Tema V: Algoritmos Paralelos de Ordenación

Problema de ordenación: *Dada una lista numérica, $A = (a_1, \ldots, a_n)$, hallar una permutación, π, de $\{1, \ldots, n\}$, tal que $a_{\pi(1)} \leqslant a_{\pi(2)} \leqslant \ldots \leqslant a_{\pi(n)}$.*

Algoritmos basados en comparaciones.

Coste en tiempo de algoritmos secuenciales que resuelven el problema: $\theta(n \cdot \lg n)$ cota inferior en los casos peor y promedio.

	Caso promedio	Caso peor
Inserción	$\theta(n^2)$	$\theta(n^2)$
Quicksort	$\theta(n \cdot \lg n)$	$\theta(n^2)$
Heapsort	$\theta(n \cdot \lg n)$	$\theta(n \cdot \lg n)$
Mergesort	$\theta(n \cdot \lg n)$	$\theta(n \cdot \lg n)$

Algoritmos paralelos de *ordenación interna*: los datos se encuentran en la memoria de la máquina.

El problema de fusión (mezcla)

Problema de fusión (mezcla): *Dadas dos listas numéricas ordenadas, $A = (a_1, \ldots, a_r)$ y $B = b_1, \ldots, b_s)$, hallar una lista ordenada, $C = (c_1, \ldots, c_{r+s})$, cuyos elementos son los de $A \cup B$, como "multiconjuntos".*

Ordenación por fusión de dos listas numéricas ordenadas.

- Supongamos que $r \leqslant s$.
- Cota inferior en el caso peor: $\theta(\text{máx}\{r, s\})$.

Estudio algorímico del problema.

- Solución secuencial óptima.
- Soluciones paralelas.
  - **SIMD** con una red de interconexión.
  - **SIMD** con memoria compartida.

Algoritmo secuencial óptimo

procedimiento fusion $((a_1, \ldots, a_r), (b_1, \ldots, b_s), (c_1, \ldots, c_{r+s}))$

$i \leftarrow 1$; $j \leftarrow 1$; $k \leftarrow 1$

mientras $i \leqslant r$ y $j \leqslant s$ hacer

si $a_i < b_j$ entonces

$c_k \leftarrow a_i$; $k \leftarrow k+1$; $i \leftarrow i+1$

si no

$c_k \leftarrow b_j$; $k \leftarrow k+1$; $j \leftarrow j+1$

si $i > r$ entonces

para $l \leftarrow 0$ hasta $s - j$ hacer

$c_{k+l} \leftarrow b_{j+l}$

si no

para $l \leftarrow 0$ hasta $r - i$ hacer

$c_{k+l} \leftarrow a_{i+l}$

Idea del algoritmo:

- i, j: posición del menor elemento de A y B (resp) no colocado aún en C.
- k: representa la posición de C que va a ser ocupada.
- Tras la ejecución del bucle *mientras*, el valor de i (resp. j) representa al siguiente elemento de A (resp. B), si existe, aún no colocado en C.

Coste en el caso peor: $\theta(\text{máx}\{r, s\})$.

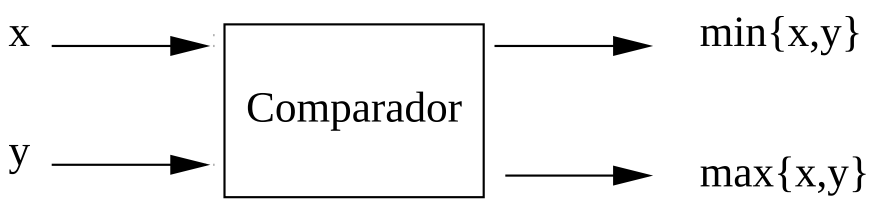
Algoritmo SIMD con una red de interconexión

Se describe un algoritmo SIMD con una red de interconexión que resuelve el problema de fusión de dos listas ordenadas.

Supongamos que $r = s = n$ es potencia de 2.

Arquitectura paralela: conjunto de procesadores idénticos (*comparadores*) comunicados entre sí a través de una red (*red de fusión*).

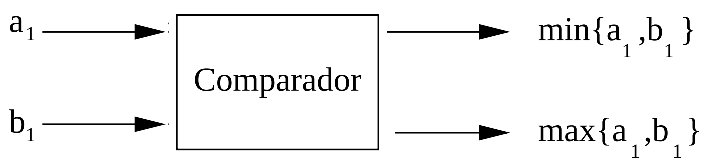
Definición: Un *comparador* es un dispositivo que admite dos datos de entrada y proporciona dos datos de salida: el menor sale por la parte superior y el mayor sale por la inferior.



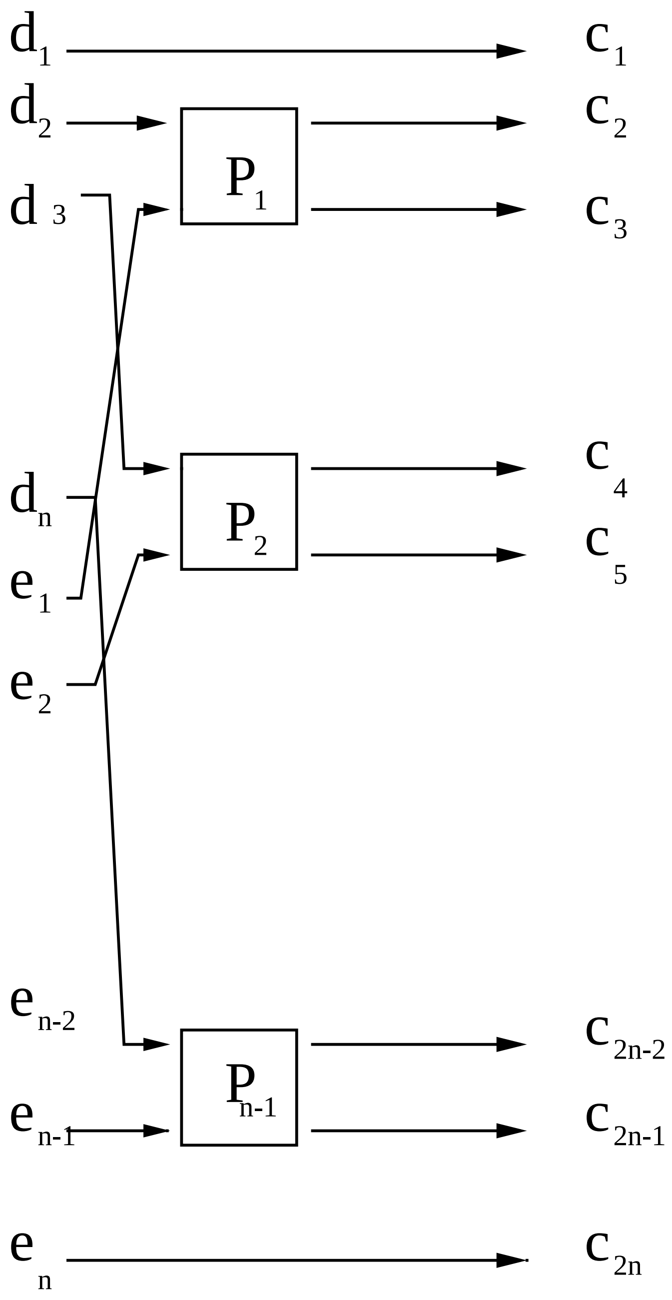
Los comparadores son los elementos básicos de la red (red de fusión), que se define recursivamente.

Definición: Las *redes de fusión de orden* n se definen recursivamente como sigue:

- Las redes de fusión de orden 1 procesan dos listas ordenadas de tamaño 1, $A = (a_1)$ y $B = (b_1)$, a través de un comparador.



- Las redes de fusión de orden n ($n > 1$, n potencia de 2) procesan dos listas ordenadas de tamaño n, $A = (a_1, \ldots, a_n)$ y $B = (b_1, \ldots, b_n)$ así:
  - ⋆ Se procesan recursivamente las listas ordenadas $A_I = (a_1, a_3, \ldots, a_{n-1})$ y $B_I = (b_1, b_3, \ldots, b_{n-1})$ mediante una red de fusión de orden $\frac{n}{2}$ (notemos $C_I = (d_1, d_2, \ldots, d_n)$ la correspondiente salida).
  - ⋆ Se procesan recursivamente las listas ordenadas $A_P = (a_2, a_4, \ldots, a_n)$ y $B_P = (b_2, b_4, \ldots, b_n)$ mediante una red de fusión de orden $\frac{n}{2}$ (notemos $C_P = (e_1, e_2, \ldots, e_n)$ la correspondiente salida).
  - ⋆ Se procesan las listas ordenadas C_I y C_P mediante $n - 1$ comparadores así:



Es decir

$$
\begin{cases}
c_1 & = & d_1 \\
c_{2n} & = & e_n \\
c_{2i} & = & \text{mín}\{d_{i+1}, e_i\}\ (i = 1, \dots, n-1) \\
c_{2i+1} & = & \text{máx}\{d_{i+1}, e_i\}\ (i = 1, \dots, n-1)
\end{cases}
$$

Las redes de fusión proporcionan un algoritmo **SIMD** para ordenar por fusión dos listas ordenadas.

Teorema: *Una red de fusión de orden $n = 2^r$ ordena por fusión dos listas ordenadas de tamaño n.*

Demostración: Por inducción fuerte sobre n.

Para $n = 1$ el resultado es inmediato. Sea $n > 1$ (n potencia de 2) y supongamos cierto el resultado $\forall r \geq 1$, $r < n$, r potencia de 2.

Sean A y B listas ordenadas de tamaño n. Una red de fusión de orden n trabaja con A y B así:

- Procesa A_I y B_I mediante una red de fusión de orden $\frac{n}{2}$. Por h.i. la lista resultante C_I está ordenada.

- Procesa A_P y B_P mediante una red de fusión de orden $\frac{n}{2}$. La lista resultante C_P está ordenada.

- Con $n - 1$ comparadores se procesan C_I y C_P, obteniendo $c_1 = d_1$, $c_{2n} = e_n$ y

  $$c_{2i} = \text{mín}\{d_{i+1}, e_i\} \ ; \ c_{2i+1} = \text{máx}\{d_{i+1}, e_i\}$$

  para cada $i = 1, \ldots, n - 1$.

  ⋆ **Lema:** *Si $c_1^* \leq c_2^* \leq \ldots \leq c_{2n-1}^* \leq c_{2n}^*$ es la lista fusión de A y B, entonces $c_i^* = c_i$.*
  Por inducción fuerte acotada sobre i.

■

Análisis del algoritmo

- *Tiempo de ejecución.* Si $F(2n)$ es el tiempo de ejecución necesario para ordenar dos listas de tamaño n, entonces:

$$F(2n) = \begin{cases} 1 & \text{si} \quad n = 1 \\ F(n) + 1 & \text{si} \quad n > 1 \ (n = 2^r) \end{cases}$$

De donde $F(2n) = 1 + \lg n \in \theta(\lg n)$.

- *Número de procesadores.* Si $k(2n)$ es el número de procesadores usados para ordenar dos listas de tamaño n, entonces:

$$k(2n) = \begin{cases} 1 & \text{si} \quad n = 1 \\ 2k(n) + n - 1 & \text{si} \quad n > 1 \ (n = 2^r) \end{cases}$$

De donde $k(2n) = 1 + n\lg n \in \theta(n\lg n)$.

- *Aceleración, coste y eficiencia.*

$$S(2n, k(2n)) \in \theta(\frac{n}{\lg n})$$

$$C(2n, k(2n)) \in \theta(n\lg^2 n)$$

$$E(2n, k(2n)) \in \theta(\frac{1}{\lg^2 n})$$

Consideraciones: Las redes de fusión de orden n

- Son *independientes* del dato de entrada (número constante de comparaciones).

- Mejoran considerablemente el tiempo de ejecución de cualquier algoritmo secuencial.

- No resuelven de manera eficiente el problema de fusión de dos listas ordenadas.

- Son *poco flexibles* (sólo sirven para listas de tamaño el orden de la red).

- No son prácticas para valores elevados de n.

Algoritmo SIMD-CREW

Diseñaremos un algorimo **SIMD–CREW** que resuelve el problema de fusión de dos listas numéricas <u>ordenadas</u> $A = (a_1, \dots, a_r)$ y $B = (b_1, \dots, b_s)$.

Supondremos que $r = s = n$ y k indicará el numero de procesadores usados en el algoritmo.

Convendremos en que si se compara un elemento de A y otro de B que son iguales, entonces decide que el de A es "menor".

El algoritmo diseñado será:

- *Flexible*.
- *Optimo*, para un número adecuado de procesadores.

Cada procesador tendrá asignada tareas de fusión secuencial y de búsqueda binaria.

En la memoria compartida existirán, básicamente, tres zonas asociadas, respectivamente, a las listas de entrada A, B (de longitud n), y a la lista de salida C (de longitud $2n$).

El algoritmo se describirá explicitando tres pasos básicos.

Descripción de un algoritmo SIMD-CREW

Entrada: $A = (a_1, \ldots, a_n)$ y $B = (b_1, \ldots, b_n)$ listas numéricas ordenadas.

Paso 1: *Se dividen A y B en k sublistas A', B' del "mismo tamaño", seleccionando $k-1$ elementos:*

para cada P_i tal que $1 \leqslant i \leqslant k-1$ hacer

$a'_i \leftarrow a_{i \cdot \lceil \frac{n}{k} \rceil}$

$b'_i \leftarrow b_{i \cdot \lceil \frac{n}{k} \rceil}$

Paso 2: *A partir de $A' = (a'_1, \ldots, a'_{k-1})$ y $B' = (b'_1, \ldots, b'_{k-1})$ se construye $C' = (c'_1, \ldots, c'_{2k-2})$ tal que c'_i es una terna del tipo (a'_t, t, A) ó (b'_t, t, B), y cuyas primeras componentes es una lista ordenada.*

para cada P_i tal que $1 \leqslant i \leqslant k-1$ hacer

P_i busca en B' el menor j tal que $b'_j \geqslant a'_i$

si j existe entonces

$c'_{i+j-1} \leftarrow (a'_i, i, A)$

si no

$c'_{i+k-1} \leftarrow (a'_i, i, A)$

para cada P_i tal que $1 \leqslant i \leqslant k-1$ hacer

P_i busca en A' el menor j tal que $a'_j > b'_i$

si j existe entonces

$c'_{i+j-1} \leftarrow (b'_i, i, B)$

si no

$c'_{i+k-1} \leftarrow (b'_i, i, B)$

Paso 3: *Cada procesador P_i fusiona elementos de una sublista de A y otra de B menores que $(c'_{2i})_1$ (primera componente de la terna, si $i \leqslant k-1$), colocándolos en C: los índices para iniciar la tarea se le proporciona mediante un vector $V(i) = (\alpha_i, \beta_i)$.*

$V(1) \leftarrow (1,1)$

para cada P_i tal que $2 \leqslant i \leqslant k$ hacer

si $c'_{2i-2} = (a'_t, t, A)$ entonces

P_i busca en B el menor j tal que $b_j \geqslant a'_t$

$V(i) \leftarrow (t \cdot \lceil \frac{n}{k} \rceil, j)$

si $c'_{2i-2} = (b'_t, t, B)$ entonces

P_i busca en A el menor j tal que $a_j \geqslant b'_t$

$V(i) \leftarrow (j, t \cdot \lceil \frac{n}{k} \rceil)$

para cada P_i tal que $1 \leqslant i \leqslant k$ hacer

P_i opera con el vector $V(i) = (\alpha_i, \beta_i)$ así:

fusiona la sublista de A que comienza por a_{α_i} y la sublista de B que comienza por b_{β_i}, colocando el resultado en C, a partir de la posición $\alpha_i + \beta_i - 1$; la fusión termina cuando

- O bien no quedan elementos de A ó B
- O bien $i \leqslant k-1$ y encuentran un elemento de A ó B que es $\geqslant$ que la primera componente de c'_{2i}

Ejemplo de ilustración

Sean $A = \{2, 5, 6, 10, 11, 19, 24, 25, 26, 28, 30, 40\}$ y $B = \{1, 6, 12, 13, 17, 18, 20, 21, 25, 27, 29, 32\}$.

En este ejemplo $r = s = n = 12$.

Usemos $k = \frac{n}{\lg n} = 4$ procesadores.

- Paso 1: Dividimos A y B en 4 sublistas de igual tamaño: $A' = \{6, 19, 26\}$ y $B' = \{12, 18, 25\}$.

- Paso 2: "Fusionamos" las listas A' y B' en ternas ordenadas:

$$C' = \{(6, 1, A), (12, 1, B), (18, 2, B), (19, 2, A), (25, 3, B), (26, 3, A)\}$$

- Paso 3: Hallemos el vector $V(i)$ sobre el que opera el procesador P_i.

$V(1) = (1, 1)$

$V(2) = (6, 3)$ ya que $(c'_2)_1 = 12 = b'_1 = b_3$ y $6 = \min\{j : a_j \geqslant b_3 = 12\}$

$V(3) = (6, 7)$ ya que $(c'_4)_1 = 19 = a'_2 = a_6$ y $7 = \min\{j : b_j \geqslant a_6 = 19\}$

$V(4) = (9, 10)$ ya que $(c'_6)_1 = 26 = a'_3 = a_9$ y $10 = \min\{j : b_j \geqslant a_9 = 26\}$

P_1 fusiona $\{\mathbf{2}, 5, \underline{6}, 10, 11\} * \{\mathbf{1}, \underline{\underline{6}}\} < (c'_2)_1 = 12$ ($a_{\alpha_1} = a_1 = 2$ y $b_{\beta_1} = b_1 = 1$) obteniendo la sublista $\{1, 2, 5, \underline{6}, \underline{\underline{6}}, 10, 11\}$ de C, que coloca a partir de la posición $\alpha_1 + \beta_1 - 1 = 1$.

P_2 fusiona $\emptyset * \{\mathbf{12}, 13, 17, 18\} < (c'_4)_1 = 19$ ($a_{\alpha_2} = a_6 = 19$ y $b_{\beta_2} = b_3 = 12$) obteniendo la sublista $\{12, 13, 17, 18\}$ de C, que coloca a partir de la posición $\alpha_2 + \beta_2 - 1 = 8$.

P_3 fusiona $\{\mathbf{19}, 24, \underline{25}\} * \{\mathbf{20}, 21, \underline{\underline{25}}\} < (c'_6)_1 = 26$ ($a_{\alpha_3} = a_6 = 19$ y $b_{\beta_3} = b_7 = 20$) obteniendo la sublista $\{19, 20, 21, 24, \underline{25}, \underline{\underline{25}}\}$ de C, que coloca a partir de la posición $\alpha_3 + \beta_3 - 1 = 12$.

P_4 fusiona $\{\mathbf{26}, 28, 30, 40\} * \{\mathbf{27}, 29, 32\}$ ($a_{\alpha_4} = a_9 = 26$ y $b_{\beta_4} = b_{10} = 27$) obteniendo la sublista $\{26, 27, 28, 29, 30, 32, 40\}$ de C, que coloca a partir de la posición $\alpha_4 + \beta_4 - 1 = 18$.

De esta manera se obtiene la lista C siguiente

$$\{1, 2, 5, \underline{6}, \underline{\underline{6}}, 10, 11, 12, 13, 17, 18, 19, 20, 21, 24, \underline{25}, \underline{\underline{25}}, 26, 27, 28, 29, 30, 32, 40\}.$$

Correción del algoritmo

Notación: $\{x_p, \ldots, x_q\} * \{y_r, \ldots, y_s\} < z$ indicará que se ha fusionado la lista $\{x_p, \ldots, x_q\}$ con la lista $\{y_r, \ldots, y_s\}$, de tal manera que $x_i < z \wedge y_j < z$ $(p \leqslant i \leqslant q \wedge r \leqslant j \leqslant s)$.

Asímismo designaremos $(c'_i)_1$ la primera componente de la terna c'_i.

Supongamos que en la ejecución del paso 3 cada procesador P_i realiza la siguiente operación

- $\{a_{p_i}, \ldots, a_{q_i}\} * \{b_{r_i}, \ldots, b_{s_i}\} < (c'_{2i})_1$, si $i \leqslant k-1$.
- $\{a_{p_k}, \ldots, a_n\} * \{b_{r_k}, \ldots, b_n\}$, si $i = k$.

Proposición: *Cada P_i ha escrito en posiciones distintas de C y, además, entre $P_1, \ldots, P_k$ han escrito <u>todas</u> las posiciones de C .*

Demostración: Hay que probar que

$$\begin{cases} p_1 = q_1 = 1 \\ p_i = q_{i-1} + 1 \ \wedge \ r_i = s_{i-1} + 1 \quad (2 \leqslant i \leqslant k) \end{cases}$$

Caso 1: Supongamos que $i = 1$.

- P_i opera con $V(1) = (1, 1)$. Luego $p_1 = r_1 = 1$.
  - Si $(c'_2)_1 = a'_t = a_{t \cdot \frac{n}{k}}$ entonces

    $q_1 = t \cdot \frac{n}{k} - 1$ y $s_1 = $máx$\{j : \ b_j < a_{t \cdot \frac{n}{k}}\}$
  - Si $(c'_2)_1 = b'_t = b_{t \cdot \frac{n}{k}}$ entonces

    $q_1 = $máx$\{j : \ a_j < b_{t \cdot \frac{n}{k}}\}$ y $s_1 = t \cdot \frac{n}{k} - 1$

Caso 2: Supongamos que $2 \leqslant i \leqslant k$.

- P_i opera con el vector $V(i) = (\alpha_i, \beta_i)$.
  - Si $(c'_{2i-2})_1 = a'_t = a_{t \cdot \frac{n}{k}}$ entonces

    $q_{i-1} = t \cdot \frac{n}{k} - 1$ y $s_{i-1} = $máx$\{j : \ b_j < a_{t \cdot \frac{n}{k}}\}$

    Por construcción de $V(i)$ se tiene que

    $p_i = t \cdot \frac{n}{k}$ y $r_i = $mín$\{j : \ b_j \geqslant a_{t \cdot \frac{n}{k}}\}$

    Luego, $p_i = q_{i-1} + 1$ y $r_i = s_{i-1} + 1$.
  - Si $(c'_{2i-2})_1 = b'_t = b_{t \cdot \frac{n}{k}}$ entonces

    $q_{i-1} = $máx$\{j : \ a_j < b_{t \cdot \frac{n}{k}}\}$ y $s_{i-1} = t \cdot \frac{n}{k} - 1$

    Por construcción de $V(i)$ se tiene que

    $p_i = $mín$\{j : \ a_j \geqslant b_{t \cdot \frac{n}{k}}\}$ y $r_i = t \cdot \frac{n}{k}$

    Luego, $p_i = q_{i-1} + 1$ y $r_i = s_{i-1} + 1$.

■

Análisis del algoritmo

- *Tiempo de ejecución en el caso peor.*
  - Paso 1: $\theta(1)$.
  - Paso 2: $\theta(\lg k)$.
  - Paso 3: $\theta(\lg n) + \theta(\frac{n}{k})$.

  Por tanto: $t(2n, k) \in \theta(\lg n + \frac{n}{k})$.

- *Número de procesadores.* $k(2n) \leqslant n$,

- *Aceleración, coste y eficiencia.*

$$S(2n, k(2n)) \in \theta(\frac{n}{\lg n + \frac{n}{k}})$$

$$C(2n, k(2n)) \in \theta(k \cdot (\lg n + \frac{n}{k}))$$

$$E(2n, k(2n)) \in \theta(\frac{n}{k \cdot (\lg n + \frac{n}{k})})$$

Si $k = \frac{n}{\lg n}$, entonces $E(2n, k(2n)) \in \theta(1)$.

Por tanto, el algoritmo descrito será **óptimo**.

Algoritmo SIMD-EREW

Adaptación del algorimo **SIMD–CREW**.

Basta simular lectura concurrente en una máquina **EREW**, de manera eficiente.

Un algoritmo **CREW-PRAM**, usando $k = 2^r$ procesadores y M registros de memoria, se puede simular en una **EREW-PRAM**, usando k procesadores y $M \cdot (2k-1)$ registros de memoria, en tiempo $O(\lg k)$.

Análisis del algoritmo en el caso peor

- $t(2n, k(2n)) \in \theta(\lg k \cdot (\lg n + \frac{n}{k})) \subseteq O(\lg^2 n + \frac{n}{k} \cdot \lg n)$.
- $S(2n, k(2n)) \in O(\frac{kn}{n\lg n + k\lg^2 n})$
- $C(2n, k(2n)) \in O(n\lg n + k\lg^2 n))$
- $E(2n, k(2n)) \in O(\frac{n}{n\lg n + k\lg^2 n})$

En el caso $k = \frac{n}{\lg n}$ se tiene que $E(2n, k) \in O(\frac{1}{\lg n})$.

El algoritmo descrito será eficiente pero **no** óptimo.

Algoritmos de ordenación

Algoritmos secuenciales de ordenación

- Ordenación por fusión o mezcla (Mergesort).
- Ordenación rápida (Quicksort).

Algoritmos paralelos de ordenación

- **SIMD**.
  - Con memoria compartida.
  - Con red de interconexión.
    - Redes de fusión.
    - Trasposición par–impar e impar–par.
    - Mezcla bitónica.
- **MIMD**.
  - Quicksort **MIMD**.

Algoritmos secuenciales óptimos

Algoritmo secuencial de ordenación por fusión

```
procedimiento ord-fusion (a[1,...,n])
    si n ⩽ n_0 entonces ord-insercion (a[1,...,n])
    si no
      b[1,...,⌊n/2⌋] ← a[1,...,⌊n/2⌋]
      c[1+⌊n/2⌋,...,n] ← a[1+⌊n/2⌋,...,n]
      ord-fusion (b[1,...,⌊n/2⌋])
      ord-fusion (c[1+⌊n/2⌋,...,n])
      fusion (b[1,...,⌊n/2⌋], c[1+⌊n/2⌋,...,n])
```

Coste del orden $\theta(n\lg n)$ en los casos peor y promedio.

Algoritmo óptimo en los casos peor y promedio.

Algoritmo secuencial de ordenación rápida

Definición: *Un elemento a_i es un corte de $(a_1, \ldots, a_n)$ si $1 \leqslant j < i \Longrightarrow a_j \leqslant a_i$ e $i < j \leqslant n \Longrightarrow a_j \geqslant a_i$.*

Fabricación de un corte de $(a_1, \ldots, a_n)$ a partir de un elemento $p = a_i$ (**pivote**).

Sean k y l punteros inicializados en 1 y n.

- Primera fase:
  - Incrementamos k hasta que $a_k > p$ ó $k = n + 1$.
  - Disminuimos l hasta que $a_l \leqslant p$ ó $l = 0$.

  El proceso se reitera mientras $k < l$
- Segunda fase: Intercambiamos a_i y a_l.

```
procedimiento ord-rapida (a[r,...,s])
    i ← r;  j ← s
    si i ⩾ j entonces terminar
    si no
      Elegimos ''adecuadamente'' un elemento a_k
      corte (a[r,...,s], k, l)
      ord-rapida (a[r,...,l−1])
      ord-rapida (a[l+1,...,s])
```

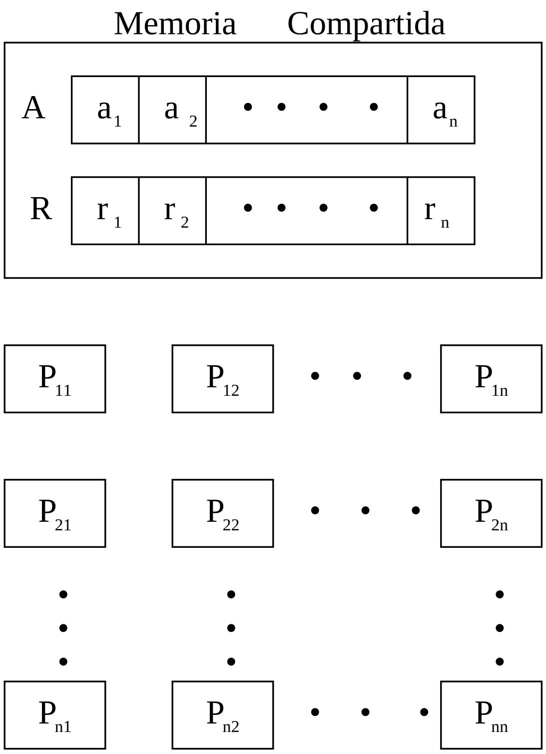
Coste del orden $\theta(n\lg n)$ en el caso promedio (óptimo).

Coste del orden $\theta(n^2)$ en en el caso peor.

Algoritmos paralelos de ordenación SIMD con memoria compartida

Algoritmo de ordenación SIMD-CRCW

- Ordenamos una lista $(a_1, \ldots, a_n)$ con una **PRAM–CRCW** con n^2 porcesadores, P_{ij}.
- Conflictos de escritura: si varios procesadores desean escribir en la misma dirección, se guardará la $suma$ de esos valores.
- Para cada i, se halla el número r_i de elementos menores que a_i (conservando la posición relativa los que son iguales). Se coloca a_i en la posición $1 + r_i$.
- Dos vectores $A[1, \ldots, n]$ y $R[1, \ldots, n]$ en la memoria compartida: A almacena la entrada, R almacena los valores r_i, y el dato de salida aparece en A.



Descripción del algoritmo:

procedimiento ordenacion$_{\text{CRCW}}$ $(A[1,\ldots,n])$
    vector $R[1,\ldots,n]$
    para cada P_{ij} tal que $1 \leqslant i,j \leqslant n$ hacer
      si $(a_i > a_j) \vee (a_i = a_j \wedge i > j)$ entonces
        P_{ij} escribe 1 en $R[i]$
      si no
        P_{ij} escribe 0 en $R[i]$
    para cada P_{i1} tal que $1 \leqslant i \leqslant n$ hacer
      P_{i1} escribe a_i en la posicion $1 + R[i]$ de A

Análisis del algoritmo SIMD-CRCW

- $t(n,n^2) \in \theta(1)$.
- $S(n,n^2) \in \theta(n\lg n)$.
- $C(n,n^2) \in \theta(n^2)$.
- $E(n,n^2) \in \theta(\frac{\lg n}{n})$.

El algoritmo **no** es óptimo ni eficiente.

Ordenación SIMD-CREW

Nos basamos en el algoritmo paralelo de fusión **CREW**.

Idea del algoritmo:

- Entrada $A = (a_1, \ldots, a_n)$; se usarán k procesadores, P_i (n múltiplo de k).
- Se divide A en k sublistas, B_i, de tamaño $\frac{n}{k}$.
- A cada procesador P_i se le asocia la sublista B_i, que ordena por fusión secuencial.
- Las k sublistas obtenidas, se fusionan dos a dos usando la técnica del árbol binario completo.

$\texttt{procedimiento ordenacion}_{\mathbf{CREW}}\ (A[1,\ldots,n])$
    $\texttt{para cada}\ P_i\ \texttt{tal que}\ 1 \leqslant i \leqslant k\ \texttt{hacer}$
      $\texttt{para cada}\ j \leftarrow 1\ \texttt{hasta}\ \frac{n}{k}\ \texttt{hacer}$
        $b_i^j \leftarrow a_{(i-1)\frac{n}{k}+j}$
    $\texttt{para cada}\ P_i\ \texttt{tal que}\ 1 \leqslant i \leqslant k\ \texttt{hacer}$
      $\texttt{ord-fusion}\ (B_i = (b_i^1, \ldots, b_i^{\frac{n}{k}}))$
      $B_i^1 \leftarrow B_i;\ P_i^1 \leftarrow \{P_i\}$
    $u \leftarrow 1;\ v \leftarrow k$
    $\texttt{mientras}\ v > 1\ \texttt{hacer}$
      $\texttt{para cada}\ m \leftarrow 1\ \texttt{hasta}\ \lfloor \frac{v}{2} \rfloor\ \texttt{hacer}$
        $P_m^{u+1} \leftarrow P_{2m-1}^u \cup P_{2m}^u$
        $B_m^{u+1} \leftarrow \texttt{fusion}_{\mathbf{CREW}-P_m^{u+1}}(B_{2m-1}^u, B_{2m}^u)$
      $\texttt{si}\ v\ \texttt{es impar entonces}$
        $P_{\lceil \frac{v}{2} \rceil}^{u+1} \leftarrow P_v^u;\ B_{\lceil \frac{v}{2} \rceil}^{u+1} \leftarrow B_v^u$
      $u \leftarrow u+1;\ v \leftarrow \lceil \frac{v}{2} \rceil$

Análisis del algoritmo SIMD-CREW

- *Tiempo de ejecución*:
  - Primera fase: $\theta(\frac{n}{k})$.
  - Segunda fase: $\theta(\frac{n}{k}\lg\frac{n}{k})$.
  - Tercera fase: $\sum_{j=1}^{\lg k}(\lg 2^{j-1} \cdot \frac{n}{k} + \frac{(2^{j-1}n/k)n}{2^j})$.

  Por tanto, el tiempo total de ejecución es del orden

  $O(\frac{n}{k} + \frac{n}{k}\lg\frac{n}{k} + \lg n \lg k + \frac{n}{k}\lg k) \subseteq O(\frac{n}{k}\lg n + \lg n \lg k)$

- *Coste:* $C(n,k) \in O(n\lg n + k\lg n\lg k)$.

Para $k = \frac{n}{\lg n}$ se tiene que $C(n,k) \in O(n\lg n)$.

Luego el algoritmo es **óptimo**.

Ordenación SIMD-EREW

Se puede describir un algoritmo de ordenación SIMD-EREW, a partir de uno SIMD-CREW, simulando la lectura concurrente (usando difusión múltiple).

Análisis del algoritmo SIMD-EREW

- *Tiempo de ejecución*:

  $t(n,k) \in O((\frac{n}{k}\mathsf{lg}n + \mathsf{lg}n\mathsf{lg}k)\mathsf{lg}k) \subseteq O((\frac{n}{k} + \mathsf{lg}k)\mathsf{lg}n\mathsf{lg}k)$

- *Coste:* $C(n,k) \in O((n + k\mathsf{lg}k)\mathsf{lg}n\mathsf{lg}k)$.

Luego el algoritmo **no** es óptimo.

Algoritmos paralelos de ordenación SIMD con red de interconexión

Ordenación SIMD por redes de fusión

Sea A una lista de tamaño $n = 2^r$ que se trata de ordenar.

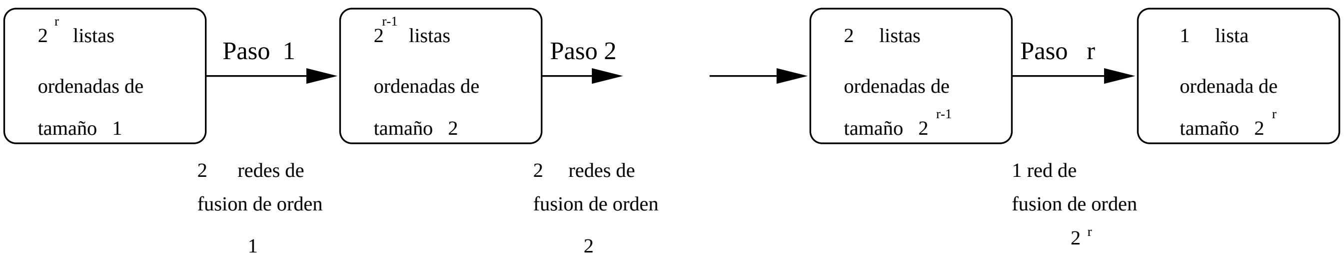
<u>Paso 1.</u> A es una lista formada por n sublistas ordenadas de tamaño $1 = 2^{1-1}$. Se ordenan mediante $\frac{n}{2}$ redes de fusión de orden $1 = 2^1 - 1$.

<u>Paso 2.</u> Las $\frac{n}{2}$ sublistas ordenadas de tamaño $2 = 2^{2-1}$, obtenidas en el paso anterior, se ordenan mediante $\frac{n}{2^2}$ redes de fusión de orden 2^{2-1}.

<u>Paso 3.</u> Las $\frac{n}{2^2}$ sublistas ordenadas de tamaño $2^2 = 2^{3-1}$, obtenidas en el paso anterior, se ordenan mediante $\frac{n}{2^3}$ redes de fusión de orden 2^{3-1}.

. . . .

<u>Paso r.</u> Las 2 sublistas ordenadas obtenidas en el paso anterior, se ordenan mediante $\frac{n}{2^r} = 1$ red de fusión de orden 2^{r-1}.



Análisis del algoritmo

(a) *Tiempo de ejecución.* Si $F(2^i)$ es el tiempo de ejecución que necesita una red de fusión de orden 2^{i-1} $(i \geq 1)$, entonces

$$t(n, k(n)) = \sum_{i=1}^{\lg n} F(2^i) = \sum_{i=1}^{\lg n} i \in \theta(\lg^2 n).$$

(b) *Número de procesadores.* Si $k^*(2^i)$ es el número de procesadores que necesita una red de fusión de orden 2^{i-1} $(i \geq 1)$, entonces

$$k(n) = \sum_{i=1}^{\lg n} 2^{(\lg n)-i} \cdot k^*(2^i) \in \theta(n \lg^2 n)$$

(c) *Aceleración, coste y eficiencia.*

$$S(n, k(n)) \in \theta\left(\frac{n}{\lg n}\right)$$

$$C(n, k(n)) \in \theta(n \lg^4 n)$$

$$E(n, k(n)) \in \theta\left(\frac{1}{\lg^3 n}\right)$$

Por tanto, este algoritmo **no** es óptimo.

Ordenación en SIMD–MC^1

Sea $A = (a_0, \ldots, a_{n-1})$ una lista numérica.

Sean $B = (b_0, \ldots, b_{n-1})$ y $T = (t_0, \ldots, t_{n-1})$ listas auxiliares (con variables locales).

Diseñamos un algoritmo $\mathbf{SIMD - MC^1}$ con n procesadores, P_i $(0 \leq i \leq n-1)$, que almacena a_i, b_i, t_i.

Algoritmo de trasposición impar–par.

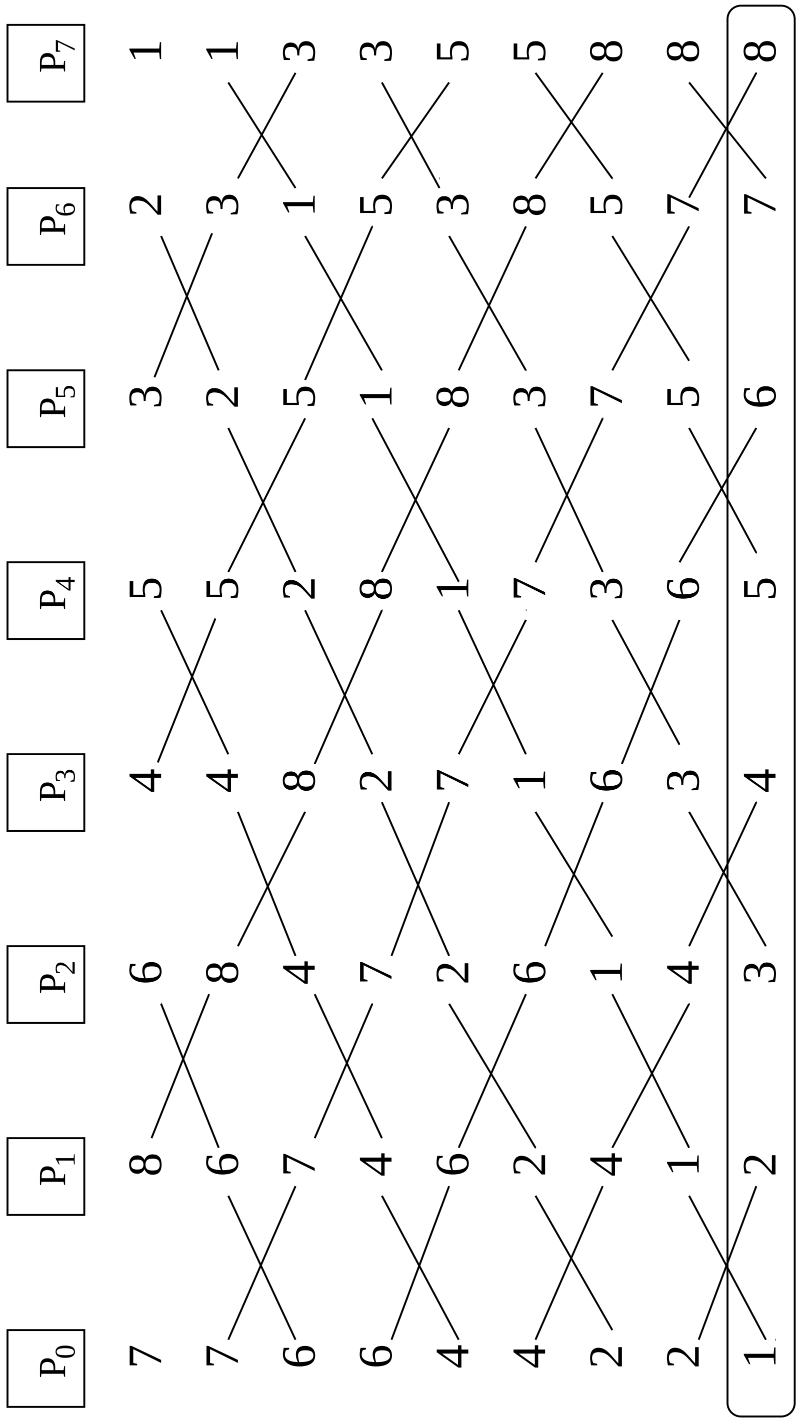
El algoritmo efectúa $\lfloor \frac{n}{2} \rfloor$ iteraciones, cada una consta de dos fases:

- Fase 1ª: Intercambio impar-par.
  - Si $0 \leq j < n-1$ y j impar, comparamos a_j con a_{j+1} y almacenamos
$$\begin{array}{lll} \text{mín}\{a_j, a_{j+1}\} & \text{en} & P_j \\ \text{máx}\{a_j, a_{j+1}\} & \text{en} & P_{j+1} \end{array}$$

- Fase 2ª: Intercambio par-impar.
  - Si $0 \leq j < n-1$ y j par, comparamos a_j con a_{j+1} y almacenamos
$$\begin{array}{lll} \text{mín}\{a_j, a_{j+1}\} & \text{en} & P_j \\ \text{máx}\{a_j, a_{j+1}\} & \text{en} & P_{j+1} \end{array}$$

De manera análoga se define el algoritmo de **trasposición par–impar**.

Ejemplo de ilustración: Ordenar por trasposición impar–par la lista $(7, 8, 6, 4, 5, 3, 2, 1)$.



Pseudocódigo del algoritmo $OTIP$.

procedimiento $OTIP(a_0, \ldots, a_{n-1})$

  para $i \leftarrow 1$ **hasta** $\lfloor \frac{n}{2} \rfloor$ **hacer**

    para cada P_j `tal que` $0 \leq j \leq n-1$ `hacer`

      si $j < n-1$ `y` j `impar` **entonces**

        $t_j \Longleftarrow a_{j+1}$

        $b_j \leftarrow \texttt{máx}\{a_j, t_j\}$

        $a_j \leftarrow \texttt{mín}\{a_j, t_j\}$

      si j `es par` **entonces**

        si $j > 0$ **entonces**

          $a_j \Longleftarrow b_{j-1}$

        $t_j \Longleftarrow a_{j+1}$

        $b_j \leftarrow \texttt{máx}\{a_j, t_j\}$

        $a_j \leftarrow \texttt{mín}\{a_j, t_j\}$

      si j `es impar` **entonces**

        $a_j \Longleftarrow b_{j-1}$

Análisis del algoritmo

- *Tiempo de ejecución:* $t(n, n) \in \theta(n)$.
- *Aceleración:* $S(n, n) \in \theta(\lg n)$.
- *Coste:* $C(n, n) \in \theta(n^2)$.
- *Eficiencia:* $E(n, n) \in \theta(\frac{\lg n}{n})$.

Luego el algoritmo $OTIP$ no es óptimo.

Teorema de optimalidad relativa: *Cualquier algoritmo* $\mathbf{SIMD - MC^1}$ *que ordene una lista de tamaño* n*, con* n *procesadores, necesita al menos un tiempo de ejecución del orden* $\theta(n)$*, en el caso peor.*

En cierto sentido, el algoritmo $OTIP$ es óptimo.

Teorema: Los algoritmos $OTIP$ y $OTPI$ son correctos para ordenar una lista numérica.

Demostración: Por inducción débil sobre el tamaño, n, del dato de entrada.

- Para $n = 2$ el resultado es inmediato.

- Supongamos que $OTIP$ y $OTPI$ son correctos para ejemplares de tamaño n.

  Veamos que $OTIP$ es correcto para ejemplares de tamaño $n + 1$ (análogo para $OTPI$).

  Sea $(a_0, \ldots, a_n)$ una lista de tamaño $n + 1$. Sean P_i $(0 \leq i \leq n)$ los procesadores que disponemos (inicialmente a_i está en P_i).

  Sea a_m =máx$\{a_0, \ldots, a_n\}$.

  ⋆ Tras la ejecución de $OTIP$, a_m estará en P_n.
  ⋆ Si $(b_0, \ldots, b_{n-1})$ es la lista deducida de la de partida eliminando a_m, la traza de $OTPI$ sobre $(b_0, \ldots, b_{n-1})$ coincide con una parte de la traza de $OTIP$ sobre $(a_0, \ldots, a_n)$ (figura página siguiente).
  ⋆ Como $OTPI$ es correcto para $(b_0, \ldots, b_{n-1})$, resulta que $OTIP$ lo será para $(a_0, \ldots, a_n)$.

■

Traza de $OTPI$ sobre $(b_0, \ldots, b_{n-1})$.

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
7	8	6	4	5	3	2	1
7	6	8	4	5	2	3	1
6	7	4	8	2	5	1	3
6	4	7	2	8	1	5	3
4	6	2	7	1	8	3	5
4	2	6	1	7	3	8	5
2	4	1	6	3	7	5	8
2	1	4	3	6	5	7	8
1	2	3	4	5	6	7	8

Mejora del algoritmo $OTIP$ mediante la reducción del número de procesadores

Simulemos dicho algoritmo utilizando $k < n$ procesadores, P_i, para ejemplares de tamaño n (supondremos k par y n múltiplo de k).

Sea $A[1, \ldots, n]$ la lista a ordenar.

- Se divide la lista A en k sublistas de tamaño $\frac{n}{k}$ (sean $B_1, \ldots, B_k$ dichas sublistas).
- Se asigna a cada procesador P_i la lista B_i.
- En paralelo, cada procesador P_i ordena secuencialmente, por *fusión*, la sublista B_i.
- A continuación se realizan $\frac{k}{2}$ iteraciones cada una con dos fases:
  - ⋆ <u>Fase impar</u>: Cada procesador P_i de índice impar ordena por fusión secuencial B_i y B_{i+1}: la primera mitad resultante se queda en P_i y la otra pasa a P_{i+1}.
  - ⋆ <u>Fase par</u>: Similar a la anterior para procesadores de índices pares.

Pseudocódigo del algoritmo paralelo.

procedimiento $trasposicion^*(A[1,\ldots,n],k)$

    Dividir A en k sublistas B_i de tamaño $\frac{n}{k}$
    para cada P_i tal que $1 \le i \le k$ **hacer**
        ord-fusion (B_i)
    para $j \leftarrow 1$ **hasta** $\frac{k}{2}$ **hacer**
        para cada P_i tal que $1 \le i \le k \wedge i$ impar **hacer**
            mezcla (B_i, B_{i+1}, C_i)
            $B_i \leftarrow (c_1,\ldots,c_{\frac{n}{k}})$
            $B_{i+1} \leftarrow (c_{\frac{n}{k}+1},\ldots,c_{\frac{2n}{k}})$
        para cada P_i tal que $1 \le i \le k \wedge i$ par **hacer**
            mezcla (B_i, B_{i+1}, C_i)
            $B_i \leftarrow (c_1,\ldots,c_{\frac{n}{k}})$
            $B_{i+1} \leftarrow (c_{\frac{n}{k}+1},\ldots,c_{\frac{2n}{k}})$

Análisis del algoritmo:

- $t(n,k) \in \theta(k) + \theta(\frac{n}{k}\cdot \lg\frac{n}{k}) + \frac{k}{2}\cdot\theta(\frac{n}{k}) = \theta(n + \frac{n}{k}\cdot\lg\frac{n}{k})$.
- $S(n,k) \in \theta(\frac{k \lg n}{k+\lg\frac{n}{k}})$.
- $C(n,k) \in \theta(k\cdot n + n\cdot\lg\frac{n}{k}) \overset{k\le\lg n}{\subseteq} O(n\lg n)$
- $E(n,k) \in \theta(\frac{\lg n}{k+\lg\frac{n}{k}})$.