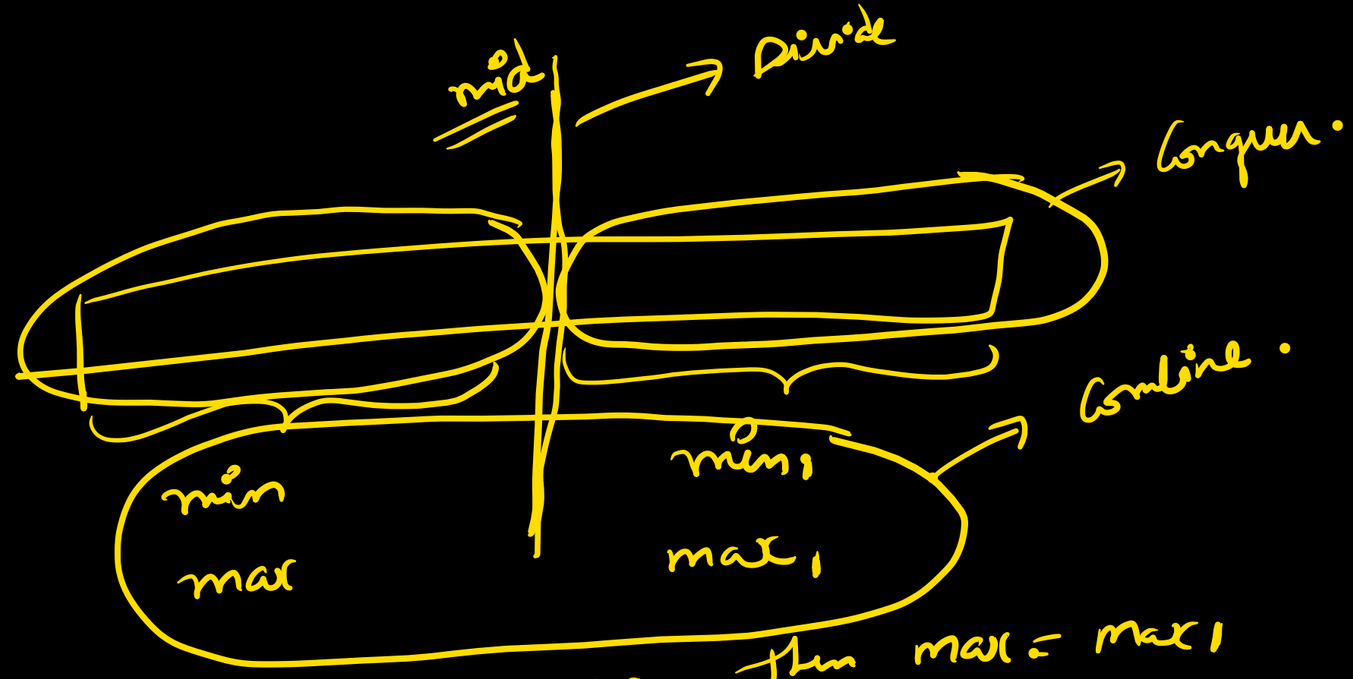


max < max1 then max = max1

min > min1 then min = min1

```
Algorithm maxmin(i, j, max, min)
{
    if (i == j) then max = min = a[i];          // 0 comparisons → small ⇒ size = 1
    else if (i == j-1) then                     // 1 comparison ⇒ small ⇒ size = 1
    {
        if (a[i] < a[j]) then
        {
            max = a[j]; min = a[i];
        }
        else
        {
            max = a[i]; min = a[j];
        }
    }
    else
    {
        mid = ⌊(i+j)/2⌋                         // divide.
        maxmin(i, mid, max, min);               // conquer
        maxmin(mid+1, j, max1, min1);
        if (max < max1) then max = max1;
        if (min > min1) then min = min1;
    }
}
```

max min | max1 min1

Both problems are divided in 1 & 2 size problem recursively.

Even divided into smaller problems till they can be solved without dividing.

→ (2).

a 1 2 3 4 5 6 7 8 9

22 13 | -5 | -8 15 | 60 17 | 31 47

1,9, 60, -8 (9)

max min

1,5 22 -8 (5)

1,3 22 -5 (3)

1 2 22 13 (1)

2

3,3,-5,-5

1

(2)

(4)

4,5 15 -8

2

max min

6,9 60 17 (8)

6,7, 60, 17

2

(6)

8 9 47,31

2

(7)

max
min

22 13

max
min

-5 -5

Size = 1 — 0 Comp

Size = 2 — 1 Compare

Size ≥ 3 — 2 Comp.

$C(n)$ is number of Comparisons required to find maximum on a size of 'n'.

$$C(n) = \begin{cases} C(n/2) + C(n/2) + 2 & n > 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$$

$$C(n) = 2C(n/2) + 2$$

$$= 2(2C(n/4) + 2) + 2$$

$$= 2^2(C(n/2^2)) + 2^2 + 2$$

$$= 2^2(2C(n/2^3 + 2) + 2^2 + 2$$

$$= 2^3 C(n/2^3) + 2^3 + 2^2 + 2^1$$

$$= \vdots \quad k$$

$$= 2^k T(n/2^k) + 2^k + 2^{k-1} + \cdots 2$$

Let $n/2^k = 2 \Rightarrow 2^k = n/2$

$$= n/2 * T(2) + \frac{2(2^k - 1)}{2 - 1}$$

$$= \frac{n}{2} + 2\left(\frac{n}{2} - 1\right) = \frac{3n}{2} - 2$$

$\left(\frac{3n}{2} - 2\right)$ is the best, average and worst case no of comparisons.

No method can give better comparisons than Dest divide and conquer max min.

Disadv: In terms of space, max min is worse than straight forward algo because it requires stack space for recursion.

Space complexity :-

$n/2^k = 1$

$\Rightarrow k = \log n$

Time Complexity for DandC max min.

$$T(n) = \begin{cases} O(1), & n = 1 \\ O(1), & n = 2 \\ T(n/2) + T(n/2) + C & \rightarrow n > 2 \end{cases}$$

TC
$O(n)$

SC
$O(\log n)$

Comp:
$\frac{3n}{2} - 2$

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

$f(n)$
C

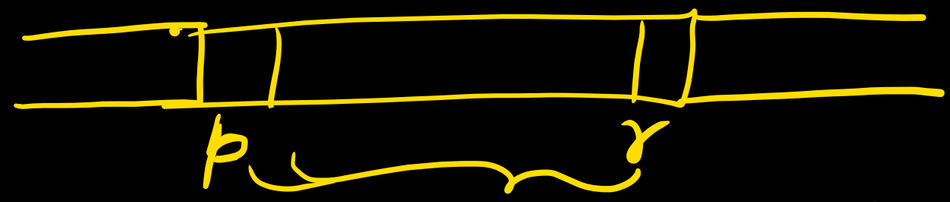
$n^{\log_b a}$
$n^{\log 2}$
n

$O(n)$.

One more example on D and C, merge sort

Dand C.

merge_sort (A, p, r)



Small if $n \leq 3$

{

if $(p < r)$ → number elements are ≥ 2

{

then $q = \lfloor \frac{p+r}{2} \rfloor$ → divide.

merge_sort (A, p, q) → Conquer

merge_sort (A, q+1, r)

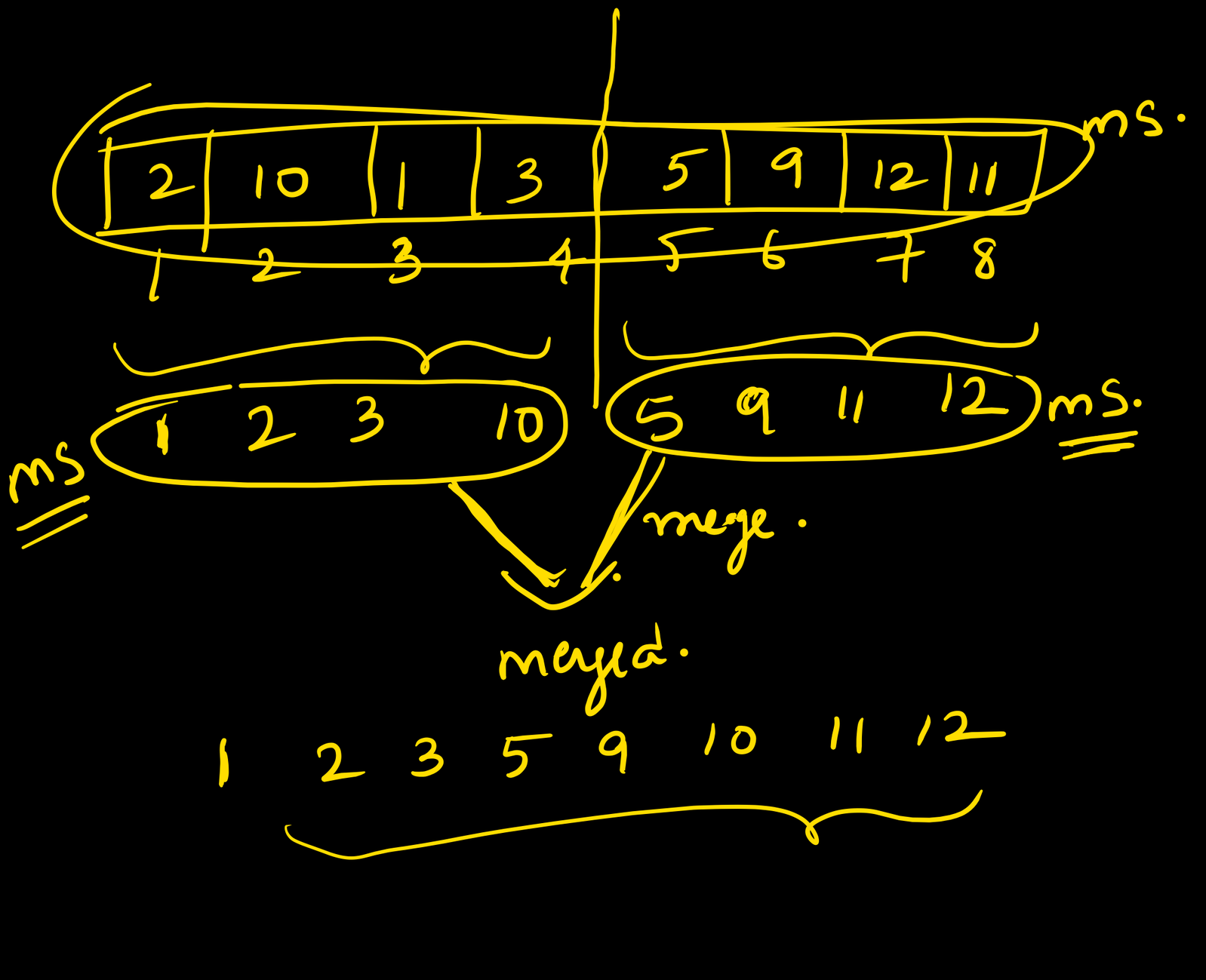
merge (A, p, q, r) → Combine.

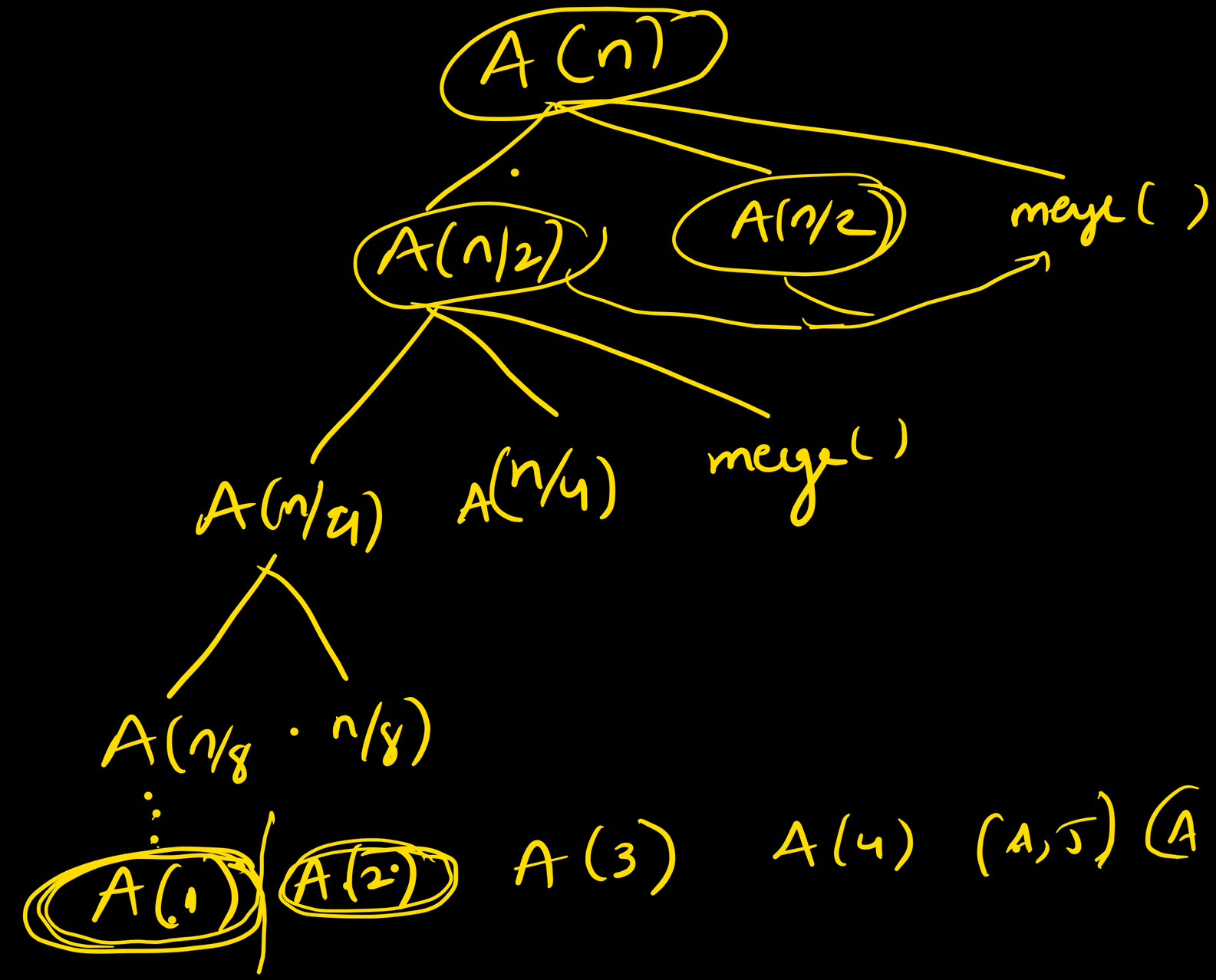
}

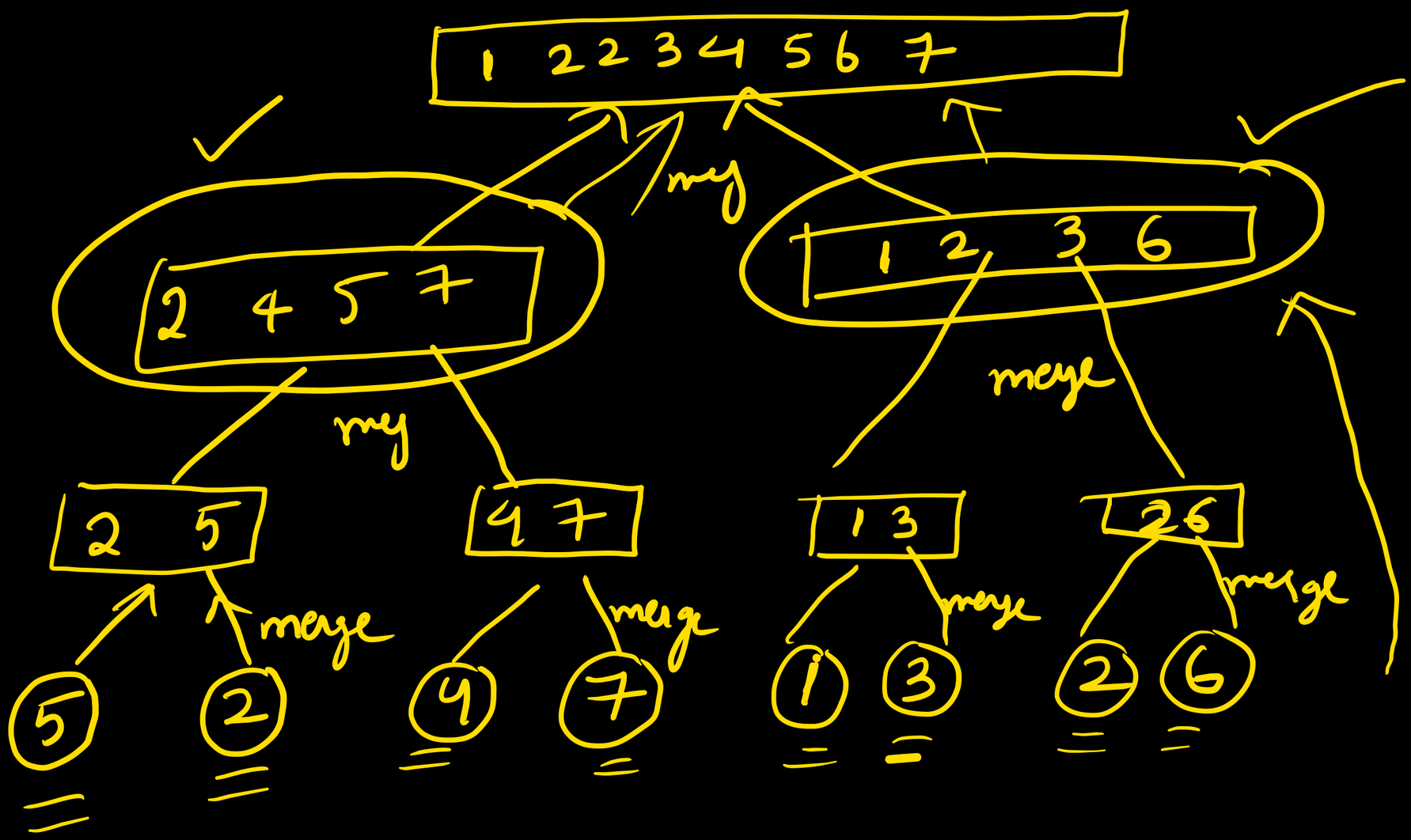
}

divide the problem into problems of size 1 A single element is already sorted.

Merge sorts.

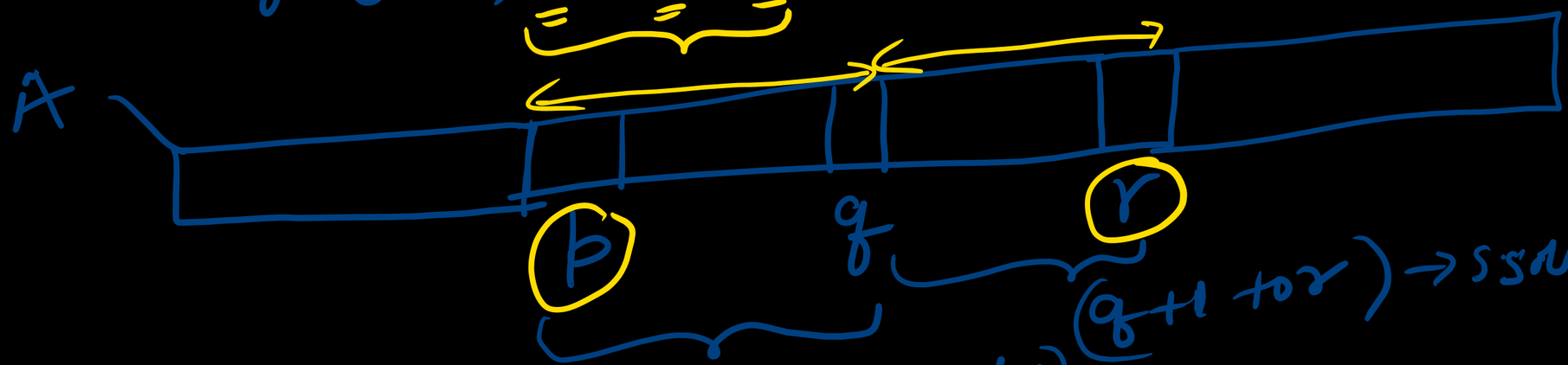

2 10 1 3 5 9 12 11
ms.
1 2 3 4 5 6 7 8
1 2 3 10
ms
5 9 11 12
ms.
merge.
merged.
1 2 3 5 9 10 11 12

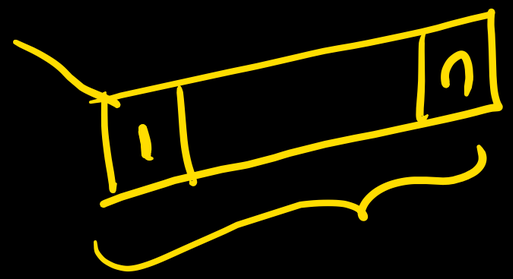

A(n)
A(n/2)
A(n/2)
merge( )
A(n/4)
A(n/4)
merge( )
A(n/8 · n/8)
A(1)
A(2)
A(3)
A(4)
(A,5)
(A 6). .


1 2 2 3 4 5 6 7
2 4 5 7
1 2 3 6
merge
merge
merge
merge
merge
merge
merge
2 5
4 7
1 3
2 6
5
2
4
7
1
3
2
6

Divide the elements into one sized array and merge them bottom up.

merge (A , p, q, r)





A [p, q] → sorted

A [q+1, r] → sorted.

we have to make A [p, r] sorted

merge (A, p, q, r)

{

$n_1 = q - p + 1$ // I sorted list

$n_2 = r - q$ // II sorted list

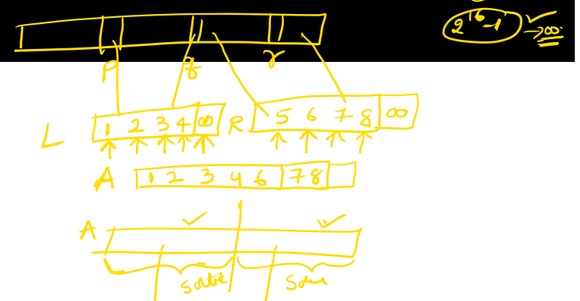
Create two arrays $L[1 \ldots n_1+1]$ $R[1 \ldots n_2+1]$

for (i = 1 to n_1)

do $L[i] = A[p+i-1]$ // Copying $A[p]$ to $A[q]$ into L

for (j = 1 to n_2)

do $R[j] = A[q+j]$ // $A[q+1][r]$ into R



$L[n_1+1] = \infty$

$R[n_2+1] = \infty$

$i = 1$

$j = 1$

~~for (k = p to r)~~

for (k = p to r)

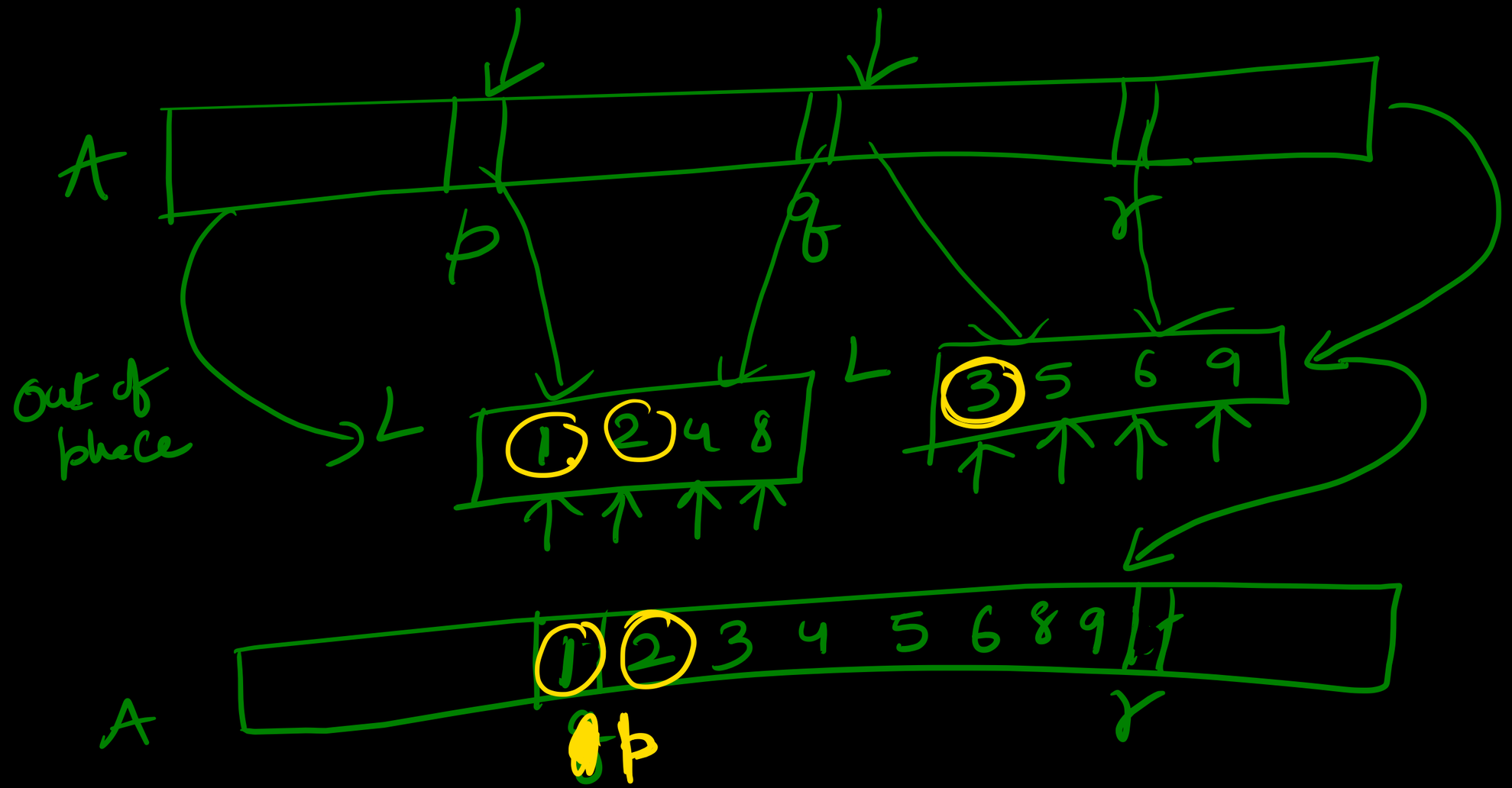
do if $L[i] \le R[j]$

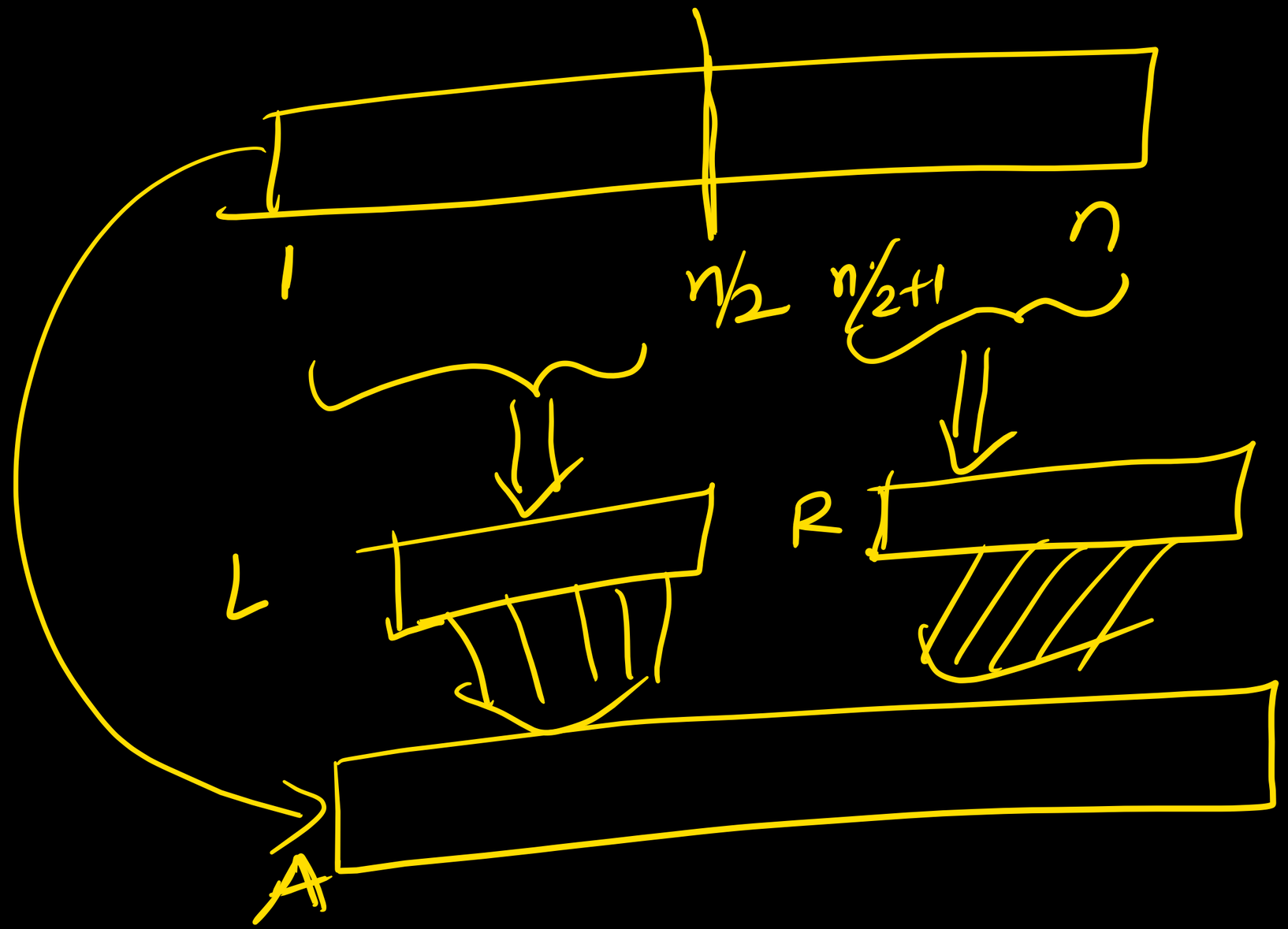
then $A[k] = L[i]$

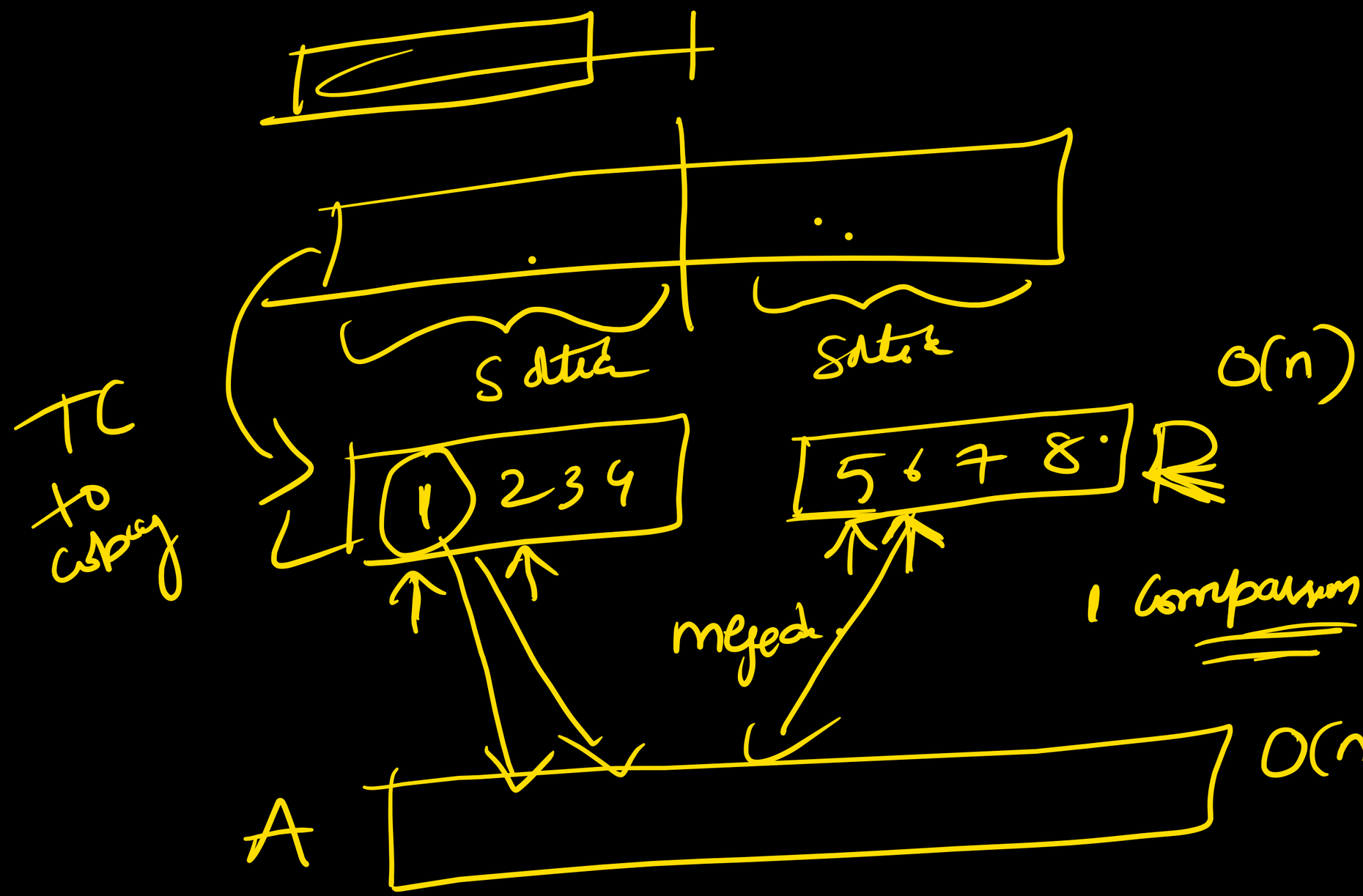
$i = i + 1$

else $A[k] = R[j]$

$j = j + 1$


A
p
q
r
Out of place
L
1
2
4
8
L
3
5
6
9
A
1
2
3
4
5
6
8
9
p
r

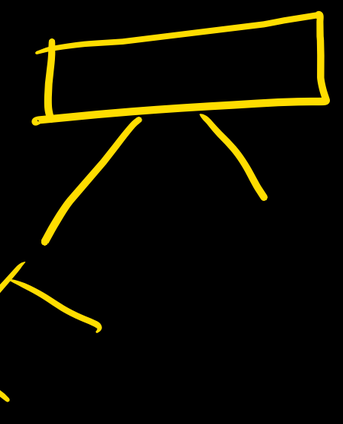

1
n/2
n/2+1
n
L
R
A


Sdtch
Sdtch
O(n)
TC
to
copy
1 2 3 4
5 6 7 8
R
merged
1 Comparison per element.
A
O(n).

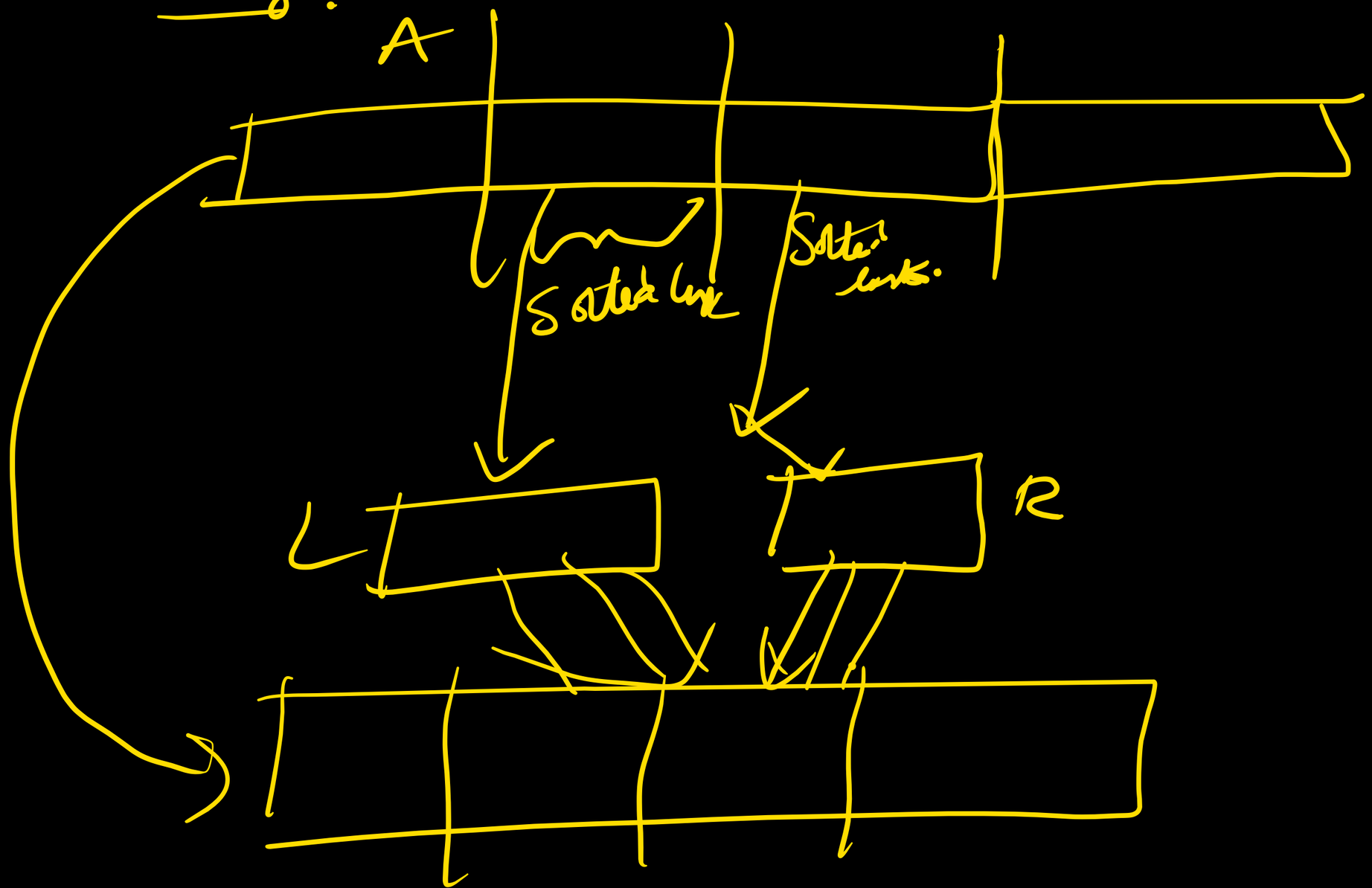
merge ⇒ copying from A to L and R. (O(n))

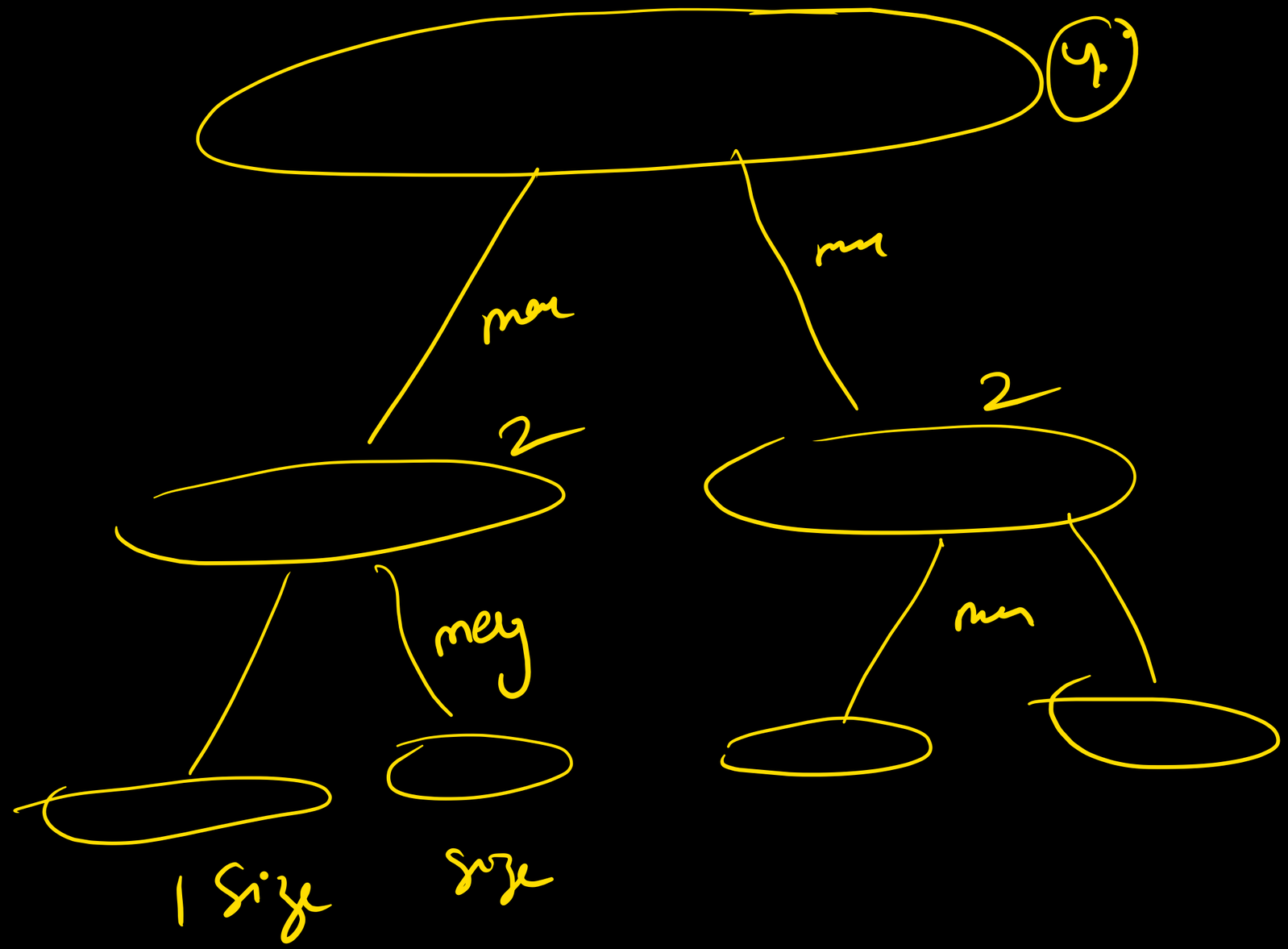
⇒ merging L and R into A. (O(n))

merge ⇒ O(n)



merge:




n
max
max
2
2
max
max
1 Size
Size

MS (n) ⇒ T(n)

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

MS (n/2) ⇒ T(n/2)

MS (n/2) ⇒ T(n/2)

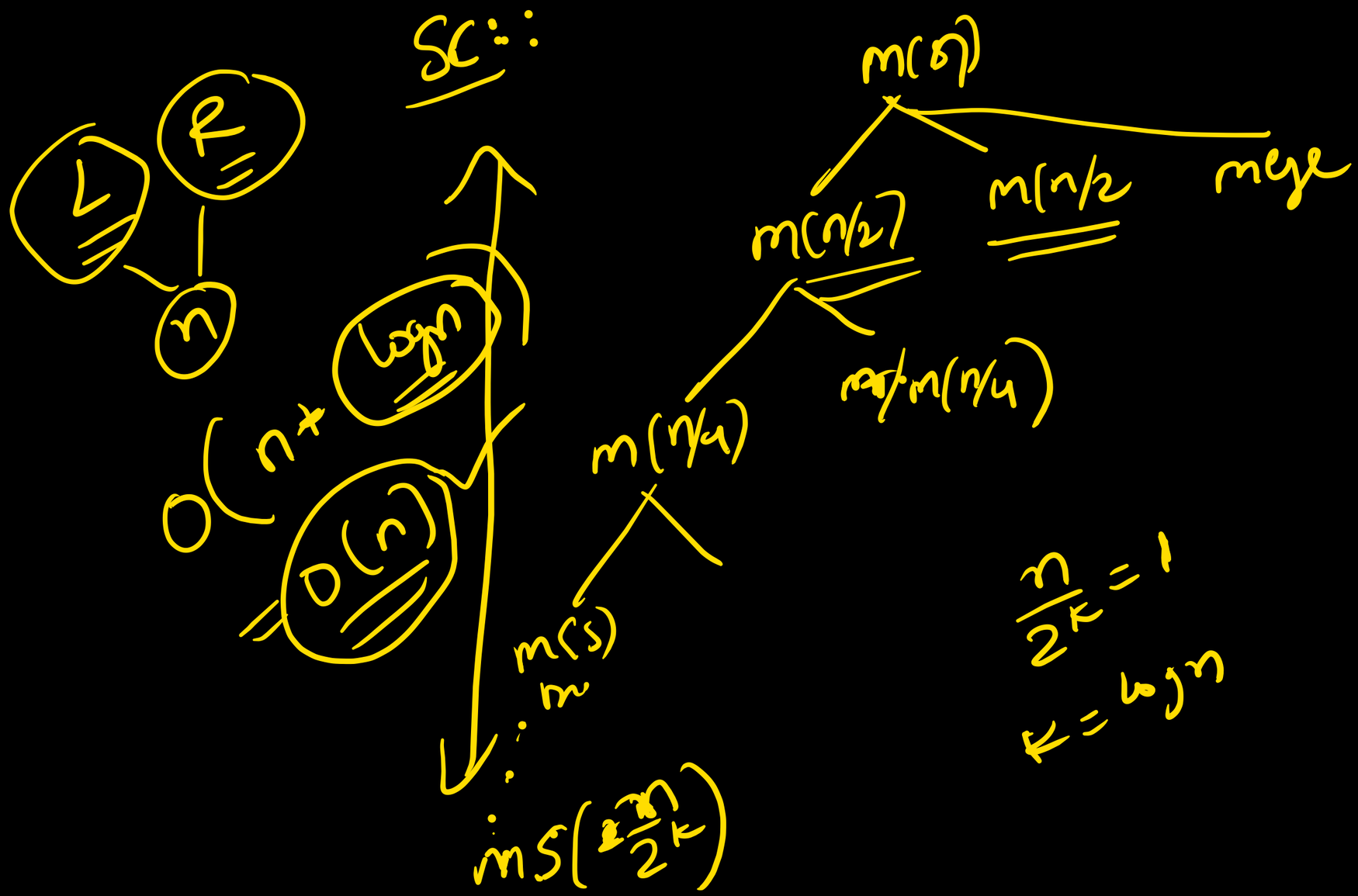
O(n) + O(n)

merge (two sorted lists)

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

$f(n)$ ⇓ cn = $n^{\log_b a}$ ⇓ $n^{\log 2}$ = n

$T(n) = O(n \log n)$



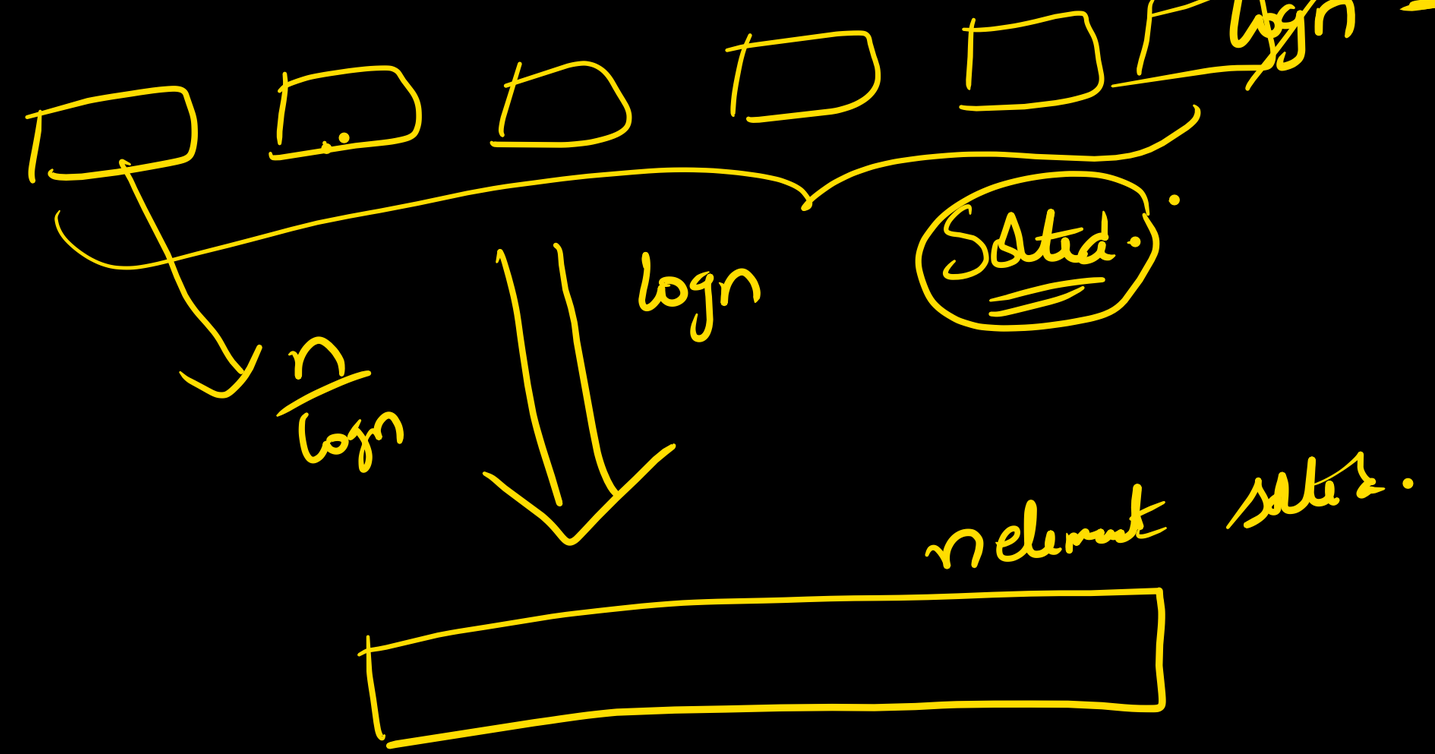
$$\frac{n}{2^k} = 1$$

$$k = \log n$$

i/p : $\log n$ sorted subarrays each of size $\frac{n}{\log n}$

o/p : one n element single sorted array

TC = ?



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

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

$\log n/2^2$

$\frac{\log n}{2}$

$\log n$ units

$O(n)$

Sorted

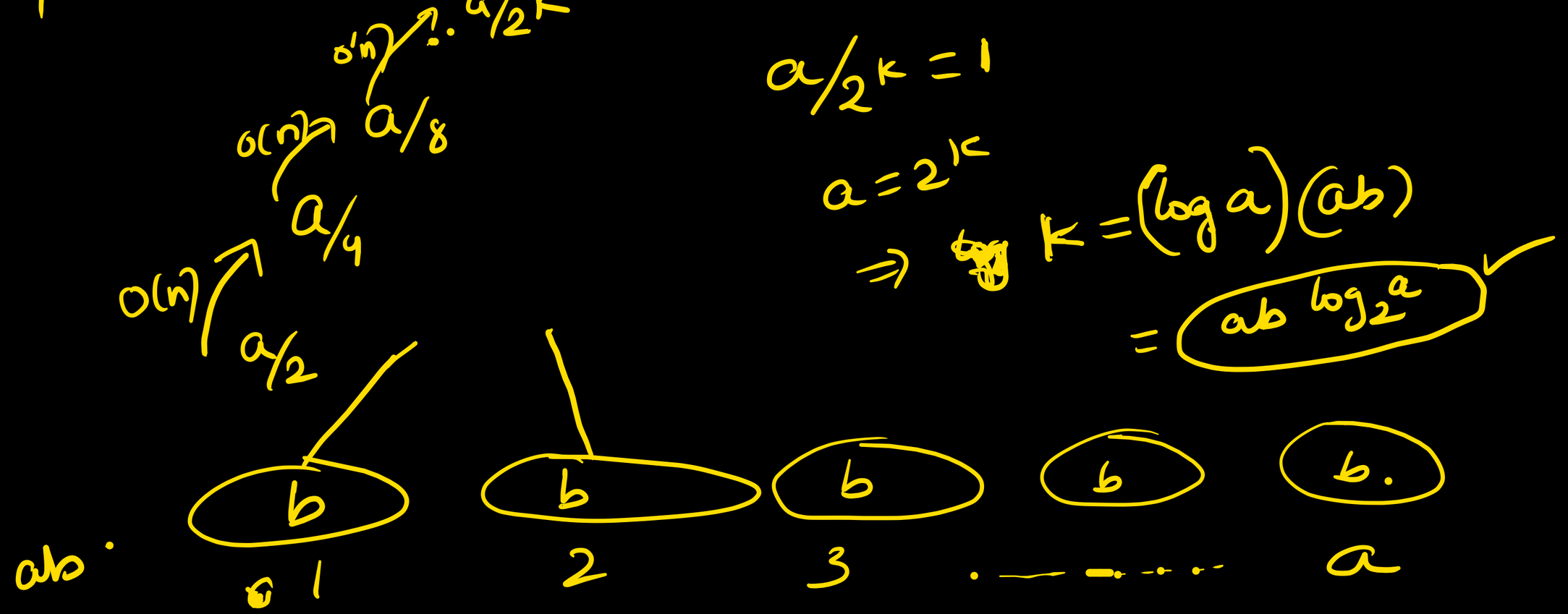
Sorted

$O(n(\log\log n))$

i/p: 'a' sorted subarrays each of size b

o/p: Find single sorted array

T.C = ?



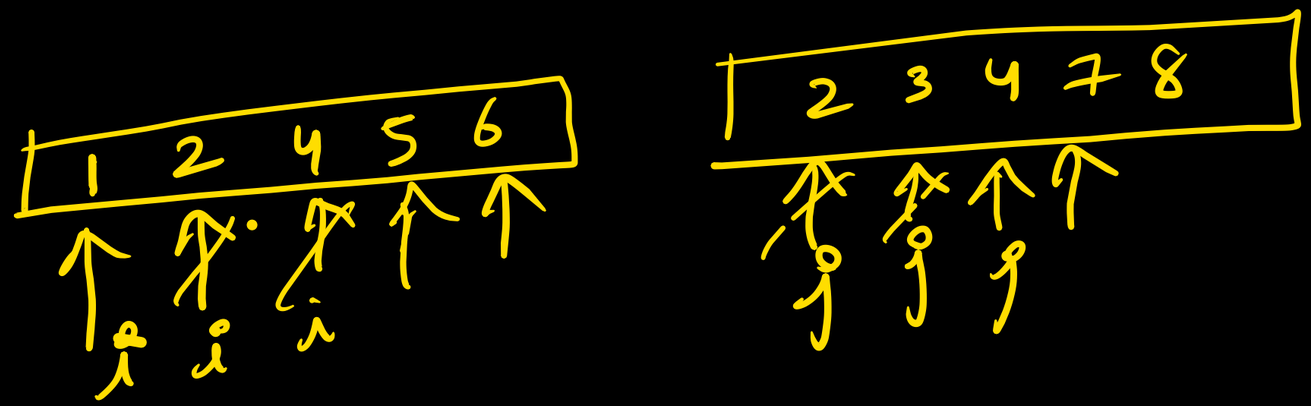
$a/2^k = 1$

$a = 2^k$

$\Rightarrow \log_2 k = (\log a)(ab)$

$= ab \log_2 a$

Given two sorted arrays, TC to find $A \cap B$.



2 4

$= O(n)$