Merge Sort

Q⇒ Given 2 sorted arrays of different sizes, combine them such that the combined array is also sorted: (Merge two Sorted Arrays)

Ex → A → [1, 5, 7, 10]$_n$

B → [2, 3, 9, 12, 16]$_m$

input given

$1 \le n, m \le 10^7$

C → [1, 2, 3, 5, 7, 9, 10, 12, 16] output

final result is also sorted

How can we solve the question ??

Brute Force → Let's just follow the expected steps in the question-

1) Combine one array after the other and then sort them.

→ A → n
  B → m
} result → C → (n+m) length

(1) We can create a new array of size n+m.

(2) We will copy the data of array A first & then array B.

(3) Sort the array C.

A → [1, 5, 7, 10]$_n$

B → [2, 3, 9, 12, 16]$_m$

C → [1, 5, 7, 10, 2, 3, 9, 12, 16]$_{n+m}$

→ sort the array C → [1, 2, 3, 5, 7, 9, 10, 12, 16]

→ if we use selection / insertion / bubble → $O((n+m)^2)$

→ quick sort / heap sort / merge sort → $O((n+m)\log(n+m))$

How can we optimize ??

$A \rightarrow [1, 5, 7, 10]_n$

$B \rightarrow [2, 3, 9, 12, 16]_m$

→ if we try to figure out the smallest element of the final output, what could be the element ??

→ It will be one of the smallest elements of the given arrays.

we're done with this element, we don't need to consider it later.

A → [1, 5, 7, 10]$_m$

B → [2, 3, 9, 12, 16]$_n$

C[0] = min (A[0], B[0]) → A[0] → 1

Q) In the final out, which element will be the Second Smallest in the output (C[1]) ??

The 2 candidates are →

1) the element that did it become the smallest
2) the smallest element of remaining Array A.

C[1] = min (A[1], B[0]) → B[0] → 2

A → [1, 5, 7, 10]$_n$

B → [2, 3, 9, 12, 16]$_m$

→ In every iteration of this logic we can eliminate exact one element by just a comparison.

So, a comparison is a constant time operation

→ Complexity → O(n+m) → time O(1) → space

i

A → [1, 5, 7, 10]$_n$

B → [2, 3, 9, 12, 16]$_m$

j

Two Pointers

A [i .. n-1]

B [j .. m-1]

C = [1, 2, 3, 5, 7, 9, 10, 12, 16]$_{n+m}$

k

if (A[i] < B[j])

C[k] = A[i]

k++

i++

if (B[j] < A[i])

C[k] = B[j]

d++

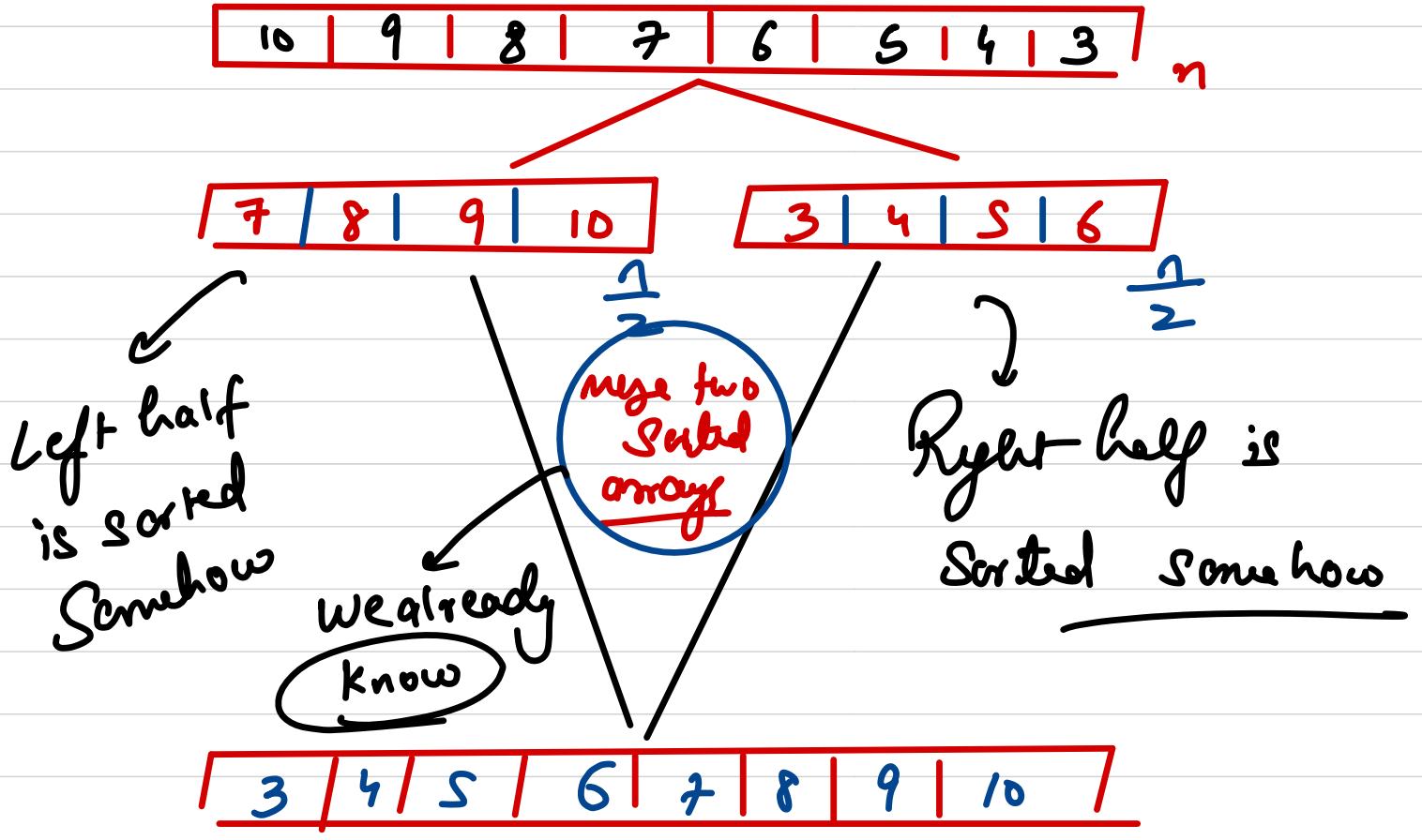
k++

→ if one of the arrays get exhausted, then all the remaining elements of the other array can be just directly inserted.

Time → O(n+m)

Space → O(1)

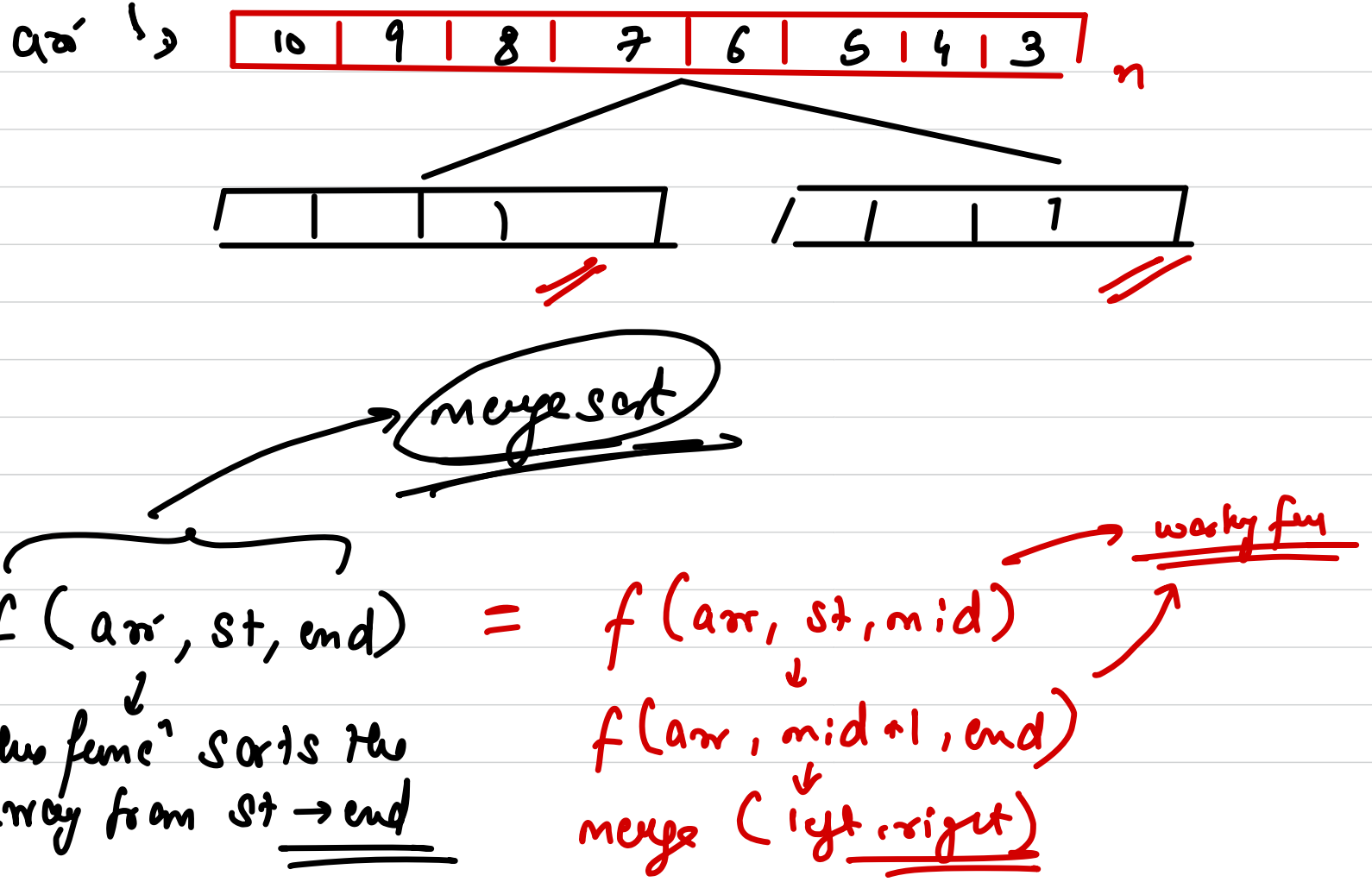
Q» Given an unsorted array, can we sort it, using the algorithm discussed previously.

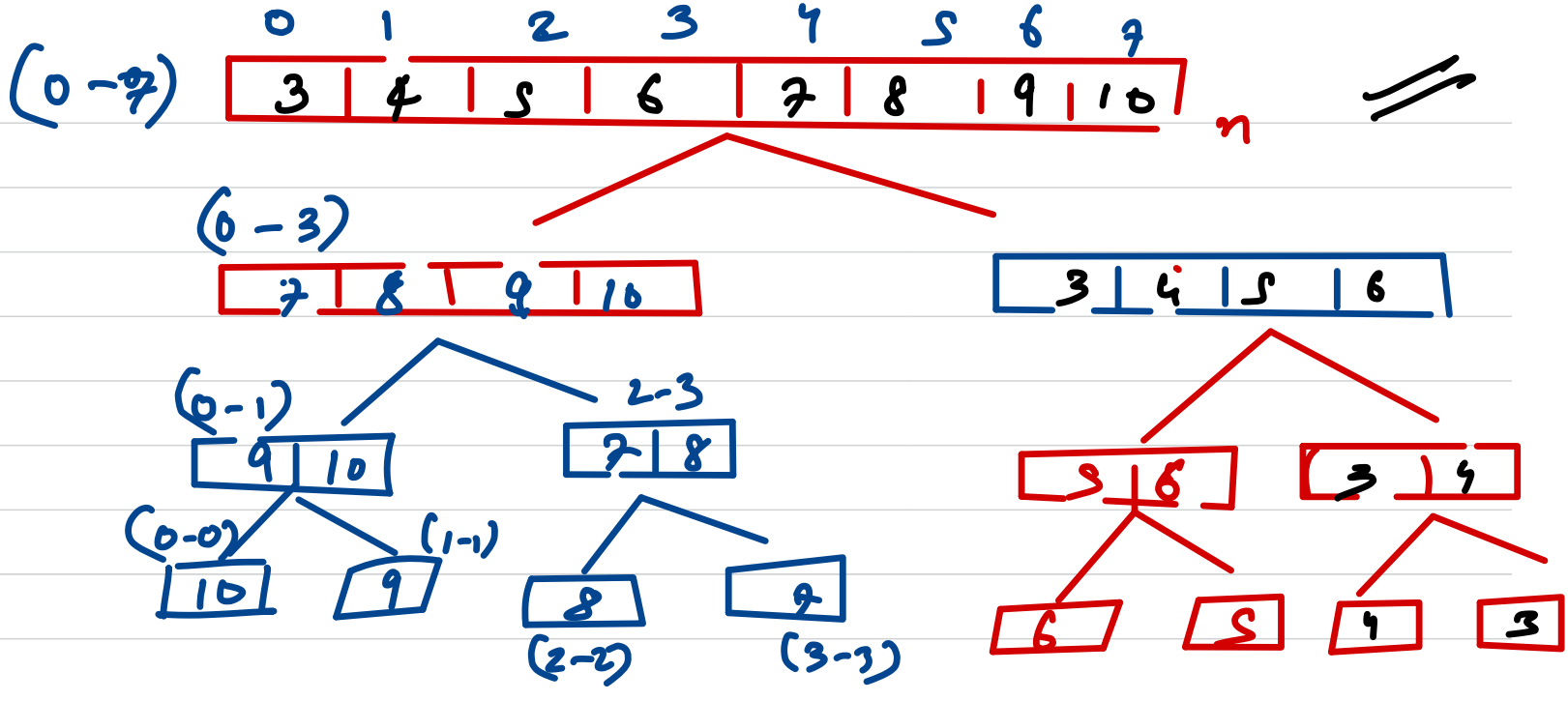

10 | 9 | 8 | 7 | 6 | 5 | 4 | 3
n
7 | 8 | 9 | 10
3 | 4 | 5 | 6
n/2
n/2
Left half is sorted somehow
merge two sorted arrays
Right half is sorted somehow
We already know
3 | 4 | 5 | 6 | 7 | 8 | 9 | 10

We will assume anyhow the first half and the second half is sorted then we can just merge both halfs with the new algo.

How the first and second half will be sorted ?:

↳ Recursion


arr 1>
10 | 9 | 8 | 7 | 6 | 5 | 4 | 3
n
mergesort
f (arr, st, end)
this fun^n sorts the array from st → end
=
f (arr, st, mid)
f (arr, mid+1, end)
merge (left, right)
working fun



algo of merge sort

levels	size	# of problems	work done

→ Merge Sort is a D&C algorithm.

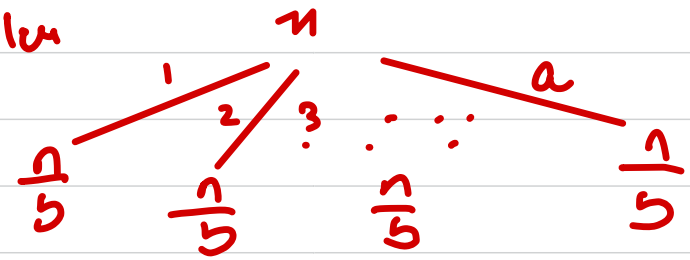
↓

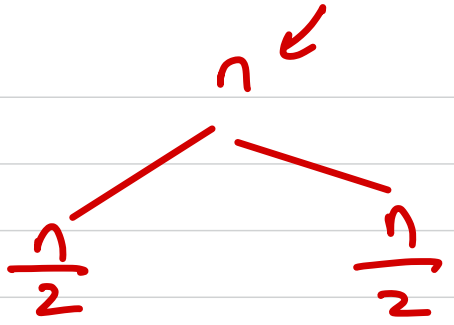
Divide and conquer

↓

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

- a → no of smaller subproblem
- $\frac{n}{b}$ → size of smaller subproblem
- $f(n)$ → operations reqd to merge answers of small sub





} 2 parts of n/2 size

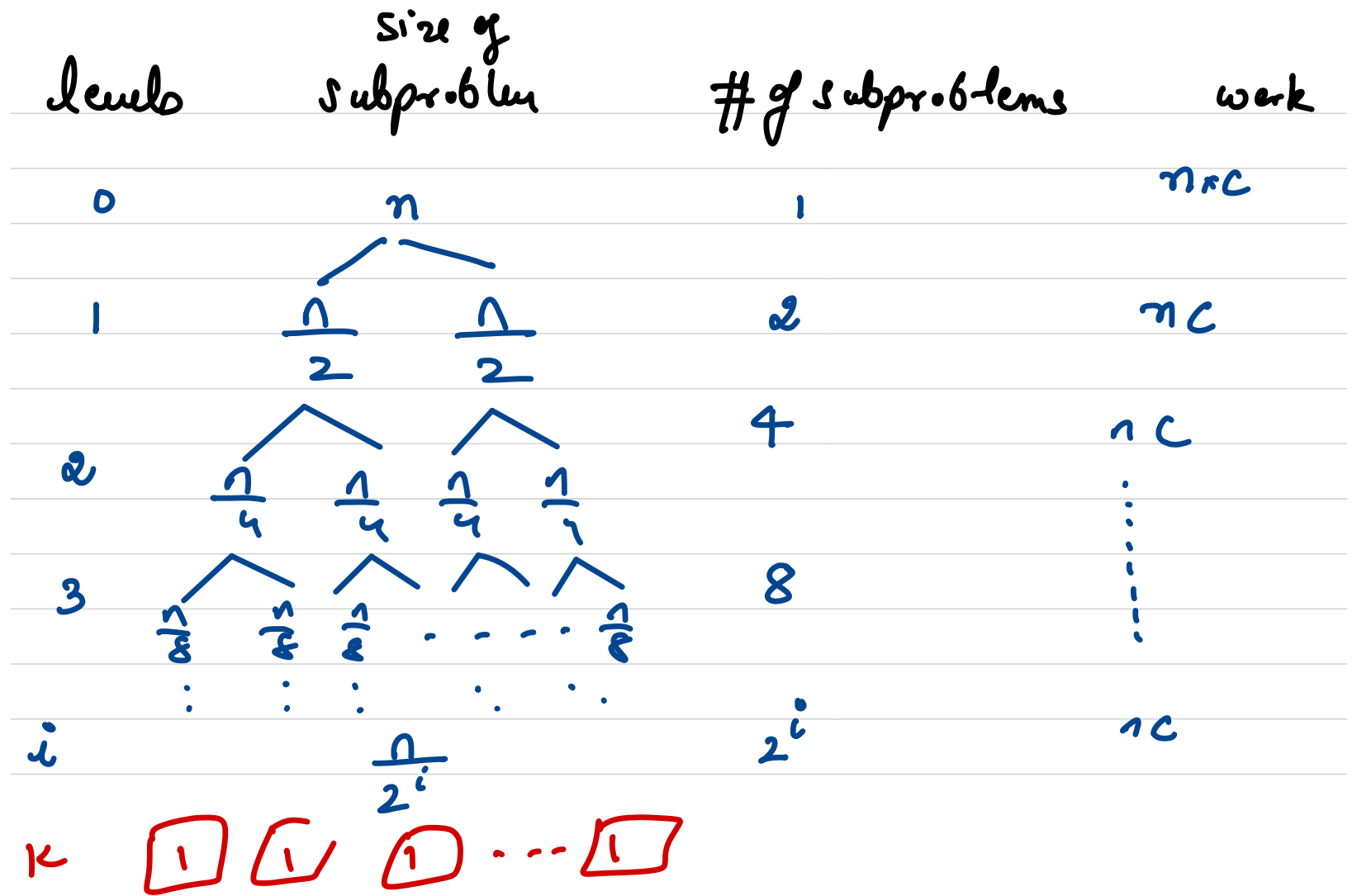
$T(n) = T(n/2) + T(n/2) + nc$

- $T(n)$: function that gives no. of operation for sorty an array of size n.
- $T(n/2)$: ops to sort left half
- $T(n/2)$: ops to sort right half
- nc: merge 2 halves

$T(n) = 2T\left(\frac{n}{2}\right) + nc$

Recurrence → $T(n) = 2T(n/2) + nc$

if we can solve this
then we can get no. of ops
to do merge sort:


Size of
levels
subproblem
# of subproblems
work
0
n
1
n*c
1
n/2
n/2
2
n*c
2
n/4
n/4
n/4
n/4
4
n*c
3
8
i
n/2^i
2^i
n*c
k
1
1
1
1

Total work done $= \sum_{i=0}^{i=\log n} nc = nc\log n$

$\downarrow$

Time $\rightarrow O(n \log n)$

What about space??

→ There are 2 type of spaces getting used →

→ 1) Call stack → O(log n) → +

2) auxillary arrays to merge →

0 1 2 3 4

(2, 3, 8, 6, 1)

$(i < j)$ and $A_i > A_j$

$(i=0, j=4)$ $2 > 1$

$(i=1, j=4)$ $3 > 1$

$(i=2, j=4)$ $8 > 1$

$(i=3, j=4)$ $6 > 1$

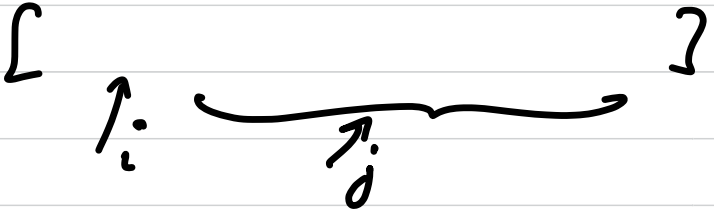
$(i=2, j=3)$ $8 > 6$

5 inversion pairs

Brute force → Generate all possible pairs & filter out the one which comply to inv pair.

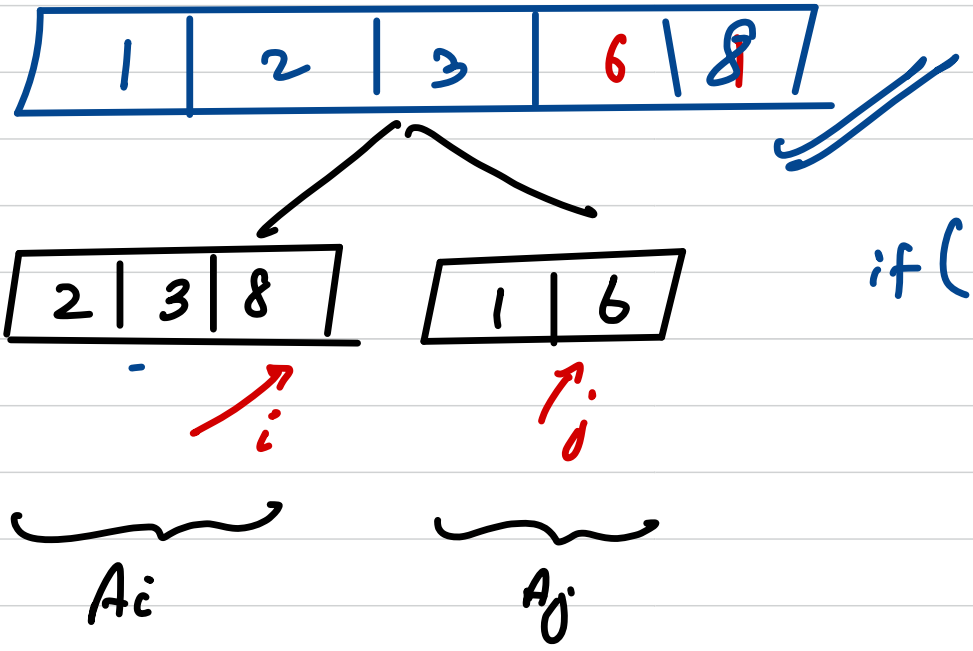
$O(n^2)$ → TLE

$n \leq 2 \times 10^5$



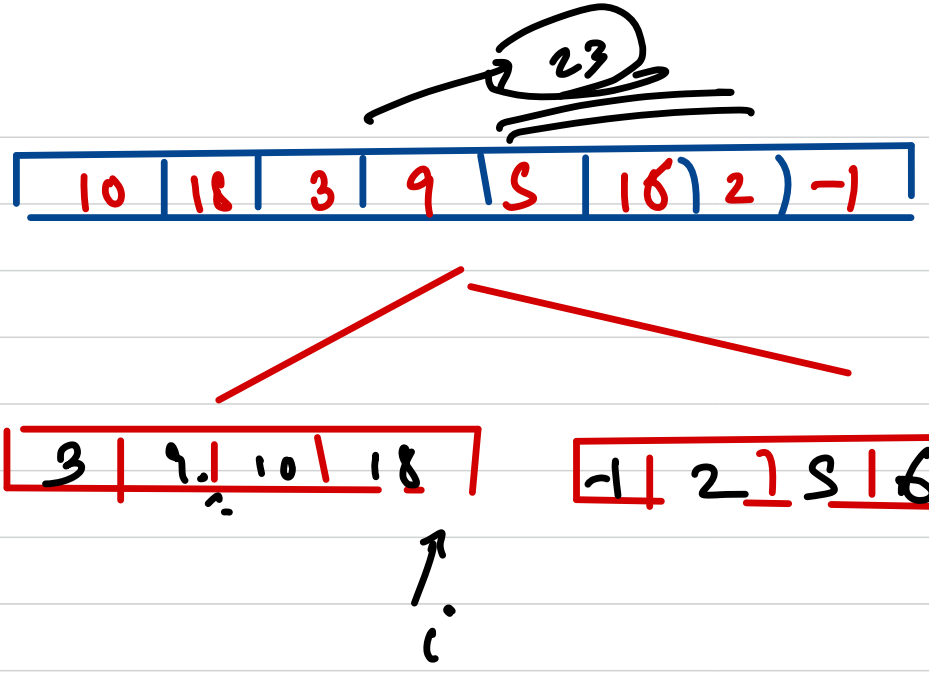
How to optimise ??

$i < j$ $A_i > A_j$



if (left[i] > right[j])

So while merging 2 sorted arrays we can count inversion pairs, such that one part of pair from left array & other from right array.



Ans = 0 + 2 + 2 + 1 + 2 + 2 + 4 + 3 + 3

$f(arr, st, end)$ = $f(arr, st, mid)$ + $f(arr, mid, end)$ + $merge()$

↓

counts inv pairs

Double inversion Count

$i < j$

$A_i > 2 A_j$

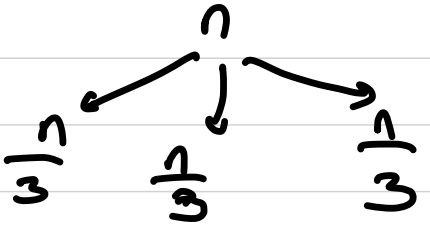
Why we only divide array into 2 parts & not 3 in merge sort ??

$2 \rightarrow \log_2 n$

$3 \rightarrow \log_3 n$

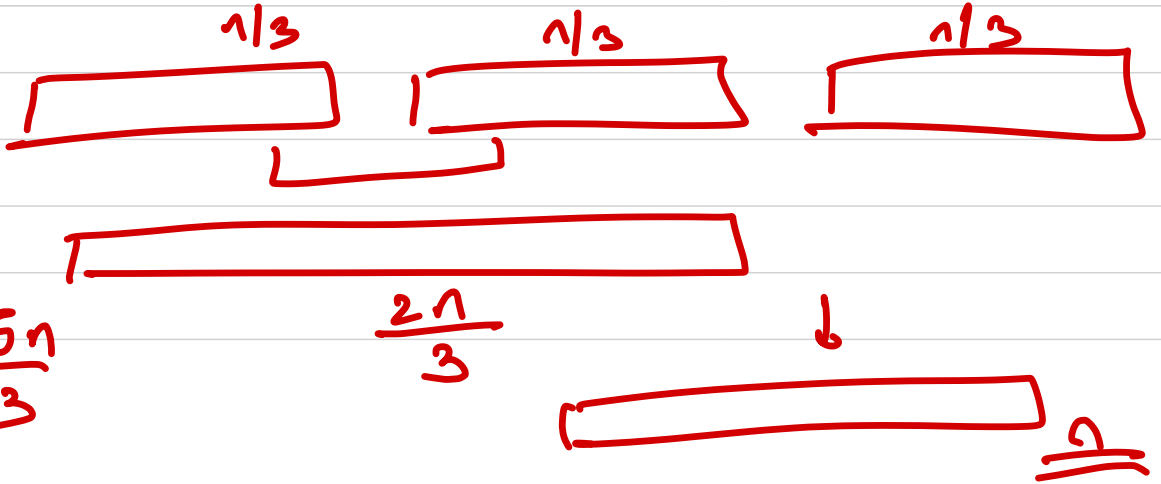
$\log_3 n < \log_2 n$

$\frac{5n}{3} > n$



no of ops to merge the array

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) + f(n)$$



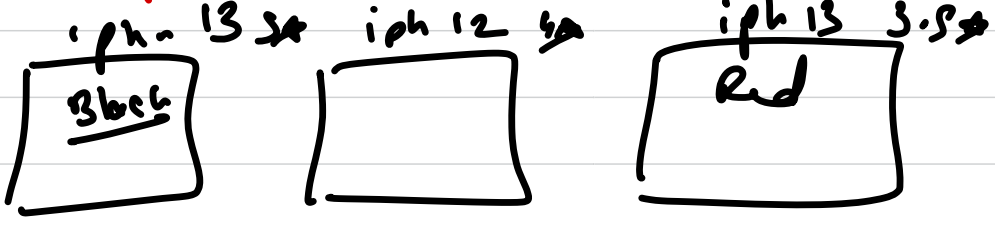
→ Is it stable ??

→ =

O_1' O_2 O_1'' O_3

O_1' O_2'' O_2 O_3 → stable

O_2'' O_1' O_2 O_3 → not stable



iph13 black

iph13 Red

iph12

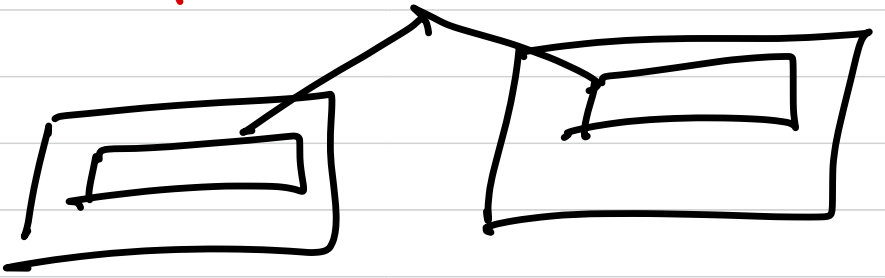
is it inplace ? ? → No

Applications → time → O(n log n)

→ Invent

Quick → O(n²)

→ Parallely



Disadvantage → Space → O(n)