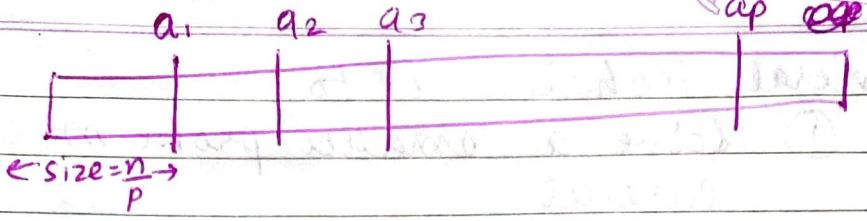
Parallel Search

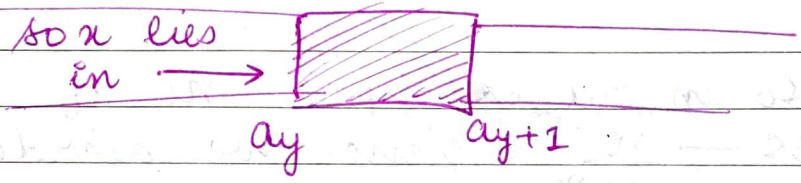


Search 'x'

(1) Check $x \to \in a_1, a_2, \ldots a_p$ in parallel using p processors.

(2) So we will know which subproblem to solve as we'll get —

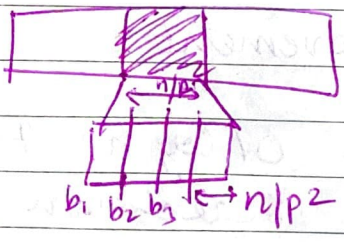
$$x < a_y \quad \& \quad x > a_{y+1}$$



So, in BS — $p = 2$

(4) So size of subproblem $= \frac{n}{p}$

(3) So, $\in \frac{n}{p}$ elements & p processors we further divide into p subproblems



Again, we check $x \in b_1, b_2 \ldots b_p$

Still making 'p' comparisons

④ Continue recursively, Eventually subproblems will have size $\leq p$ at which point we can look at each element in parallel & give answer.

Time taken by parallel search:

Size of subpr $\Rightarrow$ $n, \frac{n}{p}, \dots, p$

$$T_m = ar^{m-1}$$

$$p = n \times \left(\frac{1}{p}\right)^{m-1}$$

$$p^m = n$$

$$m = \frac{\log n}{\log p} = \log_p n$$

So, p comparisions in parall in each subproblem & $\frac{\log n}{\log p}$ subproblems

$$T(n) = T\left(\frac{n}{p}\right) + O(1)$$ ← p comparison in parallel

$$\boxed{T(n) = O(\log_p n)}$$

Work Complexity

$$W(n) = W\left(\frac{n}{p}\right) + p = \boxed{O(p \log_p n)}$$

So, time is decreasing logarithmically in p but work is increasing linearly in p.

So what 'p' to use?

	$W(n) = O(p \log_p n)$	$T(n) = O(\log_p n)$
if $p = \sqrt{n}$	$W(n) = O(\sqrt{n})$ ↑↑	$T(n) = O(1)$ ↓↓

So if $p = \sqrt{n}$, we're spending $O(\sqrt{n})$ per search, which is a lot. This is overkill in no. of processors.

if $p = 1$: $W(n) = O(\log n)$ | $T(n) = O(\log n)$

↳ same as binary search

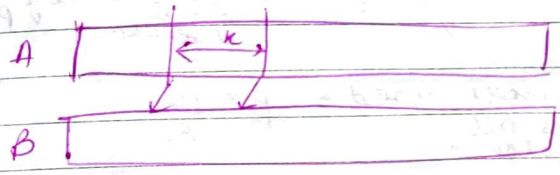
$p = \log n$: $W(n) = O\left(\frac{\log^2 n}{\log \log n}\right)$ | $T(n) = \frac{\log n}{\log \log n}$

★ Tradeoff b/w $W(n)$ & $T(n)$

if $p = n^{\varepsilon}$ $\quad T(n) = O(1)$, $\quad W(n) = O(n^{\varepsilon})$

Lecture 16 : 9th March

From parallel search to merge



using parallel search, $\leq \sqrt{n}$ processors

① Rank $\sqrt{n}$ elements of A into B using parallel search

i.e. $k = \sqrt{n}$

② ↳ for every search use $\sqrt{n}$ processors.

Time : $\log_2 n = O(1)$

Work :

for complete step = no. of processors used =

i.e. for all searches $\sqrt{n} \times \sqrt{n}$ ← searches

$\sqrt{n}$ processors in one search

∴ # processor = n

∴ Work for all searches $= O(1) \times n = O(n)$

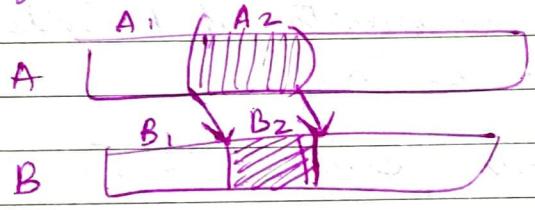
So, generally, using || search

Rank $\frac{n}{k}$ elements of A into B

Work across all searches $= \frac{n}{k} \times p$ Time across all searches $= \log_p n$

Processors used across all searches $= p \times \frac{n}{R}$

(2) Merge in sequential manner



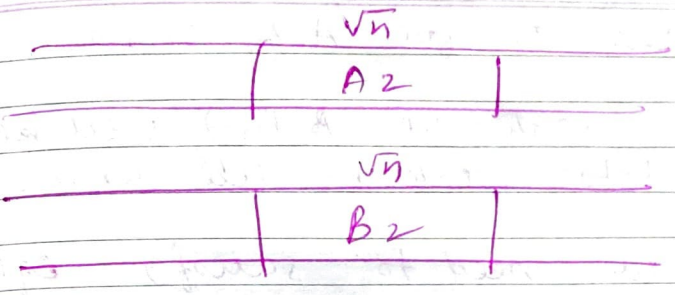
Time $= \begin{cases} O(1) \text{ for step 1} \\ O(|A_2| + |B_2|) \\ = O(\sqrt{n} + ?) \simeq O(\sqrt{n}) \end{cases}$ step 2

So, if we reduce no. of searches, no. of elements to merge go up and vice versa.

So, how to fix this?
Reduce time from $O(\sqrt{n})$ in merging

(Size of B can be reduced from ? to $\sqrt{n}$ as done in beg. of Lec (5))

Goal = Merge time t (depends on) & length $|A_2|$ $|B_2|$



if we again divide $A \& B$. —

Choosing size of subproblem :

So, we rank t elements of A_2 in B_2

$\Rightarrow \sqrt{n}$ processors

So, ~~time = $O(1)$ per search~~

processes per search $= \frac{\sqrt{n}}{t}$

$\longrightarrow$ time $= O(\log_p n)$

$= O\left(\log_{\frac{\sqrt{n}}{t}} \sqrt{n}\right)$

$= O\left(\frac{\log \sqrt{n}}{\log \sqrt{n} - \log t}\right)$

& we want

$O\left(\frac{\log \sqrt{n}}{\log \sqrt{n} - \log t}\right) = O(1)$ ——— eq (i)

$\longrightarrow$ work : $= O(p \log_p n)$ per search

$= \frac{\sqrt{n}}{t}$

Total work $= \frac{\sqrt{n}}{t} \times t = \sqrt{n}$

~~So, total work is still same~~

So work for A_2B_2 is $\sqrt{n}$

Now,

total work (A_1A_2, A_2B_2) will be $O(n)$

So, total work is still same.

But, we need to satisfy eq(i)

So, which t to pick?

$$\frac{\log \sqrt{n}}{\log \sqrt{n} - \log t} = O(1)$$

~~$\log \sqrt{n} = \log \sqrt{n} - \log t$~~

~~$\log t = 0$~~

~~$t \geq 1$~~

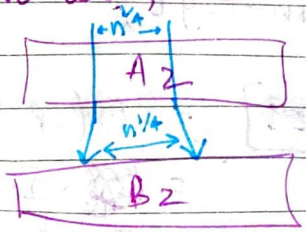
$$\frac{\log \sqrt{n}}{\log \sqrt{n} - \log t} = 2$$

$$\log \sqrt{n} = 2 \log t$$

$$t^2 = \sqrt{n}$$

$$t = \sqrt[4]{n} = n^{\frac{1}{4}}$$

But even here, $\because t = n^{\frac{1}{4}}$



merge time only went down from $n^{\frac{1}{2}}$ to $n^{\frac{1}{4}}$

So, → again divide?

$$n \to \sqrt{n} \to \sqrt{\sqrt{n}} \to n^{1/8} \to n^{\frac{1}{16}} \to O(1)$$

steps = i where

$$n^{\frac{1}{2^i}} = O(1)$$

$$n^{\frac{1}{2^i}} = 2$$

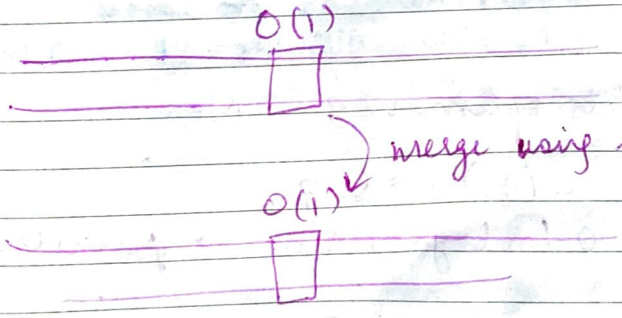
$$\frac{1}{2^i} \log n = \log 2$$

$$\log n = 2^i$$

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

$\Rightarrow$ # steps $= O(\log \log n)$

So, we do this recursively for $O(\log \log n)$ stages. At the end of these stages,



So, Total time →

t searches done in parallel, each $\leq$ O(1) time

$\Rightarrow$ O(1) time in each stage

$\Rightarrow$ So, O(log log n) time totally.

Total work for each search:

work per stage =

no. of searches in stage 1 = $\sqrt{n}$

work in each search = $\sqrt{n}$

So, work in stage 1 = O(n)

searches in stage 2 = $n^{\frac{1}{4}} = t$

work in each search = $\frac{\sqrt{n}}{t} = n^{\frac{1}{4}}$

So, work in stage 2 = O(n)

So,

log log n stages each $\leq$ O(n) work

$\Rightarrow$ O(n log log n) work.

if we do not have n processors but p, to simulate n processor $\leq$ p, time will blow up by a factor of n/p

Is this optimal? No.

O(n) $\rightarrow$ seq.

O(n log log n) $\longrightarrow$ parallel

Home-work

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

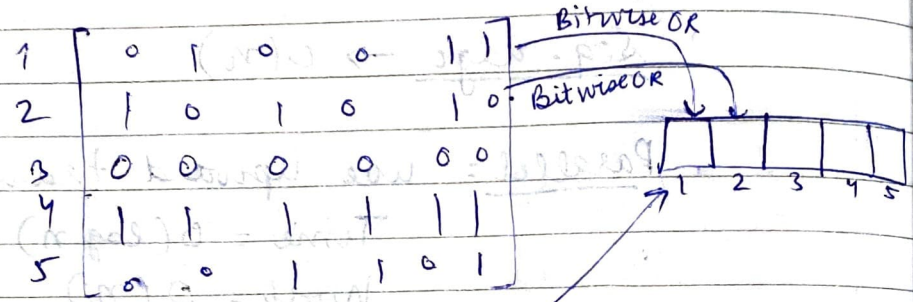
c. There will be some row ∈ all zero — ~~ex @ p d i a g o n a l s~~
so, that $A[\bullet i][0]$ will be minima
To find such row with all zeroes :

Date
Page

not O(1)
Typical

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

So,



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

uses CRCW
O(1) time

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 ≤ smaller problem size

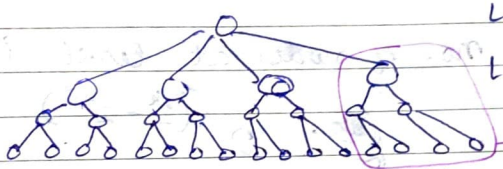
optimal

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



→ ~~exceed~~ $k+2$ levels totally.

→ Root has $\sqrt{n}$ children

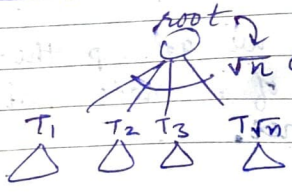
$$\sqrt{n} = \sqrt{2^{2^k}} = 2^{2^{k-1}} \text{ children.}$$

→ 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



So $\sqrt{n}$ subtrees, each $\sqrt{n}$ 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 :
$n = 2^{2^k}$
$\log n = 2^k \log 2$
$\log\log n = k$

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.

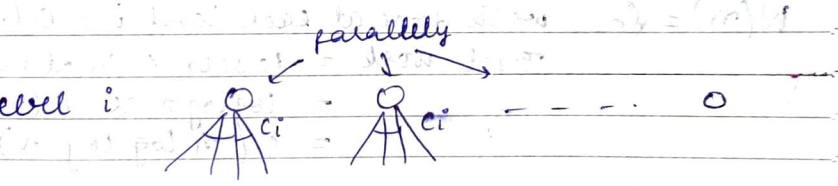
→ DLT has $\log \log n$ levels. So to get $O(\log \log n)$ time we need to spend only $O(1)$ time at each level.

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

↓

c_i^2 processors

$O(1)$ time at each level



no. of nodes at level $i = n_i$

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

$= n_i \times c_i^2$

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

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

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

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

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

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

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

$$= n$$

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~~ 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

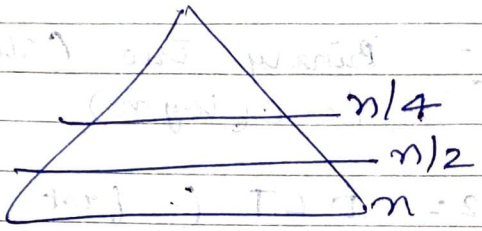
we aren't able to —
② extend solⁿ to entire problem

So, we use a new technique —

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

① → start ε slow ~~algo~~ 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.

~~Algo~~

→ Binary Tree algo – start ~~r~~ 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

W reduces to $O(n)$

As we switch to fast algo now, we need n' processors —

where $n' = \frac{n}{\log\log n}$

and $W(n') = O(n' \log\log n')$

$= O\left(\frac{n}{\log\log n} \log\log \frac{n}{\log\log n}\right)$

$= O(n)$

$\log\left(\log n - \log\log n\right) < \frac{1}{2}\log\log n$

Time = [~~$O(\log\log n)$~~ + $O(\log\log n)$]

$\hookrightarrow O(\log\log\log n) + O(\log\log n)$

BT algo 1 DL algo 2

So, time $= O(\log\log n)$