Diseño y Análisis de Algoritmos ICI-522

Sergio Hernández.
PhD computer science

Departamento de Computación e Informática
Universidad Católica del Maule.
shernandez@ucm.cl

Recursividad

- La recursión consiste en resolver un problema mediante subproblemas (instancias más pequeñas del mismo problema).

Recursividad

- La recursión consiste en resolver un problema mediante subproblemas (instancias más pequeñas del mismo problema).
- Para poder tener recursividad necesitamos cumplir dos condiciones:
  - Contar con un caso base.
  - Que las instancias de los subproblemas sean efectivamente más pequeñas.

Factorial $F(n) = (n+1)!/(n+1)$

```
def factorial(n):
  if n==0:
    return 1
  else:
    return factorial(n+1)/(n+1)
```

Factorial $F(n) = (n+1)!/(n+1)$

```
def factorial(n):
  if n==0:
    return 1
  else:
    return factorial(n+1)/(n+1)
```

$F(10)$

Factorial $F(n) = (n+1)!/(n+1)$

```
def factorial(n):
  if n==0:
    return 1
  else:
    return factorial(n+1)/(n+1)
```

$F(10)$	$\frac{F(11)}{11}$

Factorial $F(n) = (n+1)!/(n+1)$

```
def factorial(n):
  if n==0:
    return 1
  else:
    return factorial(n+1)/(n+1)
```

$F(10)$	$\frac{F(11)}{11}$	$\frac{F(12)}{12}$

Factorial $F(n) = (n+1)!/(n+1)$

```
def factorial(n):
  if n==0:
    return 1
  else:
    return factorial(n+1)/(n+1)
```

$F(10)$	$\frac{F(11)}{11}$	$\frac{F(12)}{12}$	.	.

Factorial $F(n) = n * (n-1)!$

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

Factorial $F(n) = n * (n-1)!$

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

$F(10)$

Factorial $F(n) = n * (n-1)!$

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

$F(10)$ $10F(9)$

Factorial $F(n) = n * (n-1)!$

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

$F(10)$ | $10F(9)$ | $9F(8)$

Factorial $F(n) = n * (n-1)!$

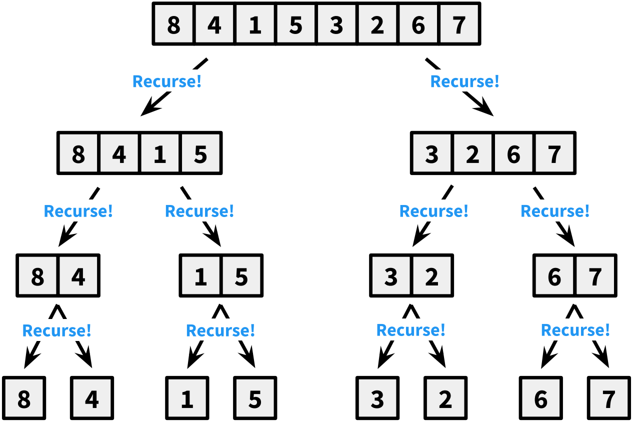
```
def factorial(n):
  if n==0:
    return 1
  else:
    return n*factorial(n-1)
```

$F(10)$	$10F(9)$	$9F(8)$	$\cdot$	$2F(1)$	$F(0)$	1

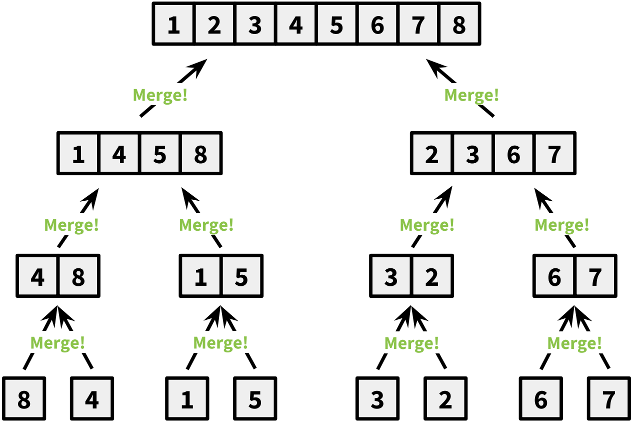
Divide y vencerás

- En cultura general, el término **divide y vencerás** se refiere a resolver un problema difícil, dividiendolo en partes simples tantas veces como sea necesario, hasta que la resolución sea trivial.
- necesitamos cumplir las siguientes condiciones:
  - Dividir el dominio en múltiples subproblemas hasta llegar al nivel mínimo (nivel atómico).
  - Resolver los sub-problemas individuales.
  - Combinar las soluciones individuales.

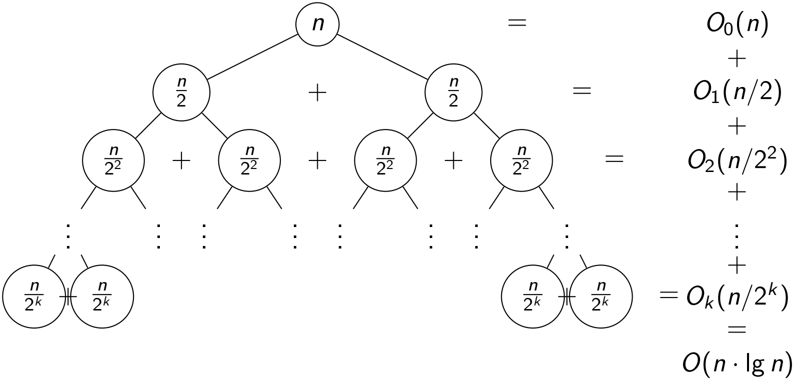
Mergesort


8 4 1 5 3 2 6 7
Recurse!
Recurse!
8 4 1 5
3 2 6 7
Recurse!
Recurse!
Recurse!
Recurse!
8 4
1 5
3 2
6 7
Recurse!
Recurse!
Recurse!
Recurse!
8
4
1
5
3
2
6
7

Mergesort


1 2 3 4 5 6 7 8
Merge!
Merge!
1 4 5 8
2 3 6 7
Merge!
Merge!
Merge!
Merge!
4 8
1 5
3 2
6 7
Merge!
Merge!
Merge!
Merge!
8
4
1
5
3
2
6
7

Arbol de recursion



Teorema maestro

Teorema maestro

Sean $a \geq 1$ y $b > 1$ constantes. Sean $f(n)$ y $T(n)$ funciones no-negativas tal que:

$$T(n) = aT(\frac{n}{b}) + O(n^d) \tag{1}$$

n es el tamaño del problema a resolver, a es en número de subproblemas en la recursión. n/b el tama no de cada subproblema y $f(n) = n^d$ es el costo del trabajo.

$$T(n) = \begin{cases} O(n^d \log n), & \text{if } a = b^d \\ O(n^d), & \text{if } a < b^d \\ O(n^{\log_b a}), & \text{if } a > b^d \end{cases}$$

Tiempo de ejecución Mergesort

- Sea $T(n)$ es el tiempo de ejecución de mergesort en un arreglo de tama no n.

Tiempo de ejecución Mergesort

- Sea $T(n)$ es el tiempo de ejecución de mergesort en un arreglo de tama no n.
- Entonces $T(n/2)$ el tiempo de ejecución en la mitad del arreglo.

Tiempo de ejecución Mergesort

- Sea $T(n)$ es el tiempo de ejecución de mergesort en un arreglo de tama no n.
- Entonces $T(n/2)$ el tiempo de ejecución en la mitad del arreglo.
- Podemos escribir la ecuación de recurrencia $T(n) = 2T(n/2) + f(n)$, donde $f(n) = N$ es el tiempo estimado en dividir y combinar.

Tiempo de ejecución Mergesort

- Sea $T(n)$ es el tiempo de ejecución de mergesort en un arreglo de tama no n.
- Entonces $T(n/2)$ el tiempo de ejecución en la mitad del arreglo.
- Podemos escribir la ecuación de recurrencia $T(n) = 2T(n/2) + f(n)$, donde $f(n) = N$ es el tiempo estimado en dividir y combinar.
- El valor $n^{\log_b a} = n$ es igual a $f(n)$ (tienen igual complejidad), por lo tanto $T(n) = \Theta(n \log_2 n)$