POWER OF CRCW – Minima

Q Find minima of n elements.

Seq. algo → $O(n)$

Parallel = use upward traversal,
Time = $O(\log n)$
Work = $O(n)$

(★) We can do better ∈ CRCW :

① → First start with $O(n^2)$ work & $O(1)$ time
② → Then sacrifice some time to gain optimality

① $O(1)$ time algo

a. Build a matrix $(n \times n)$ ← at every cell there is a processor. so n^2 processors

b. Each processor ~~compa~~ at i, j compare $A[i]$ ∈ $A[j]$
if $A[i] > A[j]$ put 1 in cell
else put 0 in cell.

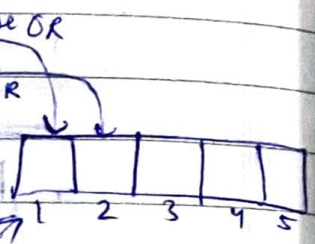
c. There will be some row ∈ all zero — ~~exceptions~~
so, that $A[\bullet i][0]$ will be minima
To find such row with all zeroes :

not O(1) ↑ difficult

① sum of row is 0 →
OR ② Bitwise OR → result = 0

So,

1	0	1	0	0	1
2	1	0	1	0	1
3	0	0	0	0	0
4	1	1	1	1	1
5	0	0	1	1	0



uses CRCW O(1) time

Bitwise OR:
For this we put S = 0 initially
↳ if B[i] = 1 write S = 1
So, we will have one S for each row

Total work = $O(n^2)$

Lecture 17: 12th March 2021

So, to reduce work we find a algo with:

W – $O(n^2)$ ⟶ $O(n \log\log n)$ ← non optimal

T – $O(1)$ ⟶ $O(\log\log n)$

start ← small problem size

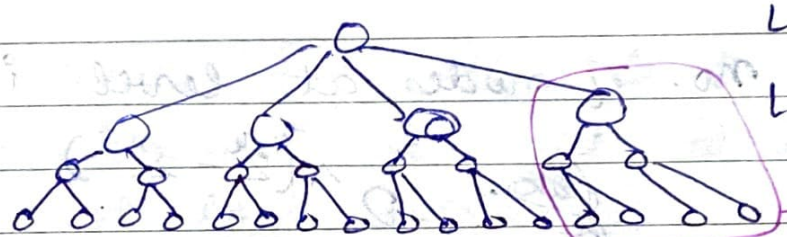
optimal

Doubly Logarithmic Tree

n = size of array = no. of element

lets assume $n = 2^{2^k}$ $(k \in \mathbb{Z})$ integer

So → in our tree we have **n leaves** and root is at level zero



→ k+2 levels totally.

→ Root has $\sqrt{n}$ children

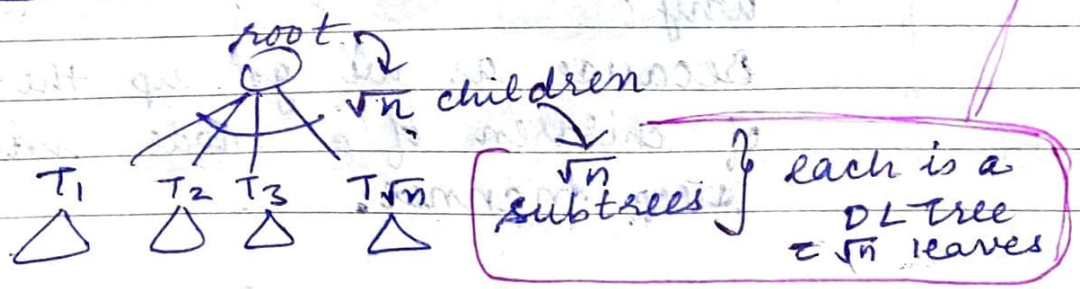
$$\sqrt{n} = \sqrt{2^{2^k}} = 2^{2^{k-1}} \text{ children.}$$

→ So in general, at level i, $2^{2^{k-i-1}}$ children

for $0 \le i \le k-1$

And each node at level k will have two leaf nodes as children.

Another way to think about this



each
so $\sqrt{n}$ subtrees, $\sqrt{n}$ leaves
so, n leaf nodes.

Some claims :

→ nodes at k^{th} level is $n/2$
(level above leaf nodes)

→ no. of nodes at level i is

$$2^{(2^k - 2^{k-i})}$$

→ depth of DLT of n leaf nodes is
$k+1 = \log\log n + 1$

proof :

$n = 2^{2^k}$

$\log n = 2^k \log 2$

$\boxed{\log\log n = k}$

COMPUTING MINIMA

- Each internal node performs min operation does not suffice.
  Why?
  Because as we go up the tree the no. of children of a node increase in a steep manner.

$\sqrt{n}$ degree at root

$n^{1/4}$ degree

$n^{1/8}$ degree

and any node with a large no. of children will not be able to find minima of its children in $O(1)$ time.

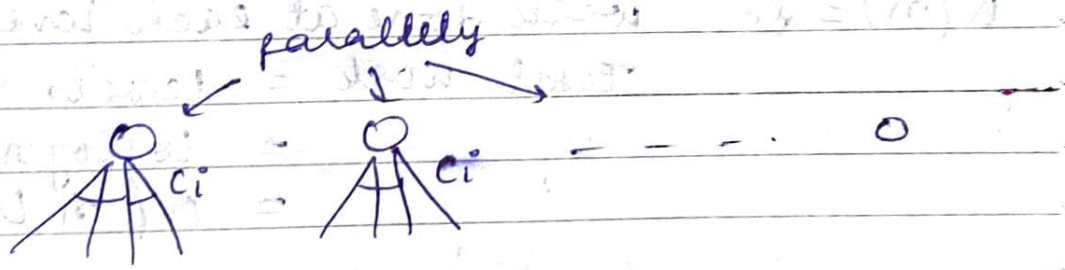
→ DLT has $\log \log n$ levels. So to get $O(\log \log n)$ time we need to spend only $O(1)$ time at each level.

So, at each intenal node at level i, if there are c_i children

$\downarrow$

c_i^2 processors

$O(1)$ time at each level



∴ no. of nodes at level $i = n_i$

So, total processors at each level $\Rightarrow n_i \times c_i^2$

$= n_i \times c_i^2$

$= 2^{2^k - 2^{k-i}} \times \left(2^{2^{k-i-1}}\right)^2$

$= 2^{2^k - 2^{k-i} + 2^{k-i-1} + 2^{k-i-1}}$

$= 2^{2^k(1-2^{-i}) + 2 \times 2^{k-i-1}}$

$= 2^{2^k(1-2^{-i}) + 2^{k-i}}$

$= 2^{2^k}$

$$= 2^{2^k - 2^{k-i}} \times \left(2^{2^{k-i-1}}\right)^2$$

$$= 2^{2^k - 2^{k-i}} \times 2^{2^{k-i-1}} \times 2^{2^{k-i-1}}$$

$$= 2^{2^k - 2^{k-i}} \times 2^{2^{k-i-1} + 2^{k-i-1}}$$

$$= 2^{2^k - 2^{k-i}} \times 2^{2 \times 2^{k-i-1}}$$

$$= 2^{2^k - 2^{k-i}} \times 2^{2^{k-i}}$$

$$= 2^{2^k - 2^{k-i} + 2^{k-i}}$$

$$= 2^{2^k}$$

$$= n$$

So, # processors at level $i = n$

$N(n)$ = So, work done at each level $i = O(n)$

Total work = levels × W at each level

$= \log\log n \times O(n)$

$= O(n \log\log n)$

(n) = Time at each level $= O(1)$

Total time = level × T at each level

$= O(\log\log n)$

p = Each level uses n processors.

$$= 2^{2^k - 2^{k-i}} \times \left(2^{2^{k-i-1}}\right)^2$$

$$= 2^{2^k - 2^{k-i}} \times 2^{2^{k-i-1}} \times 2^{2^{k-i-1}}$$

$$= 2^{2^k - 2^{k-i}} \times 2^{2^{k-i-1} + 2^{k-i-1}}$$

$$= 2^{2^k - 2^{k-i}} \times 2^{2 \times 2^{k-i-1}}$$

$$= 2^{2^k - 2^{k-i}} \times 2^{2^{k-i}}$$

$$= 2^{2^k - 2^{k-i} + 2^{k-i}}$$

$$= 2^{2^k}$$

$$= n$$

So, # processors at level $i = n$

$N(n)$ = So, work done at each level $i = O(n)$

Total work = levels × W at each level

$= \log\log n \times O(n)$

$= O(n \log\log n)$

$T(n)$ = Time at each level $= O(1)$

Total time = level × T at each level

$= O(\log\log n)$

p = Each level uses n processors.

Still, not optimal.

Till now, 2 algos.

Algo 1 = Binary Tree (slow but optimal)
T = $O(\log n)$ W = $O(n)$

Algo 2 = DLT (fast but non-op)
T = $O(\log\log n)$ W = $(n \log\log n)$

So, to go from non op to op —
starting ε smaller problem size doesn't work.
we are able to
① solve a smaller problem of roughly $\frac{n}{\log\log n}$ elements but

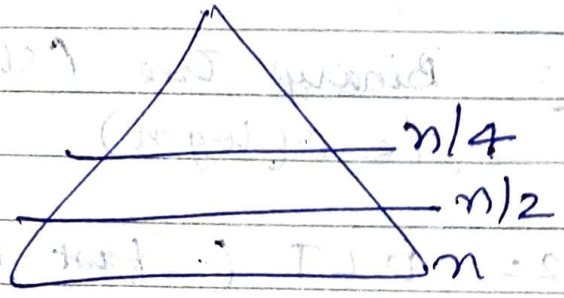
we aren't able to —
② extend soln to entire problem

So, we use a new technique —

Accelerated Cascading
↳ combine 2 algo for same problem and arrive at optimal

① → start ε slow but opt. algo till problem is small enough.

So, eg in binary tree as we go up problem size ↓es



② Then, we switch over to fast but non-optimal algo.

→ Binary Tree algo — start ⊂ size n input

In each level up, tree reduces size of input by 2

After log log log n levels,

size of input reduces to

$$\frac{n}{2^{\log\log\log n}} = \frac{n}{\log\log n}$$

We stop here, because our algo 2 is non-optimal by a factor of log log n.

So now we can use algo 2 when input is $\frac{n}{\log\log n}$ at this point