Binary Trees Class - 1

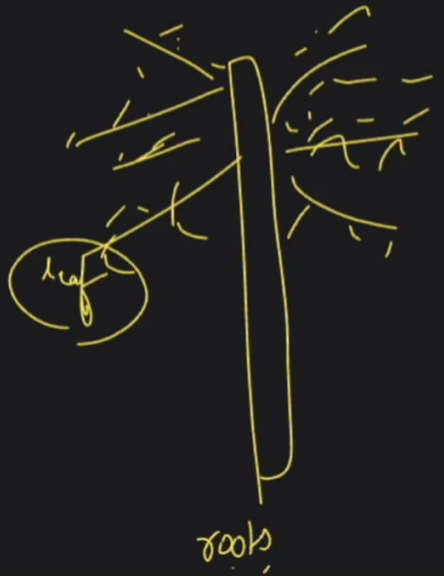
Special class

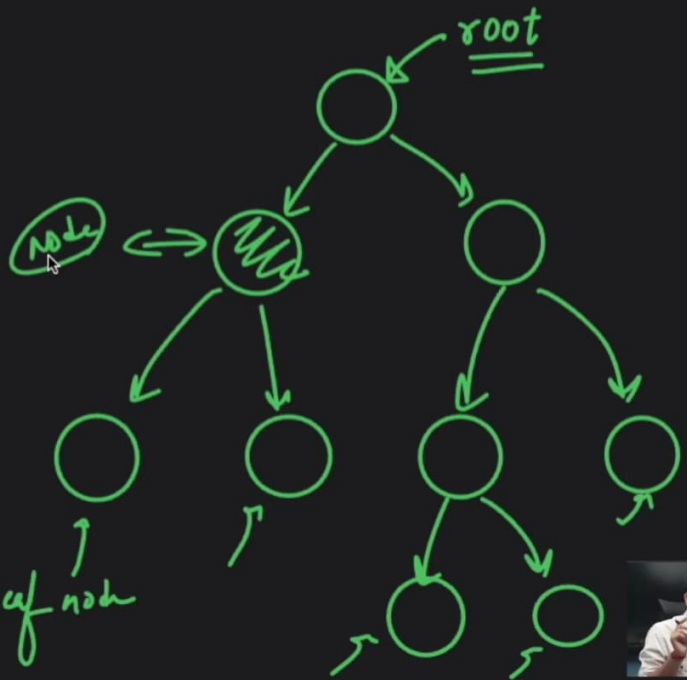
learn.codehelp.in
zeeshankhankk907@gmail.com
+919811718169

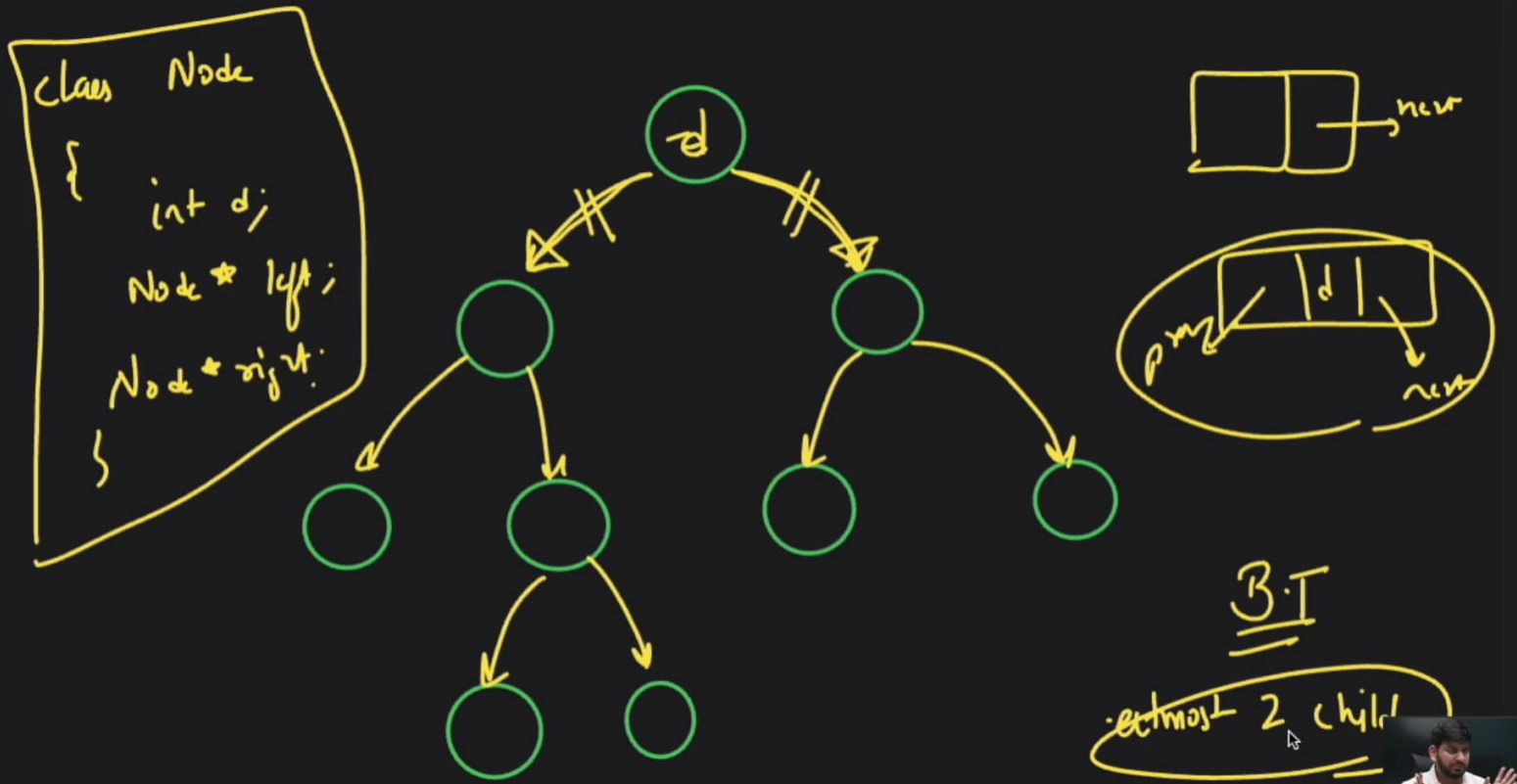
Love Babbar • Nov 30, 20

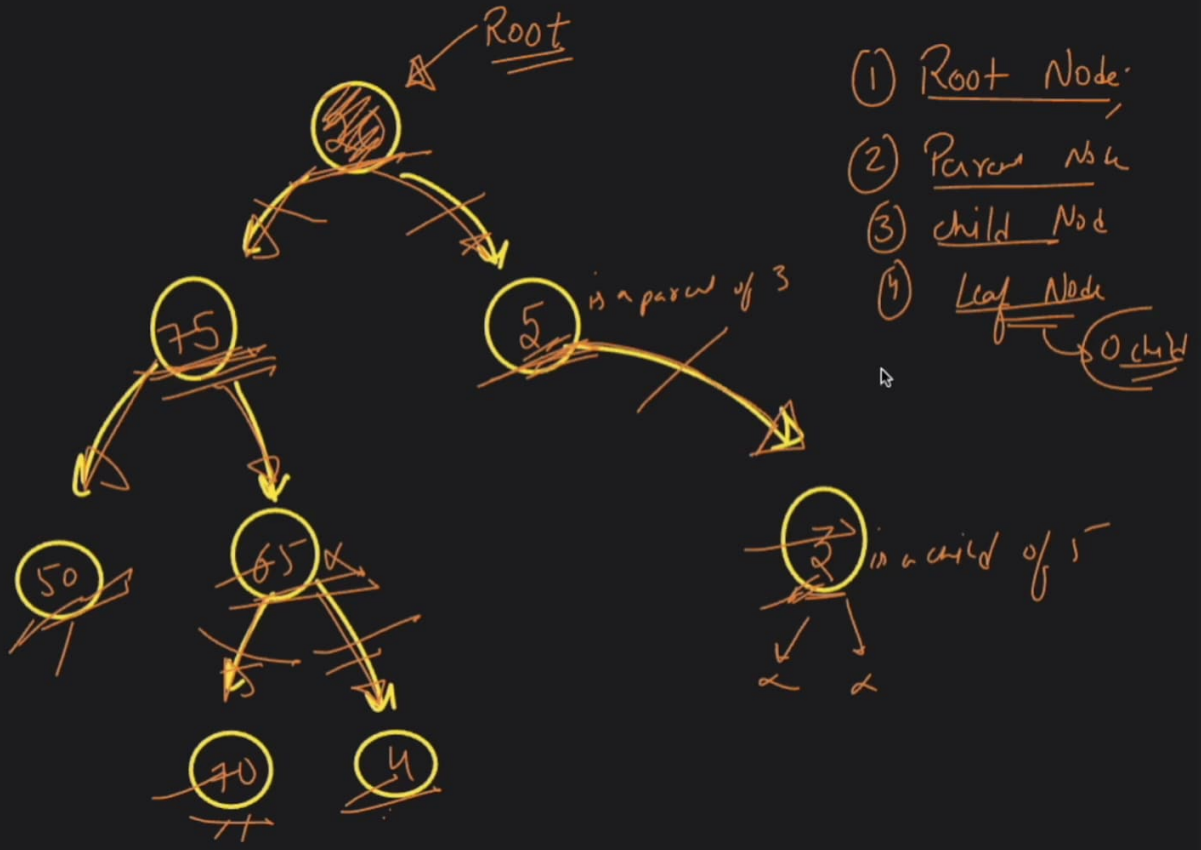
Trees

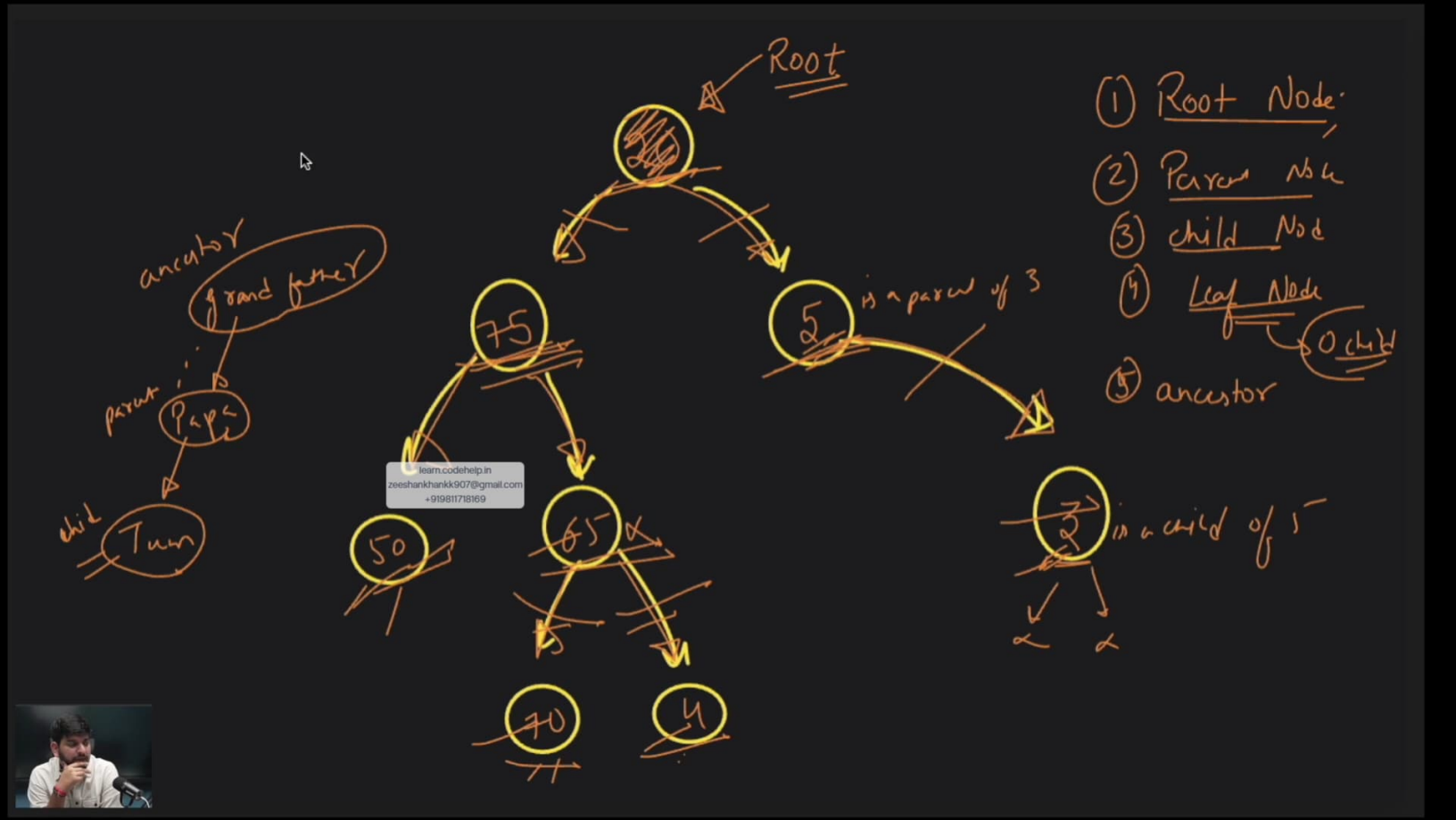
non-linear D.S

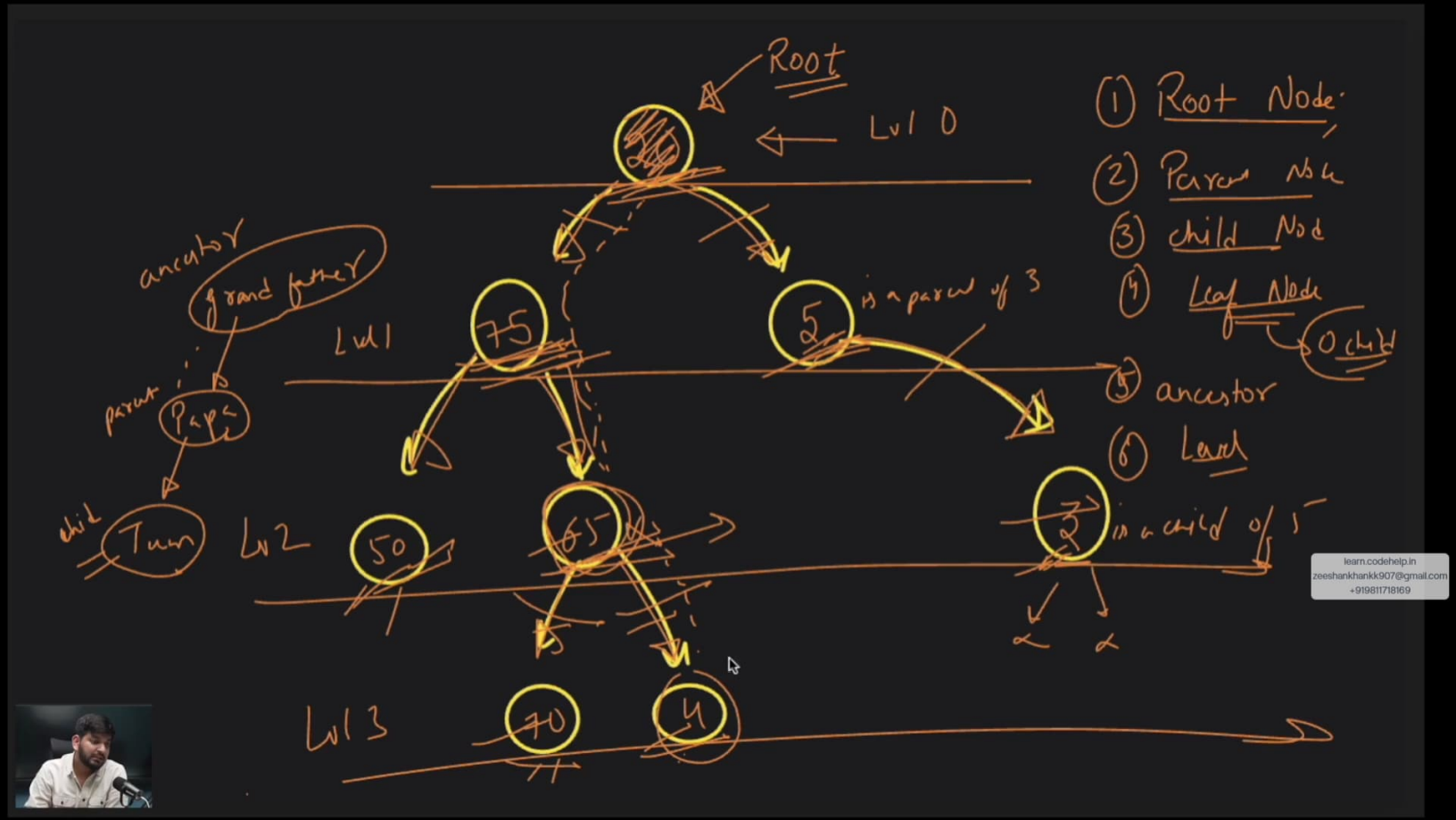


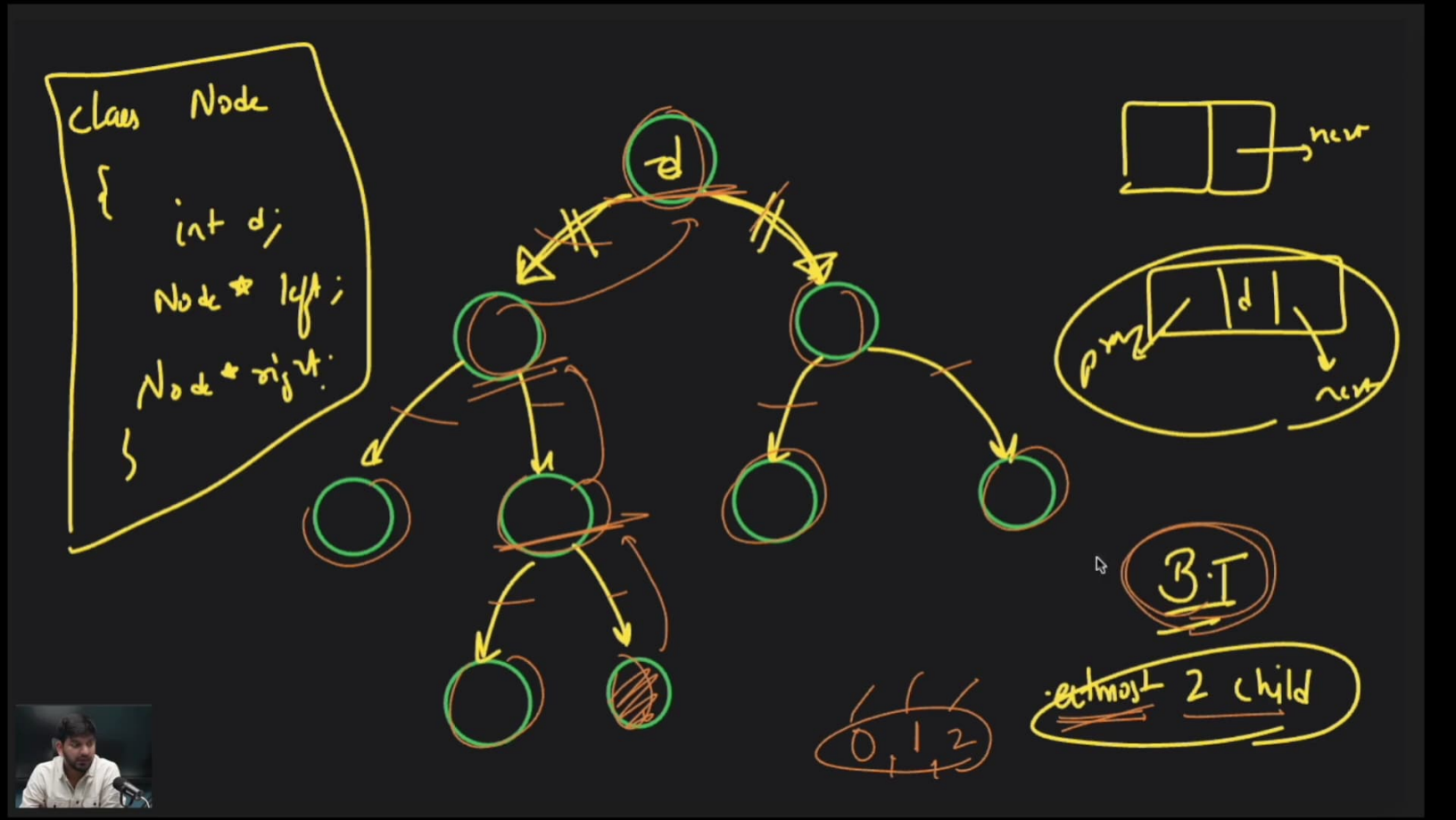


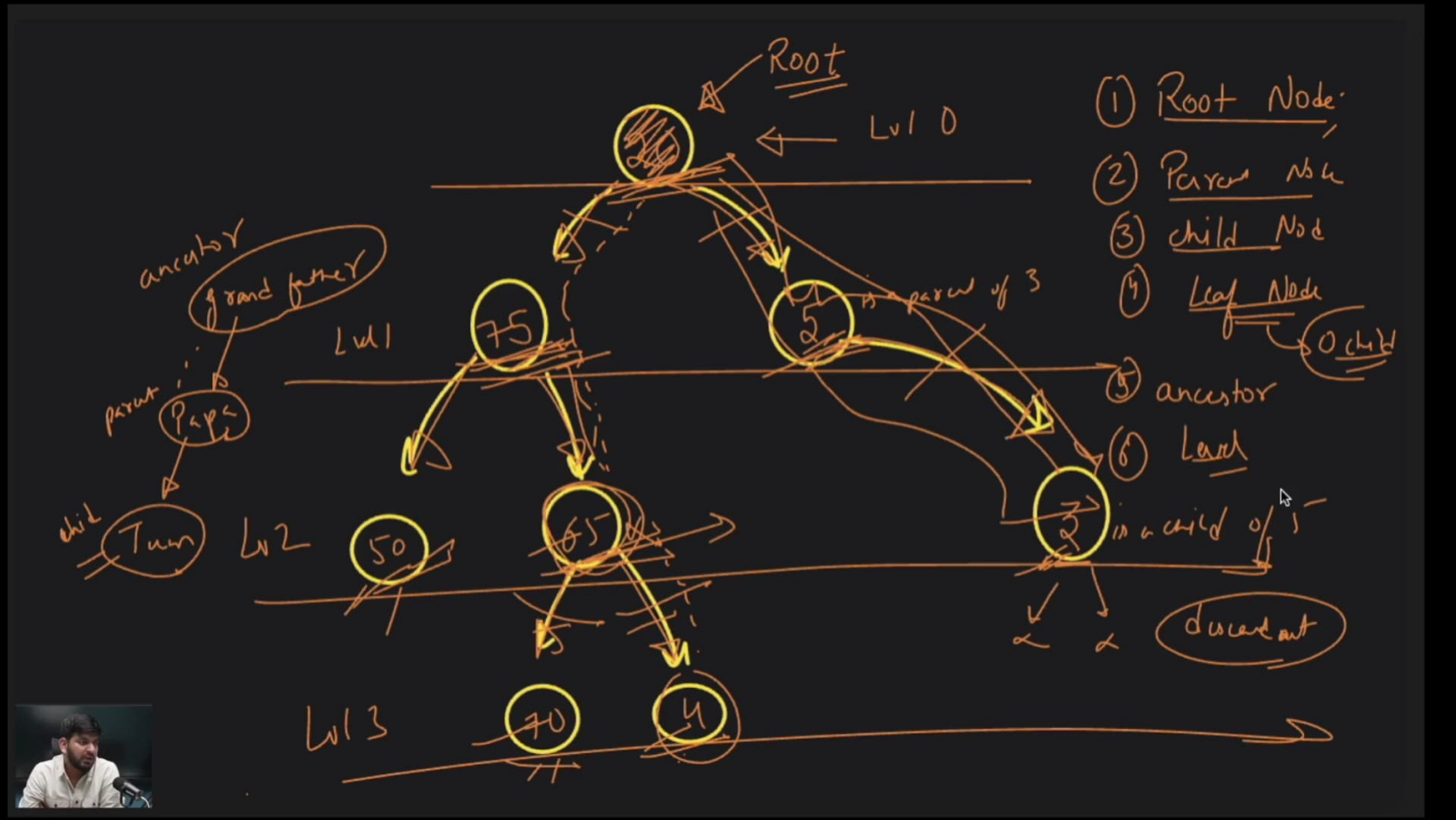

class Node
{
int d;
Node * left;
Node * right;
}
d
next
d
prev
next
3.I
almost 2 child

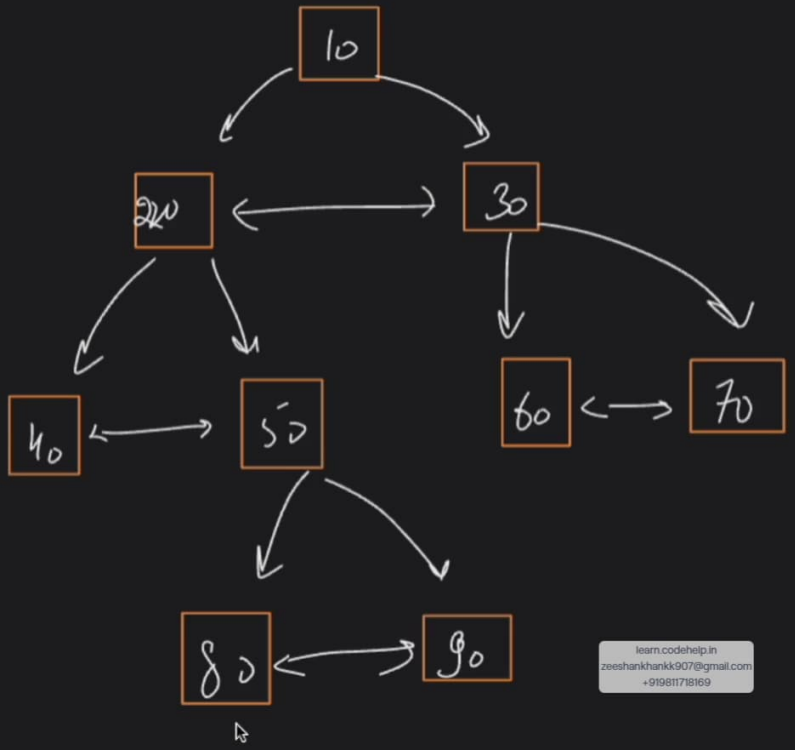

Root
75
5
is a parent of 3
50
65
3
70
4
is a child of 5
(1) Root Node
(2) Parent Node
(3) child Node
(4) Leaf Node
0 child

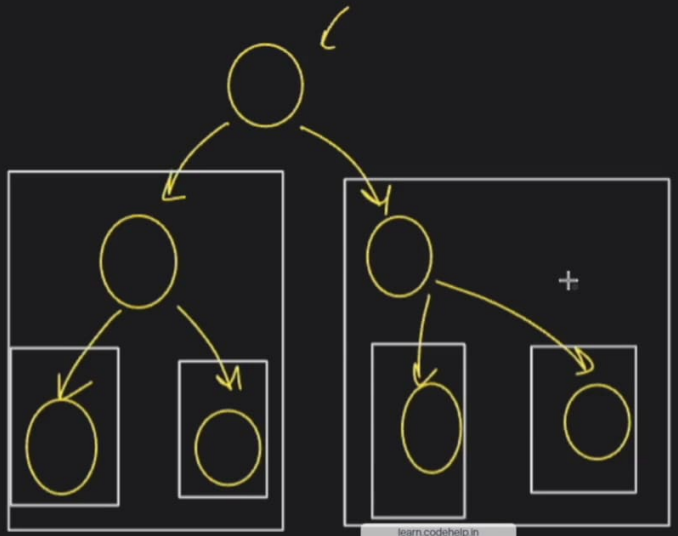

Root
(1) Root Node
(2) Parent Node
(3) child Node
(4) Leaf Node
0 child
(5) ancestor
ancestor
grand father
parent
Papa
child
Tum
75
5
is a parent of 3
50
65
3
70
4
is a child of 5
learn.codehelp.in
zeeshankhankk907@gmail.com
+919811718169

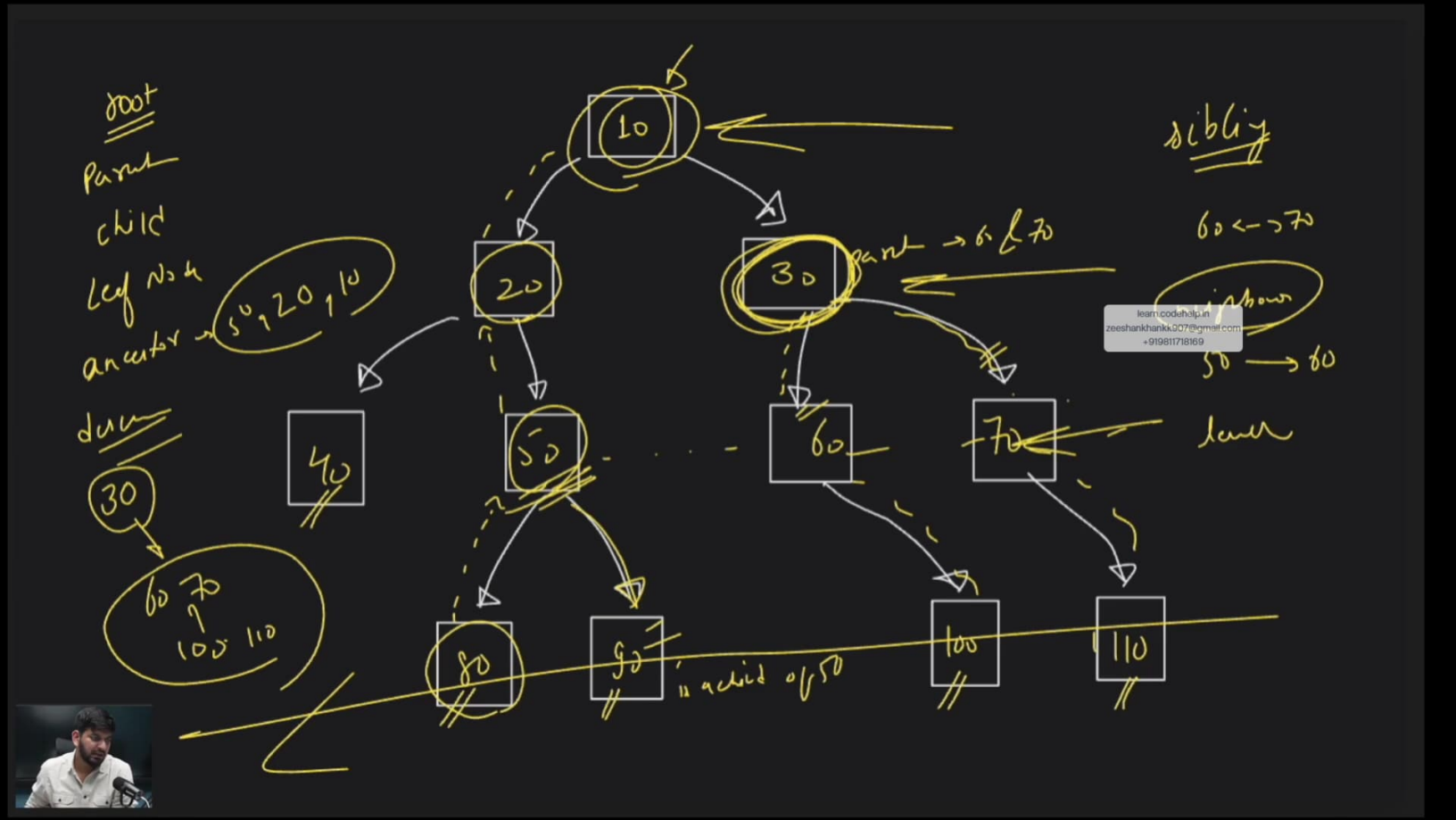

Root
Lvl 0
(1) Root Node
(2) Parent Node
(3) child Node
(4) Leaf Node
0 child
ancestor
grand father
Lvl 1
75
5
is a parent of 3
parent
Papa
(5) ancestor
(6) Level
child
Tuna
Lvl 2
50
65
27
is a child of 5
Lvl 3
70
4
learn.codehelp.in
zeeshankhankk907@gmail.com
+919811718169


class Node
{
int d;
Node * left;
Node * right;
}
d
next
d
prev
new
3.1
0, 1, 2
2 child

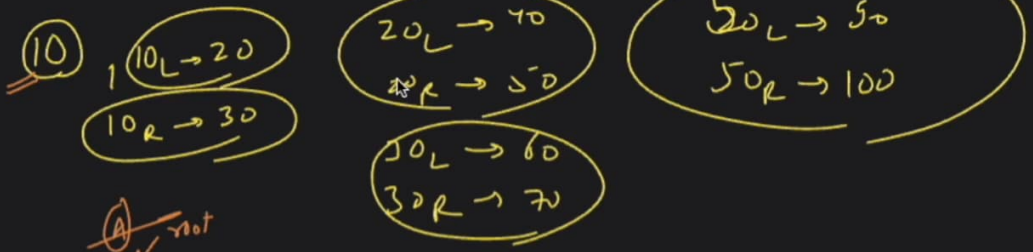

Root
Lvl 0
(1) Root Node
(2) Parent Node
(3) child Node
(4) Leaf Node
0 child
ancestor
grand father
parent
Papa
child
Tum
Lvl 1
75
2
is parent of 3
(5) ancestor
(6) Level
Lvl 2
50
65
3
is a child of 1
descendant
Lvl 3
70
4



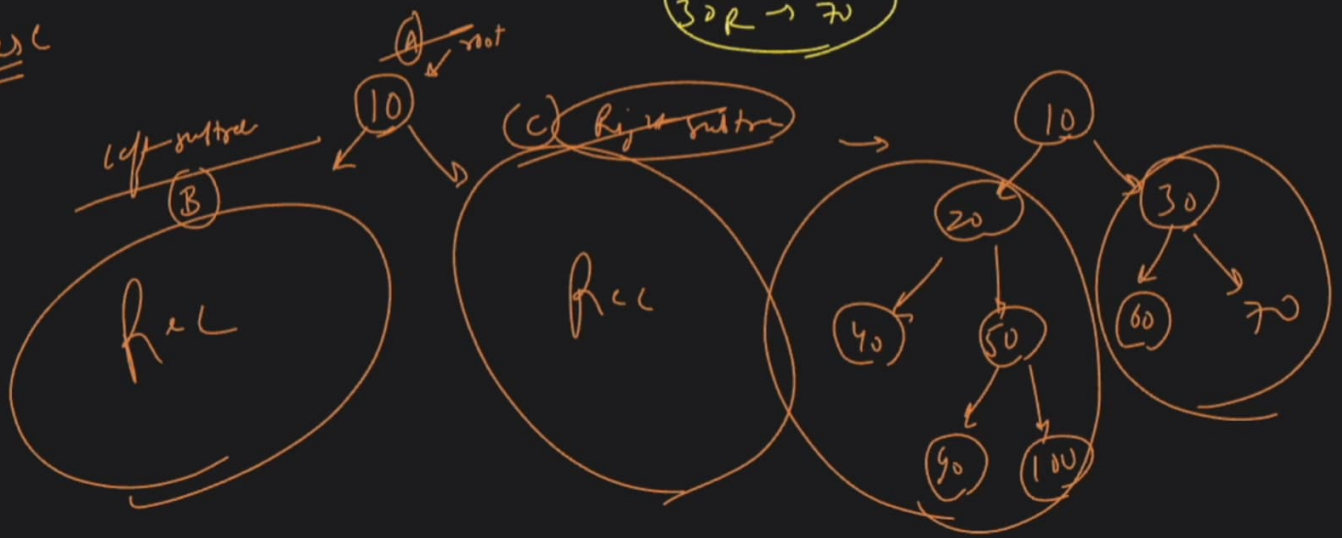


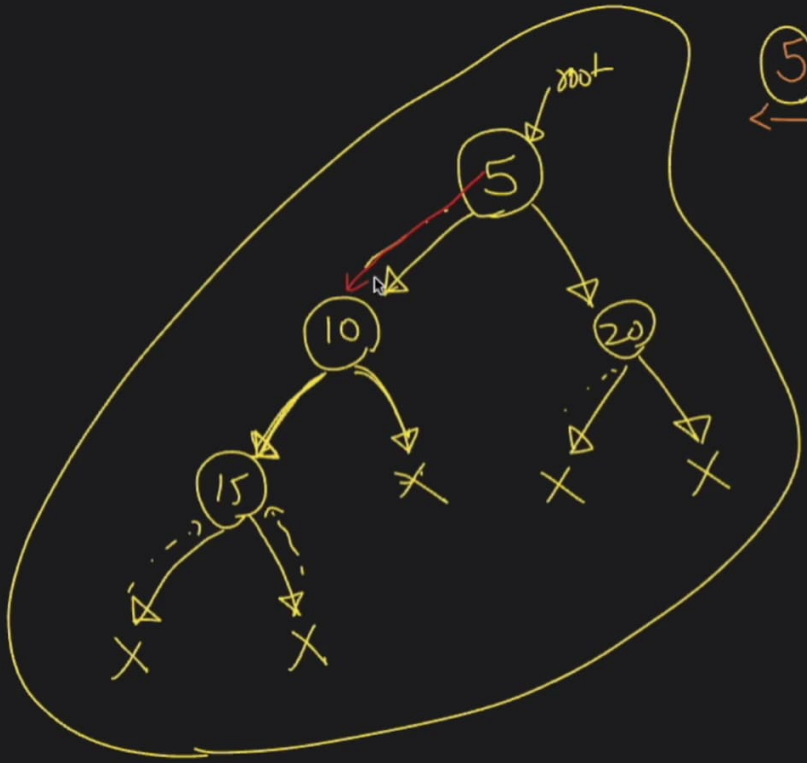

root
Parent
child
Leaf Node
ancestor → 50, 20, 10
desc
30
60 70 100 110
10
20
30
40
50
60
70
80
90
100
110
sibling
60 <-> 70
parent → of 70
50 → 60
level
learn.codehelp.in
zeeshankhankk007@gmail.com
+919811718169

→ Implement :-



1 case



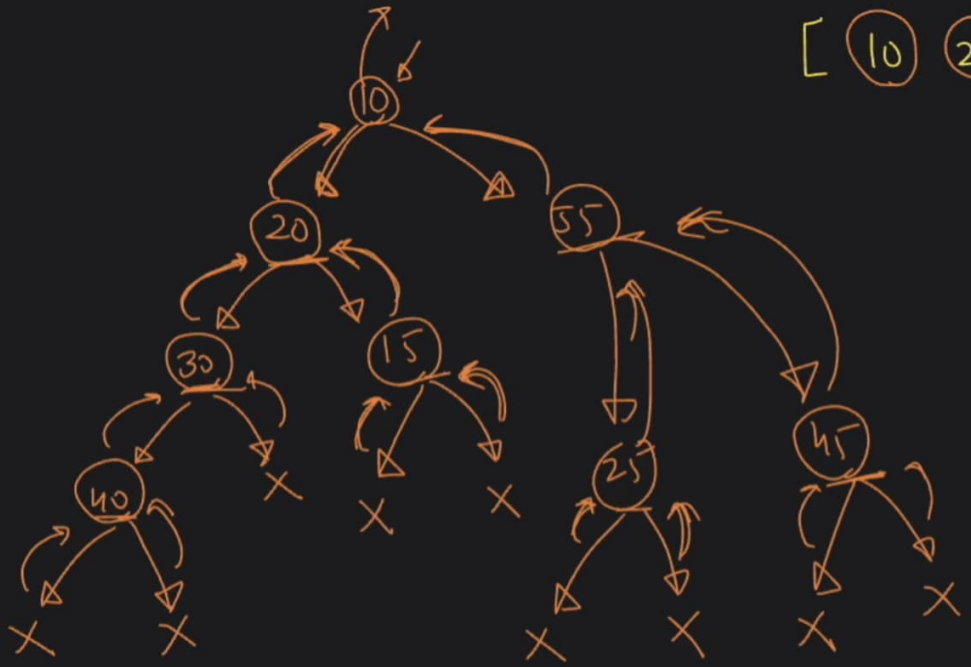


5 10 15 -1 -1 -1 20 -1 -1

Note

left subtree

right subtree

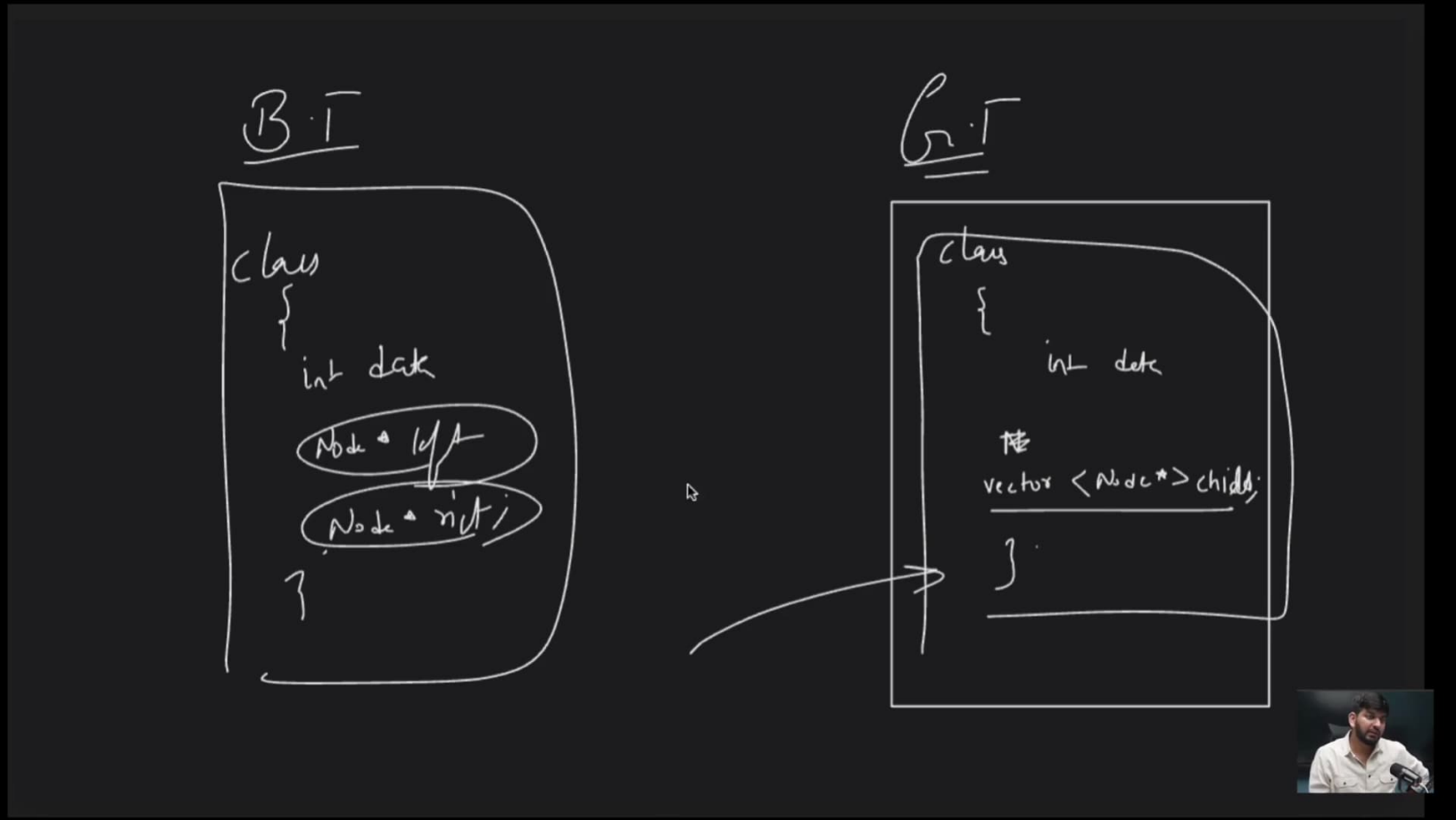


[10 20 30 40 -1 -1 -1 15 -1 -1 55 25 -1 -1 45 -1 -1]

[10 20 30 40 -1 -1 -1 15 -1 -1 55 25 -1 -1 45 -1 -1]

Binary Tree

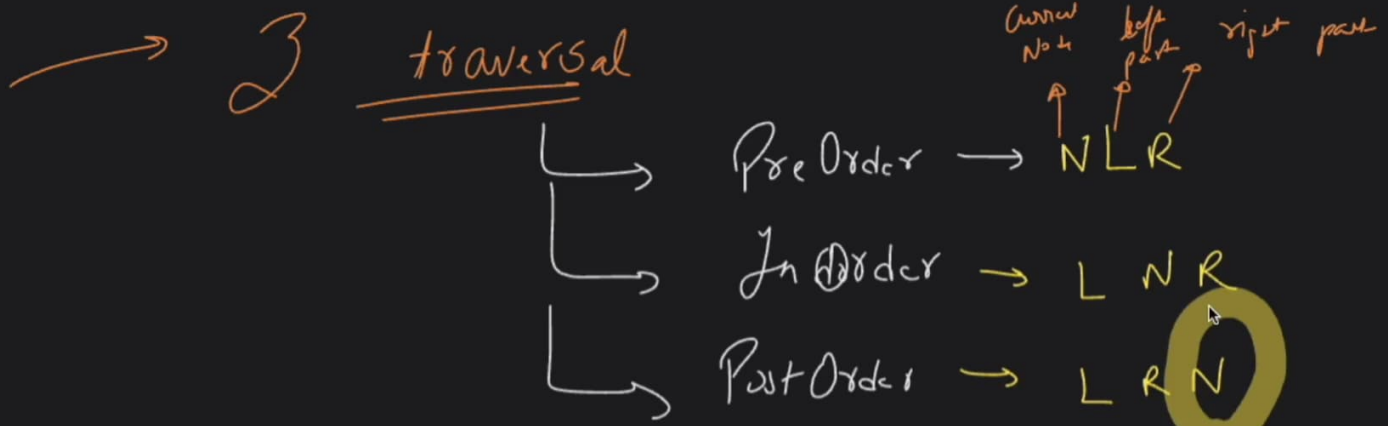
Generic Tree → n Child

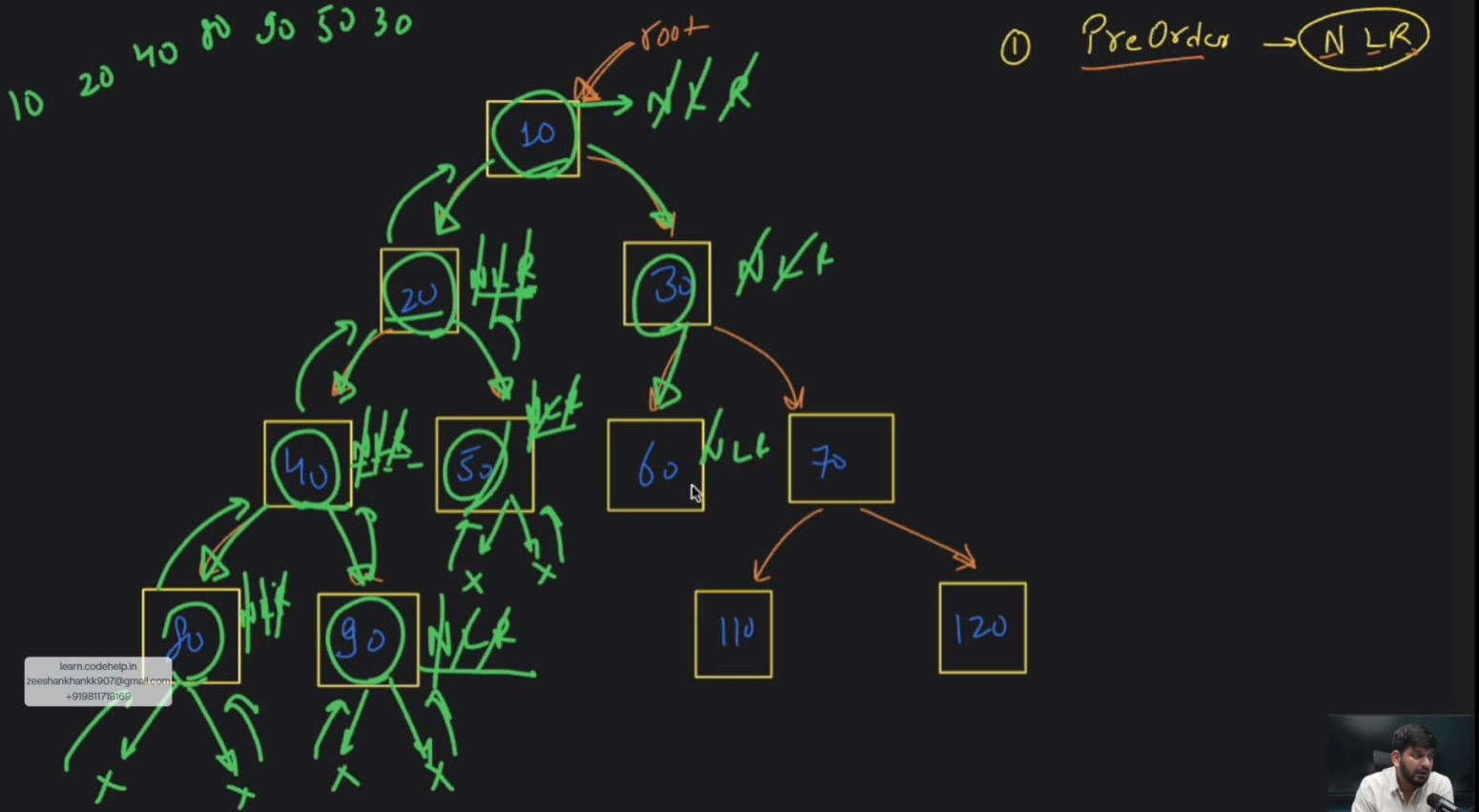

B.T
class
{
int data
Node * left
Node * right
}
G.T
class
{
int data
vector <Node*> childs;
}

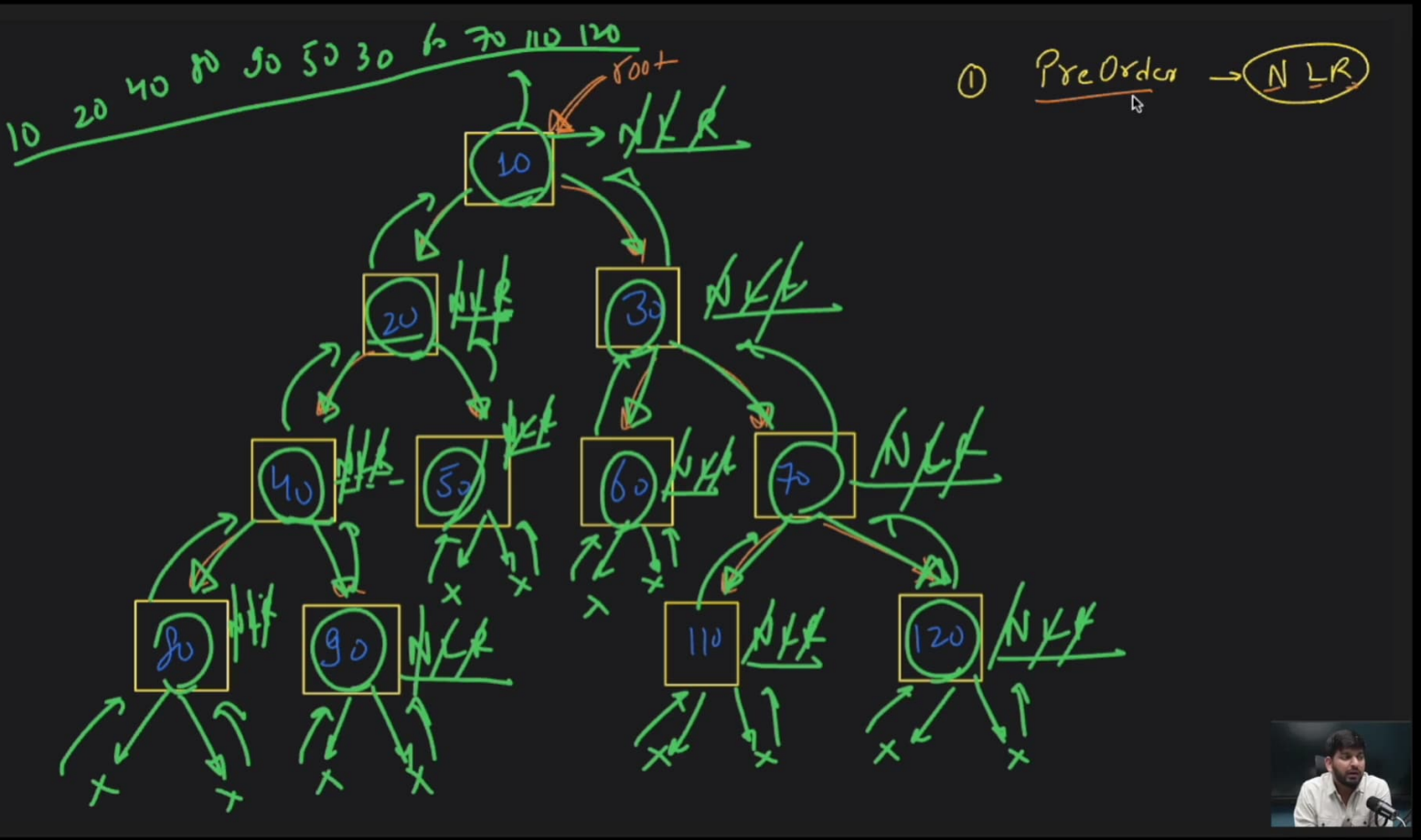
[10 20 30 40 -1 -1 -1 15 -1 -1 55 2 -1 -1 45 -1 -1]

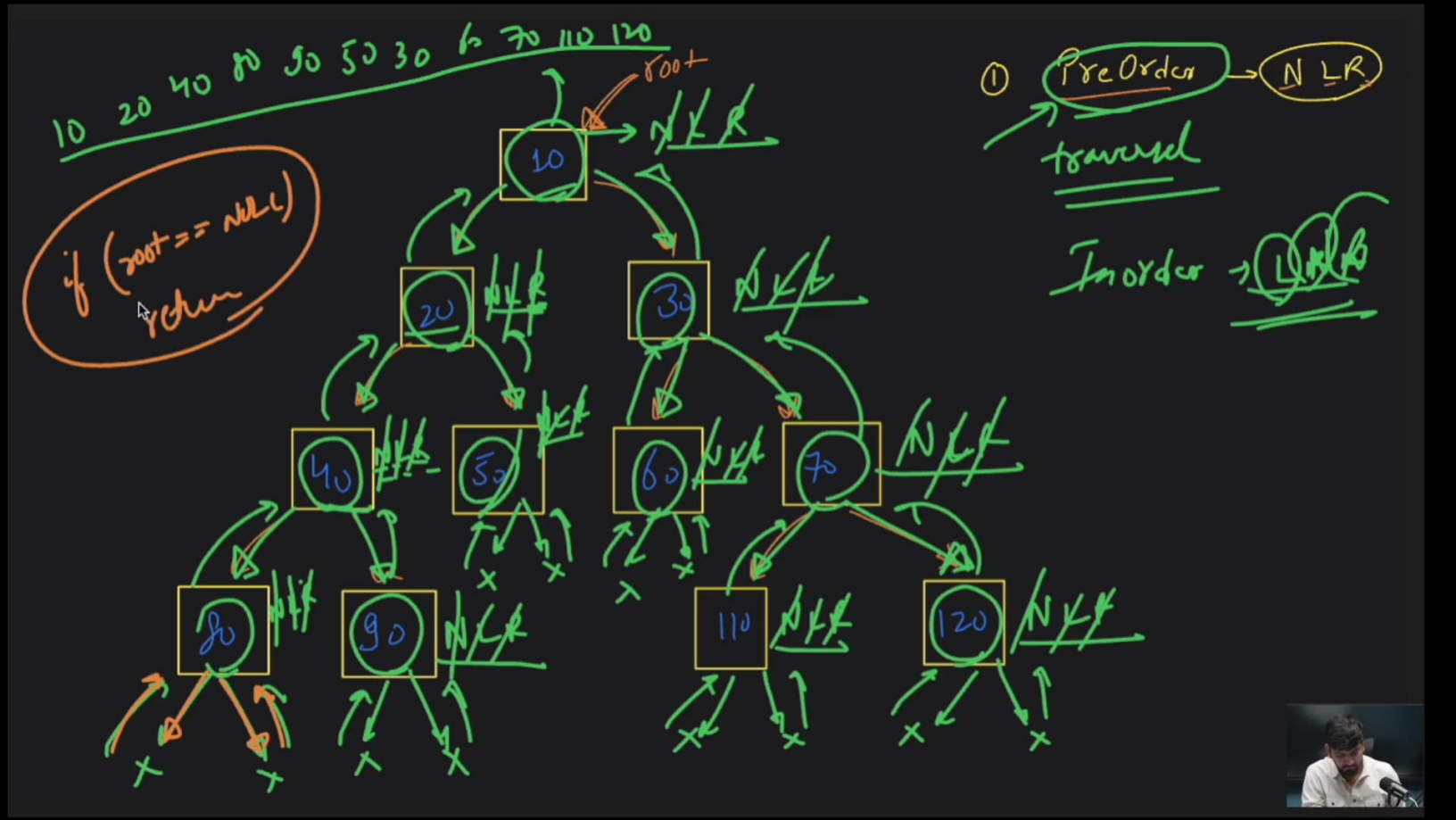
Binary Tree

Generic Tree → n Child
→ N-ary Tree


3 traversal
Current Node
left part
right part
Pre Order → N L R
In Order → L N R
Post Order → L R N

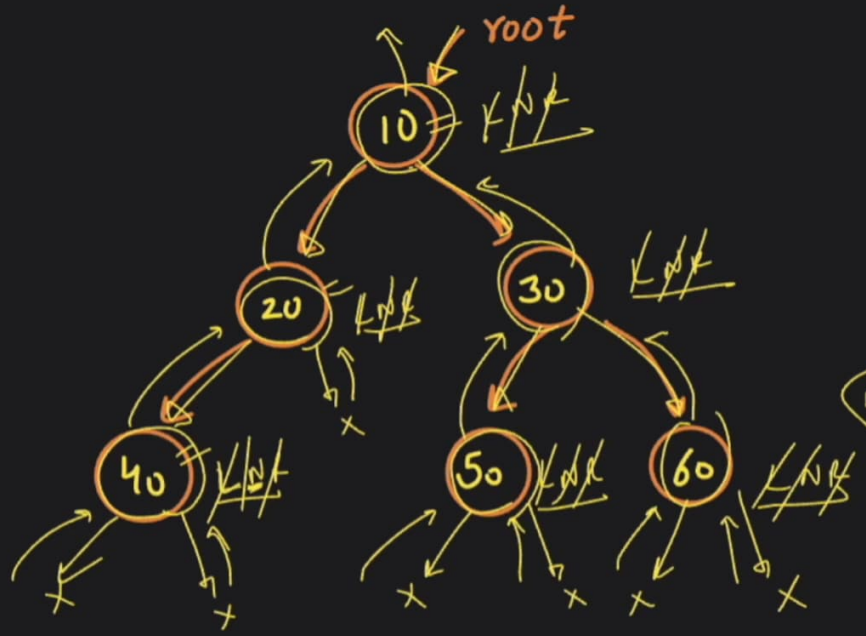

10 20 40 80 90 50 30
root
① PreOrder → N L R
10
20
30
40
50
60
70
80
90
110
120
learn.codehelp.in
zeeshankhankk907@gmail.com
+919811718169


10 20 40 80 90 50 30 60 70 110 120
root
10
20
30
40
50
60
70
80
90
110
120
① PreOrder → N L R


10 20 40 80 90 50 30 60 70 110 120
root
10
20
30
40
50
60
70
80
90
110
120
N L R
if (root == NULL)
return
① PreOrder → N L R
traversed
Inorder → L

PostOrder

LRN



Pre Order

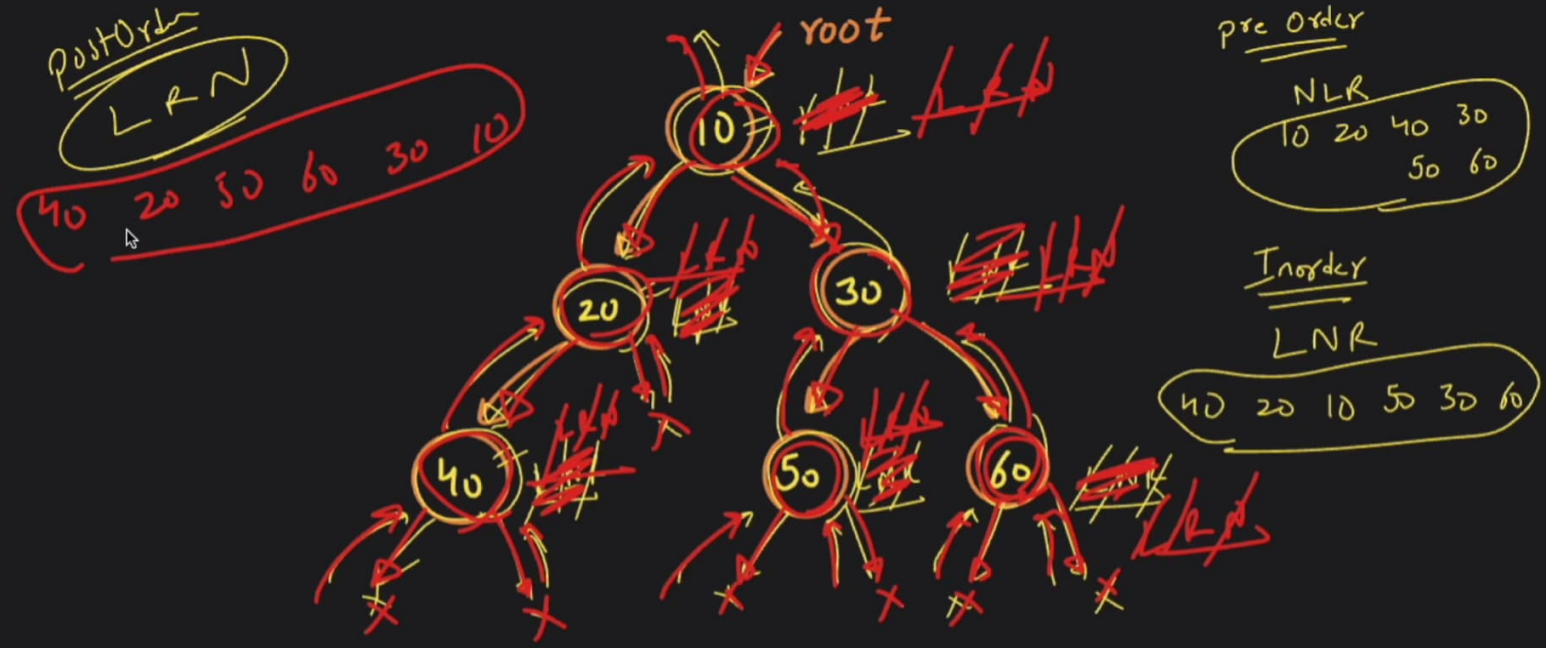
NLR

10 20 40 30 50 60

Inorder

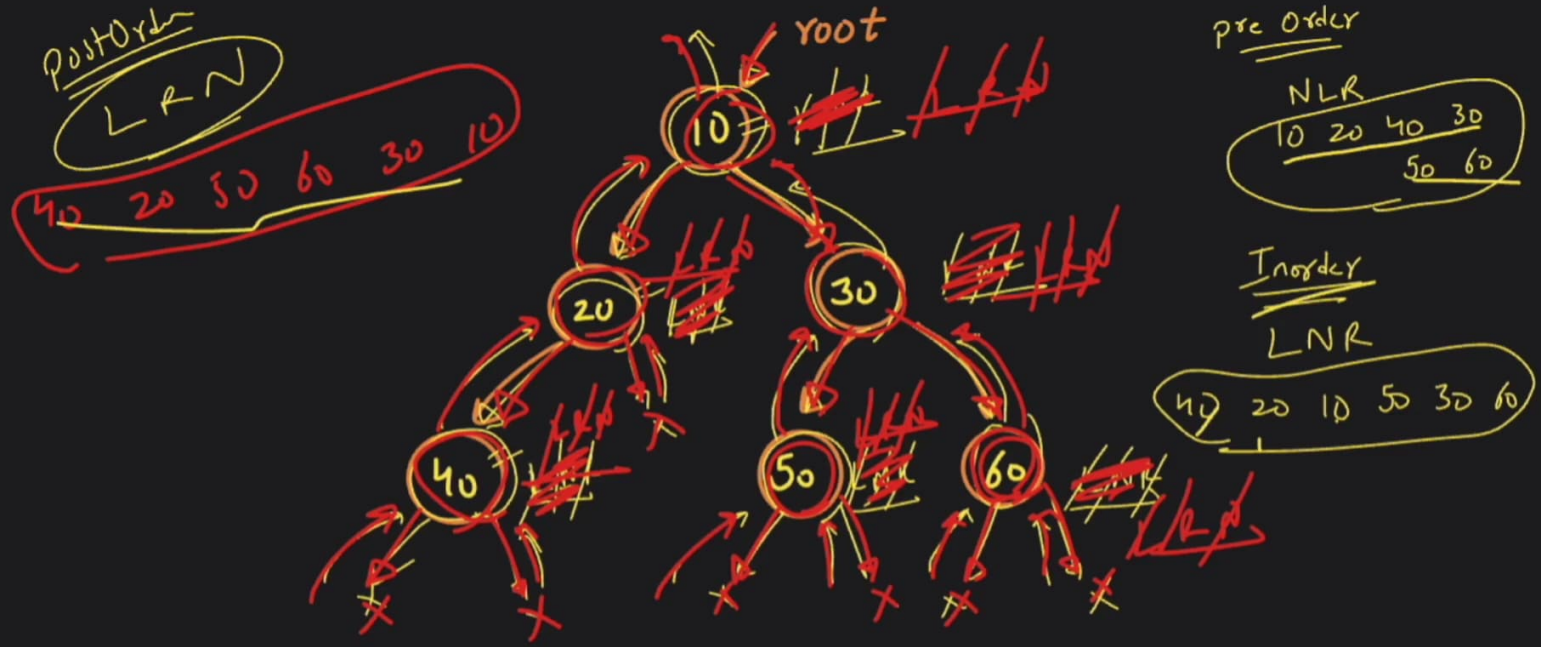
LNR

40 20 10 50 30 60

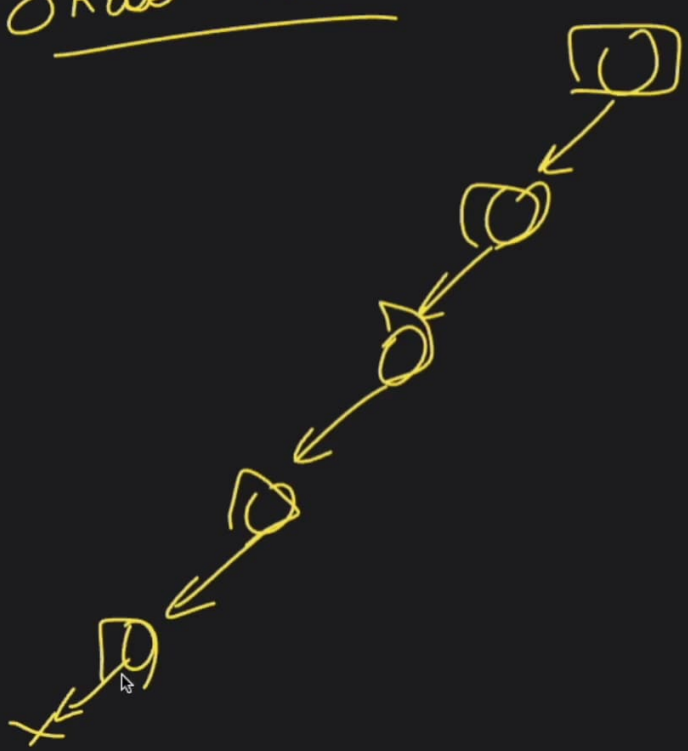

PostOrder
LRN
40 20 50 60 30 10
root
10
20
30
40
50
60
Pre Order
NLR
10 20 40 30
50 60
Inorder
LNR
40 20 10 50 30 60

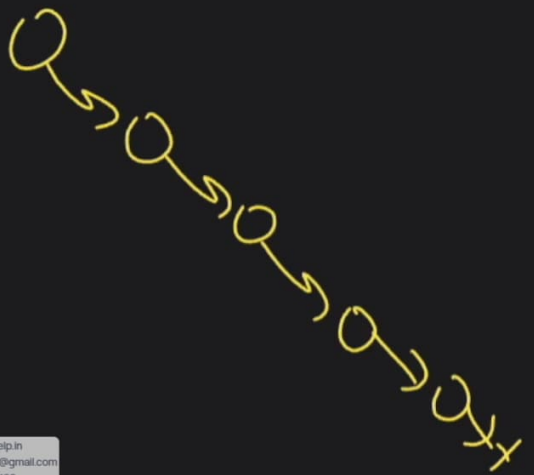
Tree input:- 10 20 40 -1 -1 -1 30 50 -1 -1

60 -1 -1


PostOrder
LRN
40 20 50 60 30 10
root
10
20
30
40
50
60
Pre Order
NLR
10 20 40 30
50 60
Inorder
LNR
40 20 10 50 30 60

Skew Tree

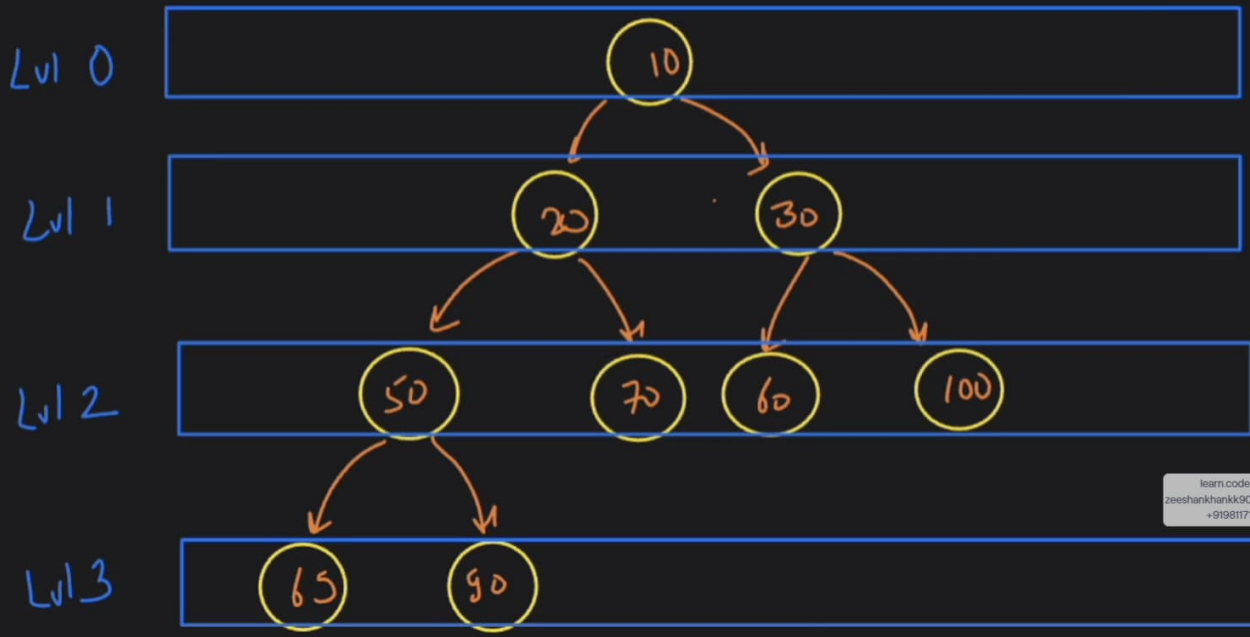




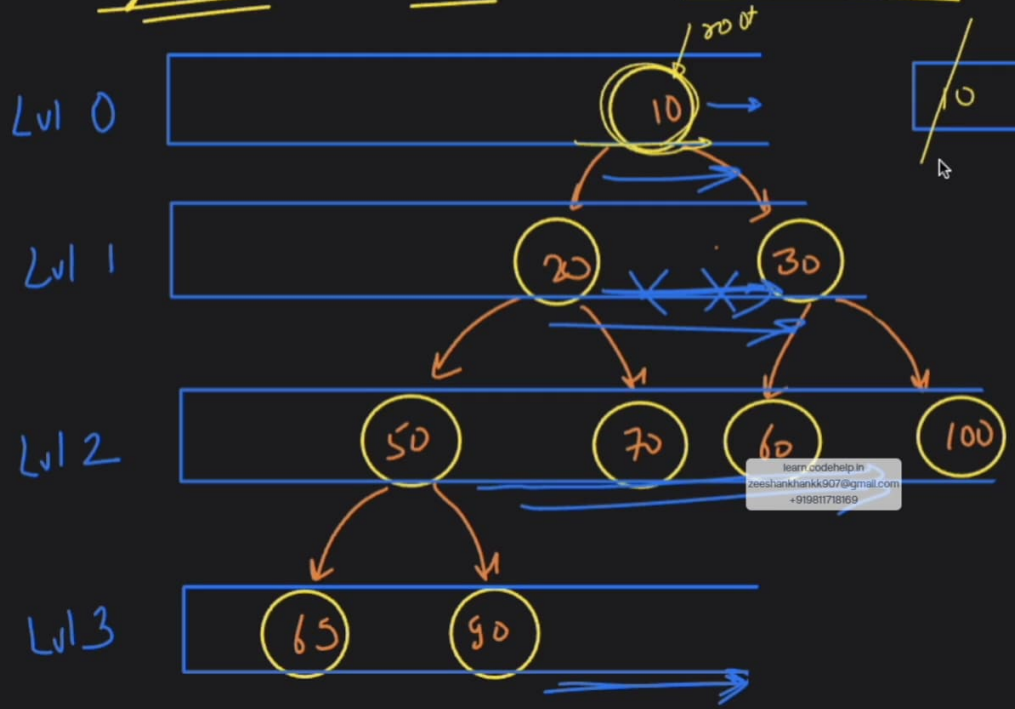
learn.codehelp.in
zeeshankhankk907@gmail.com
+919811718169

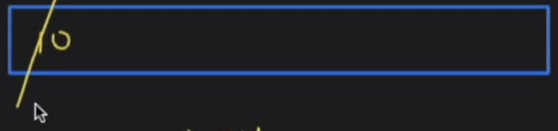
⇒ Level Order Traversal

⇒ Level Order Traversal



⇒ Level Order Traversal

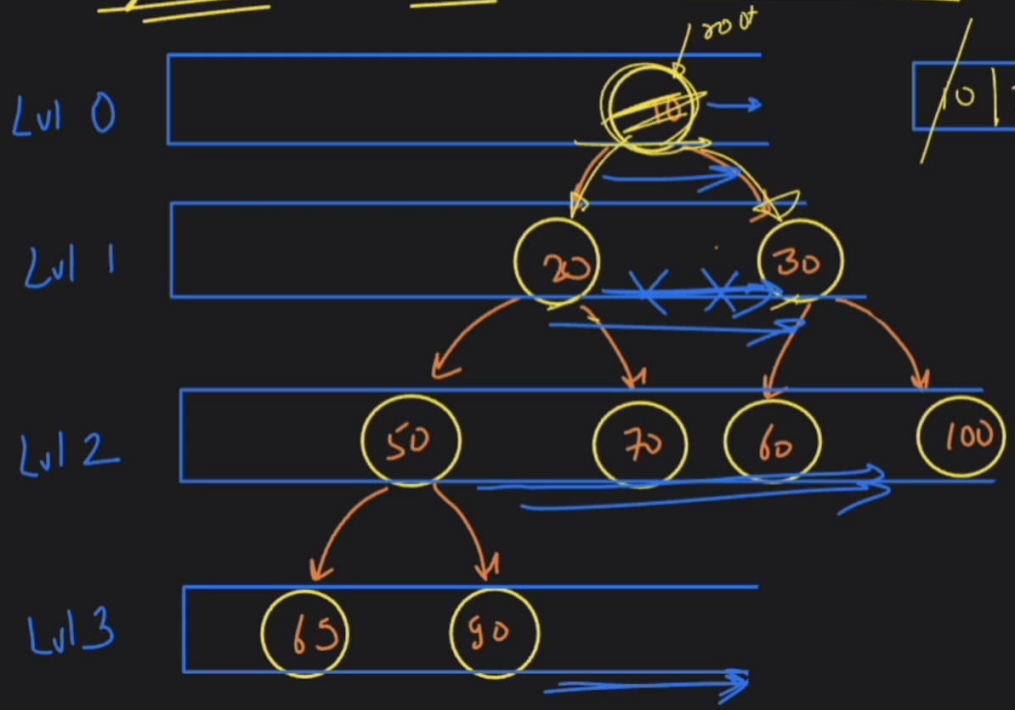




initialy
A push root node

traversal
↳ fetch front

⇒ Level Order Traversal



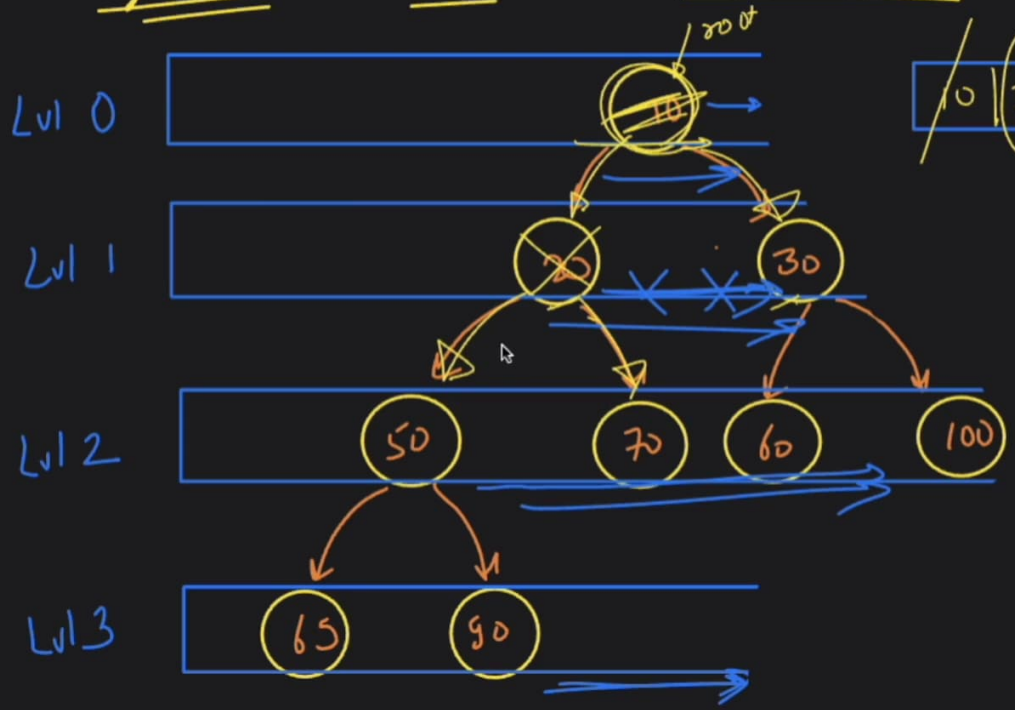
10 | 20 | 30

initially
A push root node

traversal
fetch front
pop left
push right

⇒ Level Order Traversal

10 20



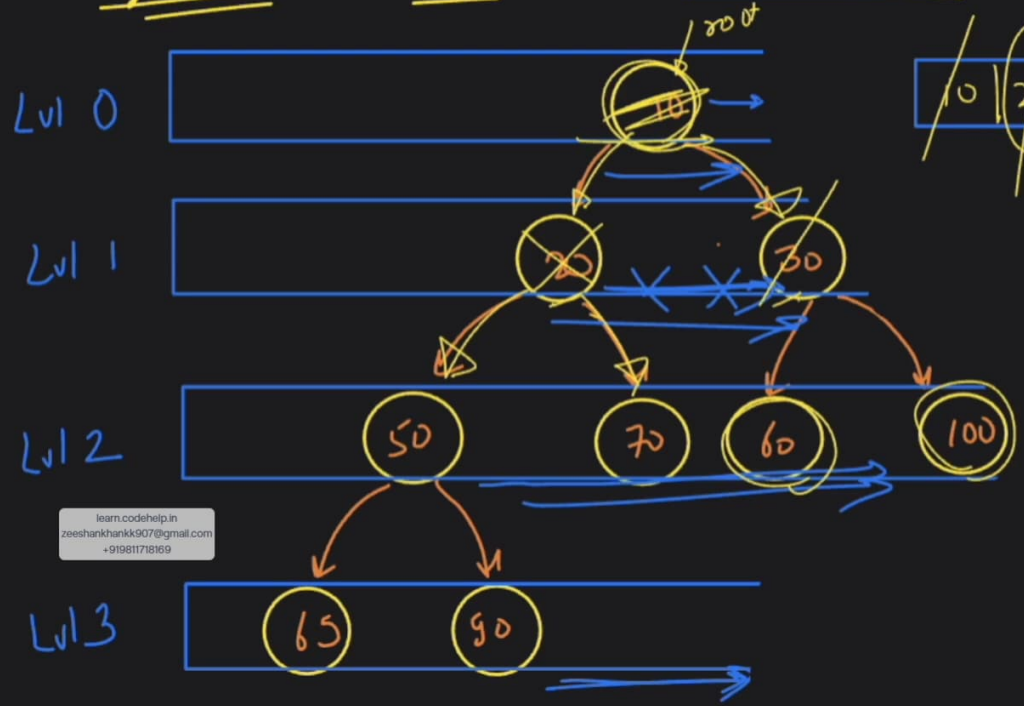
10	20	30	50	70

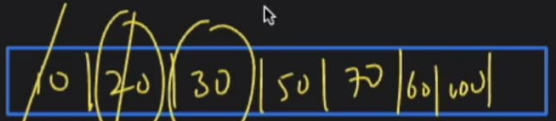
initially
A push root node

traversal
fetch front
pop left
push right

⇒ Level Order Traversal

10 20 30



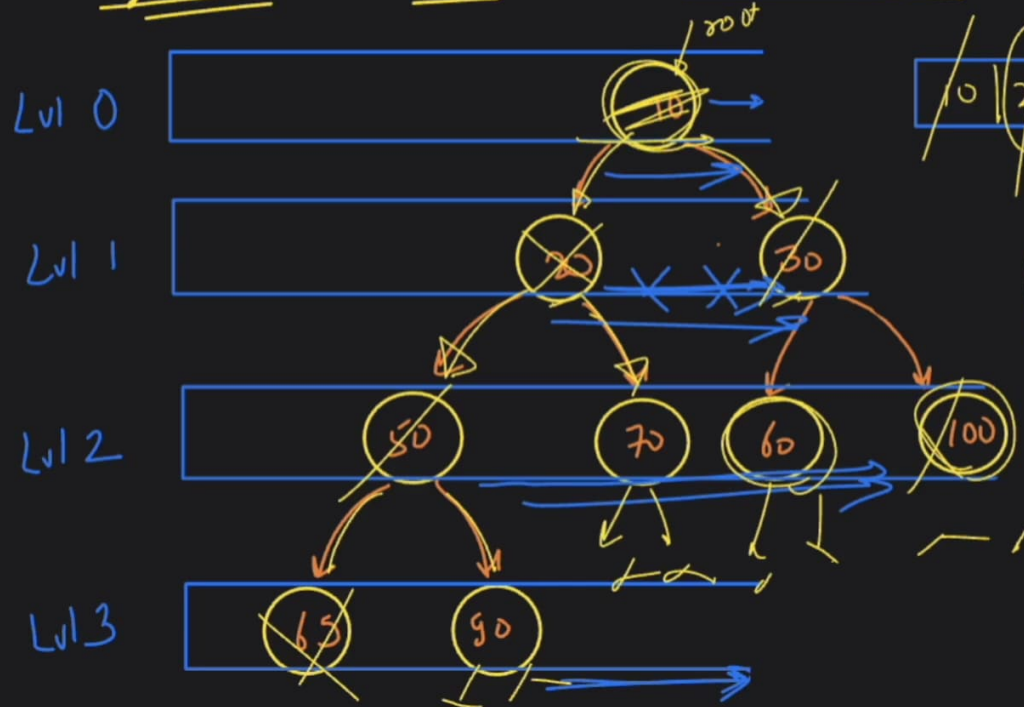


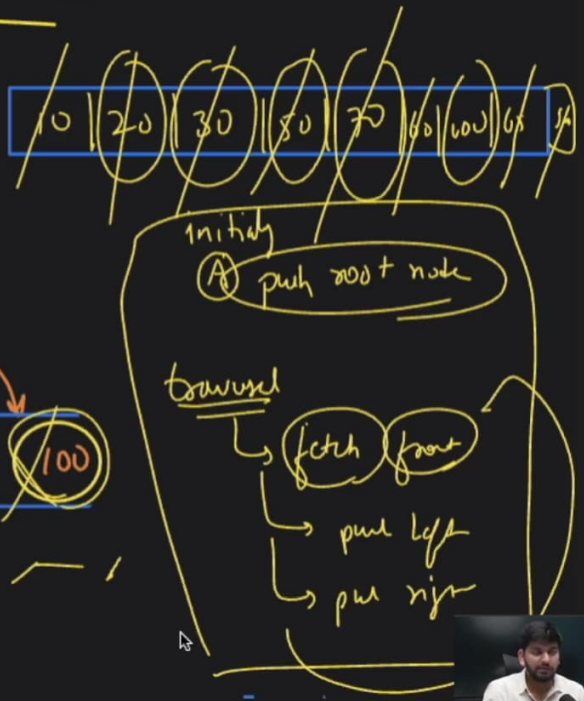
initially
A) push root node

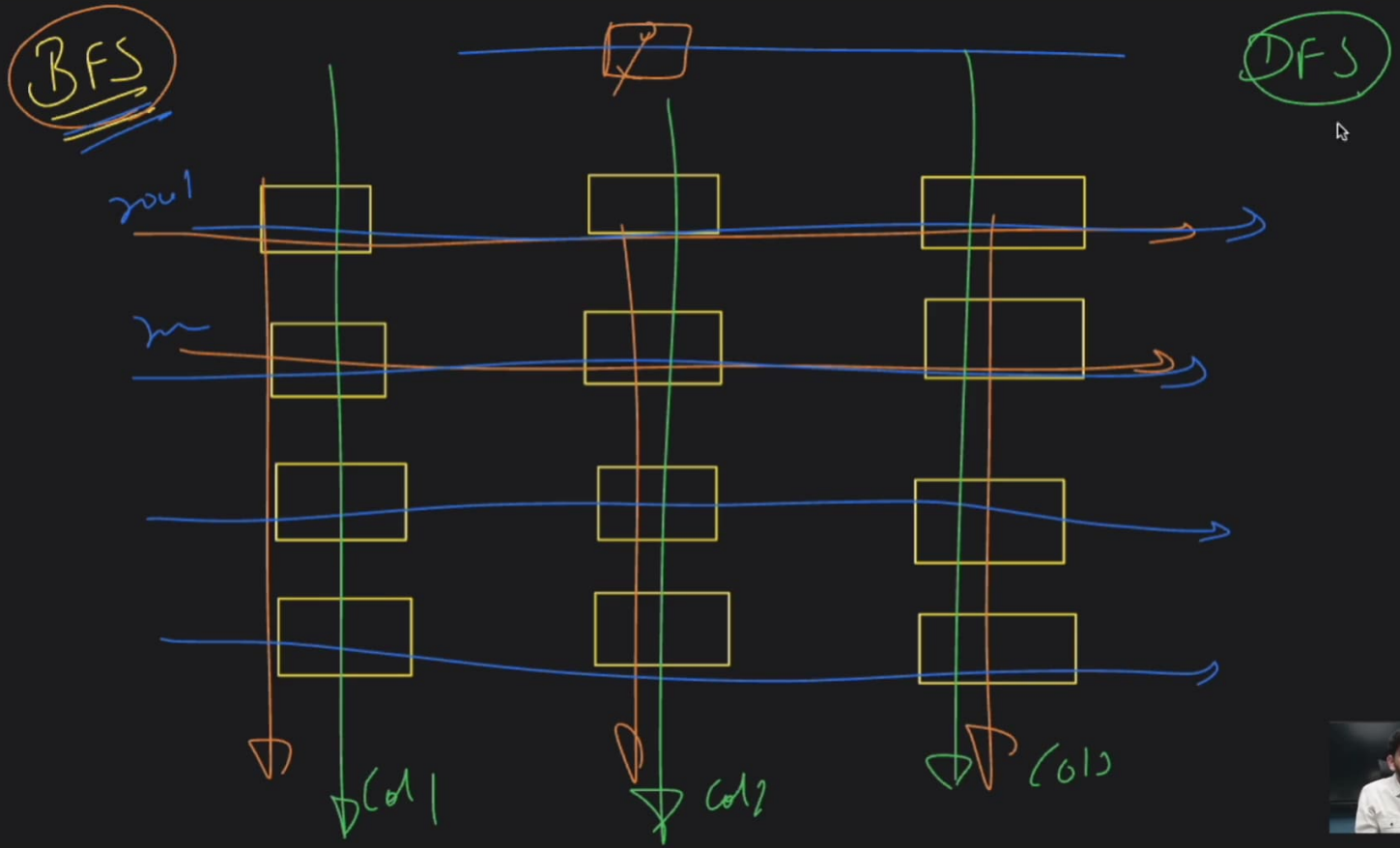
traversal
fetch front
pop left
pop right

⇒ Level Order Traversal

10 20 30 50 70 60 100 65 90

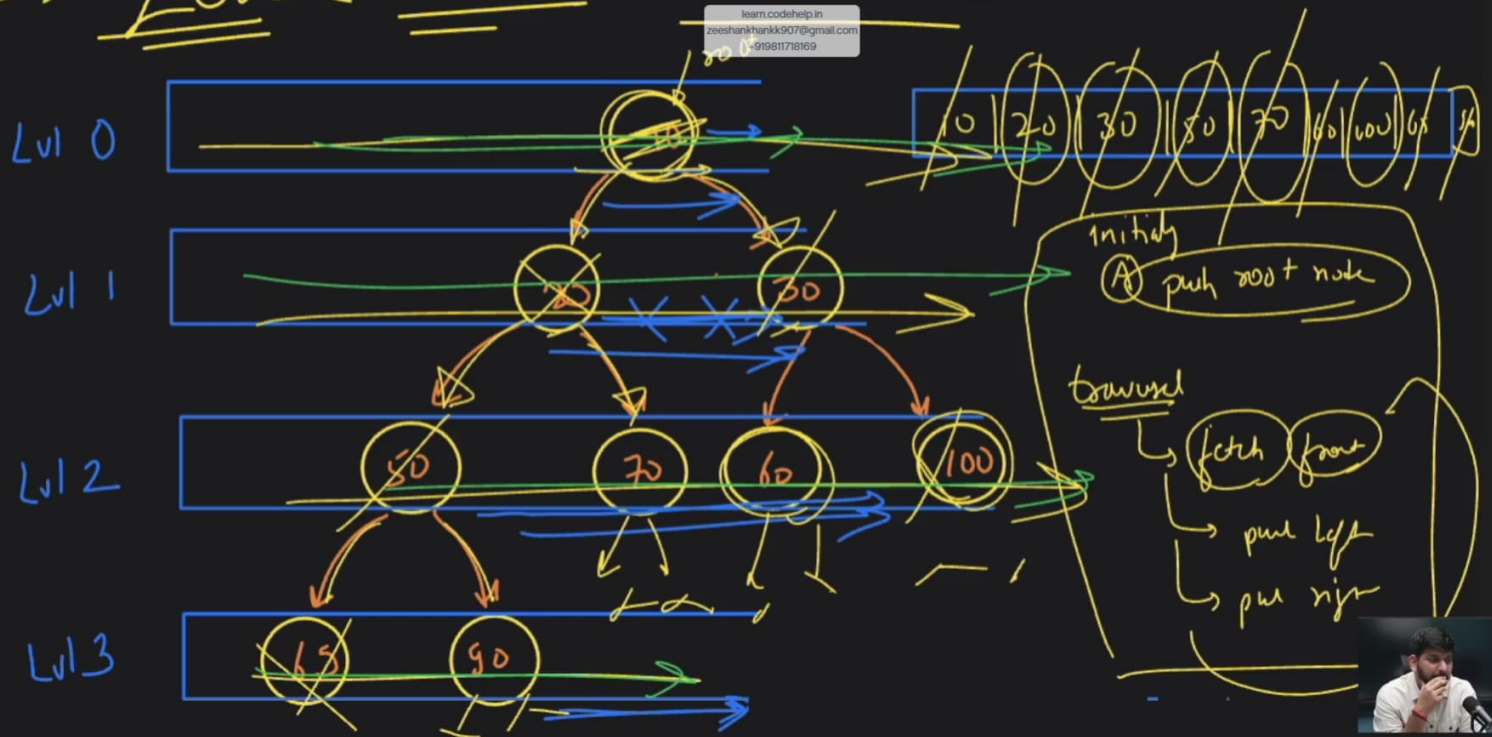


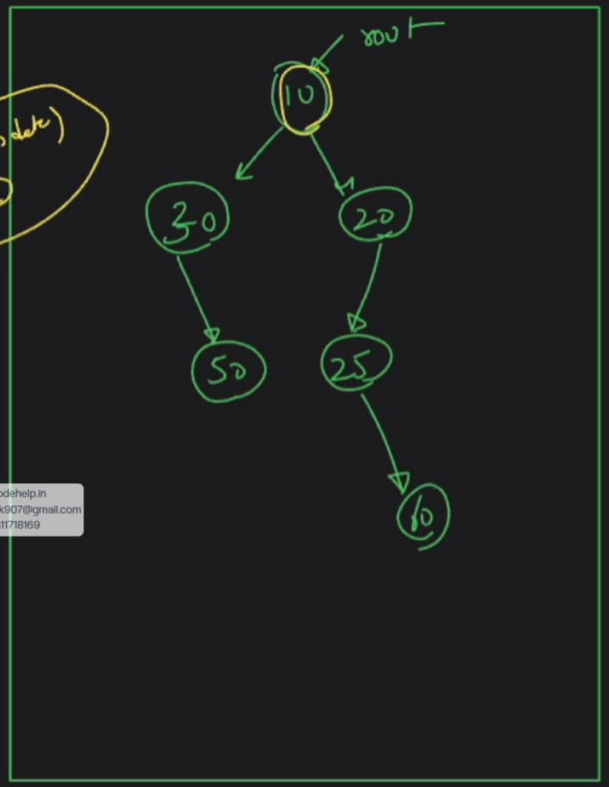



BFS
DFS
row 1
row 2
Col 1
Col 2
Col 3

⇒ Level Order Traversal

10 20 30 50 70 60 100 65 90



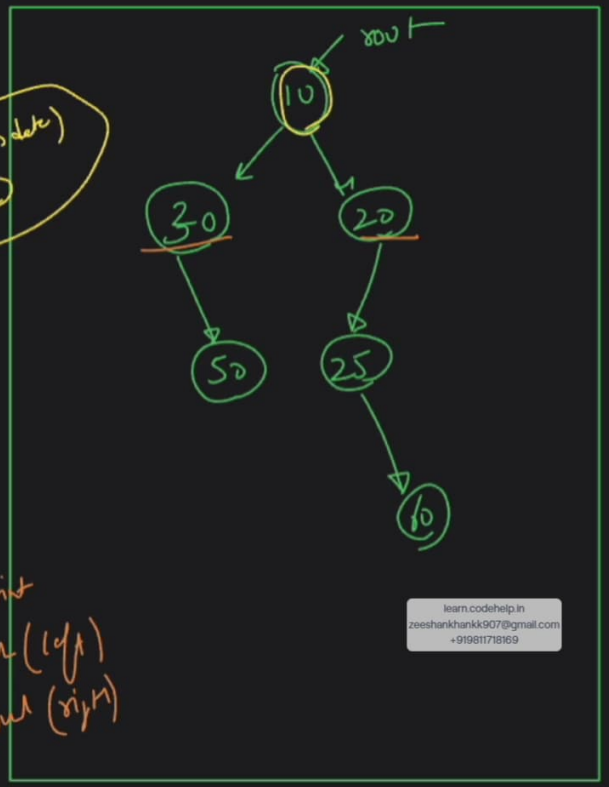


initial
q.push(root->data)
q.push(NULL)

NULL → marker
Level complete
logic
(cout << endl)

initial

q



initial
- q.push(root->data)
- q.push(NULL)

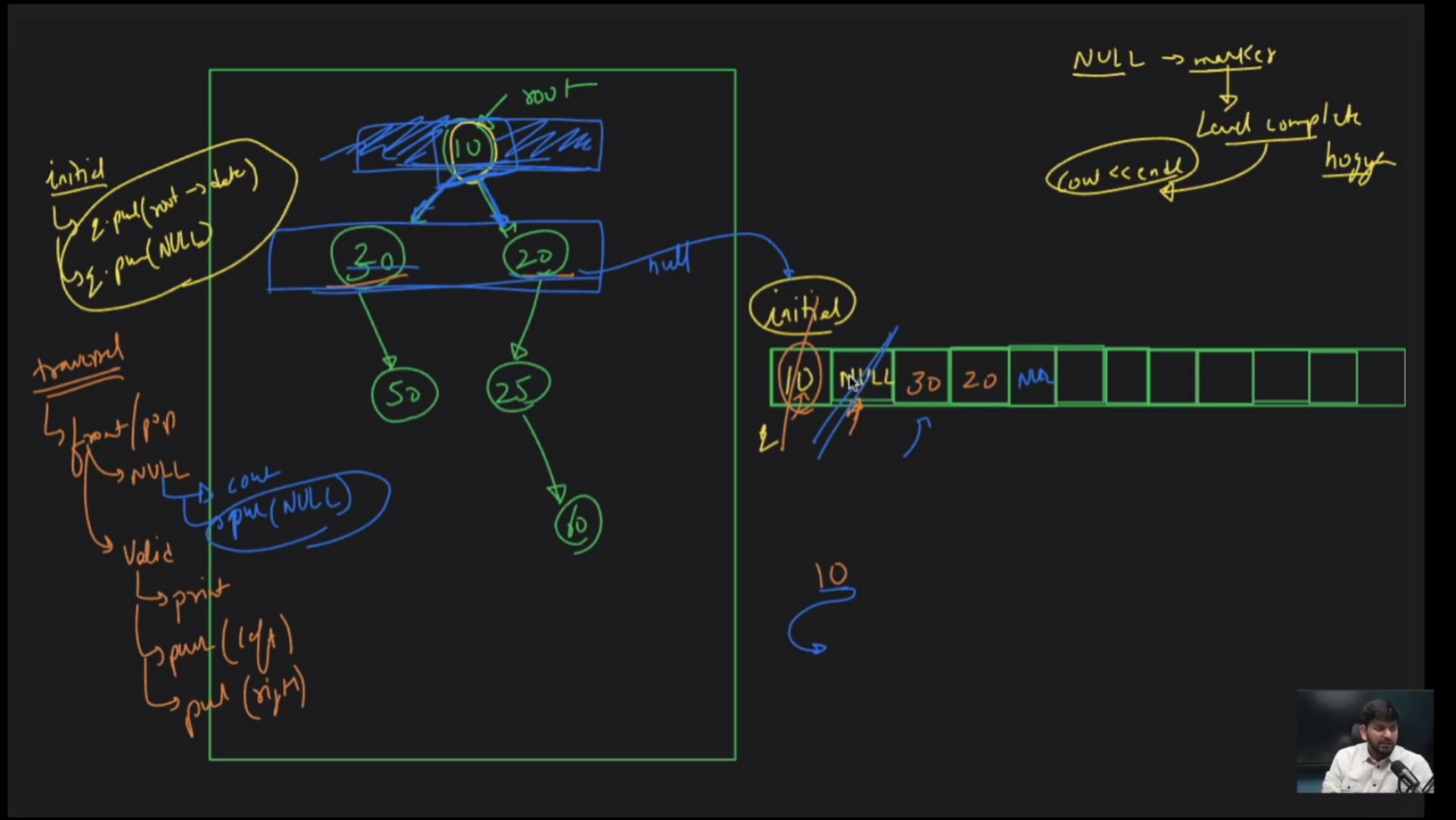
traversal
- front/pop
  - NULL
  - Valid
    - print
    - push(left)
    - push(right)

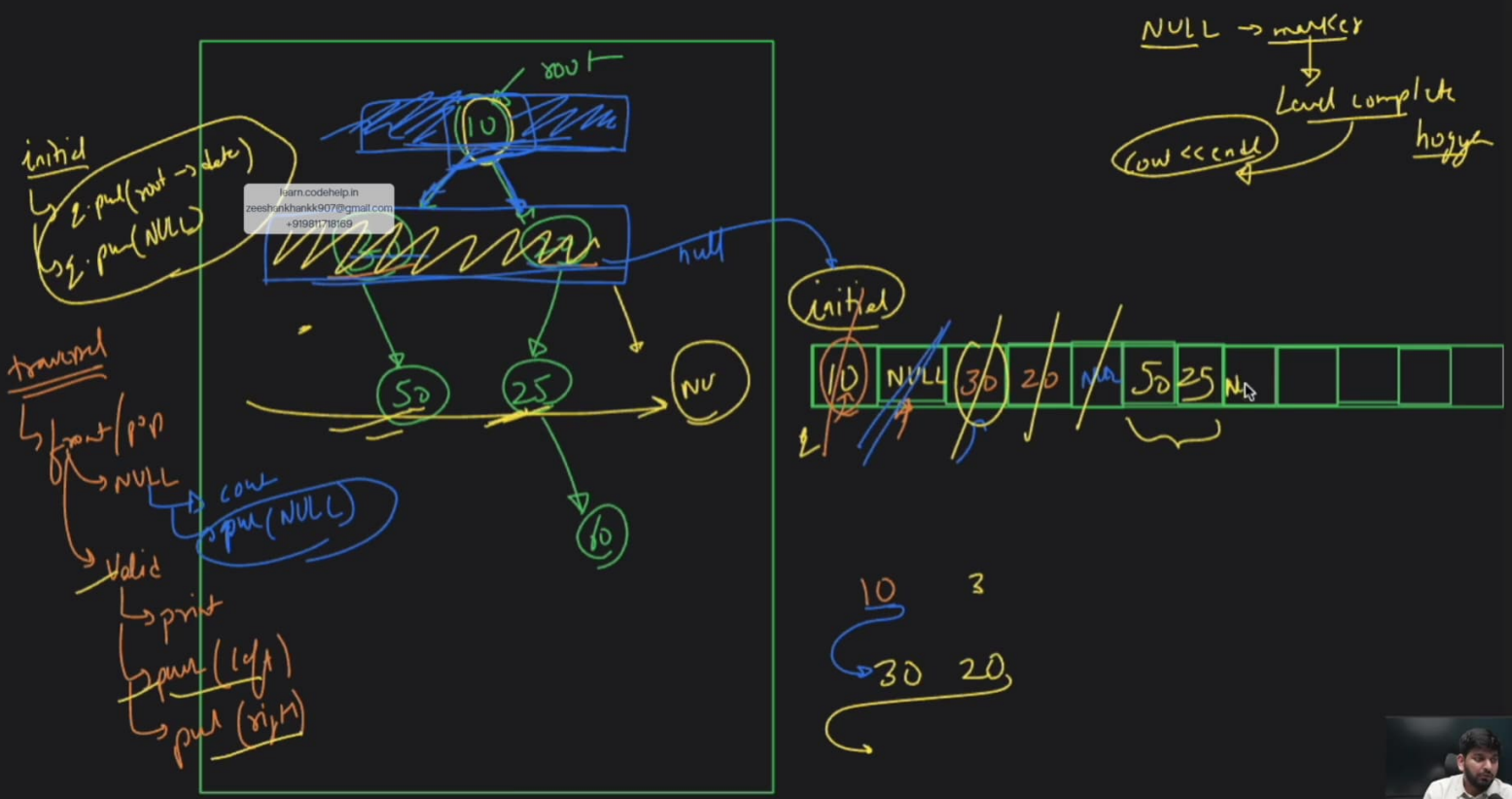
NULL → marker
- Level complete
- (low << end)
- height

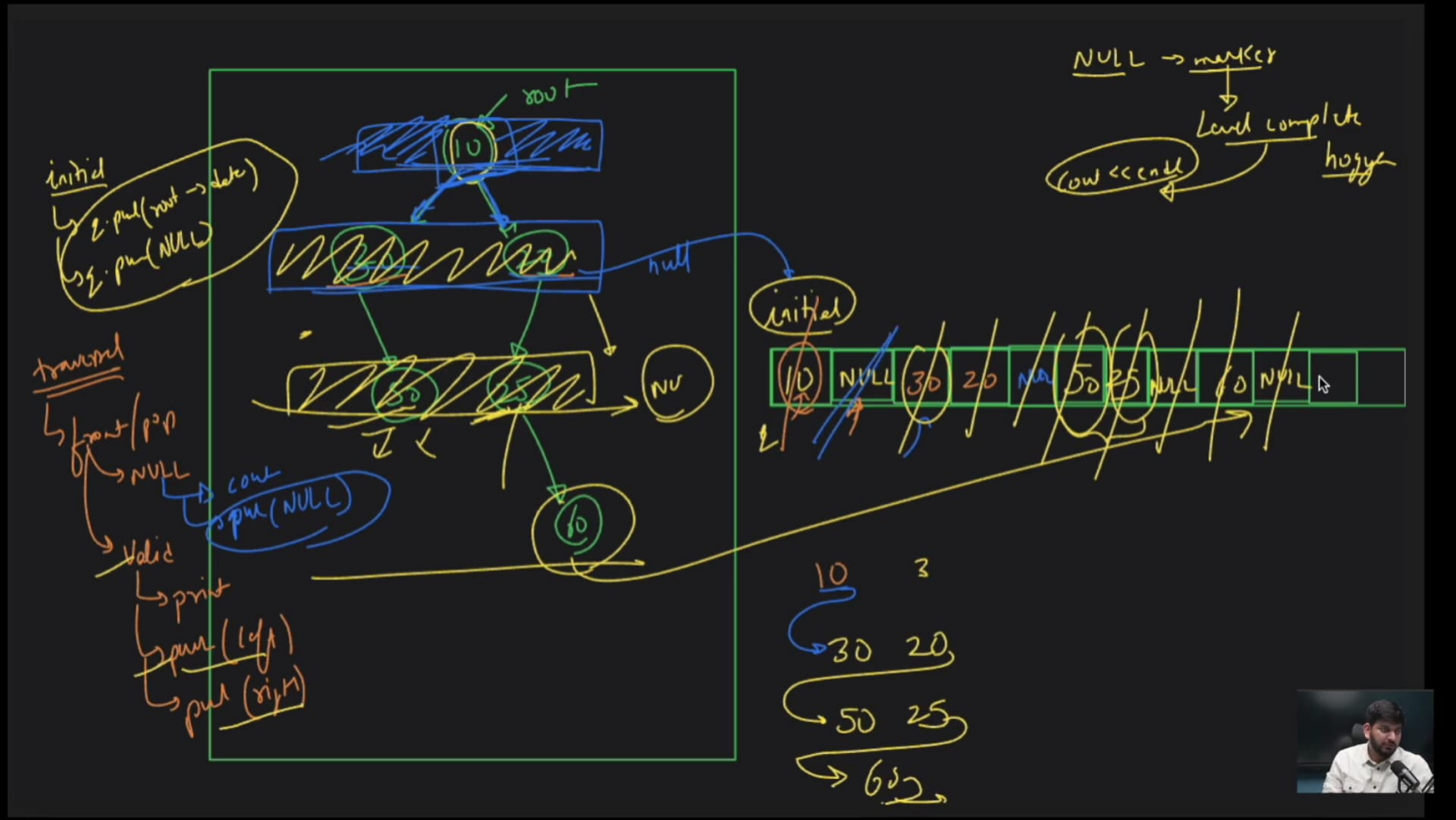
initial

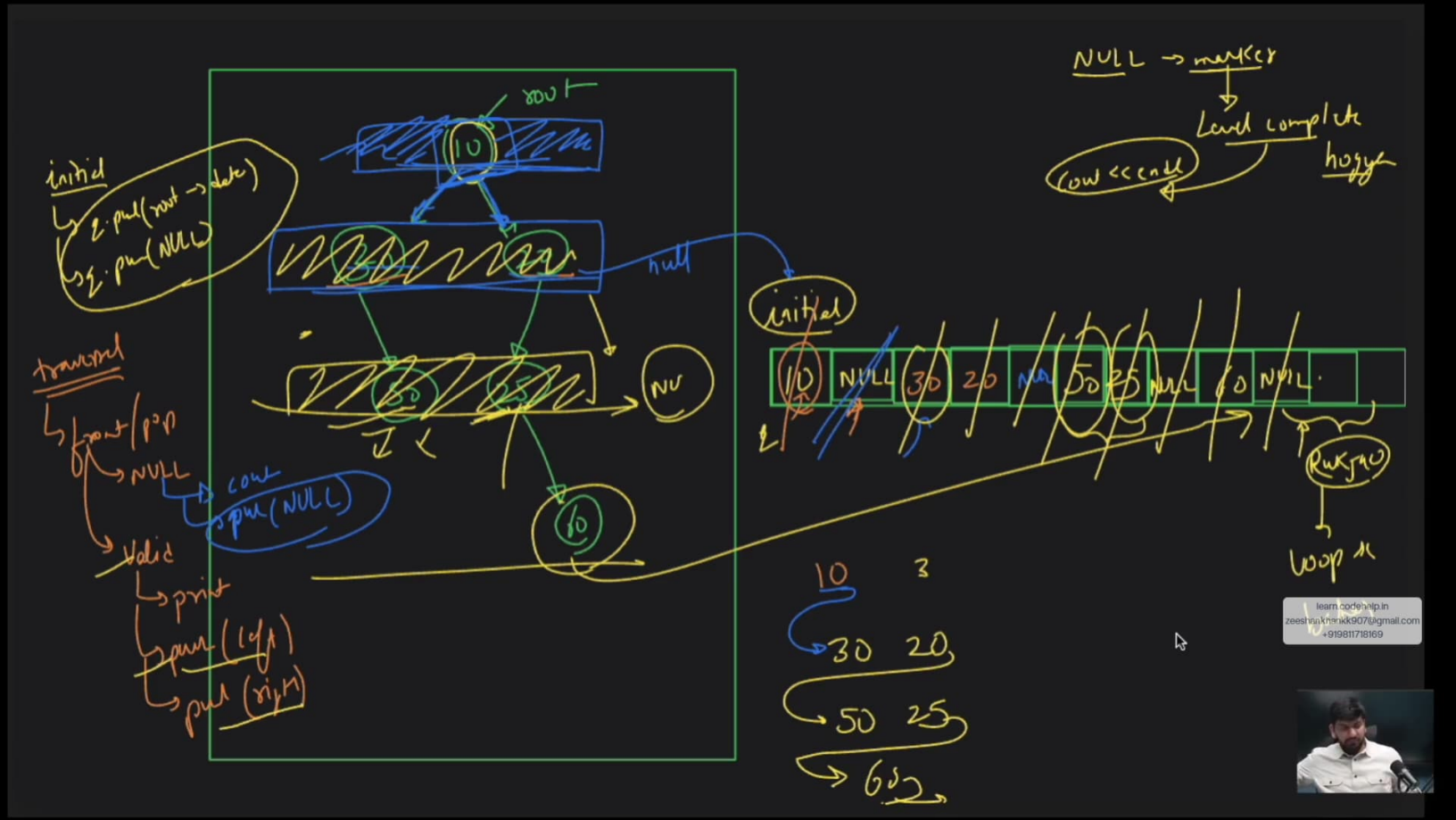
q

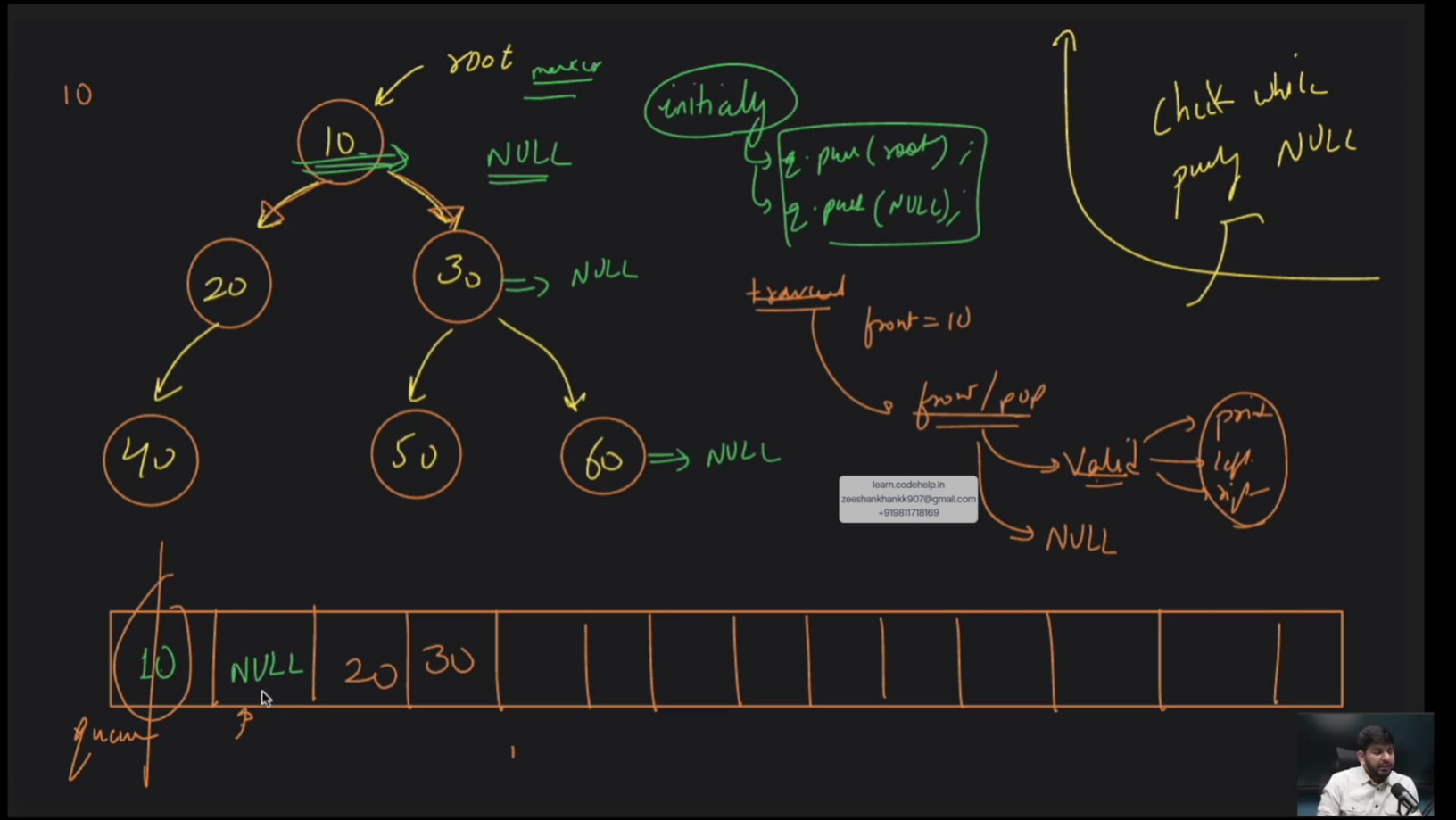
10

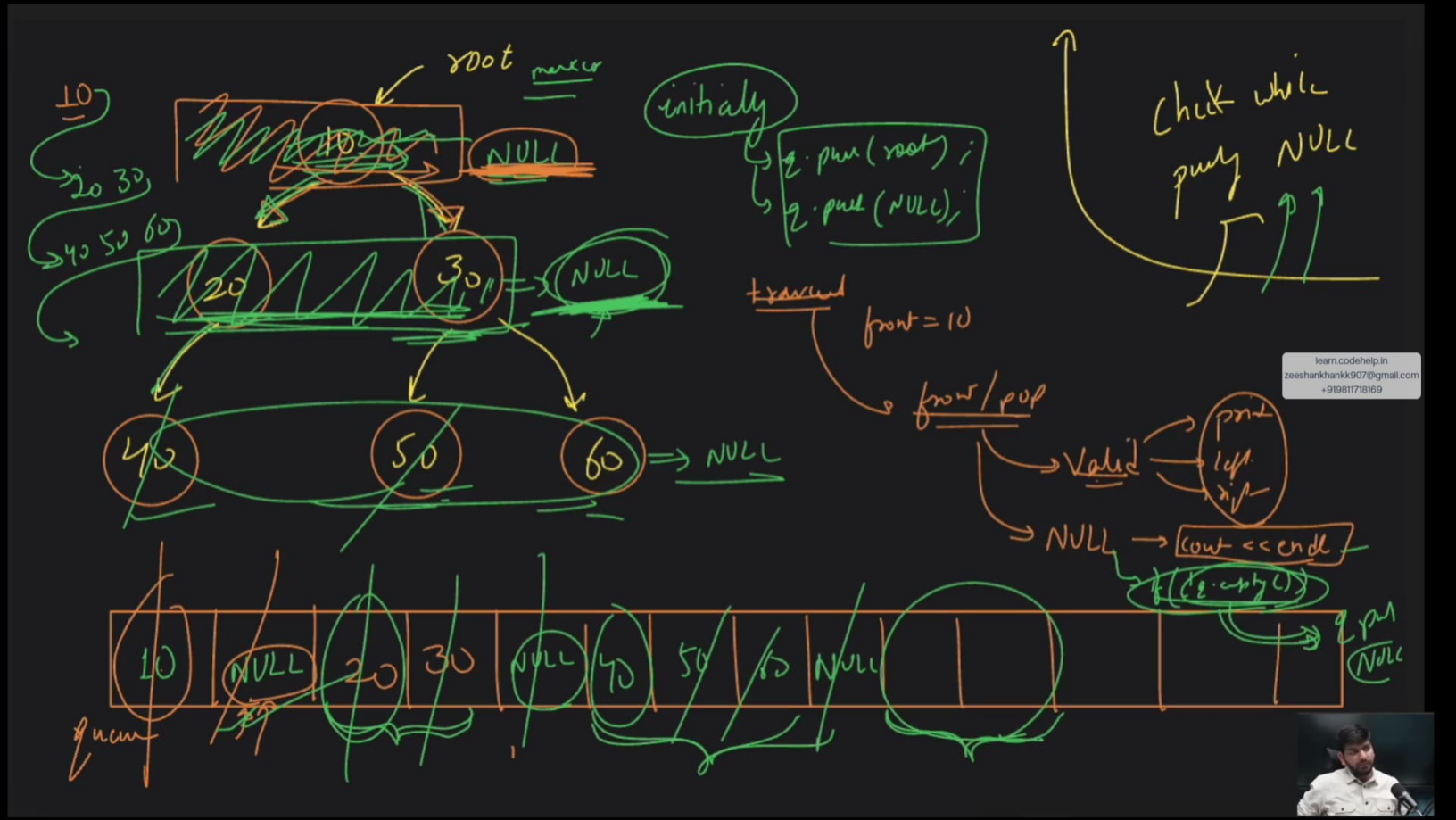

NULL → marker
Level complete
cout << endl
bigger
root
10
30
20
null
50
25
40
initial
i : push(root → data)
i : push(NULL)
traversal
front/pop
NULL
cout
push(NULL)
Valid
print
push(left)
push(right)
initial
10
NULL
30
20
NULL
10


NULL → marker
Level complete
lower
root
10
initial
push(root → data)
push(NULL)
learn.codehelp.in
zeeshankhankk907@gmail.com
+919811718169
null
50
25
NU
10
traversal
front/pop
NULL
cout
pop(NULL)
Valid
print
push(left)
push(right)
initial
10
NULL
30
20
50
25
10 3
30 20


NULL → marker
Level complete
cout << endl
initial
traversal
front/pop
NULL
Valid
print
push(left)
push(right)
cout
push(NULL)
root
null
initial
10 NULL 30 20 NULL 50 25 NULL 60 NULL
10 3
30 20
50 25
60

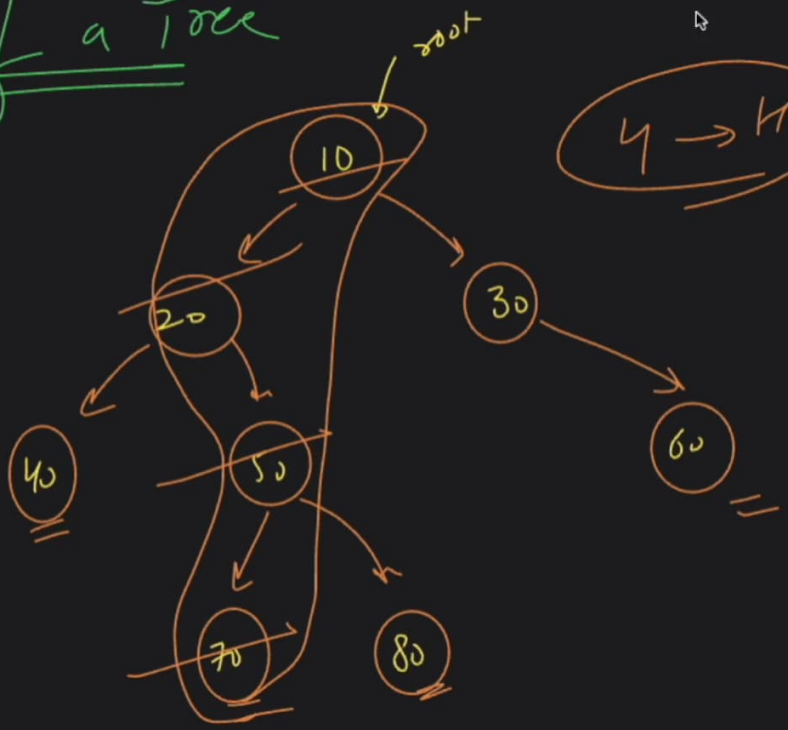

NULL → marker
Level complete
root
10
initial
traversal
front/pop
NULL
Valid
print
push (left)
push (right)
null
NULL
Loop
10 3
30 20
50 25
60


10
root
marker
10
NULL
20
30
NULL
40
50
60
NULL
initially
q.push(root);
q.push(NULL);
Check while
pop NULL
traversal
front = 10
front/pop
Valid
print
left
right
NULL
learn.codehelp.in
zeeshankhankk907@gmail.com
+919811718169
10
NULL
20
30
queue
f
r

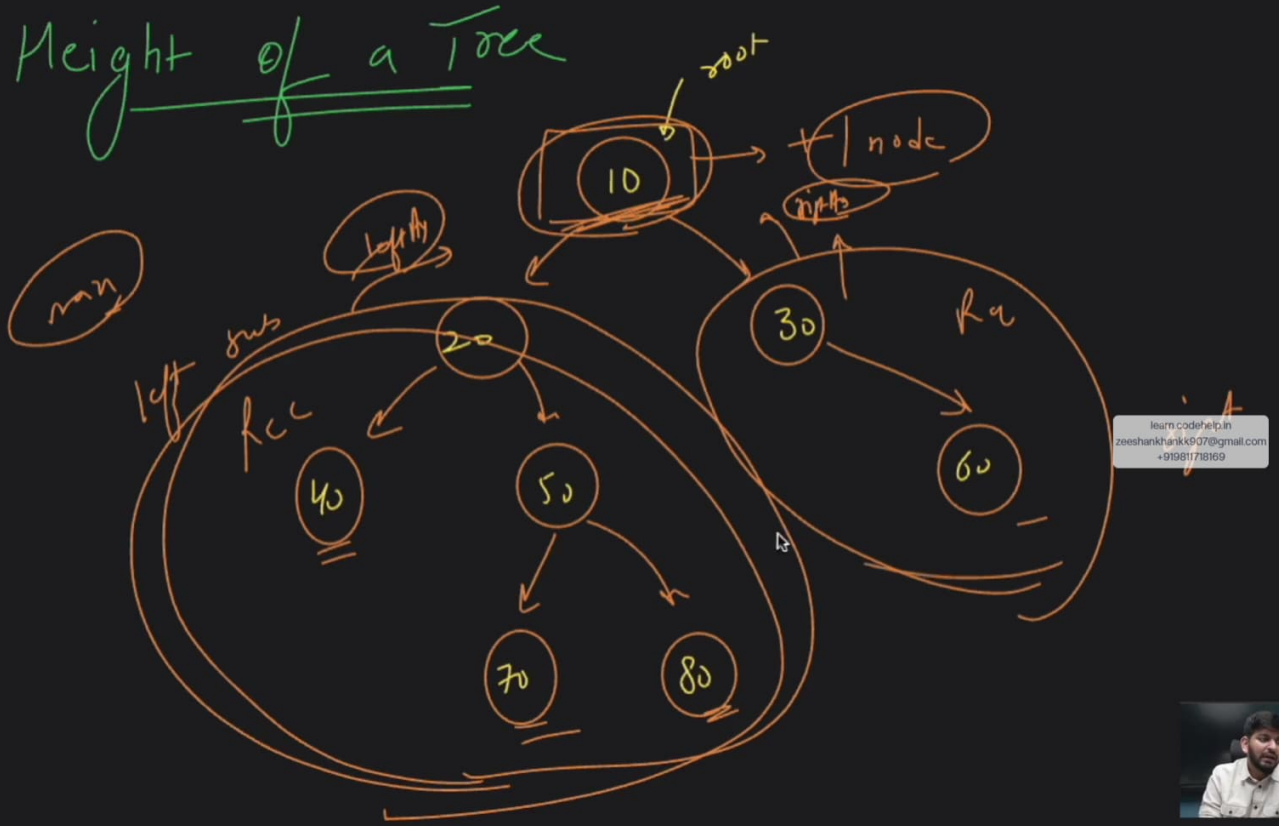

root
marker
10
NULL
20 30
40 50 60
20
30
NULL
40
50
60
NULL
initially
q.push(root);
q.push(NULL);
traversal
front = 10
front / pop
Valid
print
left
right
NULL
cout << endl
if(q.empty())
q.push
NULL
Check while
push NULL
10
NULL
20
30
NULL
40
50
60
NULL
queue
learn.codehelp.in
zeeshankhankk907@gmail.com
+919811718169

→ Height of a Tree

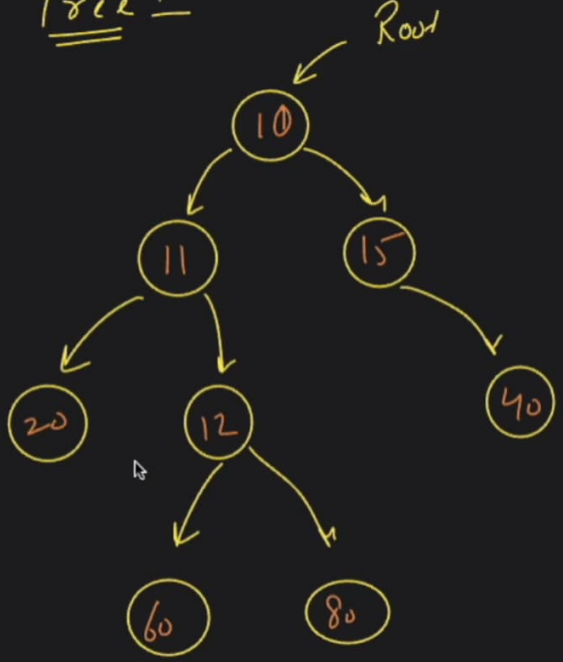
Height of a Tree



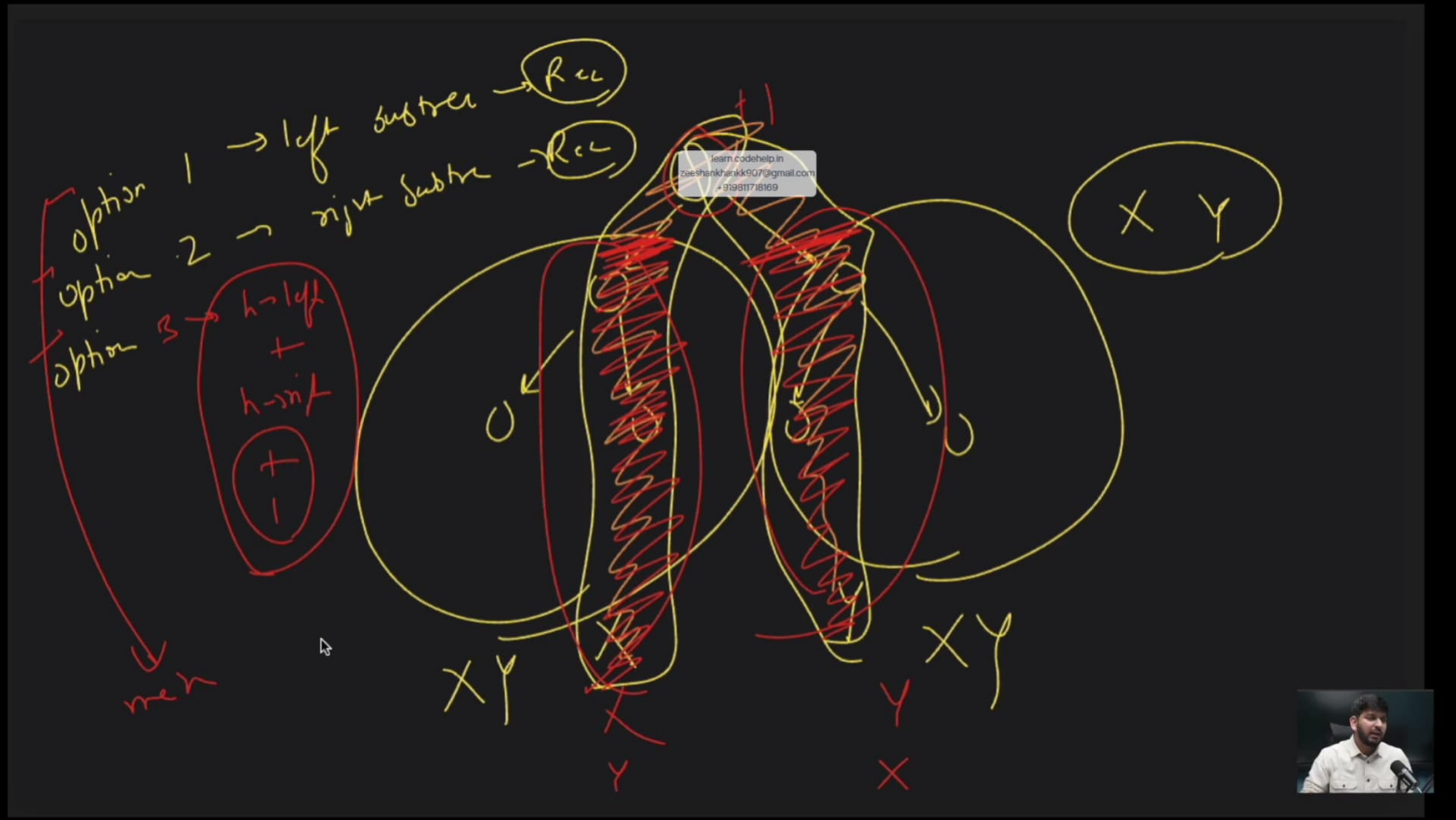
Height of a Tree

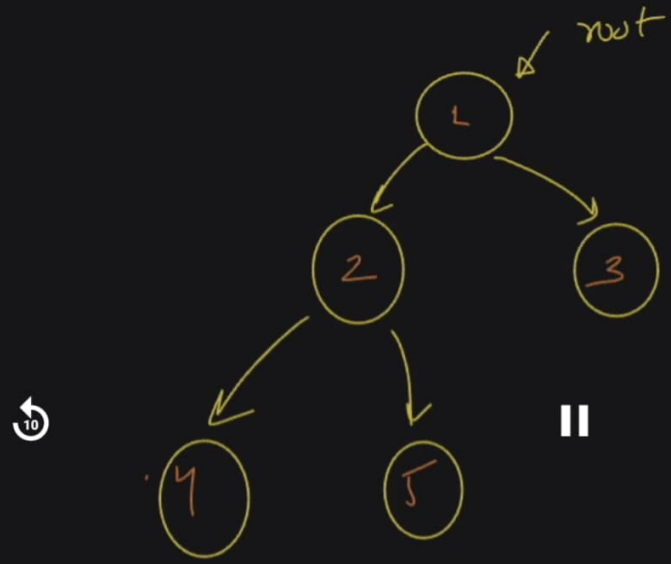


Diameter of Tree :-



→ Diameter of Tree :-

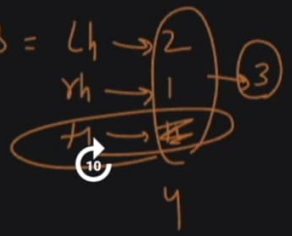

option 1 → left subtree → Rec
option 2 → right subtree → Rec
option 3 → h→left + h→right + 1
learn.codehelp.in
zeeshankhankk907@gmail.com
+919811718169
X Y
XY
XY
Y X
max

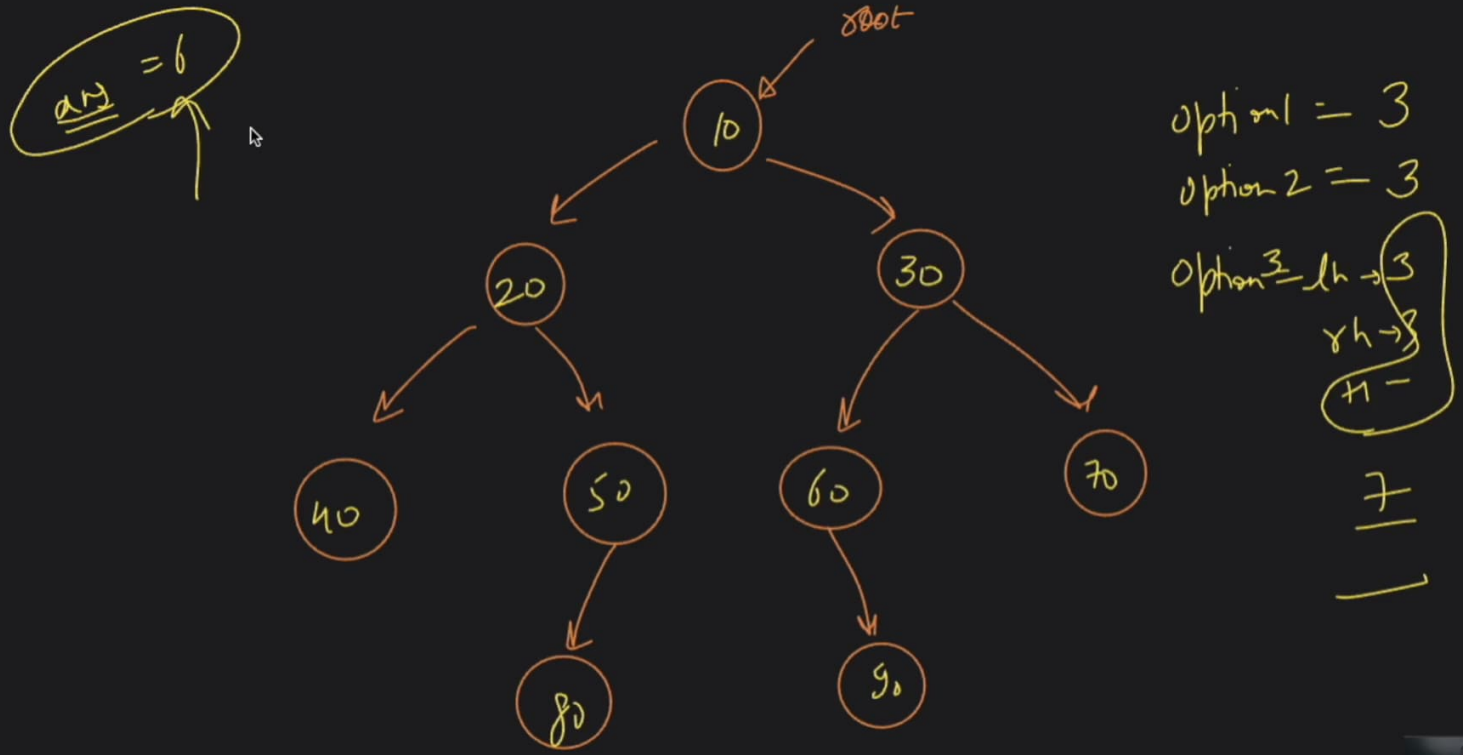


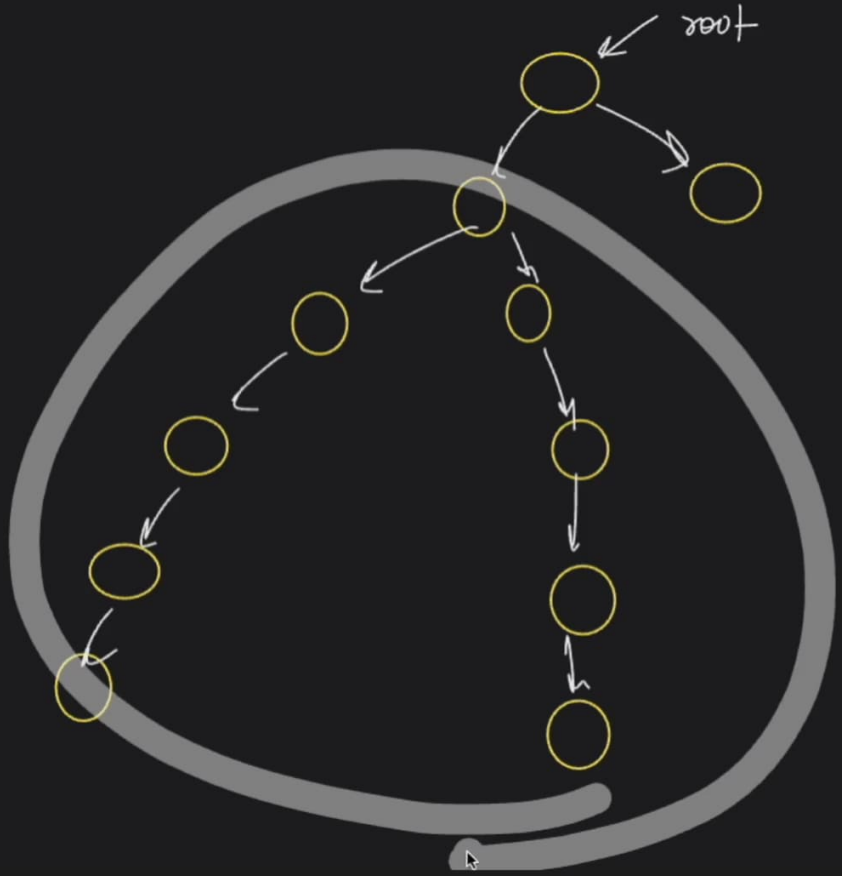
option1 => 2

option2 => 0

option3 =




ans = 6
root
10
20
30
40
50
60
70
80
90
option1 = 3
option2 = 3
option3 lh → 3
rh → 3
+1
7


root

Home - CodeHelp | Course Builder | Binary Trees Class - 2 | Balanced Binary Tree - LeetC | Lowest Common Ancestor o | Path Sum - LeetCode | Path Sum II - LeetCode

leetcode.com/problems/balanced-binary-tree/

Gmail | YouTube | Maps | CodeHelp - Dot B... | CodeHelp - Web D... | Todo | Graphy REST API... | osTicket :: Agent L... | Supreme 2023 - G... | All Bookmarks

Problem List | Dynamic Layout | Premium | 0

Description | Editorial | Solutions (5.8K) | Submissions

Height-Balanced

A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

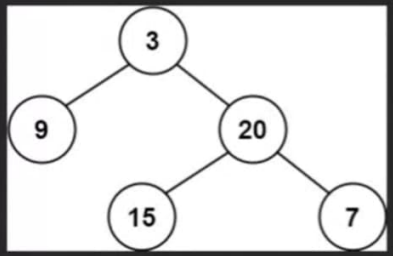
110. Balanc

Easy 10.1

Companies

Given a binary tree, determine if it is height-balanced.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: true
```

C++ | Auto

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left
(left), right(right) {}
 * };
 */
class Solution {
public:
    bool isBalanced(TreeNode* root) {

    }
};
```

Saved to local

learn.thecodehelp.in is sharing your screen. Stop sharing Hide

00:04:41 / 02:08:54