CS & IT ENGINEERING

Algorithms

Divide & Conquer

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 ← 1 to (n-1)
    for j ← (i+1) to n
        for k ← 1 to j
            c = c+1; ⊛
```

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

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

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

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

```
for i ← 1 to n
    c = c+1;
```

$$\Longrightarrow \sum_{i=1}^{n} O(1)$$

$i=1$ $\quad i=n-1$

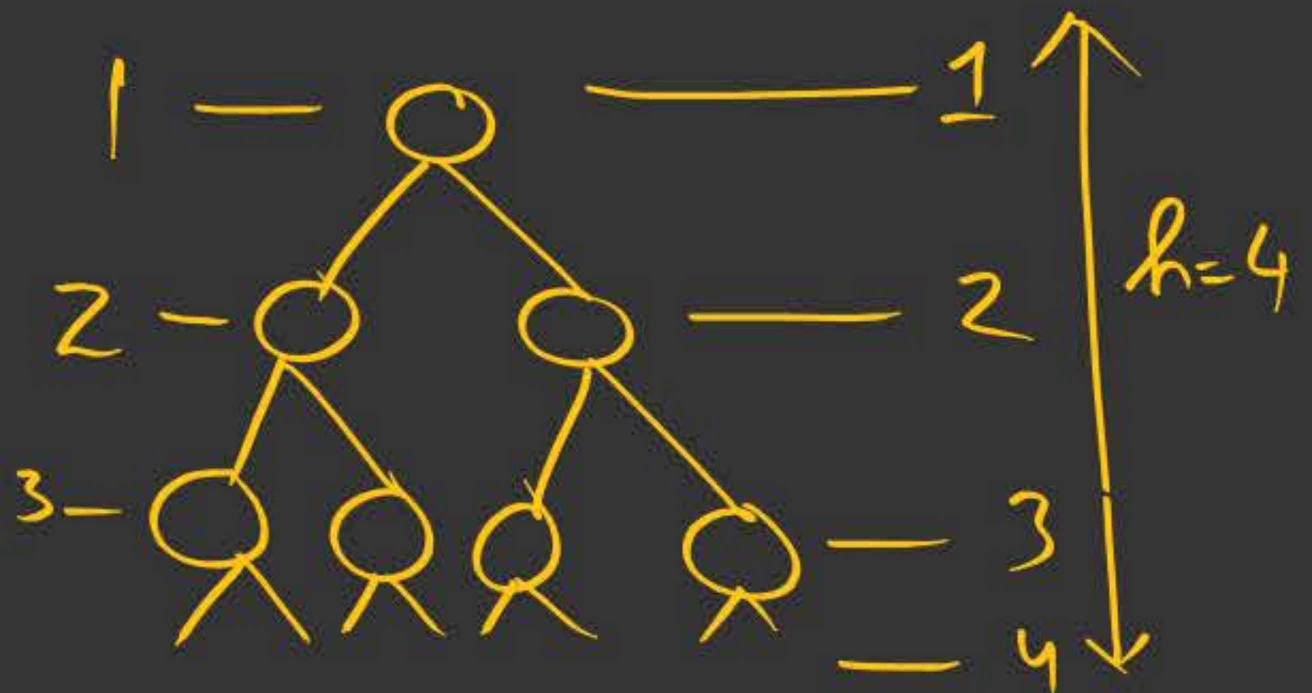
$$\left(\frac{n(n+1)}{2}-1\right)+\frac{n(n+1)}{2}-(1+2)+\cdots+\frac{n(n+1)}{2}-(1+2+\cdots+n-1)$$

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

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

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

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' $(n) = \sum_{i=1}^{h} 2^{i-1}$

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

$S_n = \frac{a(r^n - 1)}{r-1}$

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

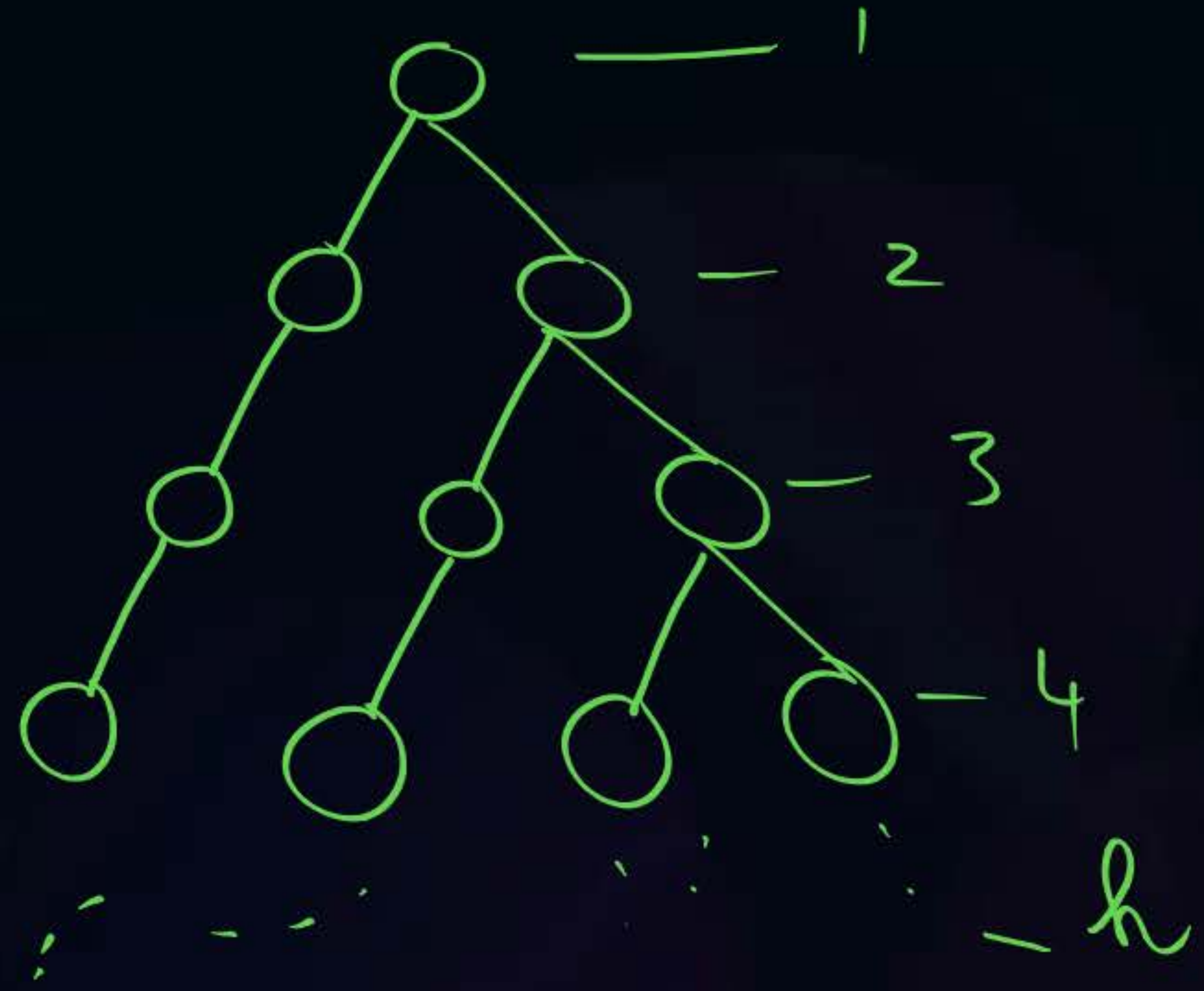
$=$

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

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

$$\boxed{h = \log_2 n+1} = O(\log_2 n)$$

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 ______.



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

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

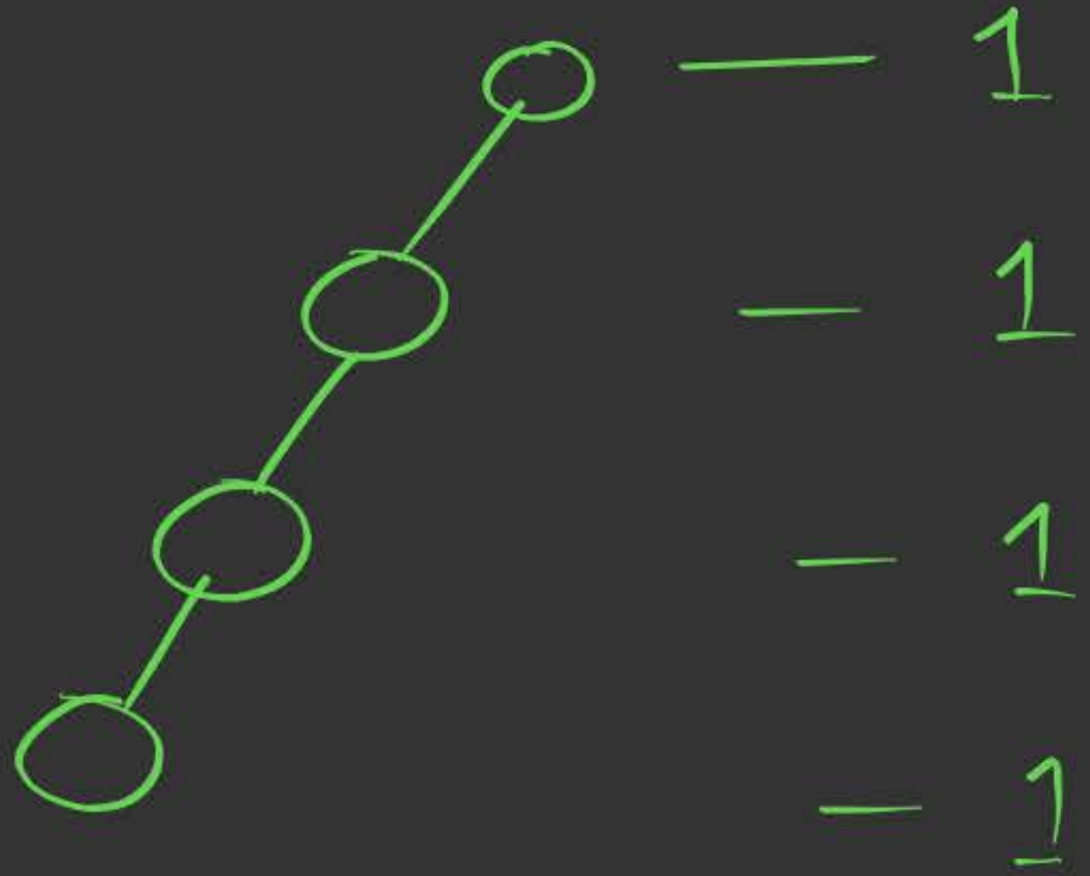
$$h^2 \sim n$$

$$h = \sqrt{n}$$

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

Slide 4

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



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

$$h = n$$

$$= O(n)$$

$$\boxed{\log n \le h \le n}$$

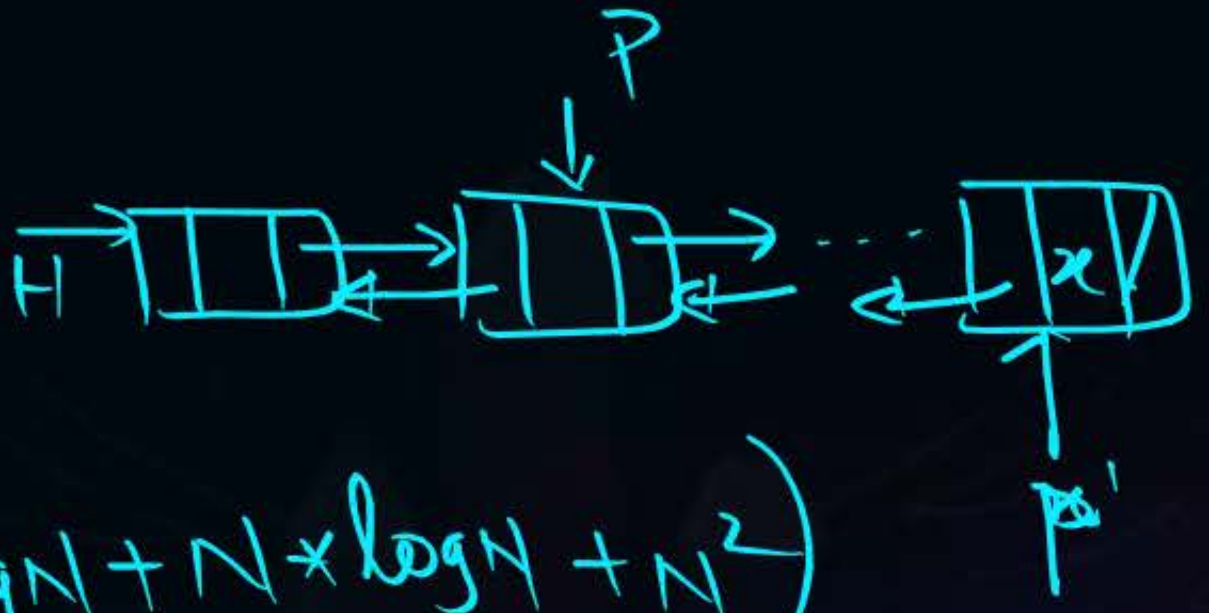
Q) N items are stored in a sorted doubly linked list. For a delete operation, a pointer is provided to the record to be deleted. For a decrease-key operation, a pointer is provided to the record on which the operation is to be performed. An algorithm performs the following operations on the list in this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

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

(b) $O(N)$

(c) $O(N^2)$

(d) $\Theta(N^2 \log N)$



Total time

1) delete - N - N*1 $\Rightarrow \left(N + N*\log N + N*\log N + N^2\right)$

2) Insert - logN - N*logN

3) find - logN - N*logN

4) Dec-key - N - N*N

$\Theta(n^2)$ ✓

Divide & Conquer :

2) Binary Search :

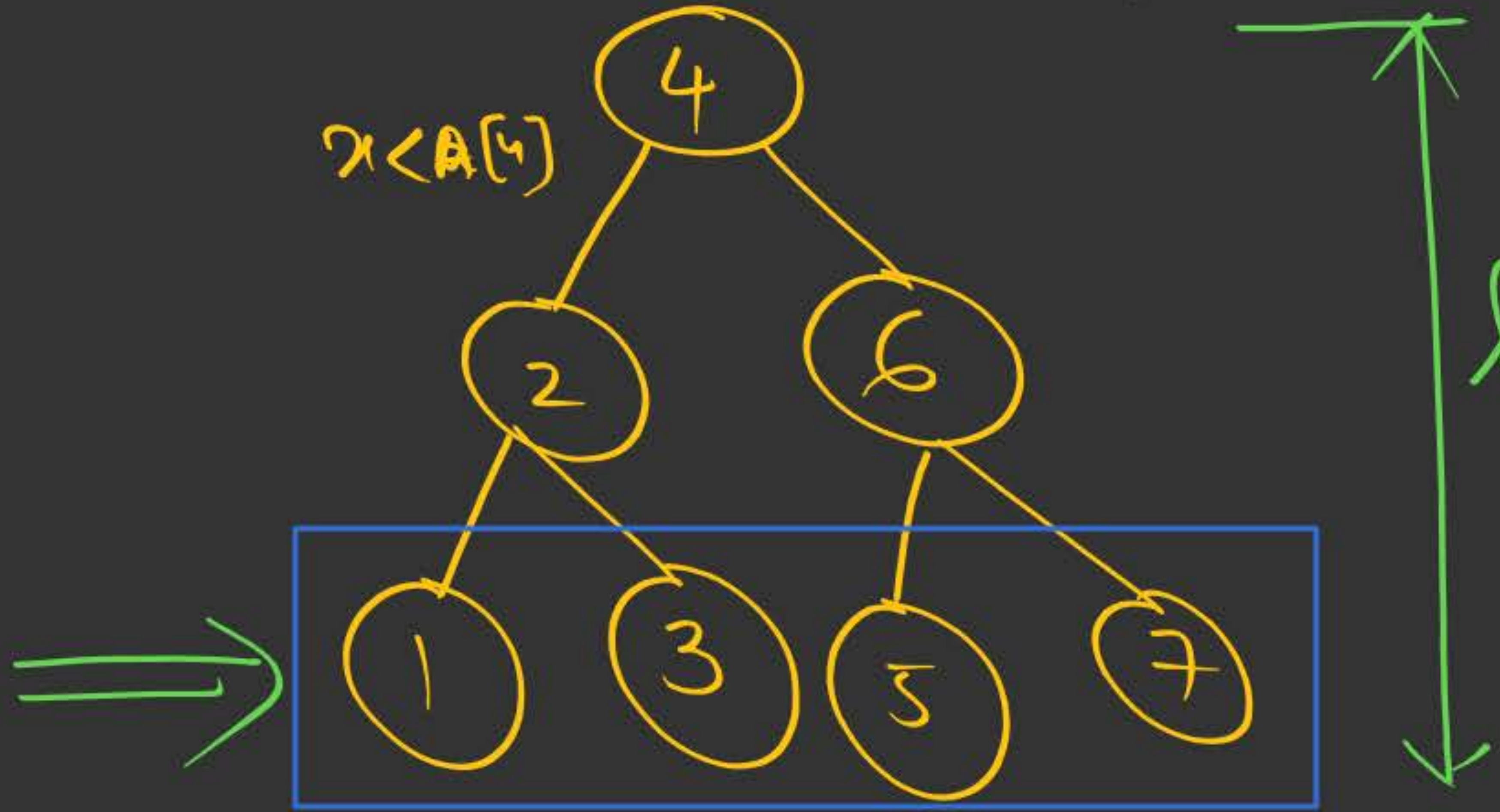
The Primary Requirement is that the list of n-elements must be in Sorted order;

$n = 7$

$$A : \langle \overbrace{1\quad 2\quad 3}^{L}\quad \bigcirc\!\!\!\!4\quad \underbrace{5\quad 6\quad 7}_{R} \rangle$$

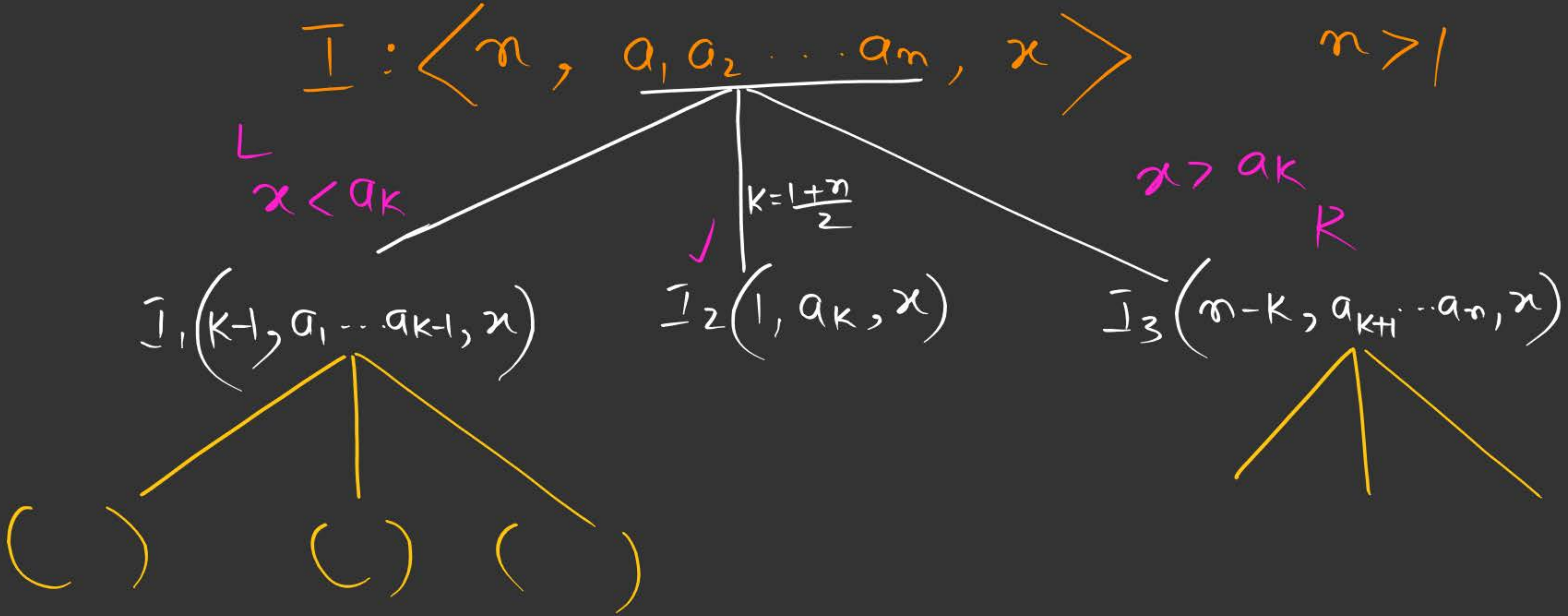
$$mid \leftarrow \left\lfloor \frac{l+h}{2} \right\rfloor$$

Key = x



height = h = O(log n)

Complete/Full B.T


$I : \langle n, a_1 a_2 \cdots a_n, x \rangle$
$n > 1$
L
$x < a_k$
$k = \frac{1+n}{2}$
✓
$x > a_k$
R
$I_1 (k-1, a_1 \cdots a_{k-1}, x)$
$I_2 (1, a_k, x)$
$I_3 (n-k, a_{k+1} \cdots a_n, x)$
( )
( )
( )

Q) Consider an array(sorted) with 'n'-elements, then if Binary Search is applied, then the D and C Recurrence arising is ______ ;

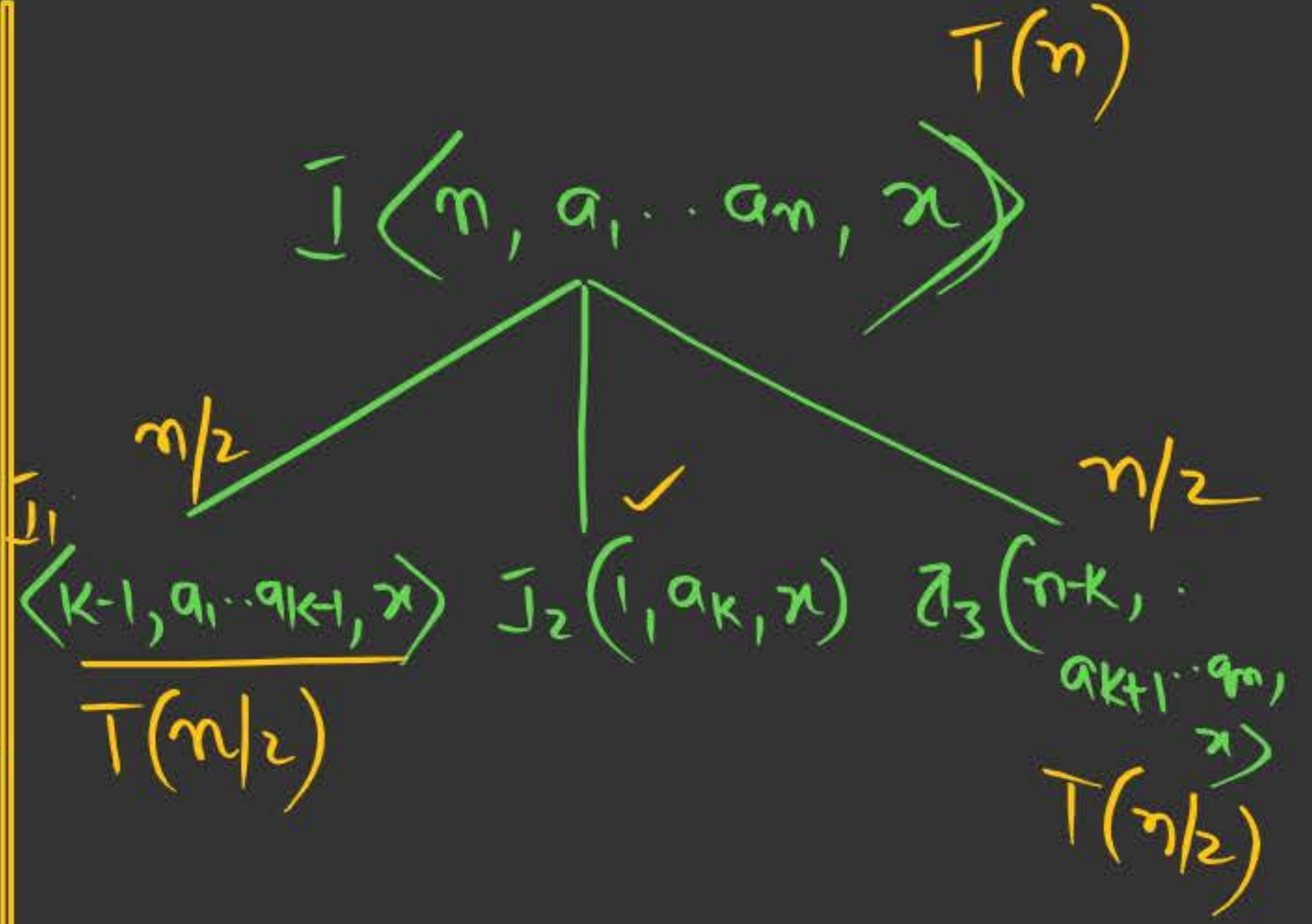
$T(n)$

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

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

$O(\log n)$ ✓

In Binary Search there is no Combine/Conquer of n



$T(n)$
$I\langle n, a_1 \cdots a_n, x\rangle$
$n/2$
✓
$n/2$
$I_1\langle k-1, a_1 \cdots a_{k-1}, x\rangle$
$T(n/2)$
$I_2(1, a_k, x)$
$I_3(n-k, a_{k+1} \cdots a_n, x\rangle$
$T(n/2)$

At every level we are Solving 2 out of 3 Subproblems

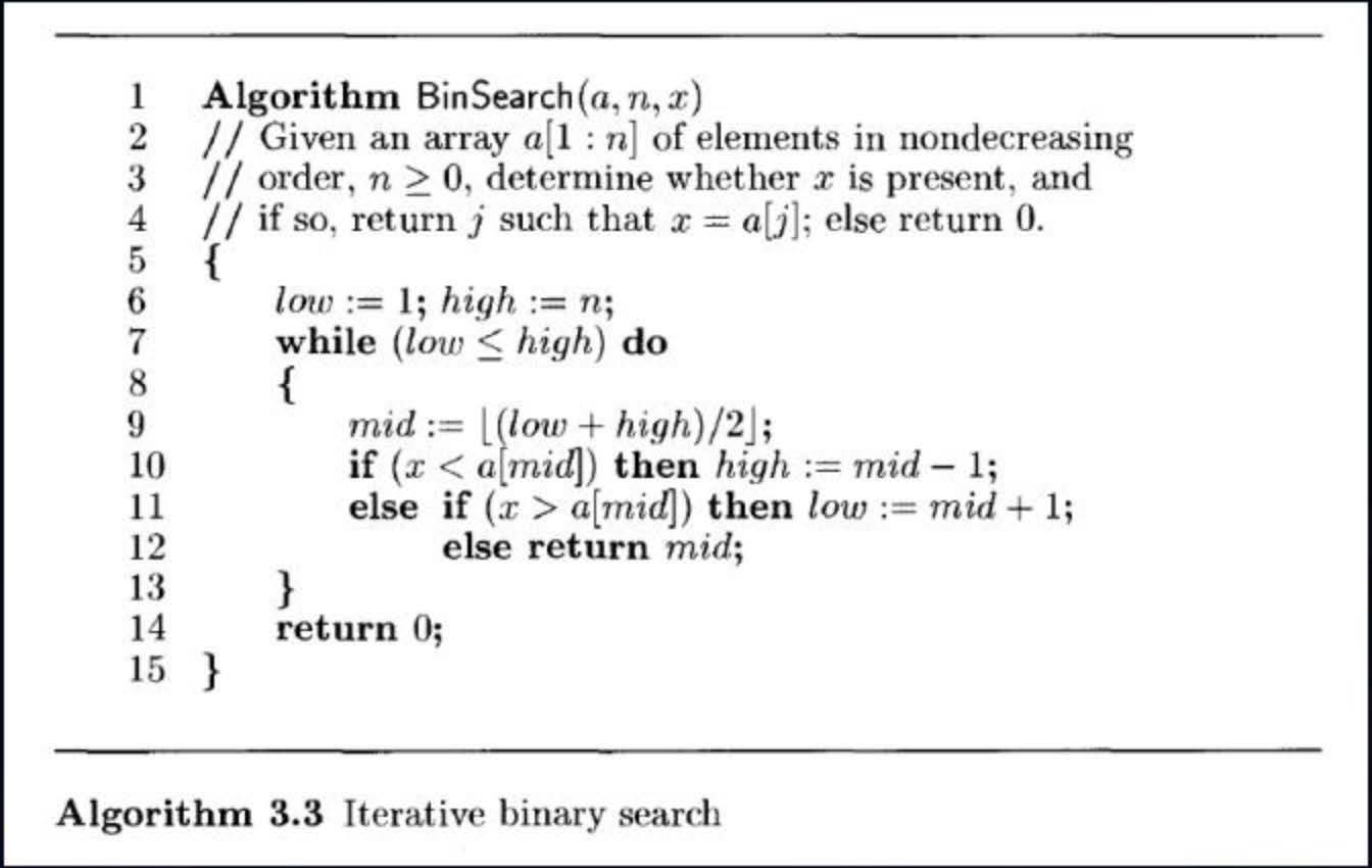
Topic : Divide and Conquer

```
Algorithm BinSearch(a,n,x)  < ITERATIVE BINARY SEARCH >
// Given an array a[1: n] of elements in nondecreasing
//order, n ≥ 0, determine whether x is present, and
// if so, return j such  that x  = a [j] ; else return 0.
{
      low : = 1; high : = n;
      While (low ≤ high) do
      {
           mid: = [(low + high)/2] ;
           If (x < a[mid]) then high: = mid –1;
           else if (x > a[mid]) then low : = mid + 1;
                      Else return mid;
      }
      Return 0;
}
```

Space Complexity

$= O(1)$

Topic : Divide and Conquer



```
Algorithm BinSearch($a, n, x$)
// Given an array $a[1:n]$ of elements in nondecreasing
// order, $n \geq 0$, determine whether $x$ is present, and
// if so, return $j$ such that $x = a[j]$; else return 0.
{
    $low := 1$; $high := n$;
    while ($low \leq high$) do
    {
        $mid := \lfloor (low + high)/2 \rfloor$;
        if ($x < a[mid]$) then $high := mid - 1$;
        else if ($x > a[mid]$) then $low := mid + 1$;
            else return $mid$;
    }
    return 0;
}
```

Algorithm 3.3 Iterative binary search

Topic : Divide and Conquer

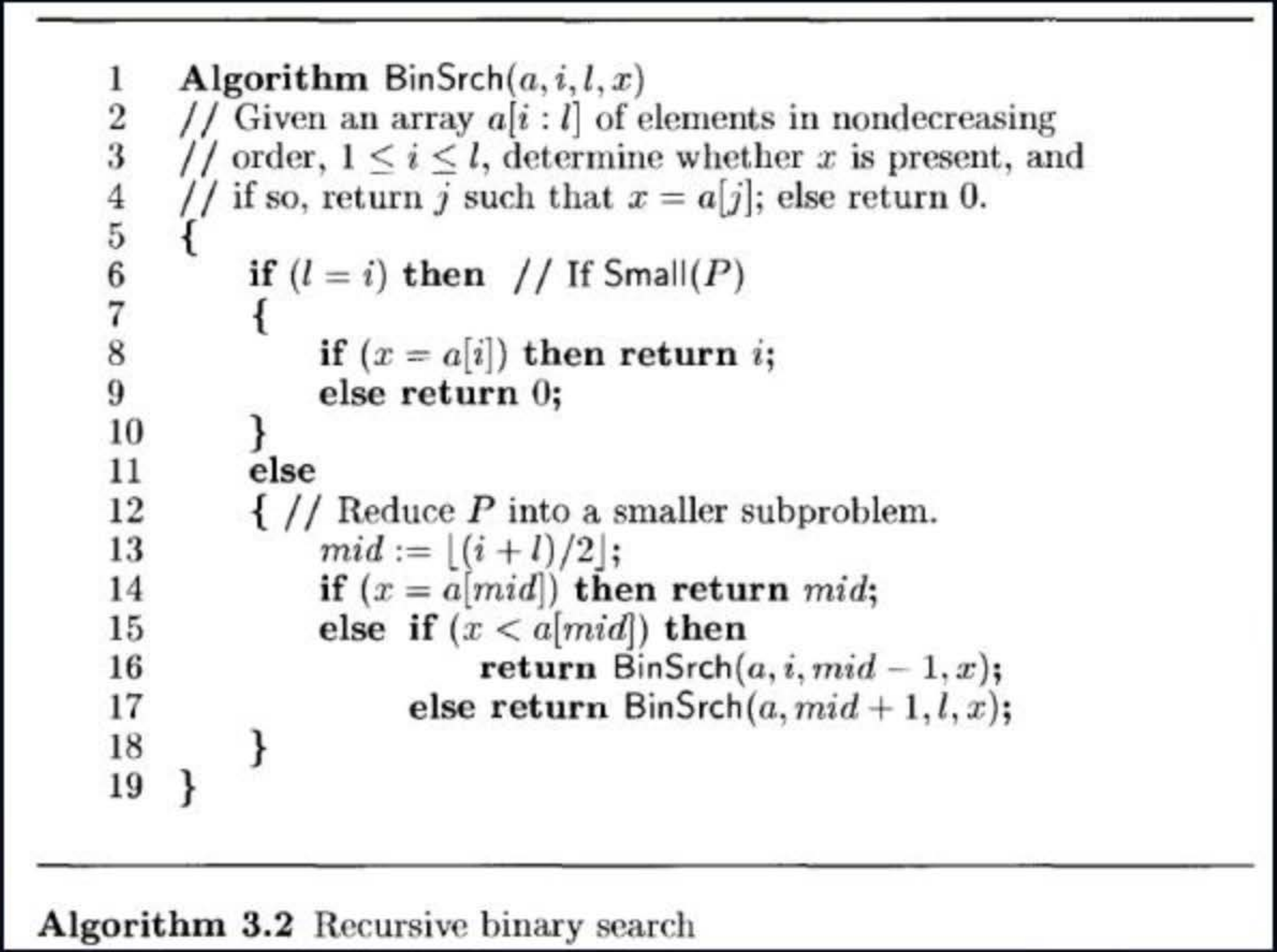
Recursive Bin-Search;

```
Algorithm BinSrch(a, i , l, x)
// Given an array a[i : l] of elements in nondecreasing
{
    if (l = i) then // If Small(P)
    {
        if (x = a[i]) then return(i;)
        else return 0;
    }
```

Topic : Divide and Conquer

```
    else    Problem is large
    { // Reduce P into a smaller subproblem.
        mid := ⌊(i + l)/2⌋ ;
        if (x = a[mid] then return mid;
        else if (x < a[mid]) then
            return BinSrch(a, i, mid — 1,x);
        else return BinSrch(a,mid + 1, l, x);
    }
}
```

Topic : Divide and Conquer



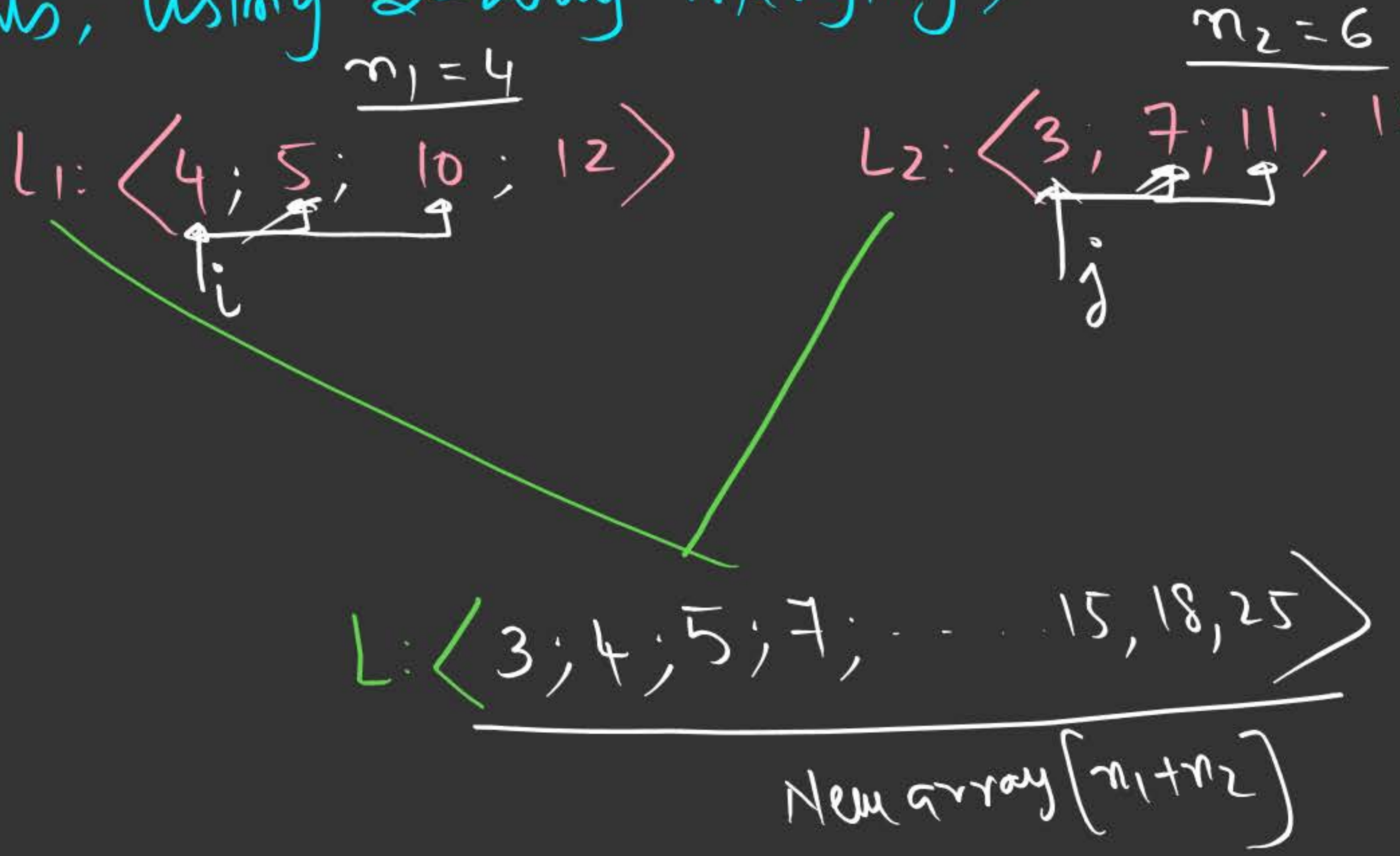
```
Algorithm BinSrch($a, i, l, x$)
// Given an array $a[i:l]$ of elements in nondecreasing
// order, $1 \leq i \leq l$, determine whether $x$ is present, and
// if so, return $j$ such that $x = a[j]$; else return 0.
{
    if ($l = i$) then  // If Small($P$)
    {
        if ($x = a[i]$) then return $i$;
        else return 0;
    }
    else
    { // Reduce $P$ into a smaller subproblem.
        $mid := \lfloor (i + l)/2 \rfloor$;
        if ($x = a[mid]$) then return $mid$;
        else  if ($x < a[mid]$) then
                  return BinSrch($a, i, mid - 1, x$);
              else return BinSrch($a, mid + 1, l, x$);
    }
}
```

Algorithm 3.2 Recursive binary search

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 having (n_1+n_2) elements, using 2-way Merging ;



$n_1 = 4$

$L_1 : \langle 4 ; 5 ; 10 ; 12 \rangle$

$n_2 = 6$

$L_2 : \langle 3 , 7 , 11 ; 15 ; 18 ; 25 \rangle$

$L : \langle 3 ; 4 ; 5 ; 7 ; \ldots 15 , 18 , 25 \rangle$

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 : $|L_1| <$ 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) $L_1 <$ First $|L_2|$ && $|L_2| <$ Last $|L_1|$

$n_1 = 4$

$L_1 : \langle 2, 3, 5, 50 \rangle$ $L_2 : \langle 8, 10, 12, 15, 18, 25, 40 \rangle$

$n_2 = 8$

$\langle 2, 3, 5, 8, 10, 12, 15, \ldots 40 \rangle$

: $(n_1 - 1) + n_2$

$(n_1 + n_2 - 1)$

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

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

Min

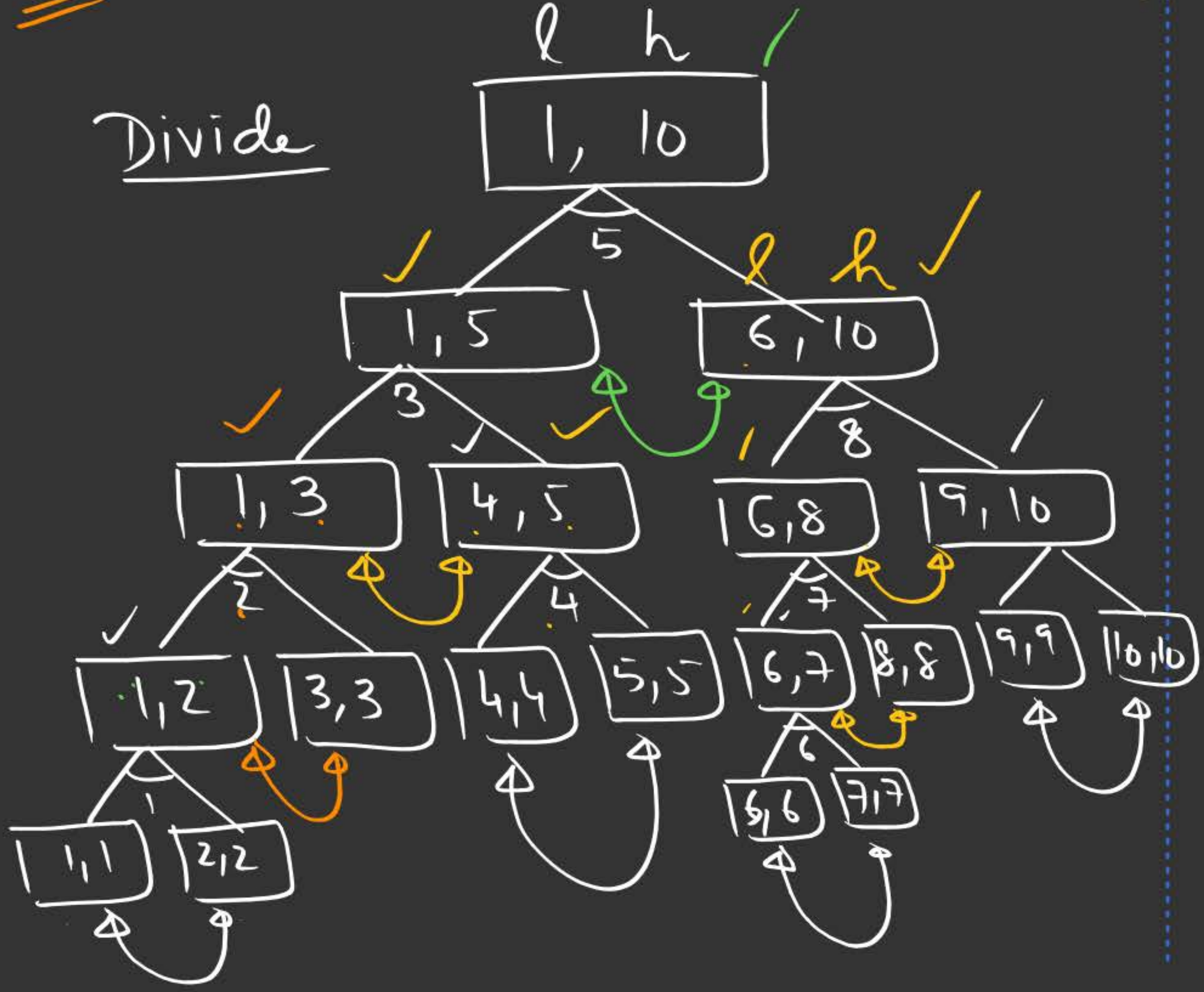
$O(n_1)$

Maximum

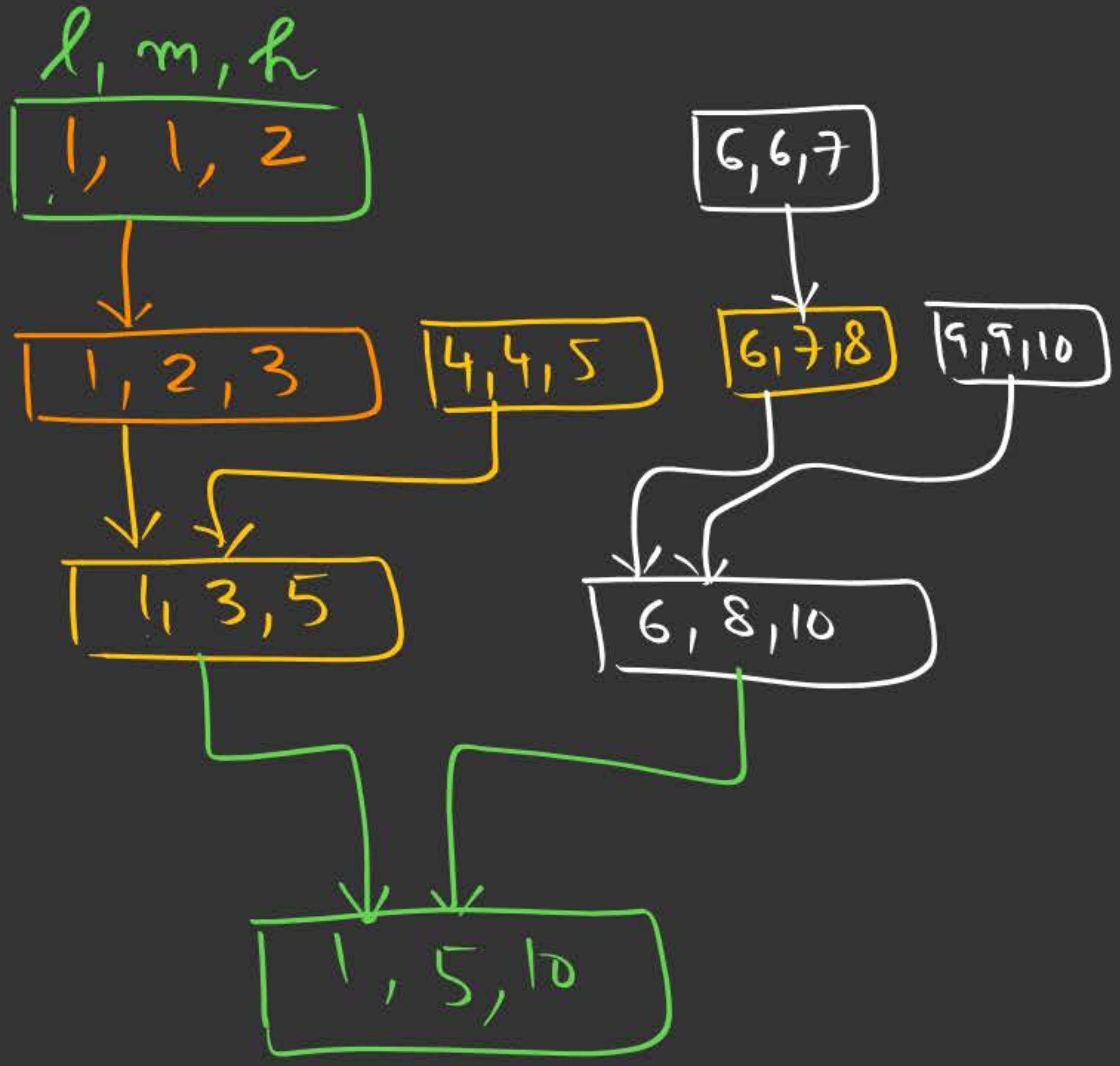
$O(n_1 + n_2)$

	1	2	3	4	5	6	7	8	9	10
IP A:	179	285	310	351	652	254	423	450	520	861
Addit Array B:	179	254	285	310	351	423	450	520	652	861

Divide



Merge-Tree



Topic : Divide and Conquer

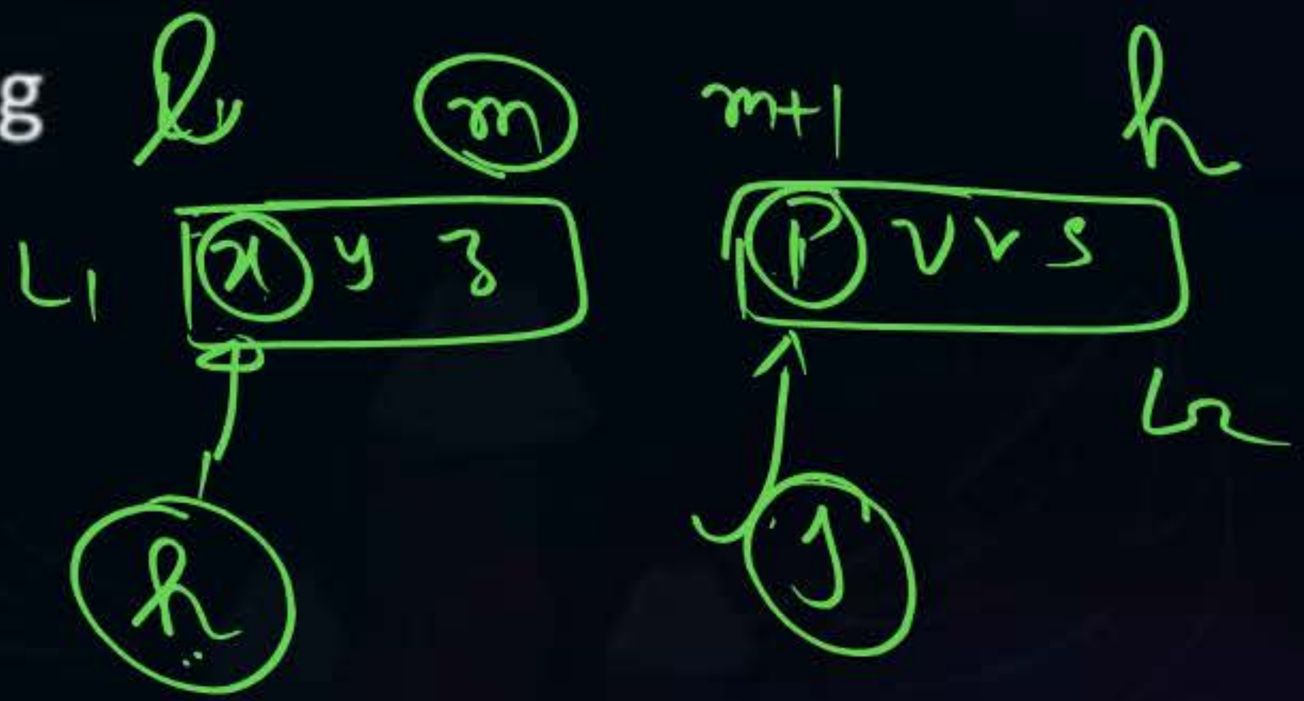
```
                            l     h
Algorithm MergeSort (low, high)
// a[low : high] is a global array to be sorted.
// Small(P) is true if there is only one element
//to sort. In this case the list is already sorted.
{
    if (low < high) then // If there are more than one element
    {
    // Divide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

Topic : Divide and Conquer

```
MergeSort(low, mid) ✓
MergeSort(mid +1, high); ✓
// Combine the solution.
     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
```



Topic : Divide and Conquer

```
  {
      b[i] := a[h];h := h +1;
  }
  else
    {
      b[i]: = a[j]; j: = j +1;
    }
    i: = l + 1;
  } *
if (h > mid) then
```

Topic : Divide and Conquer

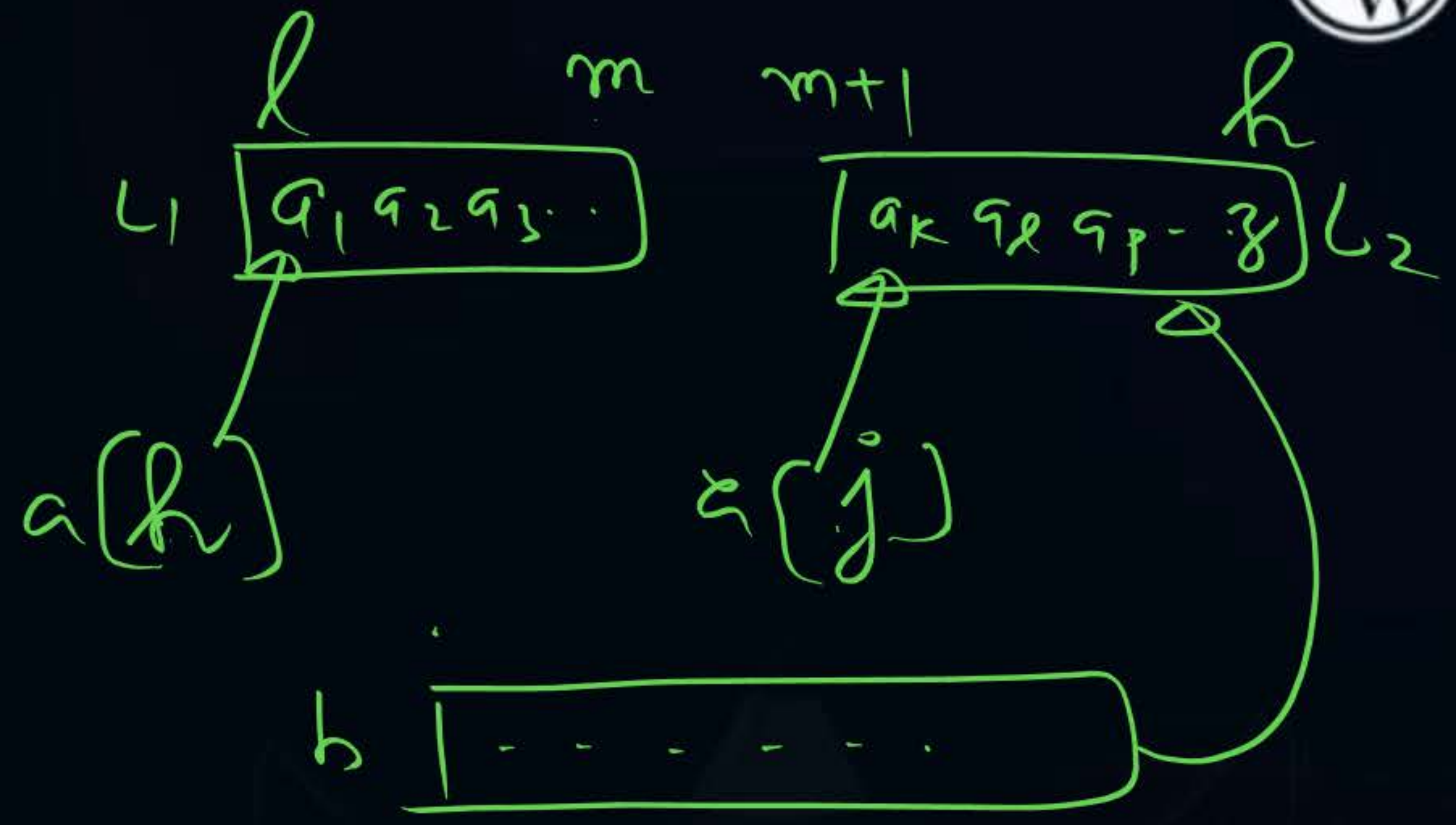
second list

```
        for k := j to high do
        {
            b[i] := a[k]; i := i+1;
        }
    else
```

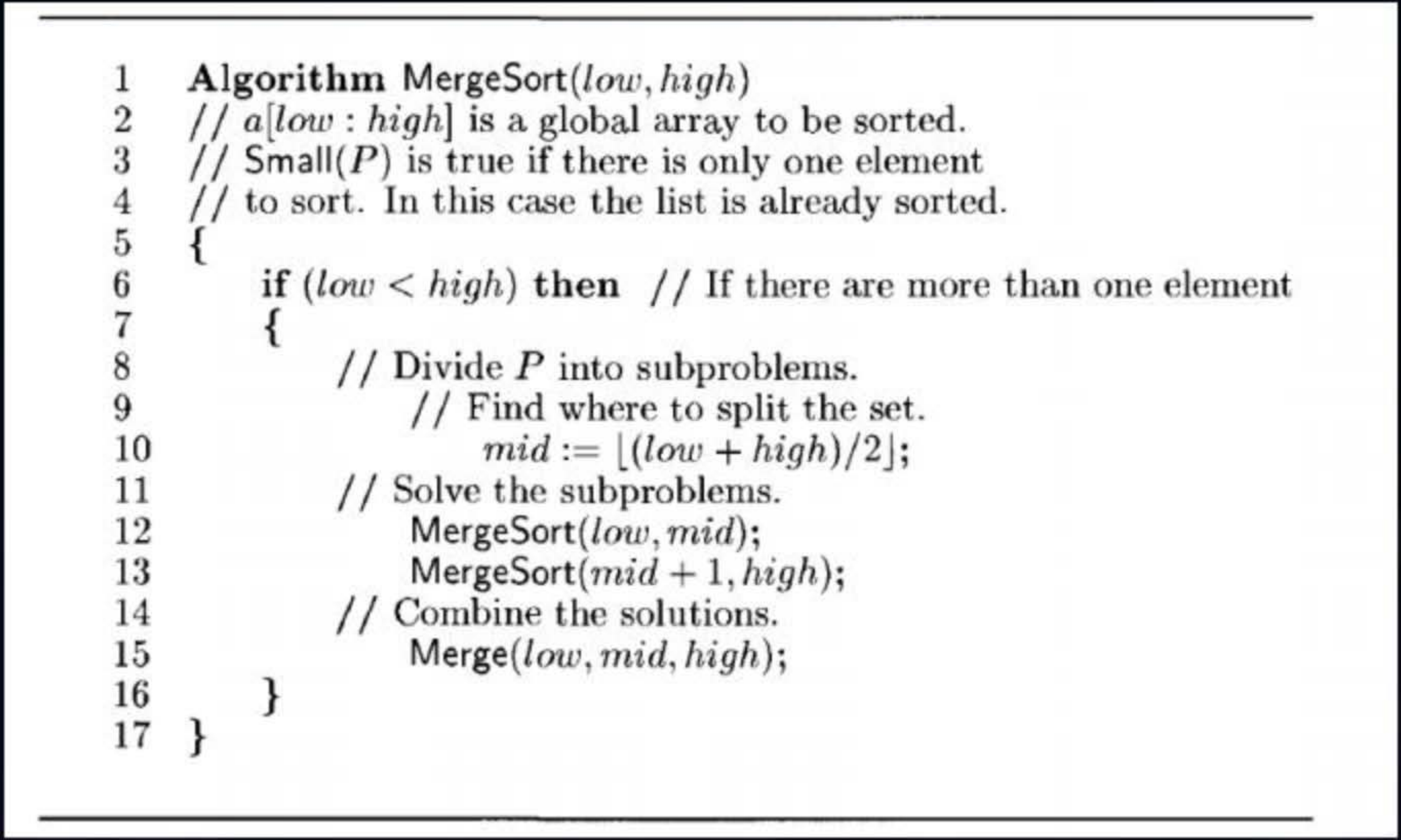
First list

```
        for k := h to mid do
        {
                    b[i] :=a[k]; i : +1;
        }
        for k := low to high do a[k] := b[k];
}
```

Copying elements from array b to a

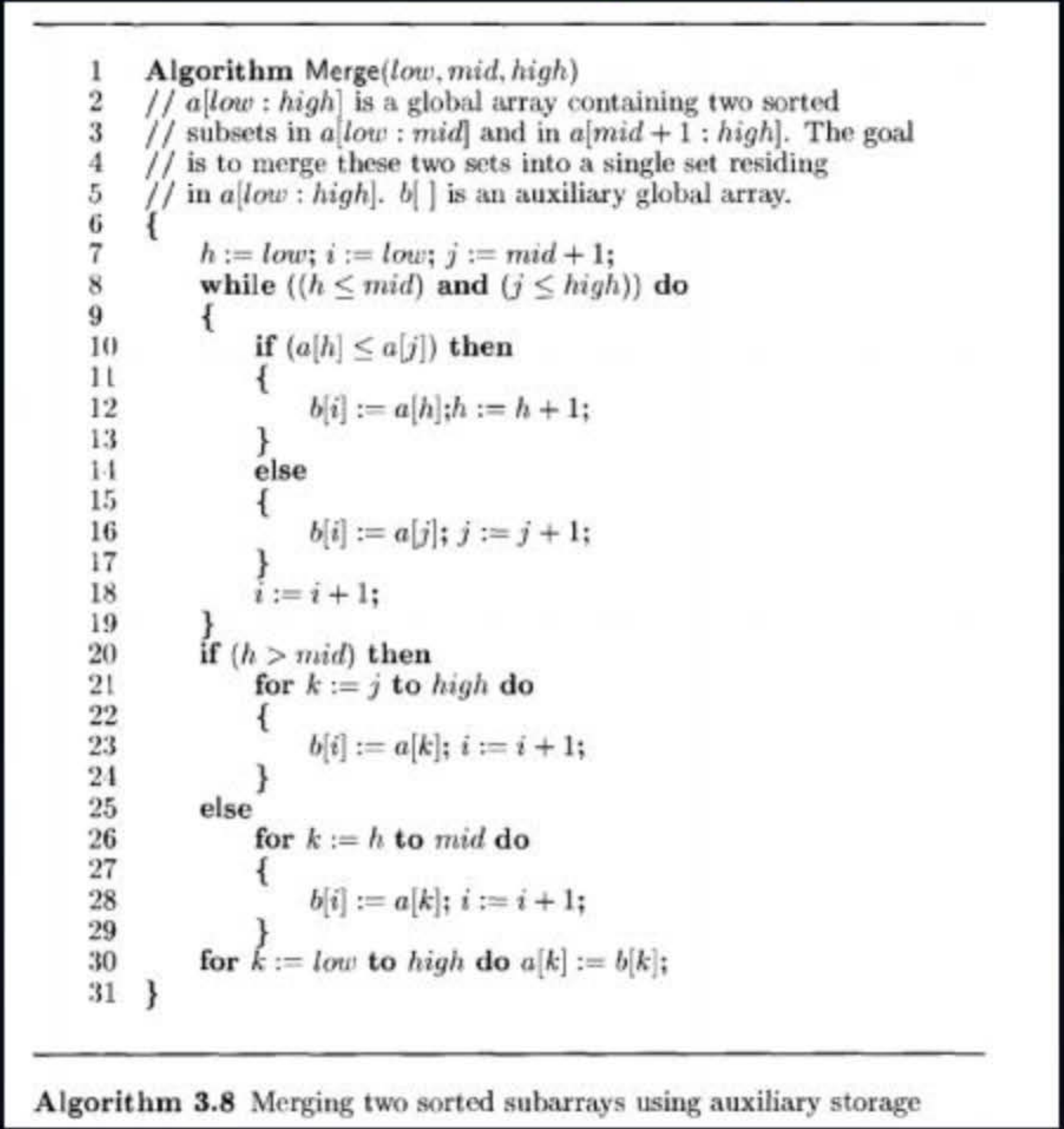


Topic : Divide and Conquer



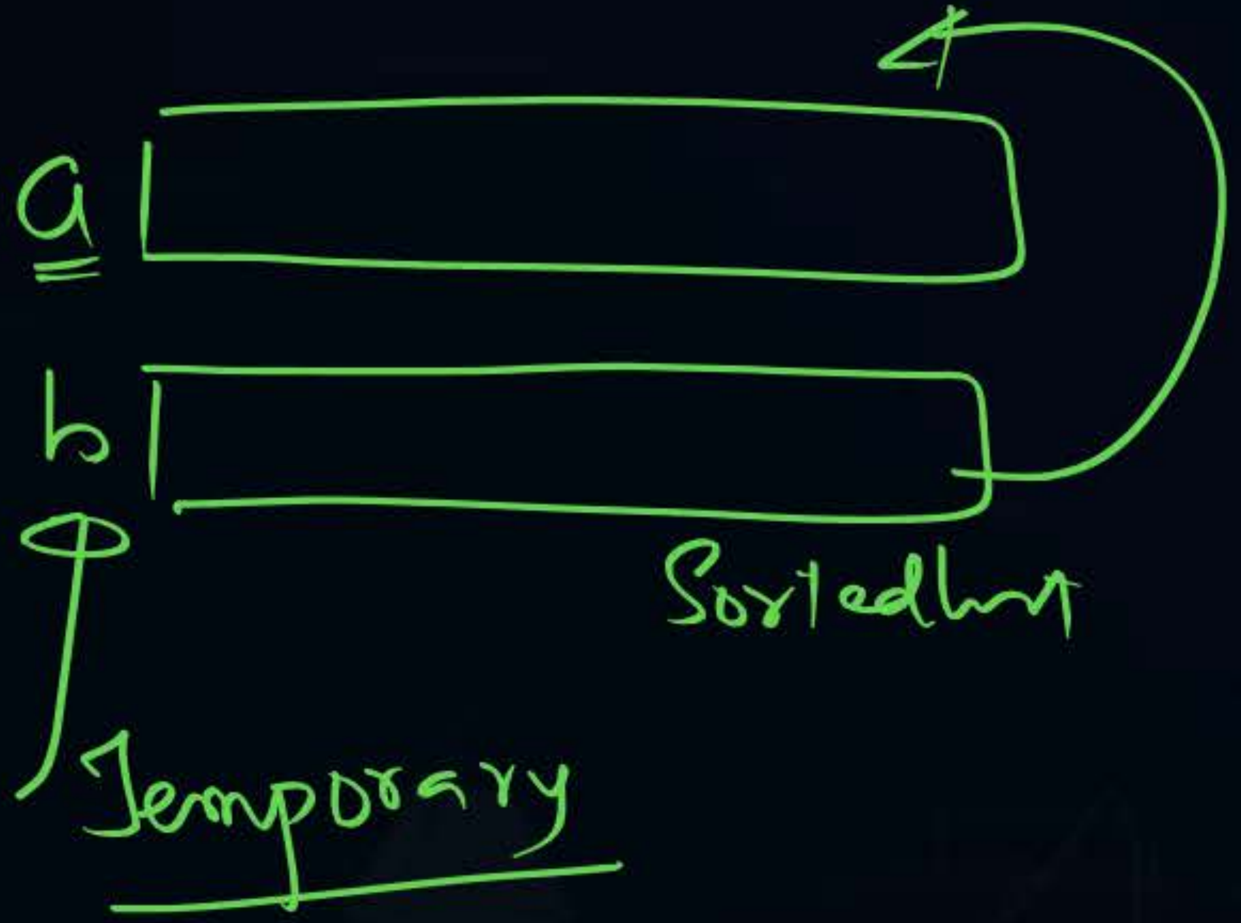
```
Algorithm MergeSort($low, high$)
// $a[low : high]$ is a global array to be sorted.
// Small($P$) is true if there is only one element
// to sort. In this case the list is already sorted.
{
    if ($low < high$) then  // If there are more than one element
    {
        // Divide $P$ into subproblems.
            // Find where to split the set.
                $mid := \lfloor (low + high)/2 \rfloor$;
        // 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 (($h \leq mid$) and ($j \leq high$)) do
    {
        if ($a[h] \leq a[j]$) 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



Performance of Mergesort:

1) **Time Complexity:**

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

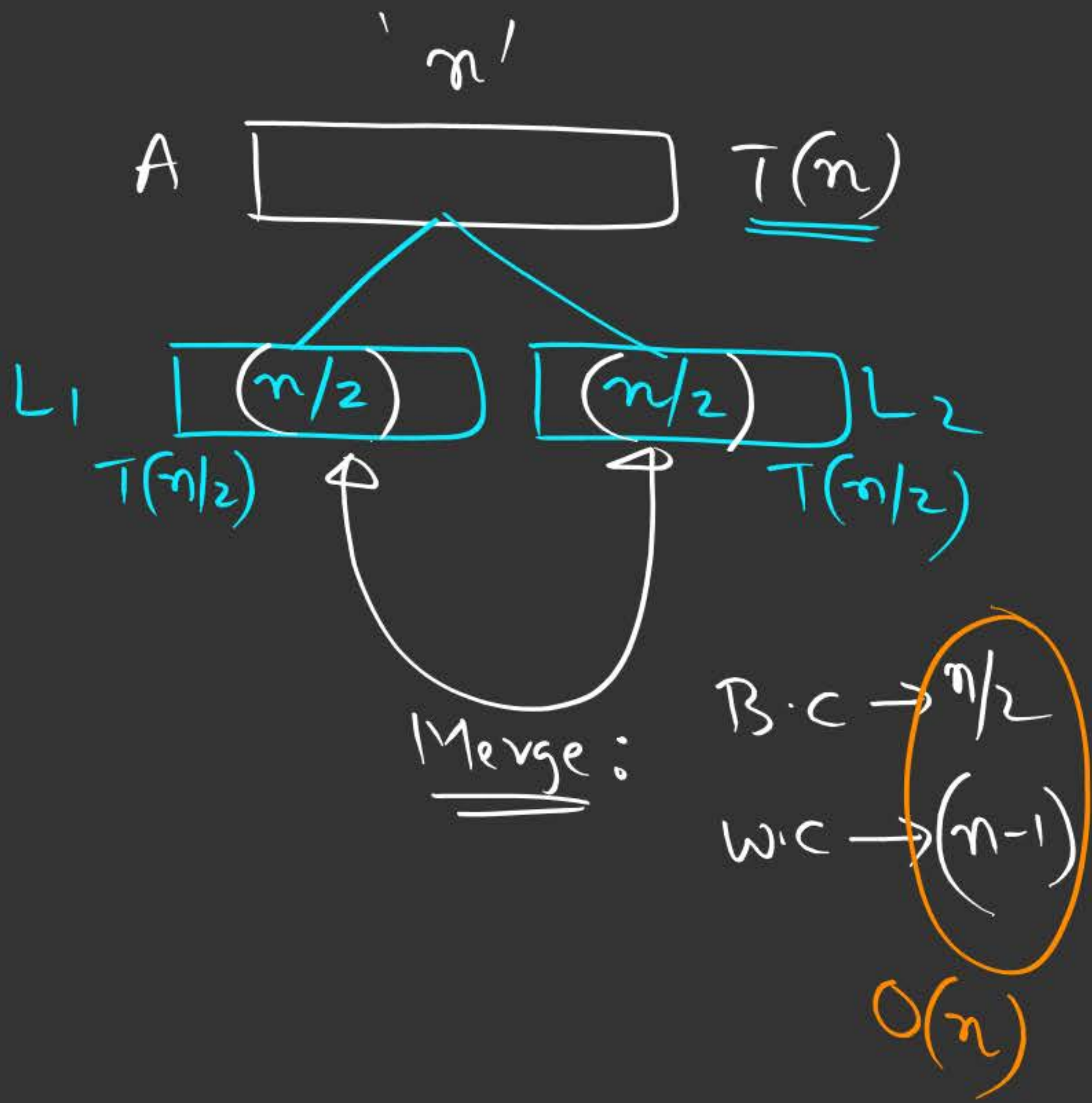
$$T(n) = c \quad , \quad n=1$$

$$= \boxed{2 \cdot T(n/2) + bn \, , \quad n>1} \qquad b>0$$

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

$$\Omega(n \log n)$$

$$\boxed{\therefore T(n) = \Theta(n \cdot \log n)}$$



THANK - YOU