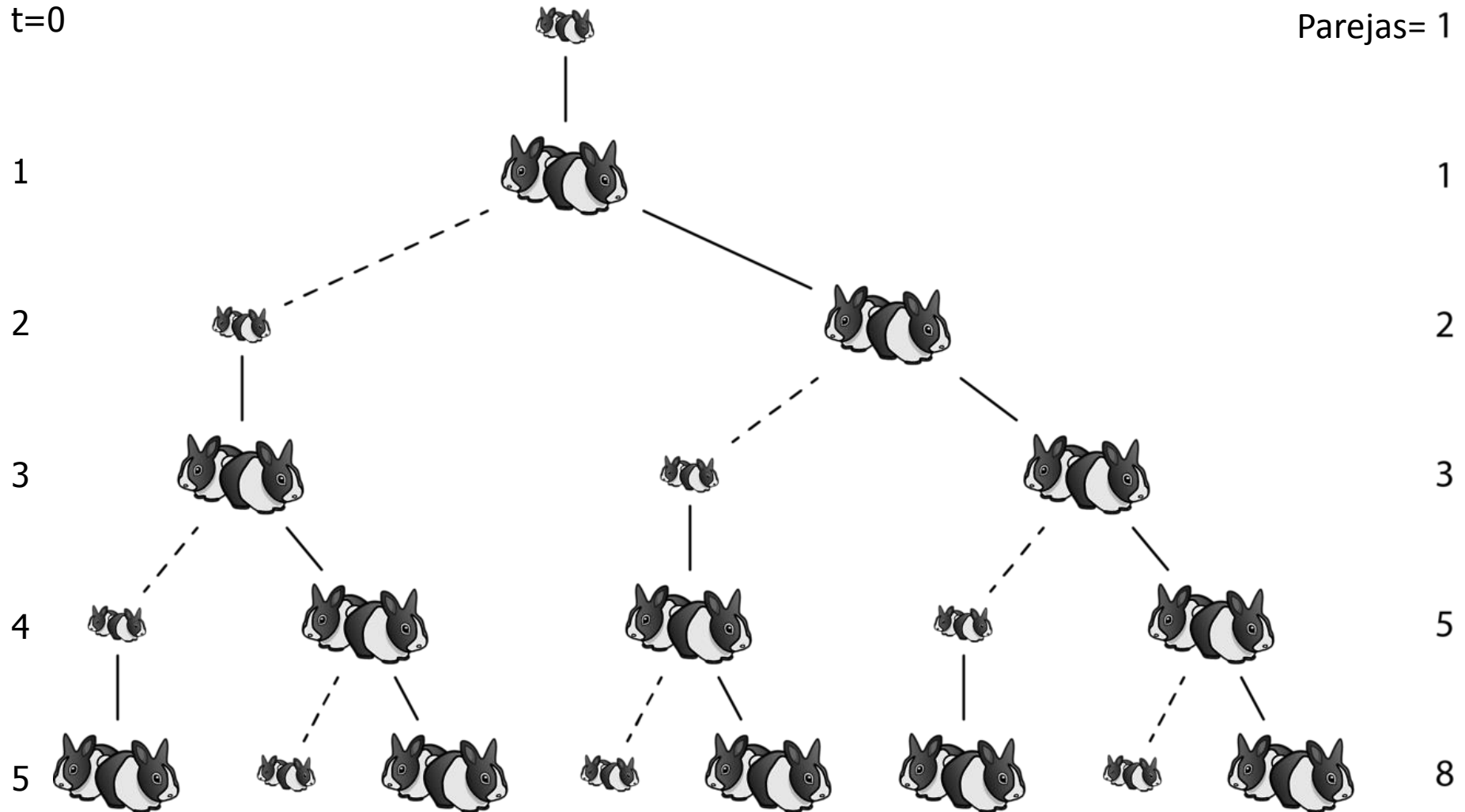




Números de Fibonacci

t=0

Parejas= 1





Números de Fibonacci

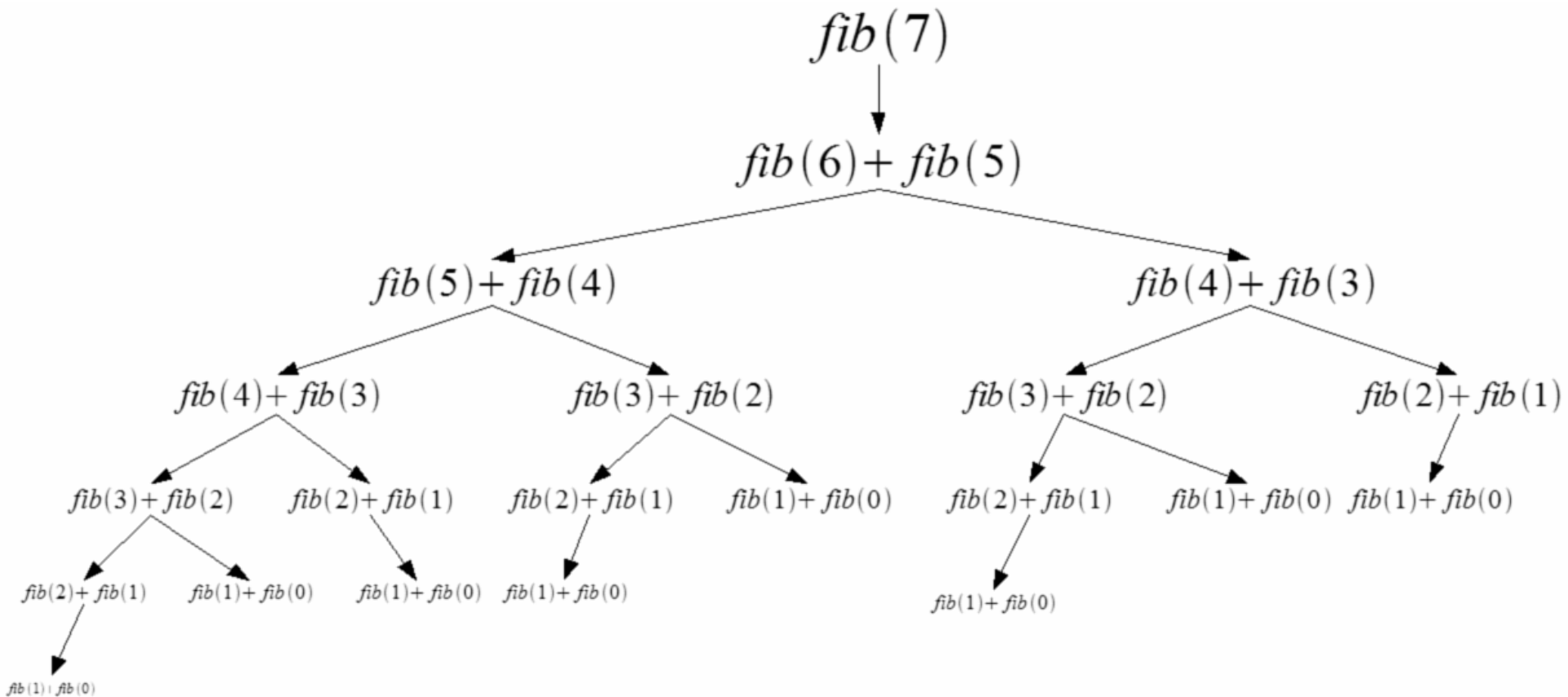
$$Fib(0) = 1$$

$$Fib(1) = 1$$

$$\forall n: n > 1 \Rightarrow Fib(n) = Fib(n - 1) + Fib(n - 2)$$



```
print fib(7)
```

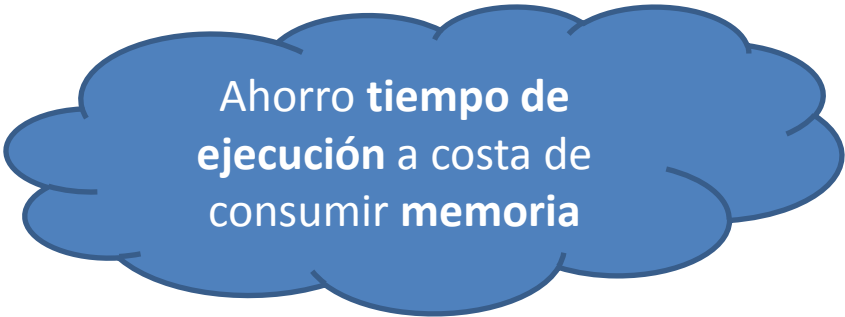


Optimizando con *mnemoizization*

- Las llamadas recursivas de Fibonacci realizan cálculos duplicados
- Esto se puede evitar a costa de almacenar los valores ya calculados

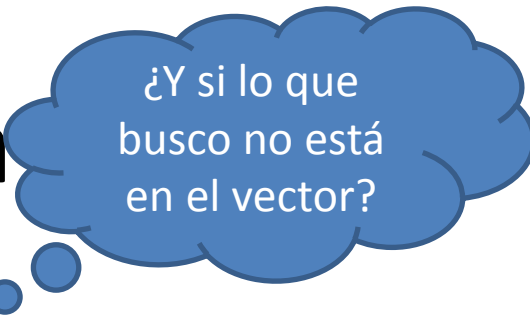
cache = {} • • •

```
def fib_m(n):  
    if n <= 1: return 1  
    if n not in cache:  
        cache[n] = fib_m(n-1) + fib_m(n-2)  
    return cache[n]
```



Ahorro **tiempo de ejecución** a costa de consumir **memoria**

Búsqueda Binaria



¿Y si lo que busco no está en el vector?

- Obtener el **elemento medio** del vector
- **Compararlo** con el elemento buscado
- ¿Lo encontré? → **¡Listo!**
- ¿Lo que busco es **más chico**? → Búsqueda Binaria en el subvector que queda a la **izquierda**
- ¿Lo que busco es **más grande**? → Búsqueda Binaria en el subvector que queda a la **derecha**

Búsqueda Binaria

```
def busqbin(vec, ini, fin, elem):
```

- Definir los tipos de datos `vec`, `ini`, `fin` y `elem`
- Implementar la función que devuelva la **posición** del elemento `e` en el vector `v` de forma recursiva
- ¿Con qué parámetros se invoca a la función la primera vez?
- ¿Precondiciones y poscondiciones?

Implementación

```
def busqbin(vec, ini, fin, elem):  
    if ini>fin: return -1  
    pos=(ini+fin)/2  
    if vec[pos]==e: return pos  
    elif vec[pos]<e: return busqbin(vec, pos+1, fin, elem)  
    else: return busqbin(vec, ini, pos-1, elem)
```



Conversión iterativa

- Todo algoritmo **recursivo** puede también implementarse de manera **iterativa**.

```
def factorial_iterativo(n):  
    res=1  
    if n >= 2:  
        for i in range(n,0,-1): res*=i  
    return res
```

Iterativo vs Recursivo

Iterativo



- Código difícil de entender para soluciones naturalmente recursivas
- A veces requiere variables adicionales
- No requiere llamada a subrutina (pila de llamadas)

Recursivo



- Código claro y conciso para soluciones naturalmente recursivas
- Más difícil de implementar, entender, hacer seguimiento, debug
- Anidamiento de llamadas a subrutinas