15
16
17
18
19

Date
Page

$A_{Parallel \to Seq}$ A_{seq}

if $T_{P \to S} = T_S$ ← work optimal

OTHER DESIGN PARADIGMS

Partitioning
└ similar to divide and conquer

D&C
- Divide
- Solve
- combine

Partitioning
- Divide
- Solve

eg. quicksort is example of sequential partitioning

In parallel algos, each subproblem that we get out of divide step can be treated independently & solved in parallel.
eg. Parallel merging, searching

MERGING IN PARALLEL

Two sorted arrays A and B to be merged into C.

Rank (x, A) = no. of elements smaller than x in A
(x ↓ ele, A ↓ sorted array)

Claim = $Rank(x, C) = Rank(x, A) + Rank(x, B)$ for every $x \in A \cup B$

For x in A, $Rank(x, A)$ = index of x in A (so immediately available)

To find $Rank(x, B)$ → [x is in A] — use binary search! (in parallel)

↓

for x in B

another x in B can be done in parallel

— because binary search is independent of another beriary search

```
Merge (A, B) {

    for each x in A {
        rx = BinaryS (x, B)
        C [rx + 1 + index(x, A)] = x
    }

    for each y in B {
        ry = BS (y, A)
        C [ry + 1 + index(y, B)] = y
    }
}
```

$|A| = n \quad |B| = n$

Time = $O(\log n)$

Work = $O(n \log n)$ — n processors

↙

Not work optimal

because ~~but~~ seq. time for merge of sorted array is $O(n)$

→ So Reduce total work to $O(n)$

① partition A into equal size pieces

② Take first element of every piece and Rank it in array B

③ Ranks of other elements of A will be b/w the two ranks in B

Soln:

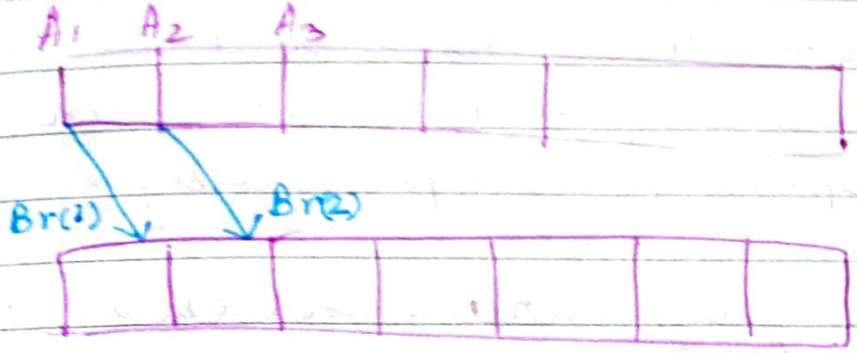
① $\log n$ elements in each partition of A

② partition B into $\log n$ element peices.

So $\frac{n}{\log n}$ partitions in A & B

$A_1, A_2 \cdots A_{\frac{n}{\log n}}$ — first element of each partition of A

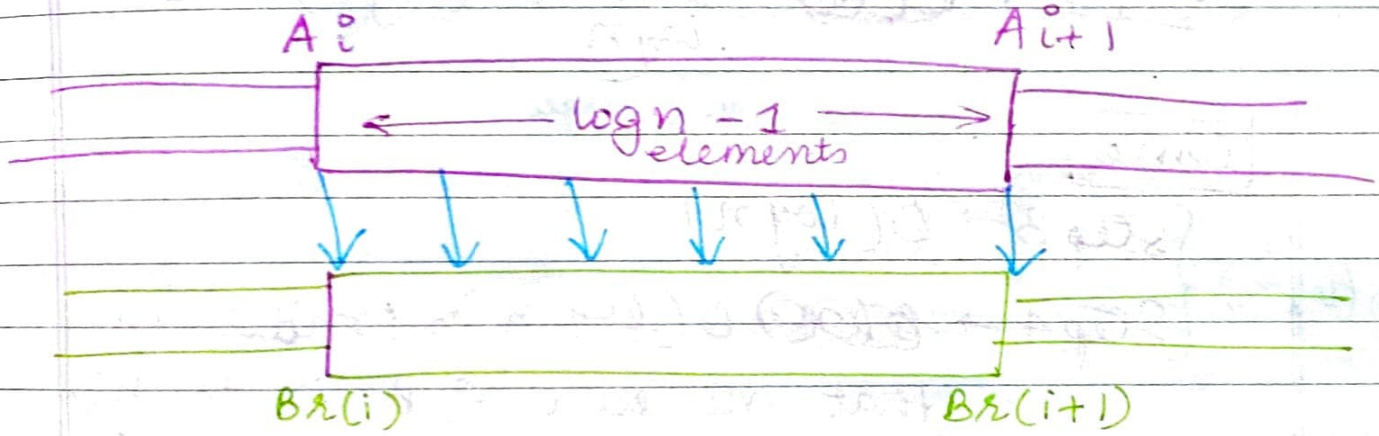
③ — and ~~rank~~ these elements are ranked in B. (using Binary Search) in parallel.



Br(i) is rank of A(i) in B

So, $A(i) \xrightarrow{\text{rank}} Br(i)$

$\Rightarrow$ A(i) to A(i+1) have ranks in Br(i) to Br(i+1)



(4) Now merge these two portions i.e. A(i) to A(i+1) and Br(i) to Br(i+1) sequentially [two-pointer]

↳ time taken for this = $O(\log n + B_{r(i+1)} - B_{r(i)})$

(5) different pieces of A can be merged sequentially with their corresponding rank pieces in B IN PARALLEL

So, all merges are happening in parallel but each merge is happening sequentially

Work: $\frac{n}{\log n}$ binary searches in parallel

$O(n)$ {
- step 3 → $\frac{n}{\log n}$
- $= \frac{n}{\log n} \times O(\log n) = O(n)$
- step 4 → $= \frac{n}{\log n} \times O(\log n) = O(n)$

($\frac{n}{\log n}$: no. of merges)

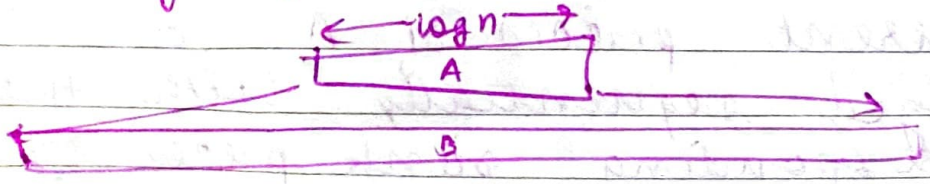
Time

$O(\log n)$ {
- step 3 → $O(\log n)$
- step 4 → $O(\log n)$ on conditions that no part of B is too big (since we are eliminating this $B[i+1] - B[i]$)

$O(\log n + (B[i+1] - B[i]))$

Lecture 15 : 5th March 2021

So, what if parts of B are of size more than $\log n$?

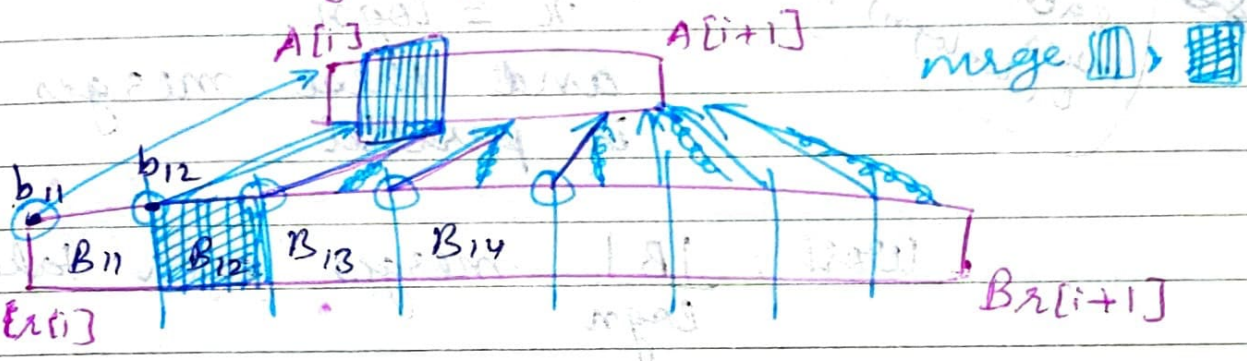


This can be solved =

We replace step 4 by this

(1) Partition each B_R part i.e. $[B_{R[i]} \ldots B_{R[i+1]}]$ into $\log n$ sized pieces.

(2) Rank (first element) each of these sub-parts of B into $\log n$-pieced A.



So, we are splitting the problem of Merge ($[A[i] \ldots A[i+1]$, $B_R[i]$ to $B_R[i+1]$) (Step 4) into subproblems:

(So if $B_{Ri} - B_{Ri+1}$ is too large, we further divide $B_{Ri} - B_{Ri+1}$ into $\log n$ parts and merge the other way round into A]

Is there any way to say that we're not creating too many subproblems? Let's count them -

$\frac{\log n}{\log \log n}$ merges

where each merge takes $O(\log \log n + \log n)$ time $\simeq O(\log n)$ time

⇒ time for merging A[i to i+1] & B

T Time : $O(\log n)$ since sequential merge of $\log n + x$ elements where B_{11} in $A_{i \to i+1}$

$x \leq \log n$.

and these merges happen in parallel.

searching each of $b_{11}, b_{12}, \ldots$ takes $O(\log \log n)$ time

Work : $\frac{|B|}{\log n}$ merges each take $O(\log n)$ time

$= O(|B|)$

So THE GENERAL TECHNIQUE for going from

Work non-optimal [Problem of size n]	→	Work optimal
$W(n)$		$B(n)$
eg. Merge : $W(n) = O(n \log n)$		$B(n) = O(n)$

General technique is to

① Solve a smaller problem in parallel

so, size of smaller problem = n'

such that

$$W(n') = B(n)$$

eg. $n' \log n' = n$

Merge →

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

So in our example, no. of BS — $\frac{n}{\log n}$ hence we divide array into $\frac{n}{\log n}$ pieces.

② Extend soln to entire problem. $n' \longrightarrow n$

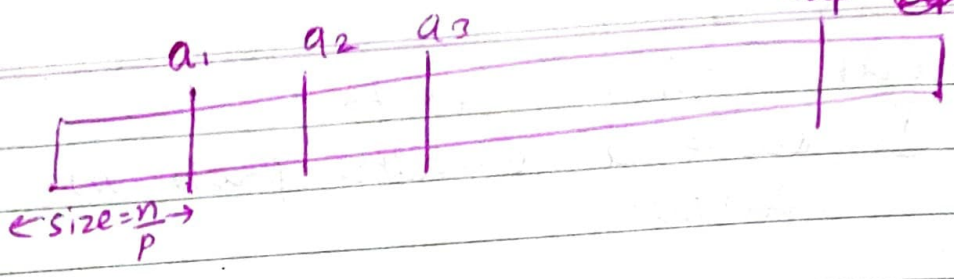
eg. Merge → use sequential merge

Further improvement:

BS takes $O(\log n)$ time.
Can we search in parallel?

PARALLEL SEARCH

Parallel Search

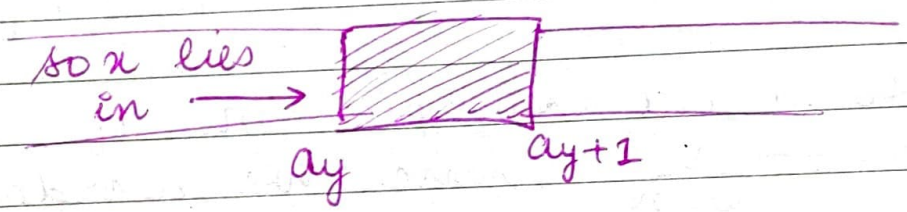


Search 'x'

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

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

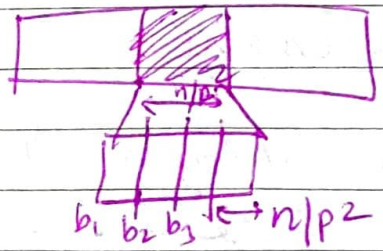
$$x < a_y \quad \& \quad x > a_{y+1}$$



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 p at which point we can look at each element in parallel & give answer.

Time taken by parallel search:

size of subpr ⇒ $n, \frac{n}{p}, \dots, p$

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

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

$$p^m = n$$

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

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

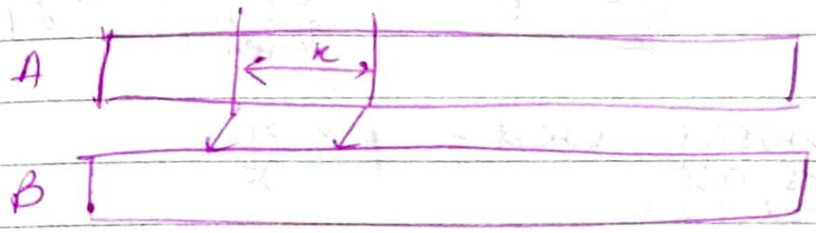
$p = \log n$: $W(n) = O\left(\frac{\log^2 n}{\log \log n}\right)$ | $T(n) = \frac{\log n}{\log \log n}$

(★) 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
↑ processors in one search

$\therefore$ # processor = n

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

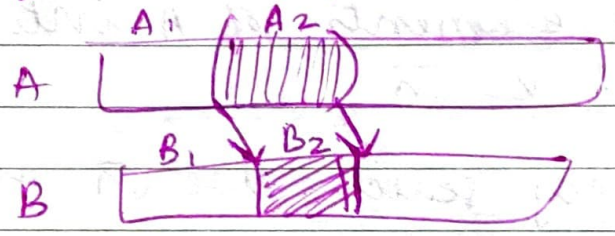
So, generally, using || search

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 $= p \times \frac{n}{k}$

(2) Merge in sequential manner



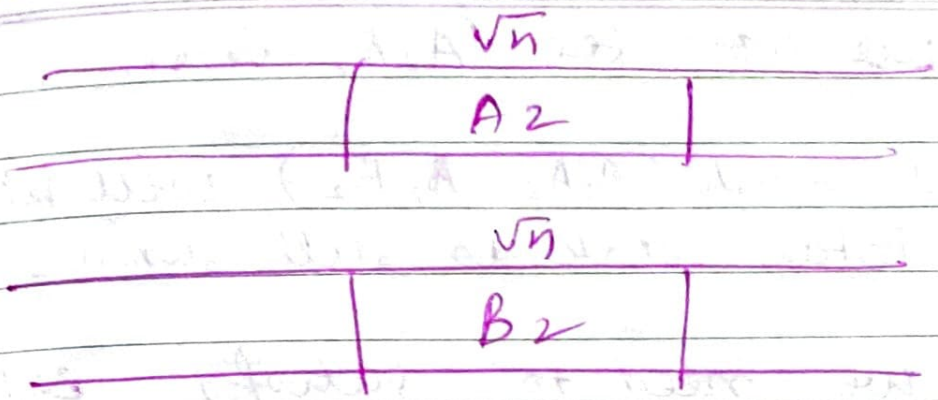
Time $= \begin{cases} O(1) \text{ for step 1} \\ O(|A_2| + |B_2|) \\ = O(\sqrt{n} + ?) \simeq O(\sqrt{n}) \end{cases}$ step 2

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 ~~total~~ 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?

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

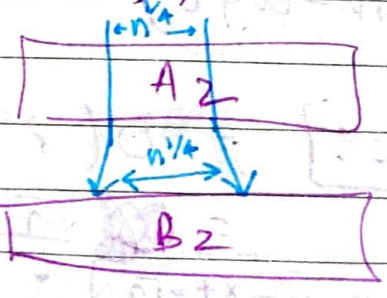
~~$\log \sqrt{n} = 2 \log \sqrt{n} - \log t$~~

~~$\log t = 0$~~
~~$t = 1$~~

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

$$\log \sqrt{n} = 2 \log t$$
$$t^2 = \sqrt{n}$$
$$t = \sqrt[4]{n} = n^{\frac{1}{4}}$$

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



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

So, → again divide?

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

steps = i where

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

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

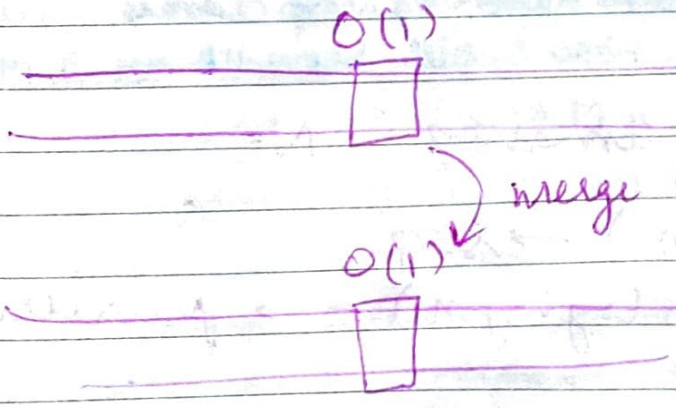
$$\frac{1}{2^i} \log n = \log 2$$

$$\log n = 2^i$$

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

$\Rightarrow$ # 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 in $O(1)$ 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 = $O(\sqrt{n})$

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