CS & IT ENGINEERING

Algorithms

Divide & Conquer

Lecture No.- 02

By- Aditya sir

Topics to be Covered

Topic

Topic

Min Max DnC

Binary Search

Topic : Min-Max Problem

Algo 2:- Without DAndC

Total no of comparisons

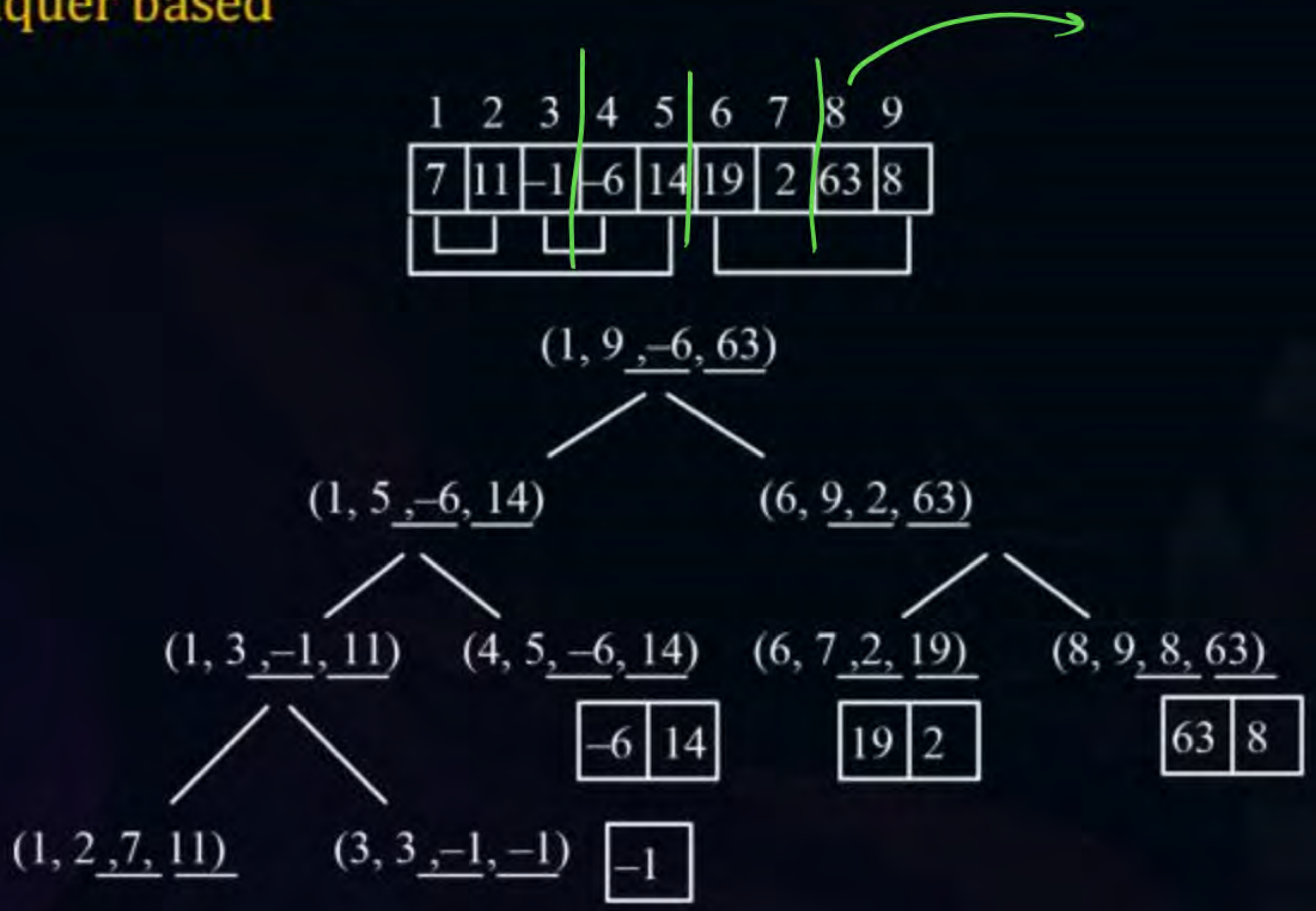
Best Case → $(n-1)$

Worst Case → $2*(n-1)$

Average Case → $\frac{3}{2}(n-1)$

Topic : Min-Max Problem

Divide & Conquer based



Topic : Min-Max Problem

```
Algorithm MinMax (i, j, min, max)
//a[1: n] is a global array. Parameters i and j are integers.
{
If (i==j) the max: min: = a[i]; // Small (P)
else if (i==j–1) then // Another case of Small (P)
  {
if (a[i] < a[j]) then
{
max:=a[j]; min : = a[i];
}
```

Topic : Min-Max Problem

```
    else
    {
            max: = a[i]; min : = a[j]
    }
}
else
{ // If P is not small, divide P into subproblems.
    // Find where to split the set.
```

Topic : Min-Max Problem

mid: $= \left\lfloor \frac{(i+j)}{2} ; \right\rfloor$

//Solve the subproblems.

MinMax (i, mid, min, max); $\longrightarrow T(n/2)$

MinMax (mid + 1, j, min1 , max1); $\longrightarrow T(n/2)$

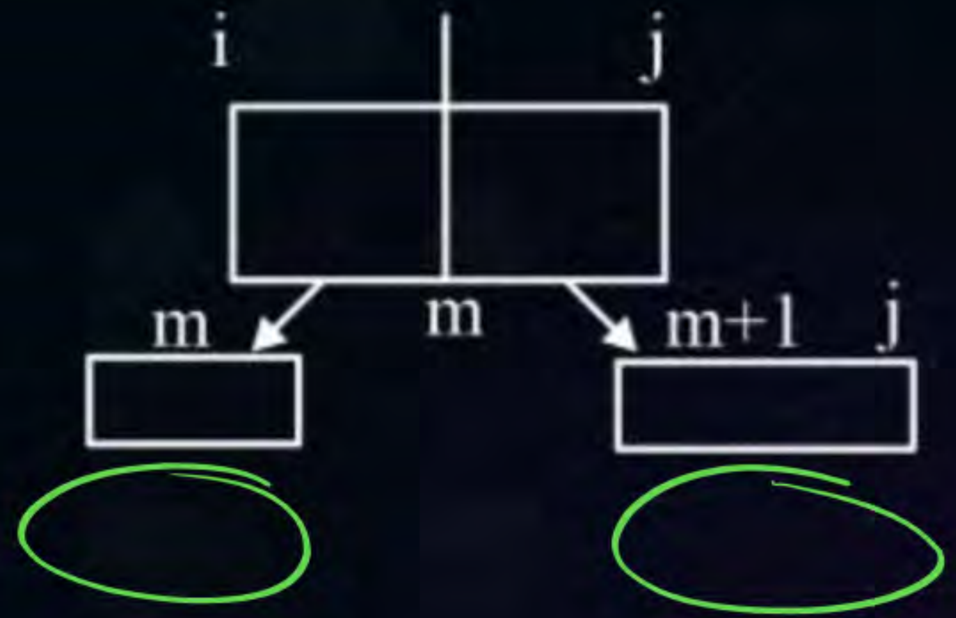
// Combine the solution.

If (max< max1) then max: =max1; $\rightarrow 1$

If ($\overset{\text{min}}{\cancel{\text{max}}}$> min1) then min: = min1; $\rightarrow 1$

}

}



Topic : Min-Max Problem

Performance of Divide & Conquer based

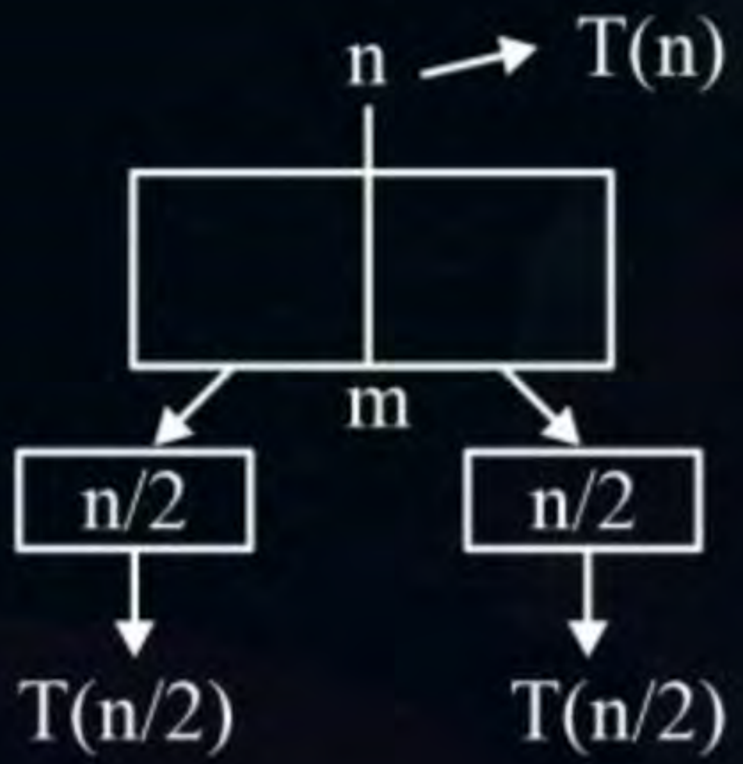
Number of element comparisons:

Let $T(n) \rightarrow$ number of element comparison required for min-max on 'n' elem.

$T(n) = 0, n = 1$

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

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



Topic : Min-Max Problem

#Q. $T(n) = 2T(n/2) + 2$

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

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

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

$$= 2^2[2T(n/2^3) + 2] + 2^2 + 2$$

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

General then

$$T(n) = 2^kT\left(n/2^k\right) + \left(2^k + 2^{k-1} \dots 2^1\right)$$

$$\Rightarrow T(n) = 2^kT\left(n/2^k\right) + \sum_{i=1}^{k} 2^i$$

Topic : Min-Max Problem

$$2^1 + 2^2 + 2^3 + \cdots + 2^k$$

GP:

- First term: a=2
- Common ratio: r=2
- Number of terms: n=k

$$\text{Sum} = \frac{a(2^k - 1)}{(2-1)}$$

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

$$= 2(2^k - 1)$$

Topic : Min-Max Problem

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

Base Condition

$n/2^k = 2$ //

$n = 2 \times \underline{2^k} \rightarrow 2^k = n/2$

$n = 2^{(k+1)}$

$(k+1) = \log_2 n$

$$\Rightarrow \frac{n}{2} \times T(2) + 2\left(\frac{n}{2} - 1\right)$$

$$\Rightarrow \frac{n}{2} \times 1 + n - 2$$

$$\Rightarrow \frac{n}{2} + n - 2 = \boxed{\frac{3n}{2} - 2}$$

$$\boxed{TC = O(n)}$$

Topic : Min-Max Problem

Summary: DnC vs D&C → Min-Max Algo:

V.V.V. Imp

	Non-DnC (Algo2)	DnC	
Best case Algo2 ~~Decr order~~	(n–1) ✓	$\left(\frac{3n}{2}-2\right)$ ✓	
Wrost case Algo2 ~~Incr order~~	2*(n–1)	$\left(\frac{3n}{2}-2\right)$ ✓	
Avg case (Random order)	$\left(\frac{3}{2}(n-1)\right)$	$\left(\frac{3n}{2}-2\right)$	

Topic : Min-Max Problem

Space complexity:-

Non-D&C (Algo2) $\rightarrow O(1)$

D&C $\rightarrow$ Recursion stack $\rightarrow O(\log_2 n)$

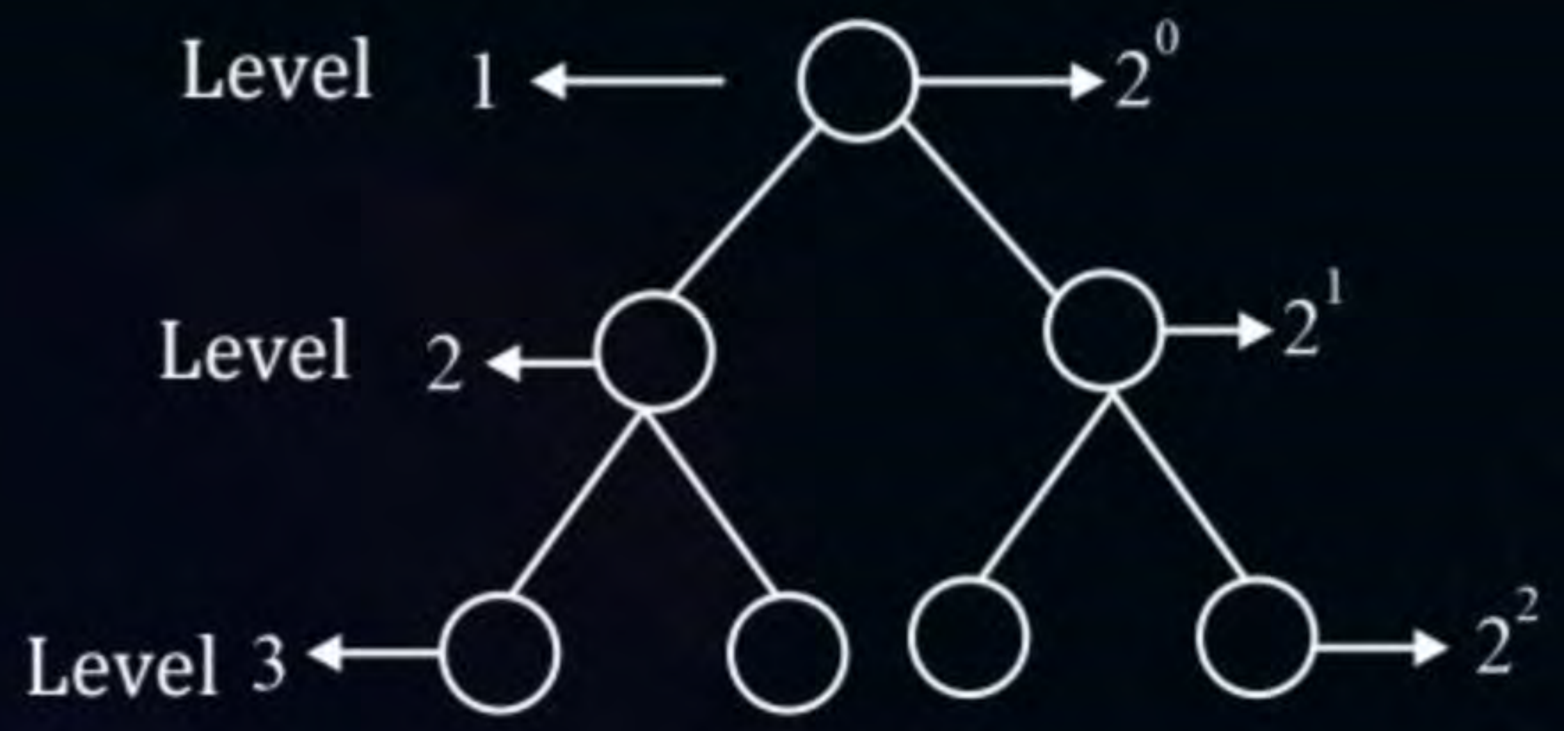
Topic : Binary Search

Binary Search:

Topic : Binary Search

#Q. Given a Binary Tree of 'n' nodes, ~~elements~~
min. Height (depth) = ?
max. Height (depth) = ?

Topic : Binary Search



Topic : Binary Search

Max number of nodes at any level 'i' in BT = $2^{(i-1)}$

10

0, 2

min

5

max

10

Topic : Binary Search

#Q. Total number nodes Pn a full Binary Tree of height h.

$$= 2^{1-1} + 2^{2-1} + 2^{3-1} \dots + 2^{h-1}$$

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

Total nodes:

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

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

$$n = 2^h - 1$$

$$2^h = n + 1$$

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

Topic : Binary Search

Level:

Every level 1 element

Height = h

Total nodes = $\sum_{i=1}^{h} 1$

$$n = \frac{(1 + 1 + \ldots 1)}{\text{h times}}$$

n = h ⇒ h = n

Max-height of a BT

Topic : Binary Search

Range of height of a Binary Tree: with n nodes
assuming root at level = 1

$$\boxed{\log_2^n \le h \le n}$$

Topic : Min-Max Problem

#Q. Given a Binary Tree
level starts at 1
every level i has exactly 'i' nodes
Height of such a BT in order of ?

A. n^2

B. $\sqrt{n}$

C. n

D. $\log n$

Topic : Design Strategies

Eg. Level i → i nodes

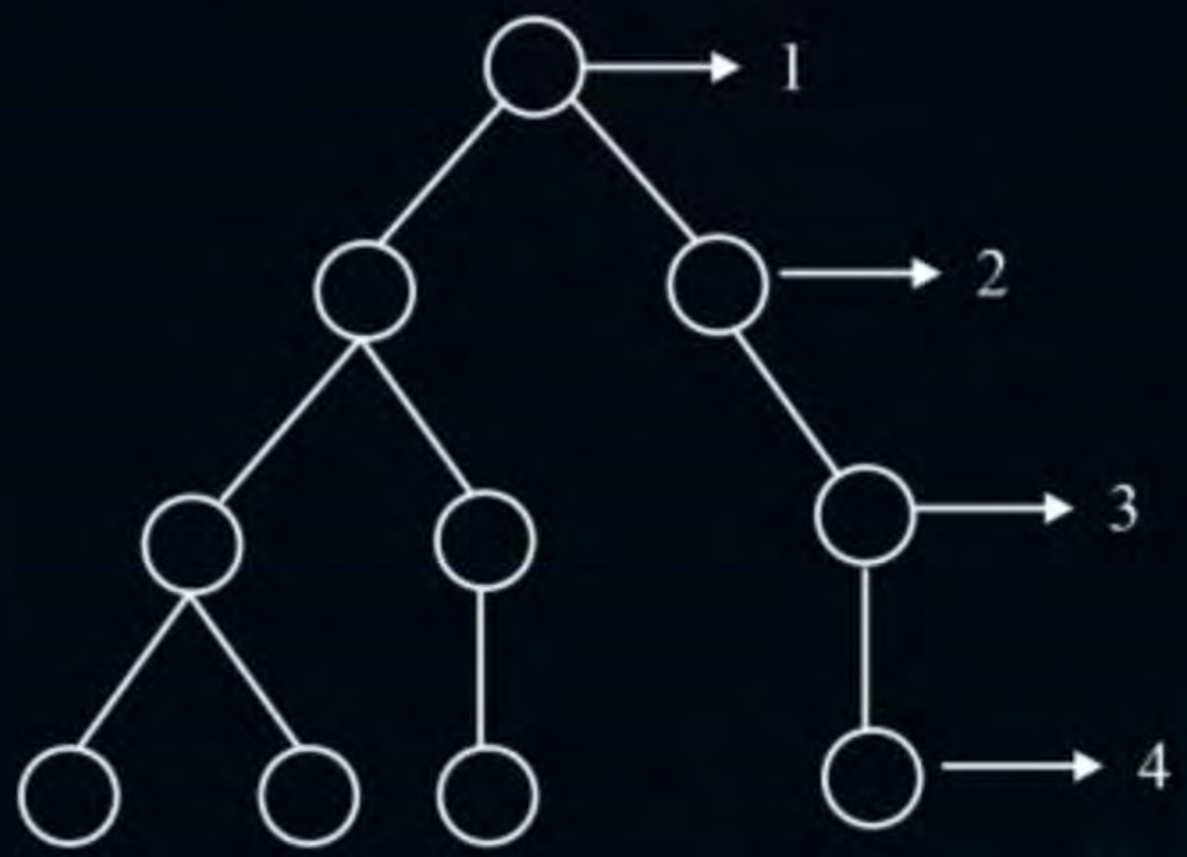
Total nodes = $\sum_{i=1}^{h} i$

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

$$\boxed{n = \frac{h^2+h}{2}}$$

$h^2 \approx n$

$h \approx O(\sqrt{n})$



Topic : Linear Search

Linear Search:-

```
Algo AJ Linear (A, n, x)
{
for (i  = 1;, i < = n; i++)
if (A[i] = = x)
{
        return i;
}
}
Printf(" Not found");
}
```

Topic : Linear Search

Linear Search TC Analysis:-

(1) Best case: $A = \begin{array}{|c|c|c|c|}\hline 2 & 20 & 1 & 10 \\ \hline \end{array}$ $x = 5$ 2

(2) Worst case: $A = \begin{array}{|c|c|c|c|}\hline 2 & 5 & 10 & 7 \\ \hline \end{array}$

$x = 7$ or $x = 8$

$x = 7 \rightarrow$ elem at last positive

$x = 8 \rightarrow$ Not present

Always work

Topic : Linear Search

Does linear search take any advantage when the input array is sorted?

→ No

Here we ~~need~~ need Binary Search for such case

will

Topic : Binary Search

Binary search → Divide & Conquer

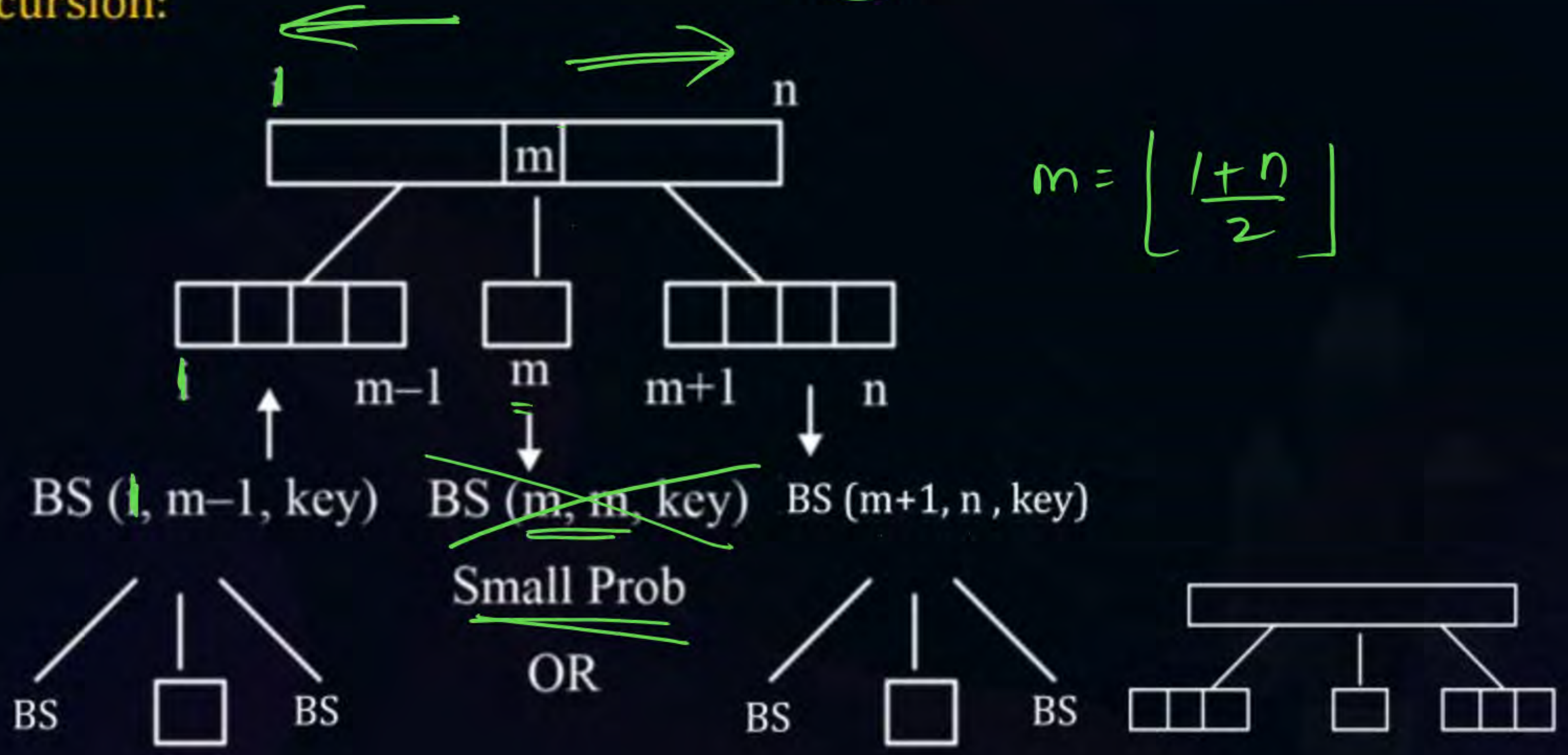
Pre-requisite: Input array should be in sorted order (Mandatory Requirement)

→ No such requirement in linear search.

Topic : Binary Search

x

Recursion:



$m = \left\lfloor \frac{1+n}{2} \right\rfloor$

Topic : Binary Search

Key = x

Recursion:

$$
\begin{cases}
C1: \quad Key = A[m] & \rightarrow \text{Small prob Solved} \\
C2: \quad Key > A[m] & \rightarrow \text{Explore BS}(\mathrm{m}+1, \mathrm{n}, \text{key}) \\
C3 \quad key < A[m] & \rightarrow \text{Explore BS}(1, \mathrm{m}-1, \text{key})
\end{cases}
$$

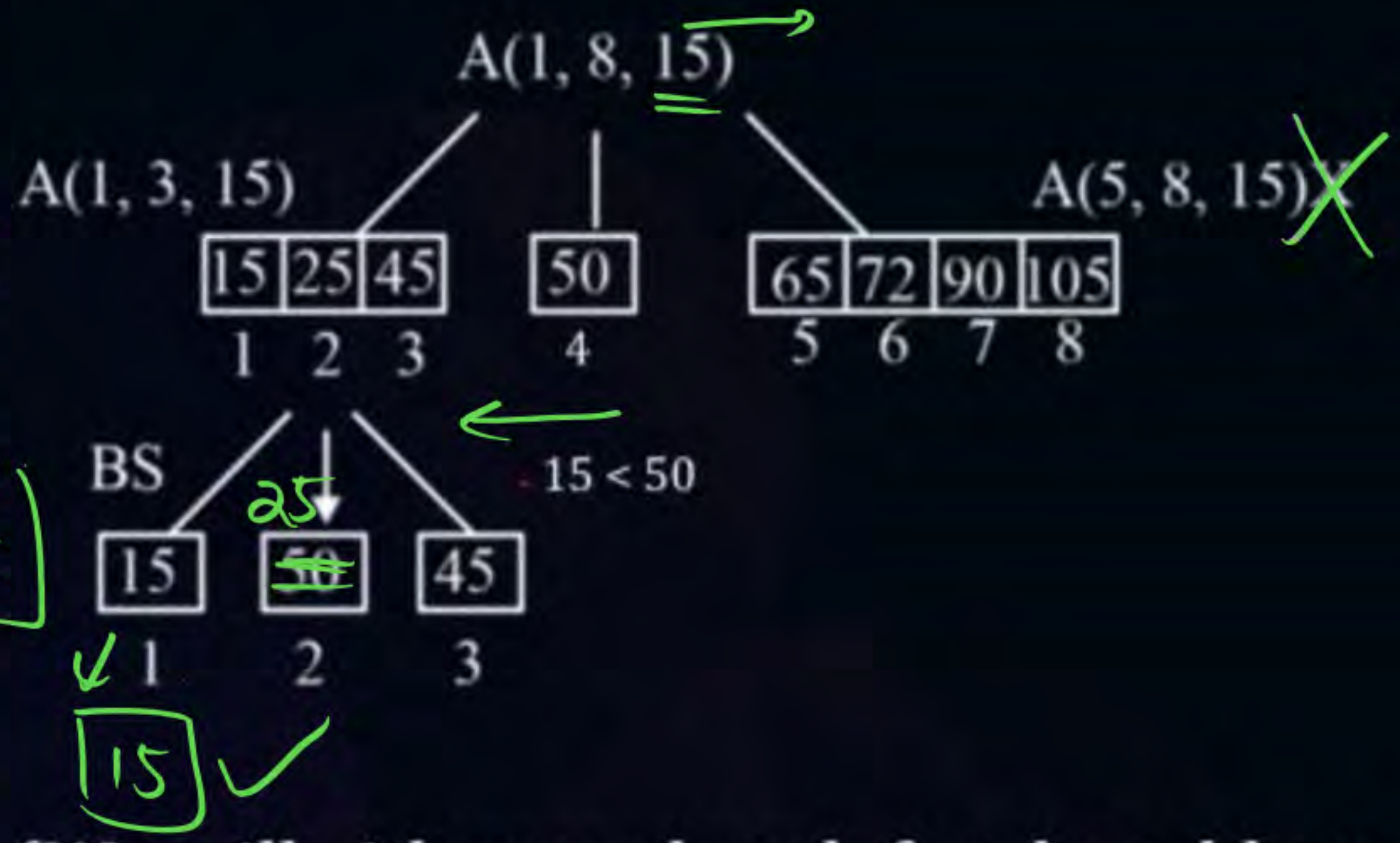
Topic : Binary Search

Eg.

15	25	45	50	65	72	90	105
1	2	3	4	5	6	7	8

, Key =15

Idea:



$m = \left\lfloor \frac{1+8}{2} \right\rfloor = 4$

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

$m = \left\lfloor \frac{1+1}{2} \right\rfloor = 1$

{We will either explore left subproblem or right subproblem BUT not both}

Topic : Divide and Conquer

Recursive

```
Algorithm BinSrch(a, i, l, x)
// Given an array a[i: l] of elements in non decreasing
// order, 1 ≤ i ≤ l, determine whether x is present, and
// if so, return j such that x = a[j]; else return 0.
{   l==i
if (l = i) then // If Small(P)
{
if (x = = a[i]) then return i;
else return 0;
}
```

i l

Topic : Divide and Conquer

```
else
{ // 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

Logic:

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

$$= b, n = 1$$

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

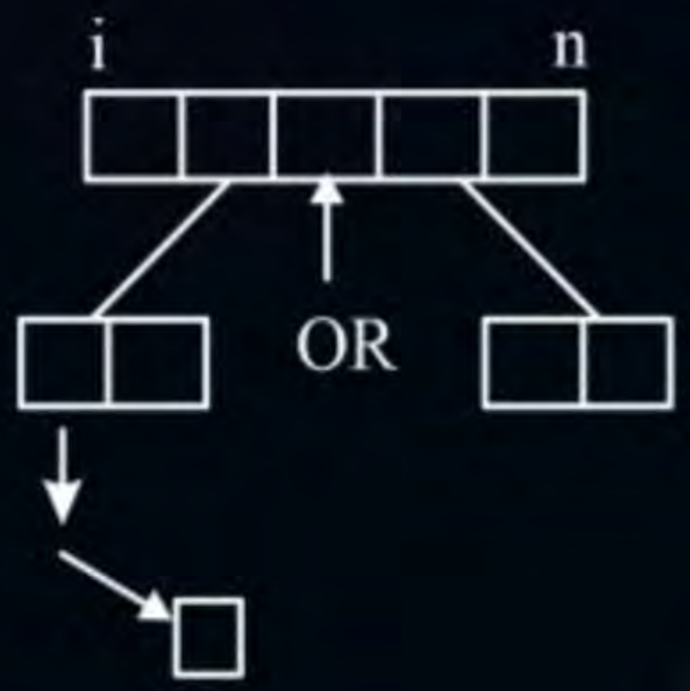
$$T(n) = T(n/2^2) + 2C$$

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

General:

$$\boxed{T(n) = (n/2^k) + k^*c}$$

for B.C, $\frac{n}{2^k} = 1 \Rightarrow 2^k = n$



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

$$= b + c^* \log_2^n$$

$$\boxed{T(n) = O(\log_2^n)}$$

Topic : Divide and Conquer

Hence, In Binary search there is only divide and no need to combine

Topic : Divide and Conquer

Non-recursive/Iterative

Algo Binary search (A, n, x)

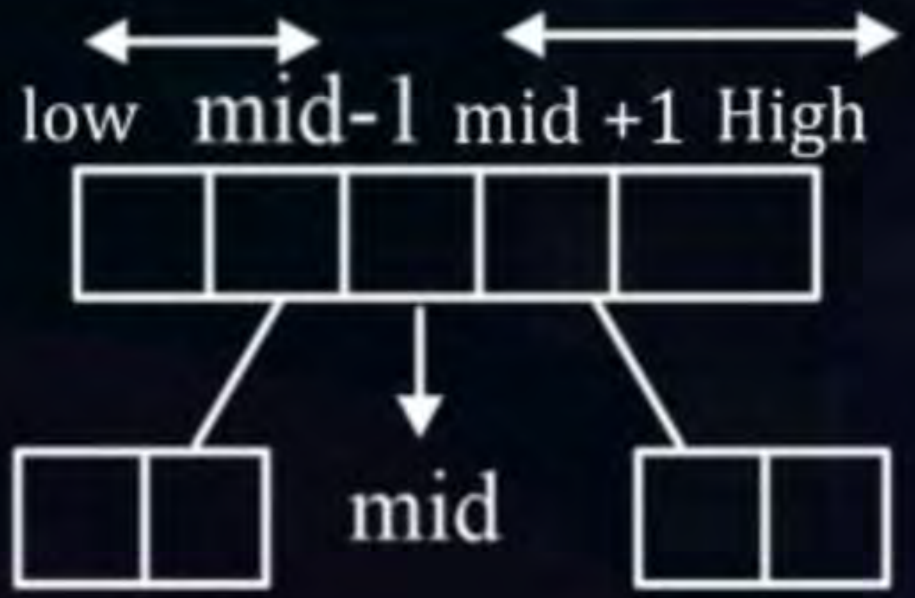
// A[i..n] → non-decreasing order

// return index of x, else o.

```
Low = 1, high = n;
While (low ≤ high){
      mid = ⌊(low+high)/2⌋;
if(A[mid] > x)
      high = mid-1;
else if (A[mid]) < x)
      low = mid+1;
else
      return mid;
   }
   return-1;
}
```

SC : O(1)

Search space [low → High]



Topic : Divide and Conquer

Space Complexity analysis

(1) Non-Recurisive

$\Rightarrow$ SC = $O(1)$

(2) Recursive

$\boxed{SC = O(\log_2^n)}$

$\downarrow$

Recursion stacks

$\boxed{k = \log_2^n}$

$n/2^k = 1$

K

$n/2^k$
$\vdots$
$n/2^2$
$n/2$
n

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

$2^k = n$

$\boxed{K = \log_2 n}$

THANK - YOU