CS & IT ENGINEERING

Data Structures & Programming

Tree

Lec- 04

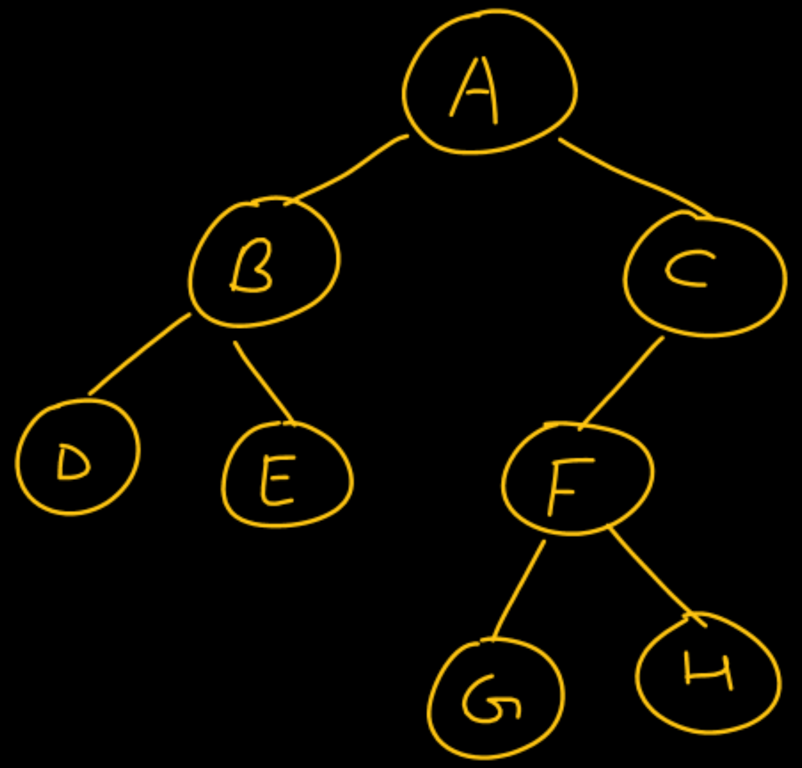
By- Pankaj Sharma Sir

TOPICS TO BE COVERED

Tree 04

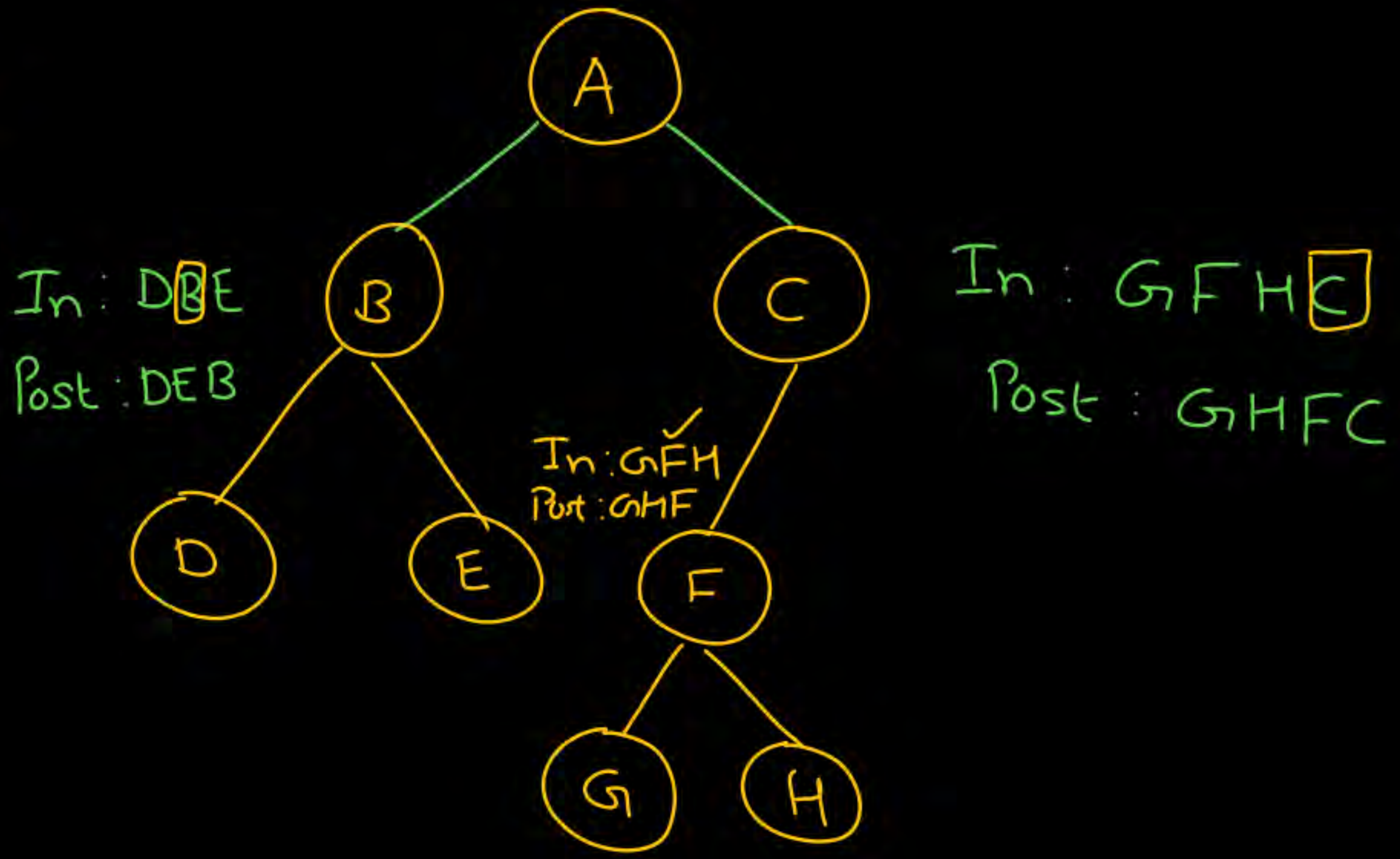
In: DBEAGFHC

Post: DEBGHFCA



In: $\boxed{\text{DBE}}$ AGFHC

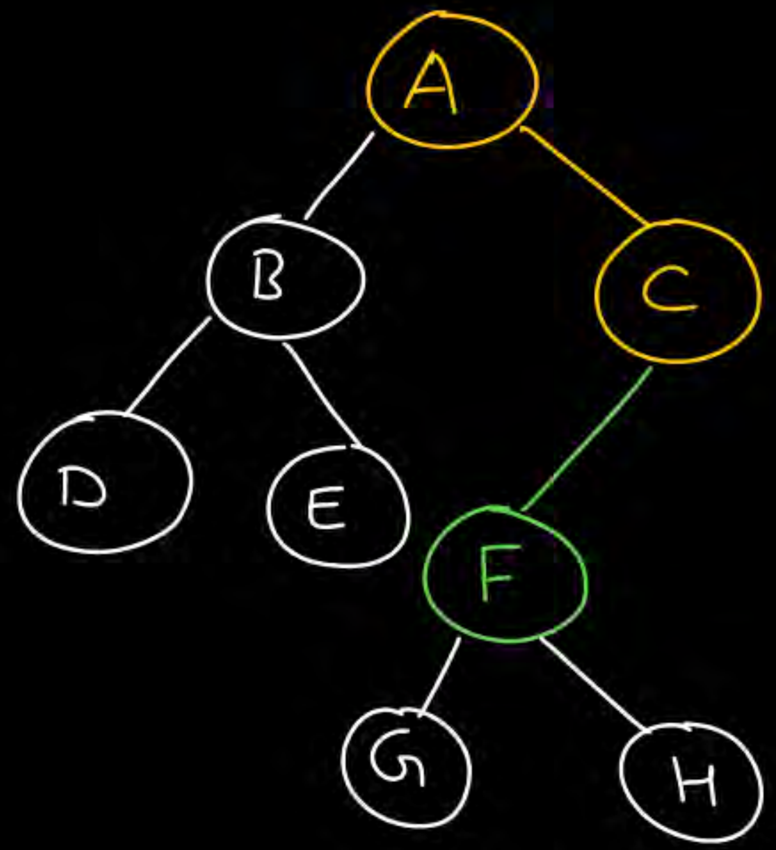
Post: DEBGHFCA



Short-trick

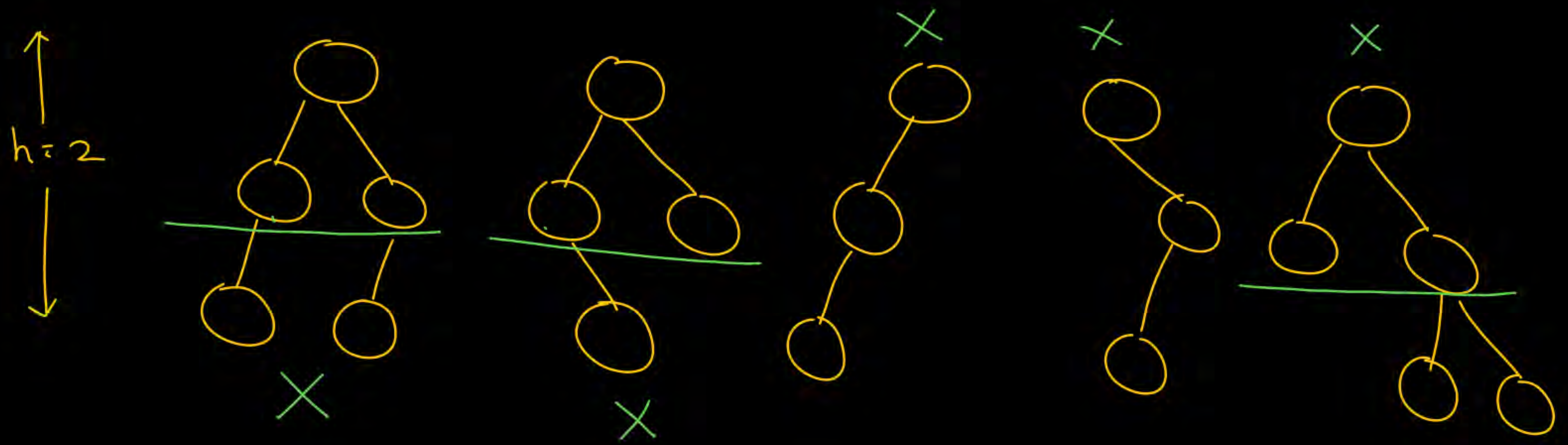
In : D B E A G F H C

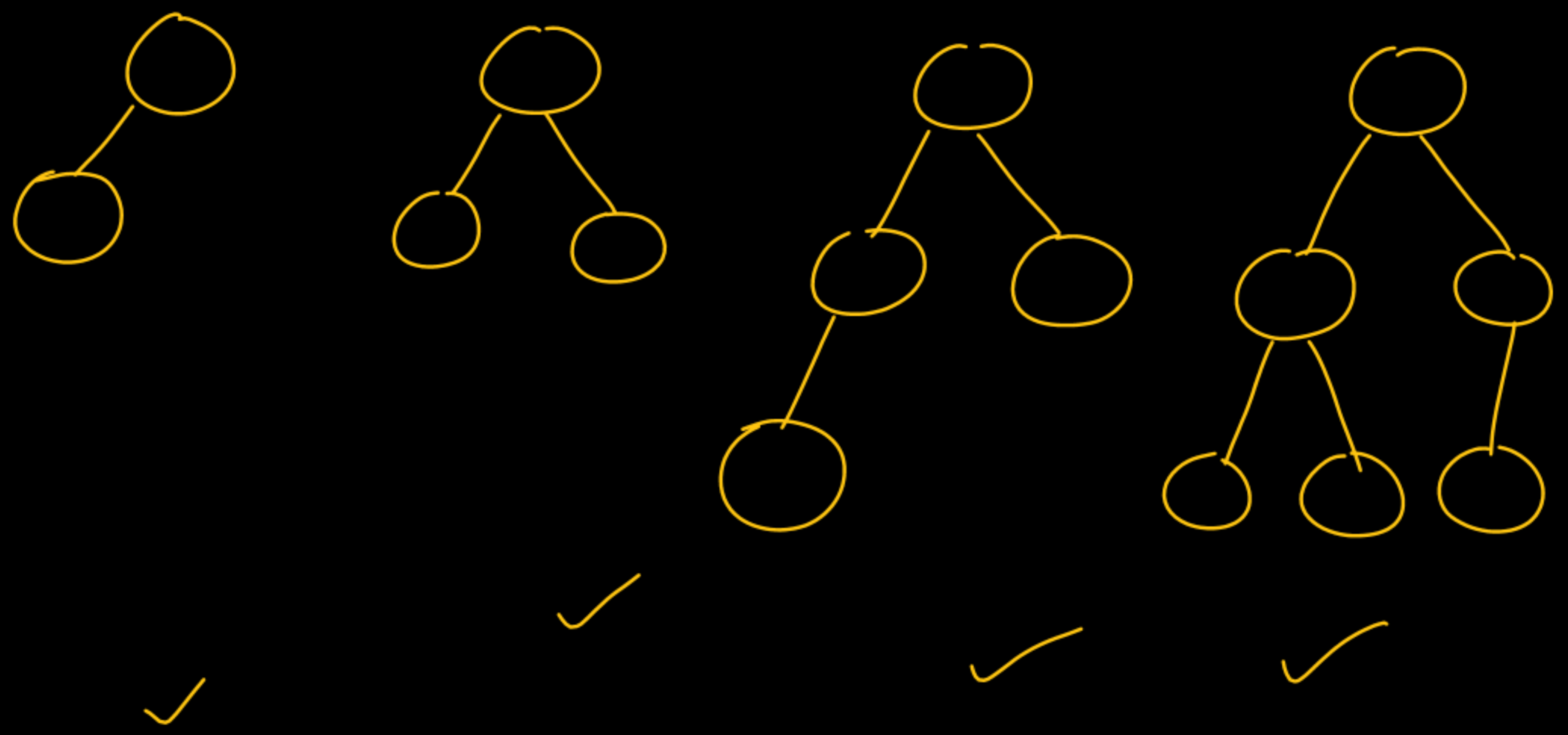
Post : D E B G H F C A



Complete Binary Tree

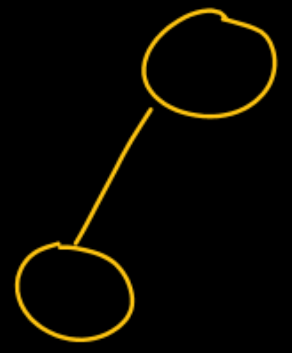
A CBT is a binary tree which is full upto second last level and nodes at last level are filled from left to right.



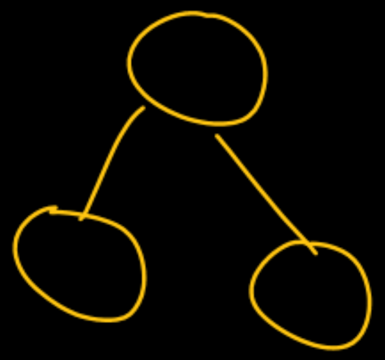


1.) structure of a CBT with 1 node

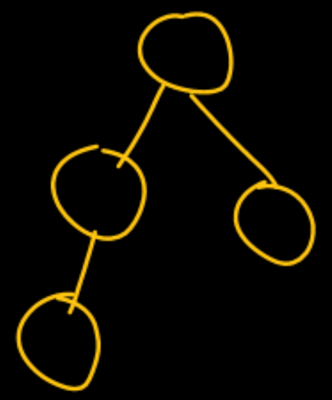
2.) structure of a CBT with 2 node.



3.) structure of a CBT with 3 nodes

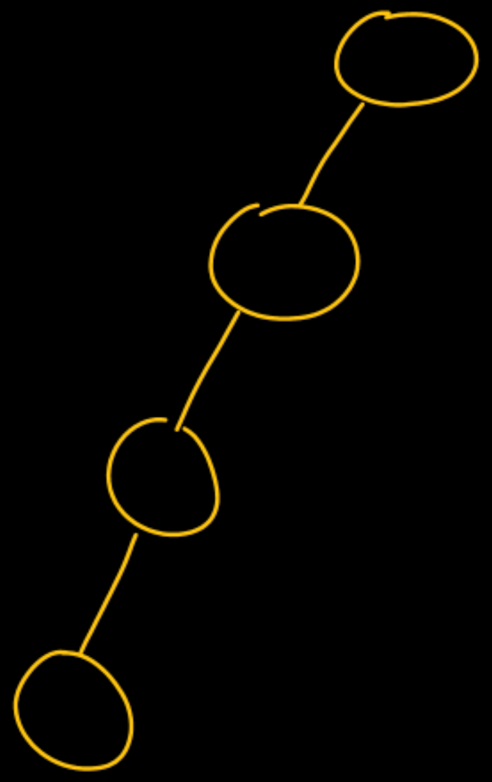


4.) - — — — 4



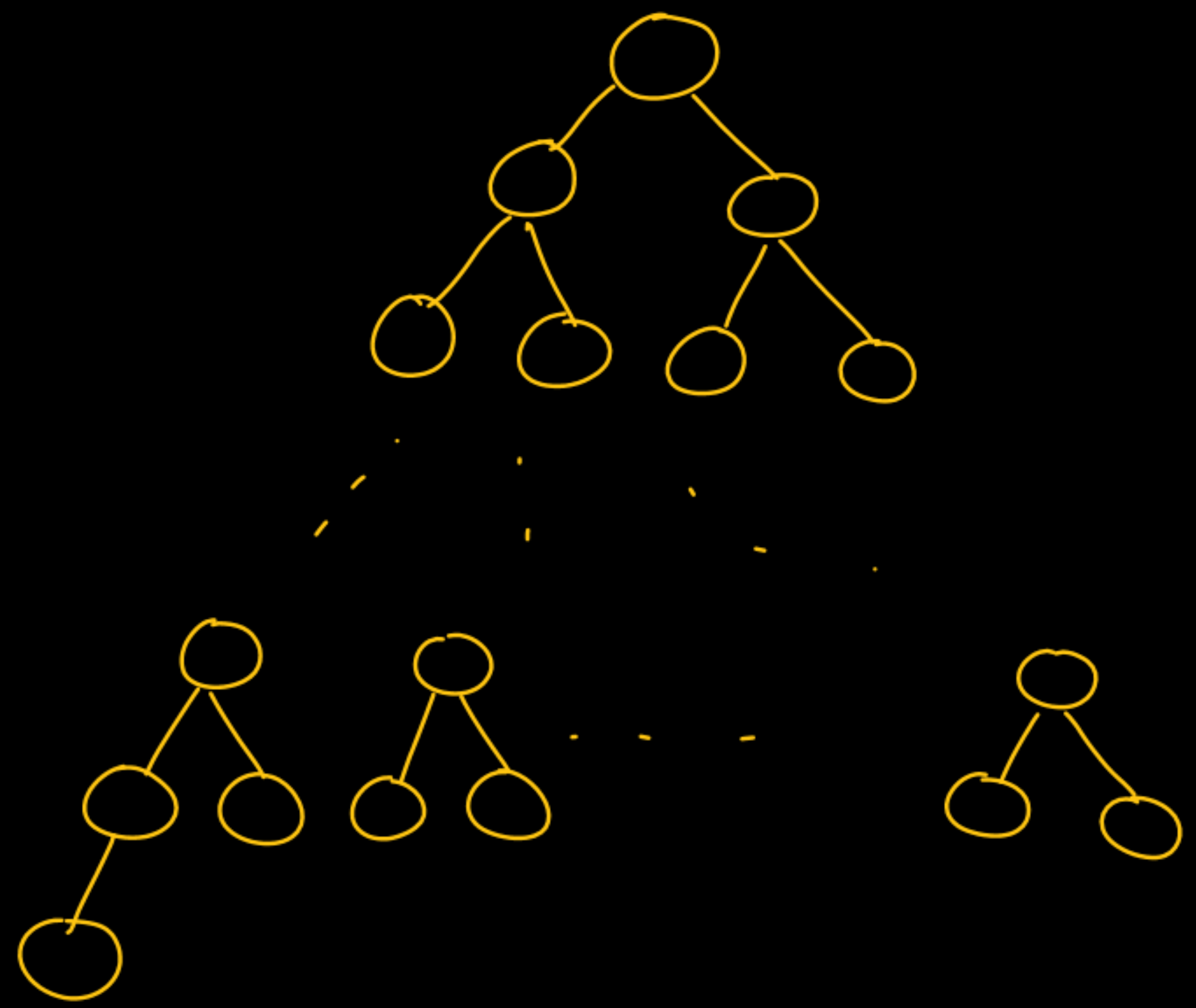
max. no. of nodes in a CBT of height $h = 2^{h+1} - 1$

min " " " $h = h+1$ X



X
CBT

# node	Level
1	0
2^1	1
2^2	2
	$\vdots$
2^{h-2}	$h-2$
2^{h-1}	$h-1$
1	h



$$n = \left(1 + 2^1 + 2^2 + \cdots + 2^{h-1}\right) + 1$$

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

$$2^h \le n$$

$$\log 2^h \le \log n$$

$$h \log 2 \le \log n$$

$$h \le \frac{\log n}{\log 2}$$

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

$$\boxed{\log_2(n+1)-1 \le h \le \log_2 n}$$

$$\boxed{2^h \le n \le 2^{h+1}-1}$$

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

$$n \le 2^{h+1}-1$$

$$n+1 \le 2^{h+1}$$

$$\log(n+1) \le (h+1) \cdot \log 2$$

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

$$h+1 \ge \log_2(n+1)$$

$$\boxed{h \ge \log_2(n+1)-1}$$

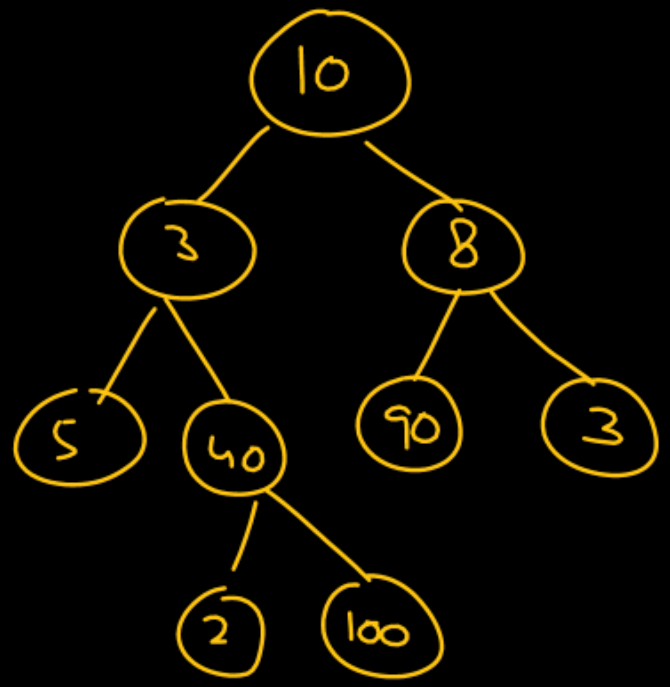
Binary Search Tree

Why?

Given a binary tree and a key, find whether the key is Present in the tree or not

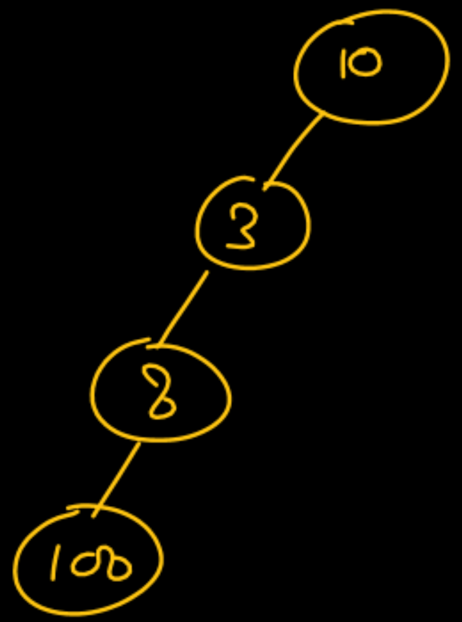
Key = 100

B.T



$O(n)$

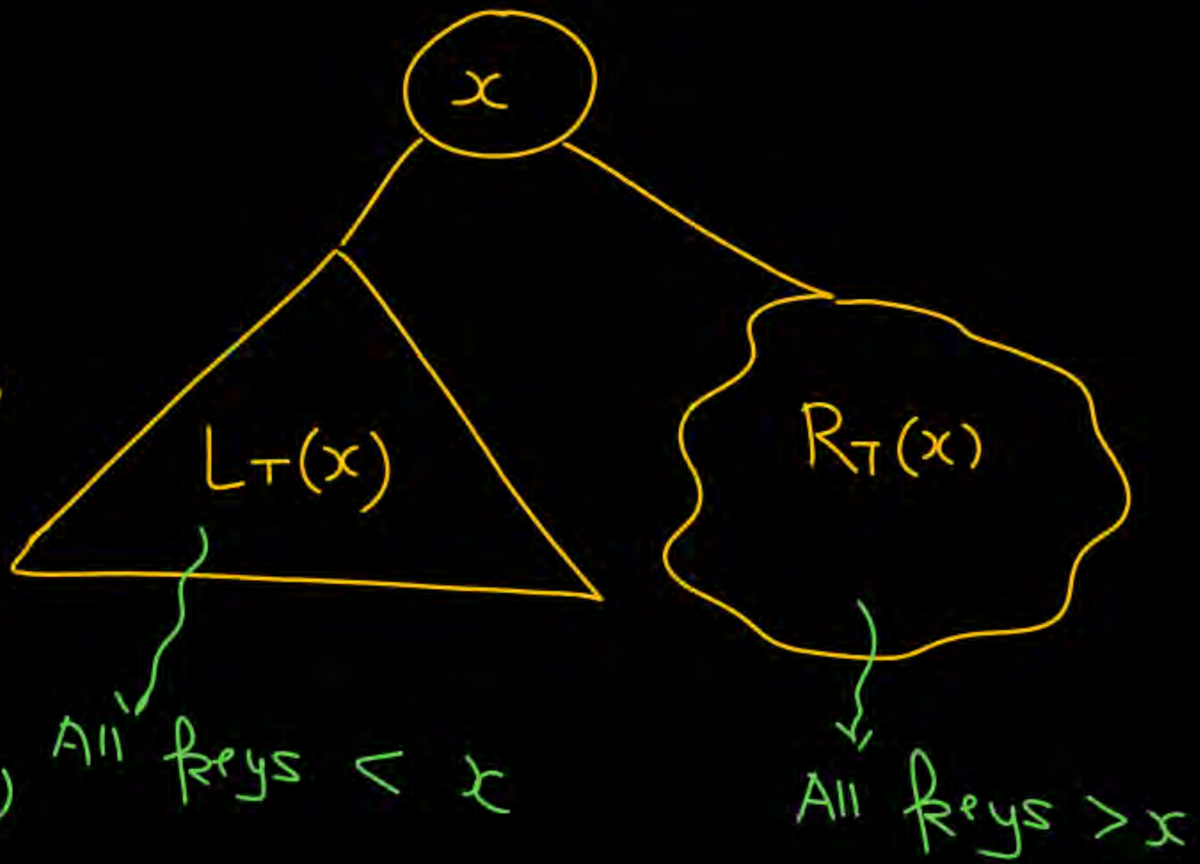
B.T

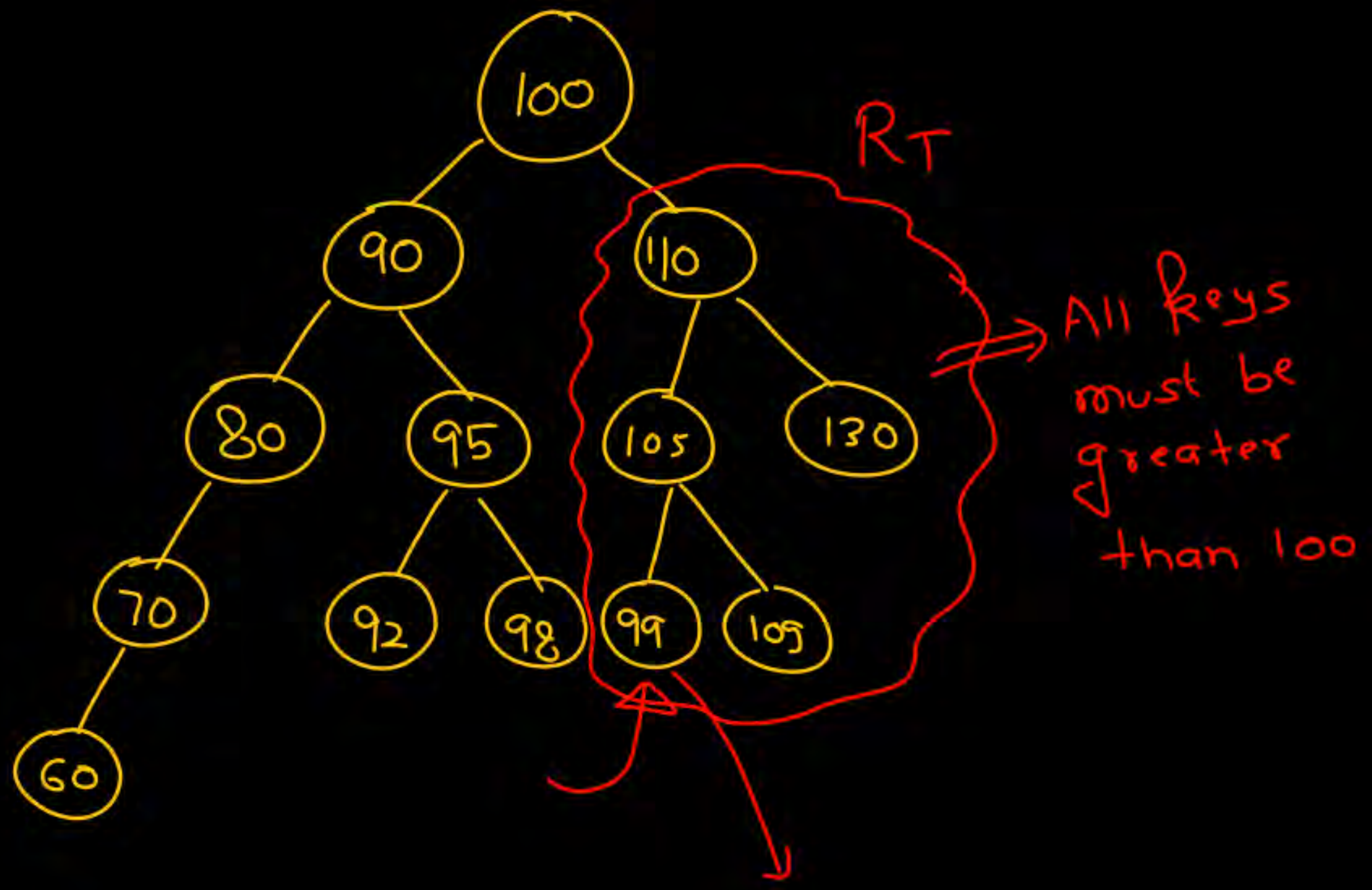


Binary Search Tree

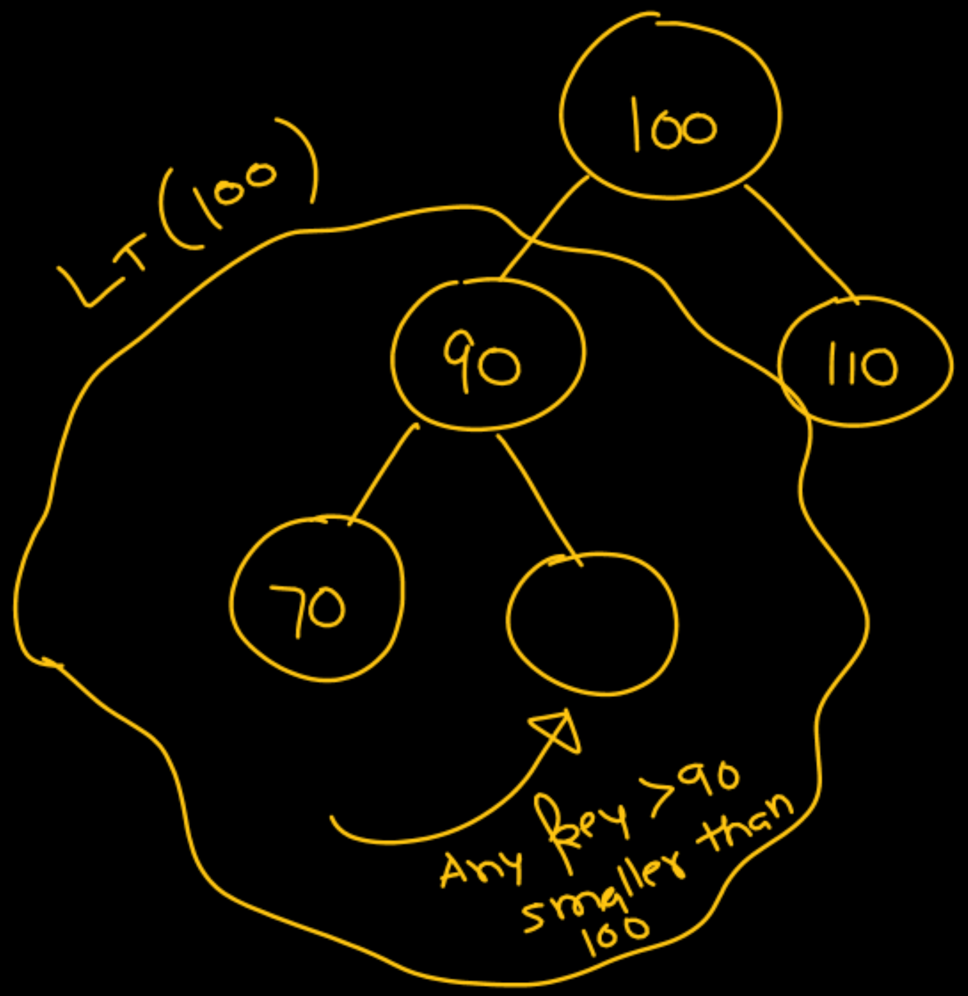
A BST is a binary tree in which every node satisfies the property:

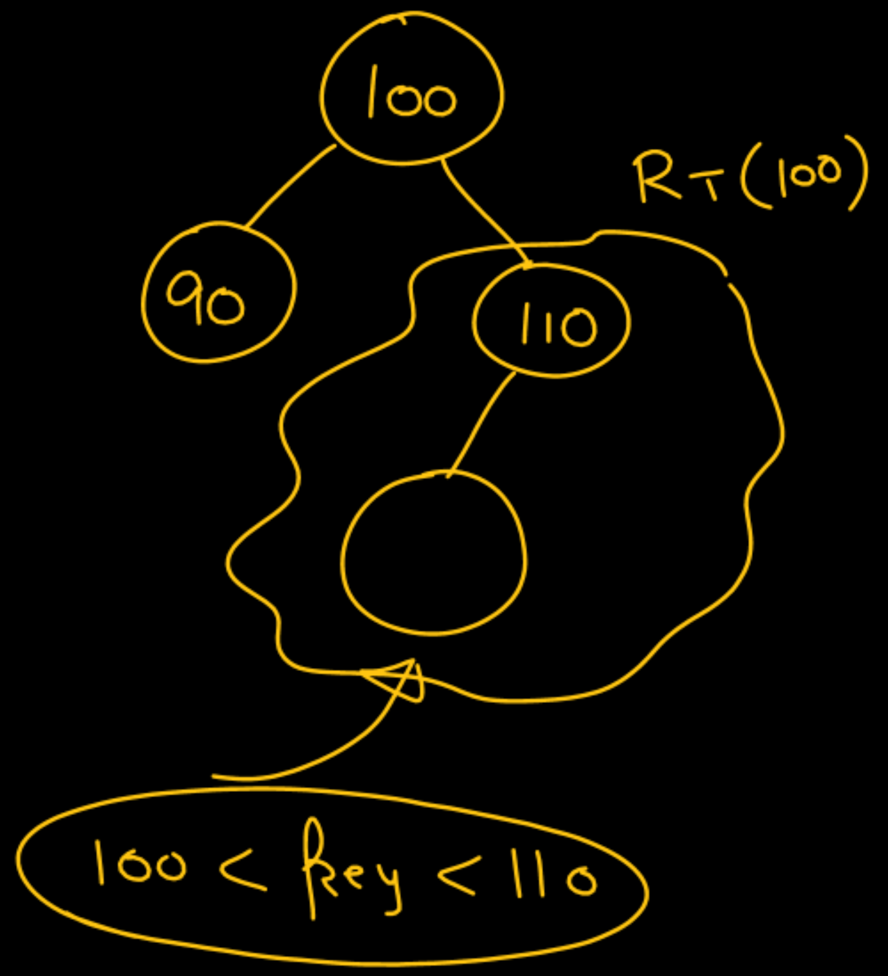
- All the keys in the left sub-tree of a node (x) are smaller than x.
- All the keys in the right subtree of node(x) are greater than x.





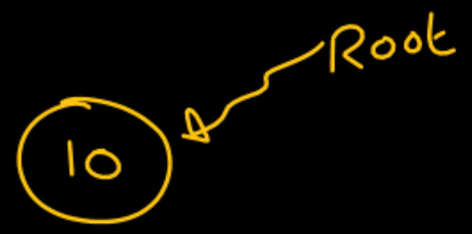
100
90
110
80
95
105
130
70
92
98
99
109
60
RT
All keys must be greater than 100


LT(100)
100
90
110
70
Any key > 90
smaller than
100

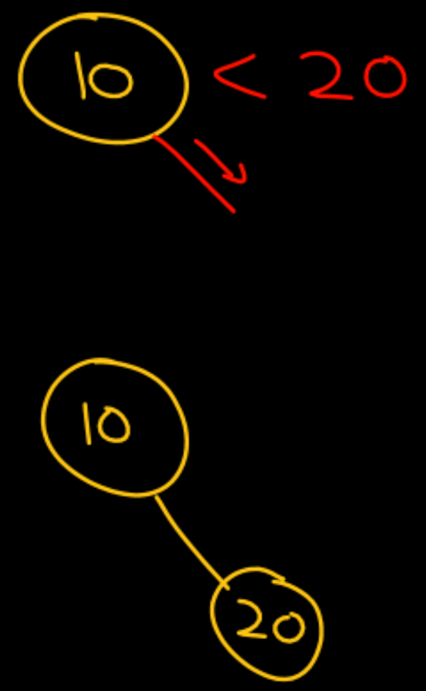

100
RT(100)
90
110
100 < key < 110

Construct a BST by inserting keys 10, 20, 30 in the same order.

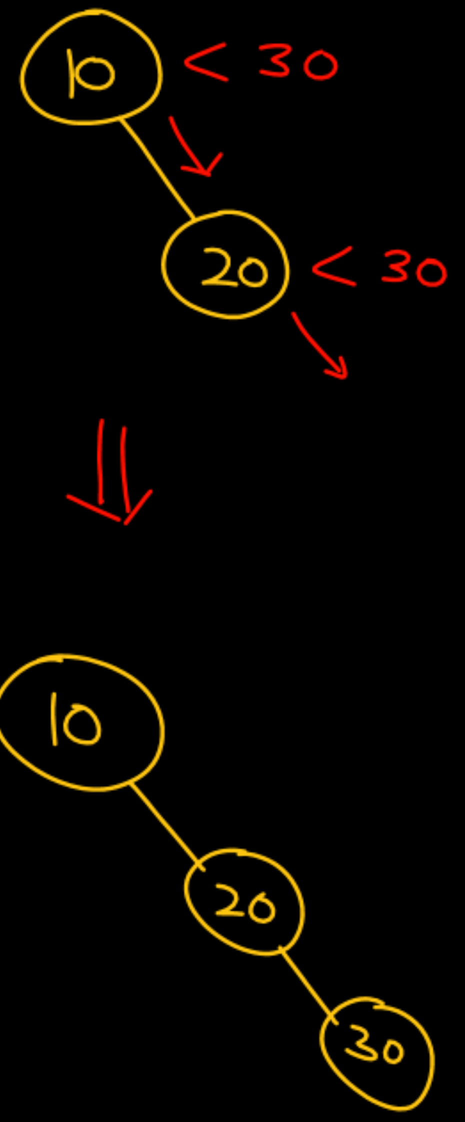
1) Insert 10



2) Insert 20

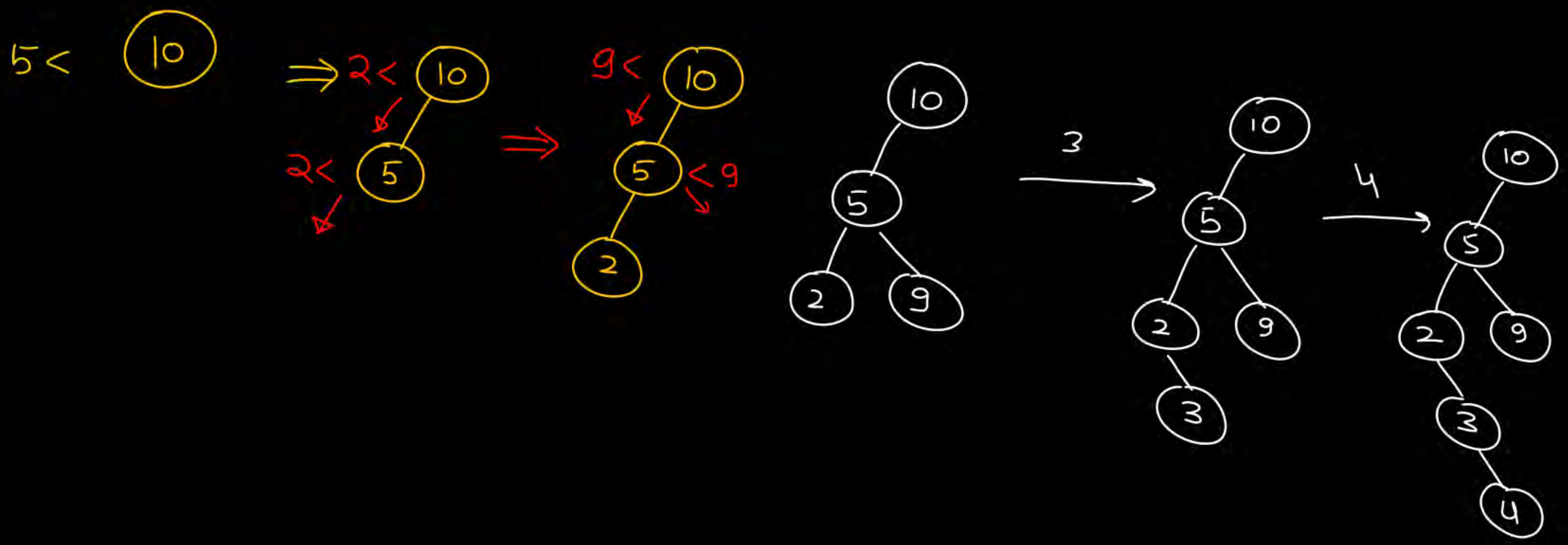


3) Insert 30



2. Construct BST by inserting following keys : 10, 5, 2, 9, 3, 4

in same order



No. of BST

Const a BST with keys 10, 20, 30 $\Longrightarrow$ 1
in same order

" " " " 10, 5, 2, 9, 3, 4 $\Rightarrow$ 1
in same order

Number of BSTs, when the insertion order of keys are given

$= 1$

Const. BST by inserting keys 10, 20, 30.

Insertion order can be

a) 10, 20, 30

b) 10, 30, 20

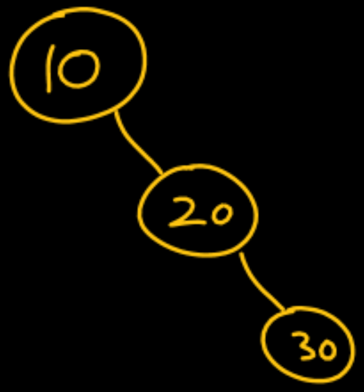
c) 20, 10, 30

d) 20, 30, 10

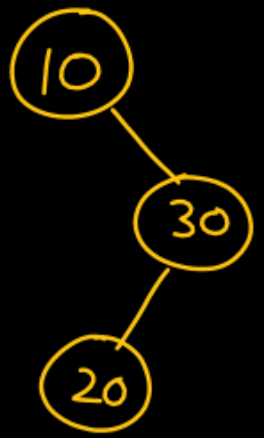
e) 30, 10, 20

f) 30, 20, 10

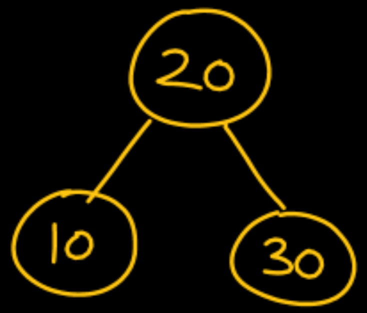
a)



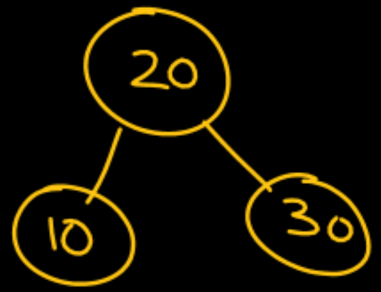
b)



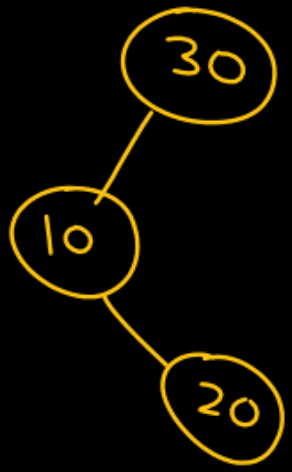
c)



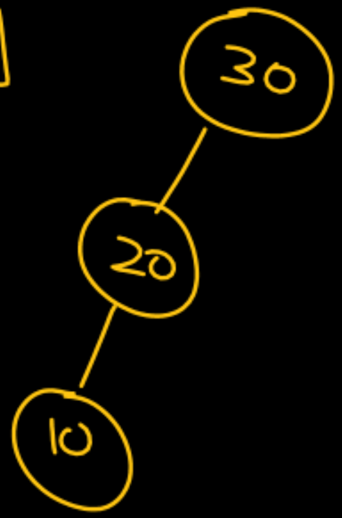
d)



e)



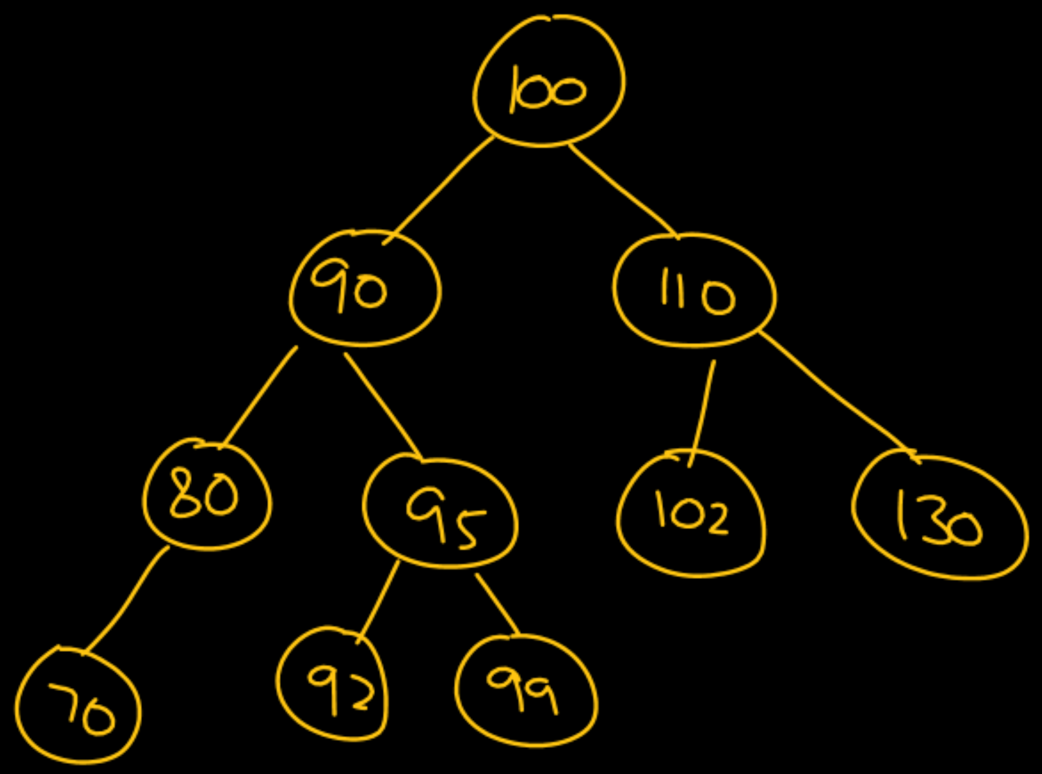
f)



$3 \text{ keys} \Longrightarrow \overset{BST}{5}$

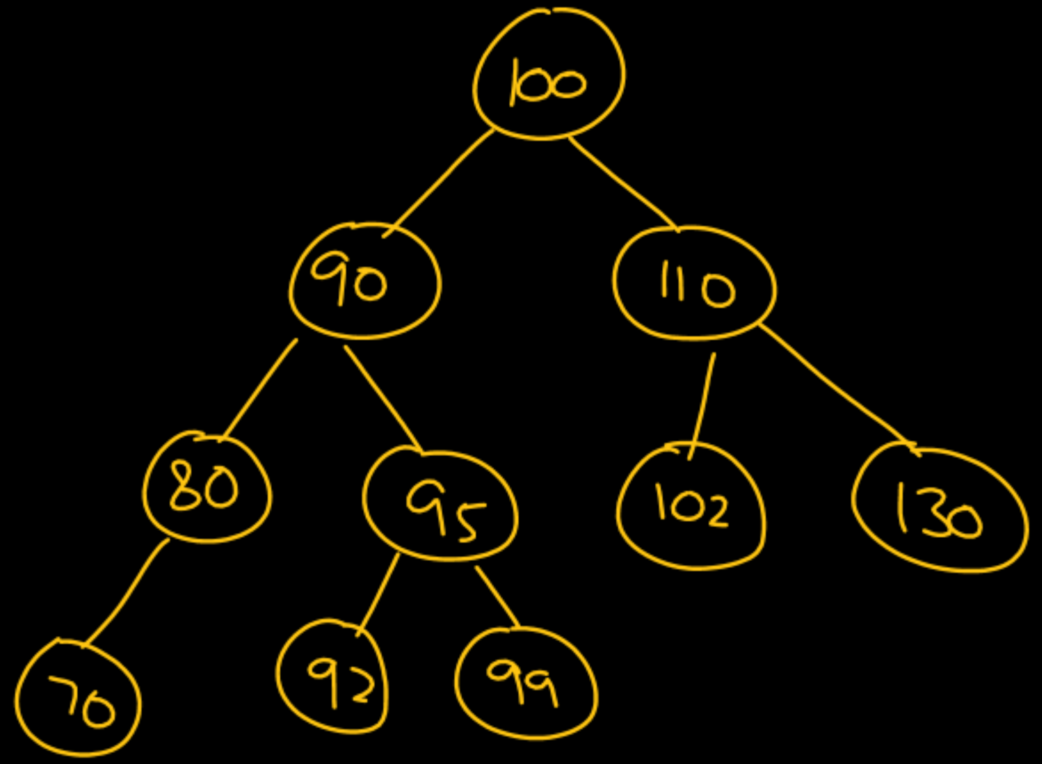
$$\boxed{\text{No. of BSTs with } n \text{ keys} = \frac{{}^{2n}C_n}{n+1}}$$

The inorder traversal of a BST is always increasing order of keys.



In : 70, 80, 90, 92, 95, 99,
100, 102, 110, 130

Pre : 100, 90, 80, 70, 95, 92, 99, 110, 102, 130



Pre : 100, 90, 80, 70, 95, 92, 99, 110, 102, 130

Q Given that the pre-order traversal of a BST is :

100, 90, 80, 70, 95, 92, 99, 110, 102, 130

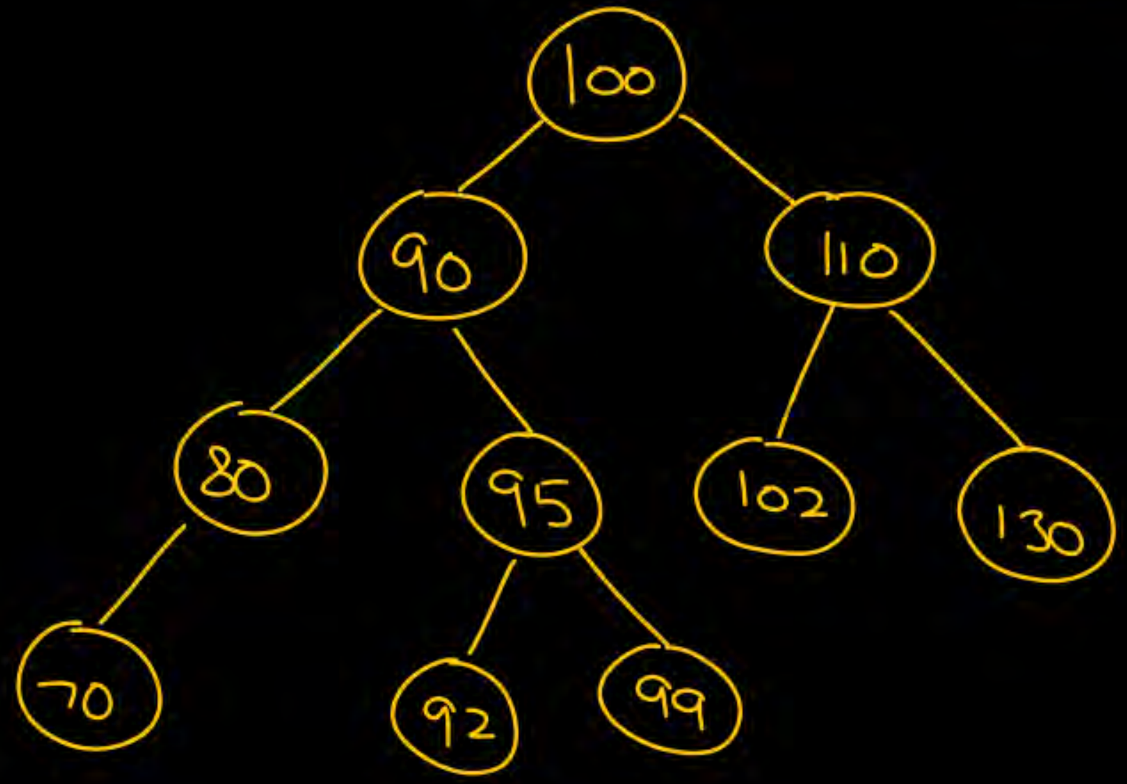
Find the postorder traversal.

In : 70 80 90 92 95 99 100 102 110 130

Pre : 100 90 80 70 95 92 99 110 102 130

In : 70 80 90 92 95 99 100 102 110 130

Pre : 100 90 80 70 95 92 99 110 102 130



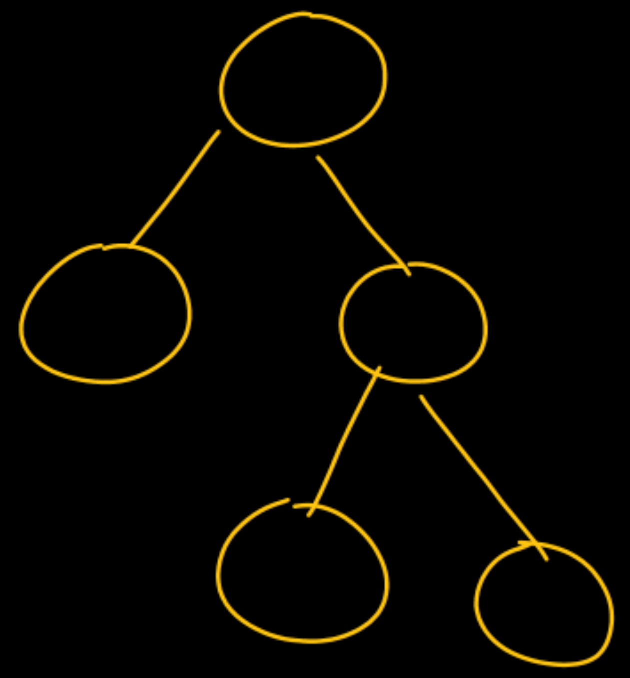
BST → inorder fix

find

Postorder

Pre :
Post : } → many trees

Given a binary tree structure with n nodes

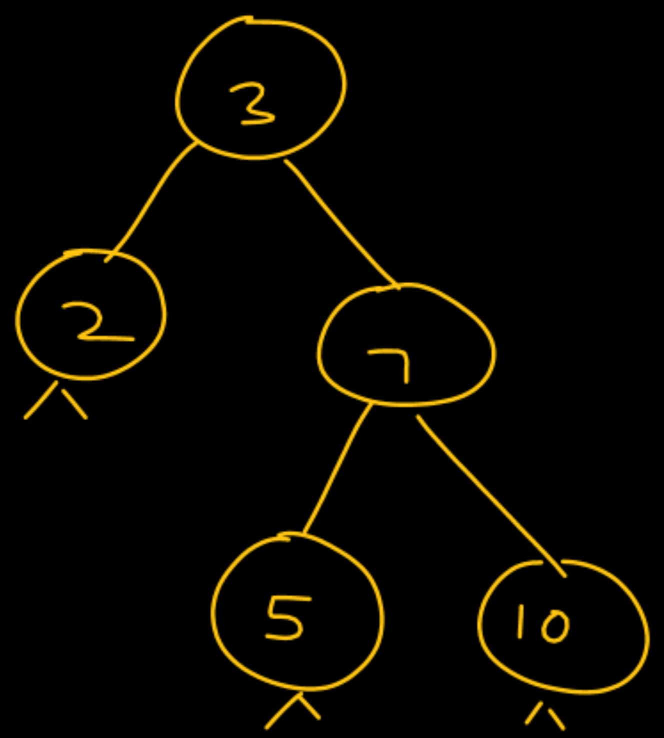


and also n keys are given

10, 2, 5, 7, 3

How many BSTs are possible

10, 2, 5, 7, 3



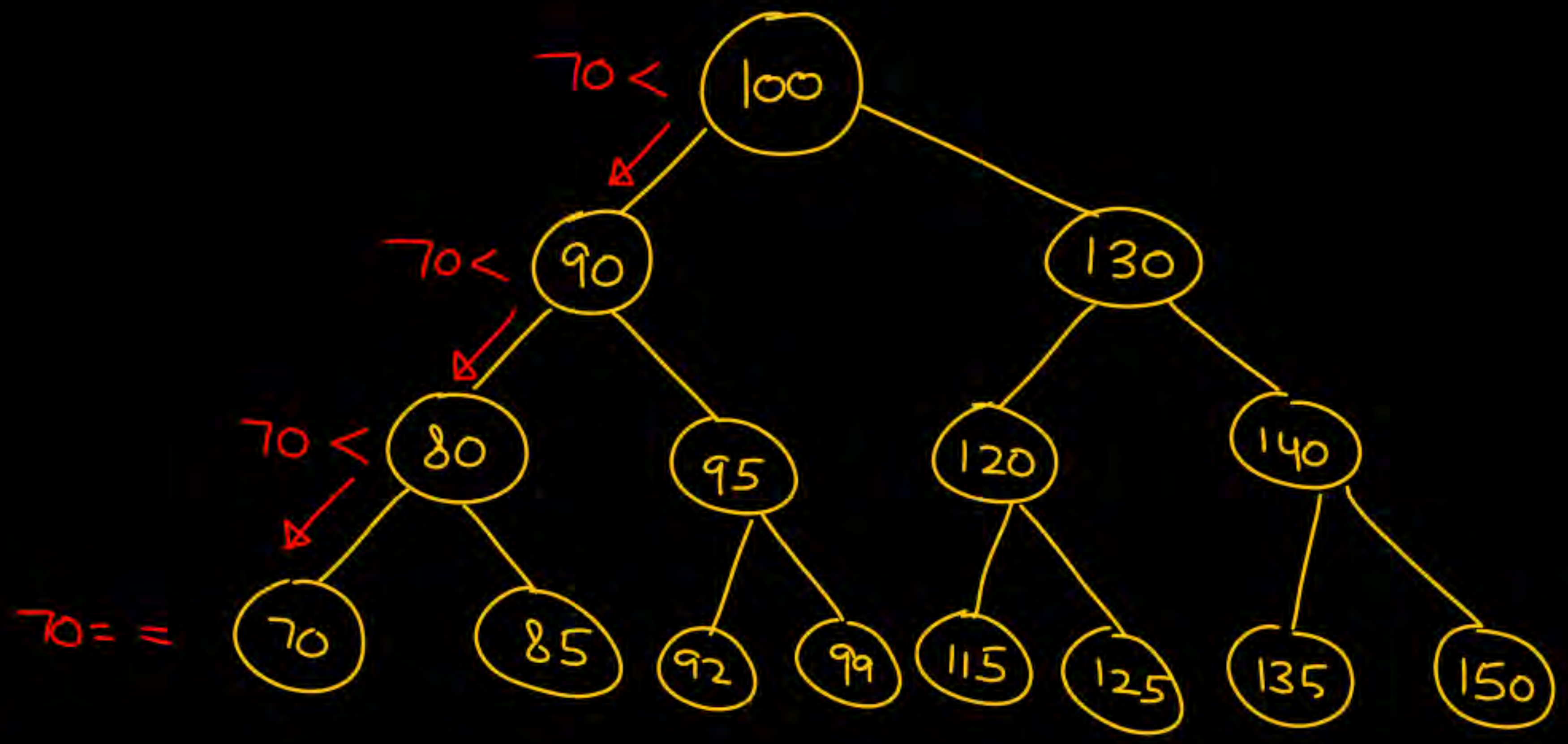
a	b	c	d	e
2	3	5	7	10

Search in a BST

Key=70

worst case

h=3



No. of comparisions = 4

No. of comparision = O(h)

h+1

Search in a BST

4 element
4 comparision

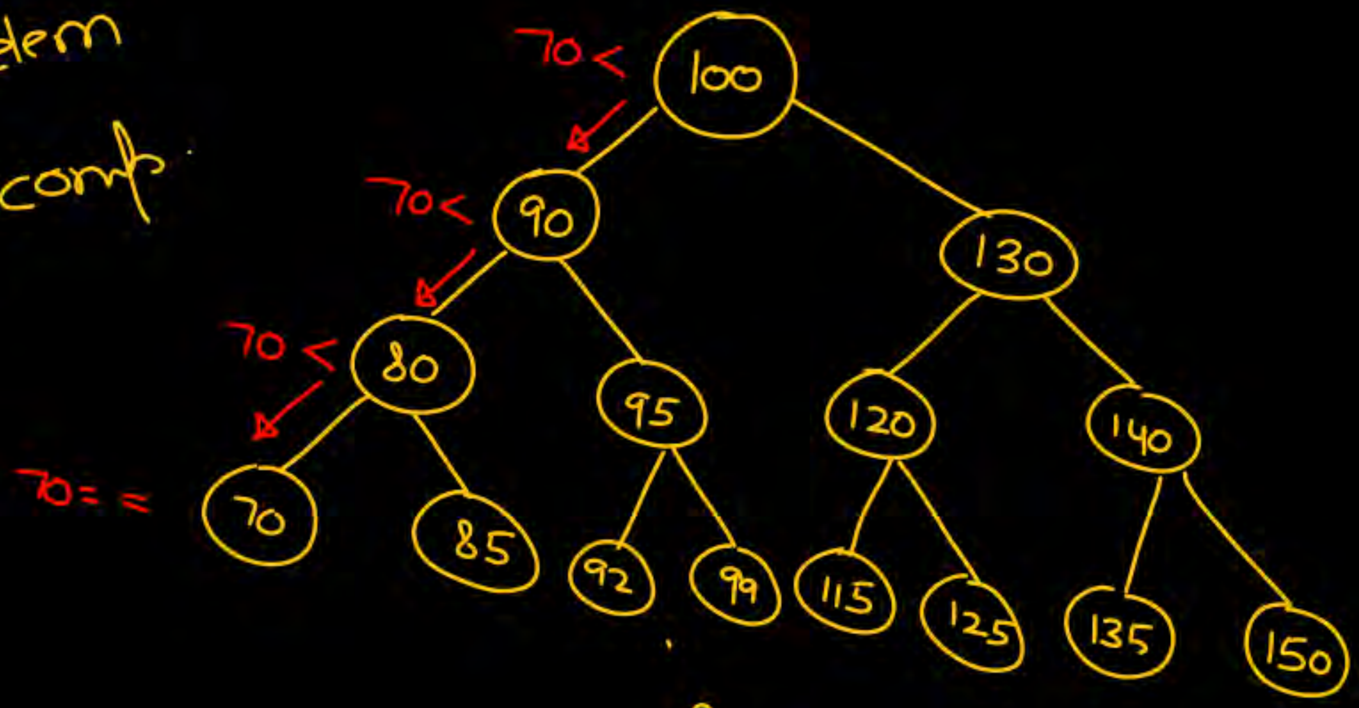
$O(n)$

Skewed tree

$\boxed{h = n-1}$

$O(n)$

15 elem
4 comp.



$h = O(\log_2 n)$

No. of comp $= h+1 = O(h)$

Deletion

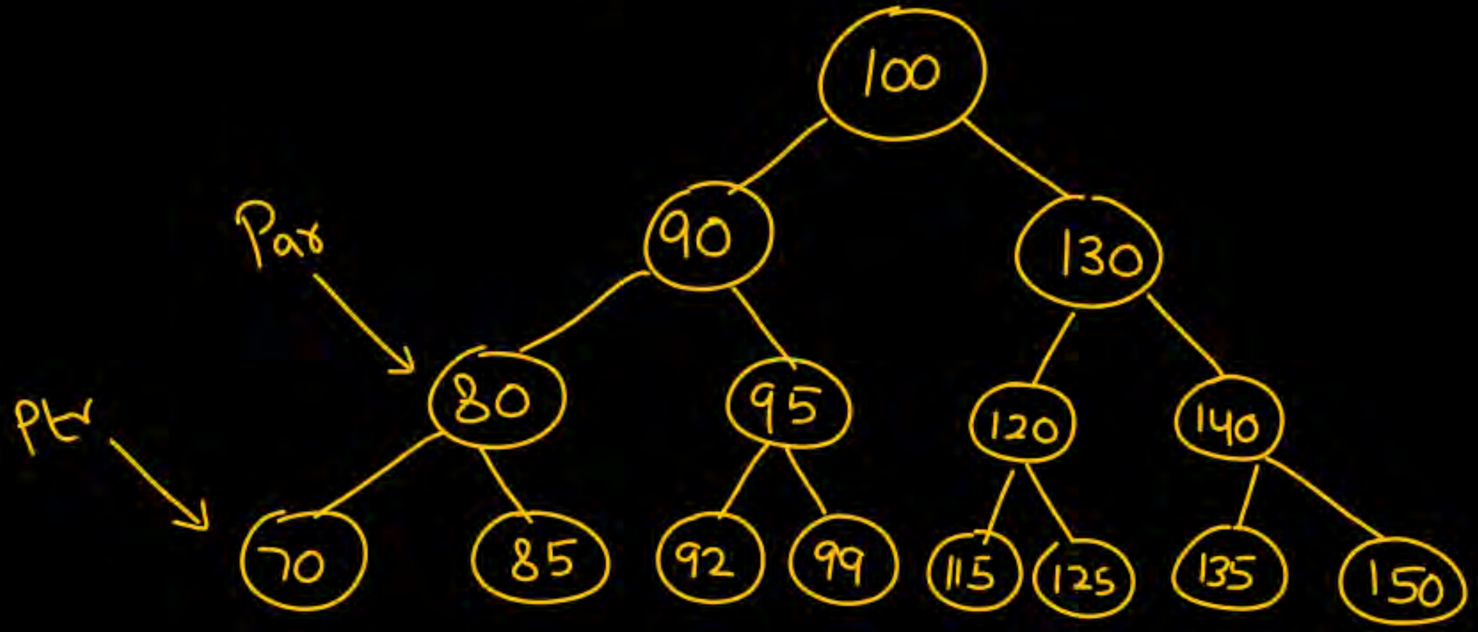
case-I : Deletion of a node having 0-child (leaf node)

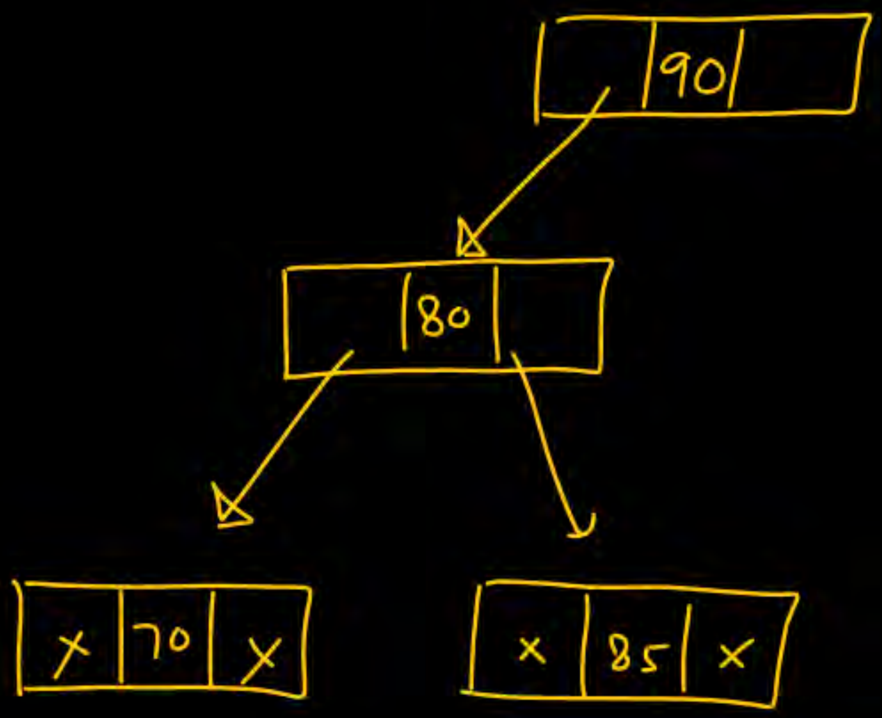
case-II Deletion of a node having 1-child

case III Deletion of a node having 2-child.

Key = 70

↳ search





We need to identify the parent pointer (of node to be deleted) which is pointing to node to be deleted. } make this pointer ⇒ NULL lect-2

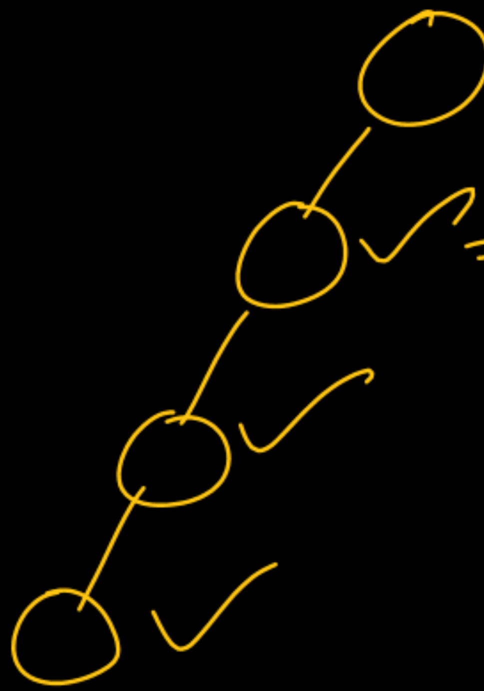
$$\boxed{n = h+1}$$

$$n_{min} = h+1$$

$$n_{max} = 2^{h+1}-1$$

$$\boxed{h = n-1}$$

X

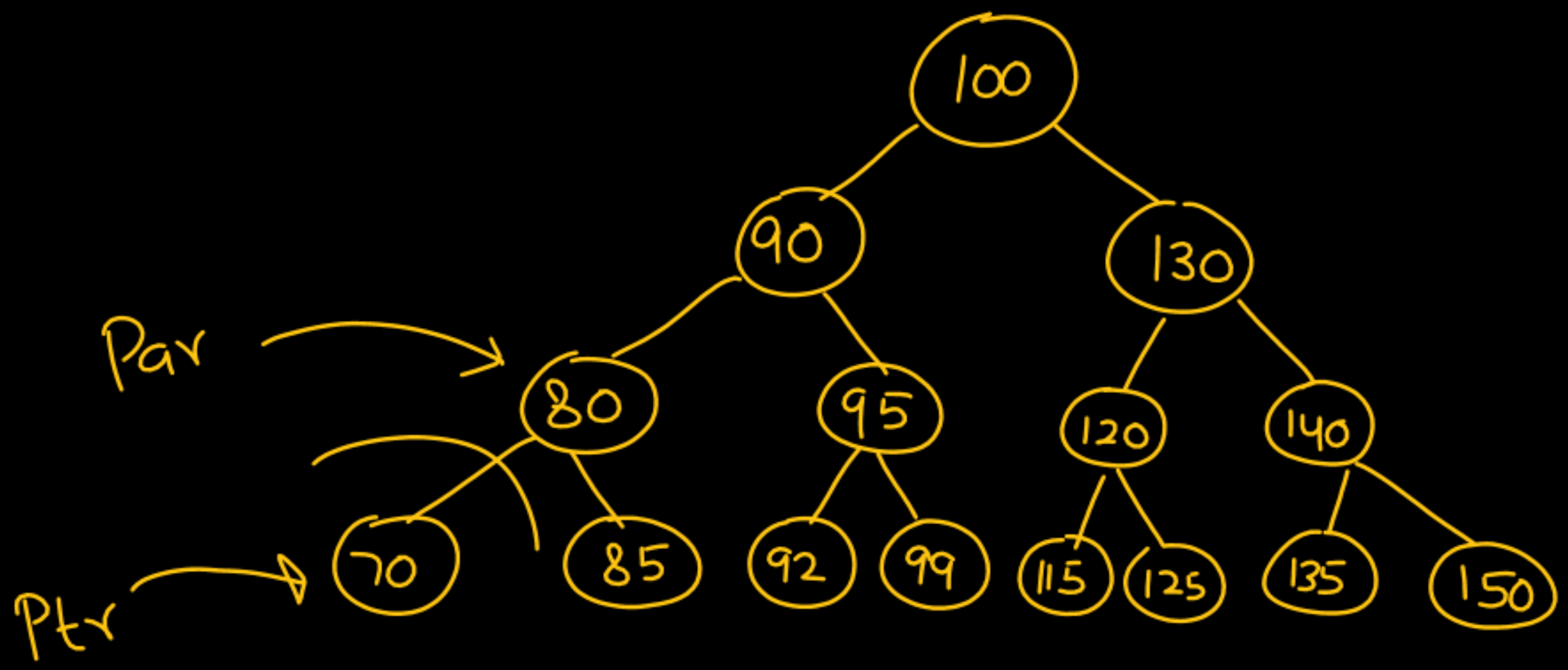


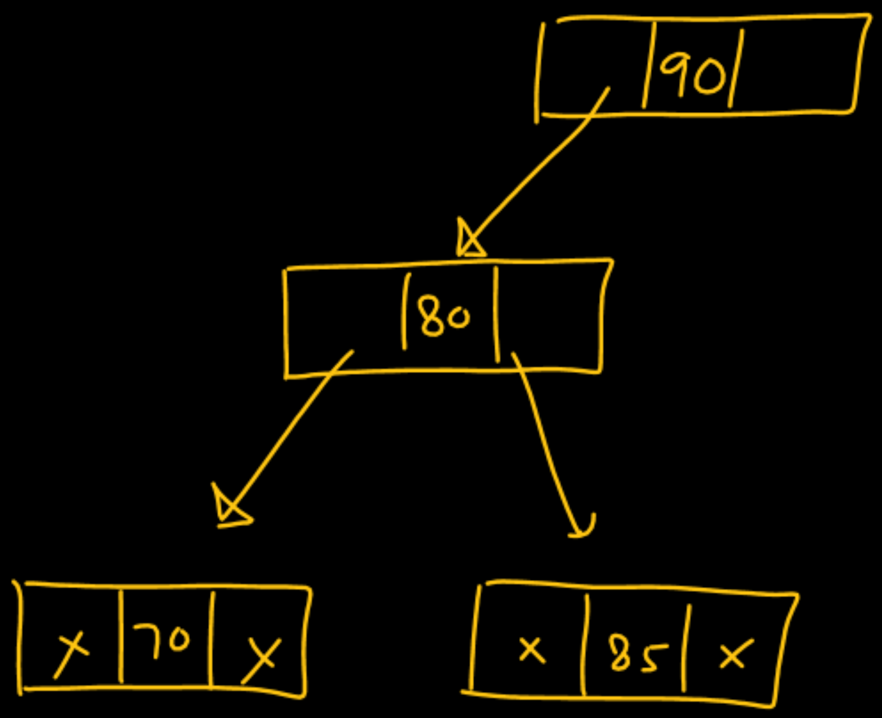
$\#$ of comp $= n$

$= O(n)$

Key = 70

→ search



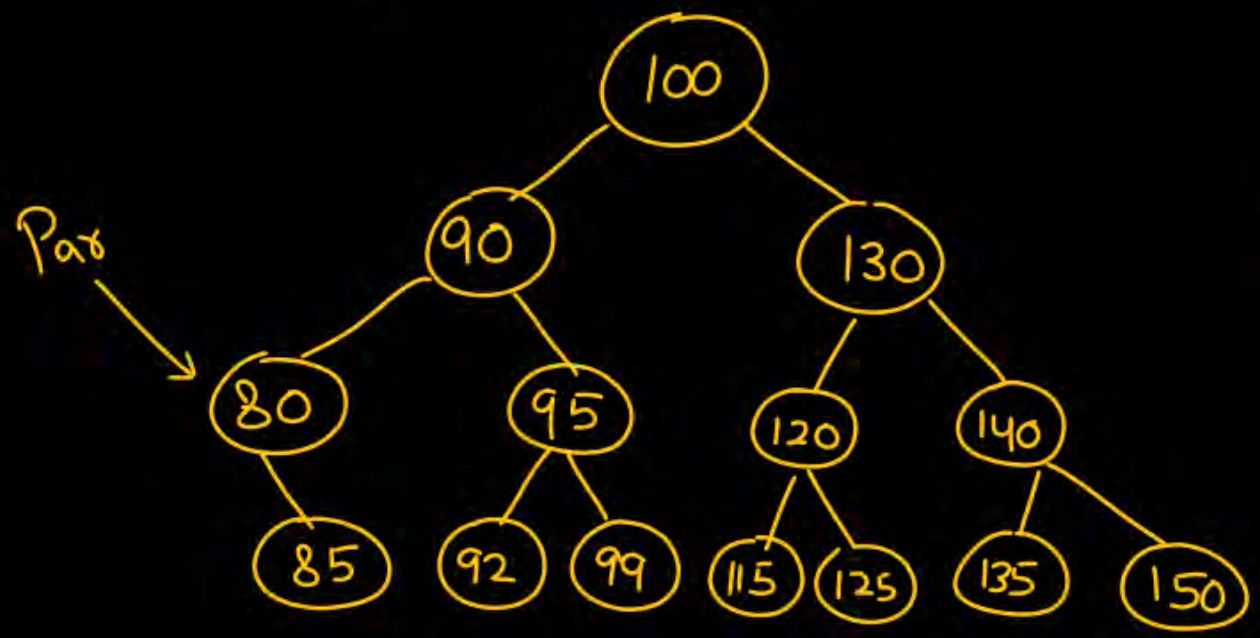


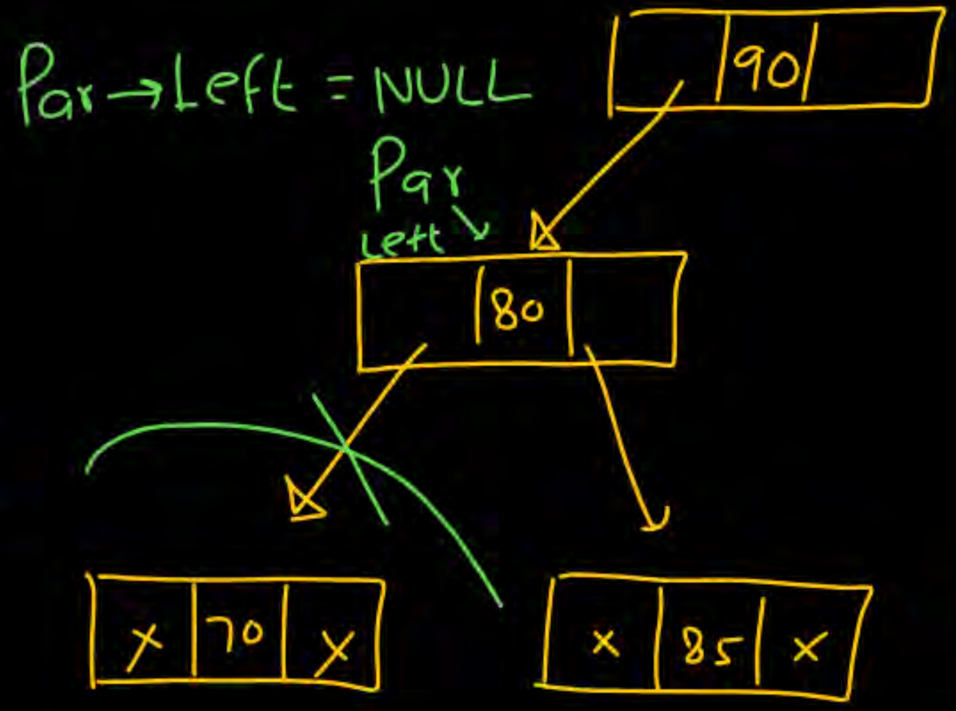

```
if ( Ptr→data < Par→data)
    {
        Par→left = NULL
        free(Ptr);
    }

else {
        Par→Right = NULL;
        free(Ptr);
    }
```

Key = 70

→ search





We need to identify the parent pointer (of node to be deleted) which is pointing to node to be deleted. } make this pointer ⇒ NULL

$$\left[\begin{array}{l}\text{1- child}\\ \text{2 -child}\end{array}\right]$$

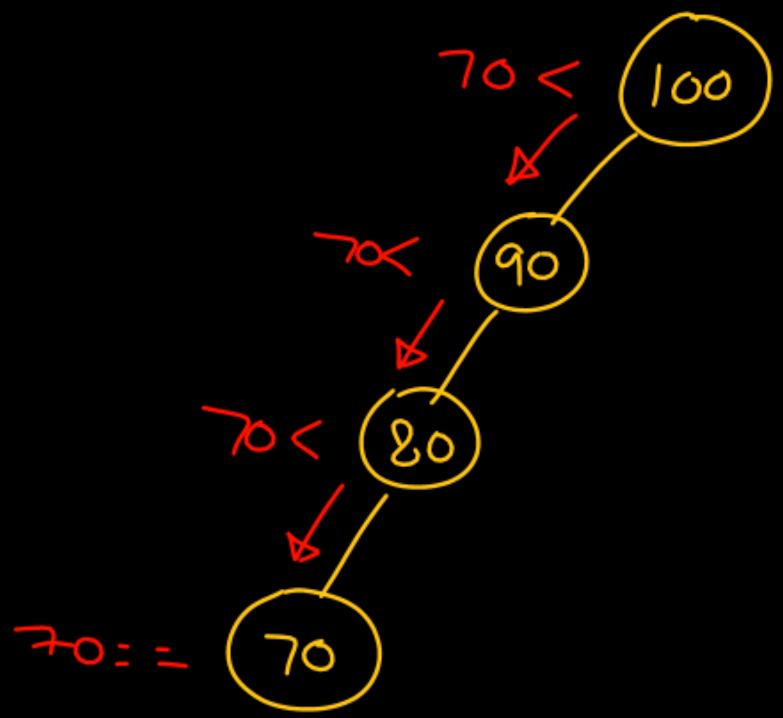
Key : 70

No. of comp = 4

$= O(h)$

$(h+1)$

$h = 3$



Keys $\longrightarrow$ Inorder ✓


THANK YOU GW
SOLDIERS !