4.3 Divide & Conquer — Merging

$A = (a_1, \dots, a_n)$; $B = (b_1, \dots, b_n)$;
$\log n$ integral; $k := n/\log n$ integral;

Algorithm 8 GenerateSubproblems

$j_0 \leftarrow 0$
$j_k \leftarrow n$
for $1 \le i \le k-1$ **pardo**
    $j_i \leftarrow \text{rank}(b_{i \log n} : A)$
for $0 \le i \le k-1$ **pardo**
    $B_i \leftarrow (b_{i \log n + 1}, \dots, b_{(i+1)\log n})$
    $A_i \leftarrow (a_{j_i + 1}, \dots, a_{j_{i+1}})$

If C_i is the merging of A_i and B_i then the sequence $C_0 \dots C_{k-1}$ is a sorted sequence.

4.3 Divide & Conquer — Merging

We can generate the subproblems in time $\mathcal{O}(\log n)$ and work $\mathcal{O}(n)$.

Note that in a sub-problem B_i has length $\log n$.

If we run the algorithm again for every subproblem, (where A_i takes the role of B) we can in time $\mathcal{O}(\log\log n)$ and work $\mathcal{O}(n)$ generate subproblems where A_j and B_j have both length at most $\log n$.

Such a subproblem can be solved by a single processor in time $\mathcal{O}(\log n)$ and work $\mathcal{O}(|A_i| + |B_i|)$.

Parallelizing the last step gives total work $\mathcal{O}(n)$ and time $\mathcal{O}(\log n)$.

the resulting algorithm is work optimal

4.4 Maximum Computation

Lemma 4
On a CRCW PRAM the maximum of n numbers can be computed in time $\mathcal{O}(1)$ with n^2 processors.

proof on board...

4.4 Maximum Computation

Lemma 5
On a CRCW PRAM the maximum of n numbers can be computed in time $\mathcal{O}(\log\log n)$ with n processors and work $\mathcal{O}(n \log\log n)$.

proof on board...

4.4 Maximum Computation

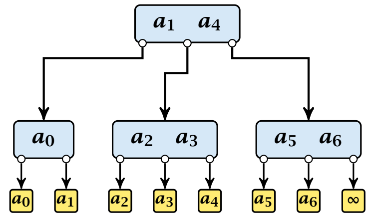
Lemma 6
On a CRCW PRAM the maximum of n numbers can be computed in time $\mathcal{O}(\log\log n)$ with n processors and work $\mathcal{O}(n)$.

proof on board...

4.5 Inserting into a $(2,3)$-tree

Given a $(2,3)$-tree with n elements, and a sequence $x_0 < x_1 < x_2 < \cdots < x_k$ of elements. We want to insert elements $x_1, \dots, x_k$ into the tree ($k \ll n$).

time: $\mathcal{O}(\log n)$; work: $\mathcal{O}(k \log n)$

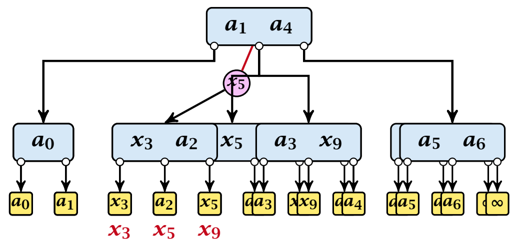


4.5 Inserting into a $(2,3)$-tree

1. determine for every x_i the leaf element before which it has to be inserted
   time: $\mathcal{O}(\log n)$; work: $\mathcal{O}(k \log n)$; CREW PRAM

   all x_i's that have to be inserted before the same element form a chain
2. determine the largest/smallest/middle element of every chain
   time: $\mathcal{O}(1)$; work: $\mathcal{O}(k)$;
3. insert the middle element of every chain
   compute new chains
   time: $\mathcal{O}(\log n)$; work: $\mathcal{O}(k_i \log n)$; k_i= #inserted elements
   (computing new chains is constant time)
4. repeat Step 3 for logarithmically many rounds
   time: $\mathcal{O}(\log n \log k)$; work: $\mathcal{O}(k \log n)$;

Step 3



- each internal node is split into at most two parts
- each split operation promotes at most one element
- hence, on every level we want to insert at most one element per successor pointer
- we can use the same routine for every level