CS & IT ENGINEERING

Algorithms

Design Strategies

Lecture No.- 03

By- Dr. Khaleel Khan Sir

Recap of Previous Lecture

- **Topic** Binary Search
- **Topic** Merge Sort
- **Topic**
- **Topic**
- **Topic**

Topics to be Covered

Topic Merge Sort

Topic Quick Sort

Topic

Topic

Topic

→ **D and C :**

Problem

P_1 P_2 P_3 Subproblems

↓

Small (one/Two basic operation)

Time Complexity: D and C Recurrence Relation

1) $T(n) = a \cdot T(n/b) + f(n)$ → Combine + Divide + Small

a: No. of Subproblems $(a \geq 1)$

n/b: Size of each of the Subproblem

2) $T(n) = T(\alpha n) + T((1-\alpha)n) + f(n)$

3) $T(n) = T(\alpha n) + T(\beta n) + T(\gamma n) + f(n)$

---

a) Max-Min : $T(n) = 2T(n/2) + 2 \Rightarrow \left(\frac{3n}{2} - 2\right) : O(n)$

Space : $O(\log n)$

---

b) Bin-Search : $T(n) = T(n/2) + C \Rightarrow O(\log n)$

Space : $O(\log n)$

---

c) Mergesort : $T(n) = 2T(n/2) + n \Rightarrow \Theta(n.\log n)$

I/P A: [1 … 9] NOT-IN-PLACE

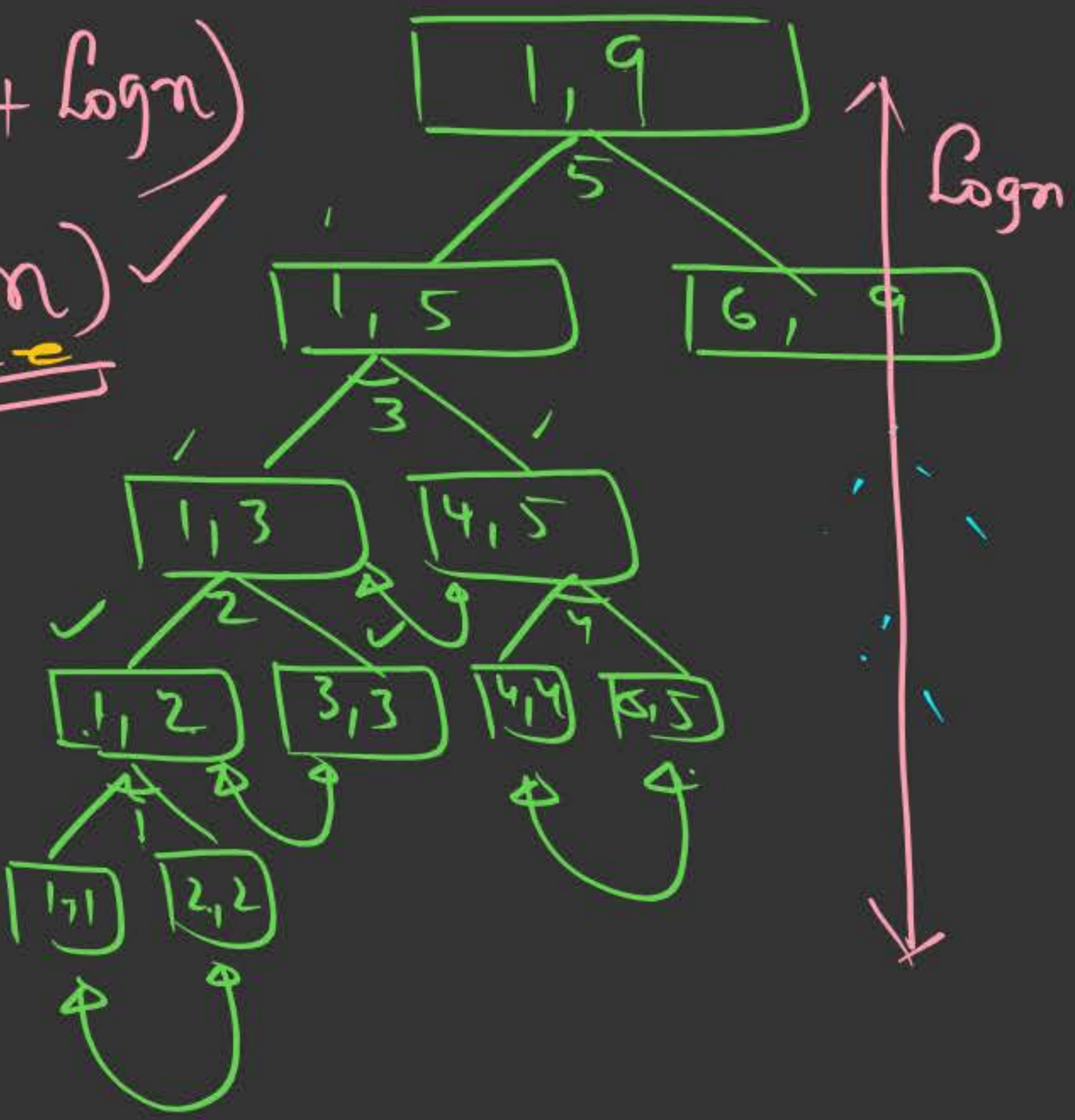
Aux B: ['n'] ⇐ Merging

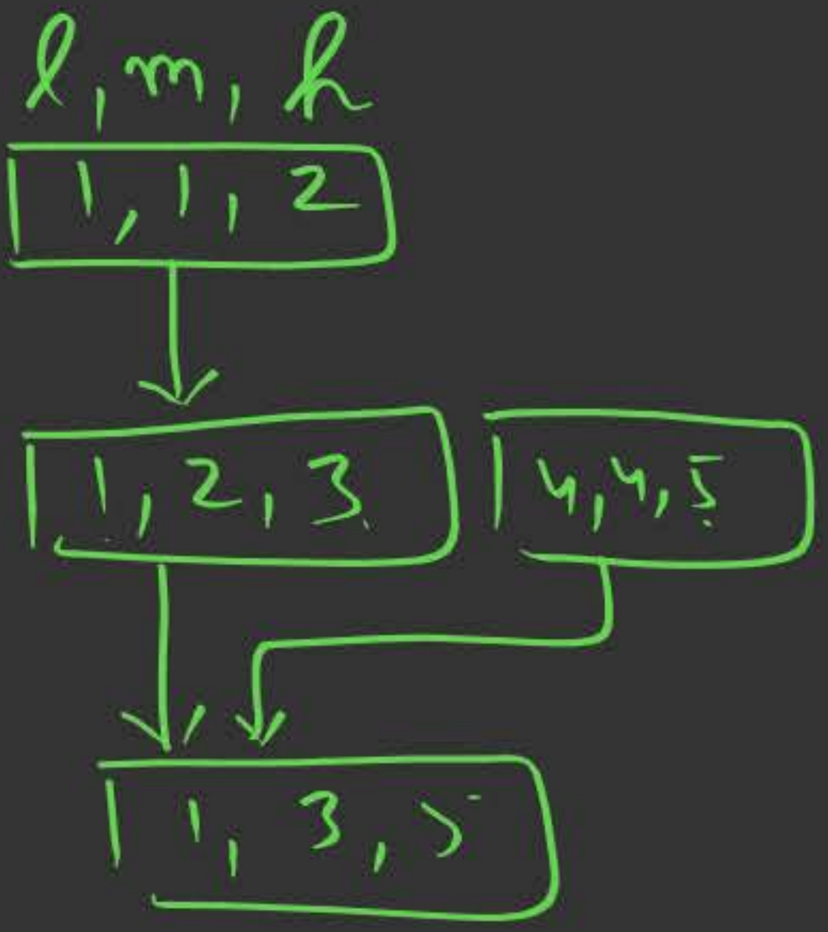
Space complexity

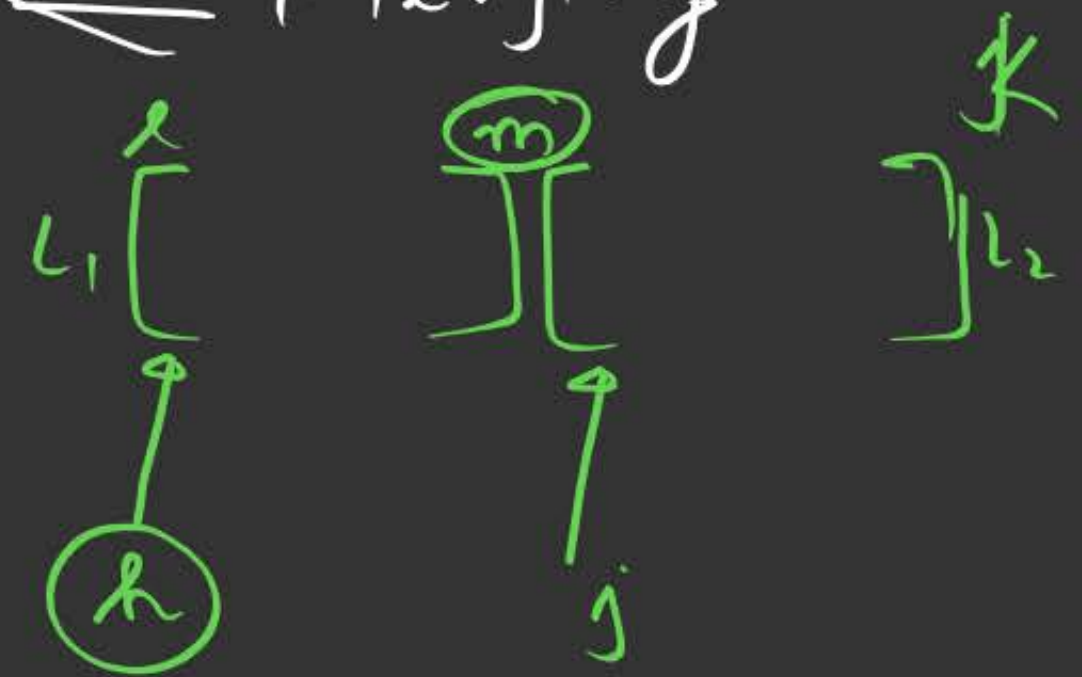
$= (n + \log n)$

$= O(n)$

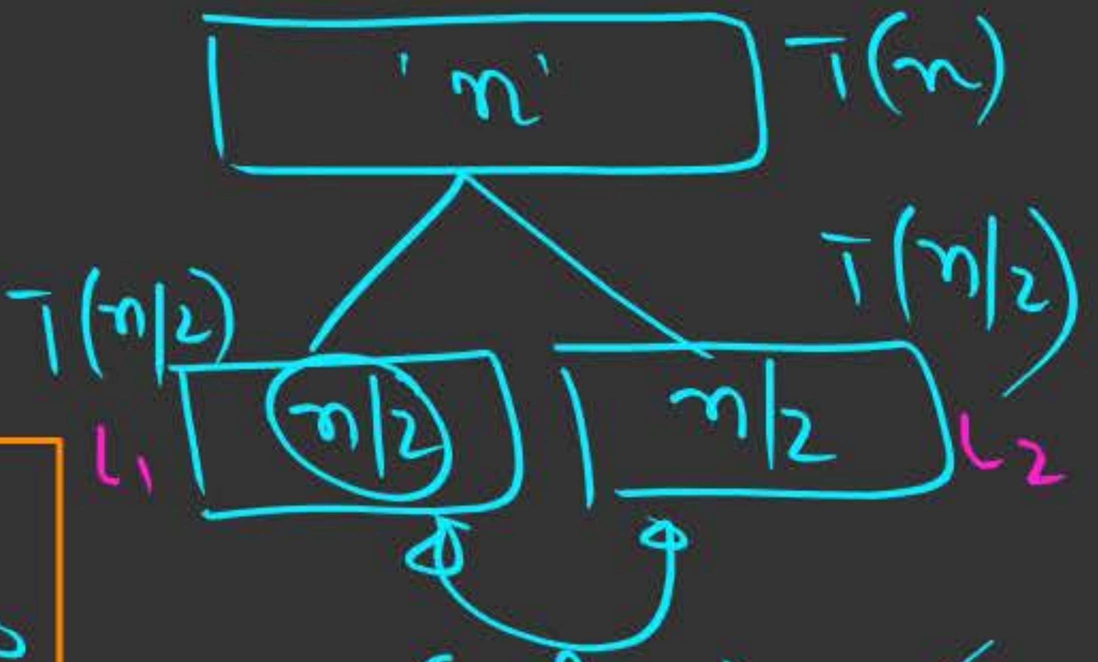
Divide







$$T(n) = c, \quad n = 1$$
$$= 2T(n/2) + bn, \quad n > 1, \; b > 0$$



$\Rightarrow \Theta(n \cdot \log n)$

1) Time : Alg. is efficient : Time is bounded by a Polynomial

2) Space : Alg is efficient :

Space requirement is atmost

$O(1)$ ✓

$O(\log n)$

recursion

Topic : Divide and Conquer

```
Algorithm MergeSort(low, high)
// a[low : high] is a global array to be sorted.
// Small(P) is true if there is only one element
// to sort. In this case the list is already sorted.
{
    if (low < high) then  // If there are more than one element
    {
        // Divide P into subproblems.
            // Find where to split the set.
                mid := ⌊(low + high)/2⌋;
        // Solve the subproblems.
            MergeSort(low, mid);
            MergeSort(mid + 1, high);
        // Combine the solutions.
            Merge(low, mid, high);
    }
}
```

Topic : Divide and Conquer

```
Algorithm Merge(low, mid, high)
// a[low : high] is a global array containing two sorted
// subsets in a[low : mid] and in a[mid + 1 : high]. The goal
// is to merge these two sets into a single set residing
// in a[low : high]. b[ ] is an auxiliary global array.
{
    h := low; i := low; j := mid + 1;
    while ((h ≤ mid) and (j ≤ high)) do
    {
        if (a[h] ≤ a[j]) then
        {
            b[i] := a[h]; h := h + 1;
        }
        else
        {
            b[i] := a[j]; j := j + 1;
        }
        i := i + 1;
    }
    if (h > mid) then
        for k := j to high do
        {
            b[i] := a[k]; i := i + 1;
        }
    else
        for k := h to mid do
        {
            b[i] := a[k]; i := i + 1;
        }
    for k := low to high do a[k] := b[k];
}
```

Algorithm 3.8 Merging two sorted subarrays using auxiliary storage

V2 : Bottom-up Mergesort / 2-way Mergesort :

Time Complexity: $n \cdot \log n$

A[i] = [ ]

list

Pass 3: [179 254 285 310 351 423 652 861] ✓

8

$n = 2^k$

$k = \log n$

Pass 2: [179 285 310 652] [254 351 423 861]

1 2

$n/4 * 2$

Pass 1: [285 310] [179 652] [351 423] [254 861]

1 2 3 4

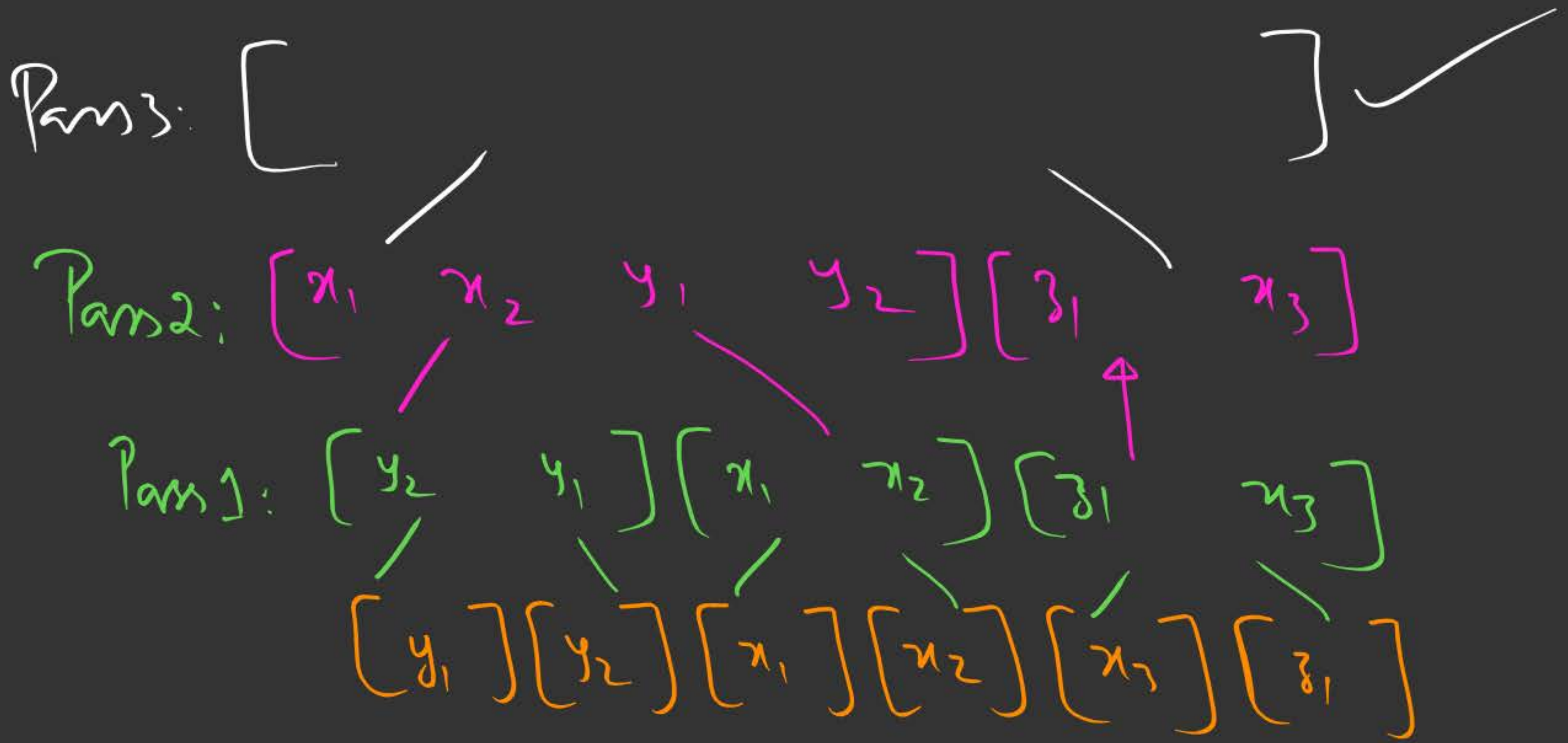
$n/4 * 3$

$n/2 * 1$

A: [310] [285] [179] [652] [351] [423] [861] [254]

1 2 3 4 5 6 7 8

$n = 2^3$



Pass 3: [ ]
Pass 2: [x_1 x_2 y_1 y_2] [z_1 x_3]
Pass 1: [y_2 y_1] [x_1 x_2] [z_1 x_3]
[y_1] [y_2] [x_1] [x_2] [x_3] [z_1]

Challege Q's :

2-way-Merge Sort

Calculate the Minimum & Maximum no. of Element Comparisons involved in 2-way-Merge Sort, Assuming $n = 2^K$; $(K > 0)$

Super Q :
Challenge

" n is not a power 2 ;

3) For merging two sorted lists of sizes m and n into a sorted list of size m+n, we require comparisons of

(a) O(m)
(b) O(n)
(c) O(m+n)
(d) O(log m+ log n)

Slide 6

2) If one uses straight two-way merge sort algorithm to sort the following elements in ascending order:

20, 47, 15, 8, 9, 4, 40, 30, 12, 17

then the order of these elements after second pass of the algorithm is:

(a) 8, 9, 15, 20, 47, 4, 12, 17, 30, 40

(b) 8, 15, 20, 47, 4, 9, 30, 40, 12, 17

(c) 15, 20, 47, 4, 8, 9, 12, 30, 40, 17

(d) 4, 8, 9, 15, 20, 47, 12, 17, 30, 40

[8 15 20 47] [ ] [4]

Pass1 : [20 47] [8 15] [4 9] [30 40] [12 17]

A : [20] [47] [15] [8] [9] [4] [40] [30] [12] [17]

1. Assume that Merge Sort takes 30sec to Sort 64 elements in worst case. What is the approximate number of elements that can be Sorted in the Worst Case using Merge Sort using 6 minutes? (n)

512 elements

$(n \cdot \log n)$ units

1) $n = 64 \longrightarrow T(\ ) = 30s$ (Platform dep.)

Apriori : $(64 * \log_2 64)$ M.S

$= (6 \times 64 \text{ units}) \longrightarrow 30\,s$

$1 \text{ unit} \longrightarrow ?$

$\frac{30}{6 \times 64} s \longrightarrow 1 \text{ unit}$

$360\,s \longrightarrow ?$

$\frac{360 \times 6 \times 64}{30}$ units $\Longrightarrow$ 'n' elements

4608

$\boxed{4608 \text{ units} \longrightarrow \text{'n' elements}}$

$n \cdot \log n = 4608$

$K \cdot 2^K = 4608$

$9 \times 512 = 4608$

Let $n = 2^K$

$n = 2^9$

$K = 9$

$n = 512$ ✓

Slide 8

4) Quick Sort [Partition-Exchange Sort] : Tony Hoare

→ Quick Sort is based on Partitioning (Pivoting) Process;

→ Partitioning Process [Divide]

Pivot = 65

O(n)

	1	Median 2	3	4	5	6	7	8	9
A :	65	70	75	80	85	60	55	50	45

(60 55 50 45) $\boxed{65}$ (70 75 80 85)

(i) Find the correct place of Pivot in the Final Sorted list;

(ii) Ensures that all elements that less than Pivot are placed to its left

(iii) " " " " " greter " " " " " " Right

A:

1	2	3	4	5	6	7	8	9	10
60	45	50	55	65	85	80	75	70	∞ (9999)

100

L   j   i   R

i

$\Rightarrow$ Impl-g QS requires an additional element (n+1) initialized to ∞ (9999)

$\rightarrow$ (Inf. loop)

$\leftarrow$ 65

i	j
1	10
2	9
3	8
4	7
5	6
6	5
10	9

Topic : Divide and Conquer

```
                         1  10
Algorithm QuickSort  (p, q)
// Sorts the elements a[p],..., a[q] which reside in the global
// array a[1 : n] into ascending order: a[n + 1] is considered to
// be defined and must be ≥ all the elements in a[1 : n].
{
    if (p < q) then // If there are more than one element
{
    // divide P into two subproblems.
      5                  1  1   10
      j := Partition(a, p, q + 1);
            // j is the position of the partitioning element.
```

Topic : Divide and Conquer

```
        // Solve the subproblems.
                                 1    4
            L:  QuickSort(p, j — 1);
            R:  QuickSort(j + 1,q);
                             6    10
            //There is no need for combining solutions.
        }
}
```

Topic : Divide and Conquer

1 10

```
Algorithm Partition  (a,m,p)
// Within a[m],a[m+1] ,....., a[p–1] the elements are
// rearranged in such a manner that if initially t = a[m],
// then after completion a[q] = t for some q between m
// and p – 1, a[k] ≤ t for m ≤ k < q, and a[k] > t
// for q < k < p. q Is returned. Set a[p] = ∞.
{
    v : = a[m]; i := m; j := p;
     repeat
     {
```

Repeat {

Loop

i
1

```
    repeat
          i:=i + 1;
    until (a[i] ≥ v);
    repeat
          j : = j + 1;
    until (a [j] ≥ v);
    if (i < j) then Interchanged (a,i , j);
} until (i ≥ j);
a[m] := a[j]; a[j] := v; return j;
}
```

L→R

R-L

Elem. Comparism

A 'n'

(n-1)

Time Complexity of

Partition: O(n)

Topic : Divide and Conquer

```
Algorithm Interchange (a,i,j)
// Exchange a[i] with a[j].
{
    p := a[i];
    a[i] := a[j]; a[j] := p;
}
```

Topic : Divide and Conquer

```
Algorithm QuickSort(p, q)
// Sorts the elements a[p], ..., a[q] which reside in the global
// array a[1 : n] into ascending order; a[n + 1] is considered to
// be defined and must be ≥ all the elements in a[1 : n].
{
    if (p < q) then  // If there are more than one element
    {
        // divide P into two subproblems.
            j := Partition(a, p, q + 1);
                // j is the position of the partitioning element.
        // Solve the subproblems.
            QuickSort(p, j − 1);
            QuickSort(j + 1, q);
        // There is no need for combining solutions.
    }
}
```

Topic : Divide and Conquer

```
Algorithm Partition(a, m, p)
// Within a[m], a[m+1], ..., a[p-1] the elements are
// rearranged in such a manner that if initially t = a[m],
// then after completion a[q] = t for some q between m
// and p-1, a[k] ≤ t for m ≤ k < q, and a[k] ≥ t
// for q < k < p. q is returned. Set a[p] = ∞.
{
    v := a[m]; i := m; j := p;
    repeat
    {
        repeat
            i := i + 1;
        until (a[i] ≥ v);            L-R

        repeat
            j := j - 1;
        until (a[j] ≤ v);            R-L

        if (i < j) then Interchange(a, i, j);

    } until (i ≥ j);

    a[m] := a[j]; a[j] := v; return j;
}
```

```
Algorithm Interchange(a, i, j)
// Exchange a[i] with a[j].
{
    p := a[i];
    a[i] := a[j]; a[j] := p;
}
```

Slide 15

Performance :

(i) Time Complexity :

$T(n)$ $I : \langle n, a_1\, a_2 \, \ldots \, a_n \rangle$

Partition : $O(n)$

Best Case

Worst Case

R. Stack

Faster

I (unsorted)

Similar II

Slower

$(\log n)$

$I_1 : \langle n, (n/2)\, \boxed{a_1}\, (n/2) \rangle$

L R

$I_2 \langle (\frac{n-1}{(n-2)}) \boxed{a_1} \rangle$

dec

2^{nd}

$([\ ]\,(\cdot)\,[\ ])\,\boxed{a_1}\,((\ )\,\boxed{\cdot}\,(\ ))$

$I_3 \langle \boxed{a_1} (\ n-1\ \boxed{\cdot}\ (n-2)\ ) \rangle$

Inc

M.S

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

$O(n \cdot \log n)$

(Sorted order)

$T(n) = O(n) + T(n-1) \Rightarrow O(n^2)$

Note: QS behaves in W.C, elements are already sorted

Space Complexity :

1) **Best-Case :** $O(\log n)$

2) **Worst Case :** $O(n)$

1. In using Quick Sort suppose the central element of the Array is always chosen as the Pivot then the worst case complexity of the Quick Sort may be $O(n^2)$.

A	1	2	3	.	m			n
	y				x			
→	x				y			

2. The Median on Array of size n can be found in O(n) time. If Median is selected as Pivot, then the worst case complexity of Quick Sort is ______.

$$\boxed{T(n) = \underbrace{O(n)}_{\text{Median}} + \underbrace{O(n)}_{\text{Partitioning}} + 2T(n/2)}$$

: $O(n \cdot \log n)$

Smallest

	1	2	3	4	5	6	7
A	8	10	15	25	30	40	50

3. In applying Quick Sort to an unsorted list if (n/4) the element is selected as Pivot then the Time Complexity of Quick Sort will be______.

L		R
n/4		3n/4

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

$T(n/4) + T(3n/4) \Rightarrow O(n \log n)$ ✓

Slide 16

4. Consider the Quicksort algorithm. Suppose there is a procedure for finding a pivot element which splits the list into two sublists each of which contains at least one-fifth of the elements. Let T(n) be the number of comparisons required to sort n elements. Then

(a) $T(n) \le 2T(n/5)+n$

(b) $T(n) \le T(n/5)+T(4n/5)+n$

(c) $T(n) \le 2T(4n/5)+n$

(d) $T(n) \le 2T(n/2)+n$

$$T(n) = O(n) + T(n/5) + T(4n/5)$$

THANK - YOU