

B.Tech.

Data Structure

Unit - 4 : Lec-1

CS / IT / CS Allied

By- Dr. Kapil Kumar
M.Tech.(CSE), Ph.D.(CSE)



Today's Target -

- What is Tree Data Structure?
- What is Binary Tree?
- Converting Algebraic Expression into Binary Tree
- Various Tree Terminology
- Binary Tree Variations : Full Binary Tree or Strict Binary Tree, Complete binary Tree (CBT),
Extended Binary Tree (EBT) or 2 - Tree, Binary Search Tree (BST) Binary sorted Tree
- Representation of Binary Tree in Memory – Linked list and Sequesntial
- Tree Traversal (Preorder, In order, Post order and Level by level Traversal of a Binary Tree)
- Binary Tree Creation using Traversals
- AKTU PYQs

Q.1 Construct a tree for the following preorder and post order and write its in order traversal.

Preorder : 24, 14, 13, 19, 17, 15, 10, 5, 8, 6, 7, 20

Post order : 13, 15, 17, 10, 19, 14, 7, 6, 20, 8, 5, 24

2014-15, 10 marks

Q.2 Following are the in order and post order traversal of a binary tree T-

(a) DKIBAEGHJFC ✓

(b) KDI EAGBFCJH ✓

Construct the tree T.

2014-15 , 5 marks

Q.3 For tree construction which is the suitable and efficient data structure and why?

2015-16, 2 marks

Q.4 Draw a binary tree which following traversal:

Inorder : D B H E A I F J C G ✓

Preorder : A B D E H C F I J G ✓

Q.5 Define complete binary tree. Give example.

Q.6 Write a program for various traversing techniques of binary tree with neat example.

Q.7 Define the following :- (i) Tree (ii) Level of a node (iii) Height of a tree

Q.8 Draw a binary tree which has the following traversal-

Inorder : D J G B A E H C F I ✓

Preorder: A B D G J C E H F I ✓

2015-16, 5 marks

2016-17, 2 marks

2016-17, 10 marks

2016-17, 2 marks

2016-17, 7 marks

Q.9 Define tree, binary tree, complete binary tree and full binary tree. Write algorithms or function to obtain traversals of binary tree in preorder, postorder and inorder.

2017-18, 7 marks

Q.10 Draw a binary tree with following traversals:

Inorder : B C A E G D H F I J

Preorder : A B C D E G F H I J

Q.11 Construct a binary tree for the following preorder and inorder traversa. Explain with a neat diagram:

Preorder: A B D I E H J C F K L G M

Inorder: D I B H J E A F L K C G M

Q.12 Construct an expression tree for the expression

Give pre order inorder and post order traversals of the expression tree so formed.

2017-18, 7 marks

2017-18, 7 Marks

2017-18, 7 marks

Q.13 If the in order traversal of a binary tree is D, J, G, B, A, E, H, C, F, I and its pre order traversal is A, B, D, G, J, C, E, H, F, I Determine the binary tree?

2018-19, 2 marks

Q.14 Define complete binary tree and full binary tree.

2018-19, 2 marks

Q.15 Draw a binary tree for the following traversals:

Preorder : A B C D E F G H I J K L

Postorder : C F E G D B K J L I H A

Q.16 Draw a binary tree for expression : $A * B - (C + D) * (P / Q)$

2018-19, 7 marks

Q.17 Number of nodes in a complete tree is 100000. find its depth.

2018-19, 2 MARKS

Q.18 Construct the binary tree for the following:

Inorder : Q, B, K, C, F, A, G, P, D, E, R, H

Preorder: G, B, Q, A, C, K, F, P, D, E, R, H

Find the post order of the tree.

2018-19, 2 MARKS

2018-19, 7 MARKS

Q.19 Define extended binary tree, full binary tree, strictly binary tree and complete binary tree.

2019-20, 2 marks

Q.20 Can you find a unique tree when any two traversals are given? Using the following traversals construct the corresponding binary tree:

INORDER: H K D B I L E A F C M J G

PREORDER: A B D H K E I L C F G J M

Also find the postorder traversal of obtained tree.

2019-20, 10 marks

Q.21 If the inorder of a binary tree is B, I, D, A, C, G, E, H, F and its post order is I, D, B, G, C, H, F, E, A then draw a corresponding binary tree with neat and clear steps from above assumption. **2020-21, 10 marks**

Q.22 Write short note of the following

- (a) Extended binary Tree
- (b) Complete Binary Tree
- (c) Threaded binary Tree

2020-21, 10 marks

Q.23. Define complete binary tree, strictly binary tree and extended binary tree.

2020-21, 2 marks

Q.24. Define Binary tree. In order and post order traversal of a tree are given below:

Preorder: H D I B J E K A F C G

Postorder: A B D H I E J K C F G

Construct the binary tree and determine the postorder traversal of the tree.

2020-21, 7 marks

Q.25. In a complete binary tree if the number of nodes is 1000000. What will be the height of complete binary tree.

2022-23, 2 marks

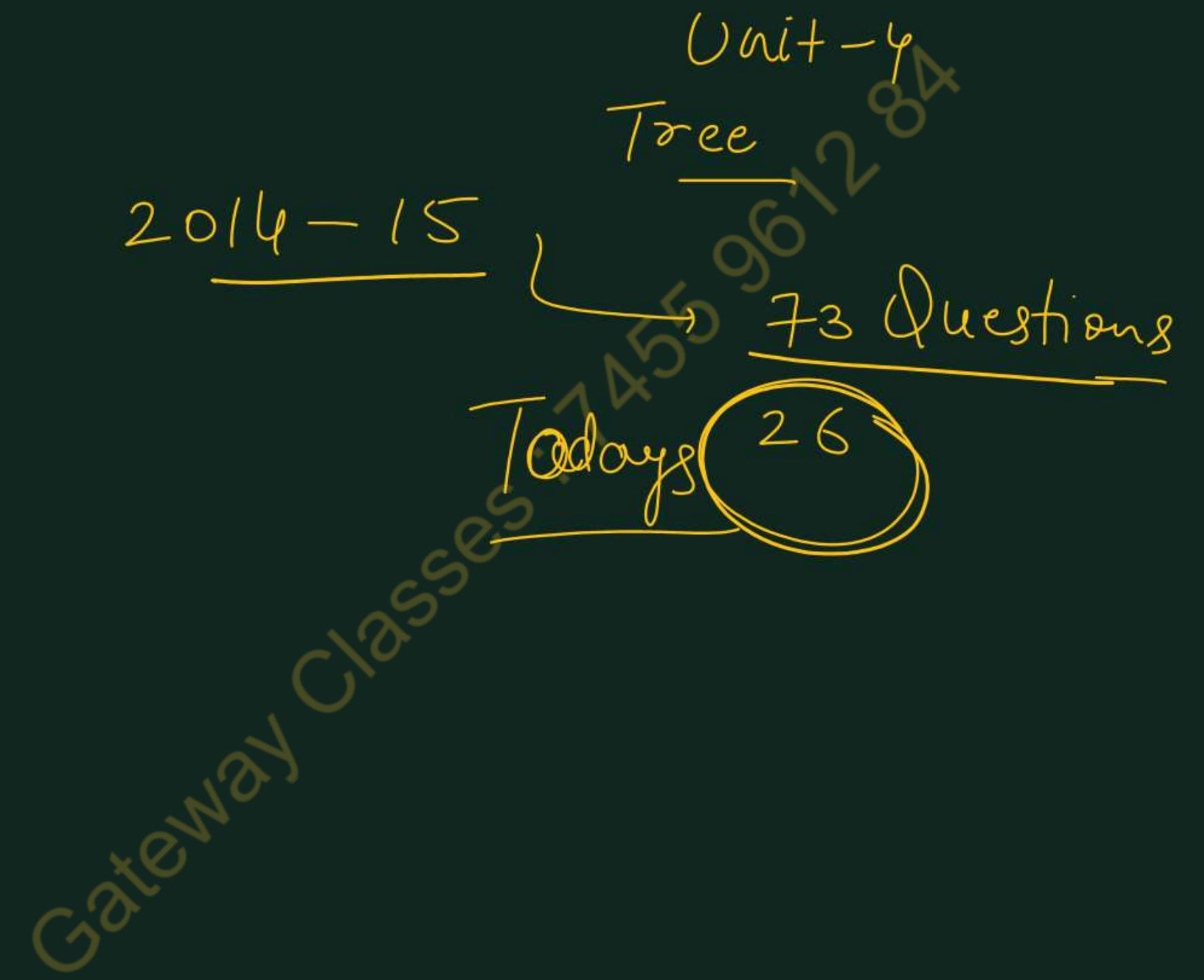
Q.26. The order of nodes of a binary tree in inorder and postorder traversals are as follows:

Inorder : B I D A C G E H F

Post order: I D B G C H F E A

- Draw the corresponding binary tree.**
- Write the pre order traversal of the same tree.**

2022-23, 10 marks



Tree Data Structure

- So far, we have been studying mainly linear types of data structures : Strings, Arrays, Linked lists, Stacks and Queues.
- Now we will discuss about a nonlinear data structure called a **tree**.
- This data structure is mainly used to represent data containing a hierarchical relationship between elements, e.g., records, family trees and tables of contents.
- Tree is a data structure which allows us to associate a parent – child relationship between various pieces of data and allows us to arrange our records, data and files in a hierarchical fashion.
- Operating systems uses tree data structure to manage our files using **hierarchical file system**.
- First we investigate a special kind of tree, called a **binary tree**, which can be easily maintained in the computer.

OS (DOS)
Windows

Root Directory

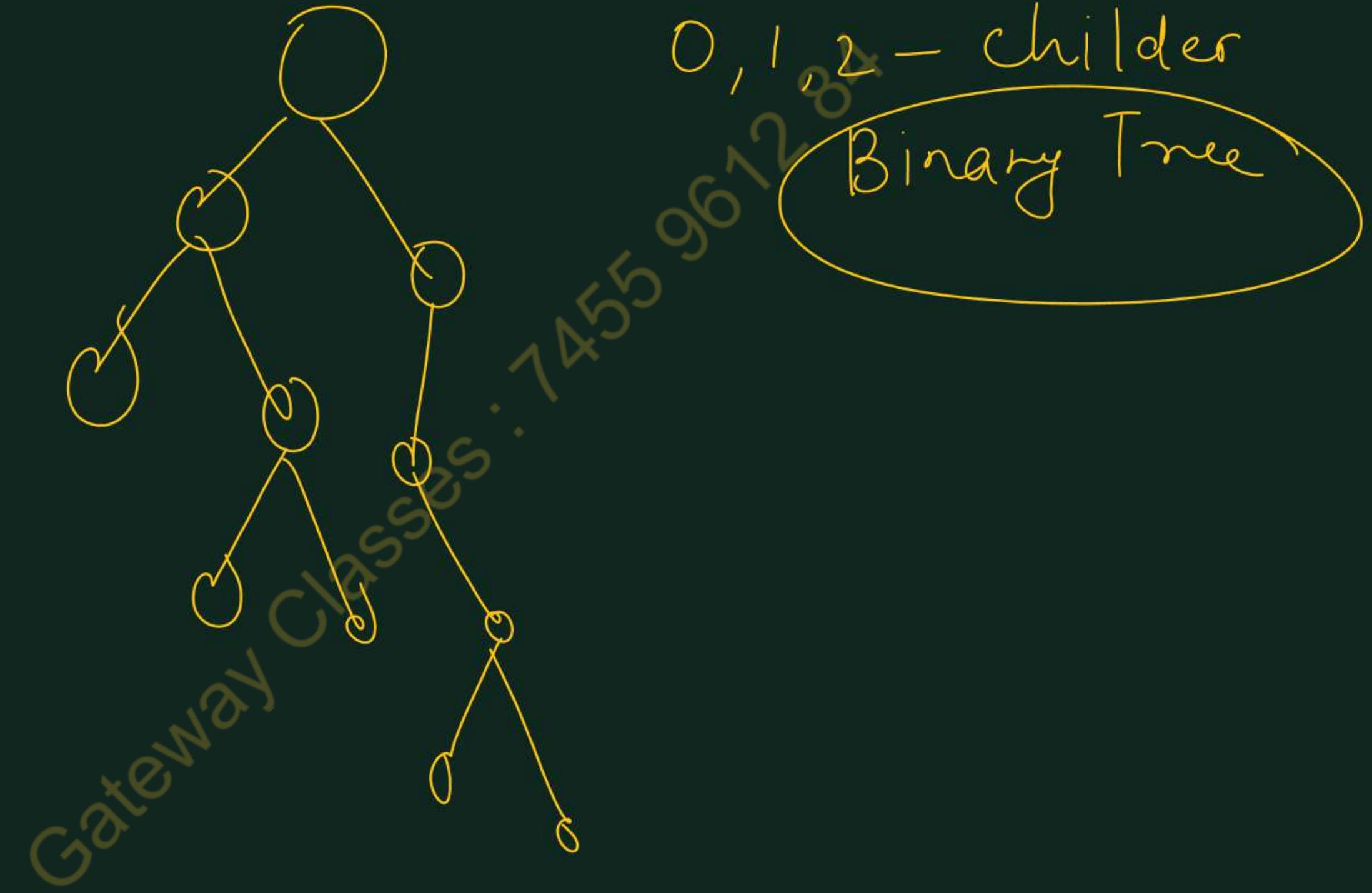


General Tree

n number
of children



Gateway Classes: 7455 9612 84



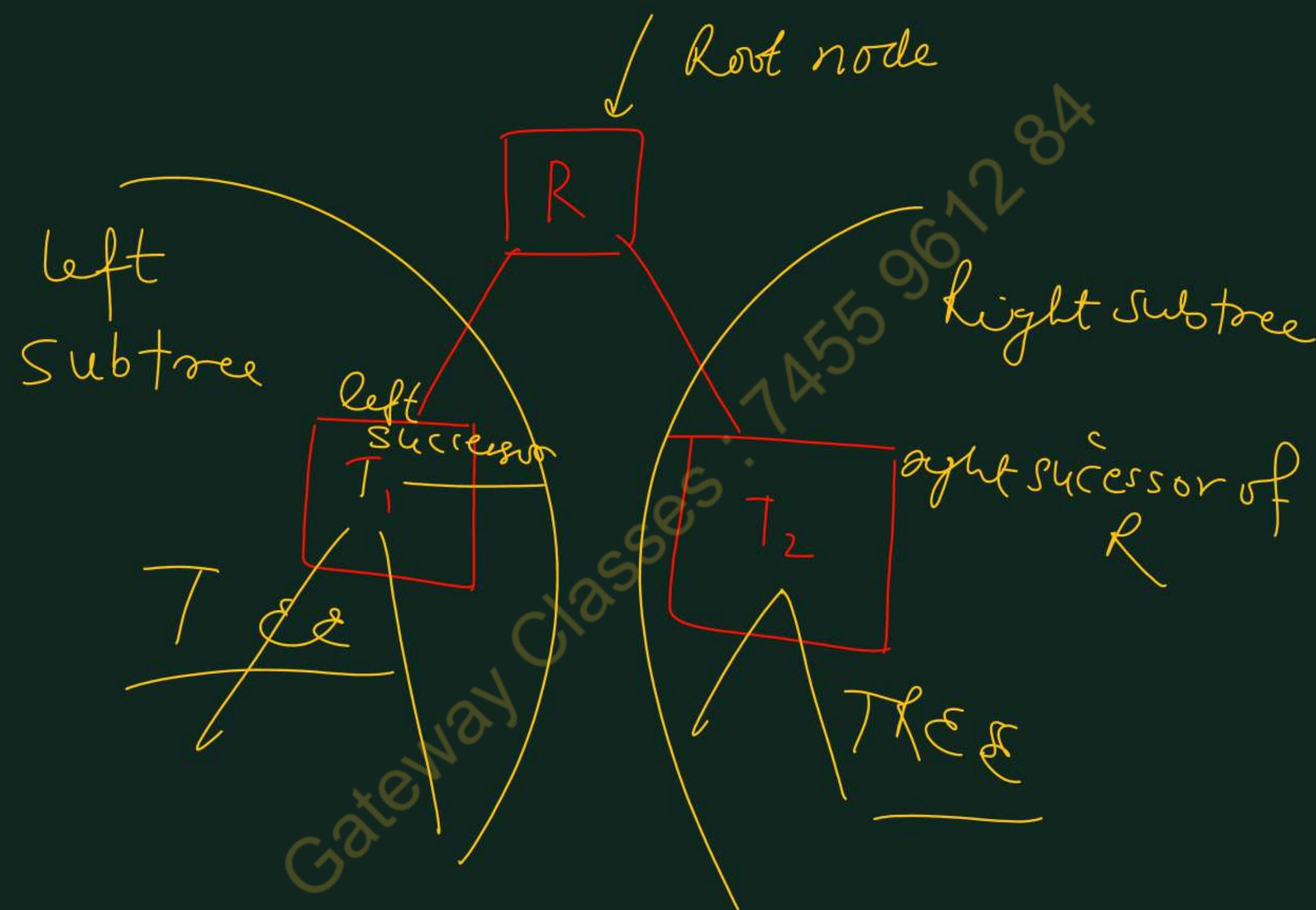
0, 1
2⁴ - children

Binary Tree

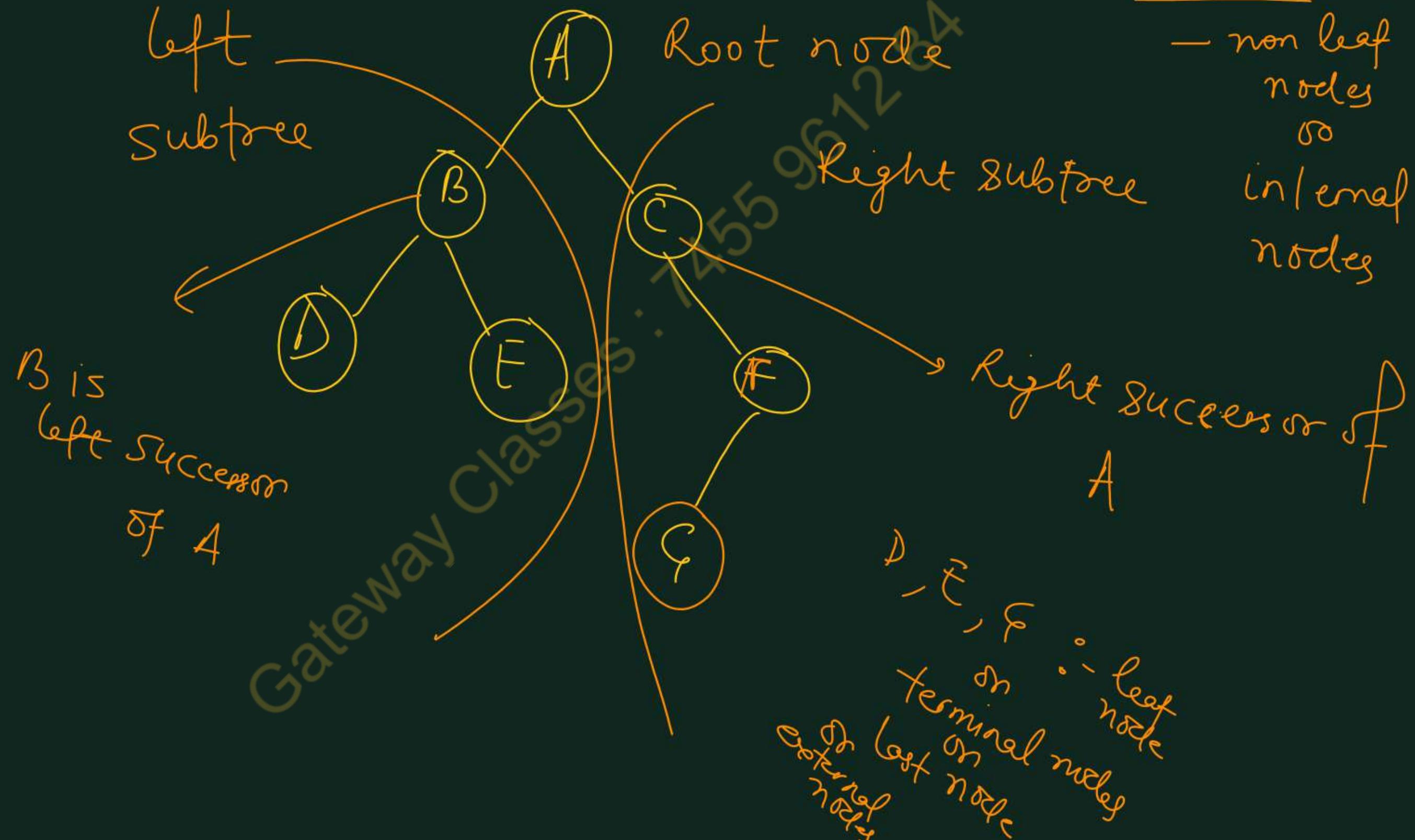
Gateway Classes : 7455 9612 84

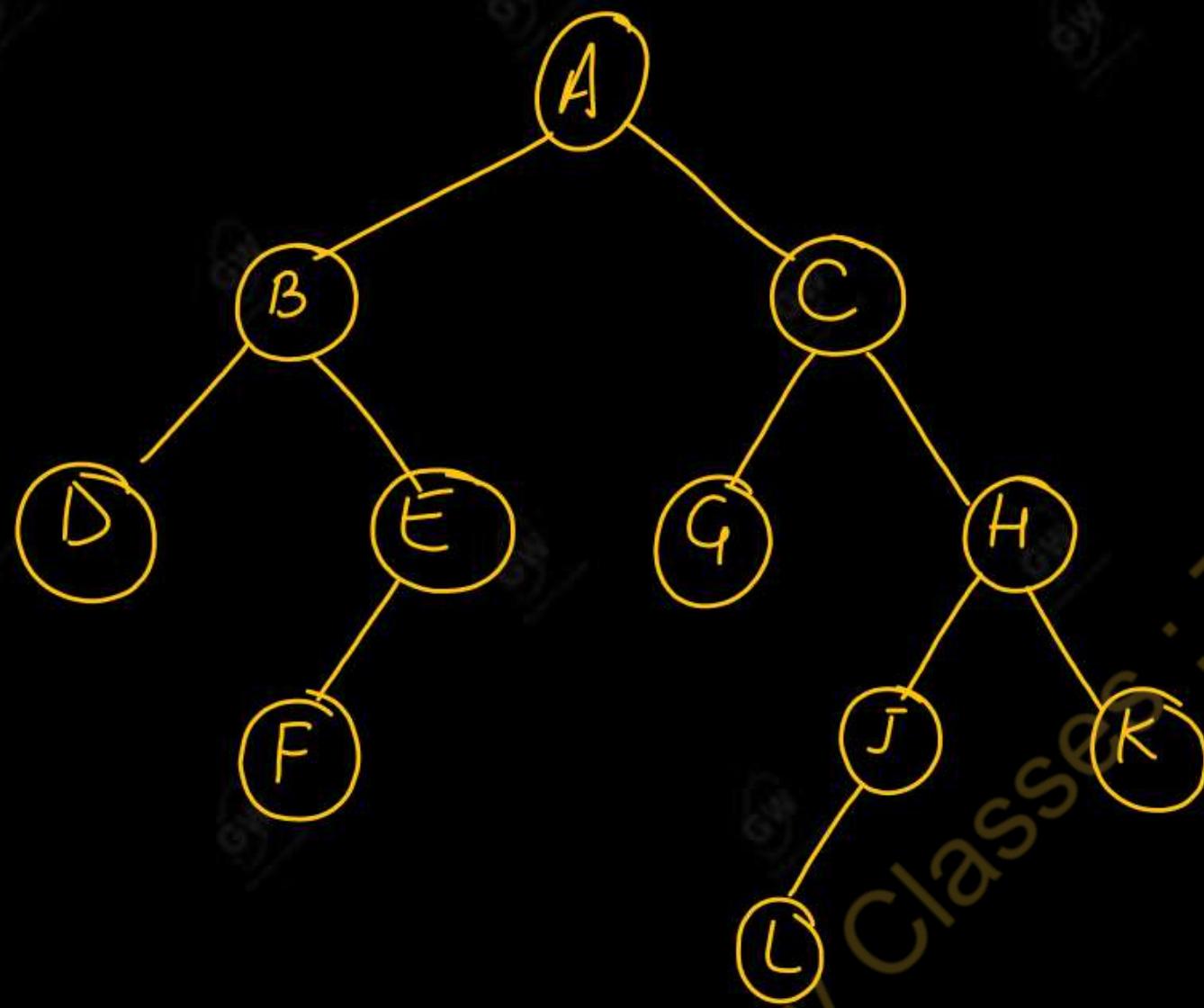
- It is a finite set of elements and each node of binary tree can have at most 2 children (branches).
- In other words, A binary tree T is defined as a finite set of elements, called nodes, such that:
 - (a) T is empty (called the null tree or *empty tree*), or
 - (b) T contains a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary trees T_1 and T_2 .
- If T does contain a root R, then the two trees T_1 and T_2 are called, respectively, the left and right subtrees of R.
- If T_1 is nonempty, then its root is called the left successor of R; similarly, if T_2 is nonempty, then its root is called the right successor of R.
- A binary tree T is frequently presented by means of a diagram.





A, B, C, F
— non leaf nodes
○ internal nodes



Consider the following Binary Tree –

- T consists of 11 nodes, represented by the letters A through L, excluding I.
- The root of T is the node A at the top of the diagram.
- A left-downward slanted line from a node N indicates a left successor of N, and a right-downward slanted line from N indicates a right successor of N.

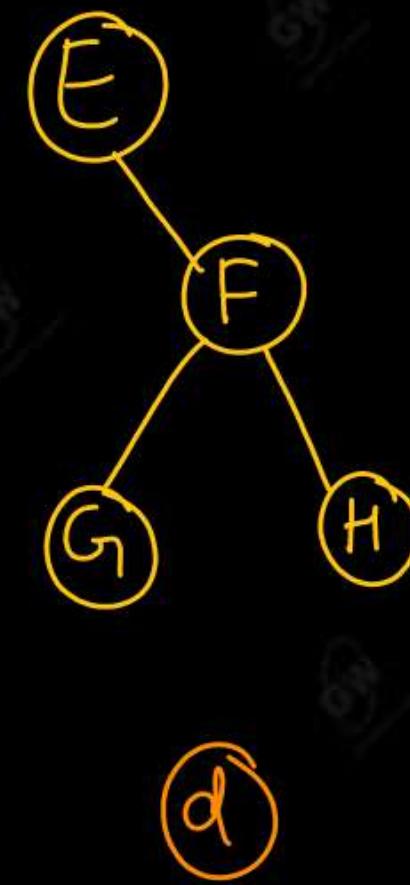
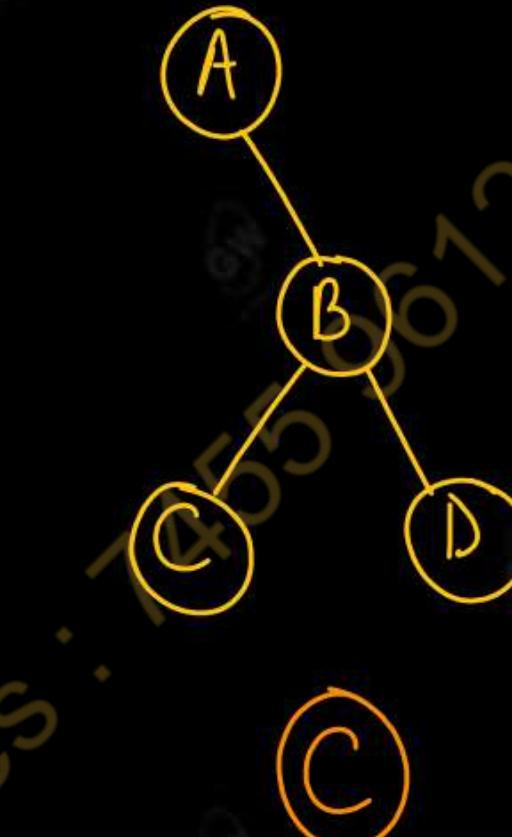
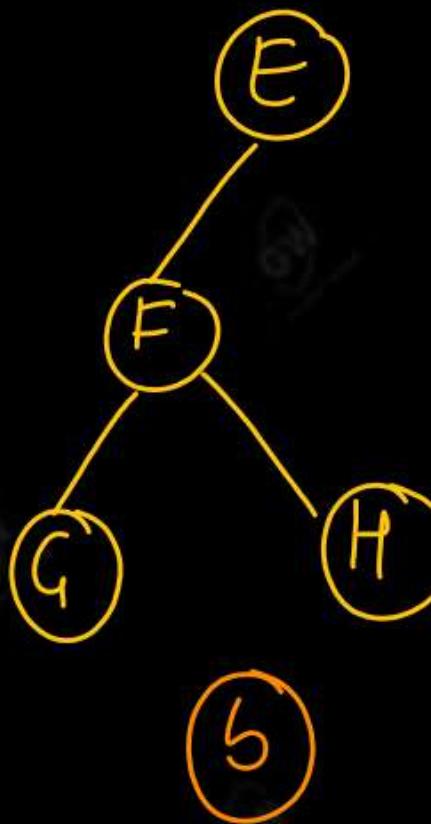
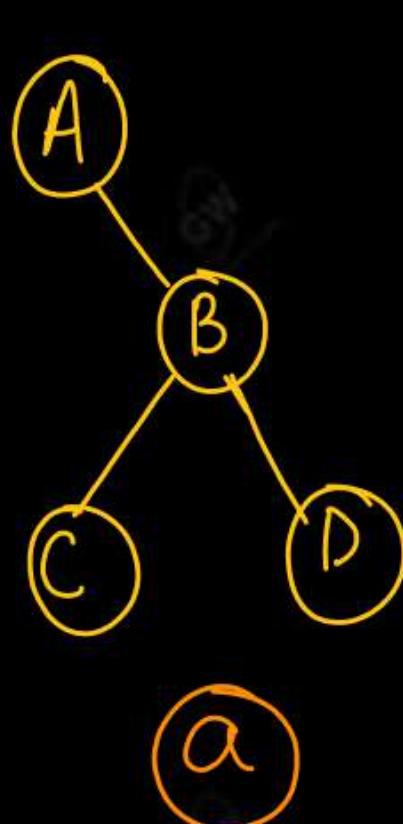
A = Root node
B onwards – left subtree (A)
C onwards – right subtree (A)
B = left successor of A
C = right successor of A
D F G L K = leaf nodes
A B E C H J = internal nodes
or
non leaf nodes

- Observe that:
 - (a) B is a left successor and C is a right successor of the node A.
 - (b) The left sub tree of the root A consists of the nodes B, D, E and F, and the right subtree of A consists of the nodes C, G, H, J, K and L.
- Any node N in a binary tree T has either 0, 1 or 2 successors.
- The nodes A, B, C and H have two successors, the nodes E and J have only one successor, and the nodes D, F, G, L and K have no successors.
- The nodes with no successors are called terminal nodes.

leaf | External | leaf

- The above definition of the binary tree T is recursive since T is defined in terms of the binary sub trees T_1 and T_2 .
- This means, in particular, that every node N of T contains a left and a right subtree.
- Moreover, if N is a terminal node, then both its left and right sub trees are empty.
- Binary trees T and T' are said to be *similar* if they have the same structure or, in other words, if they have the same shape.
- The trees are said to be *copies* if they are similar and if they have the same contents at corresponding nodes.

□ EXAMPLE : Consider the following binary trees:-



- The three trees (a), (c) and (d) are similar.
- The trees (a) and (c) are copies since they also have the same data at corresponding nodes.
- The tree (b) is neither similar nor a copy of the tree (d) because, in a binary tree, we distinguish between a left successor and a right successor even when there is only one successor.

Topic

Converting Algebraic Expressions into binary Tree

- Consider any algebraic expression E involving only binary operations, such as

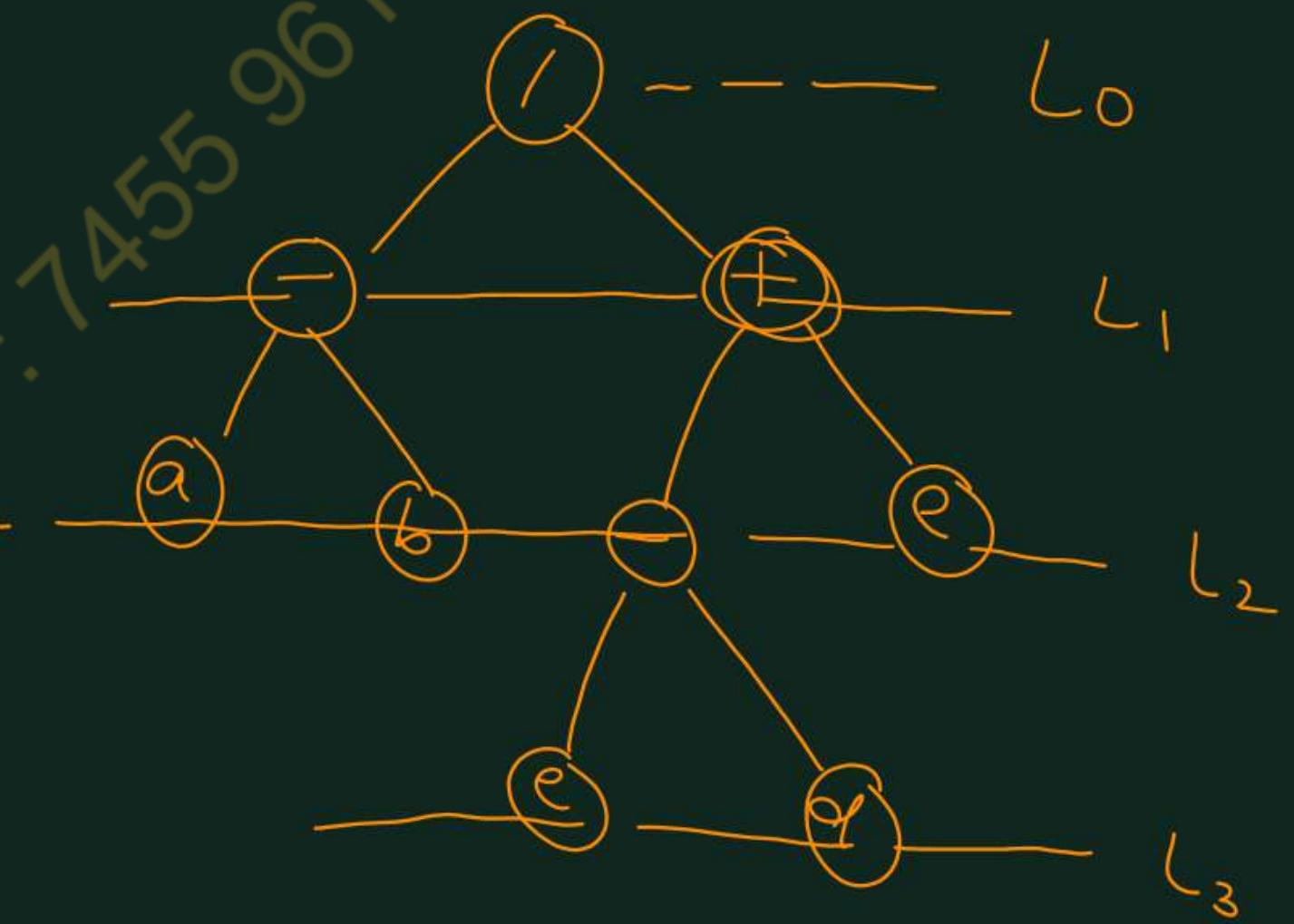
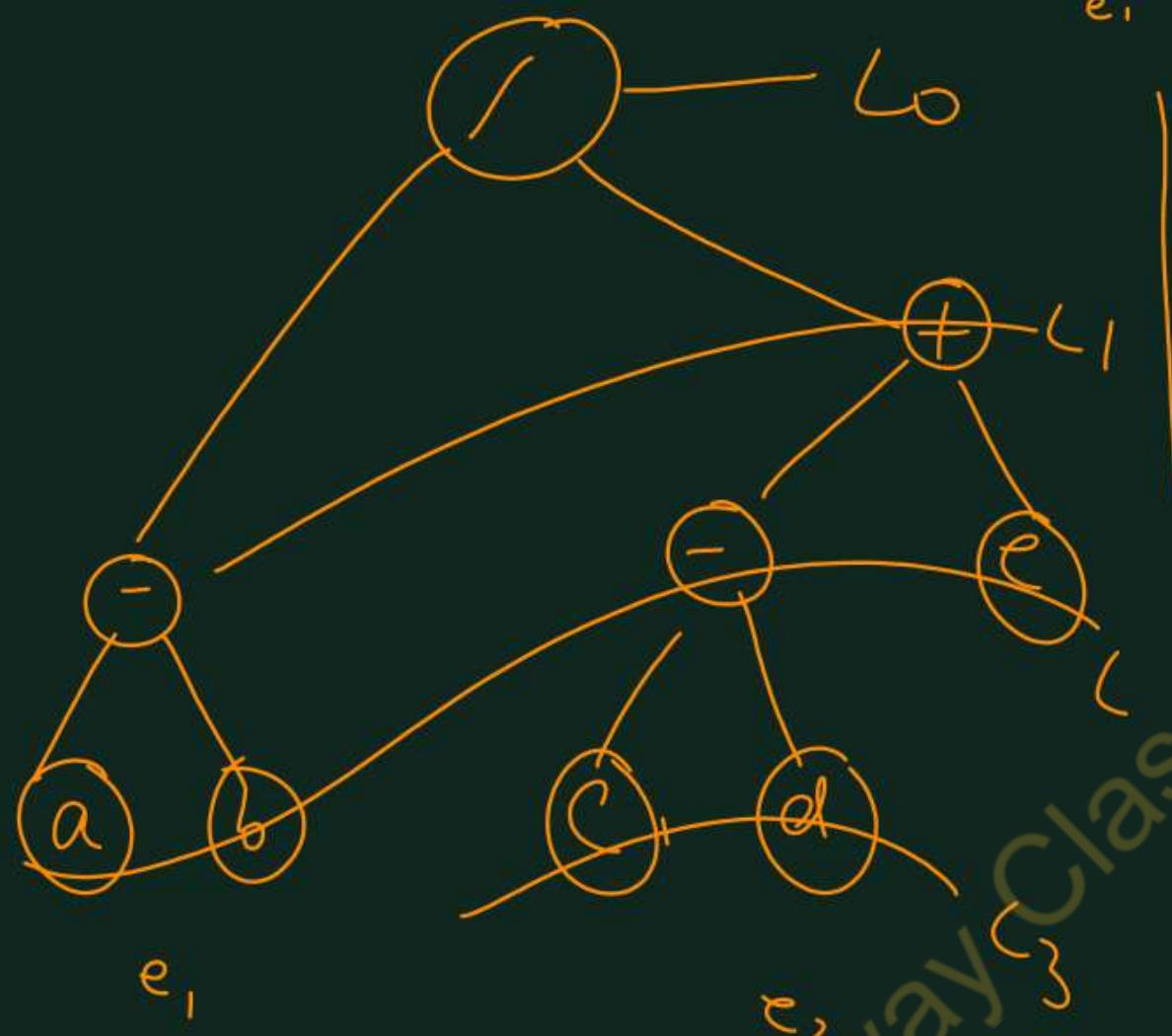
Q.

$$E = (a - b) / ((c * d) + e)$$

Q4

- E can be represented by means of the binary tree T as pictured in next figure.
- That is, each variable or constant in E appears as an "internal" node in T whose left and right subtrees correspond to the operands of the operation. For example:
 - (a) In the expression E , the operands of $+$ are $c * d$ and e .
 - (b) In the tree T , the sub trees of the node $+$ correspond to the sub expressions $c * d$ and e .
- Clearly every algebraic expression will correspond to a unique tree, and vice versa.

$$E = \left(\underbrace{a - b}_{e_1} \right) / \left(\underbrace{\left((-d) + e \right)}_{e_2, 34} \right)$$



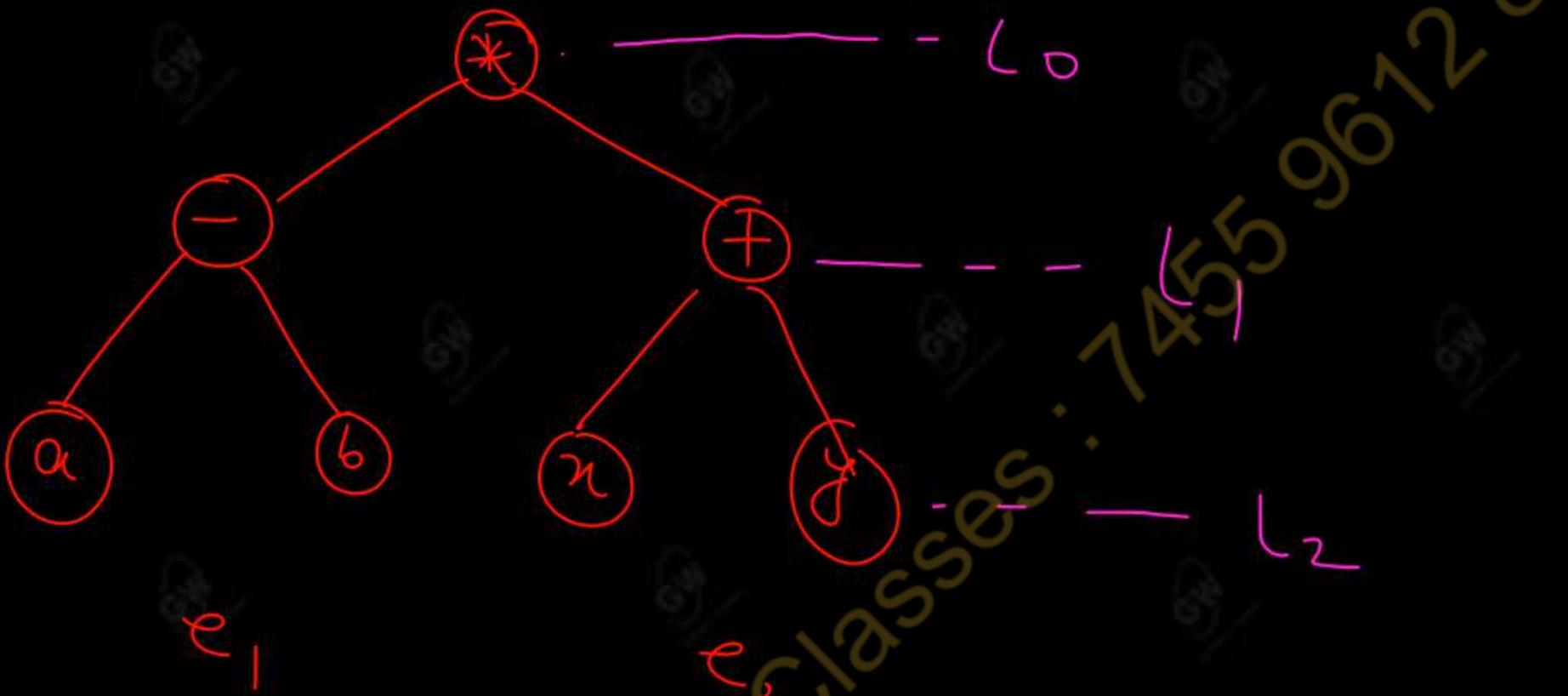
EXAMPLE : Consider the following algebraic expression E and draw binary tree:-

$$E = (2x + y) (5a - b)^3$$



EXAMPLE : Consider the following algebraic expression E and draw binary tree:-

$$E = (a - b) (x + y)$$
$$\quad \quad \quad e_1 \quad \quad \quad e_2$$



Gateway Classes : 7455961284

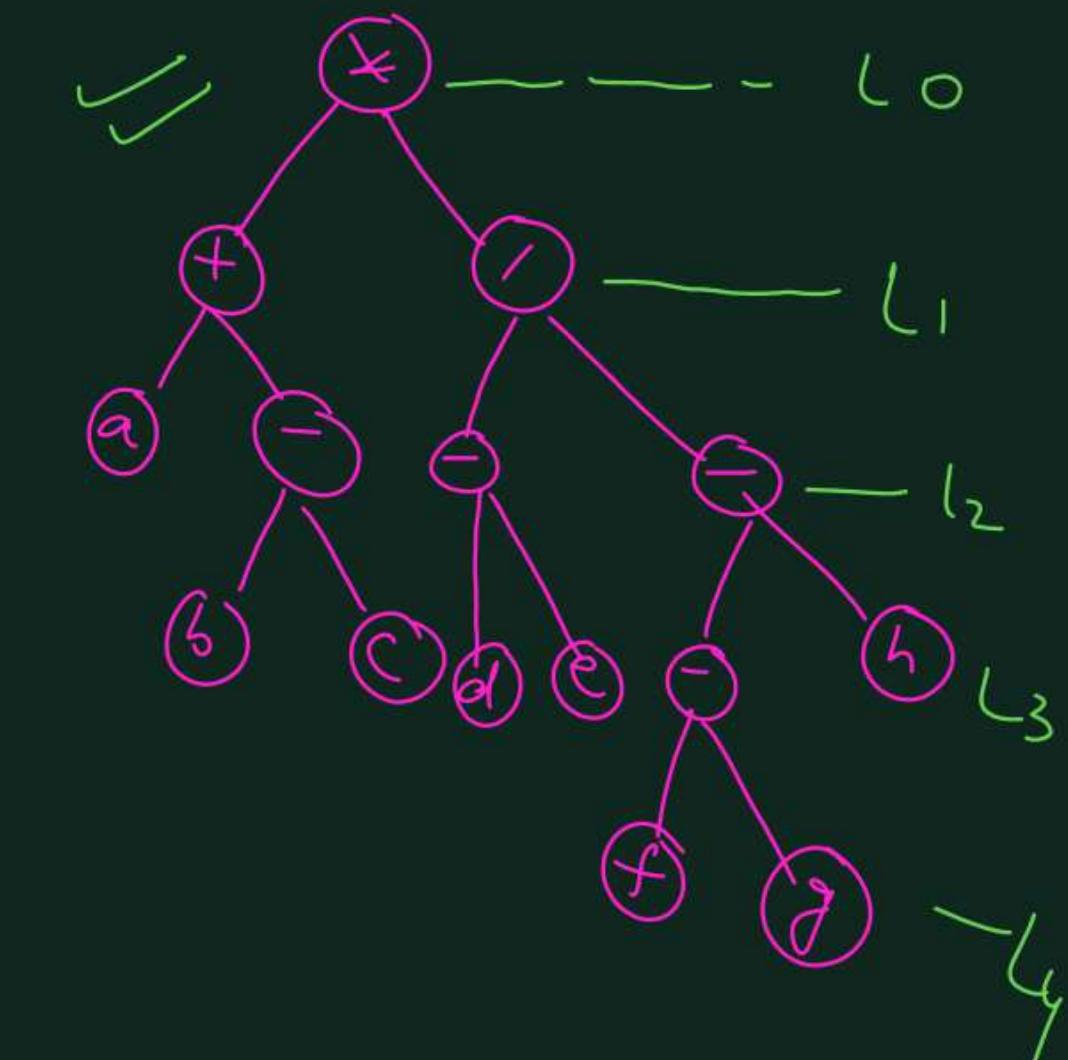
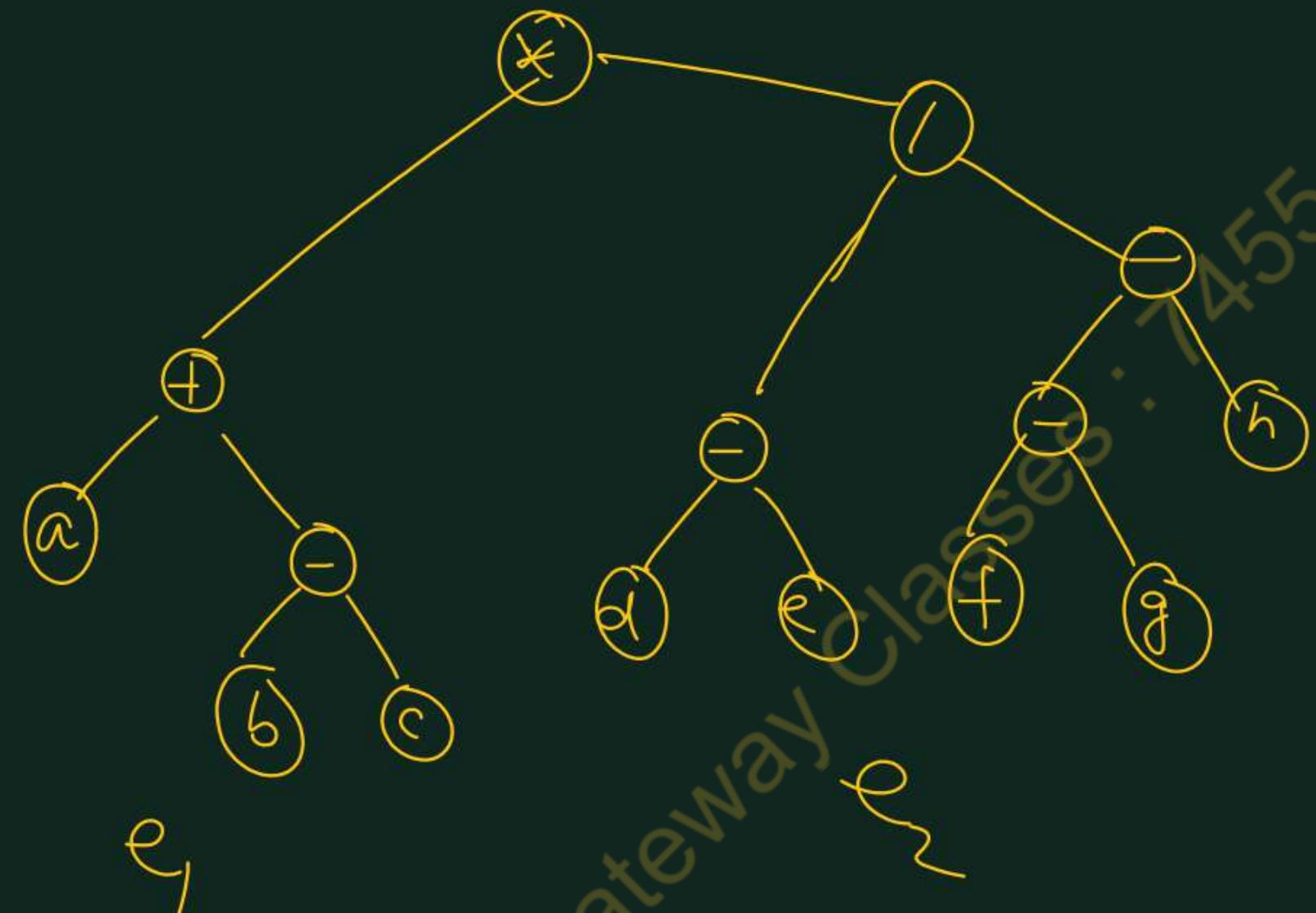
EXAMPLE : Consider the following algebraic expression E and draw binary tree:-

$$E = (\underbrace{2a + 5b}_c)^3 (\underbrace{d - 7g}_d)^3$$



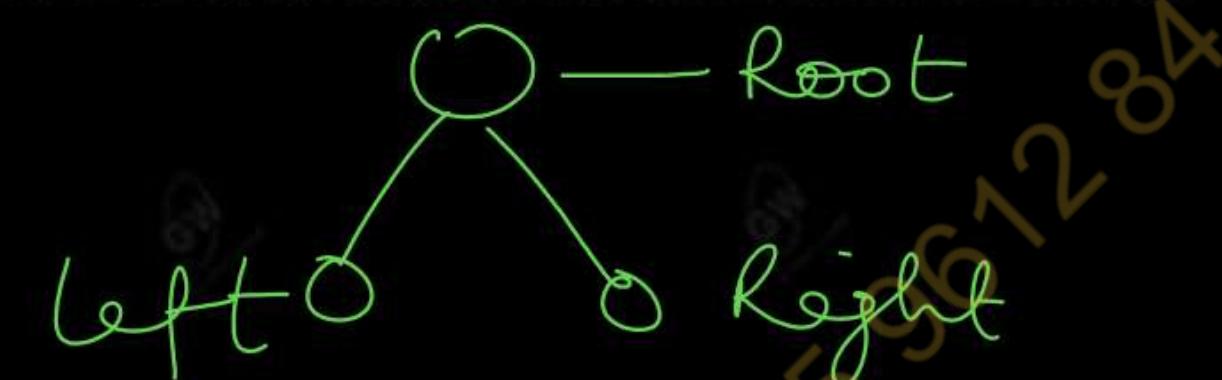
Example :-

$$\left[\underbrace{a + (b - c)}_{e_1} \right] * \left[\underbrace{(d - e) / (f - g - h)}_{e_2} \right]$$



Tree Terminology

- Root :- Tree contain a special node called the root node of the tree.



- Node:- The each element of tree is called node.
- Left and Right successor :- Suppose N is a node in T with left successor S_1 and right successor S_2 . Then N is called the *parent* (or *father*) of S_1 and S_2 .

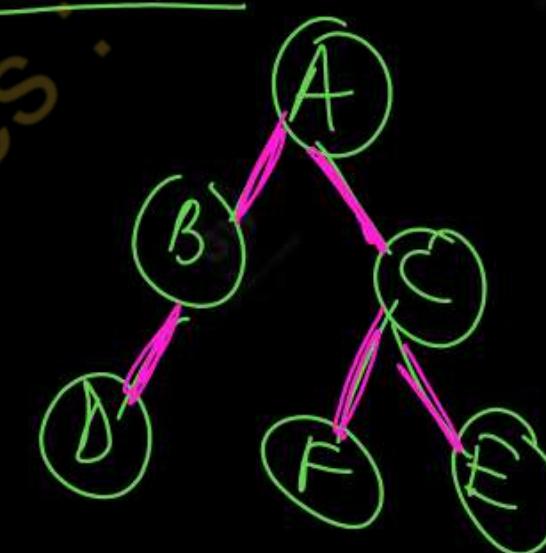


- Left and right child : - Analogously, S_1 is called the *left child* (or *son*) of N , and S_2 is called the *right child* (or *son*) of N . Furthermore, S_1 and S_2 are said to be *siblings* (or *brothers*).

- Every node N in a binary tree T , except the root, has a unique parent, called the **predecessor** of N .

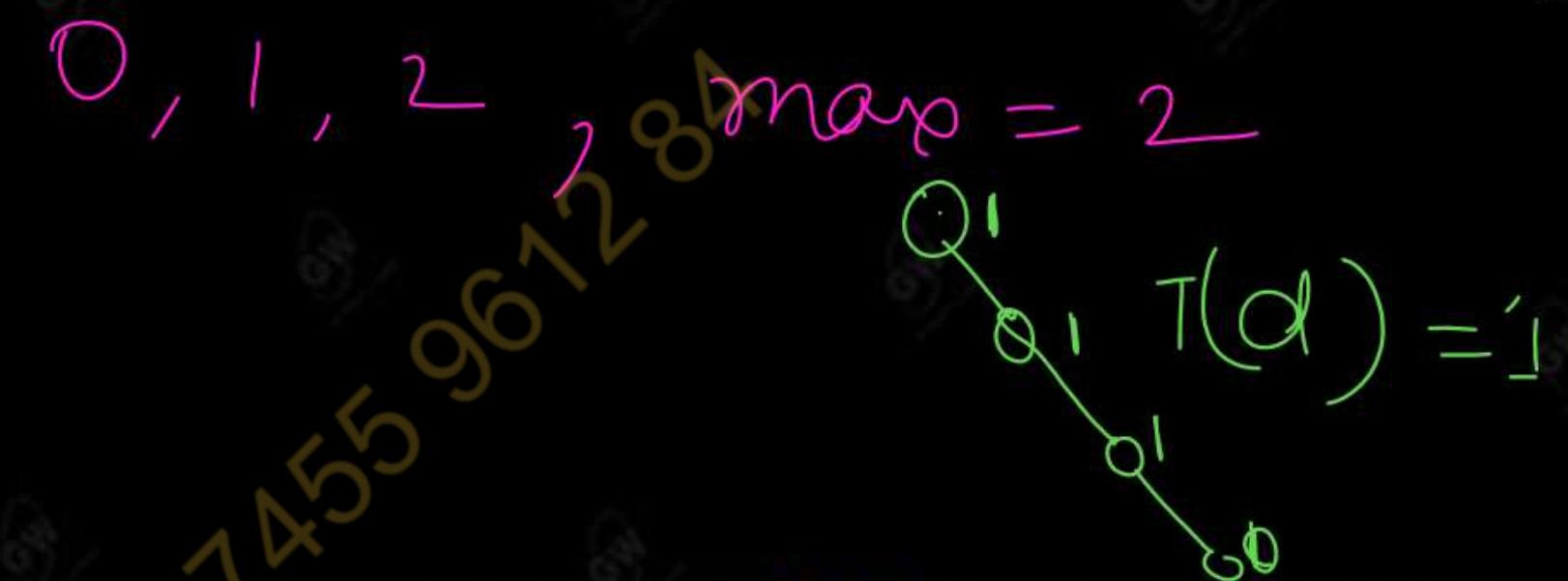


- The line drawn from a node N of T to a **successor** is called an **edge**, and a sequence of consecutive edges is called a **path**.

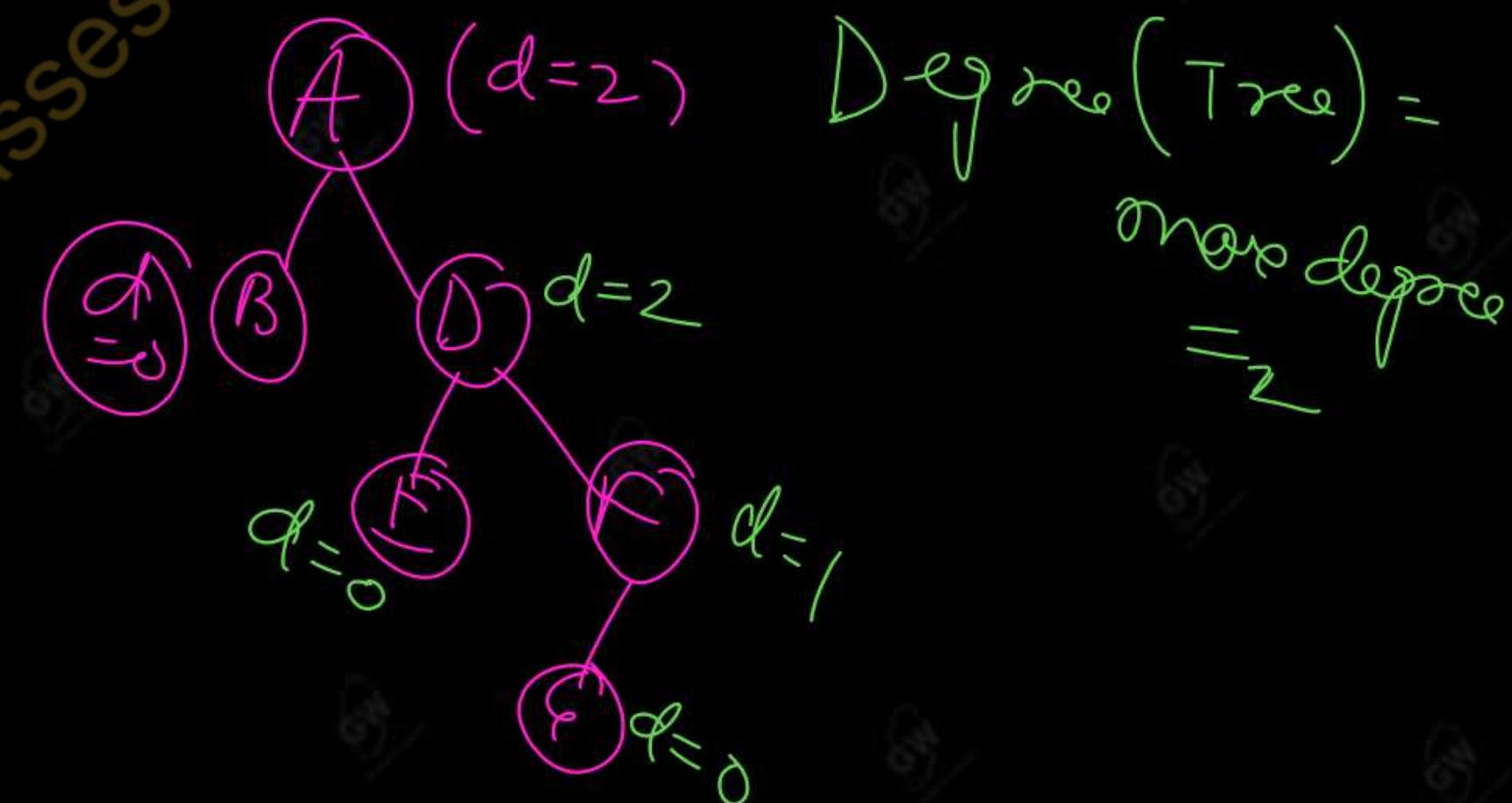


- A terminal node is called a **leaf**, and a path ending in a leaf is called a **branch**.

- Degree of a node:- In binary tree each node can have at most two children, it means the degree of a node is 2 in binary tree.



- The line drawn from a node N of T to a successor is called an **edge**, and a sequence of consecutive edges is called a **path**.



- **Terminal nodes or external nodes or leaf nodes or last nodes:** - A terminal node is called a leaf, and a path ending in a leaf is called a branch. In other words, node with degree 0 is called the terminal node.

Nodes with degree 0 = Leaf node

- **Non Terminal nodes or internal nodes:** - A binary tree can have 0,1,2 successor. A node of T which has any number of successor is called non terminal nodes.

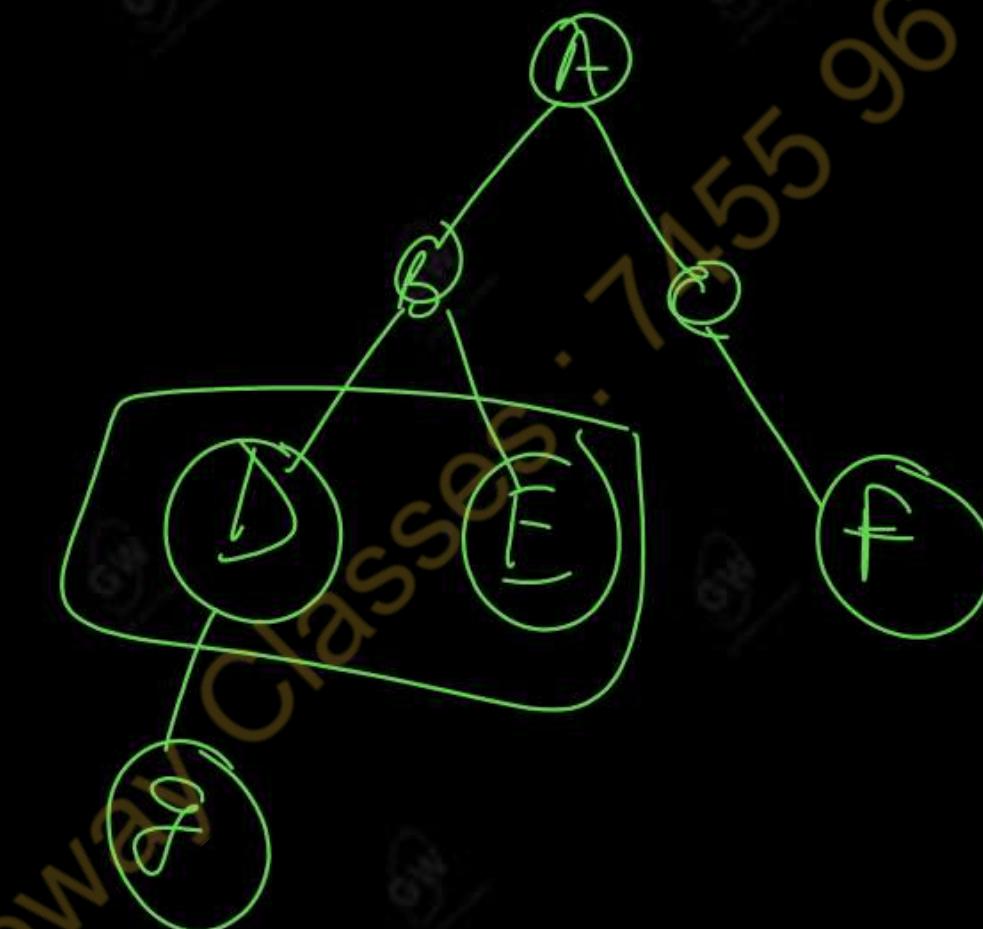
Nodes with Due
degree = Internal.

- **Level of node:-** Each node in a binary tree T is assigned a *level* number, as follows.



- The root R of the tree T is assigned the level number 0, and every other node is assigned a level number which is 1 more than the level number of its parent.

- **Generation** : - Those nodes with the same level number are said to belong to the same generation.
- **Siblings** : - The node with same parents.

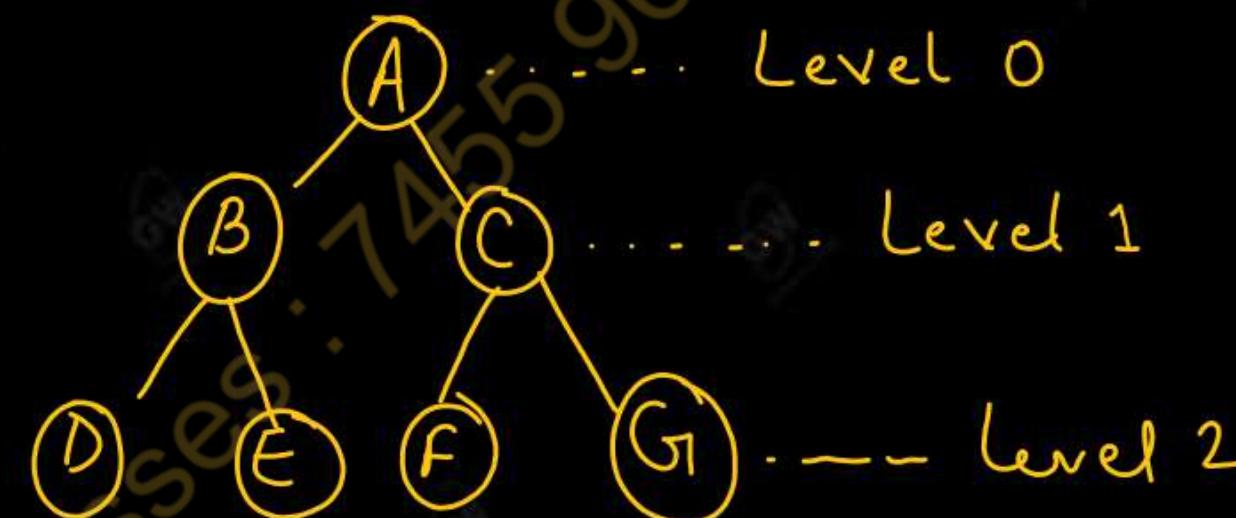


$B \& C = \text{siblings}$
 $E \& F = \text{siblings}$
SIB
YS

- The **depth** (or height) of a tree T is the maximum number of nodes in a branch of T. This turns out to be 1 more than the largest level number of T. The height of empty tree is 0. The height of a tree containing a single node is 1. The formula used to find the height of a tree is

$$h = L_{\max} + 1$$

- The following tree T has depth 3.

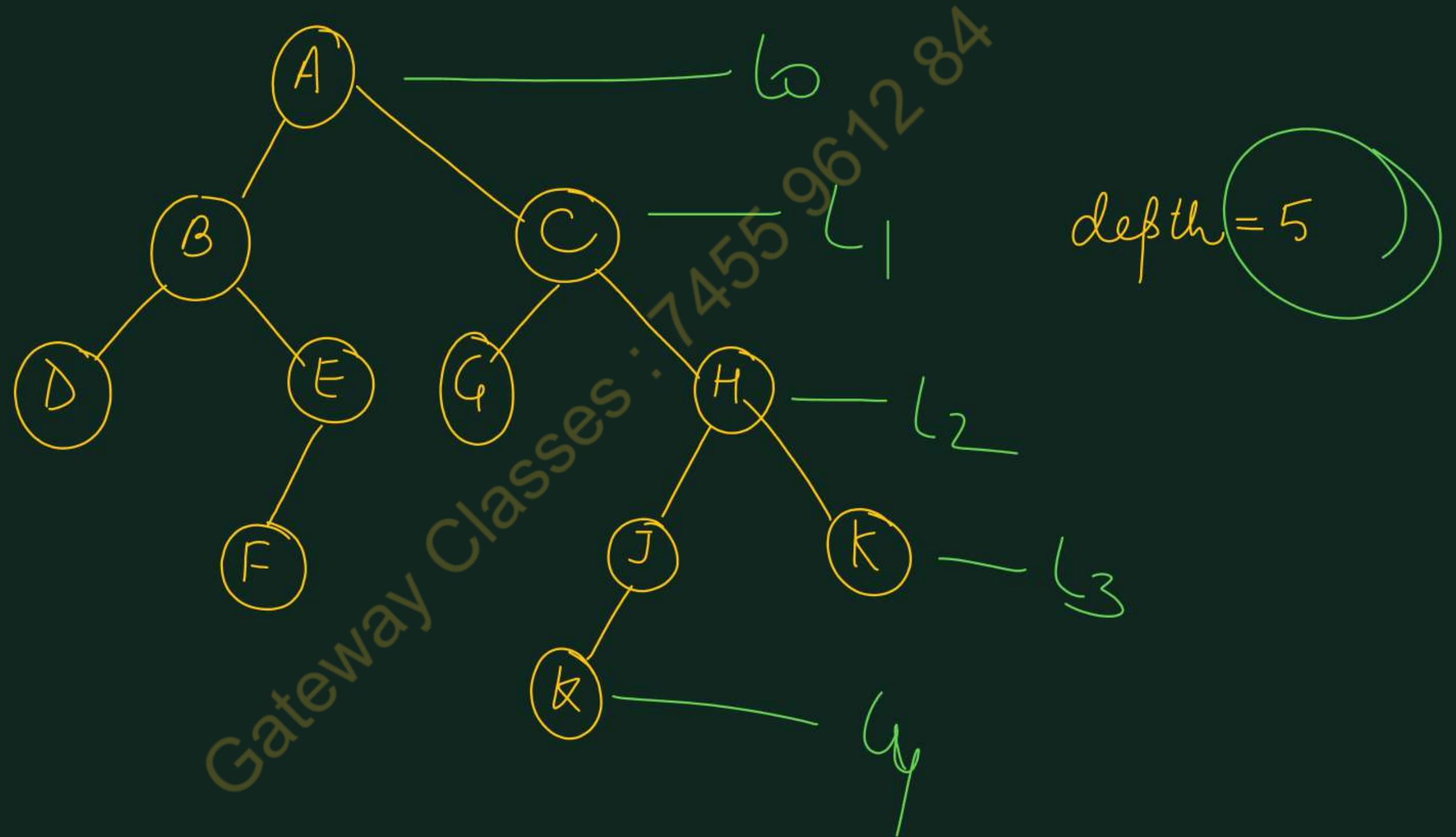


height = $2 + 1$
= 3



See one more example in next slide.

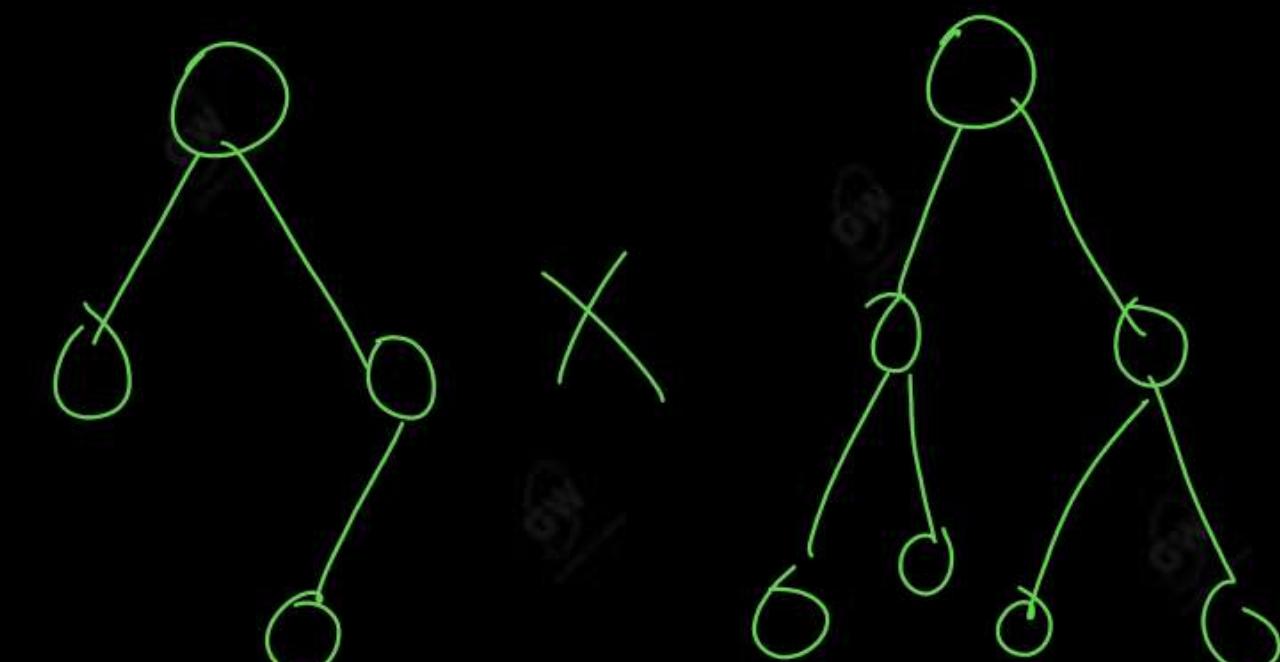
- Binary trees T and T' are said to be **similar** if they have the same structure or, in other words, if they have the same shape.
- The trees are said to be **copies** if they are similar and if they have the same contents at corresponding nodes.



Fully or Strictly Binary Trees

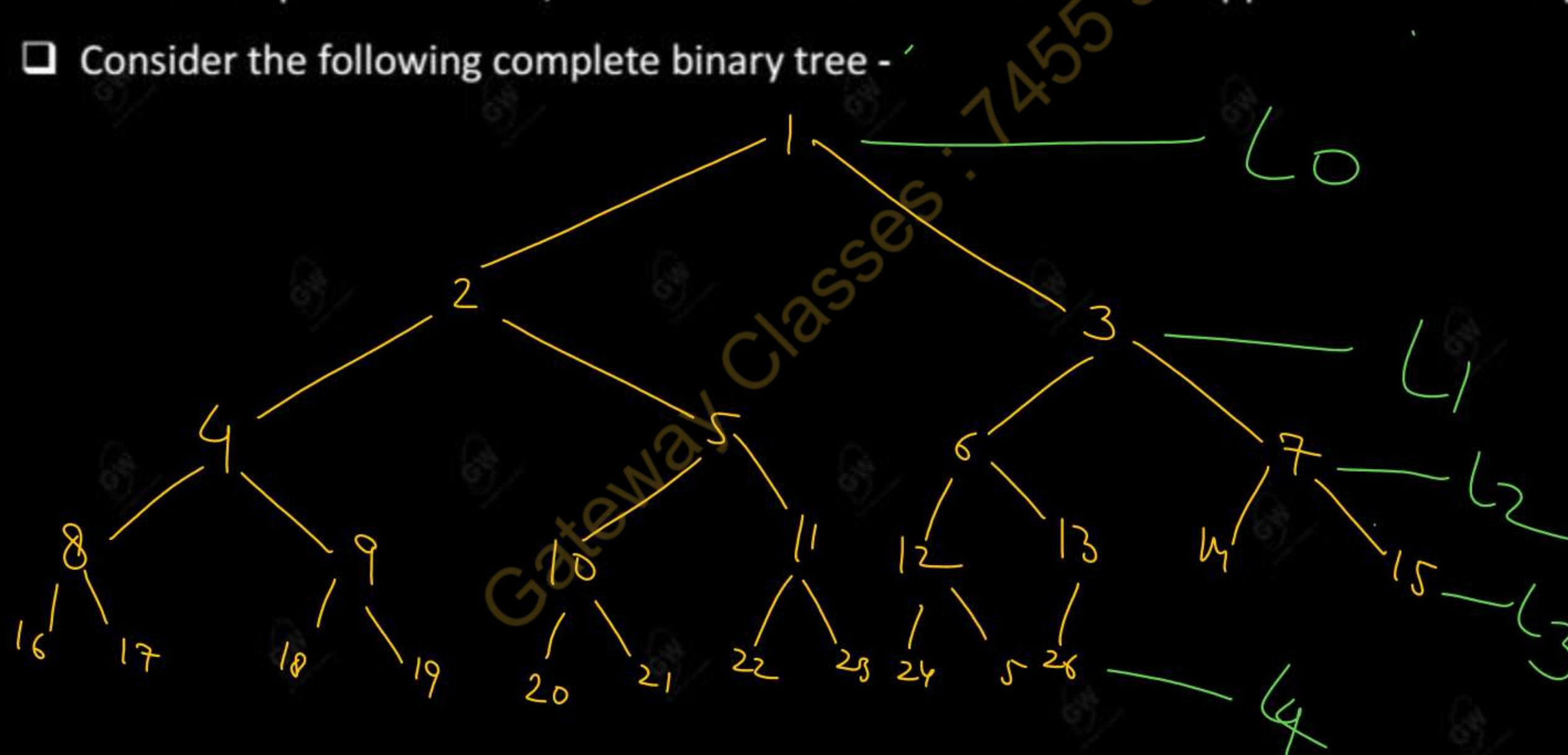
full

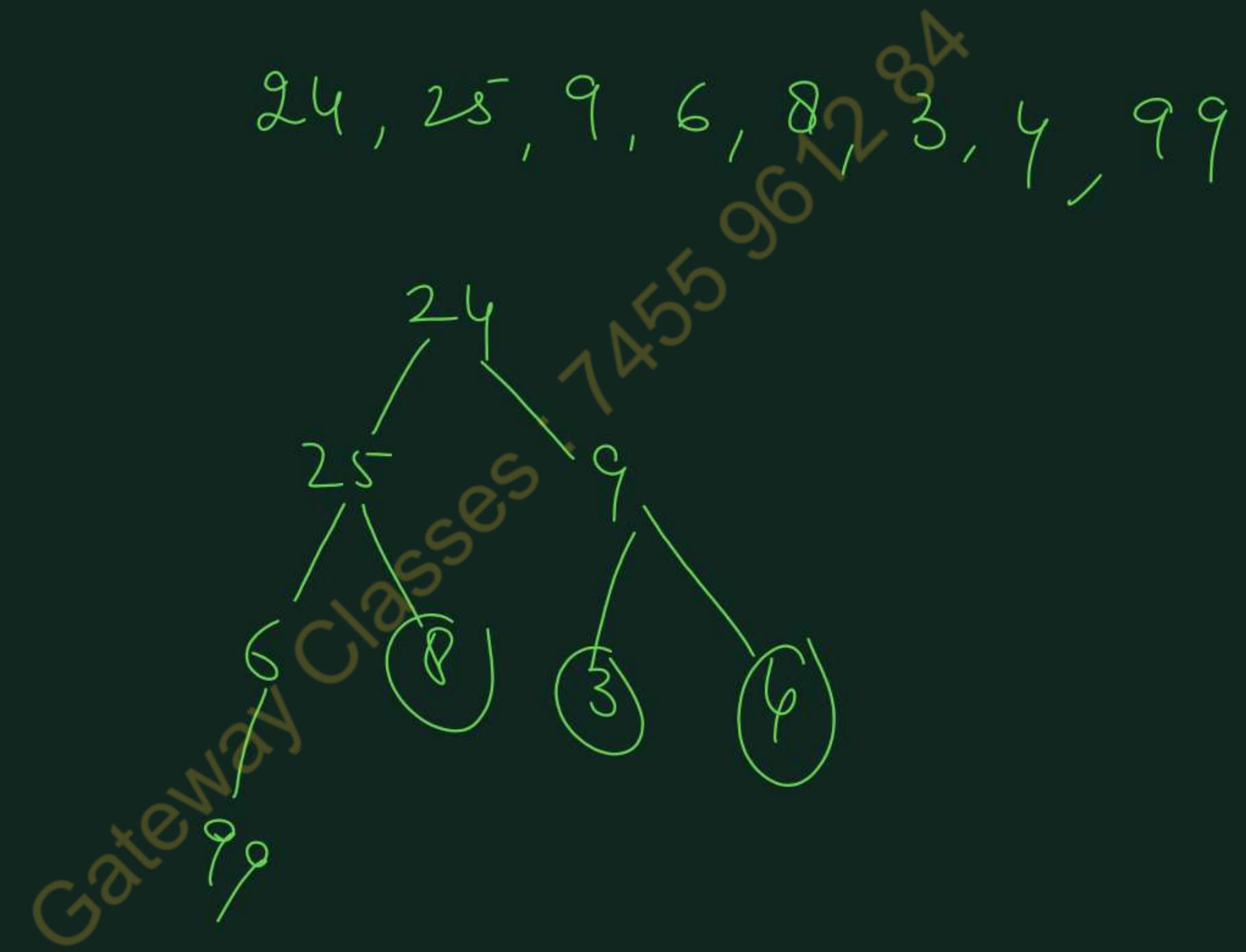
- ❑ A full binary tree is a binary tree in which all of the nodes have either 0 or 2 offspring (children).
 - ❑ In other terms, a full binary tree is a binary tree in which all nodes, except the leaf nodes, have two offspring.



Complete Binary Tree -

- Consider any binary tree T. Each node of T can have at most two children.
- Accordingly, one can show that level of T can have at most 2^r nodes.
- The tree T is said to be complete if all its levels, except possibly the last, have the maximum number of possible nodes, and if all the nodes at the last level appear as far left as possible.
- Consider the following complete binary tree -





- One can easily determine the children and parent of any node K in any complete tree T . Specifically, the left and right children of the node K are, respectively, $2 * K$ and $2 * K + 1$, and the parent of K is the node $[K / 2]$.
- For example, the children of node 9 are the nodes 18 and 19, and its parent is the node $[9 / 2] = 4$. The depth d , of the complete tree T , with n nodes is given by -

$$D = \lceil \log_2 n + 1 \rceil$$

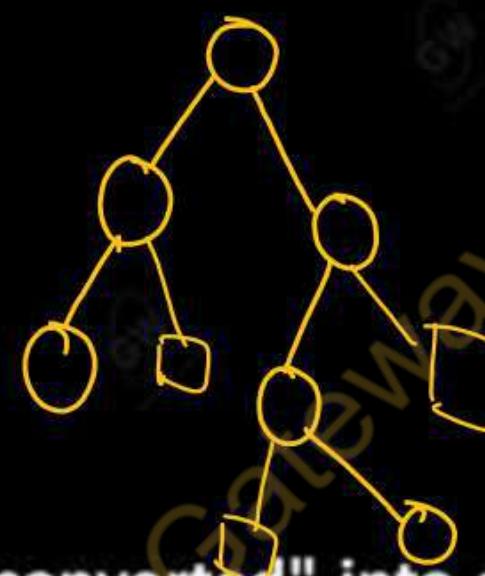
- This is a relatively small number.
- For example, if the complete tree T , has $n = 1000,000$ nodes, then its depth $D = 21$.

Imp

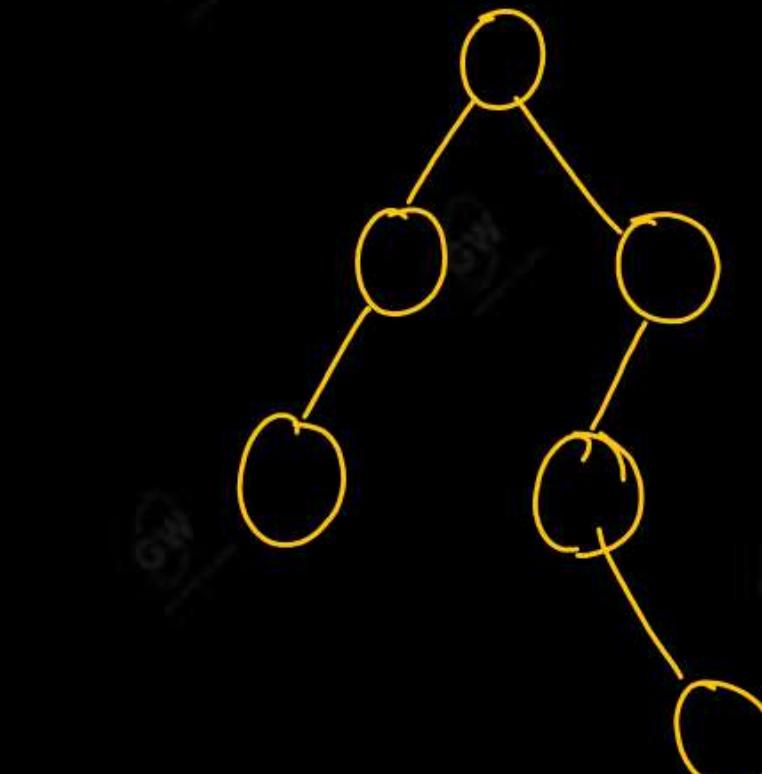
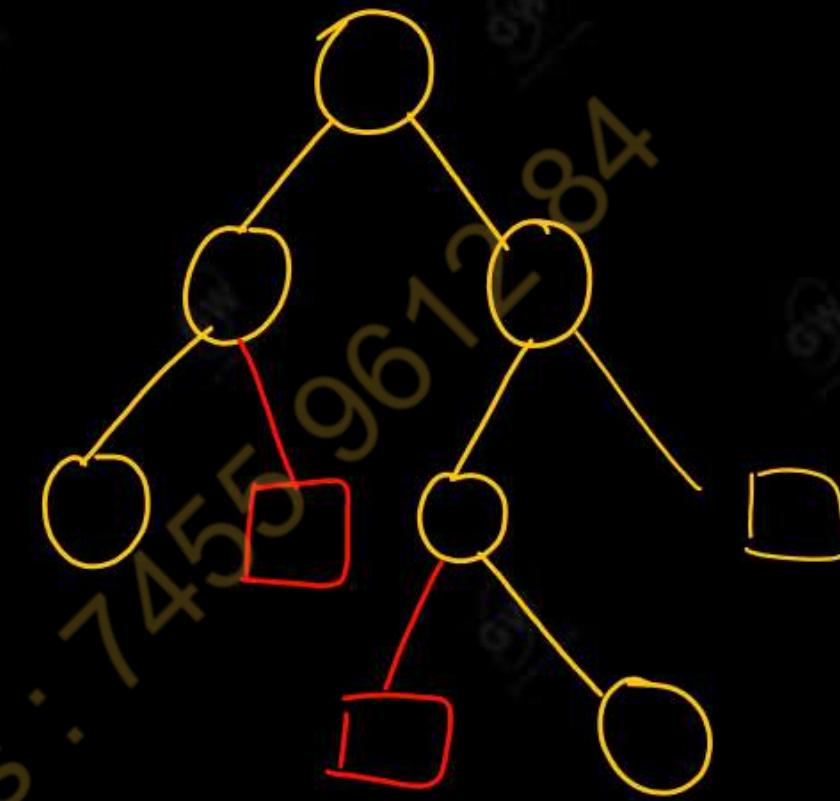
$$\begin{aligned} D &= \lceil \log_2 n + 1 \rceil \\ &= \lceil \log_2 1000000 + 1 \rceil \\ &= \lceil \log_2 2^9 + 1 \rceil \\ &= \lceil 9 + 1 \rceil \end{aligned}$$

Extended Binary Tree or 2-Tree -

- A binary tree T is said to be a 2 – tree or an *extended binary tree* if each node N has either 0 or 2 children.
- In such a case, the nodes with 2 children are called *internal nodes*, and the nodes with 0 children are called *external nodes*.
- Sometimes the nodes are distinguished in diagrams by using circles for internal nodes and squares for external nodes.
- Consider the following Binary Tree:-



- Then T may be "converted" into a 2 tree by replacing each empty subtree by a new node as shown in the next figure.

(a) Binary tree T .

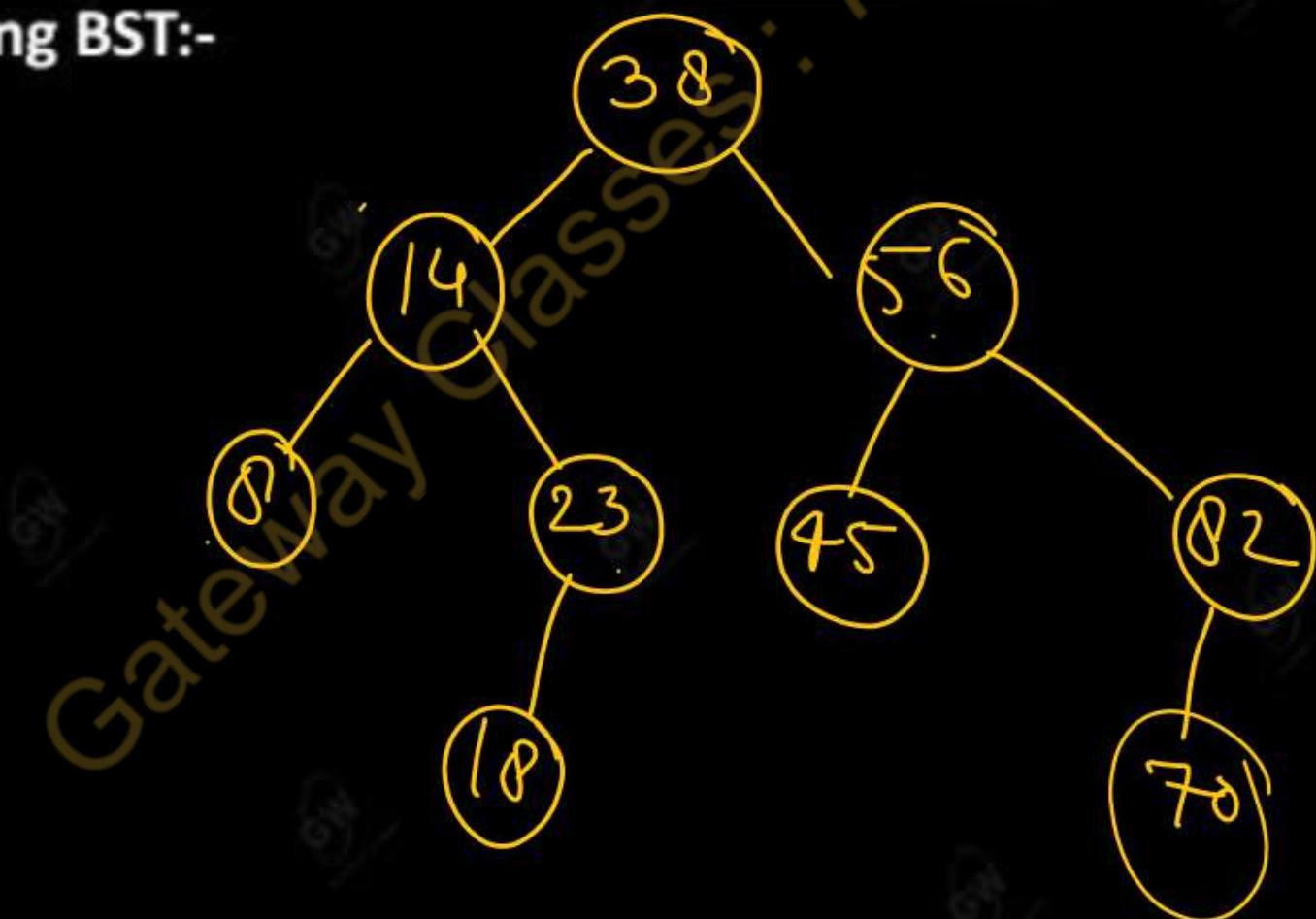
(b) Extended 2-tree.

Gateway Classes : 7455 9612 84

Binary Search Tree (BST) :-

A binary Search Tree or BST or Binary sorted Tree can be defined as:-

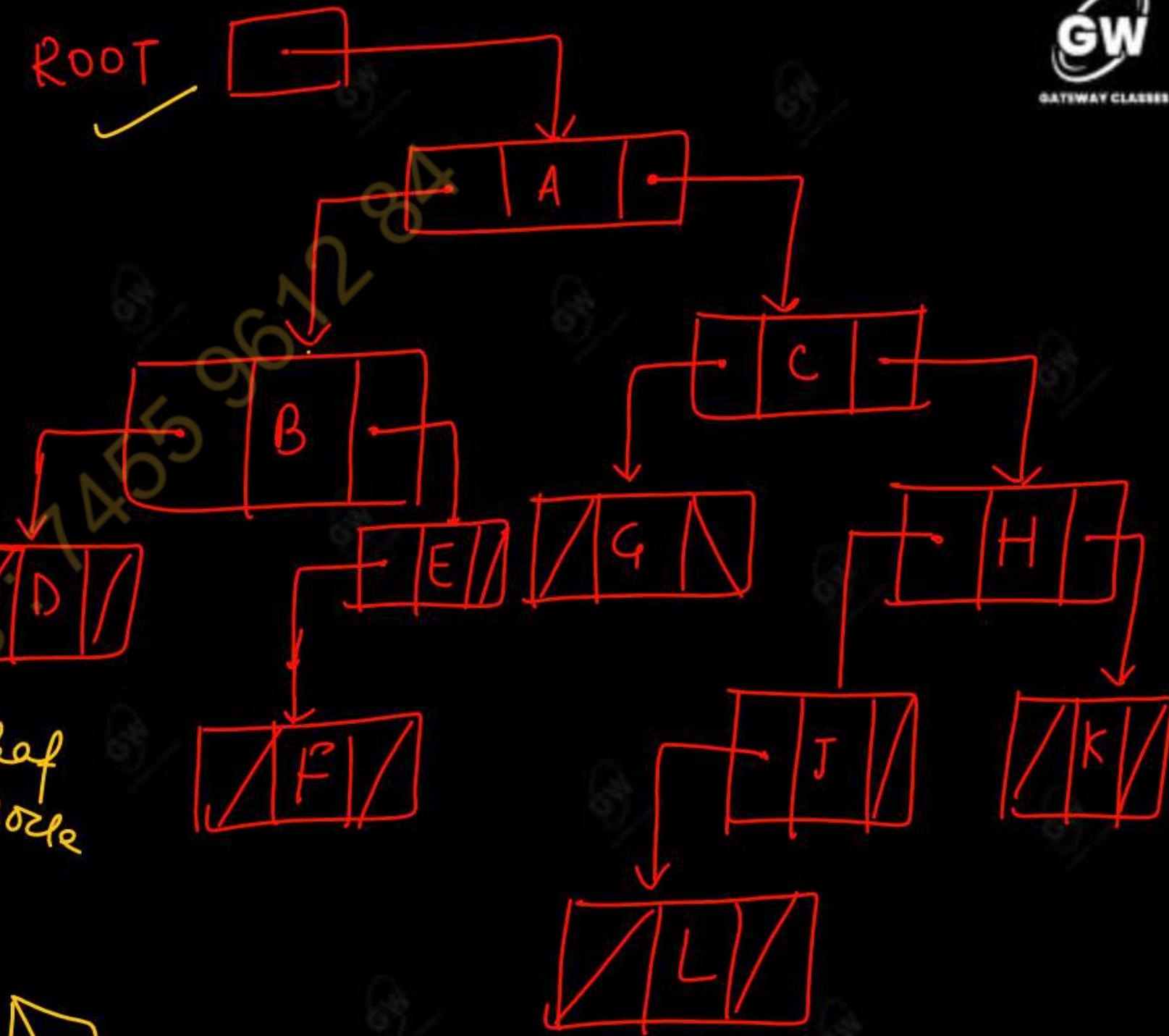
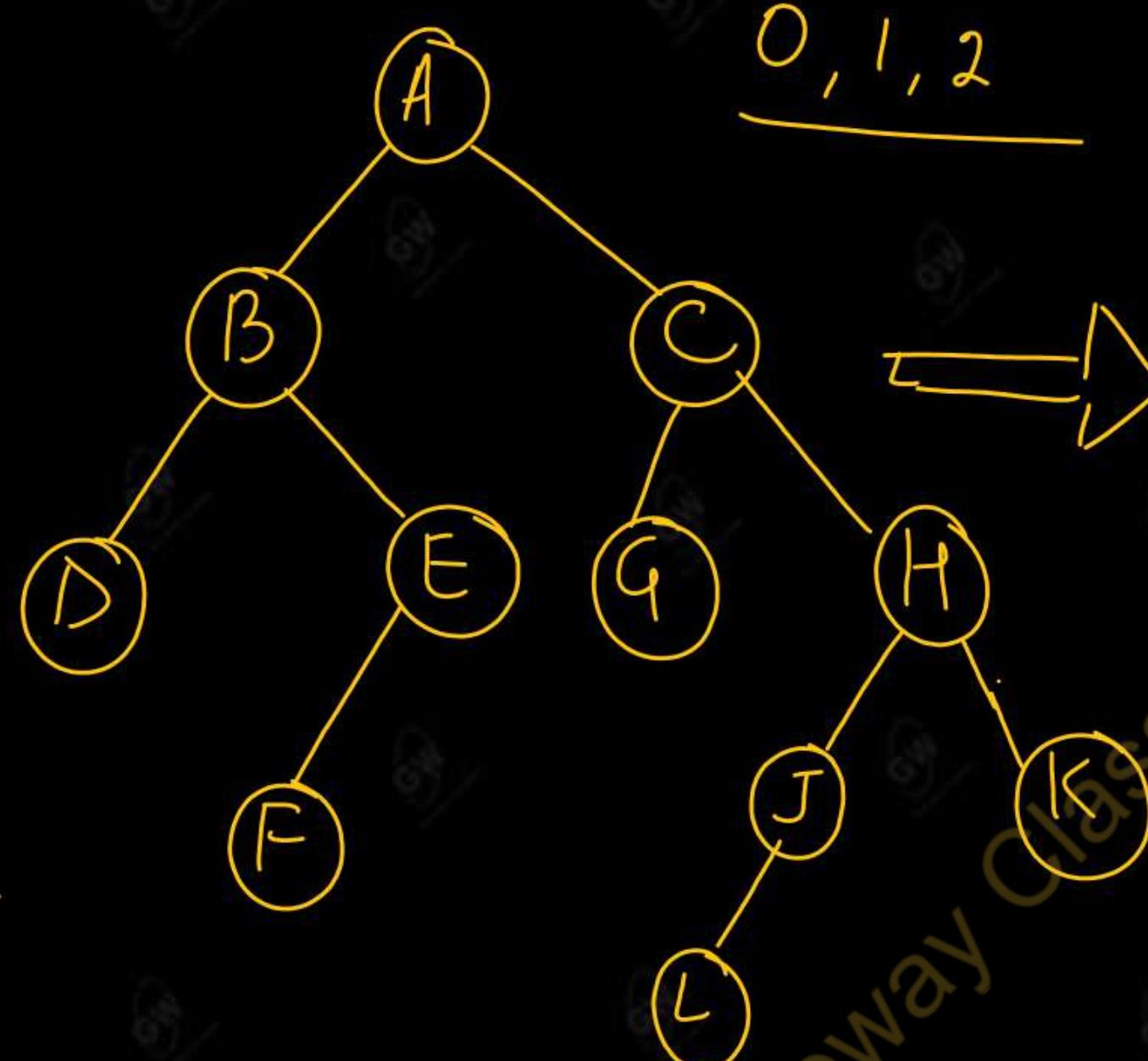
- It is a binary tree and
- The value of the left child should be less than the value of the root node and
- The value of right child should be greater than the value of root node and
- The left and right subtrees are again binary search tree.
- Consider the following BST:-



- Let T be a binary tree.
- There are two ways to represent T in memory.
 1. **Linked List representation of Binary Tree in memory** - The first and usual way is called the link representation of T and is analogous to the way linked lists are represented in memory.
 2. **Sequential representation of binary tree in Memory** - The second way, which uses a single array, called the sequential representation of T.

1. Linked Representation of Binary Trees -

- ❑ Consider a binary tree T.
- ❑ In this method, T will be maintained in memory by means of a linked representation which uses three parallel arrays, INFO, LEFT and RIGHT, and a pointer variable ROOT as follows.
- ❑ First of all, each node N of T will correspond to a location K such that:-
 - (1) INFO [K] contains the data at the node N.
 - (2) LEFT [K] contains the location of the left child of node N.
 - (3) RIGHT [K] contains the location of the right child of node N.
- ❑ Furthermore, ROOT will contain the location of the root R of T.
- ❑ If any subtree is empty, then the corresponding pointer will contain the null value; if the tree T itself is empty, then ROOT will contain the null value.



	INFO	LEFT	RIGHT
1	E	11	NULL
2	6	✓	
3	C	18	13
4			
5	A	8	3
6	7		
7	9		
8	B	10	1
9			
10	12		
11	D	NULL	NULL
12	F	NULL	NULL
13	14		
14	H	15	16
15	17		
16	J	19	NULL
17	K	NULL	NULL
18	20		
19	G	NULL	NULL
20	L	NULL	NULL

ROOT



AVAIL

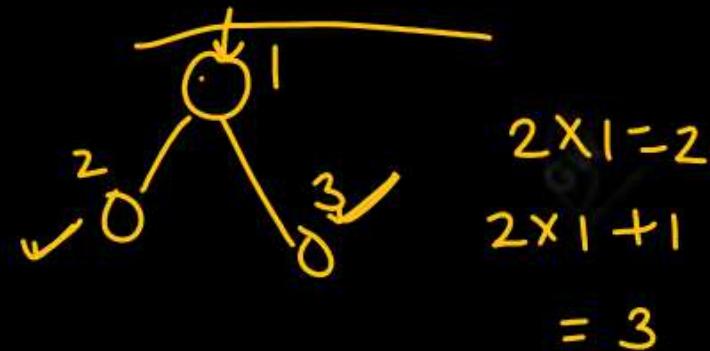


Avail

Root

2. Sequential Representation of Binary Trees -

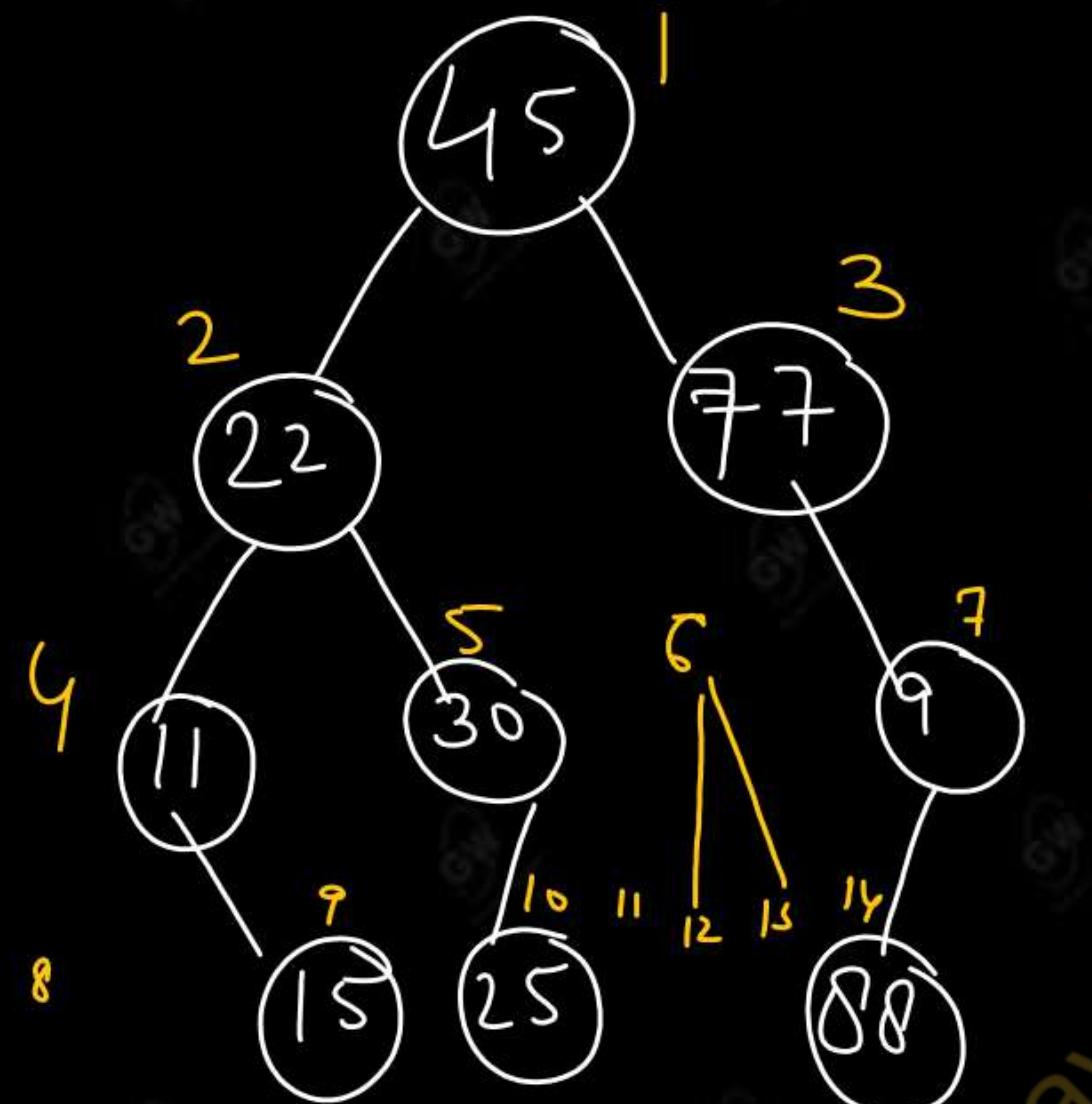
- Suppose T is a binary tree that is complete or nearly complete.
- Then there is an efficient way of maintaining T in memory called the *sequential representation* of T .
- This representation uses only a single linear array **TREE** as follows:-
 - (a) The root R of T is stored in TREE [1].
 - (b) If a node N occupies $\text{TREE} [K]$, then its left child is stored in $\text{TREE} [2K]$ and its right child is stored in $\text{TREE} [2 * K + 1]$.
- Again, **NULL** is used to indicate an empty subtree.
- In particular, $\text{TREE} [1] = \text{NULL}$ indicates that the tree is empty.
- The sequential representation of the binary tree T is represented in the next figures:



2 4 6 9 14



SEQUENTIAL REPRESENTATION OF BINARY TREE



1	45
2	22
3	77
4	11
5	30
6	NULL
7	9
8	NULL
9	15
10	25
11	
12	
13	
14	88
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	

- Observe that we require 14 locations in the array TREE even though T has only 9 nodes.
- In fact, if we included null entries for the successors of the terminal nodes, then we would actually require TREE [29] for the right successor of TREE [14].
- Generally, the sequential representation of a tree with depth d will require an array with approximately 2^{d+1} elements.
- Accordingly, this sequential representation is usually inefficient unless, as stated above, the binary tree T is complete or nearly complete.
- For example, the tree T in previous figure has 11 nodes and depth 5, which means it would require an array with approximately $2^6 = 64$ elements.

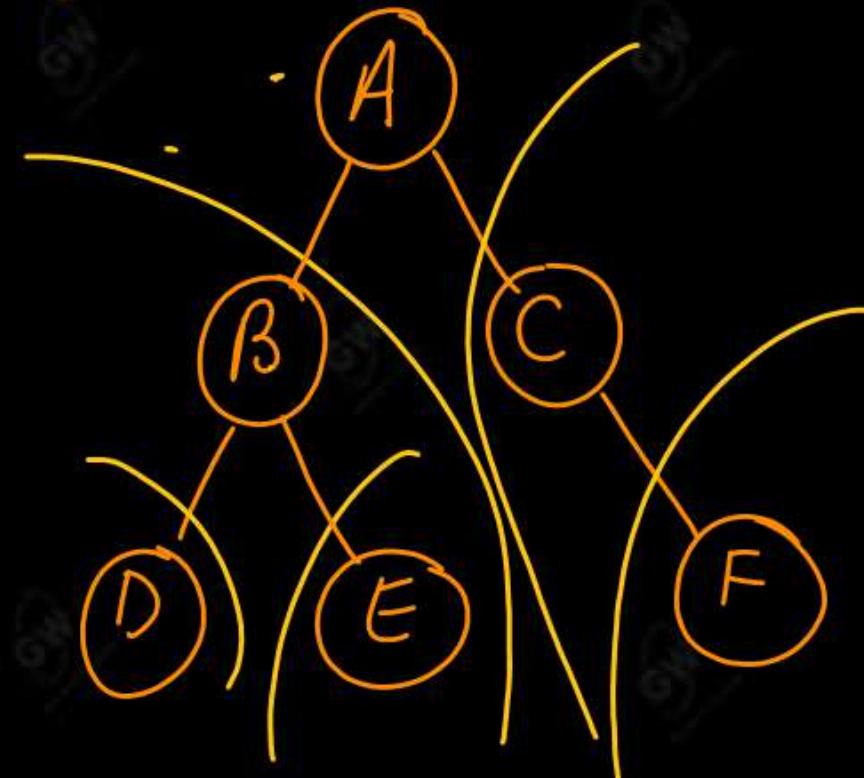
- There are three standard ways of traversing a binary tree T with root R.
- These three algorithms, called preorder, inorder and postorder, are as follows:
- **Preorder:**
 - (1) Process the root R.
 - (2) Traverse the left subtree of R in preorder.
 - (3) Traverse the right subtree of R in preorder.
- **Inorder:**
 - (1) Traverse the left subtree of R in inorder.
 - (2) Process the root R.
 - (3) Traverse the right subtree of R in inorder.
- **Postorder:**
 - (1) Traverse the left subtree of R in postorder.
 - (2) Traverse the right subtree of R in postorder.
 - (3) Process the root R.

3 ways

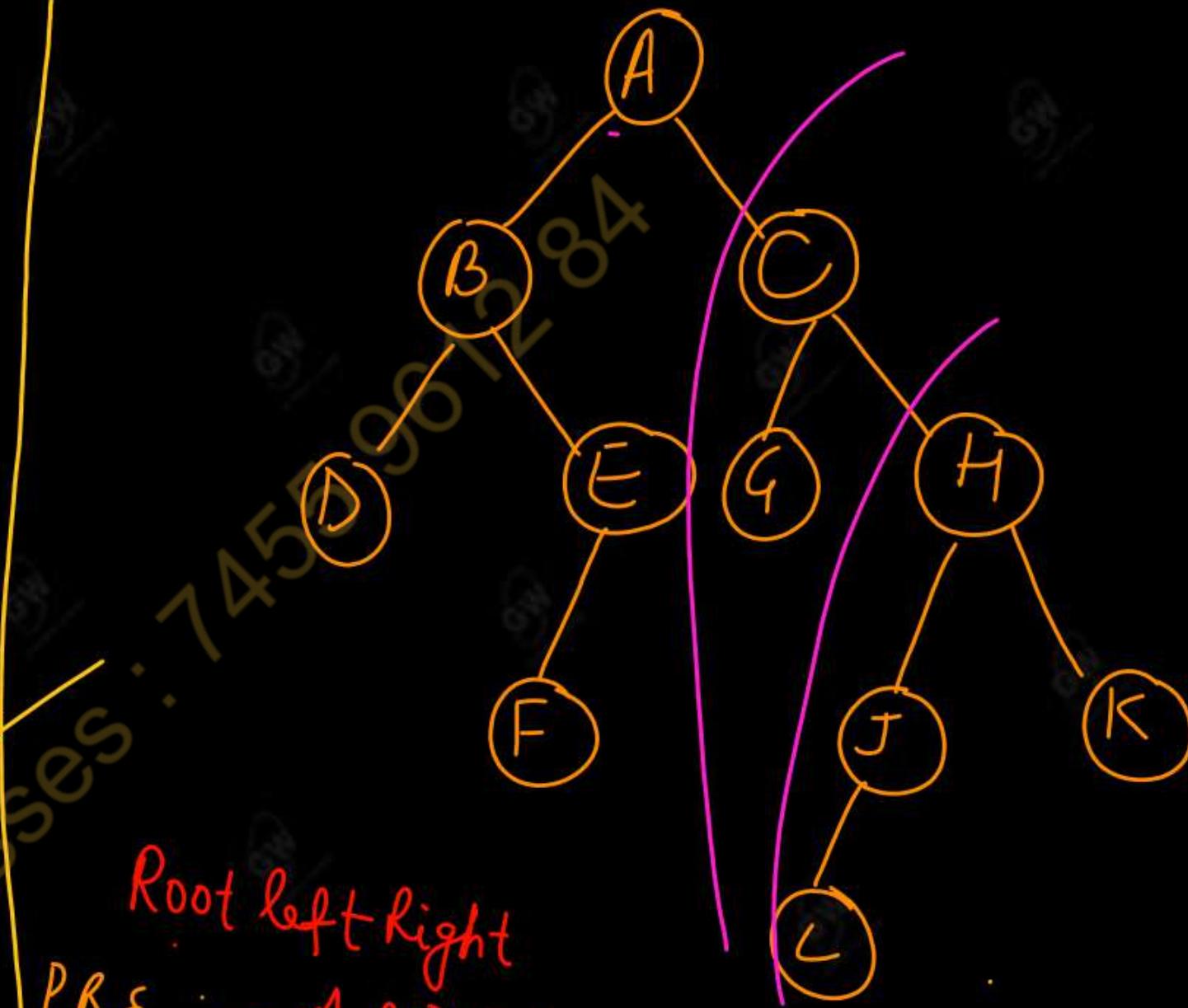
1) Pre - Root, left, right

2) In - left, Root, Right

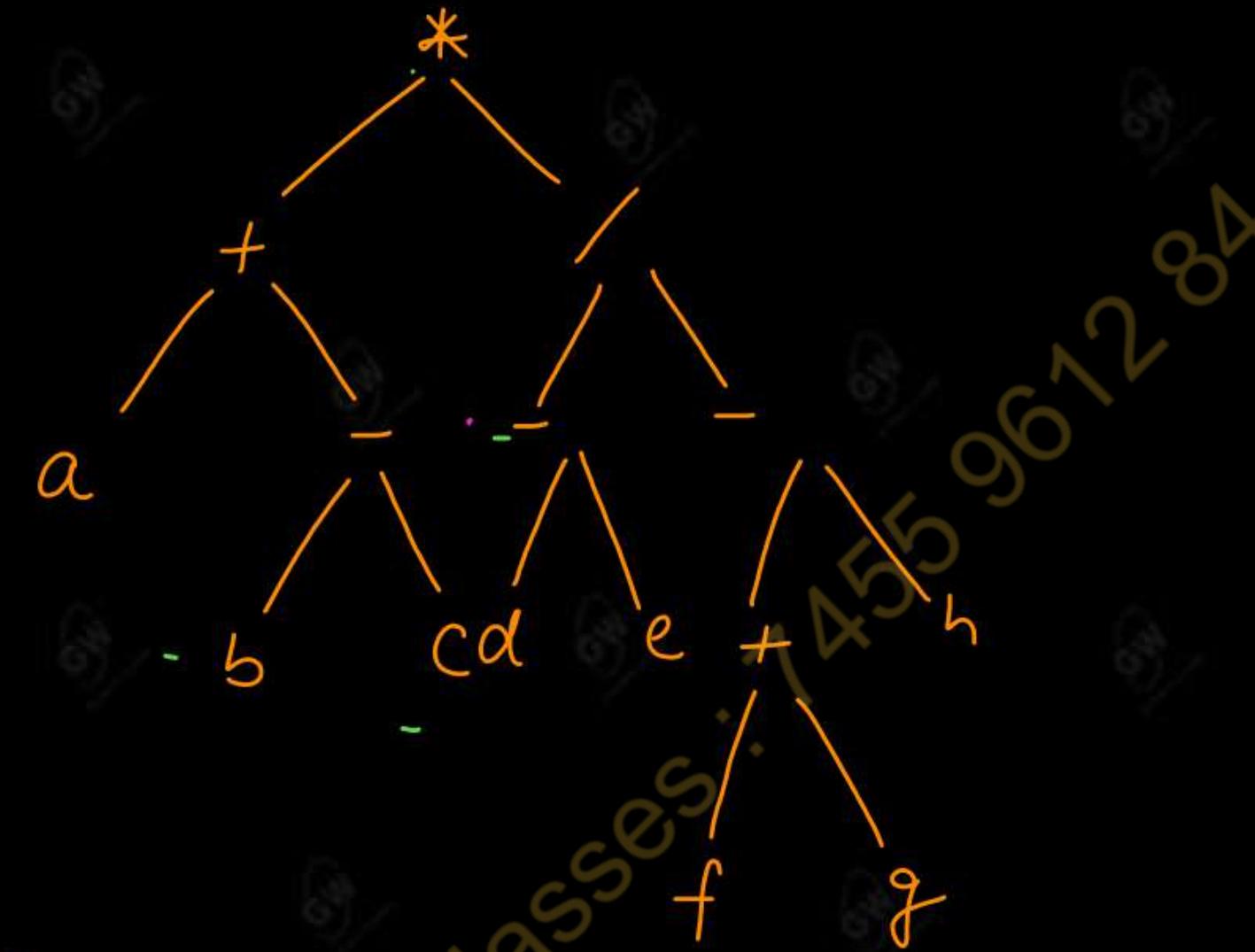
3) Post order - left, right, Root

Example:-

- 1) PRE ORDER - Root left left
- A B D E C F
- 2) IN ORDER - left Root right
- D B E A C F
- 3) POST ORDER - left right Root
- D E B F C (A)



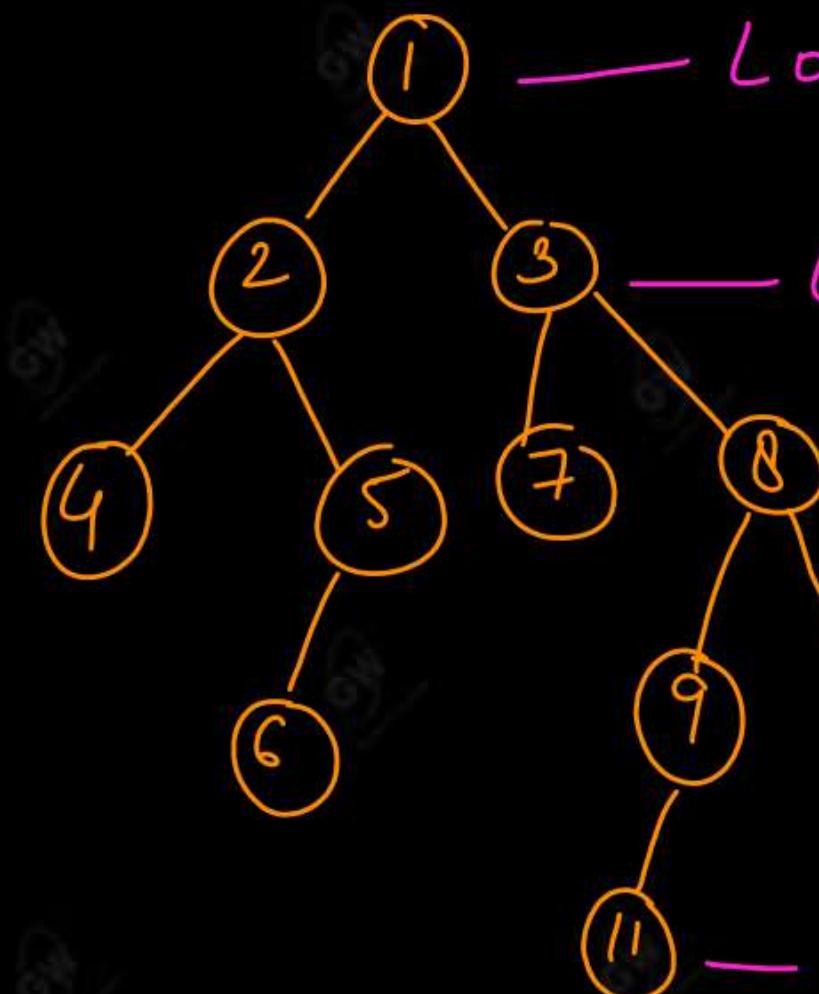
Root left right
PRE - A B D E F C G H J L K
IN - D B F E A G C L J H K
POST - D F E B G L J K H C A



PRE :- Root, left, right * + a - b c / - d e - + f g h

IN .- left Root Right a + b - c * d - e / f + g - h

POST :- left right root a b c - + d e - f g + h - / *



① PRE - 1 2 4 5 6 3 7 8 9 11 10
② IN - left Root Right 4 2 6 5 / 7 3 11 9 0 10
③ POST - left Right Root 4 6 5 2 7 11 9 10 8 3 1
④ LEVEL BY LEVEL - 1, 2, 3, 4, 5, 7, 8, 6, 9, 10, 11

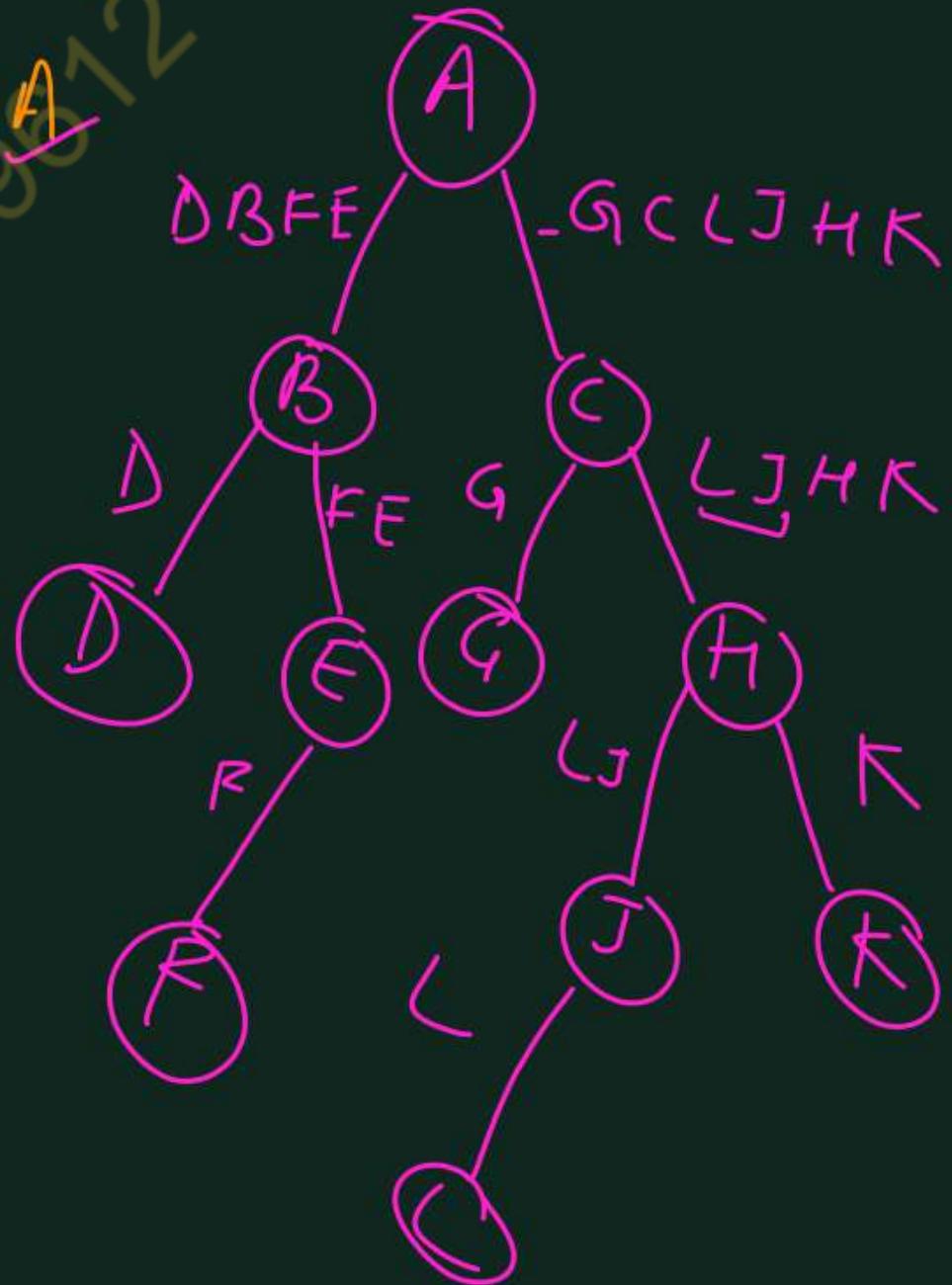
Example - Draw T.

Inorder :-

DBF E A G C L J H K

Postorder :-

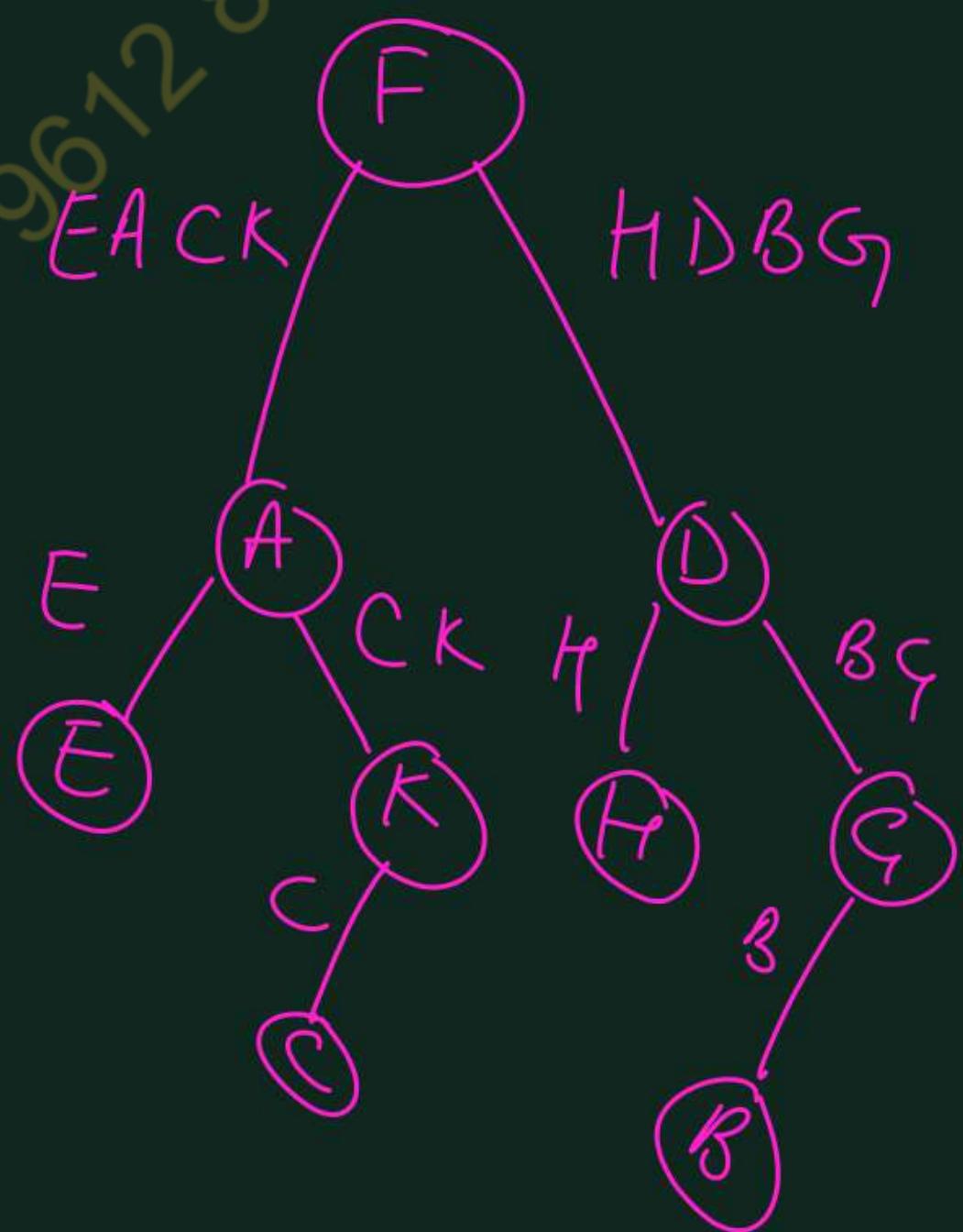
D F E B G L J C H A



Example:- Draw the tree T.

Inorder :- $E \ A \ C \ K \ F \ H \ D \ B \ G$

Preorder :- $F \ A \ E \ K \ C \ D \ H \ G \ B$



AKTU PYQs

Q.1 Construct a tree for the following preorder and post order and write its in order traversal.

Preorder : 24, 14, 13, 19, 17, 15, 10, 5, 8, 6, 7, 20

Post order : 13, 15, 17, 10, 19, 14, 7, 6, 20, 8, 5, 24

2014-15, 10 marks

Q.2 Following are the in order and post order traversal of a binary tree T-

(a) DKIBAEGHJFC

(b) KDI EAGBFCJH

Construct the tree T.

2014-15 , 5 marks

Q.3 For tree construction which is the suitable and efficient data structure and why?

2015-16, 2 marks

Array.

Dynamic
linked list

Q.4 Draw a binary tree which following traversal:

Inorder : D B H E A I F J C G

Preorder : A B D E H C F I J G

Q.5 Define complete binary tree. Give example.

Q.6 Write a program for various traversing techniques of binary tree with neat example.

Q.7 Define the following :- (i) Tree (ii) Level of a node (iii) Height of a tree

Q.8 Draw a binary tree which has the following traversal-

Inorder : D J G B A E H C F I

Preorder: A B D G J C E H F I

2015-16, 5 marks

2016-17, 2 marks

2016-17, 10 marks

2016-17, 2 marks

2016-17, 7 marks

Q.9 Define tree, binary tree, complete binary tree and full binary tree. Write algorithms or function to obtain traversals of binary tree in preorder, postorder and inorder.

2017-18, 7 marks

Q.10 Draw a binary tree with following traversals:

Inorder : B C A E G D H F I J

Preorder : A B C D E G F H I J

Q.11 Construct a binary tree for the following preorder and inorder traversa. Explain with a neat diagram:

Preorder: A B D I E H J C F K L G M

Inorder: D I B H J E A F L K C G M

Q.12 Construct an expression tree for the expression

Give pre order inorder and post order traversals of the expression tree so formed.

2017-18, 7 marks

2017-18, 7 Marks

2017-18, 7 marks

Q.13 If the in order traversal of a binary tree is D, J, G, B, A, E, H, C, F, I and its pre order traversal is A, B, D, G, J, C, E, H, F, I Determine the binary tree?

2018-19, 2 marks

Q.14 Define complete binary tree and full binary tree.

2018-19, 2 marks

Q.15 Draw a binary tree for the following traversals:

Preorder : A B C D E F G H I J K L

Postorder : C F E G D B K J L I H A

Q.16 Draw a binary tree for expression : $A * B - (C + D) * (P / Q)$

2018-19, 7 marks

Q.17 Number of nodes in a complete tree is 100000. find its depth.

2018-19, 2 MARKS

Q.18 Construct the binary tree for the following:

Inorder : Q, B, K, C, F, A, G, P, D, E, R, H

Preorder: G, B, Q, A, C, K, F, P, D, E, R, H

Find the post order of the tree.

$$\begin{aligned} \log_2 n &+ 1 \\ \log_2 100000 &\\ \log_2 2^{16} &+ \\ 16 + 1 &= 17 \end{aligned}$$

2018-19, 2 MARKS

2018-19, 7 MARKS

Q.19 Define extended binary tree, full binary tree, strictly binary tree and complete binary tree.

2019-20, 2 marks

Q.20 Can you find a unique tree when any two traversals are given? Using the following traversals construct the corresponding binary tree:

INORDER: H K D B I L E A F C M J G

PREORDER: A B D H K E I L C F G J M

Also find the postorder traversal of obtained tree.

2019-20, 10 marks

Q.21 If the inorder of a binary tree is B, I, D, A, C, G, E, H, F and its post order is I, D, B, G, C, H, F, E, A then draw a corresponding binary tree with neat and clear steps from above assumption. 2020-21, 10 marks

Q.22 Write short note of the following

- (a) Extended binary Tree
- (b) Complete Binary Tree
- (c) Threaded binary Tree

2020-21, 10 marks

Q.23. Define complete binary tree, strictly binary tree and extended binary tree.

2020-21, 2 marks

Q.24. Define Binary tree. In order and post order traversal of a tree are given below:

Preorder: H D I B J E K A F C G

Postorder: A B D H I E J K C F G

Construct the binary tree and determine the postorder traversal of the tree.

2020-21, 7 marks

Q.25. In a complete binary tree if the number of nodes is 1000000. What will be the height of complete binary tree.

2022-23, 2 marks

Q.26. The order of nodes of a binary tree in inorder and postorder traversals are as follows:

Inorder : B I D A C G E H F

Post order: I D B G C H F E A

(i) Draw the corresponding binary tree.

(ii) Write the pre order traversal of the same tree.

2022-23, 10 marks

Today's Target :

- Creation of Binary Tree with Pre and Post Order Traversal
- Applications of Tree Data Structure
- Tree Traversal using Stack (Preorder, Inorder, Postorder)
- Threaded Binary Tree (TBT) – One Way , Two Way Inorder Threading,
Threading with Header Node.
- Difference between Threaded and Normal Binary Tree
- AKTU PYQs**

Pre | Post

Inorder

Q.1 Construct a tree for the following preorder and post order and write its in order traversal.

Preorder : 24, 14, 13, 19, 17, 15, 10, 5, 8, 6, 7, 20

Post order : 13, 15, 17, 10, 19, 14, 7, 6, 20, 8, 5, 24

2014-15 10 marks

Q.2 Draw a binary tree for the following traversals:

Preorder : A B C D E F G H I J K L

Postorder : C F E G D B K J L I H A

2018-19 7 marks

Q.3 Explain threaded binary tree. How it would be useful and efficient in implementing the tree traversal?

2014-15, 10 marks

Q.4 a) Draw a binary tree which has the following traversal-

Inorder : D J G B A E H C F I

Preorder: A B D G J C E H F I

b) Explain threaded binary tree with suitable example.

2016-17, 15 marks

Q.5 What is threaded binary tree? Explain the advantages of using a threaded binary tree.

2017-18 7 marks

Q.6 Write the structure of binary search tree, threaded binary tree.

2017-18 2 marks

Q.7 What is threaded binary tree? Explain two way in order threading with suitable example?

2018-19 7 marks

Q.8 Explain threaded binary tree.

2019-20, 2 marks

Q.9 Explain the following-

(a) Extended binary Tree

2020-21, 10 marks

(b) Complete Binary Tree

(c) Threaded binary Tree

Q.10 Define threaded binary trees. Explain how threaded binary tree can be traversed.

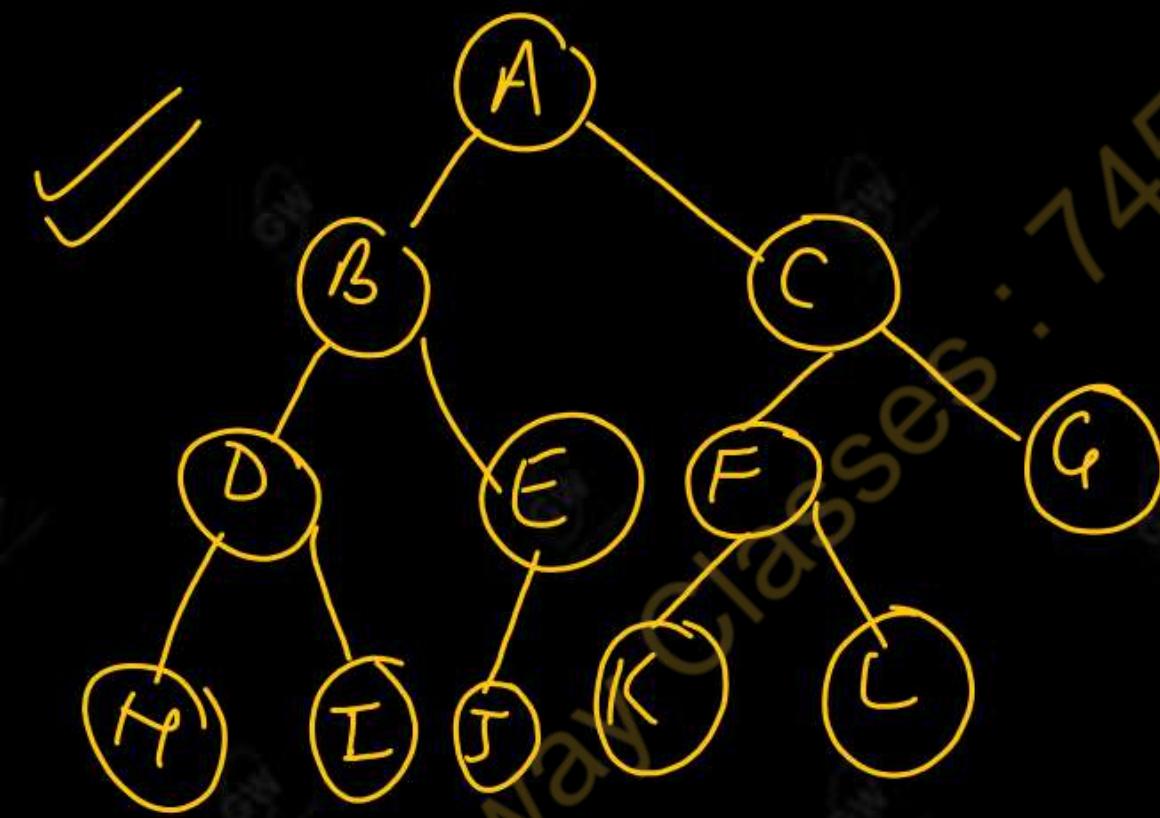
2020-21, 7 marks

Creation of Binary Tree using Preorder and Postorder Traversal

Example 1:- Construct a binary tree for the following preorder and post order and write its in order traversal.

Preorder : A B D H I E J C F K L G (Root , left , Right)
Post order : H I D J E B K L F C G A (left , right , Root)

Solution:-

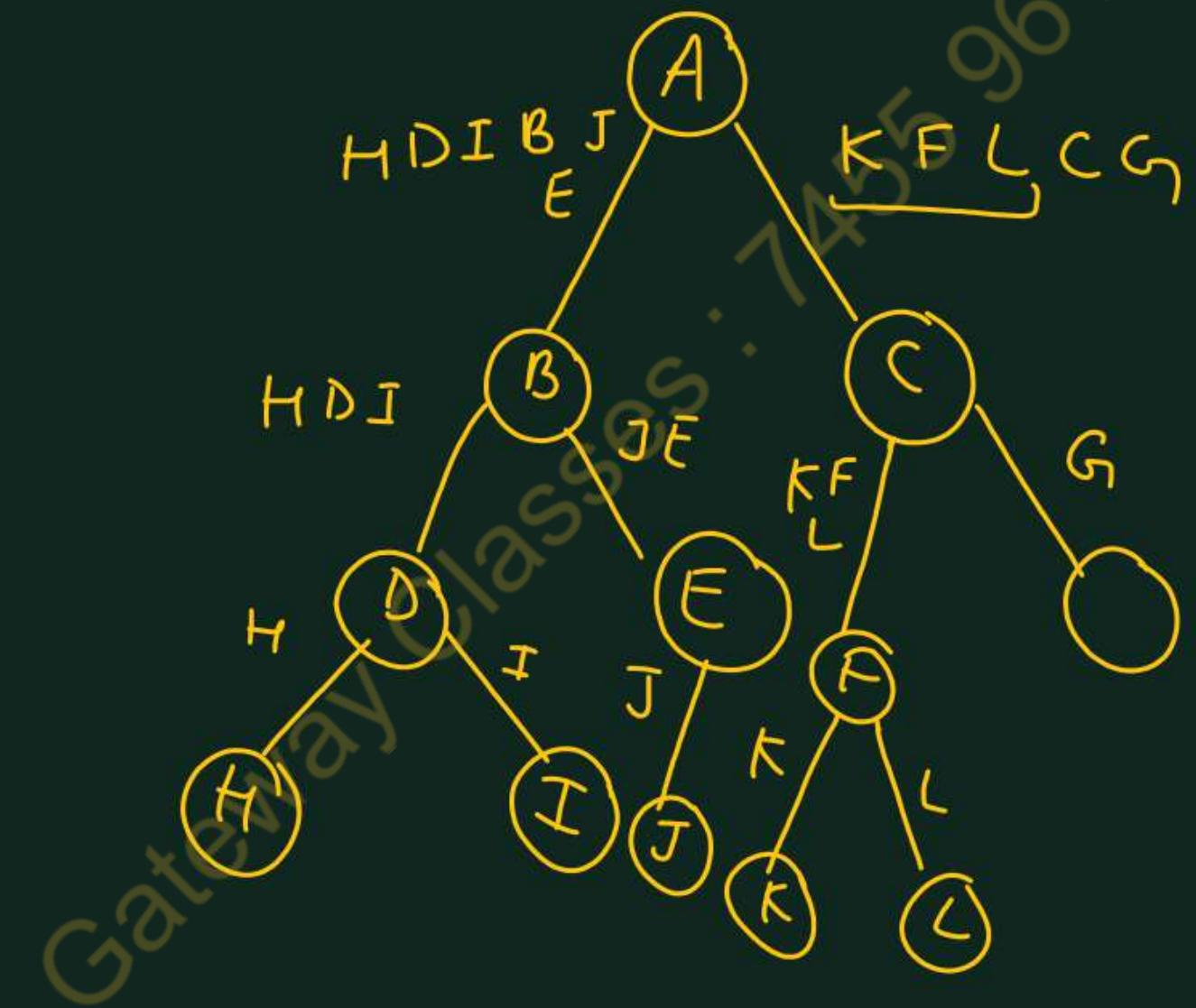


PREORDER :-
A B D H I E J C F K L G

INORDER :-
H D I B J E A K F L C G
left Root right

PREORDER :- A B D H I E J C F K L G

INORDER :- H D I B J E A K F L C G

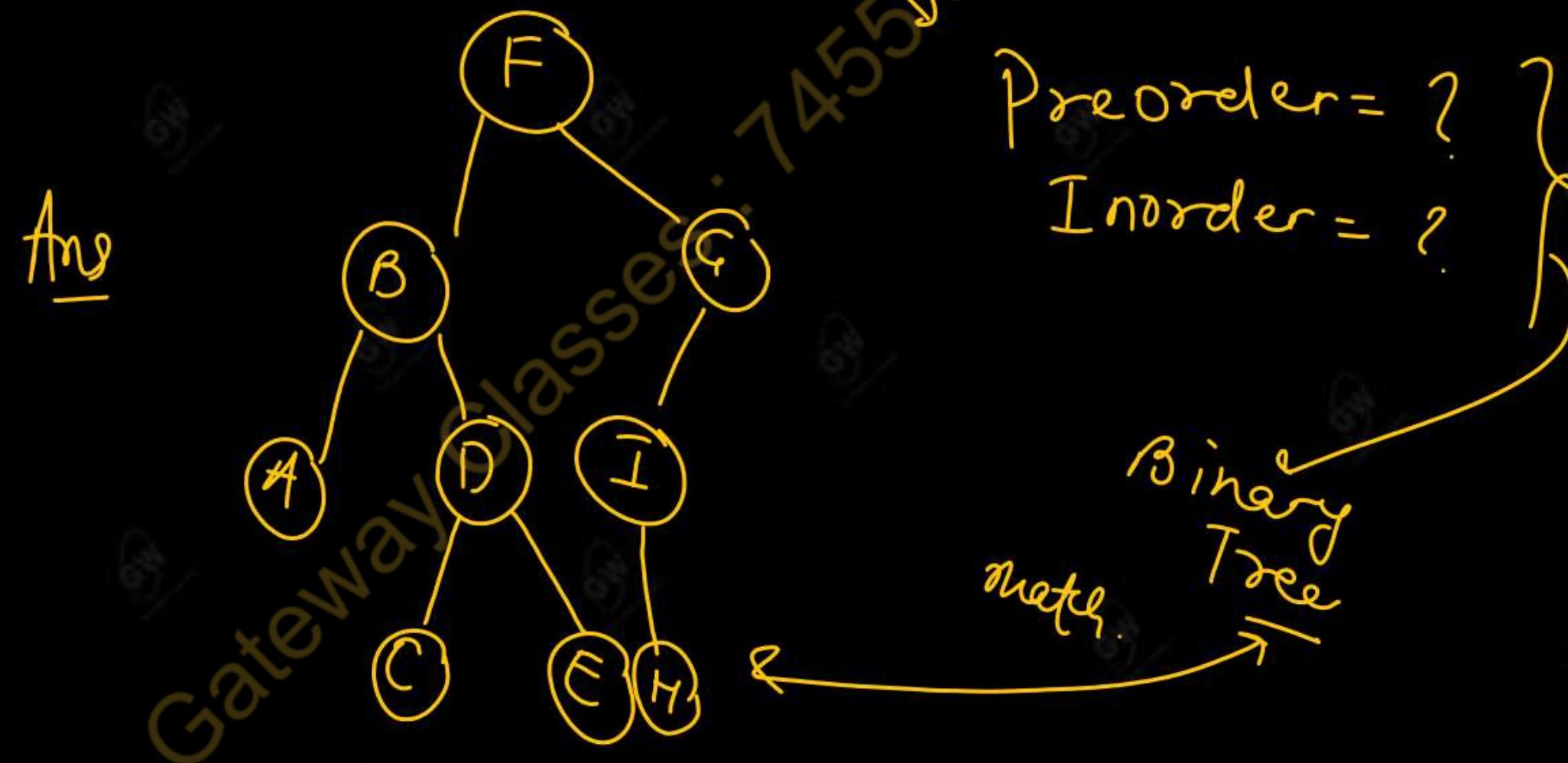


Example 2:- Construct a binary tree for the following preorder and post order and write its in order traversal.

Preorder : F B A D C E G I H

Post order : A C E D B H I G F

Solution:-



Application of Tree Data Structure:

- **File System:** This allows for efficient navigation and organization of files.
- **Data Compression:** ^{Imp.} Huffman coding is a popular technique for data compression that involves constructing a binary tree where the leaves represent characters and their frequency of occurrence. The resulting tree is used to encode the data in a way that minimizes the amount of storage required.
- **Compiler Design:** In compiler design, a syntax tree is used to represent the structure of a program.
- **Database Indexing:** B-trees and other tree structures are used in database indexing to efficiently search for and retrieve data.

B+ Tree
AVL Tree
B-Tree

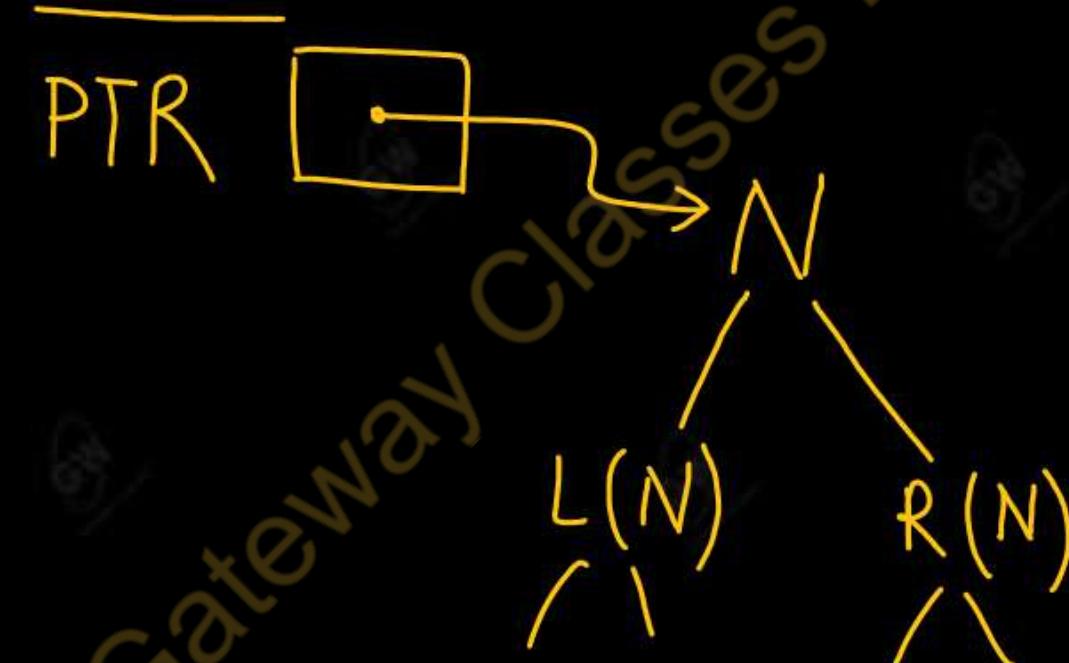
- Suppose a binary tree T is maintained in memory by some linked representation

TREE (INFO , LEFT , RIGHT , ROOT)

- Preorder Traversal :-

(Root , left , Right)

- The preorder traversal algorithm uses a variable PTR (pointer) which will contain the location of the node N currently being scanned.
- This is pictured in this figure, where $L(N)$ denotes the left child of node N and $R(N)$ denotes the right child.



- The algorithm also uses an array STACK, which will hold the addresses of nodes for future processing.

Algorithm: Pre order Traversal

Initially push NULL onto STACK and then set PTR := ROOT.

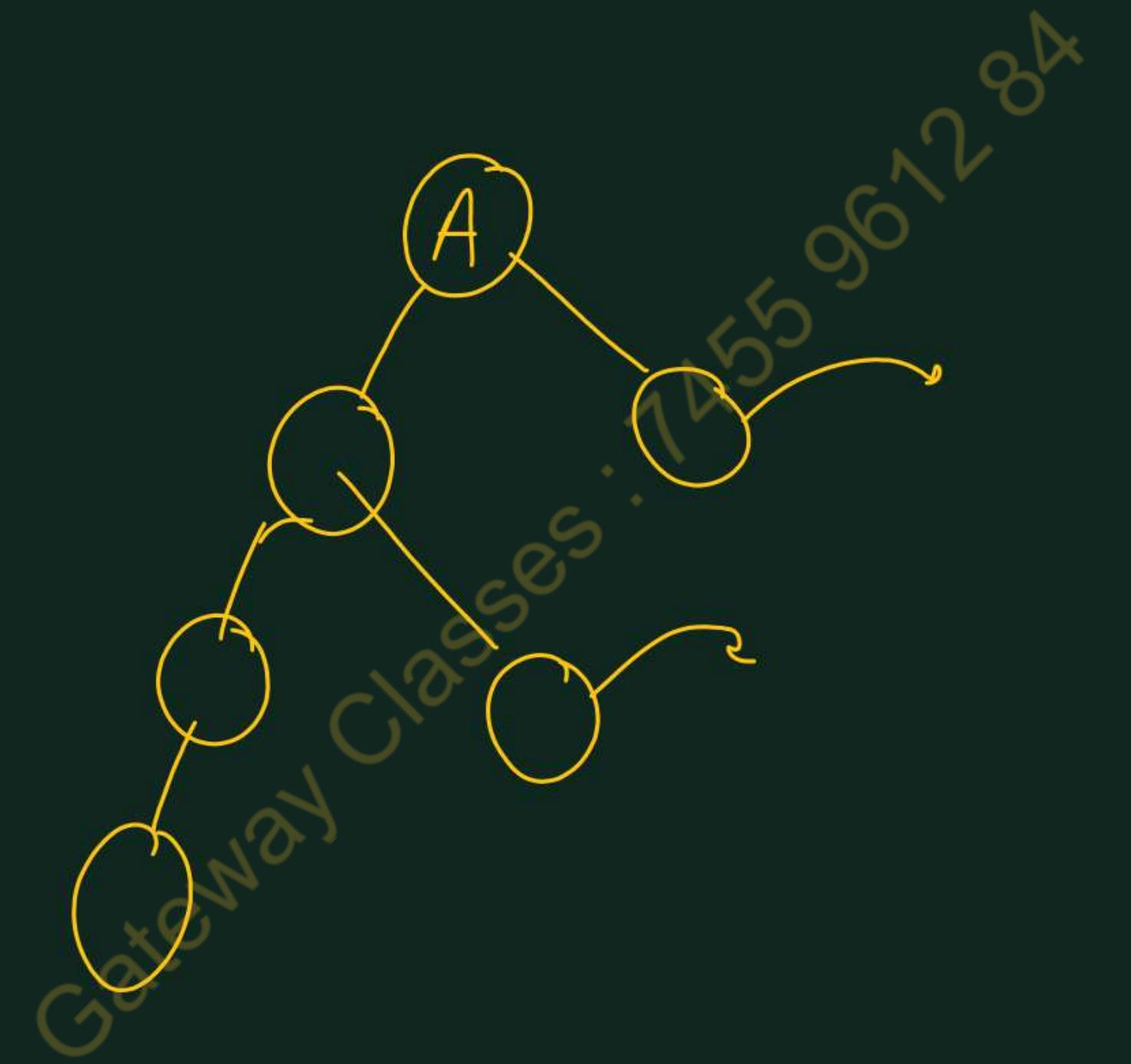
Then repeat the following steps while PTR ≠ NULL.

(a) Proceed down the left - most path rooted at PTR, processing each node N on the path and pushing each right child R (N), if any, onto STACK.

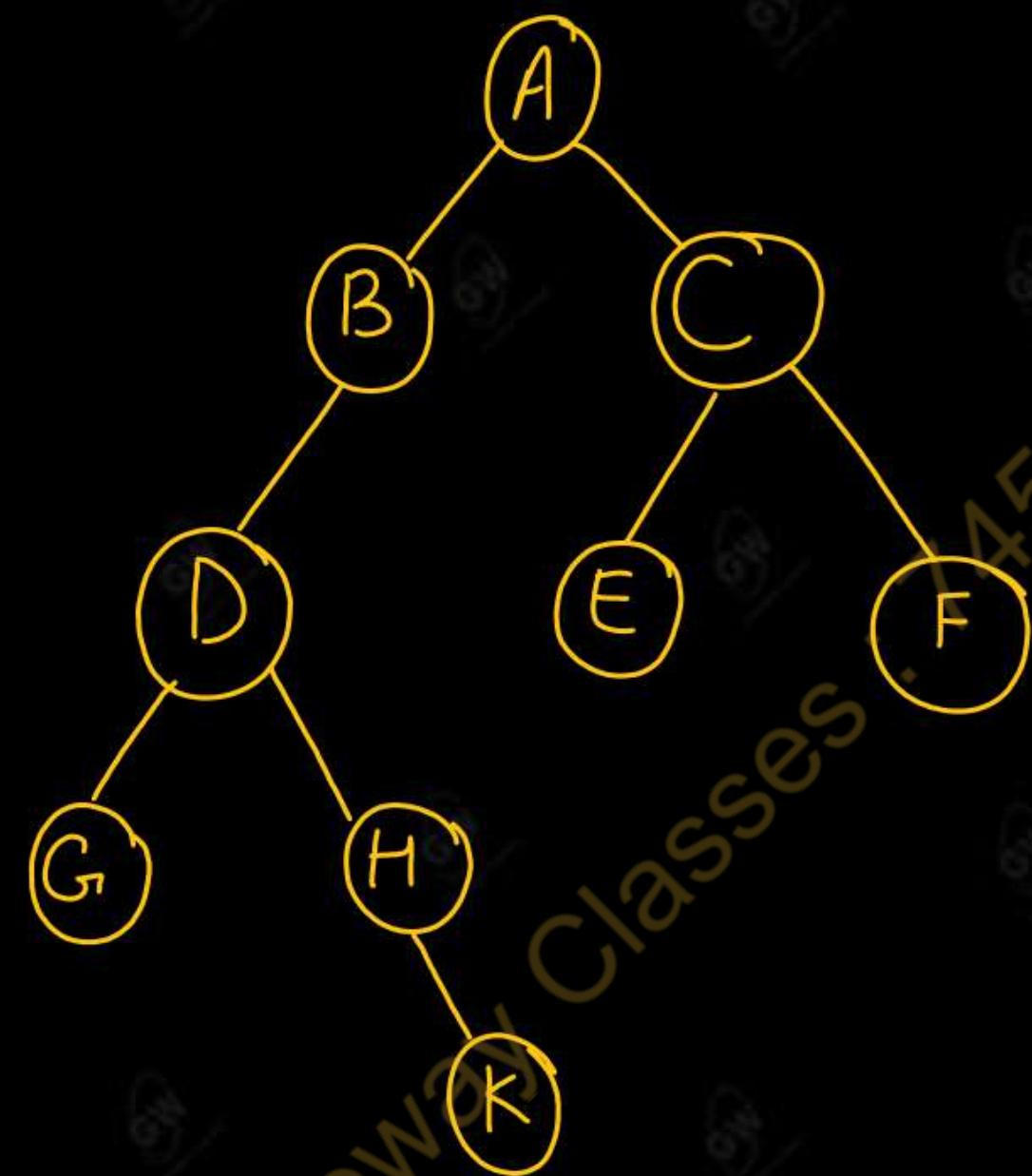
The traversing ends after a node N with no left child L (N) is processed. (Thus PTR is updated using the assignment PTR := LEFT [PTR], and the traversing stops when LEFT [PTR] = NULL)

(b) [Backtracking.] Pop and assign to PTR the top element on STACK. If PTR != NULL, then return to Step (a); otherwise Exit.

(We note that the initial element NULL on STACK is used as a sentinel.)



EXAMPLE 3: Consider the following binary tree T :-



PRE - A B D G H K C E F
455 9612 84

We simulate the above algorithm with T, showing the contents of STACK at each step.

1. Initially push NULL onto STACK:STACK: \emptyset .

$$\text{PTR} = A$$

Then set PTR = A, the root of T.**2. Proceed down the left-most path rooted at PTR = A as follows:**

(i) Process A and push its right child C onto STACK:

STACK: \emptyset, C .(ii) Process B. (There is no right child.)

(iii) Process D and push its right child H onto STACK:

STACK: \emptyset, C, H .(iv) Process G. (There is no right child.)

No other node is processed, since G has no left child.

3 [Backtracking.] Pop the top element H from STACK, and set PTR: = H. This leaves:STACK: \emptyset, C .Since PTR \neq NULL, return to Step (a) of the algorithm.

4. Proceed down the left-most path rooted at PTR = H as follows:

(v) Process H and push its right child K onto STACK:

STACK: \emptyset , C, K.

No other node is processed, since H has no left child.

5. [Backtracking.] Pop K from STACK, and set PTR := K. This leaves:

STACK: \emptyset , C.

Since PTR \neq NULL, return to Step (a) of the algorithm.

6. Proceed down the left-most path rooted at PTR = K as follows:

(vi) Process K. (There is no right child.)

No other node is processed, since K has no left child.

7. [Backtracking.] Pop C from STACK, and set PTR := C. This leaves

STACK: \emptyset .

Since PTR \neq NULL, return to Step (a) of the algorithm.

8. Proceed down the leftmost path rooted at PTR = C as follows:

(vii) Process C and push its right child F onto STACK:

STACK: \emptyset , F.

(viii) Process E. (There is no right child.)

9. [Backtracking.] Pop F from STACK, and set PTR: = F. This leaves:

STACK: \emptyset .

Since PTR \neq NULL, return to Step (a) of the algorithm.

10. Proceed down the left- most path rooted at PTR = F as follows:

(ix) Process F. (There is no right child)

No other node is processed, since F has no left child.

11. [Backtracking] Pop the top element NULL from STACK, and set PTR = NULL.

Since PTR = NULL, the algorithm is completed.

As seen from Steps 2, 4, 6, 8 and 10, the nodes are processed in the order A, B, D, G, H, K, C, E, F.

This is the required preorder traversal of T.

A formal presentation of preorder traversal algorithm is as follows: -

Algorithm : PREORDER (INFO, LEFT, RIGHT, ROOT)

A binary tree T is in memory. This algorithm does a preorder traversal of T applying an operation

PROCESS to each of its nodes. An array STACK is used to temporarily hold the addresses of nodes.

1. [Initially push NULL onto STACK, and initialize PTR.]

Set TOP := 1, STACK [1] := NULL and PTR := ROOT.

2. Repeat Steps 3 to 5 while PTR != NULL:

3. Apply PROCESS to INFO [PTR].

4. [Right Child?]

If RIGHT [PTR] != NULL, then: [Push on STACK]

Set TOP := TOP + 1, and STACK [TOP] := RIGHT [PTR].

[End of If structure.]

5. [Left child?]

If LEFT [PTR] != NULL, then:

Set PTR := LEFT [PTR].

Else: [Pop from STACK.]

Set PTR := STACK [TOP] and TOP := TOP - 1.

[End of If structure.]

[End of Step 2 loop.]

6. Exit.

Inorder Traversal

The inorder traversal algorithm also uses a variable pointer PTR, which will contain the location of the node N currently being scanned, and an array STACK, which will hold the addresses of nodes for future processing. In fact, with this algorithm, a node is processed only when it is popped from STACK.

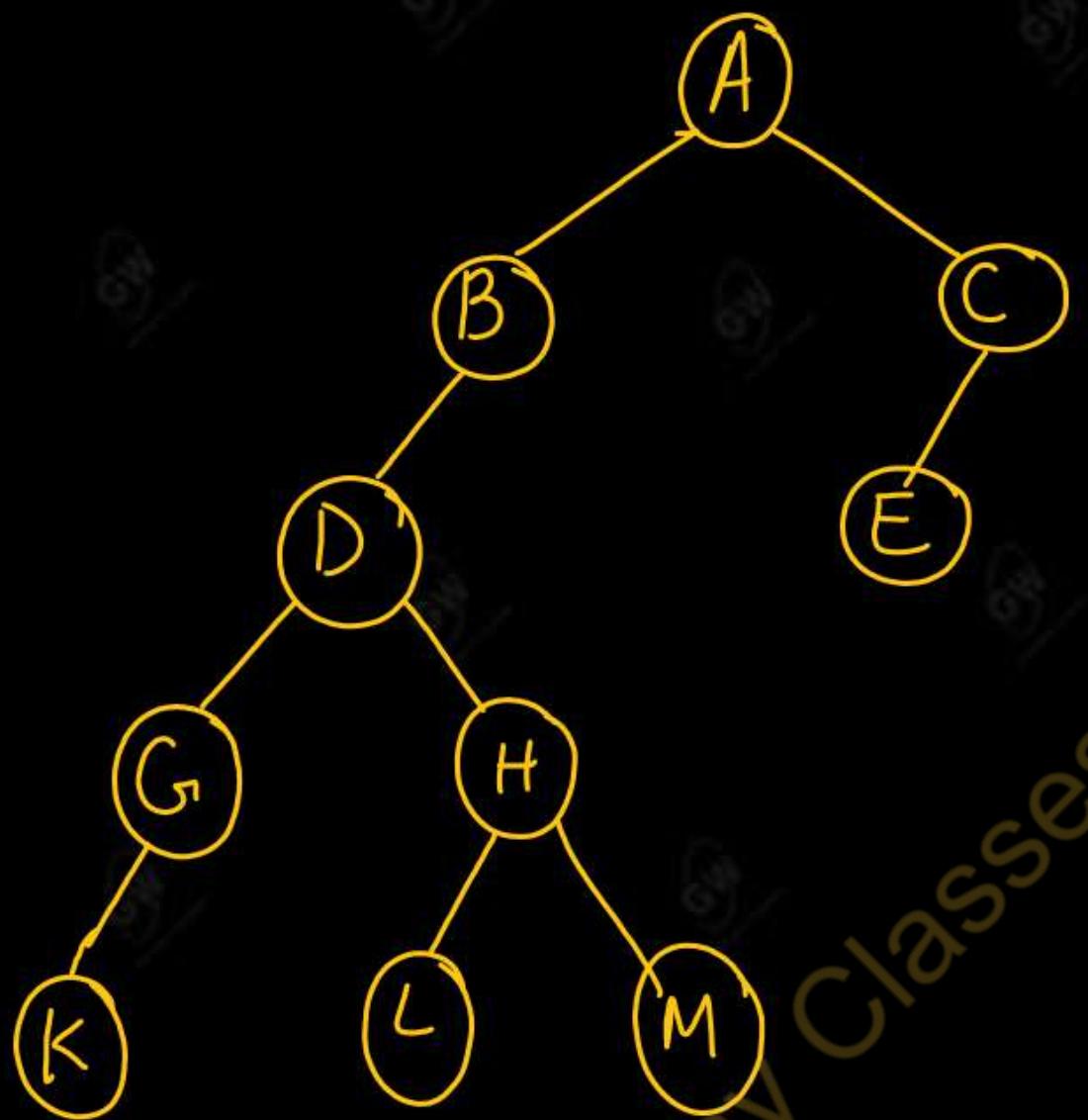
Algorithm: Initially push NULL onto STACK (for a sentinel) and then set PTR : =ROOT.

Then repeat the following steps until NULL is popped from STACK.

(a) Proceed down the left- most path rooted at PTR, pushing each node N onto STACK and stopping when a node N with no left child is pushed onto STACK.

(b) [Backtracking.] Pop and process the nodes on STACK. If NULL is popped, then Exit. If a node N with a right child R (N) is processed, set PTR = R (N) (by assigning PTR := RIGHT [PTR]) and return to Step (a).

We emphasize that a node N is processed only when it is popped from STACK.

EXAMPLE 4:- Consider the following binary tree T :

We simulate the above algorithm with T, showing the contents of STACK.

1. Initially push NULL onto STACK:STACK: \emptyset

Then set PTR: A, the root of T.

2. Proceed down the left-most path rooted at PTR = A, pushing the nodes A, B, D, G and K onto stack:STACK: \emptyset, A, B, D, G, K .

(No other node is pushed onto STACK, since K has no left child.)

3. [Backtracking.] The nodes K, G and D are popped and processed, leaving:STACK: \emptyset, A, B .

(We stop the processing at D, since D has a right child.) Then set PTR := H, the right child of D.

4. Proceed down the left-most path rooted at PTR = H, pushing the nodes H and L onto STACK:STACK: \emptyset, A, B, H, L .

(No other node is pushed onto STACK, since L has no left child.)

5. [Backtracking.] The nodes L and H are popped and processed, leaving:

STACK: \emptyset , A, B.

(We stop the processing at H, since H has a right child.) Then set PTR := M, the right child of H.

6. Proceed down the left-most path rooted at PTR = M, pushing node M onto STACK:

STACK: \emptyset , A, B, M.

(No other node is pushed onto STACK, since M has no left child.)

7. [Backtracking.] The nodes M, B and A are popped and processed, leaving:

STACK: \emptyset .

(No other element of STACK is popped, since A does have a right child.) Set PTR := C, the right child of A.

8. Proceed down the left-most path rooted at PTR = C, pushing the nodes C and E onto STACK:

STACK: \emptyset , C, E.

9. [Backtracking.] Node E is popped and processed. Since E has no right child, node C is popped and processed. Since C has no right child, the next element, NULL, is popped from STACK.

The algorithm is now finished, since NULL is popped from STACK.

As seen from Steps 3, 5, 7 and 9, the nodes are processed in the order K, G, D, L, H, M, B, A, E, C.

This is the required inorder traversal of the binary tree T.

A formal presentation of our inorder traversal algorithm is given in the next slide.

Algorithm : INORDER (INFO, LEFT, RIGHT, ROOT)

A binary tree is in memory. This algorithm does an inorder traversal of T, applying an operation **PROCESS** to each of its nodes. An array **STACK** is used to temporarily hold the addresses of nodes.

1. [Push NULL onto STACK and initialize PTR.]

Set TOP := 1, STACK [1] := NULL and PTR := ROOT.

2. Repeat while PTR != NULL : [Pushes left-most path onto STACK.]

(a) Set TOP := TOP + 1 and STACK [TOP] := PTR. [Save s node.]

(b) Set PTR:= LEFT [PTR]. [Updates PTR.]

[End of loop.]

3. Set PTR := STACK [TOP] and TOP: TOP - 1. [Pops node from STACK.]

4. Repeat Steps 5 to 7 while PTR NULL: [Backtracking.]

5. Apply PROCESS to INFO [PTR].

6. [Right child?] If RIGHT [PTR] != NULL, then:

(a) Set PTR := RIGHT [PTR].

(b) Go to Step 3.

[End of If structure.]

7. Set PTR STACK [TOP] and TOP := TOP - 1. [Pops node.]

[End of Step 4 loop.]

8. Exit.

Postorder Traversal

(left , light , Root)

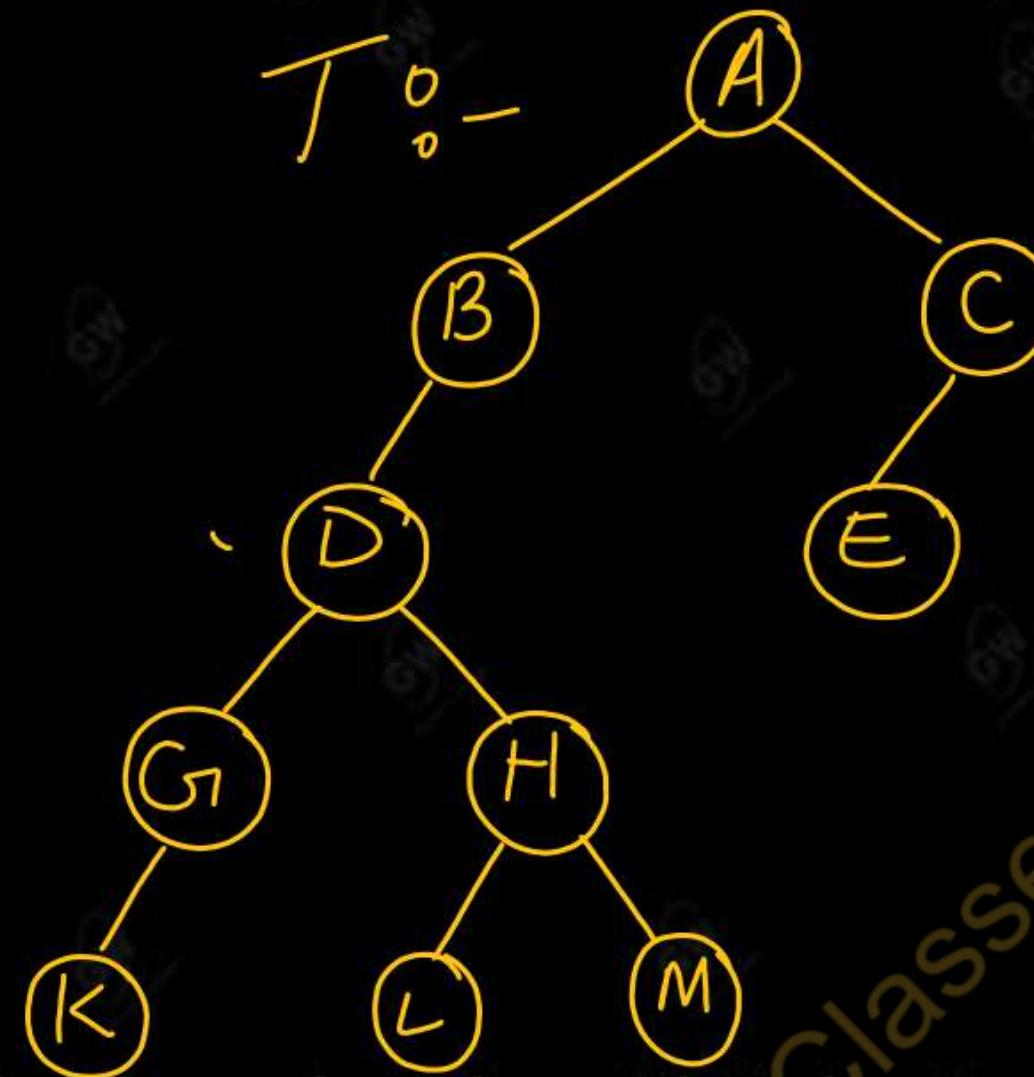
The postorder traversal algorithm is more complicated than the preceding two algorithms, because here we may have to save a node N in two different situations. We distinguish between the two cases by pushing either N or its negative, - N, onto STACK. Again, a variable PTR (pointer) is used which contains the location of the node N that is currently being scanned.

Algorithm: Initially push NULL onto STACK (as a sentinel) and then set PTR := ROOT. Then repeat the following steps until NULL is popped from STACK.

(a) Proceed down the left-most path rooted at PTR. At each node N of the path, push N onto STACK and, if N has a right child R(N), push -R(N) onto STACK.

(b) [Backtracking.] Pop and process positive nodes on STACK. If NULL is popped, then Exit. If a negative node is popped, that is, if PTR = - N for some node N, set PTR = N (by assigning PTR := -PTR) and return to Step (a).

We emphasize that a node N is processed only when it is popped from STACK and it is positive.

EXAMPLE 5:- Consider again the following binary tree:

left Right Root

POSTORDER TRAVERSAL :-

K G L M H D E C A

We simulate the above algorithm with T , showing the contents of STACK.

1. Initially, push NULL onto STACK and set PTR A, the root of T : STACK: \emptyset .
2. Proceed down the left-most path rooted at PTR=A, pushing the nodes A, B, D, G and K onto STACK.
Furthermore, since A has a right child C, push -C onto STACK after A but before B, and since D has a right child H, push -H onto STACK after D but before G. This yields: STACK: $\emptyset, A, -C, B, D, -H, G, K$.

3. [Backtracking.] Pop and process K, and pop and process G. Since -H is negative, only pop -H.

This leaves:

STACK: $\emptyset, A, -C, B, D$

Now PTR = -H. Reset PTR = H and return to Step (a),

4. Proceed down the left-most path rooted at PTR = H. First push H onto STACK. Since H has a right child

M, push -M onto STACK after H. Last, push L onto STACK. This gives: STACK: $\emptyset, A, -C, B, D, H, -M, L$

5. [Backtracking.] Pop and process L, but only pop -M. This leaves: STACK: $\emptyset, A, -C, B, D, H$

Now PTR = - M. Reset PTR = M and return to Step (a).

6. Proceed down the left-most path rooted at PTR = M. Now, only M is pushed onto STACK. This yields:

STACK: $\emptyset, A, - C, B, D, H, M$

7. [Backtracking.] Pop and process M, H, D and B, but only pop -C. This leaves:

STACK: \emptyset, A

Now PTR = -C. Reset PTR = C, and return to Step (a).

8. Proceed down the left-most path rooted at PTR = C. First C is pushed onto STACK and then E, yielding:

STACK: \emptyset, A, C, E

9. [Backtracking.] Pop and process E, C and A. When NULL is popped, STACK is empty and the algorithm is completed.

As seen from Steps 3, 5, 7 and 9, the nodes are processed in the order K, G, L, M, H, D, B, E, C, A.

This is the required postorder traversal of the binary tree T.

A formal presentation of our postorder traversal algorithm is shown in the next slide:

Algorithm : POSTORD (INFO, LEFT, RIGHT, ROOT)

A binary tree T is in memory. This algorithm does a postorder traversal of T, applying an operation **PROCESS** to each of its nodes. An array **STACK** is used to temporarily hold the addresses of nodes.

1. [Push NULL onto STACK and initialize PTR.]

Set TOP := 1, STACK [1] := NULL and PTR := ROOT.

2. [Push left-most path onto STACK.]

Repeat Steps 3 to 5 while PTR != NULL:

3. Set TOP := TOP + 1 and STACK [TOP]:= PTR.

[Pushes PTR on STACK.]

4. If RIGHT [PTR] != NULL, then: [Push on STACK.]

Set TOP := TOP + 1 and STACK [TOP]:= - RIGHT [PTR]

[End of If structure.]

5. Set PTR = LEFT [PTR]. [Updates pointer PTR.]

[End of Step 2 loop.]

6. Set PTR = STACK [TOP] and TOP := TOP - 1.

[Pops node from STACK.]

7. Repeat while PTR > 0:

(a) Apply PROCESS to INFO [PTR].

(b) Set PTR := STACK [TOP] and TOP := TOP-1.

[Pops node from STACK.]

[End of loop.]

8. If PTR < 0, then:

(a) Set PTR := - PTR.

(b) Go to Step 2.

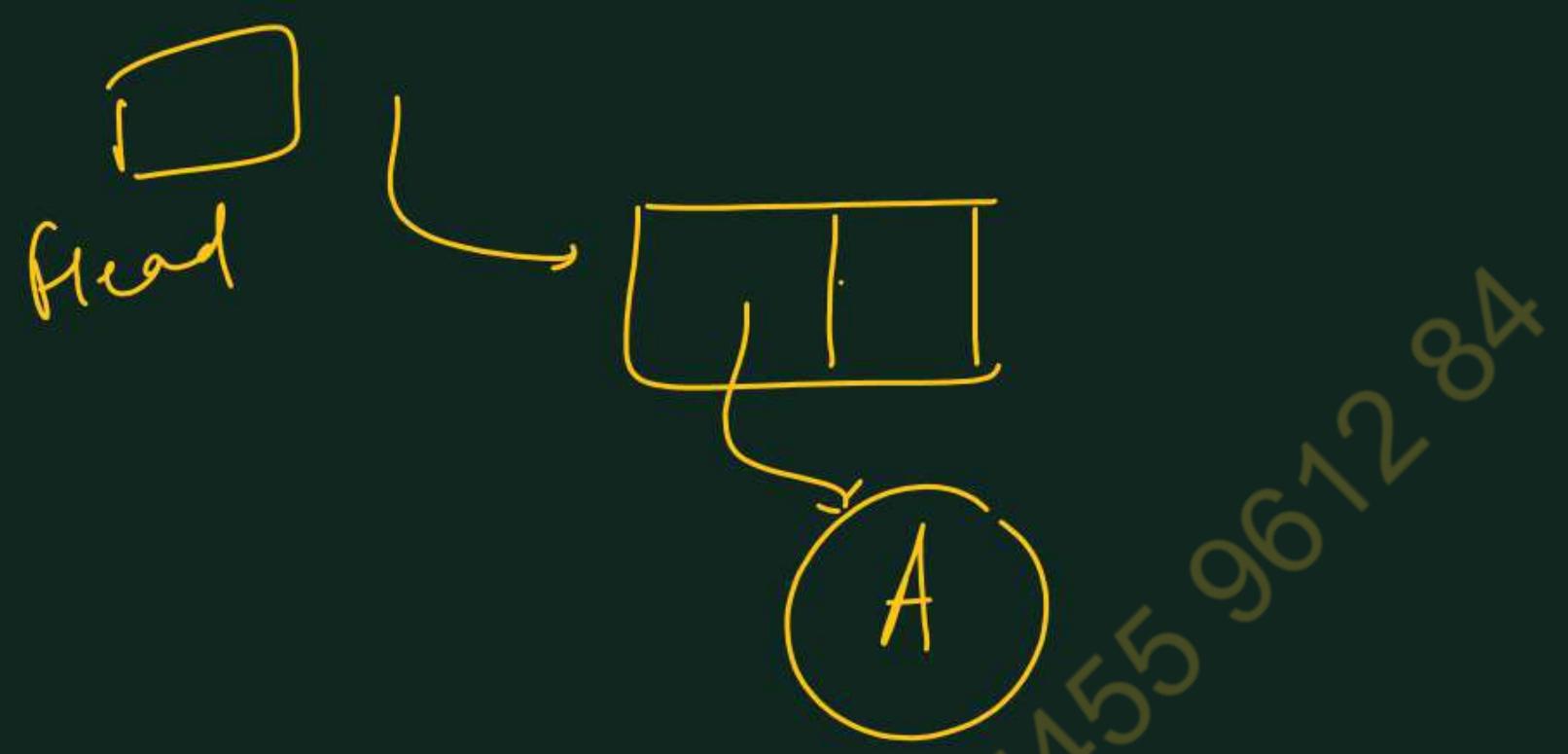
[End of If structure.]

9. Exit.

- Consider a binary tree T.
- Variations of the linked representation of T are frequently used because certain operations on T are easier to implement by using the modifications.
- Some of these variations, which are analogous to header and circular linked lists, are discussed in this section.

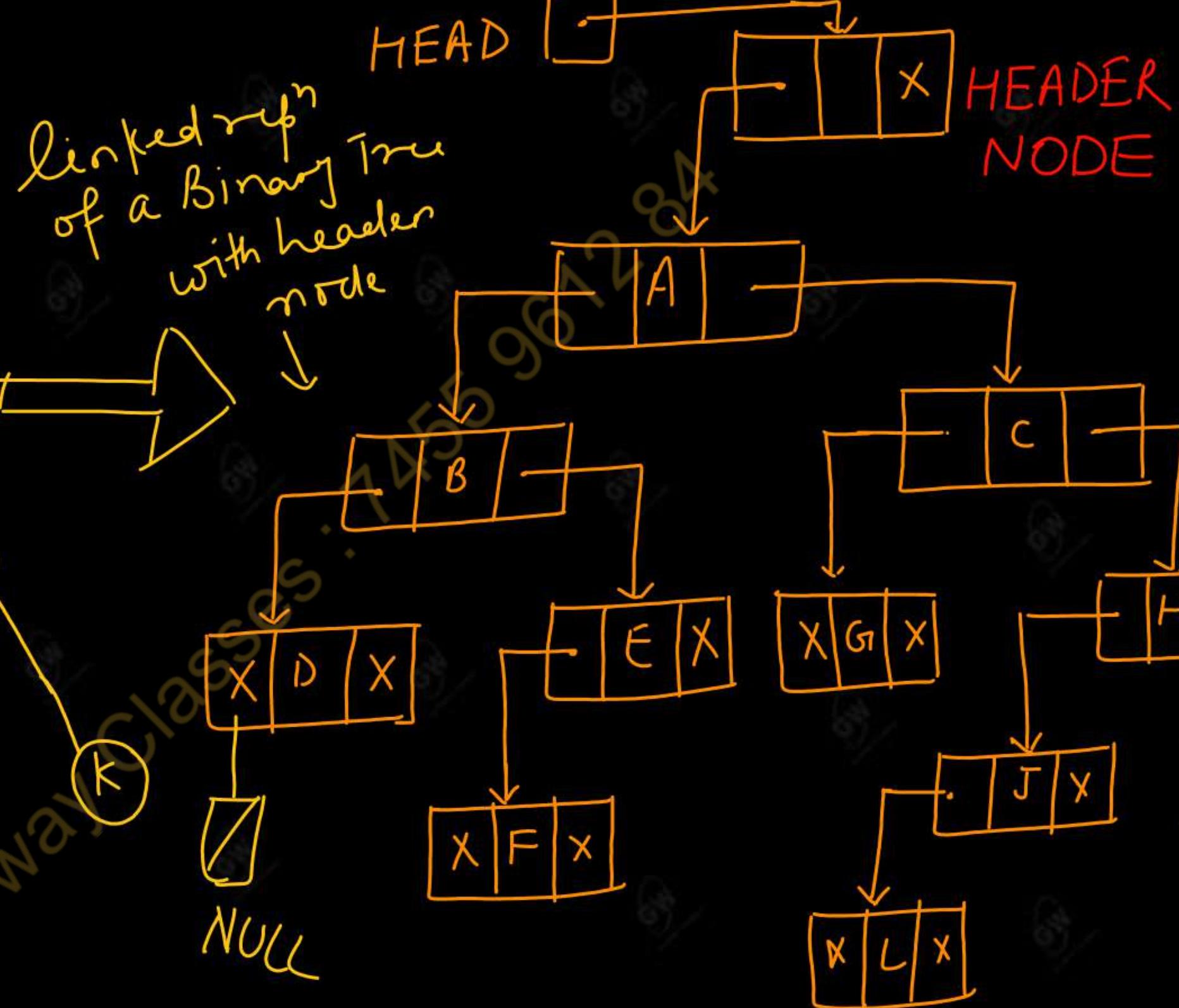
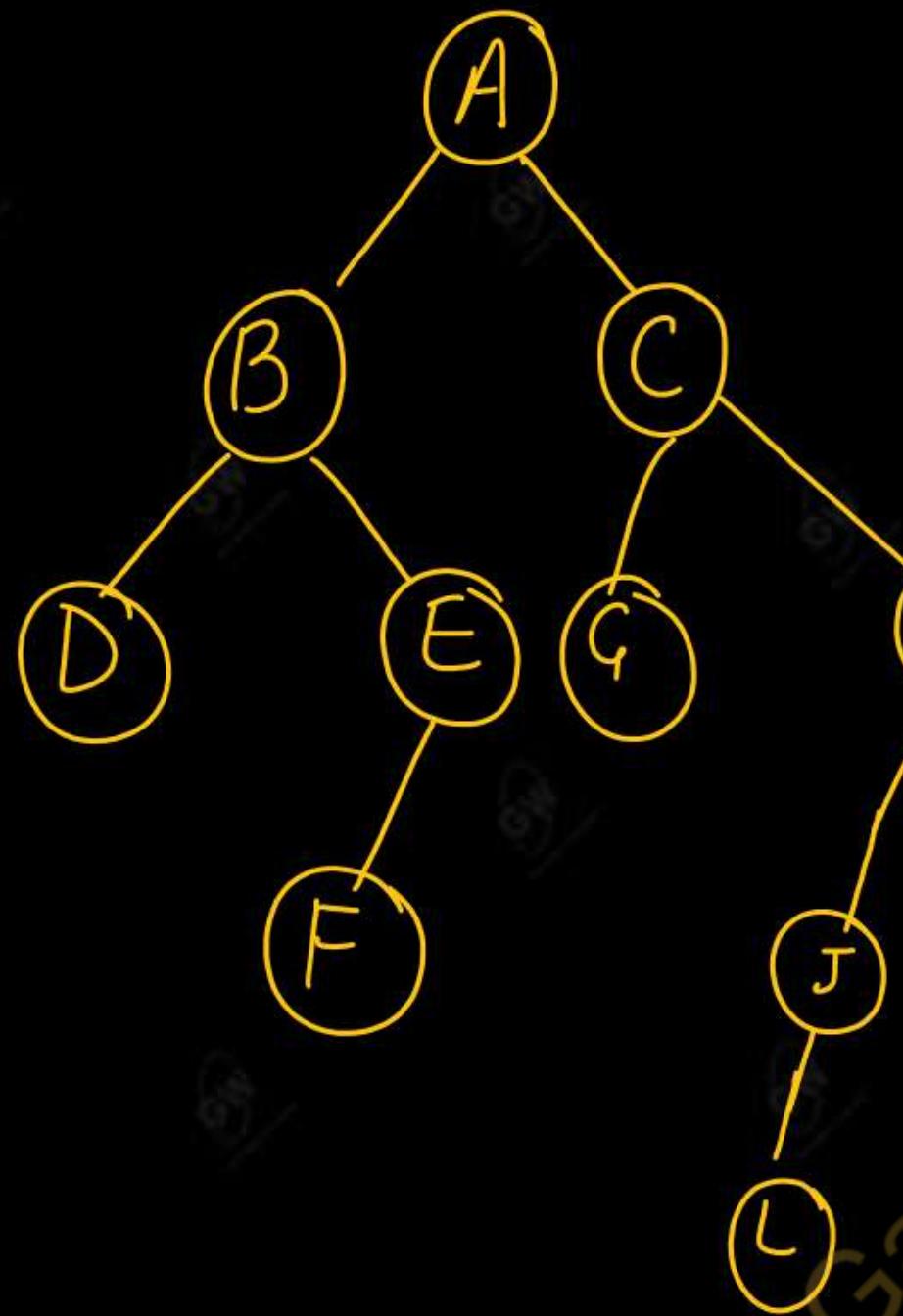
Header Nodes

- Suppose a binary tree T is maintained in memory by means of a linked representation.
- Sometimes an extra, special node, called a header node, is added to the beginning of T.
- When this extra node is used, the tree pointer variable, which we will call HEAD (instead of ROOT), will point to the header node, and the left pointer of the header node will point to the root of T. Consider the following binary tree in the next slide: -



Gateway Classes : 7455 9612 84

Binary Tree



- Following figure shows a schematic picture of the previous binary tree that uses a linked representation with a header node.

- Suppose a binary tree T is empty.
- Then T will still contain a header node, but the left pointer of the header node will contain the null value.
- Thus the condition LEFT [HEAD] = NULL will indicate an empty tree.

- Another variation of the above representation of a binary tree T is to use the header node as a sentinel.
- That is, if a node has an empty subtree, then the pointer field for the subtree will contain the address of the header node instead of the null value.
- Accordingly, no pointer will ever contain an invalid address, and the condition $\text{LEFT}[\text{HEAD}] = \text{HEAD}$ will indicate an empty subtree.

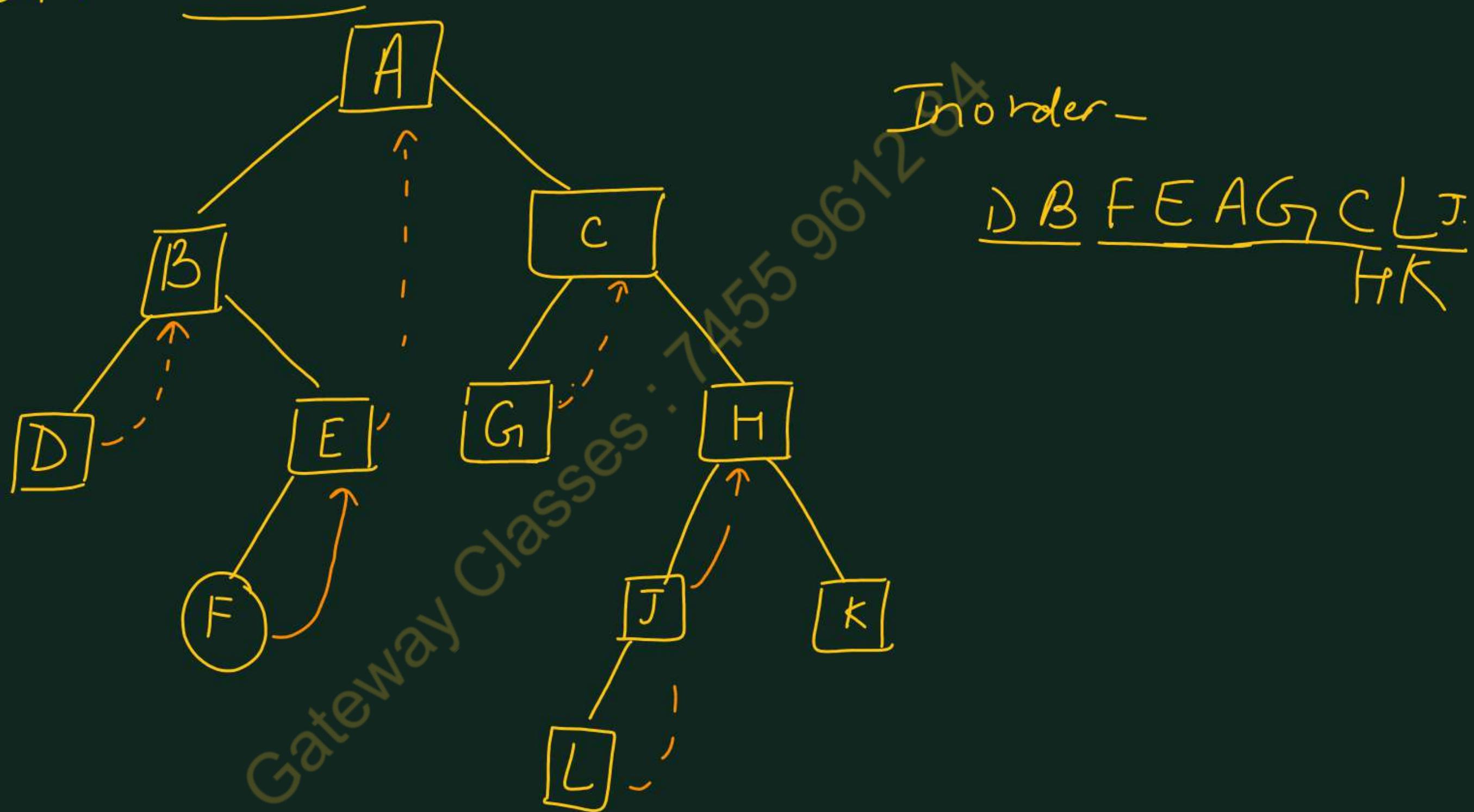
Gateway Classes : 7453 6128

Threads; Inorder Threading

- Consider again the linked representation of a binary tree T.
- Approximately half of the entries in the pointer fields LEFT and RIGHT will contain null elements.
- This space may be more efficiently used by replacing the null entries by some other type of information.
- Specifically, we will replace certain null entries by special pointers which point to nodes higher in the tree.
- These special pointers are called **threads**, and binary trees with such pointers are called threaded trees.

- The threads in a threaded tree must be distinguished in some way from ordinary pointers.
- The threads in a diagram of a threaded tree are usually indicated by dotted lines.
- In computer memory, an extra 1 - bit TAG field may be used to distinguish threads from ordinary pointers, or, alternatively, threads may be denoted by negative integers when ordinary pointers are denoted by positive integers.
- There are many ways to thread a binary tree T, but each threading will correspond to a particular traversal of T.
- Also, one may choose a one-way threading or a two-way threading.

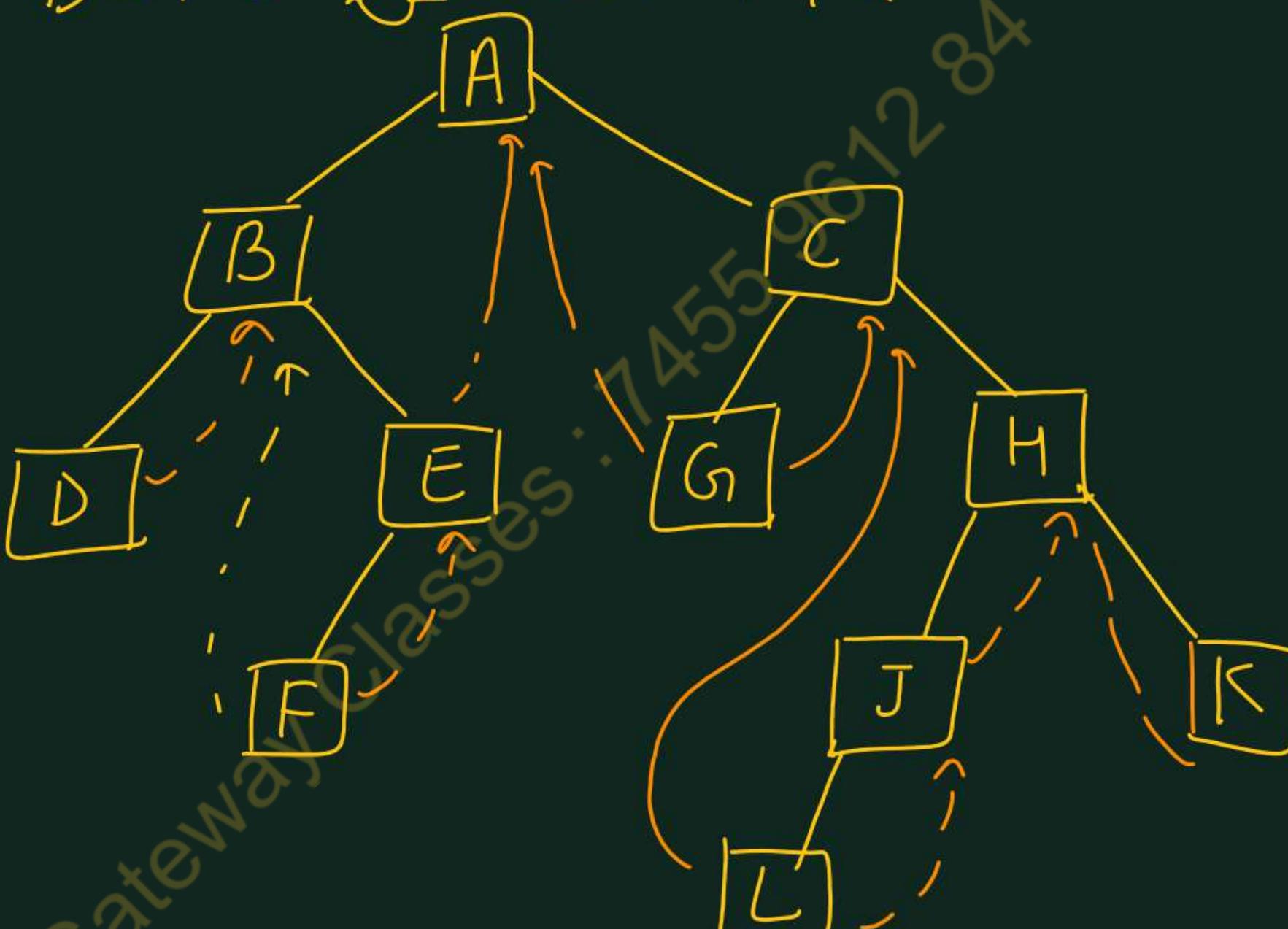
① ONE-WAY INORDER THREADING



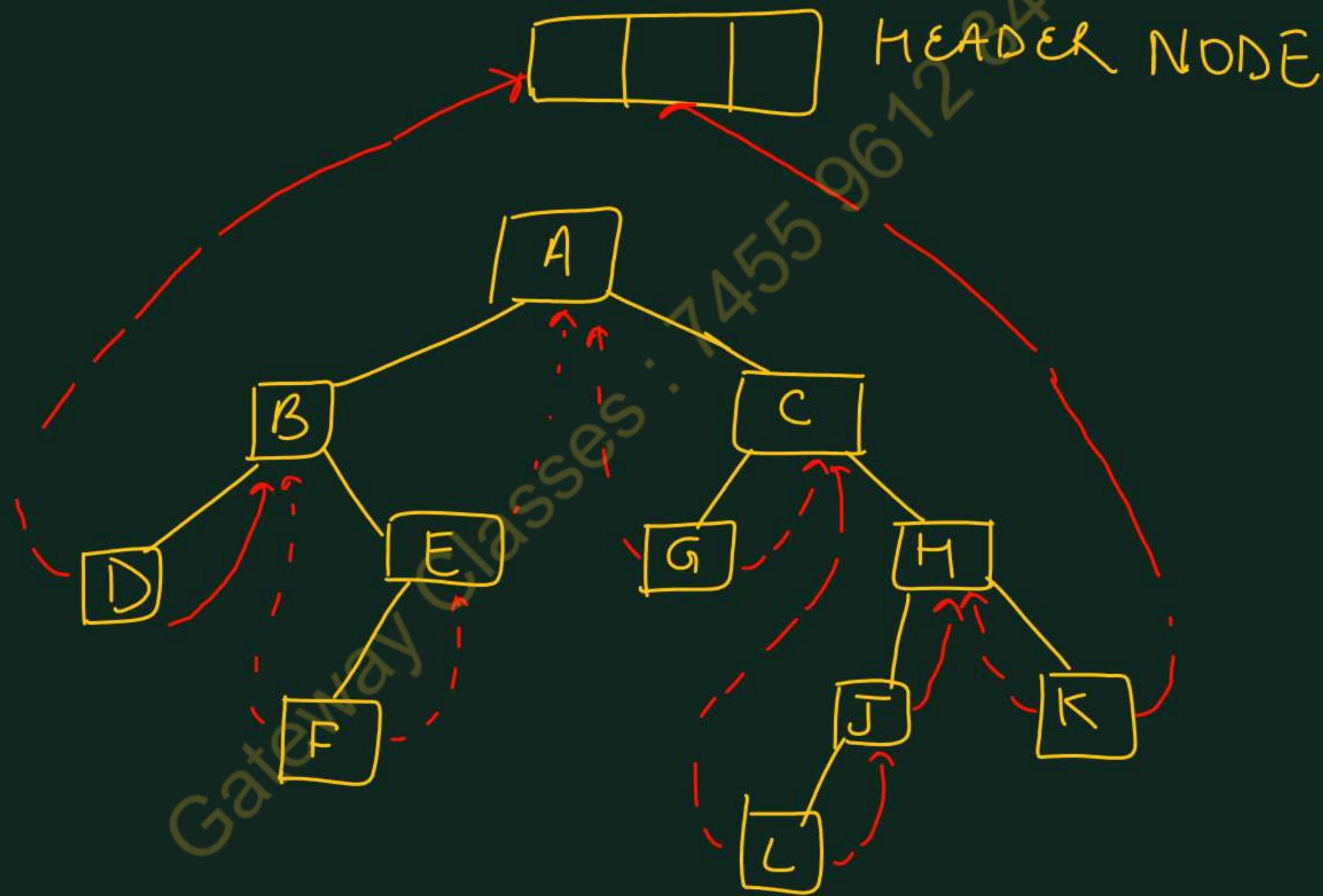
②

TWO - WAY INORDER THREADING

Inorder - D B F E A G C L J H K



③ TWO WAY THREADING WITH HEADER NODE



- Unless otherwise stated, our threading will correspond to the inorder traversal of T.
- Accordingly, in the one-way threading of T, a thread will appear in the right field of a node and will point to the next node in the inorder traversal of T; and in the two-way threading of T, a thread will also appear in the LEFT field of a node and will point to the preceding node in the inorder traversal of T.
- Furthermore, the left pointer of the first node and the right pointer of the last node (in the inorder traversal of T) will contain the null value when T does not have a header node, but will point to the header node when T does have a header node.
- There is an analogous one-way threading of a binary tree T which corresponds to the preorder traversal of T.
- On the other hand, there is no threading of T which corresponds to the postorder traversal of T.

Q How Threaded Binary Tree would be useful and efficient in implementing the tree traversal?

- A threaded binary tree is a binary tree with additional pointers, called threads, that link nodes in a specific way to facilitate more efficient tree traversal.
- As we know in a threaded binary tree, these threads can be of two types: -
- Inorder Threaded Binary Tree: In an inorder threaded binary tree, threads are added to the tree in such a way that they represent the inorder traversal sequence of the tree.
- Specifically, for each node, its right child points to its in-order successor, and its left child points to its in-order predecessor.
- Preorder Threaded Binary Tree: In a preorder threaded binary tree, threads are added to represent the preorder traversal sequence of the tree.
- The right child of each node points to its preorder successor, and the left child points to its preorder predecessor.

- The use of threaded binary trees in implementing tree traversal provides several advantages:
- Efficient Traversal :- Threaded binary trees can significantly speed up tree traversal operations. In a threaded binary tree, you can traverse the tree without the need for recursion or an explicit stack, which can be more efficient in terms of both time and space complexity.
- No Need for Stacks or Recursion: Traditional recursive or stack-based tree traversal methods use additional memory to maintain the call stack. In threaded binary trees, threads eliminate the need for such auxiliary data structures, resulting in more efficient traversal algorithms.
- Simplified Implementation: Tree traversal algorithms become simpler to implement in threaded binary trees. Without the need for recursive function calls or an explicit stack, the code becomes more straightforward and easier to understand.

- Reduced Function Call Overhead: Traditional tree traversal using recursion involves function call overhead, which can impact performance. In threaded binary trees, the traversal process is streamlined, reducing the overhead associated with function calls.
- Improved Space Efficiency: Threaded binary trees can be more space-efficient than traditional binary trees for certain traversal scenarios. The elimination of the need for an explicit stack or recursion can result in reduced memory requirements.
- Support for Reverse Traversal: In addition to forward traversals, threaded binary trees also support efficient reverse traversals without the need for additional pointers or structures.
- It's important to note that the efficiency gains of threaded binary trees in traversal operations come with some trade-offs. Insertion and deletion operations can be more complex and may require additional checks and modifications to maintain the threading.
- The choice to use threaded binary trees depends on the specific requirements of the application and the balance between traversal efficiency and the complexity of modification operations.

Advantages of Threaded Binary Tree

- In this Tree, it enables linear traversal of elements.
- It eliminates the use of stack as it performs linear traversal, so saves memory.
- Enables to find parent node without explicit use of parent pointer
- Threaded trees give forward and backward traversal of nodes by in-order fashion
- Nodes contain pointers to in-order predecessor and successor
- For a given node, we can easily find inorder predecessor and successor.
- So, searching is much more easier.
- In threaded binary tree there is no NULL pointer present. Hence memory wastage in occupying NULL links is avoided.
- The threads are pointing to successor and predecessor nodes. This makes us to obtain predecessor and successor node of any node quickly.
- There is no need of stack while traversing the tree, because using thread links we can reach to previously visited nodes.

Disadvantages of Threaded Binary Tree

- Every node in threaded binary tree need extra information (extra memory) to indicate whether its left or right node indicated its child nodes or its inorder predecessor or successor. So, the node consumes extra memory to implement.
- Insertion and deletion are way more complex and time consuming than the normal one since both threads and ordinary links need to be maintained.
- Implementing threads for every possible node is complicated.
- Increased complexity: Implementing a threaded binary tree requires more complex algorithms and data structures than a regular binary tree. This can make the code harder to read and debug.
- Extra memory usage: In some cases, the additional pointers used to thread the tree can use up more memory than a regular binary tree. This is especially true if the tree is not fully balanced, as threading a skewed tree can result in a large number of additional pointers.

Difference Between Threaded and Normal Binary Tree

S. No.	Threaded Binary Tree	Binary Tree without Threads
1.	In threaded binary trees, the NULL pointers are used as threads.	The Null pointers remains NULL.
2.	We can use NULL pointers which is a efficient way of use computers memory.	We can not use NULL pointers so it is a wastage of memory.
3	Traversal is easy and completed without using stack or recursive functions.	Traverse is not easy and not memory efficient.
4.	Structure is complex.	Less complex than threaded binary tree.
5.	Insertion and deletion takes more time.	Less time consuming than threaded binary tree.

Q.1 Construct a tree for the following preorder and post order and write its in order traversal.

Preorder : 24, 14, 13, 19, 17, 15, 10, 5, 8, 6, 7, 20

Post order : 13, 15, 17, 10, 19, 14, 7, 6, 20, 8, 5, 24

2014-15 10 marks

Q.2 Draw a binary tree for the following traversals:

Preorder : A B C D E F G H I J K L

Postorder : C F E G D B K J L I H A

2018-19 7 marks

Q.3 Explain threaded binary tree. How it would be useful and efficient in implementing the tree traversal?

2014-15, 10 marks

Q.4 a) Draw a binary tree which has the following traversal-

Inorder : D J G B A E H C F I

Preorder: A B D G J C E H F I

b) Explain threaded binary tree with suitable example.

2016-17, 15 marks

Q.5 What is threaded binary tree? Explain the advantages of using a threaded binary tree.

2017-18 7 marks

Q.6 Write the structure of binary search tree, threaded binary tree.

2017-18 2 marks

Q.7 What is threaded binary tree? Explain two way in order threading with suitable example?

2018-19 7 marks

Q.8 Explain threaded binary tree.

2019-20, 2 marks

Q.9 Explain the following-

(a) Extended binary Tree

(b) Complete Binary Tree

(c) Threaded binary Tree

2020-21, 10 marks

Q.10 Define threaded binary trees. Explain how threaded binary tree can be traversed.

2020-21, 7 marks

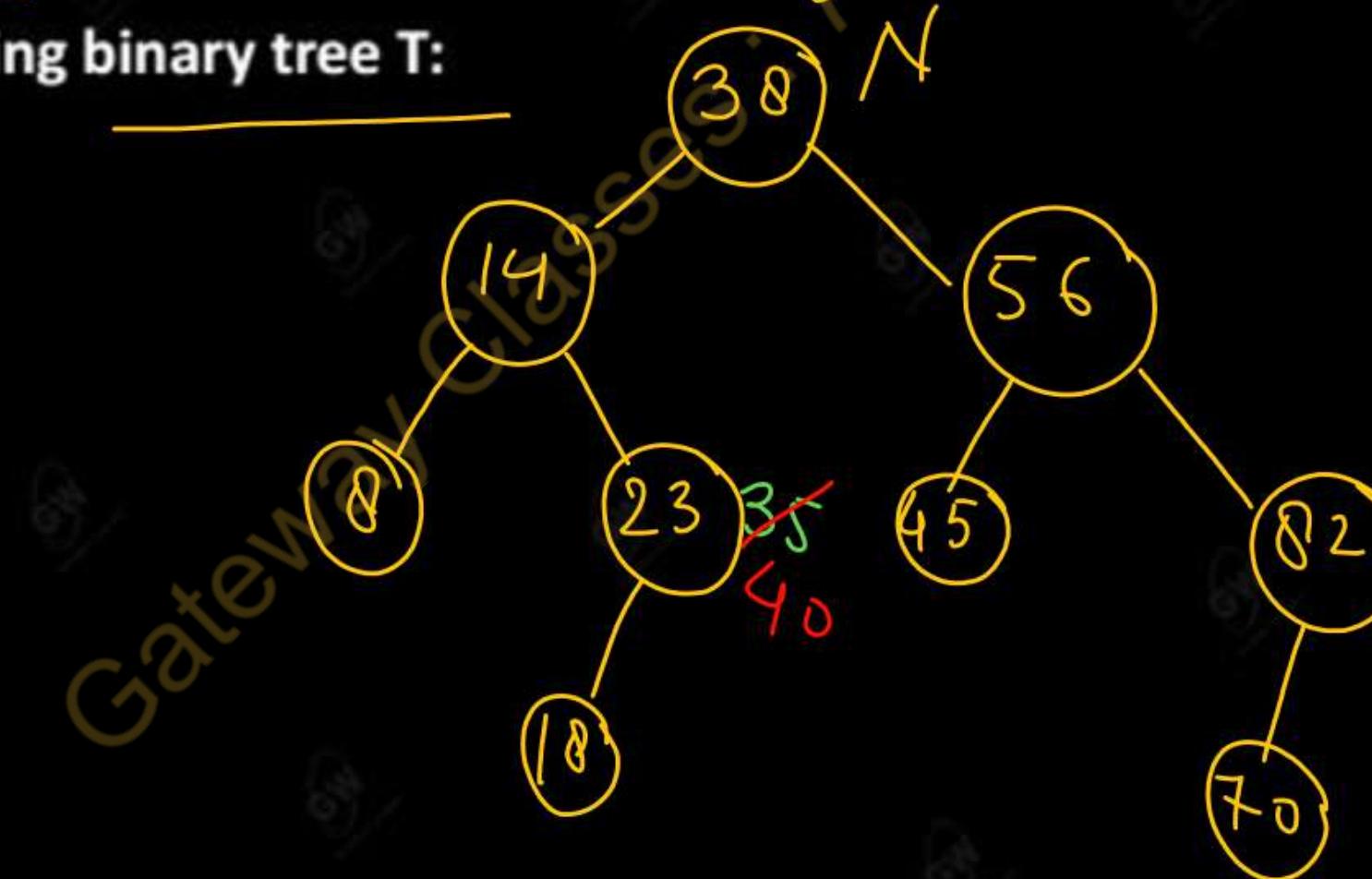
Unit 4 : Lec- 3

Today's Target

- Binary Search Tree:- Searching, Insertion and Deletion**
- AKTU PYQs**

Binary Sorted Tree

- Suppose T is a binary tree.
- Then T is called a **binary search tree** (or **binary sorted tree**) if each node N of T has the following property:
- The value at N is greater than every value in the left subtree of N and is less than every value in the right subtree of N.
- Consider the following binary tree T:



- T is a binary search tree; that is, every node N in T exceeds every number in its left subtree and is less than every number in its right subtree.
- Suppose the 23 were replaced by 35. Then T would still be a binary search tree.
- On the other hand, suppose the 23 were replaced by 40. Then T would not be a binary search tree, since the 38 would not be greater than the 40 in its left subtree.
- The definition of a binary search tree given in this section assumes that all the node values are distinct. *No duplicacy | No redundancy*
- There is an analogous definition of a binary search tree which admits duplicates, that is, in which each node N has the following property:-
 - The value at N is greater than every value in the left subtree of N and is less than or equal to every value in the right subtree of N.

SEARCHING AND INSERTING IN BINARY SEARCH TREES

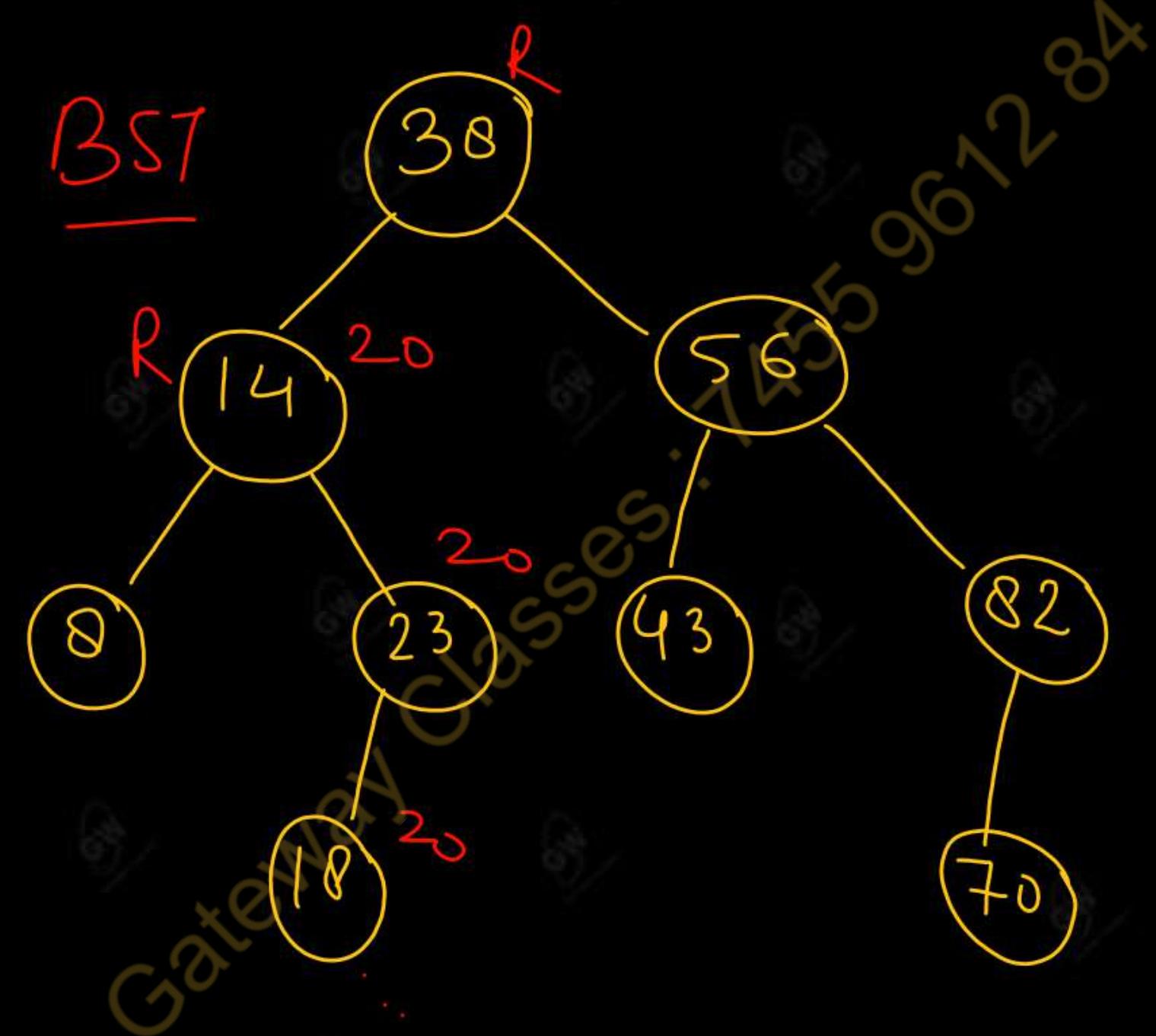
- Suppose T is a **binary search tree**. \underline{T}
- Let us discuss the **basic operations of searching and inserting** with respect to T .
- In fact, the **searching and inserting** will be given by a single **search and insertion algorithm**.
- The operation of deleting is treated separately.
- Traversing in T is the same as traversing in any binary tree.

Gateway Classes : 145596124

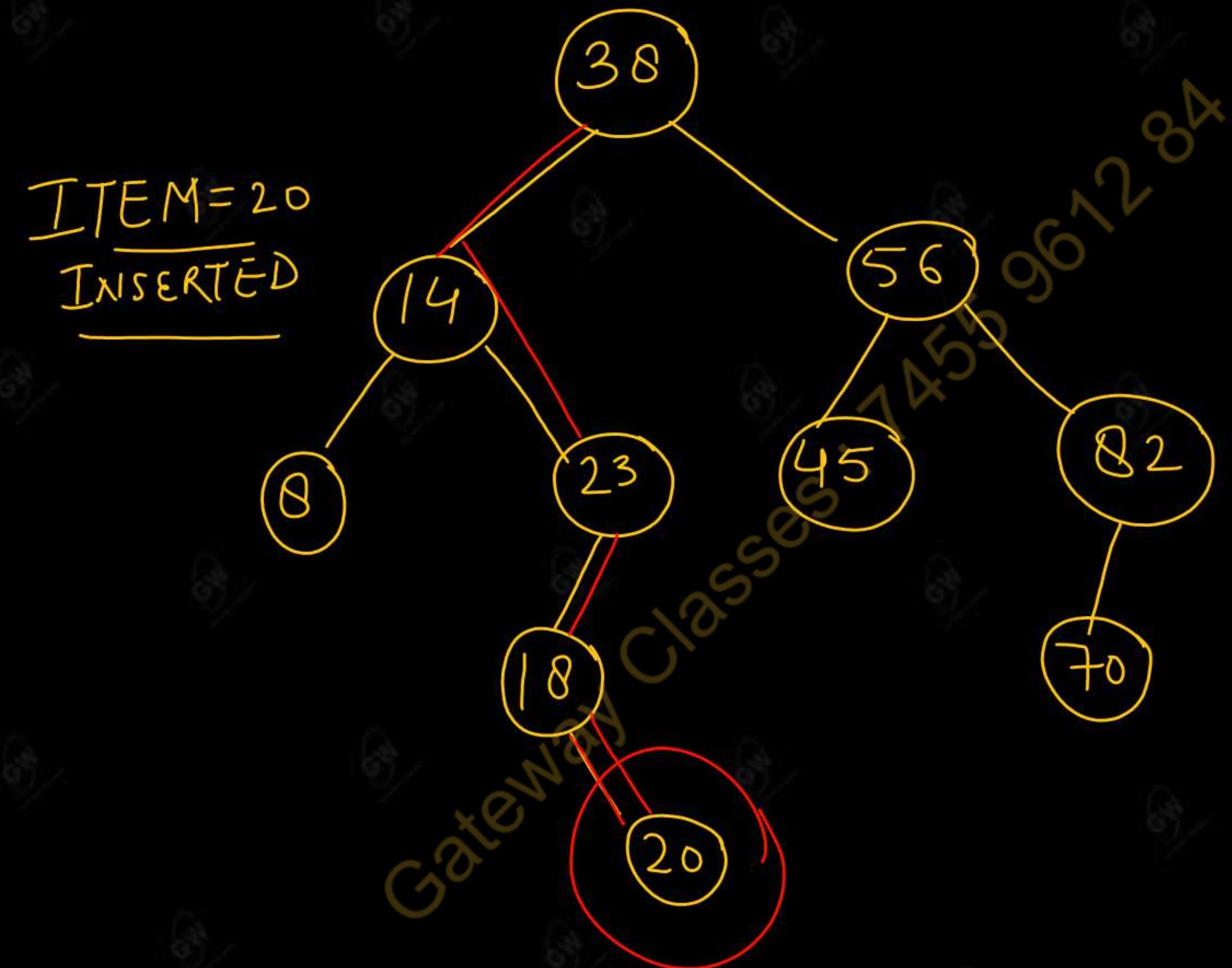
- Suppose an ITEM of information is given.
- The following algorithm finds the location of ITEM in the binary search tree T, or inserts ITEM as a new node in its appropriate place in the tree.
 - (a) Compare ITEM with the root node N of the tree.
 - (i) If ITEM < N, proceed to the left child of N.
 - (ii) If ITEM > N, proceed to the right child of N.
 - (b) Repeat Step (a) until one of the following occurs:
 - (i) We meet a node N such that ITEM = N. In this case the search is successful.
 - (ii) We meet an empty subtree, which indicates that the search is unsuccessful, and we insert ITEM in place of the empty subtree.
- In other words, proceed from the root R down through the tree T until finding ITEM in T or inserting ITEM as a terminal node in T.

EXAMPLE -

- (a) Again consider the following binary search tree T.



- Suppose ITEM = 20 is given.
- Simulating the above algorithm, we obtain the following steps:
 1. Compare ITEM = 20 with the root, 38, of the tree T. Since $20 < 38$, proceed to the left child of 38, which is 14.
 2. Compare ITEM = 20 with 14. Since $20 > 14$, proceed to the right child of 14, which is 23.
 3. Compare ITEM = 20 with 23. Since $20 < 23$, proceed to the left child of 23, which is 18.
 4. Compare ITEM = 20 with 18. Since $20 > 18$ and 18 does not have a right child, insert 20 as the right child of 18.
- The next tree shows the new tree with ITEM = 20 inserted.
- The shaded edges indicate the path down through the tree during the algorithm.



(b) Consider the following binary search tree T.



- Suppose ITEM = Davis is given.
- Simulating the above algorithm, we obtain the following steps:
 1. Compare ITEM = Davis with the root of the tree, Harris. Since Davis < Harris, proceed to the left child of Harris, which is Cohen.
 2. Compare ITEM = Davis with Cohen. Since Davis > Cohen, proceed to the right child of Cohen, which is Green.
 3. Compare ITEM = Davis with Green. Since Davis < Green, proceed to the left child of Green, which is Davis.
 4. Compare ITEM = Davis with the left child, Davis. We have found the location of Davis in the tree.

tree:

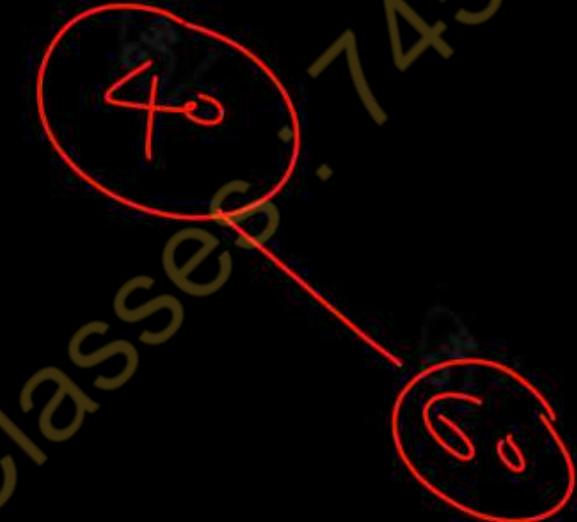
Draw each step and final BST.

Solution :-

(i) item 40

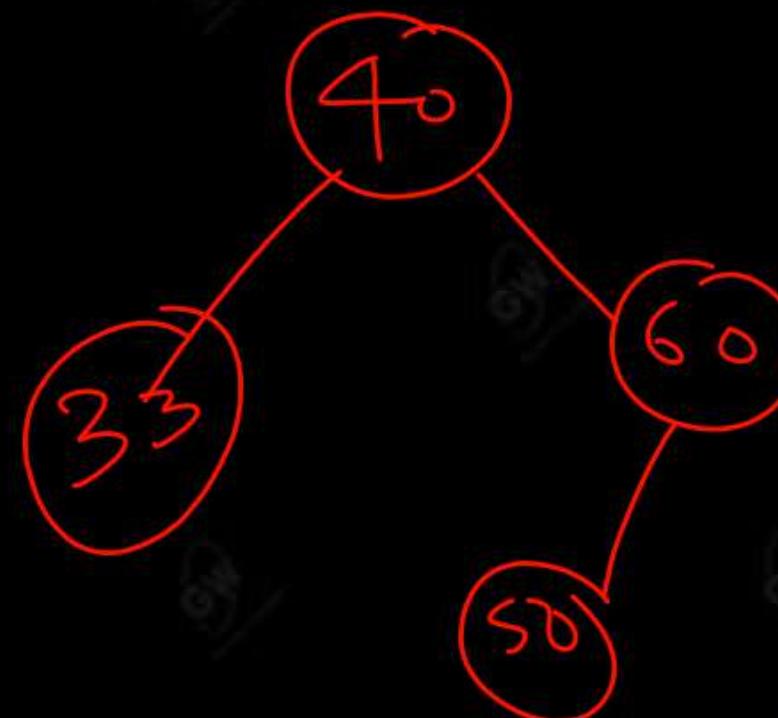
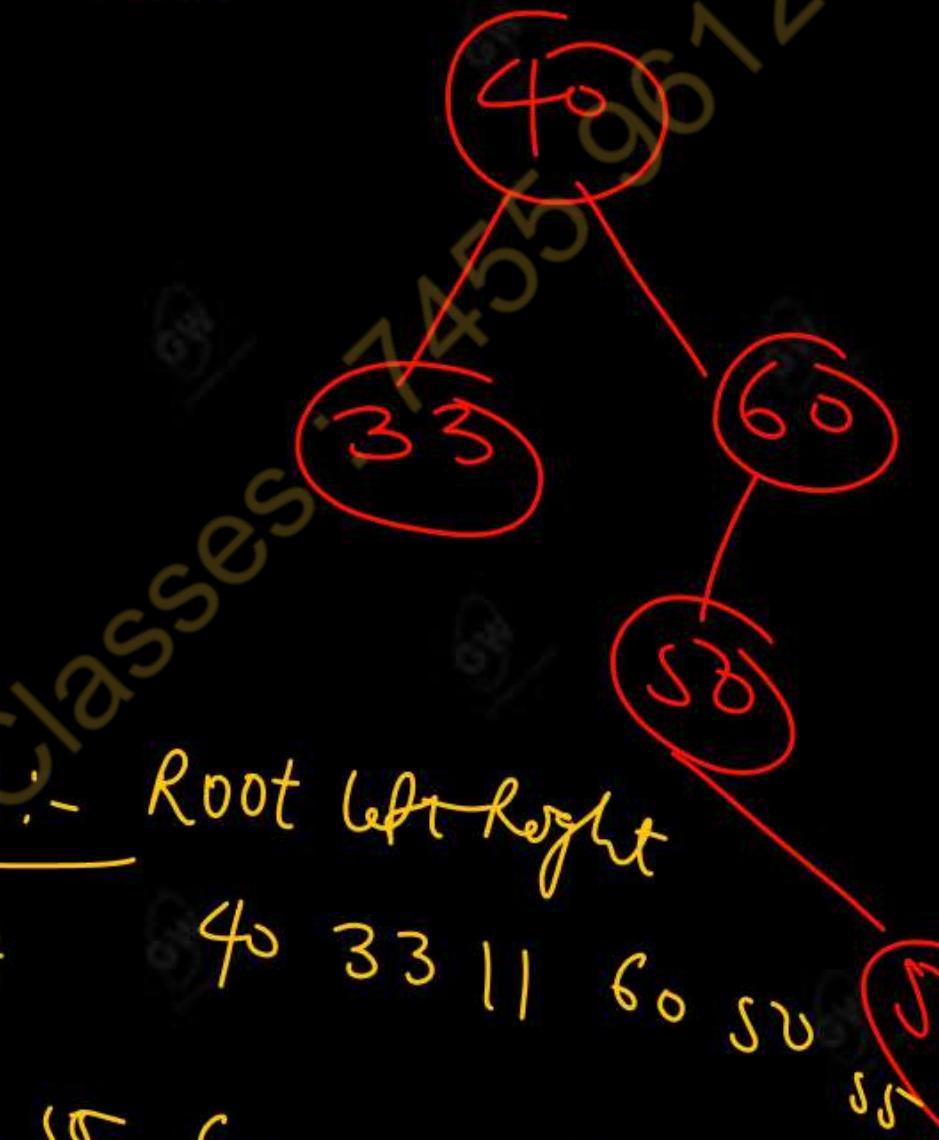
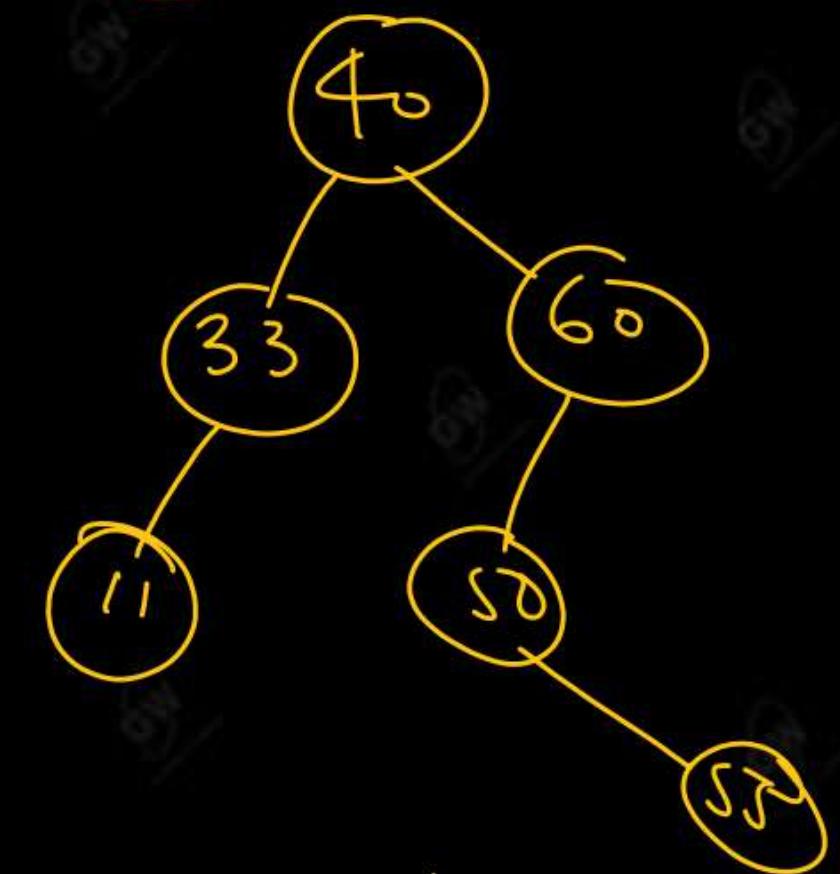


(ii) item = 60



(iii) item = 50



(IV) $i = 33$ (V) $i+n = 55^{\text{th}}$ (VI) $i+n = 11$ 

PRE :- Root left right
 $T_n = \text{left Root right}$
 11, 33, 40, 50, 55, 60

Post :- left right root
 11, 33, 55, 50, 60, 40

Amp

- The formal presentation of our search and insertion algorithm will use the following procedure, which finds the locations of a given ITEM and its parent.
- The procedure traverses down the tree using the pointer PTR and the pointer SAVE for the parent node.

Procedure: FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)

A binary search tree T is in memory and an ITEM of information is given.

This procedure finds the location LOC of ITEM in T and also the location PAR of the parent of ITEM.

There are three special cases:

(i) LOC = NULL and PAR = NULL will indicate that the tree is empty.

(ii) LOC != NULL and PAR = NULL will indicate that ITEM is the root of T.

(iii) LOC = NULL and PAR != NULL will indicate that ITEM is not in T and can be added to T as a child of the node N with location PAR.

1. [Tree empty?]

If ROOT= NULL, then: Set LOC:= NULL and PAR = NULL, and Return.

2. [ITEM at root?]

If ITEM=INFO[ROOT], then: Set LOC: = ROOT and PAR: = NULL, and Return.

3. [Initialize pointers PTR and SAVE.]

If ITEM < INFO[ROOT], then:

Set PTR = LEFT[ROOT] and SAVE:=ROOT.

Else:

Set PTR = RIGHT[ROOT] and SAVE:= ROOT.

[End of If structure.]

4. Repeat Steps 5 and 6 while PTR ≠ NULL;

5. [ITEM found?]

If ITEM=INFO[PTR], then: Set LOC:= PTR and PAR:= SAVE, and Return.

6. If ITEM < INFO[PTR], then:

Set SAVE:= PTR and PTR = LEFT[PTR].

Else:

Set SAVE:= PTR and PTR:=RIGHT[PTR]. [End of If structure.]

[End of Step 4 loop.]

7. [Search unsuccessful.] Set LOC:=NULL and PAR :=SAVE.

8. Exit.

Observe that, in Step 6, we move to the left child or the right child according to whether ITEM < INFO[PTR] or ITEM> INFO[PTR].

The formal statement of our search and insertion algorithm is as follows.

Algorithm: INSBST(INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM, LOC)

A binary search tree T is in memory and an ITEM of information is given.

This algorithm finds the location LOC of ITEM in T or adds ITEM as a new node in T at location LOC.

1. Call FIND (INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR). [Procedure Find]
2. If LOC != NULL, then Exit.
3. [Copy ITEM into new node in AVAIL list.]
 - (a) If AVAIL = NULL, then: Write: OVERFLOW, and Exit.
 - (b) Set NEW:=AVAIL, AVAIL:=LEFT [AVAIL] and INFO[NEW]:=ITEM.
 - (c) Set LOC:= NEW, LEFT[NEW]:=NULL and RIGHT[NEW]:=NULL,

4. [Add ITEM to tree.]

If PAR=NULL, then: Set ROOT:= NEW

Else if ITEM < INFO[PAR], then:

Set LEFT [PAR]:= NEW

Else:

Set RIGHT [PAR]:= NEW

[End of If structure.]

5. Exit.

Observe that, in Step 4, there are three possibilities:

- (1) The tree is empty,
- (2) ITEM is added as a left child and
- (3) ITEM is added as a right child.

DELETING IN A BINARY SEARCH TREE

- Suppose T is a binary search tree, and suppose an ITEM of information is given.
- This section gives an algorithm which deletes ITEM from the tree T .
- The deletion algorithm first uses Procedure FIND to find the location of the node N which contains ITEM and also the location of the parent node $P(N)$.
- The way N is deleted from the tree depends primarily on the number of children of node N .
- There are three cases:

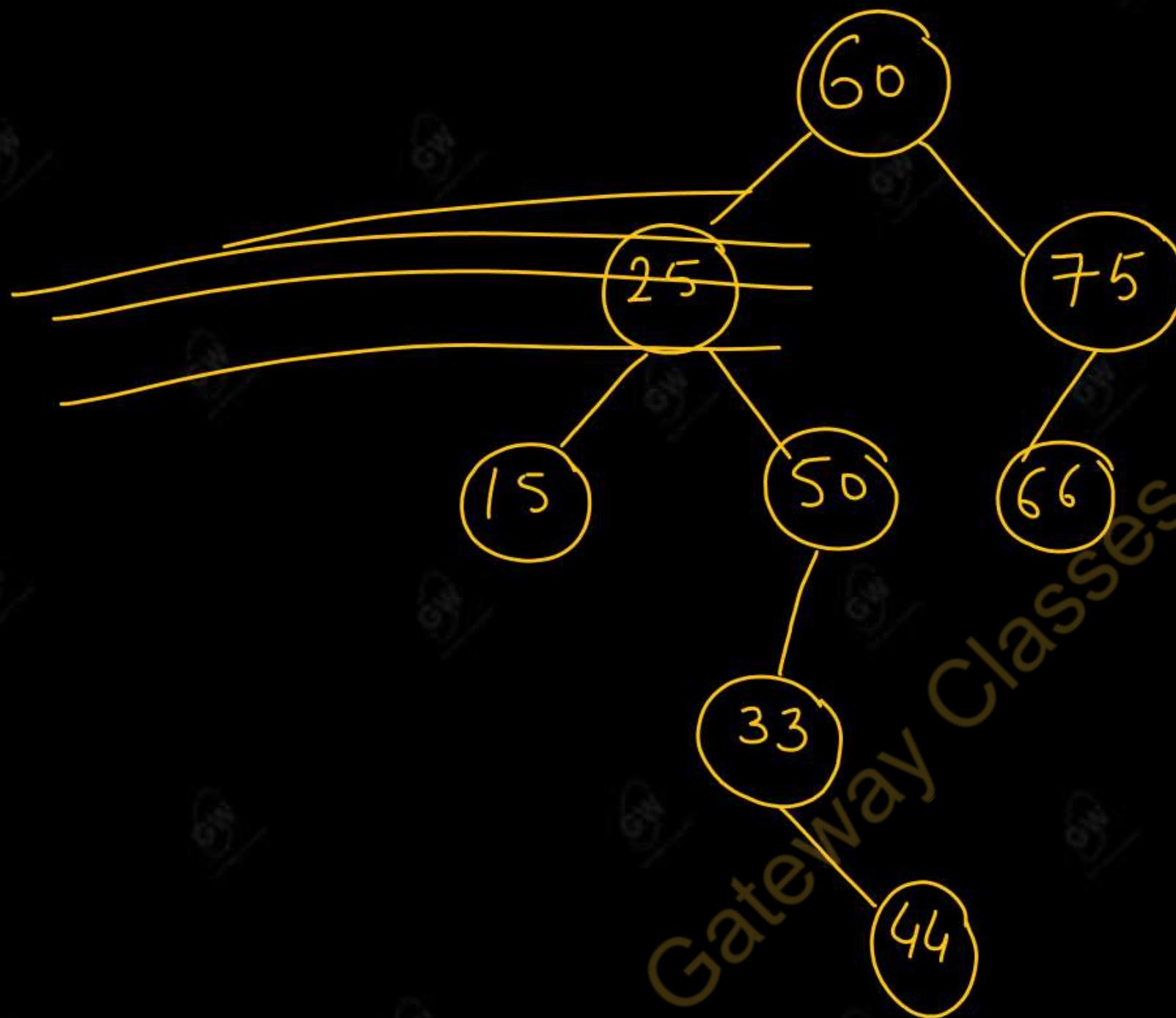
Case 1. N has no children. Then N is deleted from T by simply replacing the location of N in the parent node $P(N)$ by the null pointer.

Case 2. N has exactly one child. Then N is deleted from T by simply replacing the location of N in $P(N)$ by the location of the only child of N .

Case 3. N has two children. Let $S(N)$ denote the inorder successor of N. (The reader can verify that $S(N)$ does not have a left child.) Then N is deleted from T by first deleting $S(N)$ from T (by using Case 1 or Case 2) and then replacing node N in T by the node $S(N)$. 84

- Observe that the third case is much more complicated than the first two cases.
- In all three cases, the memory space of the deleted node N is returned to the AVAIL list.

EXAMPLE :- Consider the following binary search tree:-



BS7
BST : 14 25 33 44 50 66 75 84
Inorder :-
N
↓
15, 25, 33, 44, 50, 66, 75, 84

S(N)
↓
66 75

Suppose T appears in memory as per the following:

ROOT



AVAIL



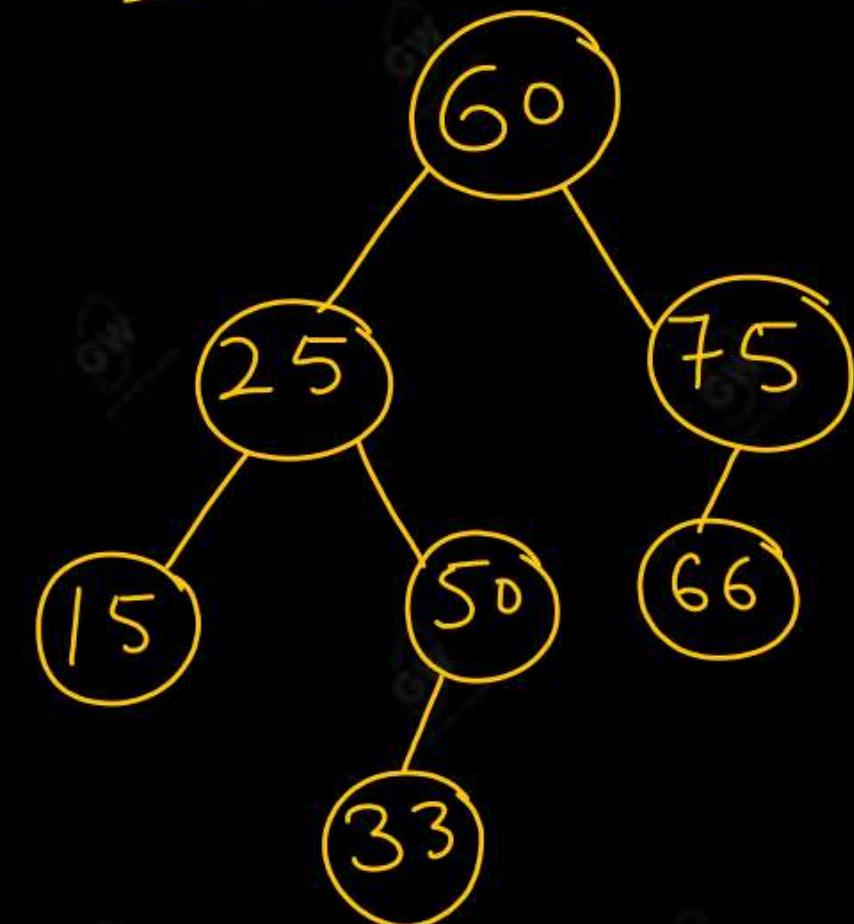
Gateway Classes

INFO LEFT RIGHT

1	33	0NULL	9
2	25	8	10
3	60	2	7
4	66	0	0
5		6	
6		0	
7	75	4	0
8	15	0	0
9	44	0	0
10	50	1	0

LINKED REPRESENTATION

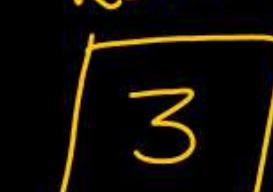
(a) Suppose we delete node 44 from the previous binary search tree. Note that node 44 has no children. So after deletion 44, the binary search tree and its linked representation will be as follows:



NODE 44 is Deleted

The deletion is accomplished by simply assigning NULL to the parent node, 33, (The shading indicates the changes.)

ROOT



AVAIL

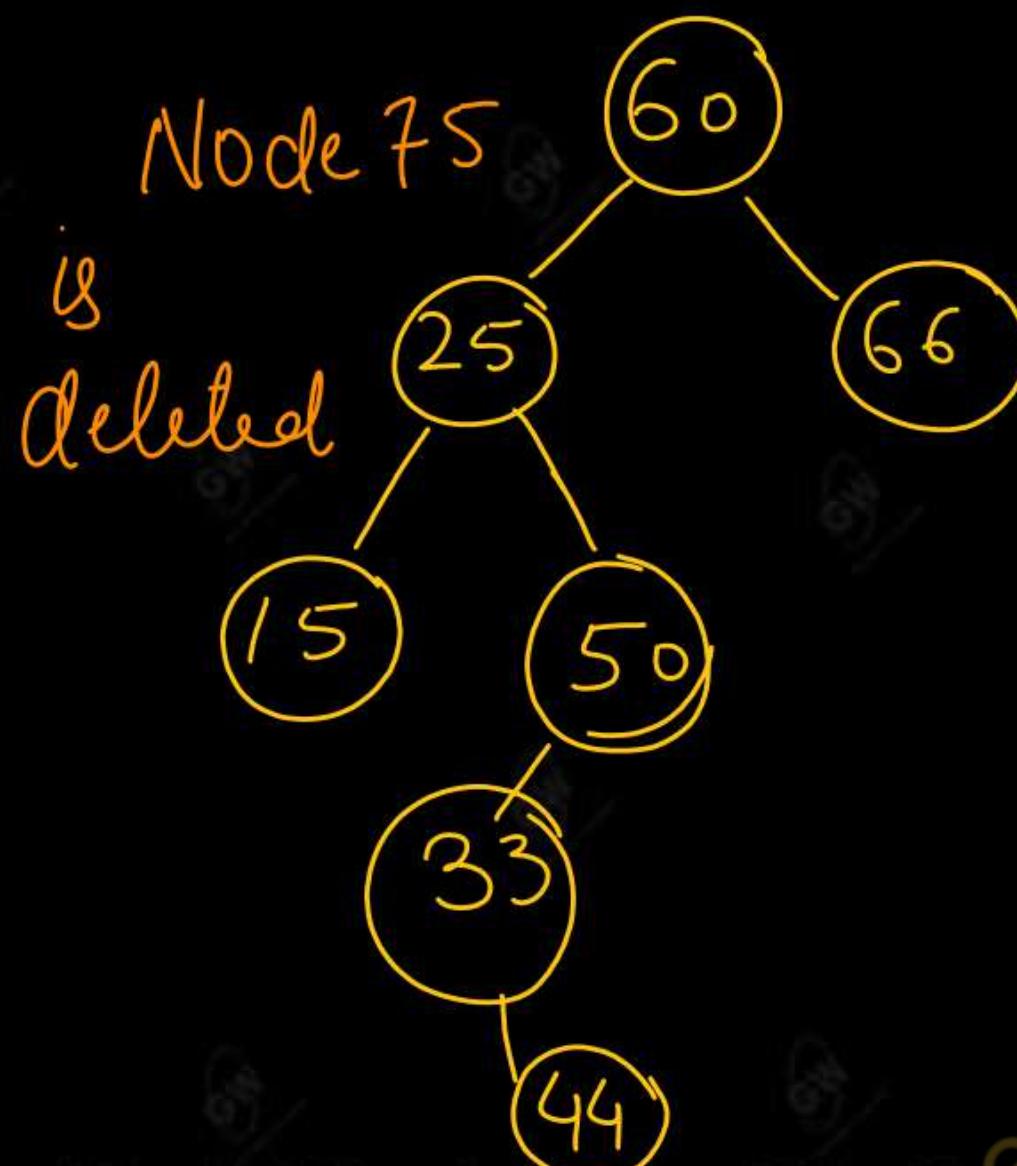


INFO LEFT RIGHT

	INFO	LEFT	RIGHT
1	33	0	0
2	25	8	10
3	60	2	7
4	66	0	0
5		6	
6		0	
7	75	4	0
8	15	0	0
9		5	
10	50	1	0

LINKED REPRESENTATION

(b) Suppose we delete node 75 from the previous binary tree instead of node 44. Note that node 75 has only one child. The following binary search tree pictures the tree after 75 is deleted.



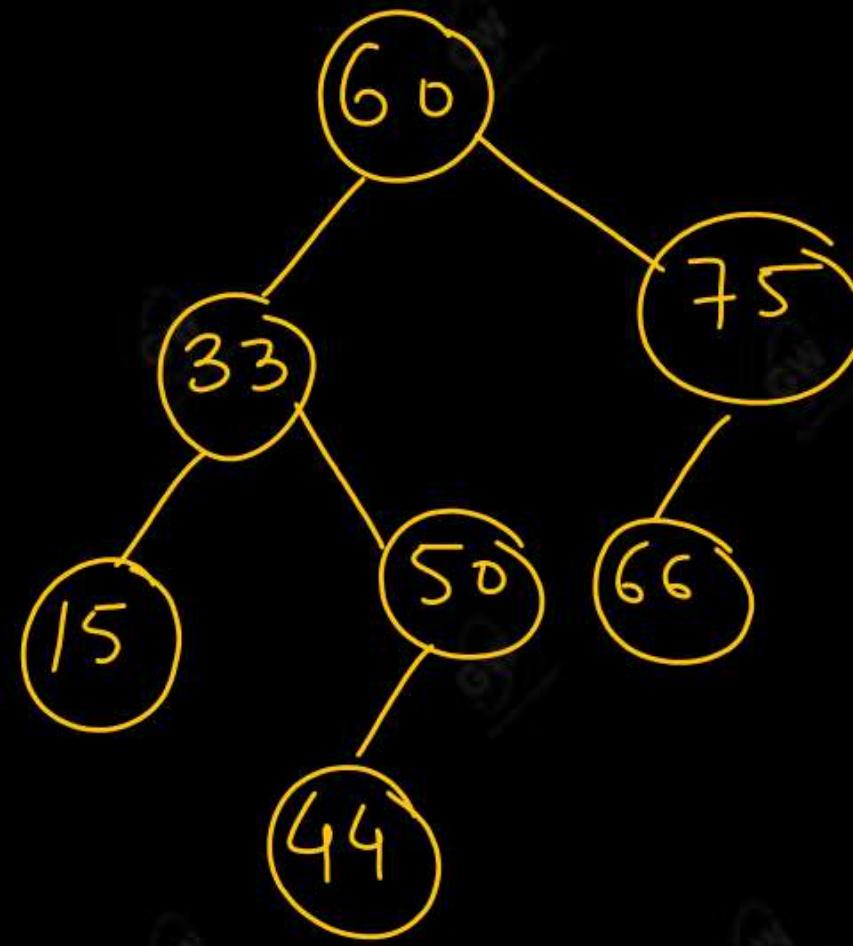
ROOT
[3]
AVAIL
[7]

	INFO	LEFT	RIGHT
1	33	0	9
2	25	0	10
3	60	2	4
4	66	0	0
5			
6			NULL
7			5
8	15	0	0
9	44	0	0
10	50	1	0

LINKED REPRESENTATION

The deletion is accomplished by changing the right pointer of the parent node 60, which originally pointed to 75, so that it now points to node 66, the only child of 75. (The shading indicates the changes.)

(c) Suppose we delete node 25 from the previous binary tree instead of node 44 or node 75. Note that node 25 has two children. Also observe that node 33 is the inorder successor of node 25. The following figure shows the tree after 25 is deleted with its linked representation.



Node 25 is deleted



	INFO	LEFT	RIGHT
1	33	8	10
2		5	
3	60	1	7
4	66	0	0
5		6	
6		0	
7	75	4	0
8	15	0	0
9	44	0	0
10	50	9	0

LINKED REPRESENTATION

The deletion is accomplished by first deleting 33 from the tree and then replacing node 25 by node 33. We emphasize that the replacement of node 25 by node 33 is executed in memory only by changing pointers, not by moving the contents of a node from one location to another. Thus 33 is still the value of INFO[1].

- Our deletion algorithm will be stated in terms of Procedures CASEA AND CASEB.
- The first procedure refers to Cases 1 and 2, where the deleted node N does not have two children; and the second procedure refers to Case 3, where N does have two children.
- There are many subcases. which reflect the fact that N may be a left child, a right child or the root. Also, in Case 2, N may have a left child or a right child.
- Procedure CASEB treats the case that the deleted node N has two children.
- We note that the inorder successor of N can be found by moving to the right child of N and then moving repeatedly to the left until meeting a node with an empty left subtree.

Procedure CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR)

This procedure deletes the node N at location LOC, where N does not have two children.

The pointer PAR gives the location of the parent of N or else PAR = NULL indicates that N is the root node.

The pointer CHILD gives the location of the only child of N, or else CHILD = NULL indicates N has no children.

1. [Initializes CHILD.]

If LEFT [LOC] = NULL and RIGHT[LOC] =NULL, then:

Set CHILD: =NULL,

Else if LEFT [LOC] != NULL, then:

Set CHILD: = LEFT[LOC]

Else

Set CHILD: =RIGHT[LOC]

[End of If structure.]

2. If PAR != NULL, then:

If LOC = LEFT [PAR] then:

Set LEFT [PAR]= CHILD

Else:

Set RIGHT [PAR]: CHILD.

[End of If structure.]

Else

Set ROOT: =CHILD.

[End of If structure.]

3. Return.

Procedure CASEB (INFO, LEFT, RIGHT, ROOT, LOC, PAR)

This procedure will delete the node N at location LOC, where N has two children.

The pointer PAR gives the location of the parent of N, or else PAR = NULL indicates that N is the root node.

The pointer SUC gives the location of the inorder successor of N, and PARSUC gives the location of the parent of the inorder successor.

1. [Find SUC and PARSUC.]

(a) Set PTR = RIGHT [LOC] and SAVE: = LOC.

(b) Repeat while LEFT[PTR] != NULL:

Set SAVE:= PTR and PTR = LEFT [PTR].

[End of loop.]

(c) Set SUC:= PTR and PARSUC:= SAVE.

2. [Delete inorder successor, using Procedure CASEA]

Call CASEA (INFO, LEFT, RIGHT, ROOT, SUC, PARSUC).

3. [Replace node N by its inorder successor.]

(a) If PAR != NULL, then:

If LOC = LEFT [PAR], then:

Set LEFT[PAR]:= SUC.

Else:

Set RIGHT [PAR]:= SUC. [End of If structure.]

Else:

Set ROOT:= SUC. [End of If structure.]

(b) Set LEFT[SUC]:= LEFT[LOC] and

RIGHT[SUC]:= RIGHT[LOC].

4. Return.

We can now formally state our deletion algorithm, using Procedures CASEA and CASEB as building blocks.

Algorithm : DEL (INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM)

A binary search tree T is in memory, and an ITEM of information is given. This algorithm deletes ITEM from the tree.

1. [Find the locations of ITEM and its parent, using Procedure FIND]

Call FIND (INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR).

2. [ITEM in tree?]

If LOC = NULL, then: Write: ITEM not in tree, and Exit.

3. [Delete node containing ITEM.]

If RIGHT [LOC] != NULL and LEFT [LOC] != NULL, then:

Call CASEB (INFO, LEFT, RIGHT, ROOT, LOC, PAR).

Else:

Call CASEA (INFO, LEFT, RIGHT, ROOT, LOC, PAR). [End of If structure.]

4. [Return deleted node to the AVAIL list.]

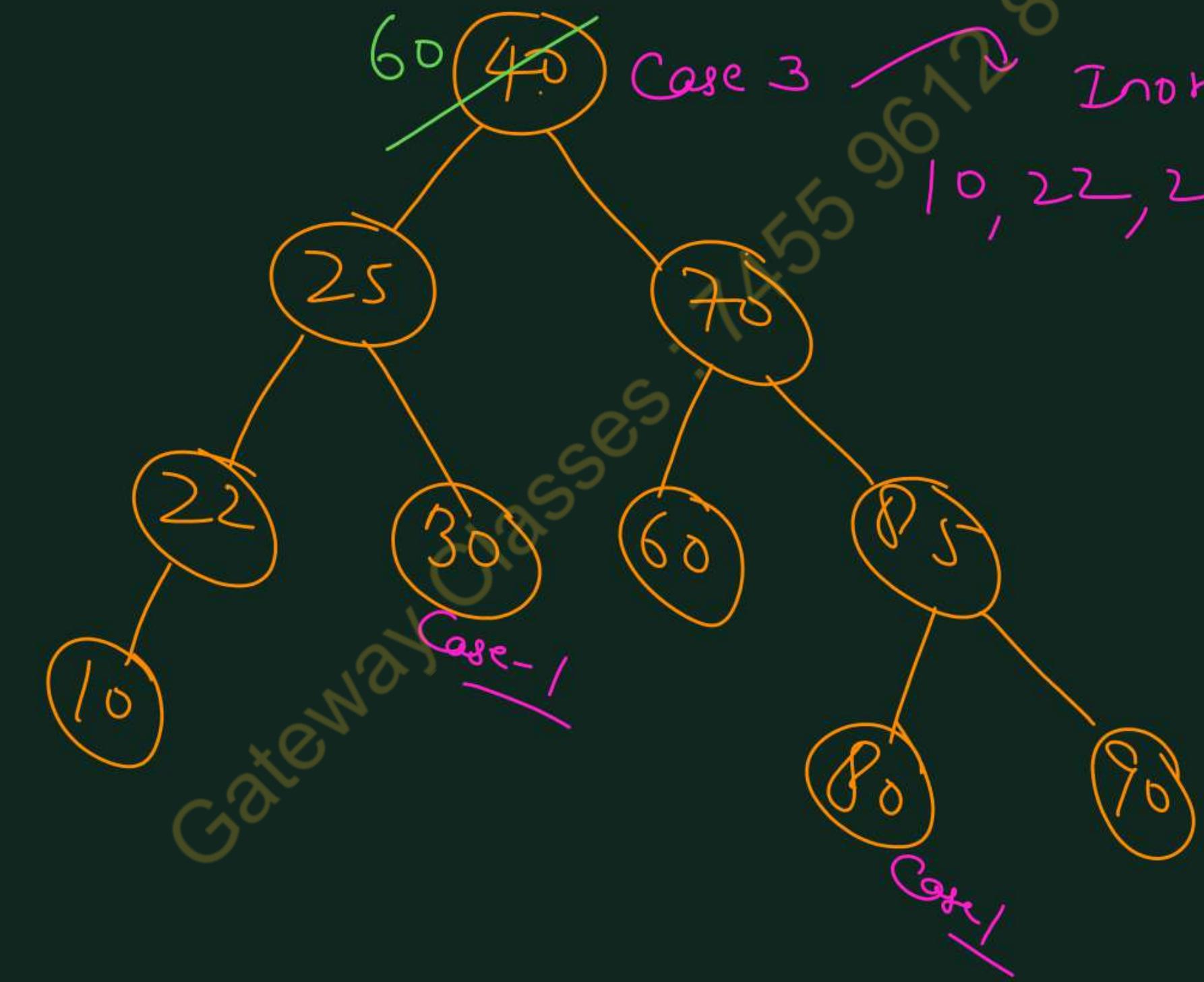
Set LEFT [LOC] := AVAIL and AVAIL := LOC.

5. Exit.

Example:- Suppose the following numbers are inserted in an empty Binary Search Tree -

40, 25, 70, 22, 85, 60, 80, 90, 10, 30

- (i) Draw the BST, T .
- (ii) Find pre order, in order and postorder traversal of T .
- (iii) Apply the following operations on T .
 - a) Delete node 30
 - b) Delete node 80
 - c) Delete node 40.



$\checkmark 40, 25, \checkmark 70, 22, \checkmark 80, 60, \checkmark 80, 90, 10, 30$
 $\checkmark 60$ ~~40~~ **Case 3**
 Inorder: 10, 22, 25, 30, 40, 60, 70
 $\checkmark 80, 85, 90$

N	$S(N)$
10, 22, 25, 30, 40, 60, 70	60, 70
80, 85, 90	80, 85, 90

Q.1 Construct a binary search tree from the given values. Consider the first value as the root node.

Values: 2, 25, 90, 82, 7, 13, 47, 49, 63

2014-15, 5 Marks

Q.2 What happens if the binary search tree is left oriented or right oriented? Explain the problem and give the solution.

2014-15, 5 Marks

Q.3 Explain Binary search tree and its operations. Make a binary search tree for the following sequence of numbers show all steps: 45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10

2015-16, 10 Marks

Q.4 Explain Binary Search Tree and its operations. Make a binary search tree for the following sequence of numbers, show all steps : 45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10

2016-17, 10 Marks

Q.5 Discuss the concept of successor and predecessor in Binary Search Tree.

2017-18, 2 Marks

Q.6 Write the structure of binary search tree, threaded binary tree.

2017-18, 2 marks

Q.7 Write an algorithm for deletion of an element in binary search tree.

2018-19, 7 MARKS

Q.8 What is the difference between the binary search tree and heap? For a given sequence of numbers, construct a heap and a BST.

34, 23, 67, 45, 12, 54, 87, 43, 98, 75, 84, 93, 31

2019-20, 10 Marks

Q.9 What is the significance of maintaining threads in binary Search Tree? Write an algorithm to insert a node in thread binary tree.

2021-22, 2 Marks

Q.10 Differentiate between binary search tree and a heap.

2022-23, 2 marks

Unit 4: Lec - 4

Today's Target

- Path Lengths : Huffman's Algorithm
- General Tree : General Tree to Binary Tree
- AKTU PYQs

Q.1 Write short note on Huffman Algorithm.

2016-17, 5 marks

Q.2 Construct a Huffman tree for given characters A, B, C, D, E, F, G, H having frequencies 22, 5, 11, 19, 2, 11, 25, 5 respectively. What will be the code of HEAD in binary?

2018-19, 7 marks

Q.3 Explain Huffman algorithm? Construct Huffman tree for MAHARASHTRA with its optional code.

2018-19, 7 MARKS

Q.4 Explain all steps used to build Huffman Tree where following characters are given with their frequencies:

a:5, b:9, c:12, d:13, e:16, f:45

2020-21, 7 marks

PATH LENGTHS; HUFFMAN'S ALGORITHM

- We know that an extended binary tree or 2-tree is a binary tree T in which each node has either 0 or 2 children.
- The nodes with 0 children are called external nodes, and the nodes with 2 children are called internal nodes.
- The following tree shows a 2-tree where the internal nodes are denoted by circles and the external nodes are denoted by squares.

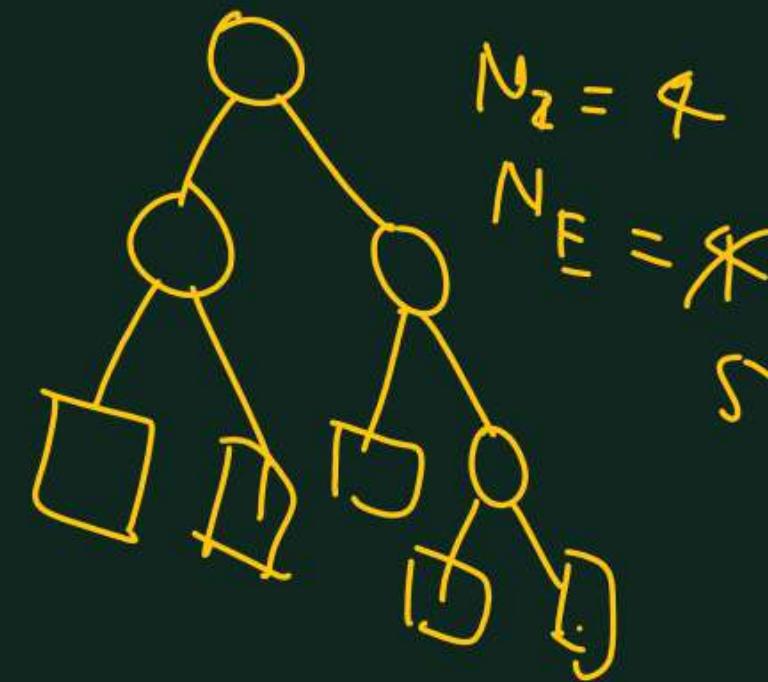
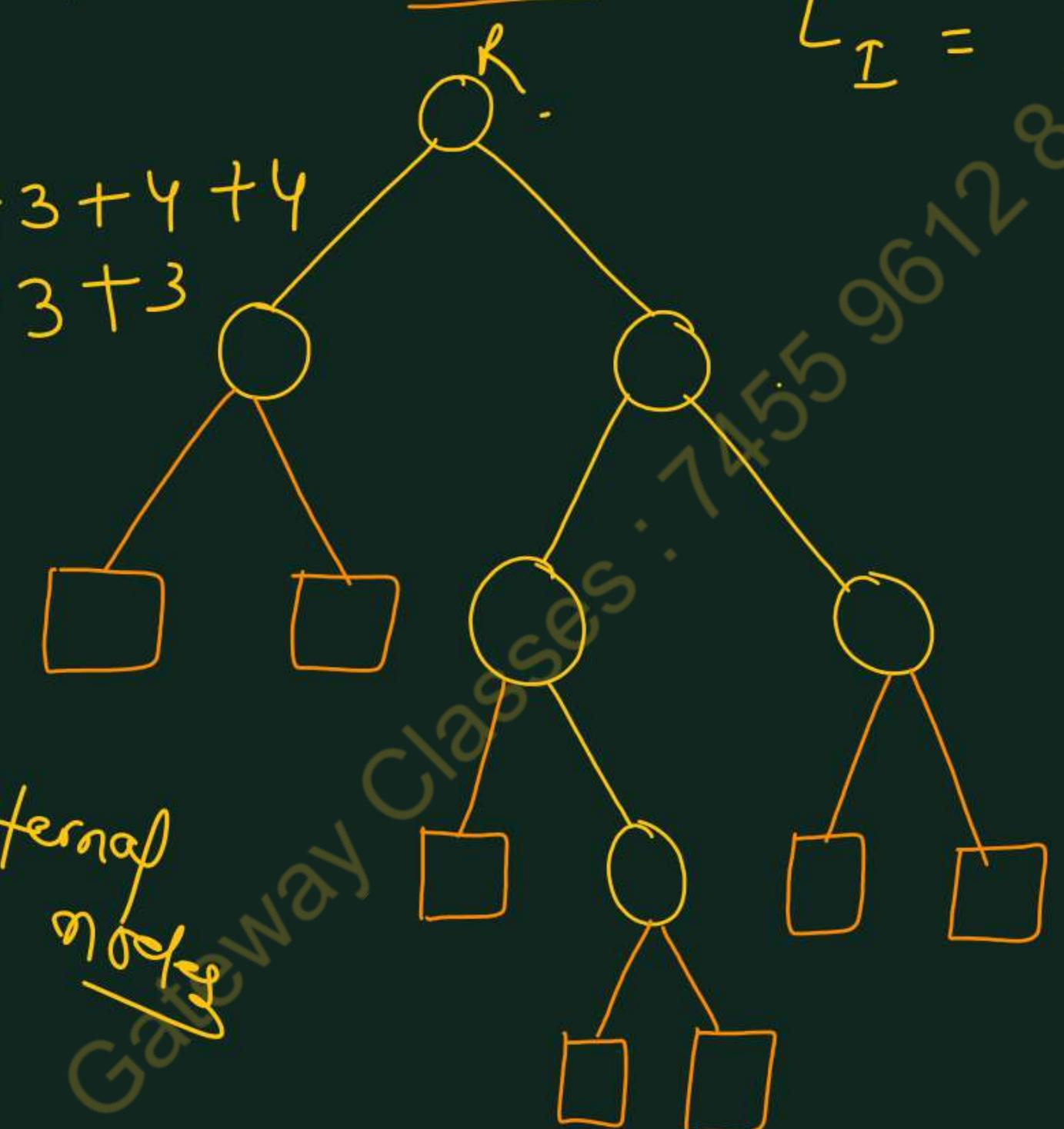
2 - TREE

$$L_E = 2 + 2 + 3 + 4 + 4 \\ = 21 + 3 + 3$$

$L_I =$

7455961284

External
Nodes
Gateway Classes : 7455961284



$$N_2 = 4$$

$$N_E = 8$$

S

- In any 2-tree, the number N_E of external nodes is 1 more than the number N_I of internal nodes, that is:

$$N_E = N_I + 1$$

- For example, for the previous 2 - tree, $N_I = 6$, and $N_E = N_I + 1 = 7$
- The running time of the algorithm may depend on the lengths of the paths in the tree.
- Therefore, we define the external path length L_E of a 2-tree T to be the sum of all path lengths summed over each path from the root R of T to an external node.
- The internal path length L_I of T is defined, using internal nodes instead of external nodes.
- For the tree in the previous figure -

External Path Length $L_E = 2 + 2 + 3 + 4 + 4 + 3 + 3 = 21$ and

Internal Path Length $L_I = 0 + 1 + 1 + 2 + 3 + 2 = 9$

- Observe that

$$\underline{L_I + 2n = 9 + 2 \cdot 6 = 9 + 12 = 21 = L_E}$$

where $n = 6$ is the number of internal nodes. In fact, the formula

$$L_E = L_I + 2n$$

is true for any 2-tree with n internal nodes.

$$\begin{aligned} 21 &= 9 + 2 \times 6 \\ 21 &= 9 + 12 \end{aligned}$$

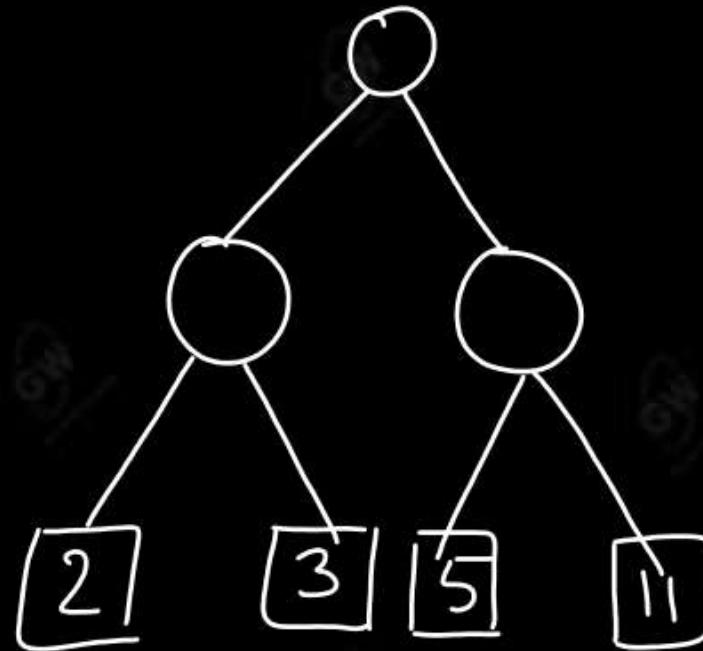
- Suppose T is a 2-tree with n external nodes, and suppose each of the external nodes assigned a is (nonnegative) weight.

- The (external) weighted path length P of the tree T is defined to be the sum of the weighted path lengths; i.e..

$$P = W_1 L_1 + W_2 L_2 + \dots + W_n L_n$$

where W_1 , and L_1 denote, respectively, the weight and path length of an external node N_1 .

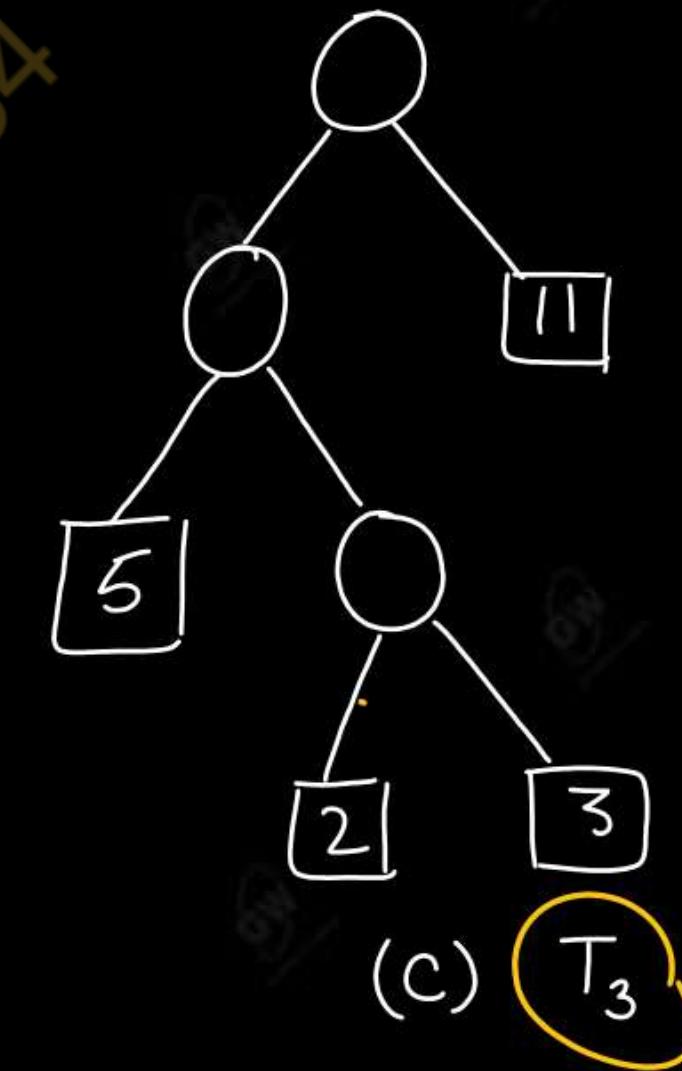
EXAMPLE : Consider following figures for three 2-trees, T_1 , T_2 and T_3 , each having external nodes with weights 2, 3, 5 and 11.



(a) T_1



(b) T_2



(c) T_3

The weighted path lengths of the three trees are as follows:

$$P_1 = \underline{2 * 2} + 3 * 2 + 5 * 2 + 11 * 2 = 42$$

$$P_2 = \underline{11 * 2} + \underline{3 * 3} + \underline{5 * 3} + \underline{2 * 1} = 48$$

$$P_3 = \underline{5 * 2} + \underline{2 * 3} + \underline{3 * 3} + \underline{11 * 1} = 36$$

□ The general problem that we want to solve is as follows.

□ Suppose a list of n weights is given:

$$W_1, W_2, \dots, \dots, W_n$$

□ Among all the 2-trees with n external nodes and with the given n weights, find a tree T with a minimum - weighted path length.

□ Huffman gave an algorithm, to find such a tree T .

A tree with min weighted Path (eth).

Huffman's Algorithm: Suppose W_1 and W_2 are two minimum weights among the n given weights $W_1, W_2 \dots W_n$. Find a tree T which gives a solution for the $n-1$ weights

$$W_1 + W_2, W_3, W_4 \dots W_n$$

Then, in the tree T' , replace the external node

$W_1 + W_2$ by the subtree



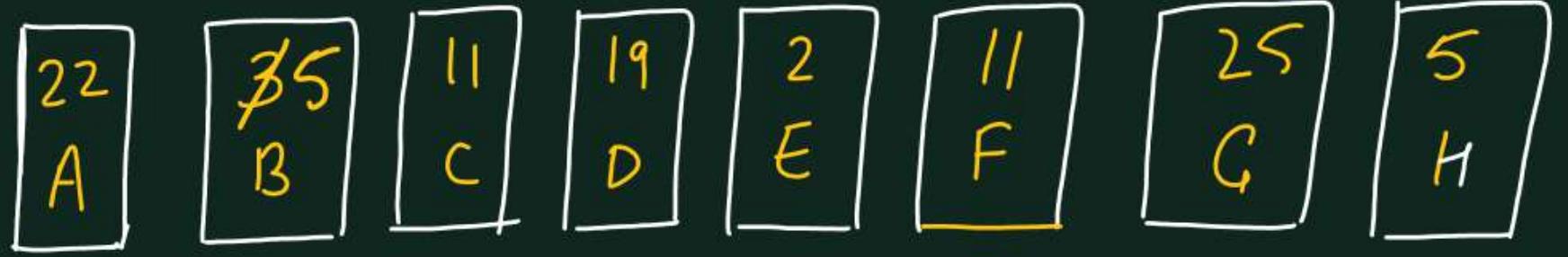
The new 2-tree T is the desired solution.

EXAMPLE :-

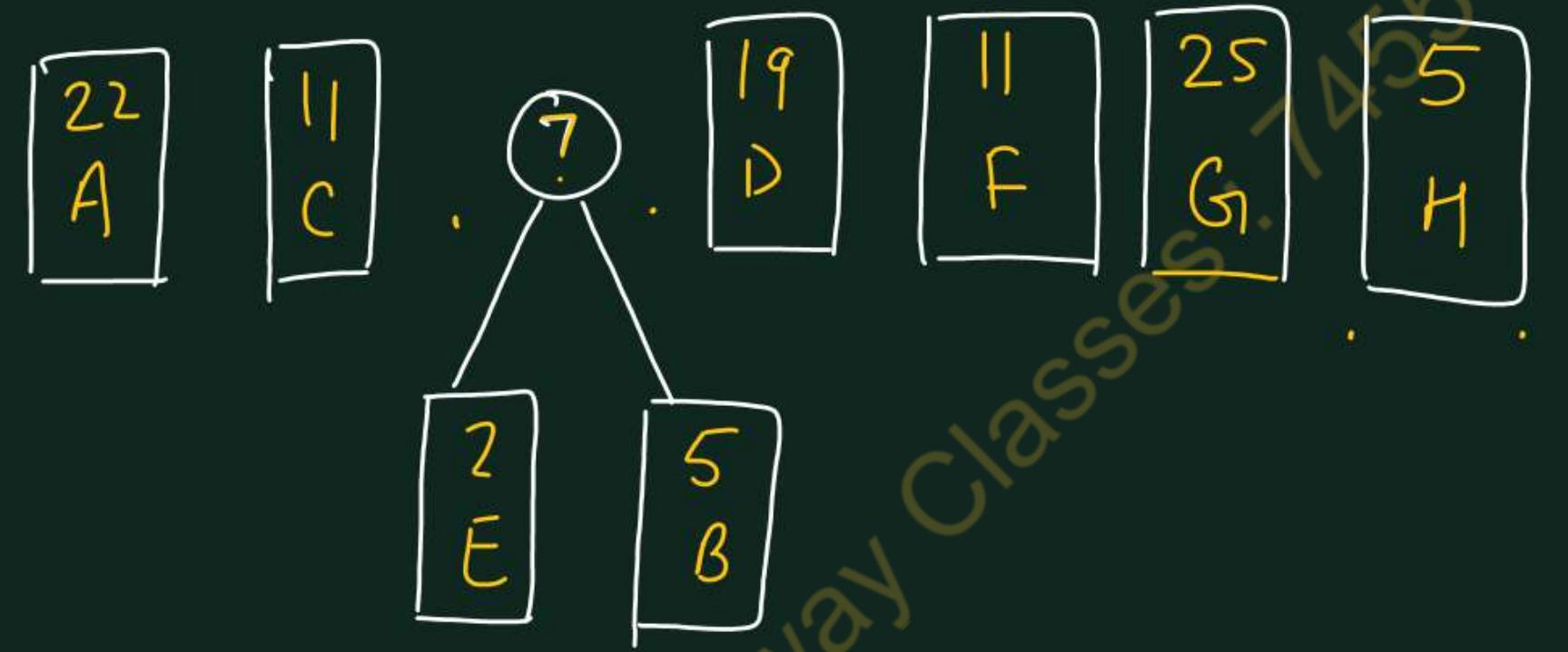
Suppose A, B, C, D, E, F, G and H are 8 data items, and suppose they are assigned weights as follows:

Data item :	A	B	C	D	E	F	G	H
Weight	22	5	11	19	26	11	25	5

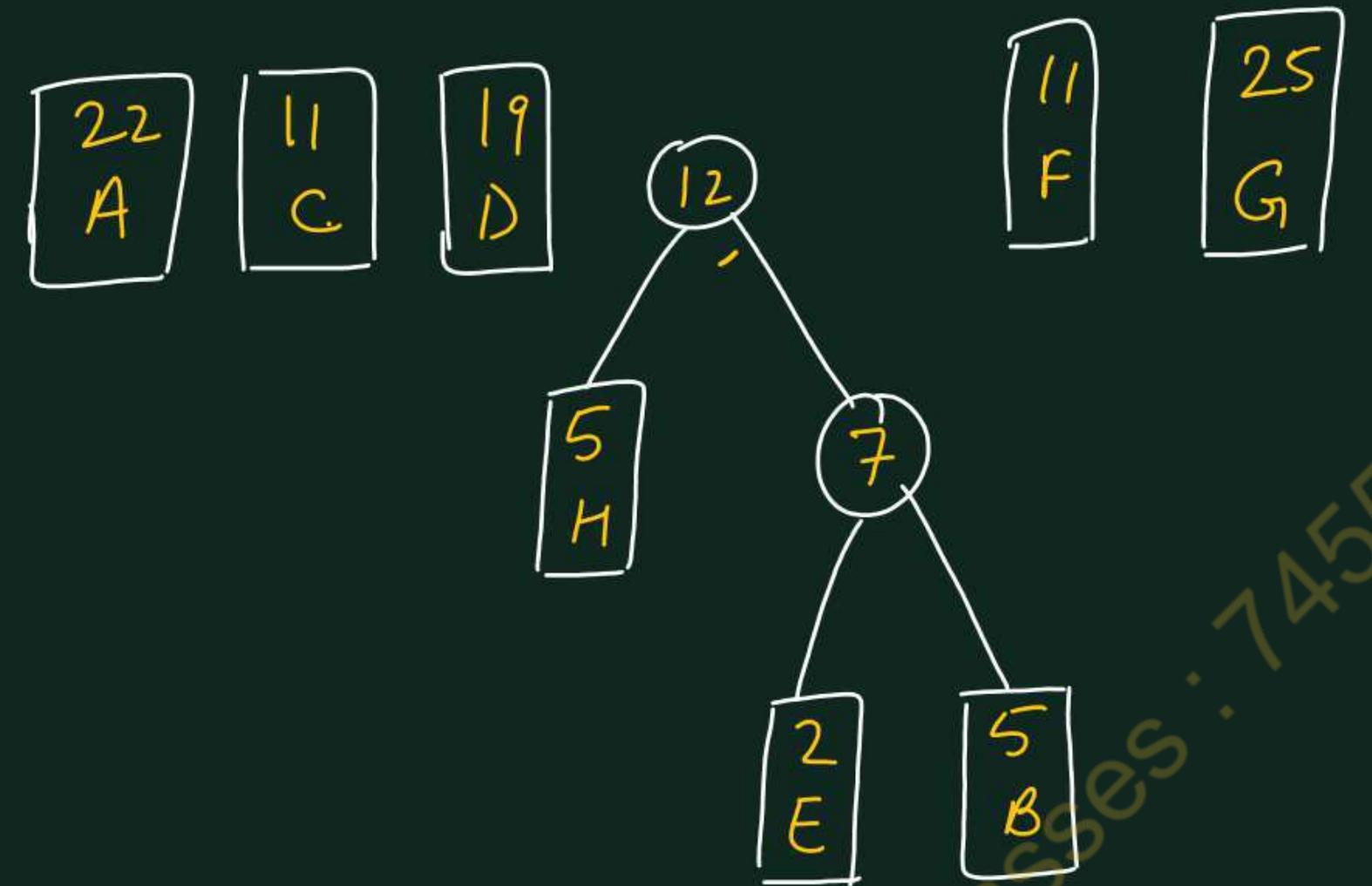
The following figures from (a) through (h) shows how to construct the tree T with minimum-weighted path length using the above data and Huffman's algorithm.



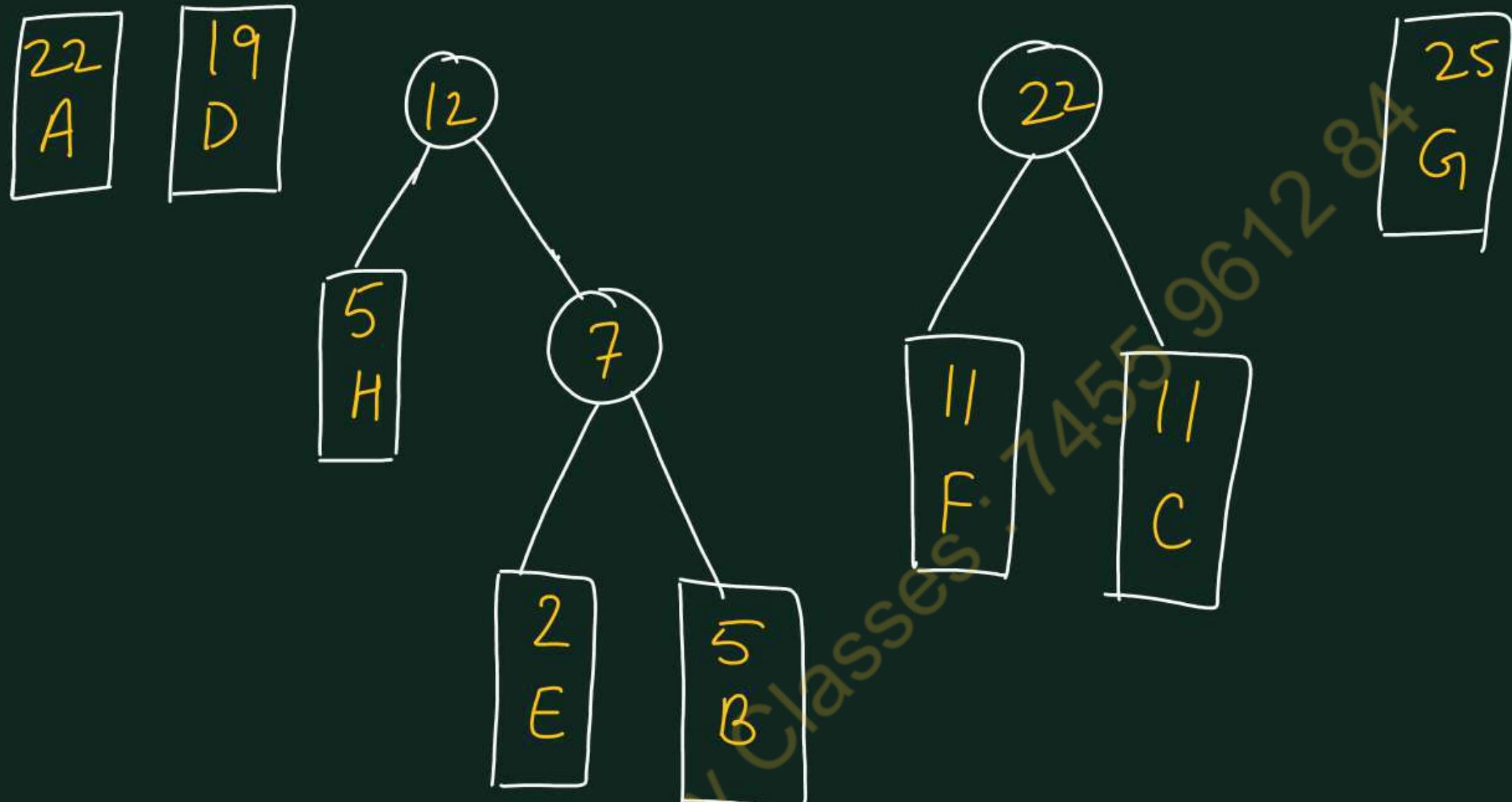
(a)



b)

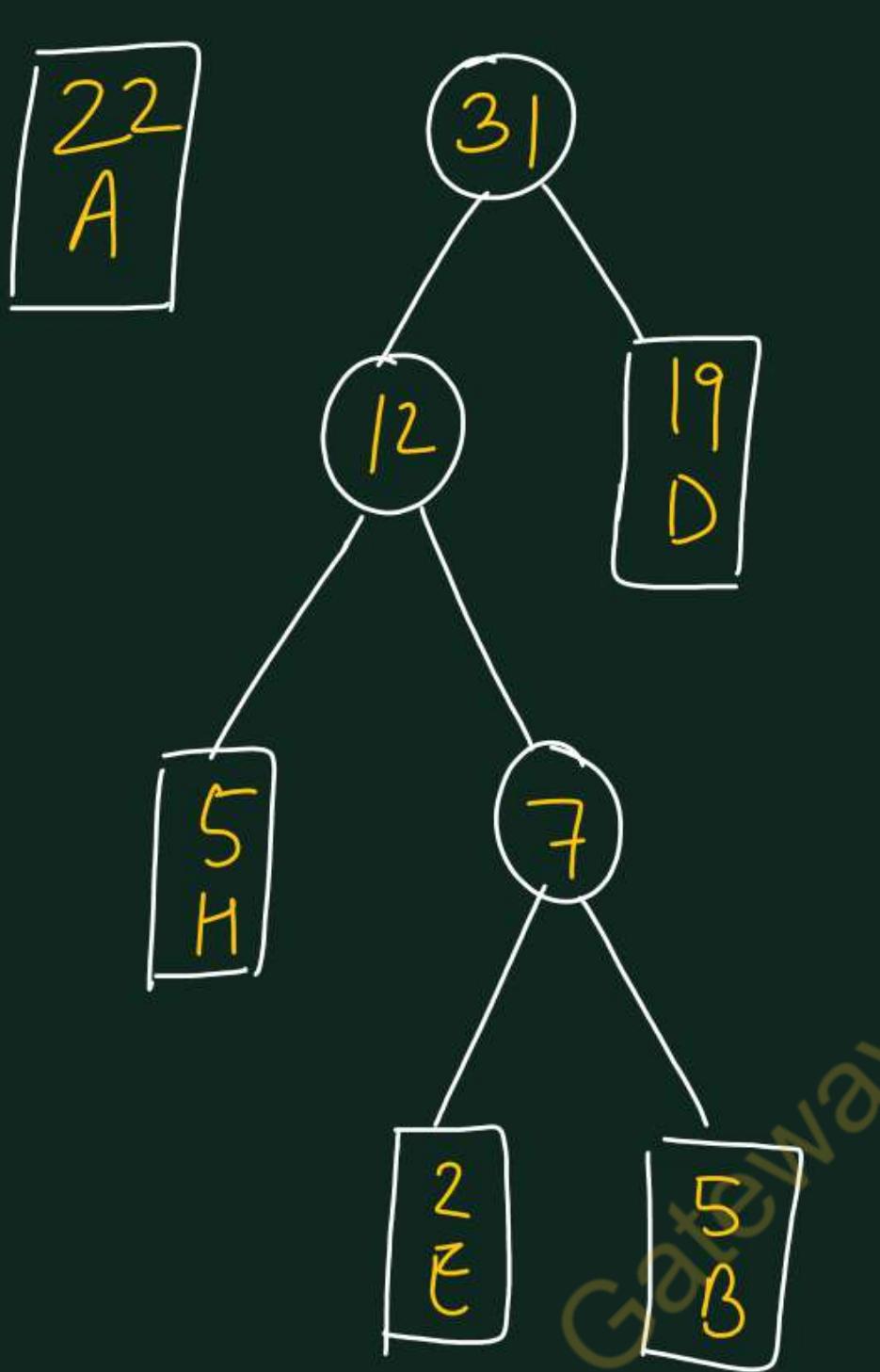


Gateway Classes : 7455 9612 84
(C)



$$\begin{array}{r} 12 \\ 22 \\ \hline 34 \\ | \\ 12 \\ | \\ 19 \\ \hline 31 \end{array}$$

(d)



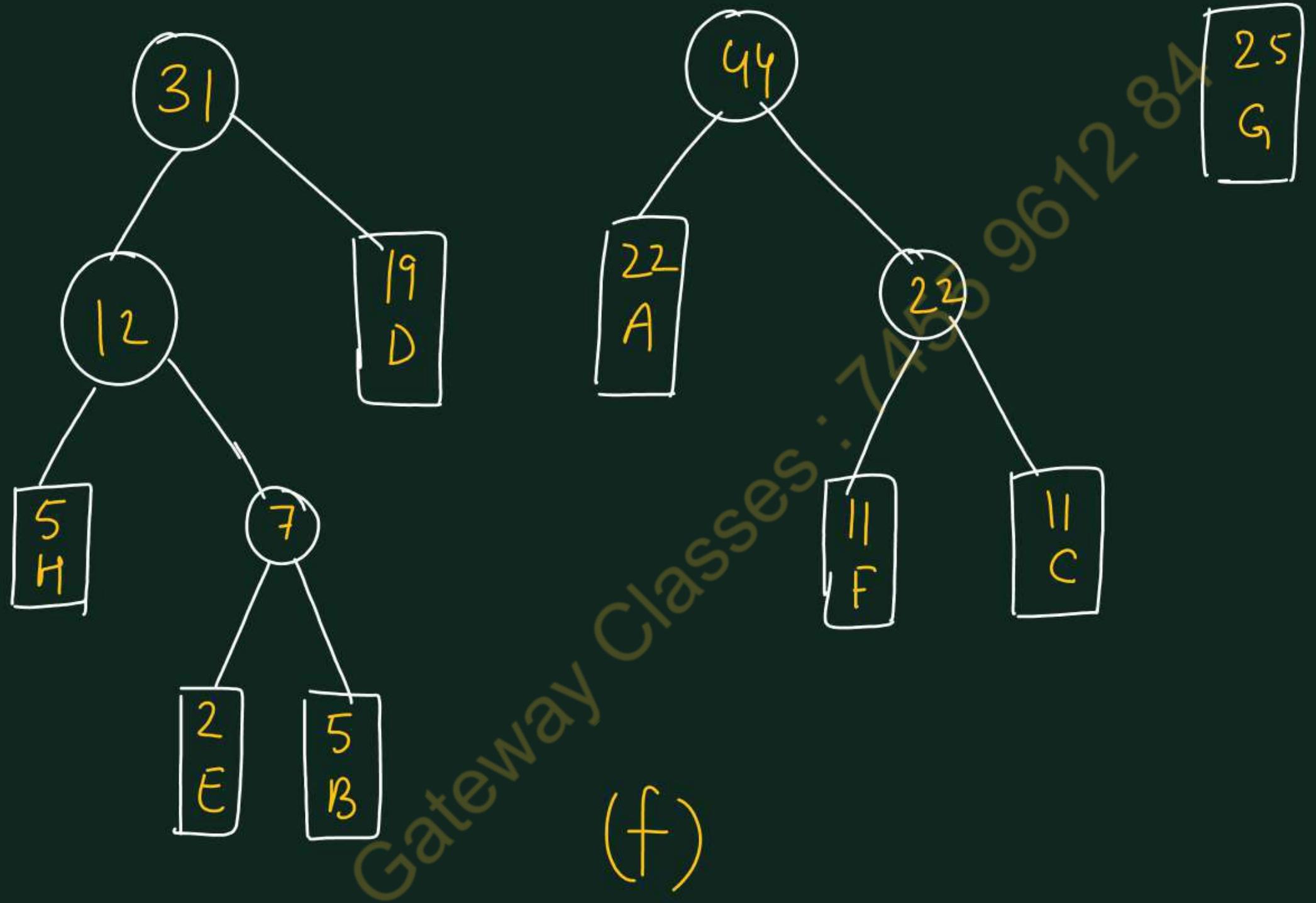
(e)

25
G

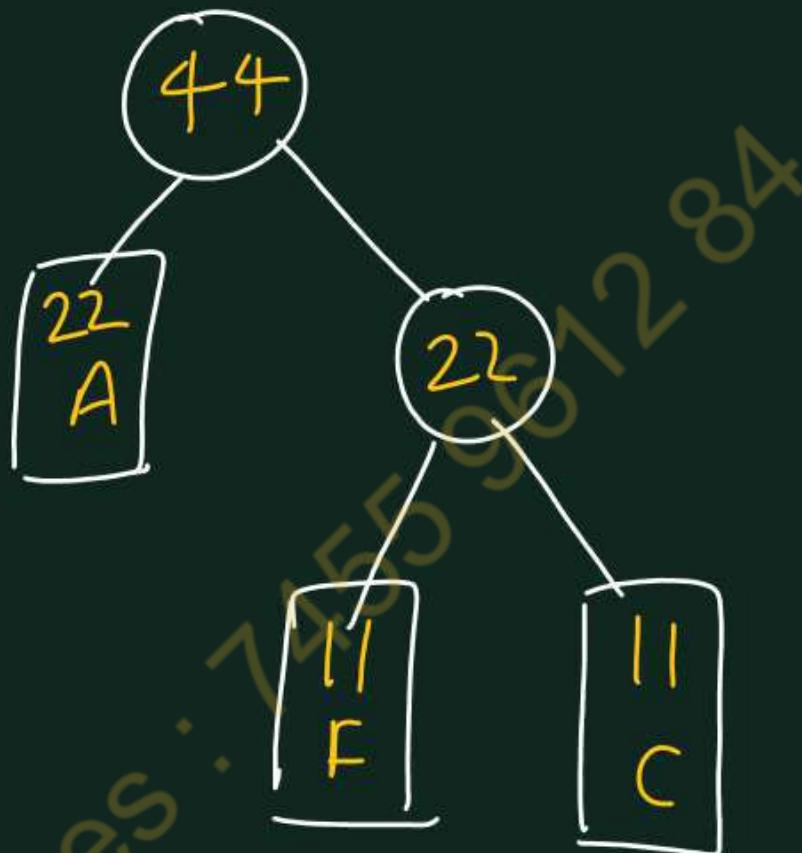
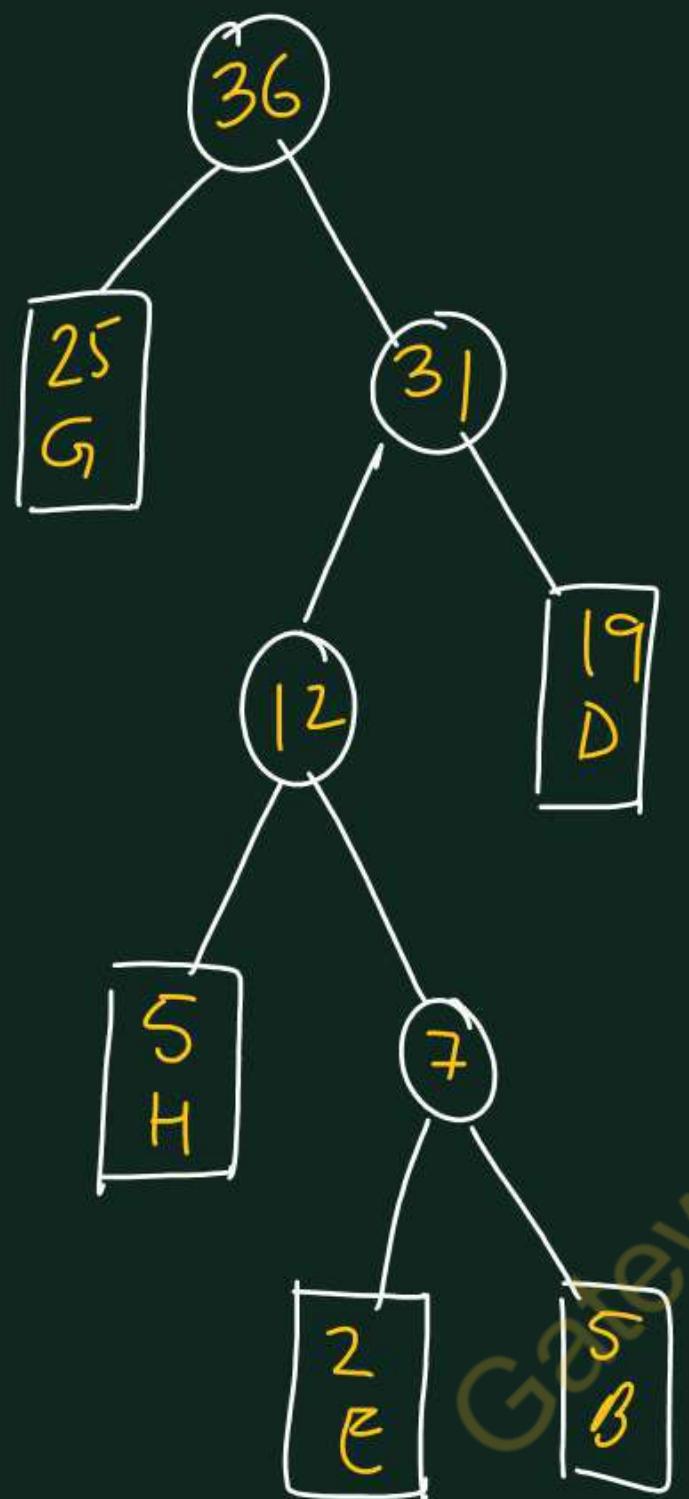
$$\begin{array}{r} 31 \\ 22 \\ 19 \\ \hline 53 \end{array}$$

$$\begin{array}{r} 22 \\ 25 \\ \hline 47 \end{array}$$

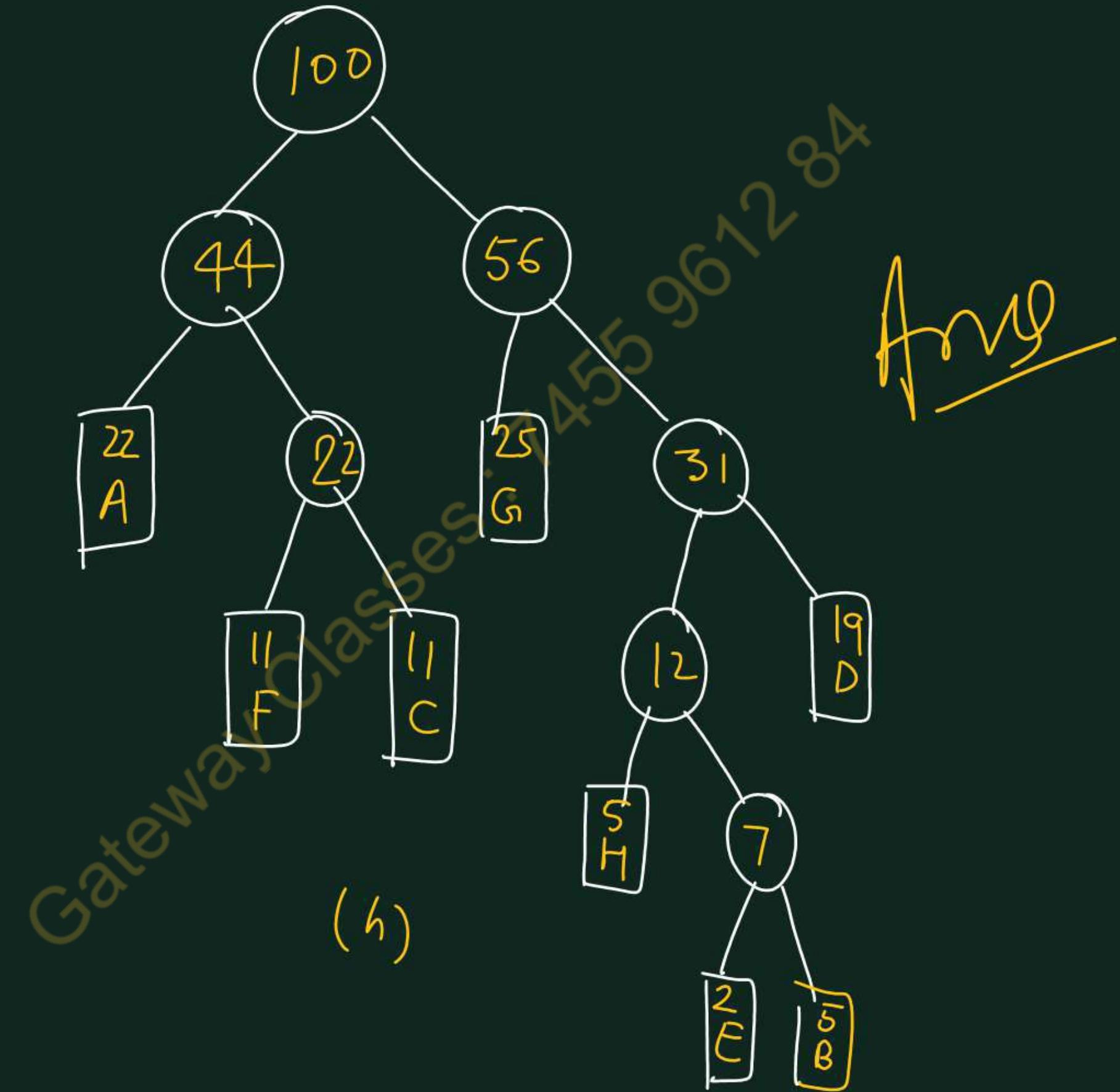
$$\begin{array}{r} 22 \\ 22 \\ \hline 44 \end{array}$$



(f)



(8)



We explain each step separately.

- (a) Here each data item belongs to its own subtree. Two subtrees with the smallest possible combination of weights, the one weighted 2 and one of those weighted 5, are shaded.
- (b) Here the subtrees that were shaded in Fig. 7-35(a) are joined together to form a subtree with weight 7. Again, the current two subtrees of lowest weight are shaded.
- (c) to (g) Each step joins together two subtrees having the lowest existing weights (always the ones that were shaded in the preceding diagram), and again, the two resulting subtree of lowest weight are shaded.
- (h) This is the final desired tree T, formed when the only two remaining subtrees are joined together

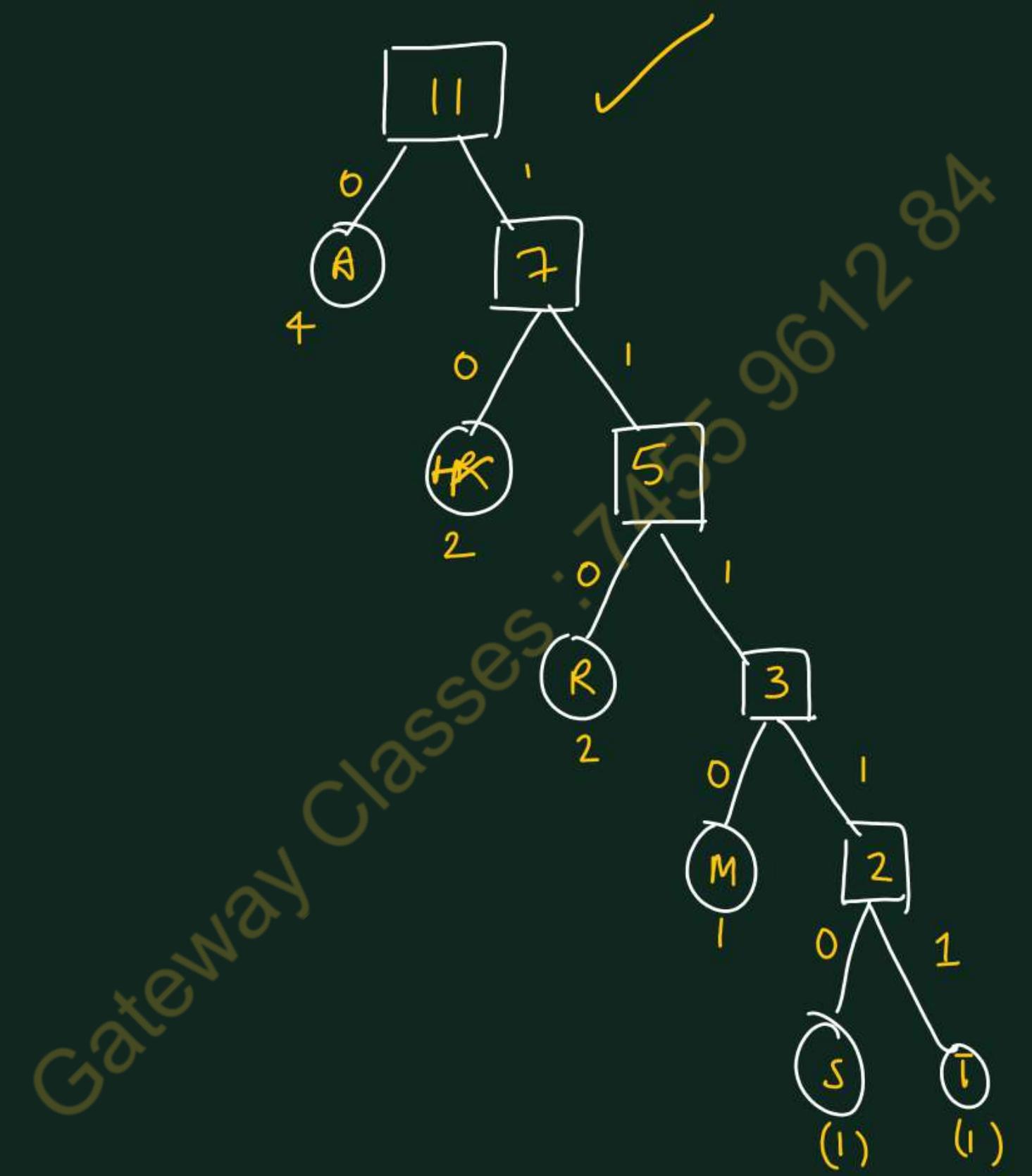
Example:- Construct a Huffman tree for**MAHARASHTRA****With its optional code.****Solution :-**

Total number of symbols in MAHARASHTRA = 6

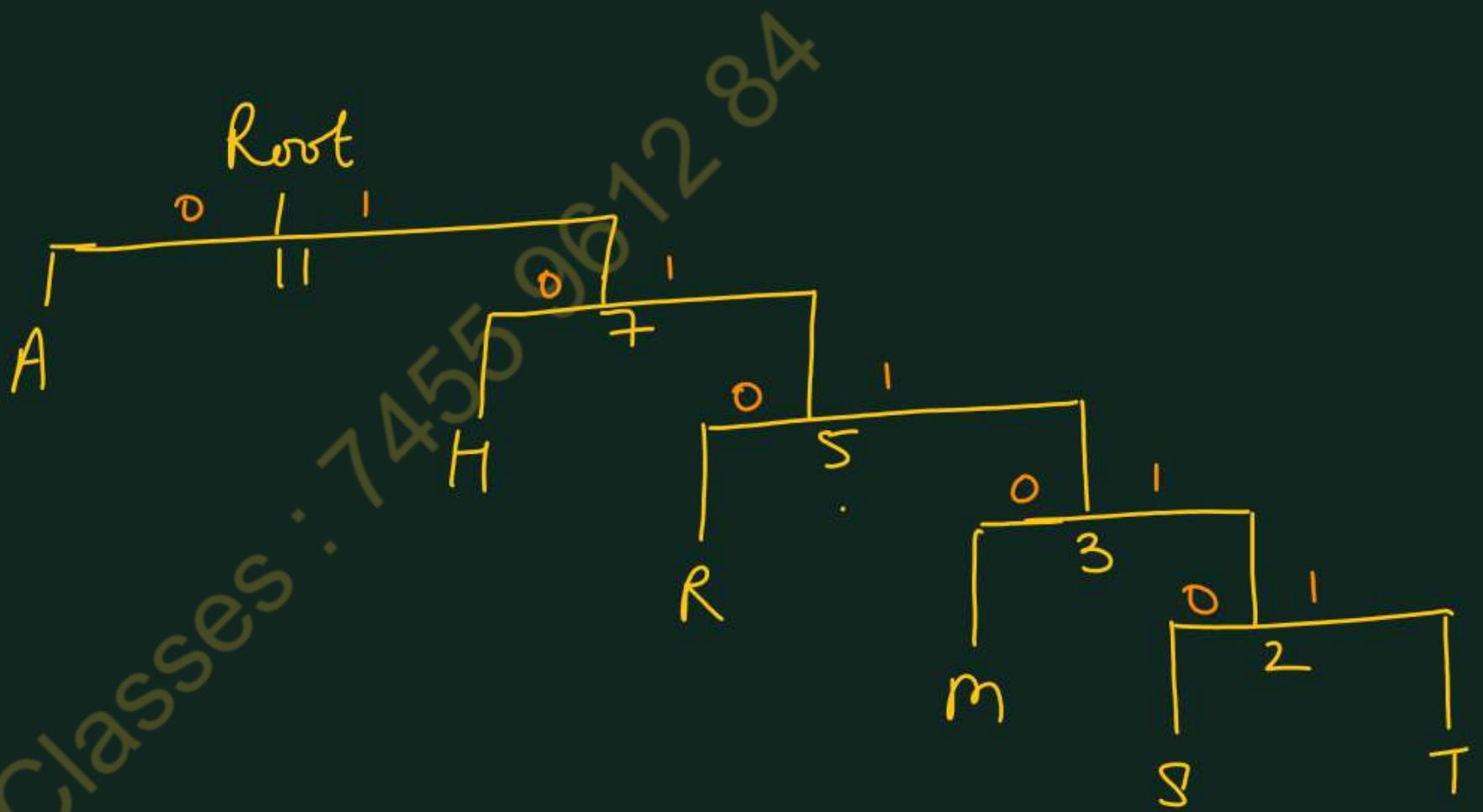
SYMBOL	FREQUENCY
M	1
A	4
H	2
R	2
S	1
T	1

Arrange in
descending
Order

SYMBOL	FREQUENCY
A	4
H	2
R	2
M	1
S	1
T	1



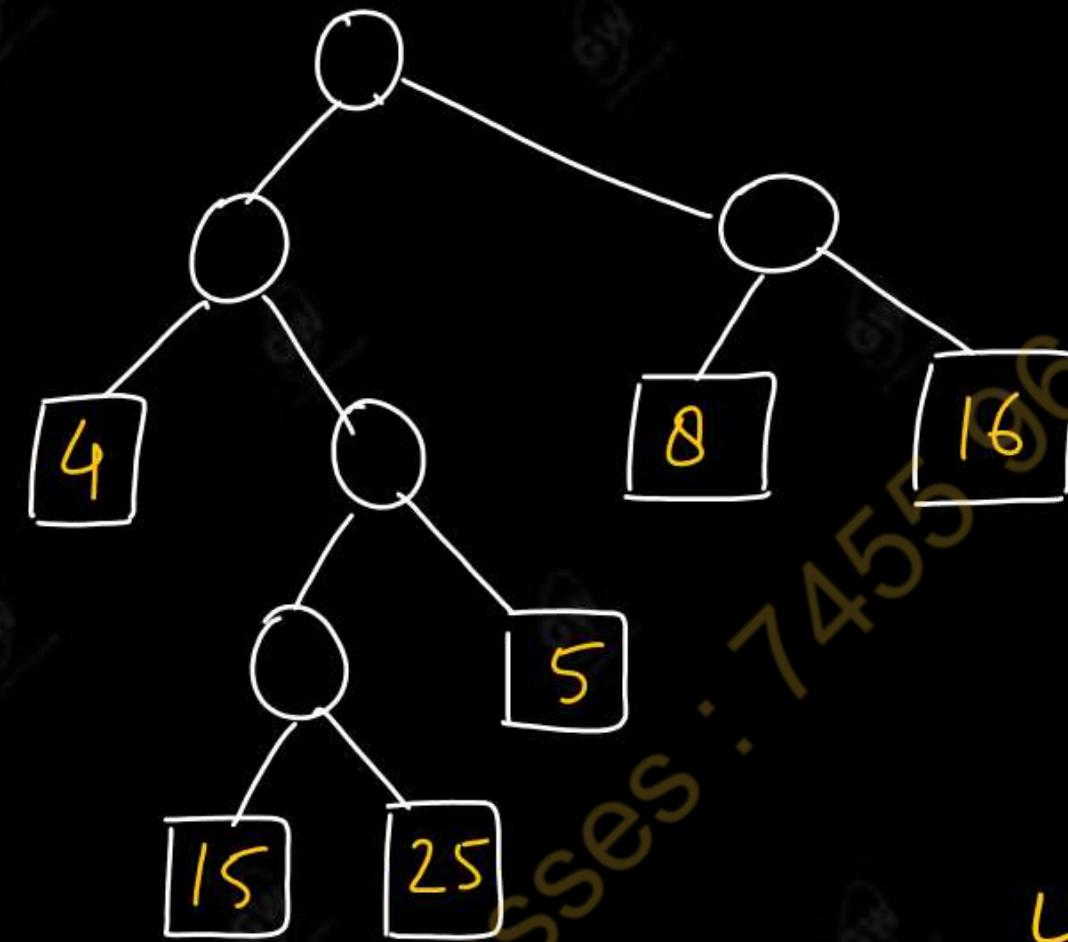
Huffman Tree is -



Codes for all the symbols -

SYMBOLS	CODE
A	0
H	1 0
R	11 0
M	111 0
S	1111 0
T	11111

Example:- Consider the following weighted 2-tree:-



Find the weighted path length P of the tree T. (Ans = 231)

Solution:-

$$\begin{aligned} P &= 4 \times 2 + 15 \times 4 + 25 \times 4 + 5 \times 3 + 8 \times 2 + 16 \times 2 \\ &= 80 + 60 + 100 + 15 + 16 + 32 \\ &= \underline{\underline{231}} \end{aligned}$$

A general tree (sometimes called a tree) is defined to be a nonempty finite set T of elements, called nodes, such that:

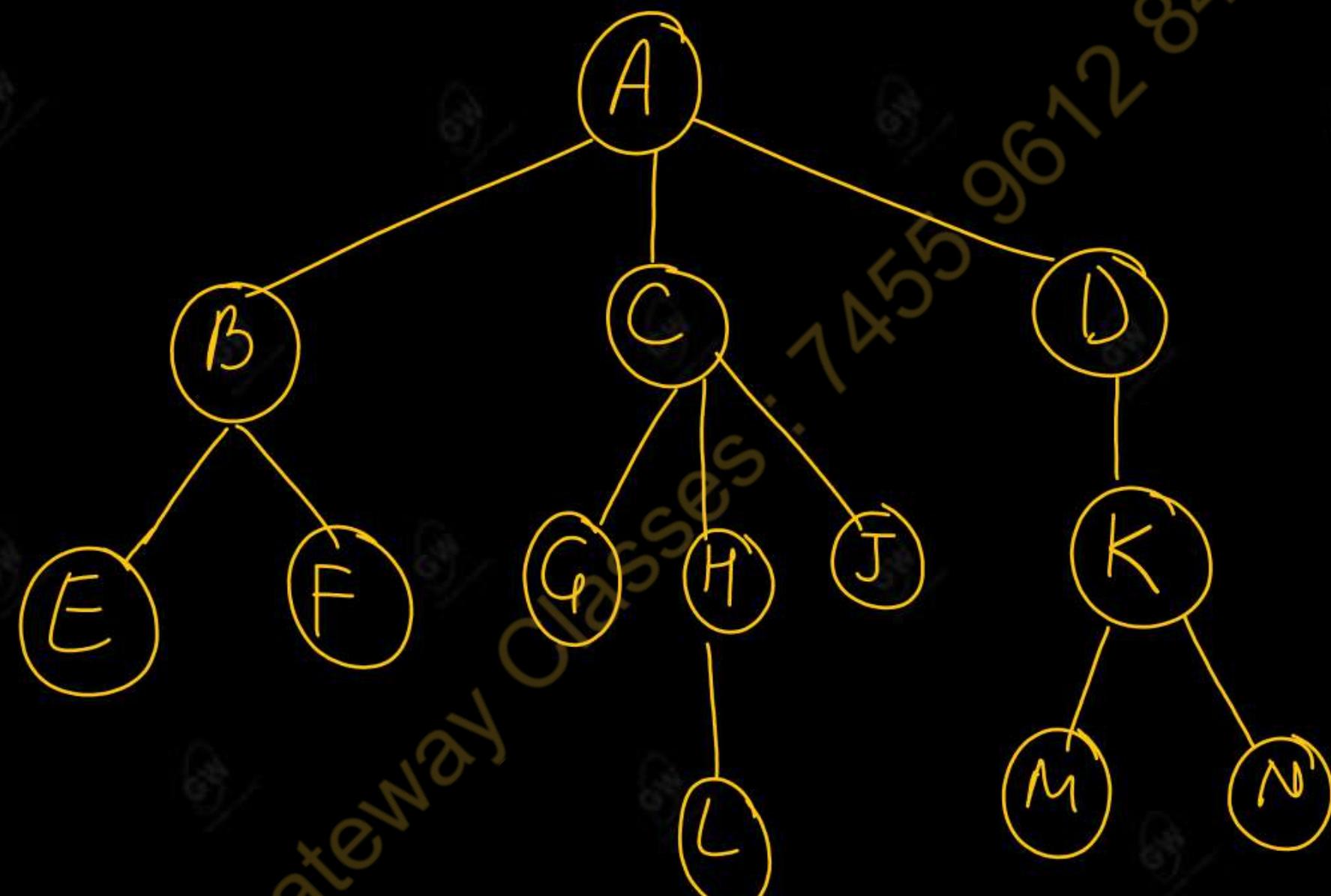
- (1) T contains a distinguished element R , called the root of T .
- (2) The remaining elements of T form an ordered collection of zero or more disjoint trees T_1, T_2, \dots, T_m .

The trees T_1, T_2, \dots, T_m are called subtrees of the root R , and the roots of T_1, T_2, \dots, T_m are called successors of R .

{ Binary Tree - 0, 1, 2 }

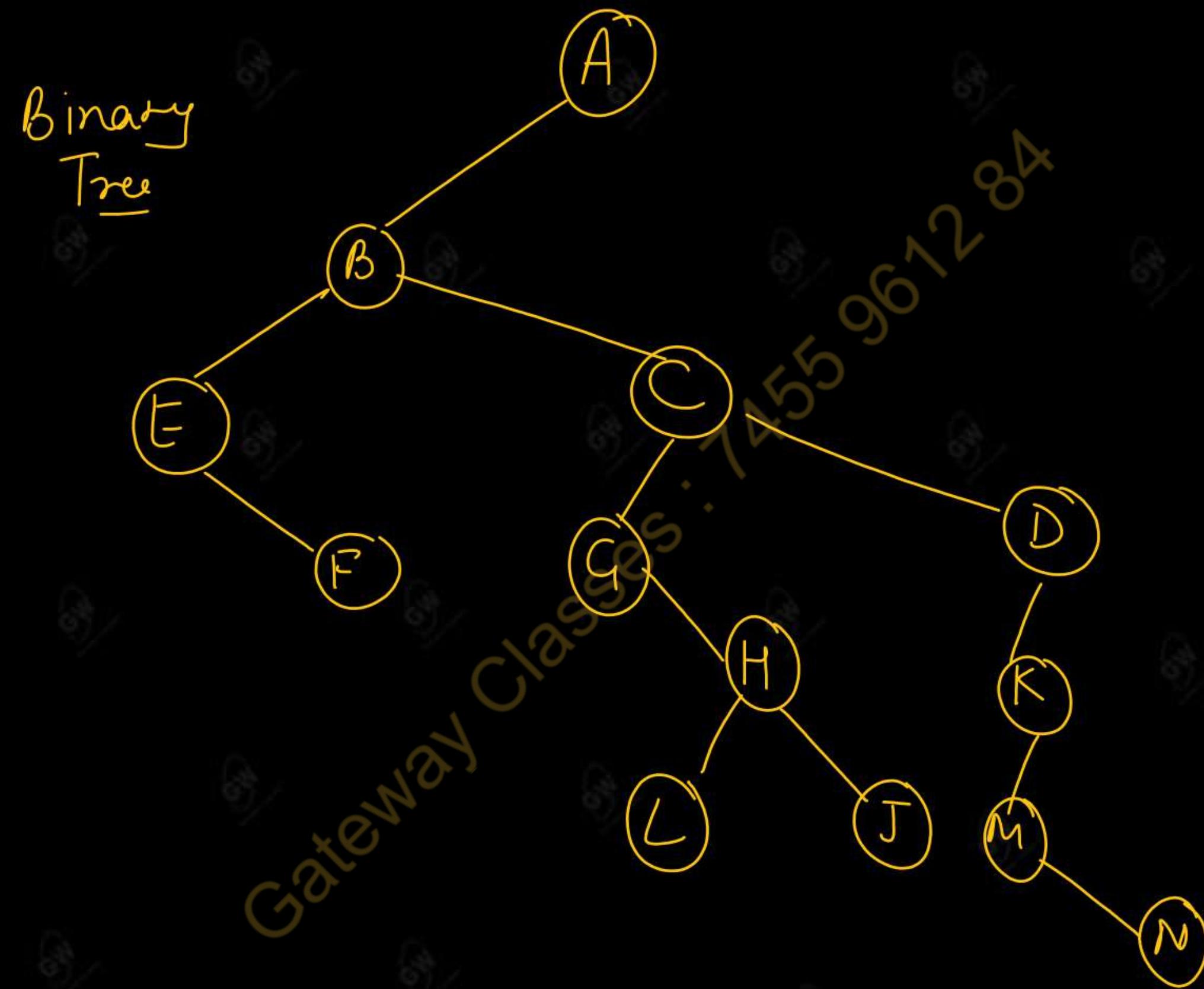
EXAMPLE : Consider a following general tree T with 13 nodes,

A, B, C, D, E, F, G, H, J, K, L, M, N

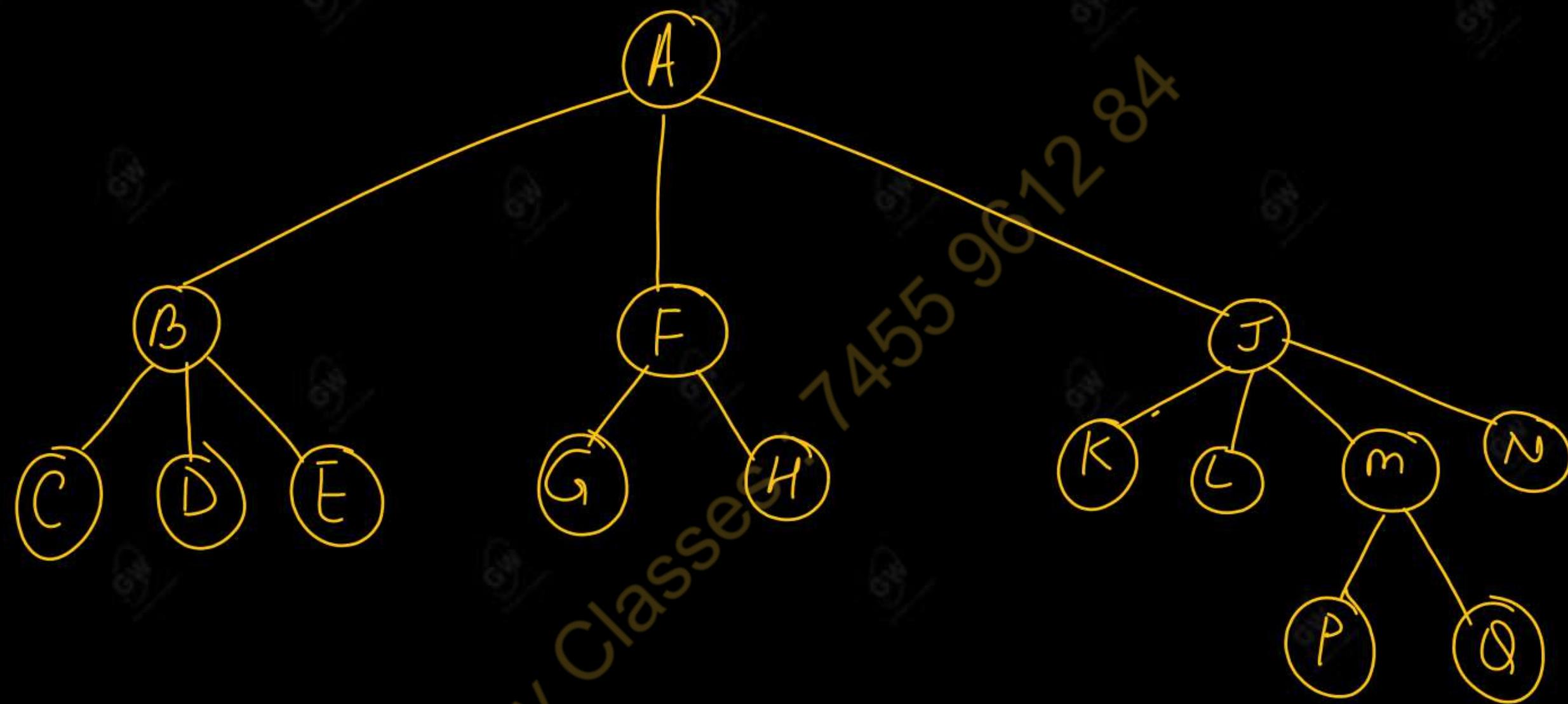


Correspondence between General Trees and Binary Trees

- Suppose T is a general tree.
- Then we may assign a unique binary tree T' to T as follows.
- First of all, the nodes of the binary tree T' will be the same as the nodes of the general tree T, and the root of T' will be the root of T.
- Let N be an arbitrary node of the binary tree T'. Then the left child of N in T' will be the first child of the node N in the general tree T and the right child of N in T' will be the next sibling of N in the general tree T.
- So the binary tree of previous general tree will be as follows:-



Example:- Consider the following general tree:-



Find the corresponding binary tree.



Q.1 Write short note on Huffman Algorithm.

2016-17, 5 marks

Q.2 Construct a Huffman tree for given characters A, B, C, D, E, F, G, H having frequencies 22, 5, 11, 19, 2, 11, 25, 5 respectively. What will be the code of HEAD in binary?

2018-19, 7 marks

Q.3 Explain Huffman algorithm? Construct Huffman tree for MAHARASHTRA with its optional code.

2018-19, 7 MARKS

Q.4 Explain all steps used to build Huffman Tree where following characters are given with their frequencies:

a:5, b:9, c:12, d:13, e:16, f:45

2020-21, 7 marks

Unit - 4 : Lec - 5

Today's Target

- AVL Tree (Its importance, Balance Factor, Its representation, Insertion, 4 Rotations (LL, RR, LR, RL), Deletion)
- AKTU PYQs

Explain the problem and give the solution.

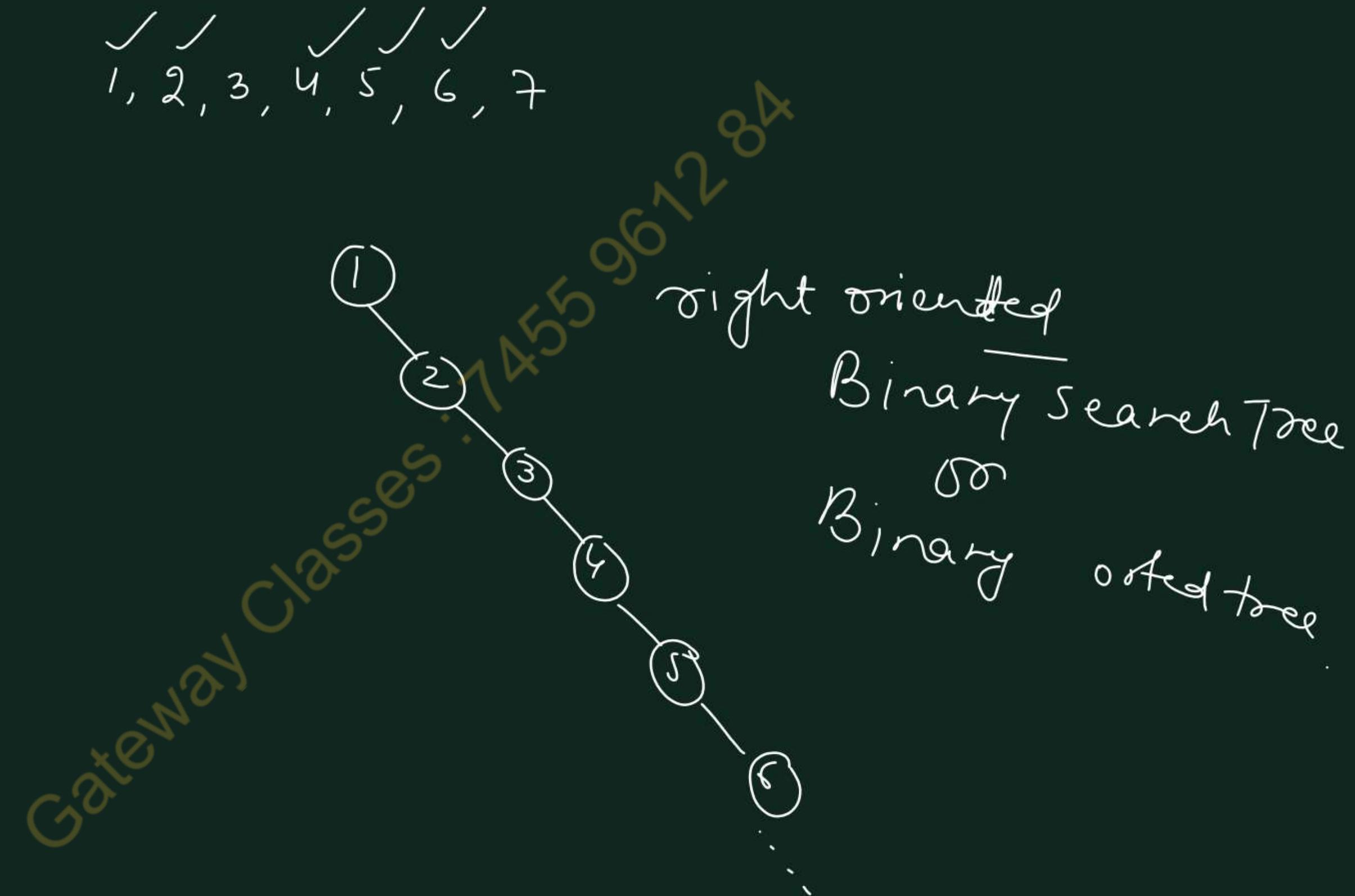
Solution:-

- In a binary search tree (BST), the orientation of the tree refers to the arrangement of nodes concerning their parent nodes.
- The terms "left-oriented" and "right-oriented" indicate the direction in which child nodes are positioned concerning their parent nodes.

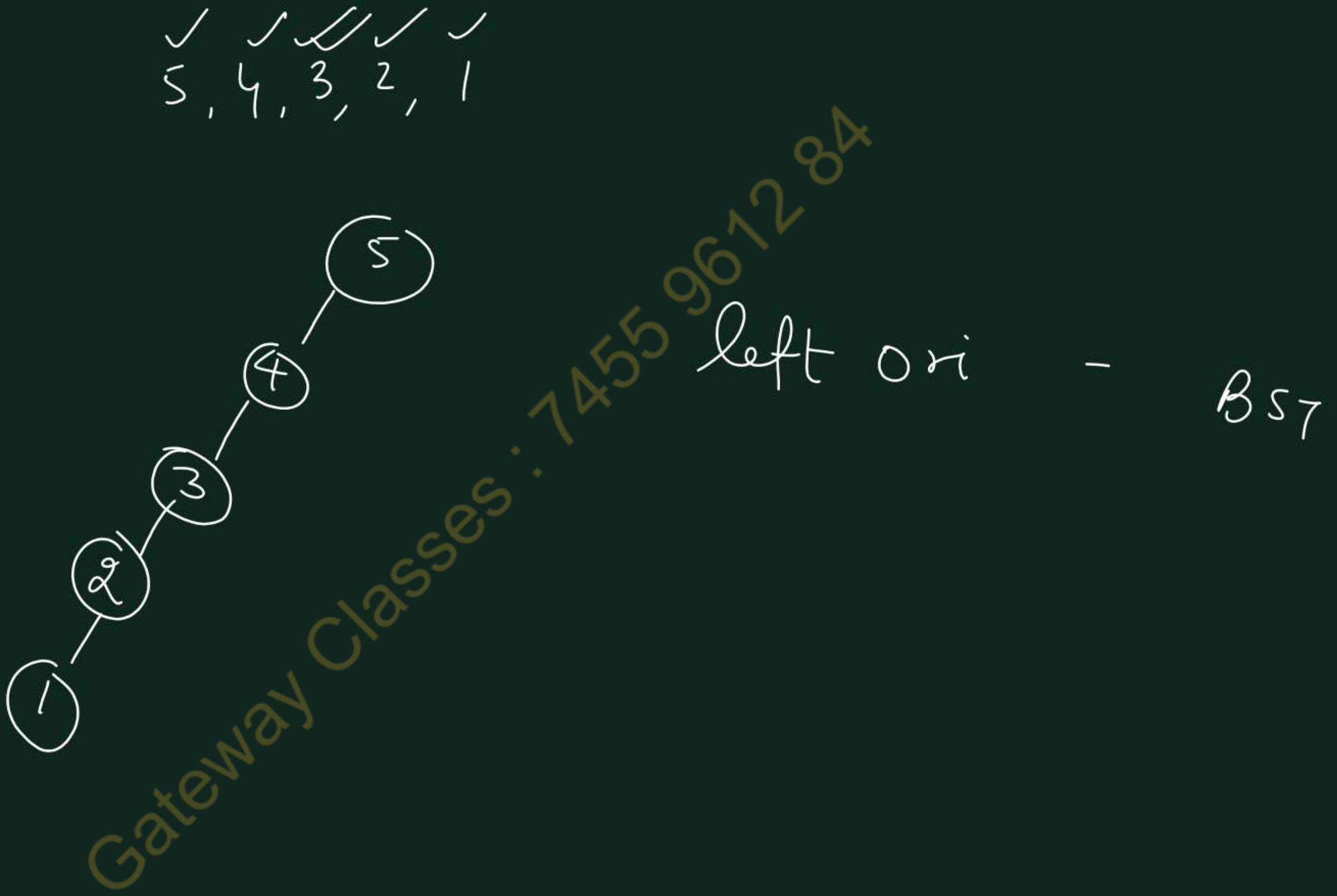
Left-Oriented Binary Search Tree:

Problem:

- In a left-oriented BST, each node has only a left child, and there are no right children.
- This leads to an imbalanced tree where the height of the tree could be significantly greater on the left side compared to the right side.



/ / /
1, 2, 3, 4, 5, 6, 7



- As a result, the time complexity of basic operations like search, insert, and delete can degrade to $O(n)$ in the worst case, where n is the number of nodes.

Solution:

- To maintain the efficiency of a BST, it is crucial to keep the tree balanced.
- Various balancing techniques, such as AVL trees or Red-Black trees, can be employed to ensure that the height of the tree remains logarithmic, providing efficient search, insert, and delete operations ($O(\log n)$).

Right-Oriented Binary Search Tree:

Problem:

- Similar to a left-oriented tree, a right-oriented BST has nodes with only right children, leading to an imbalanced tree with a skewed height on the right side.
- This can also result in inefficient search and other operations.

Solution:

- The solution is again to maintain balance in the tree.
- Balancing techniques like AVL trees or Red-Black trees can be applied to ensure that the tree remains balanced, and the time complexity of operations remains efficient.
- The main issue with left-oriented or right-oriented binary search trees is that they can become highly unbalanced, leading to poor performance in terms of time complexity for basic operations.
- AVL trees, Red-Black trees, and other self-balancing binary search tree structures are commonly used to address these issues and maintain the desired logarithmic time complexity.

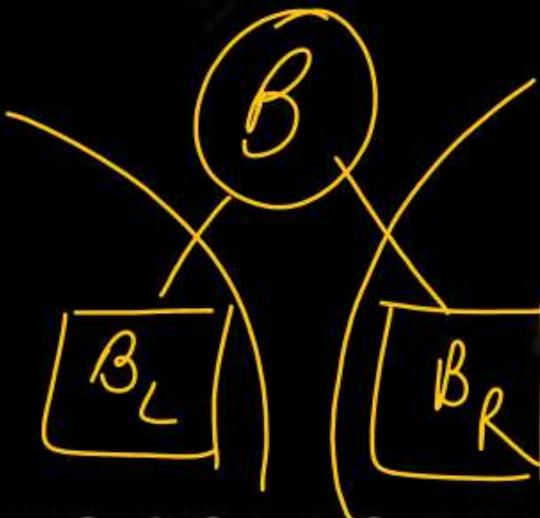
- The first balanced binary search tree was the AVL tree (named after its discoverers, Adelson Velskii and Landis).
- The AVL tree is a binary search tree that has an additional balance condition.
- The idea is to require that the left and right sub-trees have the same height.
- Recursion dictates that this idea applies to all nodes in the tree, since each node is itself a root of some sub-tree.
- This balance condition ensures that the depth of the tree is logarithmic, but it is too restrictive because it is too difficult to insert new items while maintaining balance.
- Thus the AVL trees uses a notion of balance that is somewhat weaker but still strong enough to guarantee logarithmic depth.

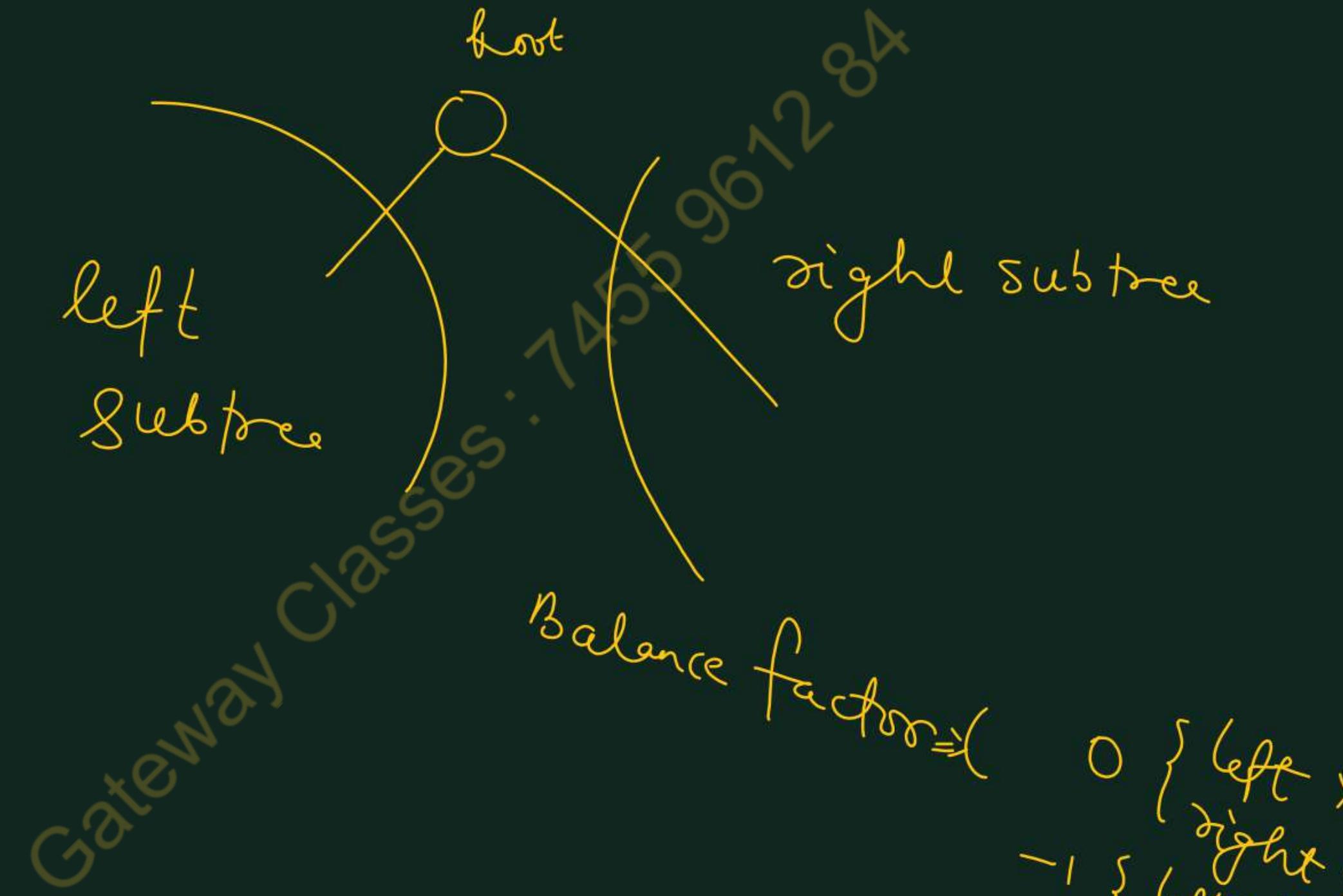
Height Balanced Tree

- AVL is a height balanced tree.
- A height balanced tree is one in which the difference in the height of the two sub-trees for any node is less than or equal to some specified amount.
- In AVL the height difference may be no more than 1.

Definition :-

- An empty binary tree B is an AVL tree.
- If B is the non-empty binary tree with B_L and B_R are its left and right sub-trees then B is an AVL tree if and only if
 - (a) B_L and B_R are AVL trees, and
 - (b) $|h_L - h_R| \leq 1$ { absolute of height of left – height of right , i.e. -1, 0, 1}
- Where h_L and h_R are the heights of B_L and B_R sub-tress, respectively.



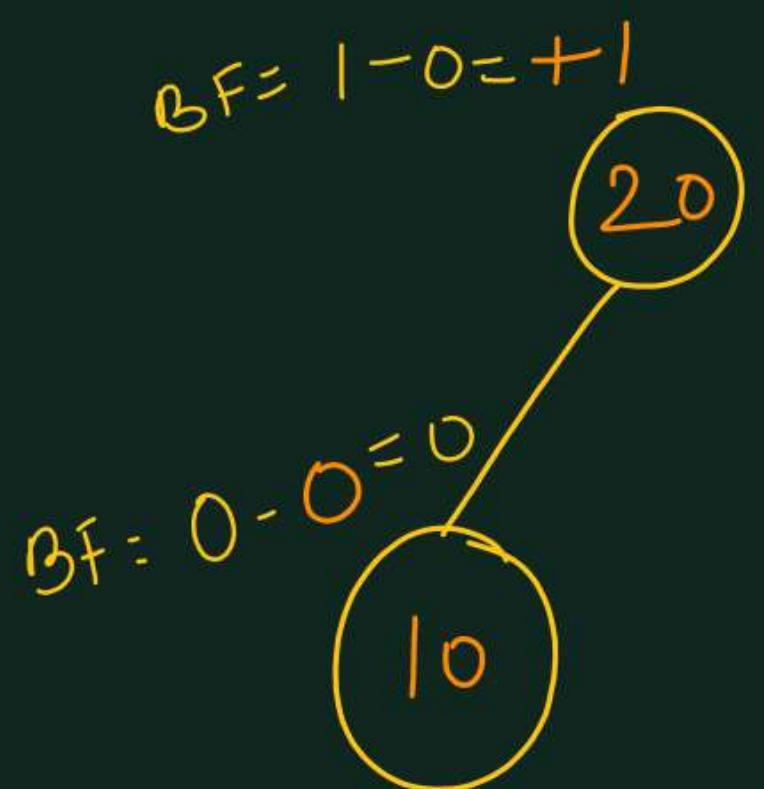


0	$\{$	$(\text{left } x)$	$\}$
-1	$\{$	$(\text{right } x)$	$\}$
$+1$	$\{$	$(\text{left } x)$	$\}$

Search

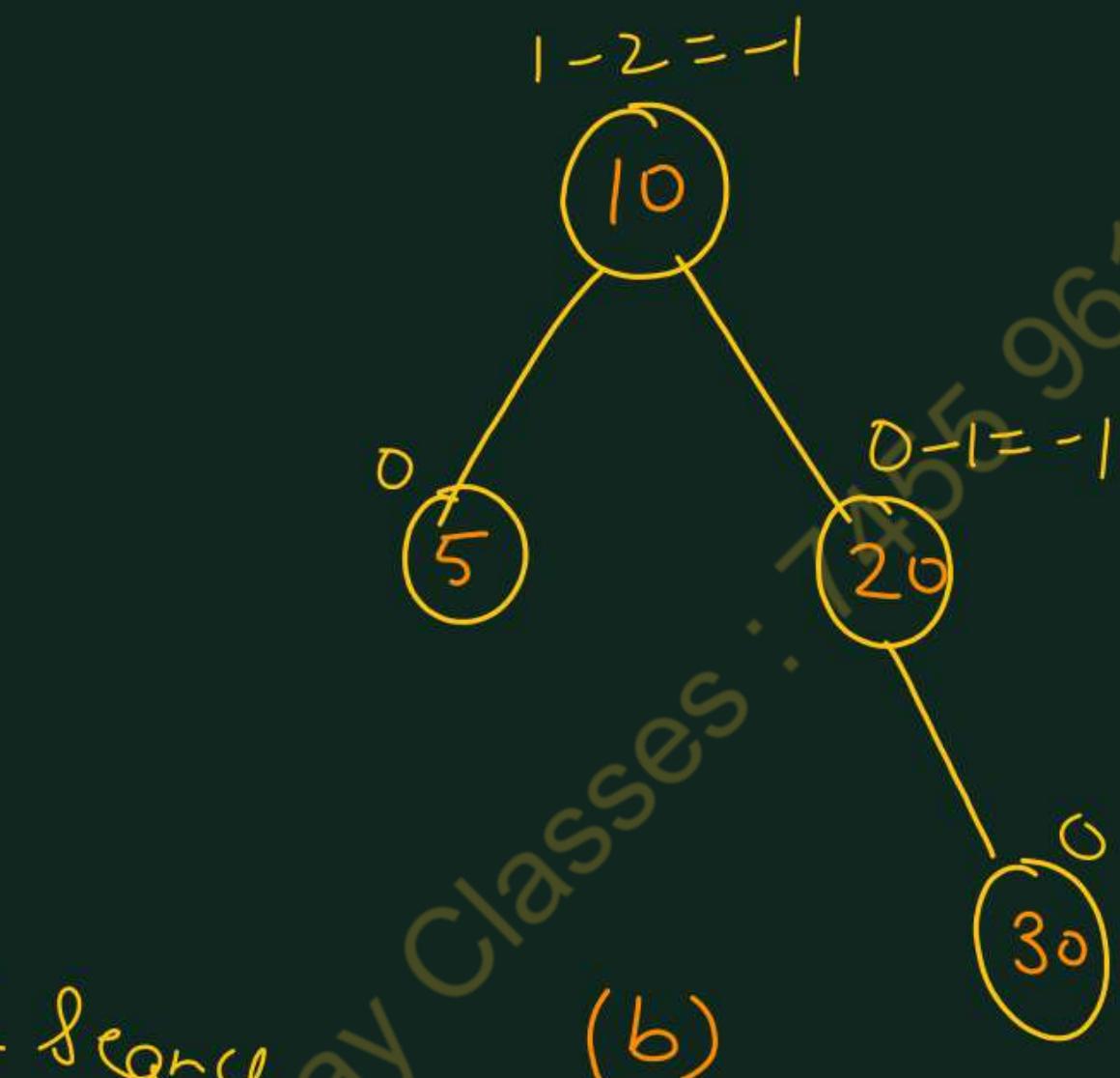
- To implement an AVL tree each node must contain a balance factor, which indicates its states of Balance relative to its sub-trees.
- If balance is defined by -

$$\text{Balance Factor} = \text{Height of left sub-tree} - \text{Height of right sub-tree} = \{-1, 0, 1\}$$
- Then the balance factors in a balanced tree can have values of -1, 0 or 1.
- In figure, numbers placed near the nodes indicate balance factors.



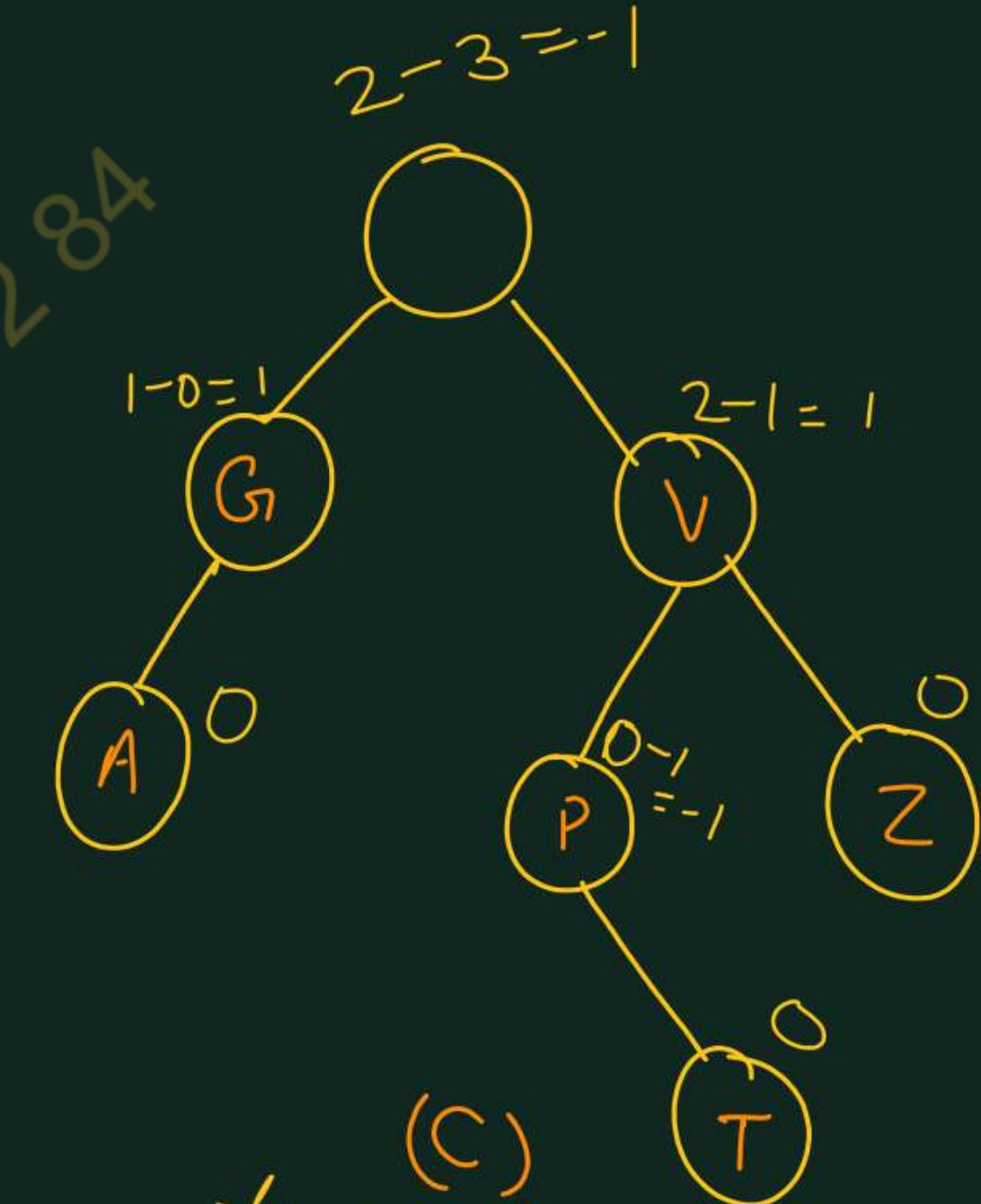
(a)

AUC second
T2C



(b)

AUC



(c)

AUC first

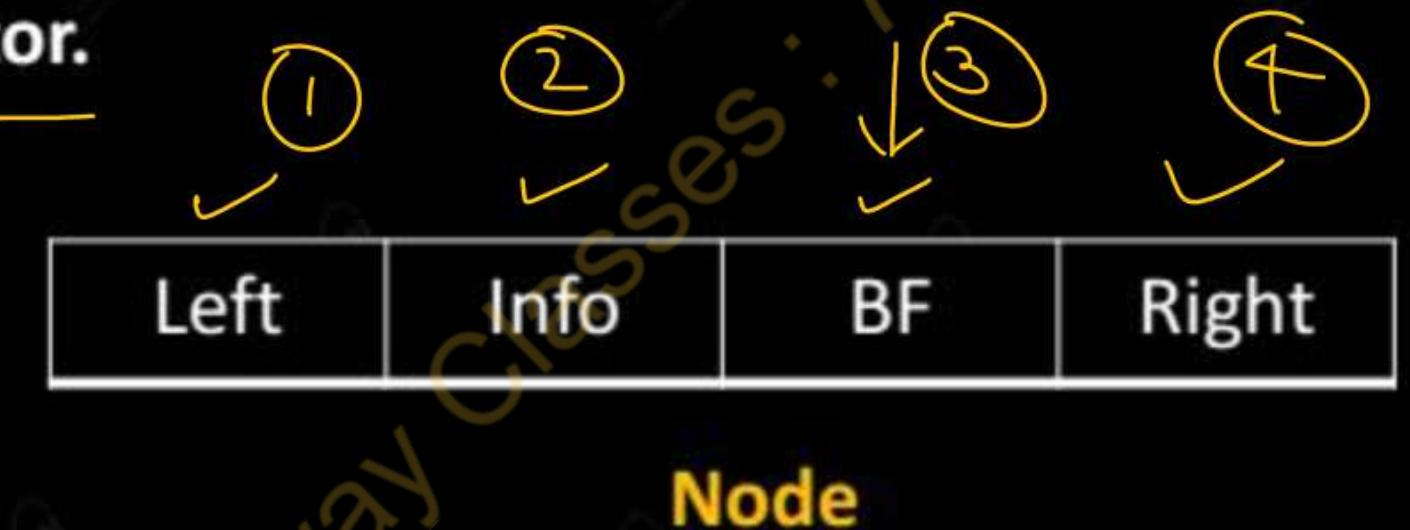
Gateway Classes:

- For an AVL tree, the value of the balance factor of any node is -1, 0, or 1.
- If it is other than these three values then the tree is not balanced or it is not an AVL tree.
- If the value of the balance factor of any node is -1, then the height of the right sub-tree of that node is one more than the height of its left sub-tree.
- If the value of the balance factor of any node is 0 then the height of its left and right sub-tree is exactly the same.
- If the value of the balance factor of any node is +1 then the height of the left subtree of that node is one more than the height of its right sub-tree.

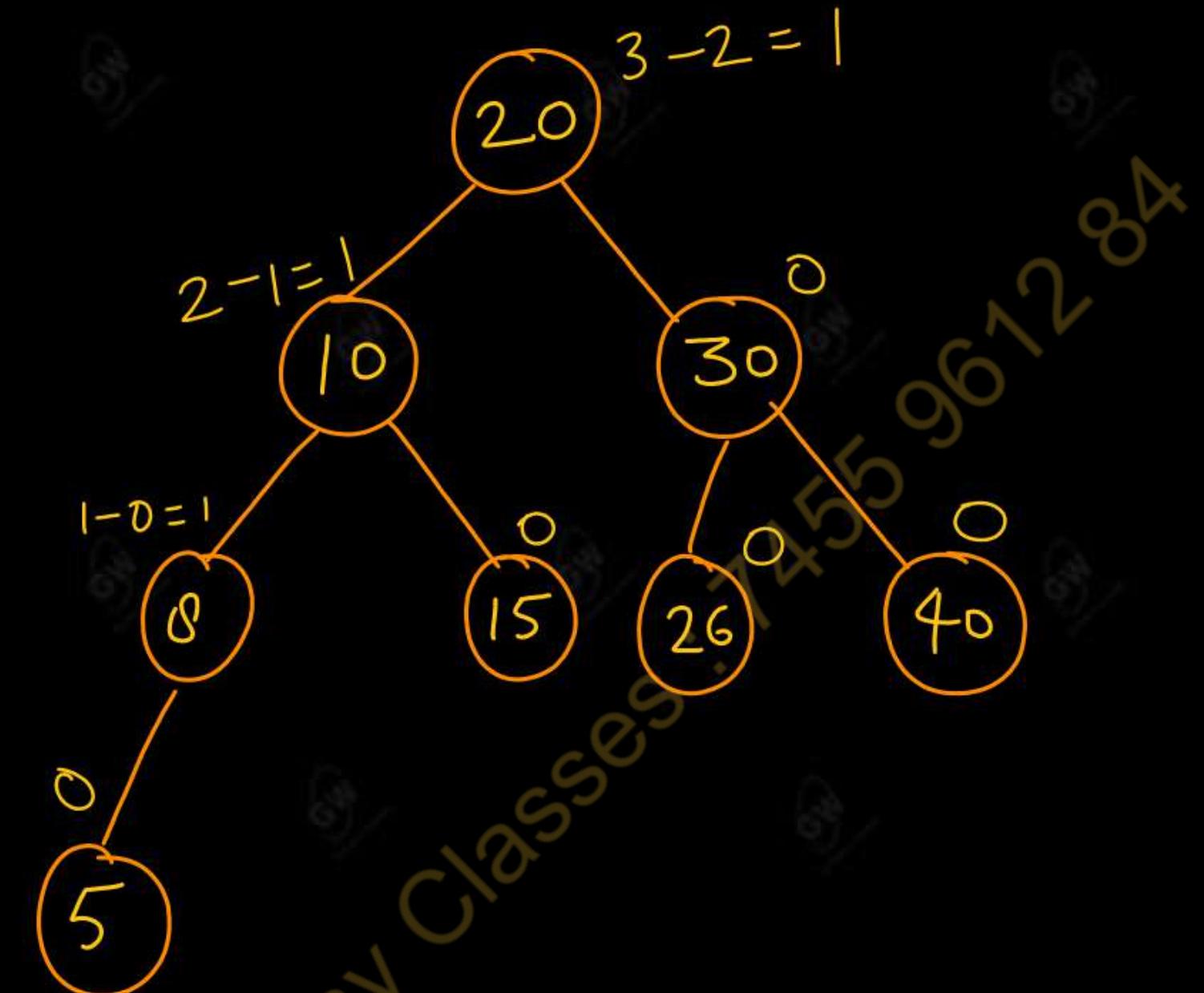
REPRESENTATION OF AN AVL TREE

AVL search trees like binary search trees are represented using a linked representation. However, every node consists of its balance factor.

So, to represent a node of an AVL tree four fields are required-one for data, two for storing the address of the left and right child and an additional field is required to hold the balance factor.



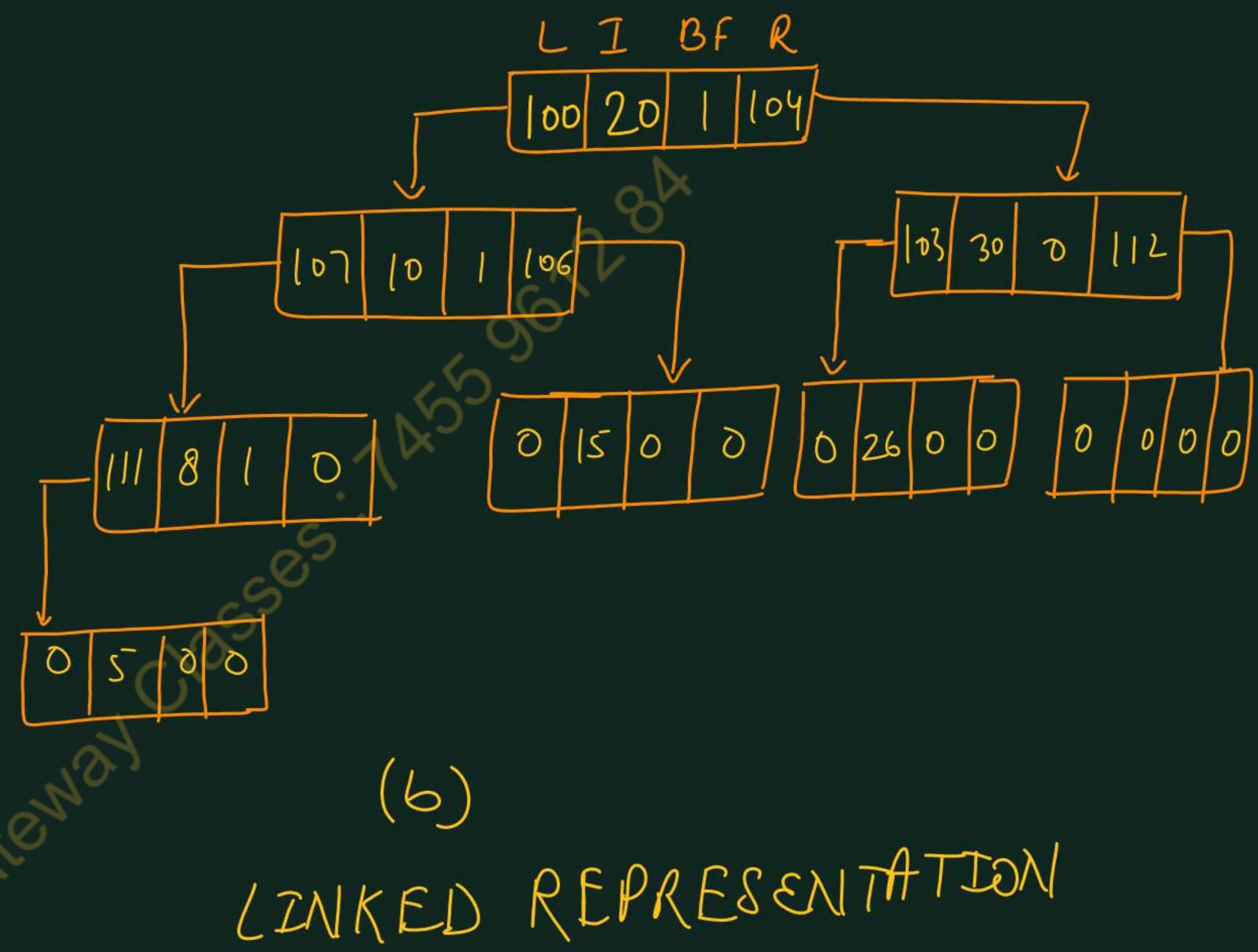
The next figure shows the representation of an AVL tree.



AVL TREE

	LEFT	INFO	BF	RIGHT
100	107	10	1	106
101	100	20	+1	104
102				
103	0	26	0	0
104	103	30	0	112
105				
106	0	15	0	0
107	111	8	1	0
108				
109				
110	0	5	0	0
111	0	40	0	0
112				
113				

(a)



LINKED REPRESENTATION

(b)

The number against each node represents its Balance Factor (BF). The linked representation of above tree is also shown in figure(5).

INSERTION IN AN AVL SEARCH TREE

□ We can insert a new node in an AVL tree by finding its appropriate position similar to that of the binary search tree.

□ But insertion of a new node involves certain overheads since the tree may become unbalanced due to the increase in its height.

1. If the data item with key 'k' is inserted into an empty AVL tree, then the node with key k is set to be the root node. In this case, the tree is height balanced.

2. If tree contains only single node, i.e., root node, then the insertion of the node with key 'k' depends upon its value. If the value of 'k' is less than the key value of the root then it is left of the root otherwise right of the root. In this case the tree is height balanced.

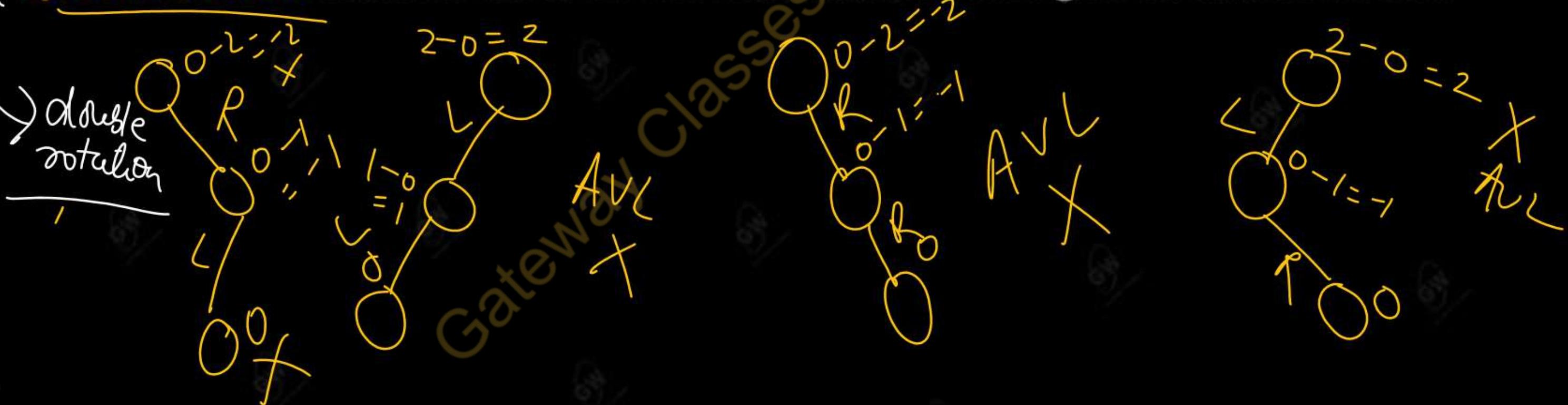
3. If on inserting the node with key 'k' the height of the right or left sub-tree of the root has increased then to maintain the balance factor "Rotations" can be performed.

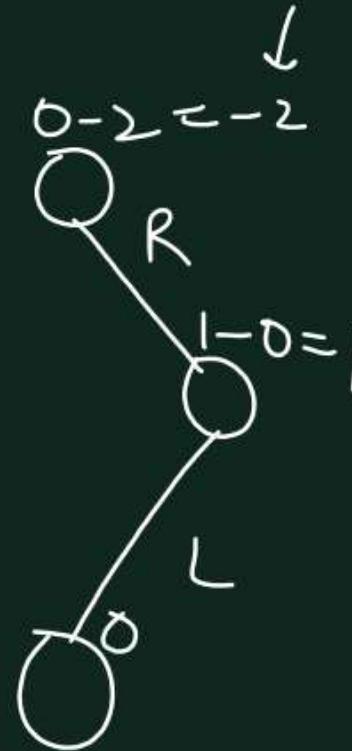
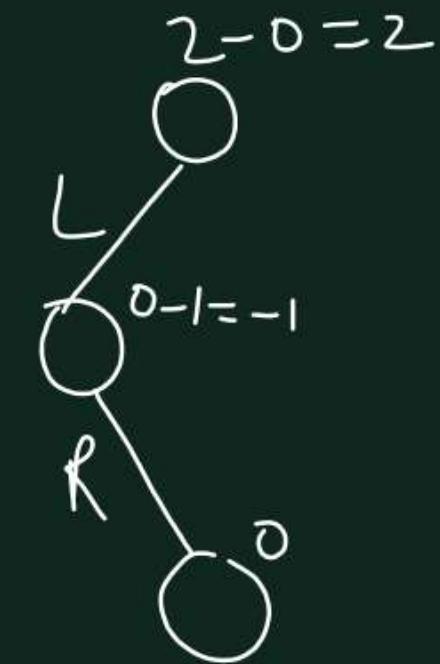
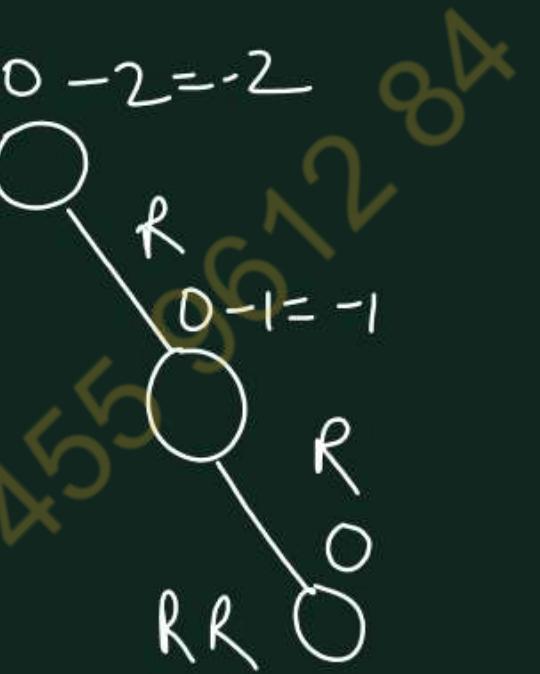
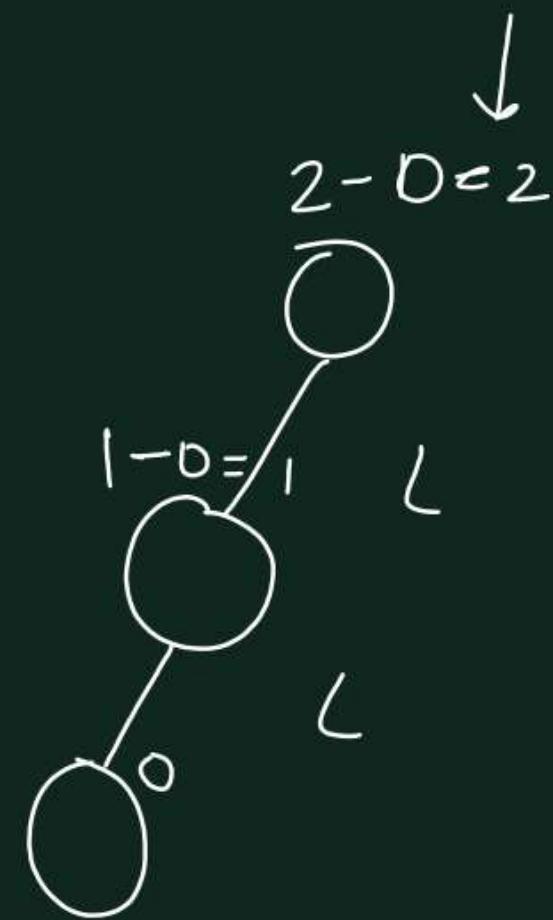
$a_1, b \Rightarrow \text{single}$

AVL Tree Rotations

In order to balance a tree, there are four cases of rotations, such as:

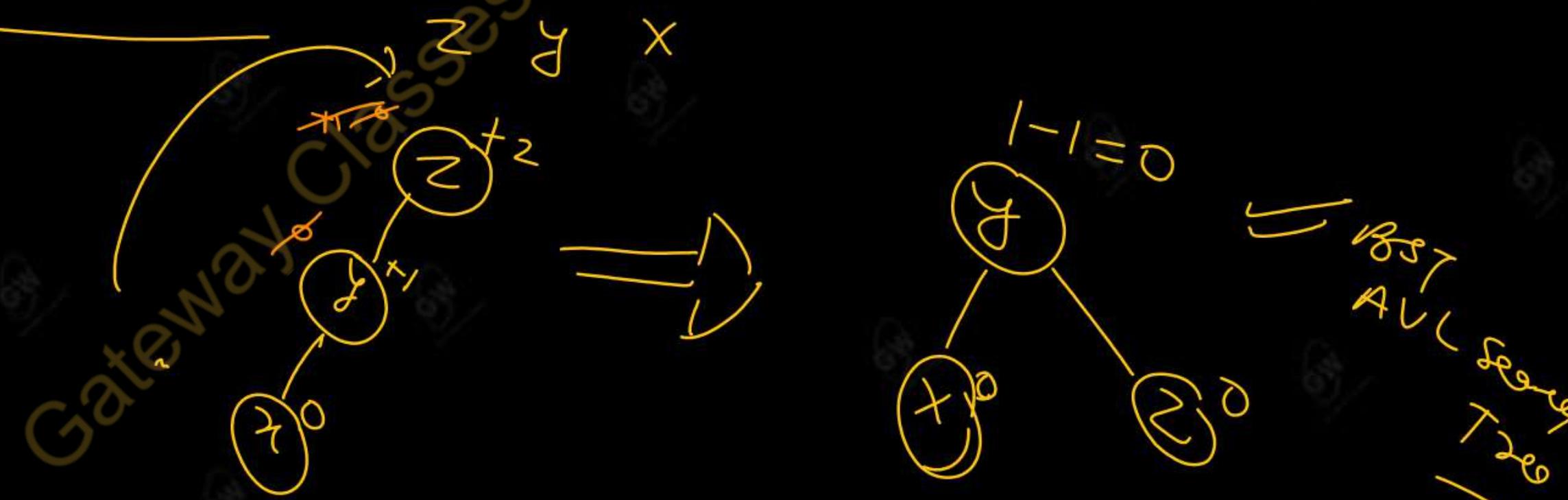
- a) **LL Rotation** : Inserted node is in the left sub-tree of left sub-tree of node.
- b) **RR Rotation** : Inserted node is in the right sub-tree of right sub-tree of node.
- c) **LR Rotation** : Inserted node is in the right sub-tree of left sub-tree of node.
- d) **RL Rotation** : Inserted node is in the left sub-tree of right sub-tree of node.

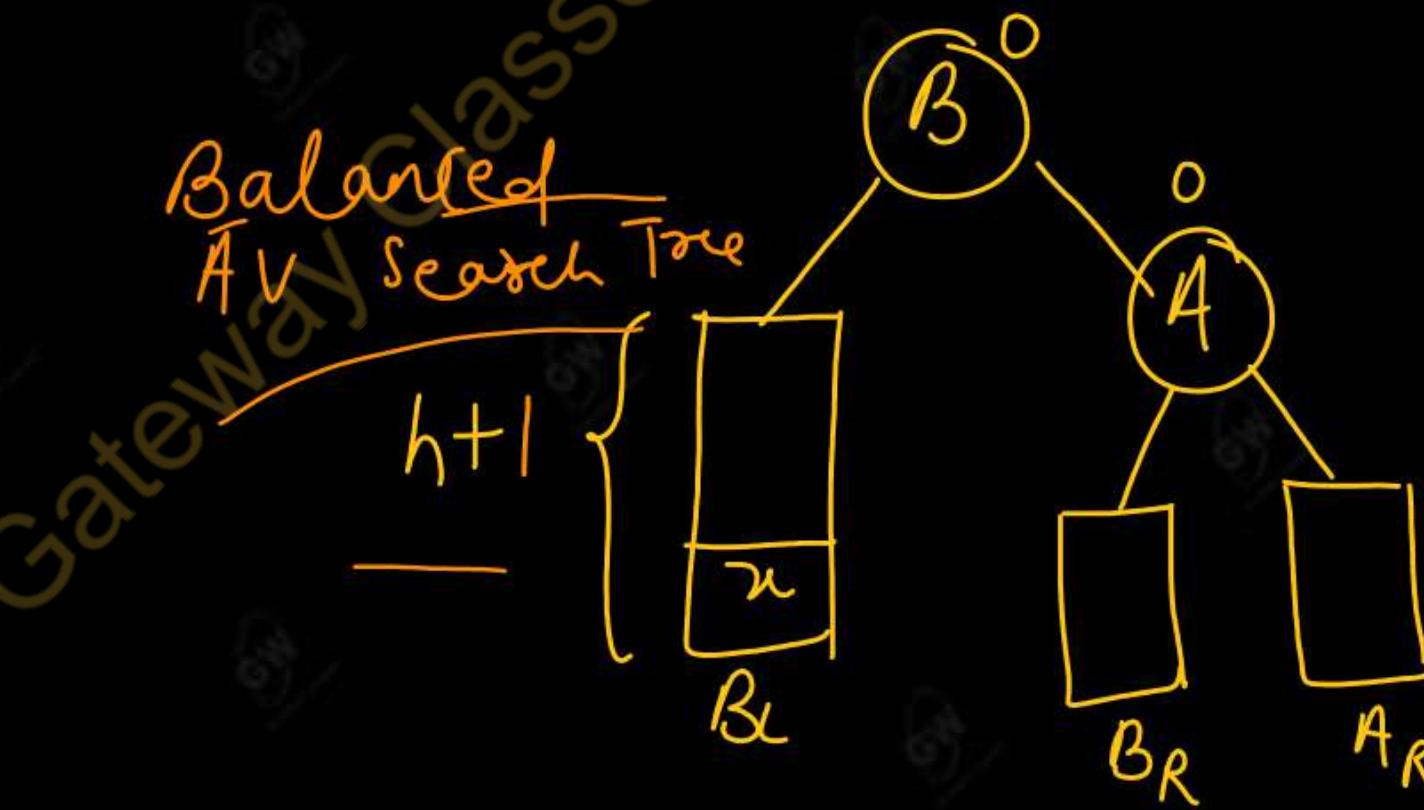
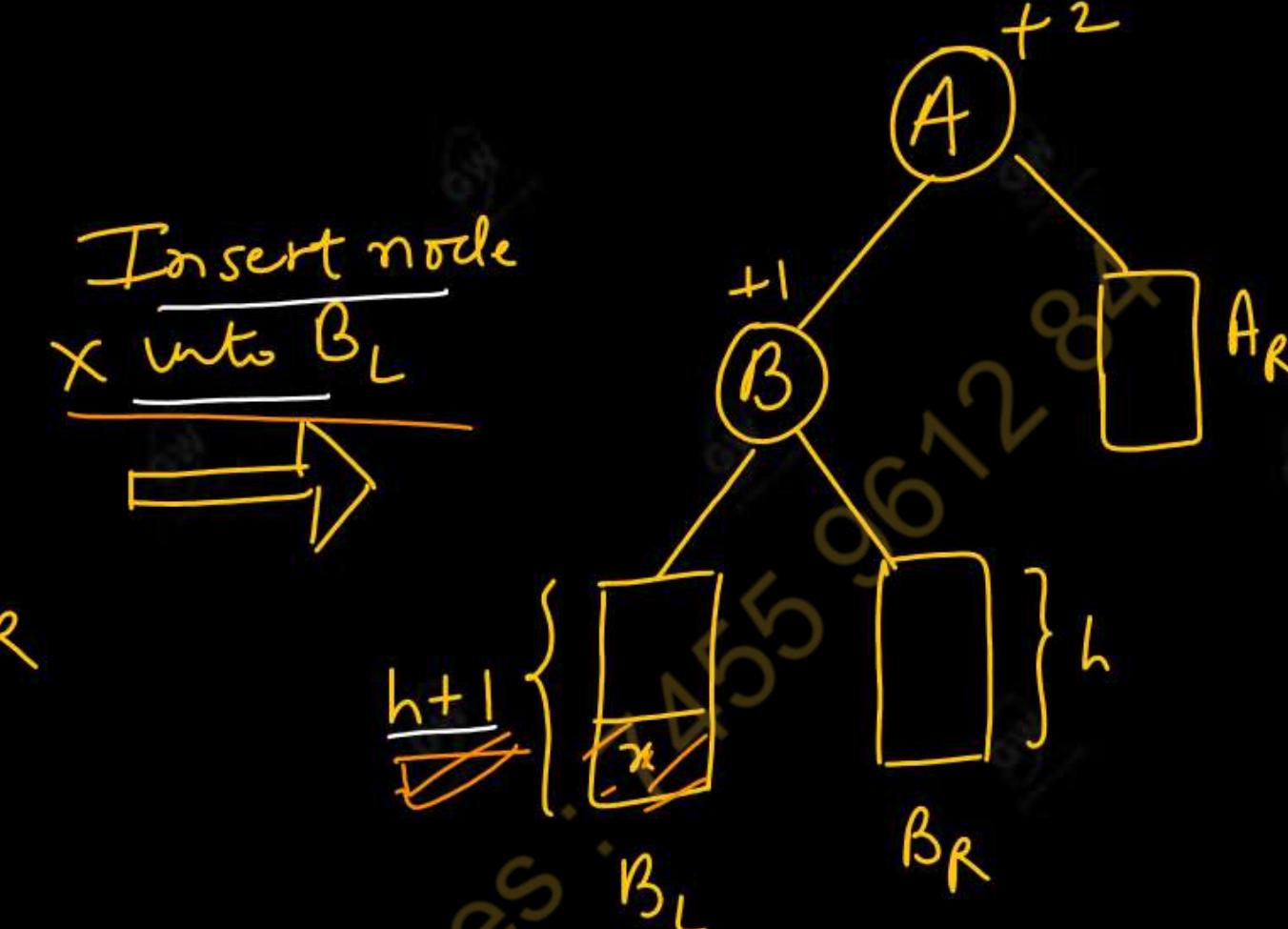
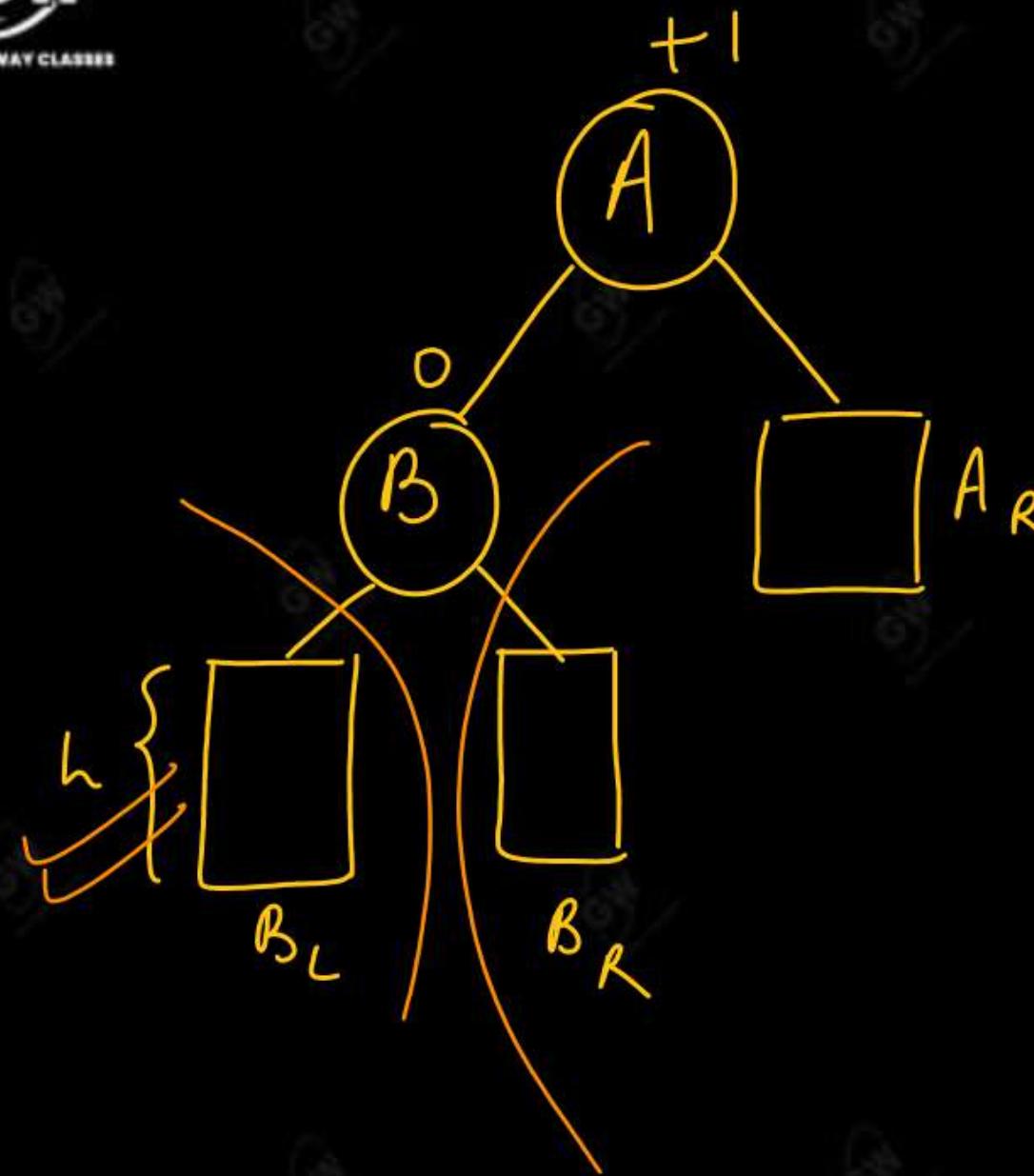




(a) LL Rotation:

- The new node x is inserted in the left sub-tree of left sub-tree of A whose balance factor becomes +2 after insertion.
- To rebalance the search tree, it is rotated so as to allow B to be the root with B_L and A as B_L is the left sub-tree and A is the right sub-tree and B_R and A_R to be the left and right sub-trees of A .





AVL

3, 2, 1

3^o

l - 0 = 1
3
3^o

AVL

+₂
3
2^o

2^o

l - 1 = 0
2
1^o
3^o

Balanced
AVL Tree
BSR

Gateway Classes : 7455987284

i.e., in LL Rotation

Left (A) \leftarrow Right (B)
Right (B) \leftarrow A

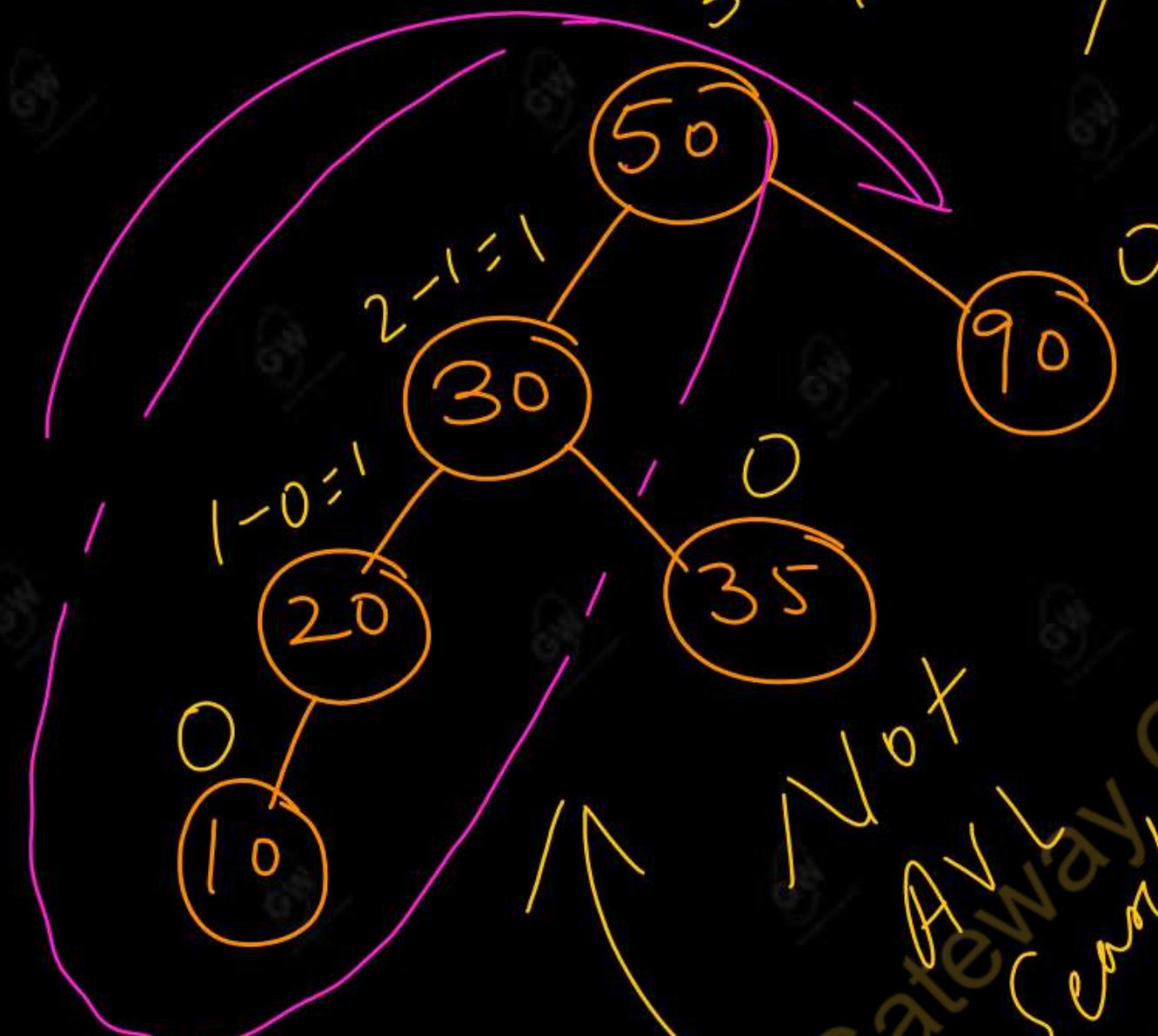
Example: Insert node 10, in



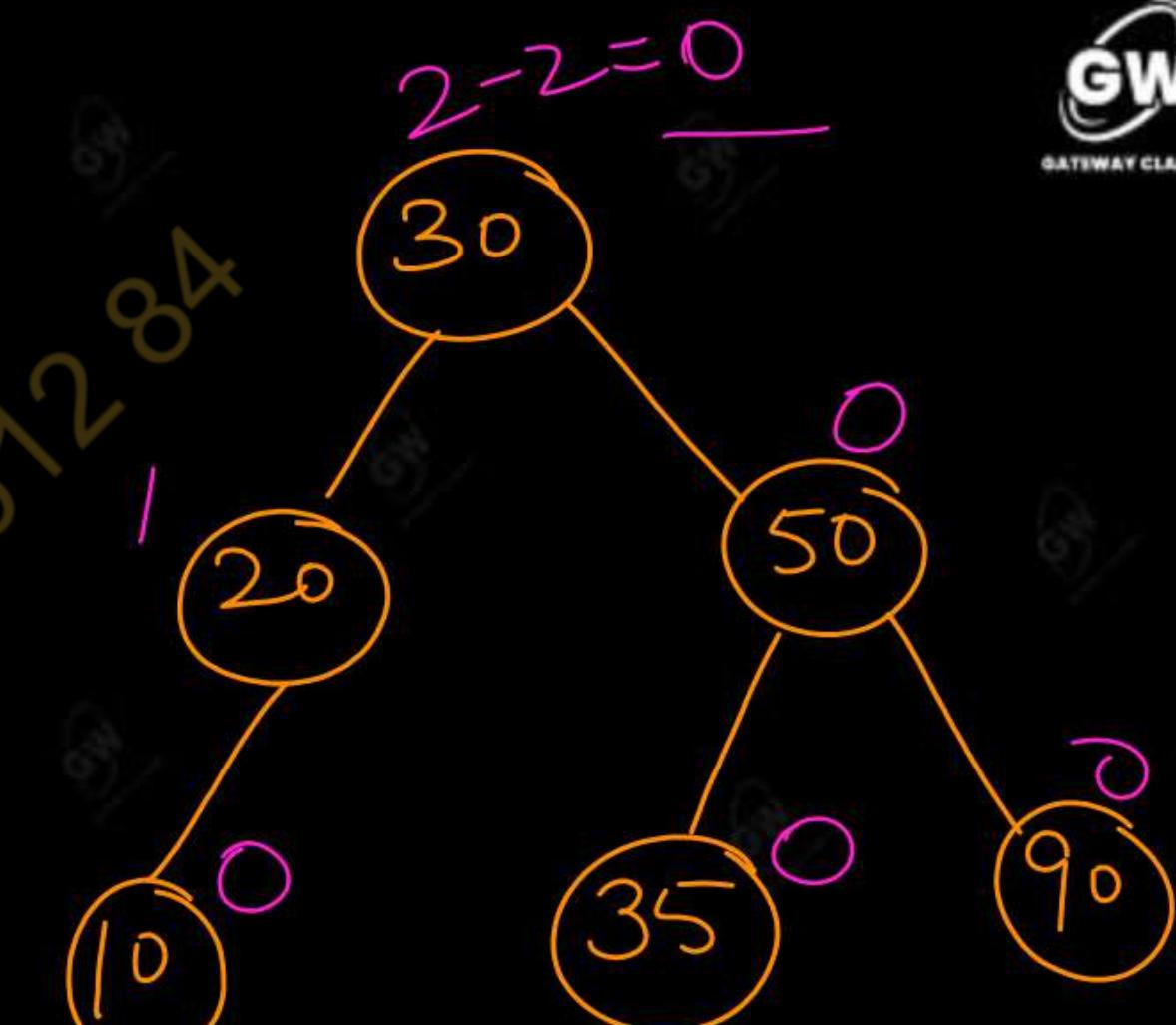
We get

$$3 - 1 = 2$$

X



→ LL Rotation

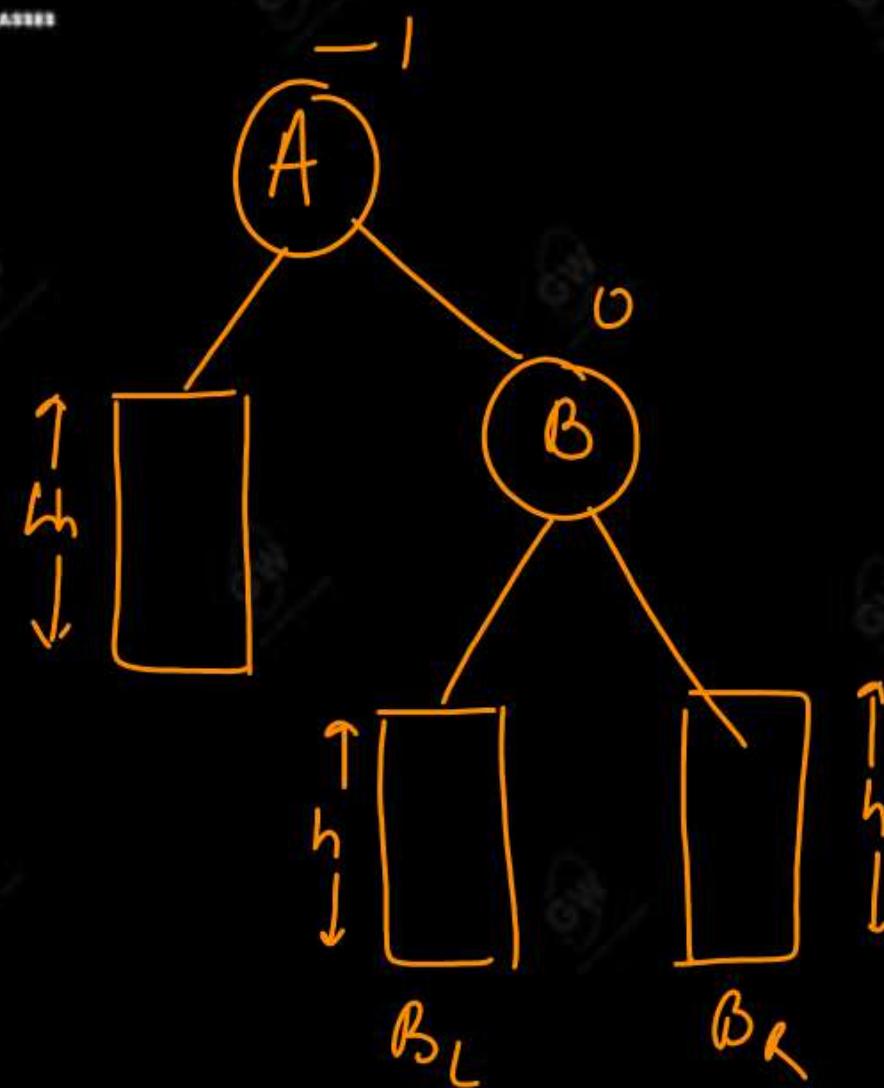


Balanced AVL
Search Tree

- Unbalance occurred due to the insertion in the right sub-tree of the right child of the pivot node.
- So, in this rotation new node is inserted at the right sub-tree of the right sub-tree of root node (pivot node).
- The next figure illustrates the balancing of an AVL search tree using RR rotation.
- Here the new node x is in the right sub-tree of A.
- The re-balancing rotation pushes B upto the root with A as its left child and B_R as its right sub-tree and A_L and B_L as the left and right sub-trees of A.
- That is, in RR rotation-

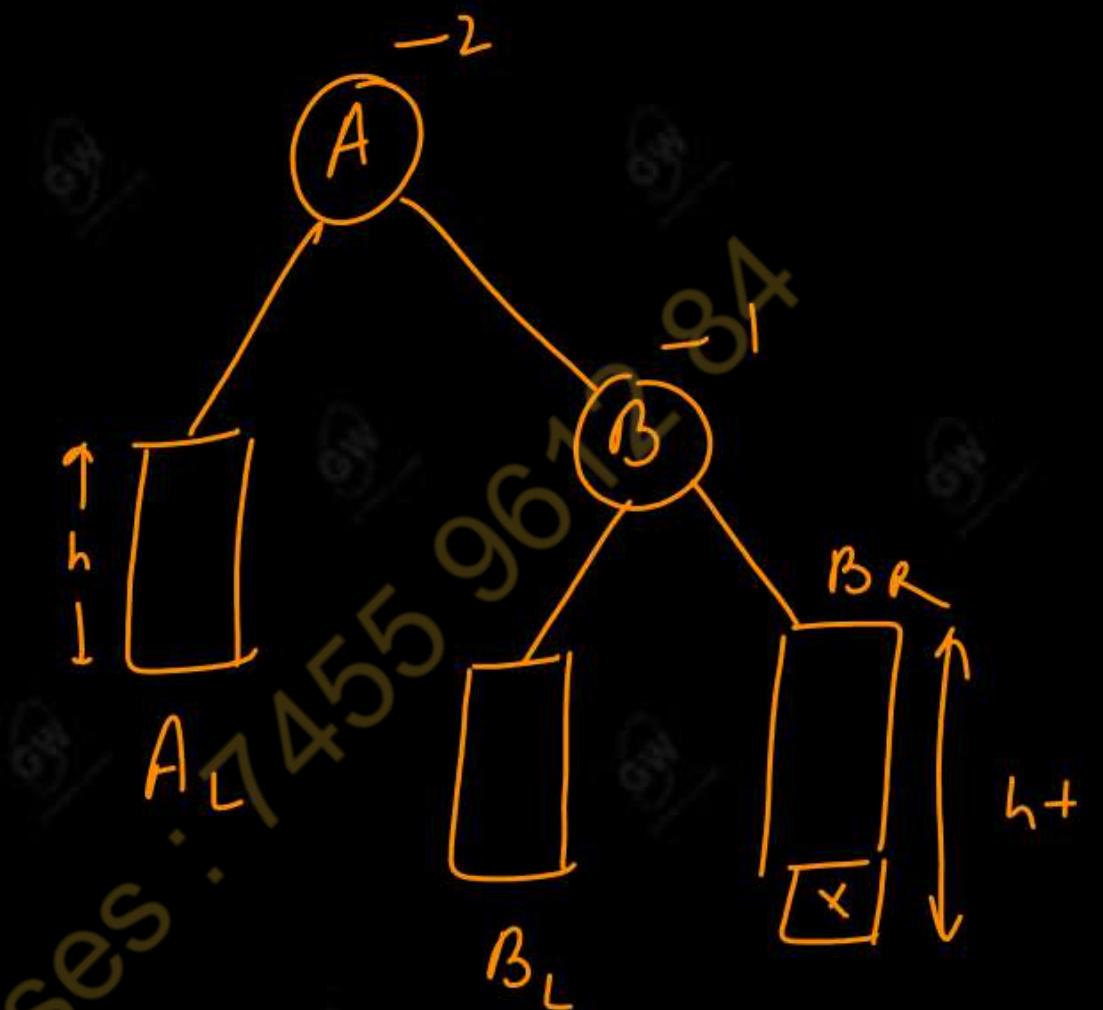
Right (A) \leftarrow left (B)

Left (B) \leftarrow A



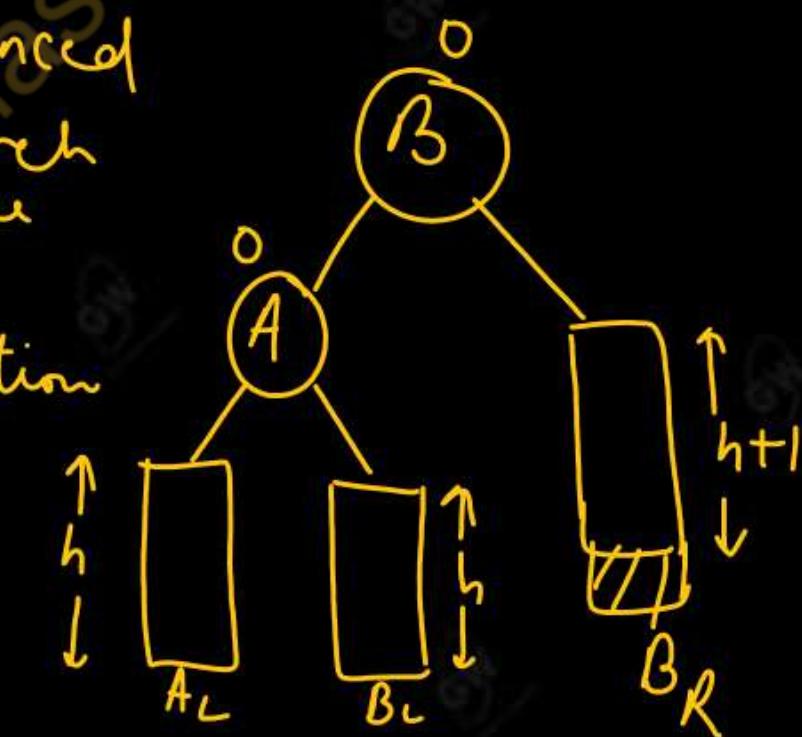
Balanced AVL
Tree

Insert X
into B_R



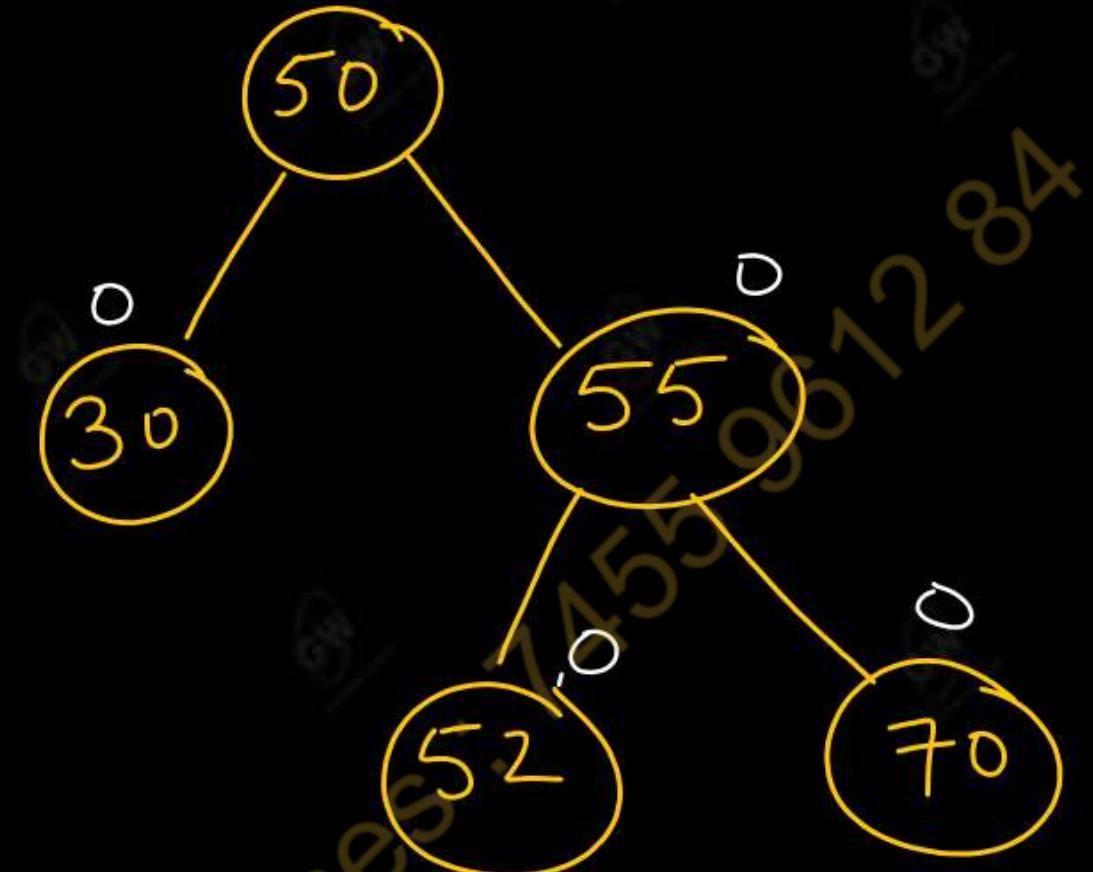
Unbalanced
AVL
Tree

Balanced
Search
Tree
af-
ter
rotation

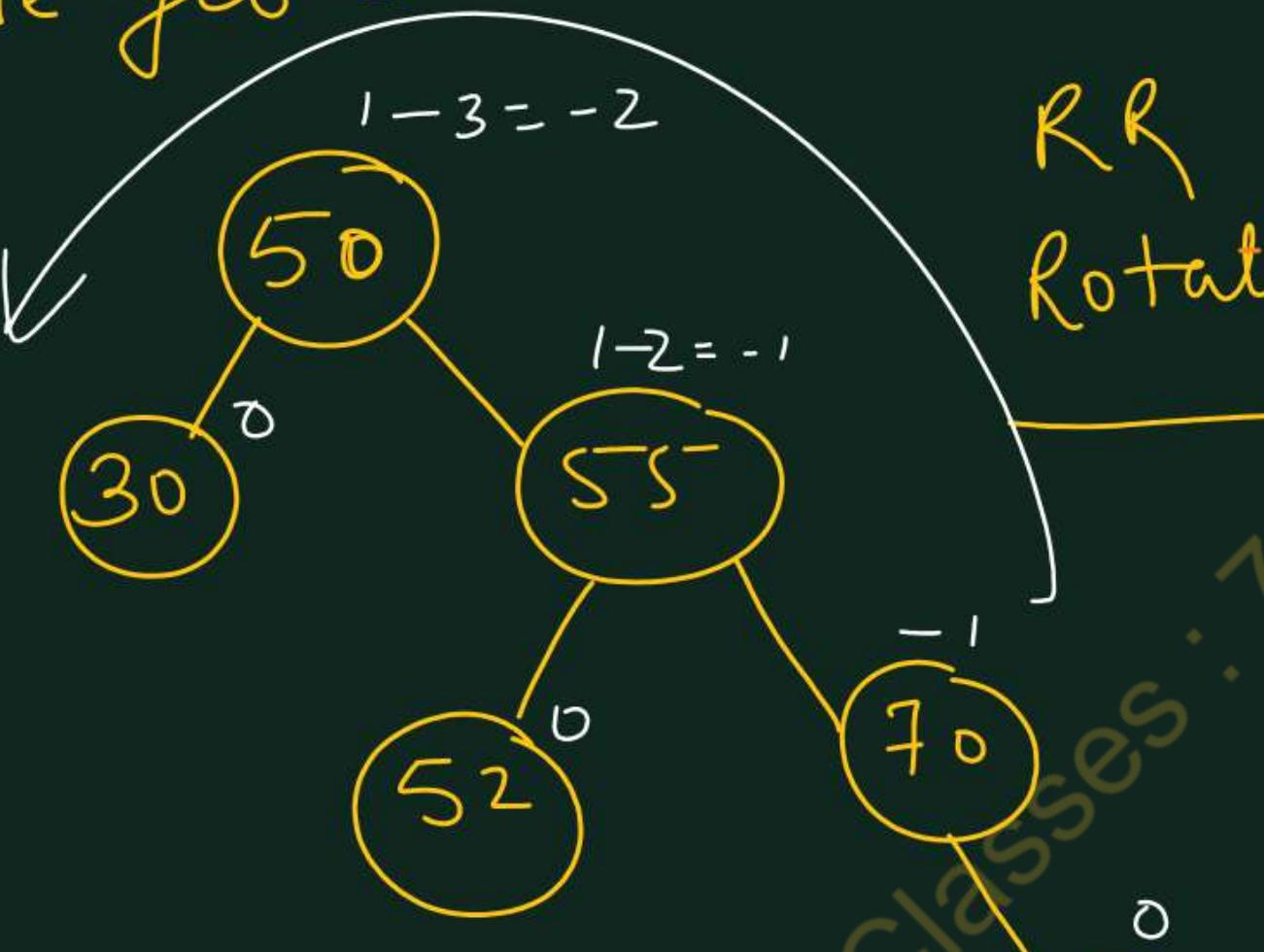


Example: Insert node 85 in.

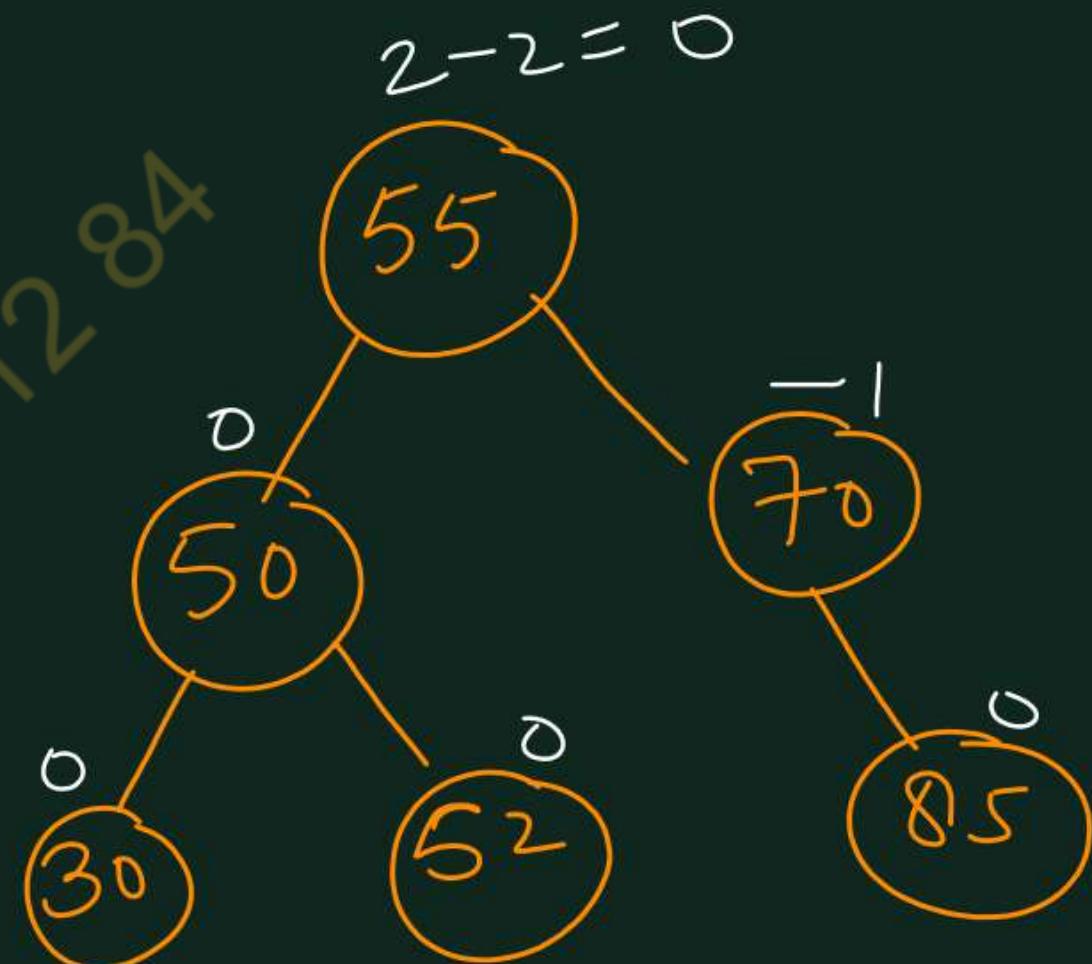
$$1 - 2 = -1$$



We get -

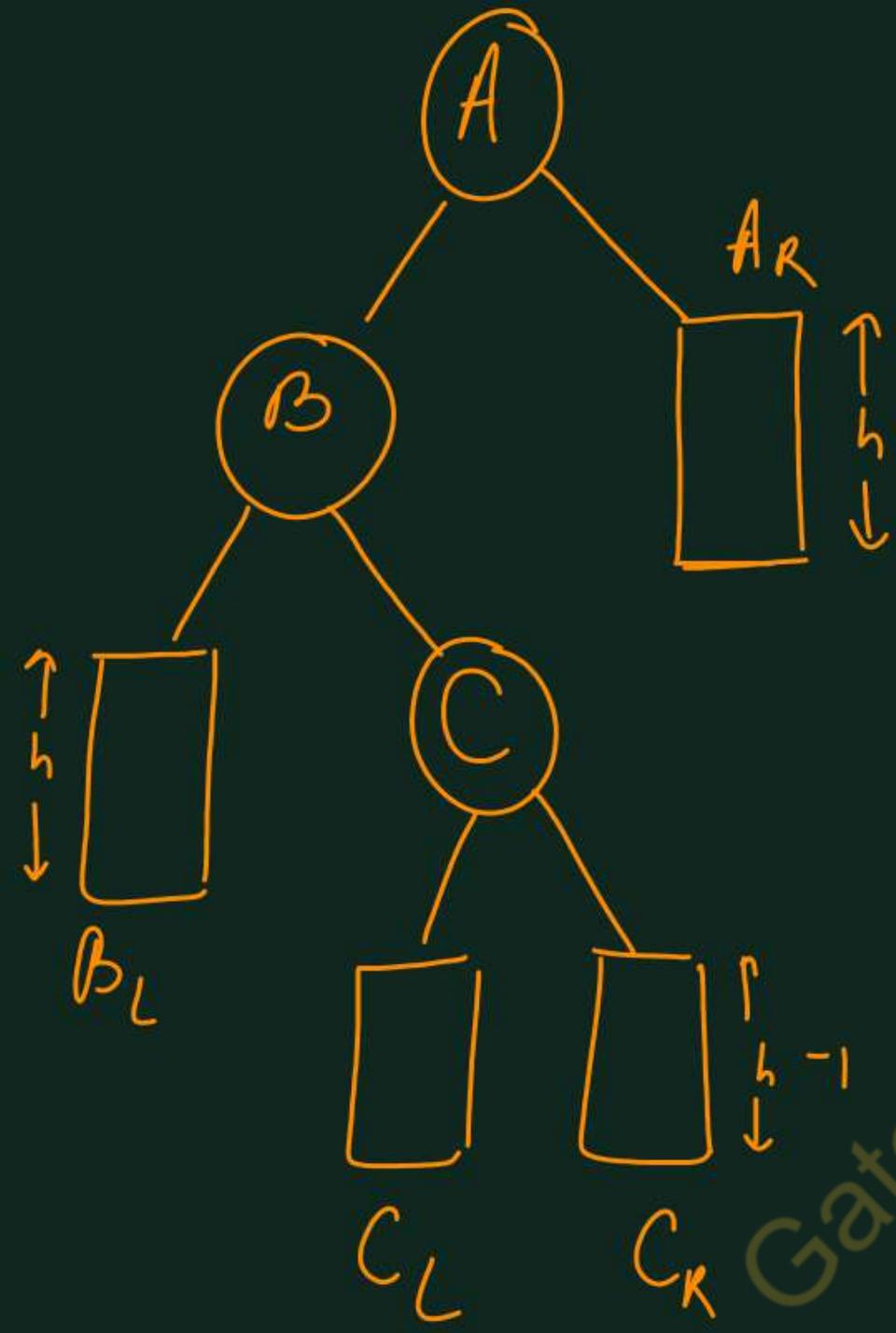


RR
Rotation

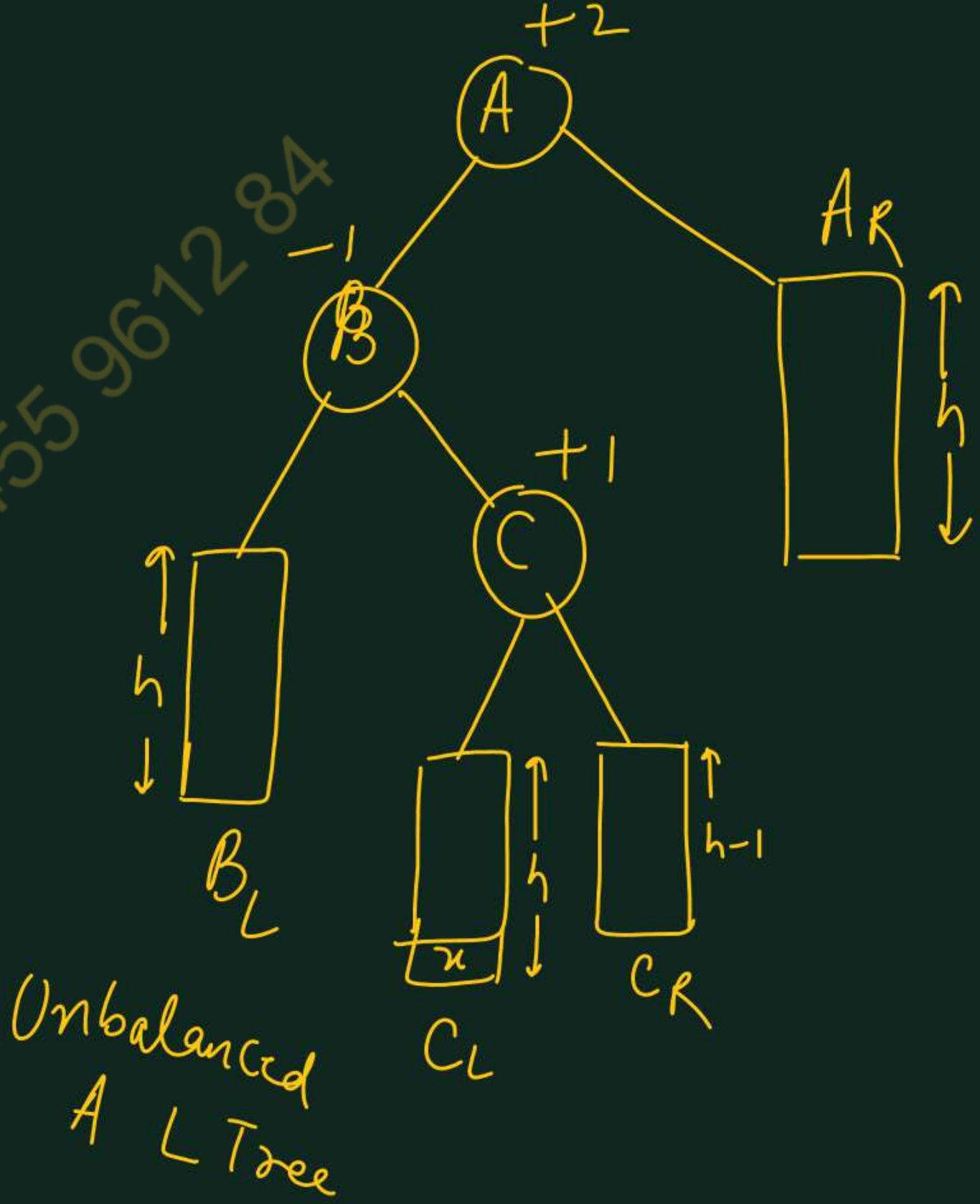


Balanced AVL
search Tree

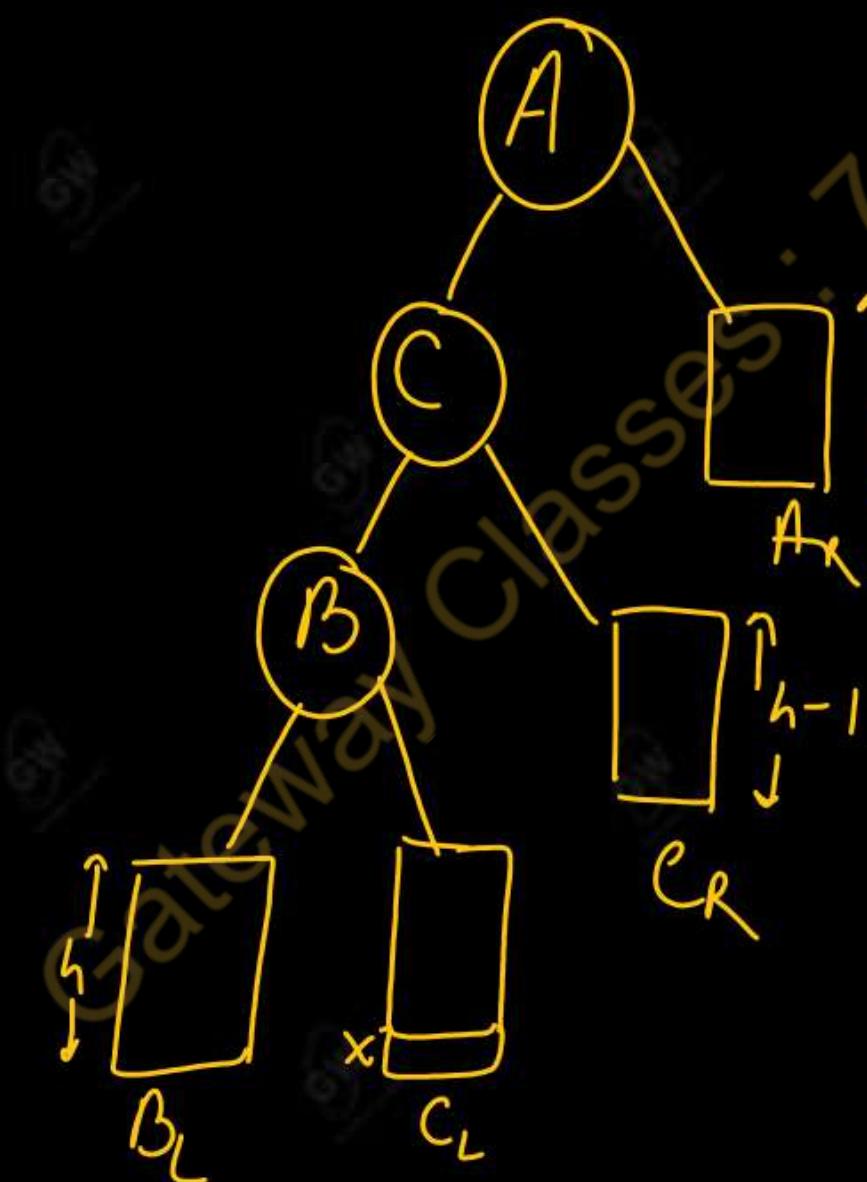
- In this, unbalance occurred due to the insertion in the right sub-tree of the left child of the root (pivot) node.
- So, this is known as left to right insertion.
- LR rotation involves two rotations for the manipulation in pointers.



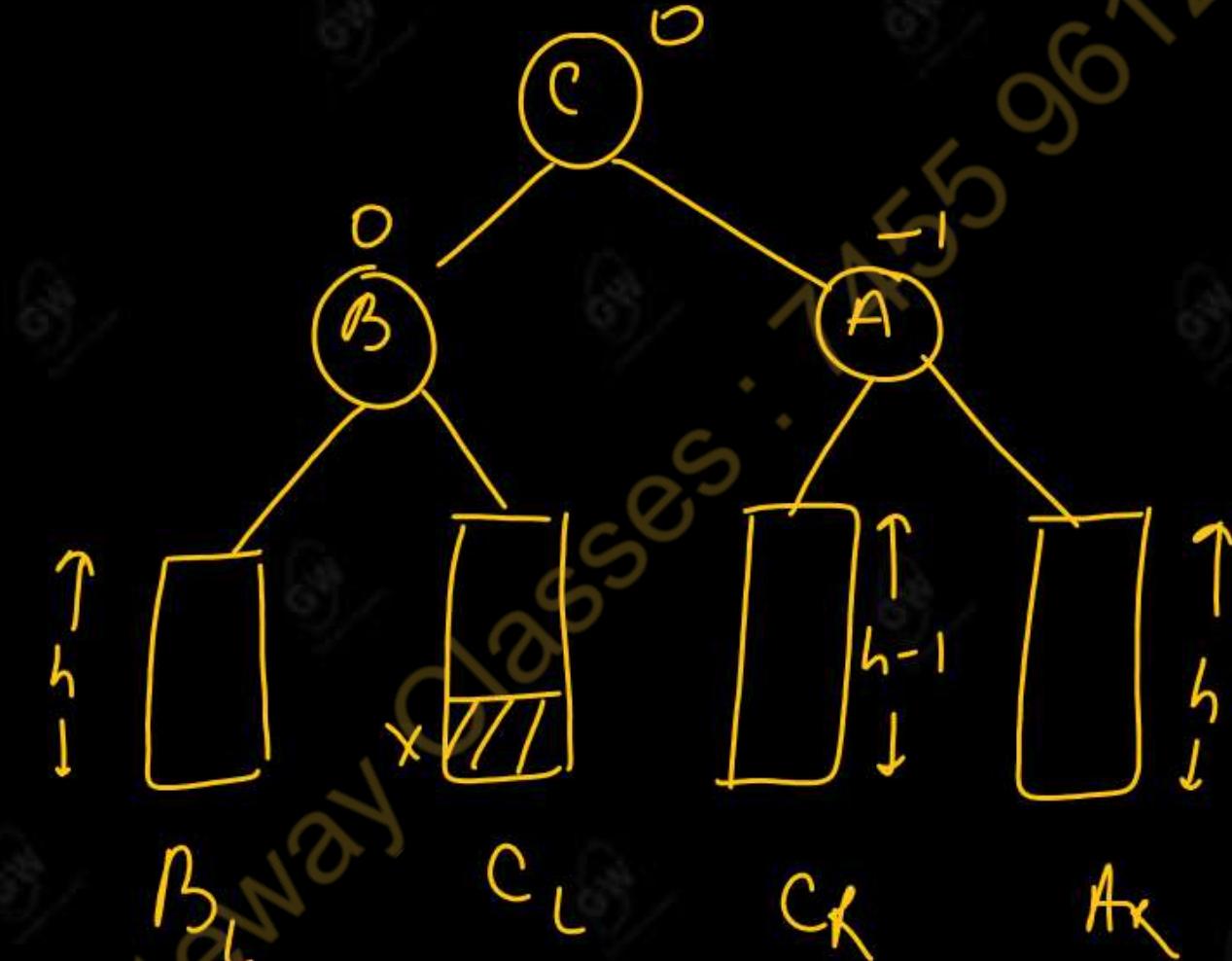
Insert
into C_L



- The left sub-tree (C_L) of the right child (C) of the left child (B) of pivot node (A) becomes the right sub-tree of the left child (B).
- The left child (B) of the pivot node (A) becomes the left child of C (i.e., RR-Rotation).



- The right sub-tree (C_R) of the left child (C) of the left child (B) of the pivot node (A) becomes the left sub-tree of A and A becomes the right child of C.



Balanced
AVL
Search
Tree

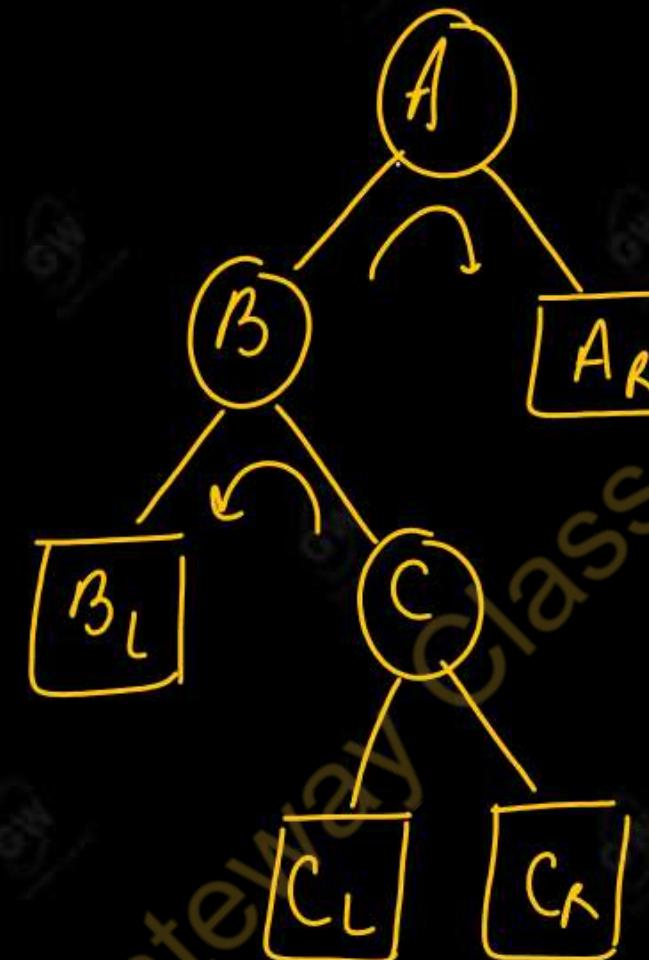
The pointer movement is:

Right (B) \leftarrow left (C)

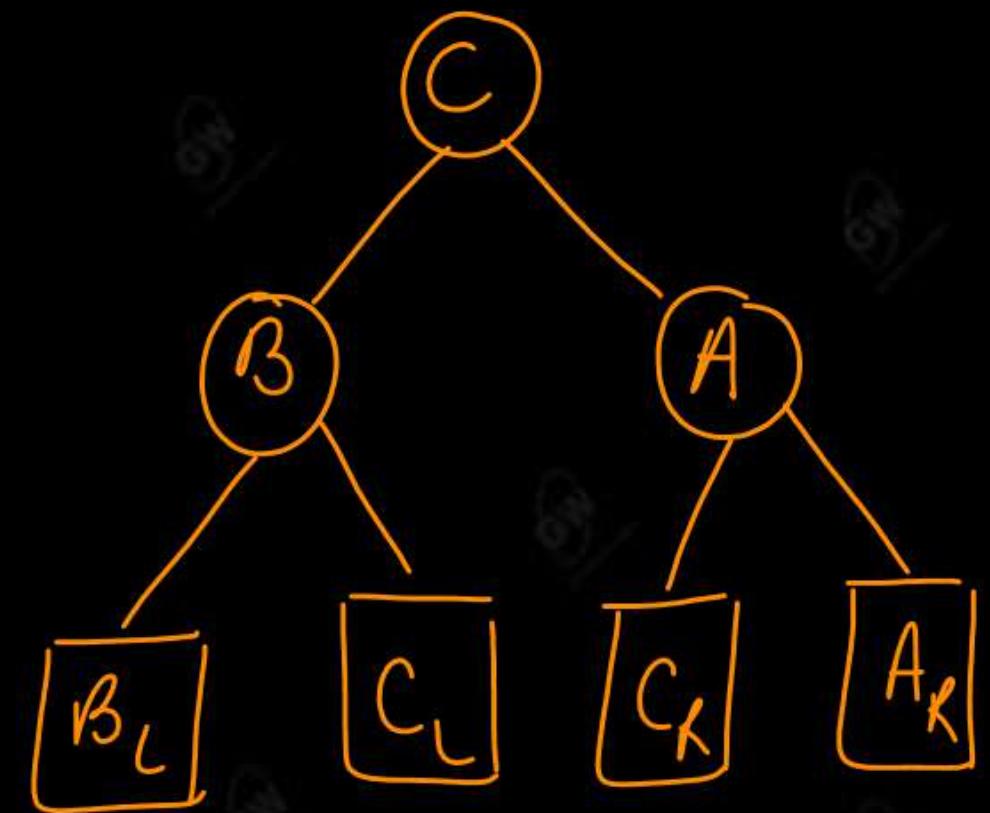
left (A) \leftarrow right (C)

left (C) \leftarrow B

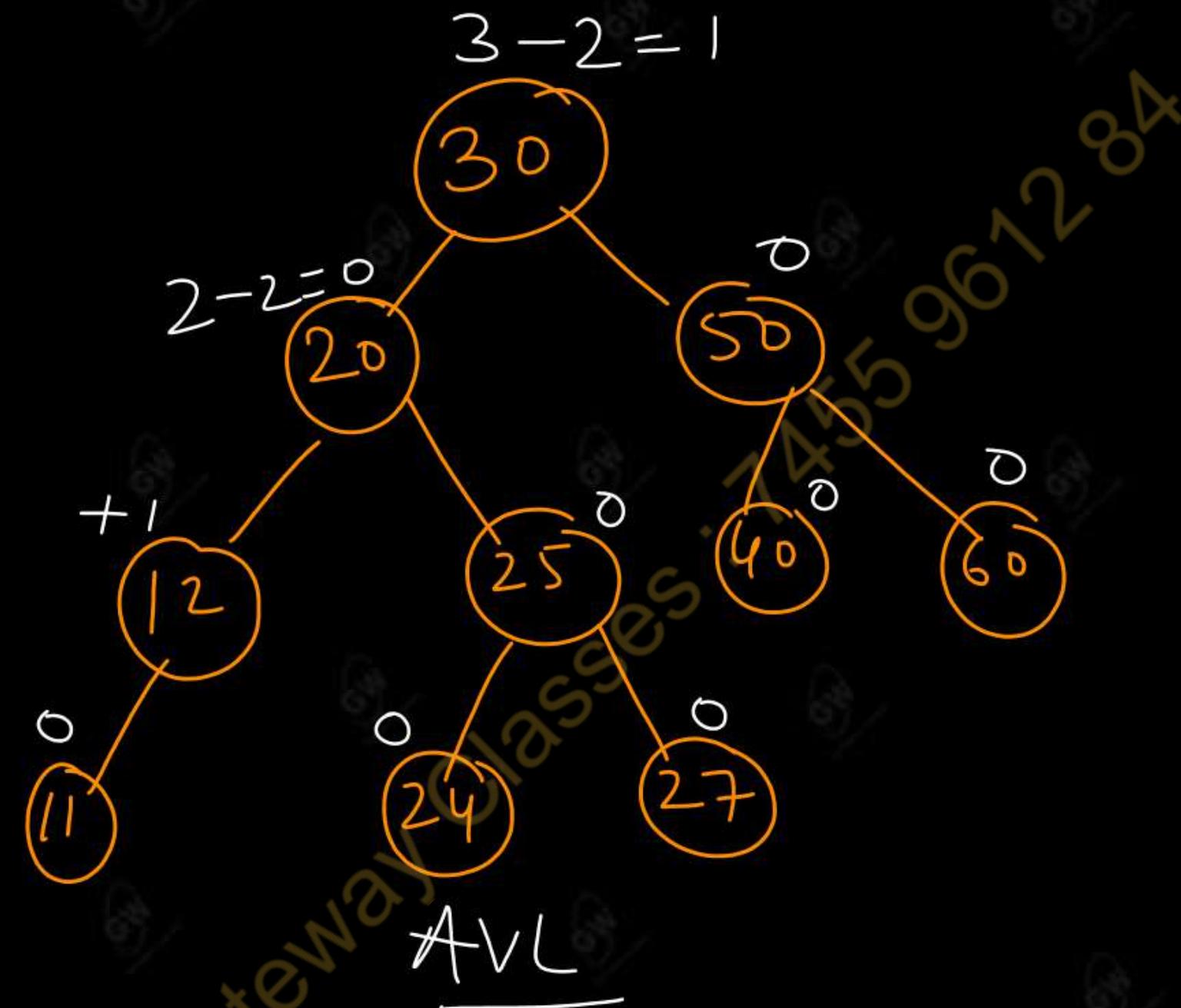
right (C) \leftarrow A



7455961284
Rotation LK

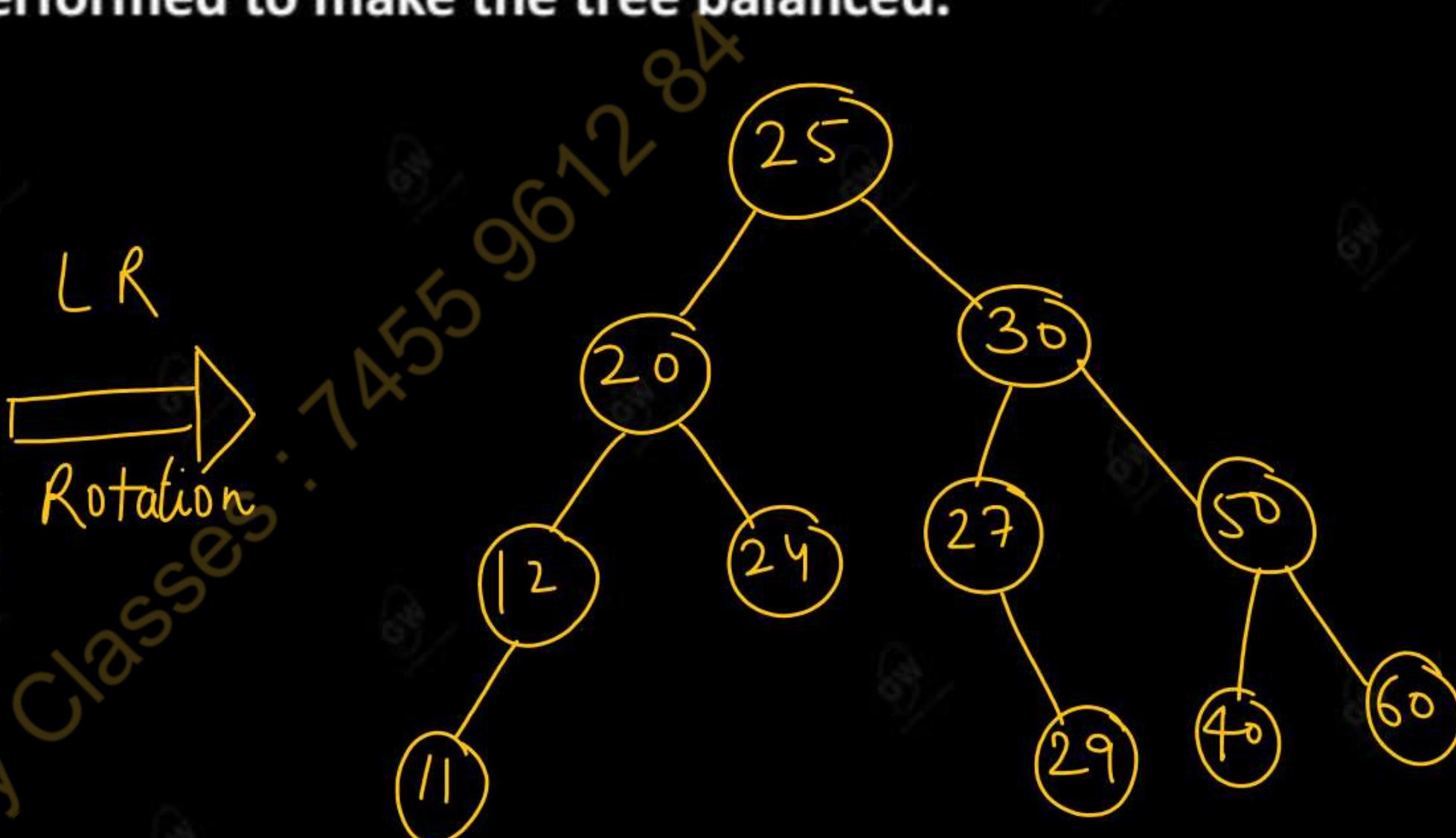
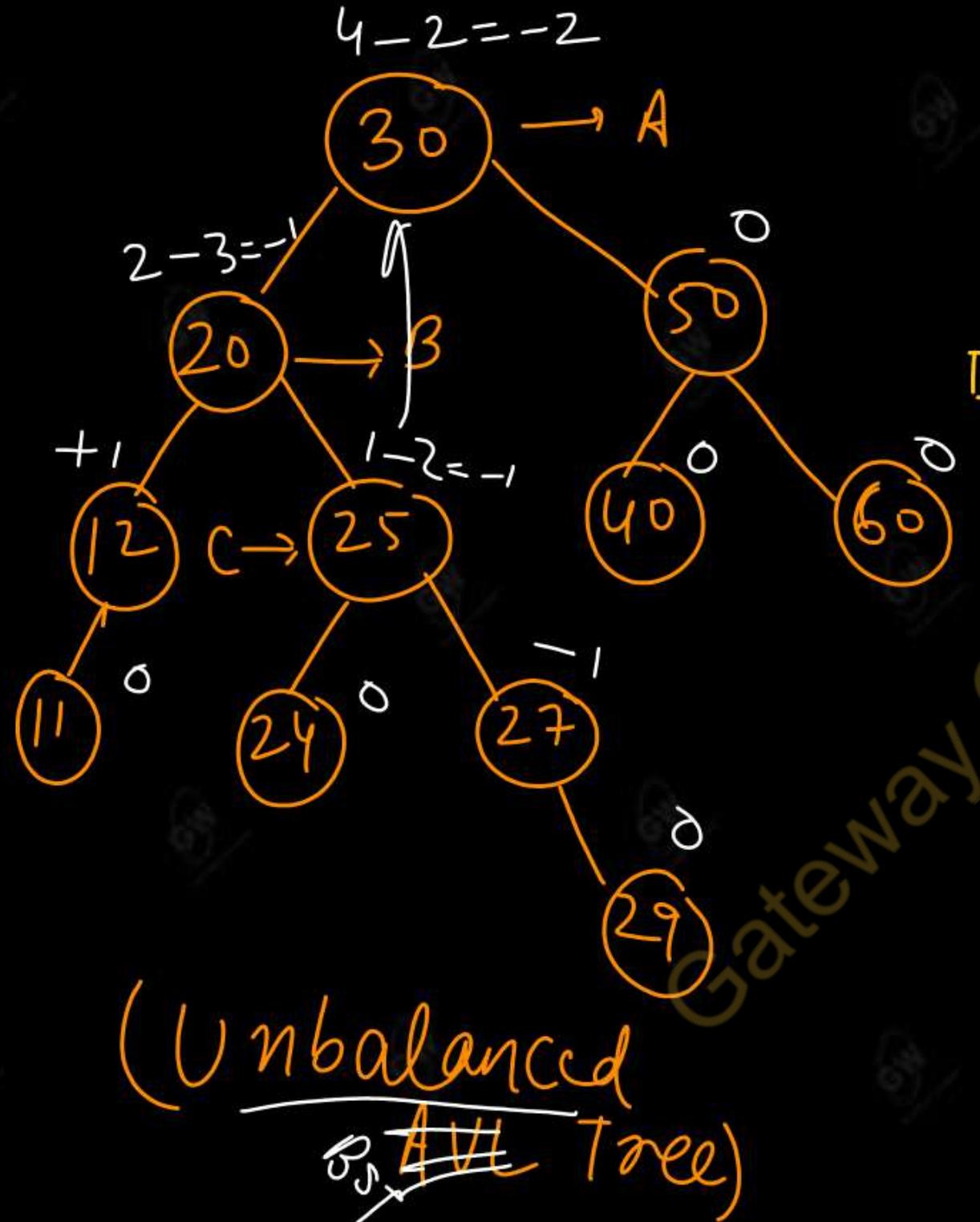


Example: Consider the following AVL tree which is height balanced:



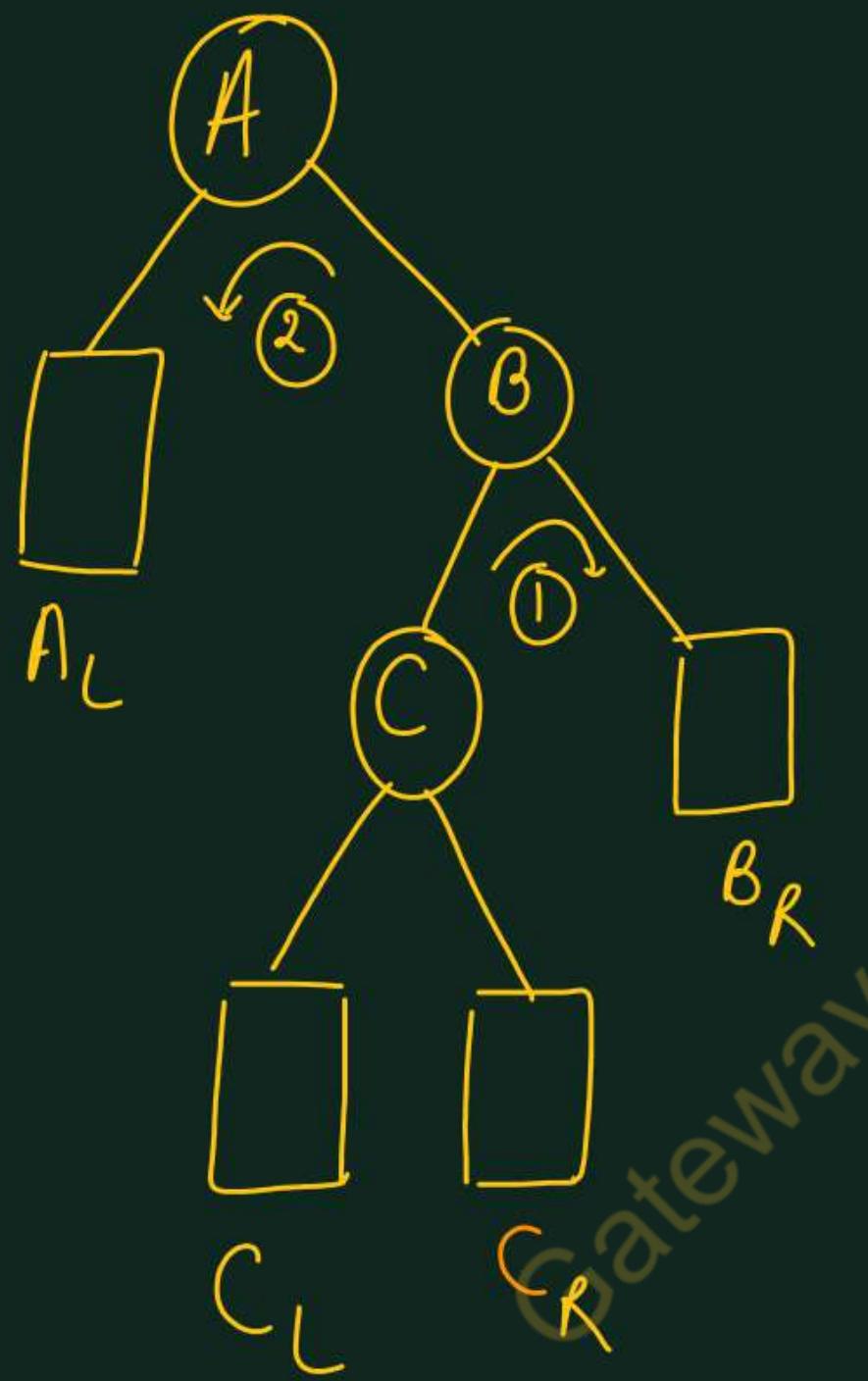
□ Insertion of the value 29 makes the tree unbalanced as shown in Fig (a).

□ Now AVL rotation has to be performed to make the tree balanced.

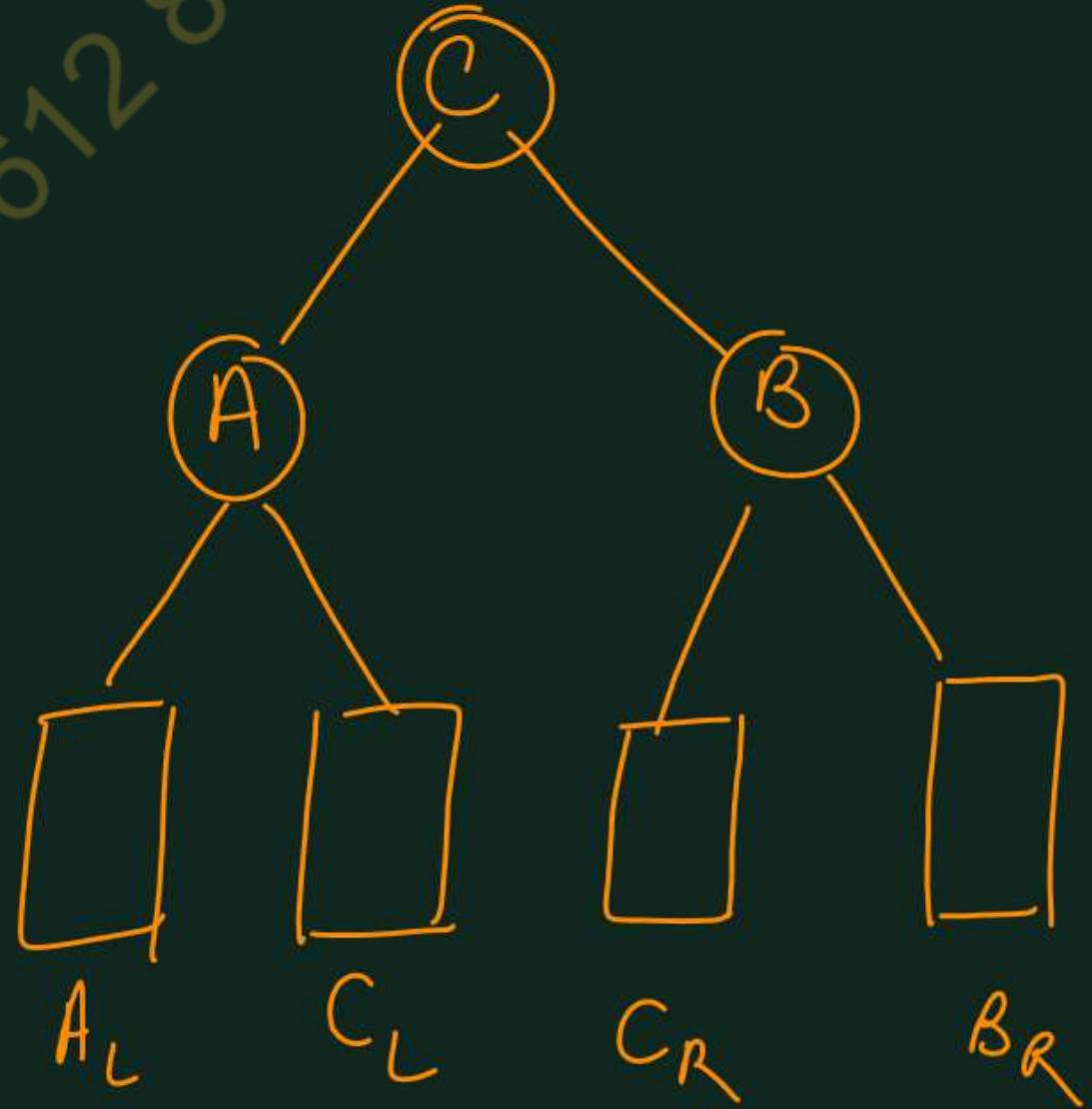


- In this, unbalance occurred due to the insertion in the left sub-tree of the right child of the r (pivot) node. This is known as **right-to-left insertion**.
- RL-Rotation is the mirror image of LR-Rotation. Next figures, illustrates the unbalance appearance and its removal due to the insertion.

Gateway Classes : 7455967284

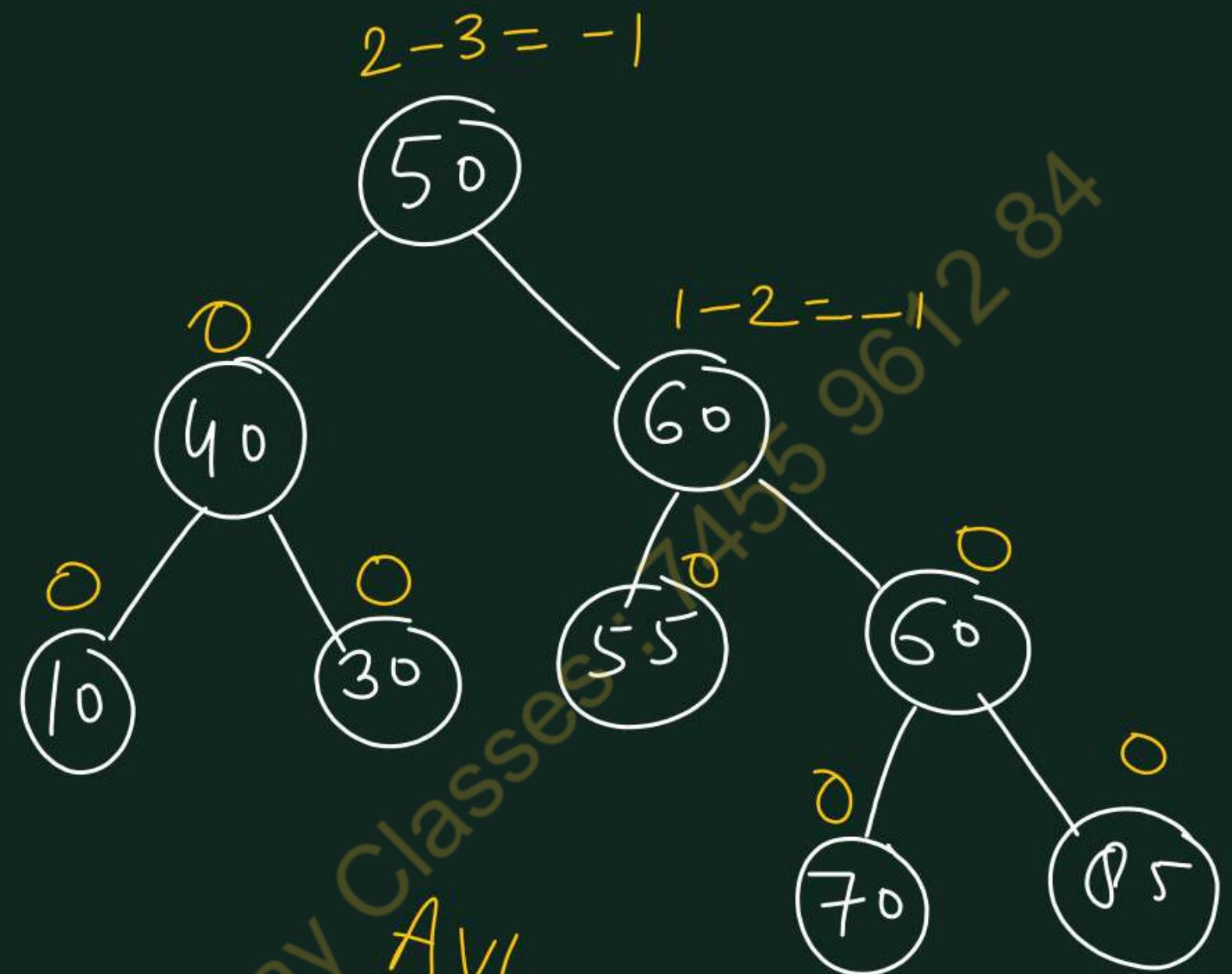


RL
Rotation



□ Here, there are two rotations for the manipulations of pointers. They are the following:

- **Rotation 1:** The right sub-tree (C_R) of the left child (C) of the right child (B) of root node (A) becomes the left sub-tree of B and the right child (B) of the root node becomes the right child of C.
- **Rotation 2:** The left sub-tree (C_L) of the left child (C) of the right child (B) of the root node becomes the right sub-tree of A.
- **Example:** Consider the AVL tree as in next figure -



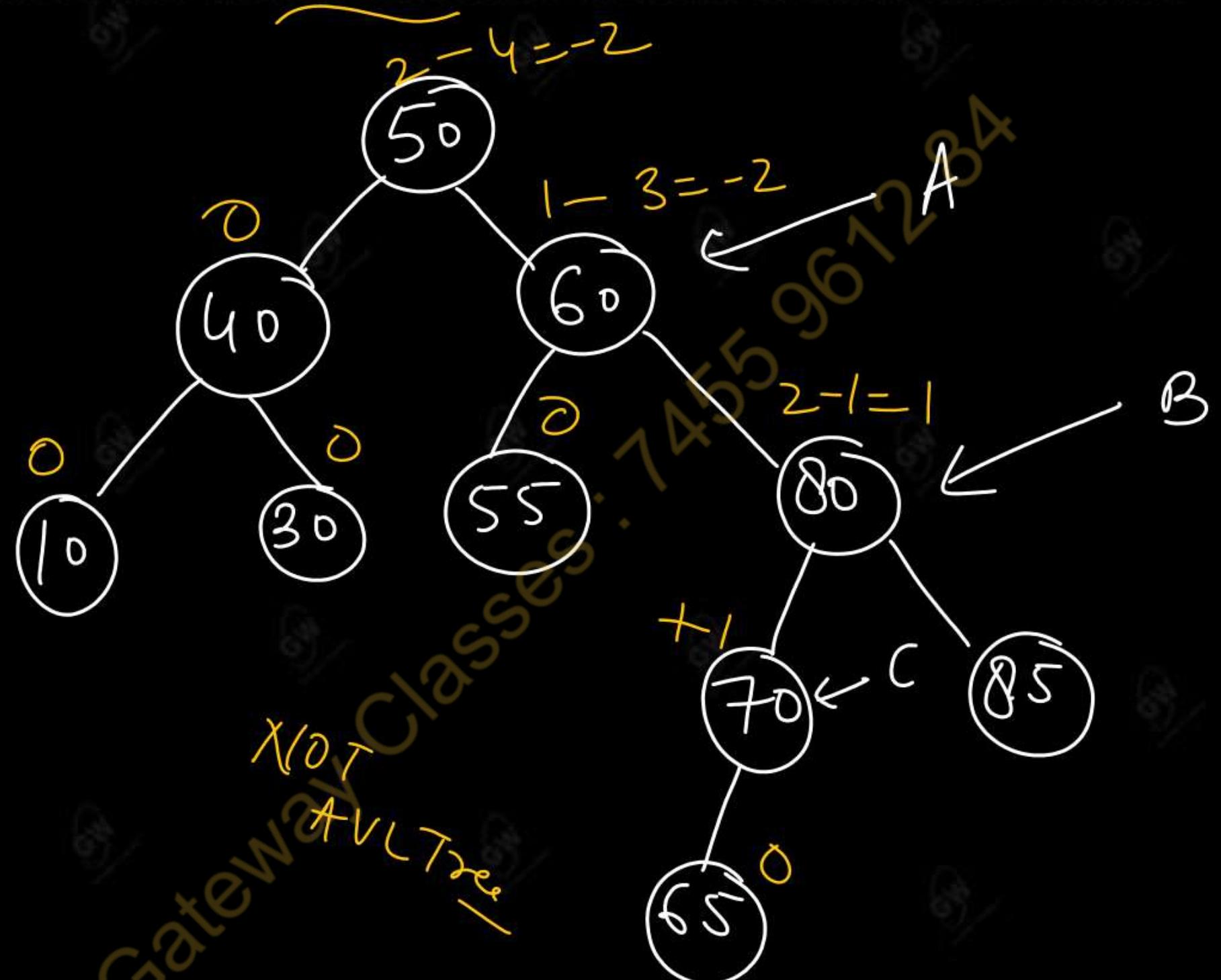
Gateway Classes AVL

$$2 - 3 = -1$$

$$1 - 2 = -1$$

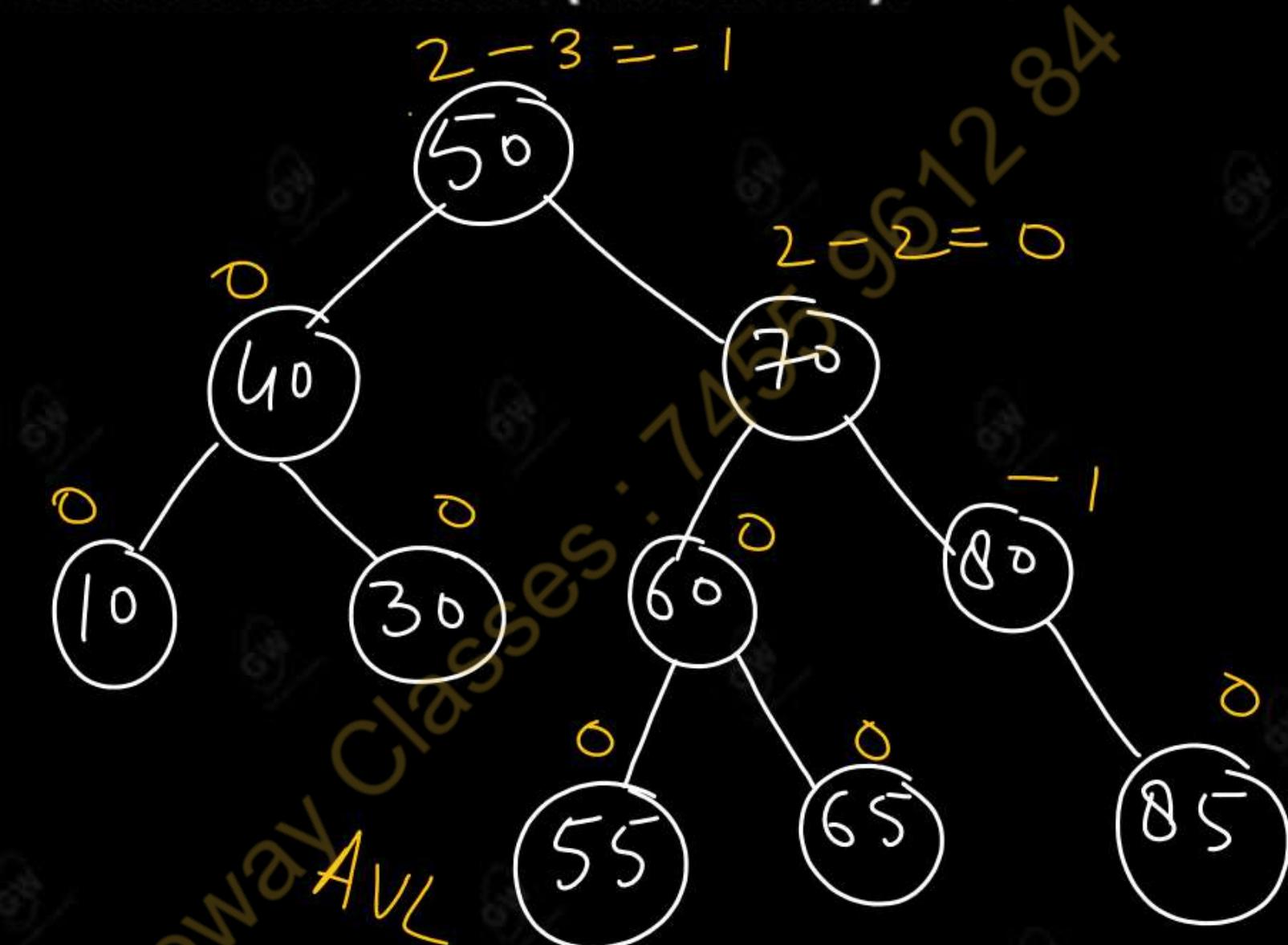
284
9672

□ This is height unbalanced. When 65 is inserted into this tree it becomes



□ Now the AVL tree is unbalanced.

□ To make it balanced, do the AVL rotations (RL rotation).



□ Now the tree has become a height balance tree.

Example: - Create an AVL tree from the given set of values:

H, I, J, B, A, E, C, F, D, G, K, L

Solution:

AVL

$$D - I = -1$$



AVL

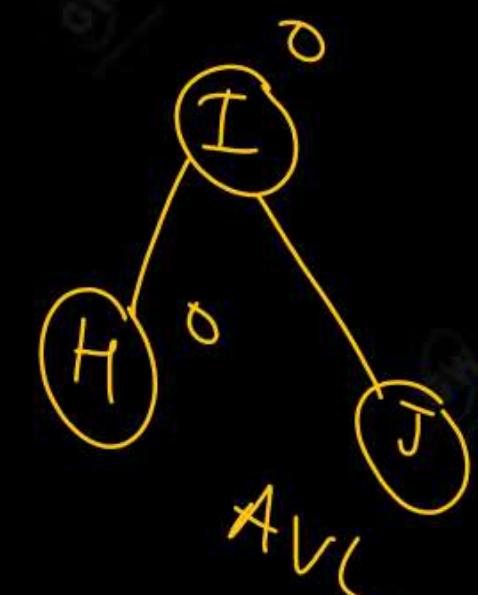


AVL

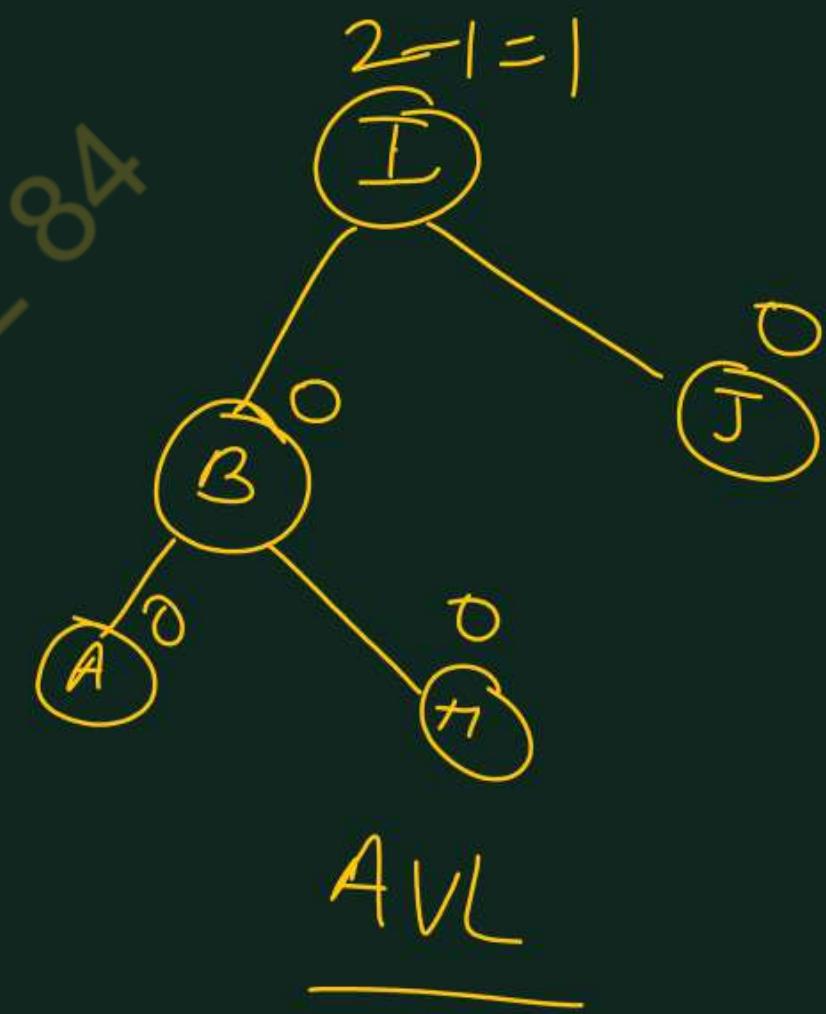
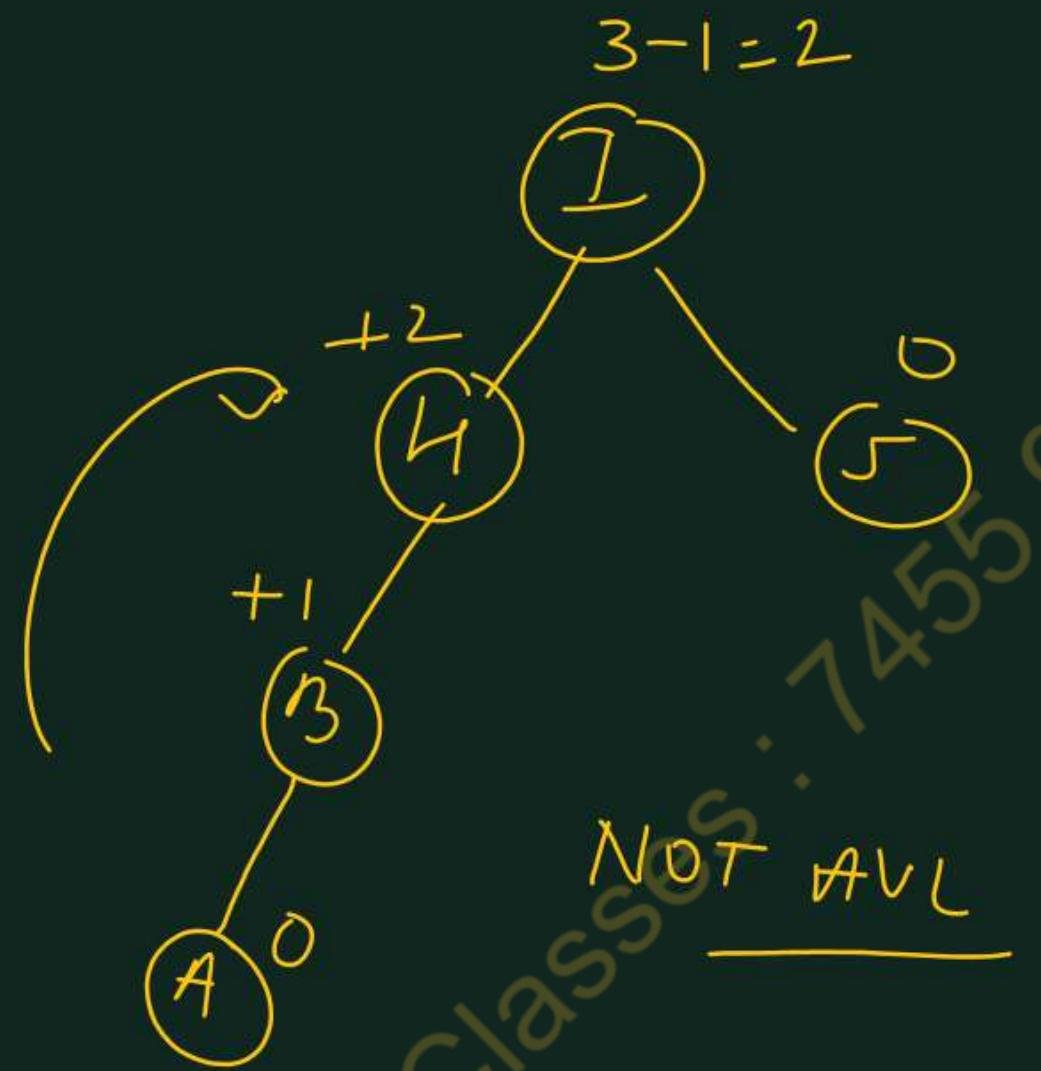
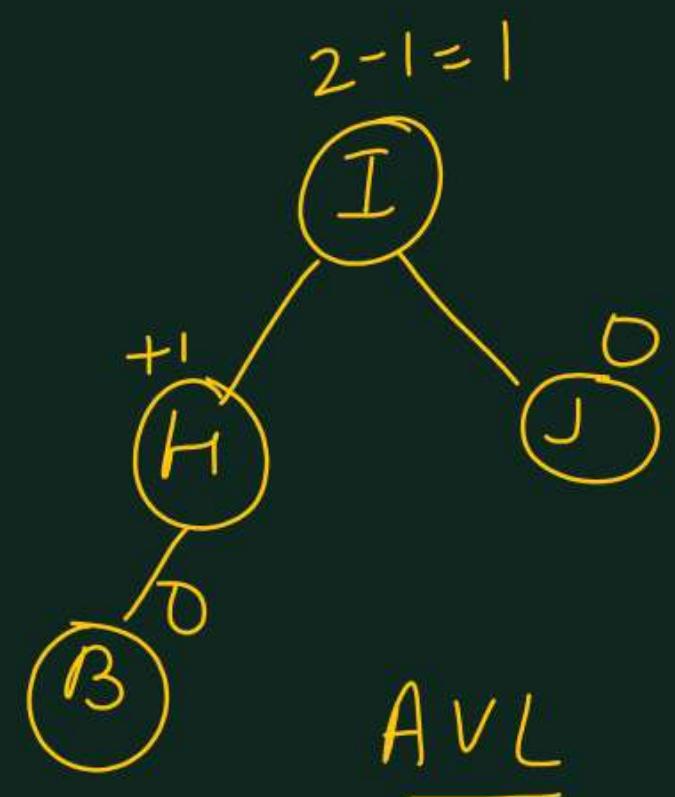
$$D - 2 = -2$$



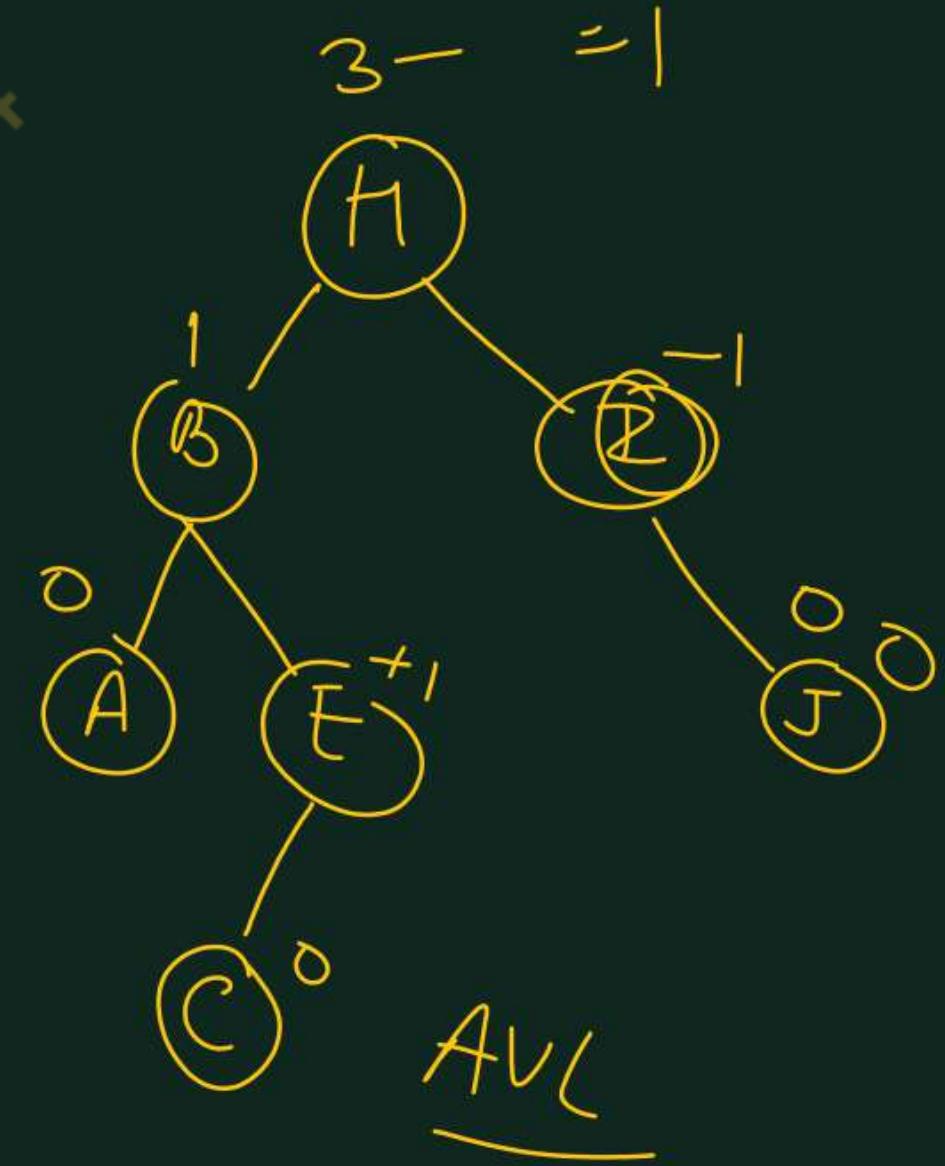
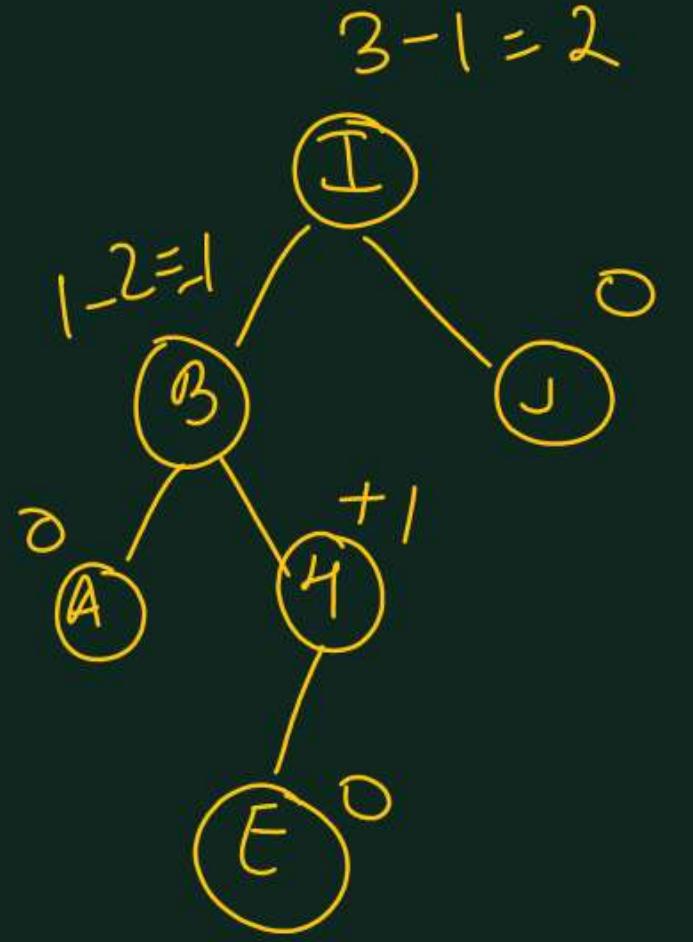
NOT AVL Tree

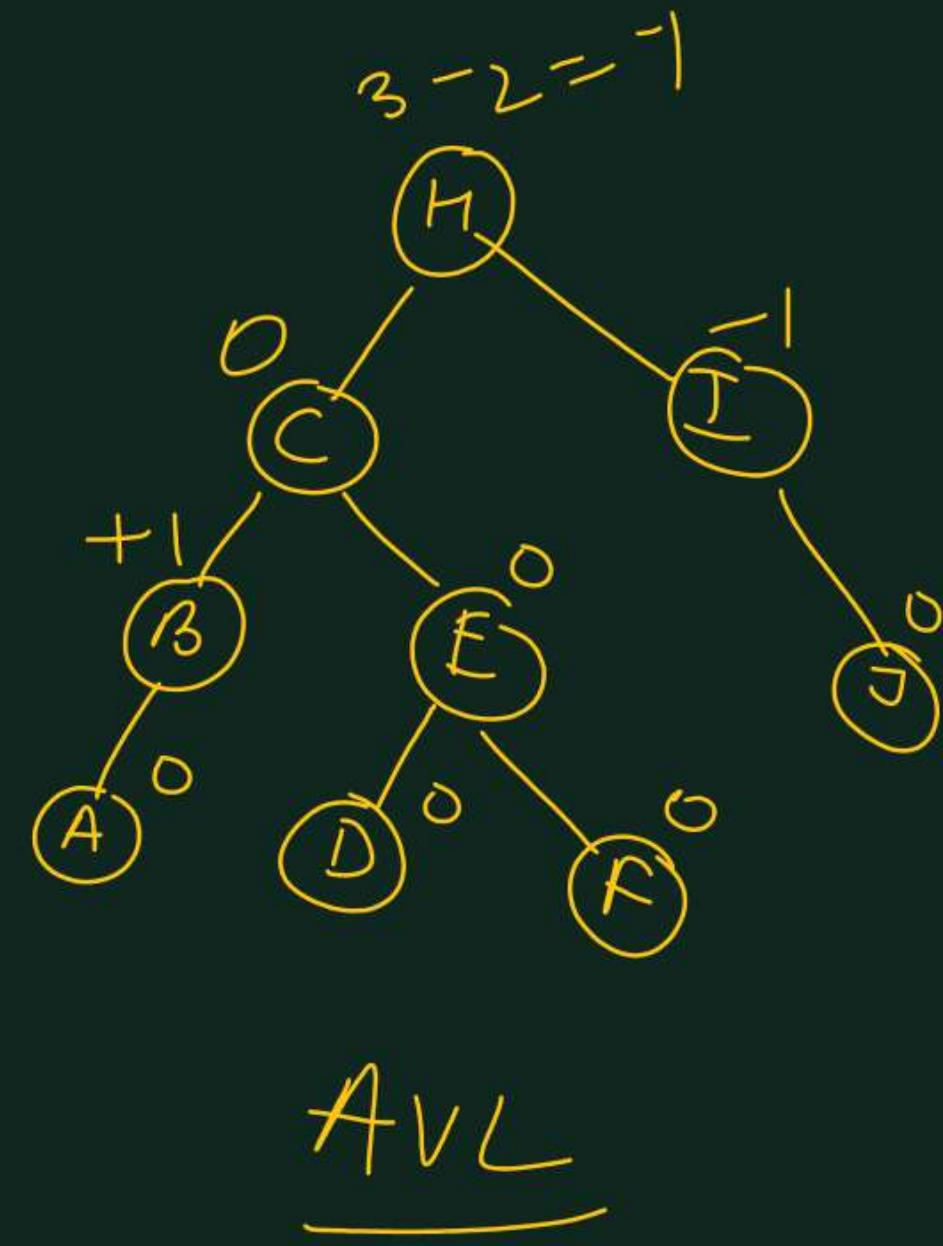
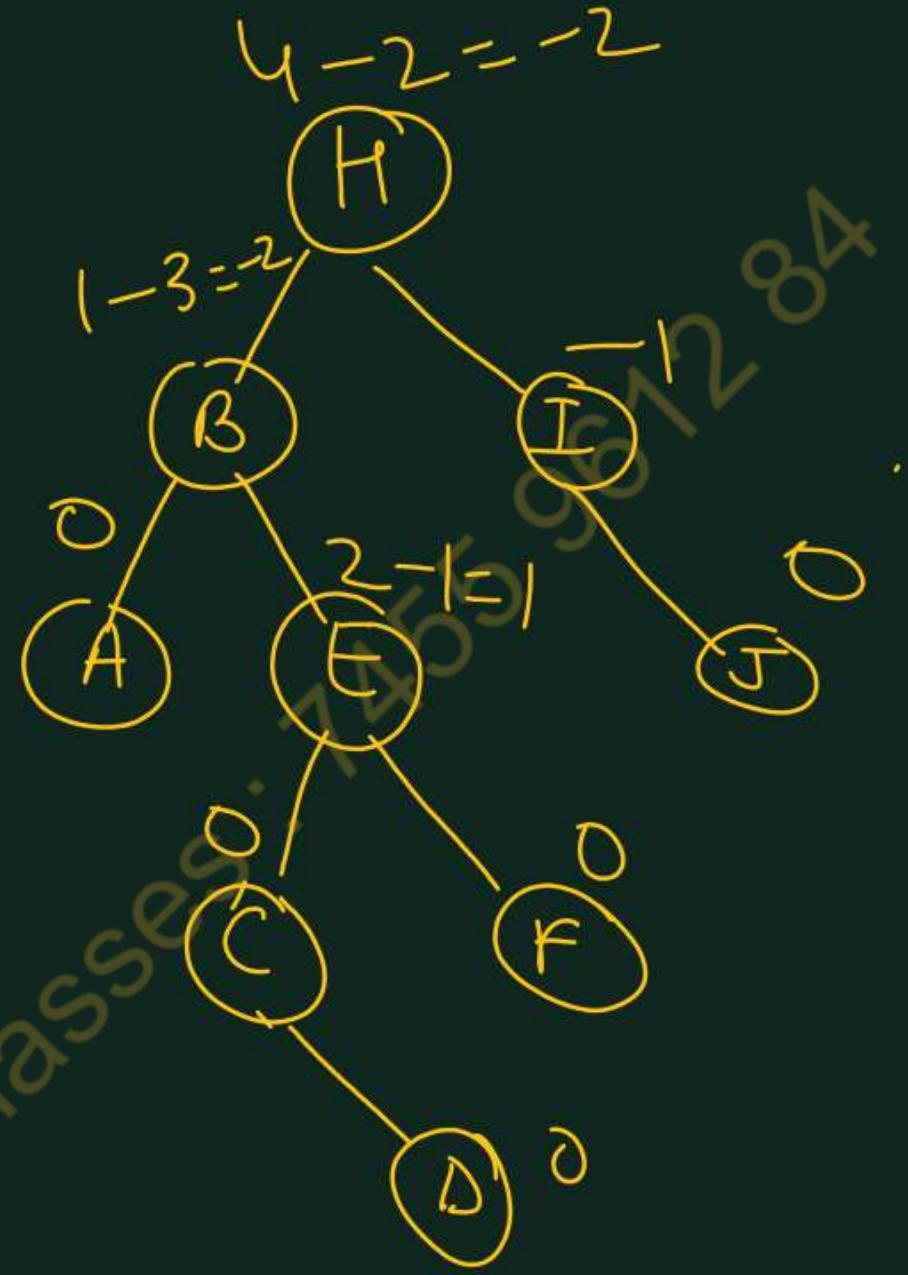
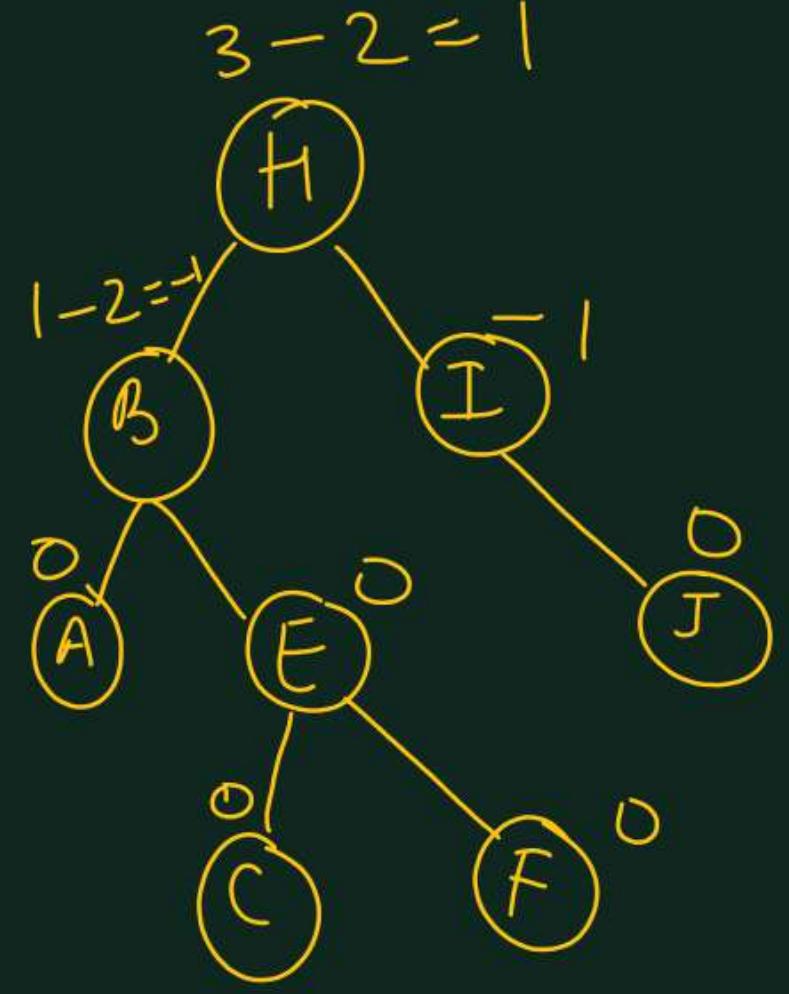


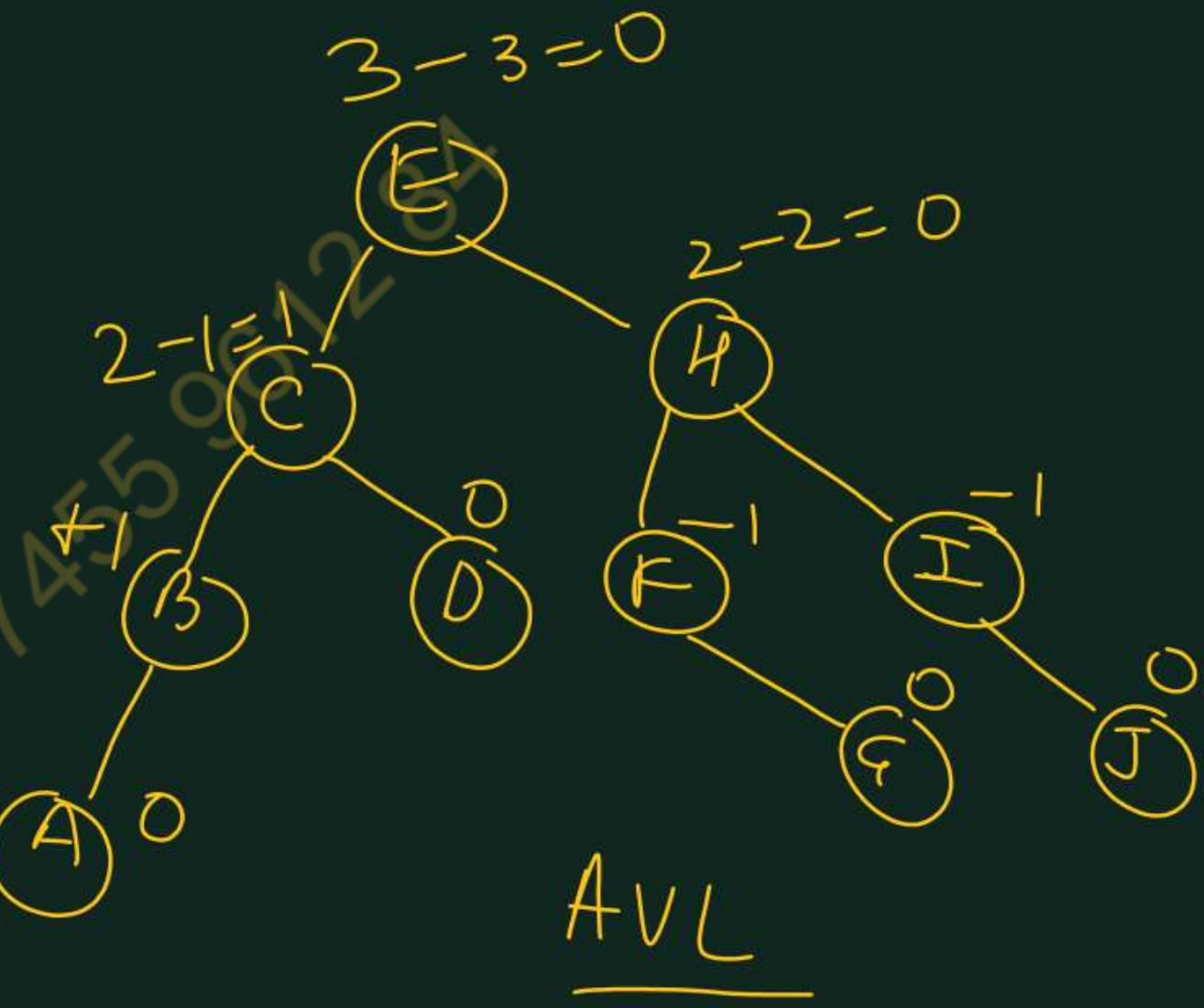
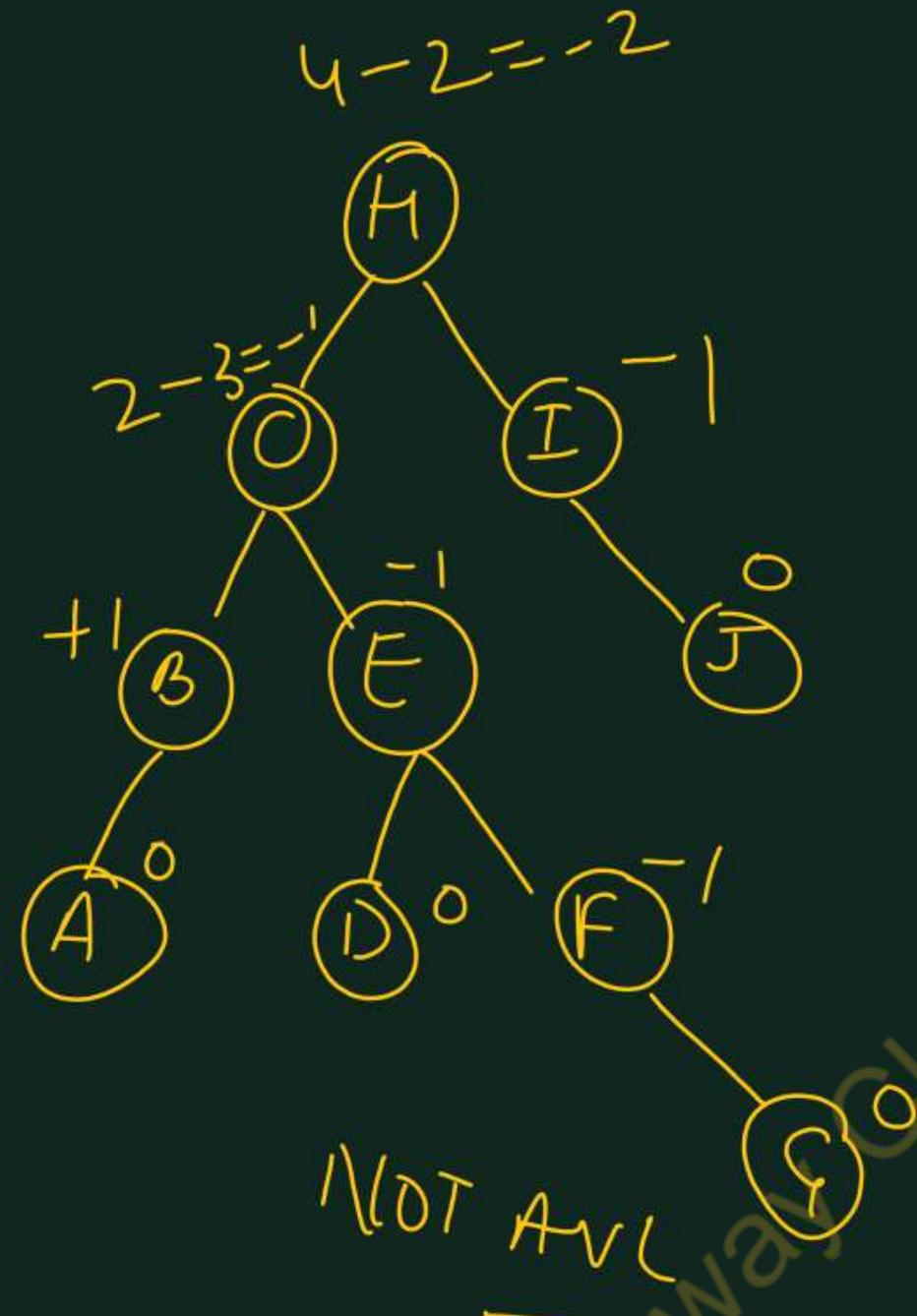
AVL

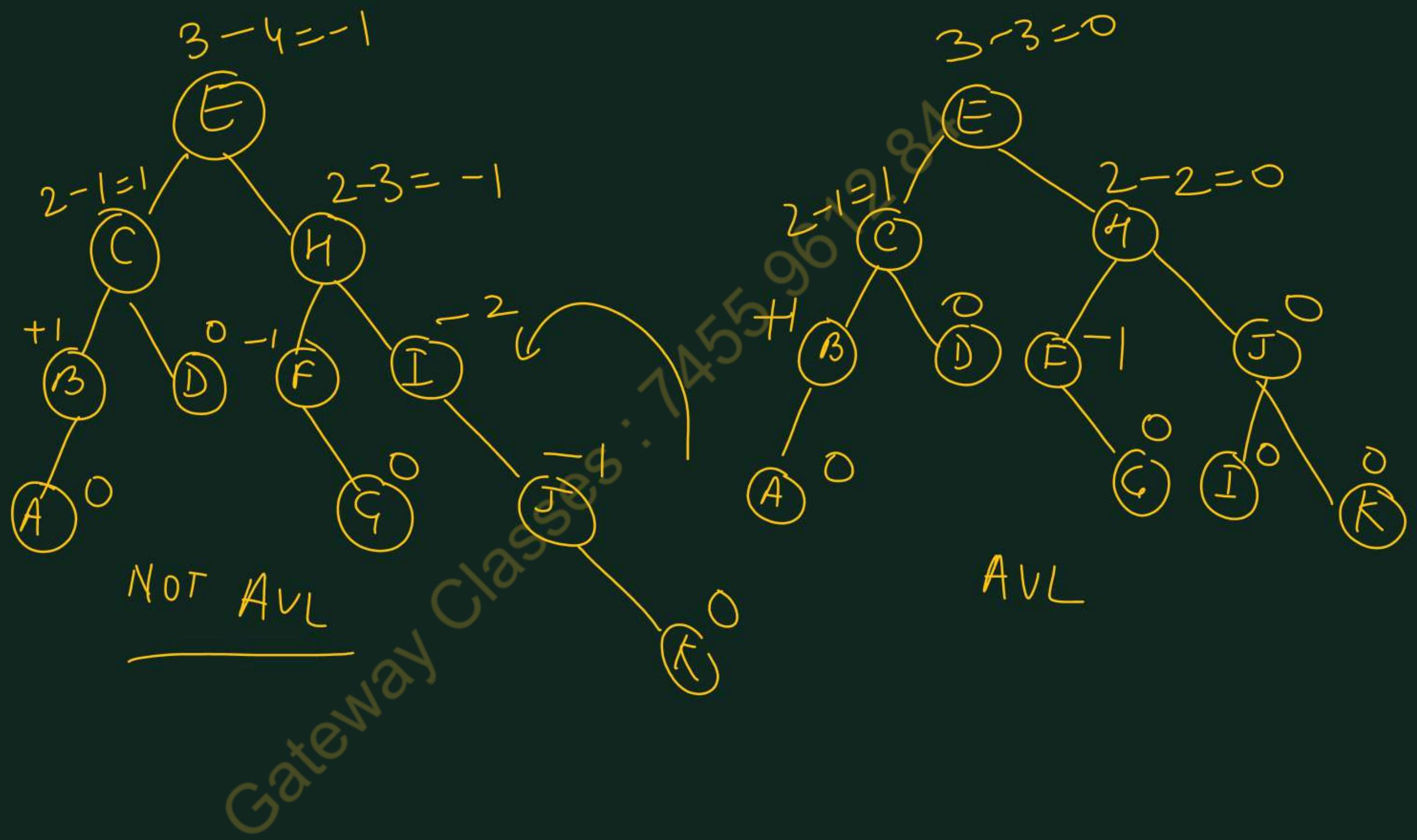


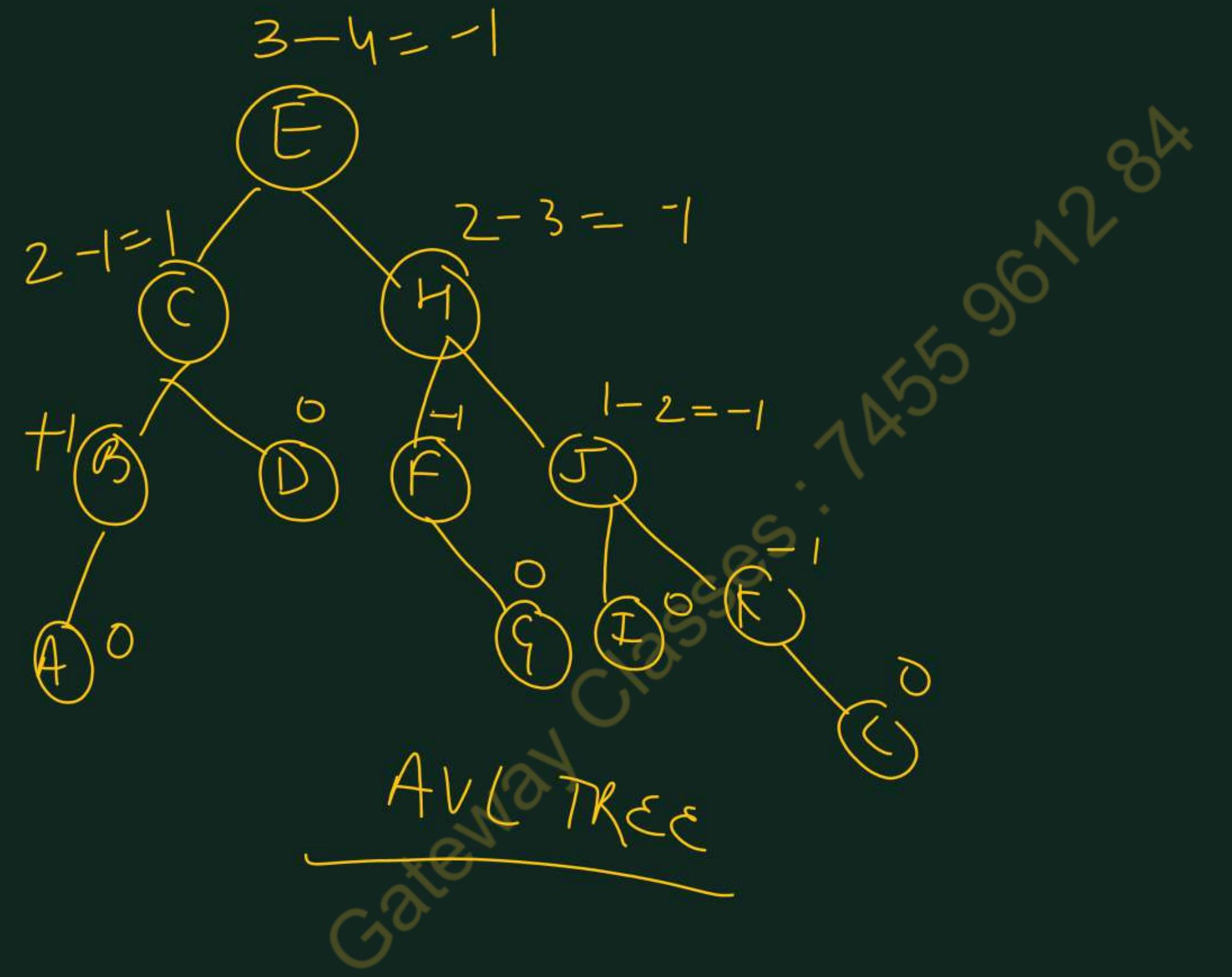
Gateway Classes : 7455 9612 84











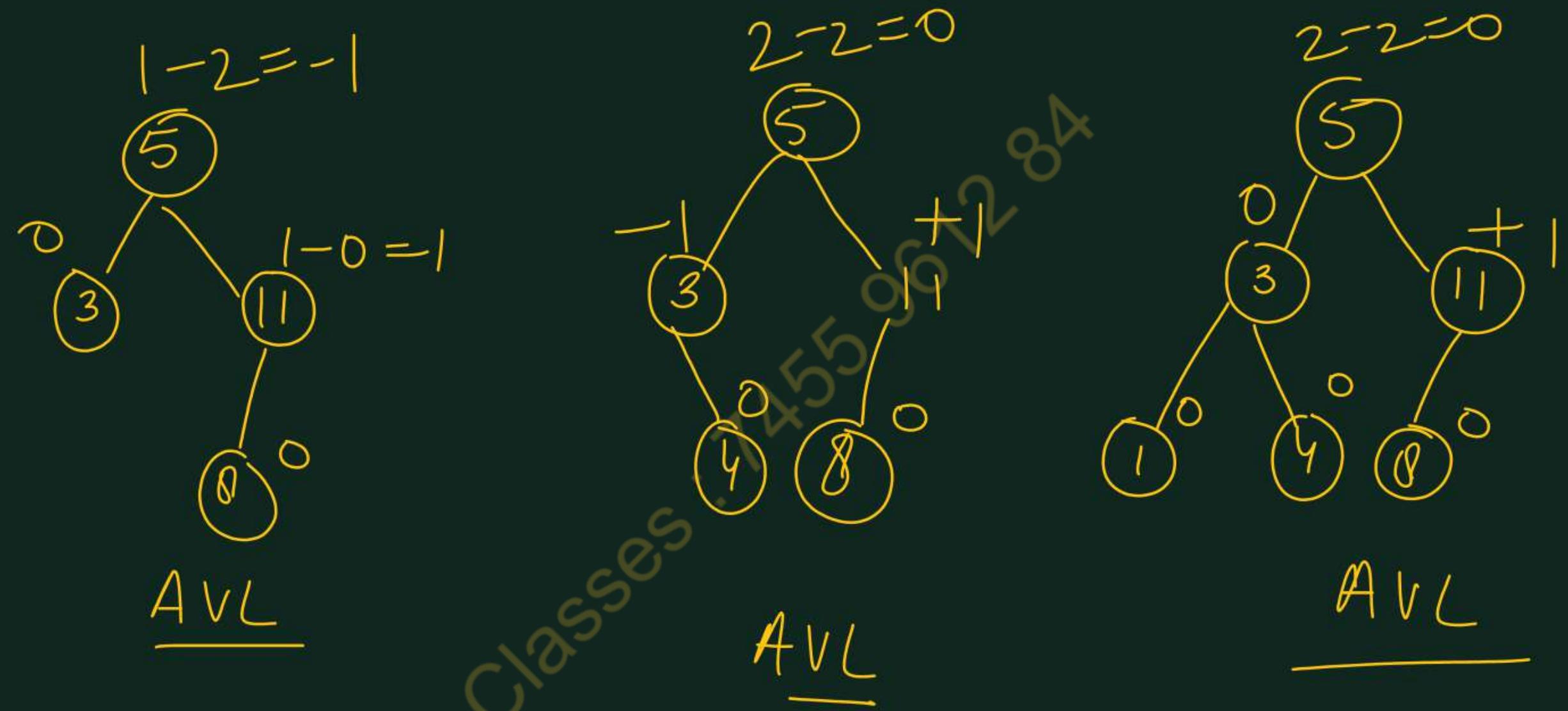
Example :- Construct an AVL Tree for the following

data elements :-

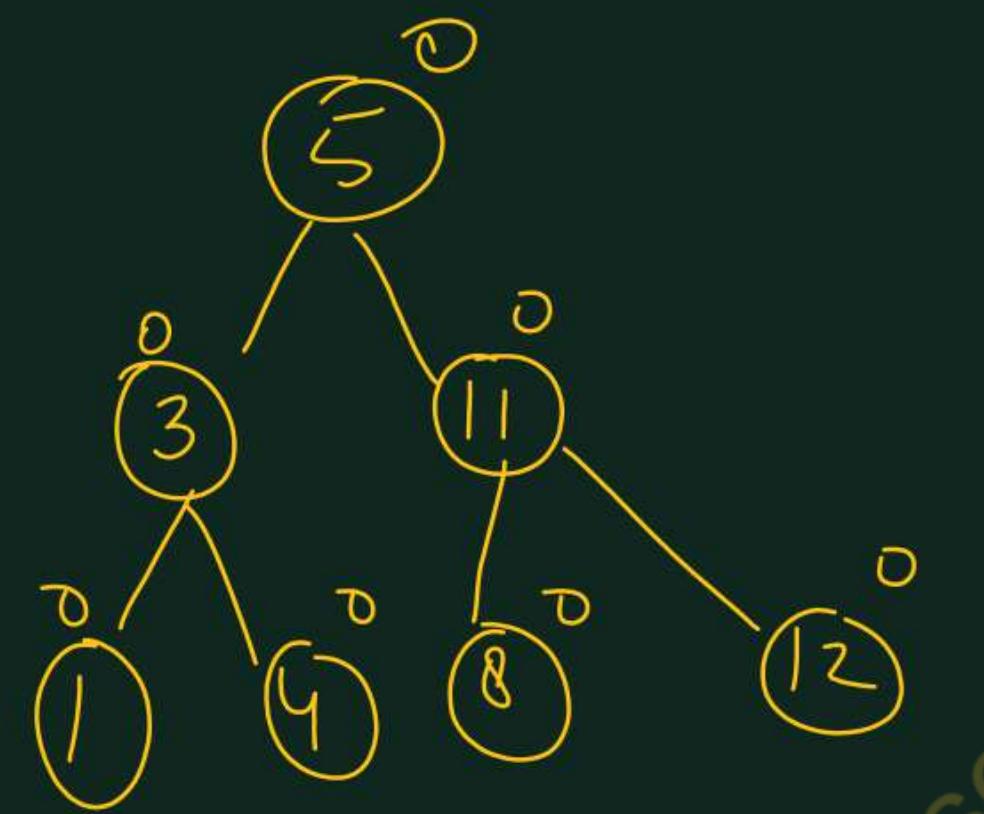
3, 5, 11, 8, 4, 1, 12, 7, 2, 6, 10

Solution :-

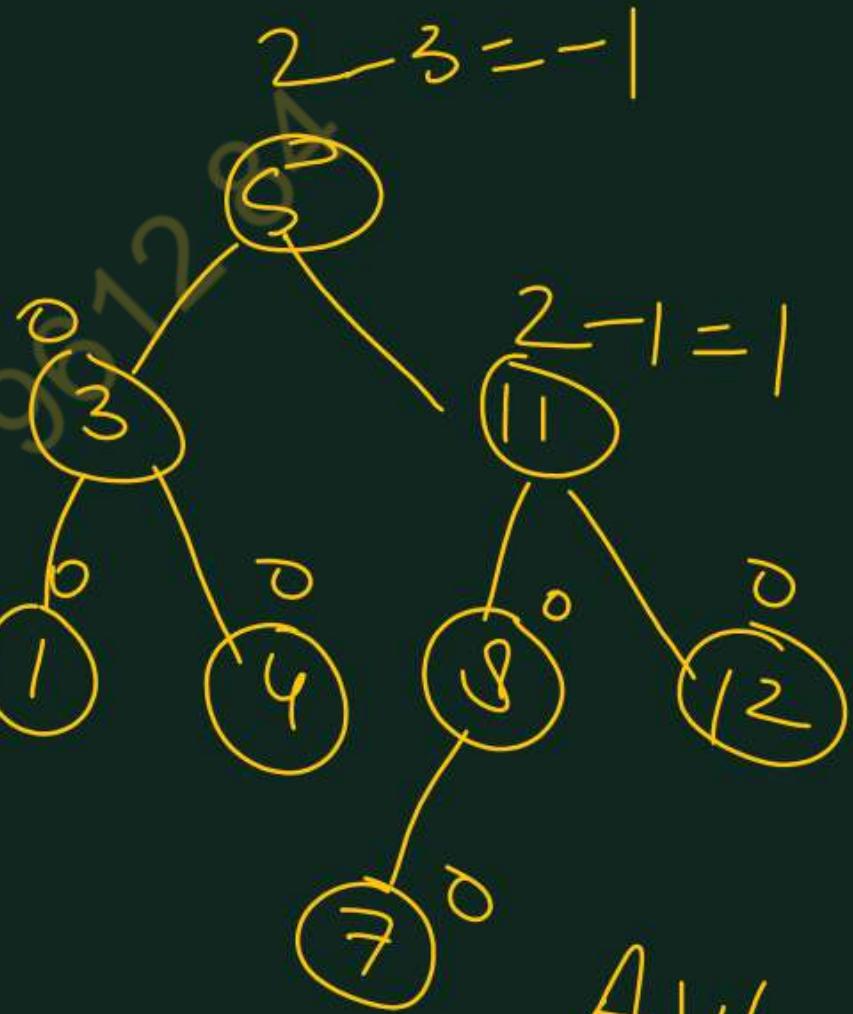




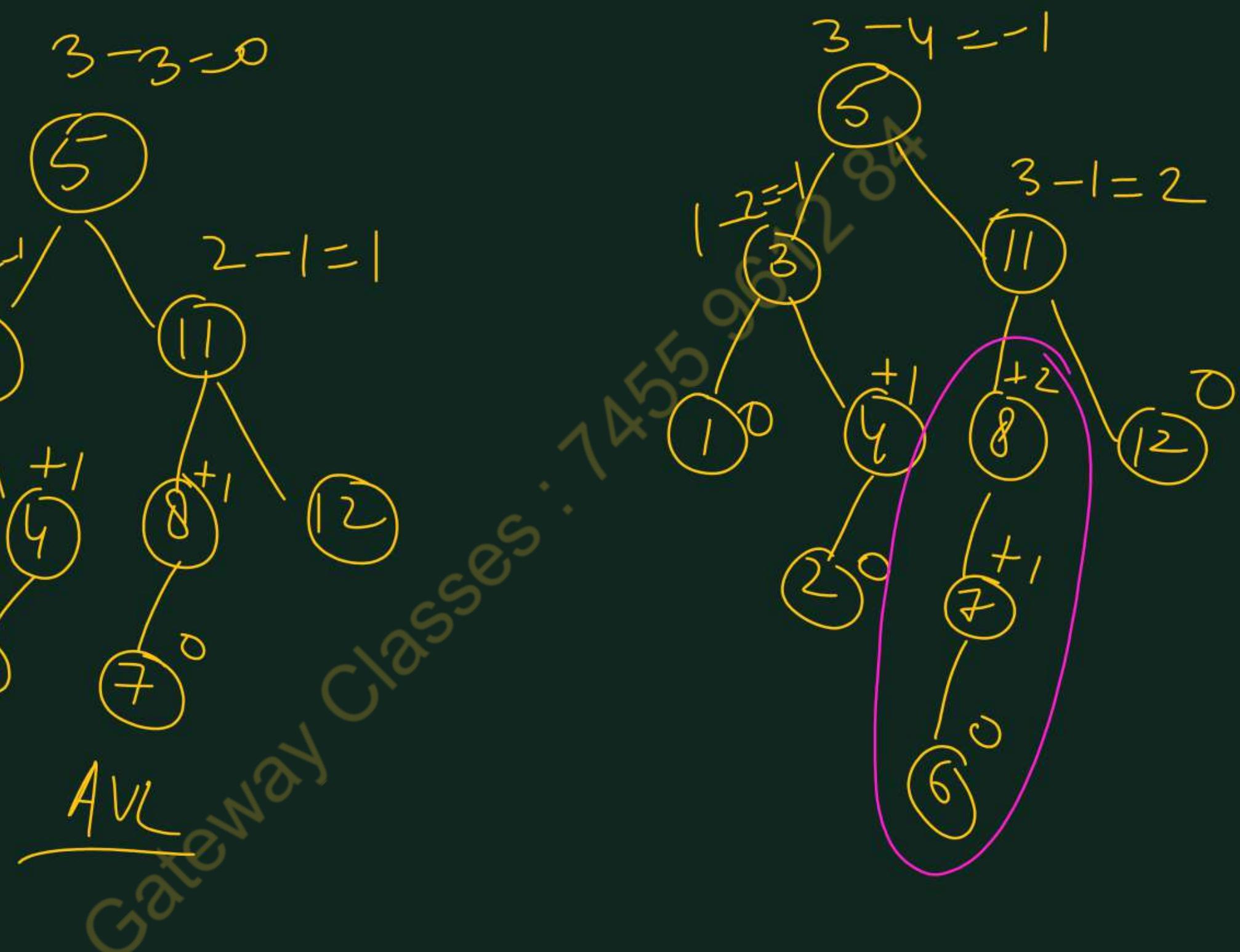
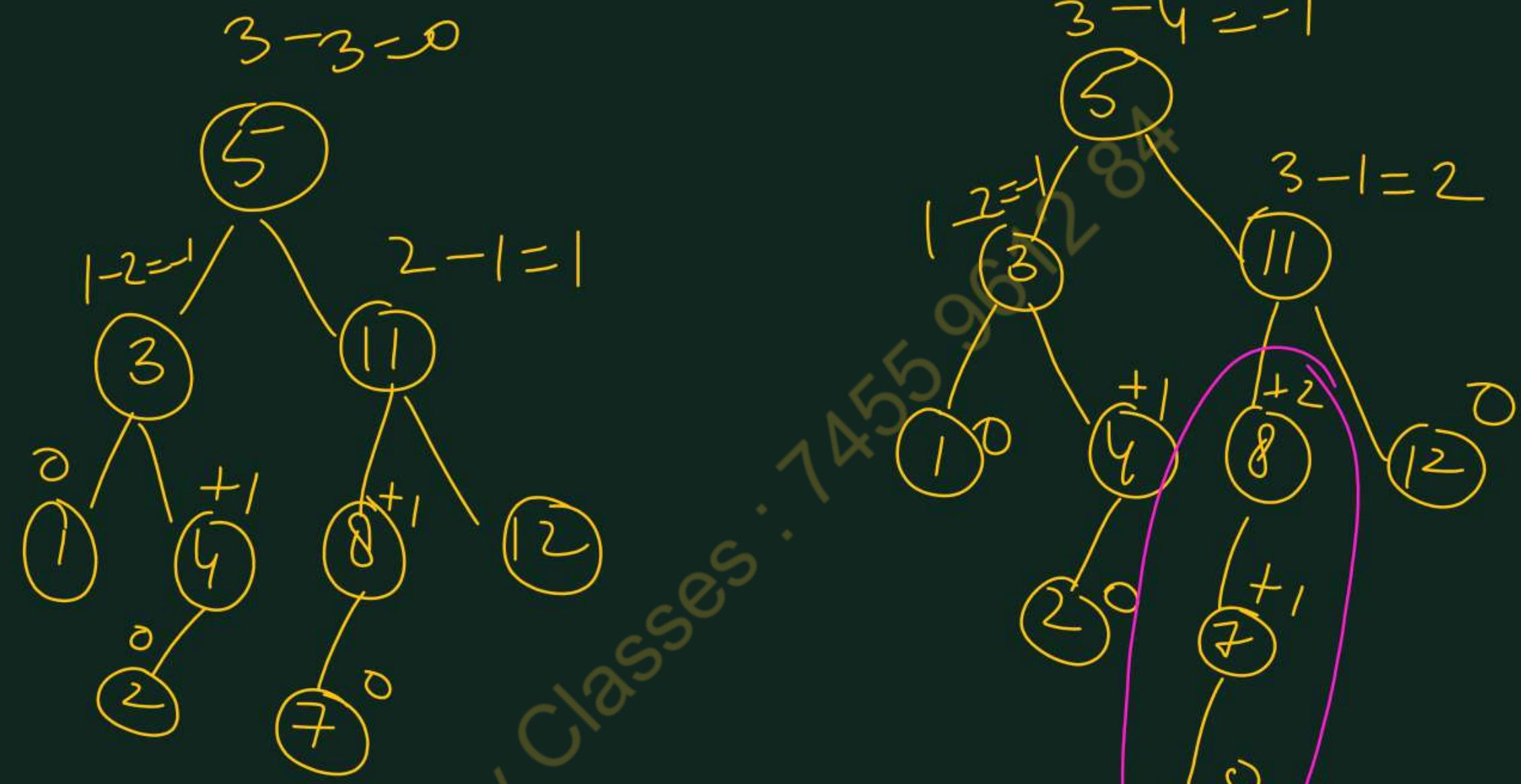
Gateway Classes

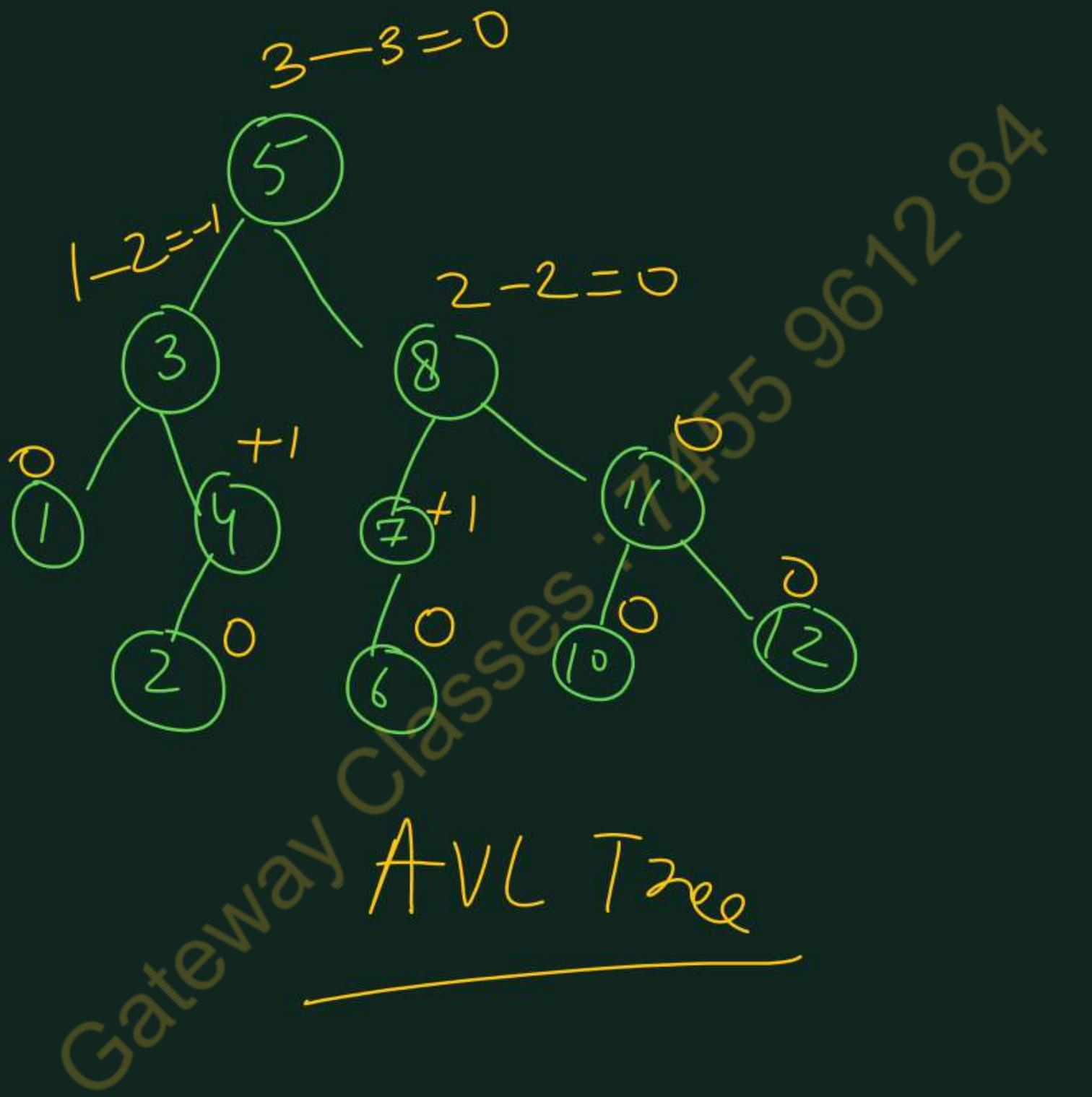


AVL



AVL

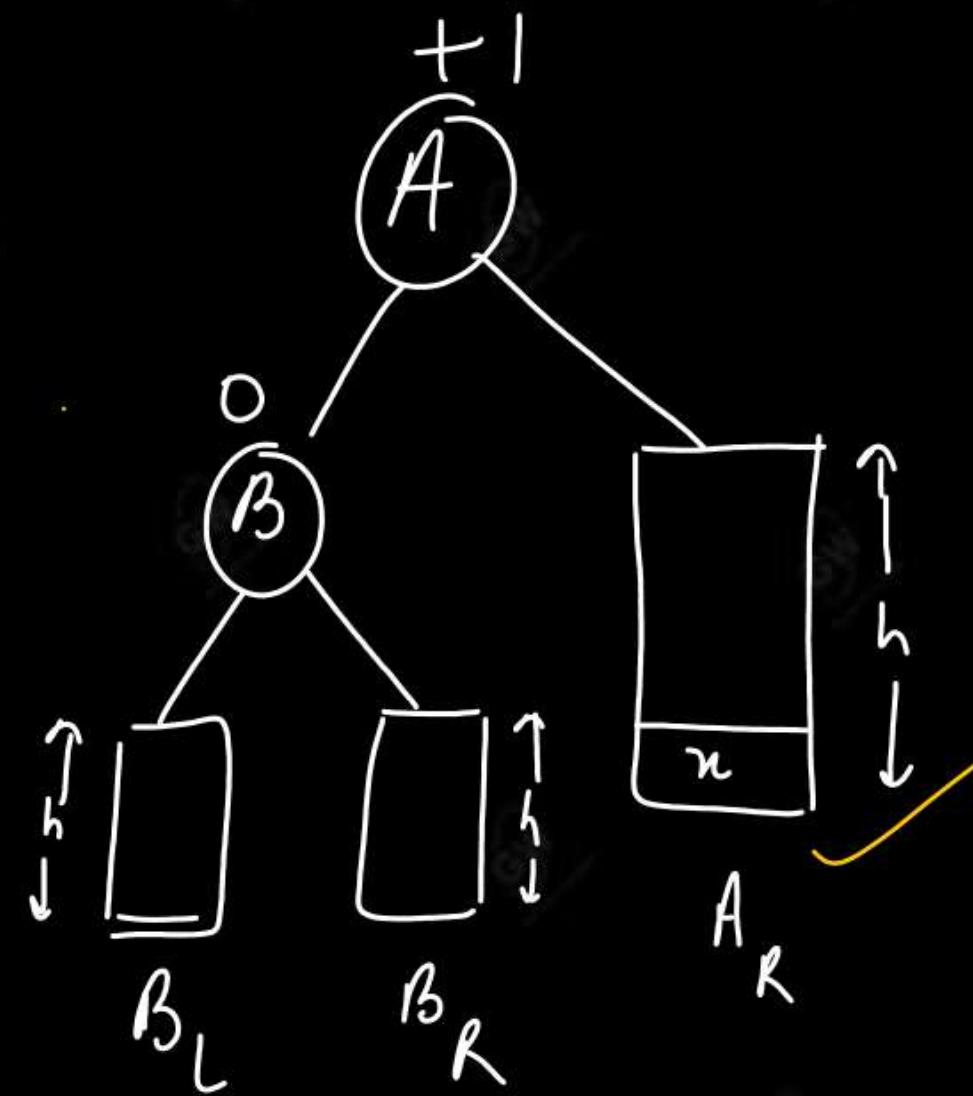




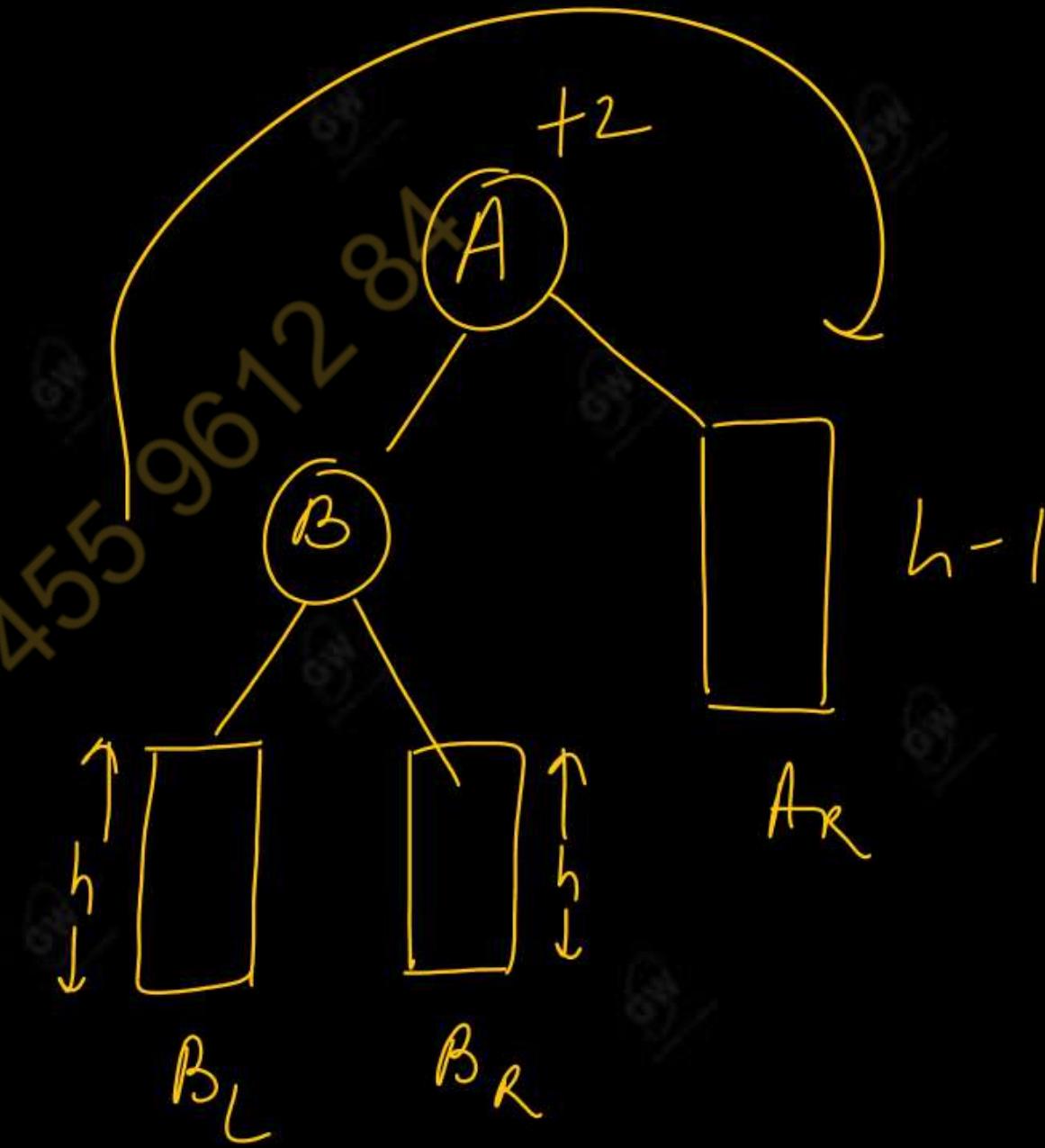
DELETION IN AN AVL TREE

- The deletion of a node from an AVL tree is exactly the same as the deletion of a node from the BST.
- The sequence of steps to be followed in deleting a node from an AVL tree is as follows:
 1. Initially, the AVL tree is searched to find the node to be deleted.
 2. The procedure used to delete a node in an AVL tree is the same as deleting the node in the binary search tree.
 3. After deletion of the node, check the balance factor of each node.
 4. Rebalance the AVL tree if the tree is unbalanced. For this, AVL rotations are used.

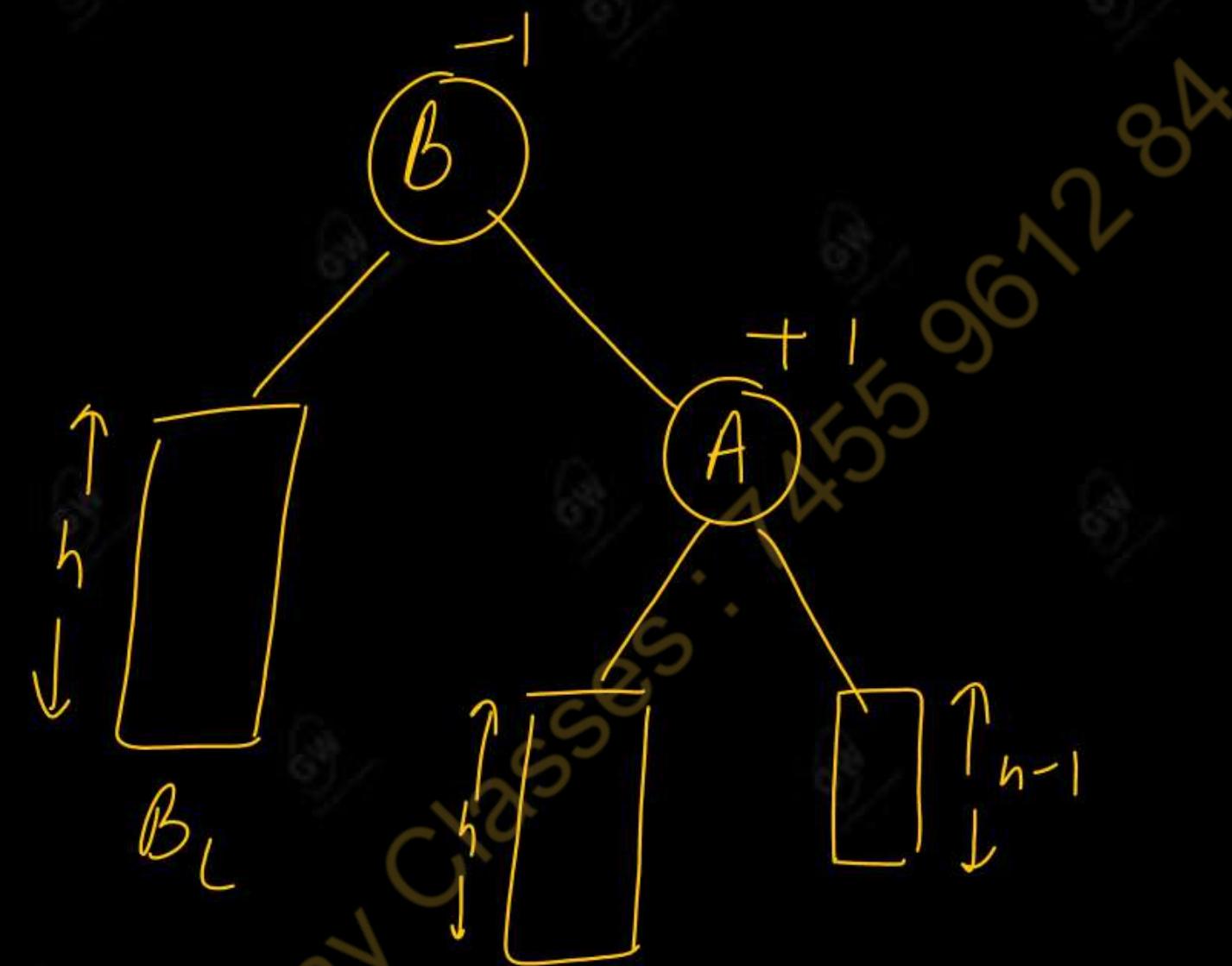
- On deletion of a node x from the AVL tree, Let A be the closest ancestor node on the path from x to the root node, with a balance factor of +2 or -2.
- To restore balance, the is classified as L or R depending on whether the deletion occurred on the left or right sub-tree of A.
- Now depending on the value of balance factor of B, where B is the root of the left or right sub tree of A, the R or L rotation is further classified as R0, R1 and R-1 or L0, L1 and L-1.
- RO Rotation and LO Rotation: When balance factor of B is 0. RO rotation is used.



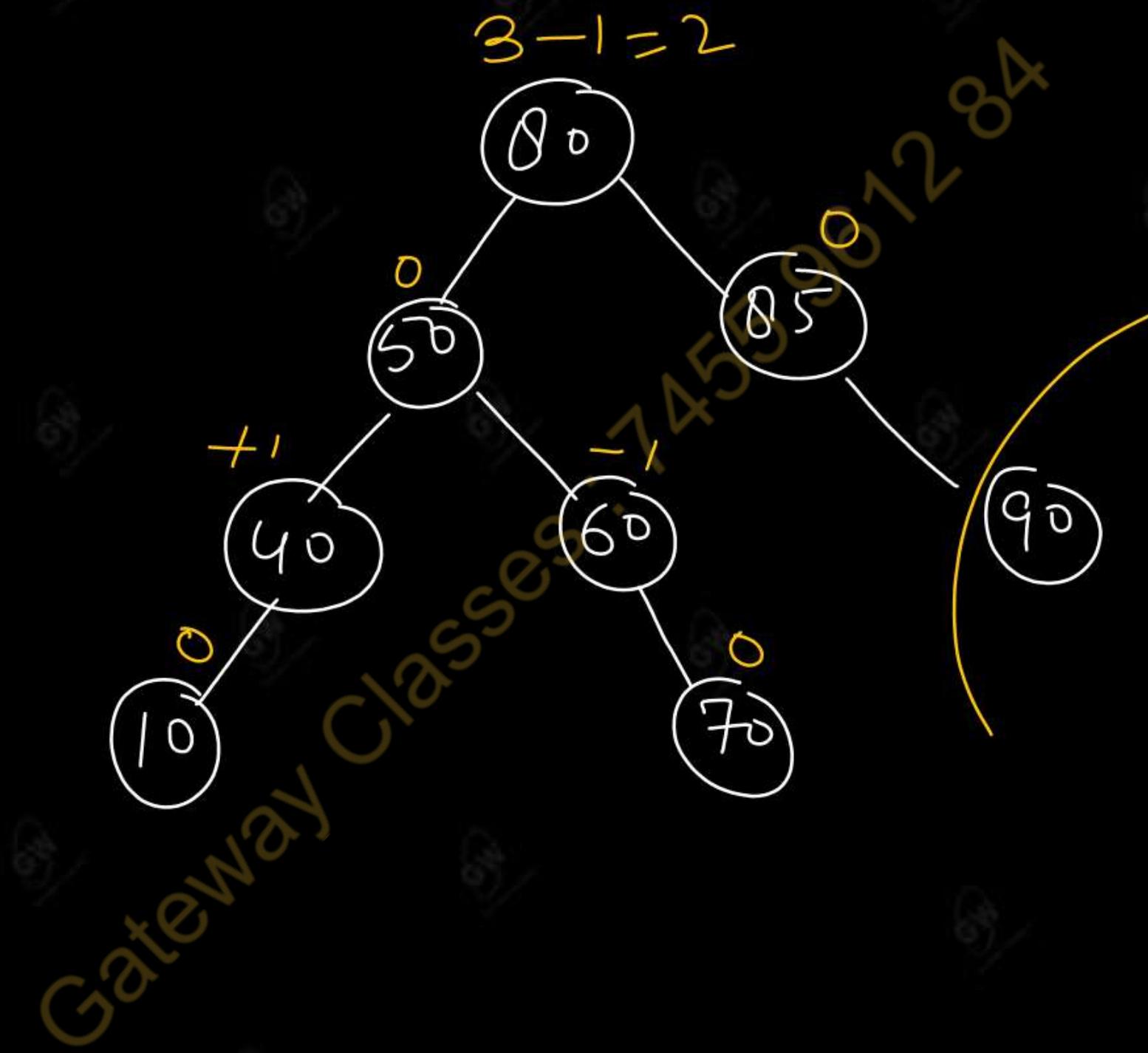
Delete
X
→



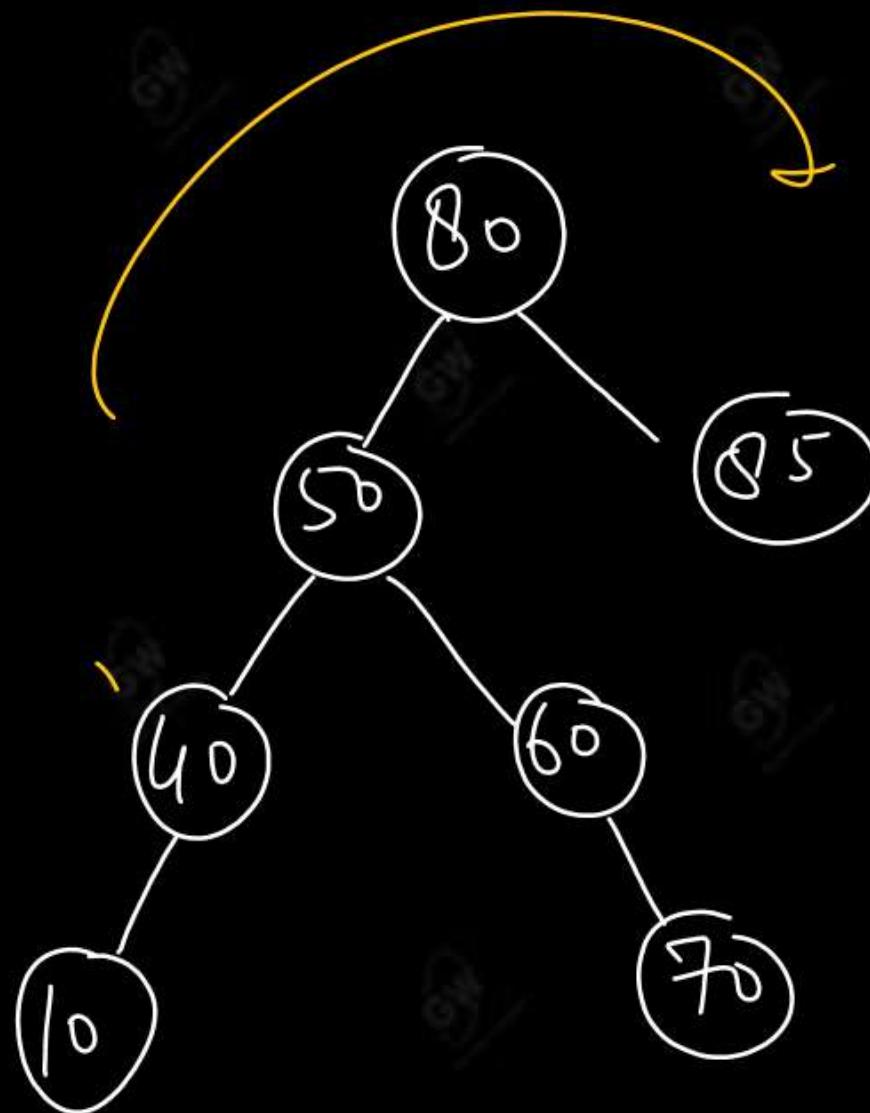
Unbalanced AVL Tree
-ter deletion



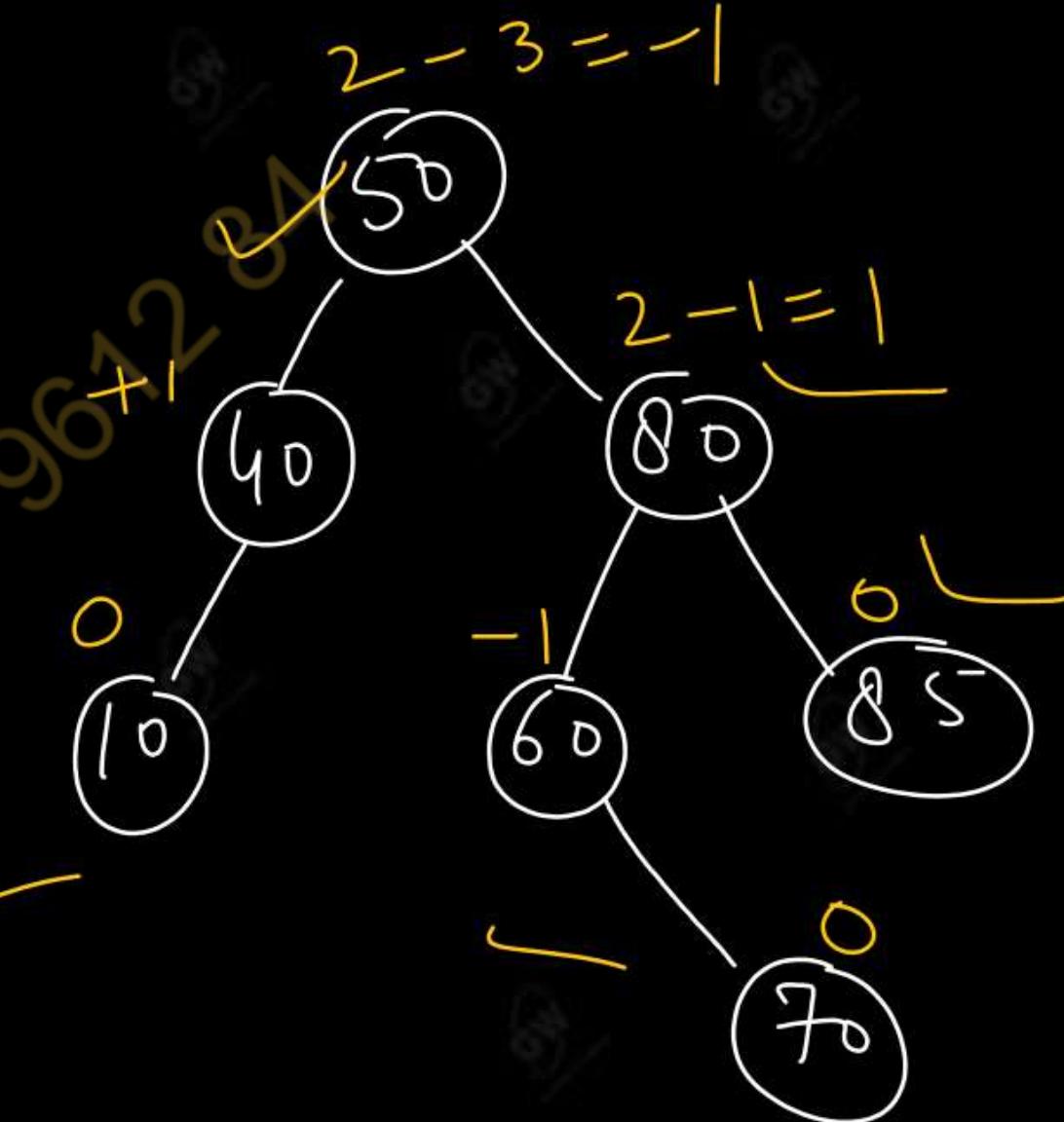
Example: Delete 90 from the AVL search tree shown in the following figure.



After deletion of 90, we get unbalanced AVL tree, as

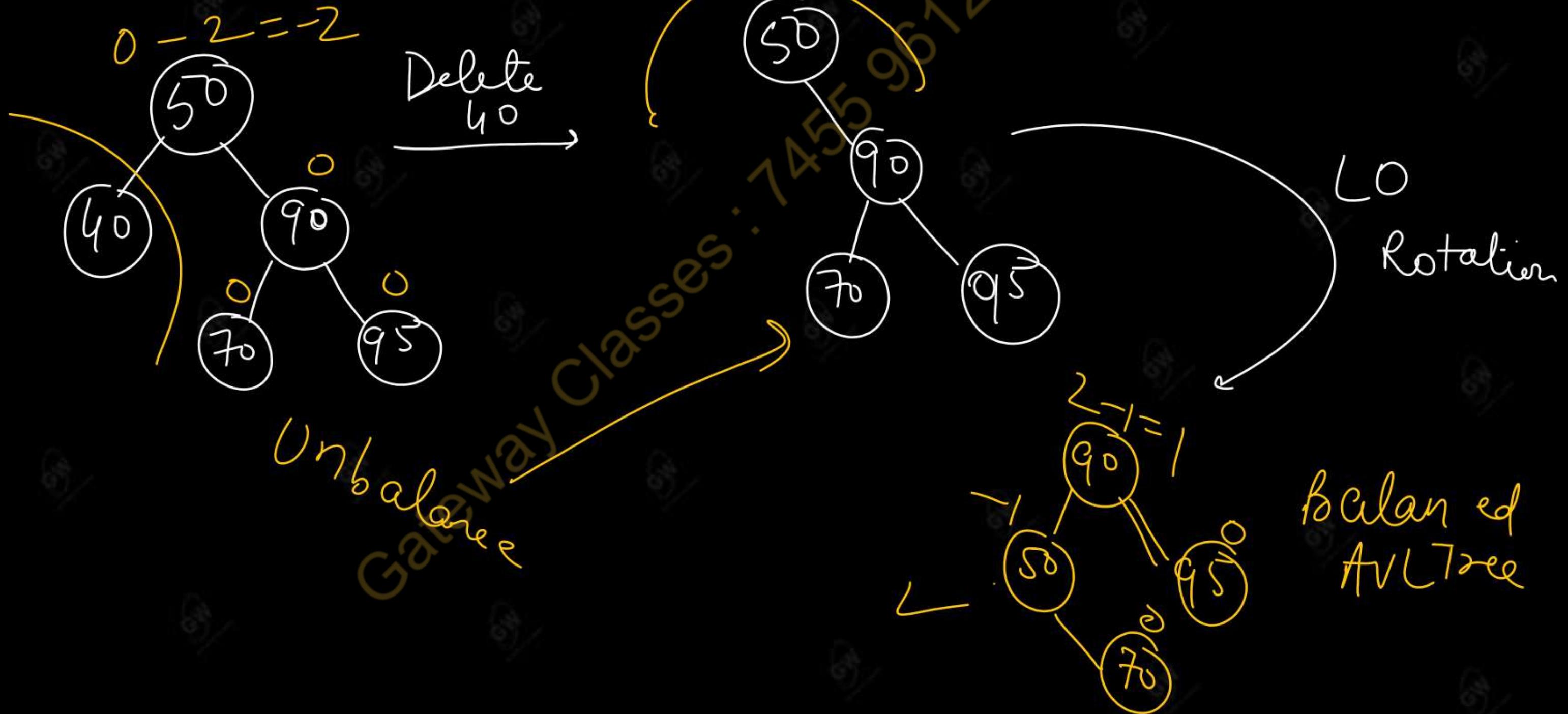


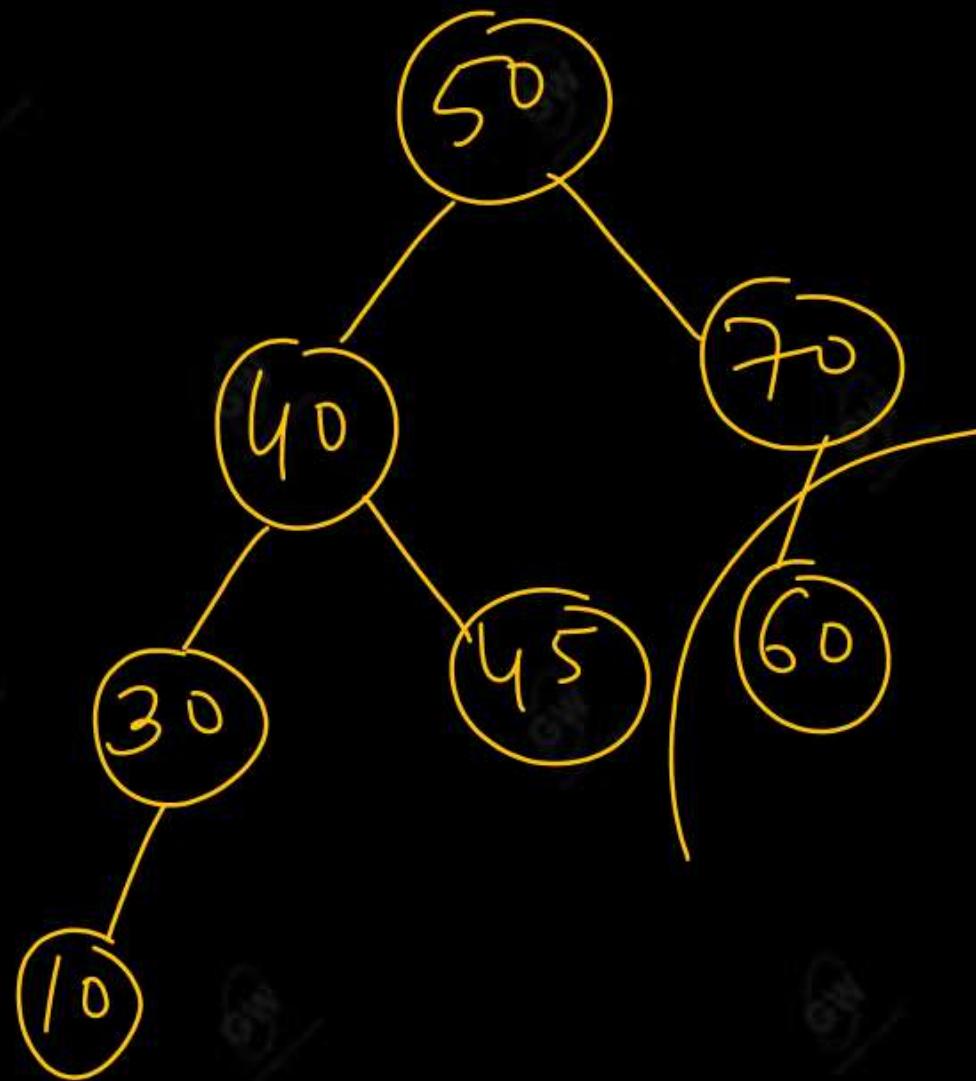
R_0
Rotation



Balanced AVL Tree
after R₀ Rotation

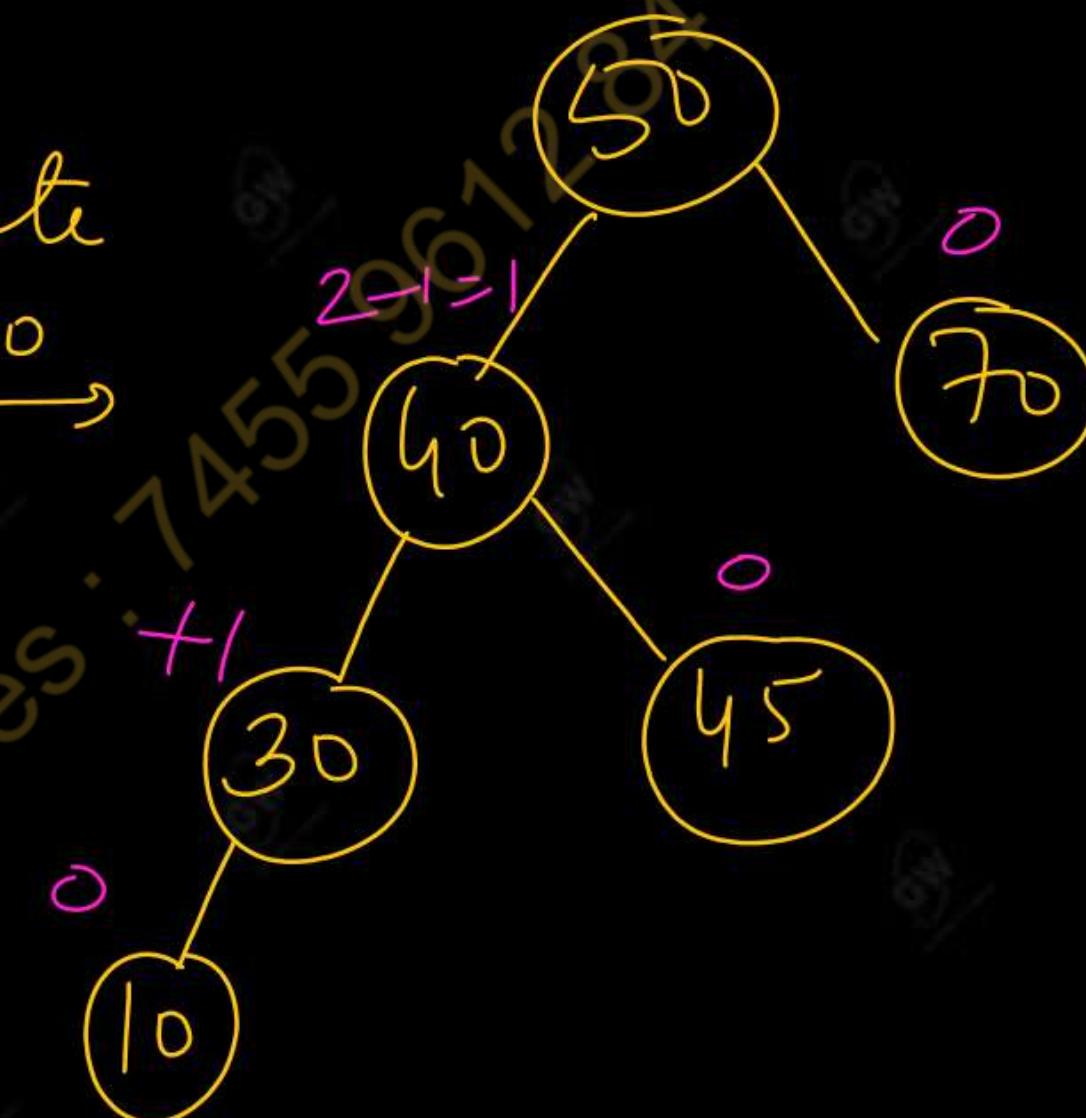
- Similarly LO rotation is executed, when balance factor of B is zero and B is the root of right sub-tree of A.
- LO rotation is executed as illustrated in the next figure.



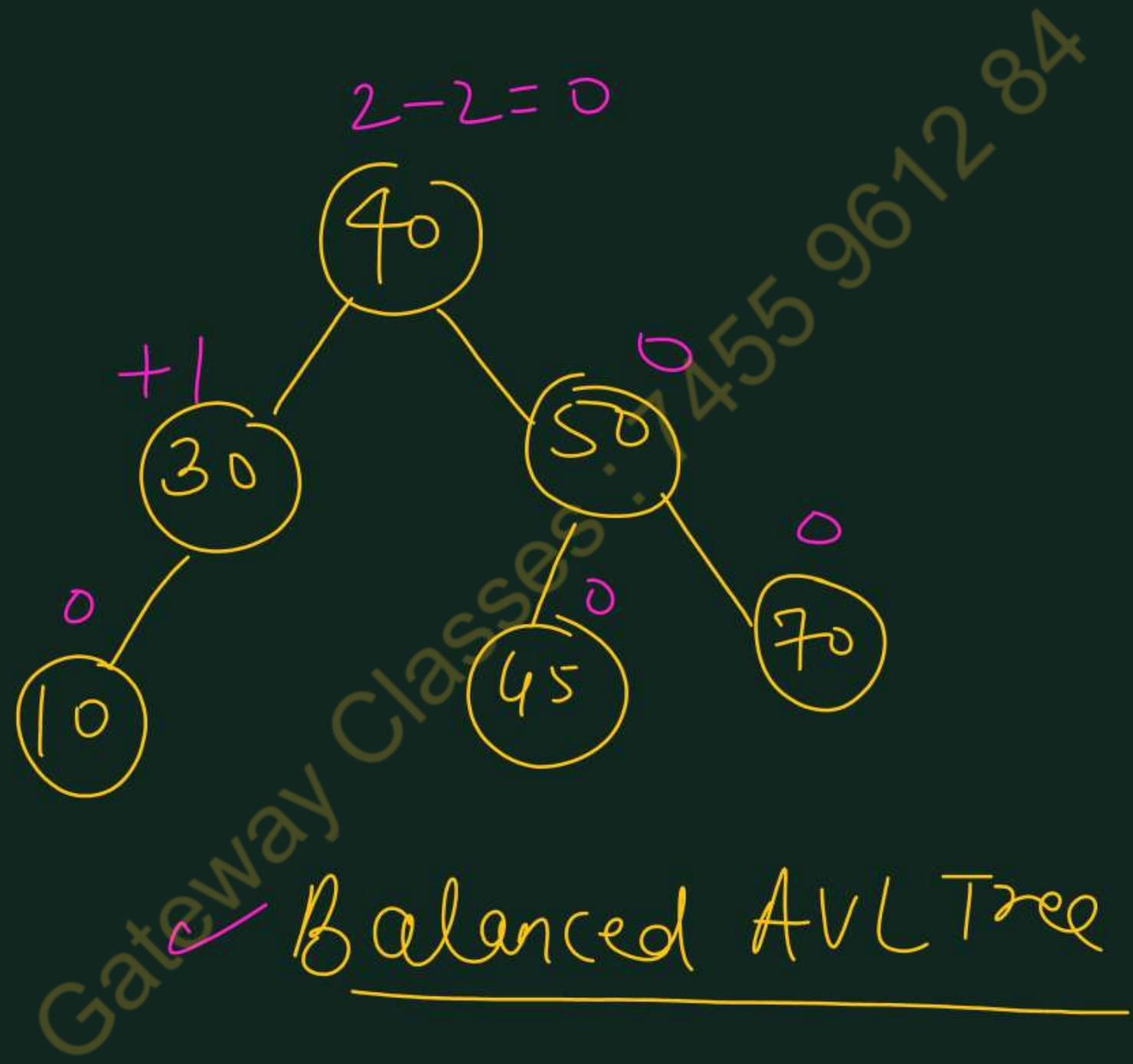
Example -

Delete
60

$$3 - 1 = 2$$

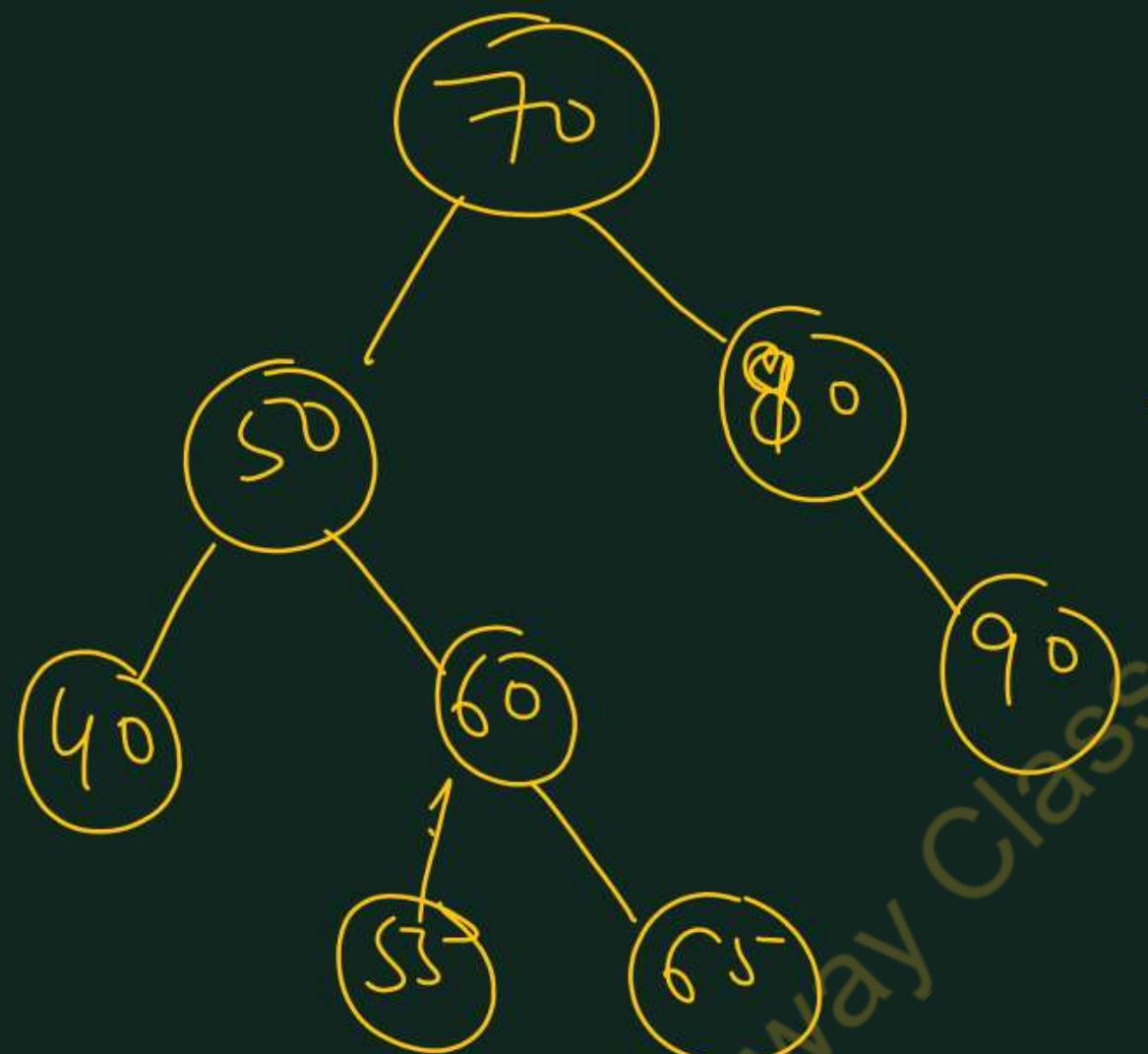


R /
Rotation

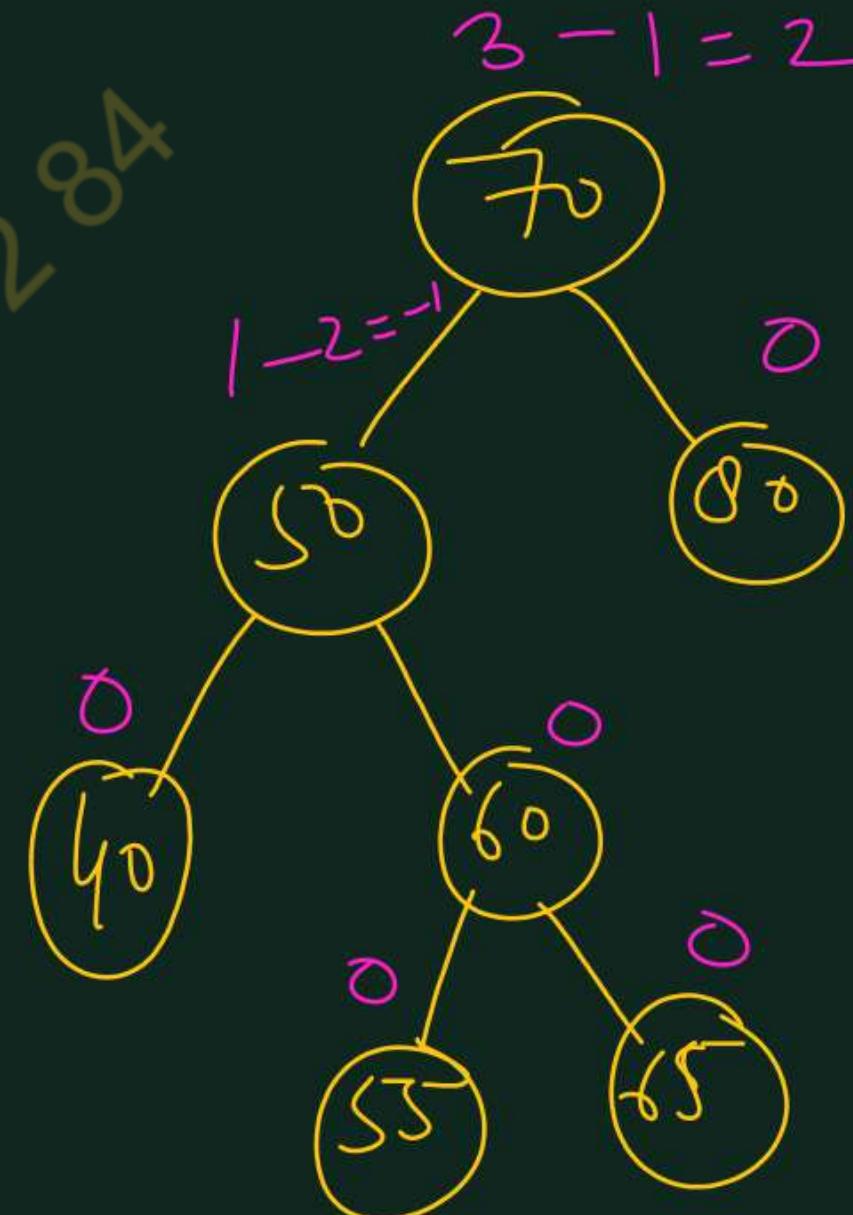


Balanced AVL Tree

Example -



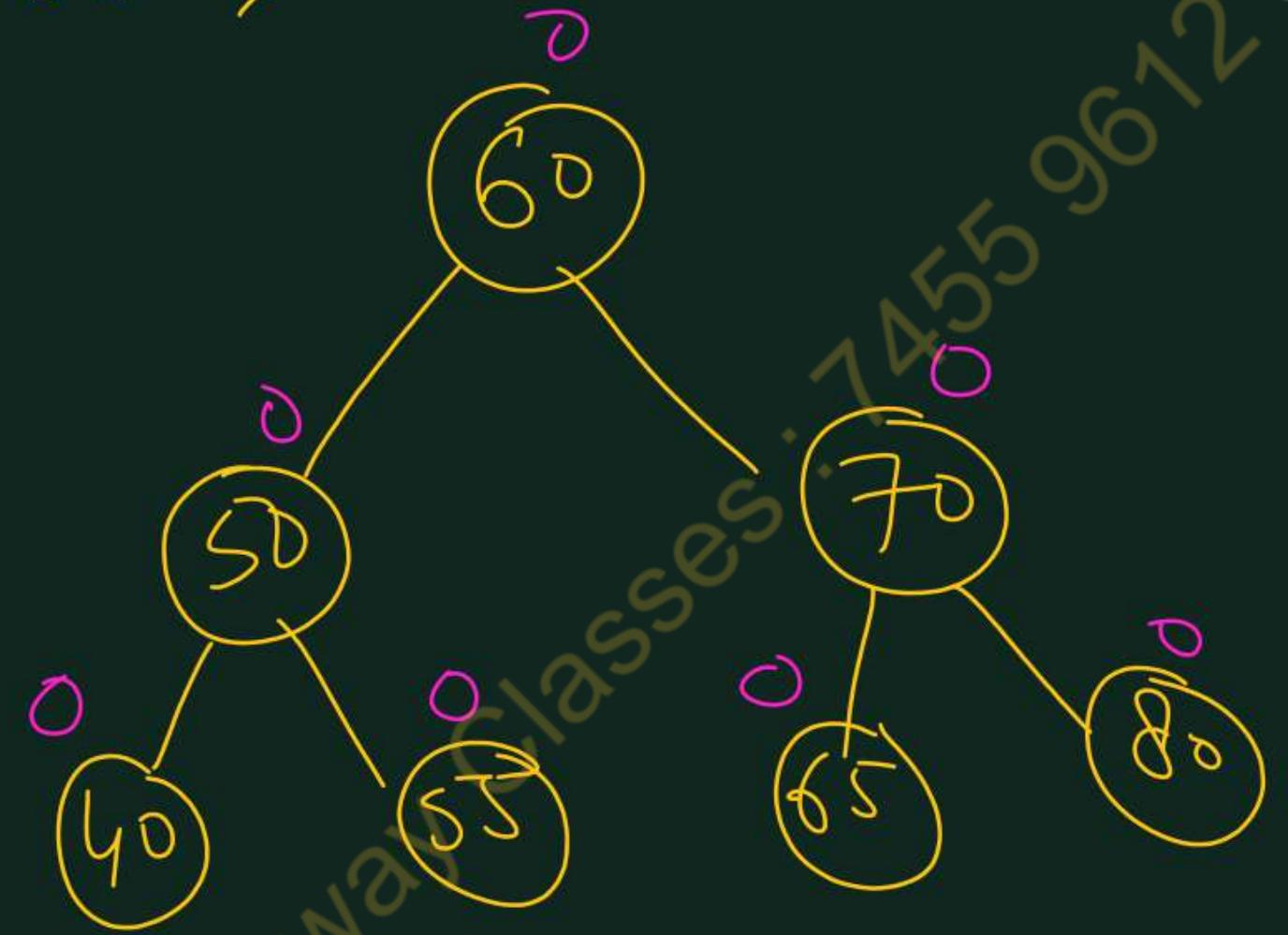
Delete 90



$$3 - 1 = 2$$

Unbalanced

By $R(-1)$ rotation, we get -



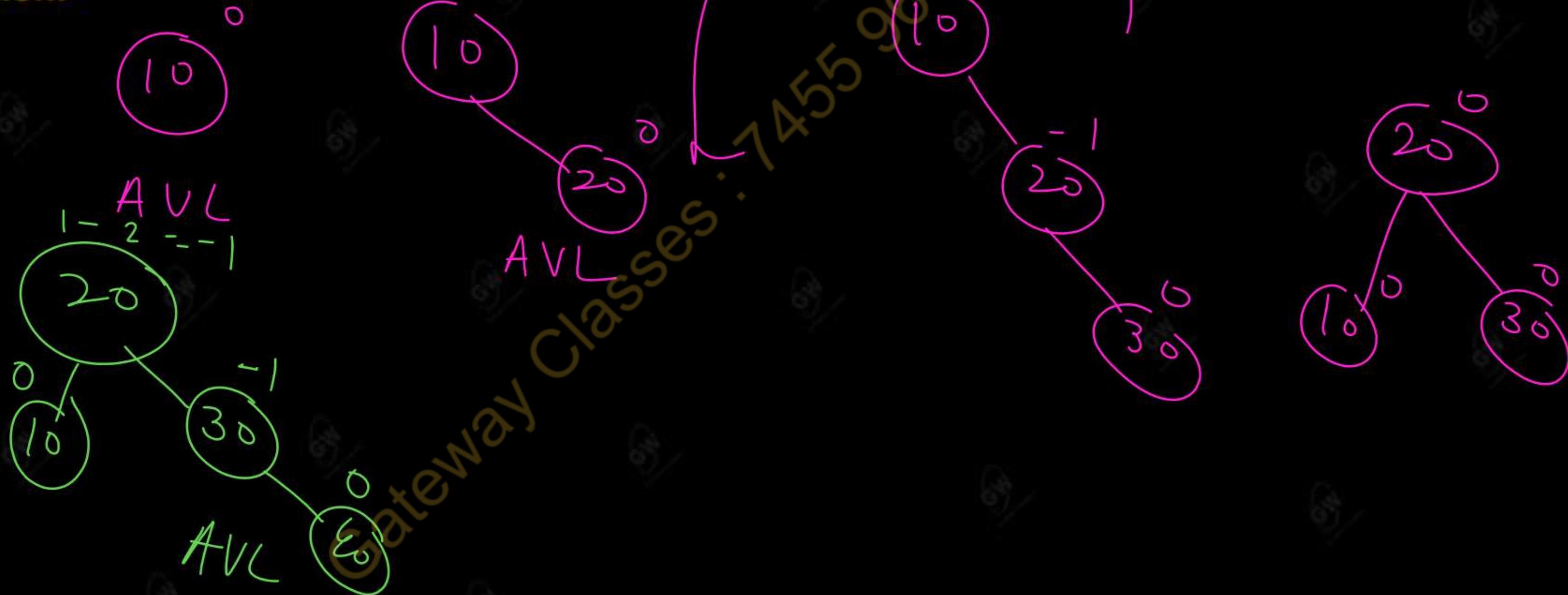
Balanced
AVL
Search Tree

Example: Insert the following keys in order shown to construct an AVL tree

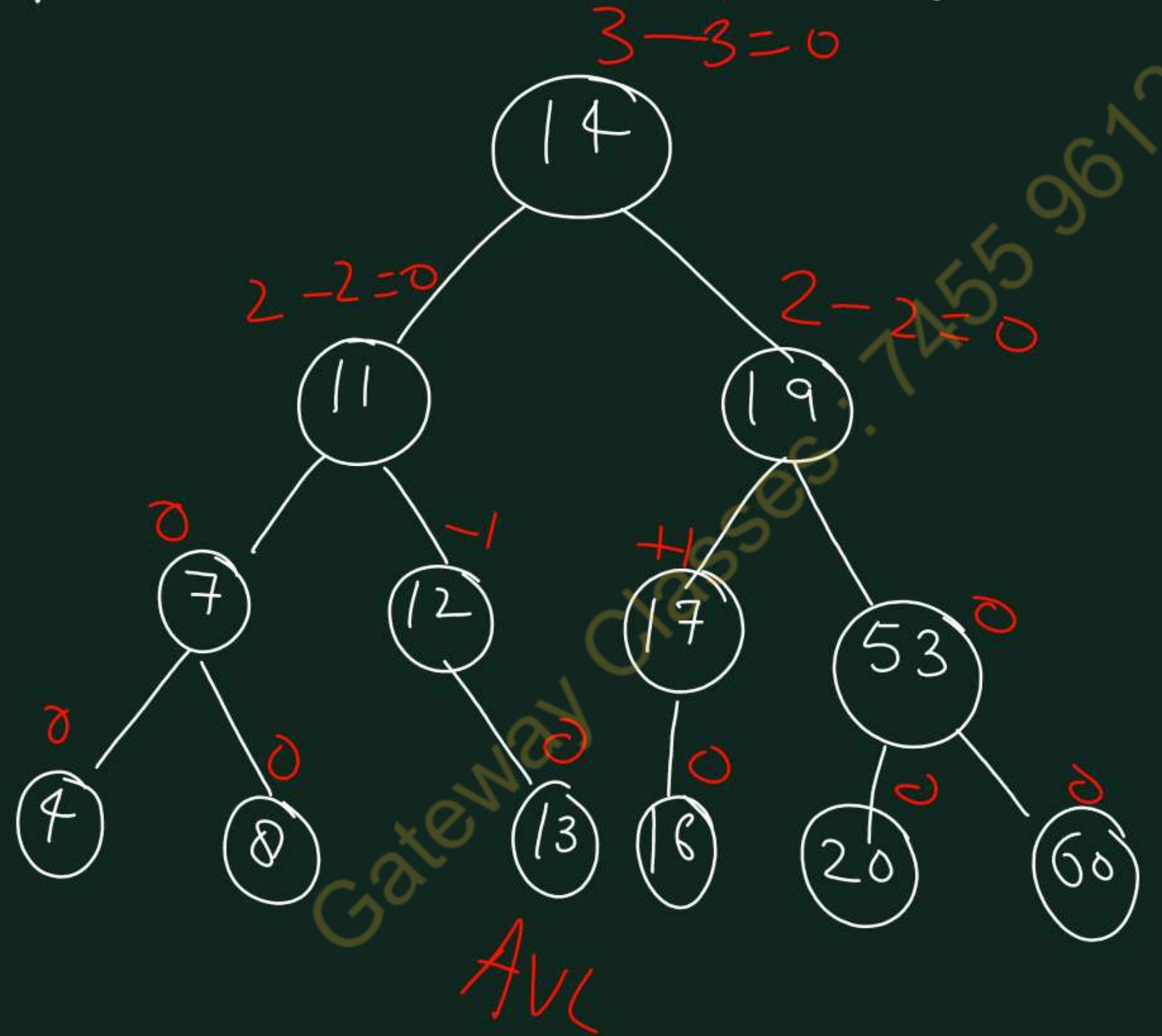
10, 20, 30, 40, 50

Delete the last two keys in the order of LIFO.

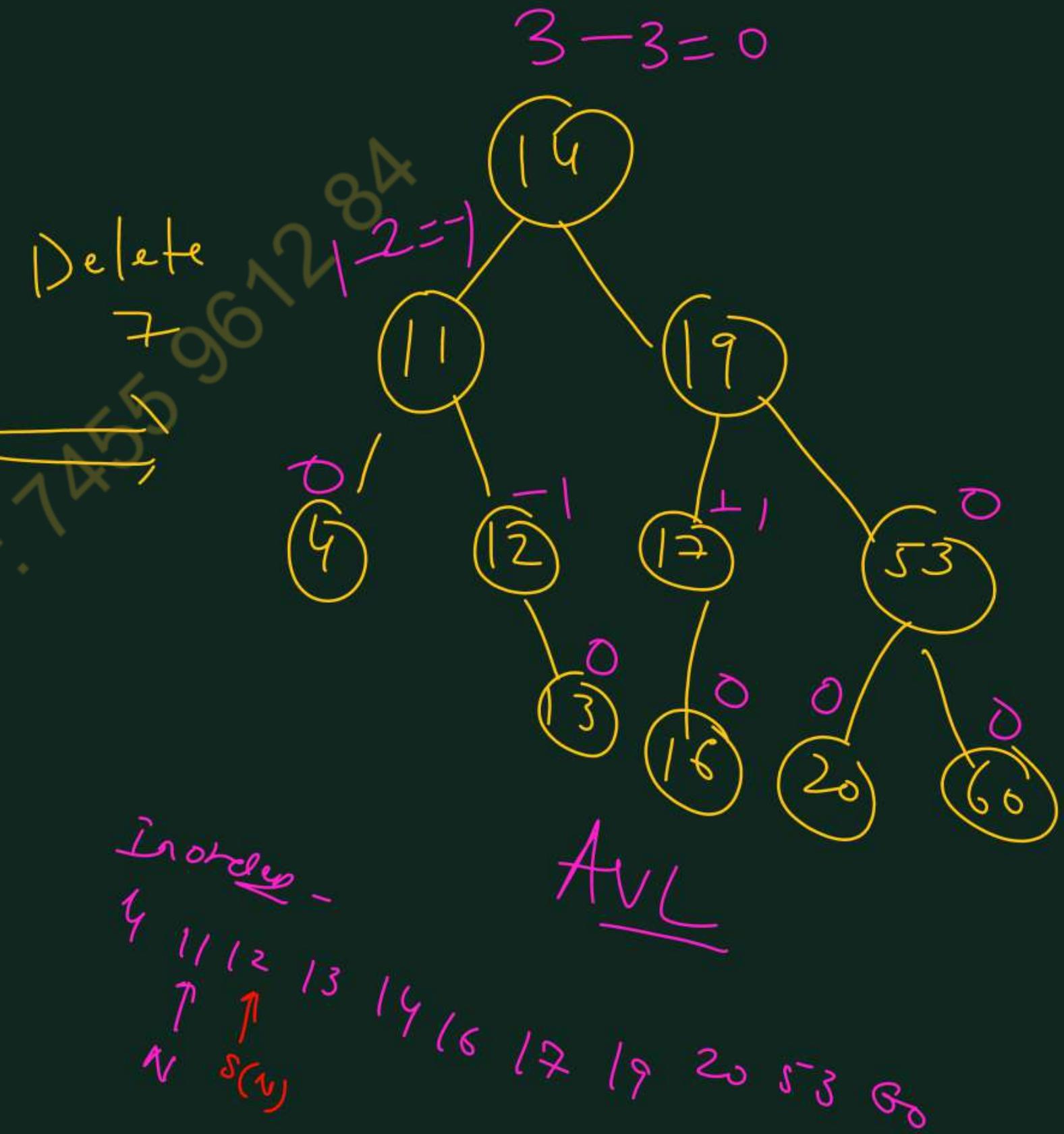
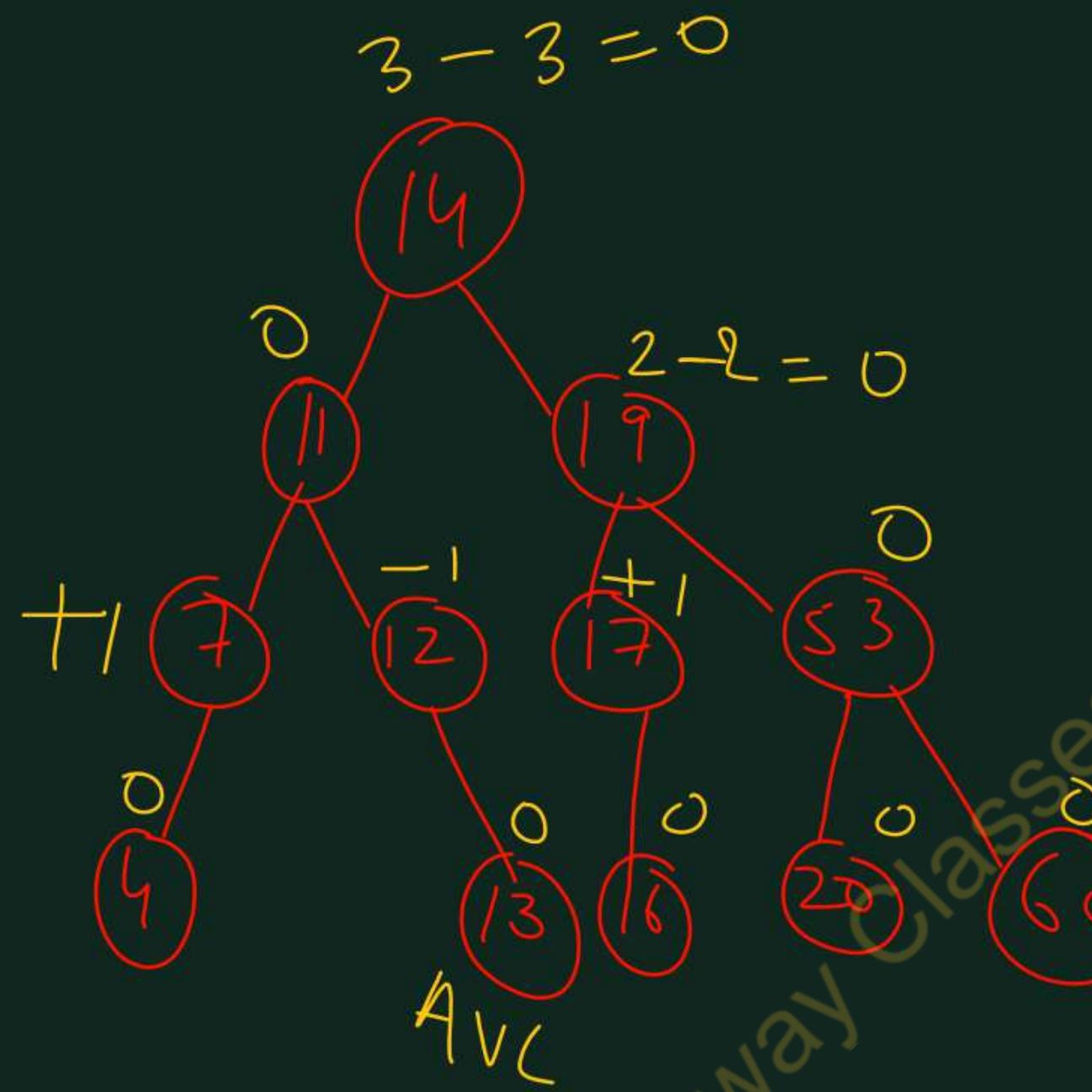
Solution:



Example :- Consider the following AVL Tree:-



Show the deletion
of 8, 7, 11, 14 &
17.



Q.1 What happens if the binary search tree is left oriented or right oriented? Explain the problem and give the solution.

2014-15 5 marks

Q.2 Construct the AVL tree with the following keys:-

35, 36, 80, 85, 67, 89, 25, 16, 10, 14, 14

2014-15 5 marks

Q.3 What is the maximum height of any AVL tree with 7 nodes?

2015-16 , 2 marks

Q.4 Describe all rotations in AVL tree. Construct AVL tree from the following nodes :

B, C, G, E, F, D, A

2015-16 5 Marks

Q.5 Define AVL trees. Explain its rotation operations with example. Construct an AVL tree with the values 10 to 1 numbers into an initially empty tree.

2016-17 15 marks

Q.6 Explain height balanced tree. List general cases to maintain the height.

2017-18 2 marks

Q.7 Construct the following AVL tree and insert 2, 12, 7 and 10 as new node. Show proper rotation to maintain the tree as AVL.

2017-18 7 marks

- Q.8 Calculate the minimum number of nodes in AVL tree with height 8.** 2017-18 2 marks
- Q.9 Describe an AVL tree. Construct an AVL tree by inserting the following elements in the order of their occurrence { 60, 2, 15, 20, 12, 115, 90 and 88 }** 2018-19, 7 marks
- Q.10 Insert the following sequence of elements into an AVL tree, starting with empty tree 71, 41, 91, 56, 60, 30, 40, 80, 50, 55 also find the minimum array size to represent this tree.** 2020-21, 10 marks
- Q.11 Define AVL tree. Insert 14, 17, 11, 7, 53, 4, 13 into an empty AVL tree.** 2020-21, 7 marks
- Q.12 Write advantages of AVL tree over BST.** 2021-22 2 marks
- Q.13 Discuss left skewed and right skewed binary tree. Construct an AVL tree by inserting the following elements in the order of their occurrence: 60, 2, 14, 22, 13, 111, 92, 86** 2022-23, 10 marks

Today's Target

- B Tree (m-way search tree, B-tree Insertion)**
- AKTU PYQs**

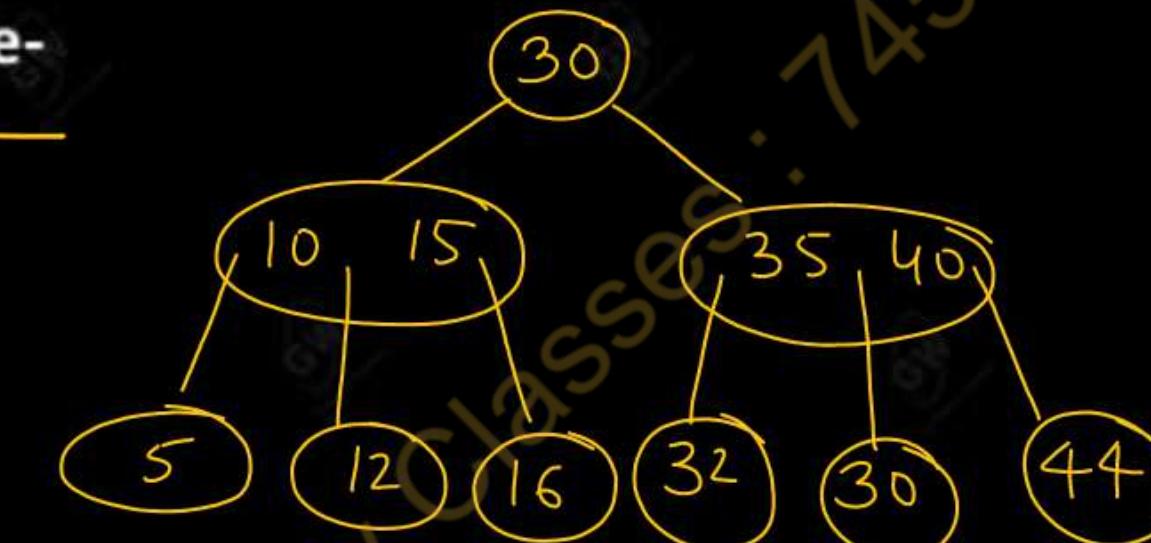
Q.1 What is m way search tree? Construct a B-tree from the following elements:-

65, 71, 70, 66, 75, 68, 72, 77, 74, 69, 83, 73, 82, 88, 67, 76, 78, 84, 85, 80

2014-15, 10 marks

Q.2 Define B-tree. What do you understand by the order of B-tree?

Consider the following B tree-



Show the B tree after the following operations- insert 43, insert 50, delete 15.

2014-15, 10 marks

Q.3 Define a B tree. What are the applications of B-tree? Draw a B-tree of order 4 by insertion of the following keys in order : Z, U, A, I, W, L, P, X, C, J, D, M, T, B, Q, E, H, S, K, N, R, G, Y, F, O, V

2015-16, 15 marks

Q.4 Construct a B tree on the following sequence of inputs:

10, 20, 30, 40, 50, 60, 70, 80, 90

Assume that the order of B-tree is 3.

Q.5 Show the results of inserting the keys F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B in order into an empty B-tree of order 5.

Q.6 Construct a B-tree of order 5 created by inserting the following elements:

3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19

Also delete elements 6, 23 and 3 from the conducted tree.

Q.7 What is B-tree? Generate a B-tree of order 4 with the alphabets (letters) arrive in the sequence as follows:

a g f b k d h m j e s l r x c l n t u p

2017-18, 7 marks

2018-19, 7 marks

2018-19, 7 marks

2019-20, 10 marks

Q.8 Construct a B tree of order 4 using following elements:

K, U, W, C, M, P, Y, A, E, Q, X, D, H, V, F, J, I, B, S, T

2020-21, 7 marks

Q.9 (i) Insert the following keys into an initially empty B tree of order 5

a, g, f, b, k, d, h, m, j, e, s, l, r, x, c, l, n, t, u, p

(ii) What will be the resultant B-tree after deleting keys j, t and d in sequence?

2021-22, 10 marks

Q.10 What is B-tree? Write the various properties of B-tree. Show the results of inserting the keys F, S, Q,

K, C, L, H, T, V, W, R, N, P, A, B in order in to an empty B-tree of order 5.

2022-23, 10 marks

- All the data structures discussed so far favour data stored in the internal memory and hence support internal information retrieval.
- However to favour retrieval and manipulation of data stored in external memory viz., storage devices such as disks etc. there is a need for some special data structures such as m-way search trees B trees and B+ trees.

WHY DO WE NEED ANOTHER TREE-STRUCTURE?

- In database programs, the data is too large to fit in memory; therefore, it is stored on secondary storage (disks or tapes).
- Disk access is very expensive; the disk I/O operation takes milliseconds while CPU processes data on the order of nanoseconds, one million times faster.

- When dealing with external storage the disk accesses dominate the running time.

Balanced binary search trees (AVL & Red-Black) have good performance if the entire data can fit in the main memory.

- These trees are not optimized for external storage and require many disk accesses, thus give poor performance for very large data.

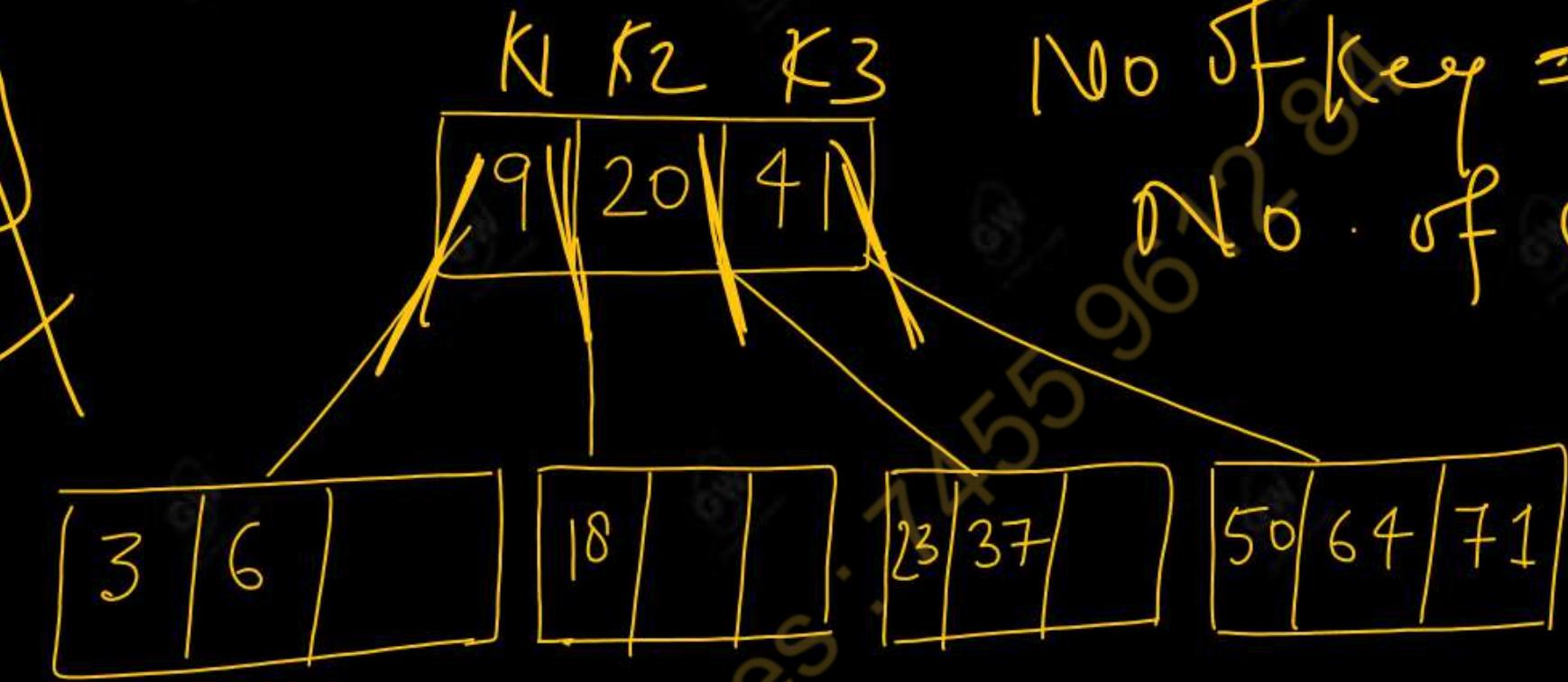
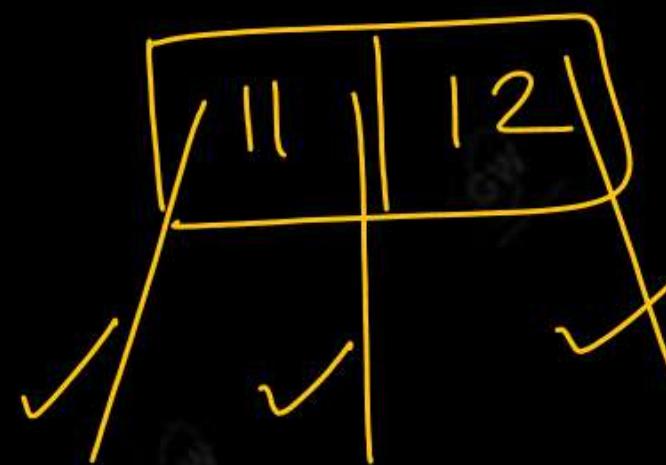
- To reduce the tree accesses –

- (i) Reduce tree height by increasing the number of children of a node.
- (ii) Store multiple records in a block on the disk.

- To achieve above mentioned goals we use multiway (m-way) search tree.

- Consider the following 4-way search tree in the next figure-





No of key = 3

No. of children = 4

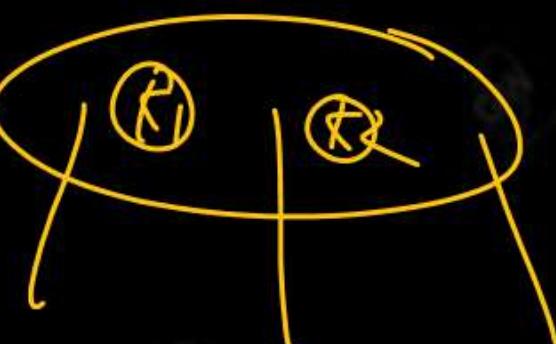


m
Two way search tree

M-WAY SEARCH TREES

- m-way search trees are generalized versions of binary search trees.
- The goals of m-way search tree is to minimize the accesses while retrieving a key from a file.
- However, an m-way search operation tree of height h calls for $O(h)$ number of accesses for an insert/delete/retrieval.
- Therefore there arises the need to maintain balanced m-way search trees.
- B trees are balanced m-way search trees.
- Trees having $(m-1)$ keys and m children are called m-way search trees.
- A binary tree is 2-way tree.
- It means that it has $m-1=2-1=1$ key (here $m = 2$) in every node and it can have maximum of two children.

$$m = 3$$

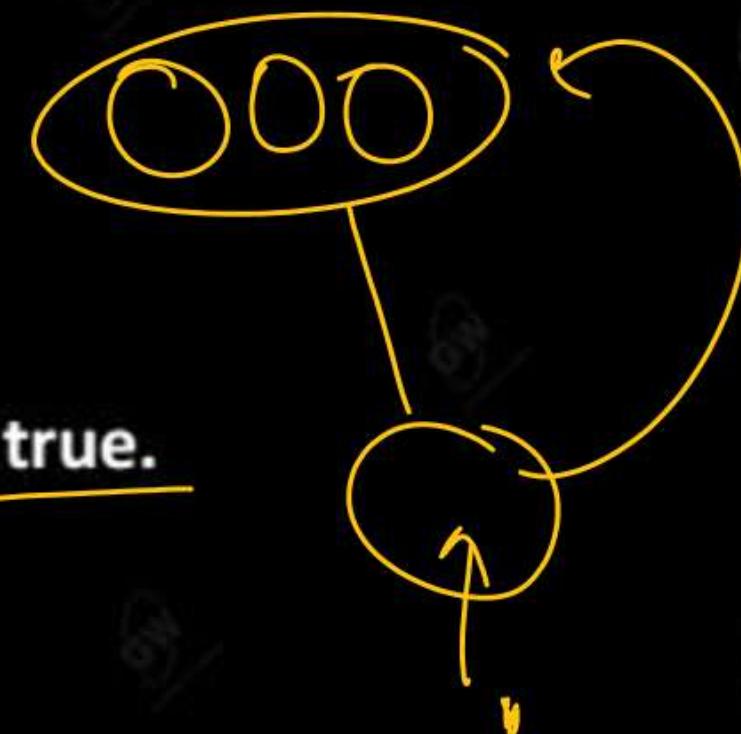


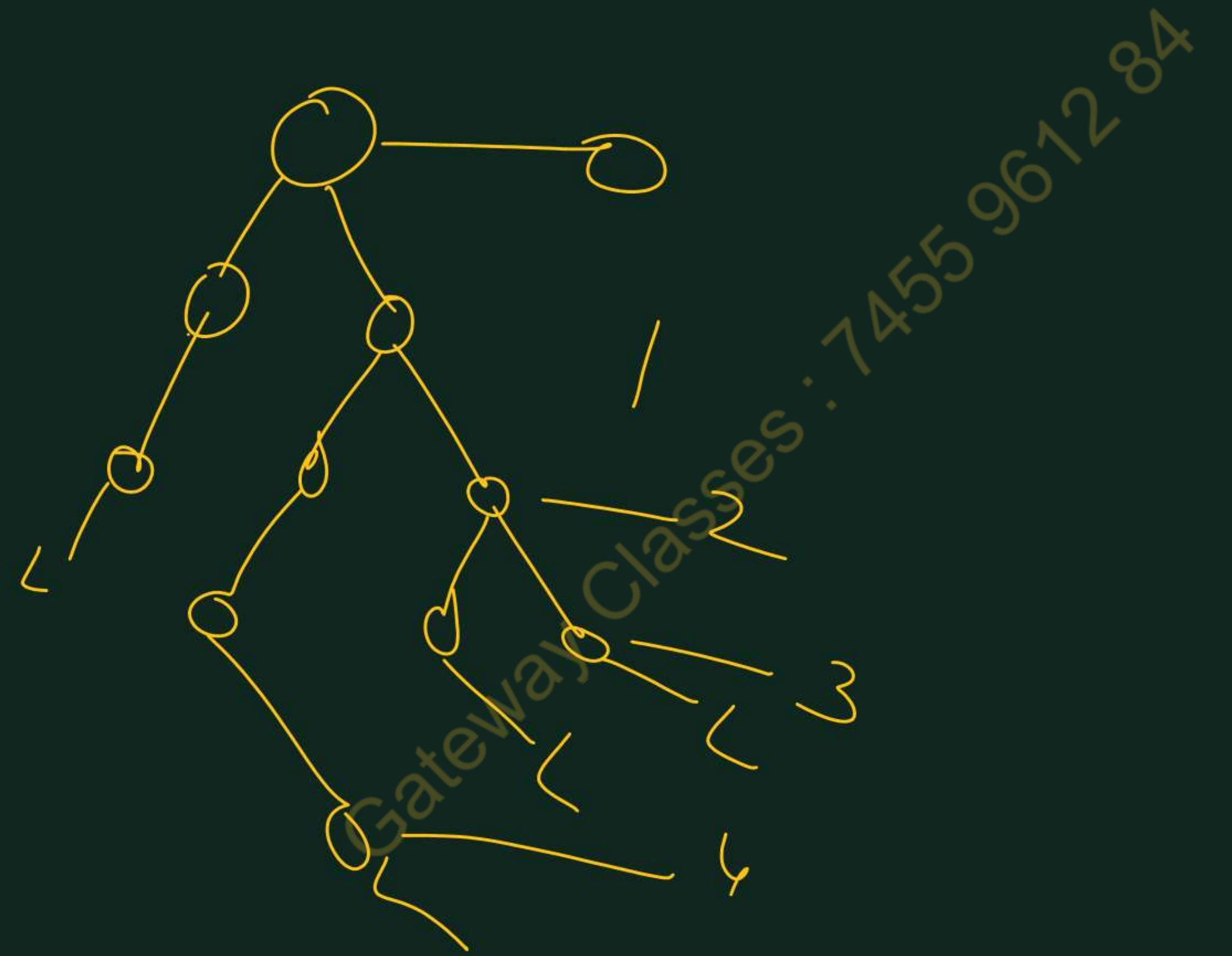
- A binary tree is also called an m-way tree of order 2.
- Similarly an m-way tree of order 3 is a tree in which key values could be either 1 or 2.

m
 $m - 1 = 3 - 1 = 2$
Children = 3.

The B-tree of order n can be defined as-

1. A B-tree is a balanced m-way tree.
2. A B-tree is also known as balanced sorted tree.
3. It finds its use in external sorting.
4. It is not a binary tree.
5. To reduce disk accesses, several conditions of the three must be true.
 - (i) the height of the tree must be kept to a minimum.
 - (ii) There must be no empty sub-trees above the leaves of the tree.
 - (iii) The leaves of the tree must all be on the same level; and
 - (iv) All nodes except the leaves must have at least some minimum number of children.





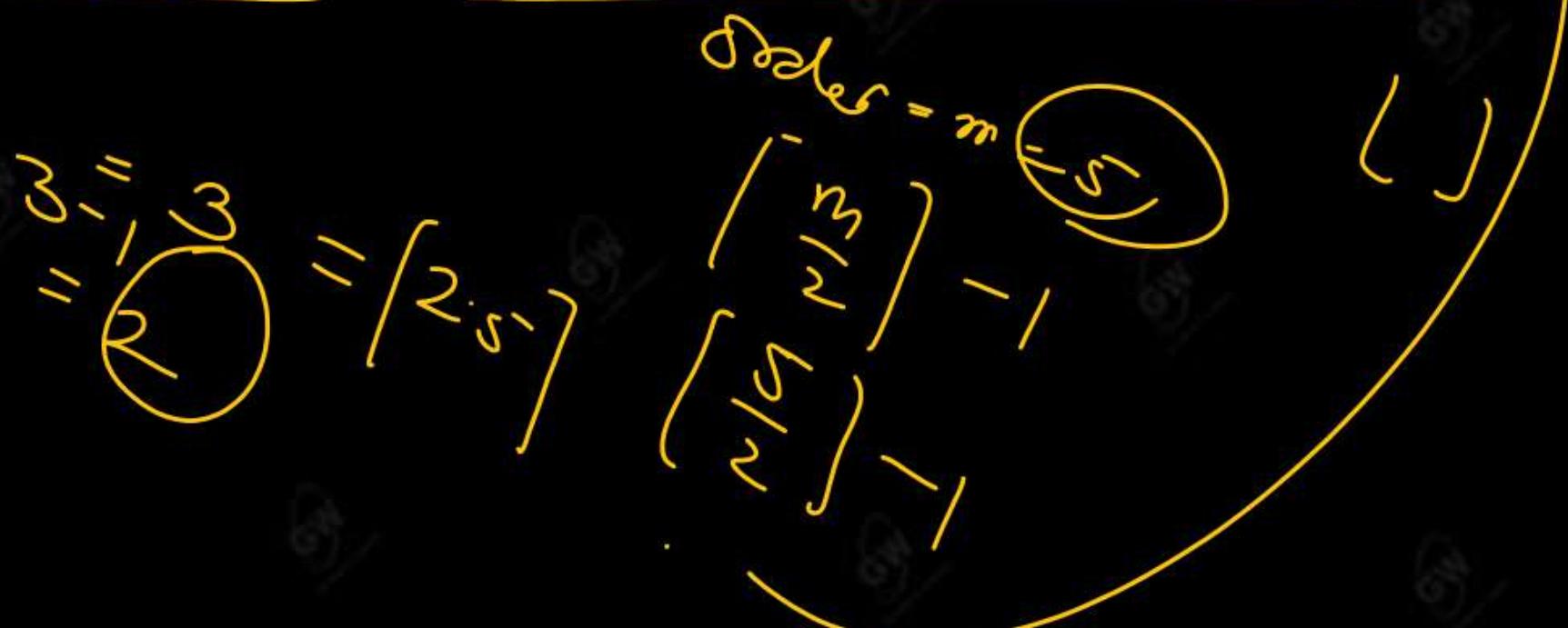
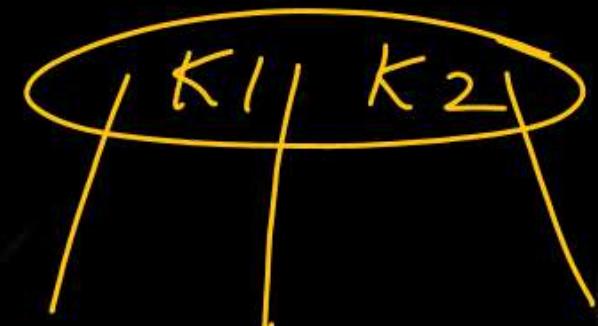
Gateway Classes: 7455 9612 84

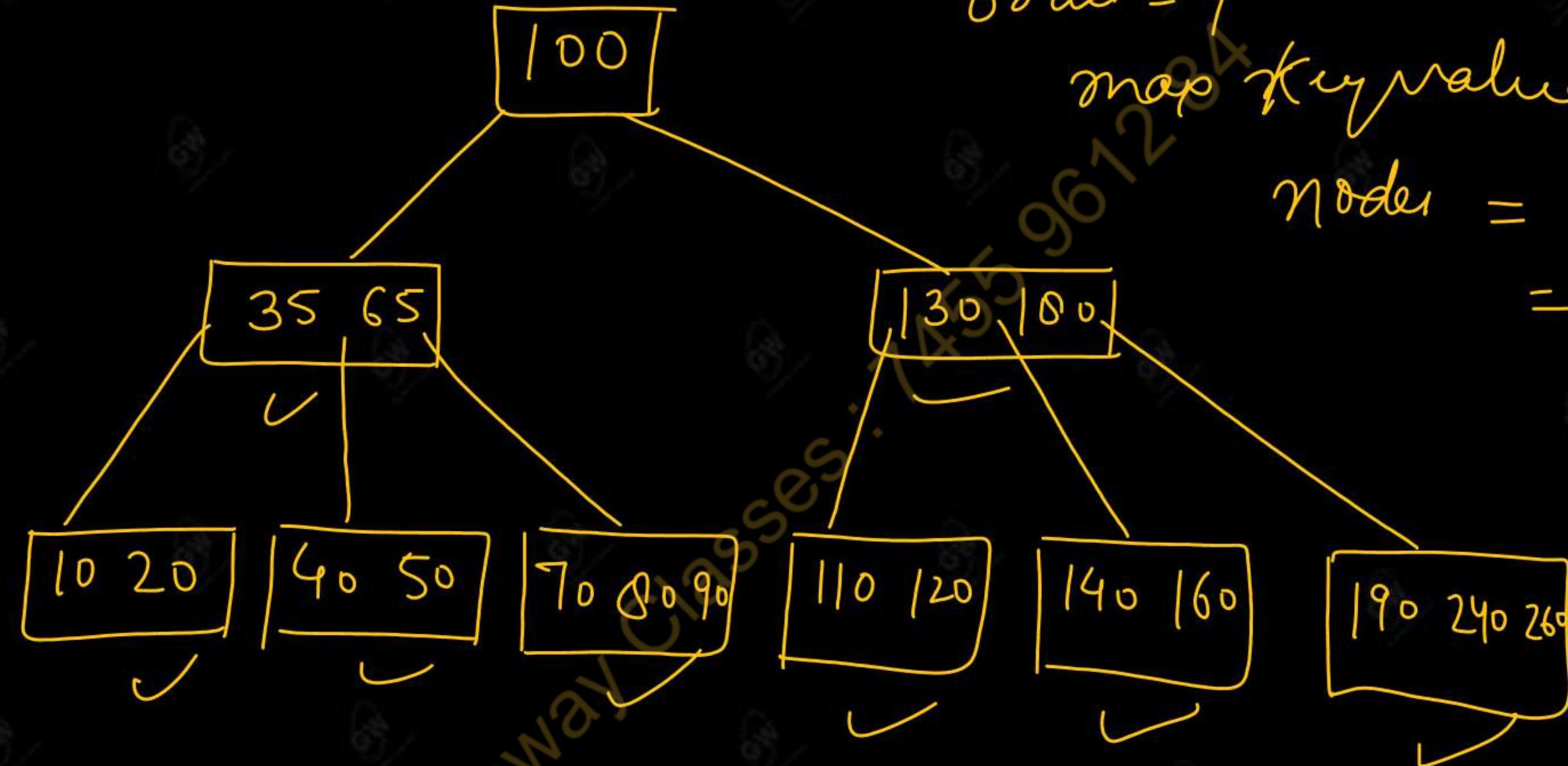
B tree has the following property:

1. All leaf nodes will be at same level.
2. Every node has maximum m children where m is order of b tree.
3. Every node has maximum $m-1$ keys.
4. Every node has minimum; root node = 1 key and all other node = $\text{Ceil}(m/2) - 1$ key. For example order = 5 the except root node all other node must have minimum 2 keys.
5. Keys in non leaf node will divide the left and right sub tree where value of left subtree keys will be less and the value of the right sub tree keys will be more than that particular key.

Let us take a B-tree of order 4:

Order = $m = 3$





- Here we can see all leaf nodes are at same level.
- All non leaf nodes have no empty sub tree and they have keys 1 less than number of their children.

Algorithm: Insertion _B-Tree

- ❑ Insertion will be always in the leaf node only.
- ❑ The insertion of a key in a B-tree requires first traversal in B-Tree.
- ❑ Through traversal it will find that key to be inserted is already existing or node.
- ❑ Suppose key does not exist in tree then through traversal it will reach leaf node.
- ❑ Now we have two cases for inserting the key:
 1. Node is not full
 2. Node is already full
- ❑ If the leaf node in which the key is to be inserted is not full, then the insertion is done in the node. A node is said to be full if it contains a maximum of $m - 1$ keys, where m is the order of the B-Tree.

- If the node is full, then insert the key in order into the existing set of keys in the node,
split the node at its median into two nodes at the same level, pushing the median
element up by one level.
- It is to be noted that the split nodes are half full.
- Accommodate the median element in the parent node if it is not full.
- Otherwise repeat the same procedure and this may even call for rearrangement of the
keys in the root node or the formation of a new root itself.
- Thus since the leaf node are all at same level, the tree grows upward.



Example: Consider building a B tree of degree 4 that is balanced 4-way tree where each node can hold three data values and have four branches. Suppose it needs to contain the following values:

1, 5, 6, 2, 8, 11, 13, 18, 20, 7, 9

Solution :-

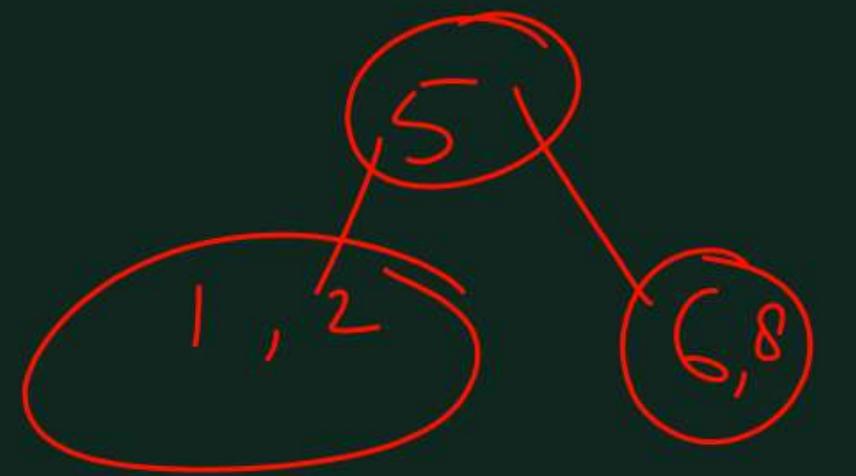
$$m = 4$$

$$\text{no of keys in a node} = \text{order} - 1 = 4 - 1 = 3$$

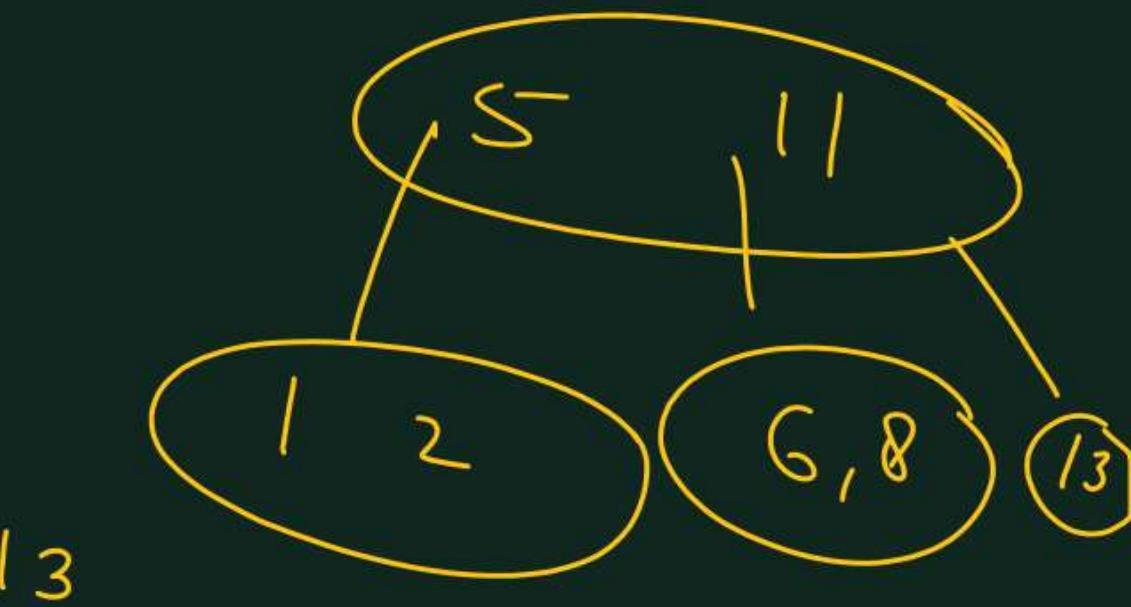


left biased tree 861,772

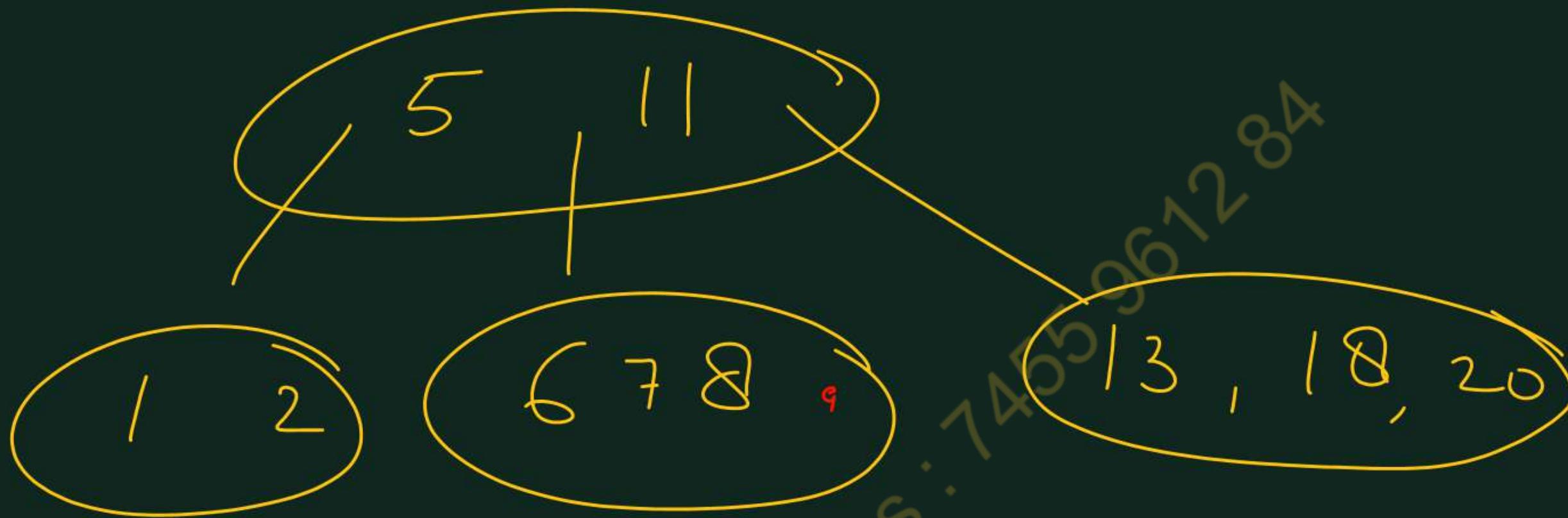




right 5 case



Gateway Classes : 7455 9612 84
right 5 case



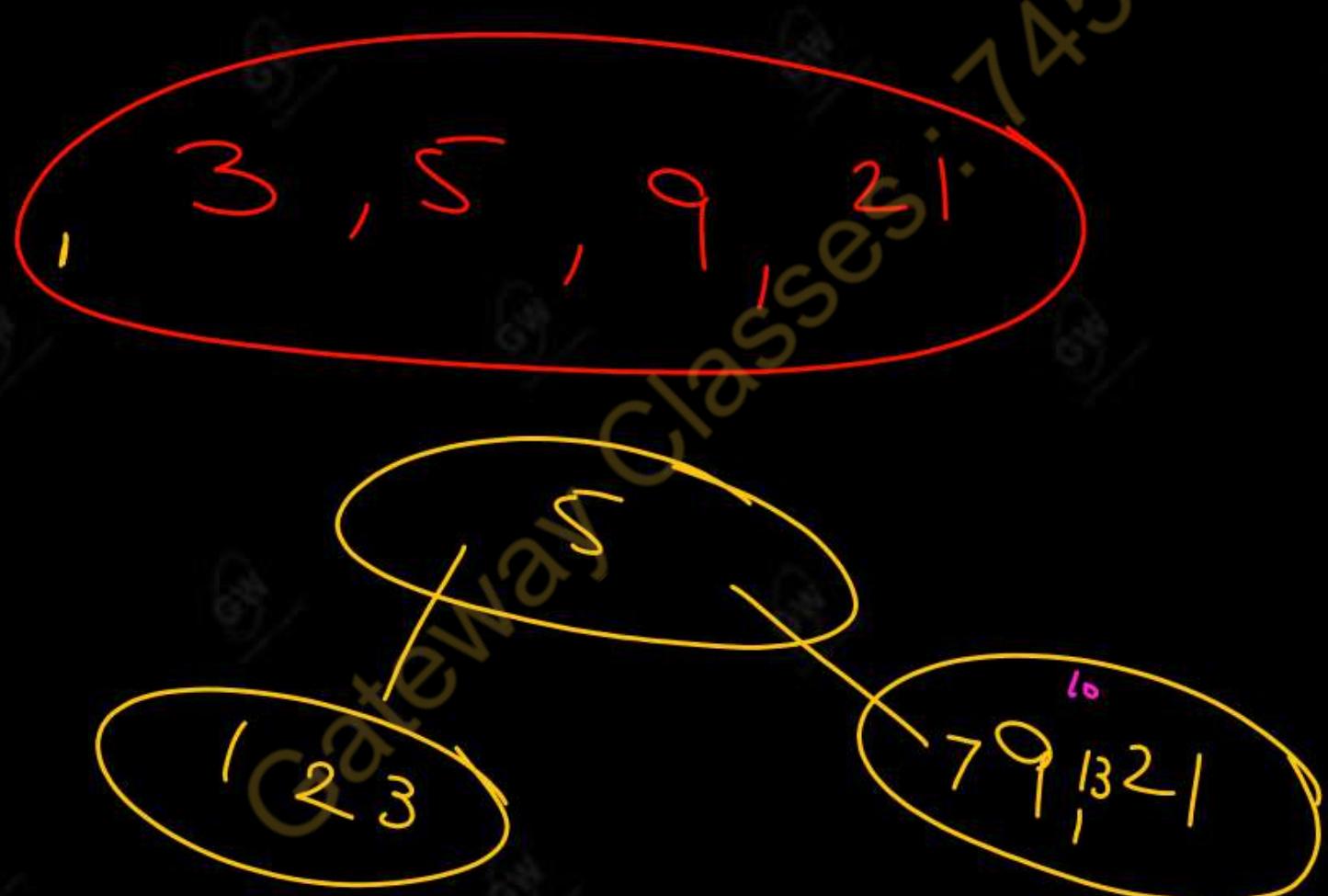
Example: Let us take a list of keys and create a B-tree of order 5.

5, 3, 21, 9, 1, 13, 2, 7, 10, 12, 4, 8

Solution :-

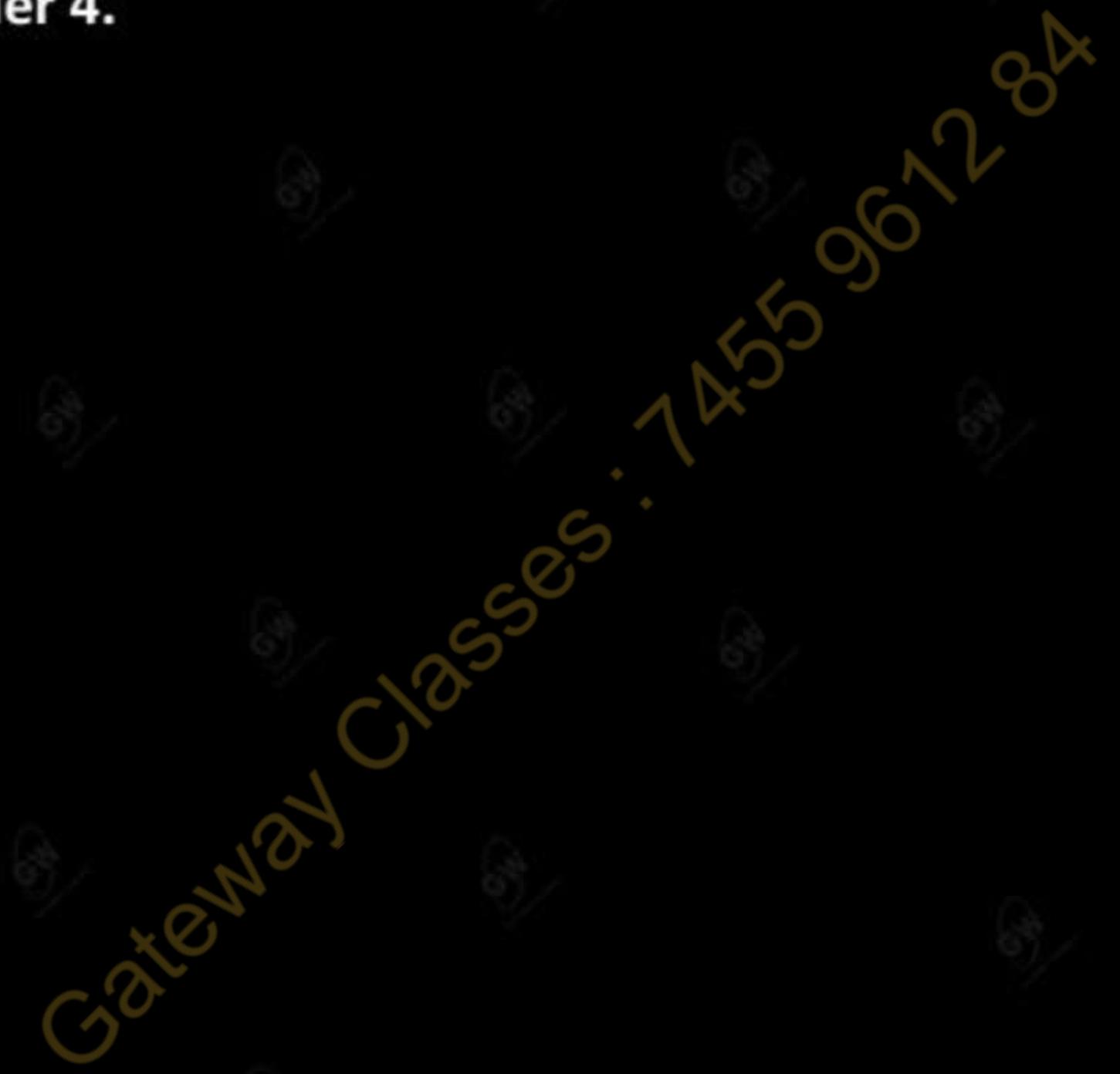
$$m=5$$

no of elements / key in a node = $5 - 1 = 4$



Example: Insert the following information 86, 23, 91, 4, 67, 18, 32, 54, 46, 45 into an empty B-tree with order 4.

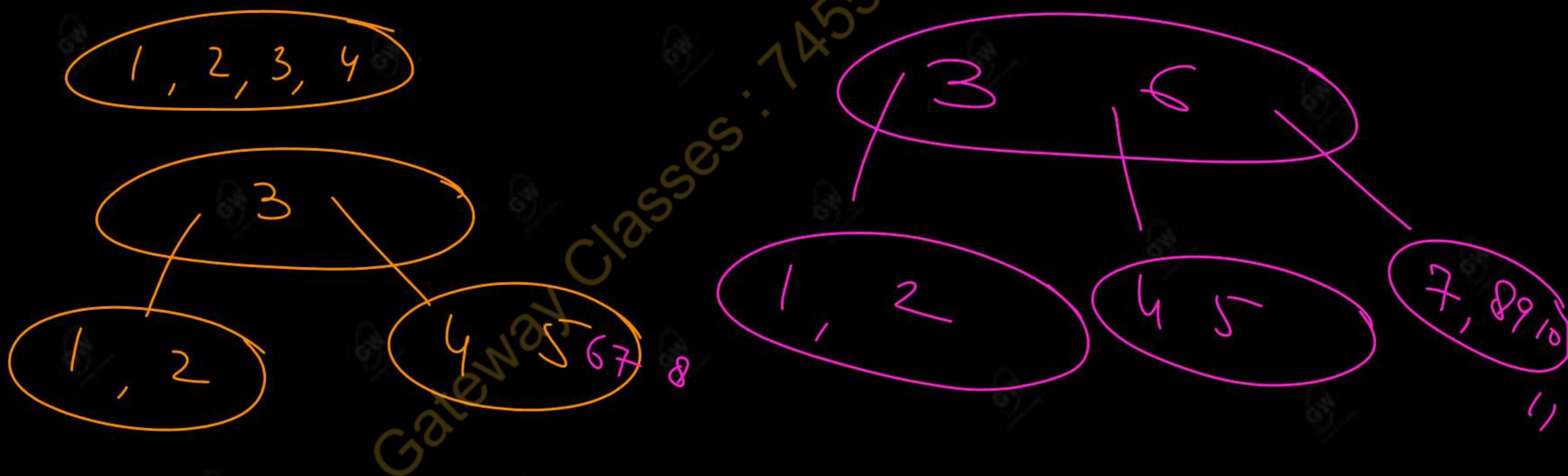
Solution:-

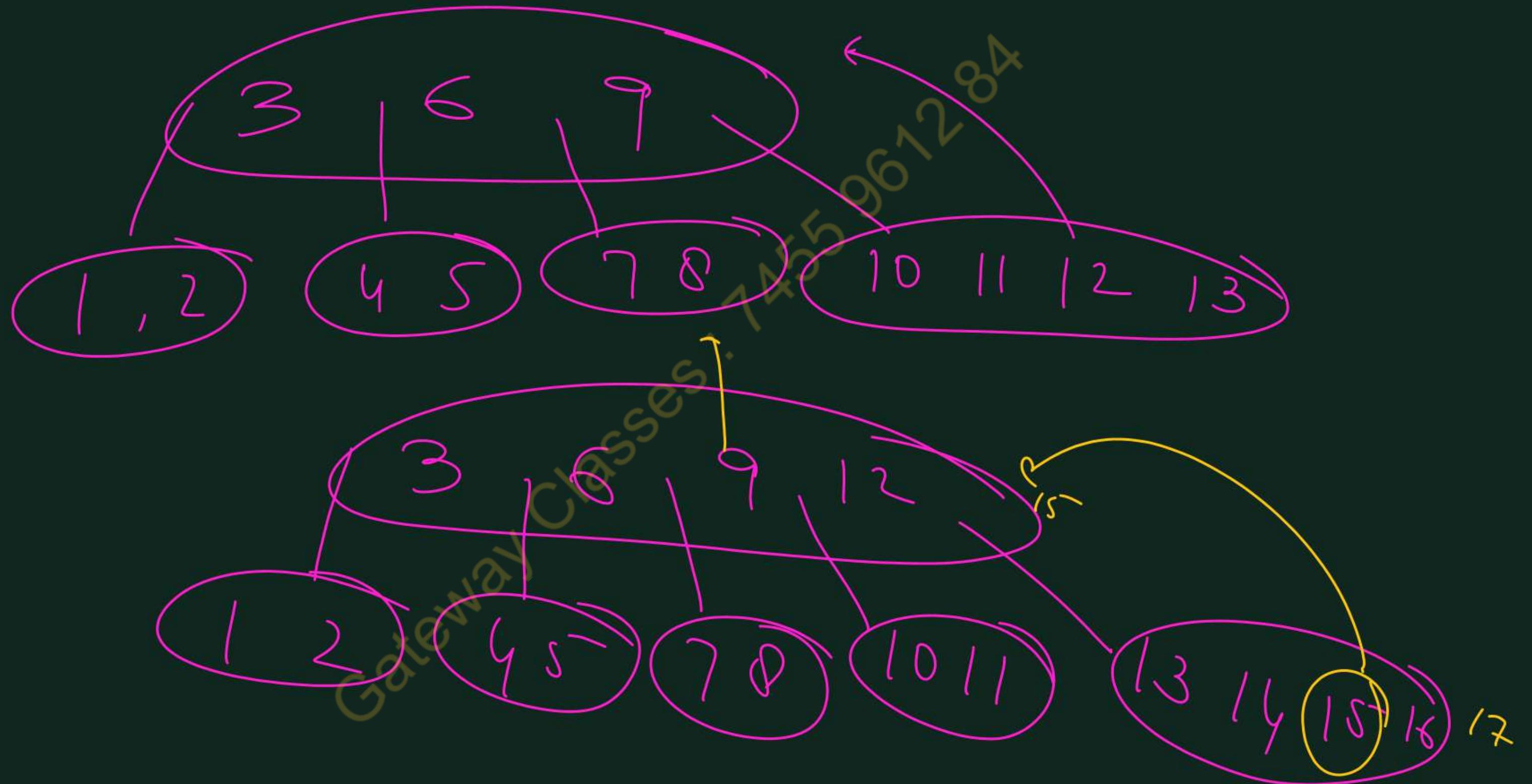


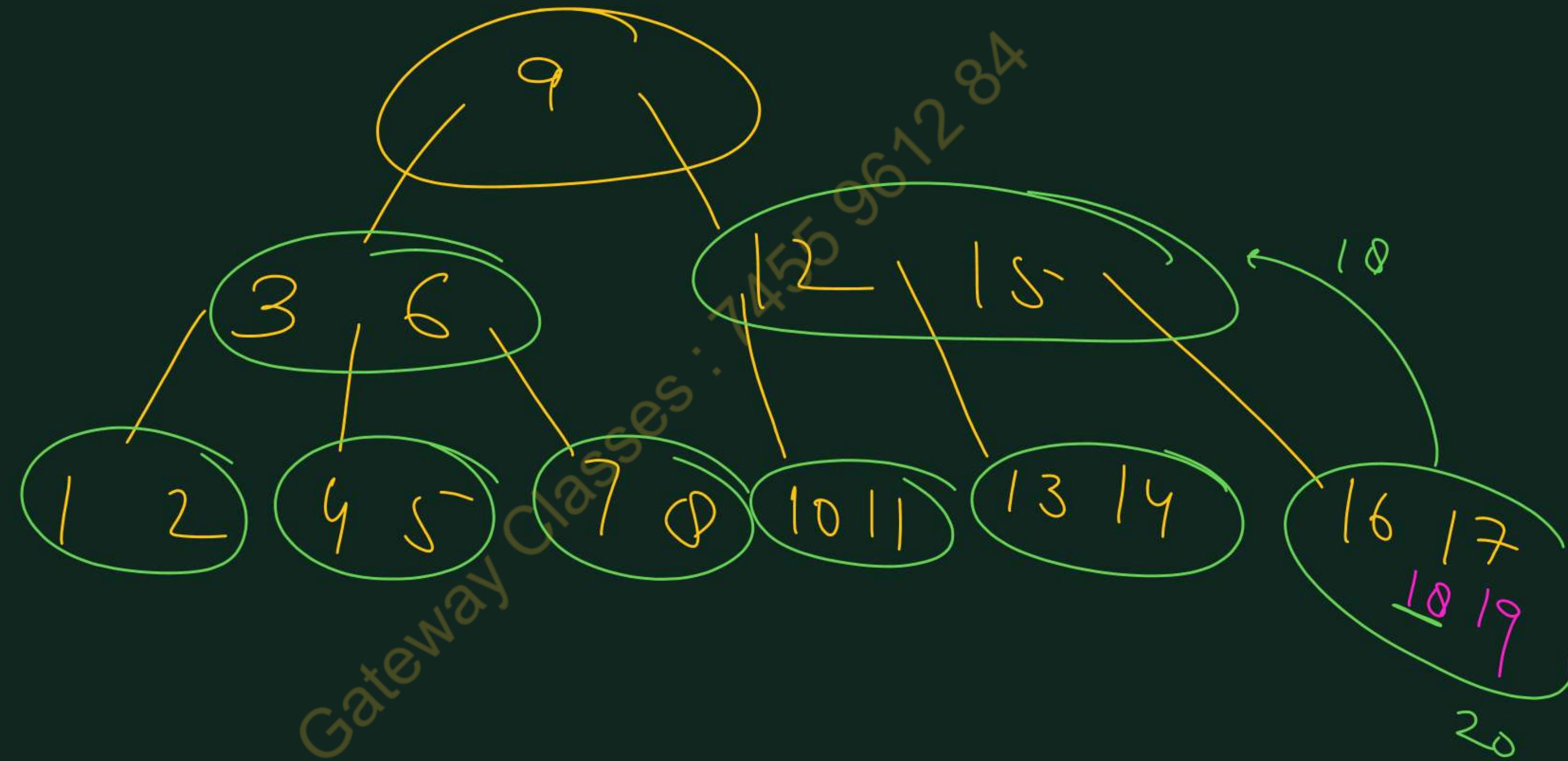
Example: Create a b tree of order 5 by inserting values from 1 to 20.**Solution:-**

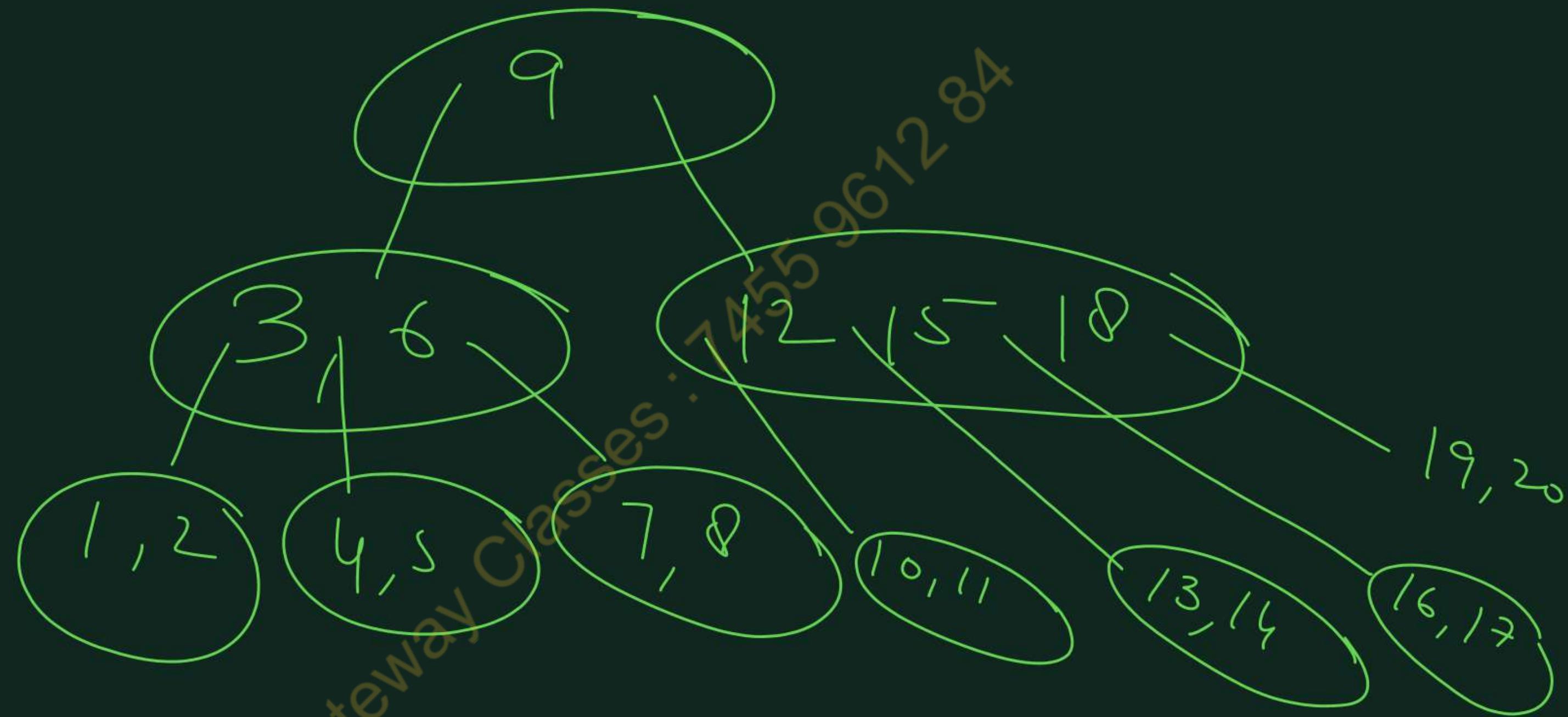
$$m=5 \quad (5-1=4)$$

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20









Ans

Unit - 4 : Lec-7

Today's Target

- B Tree (Deletion)
- Difference between B and B+ Tree
- AKTU PYQs

Q.1 Define B-tree. What do you understand by the order of B-tree?

Consider the following B tree:



Show the B tree after the following operations- insert 43, insert 50, delete 15. **2014-15, 10 marks**

Q.2 Compare and contrast the difference between B+ tree index files and B tree index file with an example. **2016-17, 10 marks**

Q.3 Construct a B-tree of order 5 created by inserting the following elements:

3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19

Also delete elements 6, 23 and 3 from the conducted tree.

2018-19, 7 marks

Q.4 (i) Insert the following keys into an initially empty B tree of order 5:

a, g, f, b, k, d, h, m, j, e, s, l, r, x, c, l, n, t, u, p

(ii) What will be the resultant B-tree after deleting keys j , t and d in sequence?

2021-22, 10 marks

Deletion in B-tree

□ Deletion of key also requires first traversal in B-tree and after reaching on particular node, two cases may occur-

- (i) Node is leaf node (Leaf node, External node)
- (ii) Node is non leaf node (Internal node)

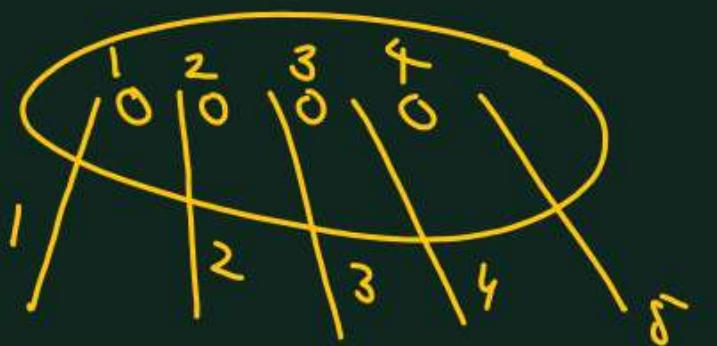
□ For the first case suppose node has more than minimum number of keys then it can be easily deleted. But suppose it has only minimum number of keys then first we will see the number of keys in adjacent leaf node if it has more than minimum number of keys then first ^{max} key of the adjacent node will go to the parent node and the key in parent node which partitioning will be combined together in one node. Suppose now parent has only less than the minimum number of keys then the same thing will be repeated until it will get the node which has more than the minimum number of keys.

Order = m (5)

(i) a node will contain max elements / key =

(ii) max no. of children for
a node = Order

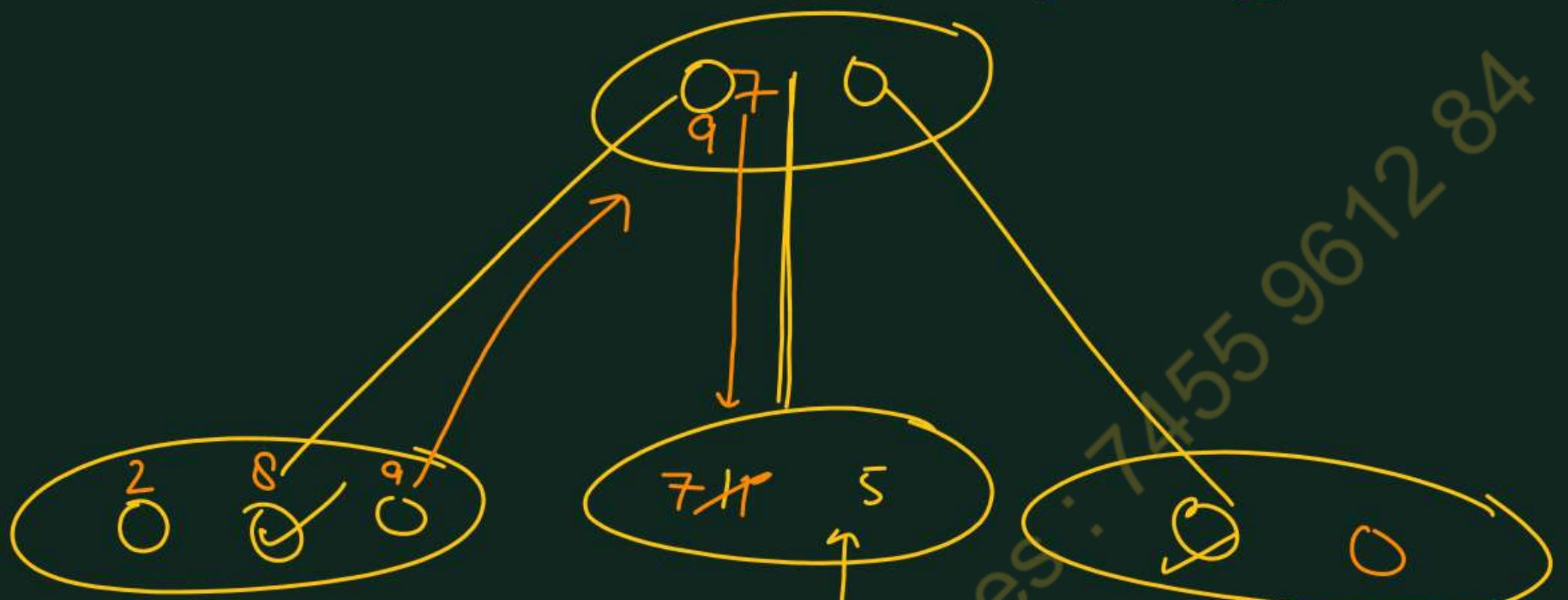
$$\begin{aligned} \text{Order - 1} &= m - 1 \\ &= 5 - 1 \\ &= 4 \end{aligned}$$



(ii) B-Tree minimum number of elements [keys] in node

$$= \left[\frac{\text{order}}{2} \right] - 1$$
$$= \left[\frac{5}{2} \right] - 1 = \frac{2 \cdot 5}{3 - 1} - 1 = 2$$

$$0:m = 5$$
$$m. = 2$$



Gateway Classes: 1455 9612 84
Delete

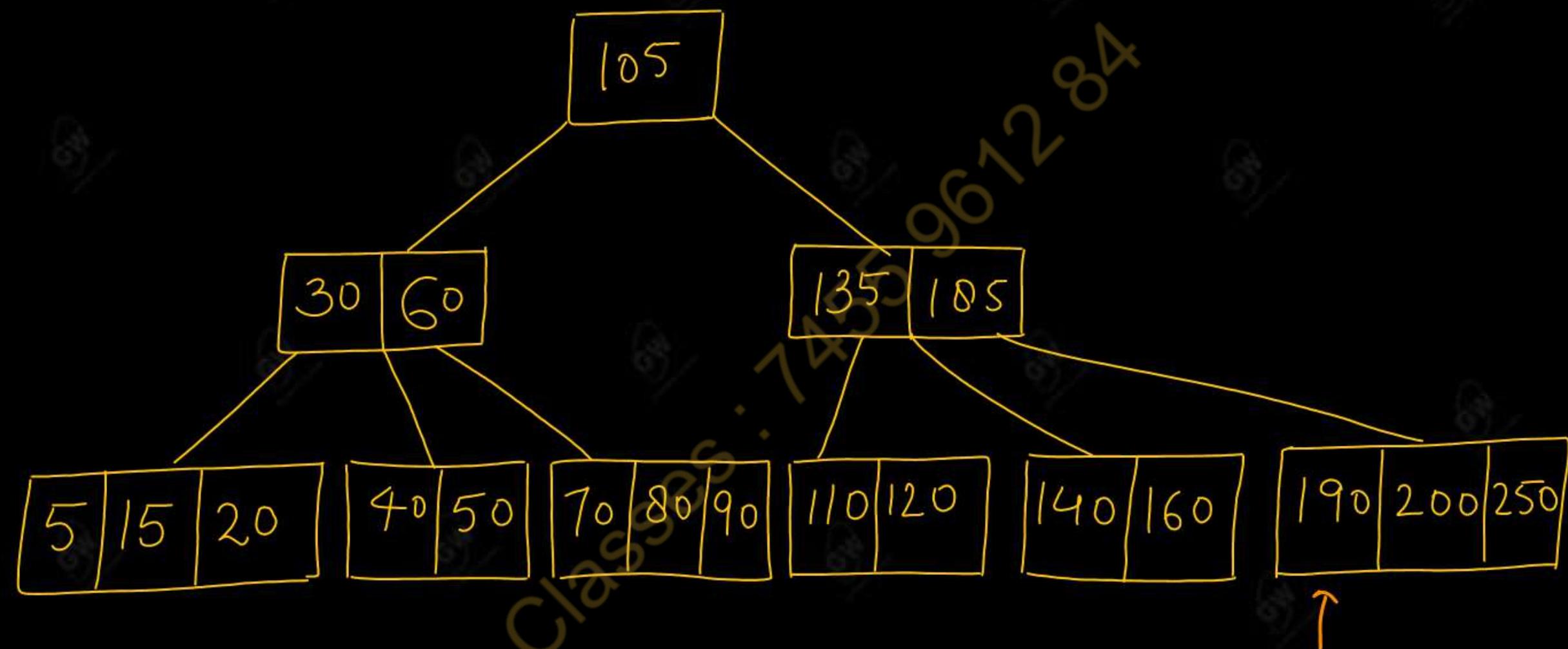
□ For the second case key will be deleted and its predecessor or successor key will come on its place. Suppose both nodes predecessor and successor key have minimum number of keys then the nodes of predecessor and successor keys will be combined.

□ Thus while removing a key from a leaf node, if the node contains more than the minimum number of keys, then key can be easily removed. If the leaf node contains just the minimum number of elements, then borrow for an element from either the left sibling node or right sibling node to fill the vacancy to make there minimum number of nodes. If the left sibling node has more than the minimum number of keys, pull the largest key up into the parent node and move down the intervening entry from the parent node to the leaf node where the key is to be deleted. Otherwise, pull the smallest key of the right sibling node to the parent node and move down the intervening parent element to the leaf node.

- If both the sibling nodes have only minimum number of entries, then create a new leaf node out of the two leaf nodes and the intervening element of the parent node, ensuring that the total number does not exceed the maximum limit for a node. If while borrowing the intervening element from the parent node, it leaves the number of keys in the parent node to be below the minimum number, then we propagate the process upwards ultimately resulting in a reduction of height of the B-tree.

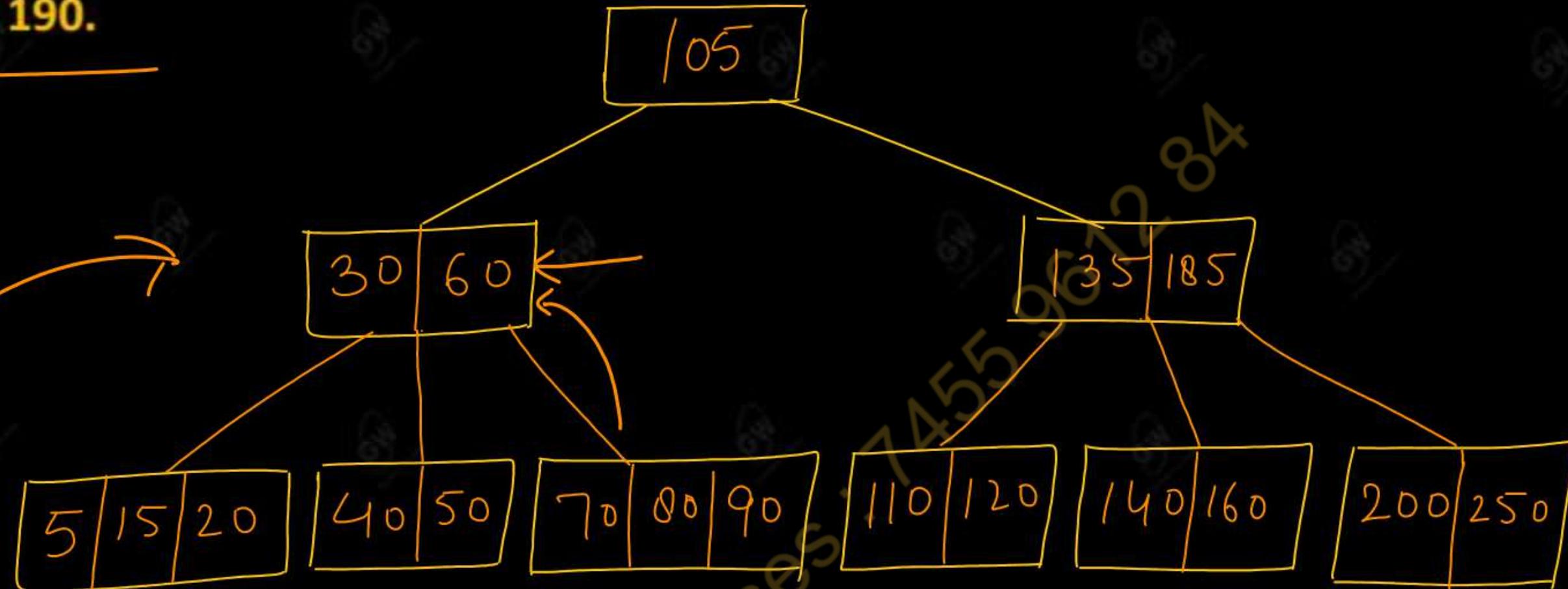
Gateway Classes

Example:- Let us take a B-tree of order 5.



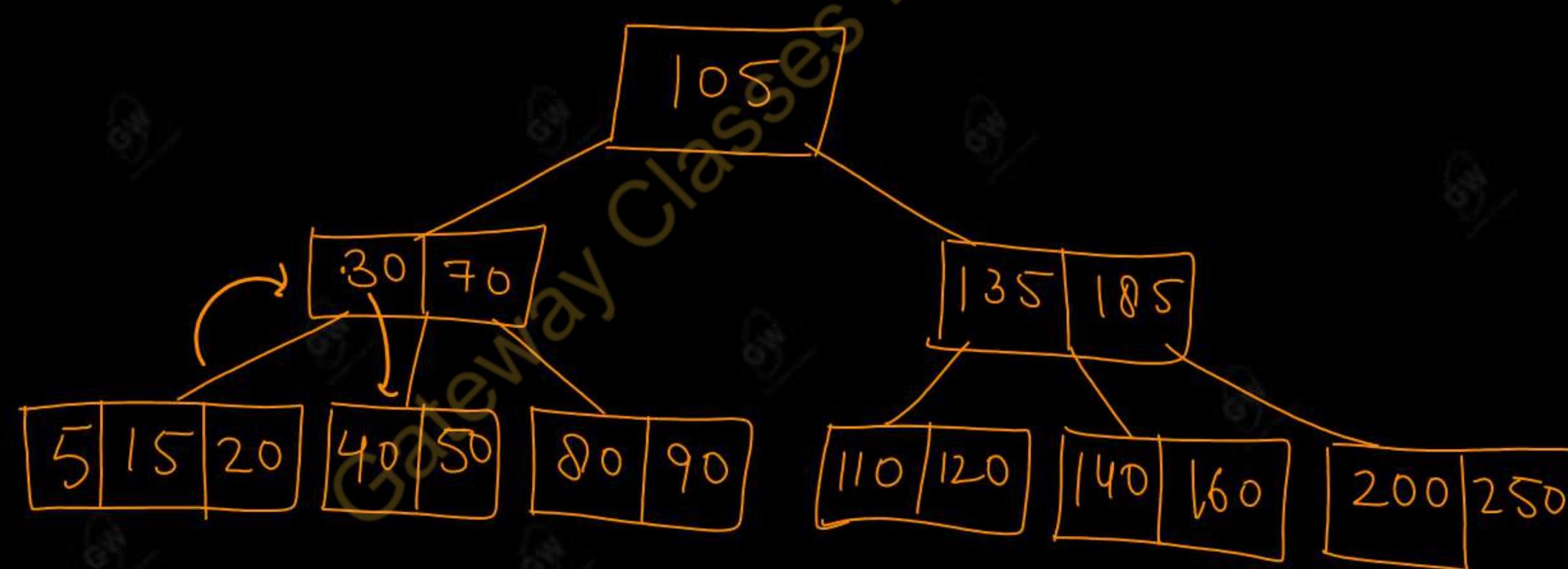
- Order 5 means node will have maximum number of keys = $4 \left(5 - 1 \right)$
- And node will have minimum number of keys = $\left[\frac{5}{2} \right] - 1 = 2 \quad \{ \text{except the root} \}$

Delete 190.

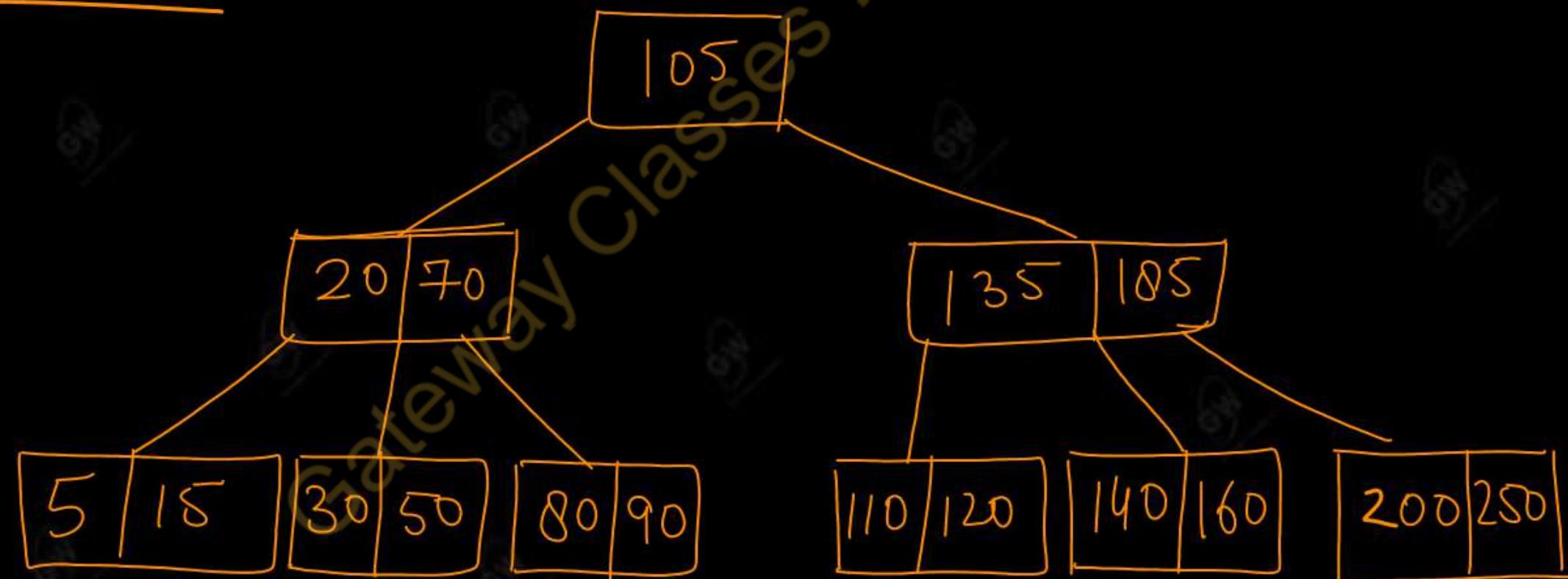


- Here 190 is in leaf node and after deleting 190 the node will have minimum number of keys so it can be deleted easily from the leaf node.
- After deletion of 190, the B tree will be as follows-

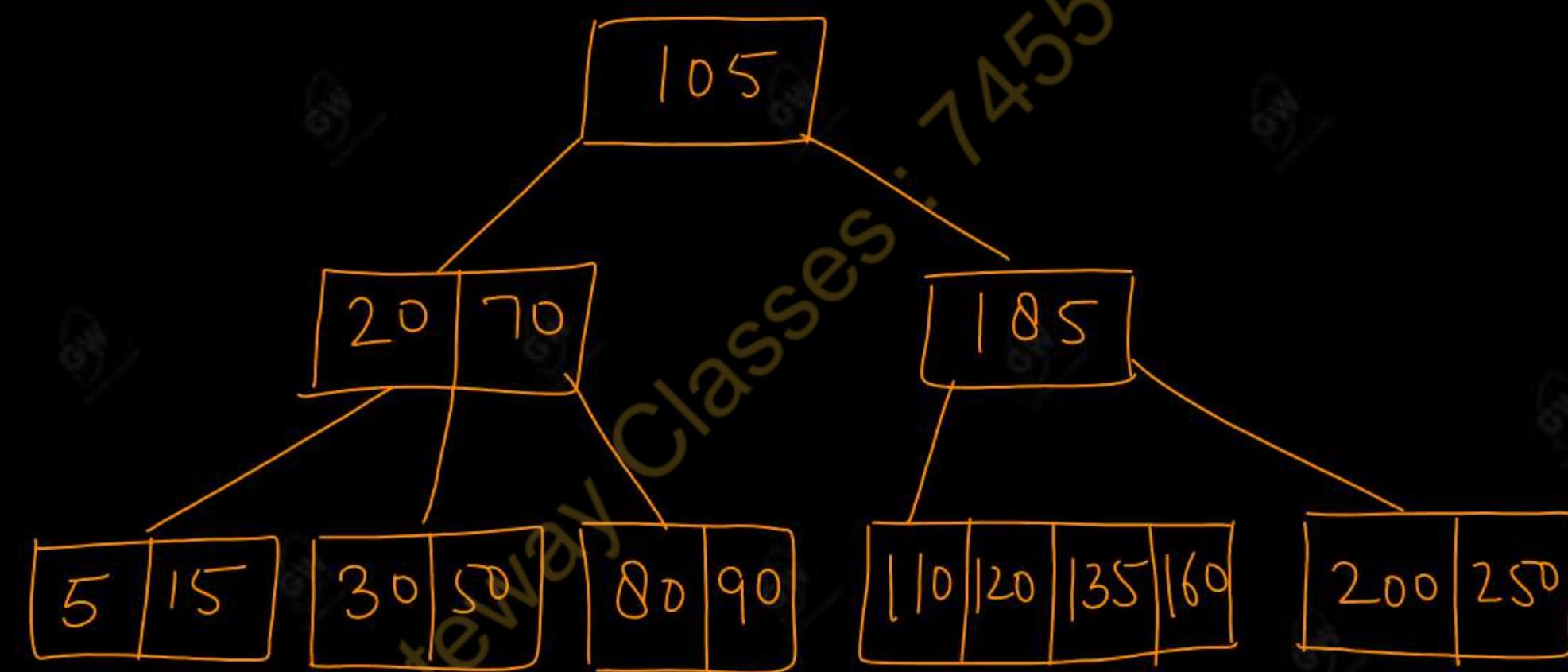
Delete 60 - Here 60 is non leaf node. This node is having minimum number of keys i.e. 2 and after deletion it will violate the rule of B tree so we have to do something to maintain minimum number of keys in this node. So first 60 will be deleted from the node and then the minimum element of right node will come in that node, that element is 70. After moving 70 to upward node, still it will maintain the minimum number of keys in a node. After this adjustment tree will be as follows:



Delete 40 - Here 40 is in leaf node. This node is having minimum number of keys i.e. 2 and after deletion it will violate the rule of B tree so we have to do something to maintain minimum number of keys in this node. So first 40 will be deleted from the leaf node and then the maximum key of the left side node will come in the parent node and the minimum key of parent node will come in leaf node. After this adjustment the B-tree will be as follows:



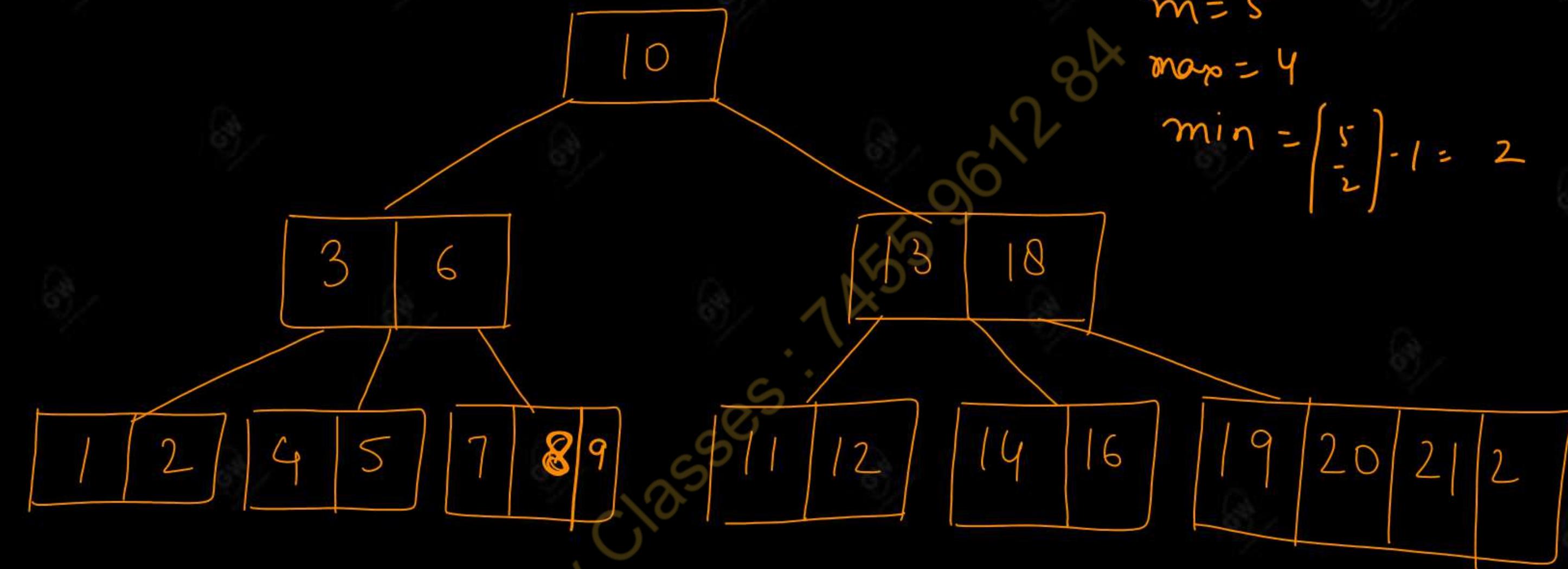
Delete 140 - Here 40 is in leaf node. This node is having minimum number of keys i.e. 2 and after deletion it will violate the rule of B tree so we have to do something to maintain minimum number of keys in this node. Its siblings and parent node is also have minimum number of keys. So we have to merge two nodes as shown in this tree.



But at least two keys must be in child node (except root node) so again we have to merge the nodes. This is shown in the following figure.



This is the final B-tree after performing all deletions.

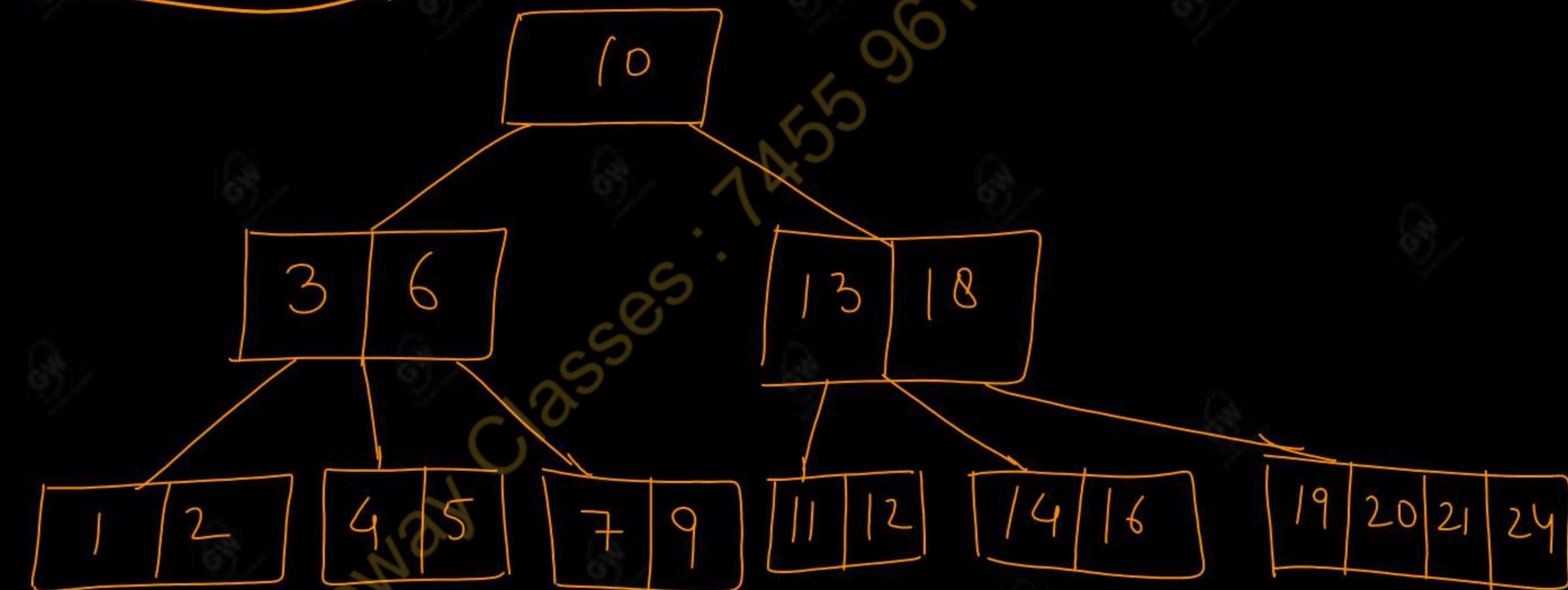
Example:- Consider the following B-tree of order 5:

$$\begin{aligned}m &= 5 \\ \max &= 4 \\ \min &= \left\lceil \frac{5}{2} \right\rceil - 1 = 2\end{aligned}$$

- Delete 8, 18, 16 and 4.

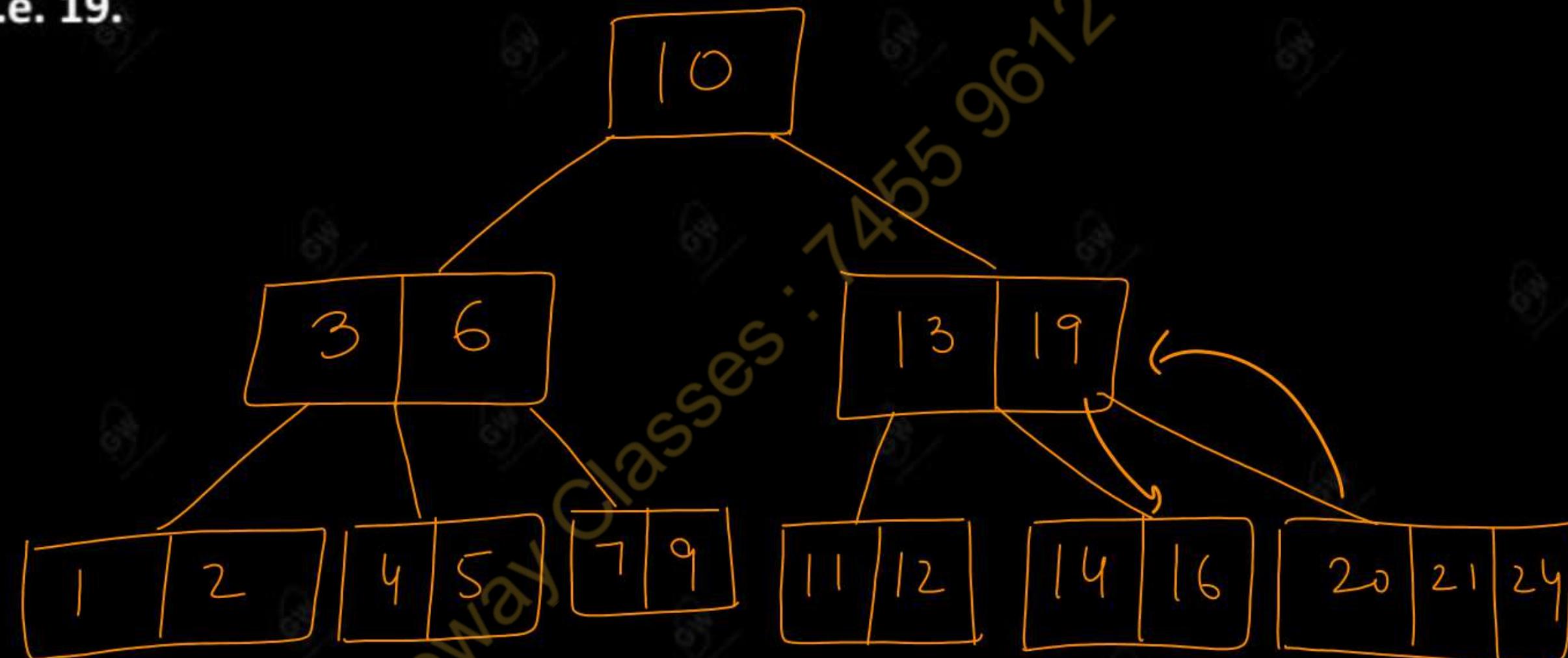
Delete 8:

- Deletion of 8 is from the leaf node with more than the minimum number of elements and hence leaves no problem.

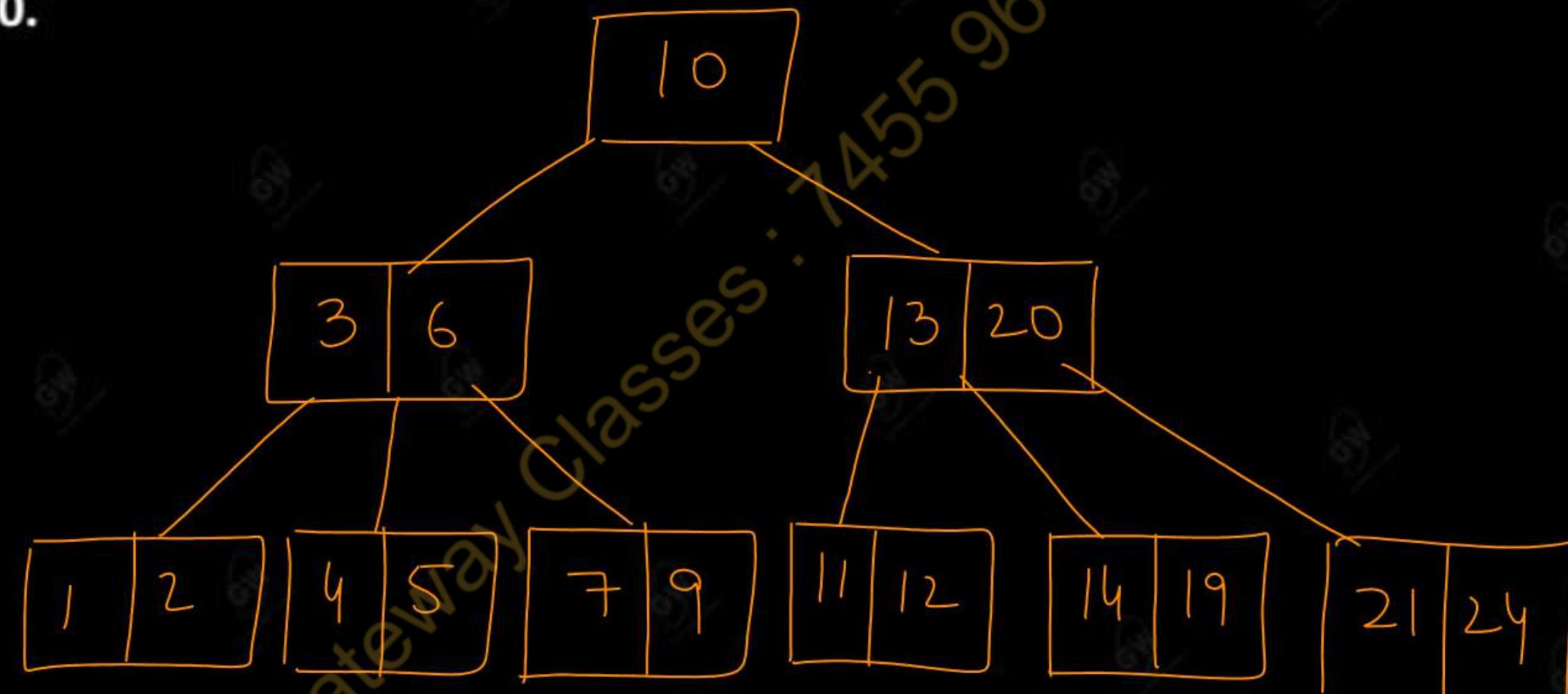


Delete 18:

- Deletion of 18 is from the non leaf node and therefore immediate successor is placed at 18, i.e. 19.

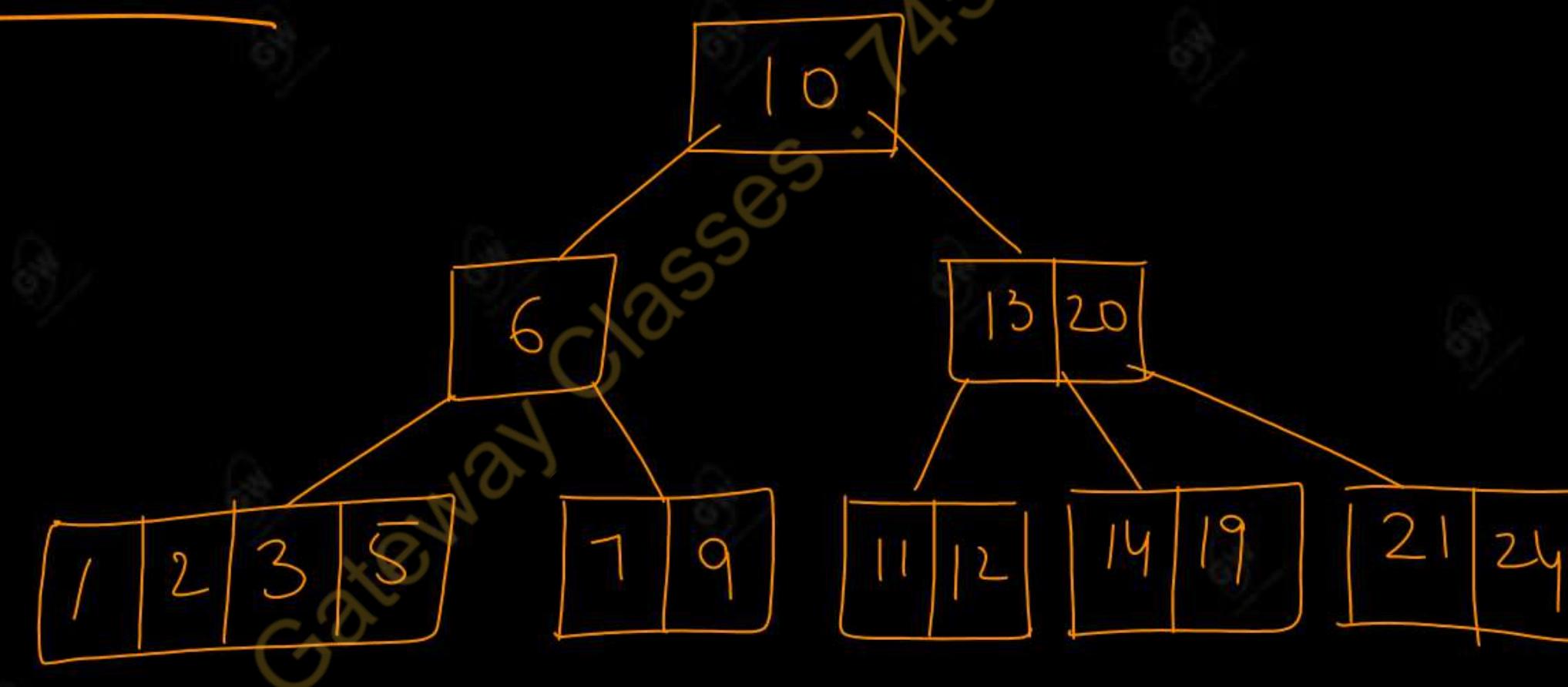


- Deletion of 16 leaves its nodes with too few elements.
- The element 19 from the parent node is therefore brought down and replaced by the element 20.

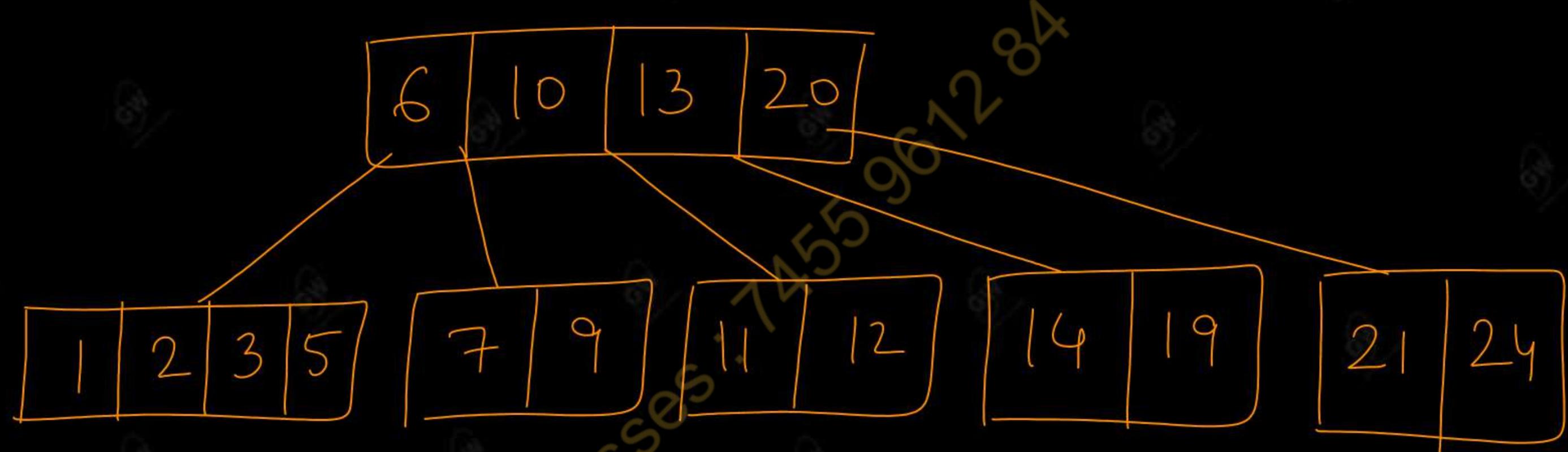


Delete 4:

- Deletion of 4 leaves the node with less than minimum number of elements and neither of its sibling nodes can spare an element.
- The node therefore is combined with one of the sibling and with the median element from the parent node.



- But at least two elements should be in the child of root so it will go in root node.



- It reduces the height of the tree also.

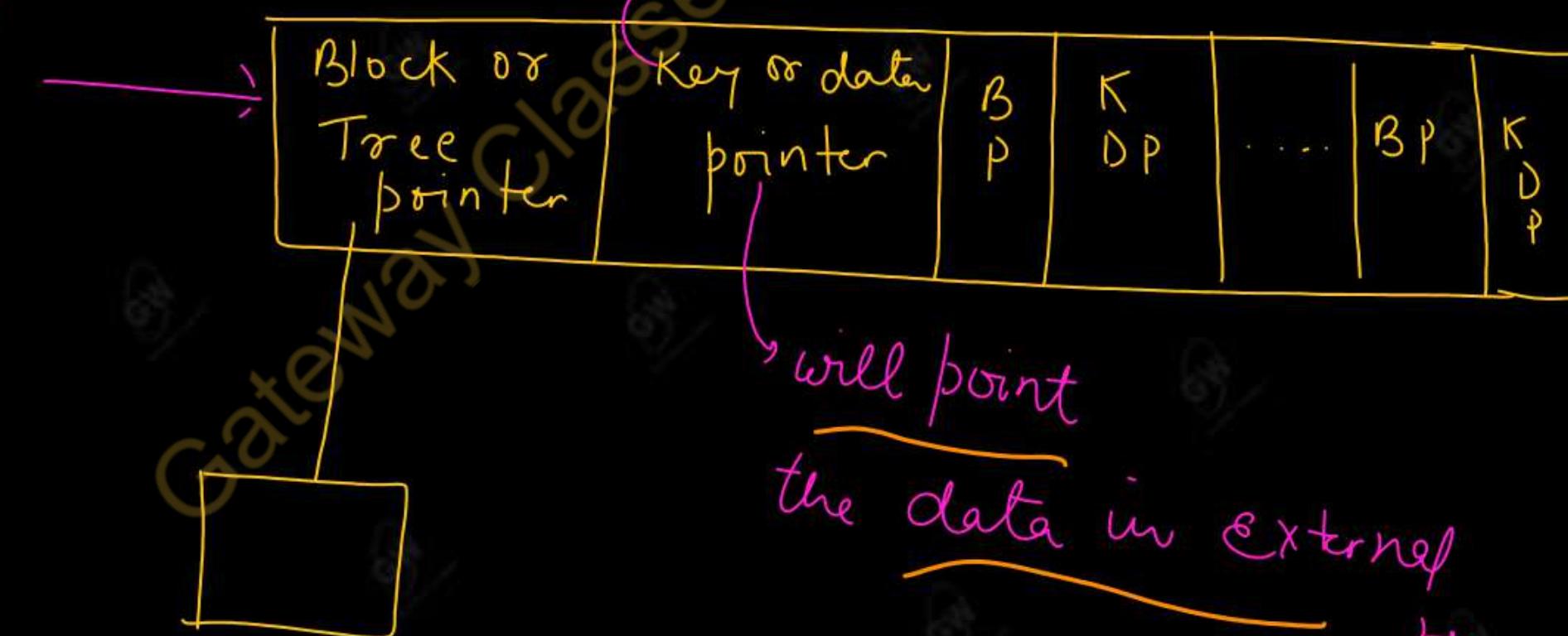
- The B-tree structure is the standard organization for indexes in a database system. There are several variations of the B-tree, most well known being the B-tree.
- The B-tree guarantees at least 50% storage utilization, that is, at any given time, the tree has each of its nodes at least 50% full.
- The B⁺-tree is a slightly different data structure, which in addition to indexed access, also allows sequential data processing and stores all data in the lowest level of the tree.
- One of the major drawbacks of the B-tree is the difficulty of traversing the keys sequentially.
- B⁺ tree retains the rapid random access property of the B-tree, while also allowing rapid sequential access.

Difference between B and B+ Tree:

Structure of B-Tree node :-

⇒ In B tree structure of a node is same for all i.e. for root node, internal and external node.

will present
children
node



- * Block or Tree pointer represent the children node.
- * Key or data pointer - Key means ⁸⁴ which is being searched, and data pointer will point the original data in second memory.

* What is the role of block pointer in leaf node as there will not be further any child?

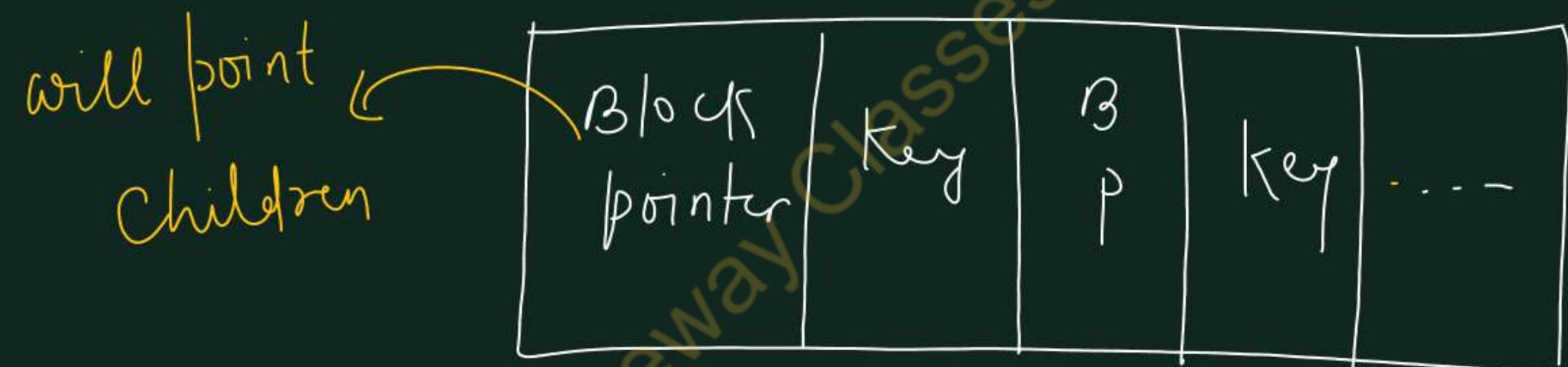
* The leaf node block pointer will contain

NULL

value.

Structure of B+ Tree Node :-

In B+ Tree the structure of internal node (including root node) is as follows -



STRUCTURE OF A NON LEAF NODE

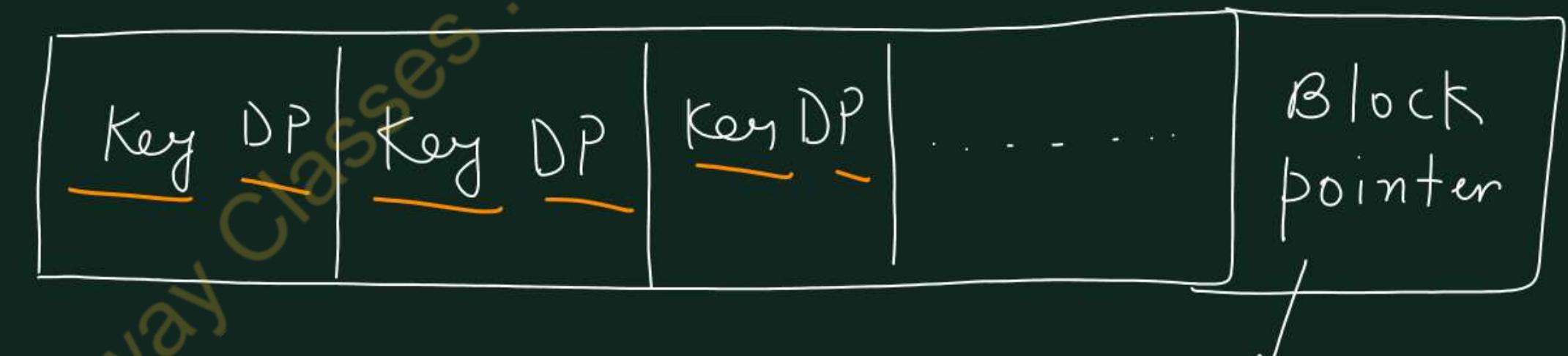
i.e. record pointer is not part of a node of B+ Tree. So more children can be accommodated in that space. So B+ Tree increases

breadth-wise { whereas B-tree increases in
depthwise }

* Where is record pointer in B+ Tree?

In B+ Tree, record pointer will be available with leaf node.

STRUCTURE OF LEAF NODE



will contain the address
of next block

B Tree

- 1) Data is stored in leaf
as well as in internal
node.
- 2) Searching is slower
because all nodes
having key values.

B+ Tree

- 1) Data is stored only in
leaf node.
- 2) Searching is faster.

B-Tree

- 3) No redundant key present.
- 4) leaf nodes not linked together.

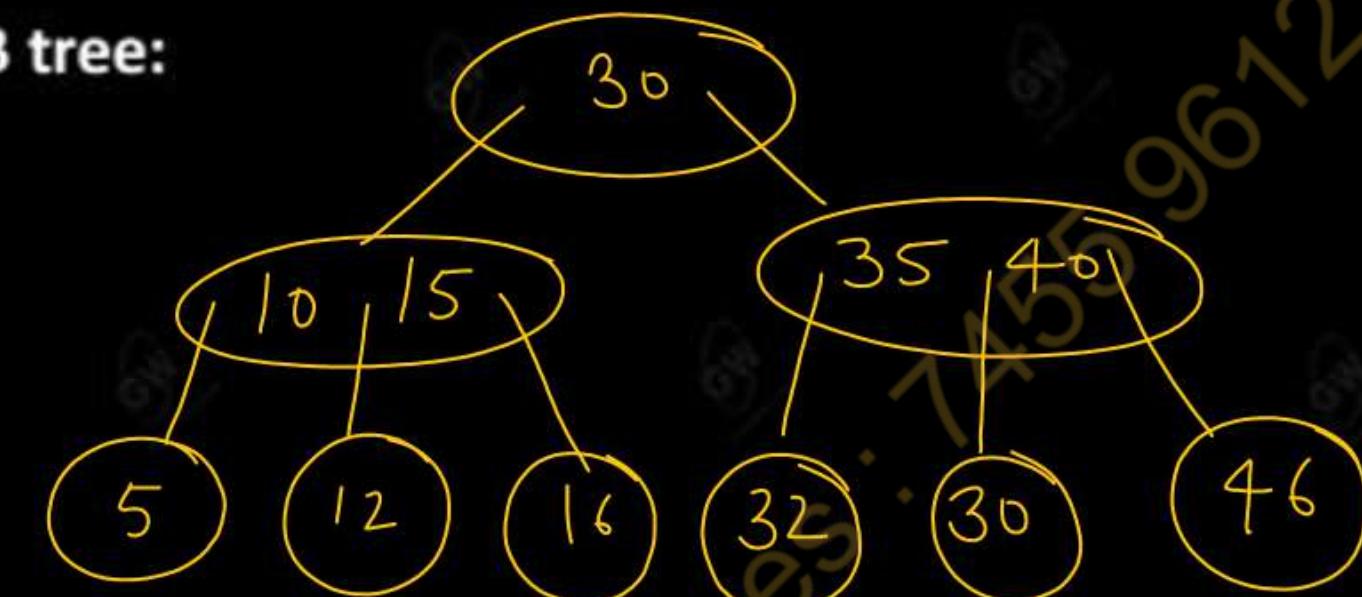
B+Tree

- 3) Redundant key may present.
- 4) leaf nodes are linked together by block pointer.

Gateway Classes: 1455961284

Q.1 Define B-tree. What do you understand by the order of B-tree?

Consider the following B tree:



Show the B tree after the following operations- insert 43, insert 50, delete 15. **2014-15, 10 marks**

Q.2 Compare and contrast the difference between B+ tree index files and B tree index file with an example. **2016-17, 10 marks**

Q.3 Construct a B-tree of order 5 created by inserting the following elements:

3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19

Also delete elements 6, 23 and 3 from the conducted tree.

2018-19, 7 marks

Q.4 (i) Insert the following keys into an initially empty B tree of order 5:

a, g, f, b, k, d, h, m, j, e, s, l, r, x, c, l, n, t, u, p

(ii) What will be the resultant B-tree after deleting keys j , t and d in sequence?

2021-22, 10 marks

Download **Gateway Classes** Application
From Google Play store
Link in Description

Next Lecture:- Unit – 5, Graph

Thank You