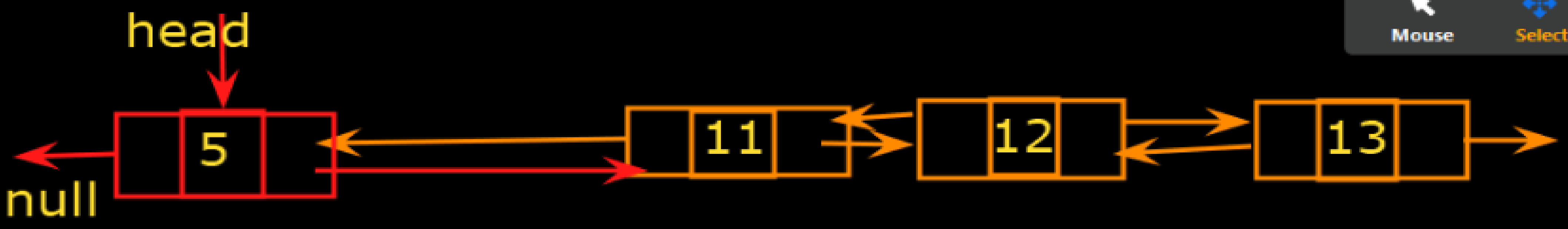
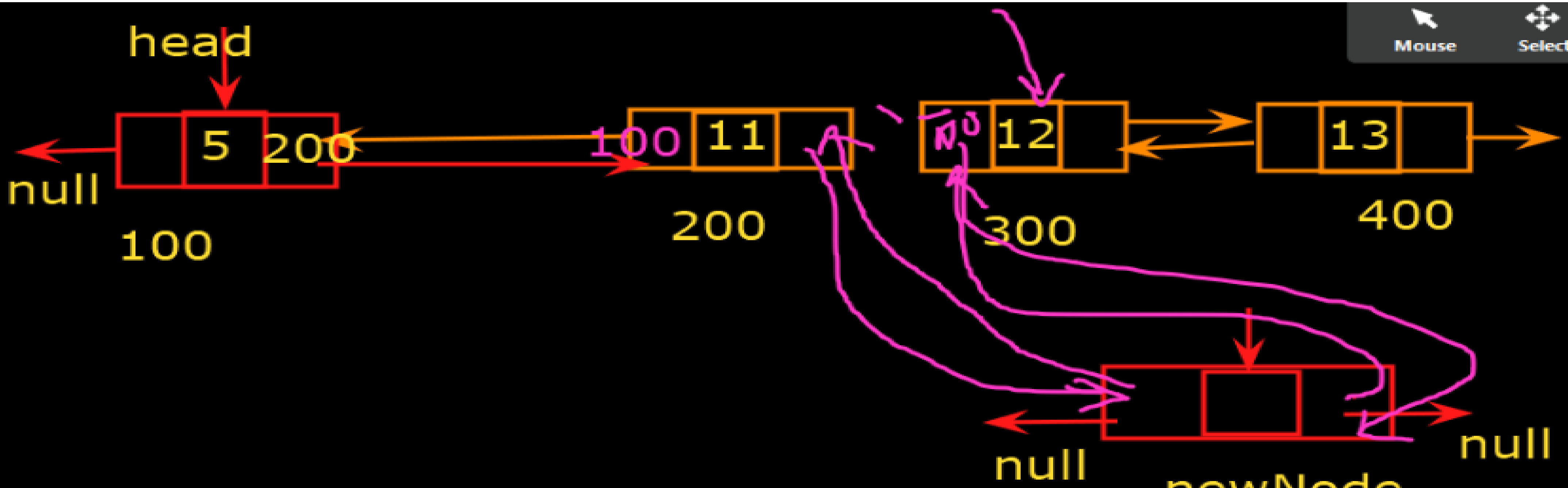
Algorithms & Data Structure

Kiran Waghmare

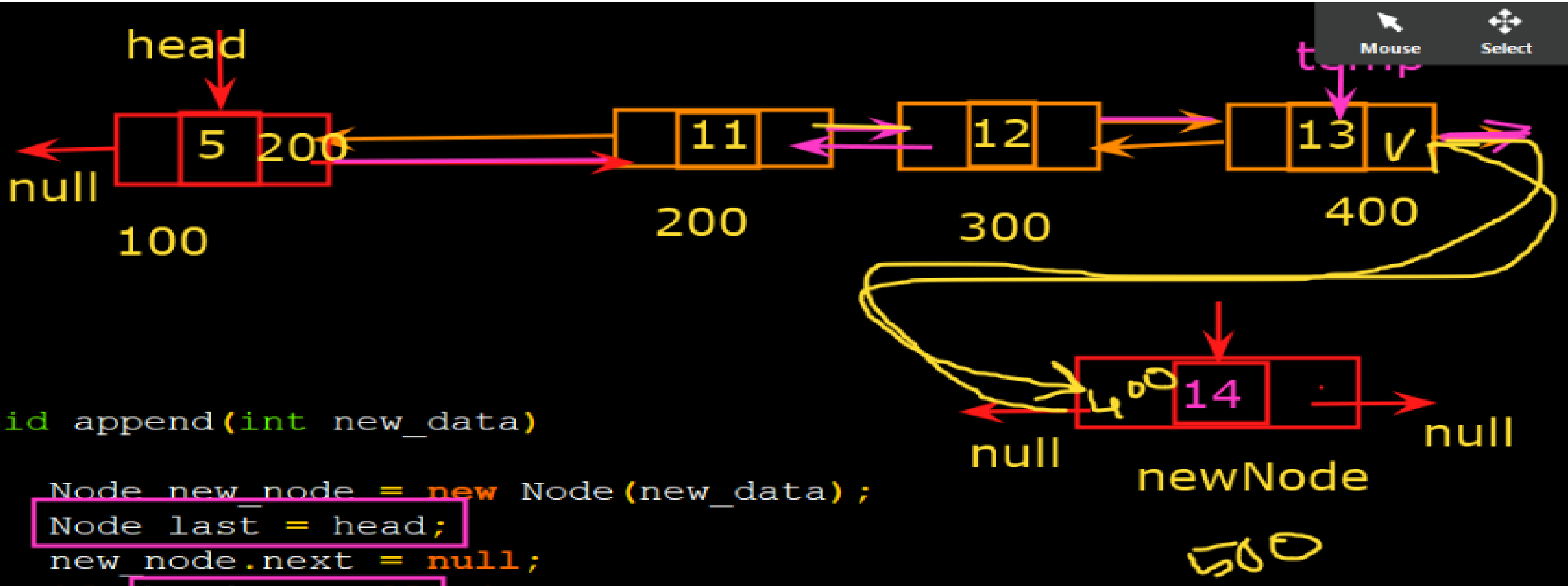
```
}
```



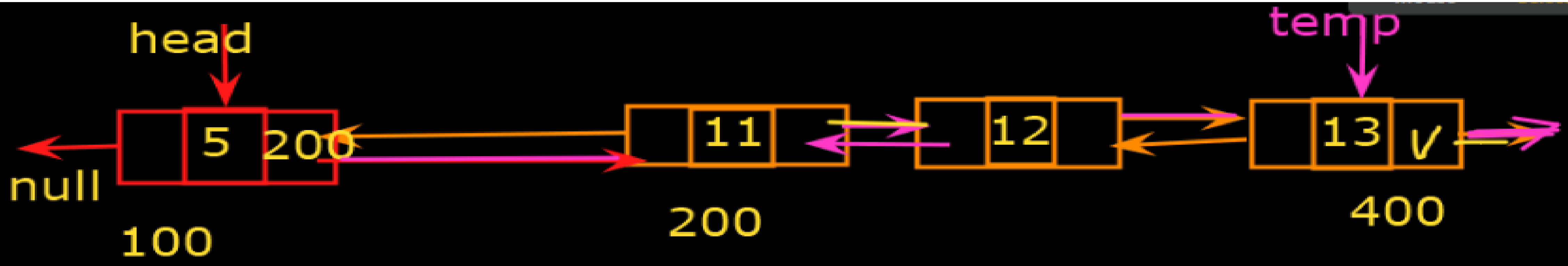

```
//Intersion at Begining
public void insert(int new_data)
{
    Node new Node = new Node(new_data);
    new_Node.next = head;
    new_Node.prev = null;
    if (head != null)
        head.prev = new_Node;
    head = new_Node;
}

}
```



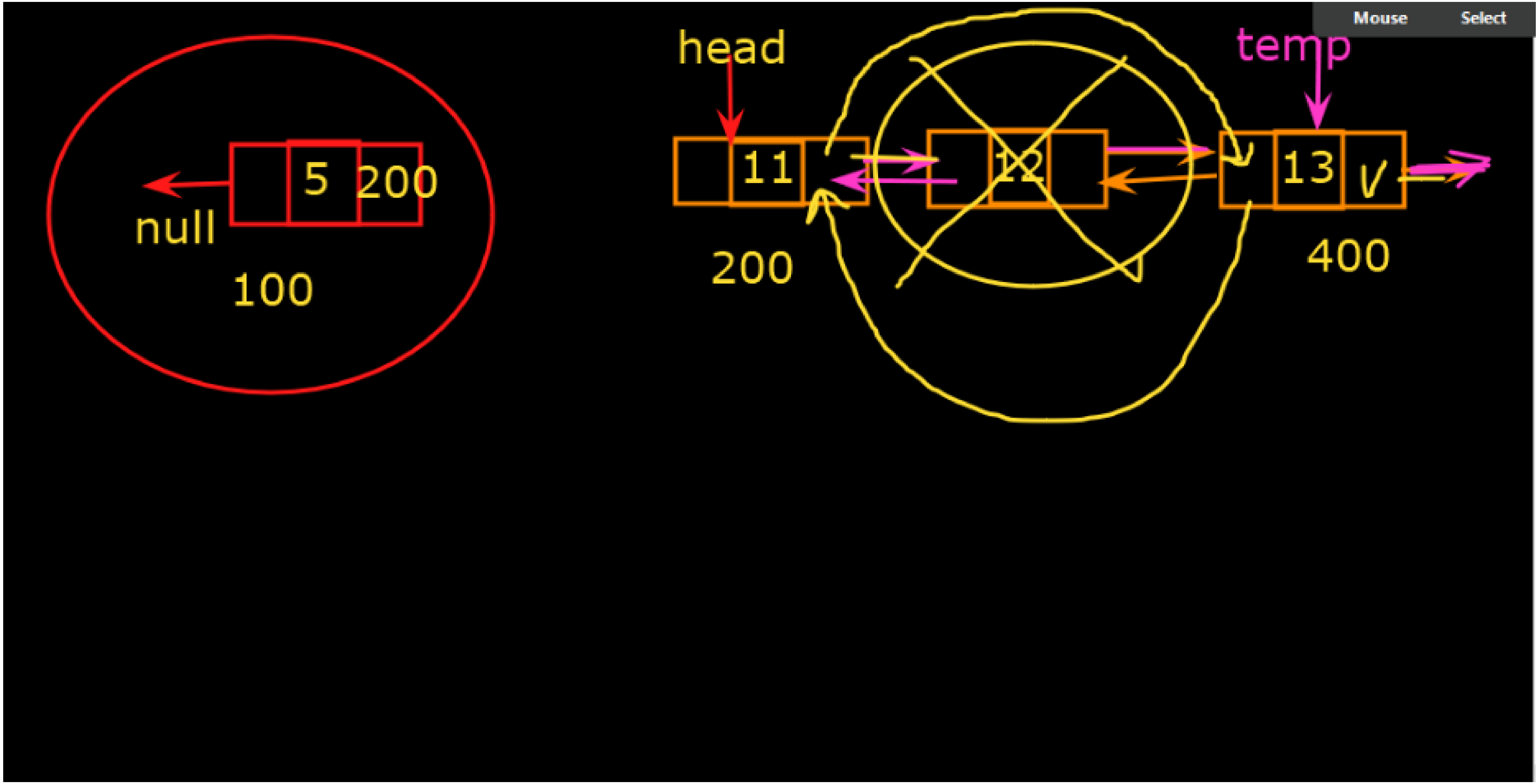

```
public void InsertAfter(Node prev_Node, int new_data)
{
    if (prev_Node == null) {
        System.out.println("The given previous node cannot be NULL ")
        return;
    }
    Node new_node = new Node(new_data);
    new_node.next = prev_Node.next;
    prev_Node.next = new_node;
    new_node.prev = prev_Node;
    if (new_node.next != null)
        new_node.next.prev = new_node;
}
```

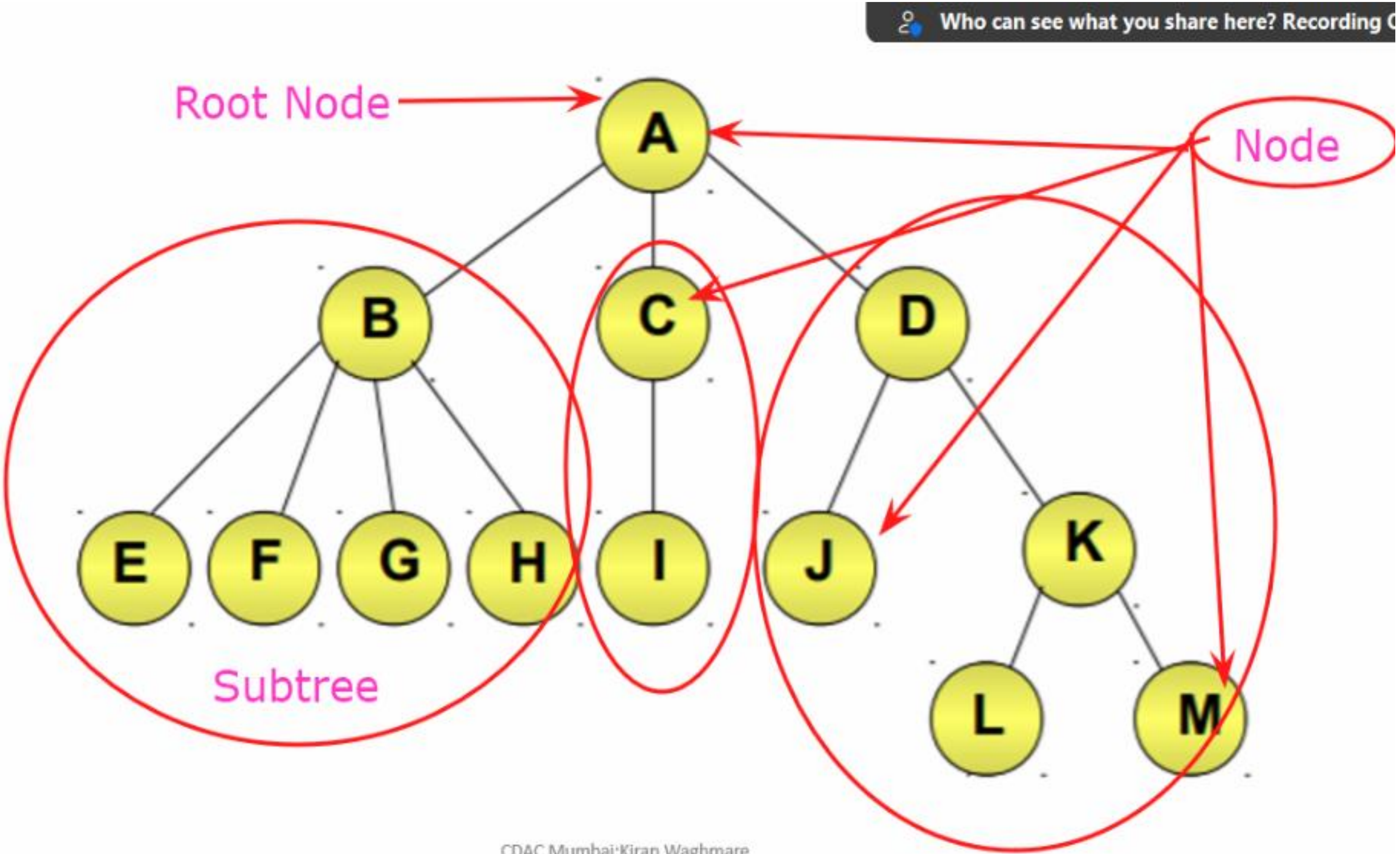


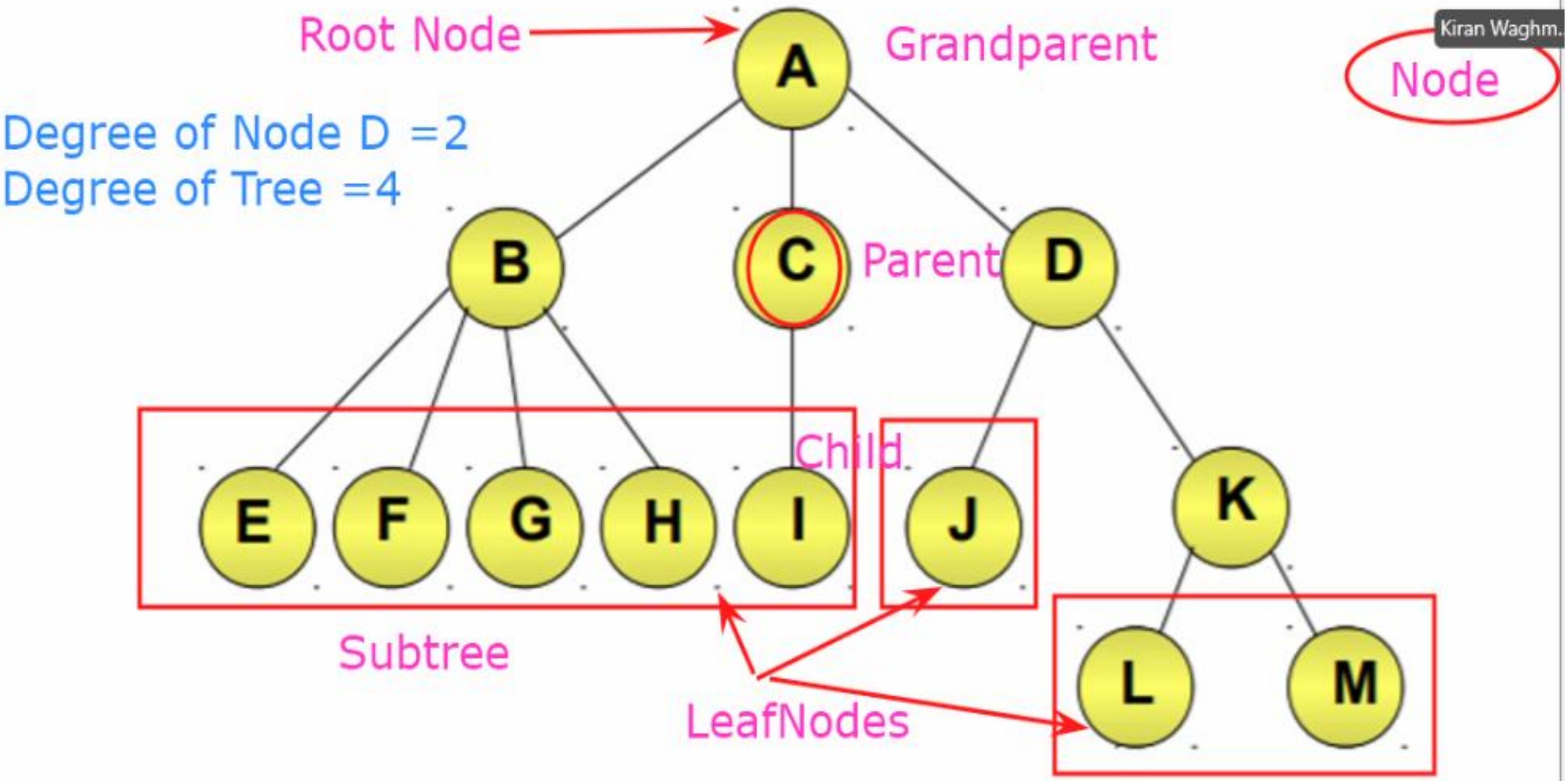

```
void append(int new_data)
{
    Node new_node = new Node(new_data);
    Node last = head;
    new_node.next = null;
    if (head == null) {
        new_node.prev = null;
        head = new_node;
        return;
    }
    while (last.next != null)
        last = last.next;
    last.next = new_node;
    new_node.prev = last;
}
```



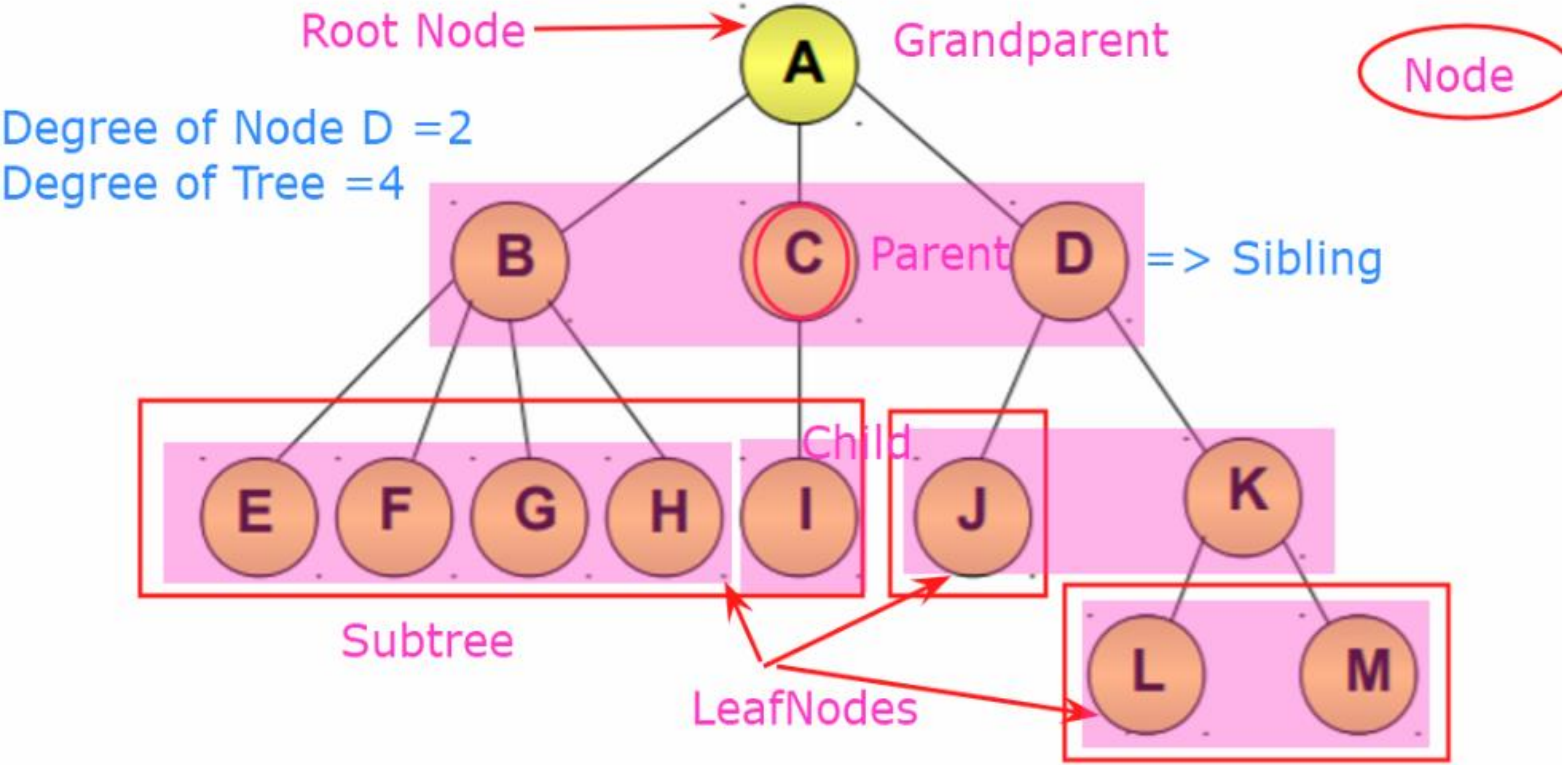

```java
static void display()
{
    Node temp = head;
    if (temp == null)
        System.out.print("Doubly Linked list empty");

    while (temp != null)
    {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
}
```

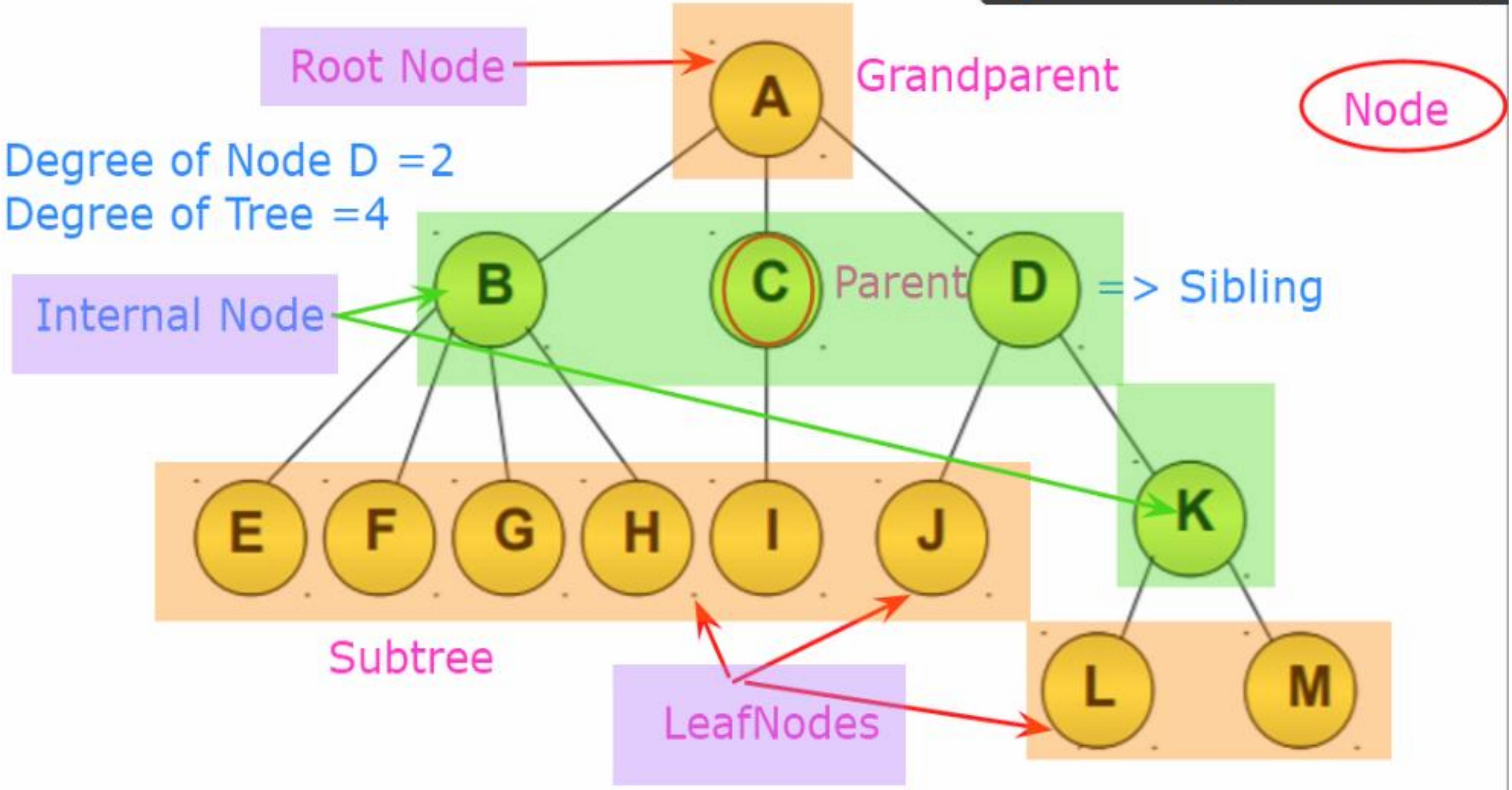

Mouse
Select
head
temp
5
200
null
100
11
12
13
200
400

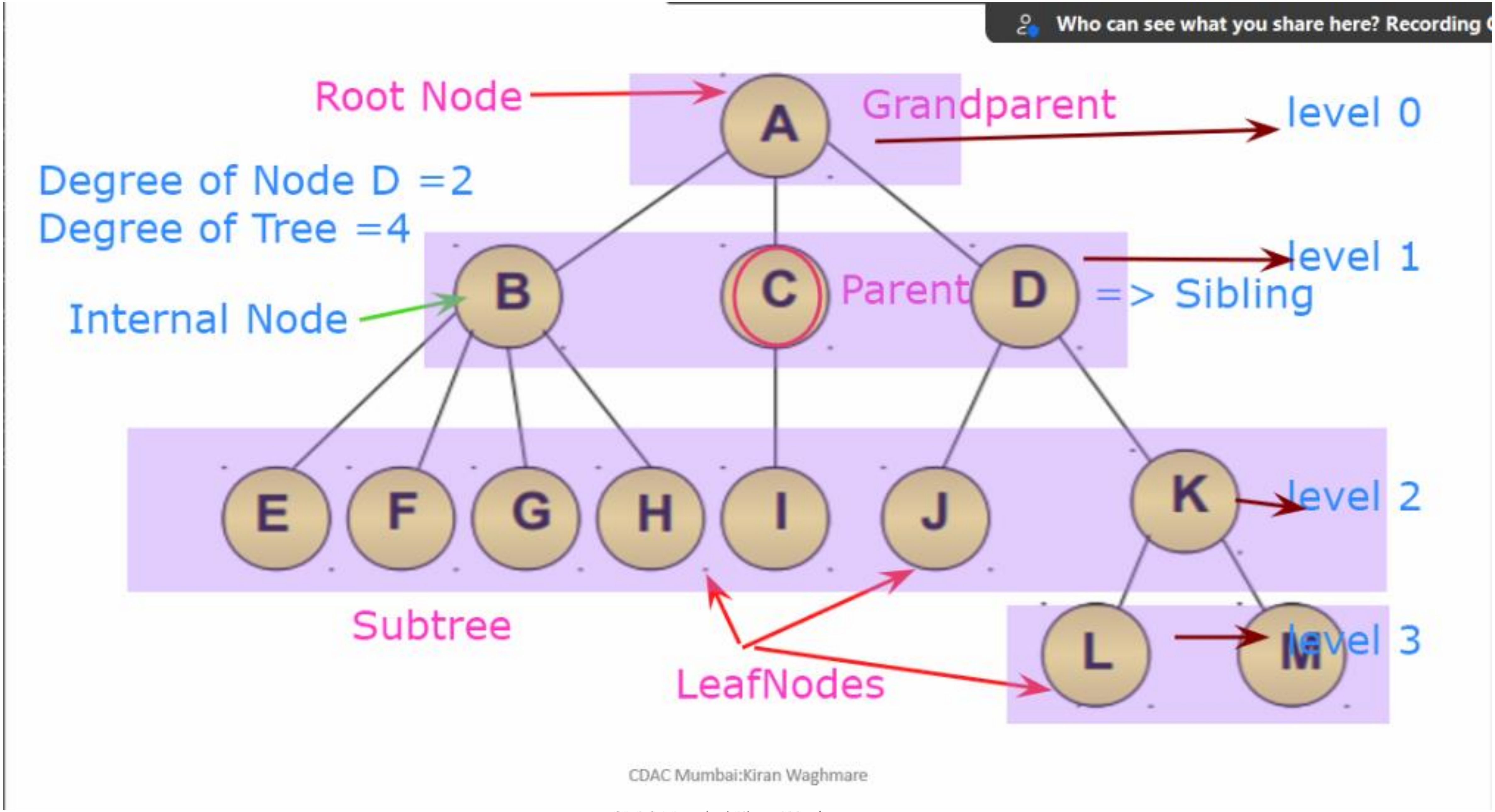

Root Node
Node
A
B
C
D
E
F
G
H
I
J
K
L
M
Subtree

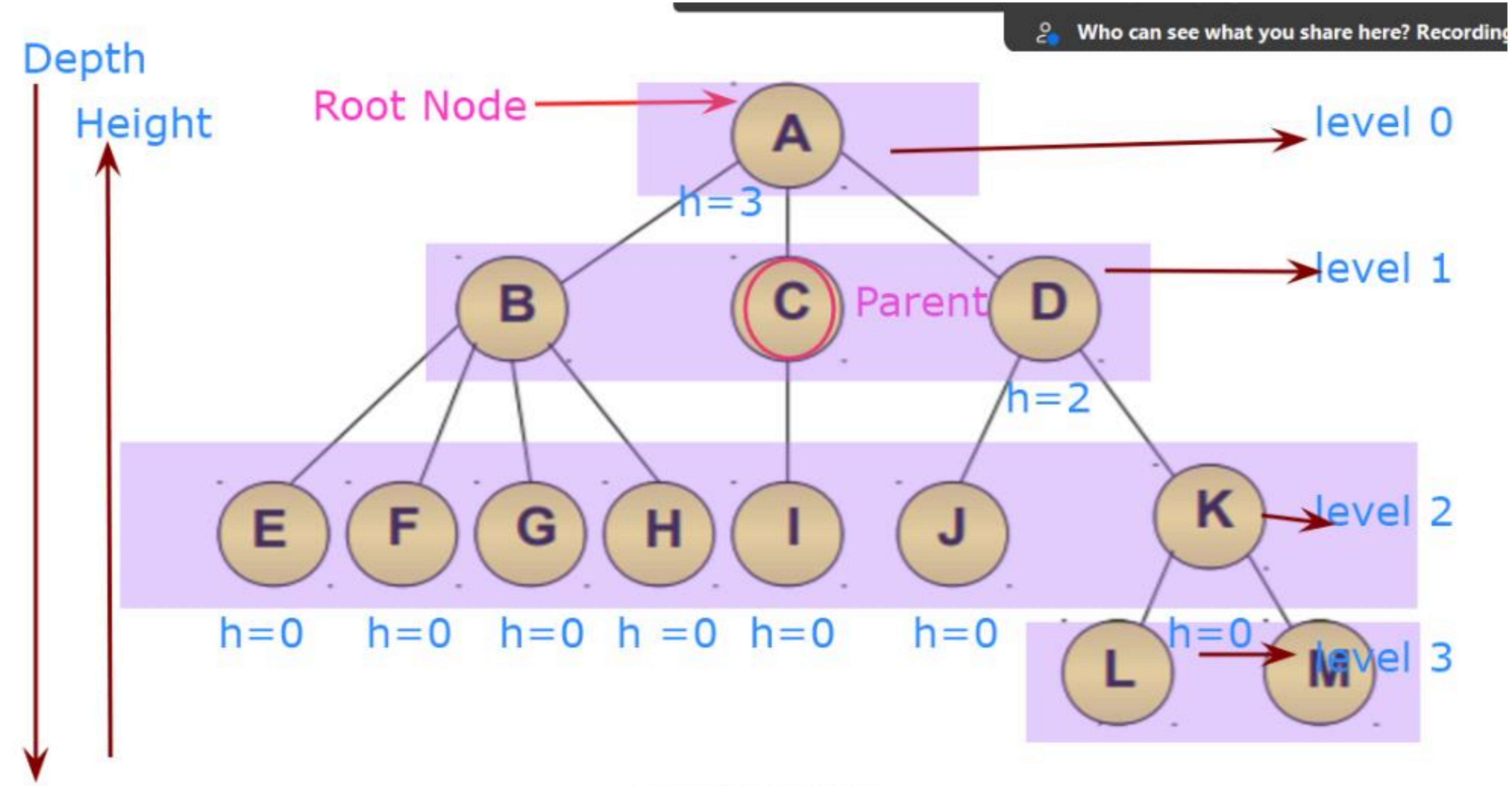


Root Node
Grandparent
Kiran Waghm...
Node
Degree of Node D =2
Degree of Tree =4
A
B
C
Parent
D
Child
E
F
G
H
I
J
K
L
M
Subtree
LeafNodes


Root Node
Grandparent
Node
Degree of Node D =2
Degree of Tree =4
A
B
C
Parent
D
=> Sibling
Child
E
F
G
H
I
J
K
L
M
Subtree
LeafNodes


Root Node
A
Grandparent
Node
Degree of Node D =2
Degree of Tree =4
B
C
Parent
D
=> Sibling
Internal Node
E
F
G
H
I
J
K
L
M
Subtree
LeafNodes

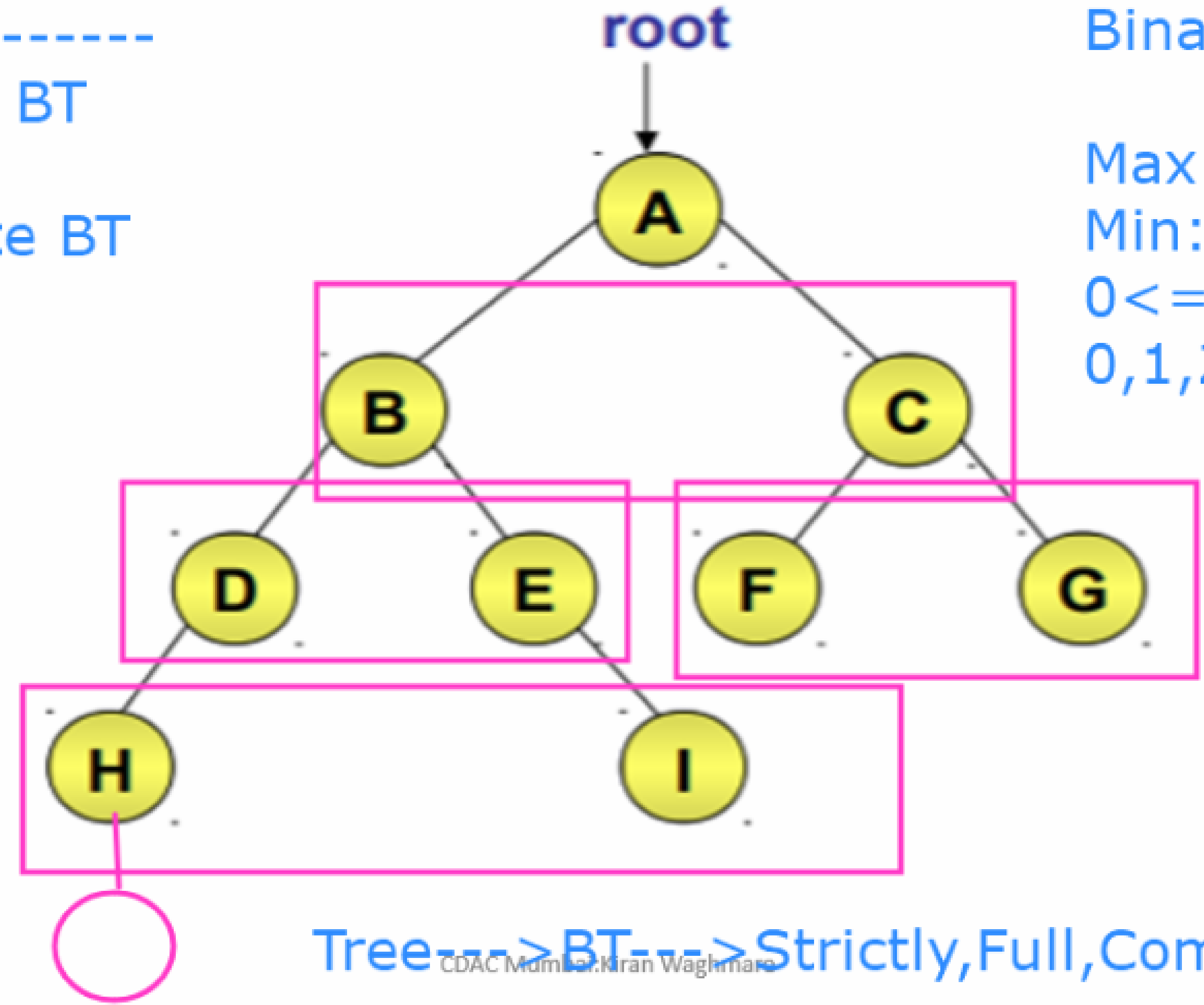

Who can see what you share here? Recording
Root Node
A
Grandparent
level 0
Degree of Node D =2
Degree of Tree =4
B
C
Parent
D
level 1
=> Sibling
Internal Node
E
F
G
H
I
J
K
level 2
Subtree
L
M
level 3
LeafNodes
CDAC Mumbai:Kiran Waghmare


Depth
Height
Root Node
A
level 0
h=3
B
C
Parent
D
level 1
h=2
E
F
G
H
I
J
K
level 2
h=0
h=0
h=0
h =0
h=0
h=0
h=0
L
M
level 3

Types of BT

----

1. Strictly BT
2.Full BT
3.Complete BT



Binary Tree

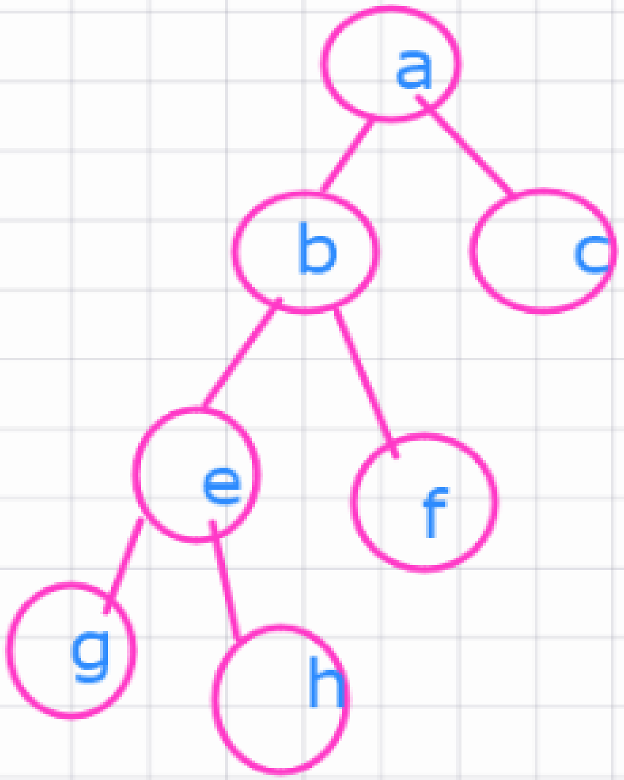
Max: 2
Min:0
0<=2
0,1,2

Tree--->BT--->Strictly,Full,Complete

Types of BT

----

1. Strictly BT
2.Full BT
3.Complete BT



Binary Tree

Max: 2
Min:0
0<=2
0,1,2

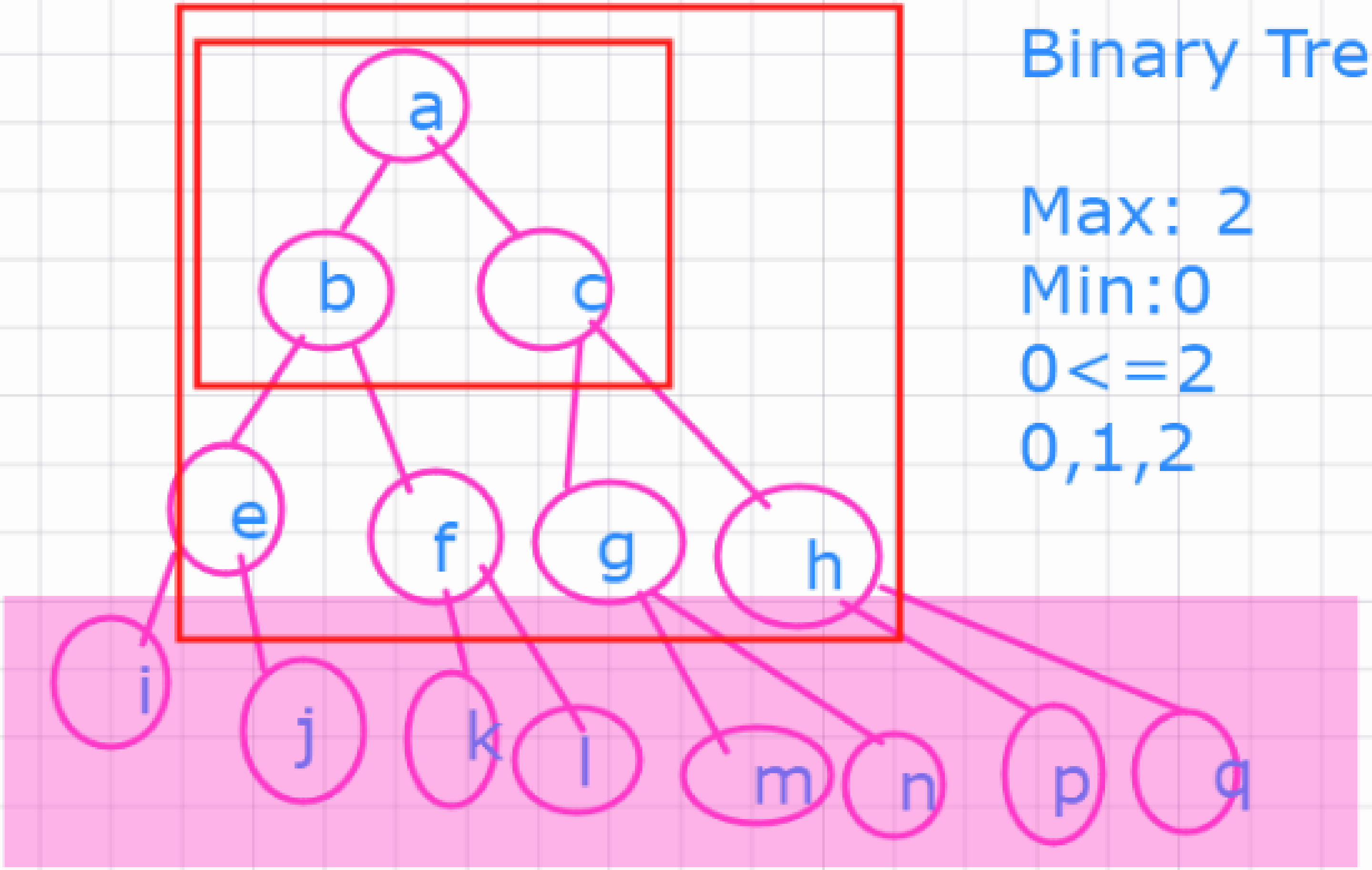
Strictly BT :every node, except for the leaf node,has non empty left and right children (2 children)

Tree--->BT--->Strictly,Full,Complete

Types of BT

----

1. Strictly BT
2. Full BT
3. Complete BT



Binary Tree

Max: 2
Min:0
0<=2
0,1,2

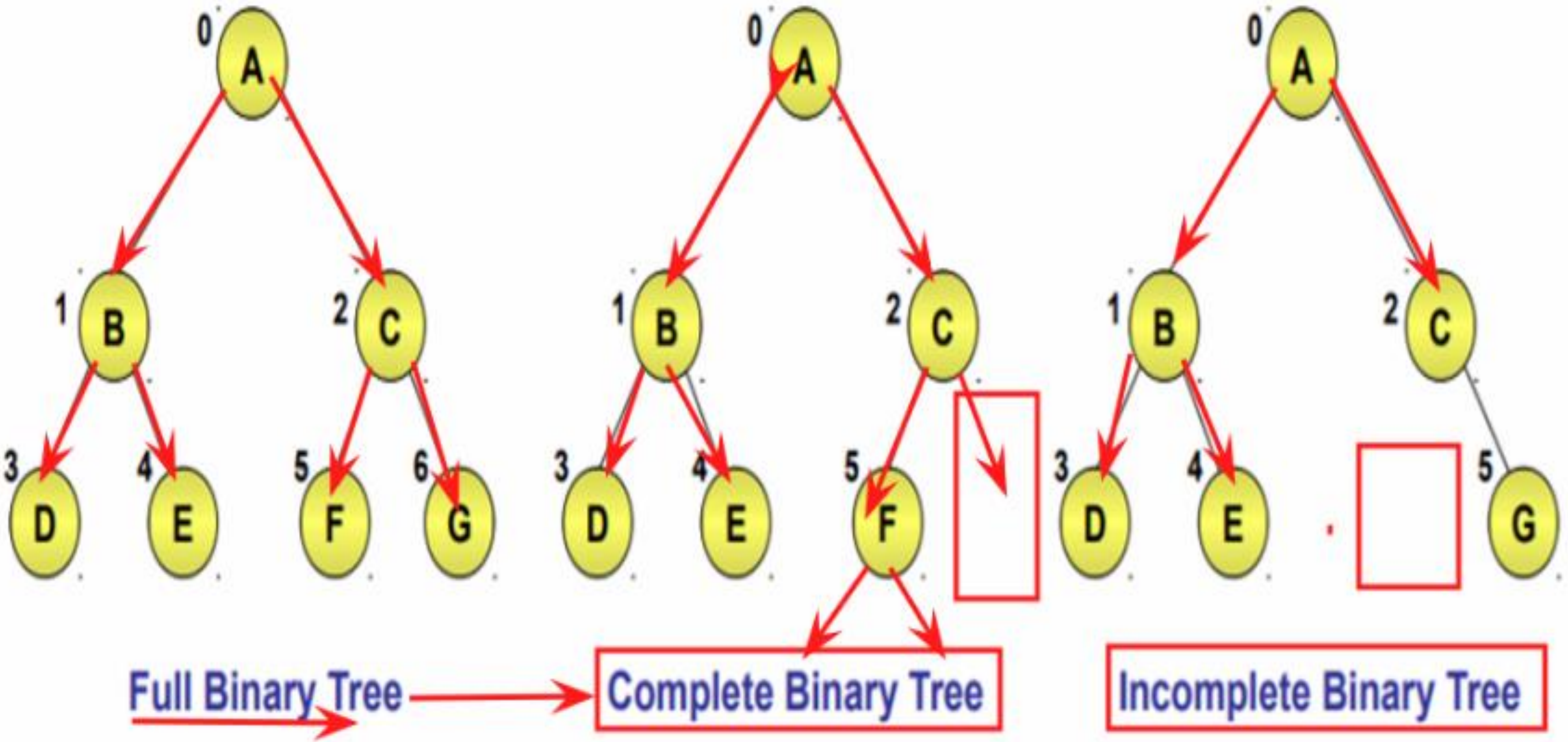
Full BT :

$2^d - 1$

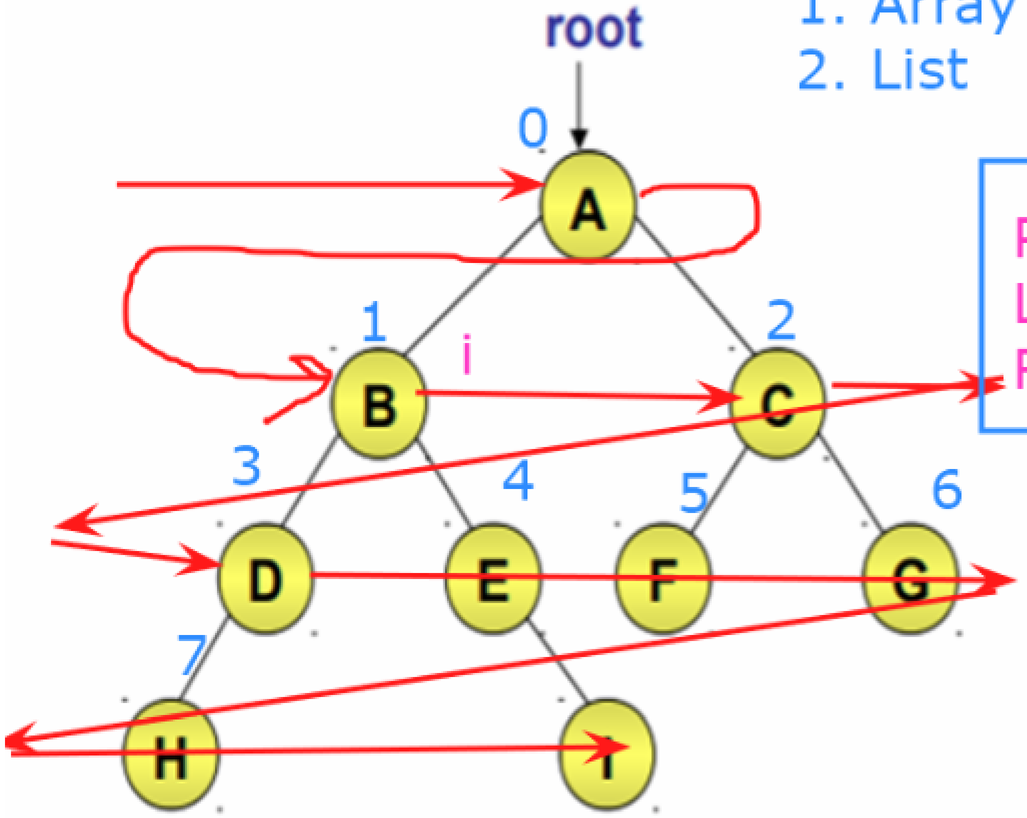
d=3

No. of nodes: $2^3 - 1 = 7$

Tree--->BT--->Strictly,Full,Complete


0
A
1
B
2
C
3
D
4
E
5
F
6
G
Full Binary Tree
0
A
1
B
2
C
3
D
4
E
5
F
Complete Binary Tree
0
A
1
B
2
C
3
D
4
E
5
G
Incomplete Binary Tree

Representation of BT
1. Array
2. List

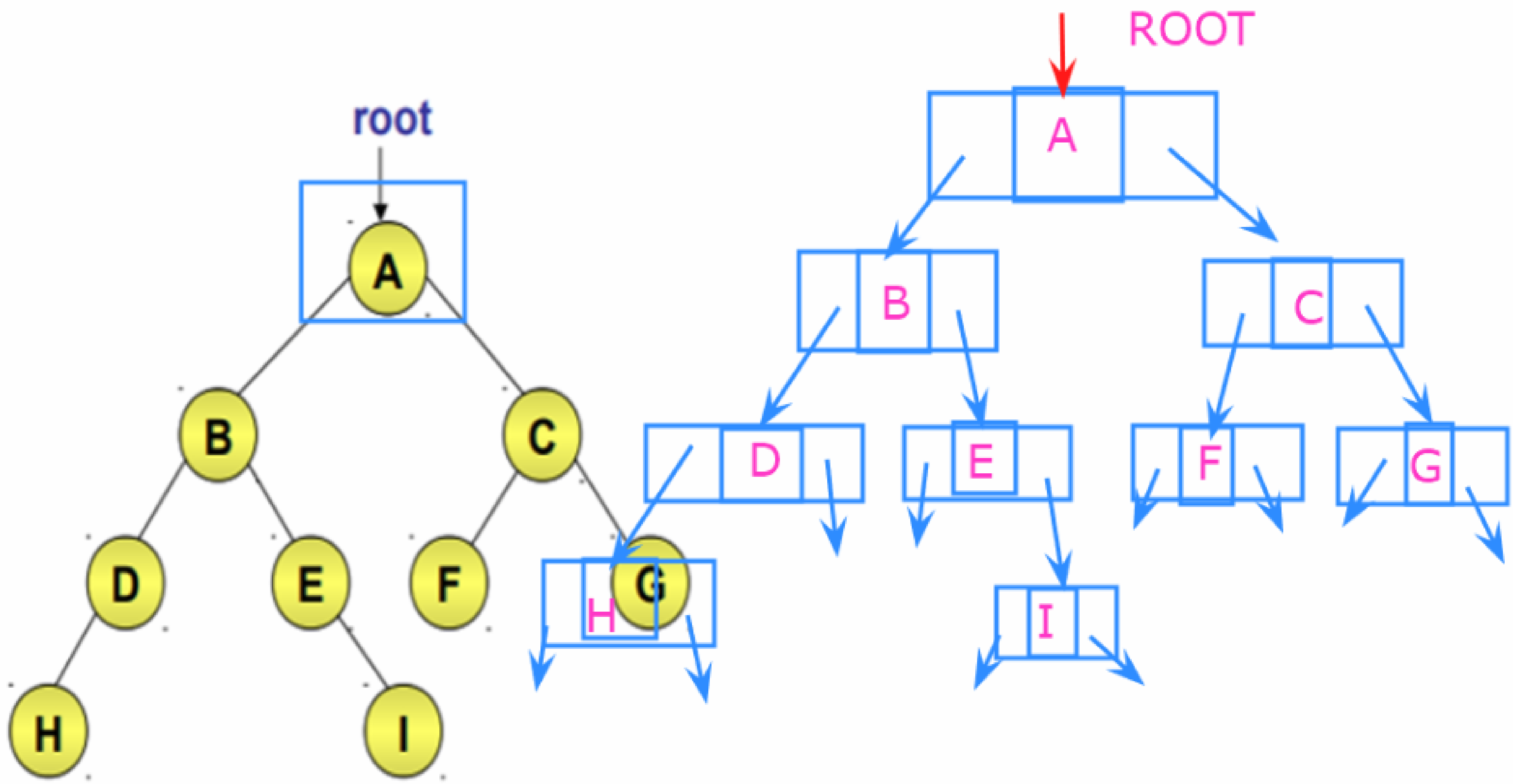


Parent(i) =(i-1)/2
Leftchild(i)=(2*i)+1
RightChild(i)=(2*i)+2

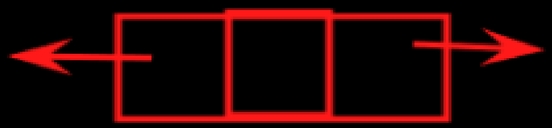
Node B=i=1
Parent = A
LC=D
RC=E

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	

CDAC Mumbai:Kiran Waghmare


root
A
B
C
D
E
F
G
H
I
ROOT
A
B
C
D
E
F
G
H
I

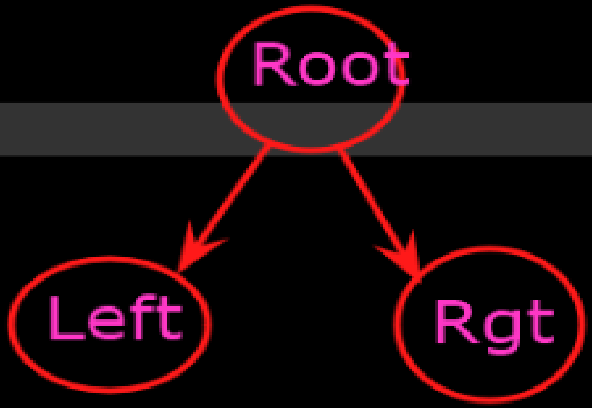
```java
class BT
{
    Node root;
static class Node
{
    int data;
    Node left,right;


    Node(int d)
    {
        data = d;
        left = right = null;
    }

}
}
```



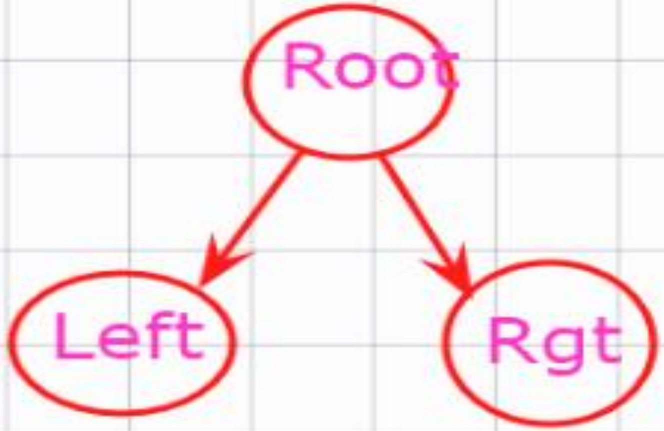
Tree Travese

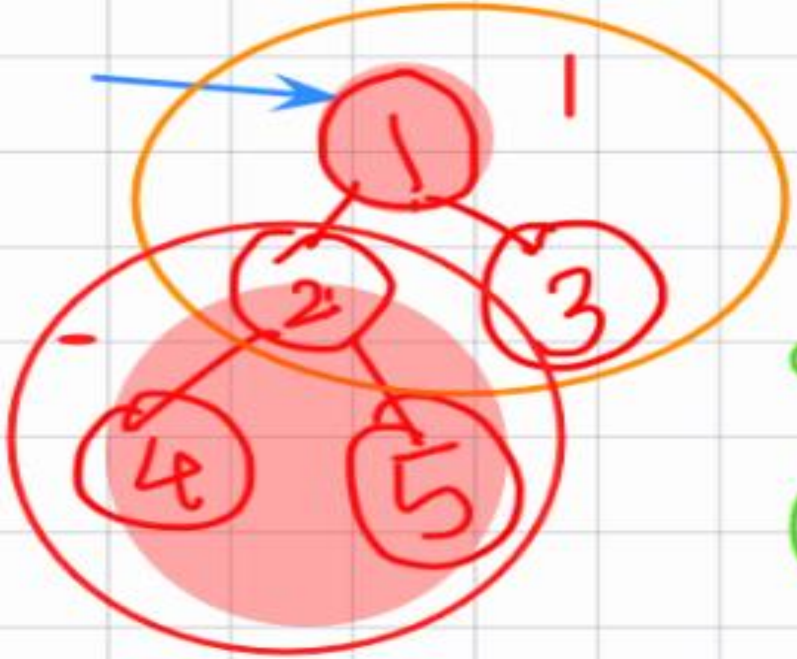
----

1. Inorder:Left,Root,Rgt
2. Preorder:Root,Left,Rgt
3. Postorder:Left,Rgt, Root



1. Inorder:Left,Root,Rgt
2. Preorder:Root,Left,Rgt
3. Postorder:Left,Rgt, Root





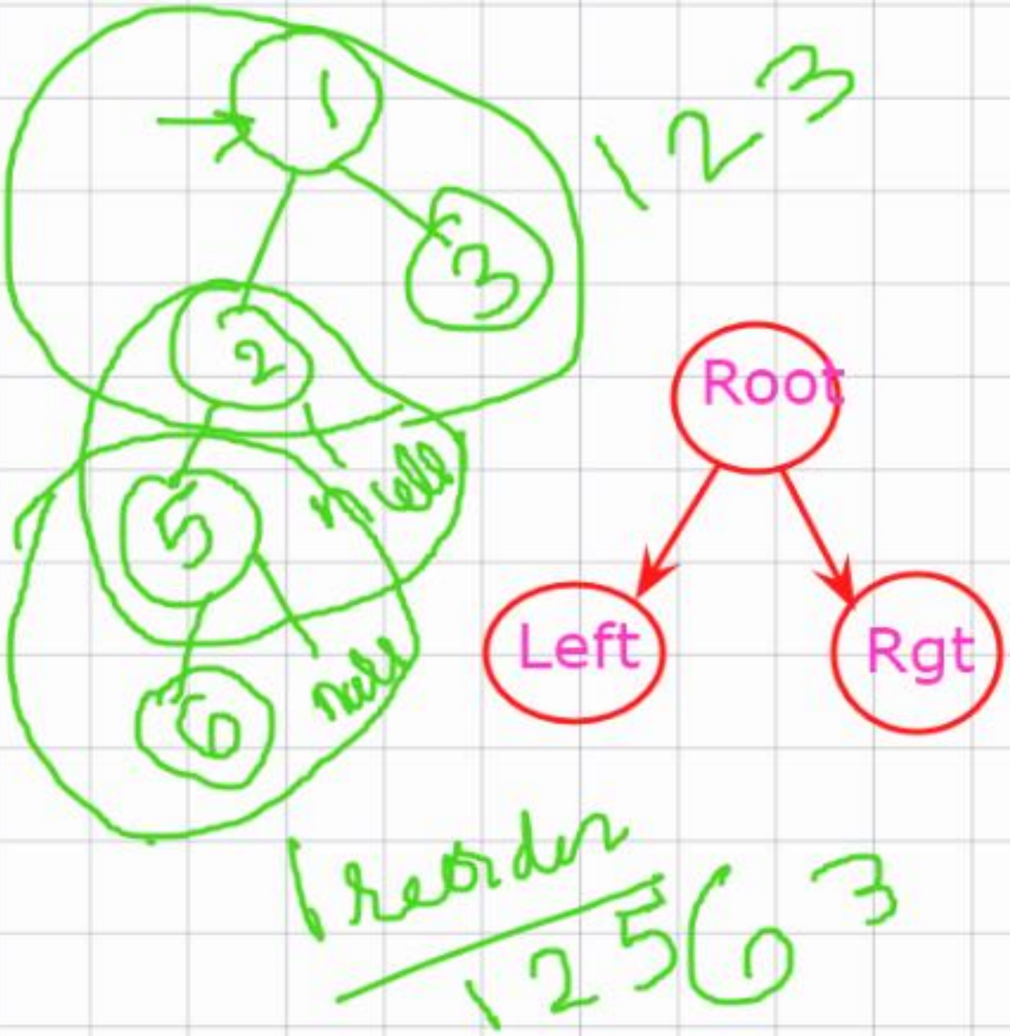
0

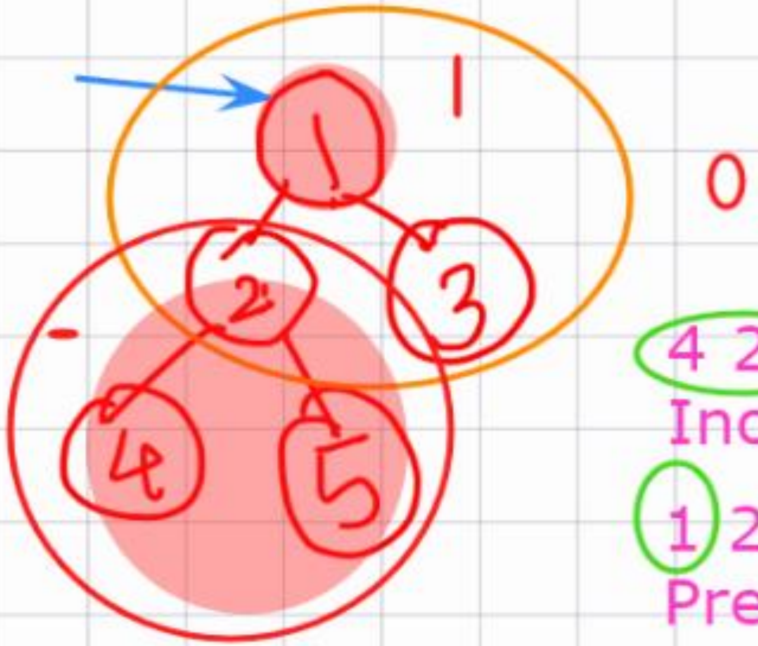
4 2 5 1 3 : Inorder

1 2 4 5 3 Preorder

4 5 2 3 1 Postorder

1. Inorder: Left, Root, Rgt
2. Preorder: Root, Left, Rgt
3. Postorder: Left, Rgt, Root





4 2 5 1 3 : Inorder

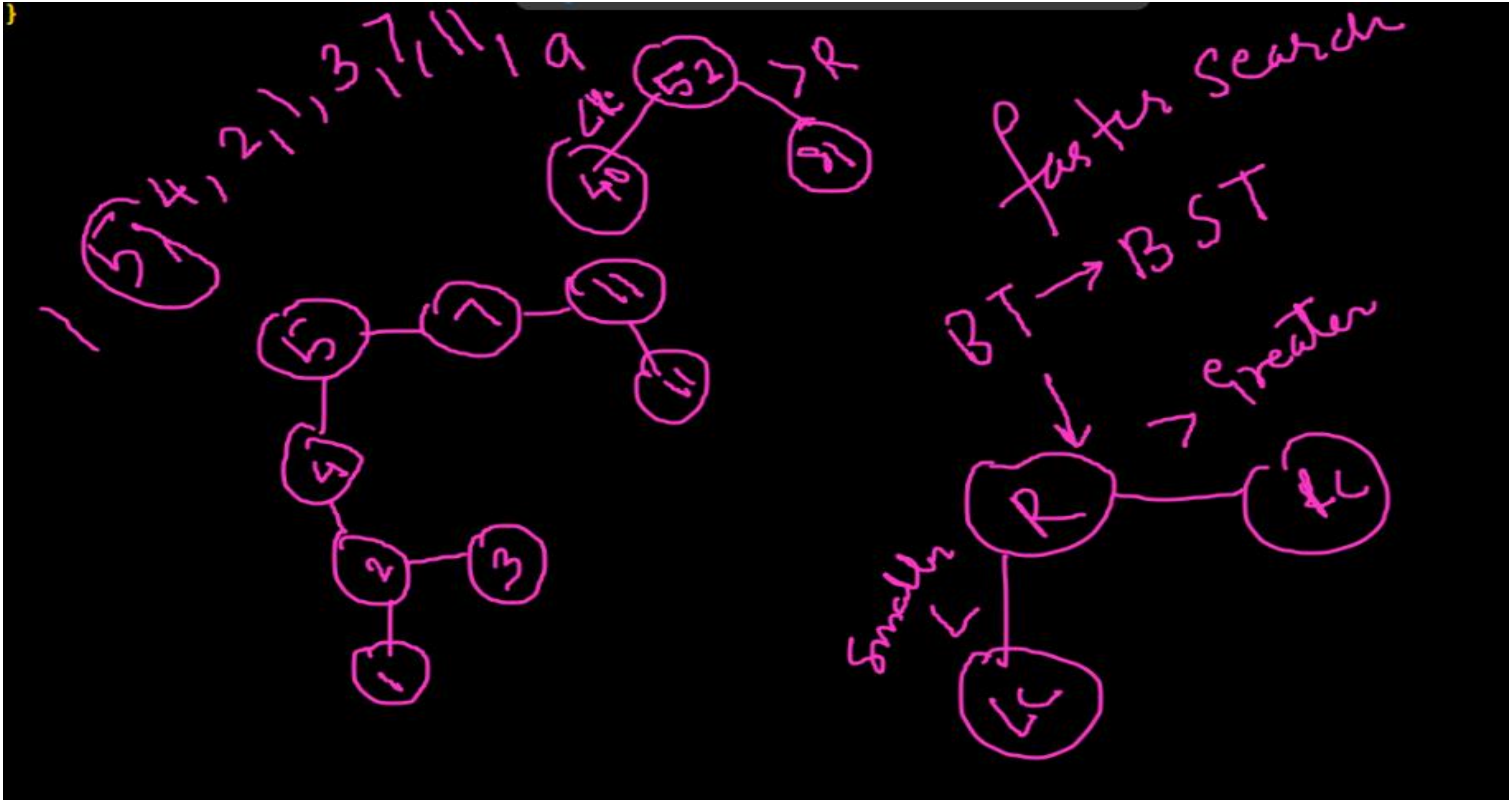
1 2 4 5 3 Preorder

4 5 2 3 1 Postorder

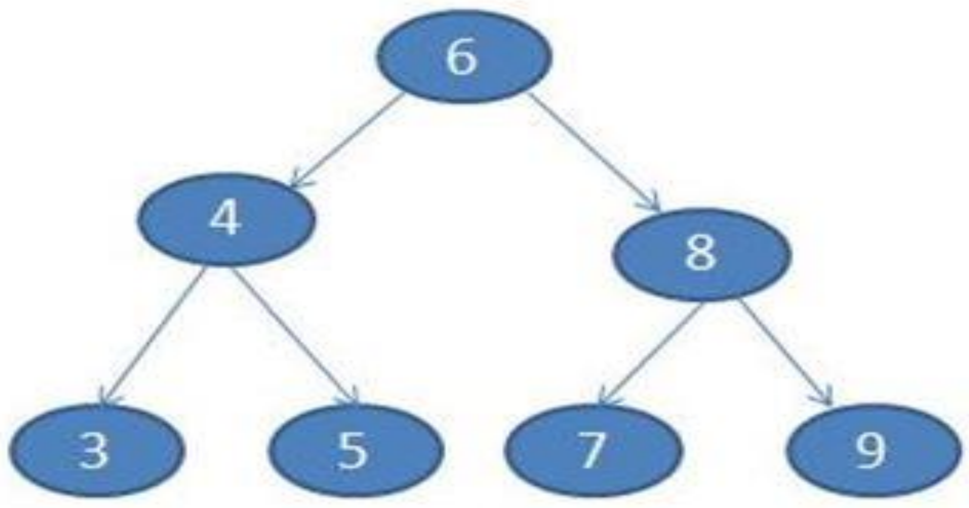
```
{
    BT t1 =new BT();
    t1.root = new Node
    t1.root.left = new
    t1.root.right = nev
    t1.root.left.left =
    t1.root.left.right

    System.out.println
    t1.Inorder();
    System.out.println
    t1.Preorder();
    System.out.println
    t1.Postorder();
    System.out.println
}
}
```

Command Prompt

```
C:\ADS>javac BT.java

C:\ADS>java BT

C:\ADS>javac BT.java

C:\ADS>java BT
Tree Traversal:
14 12 15 11 13
11 12 14 15 13
14 15 12 13 11

C:\ADS>
```


faster search
BT → BST
Greater
Smaller

Tree Traversal

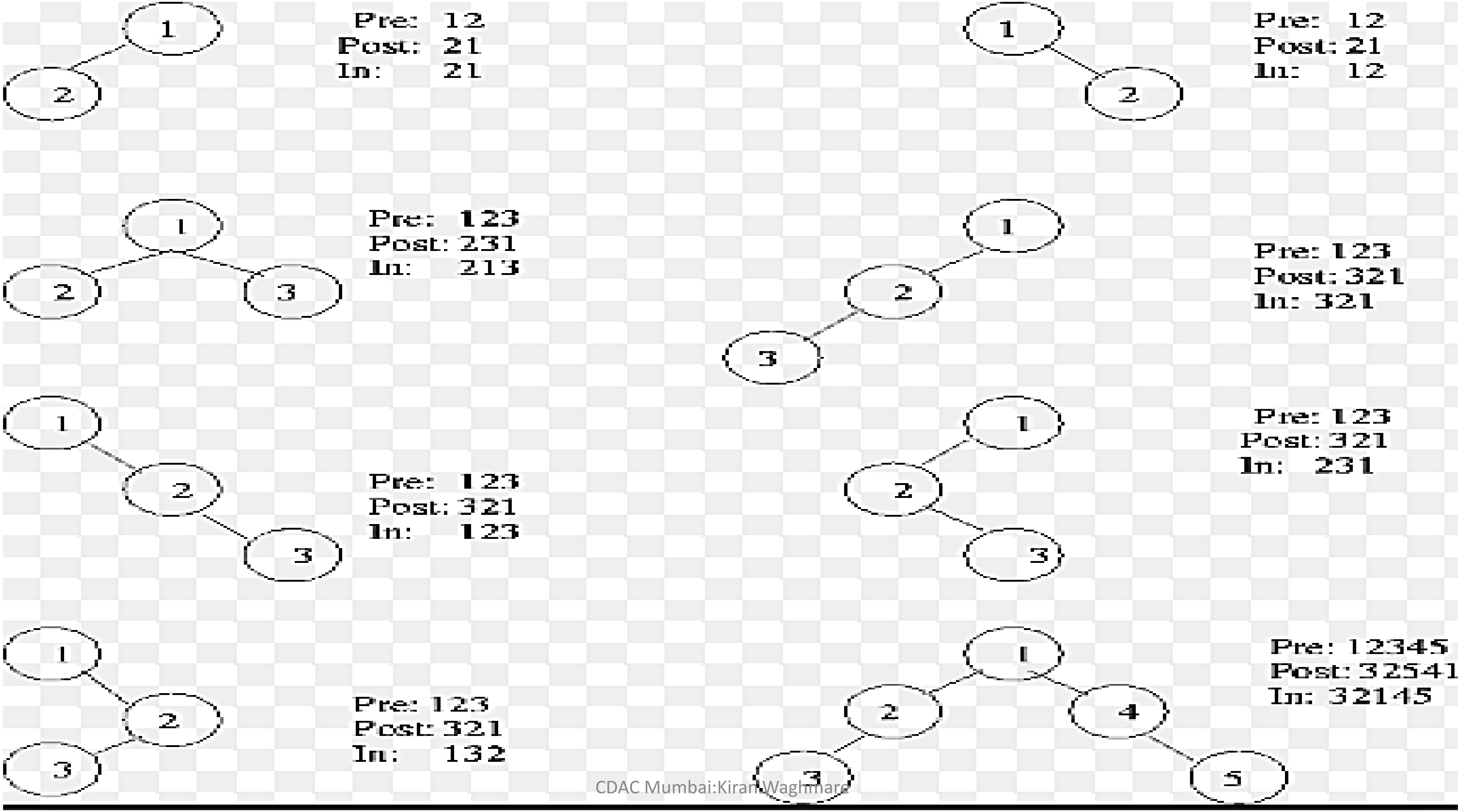


Preorder: Root-Left-Right

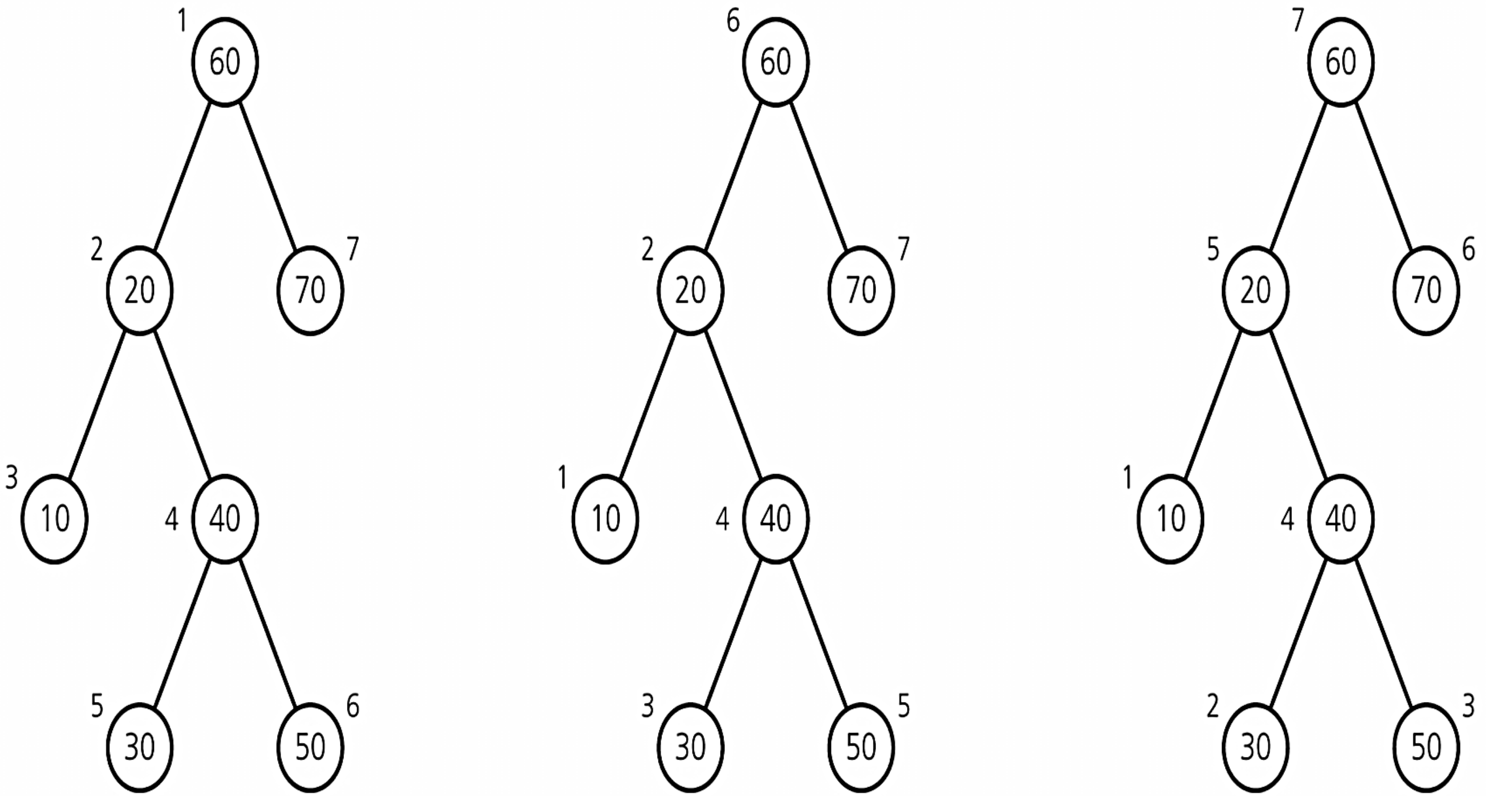
Inorder: Left-Root-Right

Postorder: Left-Right-Root

Preorder : 6, 4, 3, 5, 8, 7, 9
Inorder : 3, 4, 5, 6, 7, 8, 9
Postorder: 3, 5, 4, 7, 9, 8, 6


Pre: 12
Post: 21
In: 21
Pre: 12
Post: 21
In: 12
Pre: 123
Post: 231
In: 213
Pre: 123
Post: 321
In: 321
Pre: 123
Post: 321
In: 123
Pre: 123
Post: 321
In: 231
Pre: 123
Post: 321
In: 132
Pre: 12345
Post: 32541
In: 32145

Binary Tree Traversals

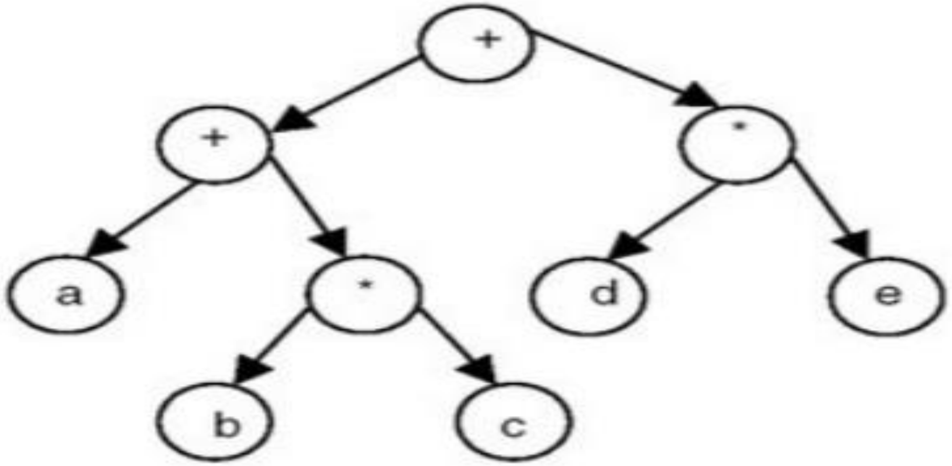


(a) Preorder: 60, 20, 10, 40, 30, 50, 70 (b) Inorder: 10, 20, 30, 40, 50, 60, 70 (c) Postorder: 10, 30, 50, 40, 20, 70, 60

(Numbers beside nodes indicate traversal order.)

Expression Binary Tree Traversal

If an expression is represented as a binary tree, the inorder traversal of the tree gives us an infix expression, whereas the postorder traversal gives us a postfix expression as shown in Figure.



Inorder : a + b * c + d * e
postorder : abc*+de*+

Q. Example : Construct a Binary Search Tree by inserting the following sequence of numbers...

10,12,5,4,20,8,7,15 and 13