Algorithms & Data Structure

Kiran Waghmare

Tree:

-nonlinear data structure

-represents hirearchical data structure

Def: Tree is a non-linear data structure that represents a hirarchical relationship between the various data elements.

Tree:

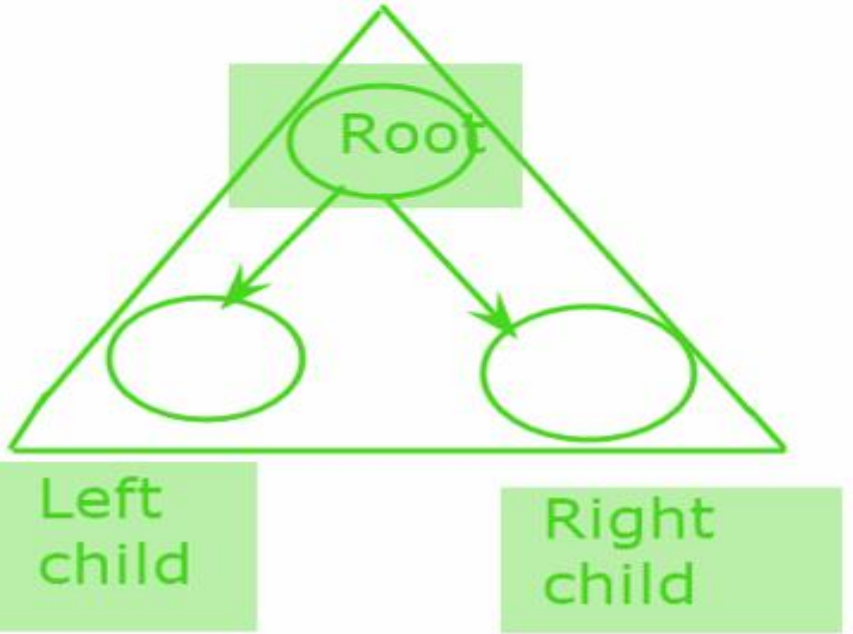
-nonlinear data structure

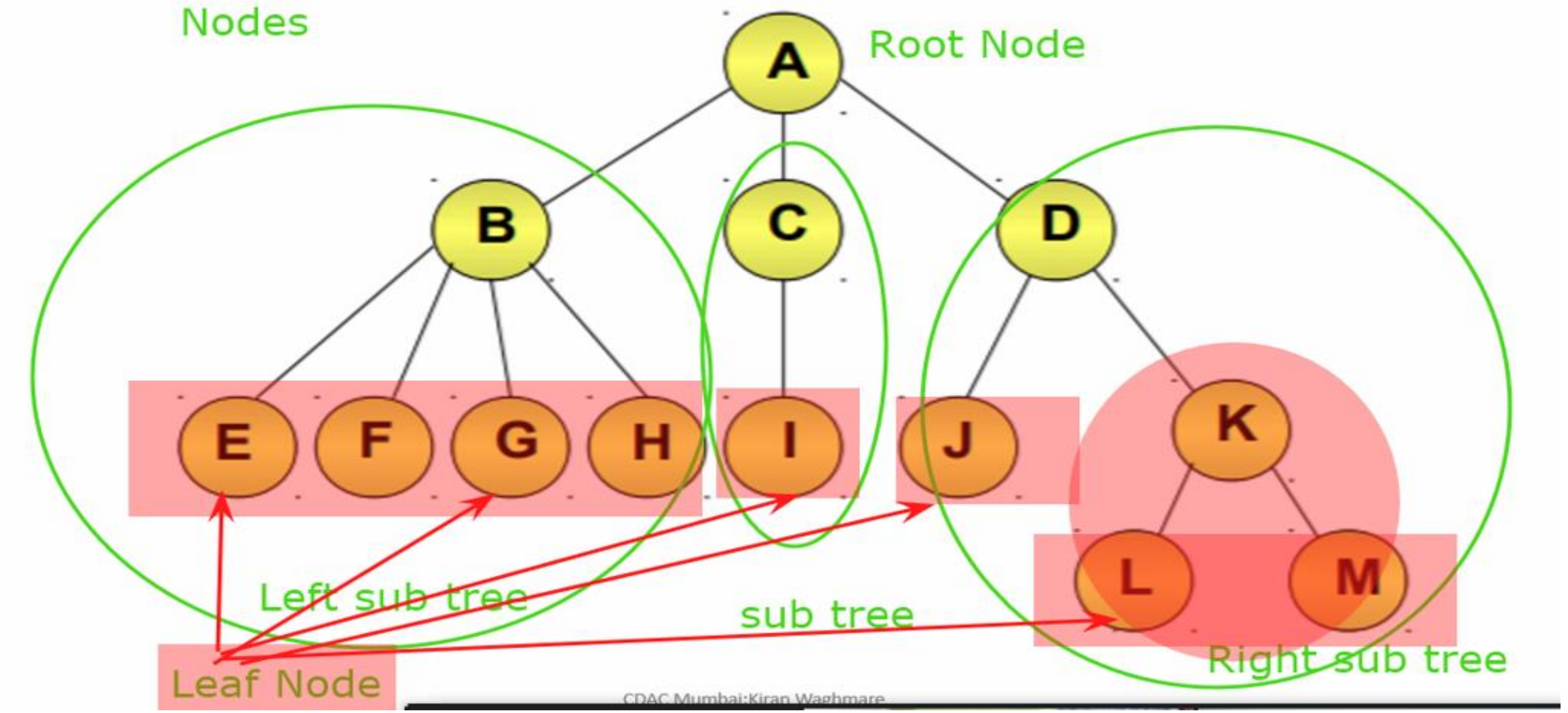
-represents hirearchical data structure

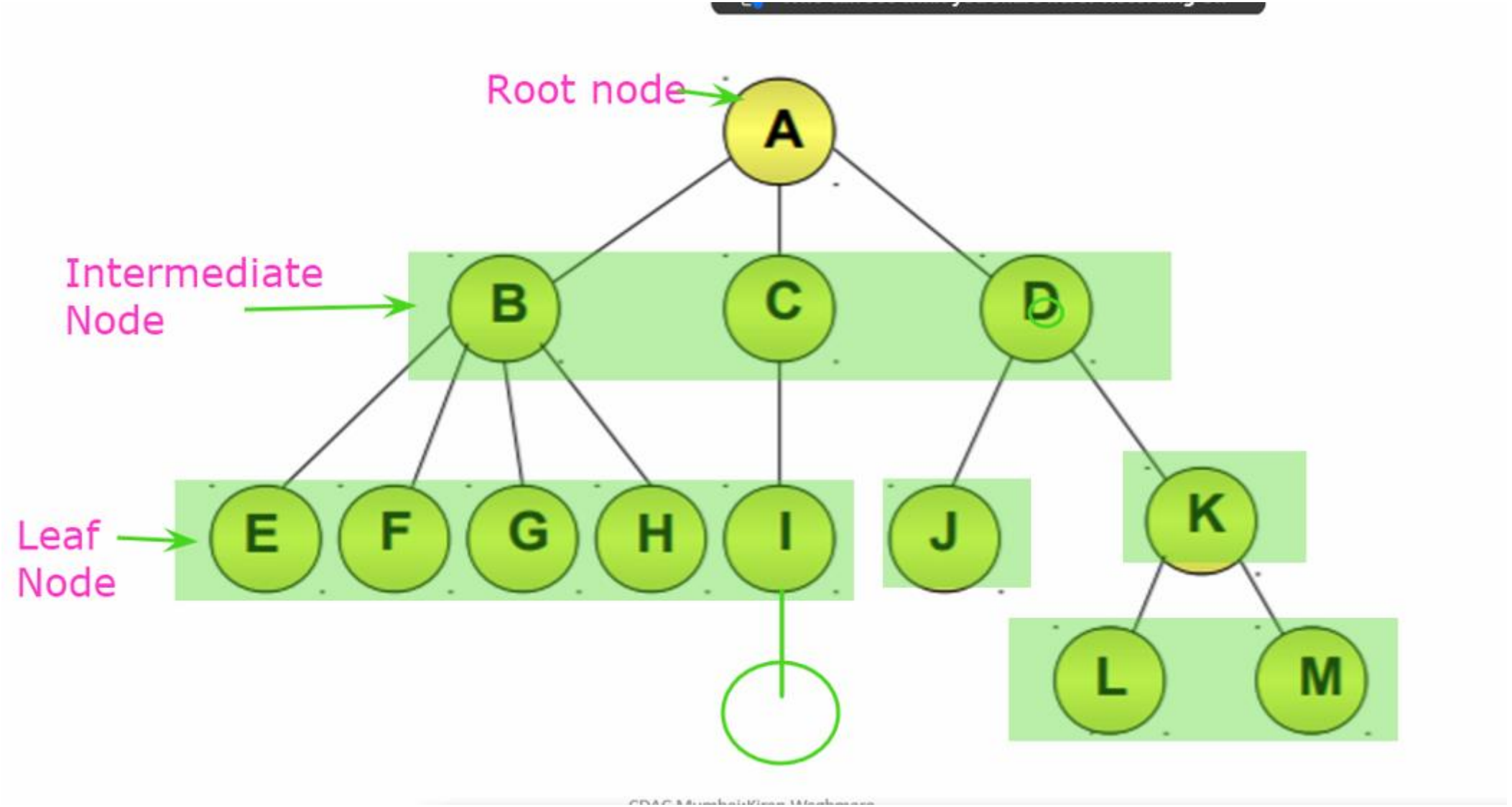
Def: Tree is a non-linear data structure that represents a hirarchical relationship between the various data elements.

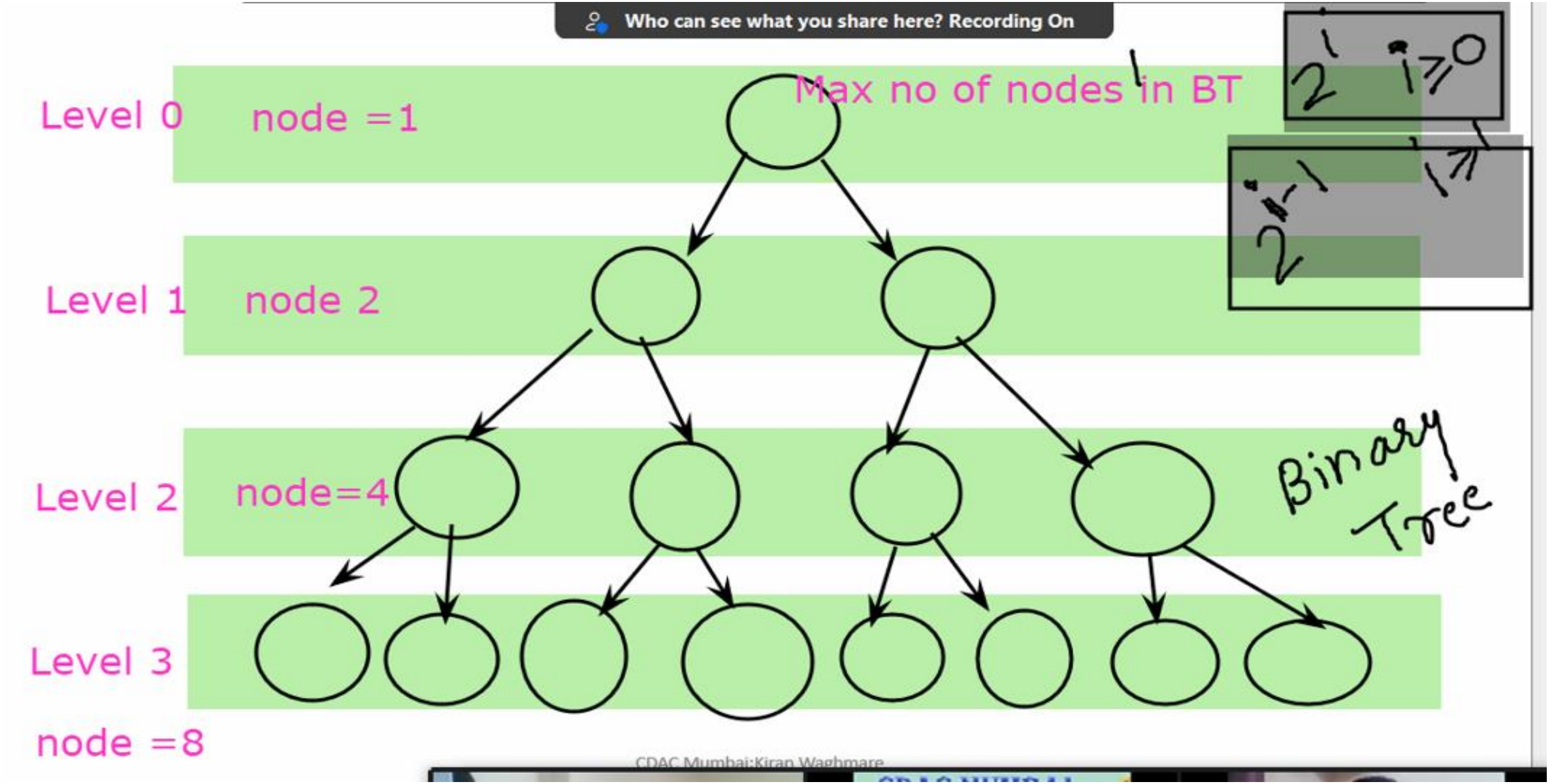
Node: which stores the element

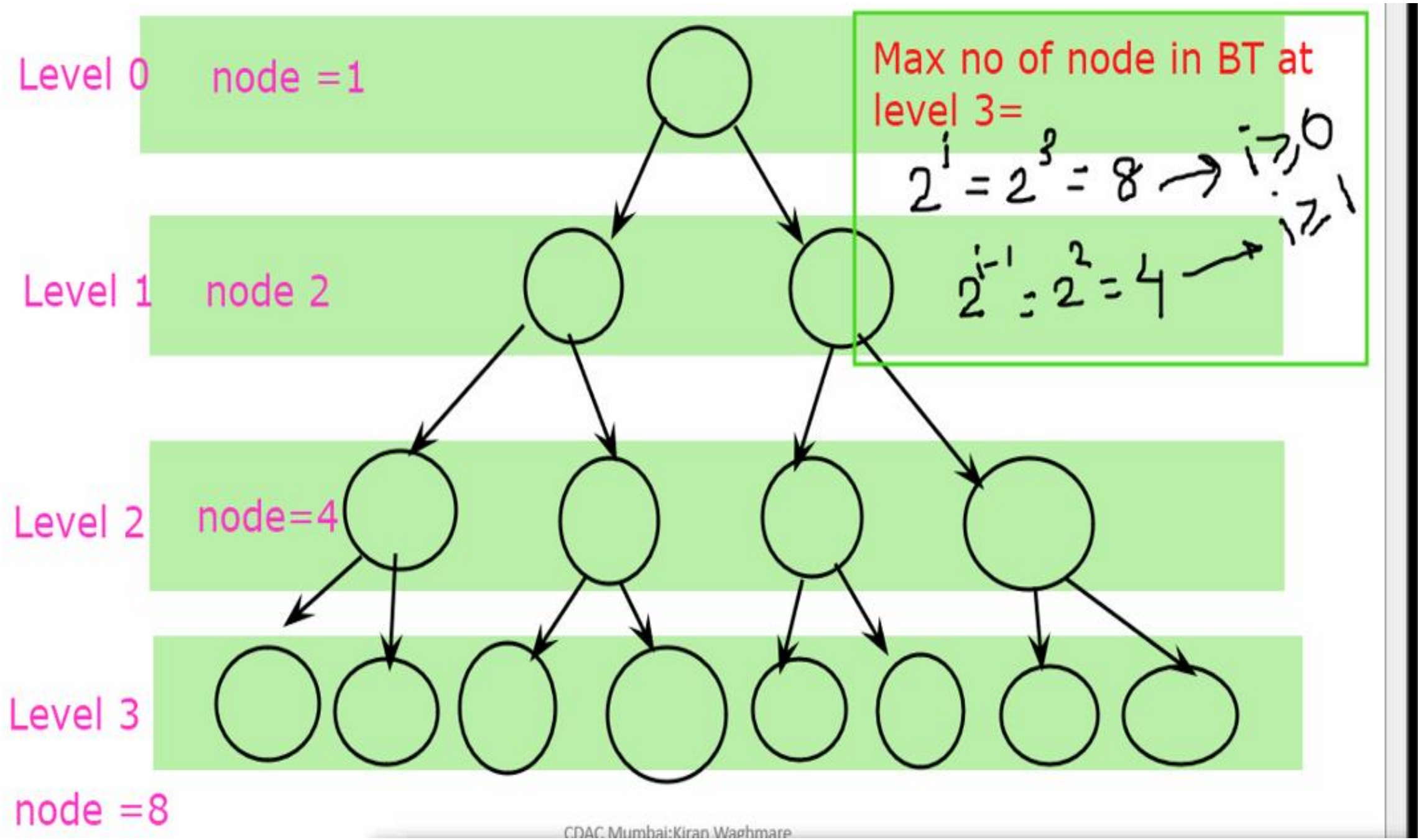
connection: lines

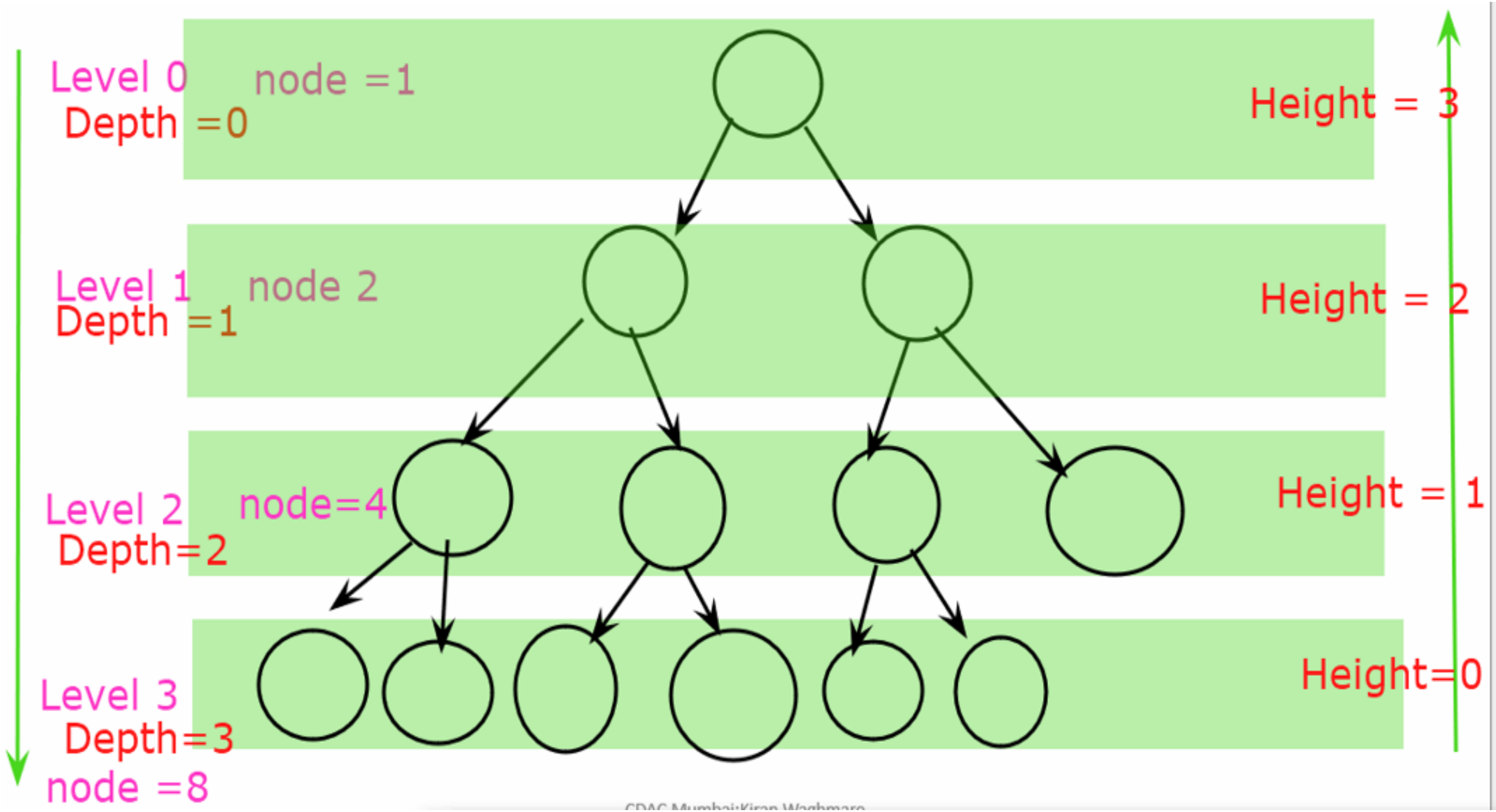


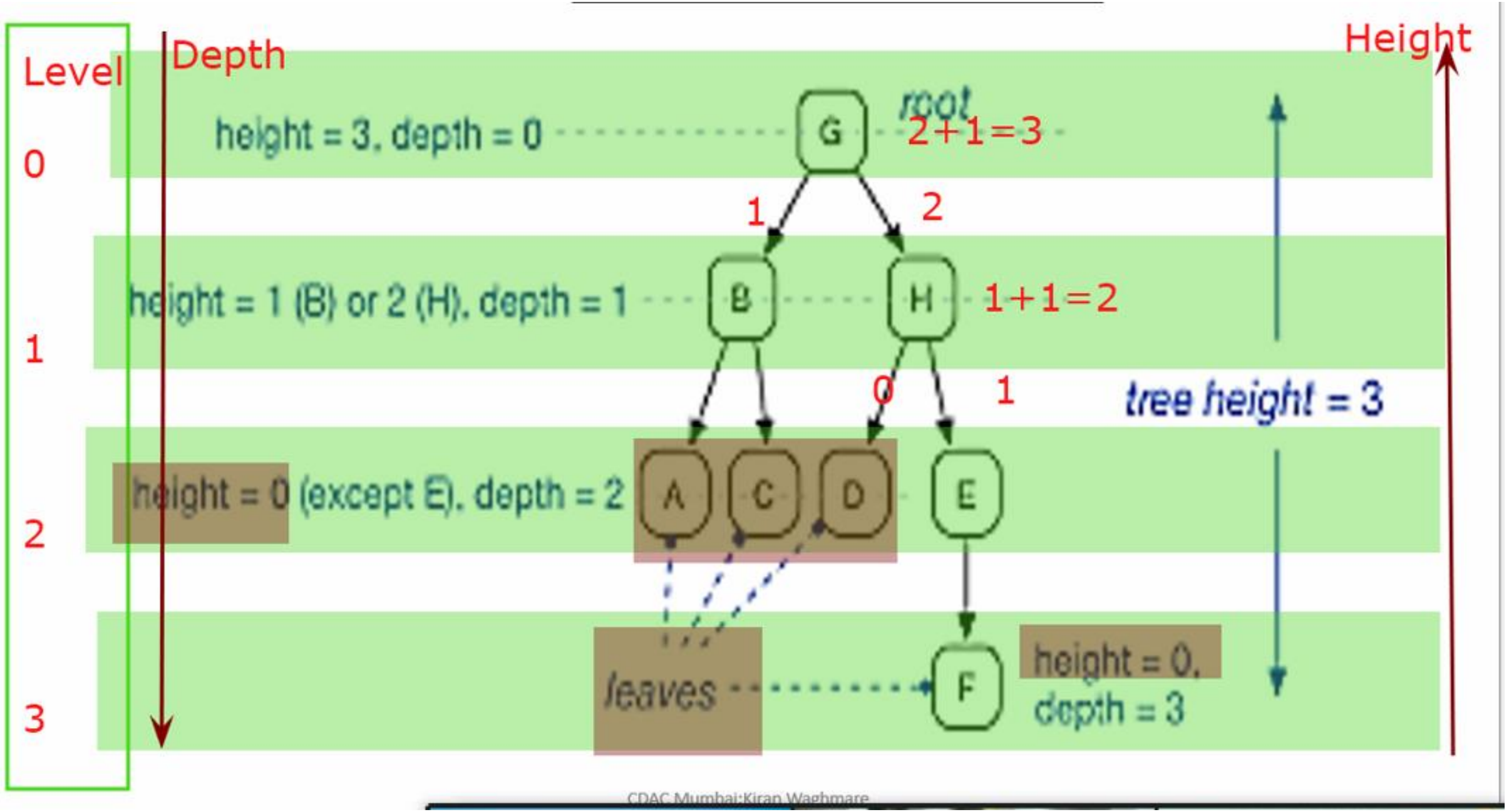

Nodes
A
Root Node
B
C
D
E
F
G
H
I
J
K
L
M
Left sub tree
sub tree
Right sub tree
Leaf Node
CDAC Mumbai:Kiran Waghmare


Root node
A
Intermediate Node
B
C
D
Leaf Node
E
F
G
H
I
J
K
L
M


Who can see what you share here? Recording On
Max no of nodes in BT
Level 0
node =1
Level 1
node 2
Level 2
node=4
Level 3
node =8
Binary Tree
CDAC Mumbai:Kiran Waghmare


Level 0
node =1
Level 1
node 2
Level 2
node=4
Level 3
node =8
Max no of node in BT at level 3=
$2^i = 2^3 = 8$ → i=0
i=1
$2^{i-1} = 2^2 = 4$ →

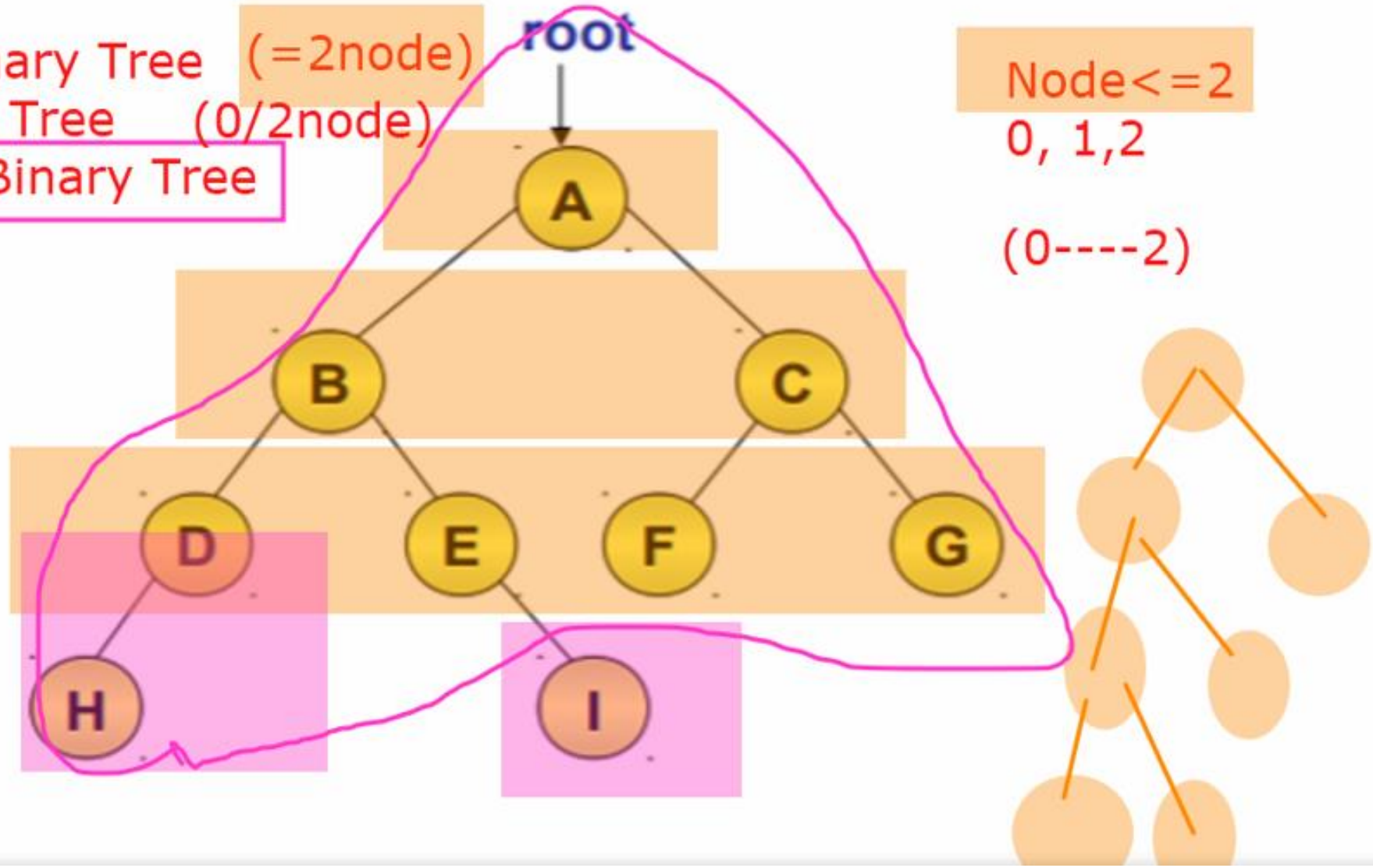

Level 0
node =1
Depth =0
Height = 3
Level 1
node 2
Depth =1
Height = 2
Level 2
node=4
Depth=2
Height = 1
Level 3
Depth=3
Height=0
node =8


Level
Depth
Height
0
1
2
3
height = 3, depth = 0
root
G
2+1=3
1
2
height = 1 (B) or 2 (H), depth = 1
B
H
1+1=2
0
1
tree height = 3
height = 0 (except E), depth = 2
A
C
D
E
leaves
F
height = 0,
depth = 3

Binary Tree:

1.Strictly Binary Tree (=2node)
2.Full Binary Tree (0/2node)
3.Complete Binary Tree

(0,1,2)



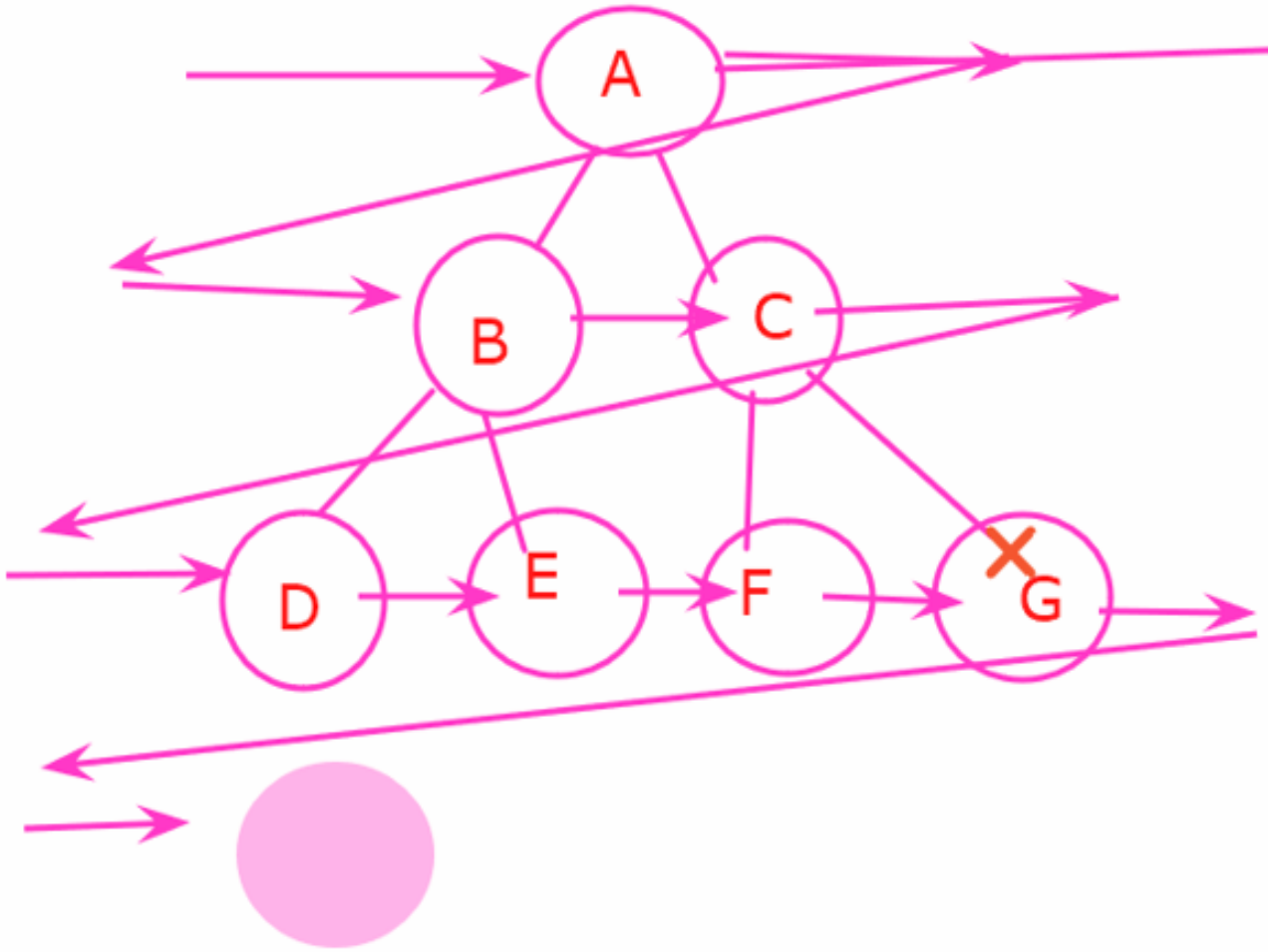
Node<=2
0, 1,2

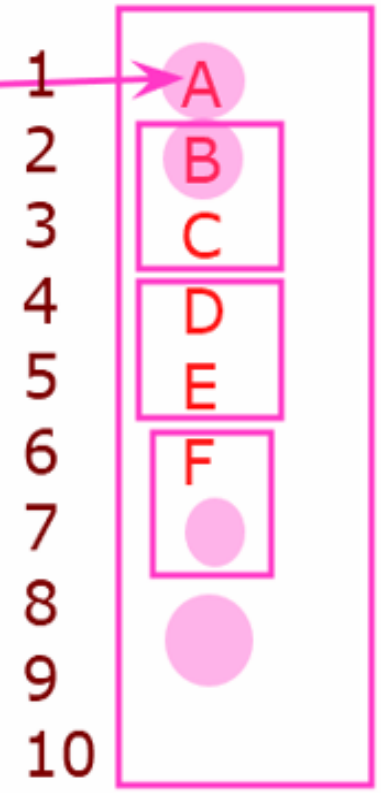
(0----2)

Representation of BT using

Kiran Waghm...

Heap Data Structure

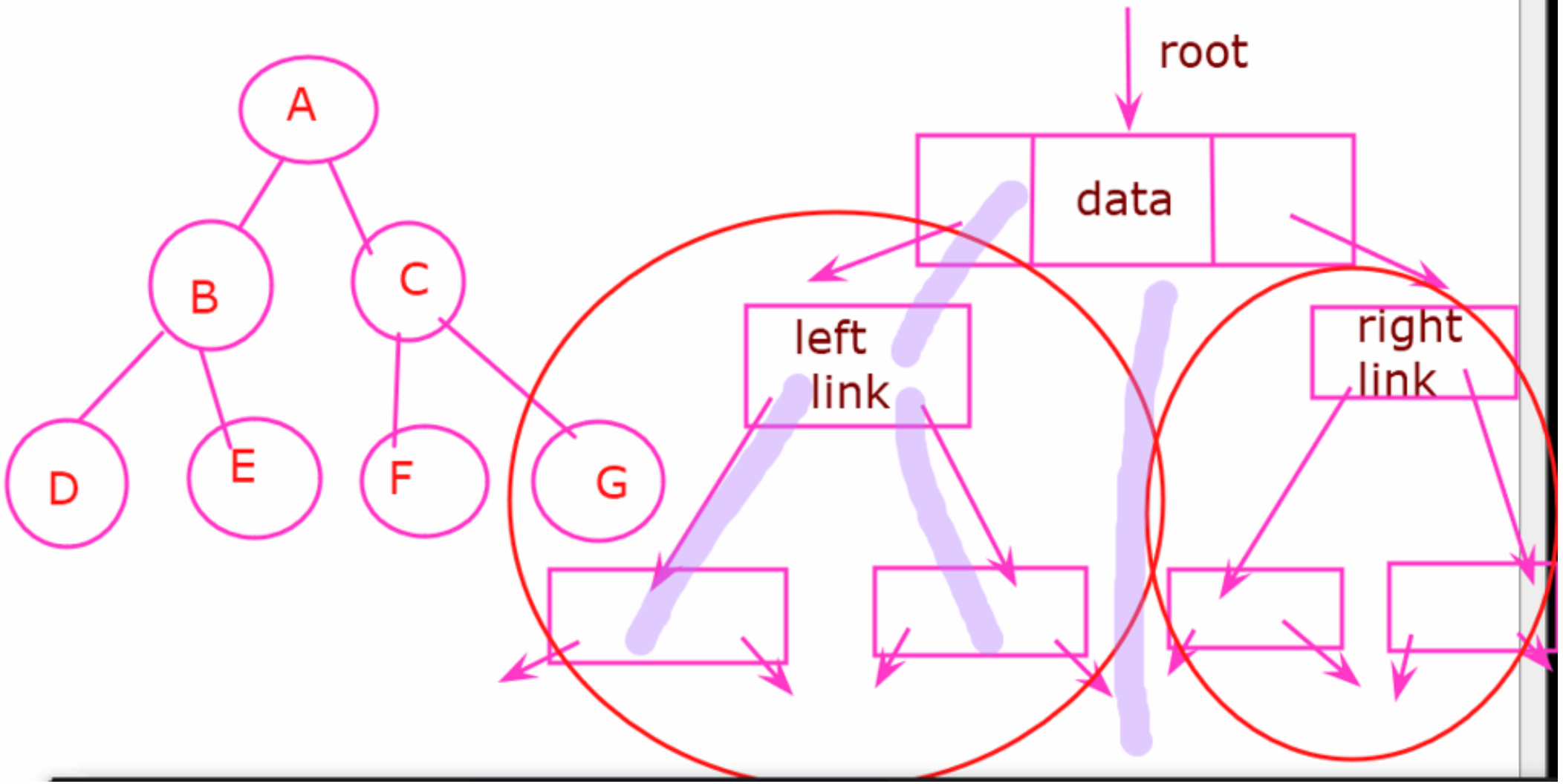




Parent : i
Lc:2i
RC:2i+1

Array Representation

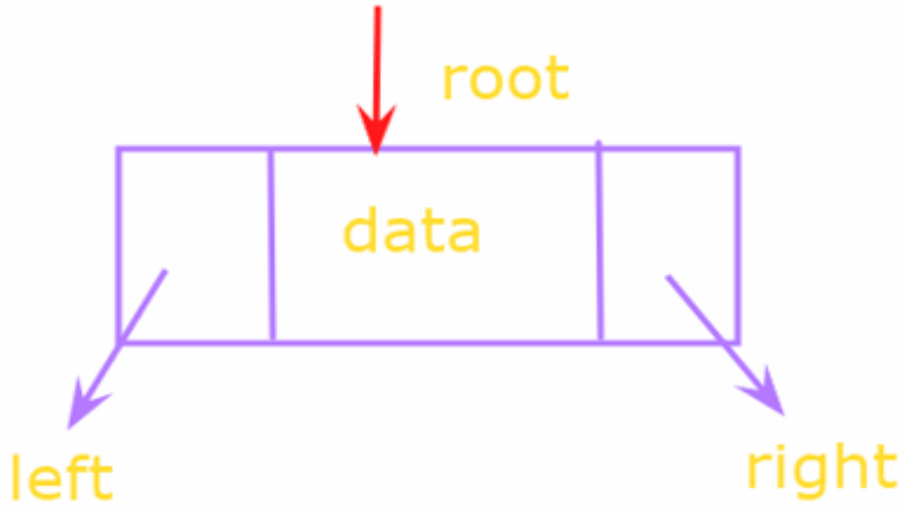
Representation of BT using Array



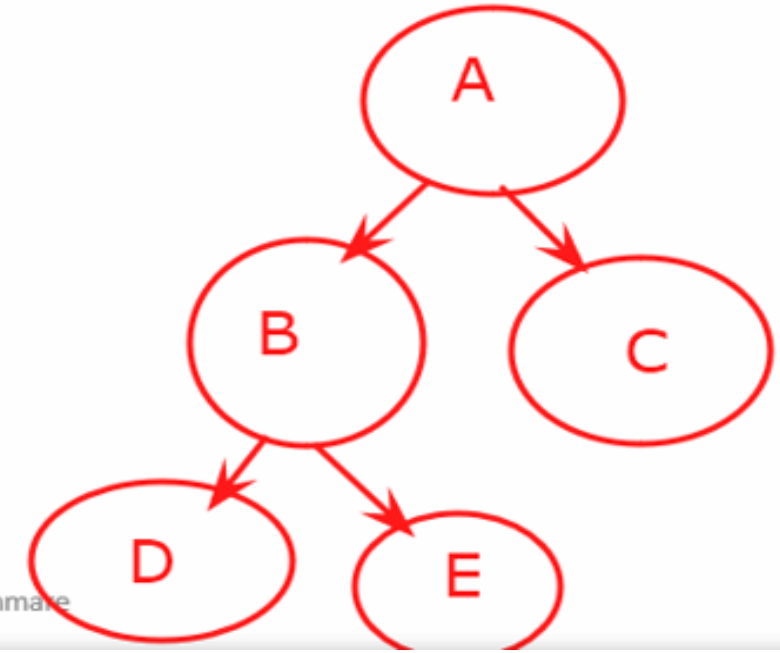
```
class Node
{
    int data;
    Node left, right;

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

Tree Traversal

1.Preorder : Root, LC, RC
2.Inorder : LC, Root, RC
3.Postorder : LC, RC, Root



Preorder:A,B,C
Inorder: B,A,C
Postorder:B,C,A



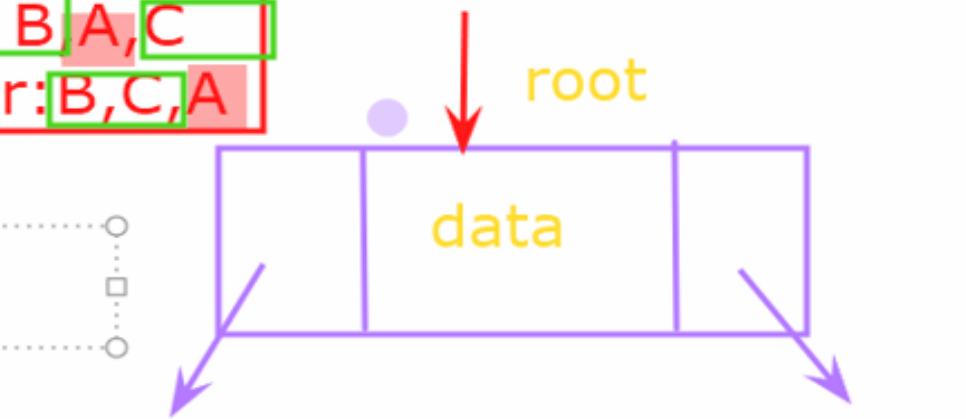
CDAC Mumbai:Kiran Waghmare

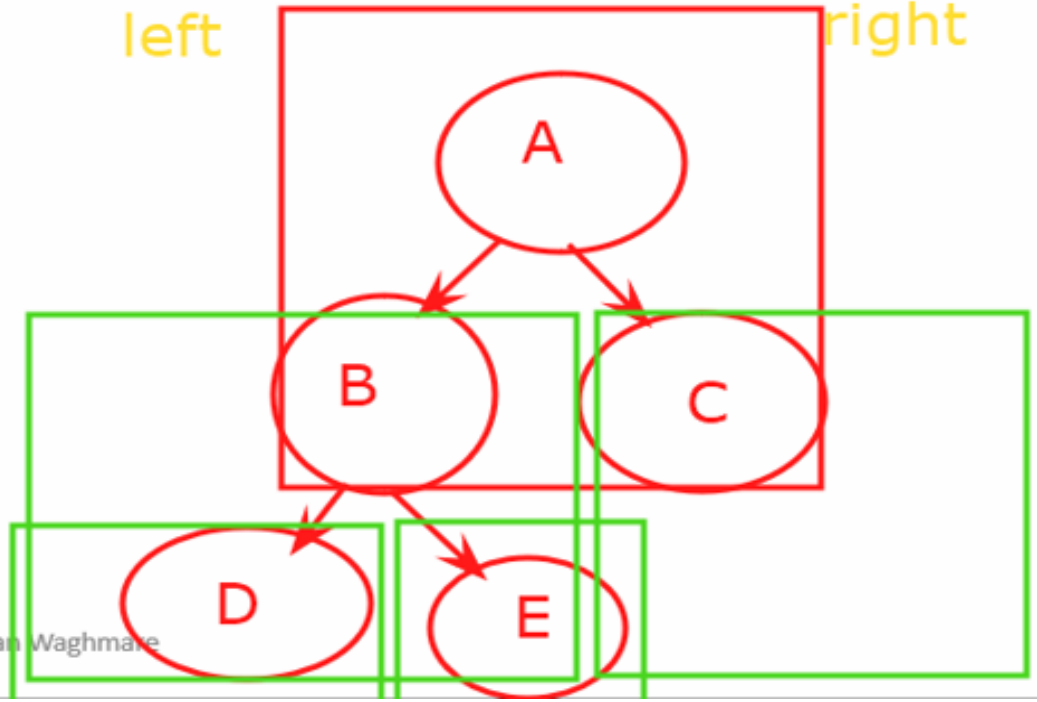
Tree Traversal

1.Preorder : Root, LC, RC
2.Inorder : LC, Root, RC
3.Postorder : LC, RC, Root

```
void preorder(Node n){
    if(node == null)
        return;
    System.out.println("node.data");
     preorder(node.left);
     preorder(node.right);
 }
```

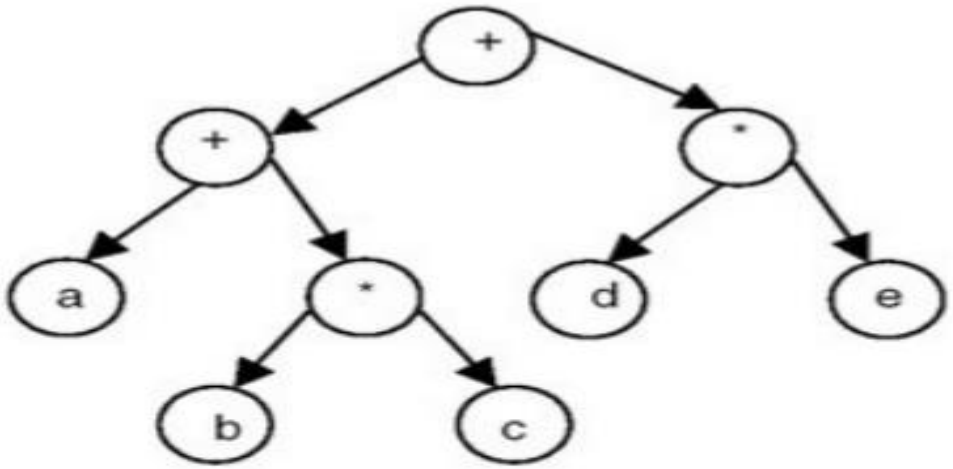
Preorder:A,B,C
Inorder: B,A,C
Postorder:B,C,A





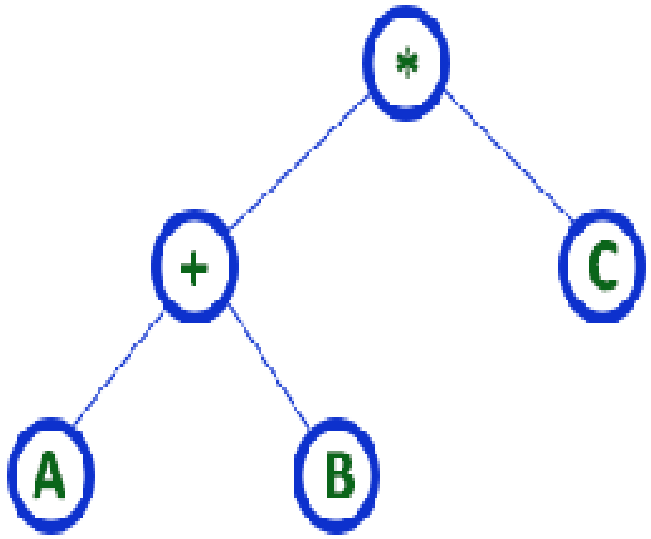
Expression Binary Tree Traversal

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.

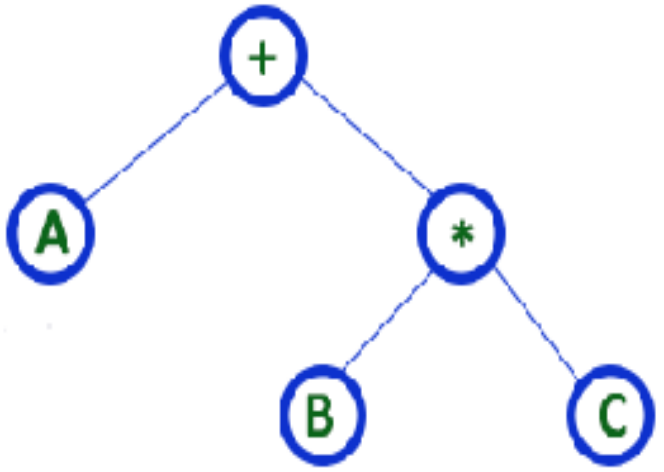


Inorder : a + b * c + d * e
postorder : abc*+de*+

Strictly binary tree data structure is used to represent mathematical expressions.



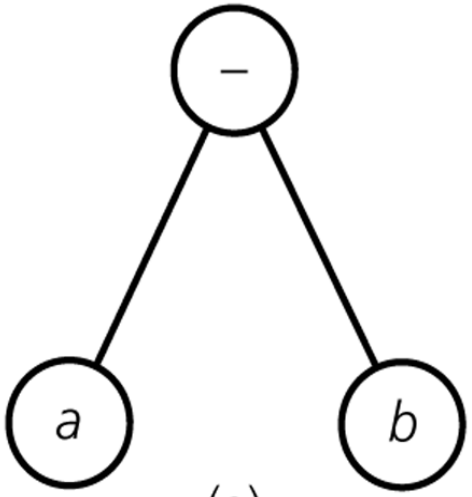
(A + B) * C



A + B * C

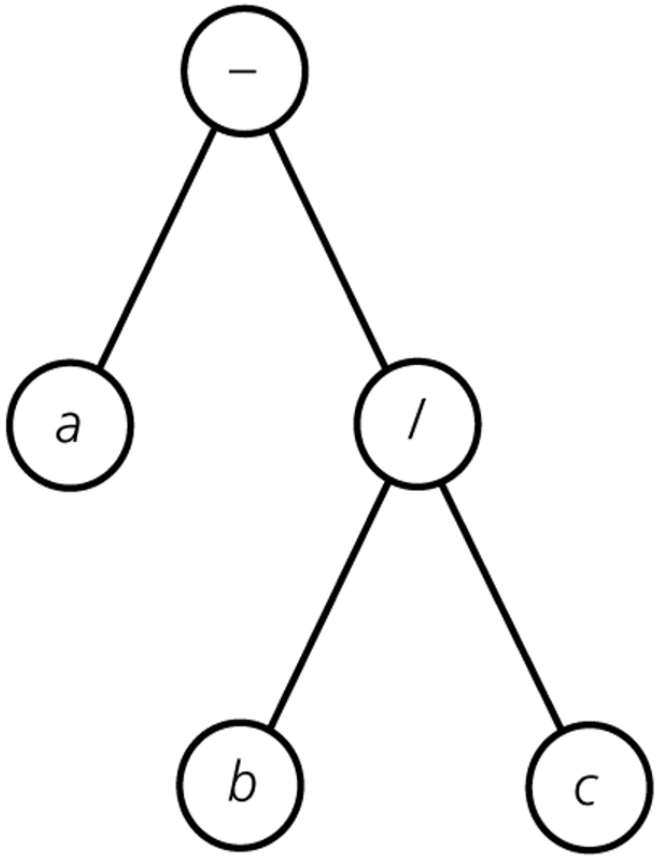
Binary Tree – Representing Algebraic Expressions

$a - b$



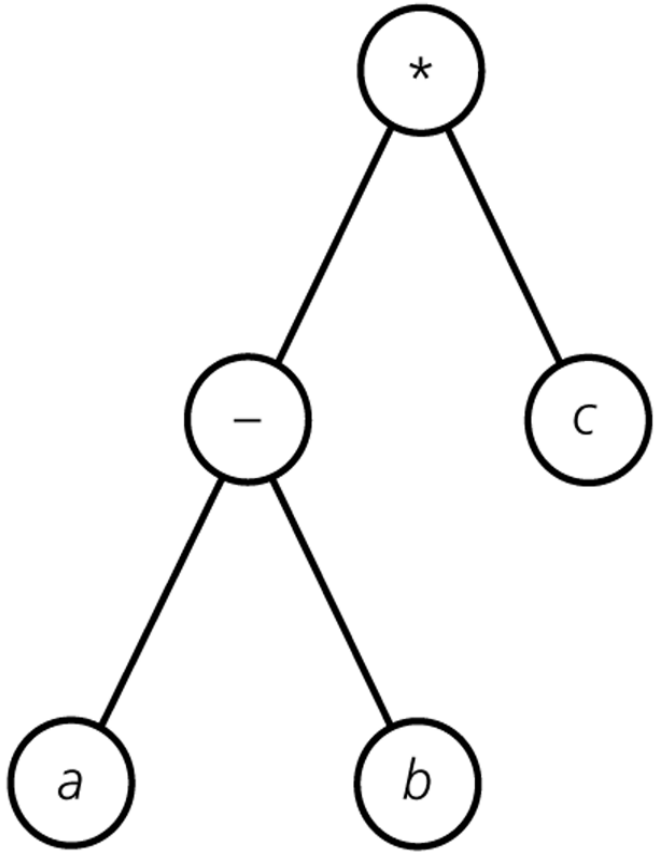
(a)

$a - b / c$

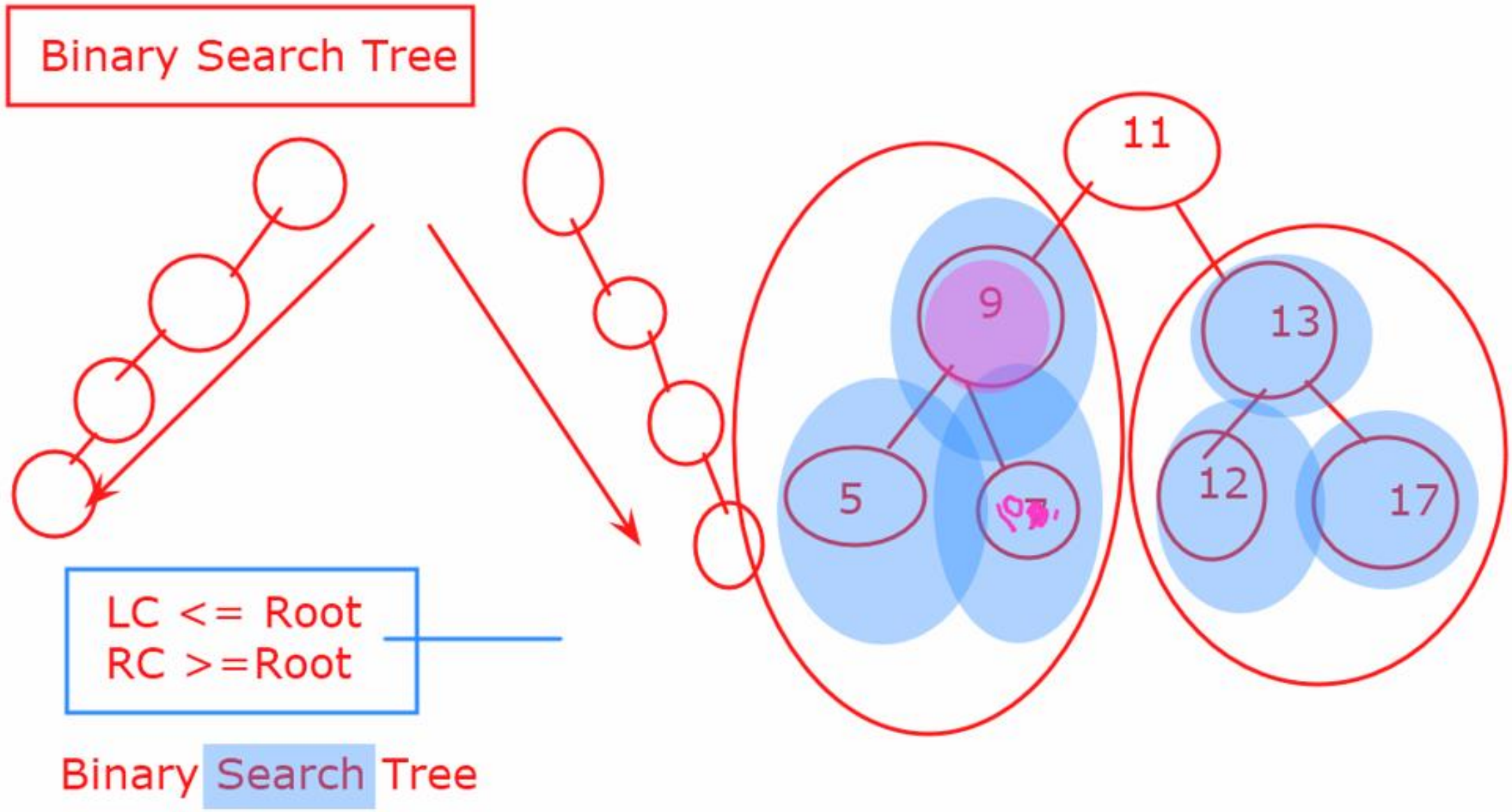


(b)

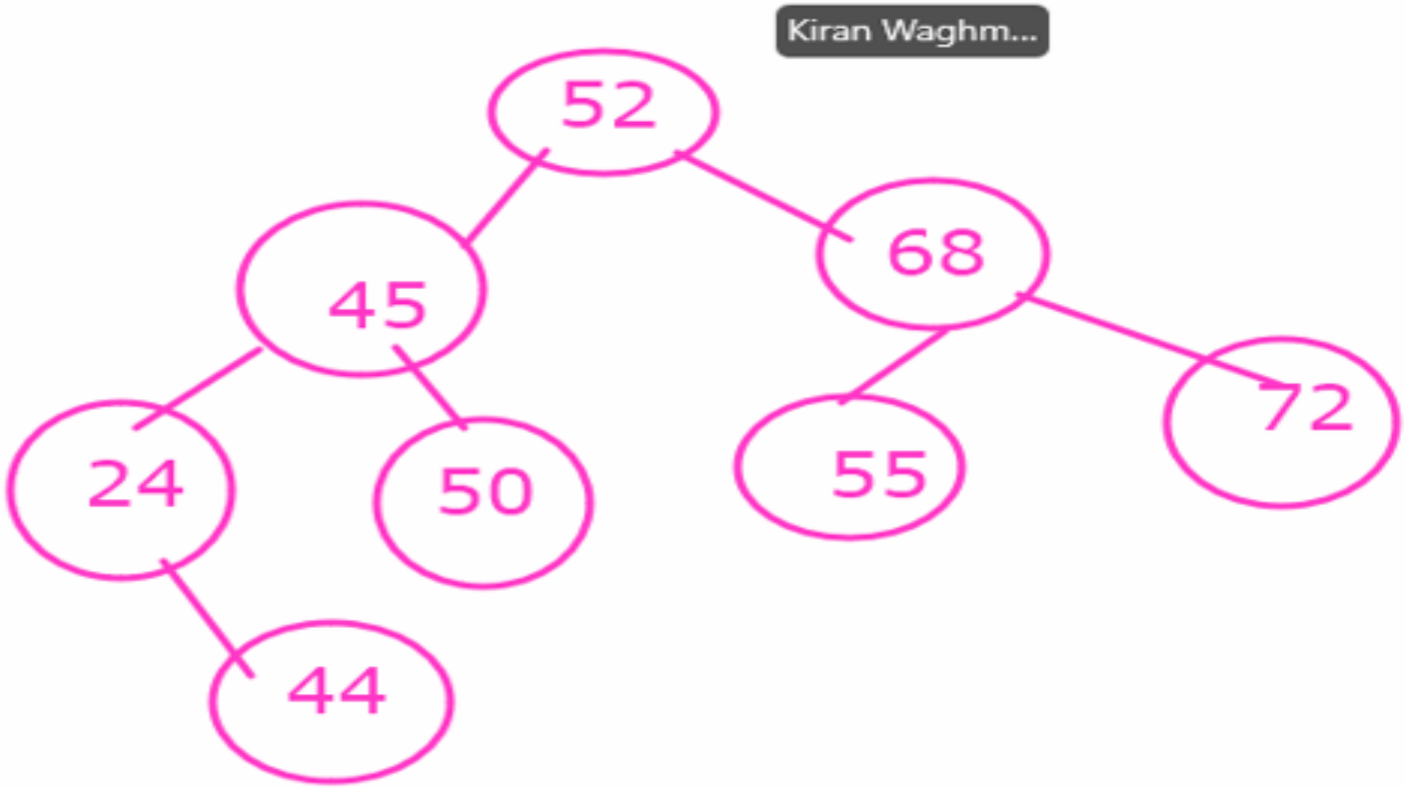
$(a - b) * c$



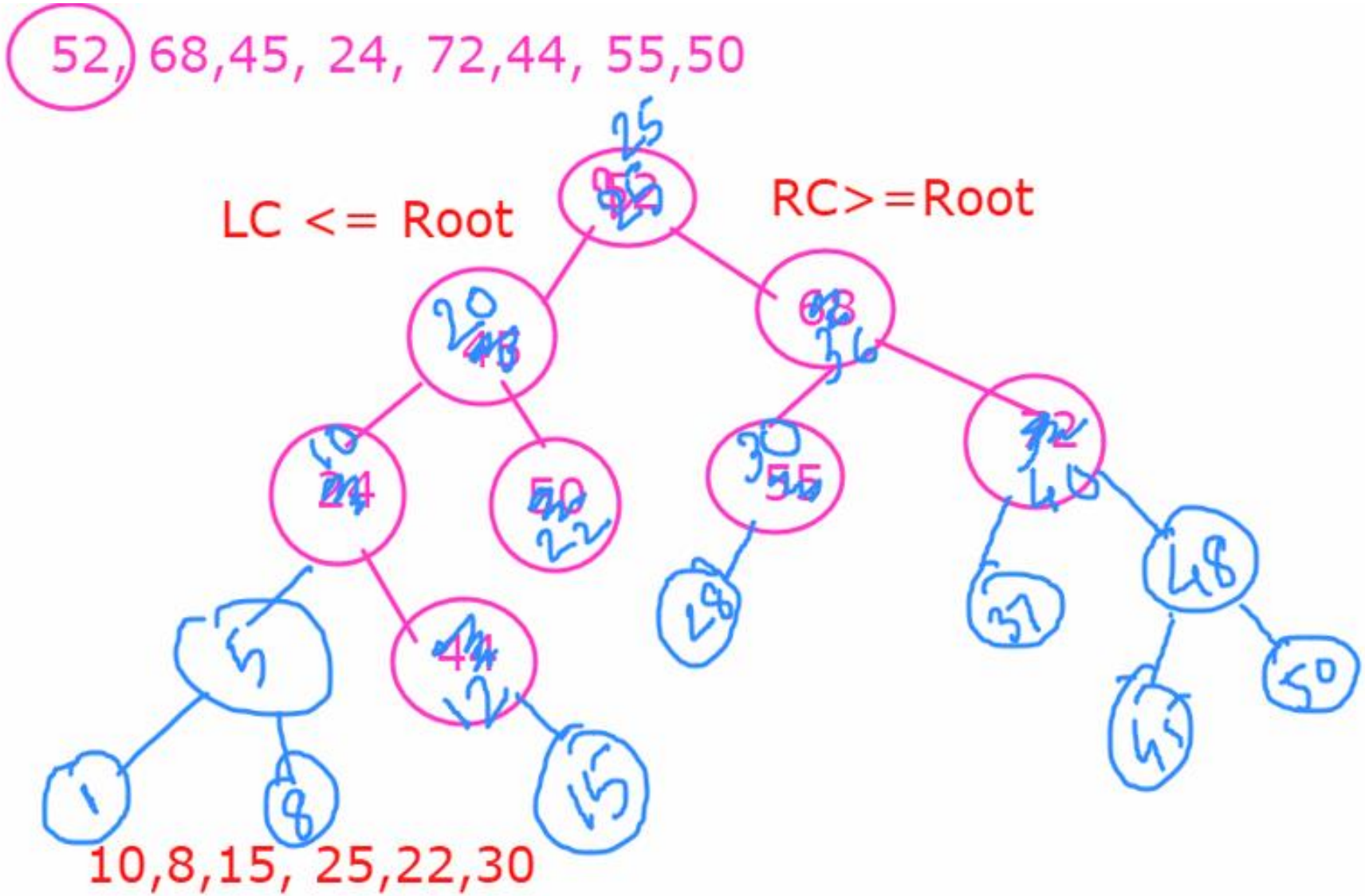
(c)


Binary Search Tree
LC <= Root
RC >=Root
Binary Search Tree
11
9
13
5
12
17

52, 68,45, 24, 72,44, 55,50

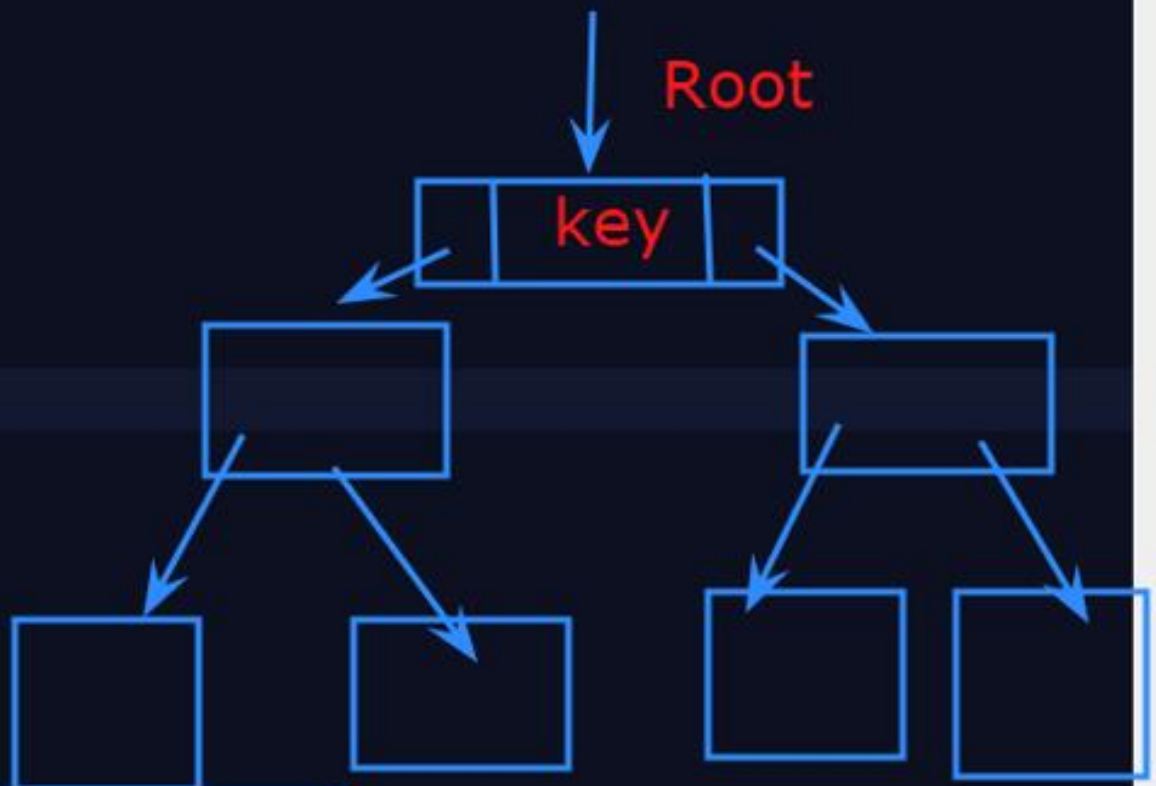


10,8,15, 25,22,30

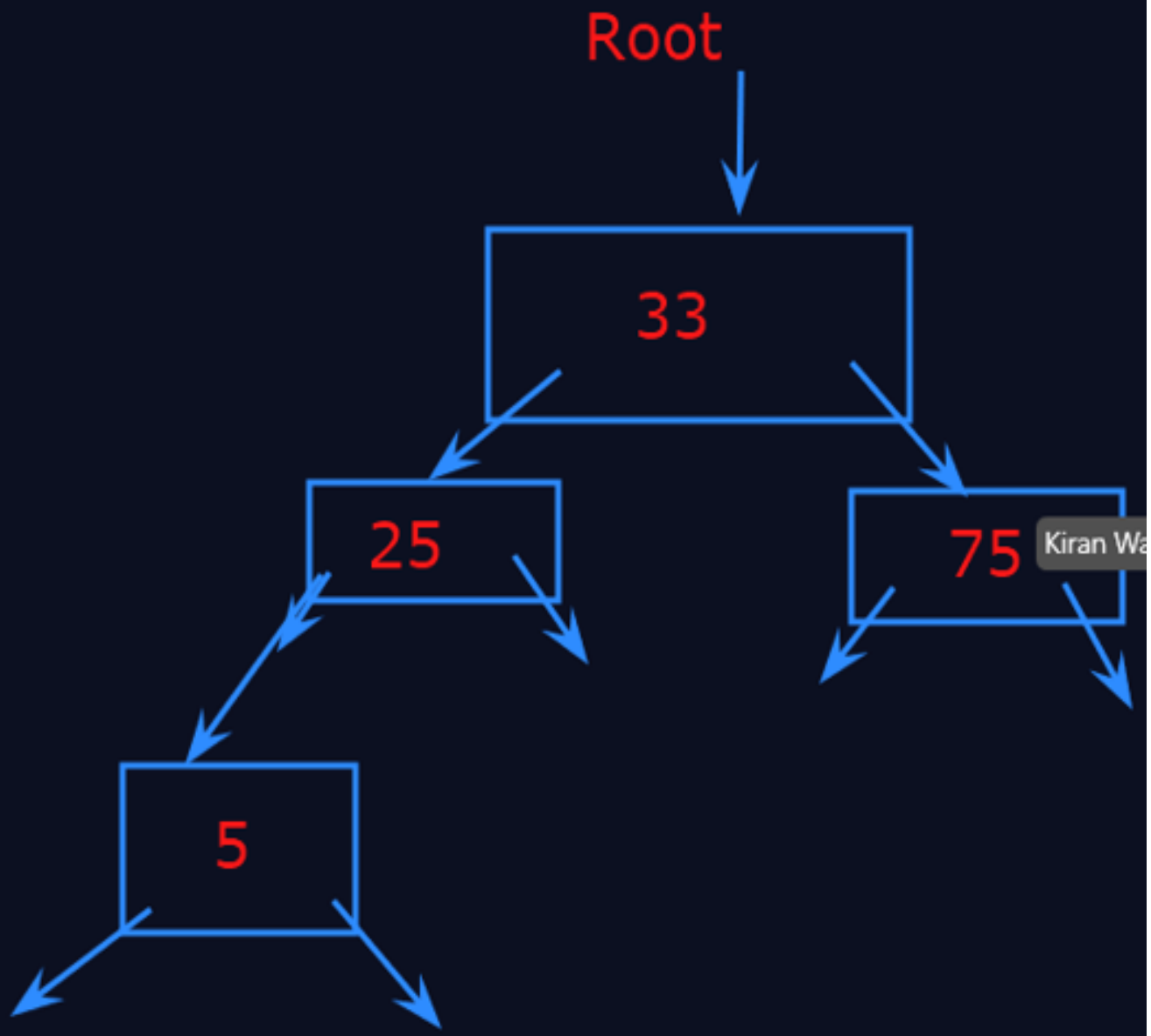


25,36,40,20,10,30,22,5,28,48,37,12,1,15,8,50,45

```
}

}


Node insertdata(Node root, int key)
{
    if(root == null)
    {
        root =new Node(key);
        return root;
    }

    if(key <= root.data)
        root.left= insertdata(root.left, key);
    else
        root.right=insertdata(root.right, key);

    return root;
}
```



Deletion:
Case 1: Leaf node
Case 2: Single node
Case 3: Parent node



Thanks