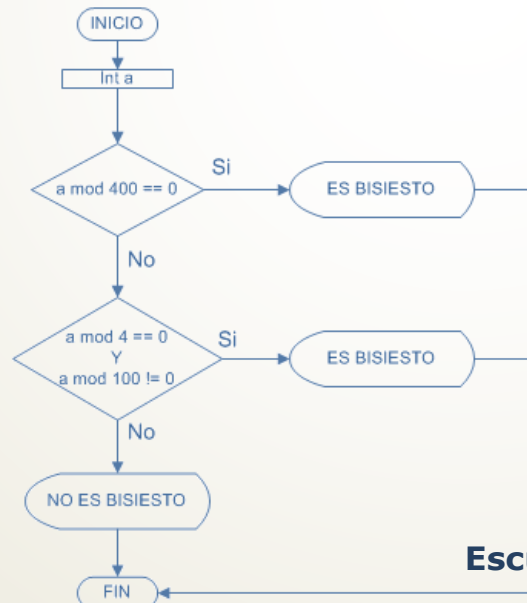


Programación Estructurada

PROGRAMACIÓN MODULAR: RECURSIVIDAD



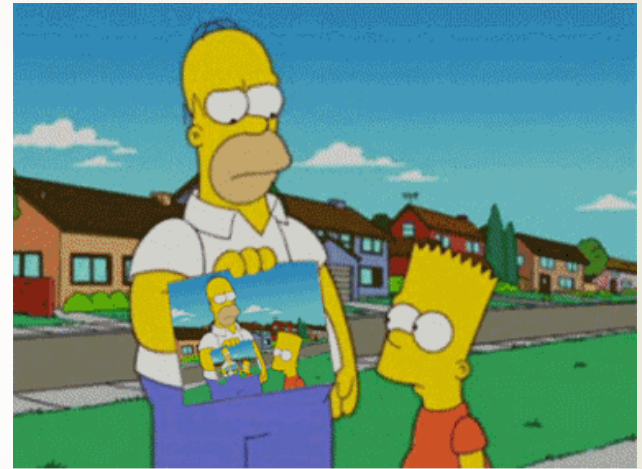
Índice

- Definición de Recursividad
- Razonamiento Recursivo
 - Caso Base
 - Regla Recursiva de Construcción
- Algoritmos Recursivos
- Funciones Recursivas
- Procedimientos Recursivos
- Tipos de Recursividad
- Ventajas y Desventajas

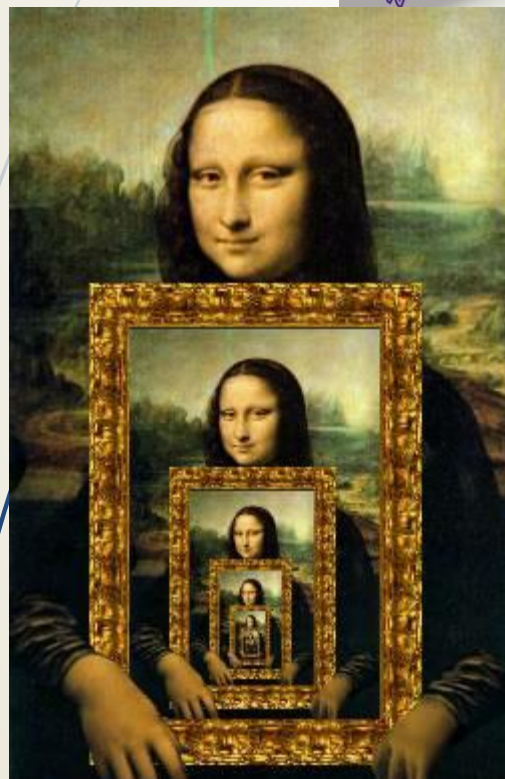


Recursividad

- La recursividad consiste en definir un concepto en términos del propio concepto.
- Una definición recursiva es válida si la referencia a sí misma es relativamente más sencilla que el caso considerado.
- La recursividad expresa un concepto complejo en función de las formas más simples del mismo concepto.



Recursividad



Razonamiento Recursivo (1)

► Partes del razonamiento recursivo:

- Caso Base: indica el problema o **caso más simple** cuya **resolución** es **directa**.
- Regla Recursiva de Construcción: plantea **versiones más simples del problema** original cuyas soluciones parciales permiten resolver el problema principal.



Razonamiento Recursivo (2)

► Consideraciones

1. la **división sucesiva del problema** original en uno o varios **problemas más pequeños**, del **mismo tipo** que el inicial;
2. la resolución de los problemas más sencillos, y
3. la **construcción de las soluciones** de los problemas complejos **a partir de las soluciones** de los problemas más sencillos.

Algoritmo Recursivo (1)

► Características

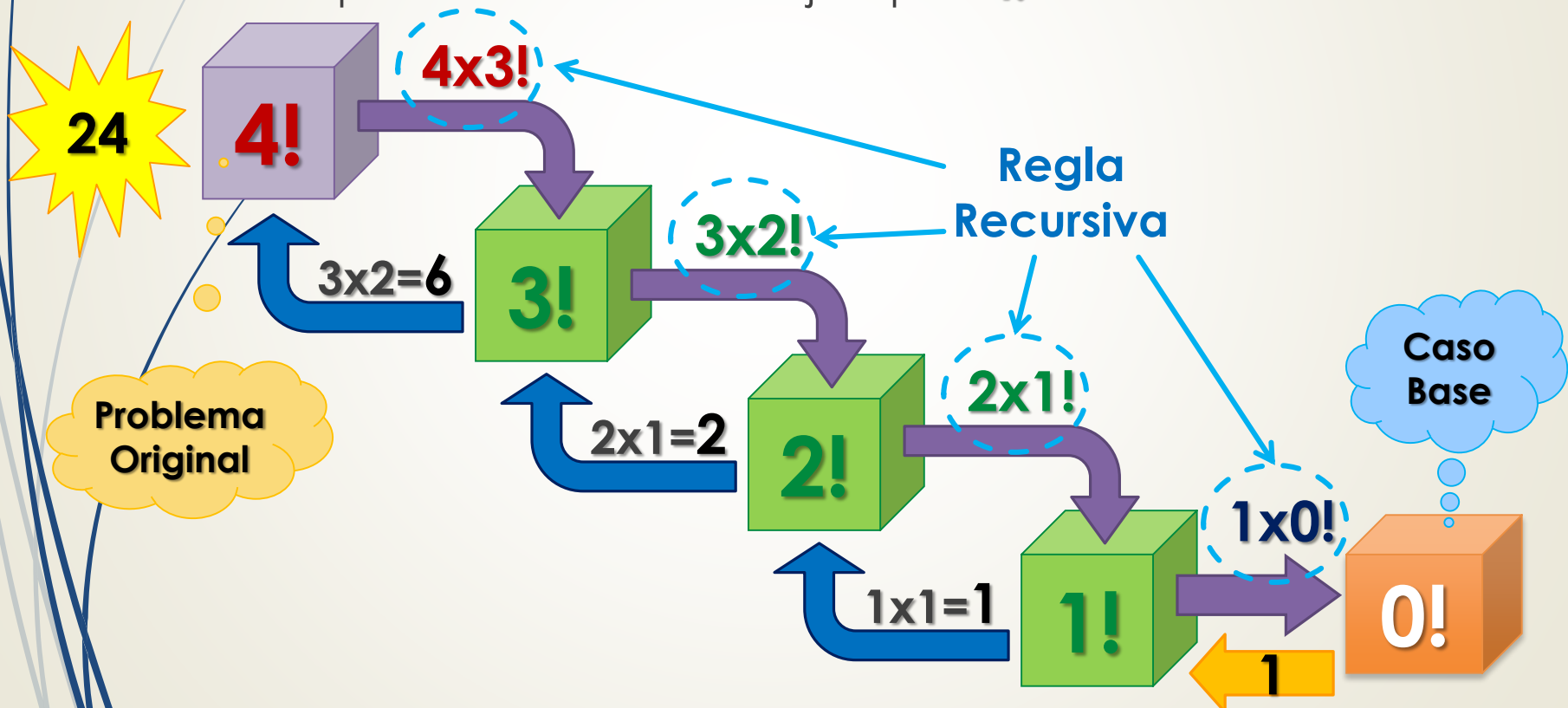
1. el algoritmo debe contener una **llamada a sí mismo**,
2. el problema planteado puede resolverse atacando el mismo **problema** pero **de tamaño menor**,
3. la **reducción del tamaño del problema** permite alcanzar el **caso base**, cuya solución es directa.

Algoritmo Recursivo (2)

- Partes del algoritmo recursivo:
 - iterativa o no recursiva
 - condición de terminación (caso base)
 - recursiva (que reduce el tamaño del problema hasta alcanzar el caso base).
- La parte recursiva y la condición de terminación son obligatorias.
- El caso base siempre debe alcanzarse, sino el algoritmo se invoca indefinidamente.

Algoritmo Recursivo (3)

- Problemas recursivos: cálculo del factorial de un número entero positivo o cero. Por ejemplo: **4!**



Funciones Recursivas

- Un función F es recursiva si:
 1. existen ciertos **argumentos**, llamados **valores base**, para los que la función no se refiere a sí misma.
 2. cada vez que la función se refiera a sí misma, el argumento de la función debe **acercarse más al valor base**.

Ejemplo 1 (1)

FUNCIÓN Factorial(E num: ENTERO): ENTERO

INICIO

SI num=0 ENTONCES

Factorial \leftarrow 1

Caso Base

SINO

Factorial \leftarrow num * Factorial(num-1)

Regla Recursiva

FINSI

FIN

```
int factorial (int num)
{
    if (num==0)
        return 1;
    else
        return num*factorial(num-1);
}
```

Ejemplo 2 (1)

FUNCIÓN Potencia(E a:entero, E b:entero): entero

INICIO

SI b=0 ENTONCES

Potencia \leftarrow 1

Caso Base

SINO

Potencia \leftarrow a * Potencia(a,b-1)

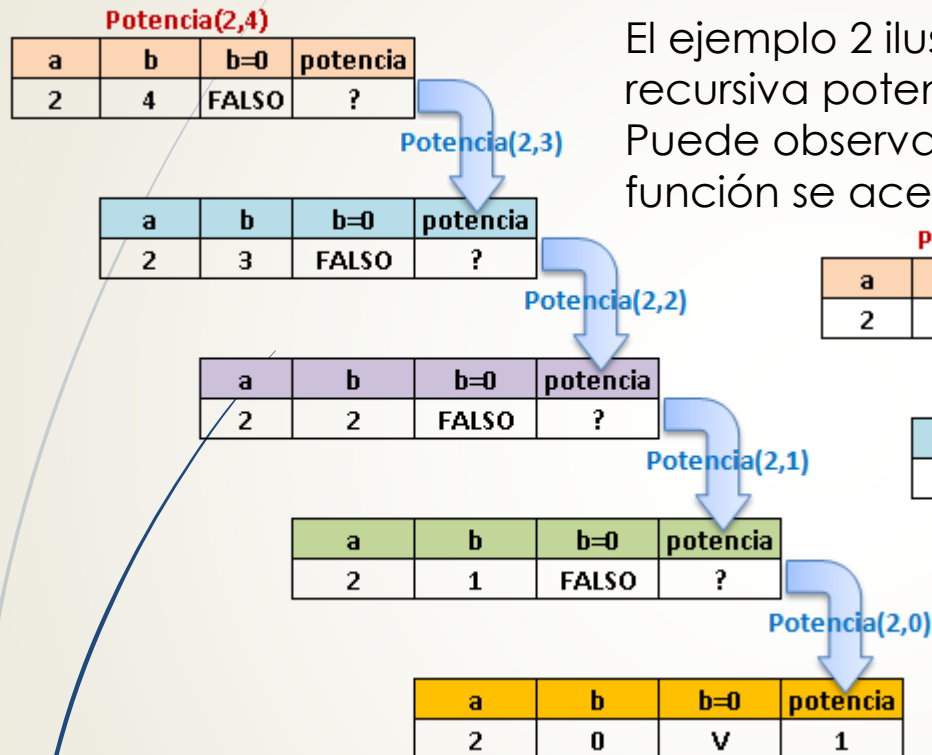
Regla Recursiva

FINSI

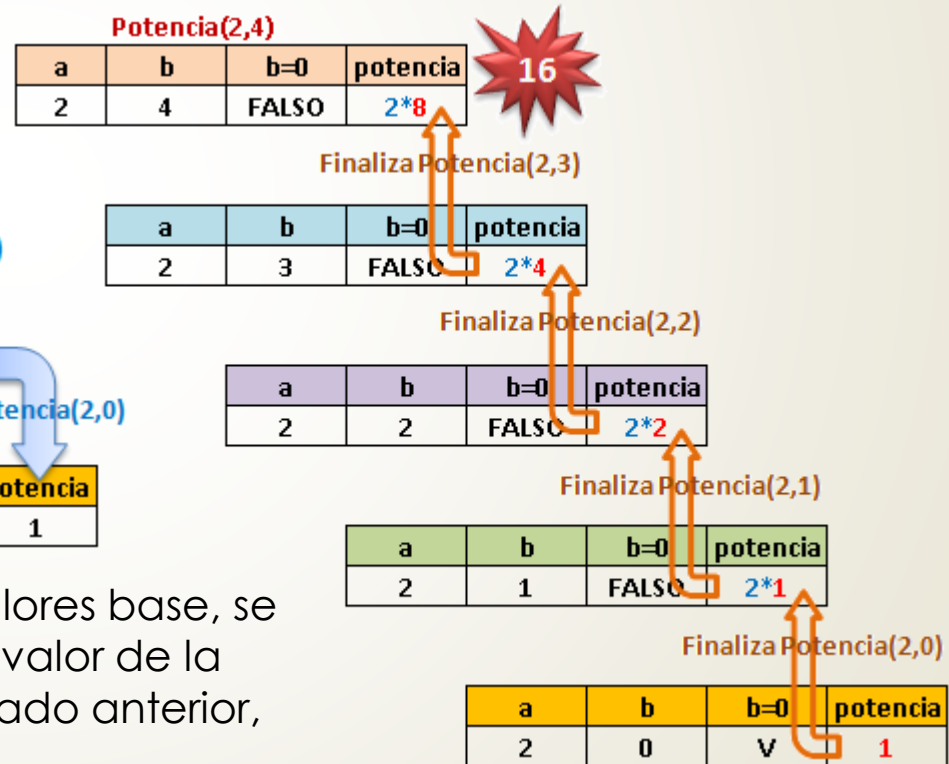
FIN

```
int potencia (int a, int b)
{
    if (b==0)
        return 1;
    else
        return a*potencia(a,b-1);
}
```


Ejemplo 2 (2)



El ejemplo 2 ilustra la ejecución de la función recursiva potencia, con argumentos 2 y 4. Puede observarse que con cada llamado, la función se acerca más a los valores base.



Una vez que se alcanzan los valores base, se obtiene la solución directa y el valor de la función se retorna a cada llamado anterior, calculándose así el valor final.

Procedimientos Recursivos

- ➡ Un procedimiento P es recursivo si:
 1. incluye un cierto criterio, llamado **caso base**, por el que el procedimiento no se llama a sí mismo.
 2. cada vez que el procedimiento se llame a sí mismo (directa o indirectamente), debe estar **más cerca del caso base**.

Ejemplo 3 (1)

PROCEDIMIENTO Mostrar_Numeros (E cantidad: entero)

INICIO

SI cantidad=1 ENTONCES
 ESCRIBIR cantidad

Caso Base

SINO

 Mostrar_Numeros (cantidad-1)
 ESCRIBIR cantidad

Regla Recursiva

FINSI

FIN

```
void mostrar_numeros (int cantidad)
{
    if (cantidad==1)
        cout << cantidad << endl;
    else
        { mostrar_numeros (cantidad-1);
          cout << cantidad << endl; }
}
```

Ejemplo 3 (2)

Mostrar_Numeros(4)

cantidad	cantidad=1	Acción
4	FALSO	?

Mostrar_Numeros(3)

cantidad	cantidad=1	Acción
3	FALSO	?

Mostrar_Numeros(2)

cantidad	cantidad=1	Acción
2	FALSO	?

Mostrar_Numeros(1)

cantidad	cantidad=1	Acción
1	VERDADERO	ESCRIBIR cantidad

El ejemplo 1 muestra la ejecución del procedimiento recursivo *mostrar_numeros*, con argumento 4. Puede observarse que con cada llamado, el procedimiento se acerca más al criterio base.

Ejemplo 3 (3)

Mostrar_Numeros(4)

cantidad	cantidad=1	Acción
4	FALSO	ESCRIBIR cantidad

Finaliza Mostrar_Numeros(3)

cantidad	cantidad=1	Acción
3	FALSO	ESCRIBIR cantidad

Finaliza Mostrar_Numeros(2)

cantidad	cantidad=1	Acción
2	FALSO	ESCRIBIR cantidad

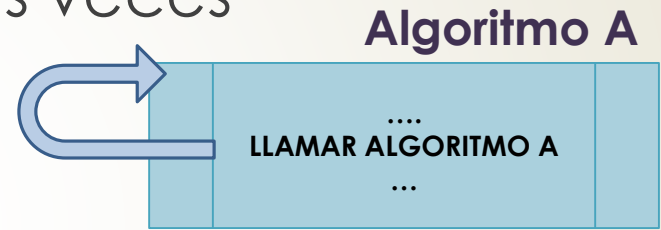
Finaliza Mostrar_Numeros(1)

cantidad	cantidad=1	Acción
1	VERDADERO	ESCRIBIR cantidad

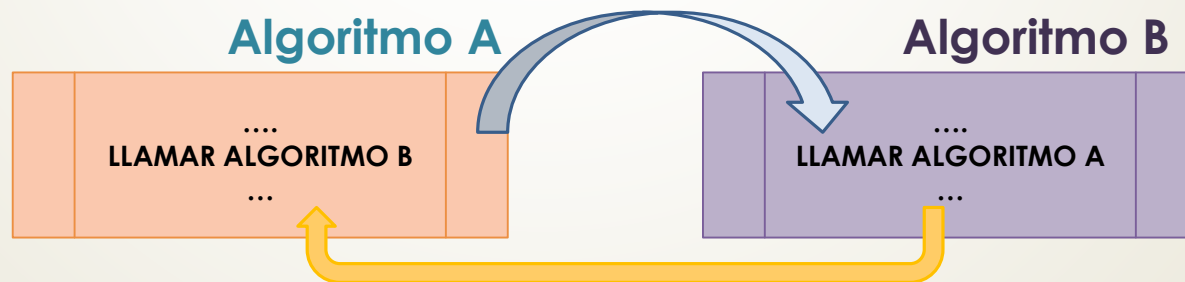
Una vez cumplido el criterio base, se realizan las acciones apropiadas y se retorna el control a cada llamado anterior.

Tipos de Recursividad

- Recursividad Directa (simple): un algoritmo se invoca a sí mismo una o más veces directamente.



- Recursividad Indirecta (mutua): un algoritmo A invoca a otro algoritmo B y éste a su vez invoca al algoritmo A.



Ventajas y Desventajas

➤ Ventajas

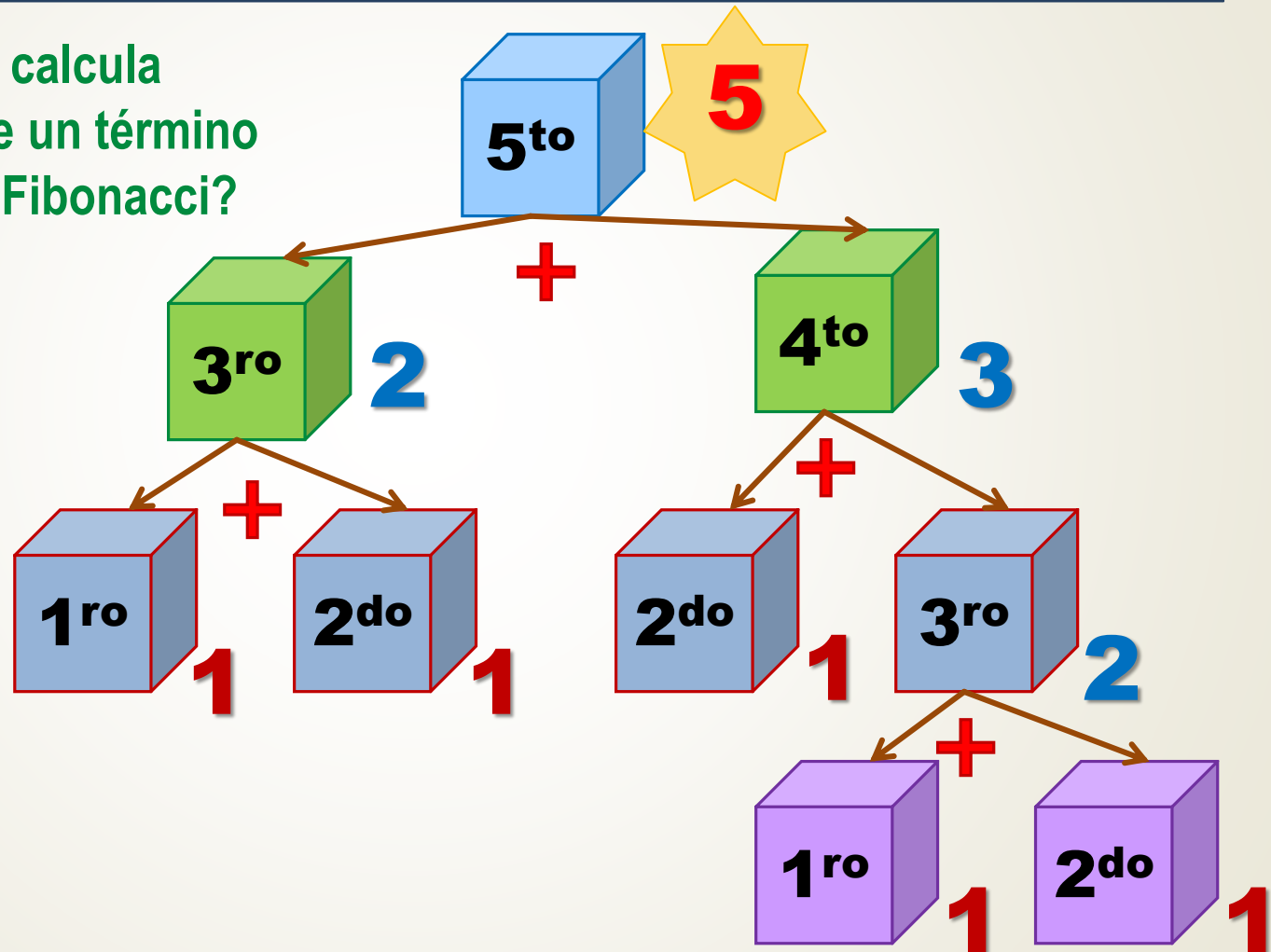
- Fácil comprensión
- Fácil comprobación
- Solución sencilla a problemas de naturaleza recursiva (versiones iterativas complicadas).

➤ Desventajas

- Las soluciones recursivas, en general, son menos eficientes que las iterativas (consumo de memoria)

Serie de Fibonacci

¿Cómo se calcula
recursivamente un término
de la serie de Fibonacci?



Bibliografía

- Sznajdleder, Pablo Augusto. Algoritmos a fondo. Alfaomega. 2012.
- López Román, Leobardo. Programación estructurada y orientada a objetos. Alfaomega. 2011.
- De Giusti, Armando *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta, 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Joyanes Aguilar, Luis. Programación en Turbo Pascal. Mc Graw Hill. 1990.