



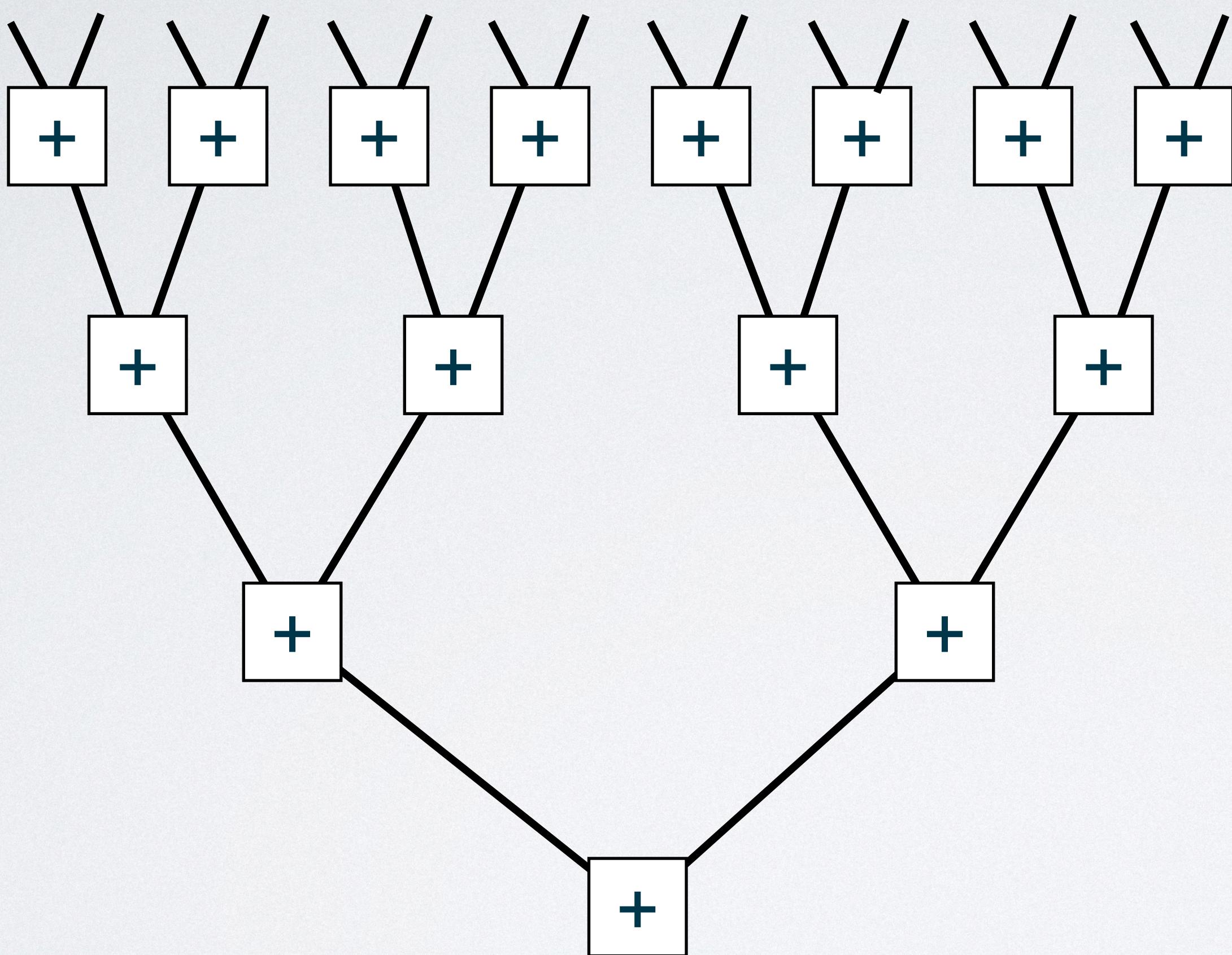
MODELO DE COSTO

EDyAI 2023

MODELO DE COSTO

- Costo secuencial: Trabajo (**W**)
 - costo con 1 procesador
 - $|E|$
- Costo paralelo: Profundidad (**S**)
 - costo con ∞ procesadores
 - $\text{Max}_{p \in \text{Path}(g)} |p|$

EJEMPLO: SUMA DE N NÚMEROS



$$W(n) \in \Theta(n)$$

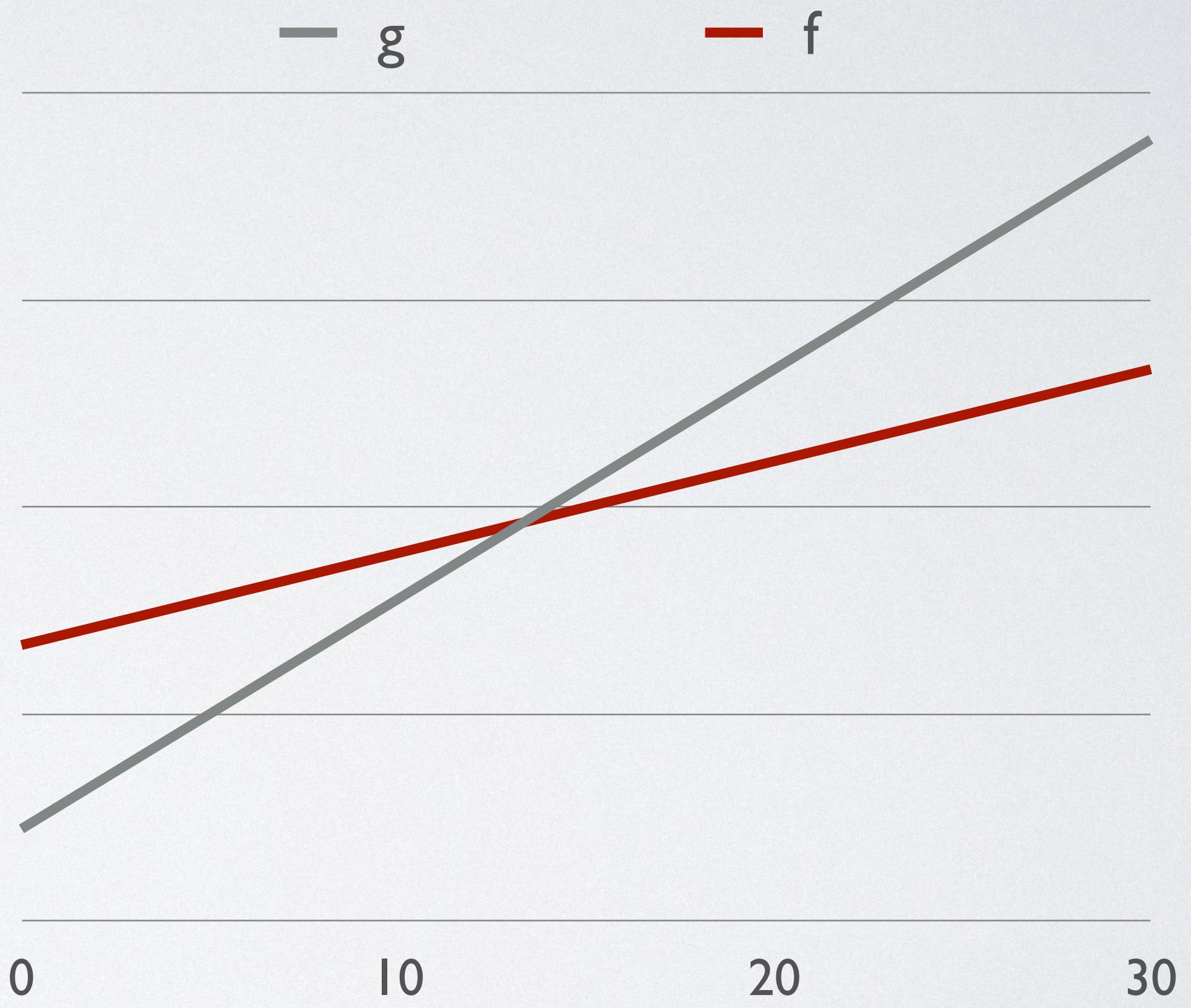
$$S(n) \in \Theta(\lg n)$$

NOTACIÓN ASINTÓTICA

- Nos da una idea de la **velocidad de crecimiento** de una función
- Nosotros la aplicaremos para **funciones de costo** (tiempo, memoria, energía, etc).

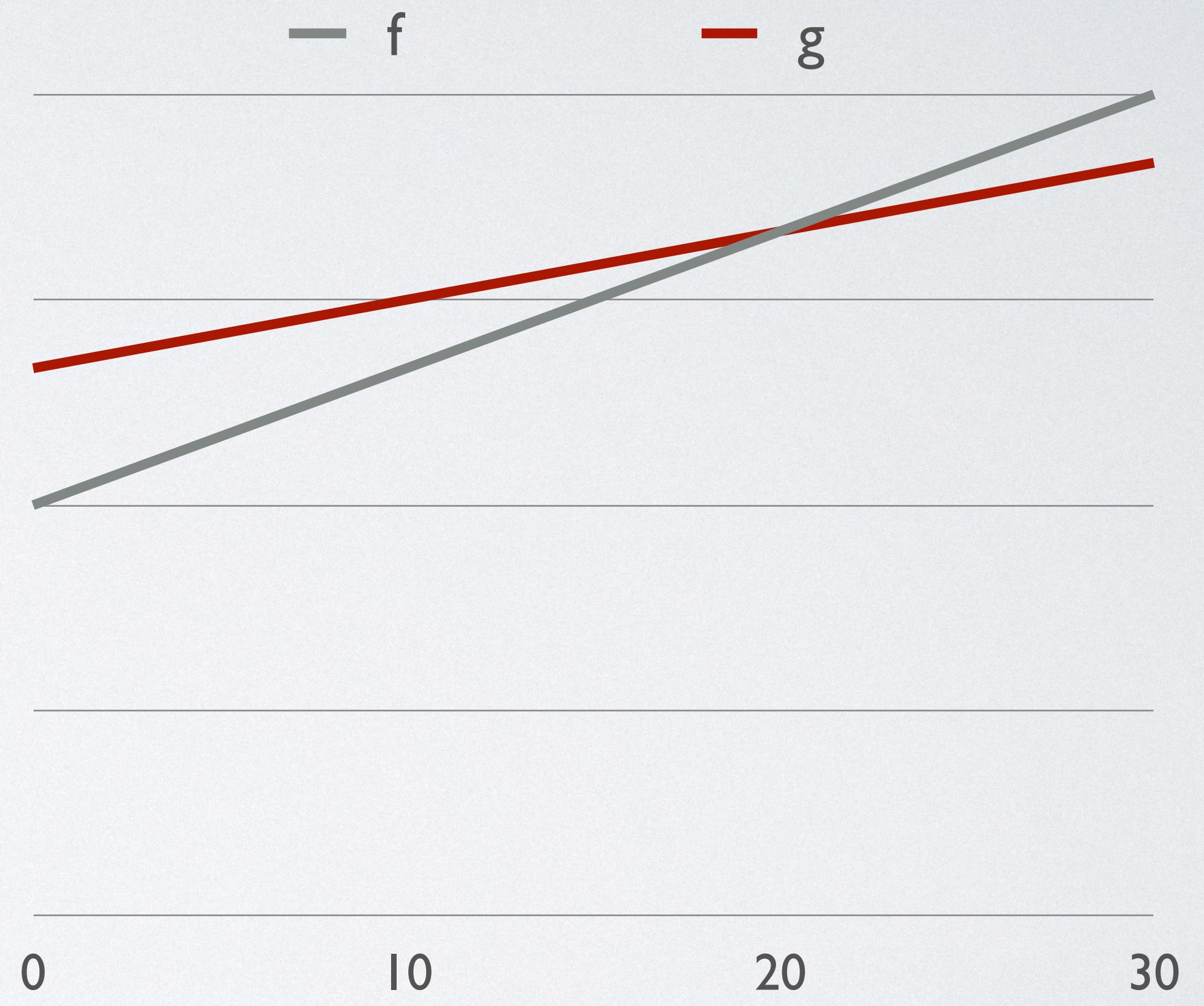
NOTACIÓN O

- Sean $f, g: \mathbb{N} \rightarrow \mathbb{R}$
- f tiene orden de crecimiento $O(g)$ (escribimos $f \in O(g)$) si existen $c \in \mathbb{R}, n_0 \in \mathbb{N}$, tal que
$$\forall n \geq n_0 . 0 \leq f(n) \leq c \cdot g(n)$$
- La notación O me da una cota superior al crecimiento de la función.



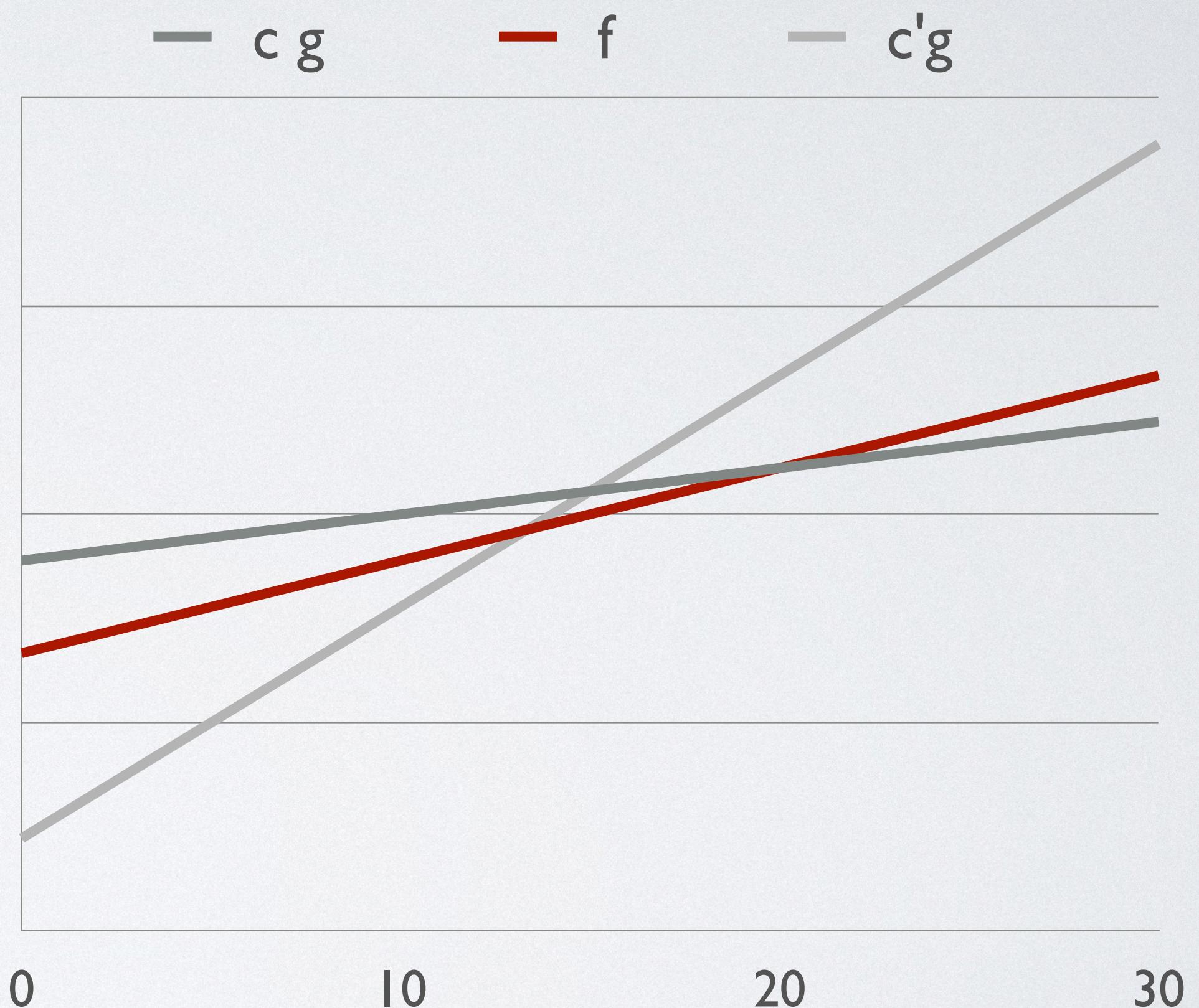
NOTACIÓN Ω

- Sean $f, g: \mathbb{N} \rightarrow \mathbb{R}$
- f tiene orden de crecimiento $\Omega(g)$ (escribimos $f \in \Omega(g)$) si existen $c \in \mathbb{R}, n_0 \in \mathbb{N}$, tal que
$$\forall n \geq n_0 . 0 \leq c \cdot g(n) \leq f(n)$$
- La notación Ω me da una cota inferior al crecimiento de la función.



NOTACIÓN Θ

- Sean $f, g: \mathbb{N} \rightarrow \mathbb{R}$
- f tiene orden de crecimiento $\Theta(g)$ (escribimos $f \in \Theta(g)$) si $f \in O(g)$ y $g \in O(f)$
- La notación Θ me da una cota ajustada del crecimiento de la función.



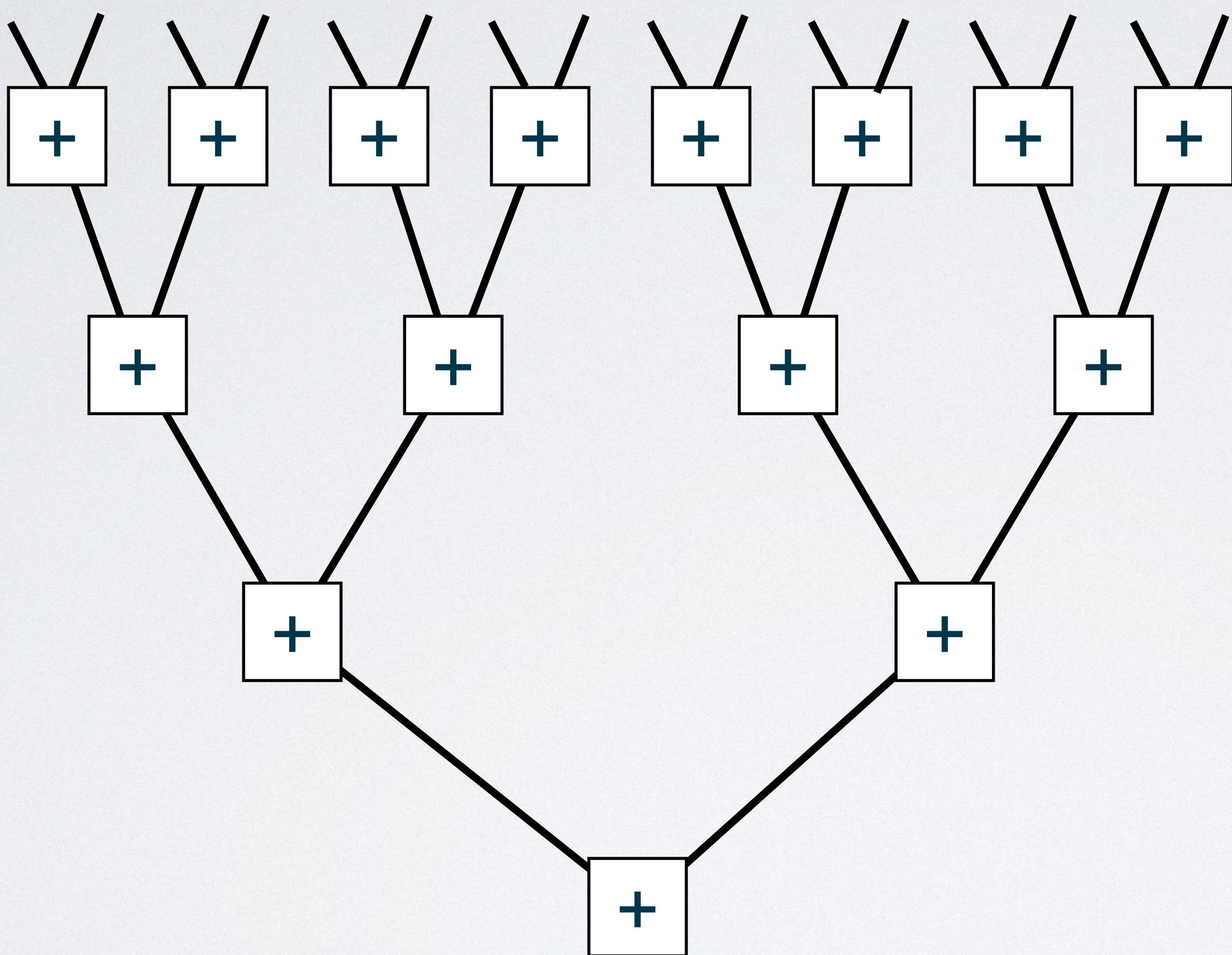
EJEMPLOS TÍPICOS

- Las siguientes son ejemplos típicos, en orden de velocidad de crecimiento:

$$\Theta(1) \quad \Theta(\lg n) \quad \Theta(n) \quad \Theta(n \lg n) \quad \Theta(n^2) \quad \Theta(n^3) \quad \Theta(2^n) \quad \Theta(n!)$$

$$O(1) \subset O(\lg n) \subset O(n) \subset O(n \lg n) \subset O(n^2) \subset O(n^3) \subset O(2^n) \subset O(n!)$$

EJEMPLO: SUMA DE N NÚMEROS



$$W(n) \in \Theta(n)$$

$$S(n) \in \Theta(\lg n)$$

PARALELISMO

$$P = \frac{W}{S}$$

- Indica cuantos procesadores podemos usar eficientemente
- Sumar n números es

$$P \leq \frac{kn}{k' \lg n} \in O\left(\frac{n}{\lg n}\right)$$

PARALELISMO

$$P = \frac{W}{S}$$

- Queremos algoritmos con W del orden del mejor algoritmo secuencial
- Entre estos, elegimos el de mayor P

SCHEDULER VORAZ

- El scheduler reparte tareas entre los procesadores
- Un scheduler es **voraz** si cuando
 - hay un procesador libre y
 - hay tareas para ejecutar
 - la tarea es asignada inmediatamente.

PRINCIPIO DEL SCHEDULER VORAZ (BRENT)

Una expresión con trabajo W y prof. S puede correr en una máquina con p procesadores que usa un scheduler voraz en tiempo

$$T < \frac{W}{p} + S$$

PRINCIPIO DEL SCHEDULER VORAZ

$$T < \frac{W}{p} + S$$

- La cota es buena
- En términos de paralelismo

$$T < \frac{W}{p} + S = \frac{W}{p} + \frac{W}{P} = \frac{W}{p} \left(1 + \frac{p}{P}\right)$$

- Si $p \ll P$ la cota es prácticamente óptima ya que entonces $\frac{p}{P} \rightarrow 0$.

SUPUESTOS

- Costo de comunicación bajo
- Latencia
- Ancho de Banda
- Scheduler Voraz

ANÁLISIS DE ALGORITMOS

- Es necesario especificar el modelo de costo
- No buscamos estimar tiempo de ejecución si no tener una cota asintótica

MODELOS DE COSTO BASADOS EN

Máquinas

(RAM,PRAM,VRAM)

Lenguajes

UN LENGUAJE

- Constantes: 0 , 1 , 2 , ... True , False , []
- Operadores:
+, -, *, /, <, >, &&, ||,
if_then_else, ▷
- Pares ordinarios y paralelos
(3 , 4) (3+4 | | 5+6)
- Expresiones let (nombres locales)
let x = 3+4 in x+x
- Secuencias
[x*2 | x ← xs]

TRABAJO

$$W(c) = 1$$

$$W(op\ e) = 1 + W(e)$$

$$W(e_1, e_2) = 1 + W(e_1) + W(e_2)$$

$$W(e_1 \parallel e_2) = 1 + W(e_1) + W(e_2)$$

$$W(\text{let } x = e_1 \text{ in } e_2) = 1 + W(e_1) + W(e_2(x \mapsto \text{Eval}(e_1)))$$

$$W([f(x) \mid x \leftarrow xs]) = 1 + \sum_{x \in xs} W(f(x))$$

PROFUNDIDAD

$$S(c) = 1$$

$$S(op\ e) = 1 + S(e)$$

$$S(e_1, e_2) = 1 + S(e_1) + S(e_2)$$

$$S(e_1 \parallel e_2) = 1 + \max(S(e_1), S(e_2))$$

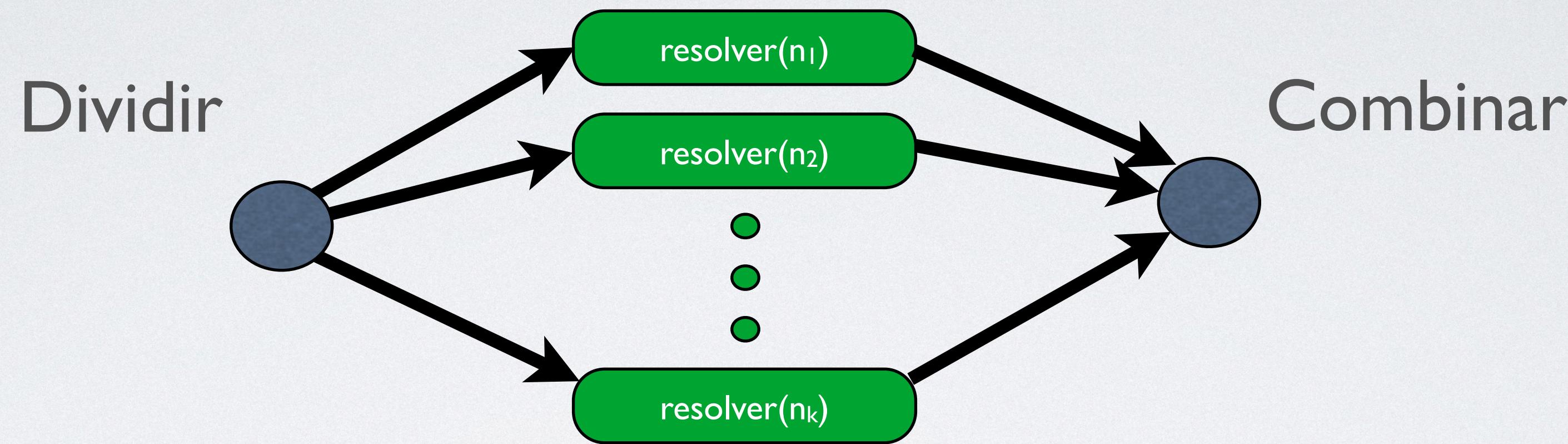
$$S(\text{let } x = e_1 \text{ in } e_2) = 1 + S(e_1) + S(e_2(x \mapsto \text{Eval}(e_1)))$$

$$S([f(x) \mid x \leftarrow xs]) = 1 + \max_{x \in xs} S(f(x))$$

DIVIDE & CONQUER

- Caso Base:
 - Problema chico, resolver directamente
- Caso Recursivo:
 1. dividir el problema en subproblemas
 2. resolver cada subproblema recursivamente
 3. combinar las soluciones en una solución al problema general

DIVIDE & CONQUER

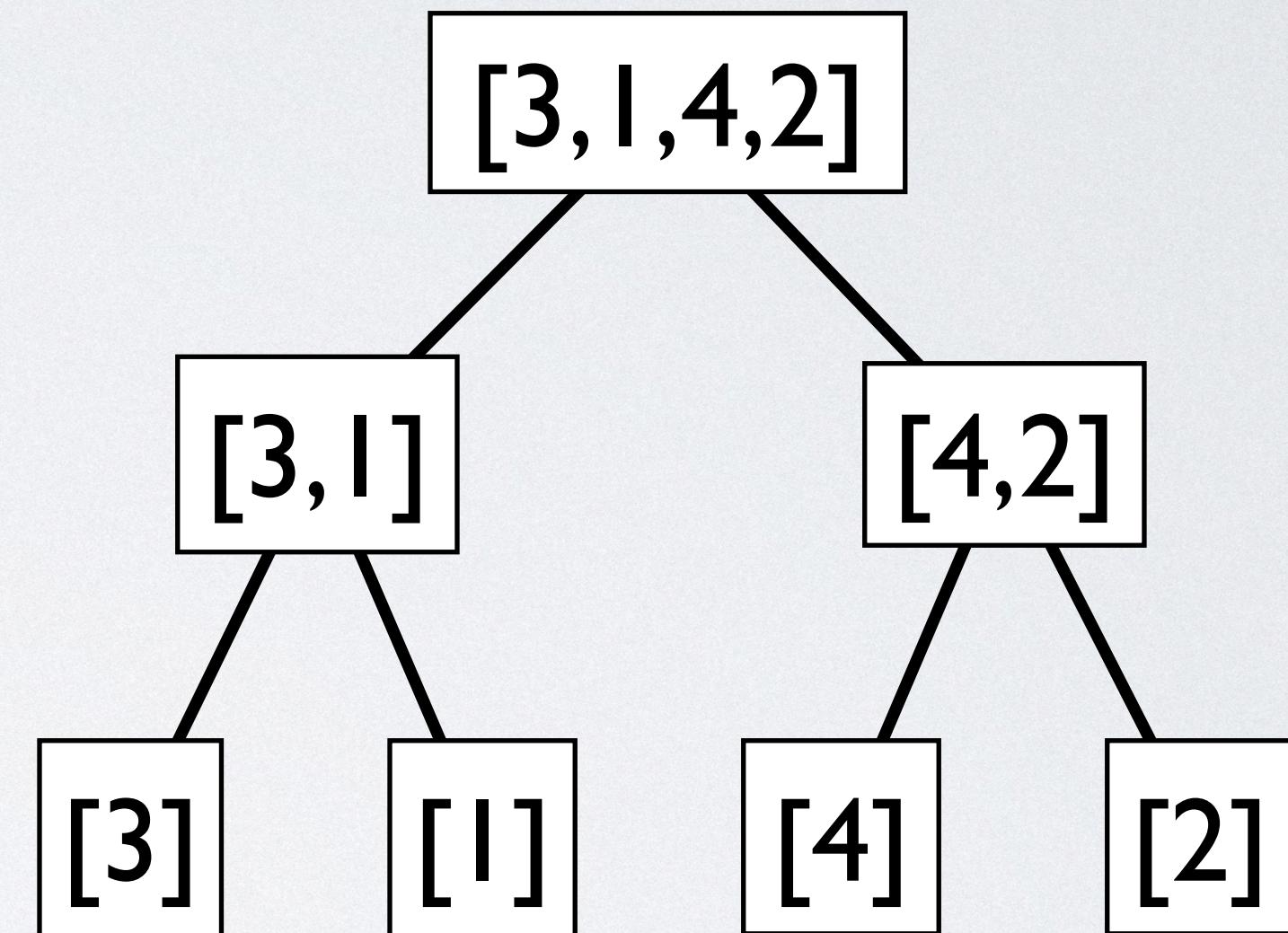


$$W(n) = W_{dividir}(n) + \sum_{i=1}^k W(n_i) + W_{combinar}(n)$$

$$S(n) = S_{dividir}(n) + \max_{i=1}^k S(n_i) + S_{combinar}(n)$$

MERGESORT

- Dividir la lista en 2 sublistas
- Ordenarlas recursivamente
- Juntar los resultados



MERGESORT

```
msort      : [ Int ] → [ Int ]
msort [ ] = [ ]
msort [ x ]= [ x ]
msort xs = let
            (ls ,rs ) = split xs
            (ls' ,rs' ) = msort ls || msort rs
        in
            merge (ls' ,rs' )
```

MERGESORT

```
split          : [Int] → [Int]×[Int]
split []       = ([],[])
split [x]       = ([x],[])
split (x>y>zs) = let
                    (xs,ys) = split zs
                    in
                    (x>xs,y>ys)
```

MERGESORT

```
merge          :: [Int]×[Int] → [Int]
merge ( [ ] , ys ) = ys
merge ( xs , [ ] ) = xs
merge ( x▷xs , y▷ys ) = if x <= y
                           then x▷merge(xs , y▷ys)
                           else y▷merge(x▷xs , ys )
```

MERGESORT: TRABAJO

```
msort      : [Int] → [Int]
msort [] = []
msort [x] = [x]
msort xs = let
            (ls ,rs ) = split xs
            (ls',rs') = msort ls || msort rs
        in
            merge (ls',rs')
```

$$W_{msort}(0) = c_0$$

$$W_{msort}(1) = c_1$$

$$W_{msort}(n) = W_{split}(n) + 2W_{msort}(n/2) + W_{merge}(n) + c_2 \quad \text{si } n > 1$$

MERGESORT :TRABAJO

```
split          : [ Int ] → [ Int ] × [ Int ]
split []       = ( [ ], [ ] )
split [ x ]    = ( [ x ], [ ] )
split ( x ▷ y ▷ z s ) = let
                           ( xs , ys ) = split z s
                           in
                           ( x ▷ xs , y ▷ ys )
```

$$W_{\text{split}}(0) = c_3$$

$$W_{\text{split}}(1) = c_4$$

$$W_{\text{split}}(n) = W_{\text{split}}(n - 2) + c_5 \quad \text{si } n > 1$$

$O(n)$

MERGESORT: TRABAJO

```
merge : [Int]×[Int] → [Int]
merge ( [], ys ) = ys
merge ( xs, [] ) = xs
merge ( xs, ys ) = if x <= y
                    then xs ∘ merge ( xs, ys )
                    else ys ∘ merge ( xs, ys )
```

$$W_{merge}(0) = c_6$$

$$W_{merge}(n) = W_{merge}(n - 1) + c_7$$

$O(n)$

n es la suma de las longitudes de las listas argumento

MERGESORT: TRABAJO

$$W_{msort}(0) = c_0$$

$$W_{msort}(1) = c_1$$

$$W_{msort}(n) = W_{split}(n) + 2W_{msort}(n/2) + W_{merge}(n) + c_2 \quad \text{si } n > 1$$

$$W_{msort}(n) = 2W_{msort}(n/2) + c_3 n \quad \text{si } n > 1$$

$O(n \lg n)$

MERGESORT: PROFUNDIDAD

```
msort      : [ Int ] → [ Int ]
msort [ ] = [ ]
msort [ x ]= [ x ]
msort xs = let
            ( ls , rs ) = split xs
            ( ls' , rs' ) = msort ls || msort rs
        in
            merge ( ls' , rs' )
```

$$S_{msort}(0) = k_0$$

$$S_{msort}(1) = k_1$$

$$S_{msort}(n) = S_{split}(n) + S_{msort}(n/2) + S_{merge}(n) + k_2 \quad \text{si } n > 1$$

MERGESORT: PROFUNDIDAD

```
split          : [Int] → [Int]×[Int]
split []       = ([],[])
split [x]      = ([x],[])
split (x▷y▷zs) = let
                  (xs,ys) = split zs
                in
                  (x▷xs,y▷ys)
```

$$S_{split}(0) = k_3$$

$$S_{split}(1) = k_4$$

$$S_{split}(n) = S_{split}(n - 2) + k_5 \quad \text{si } n > 1$$

$O(n)$

MERGESORT: PROFUNDIDAD

```
merge : [Int]×[Int] → [Int]
merge ( [], ys ) = ys
merge ( xs, [] ) = xs
merge ( xs, ys ) = if x <= y
                    then xs ∘ merge ( xs, ys )
                    else ys ∘ merge ( xs, ys )
```

$$S_{merge}(0) = k_6$$

$$S_{merge}(n) = S_{merge}(n - 1) + k_7$$

$O(n)$

n es la suma de las longitudes de las listas argumento

MERGESORT: PROFUNDIDAD

$$S_{msort}(0) = k_0$$

$$S_{msort}(1) = k_1$$

$$S_{msort}(n) = S_{split}(n) + S_{msort}(n/2) + S_{merge}(n) + k_2 \quad \text{si } n > 1$$

$$S_{msort}(n) = S_{msort}(n/2) + k_3 n$$

$O(n)$

RESUMEN

- Definimos el costo directamente sobre el lenguaje
- El trabajo W nos da todo el trabajo a realizar
- La profundidad S nos da las dependencias entre computaciones
- Al intentar calcular W y S surgen recurrencias que hay que resolver