# Searching

An extension of binary search with $p$ processors gives that one can find the rank of an element in

$$\log_{p+1}(n) = \frac{\log n}{\log(p+1)}$$

many parallel steps with $p$ processors. (not work-optimal)

This requires a CREW PRAM model. For the EREW model searching cannot be done faster than $\mathcal{O}(\log n - \log p)$ with $p$ processors even if there are $p$ copies of the search key.

# Merging

Given two sorted sequences $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$, compute the sorted squence $C = (c_1, \ldots, c_n)$.

**Definition 1**

Let $X = (x_1, \ldots, x_t)$ be a sequence. The rank $\mathrm{rank}(y : X)$ of $y$ in $X$ is

$$\mathrm{rank}(y : X) = |\{x \in X \mid x \leq y\}|$$

For a sequence $Y = (y_1, \ldots, y_s)$ we define $\mathrm{rank}(Y : X) := (r_1, \ldots, r_s)$ with $r_i = \mathrm{rank}(y_i : X)$.

# Merging

We have already seen a merging-algorithm that runs in time $\mathcal{O}(\log n)$ and work $\mathcal{O}(n)$.

Using the fast search algorithm we can improve this to a running time of $\mathcal{O}(\log\log n)$ and work $\mathcal{O}(n\log\log n)$.

# Merging

Input: $A = a_1, \ldots, a_n$; $B = b_1, \ldots, b_m$; $m \le n$

1. if $m < 4$ then rank elements of $B$, using the parallel search algorithm with $p$ processors. Time: $\mathcal{O}(1)$. Work: $\mathcal{O}(n)$.
2. Concurrently rank elements $b_{\sqrt{m}}, b_{2\sqrt{m}}, \ldots, b_m$ in $A$ using the parallel search algorithm with $p = \sqrt{n}$. Time: $\mathcal{O}(1)$. Work: $\mathcal{O}(\sqrt{m} \cdot \sqrt{n}) = \mathcal{O}(n)$

   $j(i) := \mathrm{rank}(b_{i\sqrt{m}} : A)$
3. Let $B_i = (b_{i\sqrt{m}+1}, \ldots, b_{(i+1)\sqrt{m}-1})$; and $A_i = (a_{j(i)+1}, \ldots, a_{j(i+1)})$.

   Recursively compute $\mathrm{rank}(B_i : A_i)$.
4. Let $k$ be index not a multiple of $\sqrt{m}$. $i = \lceil \frac{k}{\sqrt{m}} \rceil$. Then $\mathrm{rank}(b_k : A) = j(i) + \mathrm{rank}(b_k : A_i)$.

The algorithm can be made work-optimal by standard techniques.

proof on board...

# Mergesort

**Lemma 2**

*A straightforward parallelization of Mergesort can be implemented in time $\mathcal{O}(\log n \log\log n)$ and with work $\mathcal{O}(n \log n)$.*

*This assumes the CREW-PRAM model.*

# Mergesort

Let $L[v]$ denote the (sorted) sublist of elements stored at the leaf nodes rooted at $v$.

We can view Mergesort as computing $L[v]$ for a complete binary tree where the leaf nodes correspond to nodes in the given array.

Since the merge-operations on one level of the complete binary tree can be performed in parallel we obtain time $\mathcal{O}(h \log\log n)$ and work $\mathcal{O}(hn)$, where $h = \mathcal{O}(\log n)$ is the height of the tree.

# Pipelined Mergesort

We again compute $L[v]$ for every node in the complete binary tree.

After round $s$, $L_s[v]$ is an **approximation** of $L[v]$ that will be improved in future rounds.

For $s \geq 3\,\mathrm{height}(v)$, $L_s[v] = L[v]$.

# Pipelined Mergesort

In every round, a node $v$ sends $\text{sample}(L_s[v])$ (an approximation of its current list) upwards, and receives approximations of the lists of its children.

It then computes a new approximation of its list.

A node is called active in round $s$ if $s \leq 3\,\text{height}(v)$ (this means its list is not yet complete at the start of the round, i.e., $L_{s-1}[v] \neq L[v]$).

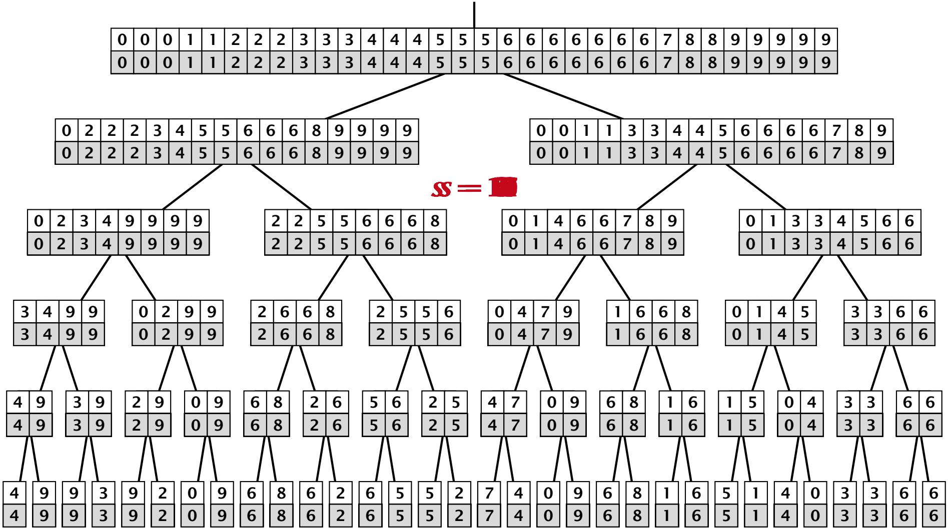
# Pipelined Mergesort

**Algorithm 11** ColeSort()

```
initialize L_0[v] = A_v for leaf nodes; L_0[v] = ∅ otw.
for s ← 1 to 3 · height(T) do
    for all active nodes v do
        // u and w children of v
        L'_s[u] ← sample(L_{s-1}[u])
        L'_s[w] ← sample(L_{s-1}[w])
        L_s[v] ← merge(L'_s[u], L'_s[w])
```

$$\text{sample}(L_s[v]) = \begin{cases} \text{sample}_4(L_s[v]) & s \le 3\,\text{height}(v) \\ \text{sample}_2(L_s[v]) & s = 3\,\text{height}(v) + 1 \\ \text{sample}_1(L_s[v]) & s = 3\,\text{height}(v) + 2 \end{cases}$$

# Colesort


0 0 0 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6 6 6 6 6 7 8 8 9 9 9 9 9
0 2 2 2 3 4 5 5 6 6 6 8 9 9 9 9
0 0 1 1 3 3 4 4 5 6 6 6 6 7 8 9
ss =
0 2 3 4 9 9 9 9
2 2 5 5 6 6 6 8
0 1 4 6 6 7 8 9
0 1 3 3 4 5 6 6
3 4 9 9
0 2 9 9
2 6 6 8
2 5 5 6
0 4 7 9
1 6 6 8
0 1 4 5
3 3 6 6
4 9
3 9
2 9
0 9
6 8
2 6
5 6
2 5
4 7
0 9
6 8
1 6
1 5
0 4
3 3
6 6
4 9 9 3 9 2 0 9 6 8 6 2 6 5 5 2 7 4 0 9 6 8 1 6 5 1 4 0 3 3 6 6

# Pipelined Mergesort

## Lemma 3

*After round* $s = 3\,\mathrm{height}(v)$*, the list* $L_s[v]$ *is complete.*

**Proof:**

- clearly true for leaf nodes
- suppose it is true for all nodes up to height $h$;
- fix a node $v$ on level $h+1$ with children $u$ and $w$
- $L_{3h}[u]$ and $L_{3h}[w]$ are complete by induction hypothesis
- further $\mathrm{sample}(L_{3h+2}[u]) = L[u]$ and $\mathrm{sample}(L_{3h+2}[v]) = L[v]$
- hence in round $3h+3$ node $v$ will merge the complete list of its children; after the round $L[v]$ will be complete

# Pipelined Mergesort

**Lemma 4**

*The number of elements in lists $L_s[v]$ for active nodes $v$ is at most $\mathcal{O}(n)$.*

proof on board...

## Definition 5

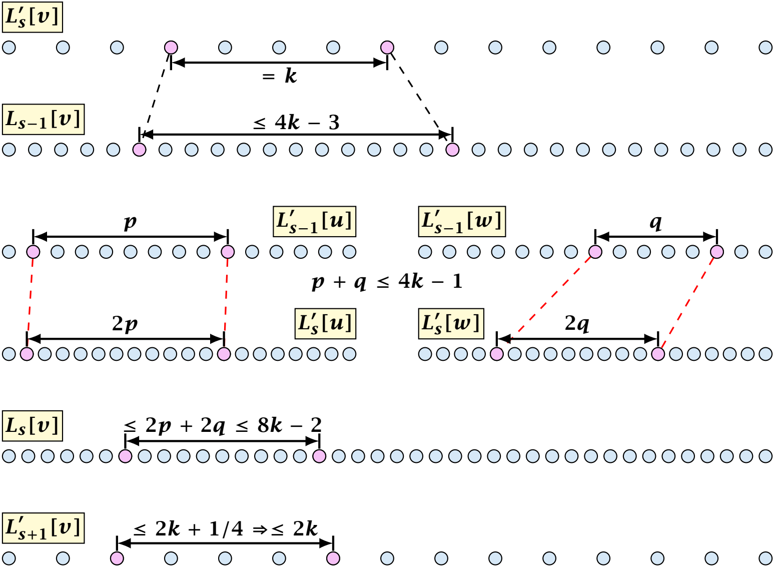
A sequence $X$ is a $c$-cover of a sequence $Y$ if for any two consecutive elements $\alpha, \beta$ from $(-\infty, X, \infty)$ the set $|\{y_i \mid \alpha \leq y_i \leq \beta\}| \leq c$.

# Pipelined Mergesort

**Lemma 6**
*$L'_s[v]$ is a 4-cover of $L'_{s+1}[v]$.*

If $[a,b]$ fulfills $|[a,b] \cap (A \cup \{-\infty, \infty\})| = k$ we say $[a,b]$ intersects $(-\infty, A, +\infty)$ in $k$ items.

**Lemma 7**
*If $[a,b]$ with $a, b \in L'_s[v] \cup \{-\infty, \infty\}$ intersects $(-\infty, L'_s[v], \infty)$ in $k \geq 2$ items, then $[a,b]$ intersects $(-\infty, L'_{s+1}, \infty)$ in at most $2k$ items.*




Note that the last step holds as long $L'_{s+1}[v] = \text{sample}_4(L_s[v])$. Otw. $L_{s-1}[v]$ has already been full, and hence, $L'_s[v], L'_{s+1}[v], L'_{s+2}[v]$ are 4-covers of the complete list $L[v]$, and also 4-covers of each other.

# Merging with a Cover

**Lemma 8**

*Given two sorted sequences $A$ and $B$. Let $X$ be a $c$-cover of $A$ and $B$ for constant $c$, and let $\operatorname{rank}(X : A)$ and $\operatorname{rank}(X : B)$ be known.*

*We can merge $A$ and $B$ in time $\mathcal{O}(1)$ using $\mathcal{O}(|X|)$ operations.*

# Merging with a Cover

**Lemma 9**
*Given two sorted sequences* $A$ *and* $B$*. Let* $X$ *be a* $c$*-cover of* $B$ *for constant* $c$*, and let* $\text{rank}(A : X)$ *and* $\text{rank}(X : B)$ *be known.*

*We can compute* $\text{rank}(A : B)$ *using* $\mathcal{O}(|X| + |A|)$ *operations.*

# Merging with a Cover

**Lemma 10**

*Given two sorted sequences $A$ and $B$. Let $X$ be a $c$-cover of $B$ for constant $c$, and let $\text{rank}(A:X)$ and $\text{rank}(X:B)$ be known.*

*We can compute $\text{rank}(B:A)$ using $\mathcal{O}(|X|+|A|)$ operations.*

Easy to do with concurrent read. Can also be done with exclusive read but non-trivial.

In order to do the merge in iteration $s+1$ in constant time we need to know

$$\mathrm{rank}(L_s[v]:L'_{s+1}[u]) \text{ and } \mathrm{rank}(L_s[v]:L'_{s+1}[w])$$

and we need to know that $L_s[v]$ is a $4$-cover of $L'_{s+1}[u]$ and $L'_{s+1}[w]$.

**Lemma 11**
*$L_s[v]$ is a 4-cover of $L'_{s+1}[u]$ and $L'_{s+1}[w]$.*

- $L_s[v] \supseteq L'_s[u], L'_s[w]$
- $L'_s[u]$ is 4-cover of $L'_{s+1}[u]$
- Hence, $L_s[v]$ is 4-cover of $L'_{s+1}[u]$ as adding more elements cannot destroy the cover-property.

# Analysis

## Lemma 12

*Suppose we know for every internal node $v$ with children $u$ and $w$*

- $\text{rank}(L'_s[v] : L'_{s+1}[v])$
- $\text{rank}(L'_s[u] : L'_s[w])$
- $\text{rank}(L'_s[w] : L'_s[u])$

*We can compute*

- $\text{rank}(L'_{s+1}[v] : L'_{s+2}[v])$
- $\text{rank}(L'_{s+1}[u] : L'_{s+1}[w])$
- $\text{rank}(L'_{s+1}[w] : L'_{s+1}[u])$

*in constant time and $\mathcal{O}(|L_{s+1}[v]|)$ operations, where $v$ is the parent of $u$ and $w$.*

Given

- $\operatorname{rank}(L'_s[u] : L'_{s+1}[u])$ **(4-cover)**
- $\operatorname{rank}(L'_s[w] : L'_s[u])$
- $\operatorname{rank}(L'_s[u] : L'_s[w])$
- $\operatorname{rank}(L'_s[w] : L'_{s+1}[w])$ **(4-cover)**

Compute

- $\operatorname{rank}(L'_{s+1}[w] : L'_s[u])$
- $\operatorname{rank}(L'_{s+1}[u] : L'_s[w])$

Compute

- $\operatorname{rank}(L'_{s+1}[w] : L'_{s+1}[u])$
- $\operatorname{rank}(L'_{s+1}[u] : L'_{s+1}[w])$

**ranks between siblings can be computed easily**

Given

- $\mathrm{rank}(L'_s[u] : L'_{s+1}[u])$ (**4-cover** → $\mathrm{rank}(L'_{s+1}[u] : L'_s[u])$)
- $\mathrm{rank}(L'_s[w] : L'_{s+1}[u])$
- $\mathrm{rank}(L'_s[u] : L'_{s+1}[w])$
- $\mathrm{rank}(L'_s[w] : L'_{s+1}[w])$ (**4-cover** → $\mathrm{rank}(L'_{s+1}[w] : L'_s[w])$)

Compute (recall that $L_s[v] = \mathrm{merge}(L'_s[u], L'_s[w])$)

- $\mathrm{rank}(L_s[v] : L'_{s+1}[u])$
- $\mathrm{rank}(L_s[v] : L'_{s+1}[w])$

Compute

- $\mathrm{rank}(L_s[v] : L_{s+1}[v])$ (by adding)
- $\mathrm{rank}(L'_{s+1}[v] : L'_{s+2}[v])$ (by sampling)