GATE

CRASH COURSE

DS & AI

Algorithms

Divide & Conquer

(Part 01) (Lec 04)

By - Aditya sir

Topics to be covered

1.

2. Divide and Conquer (DnC)

About Aditya Jain sir

1. Appeared for GATE during BTech and secured AIR 60 in GATE in very first attempt - City topper
2. Represented college as the first Google DSC Ambassador.
3. The only student from the batch to secure an internship at Amazon. (9+ CGPA)
4. Had offer from IIT Bombay and IISc Bangalore to join the Masters program
5. Joined IIT Bombay for my 2 year Masters program, specialization in Data Science
6. Published multiple research papers in well known conferences along with the team
7. Received the prestigious excellence in Research award from IIT Bombay for my Masters thesis
8. Completed my Masters with an overall GPA of 9.36/10
9. Joined Dream11 as a Data Scientist
10. Have mentored working professions in field of Data Science and Analytics
11. Have been mentoring GATE aspirants to secure a great rank in limited time
12. Have got around 27.5K followers on Linkedin where I share my insights and guide students and professionals.

Topic : (Lecture Schedule)

2. Divide & Conquer

1. General Method →
2. Max-Min Problem
3. Binary Search
4. Merge Sort
5. Quick Sort
6. Master Method for D and C Recurrences

} TC analysis

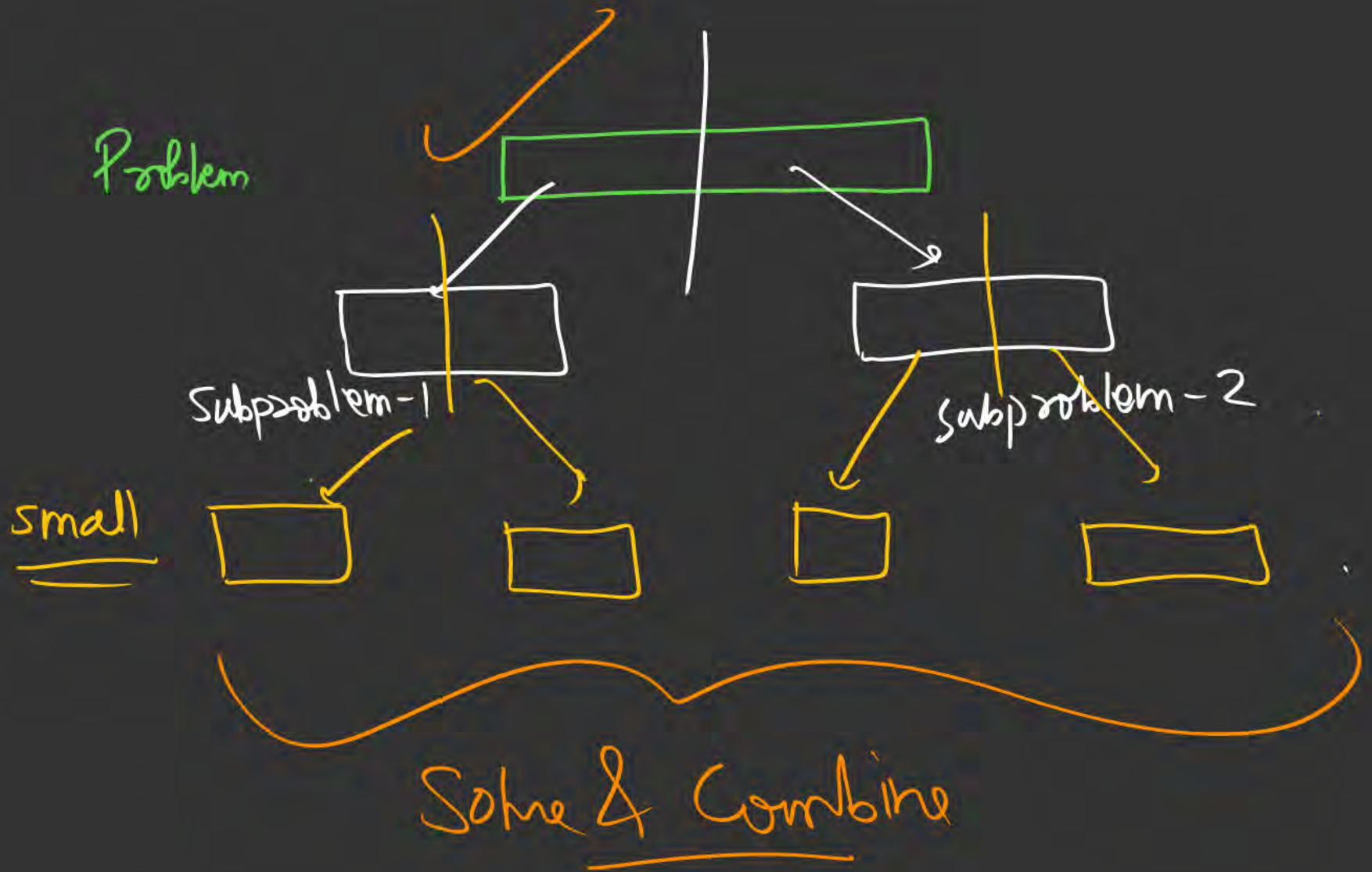
(Shortcut)

Topic : Design Strategies

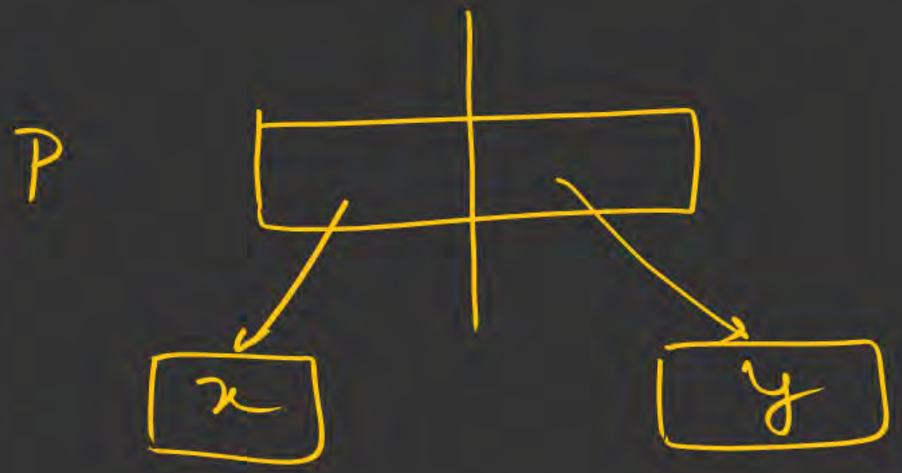
→ When the problem becomes large/complex, then divide the problem into sub problems, into further subproblems, until the subproblem becomes small.

→ Solve the smaller problem, combine their results if required to get the solution of original problem.

→ In general, a problem is said to be small, if it can be solved in one/two basic operations.


Problem
Subproblem-1
subproblem-2
small
Solve & Combine


Ruby



TC

$$T(P) = T(x) + T(y) + F(n)$$

↓

DnC

Types of TC Recurrences

1) Symmetric

2) Asymmetric

Symmetric :

(subproblems of equal size)

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

D & C

where

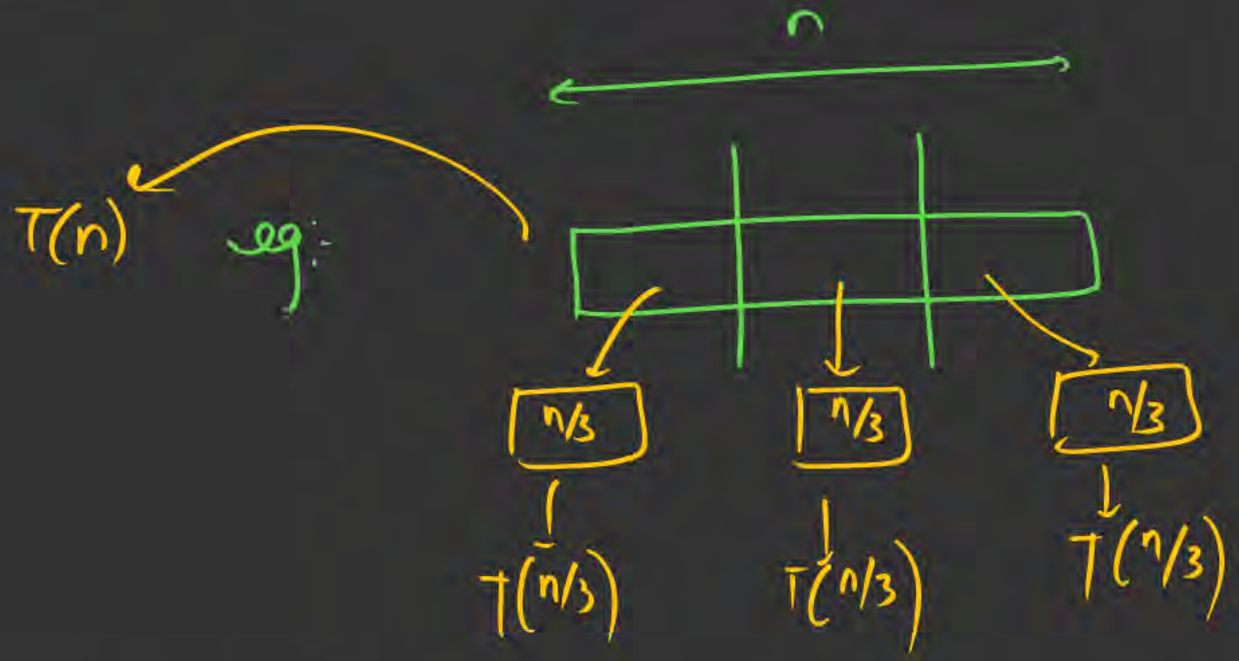
$a \geq 1$

$b > 1$

$f(n) =$ +ve

no. of subproblems

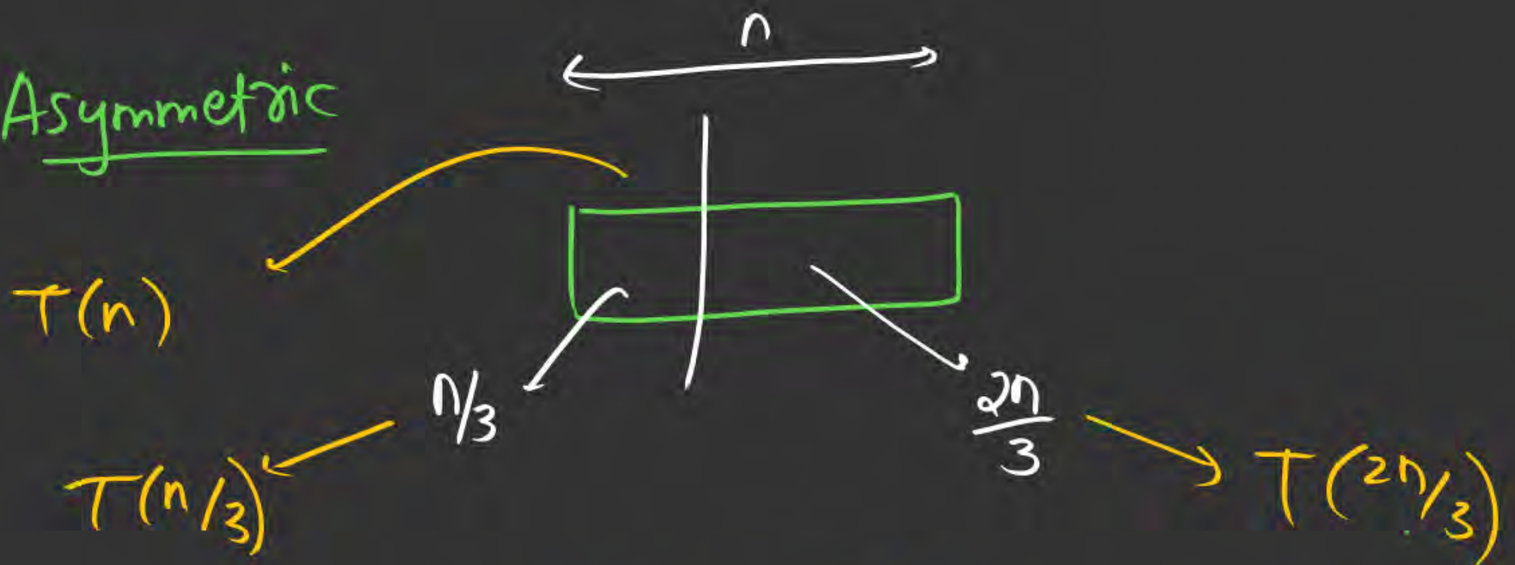
Size of each subproblem



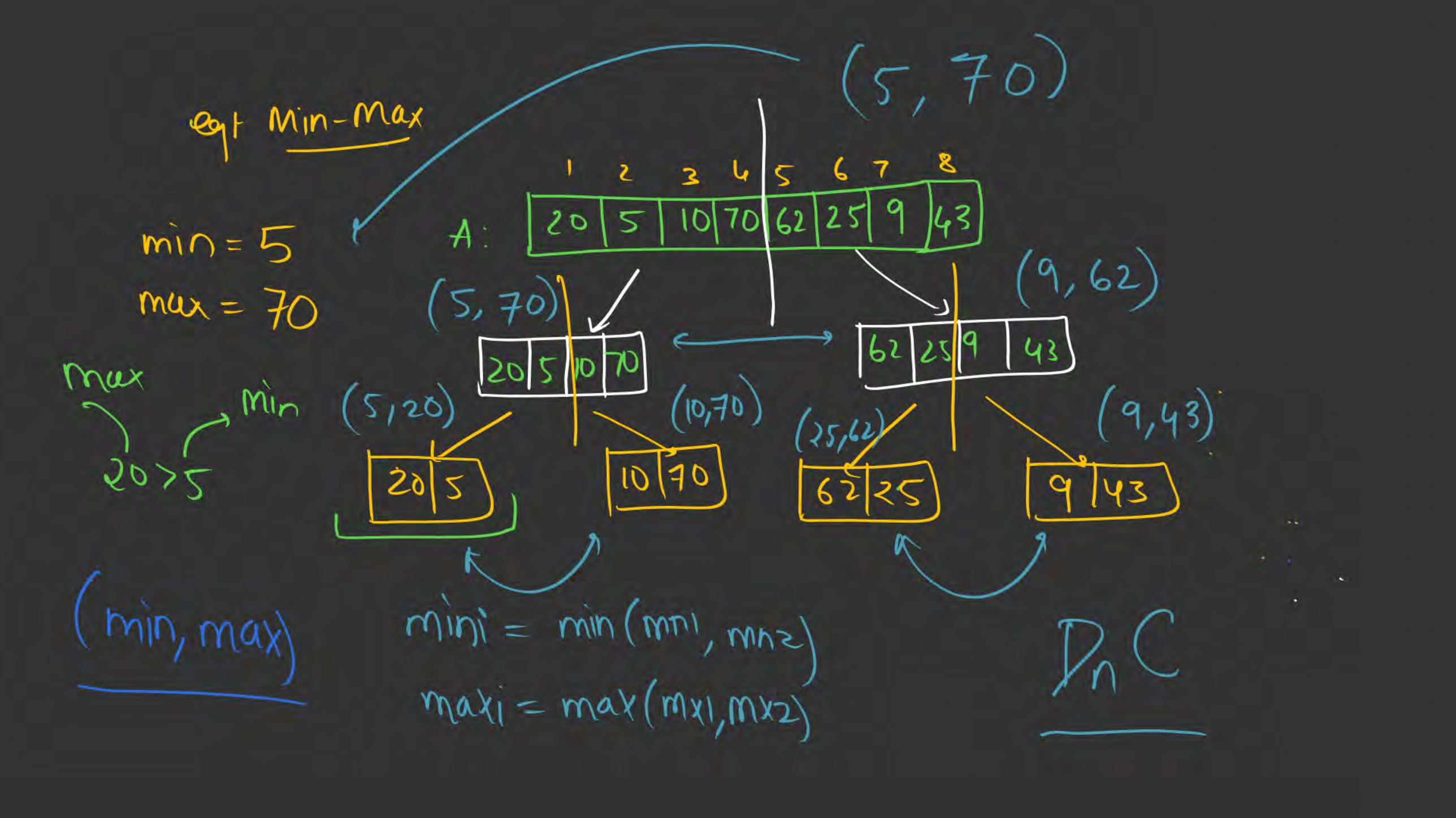
TC: $T(n) = T(n/3) + T(n/3) + T(n/3) + f(n)$

$T(n) = 3 * T(n/3) + f(n)$

2) Asymmetric



$$T(n) = T(n/3) + T\left(\frac{2n}{3}\right) + f(n)$$

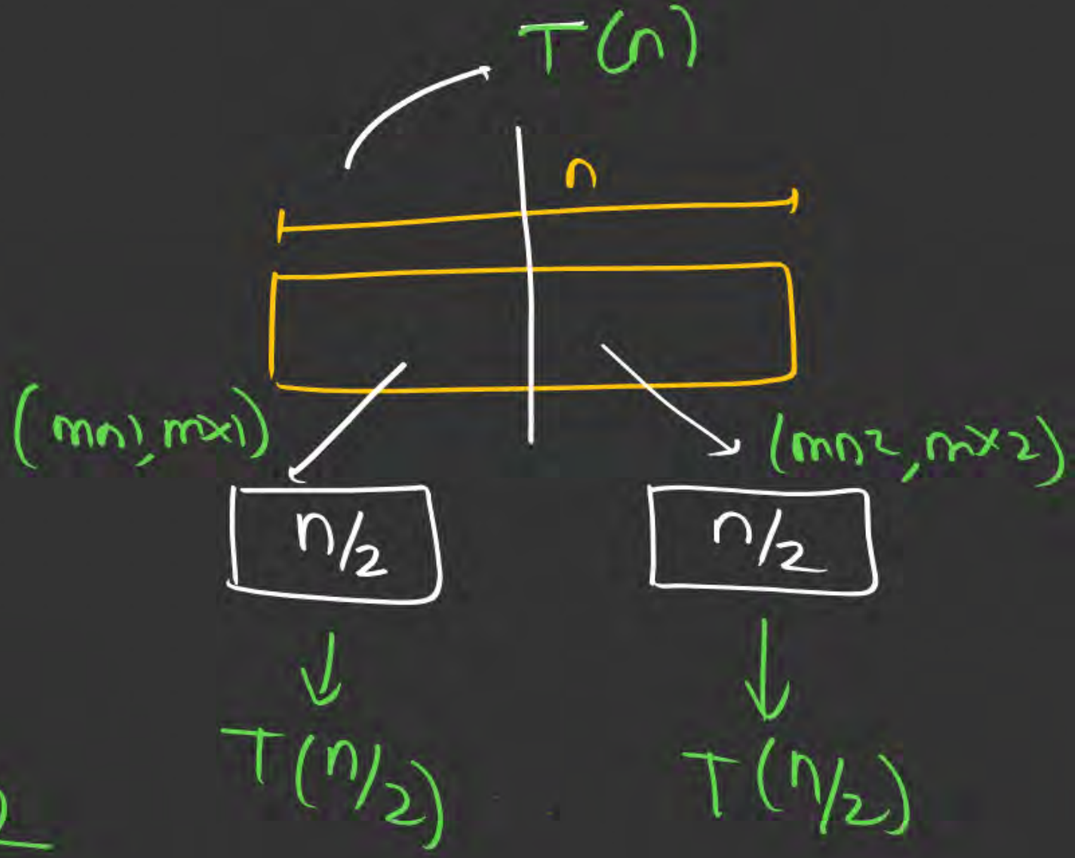

eg: Min-Max
(5, 70)
1 2 3 4 5 6 7 8
A: 20 5 10 70 62 25 9 43
min = 5
max = 70
(5, 70)
20 5 10 70
(9, 62)
62 25 9 43
max
min
20 > 5
(5,20)
20 5
(10,70)
10 70
(25,62)
62 25
(9,43)
9 43
(min, max)
mini = min(mn1, mn2)
maxi = max(mx1, mx2)
DnC

Recurrence

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

$$\boxed{T(n) = 2T(n/2) + 2}, \quad n > 2$$

$$T(n) = 1, \quad n = 2$$



$$T(n) = 2T(n/2) + 2 \quad \text{— (1)}$$

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

$$T(n) = 2^2 T(n/2^2) + (2^1 + 2^2)$$

$$T(n) = 2^3 T(n/2^3) + (2^1 + 2^2 + 2^3) \quad ----$$

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

$$n/2^k = 2 \longrightarrow \boxed{2^k = n/2}$$

$$T(n) = 2^k T(1) + (2^1 + 2^2 \cdots 2^k)$$

↓ GP

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

$$= n/2 + 2(n/2 - 1)$$

$$= n/2 + n - 1$$

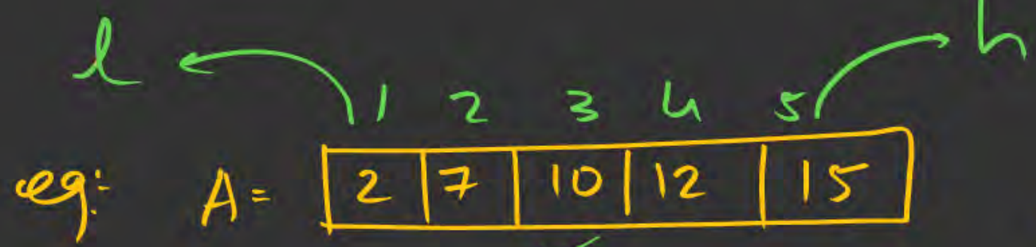
$$\boxed{T(n) = \frac{3n}{2} - 1}$$

$$TC = O(n)$$

(2) Binary Search :

Linear :

→ Pre-requisite : (Sorted array as i/p)

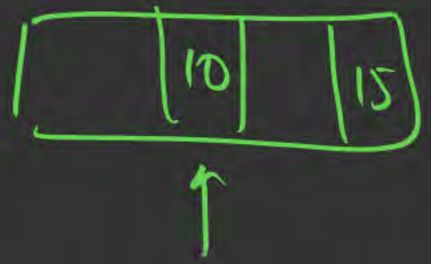


Key = 15

$mid = \left\lfloor \frac{l+h}{2} \right\rfloor$

[l, h]

$mid = \left\lfloor \frac{1+5}{2} \right\rfloor = 3$



Cases:

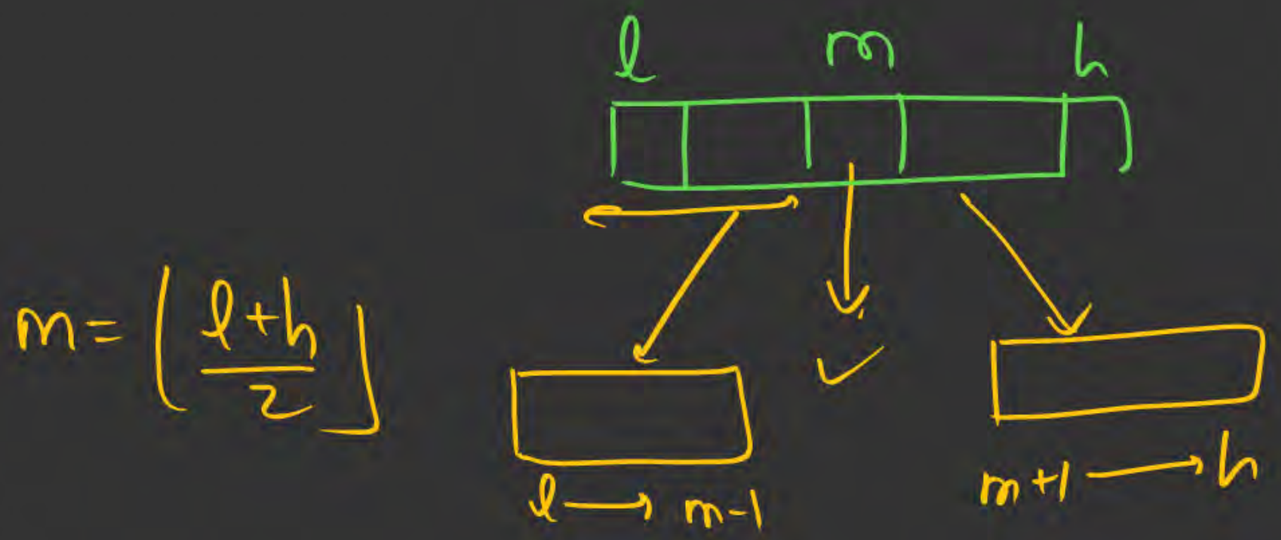
1) A[mid] = Key, Stop

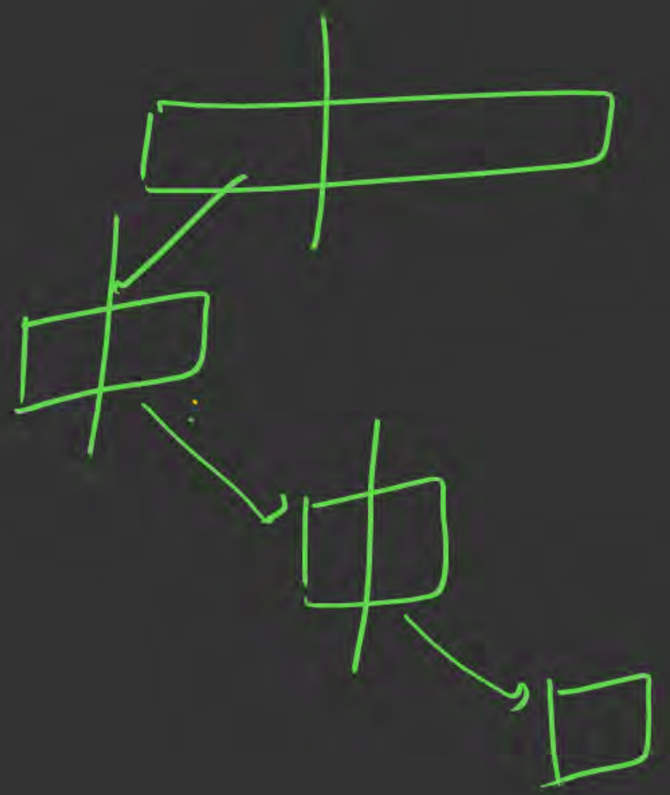
OR

2) A[mid] > Key, explore left

OR

3) A[mid] < Key, explore Right

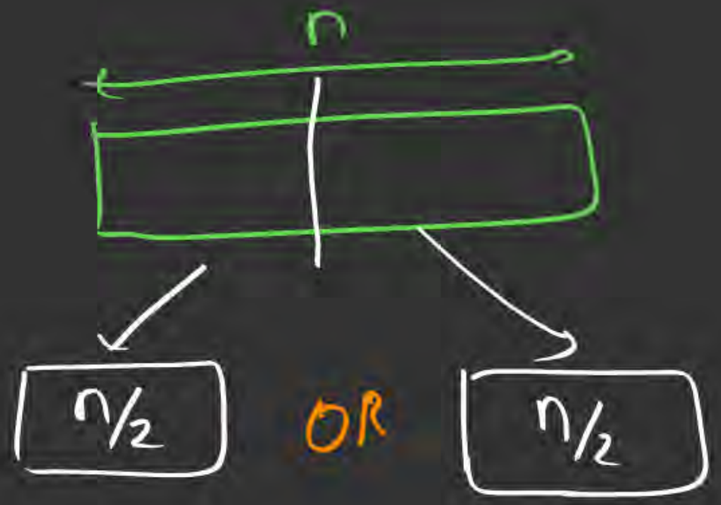




$BS(l, h)$

$BS(l, m-1)$ OR $BS(m+1, h)$

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



$T(n) = T(n/2) + C$ —— (1)

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

$T(n) = T(n/2^2) + 2C$ —— (2)

$T(n) = T(n/2^3) + 3c$

:

general :

$T(n) = T(n/2^k) + k*c$

For Base Condition $n/2^k = 1$

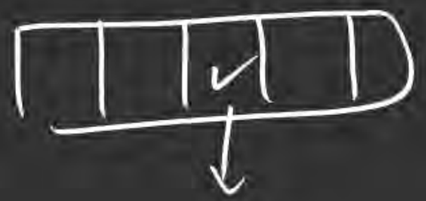
$2^k = n$ $\quad (k = \log_2 n)$

$T(n) = T(1) + c * \log_2 n$

$T(n) = a + c * \log_2 n$

$TC = O(\log_2 n)$

Best Case :



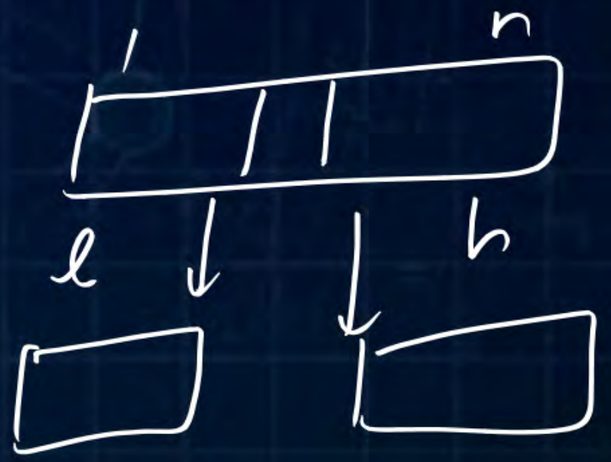
1 Comp $\longrightarrow \Omega(1)$

Worst Case

$\longrightarrow O(\log_2 n)$

Topic : Divide and Conquer

```
1.Algorithm BinSearch(a,n,x)        <ITERATIVE BINARY SEARCH>
2.// Given an array a[1: n] of elements in nondecreasing
3.//order, n > 0, determine whether x is present, and
4.// if so, return j such  that x  = a [j] ; else return 0.
5.{
 6.          low : = 1; high : = n;
 7.          While (low < high) do
 8.          {
 9.                mid: = [(low + high)/2] ;
10.                If (x < a[mid]) then high: = mid –1;
11.                else if (x > a[mid]) then low : = mid + 1;
12.                                Else return mid;
13.          }
14.          Return 0;
15.  }
```



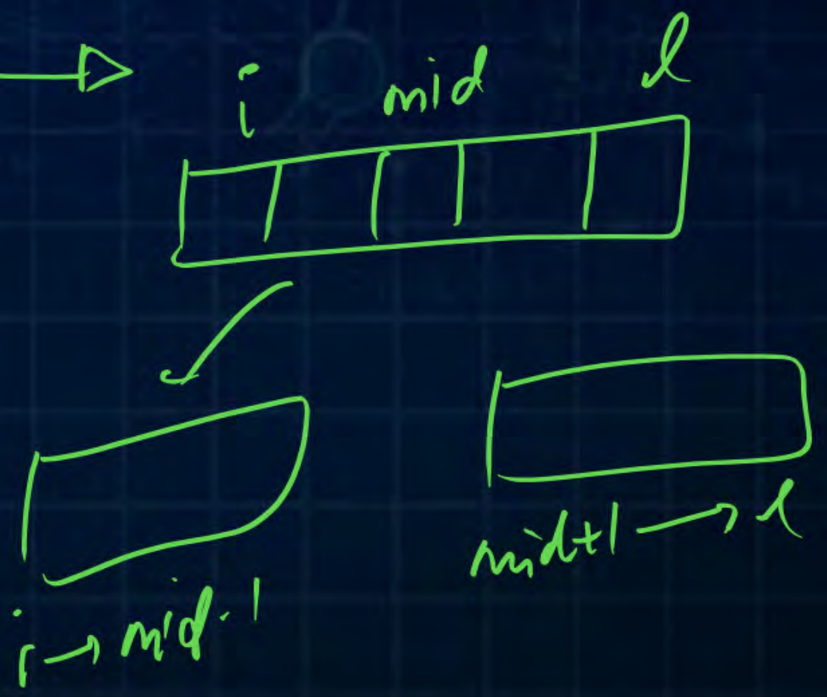
Topic : Divide and Conquer

```
1.      Algorithm BinSrch(a, i , l, x)
2.      // Given an array a[i : l] of elements in nondecreasing
3.      // order, 1 ≤ i ≤ l, determine whether x is present, and
4.    // if so, return j such that x = a[j]; else return 0.
5.      {
6.          if (l = i) then // If Small(P)
7.          {
8.                  if (x = a[i]) then return i;
9.                  else return 0;
10.     }
```

H.W

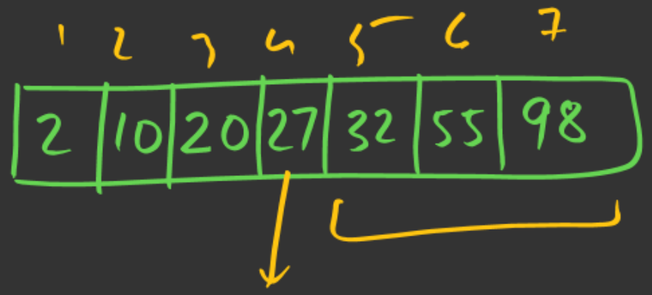
Topic : Divide and Conquer

```
11.     else
12.     { // Reduce P into a smaller subproblem.
13.     mid := [(i + l)/2];
14.     if (x = a[mid] then return mid;
15.     else if (x < a[mid]) then
16.                 return BinSrch(a, i, mid — 1,a:);
17.         else return BinSrch(a,mid + 1, l, x);
18.         }
19.     }
```



Binary Search

A:



Key = 55

27 < 55

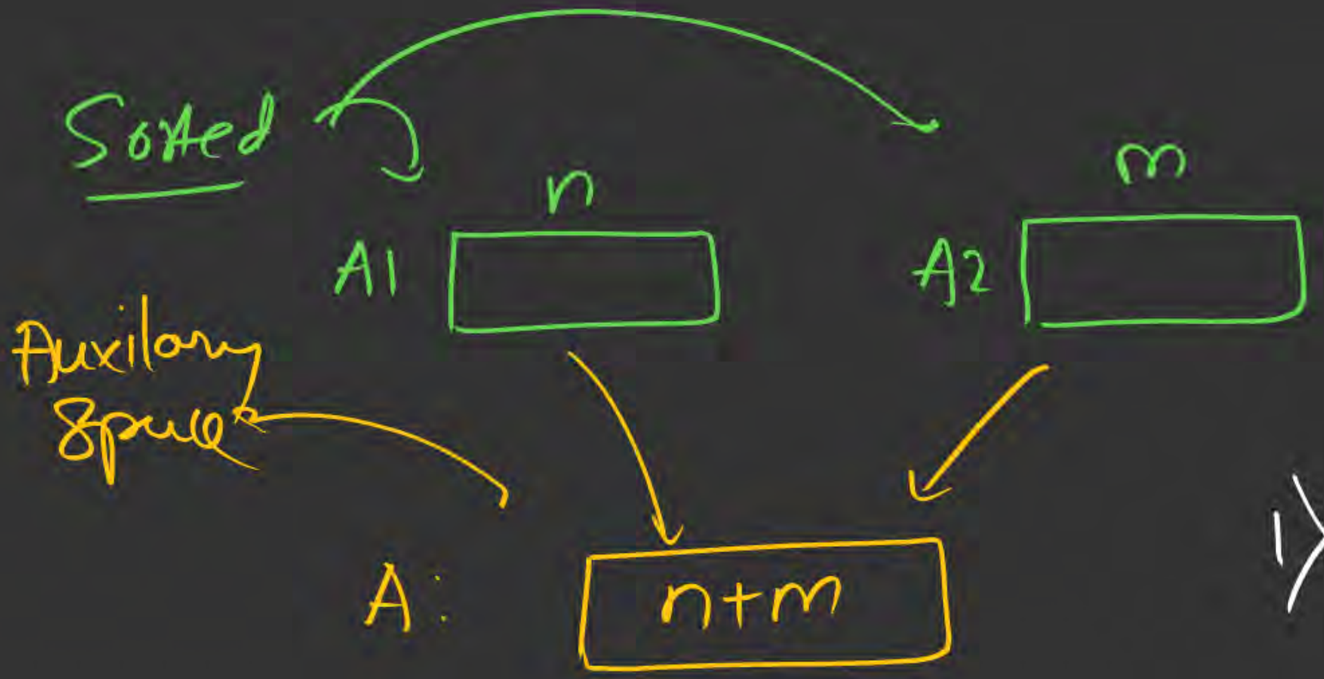
Comparisons = ?

① 55 = key ✓

② 2 Comp ✓

③ Merge Sort : DnC Sorting

↳ {Principle of merging} Conquer/Combine

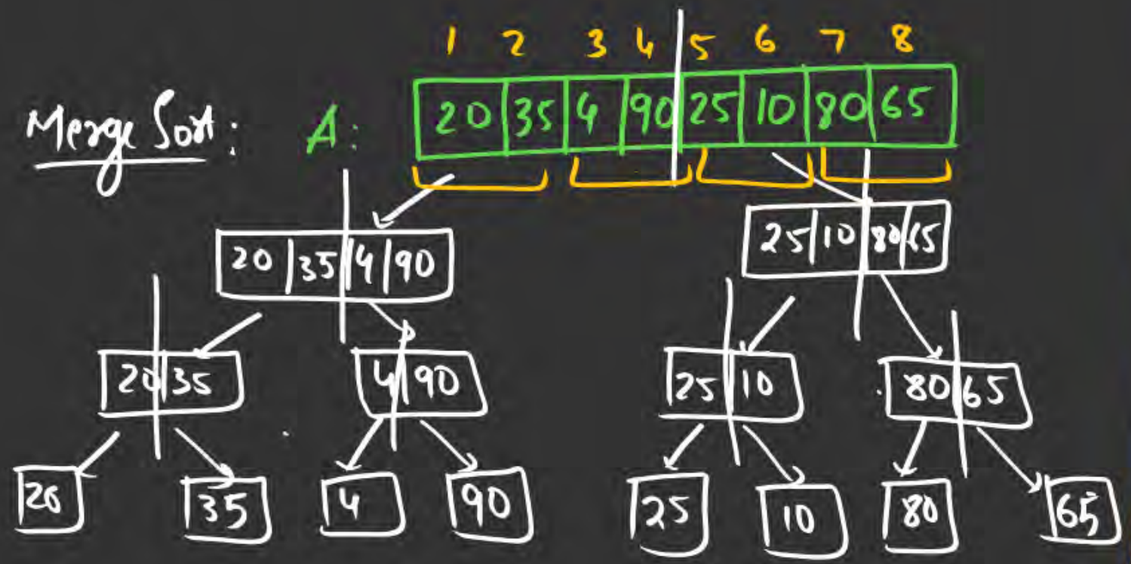


Default: (2-way merging)

No. of elem- Comparisons

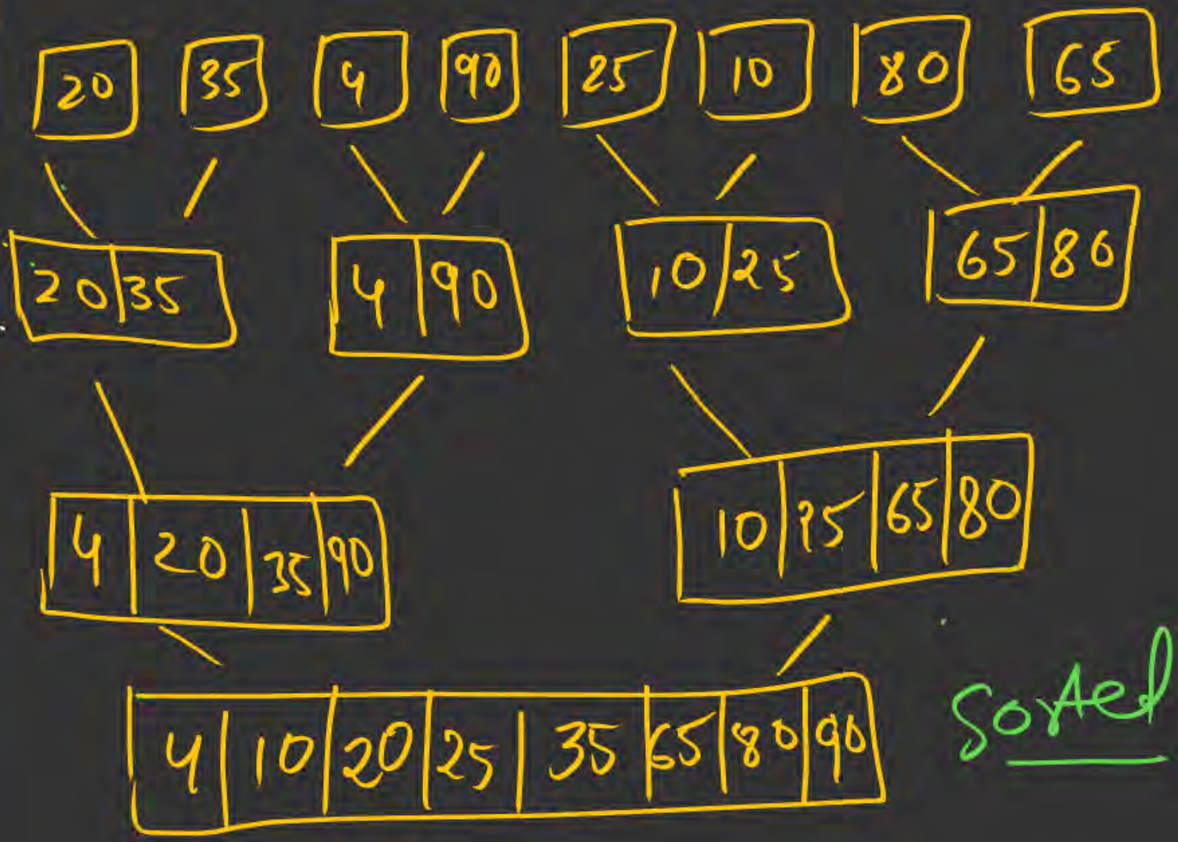
1) Minimum = $\min(n, m)$

2) Maximum = $(m + n - 1)$



Divide

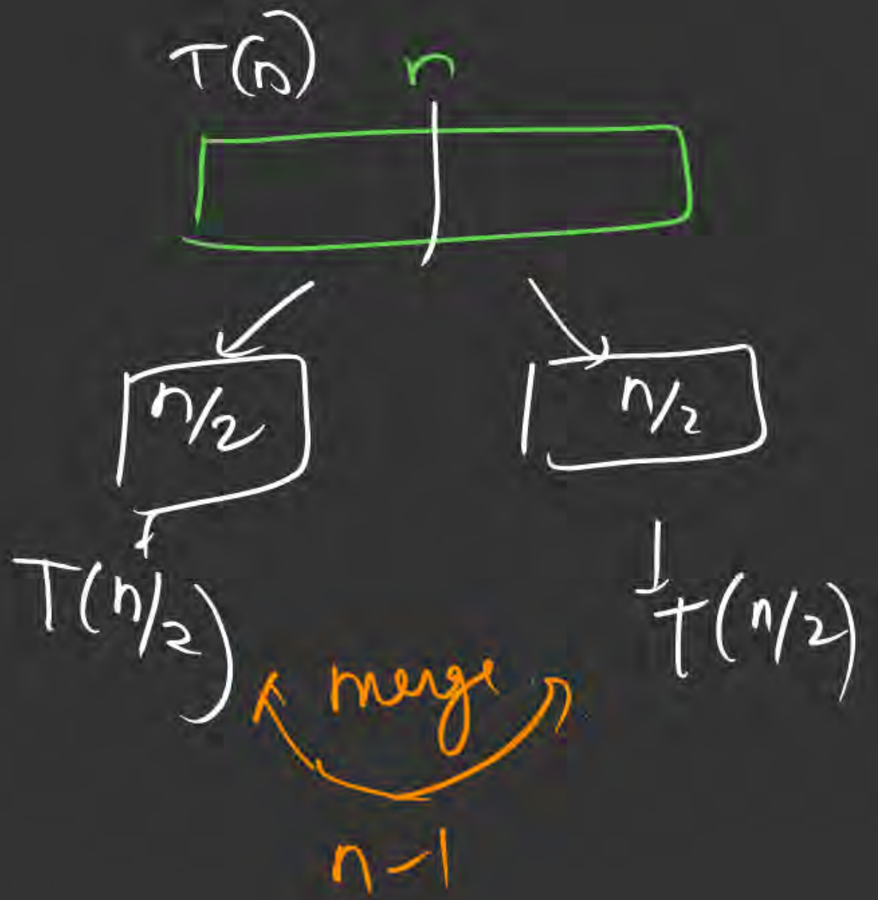
Combine



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

$$\boxed{T(n) = 2T(n/2) + (n-1)}$$

You solve

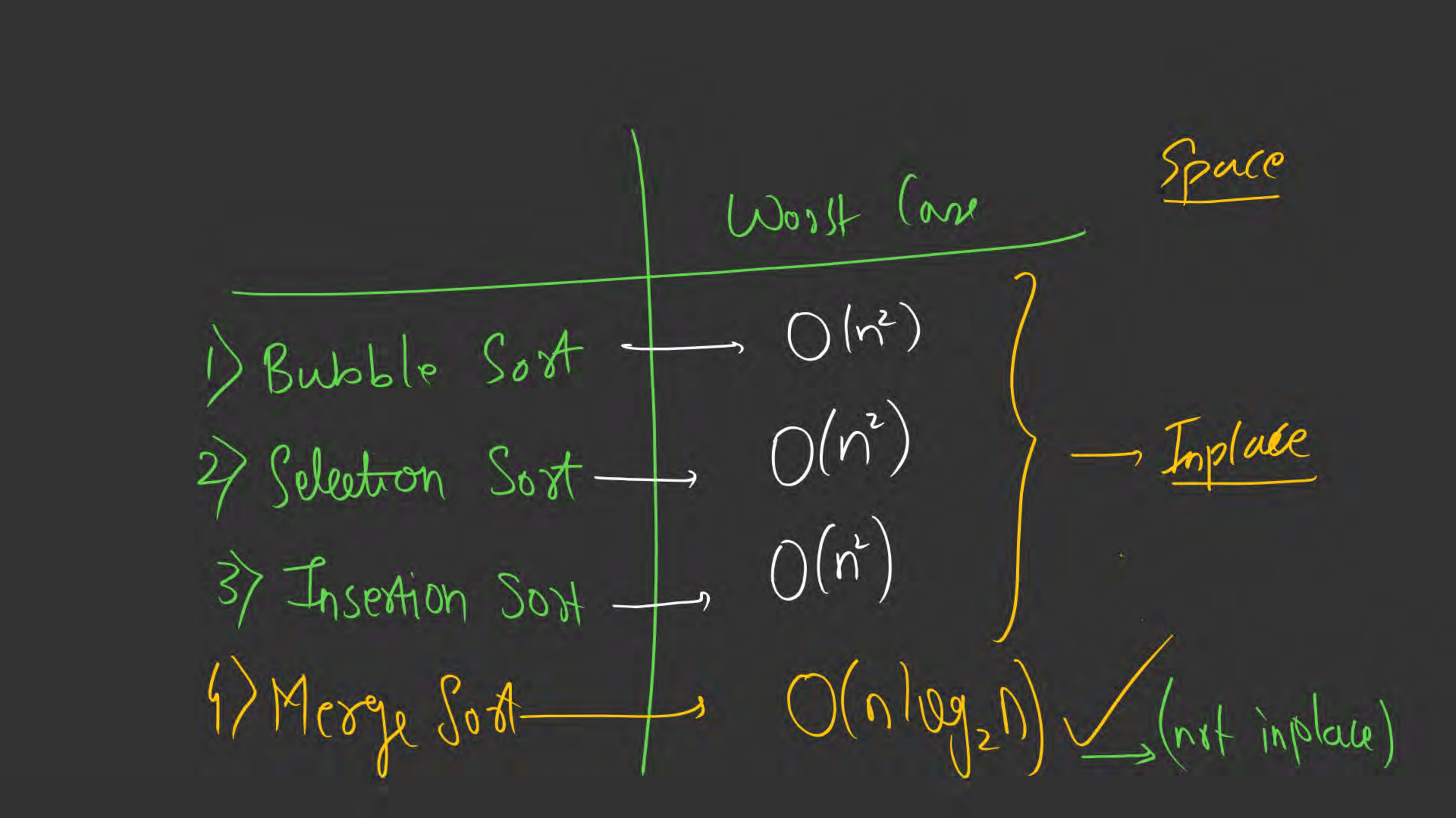


$TC = O(n \log_2 n)$

Best Case: $\Omega(n \log_2 n)$

Worst Case: $O(n \log_2 n)$

$O(n \log_2 n)$


Space
Worst Case
1) Bubble Sort → O(n²)
2) Selection Sort → O(n²)
3) Insertion Sort → O(n²)
→ Inplace
4) Merge Sort → O(n log₂n) ✓ (not inplace)

* Merge Sort Space Complexity

B(n)

1) Merging Algo : temp array — O(n)

2) Recursion Stack : $O(\log_2 n)$

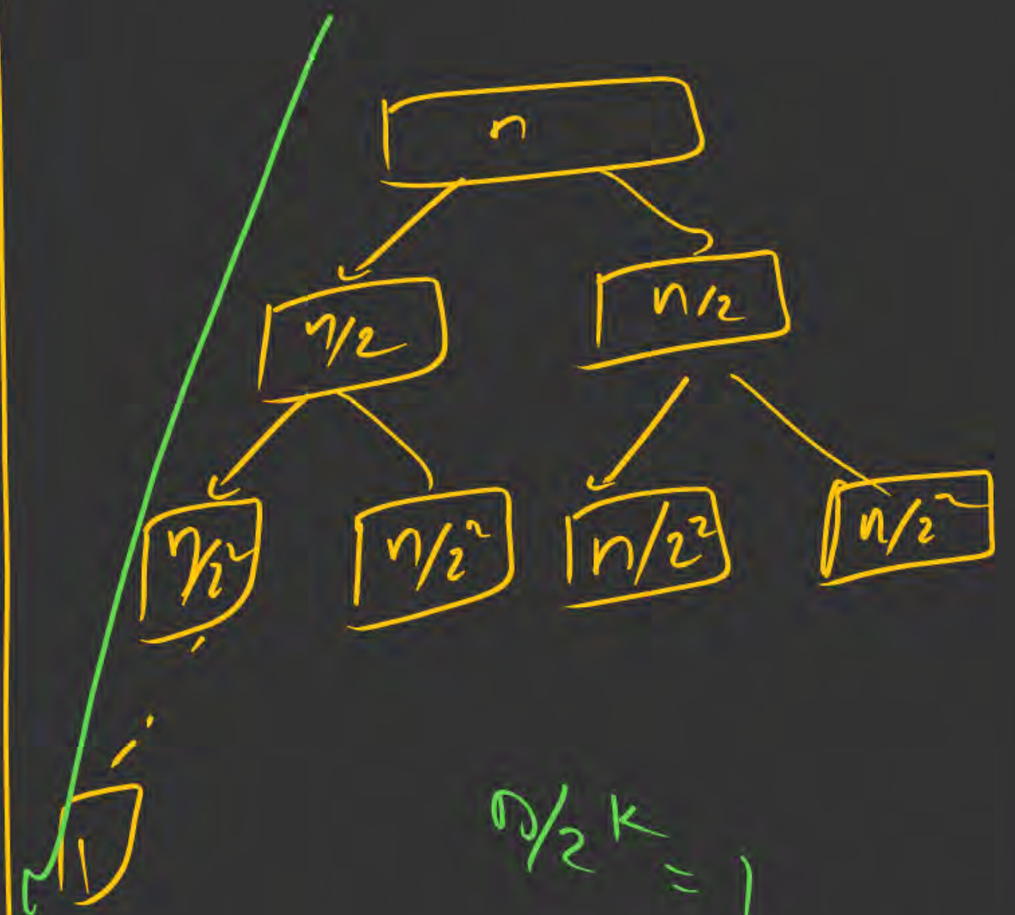
$O(n + \log n) = O(n)$

(not inplace)

$n^2 > n\log_2 n$

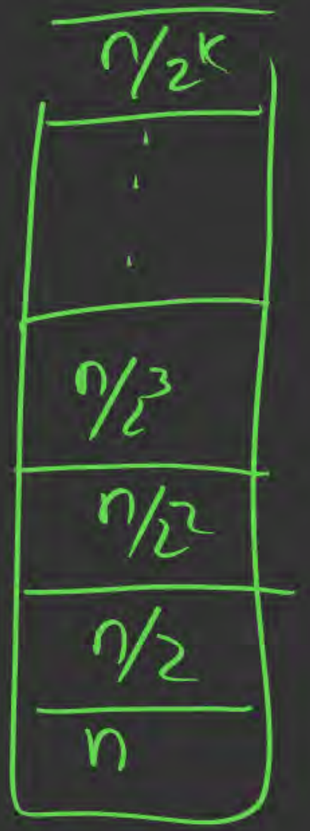
$n \times n > n \times \log n$

$n > \log n$



$n/2^k = 1$

$(k = \log_2 n)$



4) Quick Sort

↳ DnC Sorting

partitioning Algo (Divide)

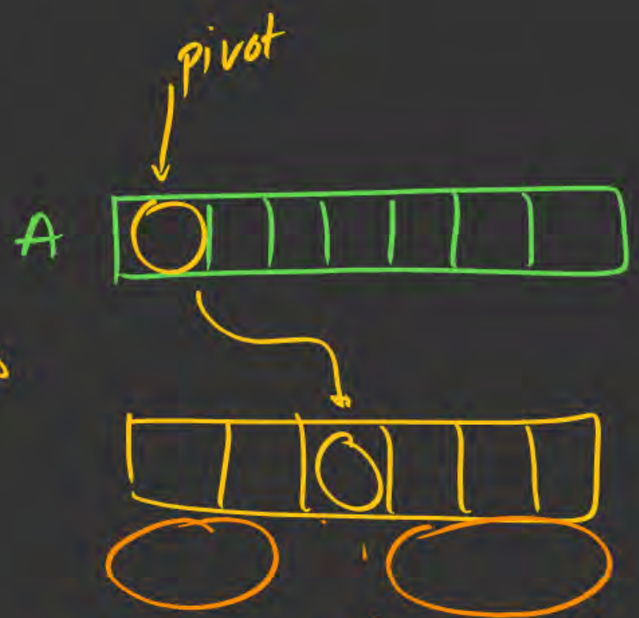
A[n]

↓

O(n)

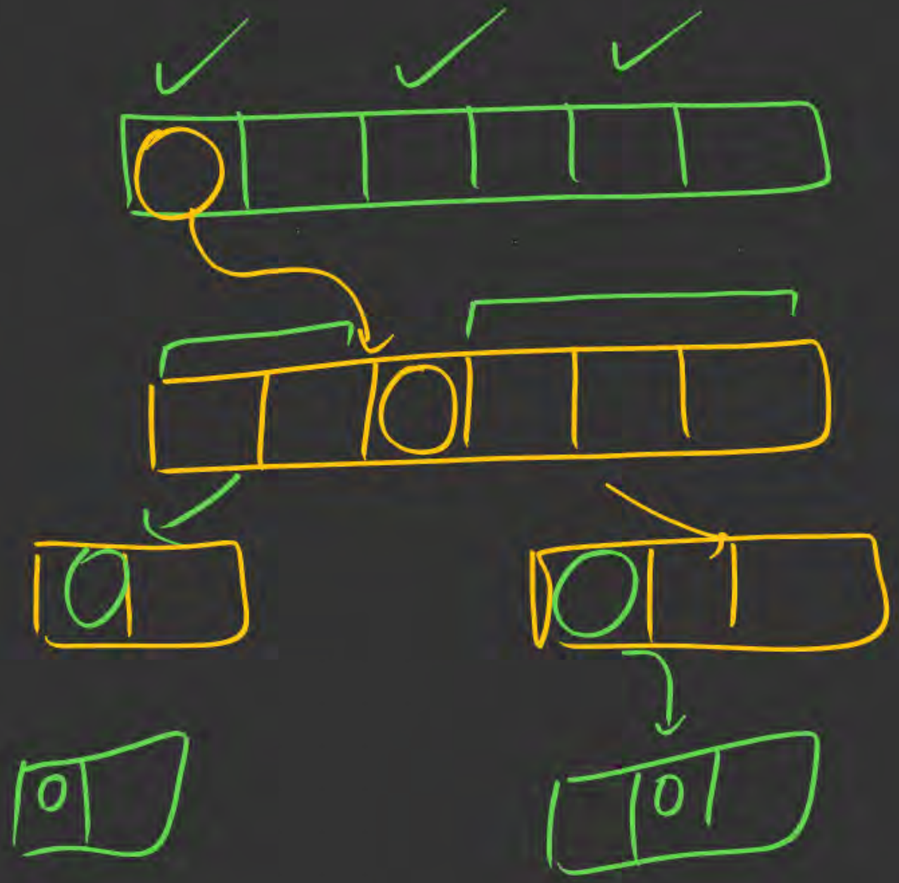
Partition Algo

default: (1st elem is pivot)



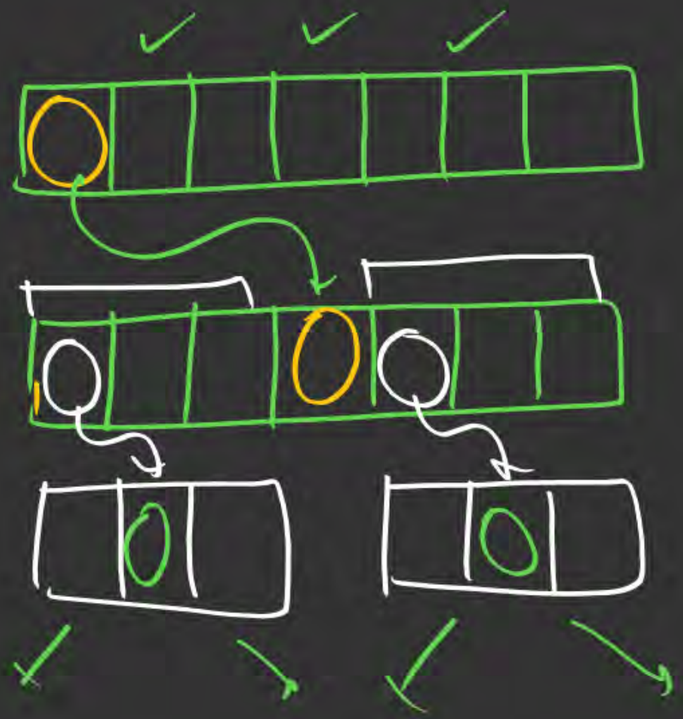
1) Pivot → gets placed at its correct position in sorted array
2) all elems (less than or equal to it) are on its left.
3) all elems (greater than " ") " " Right.

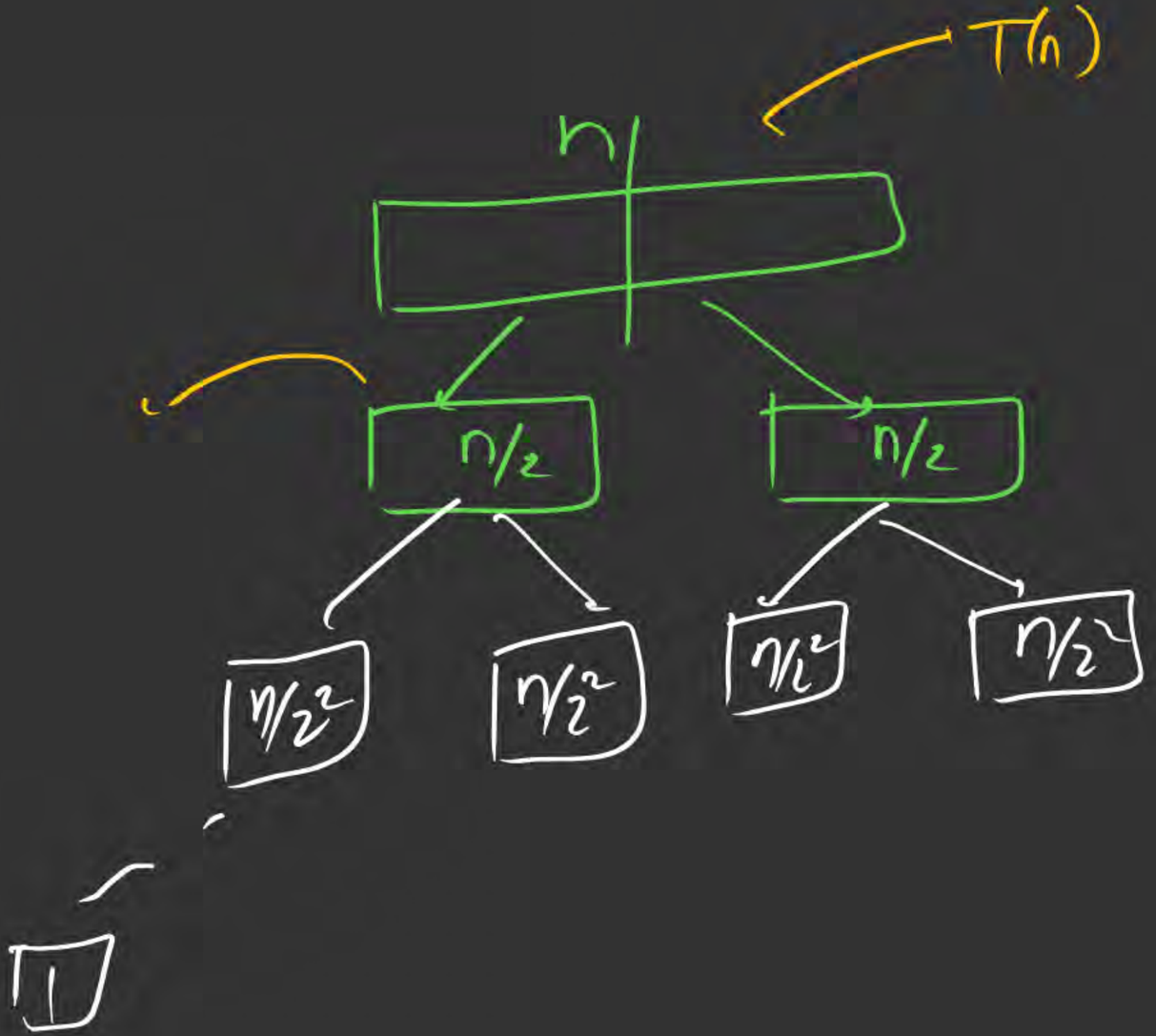
Quick Sort:

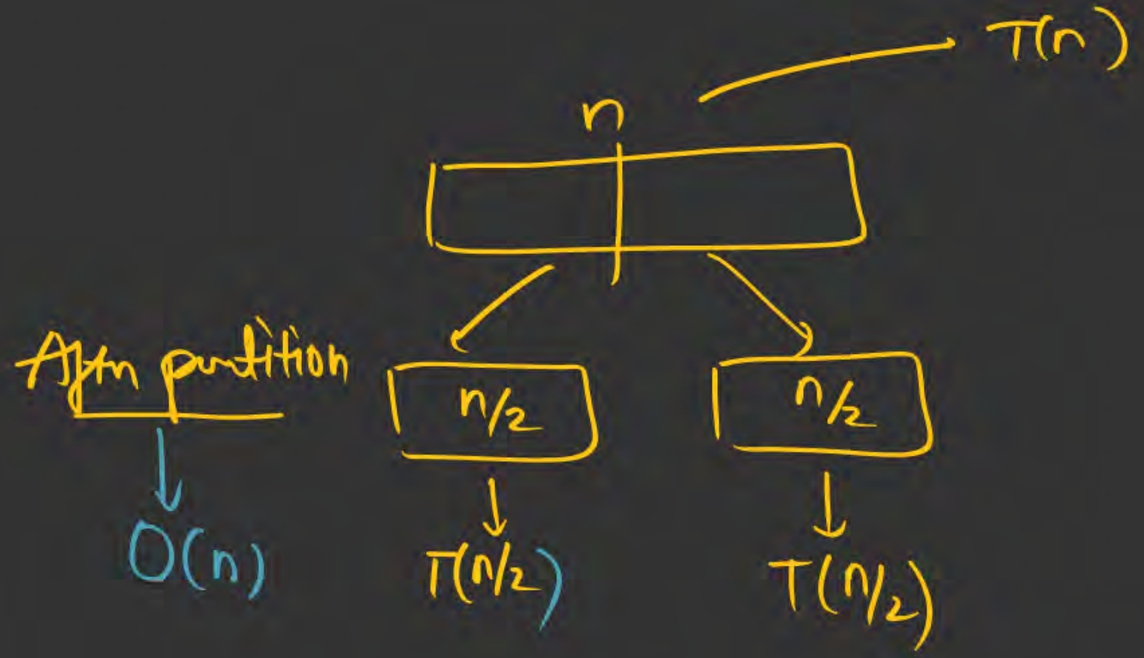


Case 1: Pivot always gets placed at mid position, after partitioning.

I/P : A :




T(n)
n
n/2
n/2
n/2²
n/2²
n/2²
n/2²
1



Partition

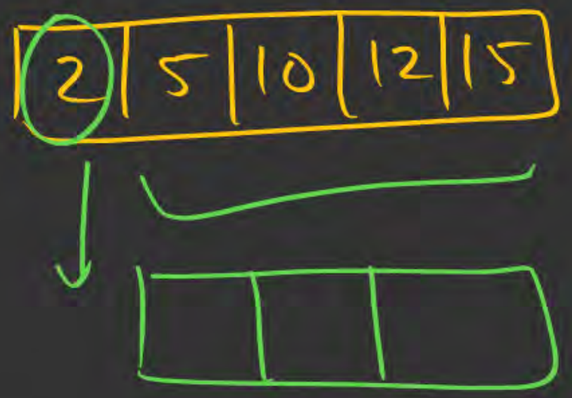
$$T(n) = T(n/2) + T(n/2) + O(n)$$

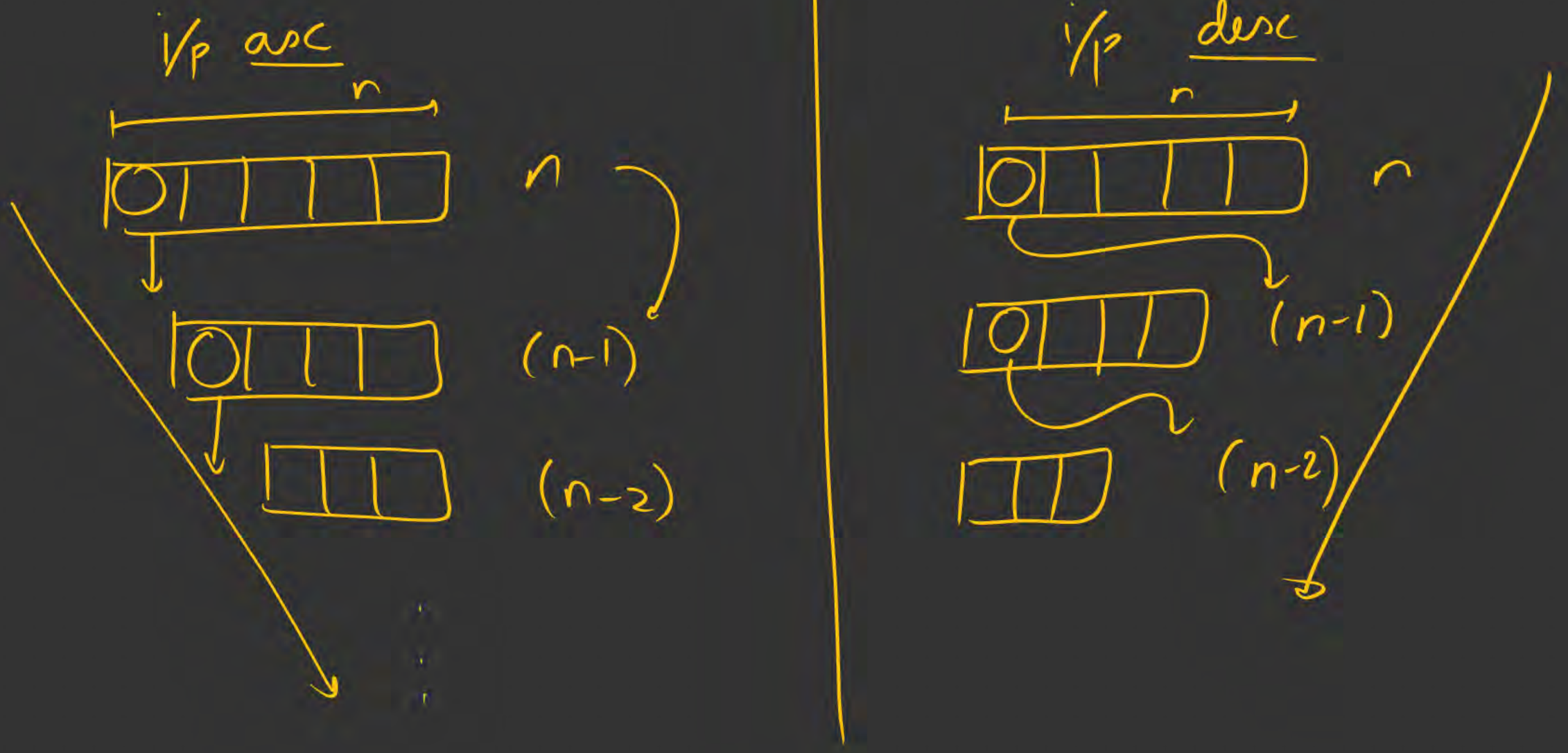
$$T(n) = 2T(n/2) + O(n)$$

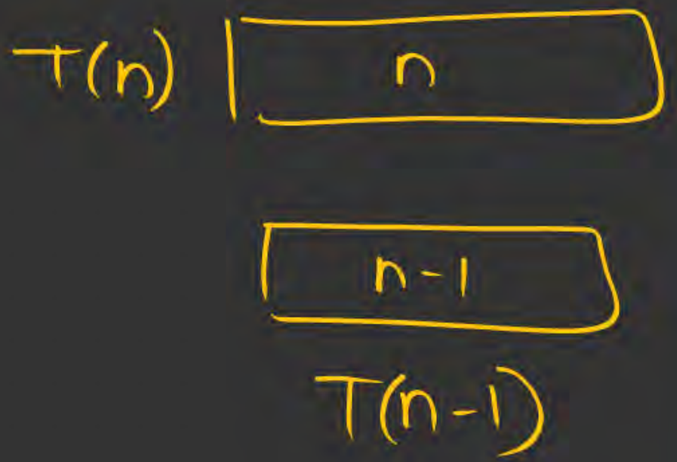
$$T(n) = 2T(n/2) + c * n$$

$$TC : O(n \log_2 n)$$

Case 2 : I/p array is already Sorted (Asc/desc)




i/p asc
n
n
(n-1)
(n-2)
i/p desc
n
n
(n-1)
(n-2)



Partition

$$T(n) = T(n-1) + O(n)$$

$$T(n) = T(n-1) + c * n$$

$T(n) = T(n-1) + c*n$

$T(n-1) = T(n-2) + c*(n-1)$

$T(n) = T(n-2) + c(n-1) + c(n)$

$T(n) = T(n-3) + c(n-2) + c(n-1) + c(n)$

general

$T(n) = T(n-k) + c*(n + (n-1) + \cdots + (n-(k-1)))$

For Base Condition

$n-k=1$

$k=(n-1)$

$$T(n)=T(n-k)+C[n+(n-1)\cdots\cdots(n-k+1)]$$

$$=T(1)+C[n+(n-1)\cdots\cdots+(n-(n-1)+1)]$$

$$=T(1)+C[\underbrace{n+(n-1)\cdots\cdots+2}]$$

$$=a+C\left[\frac{n(n+1)}{2}-1\right]$$

$$=a+C\left[\frac{n^2+n-2}{2}\right] \simeq \underline{O(n^2)}$$

(Unsorted i/p)

Case1 ⟶ $O(n \log_2 n)$ ⟶ Best Case : $\Omega(n \log n)$

Case2 ⟶ i/p is already sorted (Asc/desc) ⟶ $O(n^2)$

$= \underline{O(n^2)}$

Summary:

1) Framework DnC

2) Problems / Applications

Thank

THANK

Keep Hustling!