

CS & IT ENGINEERING

Algorithms

Design Strategies

Lecture No. - 02

By- Dr. Khaleel Khan
Sir

Recap of Previous Lecture



Topic

Divide and Conquer - Introduction

Topic

Max – Min Problem

Topic

Topic

Topic

Topics to be Covered



Topic

Binary Search

Topic

Merge Sort

Topic

Topic

Topic

for $i \leftarrow 1$ to $(n-1)$
 for $j \leftarrow (i+1)$ to n
 for $k \leftarrow 1$ to j
 $c = c + 1; *$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^j O(1)$$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n j$$

$$i=1 \quad i=2 \quad i=3$$

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

for $i \leftarrow 1$ to n
 $c = c + 1; \Rightarrow \sum_{i=1}^n O(1)$
 $i=1 \quad i=n-1$

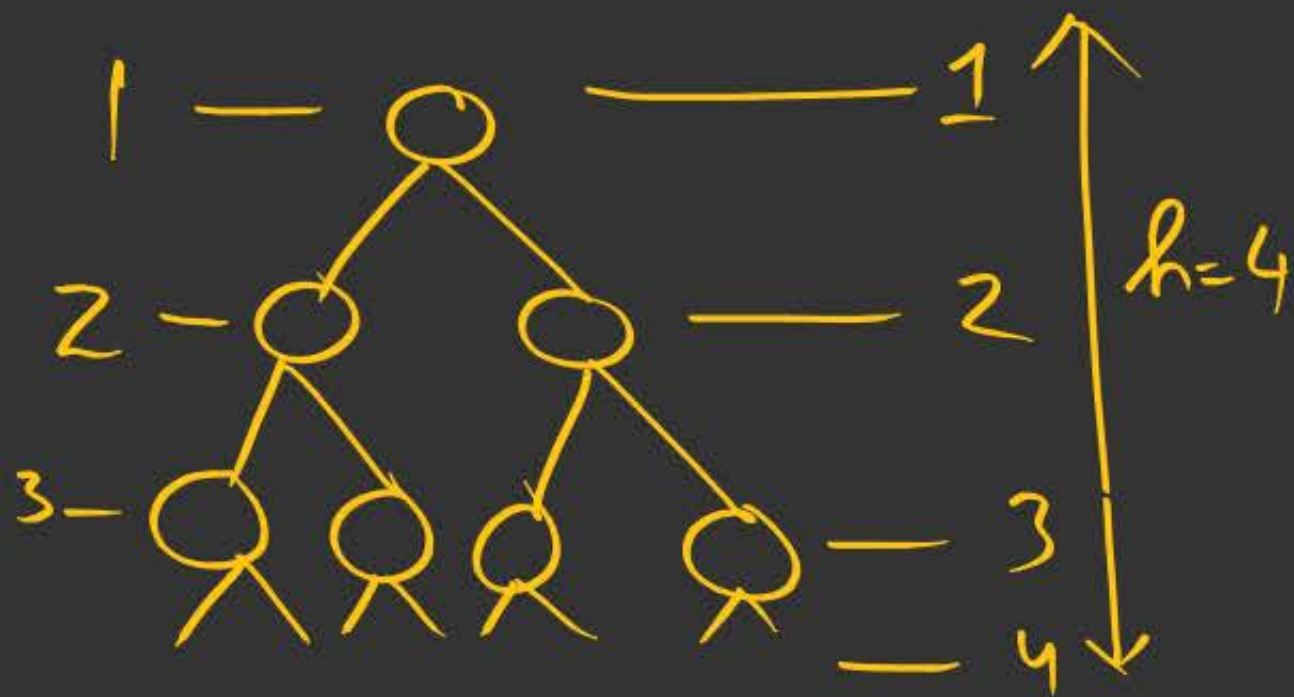
$$\left(\frac{n(n+1)}{2} - 1 \right) + \frac{n(n+1)}{2} - (1+2) + \dots + \frac{n(n+1)}{2} - (1+2+\dots+n-1)$$

$$\frac{n(n+1)}{2} (n-1) - \left[1 + (1+2) + (1+2+3) + \dots + (1+2+3+\dots+n-1) \right]$$

$$= \frac{(n-1)(n)(n+1)}{2} - \sum_{j=1}^{n-1} \left(\frac{j(j+1)}{2} \right)$$

$$= \frac{(n-1)(n)(n+1)}{2} - \frac{1}{2} \left[\sum_{j=1}^{n-1} j^2 + \sum_{j=1}^{n-1} j \right] = \frac{n(n+1)(n-1)}{3}$$

Q) Given a Full Binary Tree with n -nodes, then the height/depth of the tree is _____;



TIFR

→ Max. # of Nodes @ any level ' i ' of a Binary Tree $= 2^{i-1}$

→ Total No. of Nodes in a Binary Tree of height ' h ' $= \sum_{i=1}^h 2^{i-1}$

$$n = \sum_{i=1}^h 2^{i-1} = \frac{1}{2} \sum_{i=1}^h 2^i = \frac{2^{h+1} - 2}{2} \quad S_n = \frac{a(r^n - 1)}{r - 1}$$

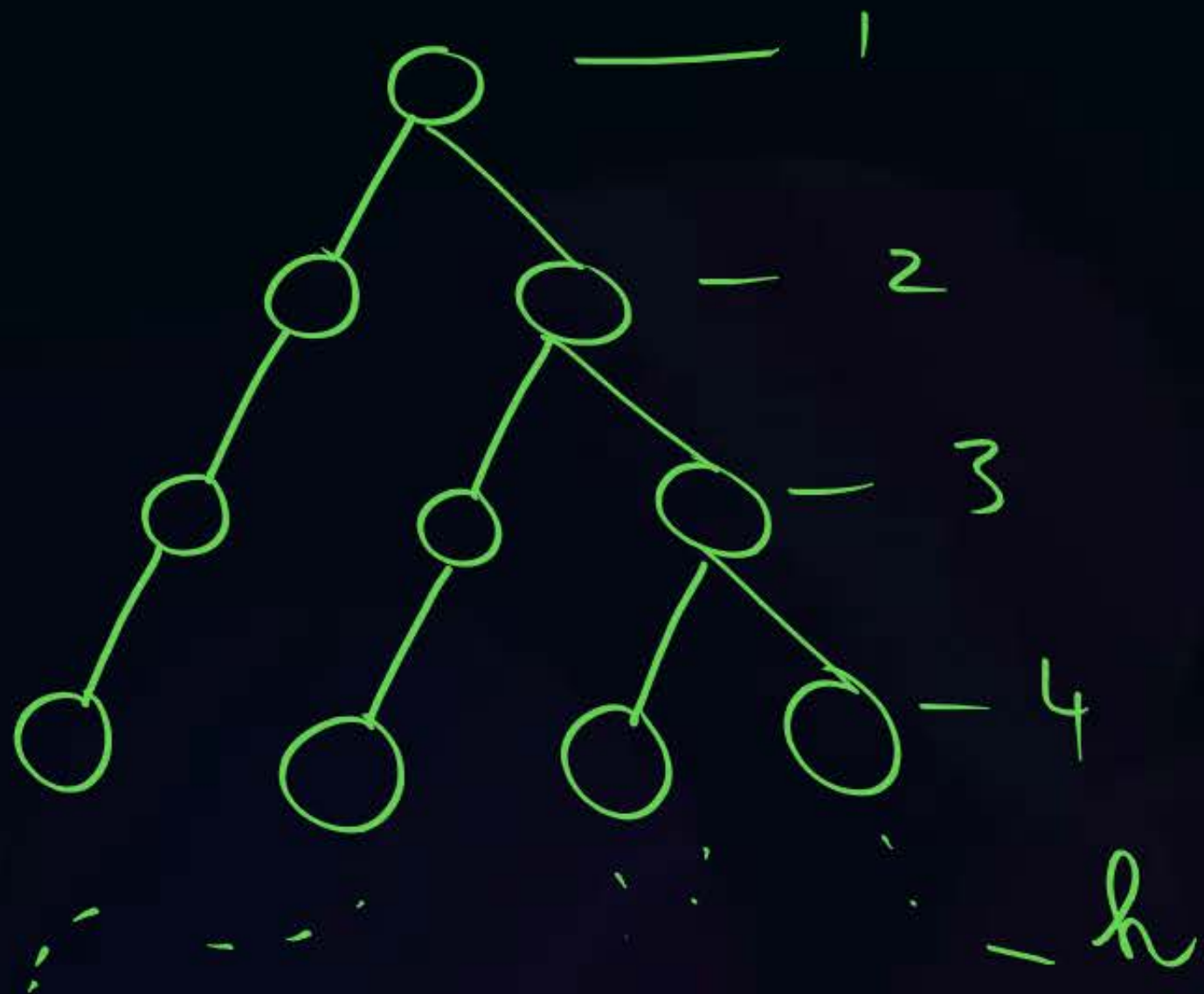
$$= \frac{2(2^h - 1)}{2 - 1}$$

$$n = \frac{1}{2} [2^{h+1} - 2] = 2^h - 1$$

$$\therefore n = 2^h - 1 \Rightarrow n + 1 = 2^h$$

$$h = \log_2(n+1) = O(\log_2 n)$$

Q) Consider a Binary Tree where root is at level 1 and each other level 'i' of the binary Tree has exactly 'i' nodes. The height of such a binary Tree having 'n' nodes is order of _____.



$$n = \sum_{i=1}^h i = \frac{h(h+1)}{2}$$

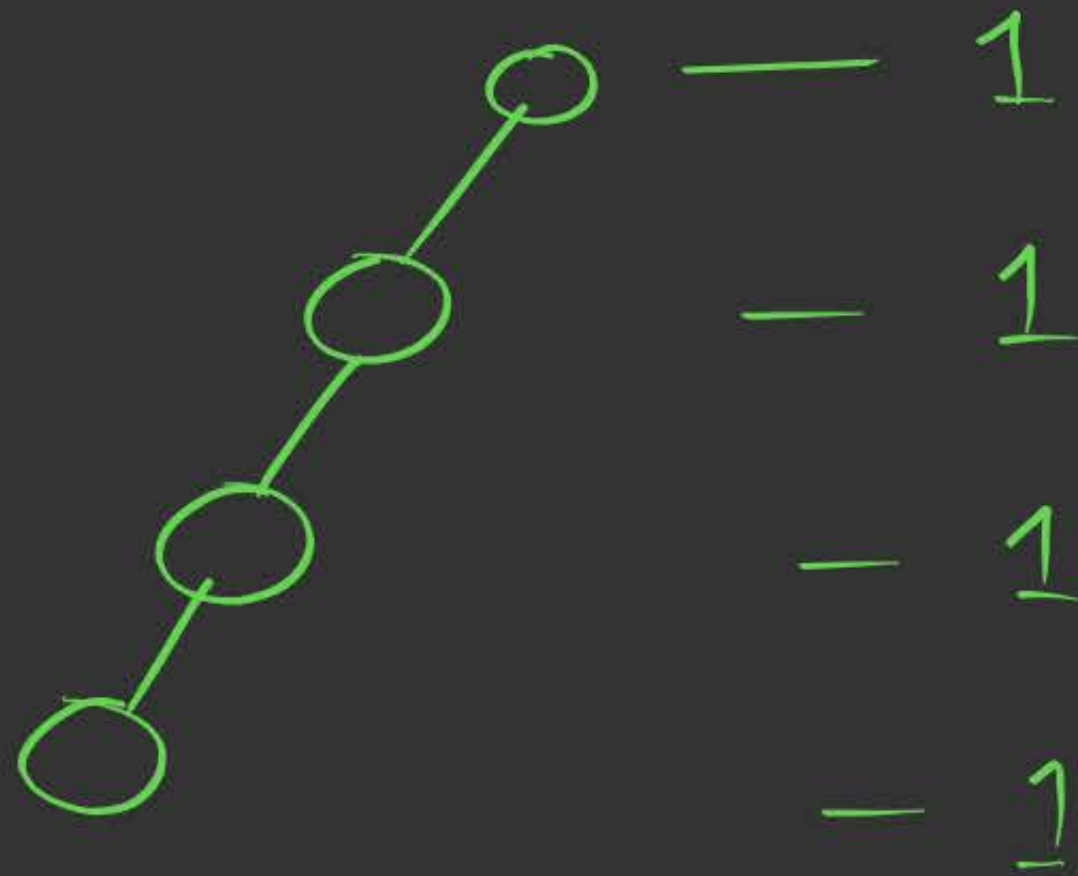
$$h^2 + h = 2n$$

$$h^2 \sim n$$

$$h = \sqrt{n}$$

$$\therefore h = O(\sqrt{n})$$

Max Height of a Binary Tree with n -elements is —



$$n = \sum_{i=1}^h 1 = h$$

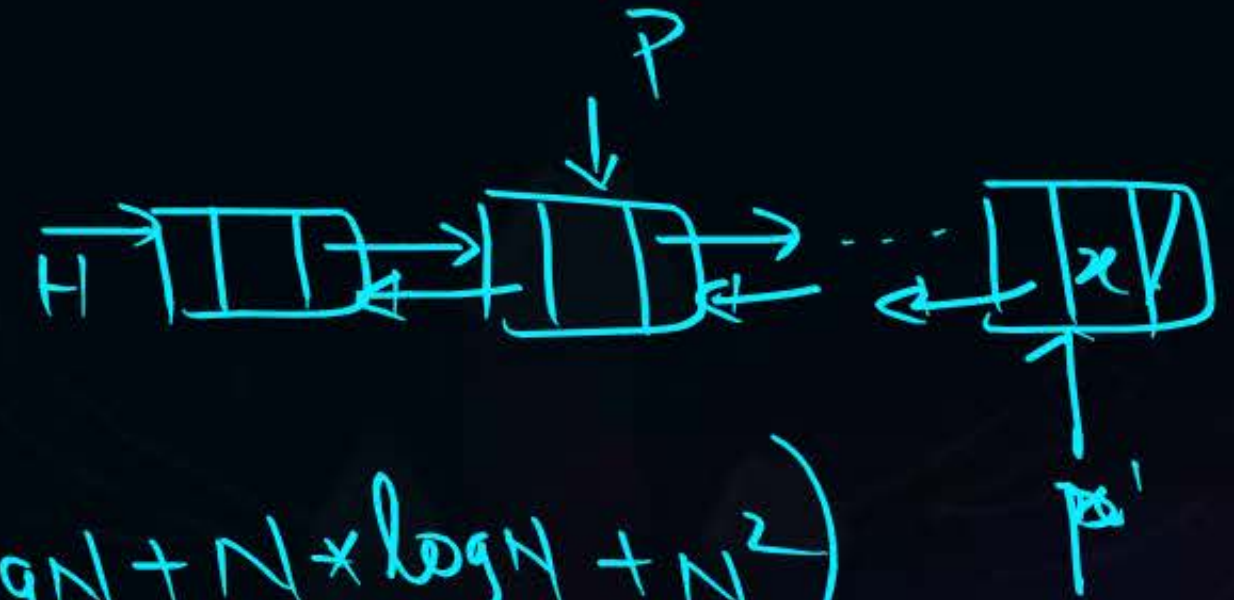
$$h = n \\ = O(n)$$

$$\log n \leq h \leq n$$

Q) N items are stored in a sorted doubly linked list. For a delete operation, a pointer is provided to the record to be deleted. For a decrease-key operation, a pointer is provided to the record on which the operation is to be performed. An algorithm performs the following operations on the list in this order: $\Theta(N)$ delete, $O(\log N)$ insert, $O(\log N)$ find, and $\Theta(N)$ decrease-key. What is the time complexity of all these operations put together?

- (a) $O(\log^2 N)$
- (c) $O(N^2)$

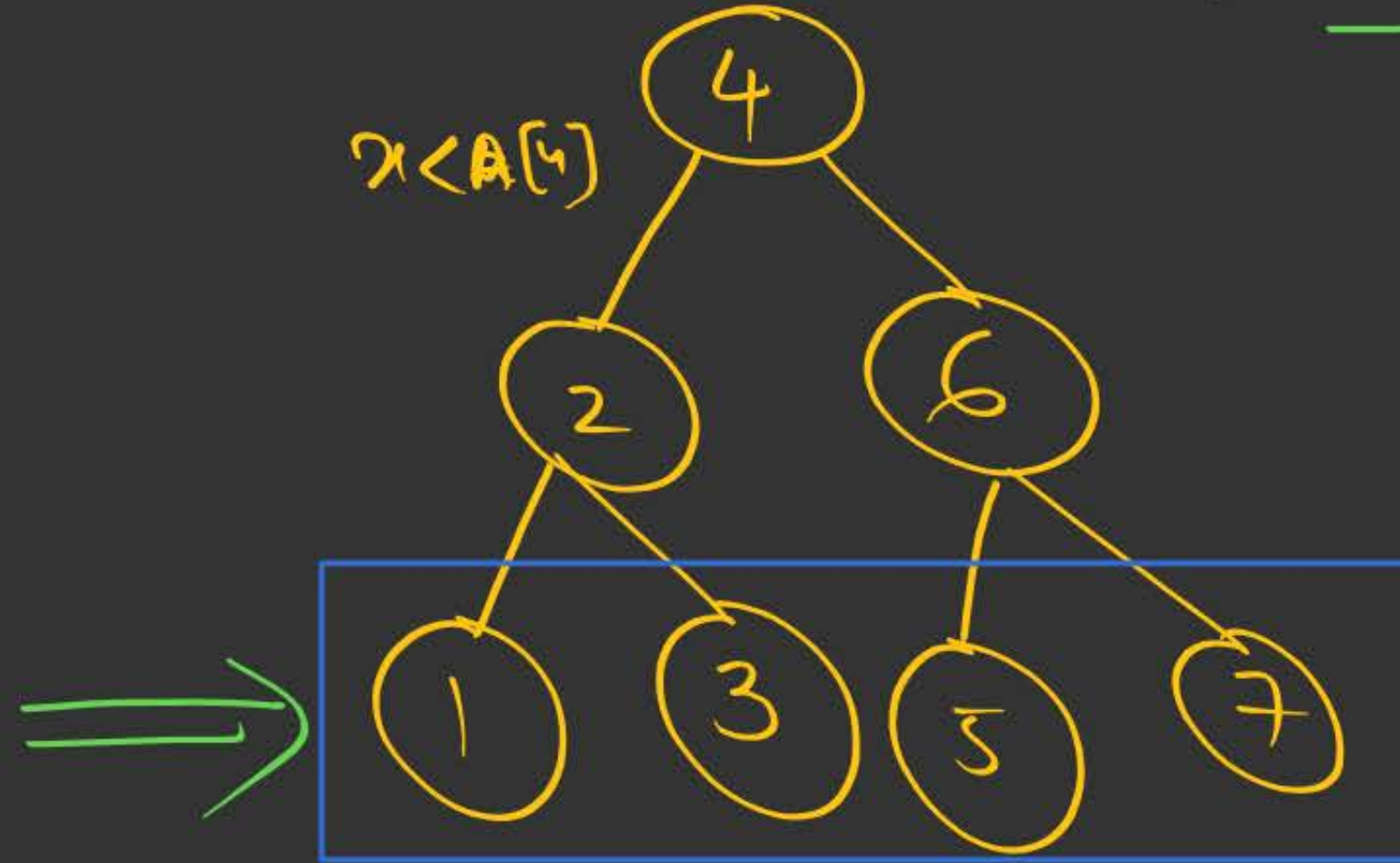
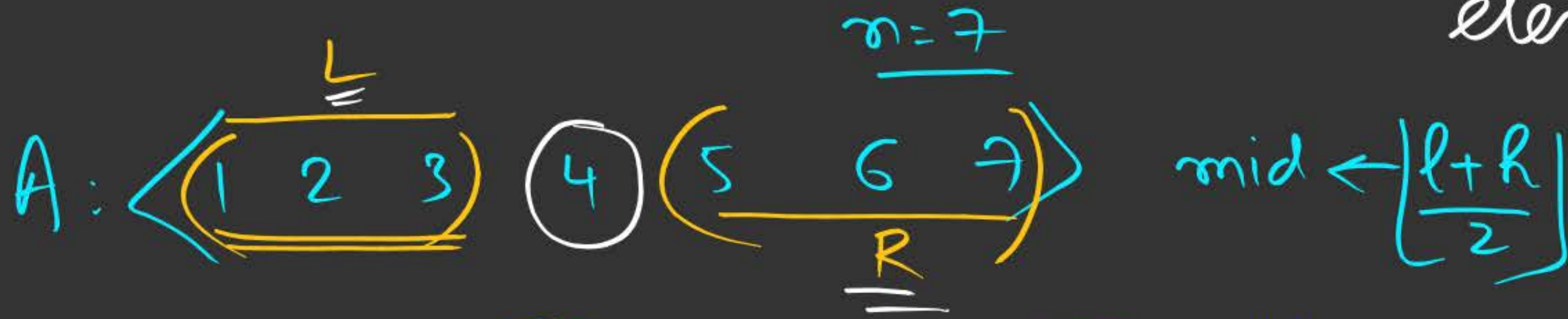
- (b) $O(N)$
- (d) $\Theta(N^2 \log N)$



$$\begin{aligned}
 &\text{Total time} \\
 &1) \text{ delete} - N - N * 1 \Rightarrow (N + N * \log N + N * \log N + N^2) \\
 &2) \text{ Insert} - \log N - N * \log N \\
 &3) \text{ find} - \log N - N * \log N \\
 &4) \text{ Dec-key} - N - N * N \\
 &\quad \quad \quad \Theta(n^2) \checkmark
 \end{aligned}$$

Divide & Conquer:

2) Binary Search: The Primary Requirement is that the list of n -elements must be in Sorted order,



Complete/Full B.T

Key = x

height = $h = O(\log n)$

$$I: \langle n, a_1, a_2, \dots, a_n, x \rangle \quad n > 1$$

$$L \quad x < a_k$$

$$I_1(k-1, a_1, \dots, a_{k-1}, x)$$



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

$$I_2(1, a_k, x)$$

$$R \quad x > a_k$$

$$I_3(n-k, a_{k+1}, \dots, a_n, x)$$



Q) Consider an array (sorted) with 'n' - elements, then if Binary Search is applied, then the DandC Recurrence arising is _____,

$$T(n)$$

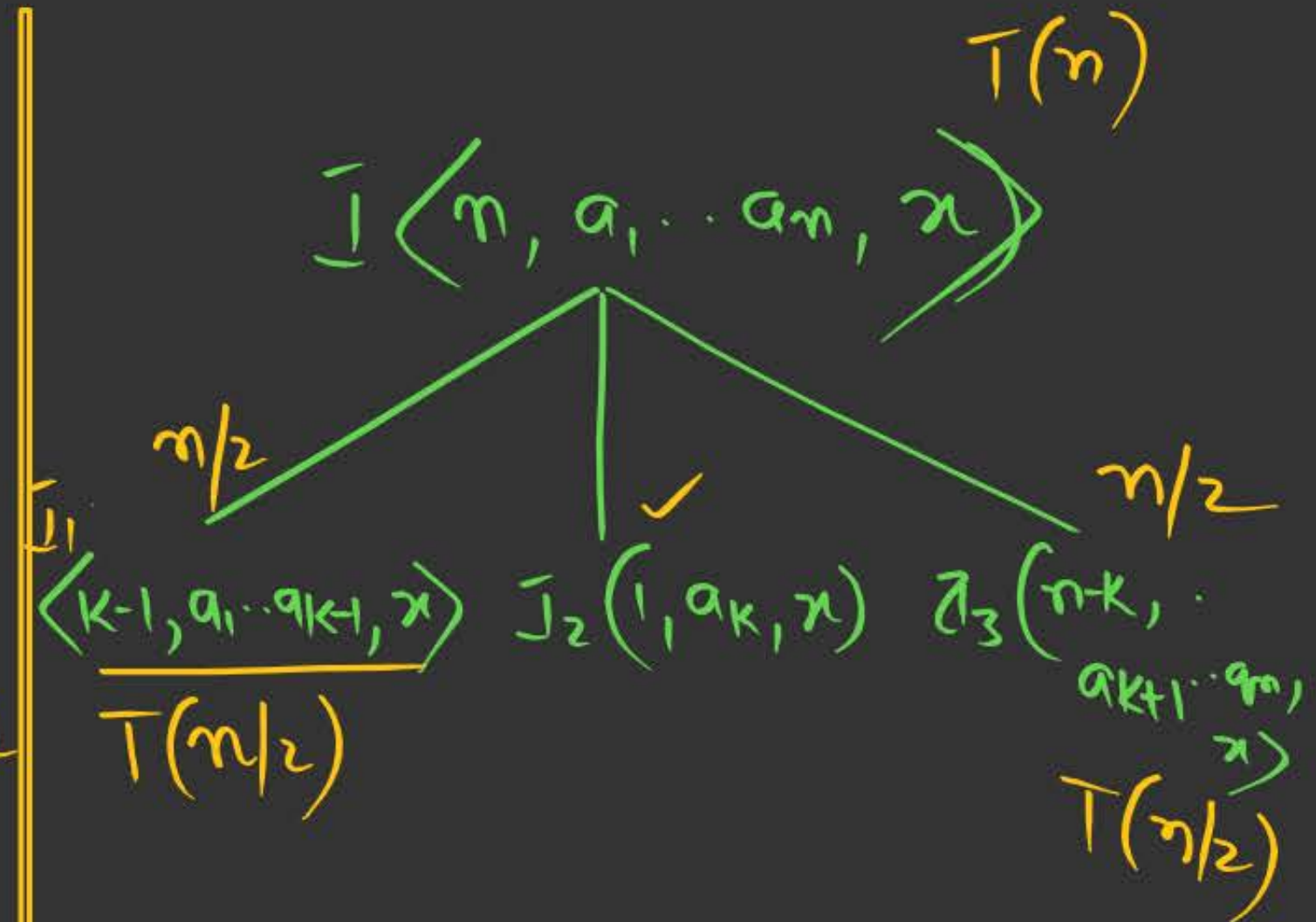
$$T(n) = C, \quad n=1$$

$$= a + T(n/2), \quad n > 1$$

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

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

In Binary Search there is no Combine/Conquer of 'n'



At every level we are solving 2 out of 3 Subproblems



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,  $n \geq 0$ , 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  $\leq$  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. }
```

Space Complexity
 $= O(1)$



Topic : Divide and Conquer

```
1  Algorithm BinSearch( $a, n, x$ )
2  // Given an array  $a[1 : n]$  of elements in nondecreasing
3  // order,  $n \geq 0$ , 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 ( $low \leq high$ ) do
8      {
9           $mid := \lfloor (low + high)/2 \rfloor;$ 
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 }
```

Algorithm 3.3 Iterative binary search



Topic : Divide and Conquer

Recursive Bin-Search;

1. Algorithm BinSrch(a, i, l, x)
2. // Given an array $a[\underline{i} : \underline{l}]$ of elements in nondecreasing
5. {
6. if ($l = i$) then // If Small(P)
7. {
8. if ($x = a[i]$) then return(i);
9. else return 0;
10. }



Topic : Divide and Conquer

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




Topic : Divide and Conquer

```
1  Algorithm BinSrch( $a, i, l, x$ )
2  // Given an array  $a[i : l]$  of elements in nondecreasing
3  // order,  $1 \leq i \leq l$ , 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     }
11     else
12     { // Reduce  $P$  into a smaller subproblem.
13          $mid := \lfloor (i + l) / 2 \rfloor$ ;
14         if ( $x = a[mid]$ ) then return  $mid$ ;
15         else if ( $x < a[mid]$ ) then
16             return BinSrch( $a, i, mid - 1, x$ );
17         else return BinSrch( $a, mid + 1, l, x$ );
18     }
19 }
```

Space
Complexity
 $= O(\log n)$

3) Merge Sort: Principle of Merging (Conquer)

→ Given Two Sorted lists $L_1(n_1)$ & $L_2(n_2)$, where $n_1 \leq n_2$
it is required to Merge them into a Single Sorted Array
elements, using 2-way Merging; $(n_1 + n_2)$

$n_1 = 4$
 $L_1: \langle 4; 5; 10; 12 \rangle$
↑
 i

$n_2 = 6$
 $L_2: \langle 3; 7; 11; 15; 18; 25 \rangle$
↑
 j

$L: \langle 3; 4; 5; 7; \dots; 15, 18, 25 \rangle$
New array $[n_1 + n_2]$

Q) Given Two Sorted lists $L_1(n_1)$ & $L_2(n_2)$ $n_1 \leq n_2$
 the Min. & Max. no. of Comparisons needed to Merge them
 to get a single Sorted list (n_1+n_2) is _____,

① Minimum: $|L_1| < \text{First } |L_2|$ $L_1: \langle 2, 3, 5 \rangle$ $L_2: \langle 8, 10, 12, 15, 18 \rangle$
 Min. Comp's : n_1 $[n_1 \leq n_2]$

② Maximum: $(n_1-1) |L_1| < \text{First } |L_2|$ & $|L_2| < \text{Last } |L_1|$
 $n_1=4$
 $L_1: \langle 2, 3, 5, 50 \rangle$ $L_2: \langle 8, 10, 12, 15, 18, 25, 40 \rangle$
 $n_2=8$

$\langle 2, 3, 5, 8, 10, 12, 15, \dots, 40 \rangle$ $:= \frac{(n_1-1) + n_2}{(n_1+n_2-1)}$

→ The no. of Comparisons required to Merge Two Sorted lists $L_1(n_1)$ & $L_2(n_2)$, $n_1 \leq n_2$ lies between

(n_1) & $(n_1 + n_2 - 1)$

Min

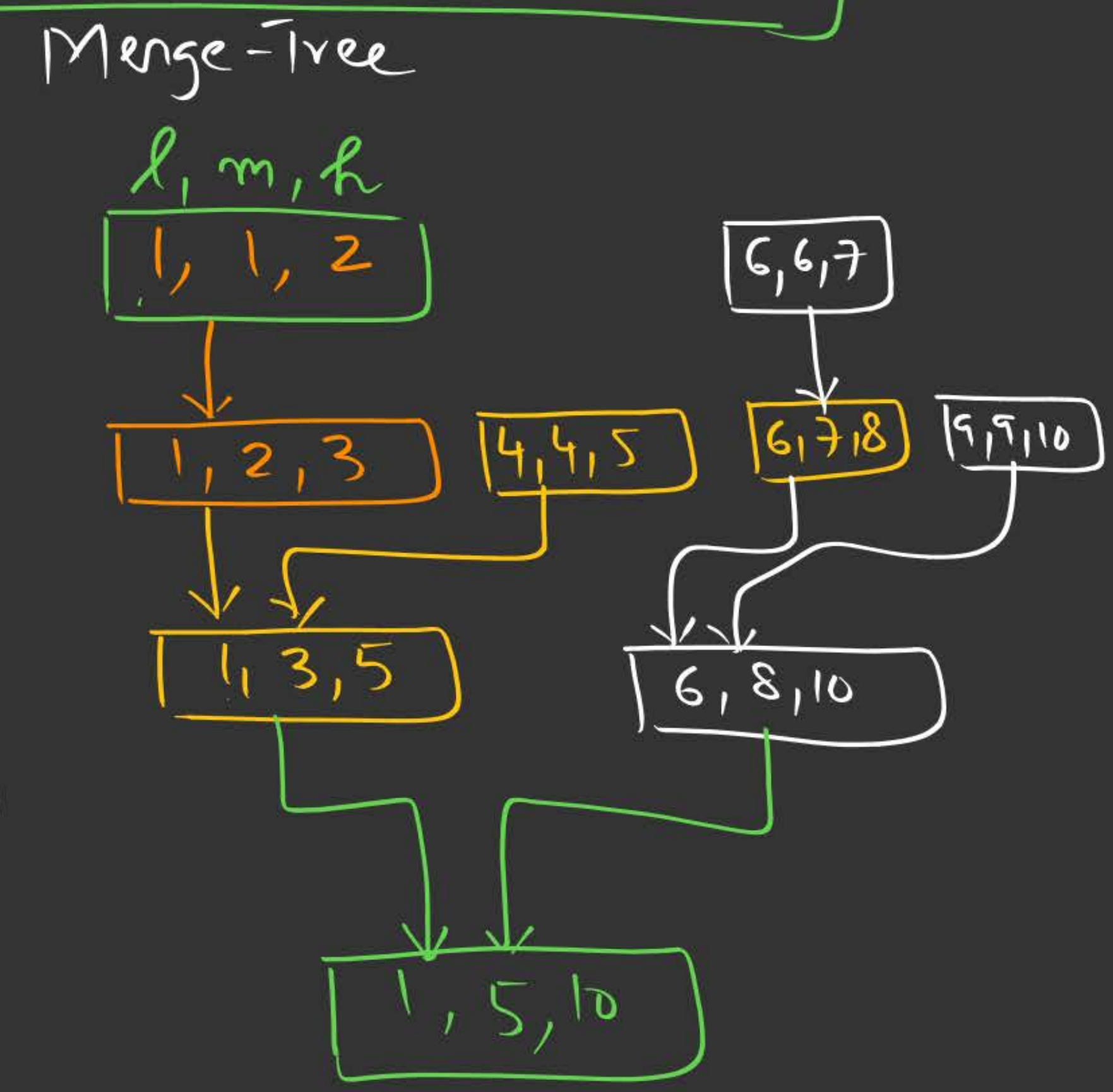
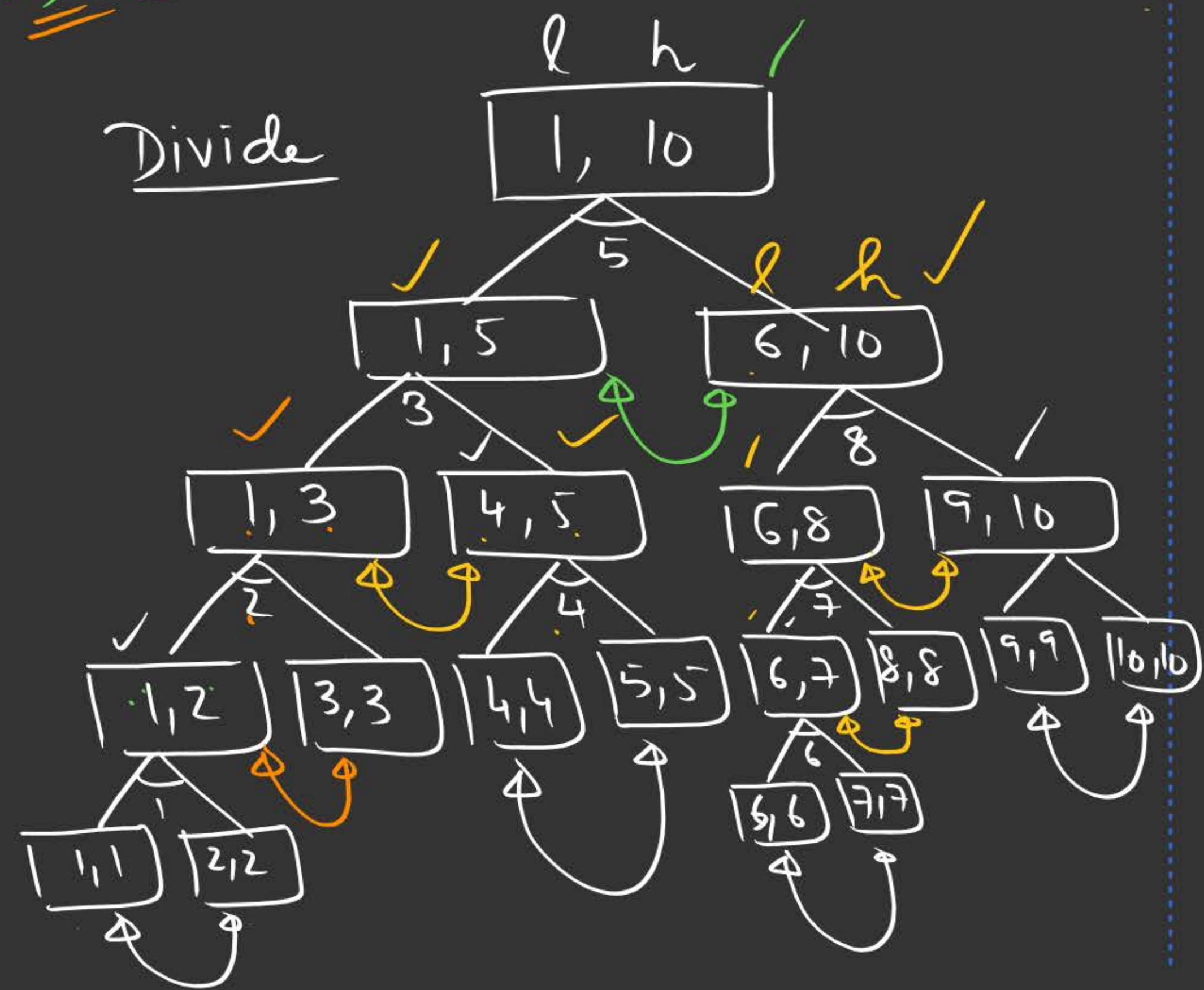
$O(n_1)$

Maximum

$O(n_1 + n_2)$

IP \nearrow A: $\left[\begin{array}{ccccc} 1 & 7 & 2 & 7 & 3 \\ 179 & 285 & 310 & 351 & 652 \end{array} \right] \left[\begin{array}{ccccc} 6 & 7 & 8 & 9 & 10 \\ 254 & 423 & 450 & 520 & 861 \end{array} \right]$

Addit Array \nearrow B: $\left[179 \quad 254 \quad 285 \quad 310 \quad 351 \quad 423 \quad 450 \quad 520 \quad 652 \quad 861 \right]$





Topic : Divide and Conquer

1. Algorithm MergeSort (^llow, ^hhigh)
2. // a[low : high] is a global array to be sorted.
3. // Small(P) is true if there is only one element
4. //to sort. In this case the list is already sorted.
5. {
6. if (low < high) then // If there are more than one element
7. {
8. // Divide P into subproblems.
9. // Find where to split the set.
10. mid: = $\lfloor (low + high) / 2 \rfloor$;
11. // Solve the subproblems.



Topic : Divide and Conquer

12. MergeSort(low, mid) ✓

13. MergeSort(mid + 1, high); ✓

14. // Combine the solution.

15. Merge(low, mid, high);

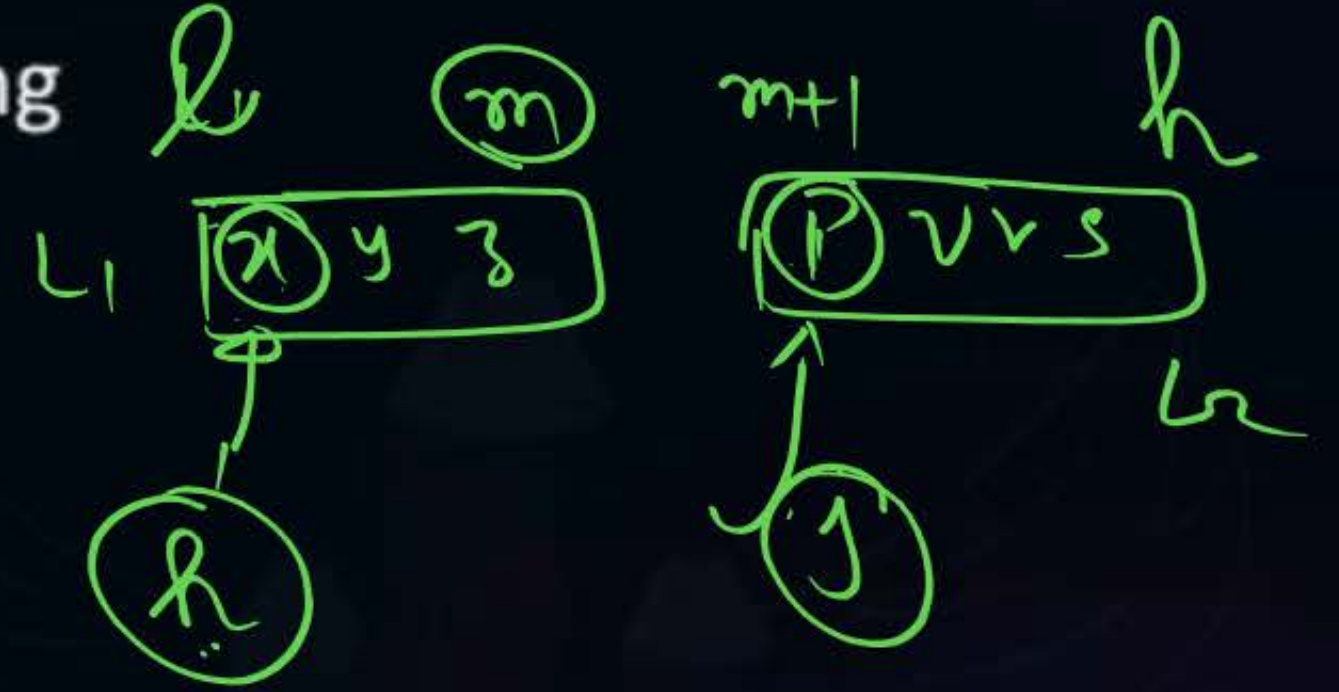
16. }

17. }



Topic : Divide and Conquer

1. Algorithm Merge (low, mid, high)
2. // a[low : high] is a global array containing two sorted
3. // subsets in a [low : mid] and in a[mid + 1 : high]. The goal
4. // is to merge these two sets into a single set residing
5. // in a[low : high]. b[] is an auxiliary global array.
6. {
7. h := low; i := low; j := mid + 1;
8. while ((h ≤ mid) and (j < high)) do
9. * {
10. if (a[h] ≤ a[j]) then





Topic : Divide and Conquer

```
11.  {  
12.      b[i] := a[h]; h := h + 1;  
13.  }  
14.  else  
15.      {  
16.          b[i] := a[j]; j := j + 1;  
17.      }  
18.      i := i + 1;  
19.  } ✱  
20.  if (h > mid) then
```

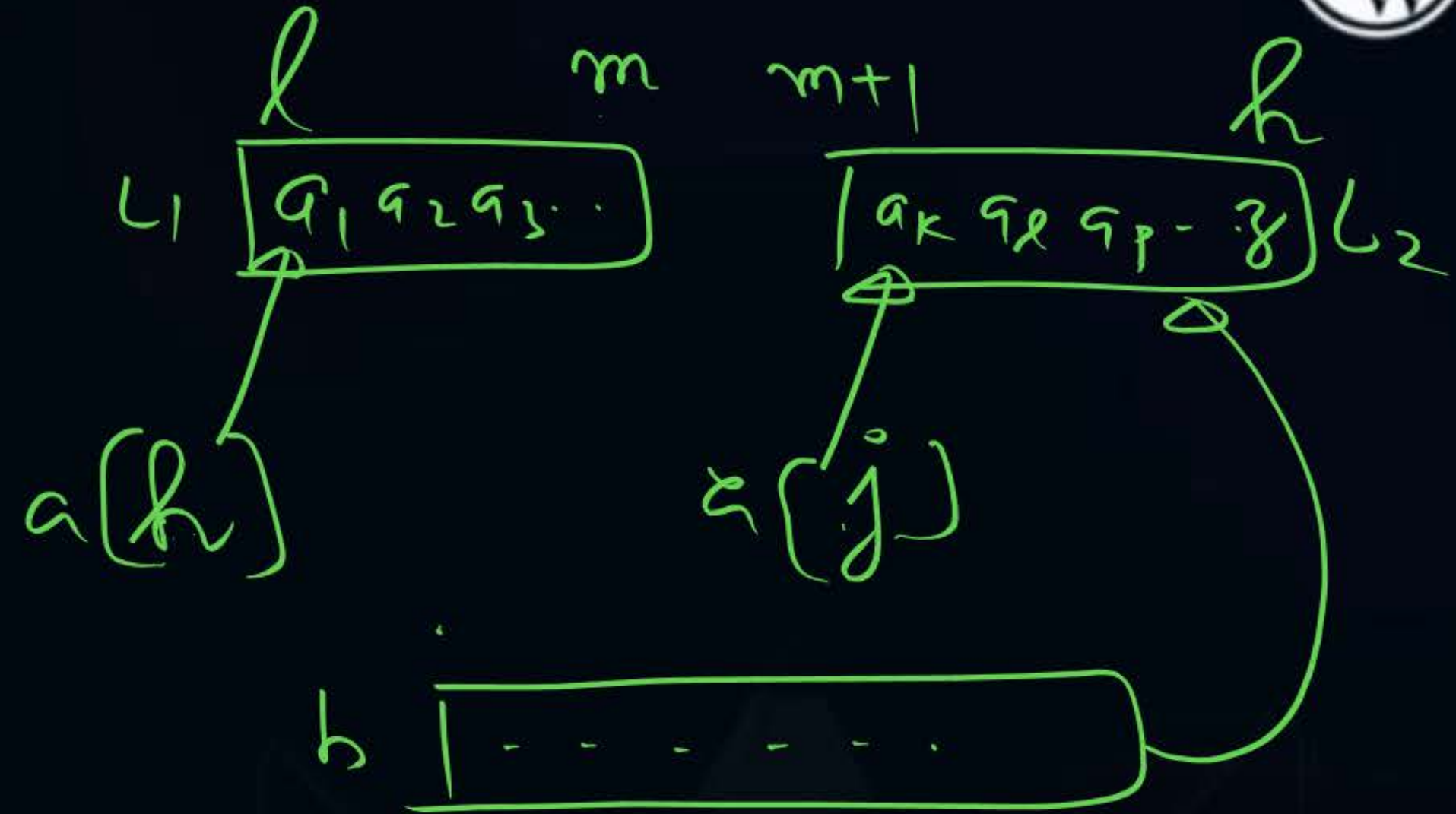



Topic : Divide and Conquer

```
21.   for k := j to high do
22.   {
23.       b[i] := a[k]; i := i+1;
24.   }
25.   else
26.       for k := h to mid do
27.       {
28.           b[i] := a[k]; i := i+1;
29.       }
30.   for k := low to high do a[k] := b[k];
31. }
```

Second list

First list



Copying elements from array b to a



Topic : Divide and Conquer

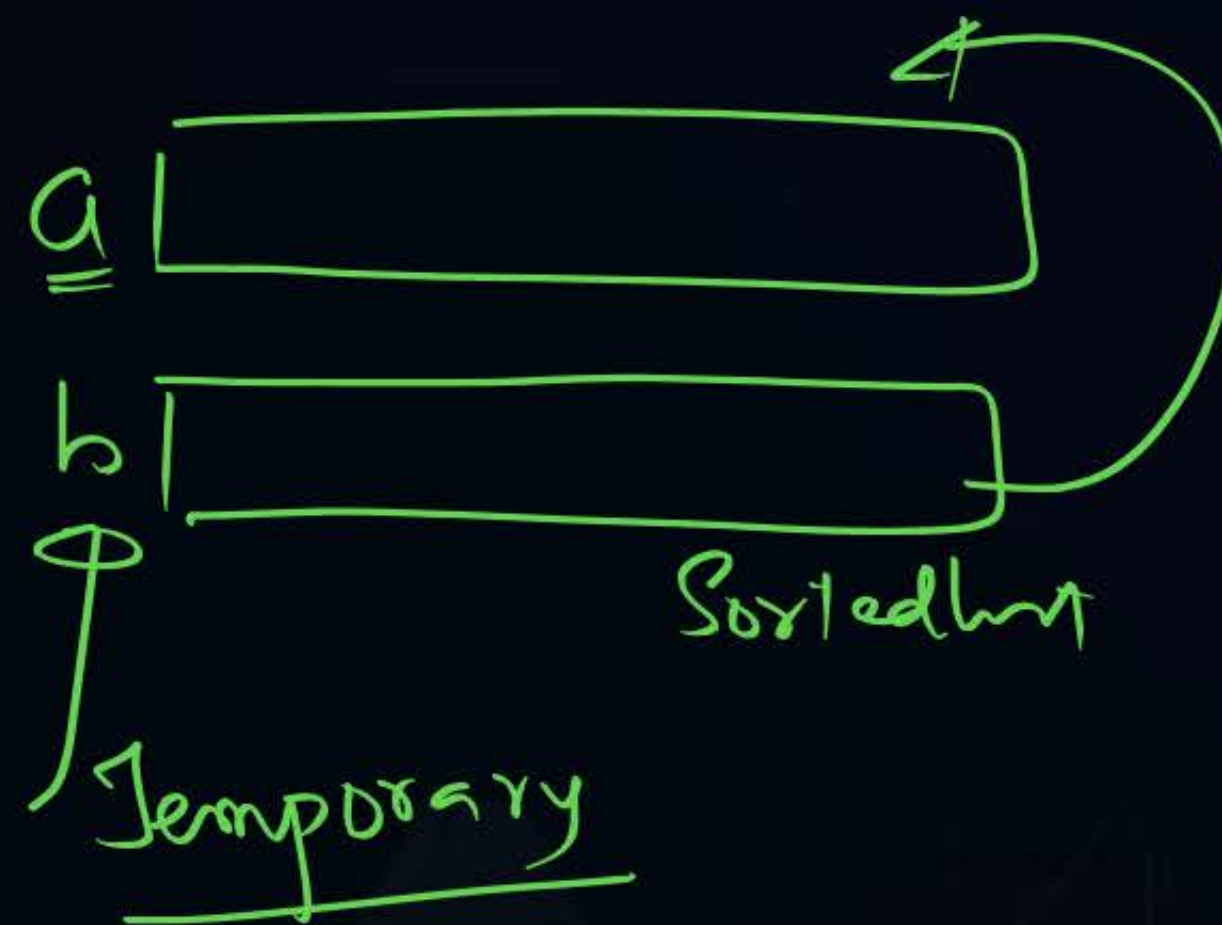
```
1  Algorithm MergeSort(low, high)
2  // a[low : high] is a global array to be sorted.
3  // Small(P) is true if there is only one element
4  // to sort. In this case the list is already sorted.
5  {
6      if (low < high) then // If there are more than one element
7      {
8          // Divide P into subproblems.
9          // Find where to split the set.
10         mid :=  $\lfloor (low + high) / 2 \rfloor$ ;
11         // Solve the subproblems.
12         MergeSort(low, mid);
13         MergeSort(mid + 1, high);
14         // Combine the solutions.
15         Merge(low, mid, high);
16     }
17 }
```




Topic : Divide and Conquer



```
1  Algorithm Merge(low, mid, high)
2  // a[low : high] is a global array containing two sorted
3  // subsets in a[low : mid] and in a[mid + 1 : high]. The goal
4  // is to merge these two sets into a single set residing
5  // in a[low : high]. b[ ] is an auxiliary global array.
6  {
7      h := low; i := low; j := mid + 1;
8      while ((h ≤ mid) and (j ≤ high)) do
9      {
10         if (a[h] ≤ a[j]) then
11         {
12             b[i] := a[h]; h := h + 1;
13         }
14         else
15         {
16             b[i] := a[j]; j := j + 1;
17         }
18         i := i + 1;
19     }
20     if (h > mid) then
21         for k := j to high do
22         {
23             b[i] := a[k]; i := i + 1;
24         }
25     else
26         for k := h to mid do
27         {
28             b[i] := a[k]; i := i + 1;
29         }
30     for k := low to high do a[k] := b[k];
31 }
```



Performance of MergeSort:

1) Time Complexity:

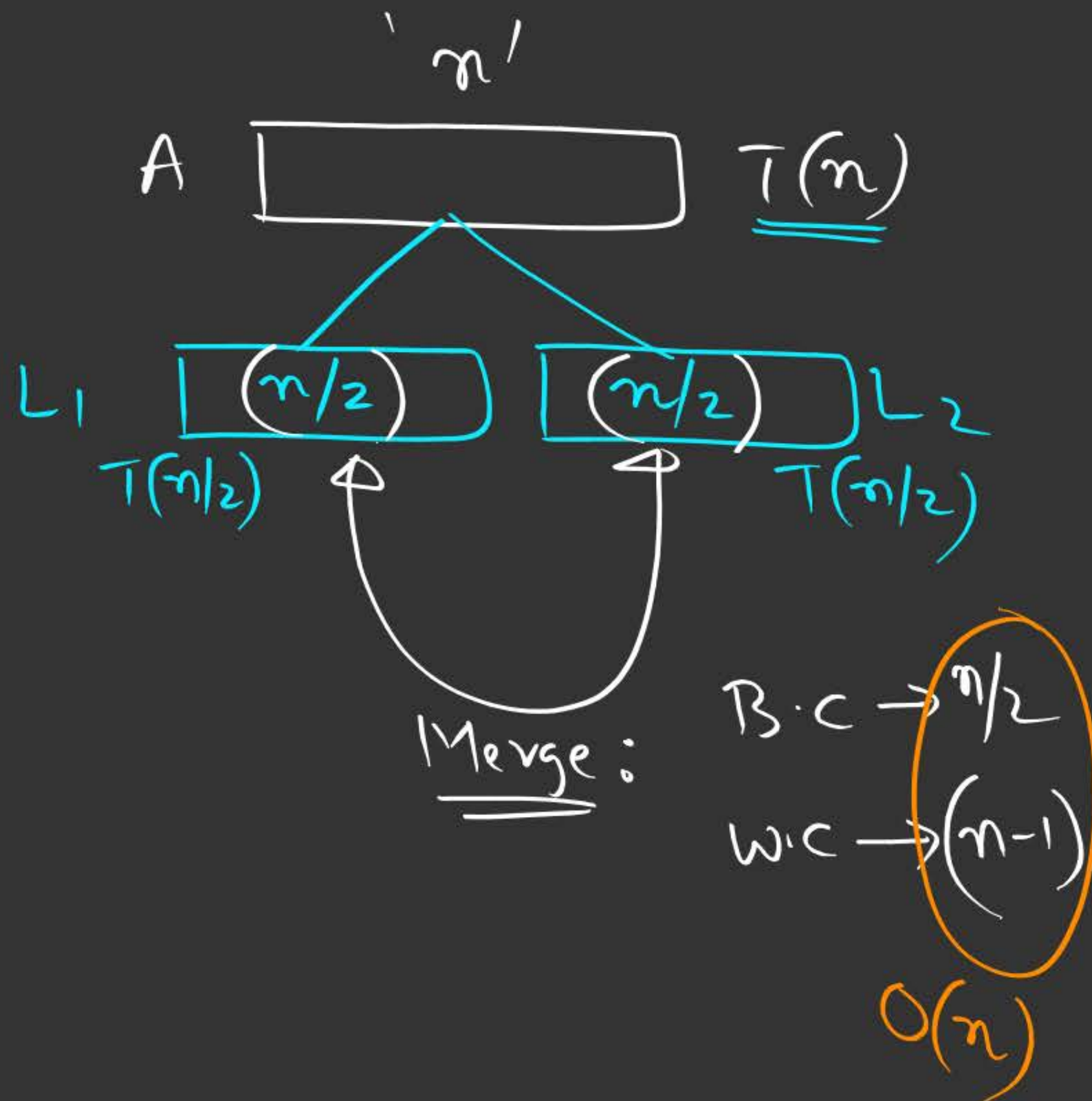
Let $T(n)$ repr. Time Complexity
of DandC-MS(n);

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

$$= 2.T(n/2) + bn, \quad n > 1 \quad \underline{b > 0}$$

$$\rightarrow O(n \cdot \log n)$$
$$\sim \Omega(n \log n)$$

$$\therefore T(n) = \Theta(n \cdot \log n)$$



THANK - YOU