

Recursividad

¿Qué es una función o un método recursivo?

Una función que se llama a sí misma se denomina **recursiva**

$$n! = \left\{ \begin{array}{ll} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n > 0 \end{array} \right|$$

$$x^n = \left\{ \begin{array}{ll} 1 & \text{si } n = 0 \\ x \cdot x^{n-1} & \text{si } n > 0 \end{array} \right|$$

Utilidad

Cuando la solución de un problema se puede expresar en términos de la resolución de un problema de la misma naturaleza, aunque de menor complejidad.

Divide y vencerás:

**Un problema complejo se divide en otros problemas más sencillos
(del mismo tipo)**

Utilidad

Sólo tenemos que conocer la solución no recursiva para algún caso sencillo (denominado caso base) y hacer que la división de nuestro problema acabe recurriendo a los casos base que hayamos definido.

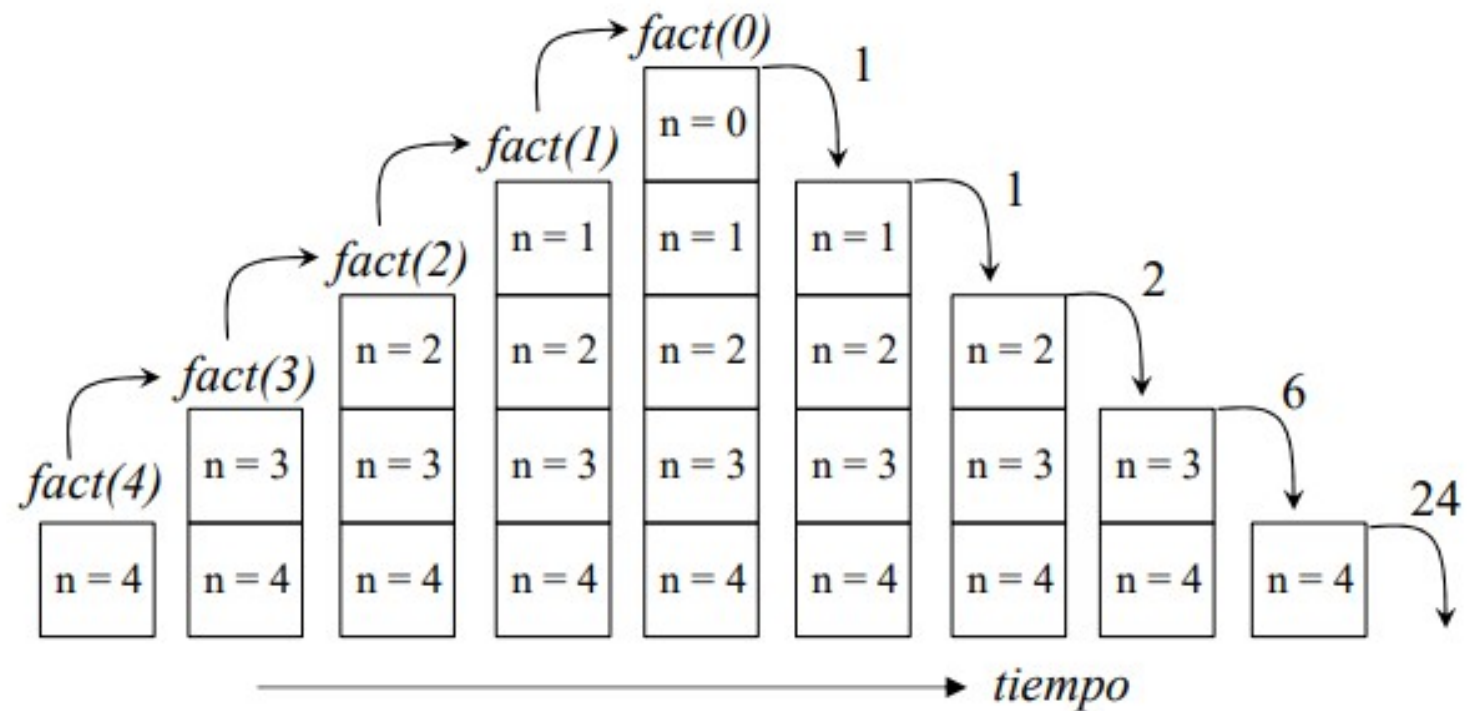


Ejemplo

¿Cuántas formas hay de colocar n objetos en orden?

- Podemos colocar cualquiera de los n objetos en la primera posición.
- A continuación, colocamos los $(n-1)$ objetos restantes.
- Por tanto: $P(n) = n P(n-1) = n!$

Factorial de un número



Factorial

```
static int factorial (int n)
{
    int resultado;

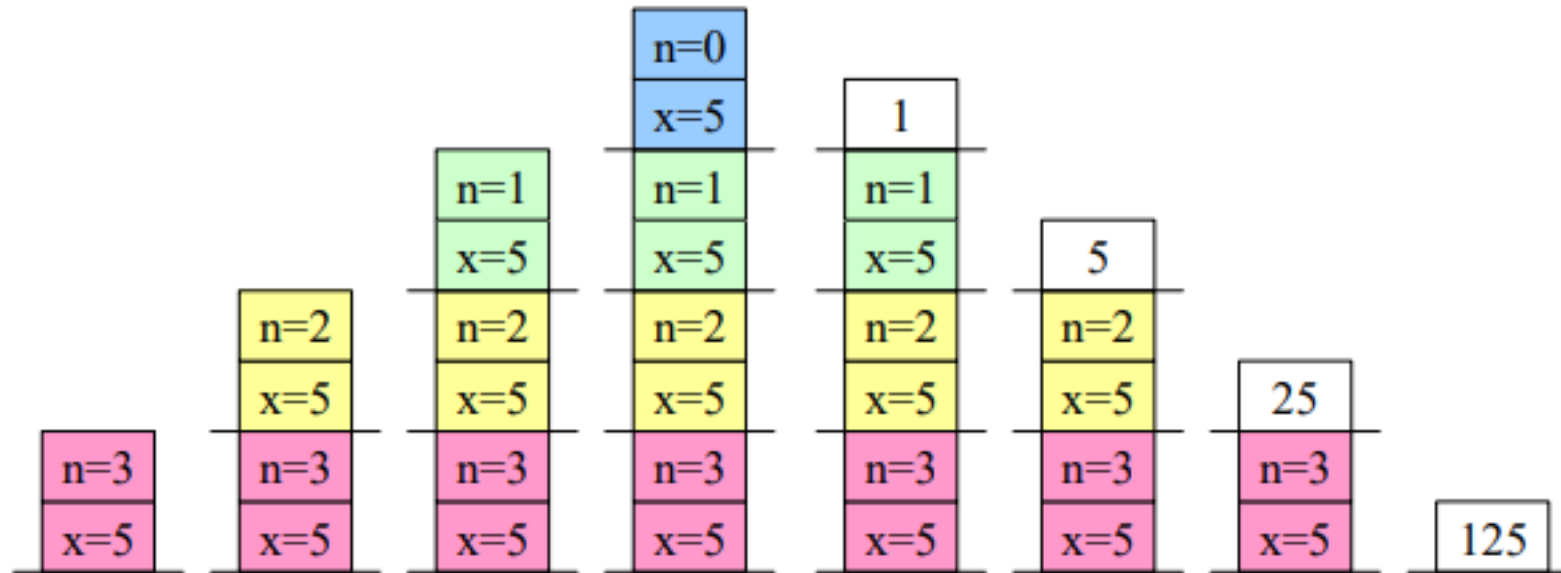
    if (n==0)                                // Caso base
        resultado = 1;
    else                                      // Caso general
        resultado = n*factorial(n-1);

    return resultado;
}
```

Funcionamiento de un algoritmo recursivo

- Se descompone el problema en problemas de menor complejidad (algunos de ellos de la misma naturaleza que el problema original).
- Se resuelve el problema para, al menos, un caso base.
- Se compone la solución final a partir de las soluciones parciales que se van obteniendo.

Potencia de un número



Cálculo de 5^3

Potencia de un número

```
static int potencia (int x, int n)
{
    if (n==0)                                // Caso base
        return 1;
    else                                     // Caso general
        return x * potencia(x,n-1);
}
```

Diseño de algoritmos recursivos

1. Resolución de problema para los casos base:

- Sin emplear recursividad.
- Siempre debe existir algún caso base.

2. Solución para el caso general:

- Expresión de forma recursiva.
- Pueden incluirse pasos adicionales (para combinar las soluciones parciales).

Diseño de algoritmos recursivos

Siempre se debe avanzar hacia un caso base:

Las llamadas recursivas simplifican el problema y, en última instancia, los casos base nos sirven para obtener la solución.

- Los casos base corresponden a situaciones que se pueden resolver con facilidad.
- Los demás casos se resuelven recurriendo, antes o después, a alguno(s) de los casos base.

De esta forma, podemos resolver problemas complejos que serían muy difíciles de resolver directamente.

Recursividad vs Iteración

Aspectos que hay que considerar
al decidir cómo implementar la solución a un problema
(de forma iterativa o de forma recursiva):

- La **carga computacional**
(tiempo de CPU y espacio en memoria)
asociada a las llamadas recursivas.
- La **redundancia**
(algunas soluciones recursivas
resuelven el mismo problema en repetidas ocasiones).
- La **complejidad de la solución**
(en ocasiones, la solución iterativa es muy difícil de encontrar).
- La concisión, legibilidad y elegancia del **código resultante**
(la solución recursiva del problema puede ser más sencilla).