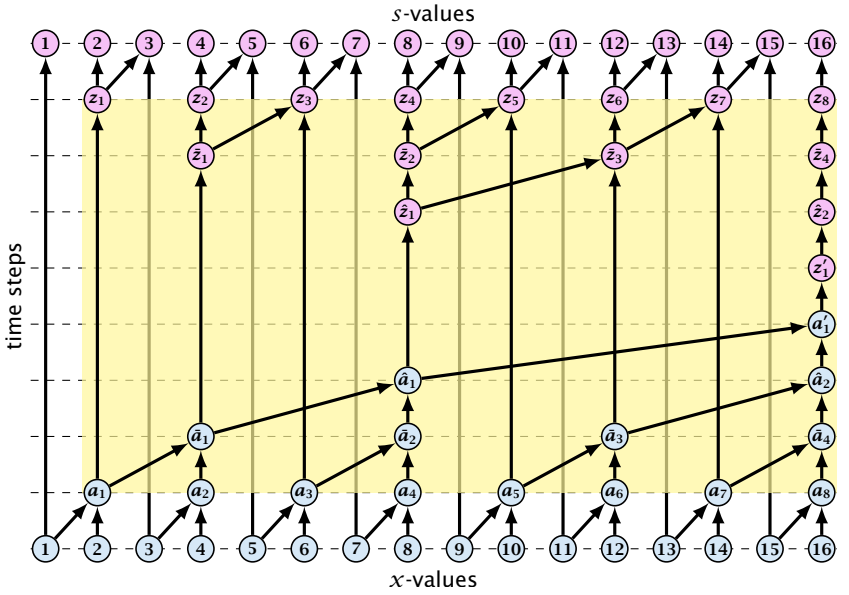
Prefix Sum

input: $x[1]\dots x[n]$
output: $s[1]\dots s[n]$ with $s[i] = \sum_{j=1}^{i} x[i]$ (w.r.t. operator $*$)

Algorithm 6 PrefixSum$(n, x[1]\dots x[n])$

1: // compute prefixsums; $n = 2^k$
2: **if** $n = 1$ **then** $s[1] \leftarrow x[1]$; **return**
3: **for** $1 \le i \le n/2$ **pardo**
4: $\quad a[i] \leftarrow x[2i-1] * x[2i]$
5: $z[1],\dots,z[n/2] \leftarrow$ PrefixSum$(n/2, a[1]\dots a[n/2])$
6: **for** $1 \le i \le n$ **pardo**
7: $\quad i$ even $\quad : s[i] \leftarrow z[i/2]$
8: $\quad i = 1 \quad : s[1] = x[1]$
9: $\quad i$ odd $\quad : s[i] \leftarrow z[(i-1)/2] * x[i]$

Prefix Sum



Prefix Sum

The algorithm uses work $\mathcal{O}(n)$ and time $\mathcal{O}(\log n)$ for solving Prefix Sum on an EREW-PRAM with n processors.

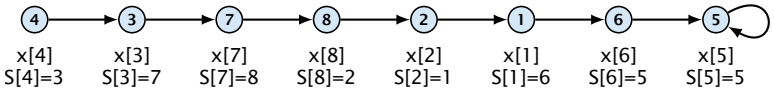
It is clearly work-optimal.

Theorem 1
On a CREW PRAM a Prefix Sum requires running time $\Omega(\log n)$ regardless of the number of processors.

Parallel Prefix

Input: a linked list given by successor pointers; a value $x[i]$ for every list element; an operator $*$;

Output: for every list position ℓ the sum (w.r.t. $*$) of elements after ℓ in the list (including ℓ)



Parallel Prefix

Algorithm 7 ParallelPrefix

```
1: for 1 ≤ i ≤ n pardo
2:     P[i] ← S[i]
3:     while S[i] ≠ S[S[i]] do
4:         x[i] ← x[i] * x[S[i]]
5:         S[i] ← S[S[i]]
6:     if P[i] ≠ i then x[i] ← x[i] * x[S(i)]
```

The algorithm runs in time $\mathcal{O}(\log n)$.

It has work requirement $\mathcal{O}(n \log n)$. non-optimal

This technique is also known as pointer jumping

4.3 Divide & Conquer — 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 2
Let $X = (x_1, \ldots, x_t)$ be a sequence. The rank $\text{rank}(y : X)$ of y in X is

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

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

4.3 Divide & Conquer — Merging

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

Observation:
We can assume wlog. that elements in A and B are different.

Then for $c_i \in C$ we have $i = \text{rank}(c_i : A \cup B)$.

This means we just need to determine $\text{rank}(x : A \cup B)$ for all elements!

Observe, that $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$.

Clearly, for $x \in A$ we already know $\text{rank}(x : A)$, and for $x \in B$ we know $\text{rank}(x : B)$.

4.3 Divide & Conquer — Merging

Compute $\text{rank}(x : A)$ for all $x \in B$ and $\text{rank}(x : B)$ for all $x \in A$.
can be done in $\mathcal{O}(\log n)$ time with $2n$ processors by binary search

Lemma 3
On a CREW PRAM, Merging can be done in $\mathcal{O}(\log n)$ time and $\mathcal{O}(n \log n)$ work.

not optimal