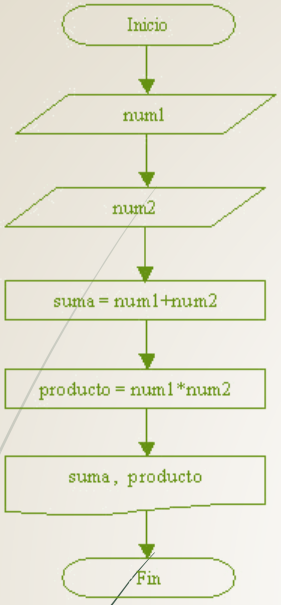
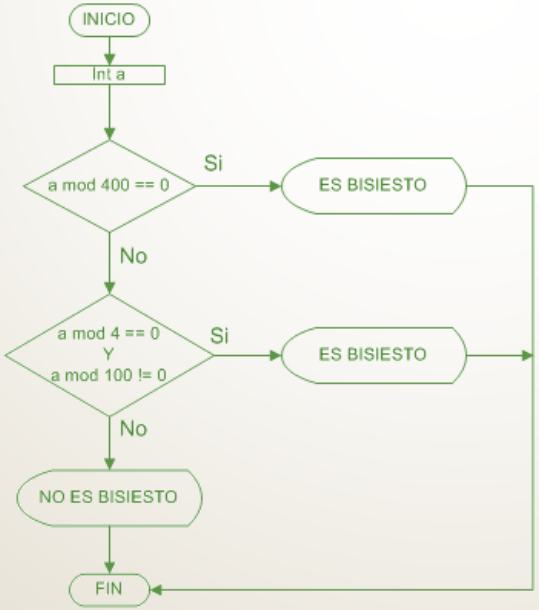


Programación Estructurada

Modularidad: Recursividad



Facultad de Ingeniería
Universidad Nacional de Jujuy

Índice

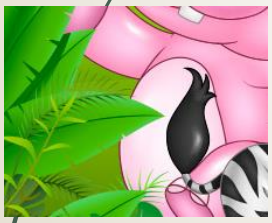
- Definición de Recursividad
- Razonamiento Recursivo
- Algoritmos Recursivos
  - Funciones Recursivas
  - Procedimientos Recursivos
- Tipos de Recursividad
- Ventajas y Desventajas

Problema: Rompecabezas (1)



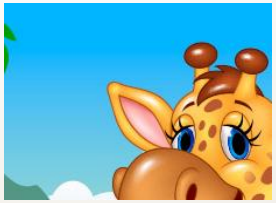




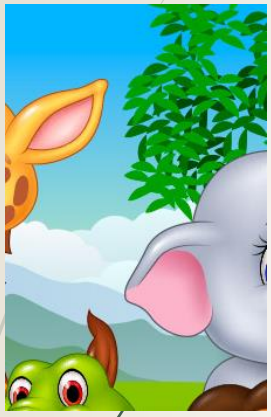










Problema: Rompecabezas (2)



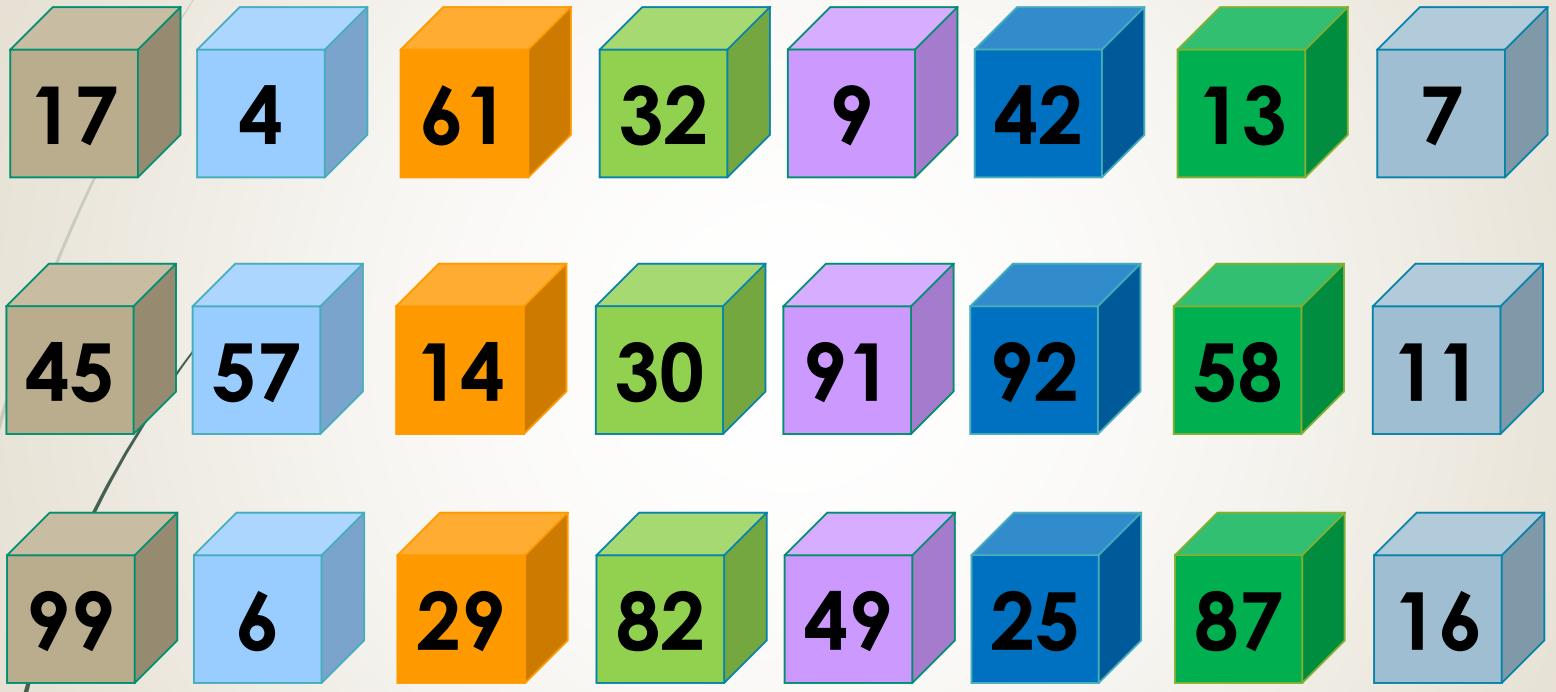
Problema: Rompecabezas (3)

Problema: Rompecabezas (4)

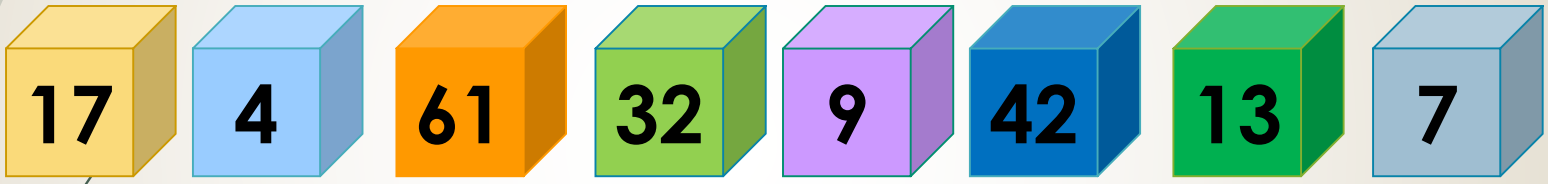
Problema: Ordenar cubos (1)

17	4	61	32	9	42	13	7
22	17	98	33	16	5	38	76
45	57	14	30	91	92	58	11
3	48	18	71	77	12	1	66
99	6	29	82	49	25	87	16

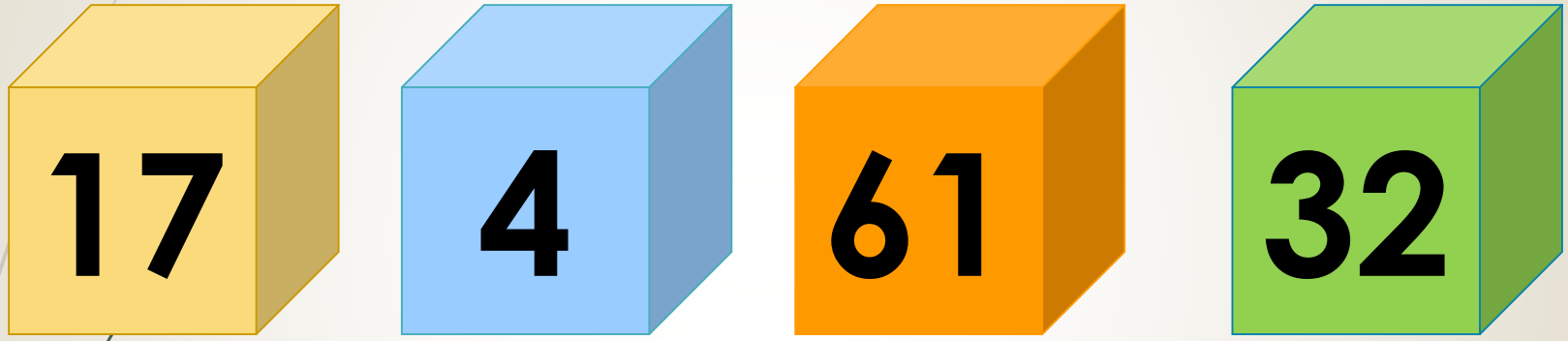
Problema: Ordenar cubos (1)



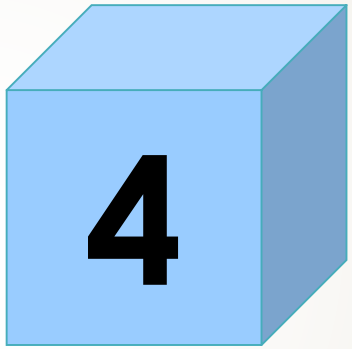
Problema: Ordenar cubos (3)



Problema: Ordenar cubos (4)

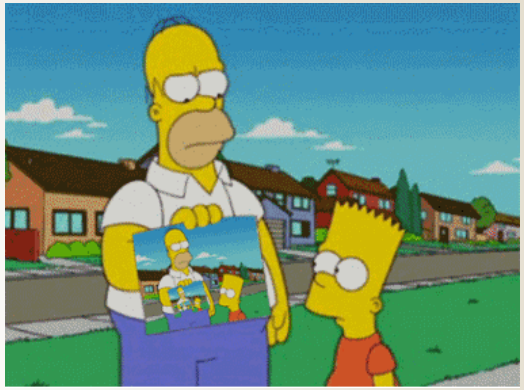


Problema: Ordenar cubos (1)



Recursividad

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



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)

Factorial: el factorial de un número n se define como el producto de los valores naturales entre 1 y n.

Por ejemplo: 4! = 4 x 3 x 2 x 1

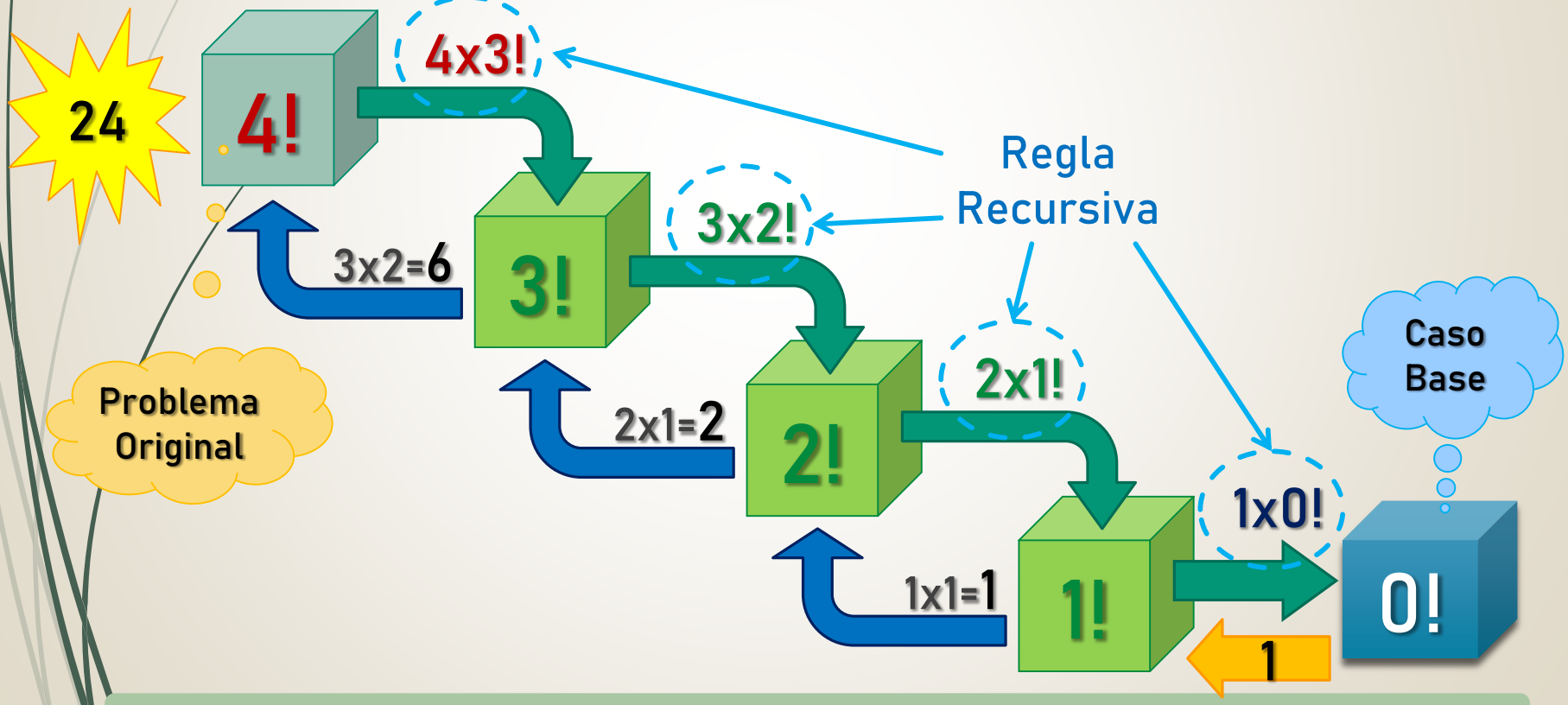
Razonamiento recursivo

4! = 3! ?	4! = 3! x 4	4! = 6 x 4	24
3! = 2! ?	3! = 2! x 3	3! = 2 x 3	
2! = 1! ?	2! = 1! x 2	2! = 1 x 2	
1! = 0! ?	1! = 0! x 1	1! = 1 x 1	

0! = 1 Caso base

Algoritmo Recursivo (4)

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



Funciones Recursivas

- Una 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: Factorial

```
FUNCIÓN Factorial(E num: ENTERO): ENTERO
INICIO
        SI num=0 ENTONCES
             Factorial←1
        SINO
             Factorial←num * Factorial(num-1)
        FINSI
FIN
```

Caso Base

Regla Recursiva

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

Ejemplo: Potencia (1)

Calcular la potencia mediante recursividad

Por ejemplo: $2^4 = 2 \times 2 \times 2 \times 2$

Razonamiento recursivo

$2^4 = 2^3$?	$2^4 = 2^3 \times 2$	$2^4 = 8 \times 2$	16
$2^3 = 2^2$?	$2^3 = 2^2 \times 2$	$2^3 = 4 \times 2$	
$2^2 = 2^1$?	$2^2 = 2^1 \times 2$	$2^2 = 2 \times 2$	
$2^1 = 2^0$?	$2^1 = 2^0 \times 2$	$2^1 = 1 \times 2$	

$2^0 = 1$ Caso base

Ejemplo: Potencia (2)

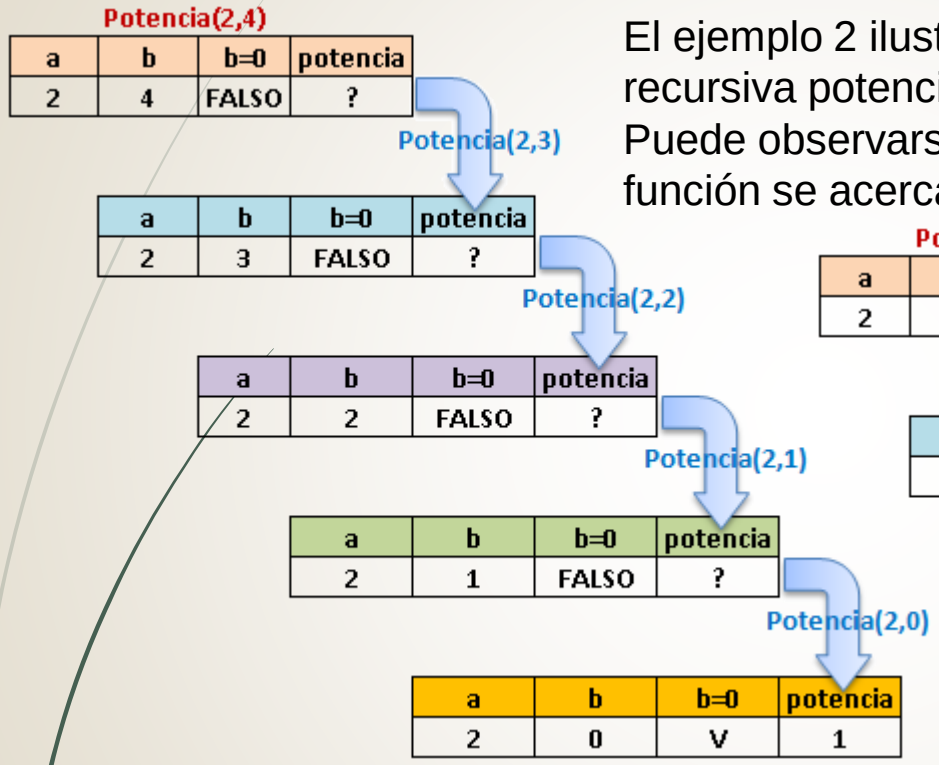
```
FUNCIÓN Potencia(E a:entero, E b:entero): entero
INICIO
        SI b=0 ENTONCES
             Potencia←1
        SINO
             Potencia←a * Potencia(a,b-1)
        FINSI
FIN
```

Caso Base

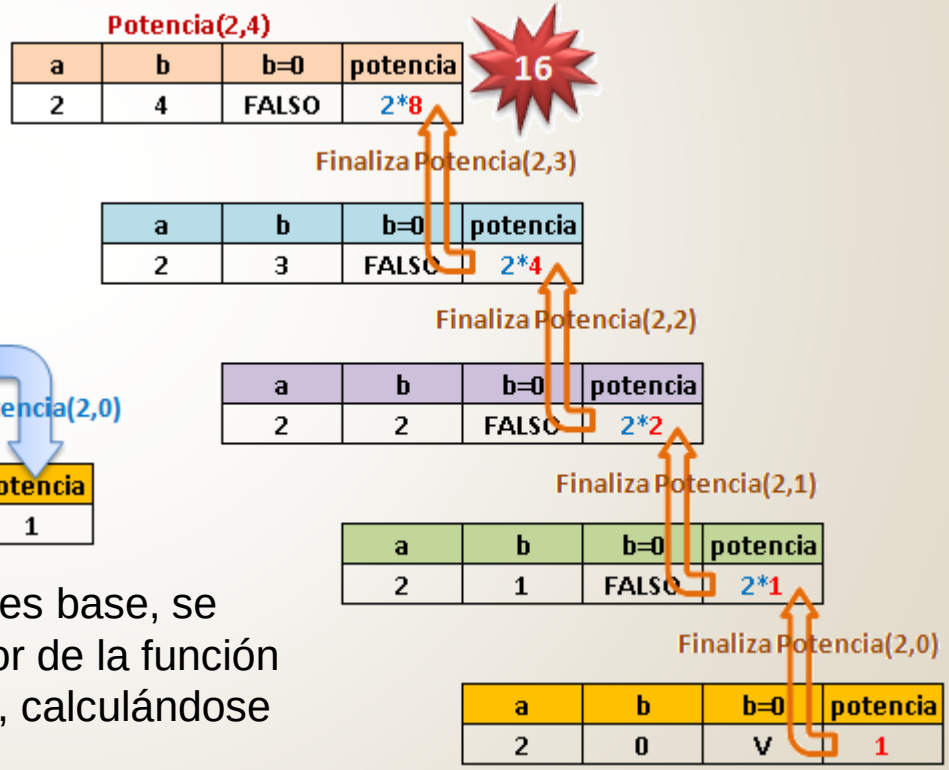
Regla Recursiva

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

Ejemplo: Potencia (3)



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: Procedimiento recursivo

```
PROCEDIMIENTO Mostrar_Numeros(E limite: entero)
INICIO
    SI limite=1 ENTONCES
       ESCRIBIR limite
    SINO
       Mostrar_Numeros(limite-1)
       ESCRIBIR limite
    FINSI
FIN
```

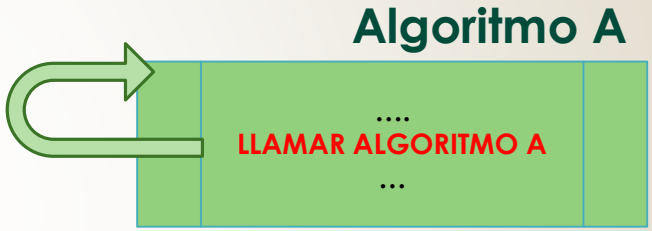
Caso Base

Regla Recursiva

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

Tipos de Recursividad

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



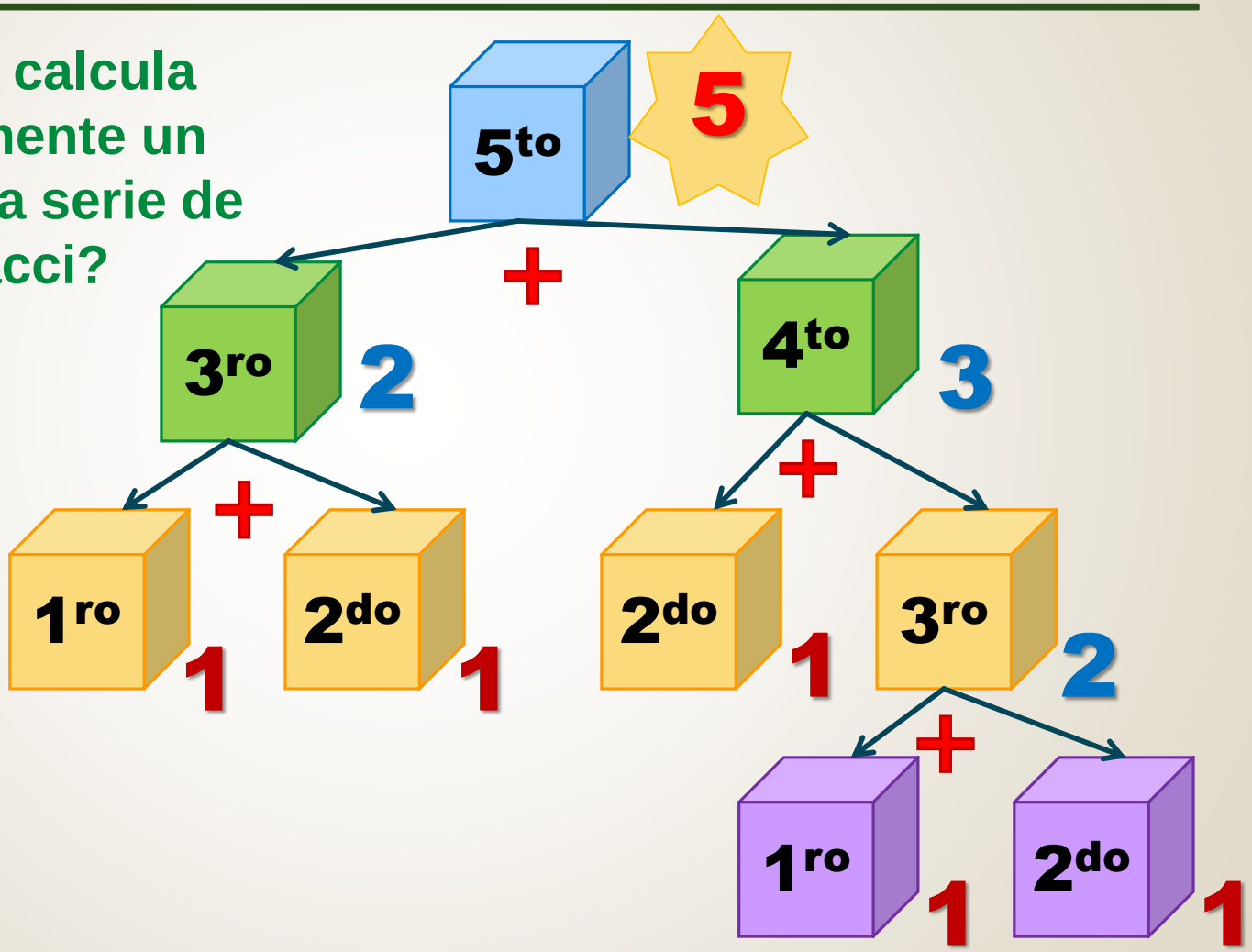
- Recursividad Indirecta (mutua): un algoritmo *A* invoca a otro algoritmo *B* y éste a su vez invoca al algoritmo *A*.

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.