Merge Sort → Merge Procedure

0	1	2	3	4	5	6	7
50	20	40	70	10	60	39	64

n = 8

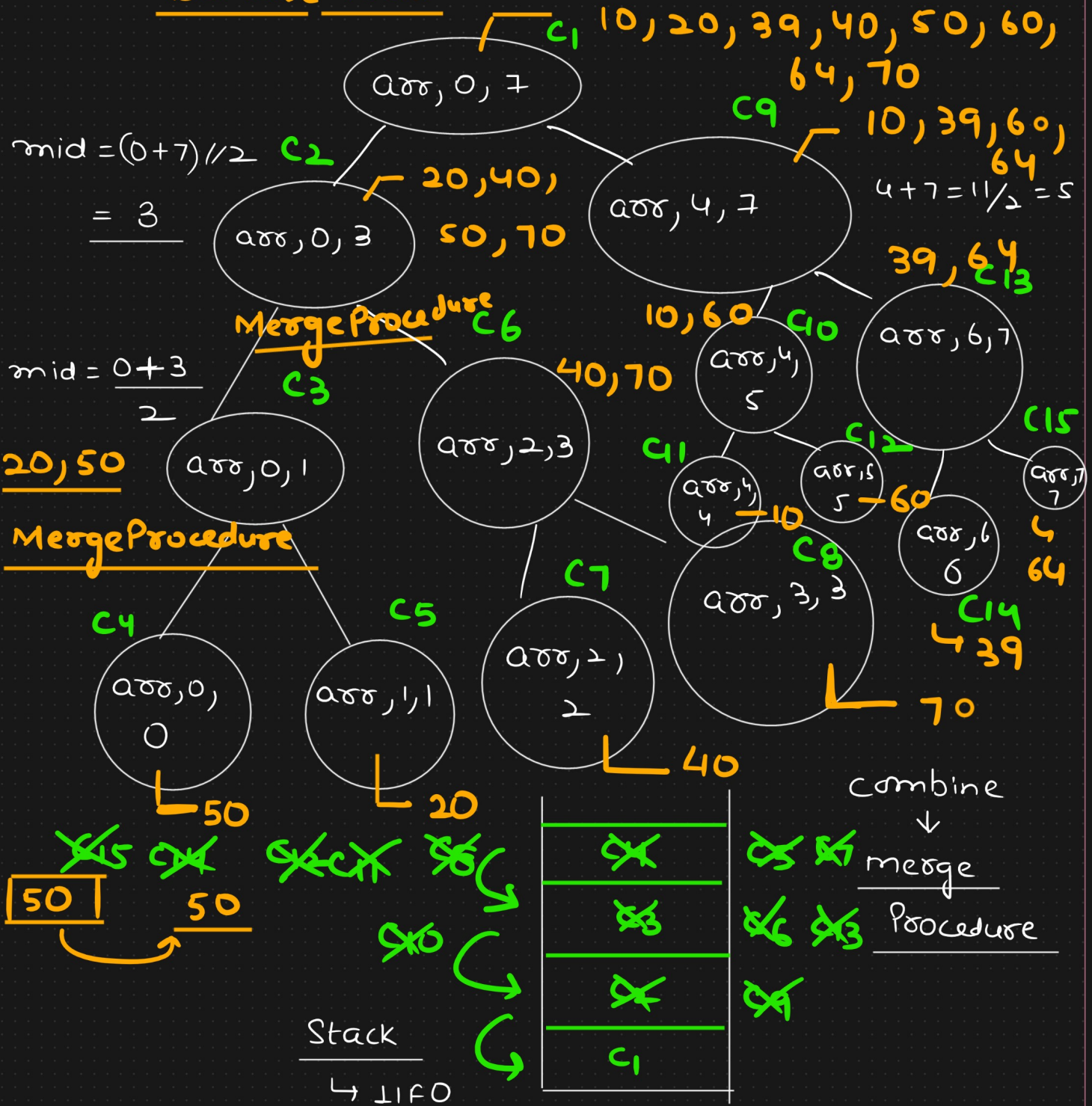
Divide & conquer

n == 1 ⟶ return arr(i)

n > 1

↳ D & C

Recursive Tree

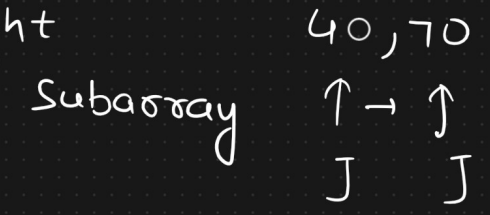


Left Subarray

Merge Procedure

20	40	50	70

Right Subarray



m = 4

Merge Procedure

m = 4

20,40
50,40
50,70

original LS i RS

10	20	30	40	11	21	31	41
0	1	2	3	4	5	6	7

m = 8

i → i → i → i
J → J → j → J

new

extra array

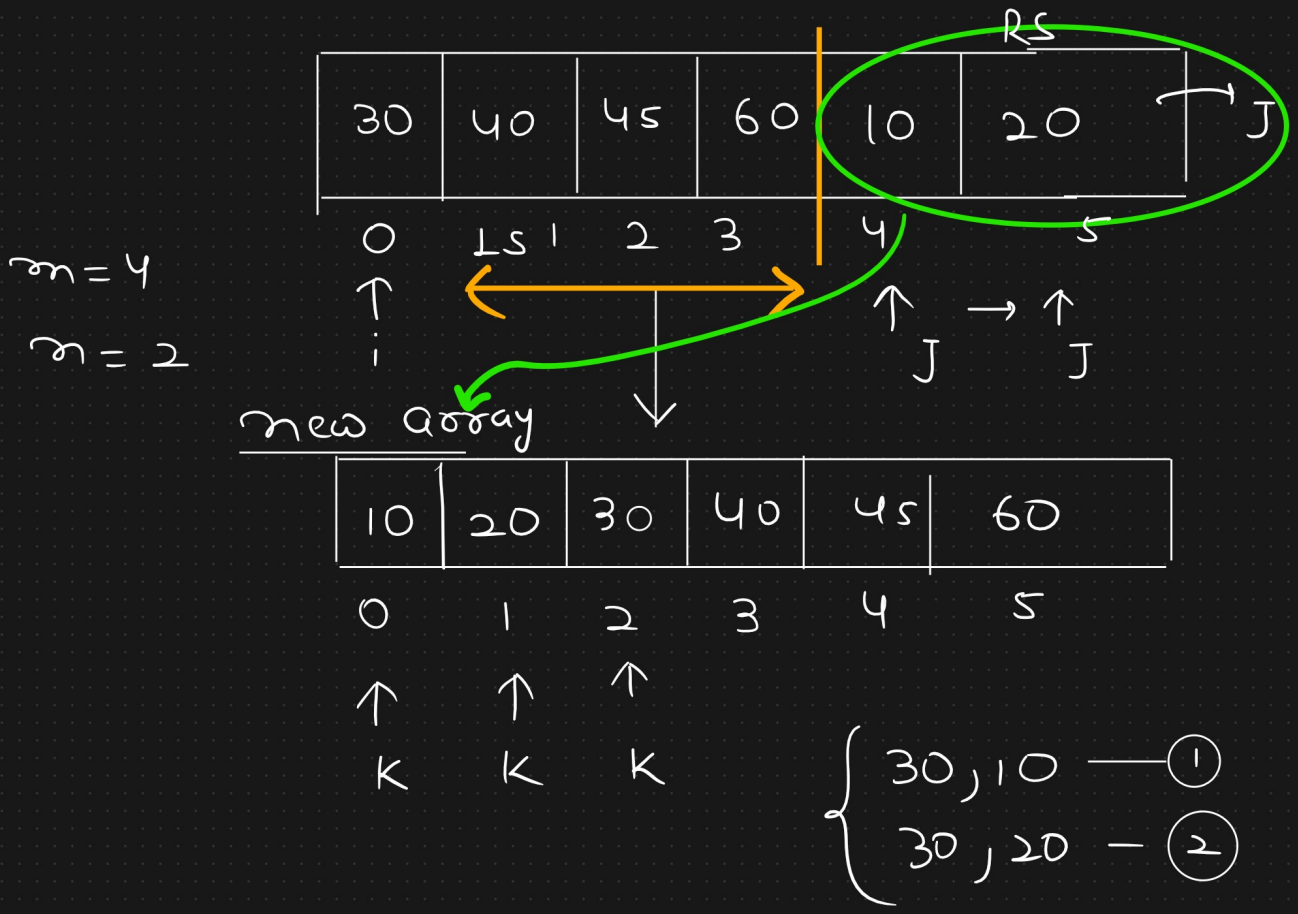
10	11	20	21	30	31	40	41
0	1	2	3	4	5	6	7

K → K → K → K → K → K → K → K

10,11 — ①
20,11 — ②
20,21 — ③
30,21 — ④
30,31 — ⑤
40,31 — ⑥
40,41 — ⑦

Worst case number of comparisons $\Rightarrow m+m-1$

$$\begin{cases} LS \to m \\ RS \to m \end{cases} \qquad m+m = N$$



Best case ⇒ # comparisons = $\underline{\min(m,n)}$

$\min(4,2) = 2$

movements = $m+n$

Time complexity → # comparisons + # move.

Worst case comp. → $(m+n-1) + (m+n)$

→ $O(m+n)$

$m+n = N$

→ $O(N)$

Best case → $\min(m,n) + (m+n)$

Scenario

$\Rightarrow O(N)$

Psuedocode → T(N)

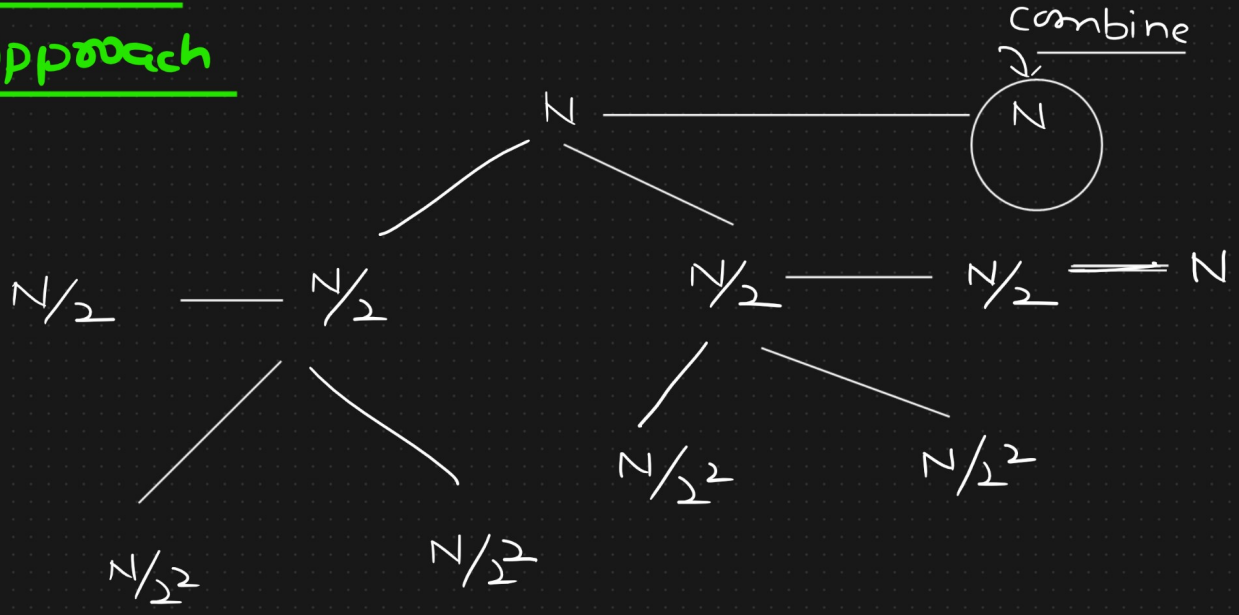
```
mergeSort(arr, p, q):
    if p == q:                      ← c  (Small problem)
        return arr(p)
    else:
        mid = p + (q-p)//2          ← c  (Divide)
        mergeSort(arr, p, mid)      ← T(N/2)
        mergeSort(arr, mid+1, q)    ← T(N/2)   (Conquer)
        mergeProcedure(arr, p, mid, q)   ← N   (Combine)
    return arr
```

Recurrence Relation

$$T(N) = \begin{cases} c & ; N = 1 \\ 2T(N/2) + N & ; N > 1 \end{cases}$$

Recursive Tree Approach

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



$a = 2$

$b = 2$

$\log_b^a = \log_2^2 = 1$

$k = 1$

$\log_b^a = k$

$p = 0$

$p > -1$

$O(n \log n)$

↓

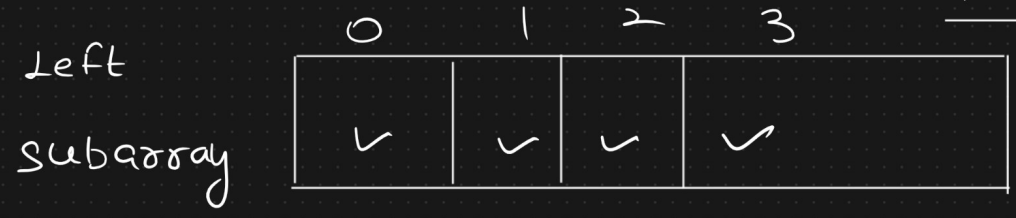
Every case

- $O(n^2)$
- $O(n \log n)$ → Better

Space complexity → Outplace sorting
↳ extra space + Stack algorithm
↳ O(N) space
↳ logN
⇒ O(N)

↳ Stable algorithm

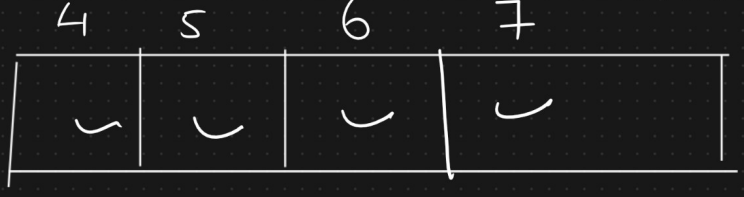
mid = 3 , i = 0



Higher index − lower index + 1

mid − i + 1

mid + 1 , J = 7



J − (mid + 1) + 1

J − mid − 1 + 1 = J − mid

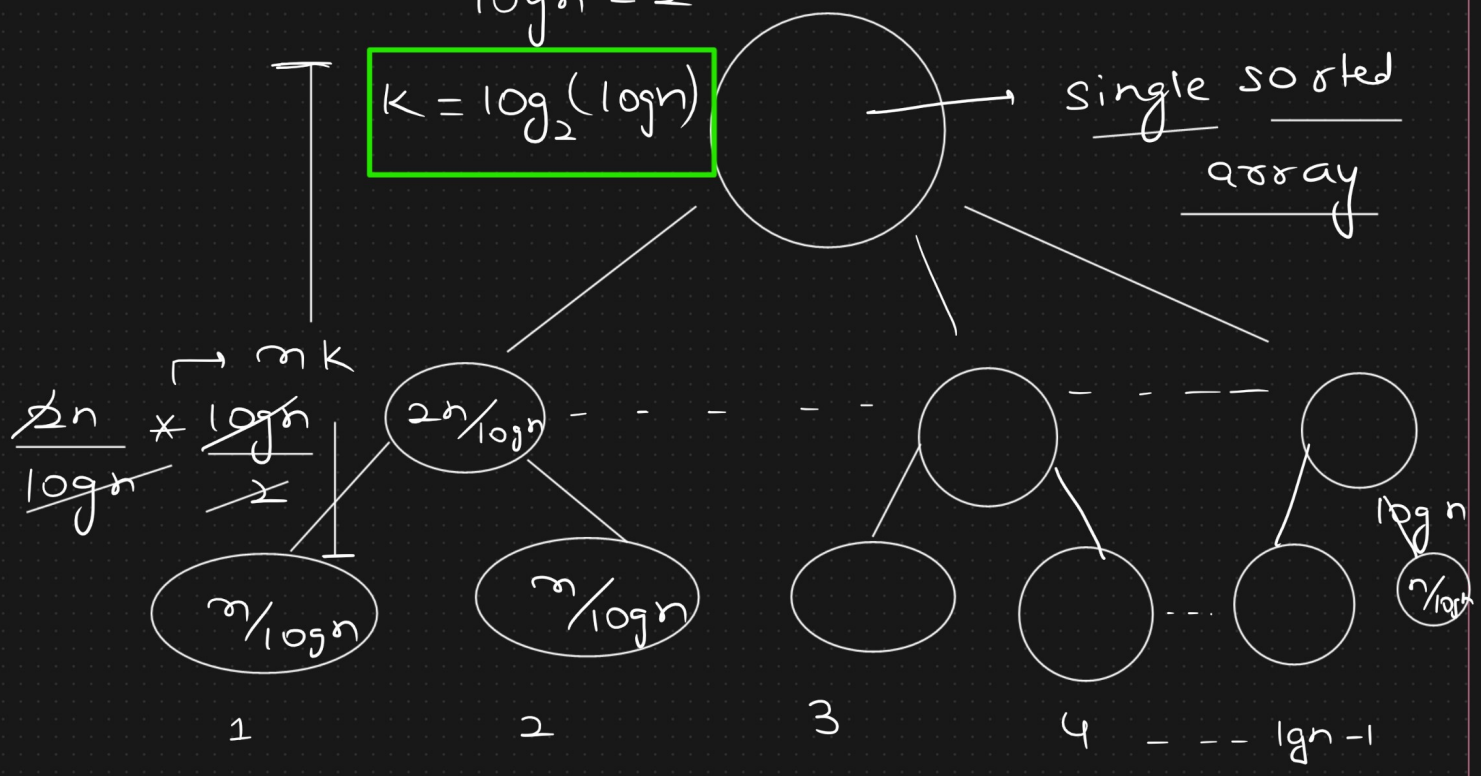
1) $\log n$ sorted subarrays each of size of $\frac{n}{\log n}$

time complexity of single sorted array?

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

$\log n = 2^k$

$k = \log_2(\log n)$



Sorted subarray

$\frac{n}{\log n} * \log n = n$

Time complexity → $O(n \log \log n)$

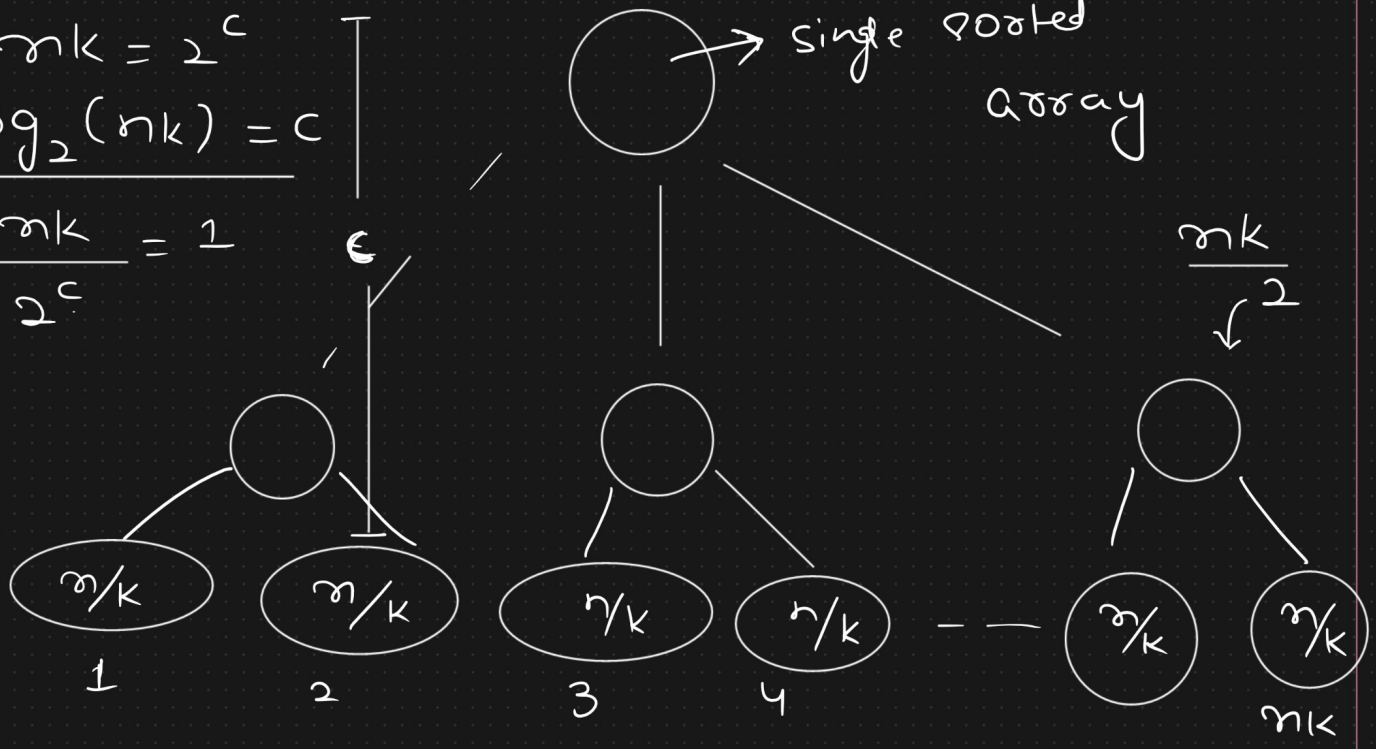
2) nk sorted subarrays each of size n/k

Time Complexity →

$nk = 2^c$

$\log_2(nk) = c$

$\frac{nk}{2^c} = 1$



$\frac{n}{k} * nk = n^2$

$TC \rightarrow O(n^2 * \log_2(nk))$