Theorem (Valiant 1981)

For general keys and deterministic algorithms;

Finding the max of \( n \) numbers using \( n \) processors needs \( \Omega(\log \log n) \) time.

A parallel comparison tree was used by Valiant to prove this theorem. A parallel comparison tree only accounts for the number of comparisons made and hence it is a model that is more powerful than the PRAMs. As a result, the same lower bound readily applies on any of the PRAM models as well.

Fact:

Finding the max of \( n \) integers in the range \([1, n^c]\) can be done in \( O(1) \) time using \( n \) Common CRCW PRAM processors, where \( c \) is any constant.

Fact:

We can find the max of \( n \) elements in \( O(\log \log n) \) time using \( \frac{n}{\log \log n} \) Common CRCW PRAM processors.

LEMMA:

We can find the max on \( n \) elements in \( \tilde{O}(1) \) time using \( n \) arbitrary CRCW PRAM processors.

Proof:

Input:

\[ X = k_1, k_2, k_3, \ldots, k_n \]

Idea:

1) Pick a random sample \( S \) of size \( \sqrt{n} \). \( O(1) \) Time
2) Find the max \( M \) of this sample. \( O(1) \) Time
3) for \( 1 \leq i \leq n \) in parallel do
   if \( k_i < M \) then delete \( k_i \);
   The number of surviving keys is \( \tilde{O}(\sqrt{n \log n}) = \tilde{O}(n^{0.51}) \)
4) Find and output the max of the surviving keys using the following fact.

Fact:
We can find the max of \( n \) elements in \( O(1) \) time using \( n \sqrt{n} \) processors.

**Idea:**

\[ k_1, k_2, \ldots, k_{\sqrt{n}} \quad k_{\sqrt{n} + 1}, \ldots, k_{2\sqrt{n}} \quad \ldots \ldots \quad \ldots, \ldots, \ldots, k_n \]

\[ M_1 \quad M_2 \quad M_{\sqrt{n}} \]

**A problem:**

We have to collect the surviving keys and write them in successive cells so that we can proceed with step 4.

We’ll place the surviving keys in a region of size \( n^{2/3} \) so that each cell in this region will have at most one surviving key.

**A Round:**

a) Each processor with a surviving key picks a random cell \( j \);
b) The processor reads from \( j \) and if \( j \) is occupied, it waits for the next round.
c) If the cell \( j \) is empty, the processor tries to write its key in \( j \);
d) The processor reads from \( j \);
e) If \( j \) has its key, the processor is done; otherwise, it waits for the next round;

**Placement algorithm:**

\[ \text{REPEAT} \]

Each processor with a live key participates in a Round.
UNTIL all the keys are placed.

Analysis:

![Diagram](attachment:image.png)

# of elements that will be placed = $O(n^{0.51})$

In any given round, the probability that a processor does not succeed is $\leq \frac{n^{0.51}}{n^{2/3}} = O(n^{-0.15})$

∴ Probability of failure in $c \propto$ successive rounds is $\leq (n^{-0.15} c \propto)$

R.H.S $\leq n^{-\infty}$ if $c \geq \frac{1}{0.15} = \frac{20}{3}$

Prefix Computation

**Input:**

$$k_1, k_2, k_3, \ldots, k_n \in \Sigma$$

**Output:**

$$K_1, K_1 \oplus K_2, K_1 \oplus K_2 \oplus K_3, \ldots \ldots \ldots, K_1 \oplus K_2 \ldots \ldots \oplus K_n$$

Where $\oplus$ is any binary, associative, and unit time operation.