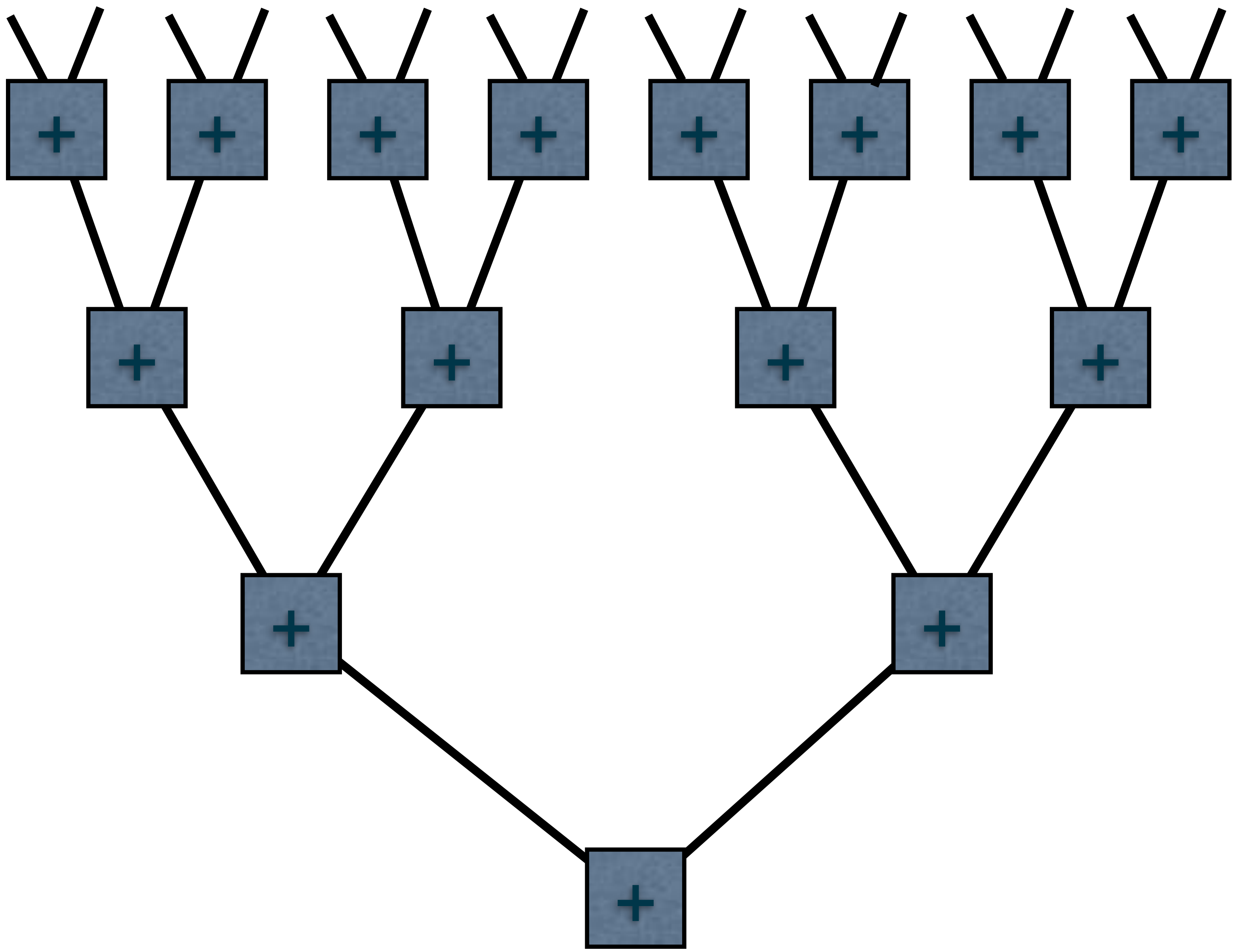
Modelo de Costo

EDyAII 2014

Modelo de Costo

- Costo secuencial: Trabajo (W)
  - costo con 1 procesador
- Costo paralelo: Profundidad (S)
  - costo con ∞ procesadores

Ejemplo: suma de n números



$W \in O(n)$

$S \in O(\lg n)$

Paralelismo

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

- Indica cuantos procesadores podemos usar eficientemente
- Sumar *n* números es $P \leq \frac{kn}{k' \lg n} \in O(\frac{n}{\lg n})$

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

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

- Si $p \ll P$ la cota es prácticamente óptima

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

Trabajo

$$
\begin{aligned}
W(c) &= 1 \\
W(op\ e) &= 1 + W(e) \\
W(e_1, e_2) &= 1 + W(e_1) + W(e_2) \\
W(e_1 || e_2) &= 1 + W(e_1) + W(e_2)) \\
W(\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2) &= 1 + W(e_1) + W(e_2[Eval(e_1)/x]) \\
W(\{f(x) : x \in A\}) &= 1 + \sum_{x \in A} W(f(x))
\end{aligned}
$$

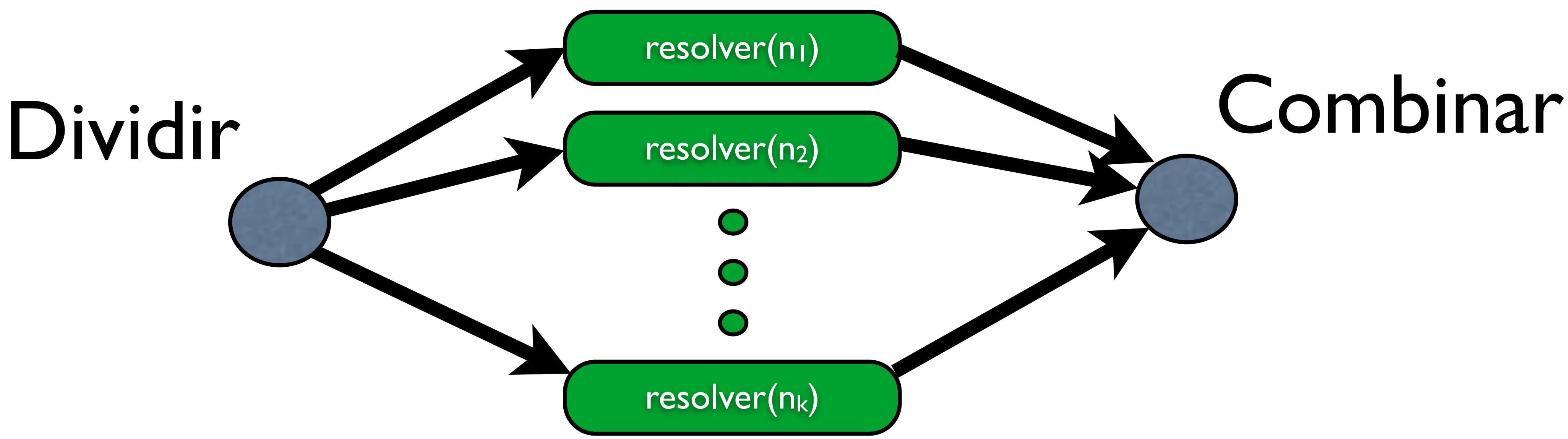
Profundidad

$$S(c) = 1$$
$$S(op\ e) = 1 + S(e)$$
$$S(e_1, e_2) = 1 + S(e_1) + S(e_2)$$
$$S(e_1 || e_2) = 1 + \max(S(e_1), S(e_2))$$
$$S(\textsf{let } x = e_1 \textsf{ in } e_2) = 1 + S(e_1) + S(e_2[Eval(e_1)/x])$$
$$S(\{f(x) : x \in A\}) = 1 + \max_{x \in A} S(f(x))$$

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

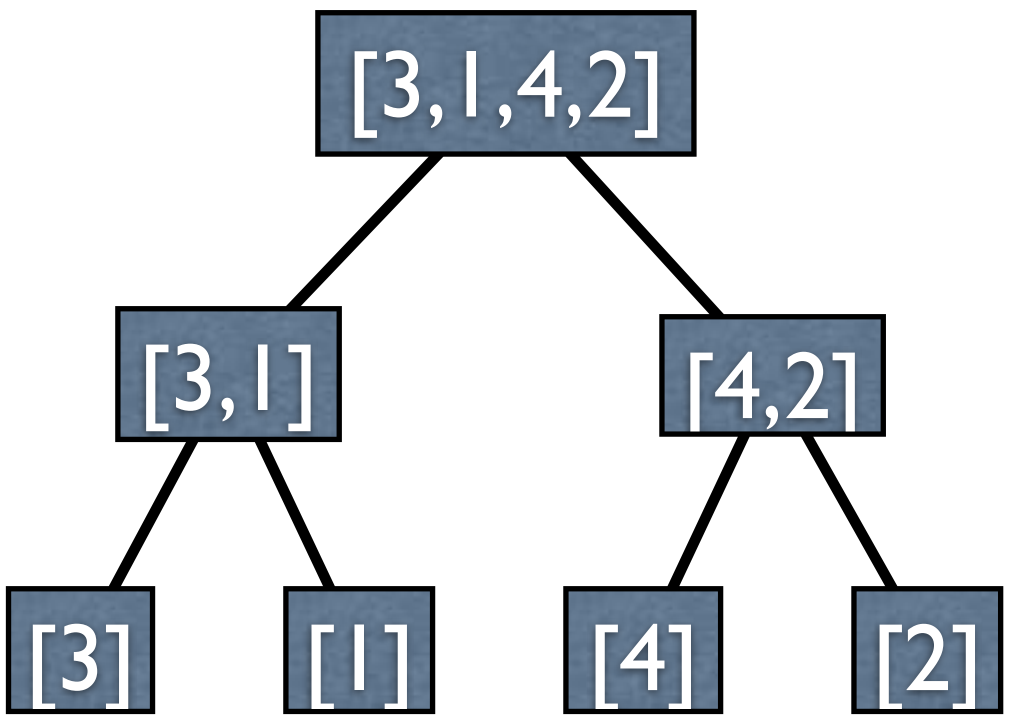
Divide & Conquer



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

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

Mergesort



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

Mergesort

$$
\begin{array}{lcl}
msort & : & [Int] \to [Int] \\
msort\ [] & = & [] \\
msort\ [x] & = & [x] \\
msort\ xs & = & \textsf{let}\ \ (ls, rs) \quad = split\ xs \\
 & & \qquad\ (ls', rs') = (msort\ ls \mid\mid msort\ rs) \\
 & & \textsf{in}\ merge(ls', rs')
\end{array}
$$

Mergesort

$$
\begin{array}{lcl}
split & : & [Int] \to [Int] \times [Int] \\
split\ [] & = & ([], []) \\
split\ [x] & = & ([x], []) \\
split\ (x \triangleleft y \triangleleft zs) & = & \mathsf{let}\ (xs, ys) = split\ zs \\
 & & \mathsf{in}\ (x \triangleleft xs, y \triangleleft ys)
\end{array}
$$

Mergesort

$$
\begin{array}{lcl}
merge & : & [Int] \times [Int] \to [Int] \\
merge\ ([], ys) & = & ys \\
merge\ (xs, []) & = & xs \\
merge\ (x \triangleleft xs, y \triangleleft ys) & = & \text{if } x \leq y \\
 & & \quad \text{then } x \triangleleft merge(xs, y \triangleleft ys) \\
 & & \quad \text{else } y \triangleleft merge(x \triangleleft xs, ys)
\end{array}
$$

Mergesort: Trabajo

$$
\begin{array}{lcl}
msort & : & [Int] \to [Int] \\
msort\ [] & = & [] \\
msort\ [x] & = & [x] \\
msort\ xs & = & \text{let } \begin{array}{l} (ls, rs) \quad = split\ xs \\ (ls', rs') = (msort\ ls \mid\mid msort\ rs) \end{array} \\
 & & \text{in } merge(ls', rs')
\end{array}
$$

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

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

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

Mergesort : Trabajo

$$
\begin{array}{lcl}
split & : & [Int] \to [Int] \times [Int] \\
split\ [] & = & ([], []) \\
split\ [x] & = & ([x], []) \\
split\ (x \triangleleft y \triangleleft zs) & = & \textsf{let}\ (xs, ys) = split\ zs \\
 & & \textsf{in}\ (x \triangleleft xs, y \triangleleft ys)
\end{array}
$$

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

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

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

$O(n)$

Mergesort: Trabajo

$$
\begin{array}{lcl}
merge & : & [Int] \times [Int] \rightarrow [Int] \\
merge\ ([\,], ys) & = & ys \\
merge\ (xs, [\,]) & = & xs \\
merge\ (x \triangleleft xs, y \triangleleft ys) & = & \text{if } x \leq y \\
 & & \quad \text{then } x \triangleleft merge(xs, y \triangleleft ys) \\
 & & \quad \text{else } y \triangleleft merge(x \triangleleft xs, ys)
\end{array}
$$

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

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

$$O(n)$$

n es la suma de las longitudes de las listas argumento

Mergesort: Trabajo

$$W_{msort}(0) = c_0$$
$$W_{msort}(1) = c_1$$
$$W_{msort}(n) = W_{split}(n) + 2W_{msort}(\frac{n}{2}) + W_{merge}(n) + c_2 \quad \text{si } n > 1$$

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

$$O(n \lg n)$$

Mergesort: Profundidad

$$
\begin{array}{lcl}
msort & : & [Int] \rightarrow [Int] \\
msort\ [] & = & [] \\
msort\ [x] & = & [x] \\
msort\ xs & = & \text{let } \begin{array}{l} (ls, rs) \quad = split\ xs \\ (ls', rs') = (msort\ ls\ ||\ msort\ rs) \end{array} \\
 & & \text{in } merge(ls', rs')
\end{array}
$$

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

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

$$S_{msort}(n) = S_{split}(n) + S_{msort}(\frac{n}{2}) + S_{merge}(n) + k_3$$

Mergesort: Profundidad

$$
\begin{array}{lcl}
split & : & [Int] \to [Int] \times [Int] \\
split\ [] & = & ([], []) \\
split\ [x] & = & ([x], []) \\
split\ (x \triangleleft y \triangleleft zs) & = & \mathsf{let}\ (xs, ys) = split\ zs \\
 & & \mathsf{in}\ (x \triangleleft xs, y \triangleleft ys)
\end{array}
$$

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

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

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

$O(n)$

Mergesort: Profundidad

$$
\begin{array}{lcl}
merge & : & [Int] \times [Int] \rightarrow [Int] \\
merge\ ([], ys) & = & ys \\
merge\ (xs, []) & = & xs \\
merge\ (x \triangleleft xs, y \triangleleft ys) & = & \text{if } x \leq y \\
 & & \quad \text{then } x \triangleleft merge(xs, y \triangleleft ys) \\
 & & \quad \text{else } y \triangleleft merge(x \triangleleft xs, ys)
\end{array}
$$

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

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

$O(n)$

Mergesort: Profundidad

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

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

$$S_{msort}(n) = S_{split}(n) + S_{msort}(\frac{n}{2}) + S_{merge}(n) + k_3$$

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

$$O(n)$$

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