~~15~~
~~16~~
~~17~~
~~18~~
19

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

if $\boxed{T_{P \to S} = T_S}$ ← 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

Date
Page

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

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

To find $Rank(x, B) \rightarrow$ [$x$ 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 binary 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 best 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

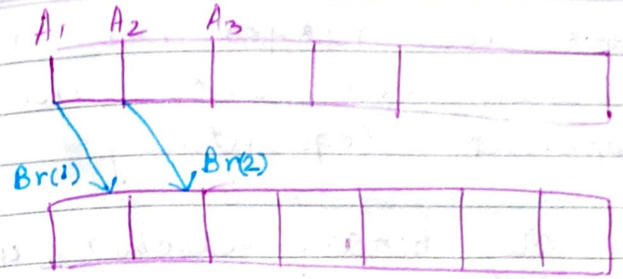
Soln:

1. $\log n$ elements in each partition of A
2. partition B into $\log n$ element pieces

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

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

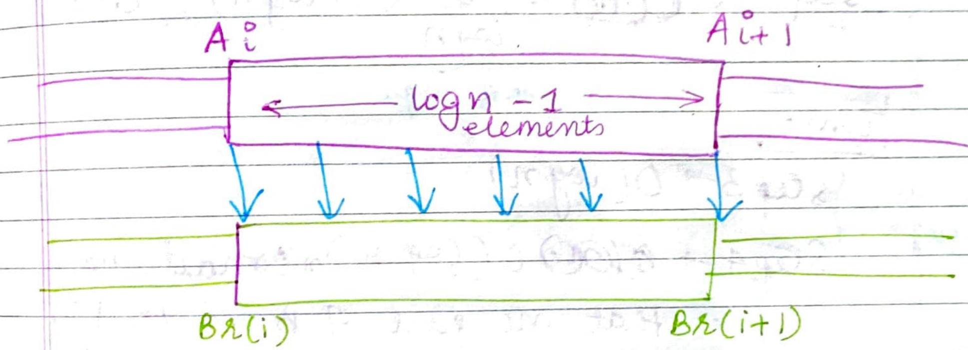
3. — and ~~[illegible]~~ these elements are ranked in B. (using Binary Search) in parallel




$Br(i)$ 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 = $O(\log n + Br(i+1) - Br(i))$

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

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

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

$O(n)$

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

step 4 → $= \underbrace{\frac{n}{\log n}}_{\text{no. of merges}} \times O(\log n) = O(n)$

$\boxed{\text{Time}}$

$O(\log n)$

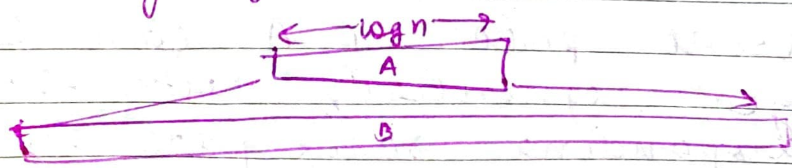
step 3 → $O(\log n)$

step 4 → $O(\log n)$ on conditions that no part of B is too big (since we are eliminating this

$O(\log n + \underbrace{(B_{r(i+1)} - B_{r(i)})}_{B})$

## Lecture 15 : 5th March 2021

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



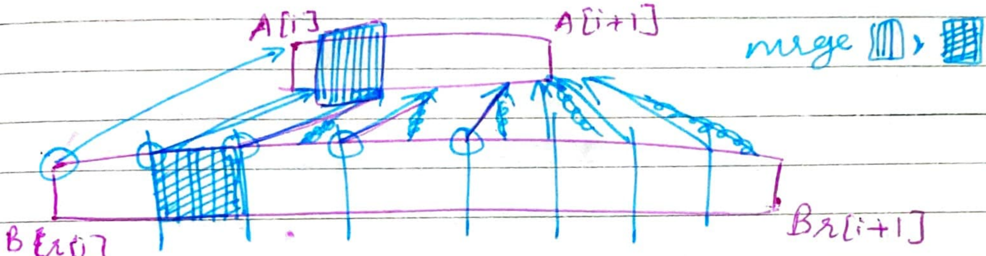

This can be solved =

We replace step 4 by this :

① Partition each B part i.e. [$B_{r[i]} \ldots B_{r[i+1]}$]

into log n sized pieces.

② Rank first element each of thes sub-parts of B into log n-pieced A.




So, we are splitting the problem of Merge([A[i] ... A[i+1]], Br[i] to Br[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 $O(\log\log n + \frac{\log\log n}{\log n})$ time $\approx O(\log n)$ time~~

$\Rightarrow$ time for merging $A[i \text{ to } i+1]$
$\propto B$

---

So THE GENERAL TECHNIQUE for going from

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