# Sorting

The Ugly…

# Parallel Merge sort

**Recall:**
Sorting *n* elements using one thread for each merge operation gave:

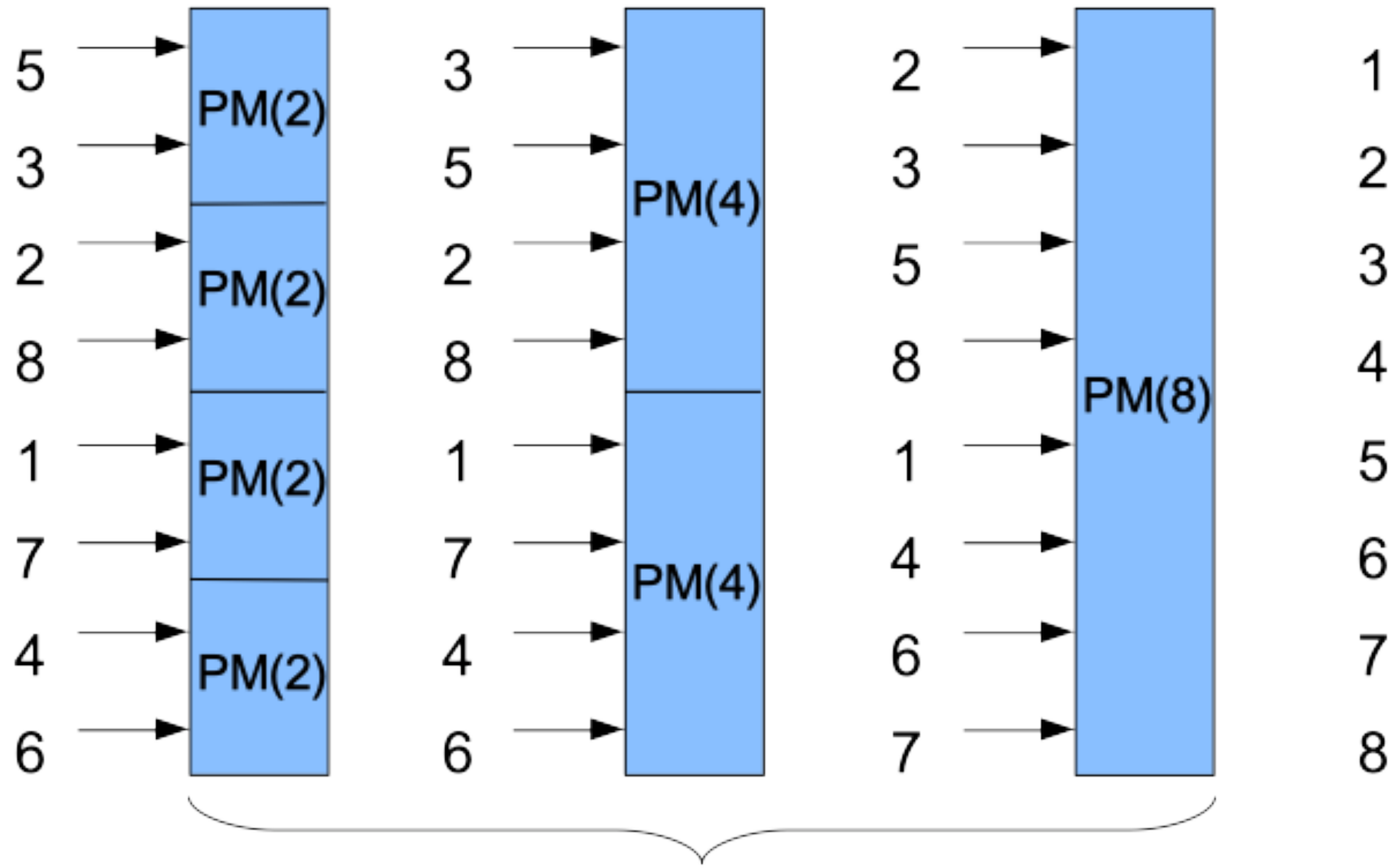
- $T_p = \Omega(n)$
- $S(p) \leq \frac{1}{2}\log(n)$

**To get better performance: Must parallelize the merge operation!**

Given sorted lists $a = [a_0, a_1, \ldots, a_r]$ and $b = [b_0, b_1, \ldots, b_r]$
Compute Merge(a,b)

Any ideas?

# Sorting using parallel merge




If p = n we get log(n) stages
If p < n we get log(p) stages

$$T_n = \sum_{i=1}^{\log(n)} PM(2^i) \qquad T_p = \sum_{i=1}^{\log(p)} PM(n/p * 2^i)$$

# Parallel Merge sort

**Want to create p independent tasks**

Partition $a = a_1, a_2, \ldots, a_r$ and $b = b_1, b_2, \ldots, b_r$ into *p* parts $a =$ ($A_1$,$A_2$,…,$A_p$) and *b* = ($B_1$,$B_2$,…,$B_p$) so that:

Merge(*a*,*b*) = Merge($A_1$,$B_1$), Merge($A_2$,$B_2$), … , Merge($A_p$,$B_p$)

Also, would like $|A_i| + |B_i| = 2r/p$ i.e. an even load balance

**Example:**

$a = 0,1,3,4,8,10$ and $b = 2,5,6,7,9,11$ r = 6, p = 3 2r/p = 4

| | | |
|---|---|---|
| $A_1$ ={0,1,3} $B_1$ ={2} | $A_2$ ={4} $B_2$ ={5,6,7} | $A_3$ ={8,10} $B_3$ ={9,11} |
| $t_1$ | $t_2$ | $t_3$ |
| 0,1,2,3 | 4,5,6,7 | 8,9,10,11 |

# Parallel Merge sort

**Want to create p independent tasks**

Partition $a = a_1, a_2, \ldots, a_r$ and $b = b_1, b_2, \ldots, b_r$ into *p* parts

$a =$ $(A_1, A_2, \ldots, A_p)$ and $b = (B_1, B_2, \ldots, B_p)$ so that:

Merge($a,b$) = Merge($A_1,B_1$), Merge($A_2,B_2$), … , Merge($A_p,B_p$)

Also, would like $|A_i| + |B_i| = 2r/p$ i.e. an even load balance

Must have $\max\{A_i, B_i\} \leq \min\{A_{i+1}, B_{i+1}\}$ for all $i$

Wish to compute largest element from a and b that will fit in each interval:

$$c = [c_1, c_{2r/p}],\ [c_{1+2r/p}, c_{4r/p}],\ \ldots\ ,\ [c_{1+2r-2r/p}, c_{2r}]$$

$$t_1 \qquad\qquad t_2 \qquad\qquad\qquad\qquad t_p$$

where $c_i$ is the i´th smallest element in the final merged sequence

# Parallel Merge sort

**New problem:**

Determine position of max element from *a* and *b* in each $[c_{1+(i-1)*2r/p}, c_{i*2r/p}]$

Can be done (in parallel) for each i by thread $t_i$ using a sequential algorithm

**Sequential problem:**

Let k = i*2r/p. Determine max elements from a and b that are $\leq c_k$.

Will maintain two intervals $[l_a,u_a]$ and $[l_b,u_b]$ such that:

- both $[a_1,a_{l_a-1}]$ and $[b_1,b_{l_b-1}]$ belong to the k smallest elements
- the k smallest elements are contained in the union of $[a_1,a_{u_a}]$ and $[b_1,b_{u_b}]$

Follows that the index of the last element of $A_i$ is in $[l_a-1,u_a]$ (and similar for $B_i$).

# Parallel Merge sort

**New problem:**

Determine position of max element from *a* and *b* in each $[c_{1+(i-1)*2r/p}, c_{i*2r/p}]$

Can be done (in parallel) for each i by thread $t_i$ using a sequential algorithm

**Example:**

$a = 0,1,3,4,8,10$ and $b = 2,5,6,7,9,11$ r = 6, p = 3 2r/p = 4

Compute $A_1$ and $B_1$. Searching for 4th smallest element

Start with $l_a = 1$ $u_a = 6$ and $l_b = 1$ $u_b = 6$

Compare middle elements 0,1,**3**,4,8,10 2,5,**6**,7,9,11 → **3** < **6**

**3** has index 3 in *a*, **6** has index 3 in b → 3 + 3 = 6 elements ≤ **6**

4th smallest cannot be in 6,7,9,11 → Can set $u_b = 2$

# Parallel Merge sort

**Example:**

$a = 0,1,3,4,8,10$ and $b = 2,5,6,7,9,11$ $r = 6$, $p = 3$ $2r/p = 4$

Compute $A_1$ and $B_1$.

Initially $l_a = 1, u_a = 6$ and $l_b = 1$, $u_b = 6$.

After first comparison $l_a = 1, u_a = 6$ and $l_b = 1$, $u_b = 2$

**2. step:** Compare $a_3 = 3$ and $b_1 = 2$ (Since $(l_b + u_b)/2 = 1$)

**3** > **2**, How many elements can be $\leq 2$? Only 1 in b (2 itself) and at most 2 in $a$ (0 and 1). Thus 2 must be part of the 4 smallest elements. Can set $l_b = 2$.

**3. step:** Compare $a_3 = 3$ and $b_2 = 5$ (Since $(l_b + u_b)/2 = 2$)

**3** < **5**, How many elements are < 5? At least 3 in a and 1 in b. Thus 5 is not part of the 4 smallest elments. Set $u_b = 1$.

Since $u_b < l_b$ it follows that $b_{1=u_b}$ is the last element from b in $B_1$. Then element $a_3$ must be the last element from $a$ in $A_1$ .

# Parallel Merge sort

**In general:**

Will show how we can cut one of $[l_a, u_a]$ and $[l_b, u_b]$ in half:

Let $m_a$ be middle index of $[l_a, u_a]$ and $m_b$ the middle index of $[l_b, u_b]$

Assume $a_{m_a} < b_{m_b}$ (other case is symmetric)

- If $k < m_a + m_b$, then the elements in $b[m_b..u_b]$ cannot belong to the $k$ smallest elements. Can set $u_b = m_b - 1$.
- If $k \geq m_a + m_b$, then all elements in $a[l_a..m_a]$ belong to the $k$ smallest elements. Can set $l_a = m_a + 1$.

Repeat until one of the ranges is empty: $u_a = l_a - 1$ or $u_b = l_b - 1$

Set remaining value $u_b = k - u_a$ or $u_a = k - u_b$

# Parallel Merge sort

**Complexity of one step:**

One of a and b is cut in half in each iteration:   At most $2 \log r$ iterations

**Outline of algorithm:**

First each thread sorts n/p elements

For i = 0 to $\log p$ - 1:

  Use $2^{i+1}$ threads to merge two sequences of lengths $2^i$ n/p

**Complexity of algorithm:**

Initial sorting:       O(n/p log (n/p))

Merging sequences of length $2^i$ n/p :

  Splitting: $O(\log(2^i \cdot n/p)) = O(\log(n))$

  Merging: *O(n/p)*

$$T_p = O(n/p \log(n/p) + \log(p)(\log(n) + n/p)) = O(\log(n)^2 + n \log(n)/p)$$

# Odd – Even Merge sort

PM(n): Two sorted lists: $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$

Partition each list:

Odd index: $a_1, a_3, \ldots, a_{n-1}$ and $b_1, b_3, \ldots, b_{n-1}$

Even index: $a_2, a_4, \ldots, a_n$ and $b_2, b_4, \ldots, b_n$

Apply PM(n/2) to $a_1, a_3, \ldots, a_{n-1}$ and $b_1, b_3, \ldots, b_{n-1}$ giving $c_1, c_2, \ldots, c_n$

Apply PM(n/2) to $a_2, a_4, \ldots, a_n$ and $b_2, b_4, \ldots, b_n$ giving $d_1, d_2, \ldots, d_n$

Final merge (in parallel):

$e_1 = c_1$,

$e_{2i} = \min\{c_{i+1}, d_i\}$, $1 < i < n$

$e_{2i+1} = \max\{c_{i+1}, d_i\}$, $1 < i < n$

$e_{2n} = d_n$

Note the number of stages in PM(n) is 1 + PM(n/2) = O(log(n))

# Example

a = [2, 4, 5, 8]  b =[1, 3, 6, 7]

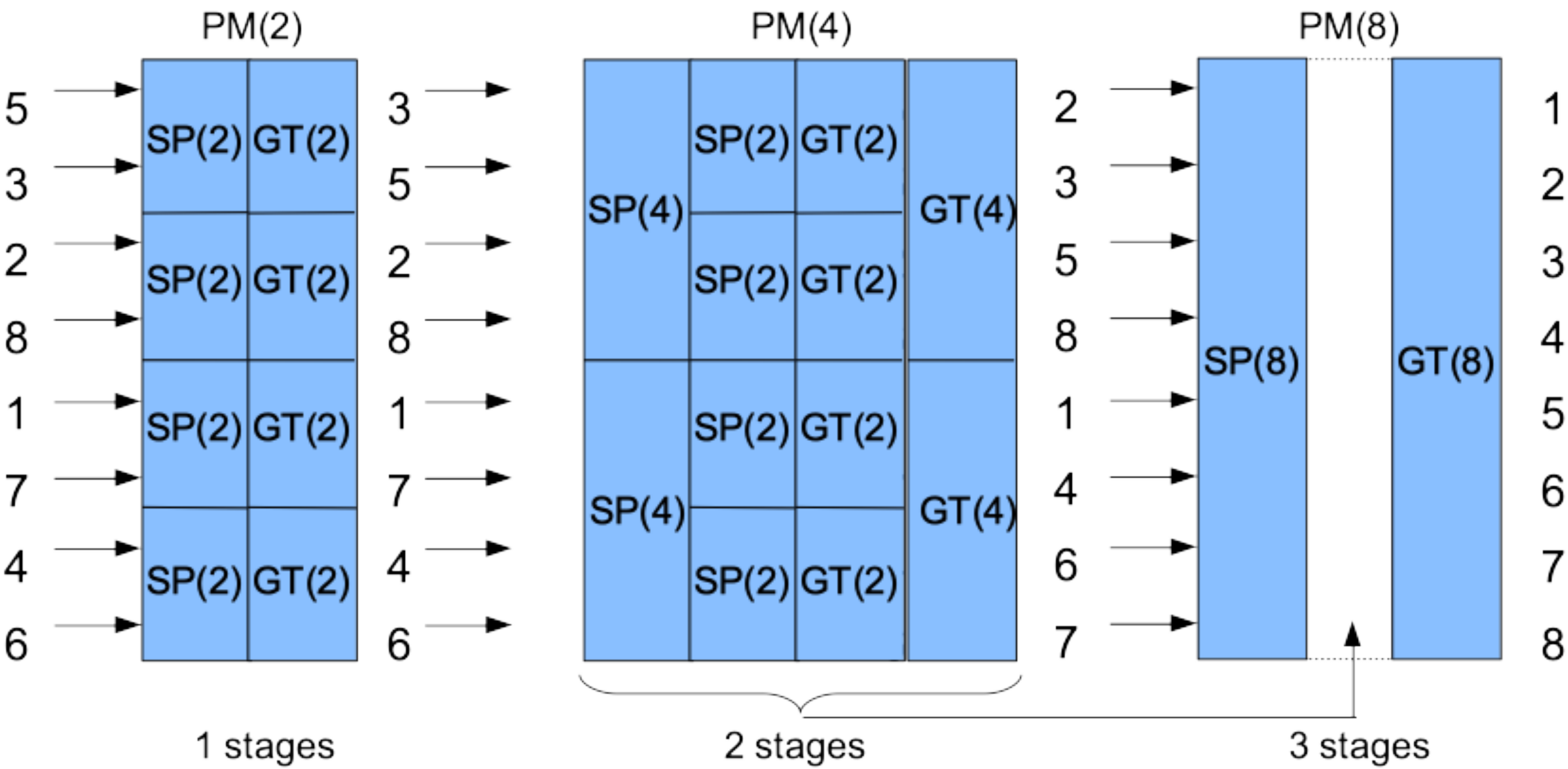
Odd: [2, 5] [1, 6]    Even: [4, 8] [3, 7]

c = [1, 2, 5, 6]    d = [3, 4, 7, 8] (Done recursively)

$e_1 = \quad 1$
$e_2 = \min\{2,3\} = 2$
$e_3 = \max\{2,3\} = 3$
$e_4 = \min\{5,4\} = 4$
$e_5 = \max\{5,4\} = 5$
$e_6 = \min\{6,7\} = 6$
$e_7 = \max\{6,7\} = 7$
$e_8 = \quad 8$

# A full algorithm

SP(n): Splitt two lists of length n into four lists depending on indices
GT(n): Gather result from two sorted lists each of length n.




If list is of length $2^i$ then we need i stages in PM($2^i$).

# Analysis

Need i stages to output sorted lists of length $2^i$ from sorted lists of length $2^{i-1}$

The total sequential time of each stage is O(n).

To go from lists of lengths $2^{i-1}$ to lists of length $2^i$ takes time O(n*i).

Must go from lists of length $2^0$ to $2^1$ to $2^2$ to $2^3$ all the way up to $2^{\log n} = n$.

For the whole algorithm:

$$T_s = O(n) \sum_{i=1}^{\log(n)} i = O(n \log^2 n)$$

In parallel when using n threads, each stage takes O(1) time,

Thus $T_n = \sum_{i=1}^{\log(n)} i = O(\log^2(n))$

$S_n = n \log(n) / \log^2(n) = n/\log(n)$ Fairly close to optimal: O(n)

# Proof of correctness

Assume that all elements in A and B are distinct and consider element $c_{i+1}$ from the sorted list C.

Assume that $c_1, c_2, \ldots, c_i$ contains r elements from A and q elements from B. Thus $i = r + q$.

The r elements from A in C that are smaller than $c_{i+1}$ are then $\{a_1, a_3, a_5, a_7, \ldots, a_{2r-1}\}$ and it follows that $a_{2r-1} < c_{i+1} \leq a_{2r+1}$, with equality only if $c_{i+1}$ is the (2r+1)st element in A.
Similarly $b_{2q-1} < c_{i+1} \leq b_{2q+1}$.

If $c_{i+1}$ is from A then $c_{i+1} = a_{2r+1}$ and it follows that $a_{2r} < c_{i+1}$ and there are exactly 2r elements from A smaller than $c_{i+1}$. Then $c_{i+1} < b_{2q+1}$ and the only element that has not been placed relative to $c_{i+1}$ is $b_{2q}$. Thus we either have $2r + 2q - 1 = 2i - 1$ or $2r + 2q = 2i$ elements smaller than $c_{i+1}$ (if $c_{i+1} < b_{2q}$ or not) and $c_{i+1}$ must be placed in position 2i or 2i+1.

If $c_{i+1}$ is from B then there are 2q elements from B smaller than $c_{i+1}$ and the only element that has not been placed relative to $c_{i+1 \text{ is}}$ is $a_{2r}$. Still $c_{i+1}$ again has to be placed either in position 2i or 2i+1 of E.

A similar argument shows that $d_i$ must also be placed in either position 2i or 2i+1.

# What if p < n?

- Start by sorting sequences of length n/p sequentially
- When the split operation has created 2p sequences, switch to sequential merge