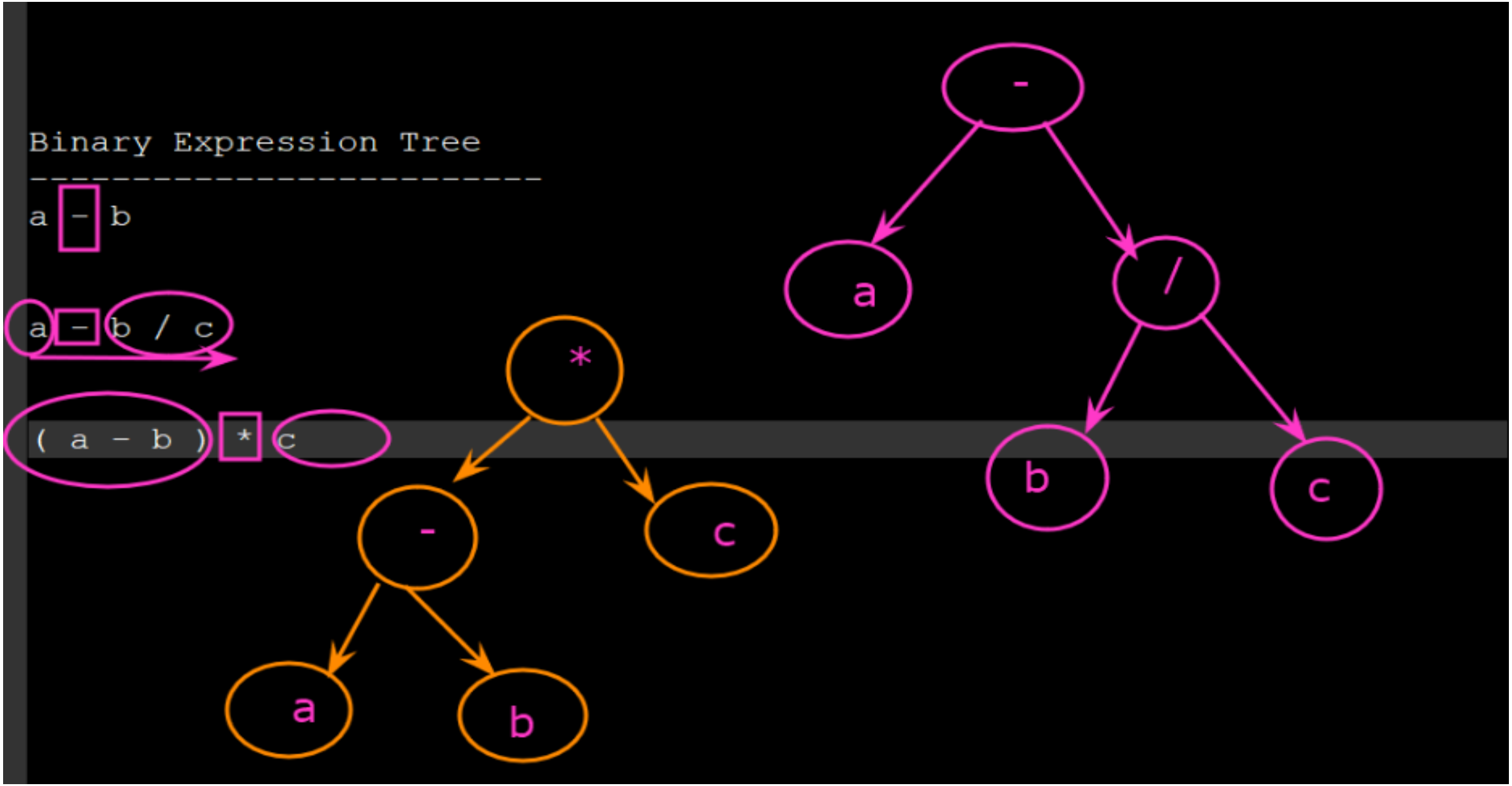
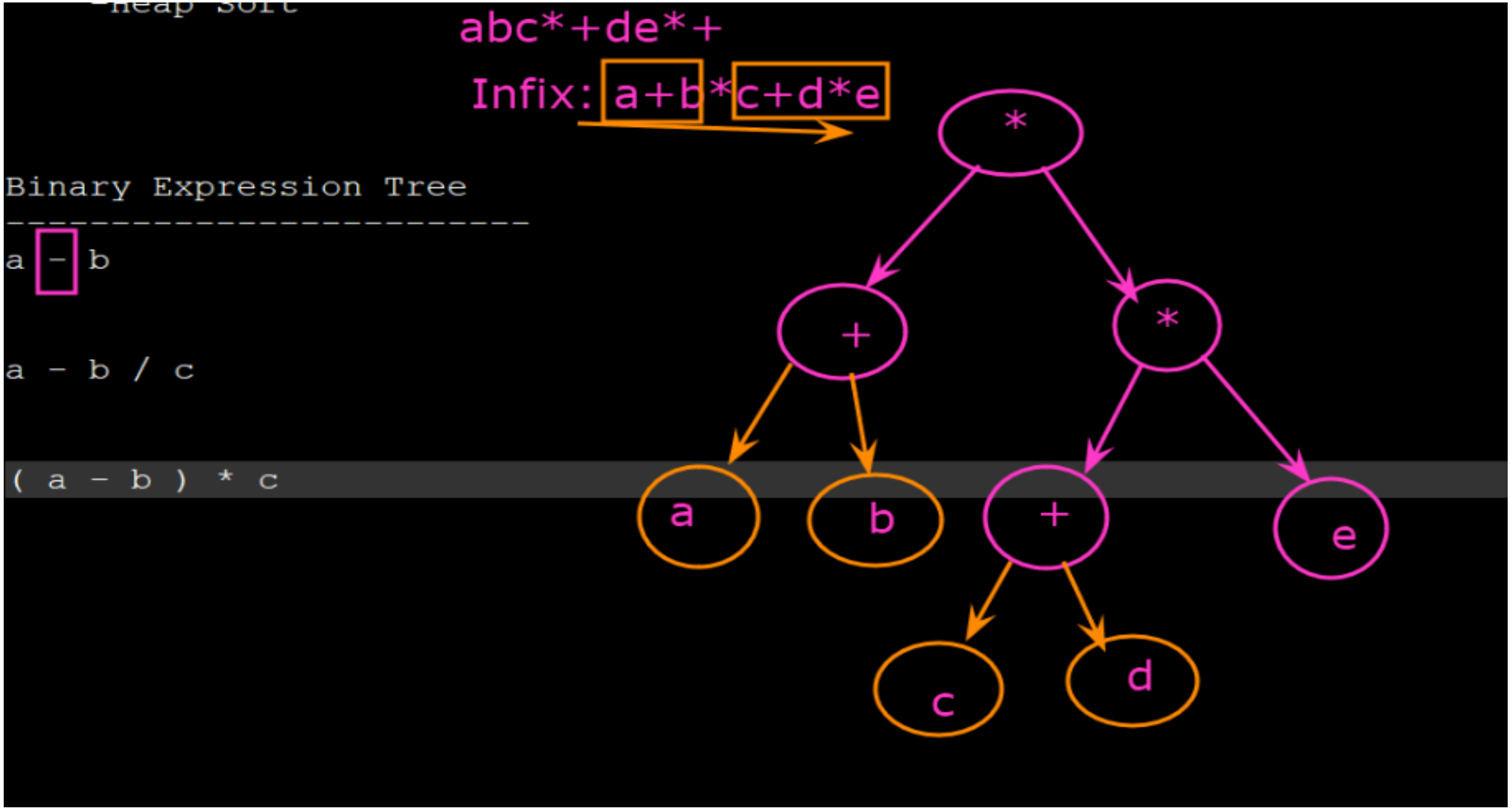
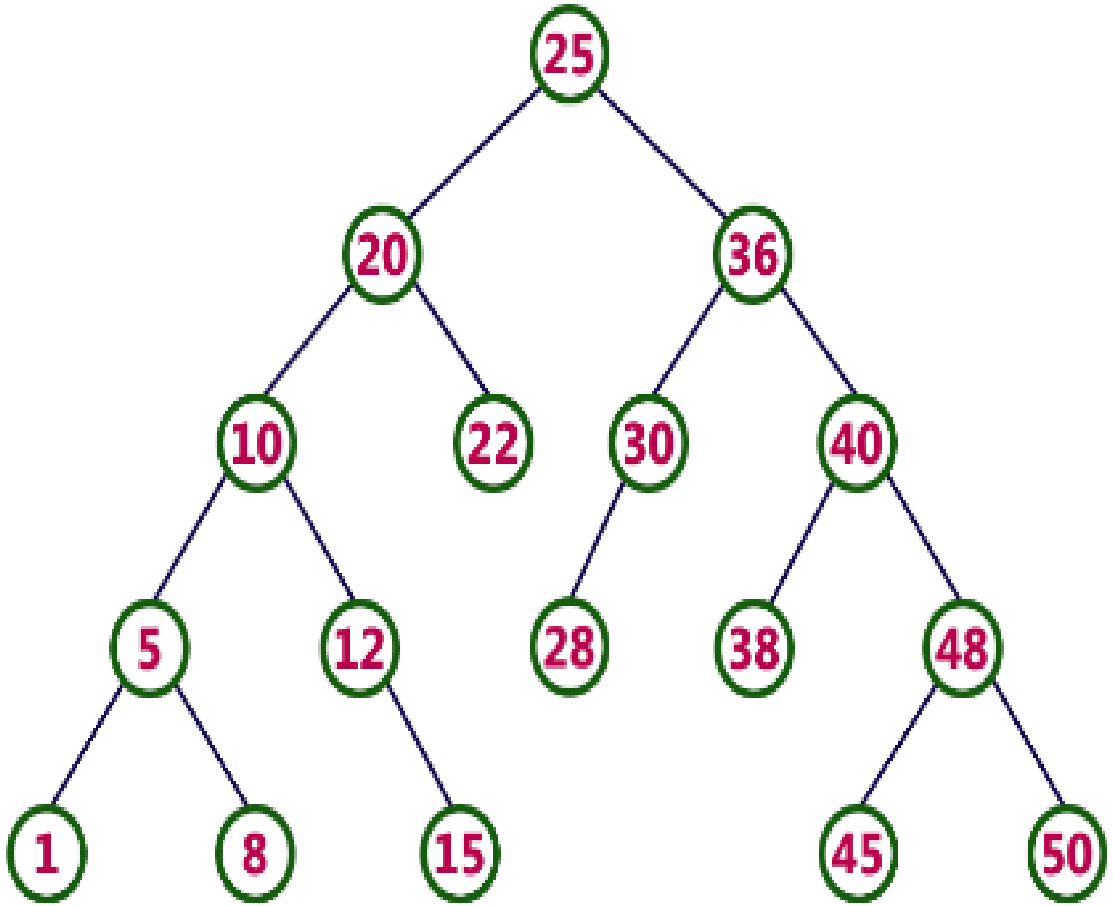
Algorithms & Data Structure

Kiran Waghmare

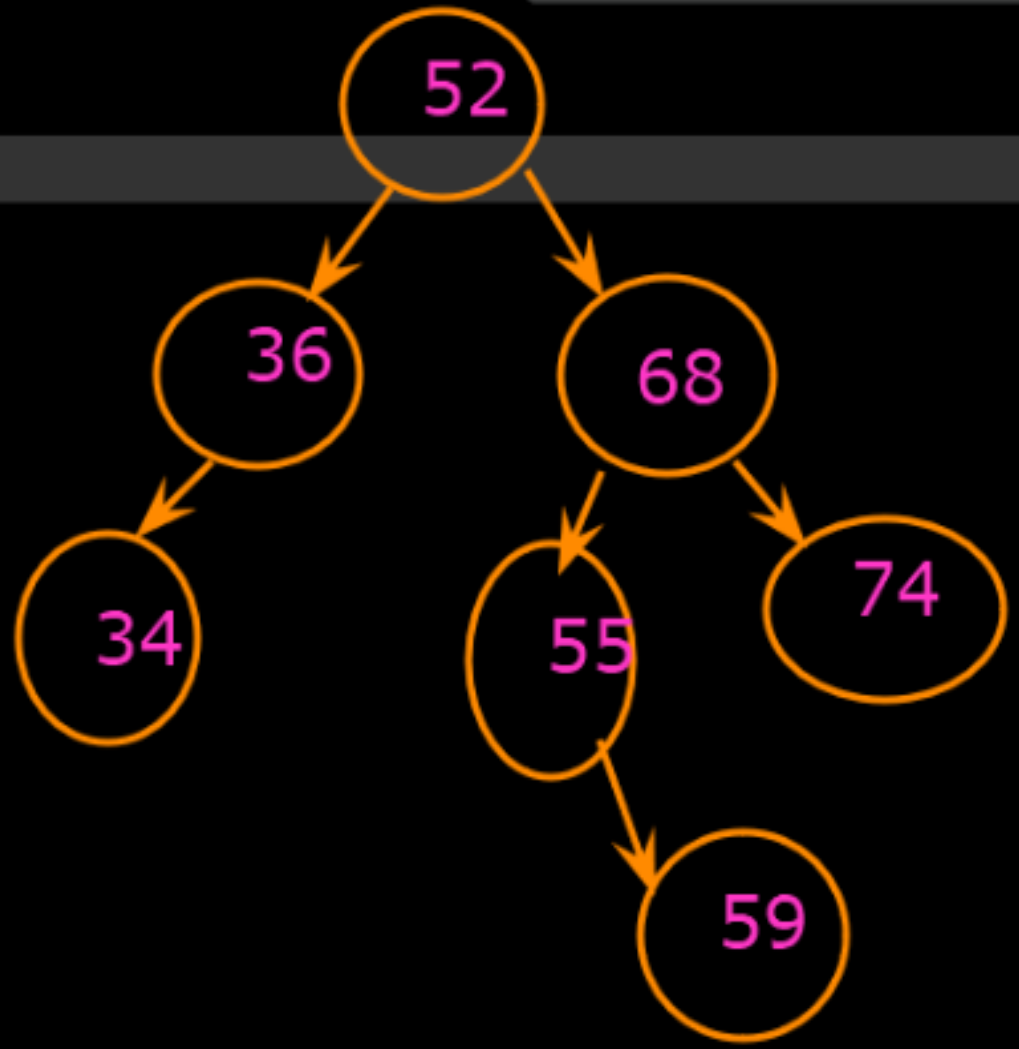

Binary Expression Tree
a - b
a - b / c
(a - b) * c
-
a
/
b
c
*
-
c
a
b


abc*+de*+
Infix: a+b*c+d*e
Binary Expression Tree
a - b
a - b / c
(a - b) * c
*
+
*
a
b
+
e
c
d



Every Binary Search Tree is a binary tree but all the Binary Trees need not to be binary search trees.

```
Binary Search Tree
----------------------
52, 36, 68, 74, 34,55

First node: Root
value < Root ---> LC
value > Root ---->RC
```

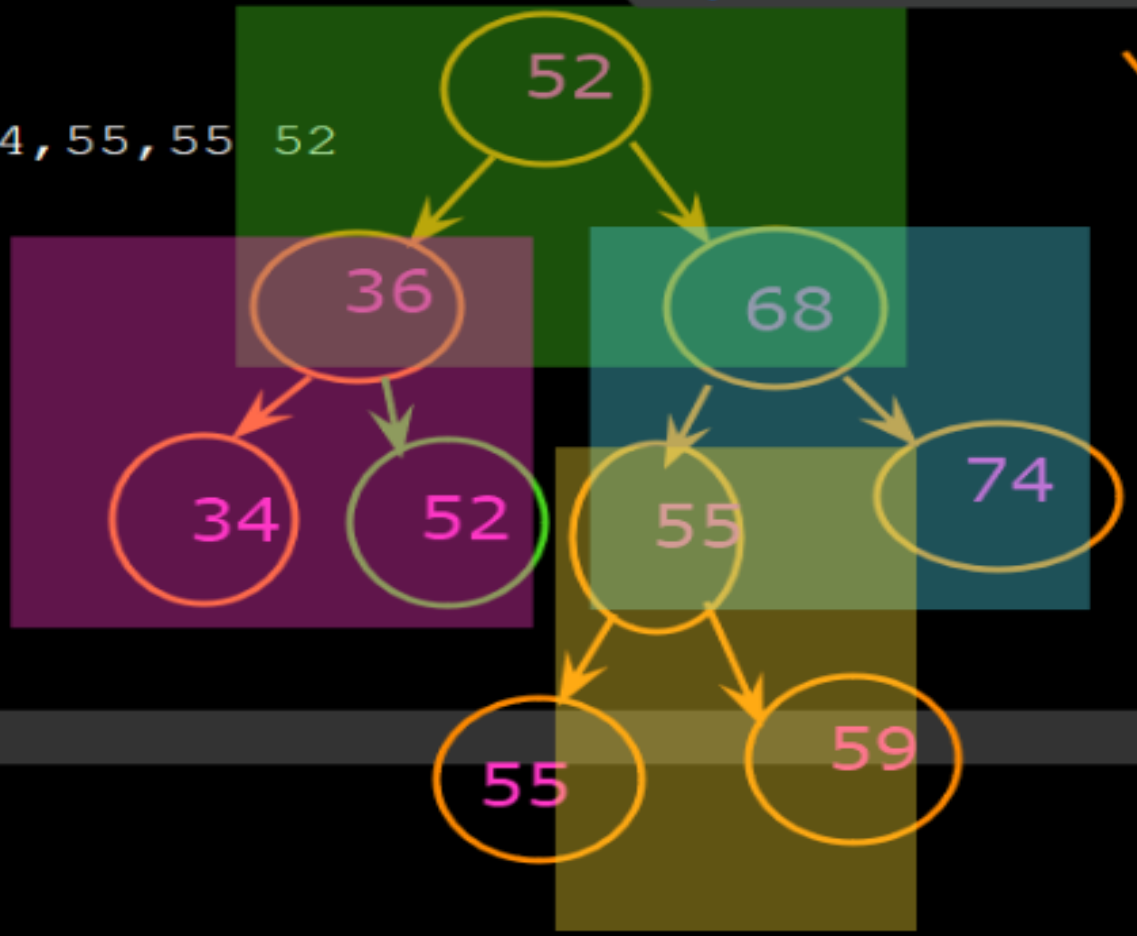


Operations on a Binary Search Tree

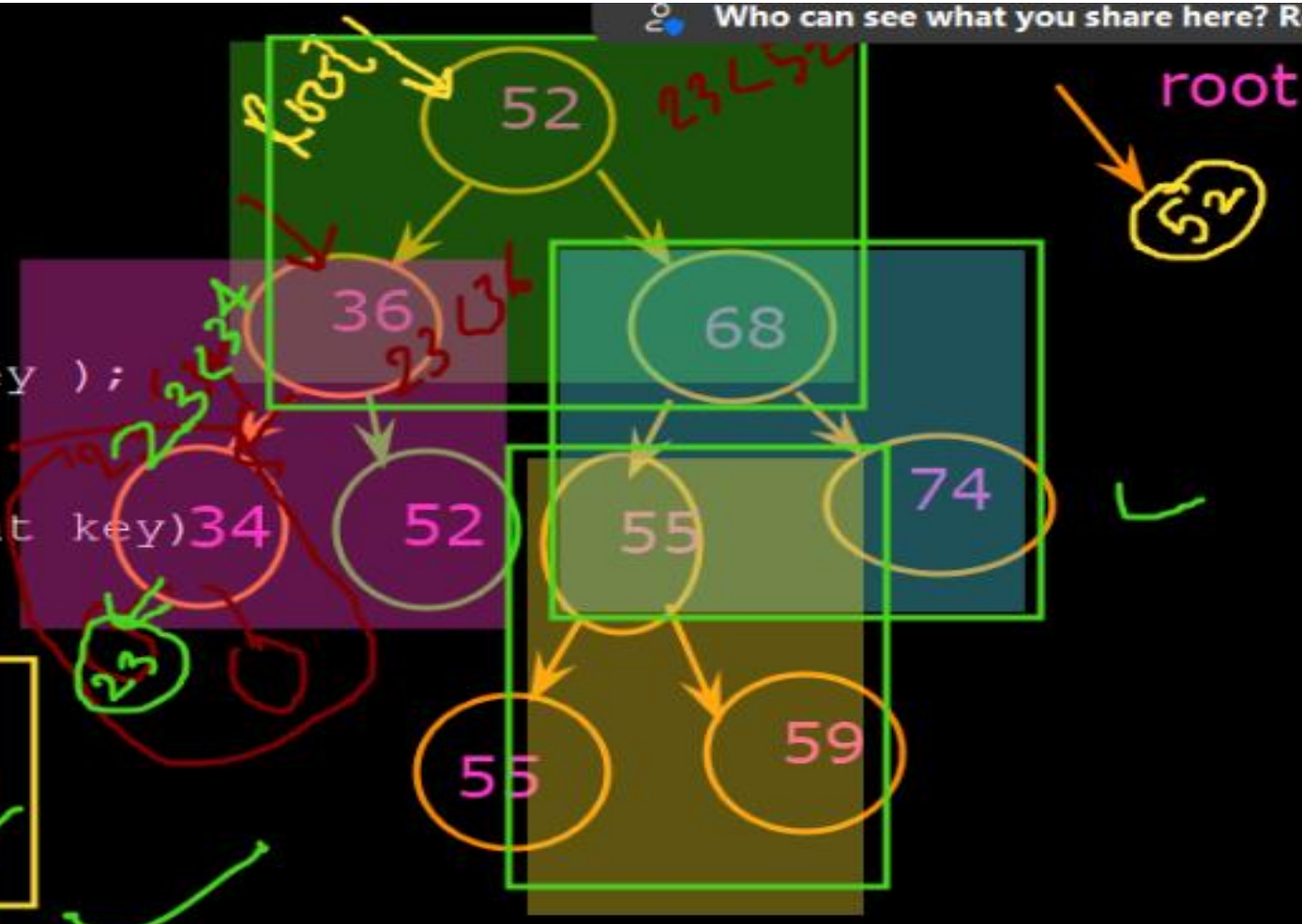
- The following operations are performed on a binary earch tree...
  - Search
  - Insertion
  - Deletion
  - Traversal

Who can see what you share here? Record

```
Binary Search Tree
----------------------
Insertion : 52, 36, 68, 74, 34,55,55 52

First node: Root
value <= Root ---> LC
value > Root ----->RC



Node root;
root=null;//Constructor

    insert (int  key);
```



```
Node root;
root=null;//Constructor

void insert(int key)
{
    root = insertData(root,key );
}

Node insertData(Node root, int key)
{
//Empty tree
    if(root == null)
    {
        root = new Node(key);
        return root;
    }
    //Existing tree
    if(key <=root.data)
        root.left = insertData(root.left,key);
    else if(key >root.data)
        root.right = insertData(root.right,key);
        return root;
```



Deletion in BST:

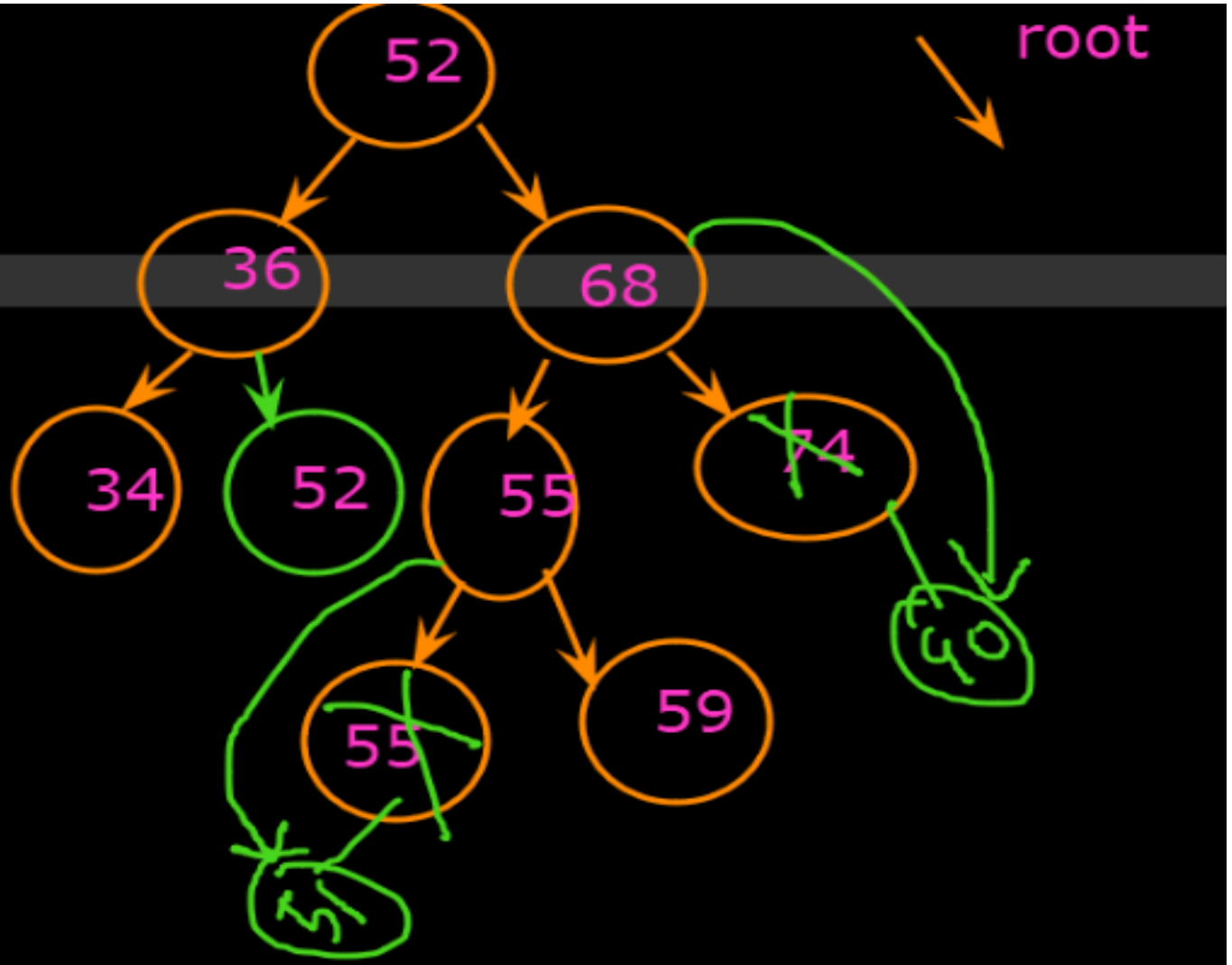
-keep track of child nodes

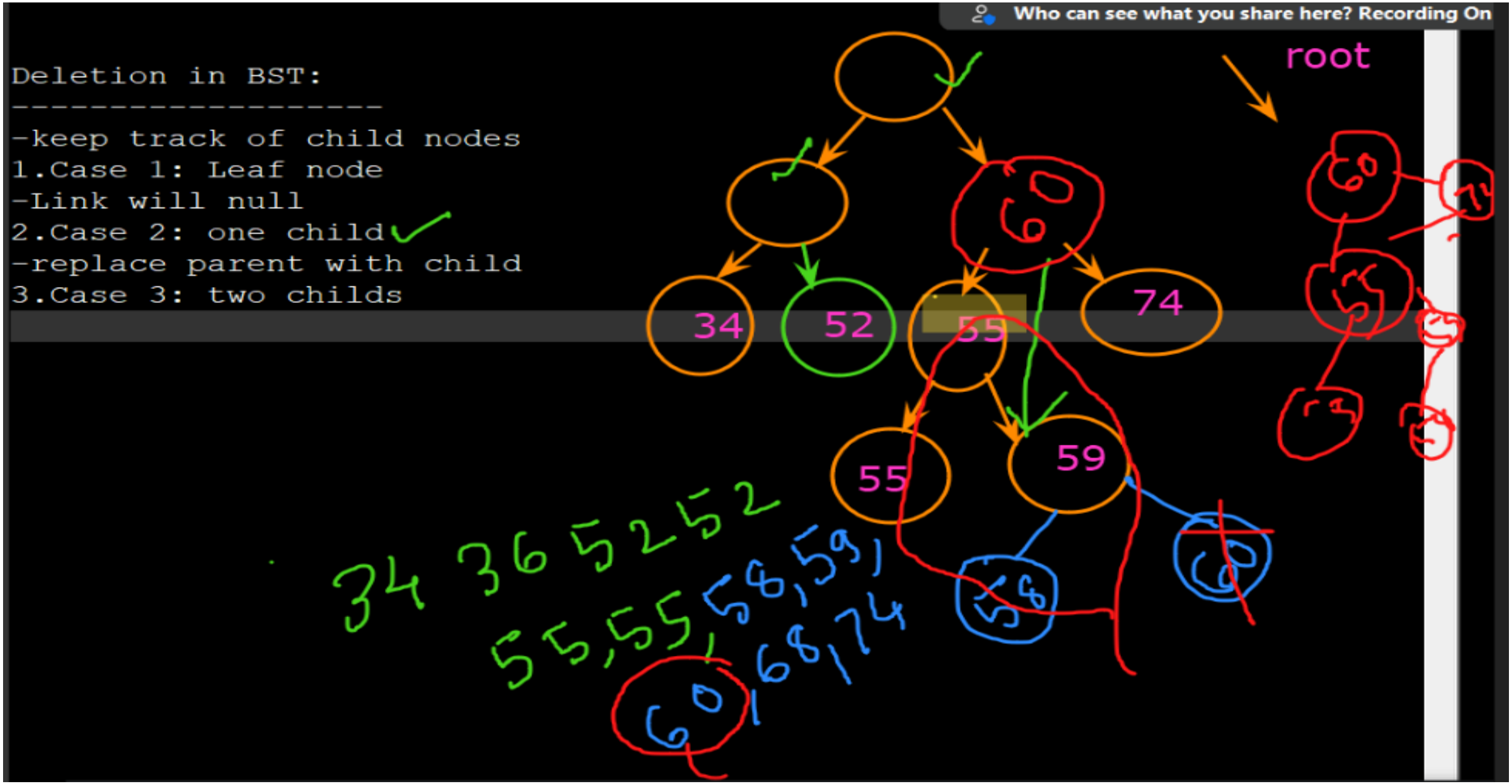
1.Case 1: Leaf node

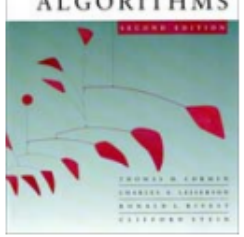
-Link will null

2.Case 2: one child

3.Case 3: two childs




Who can see what you share here? Recording On
Deletion in BST:
-keep track of child nodes
1.Case 1: Leaf node
-Link will null
2.Case 2: one child
-replace parent with child
3.Case 3: two childs
root
34
52
55
74
55
59
34 36 52 52
55,55,58,59,
68,74



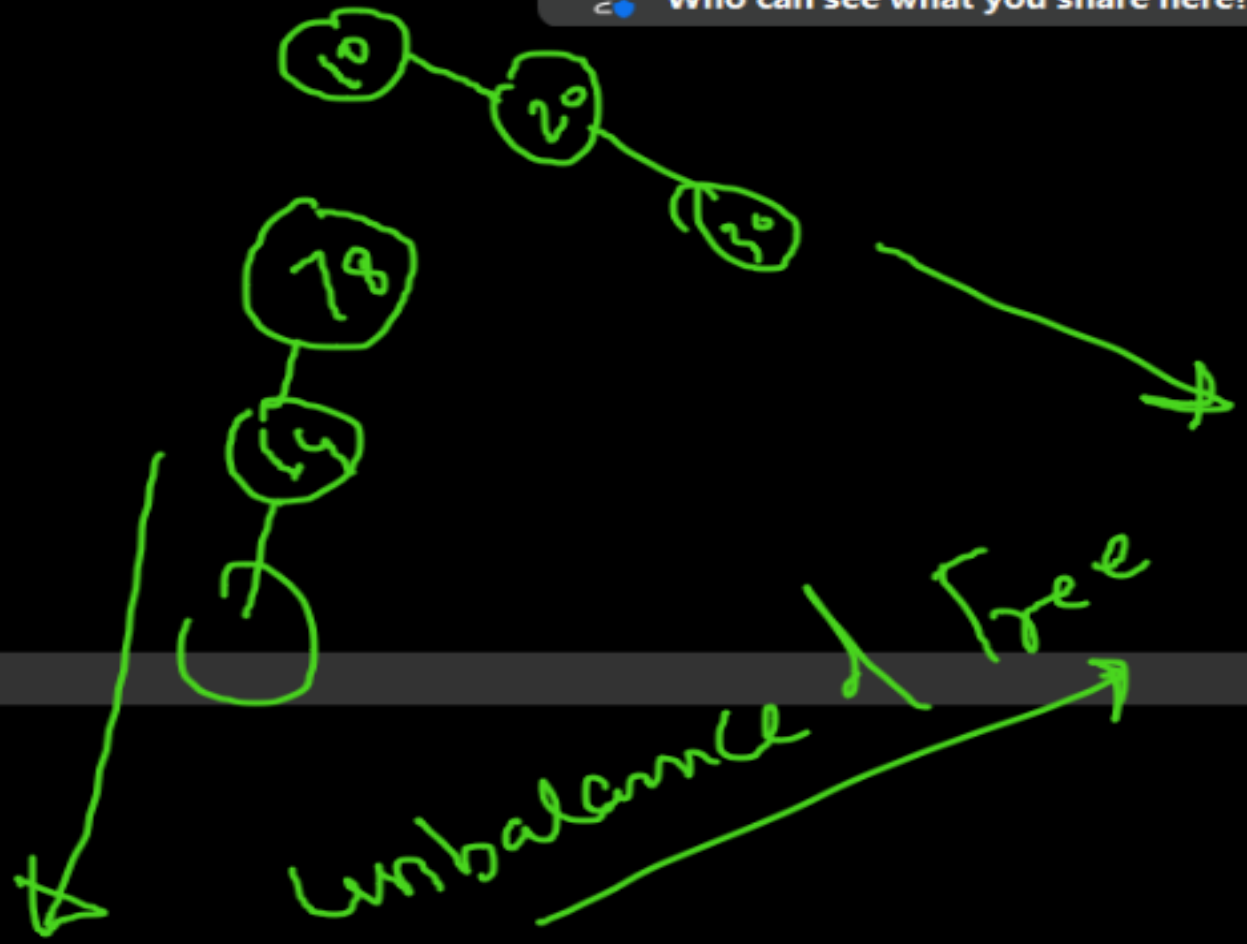
Balanced search trees

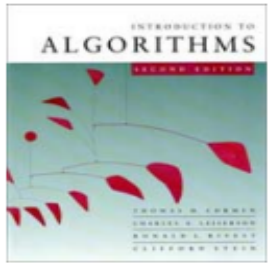
Balanced search tree: A search-tree data structure for which a height of $O(\lg n)$ is guaranteed when implementing a dynamic set of n items.

Examples:

- AVL trees
- 2-3 trees
- 2-3-4 trees
- B-trees
- Red-black trees

```
    int minv = root.data;
    while (root.left != null)
    {
        minv = root.left.data;
        root =root.left;

    }
    return minv;

}

BST:

BST1 : 10,20,30,40,50,60,70

Bst2 : 78,64,59,35,12,8
```



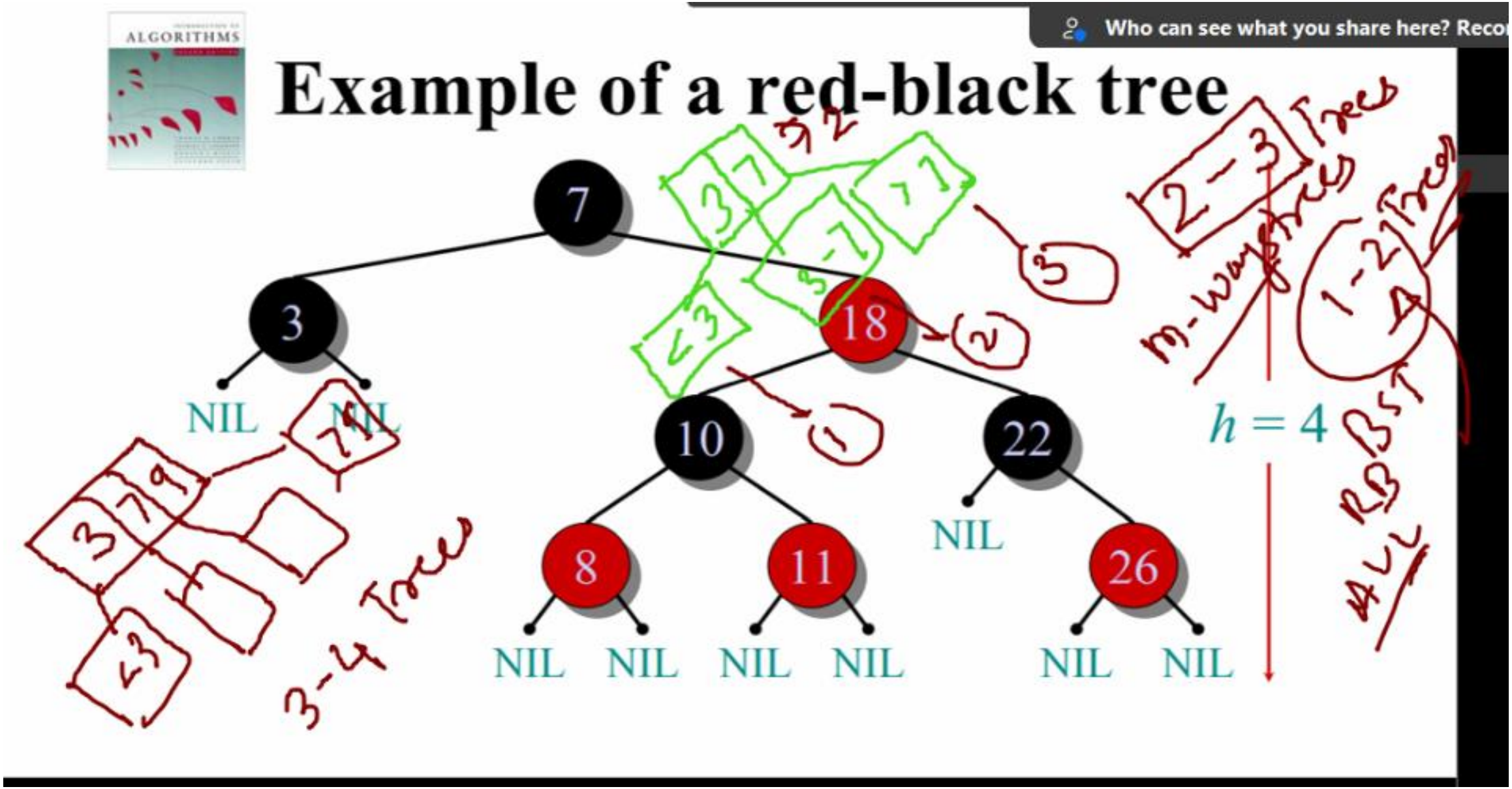


Red-black trees

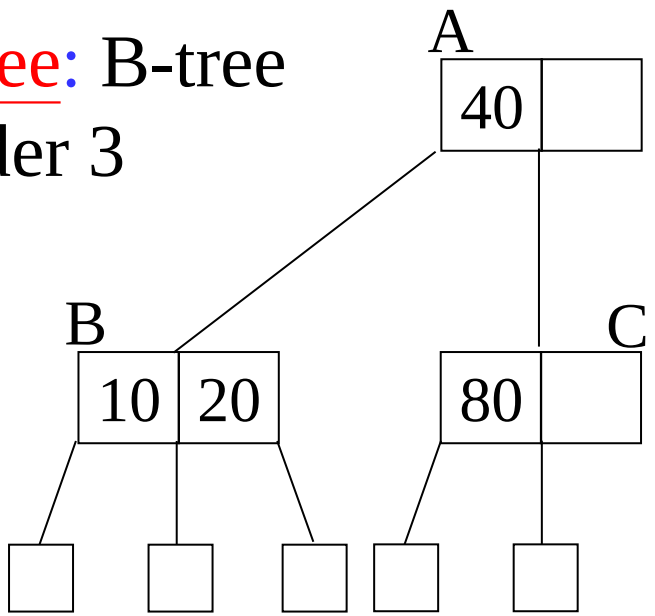
This data structure requires an extra one-bit color field in each node.

Red-black properties:

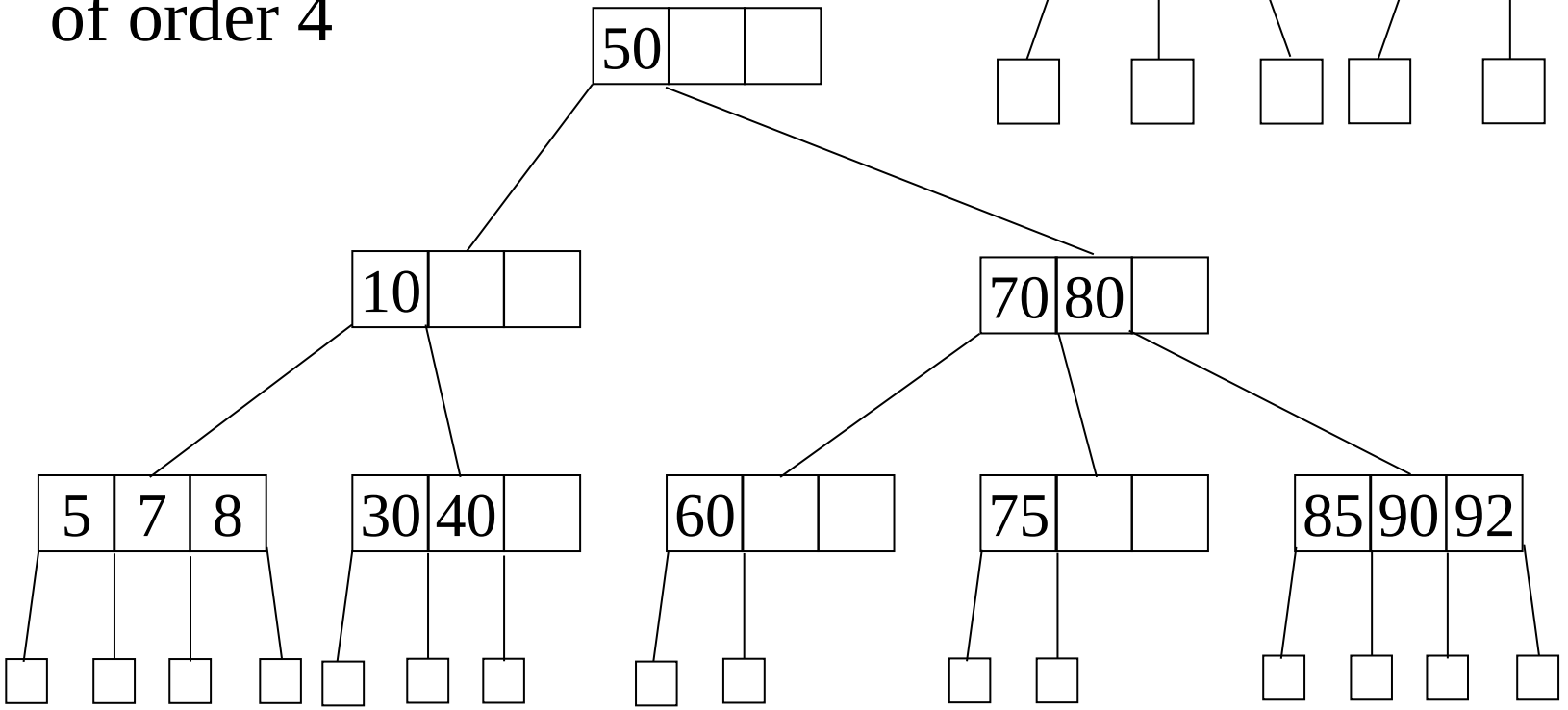
1. Every node is either red or black.
2. The root and leaves (NIL's) are black.
3. If a node is red, then its parent is black.
4. All simple paths from any node x to a descendant leaf have the same number of black nodes = black-height(x).

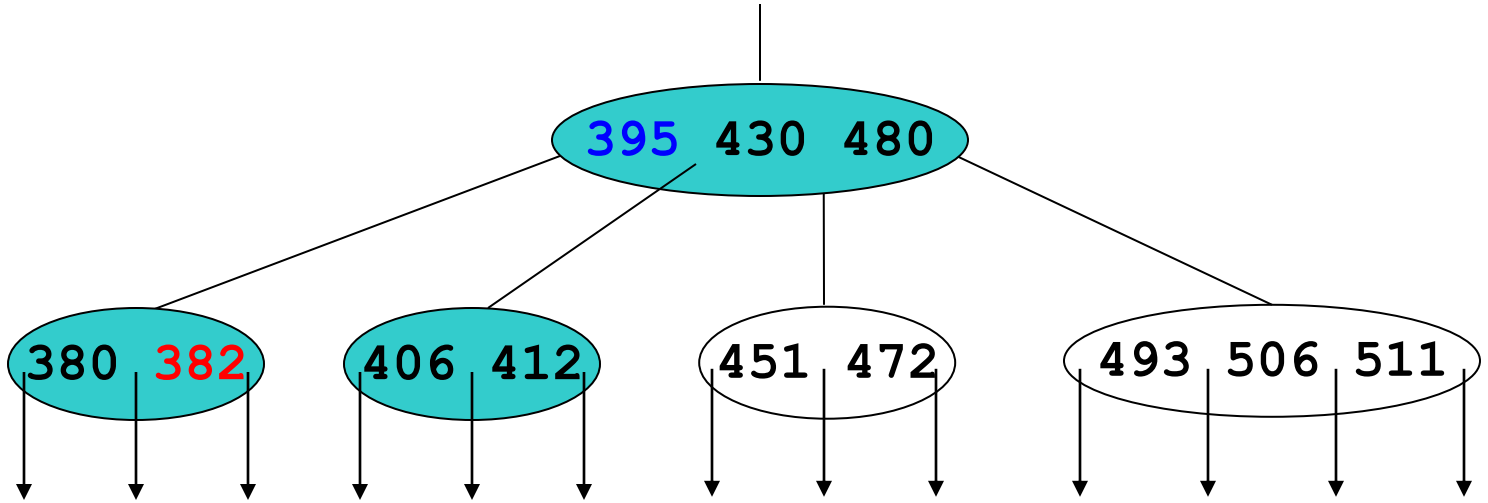

Who can see what you share here? Reco
Example of a red-black tree
7
3
18
NIL
NIL
10
22
8
11
NIL
26
NIL
NIL
NIL
NIL
NIL
NIL
h = 4
3-4 Trees
AVL RB BST

2-3 tree: B-tree of order 3

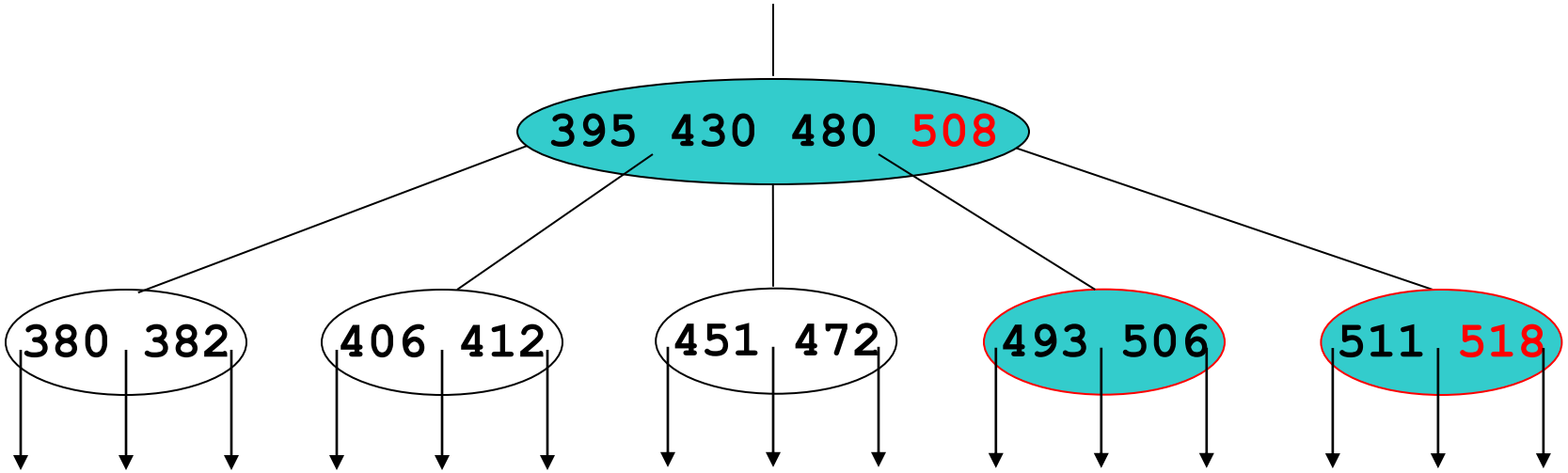


2-3-4 tree: B-tree of order 4





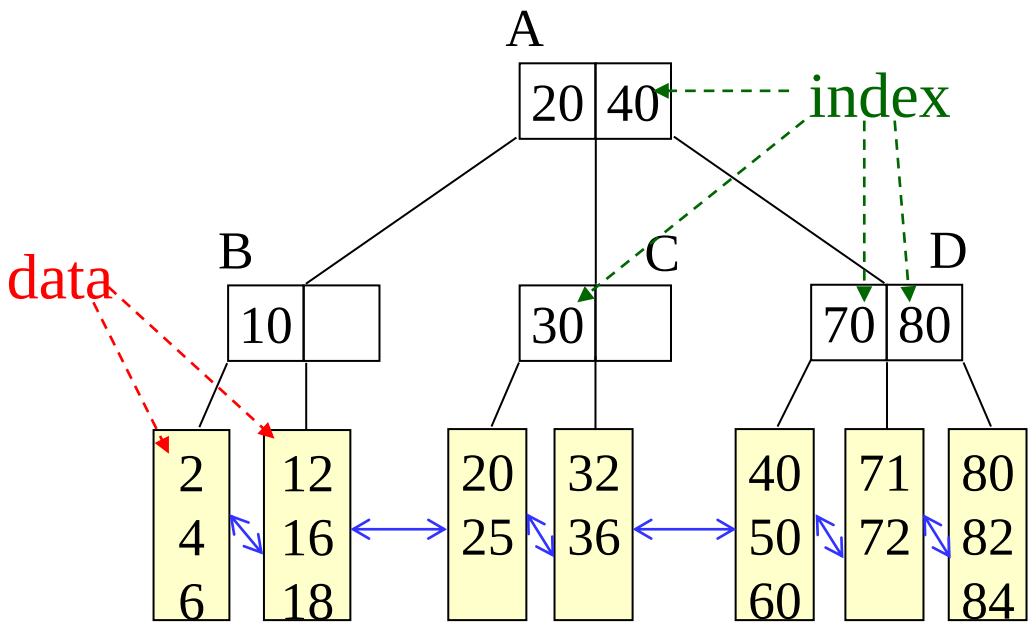
(b) After inserting 382 (Split the full node)



(c) After inserting 518 and 508

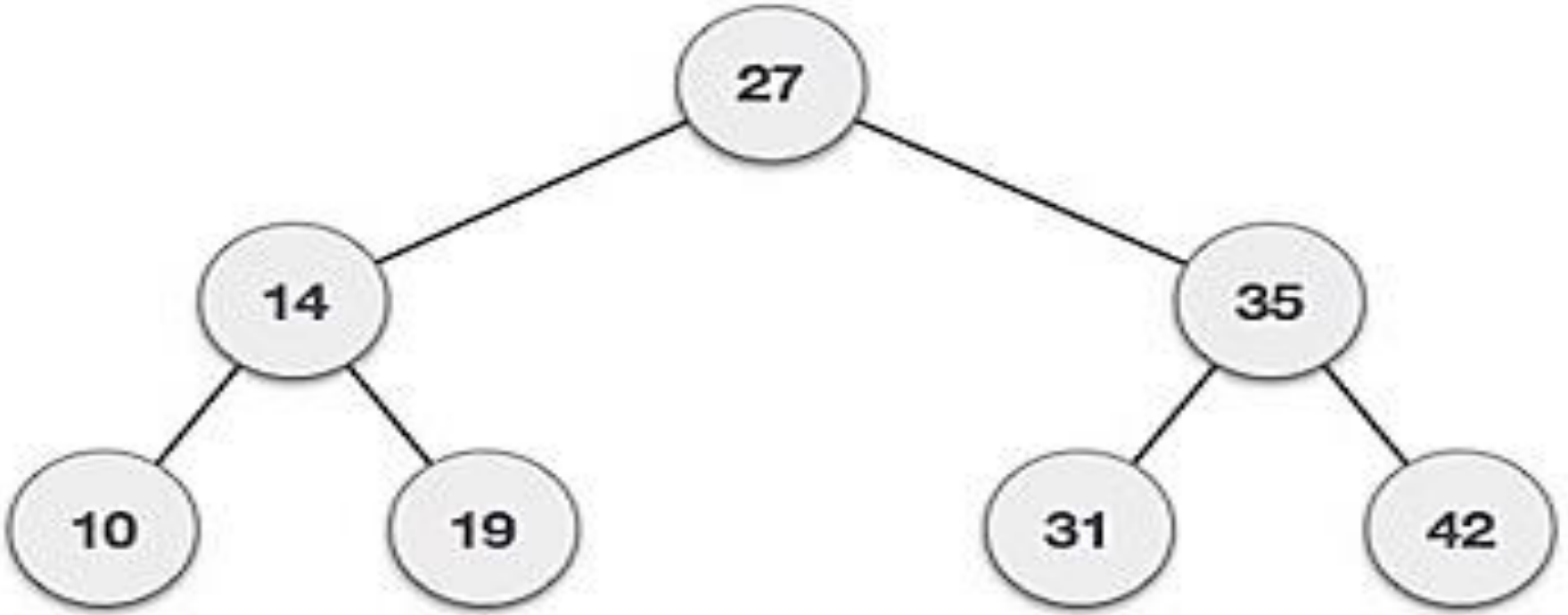
B^+-trees

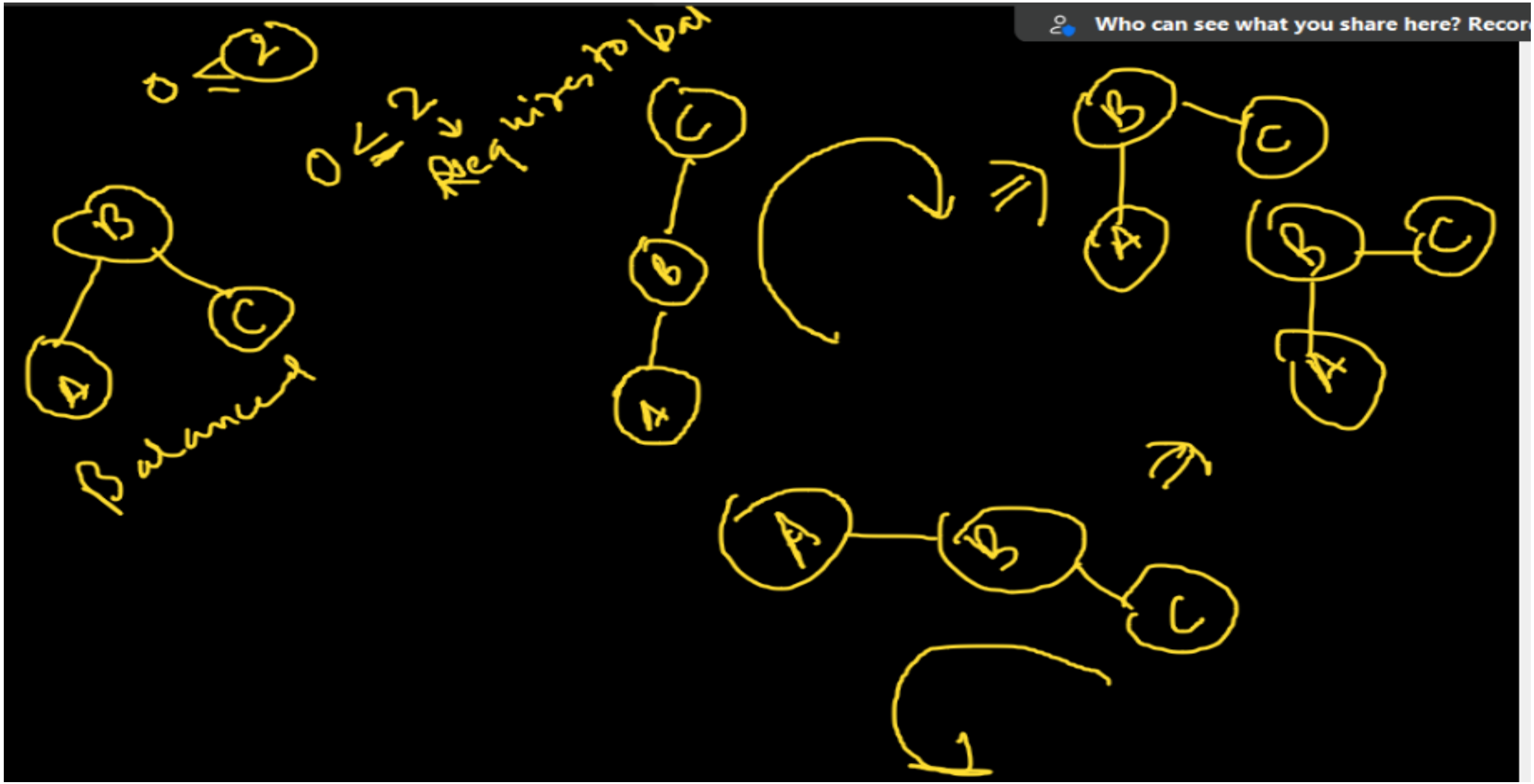
- Index node: internal node, storing keys (not elements) and pointers
- Data node: external node, storing elements (with their keys)
- Data nodes are linked together to form a doubly linked list.



Example:

Example AVL Tree

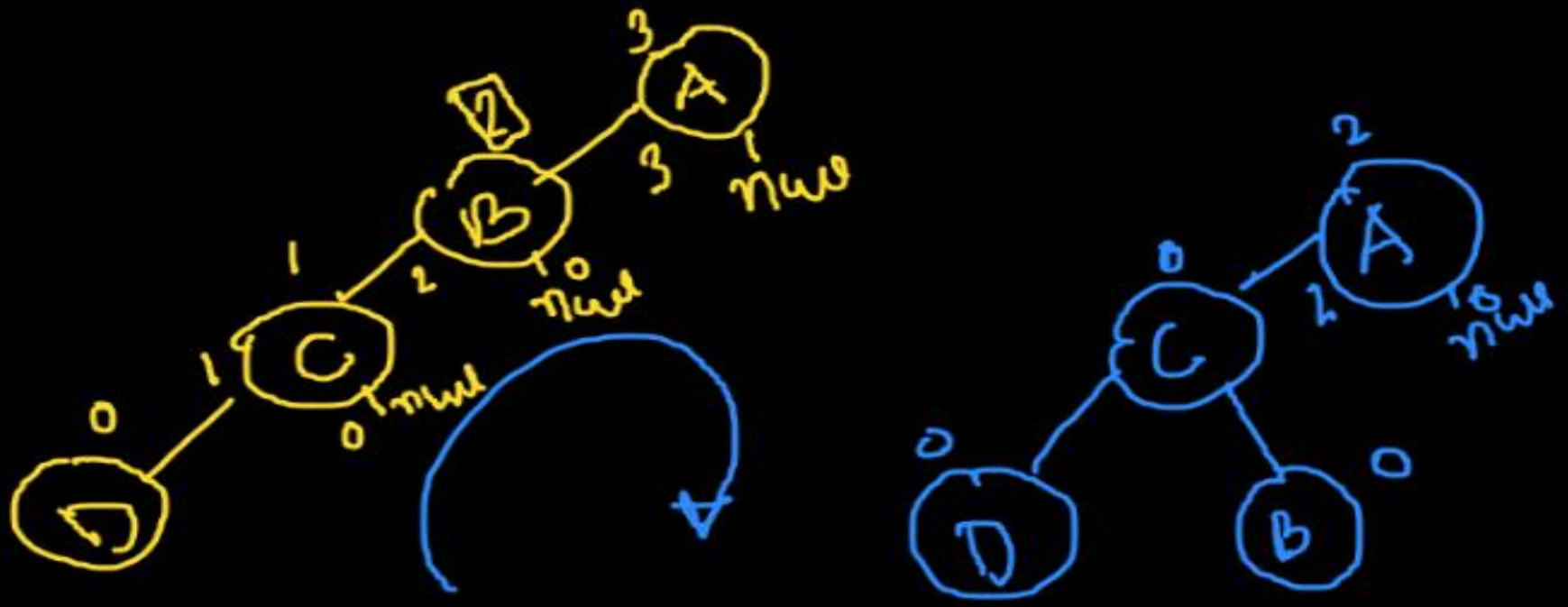



Who can see what you share here? Recor
Balanced
Requires to bal

-Balance Factor:

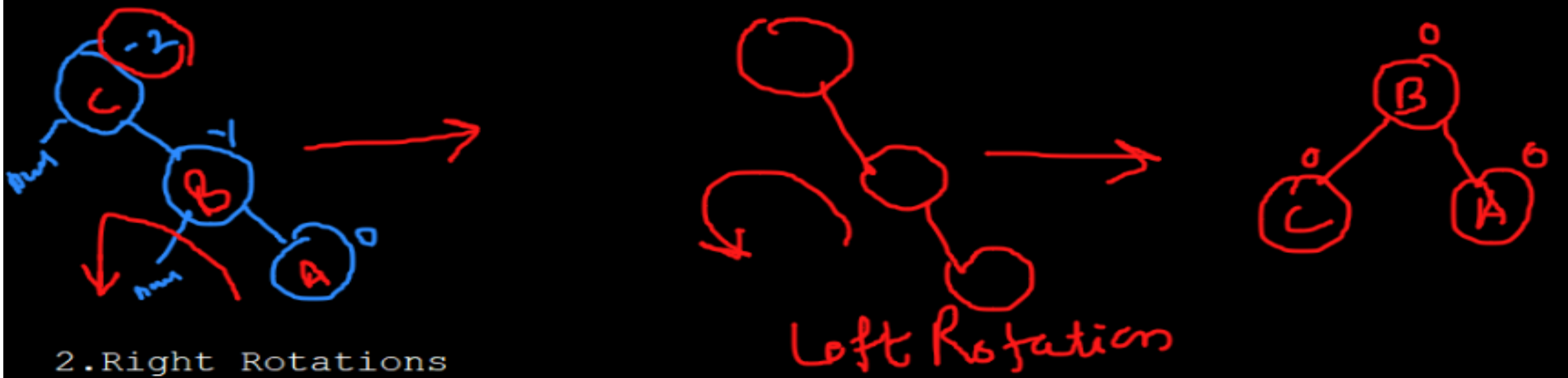
Formula:

BF =height(LST) - height(RST)

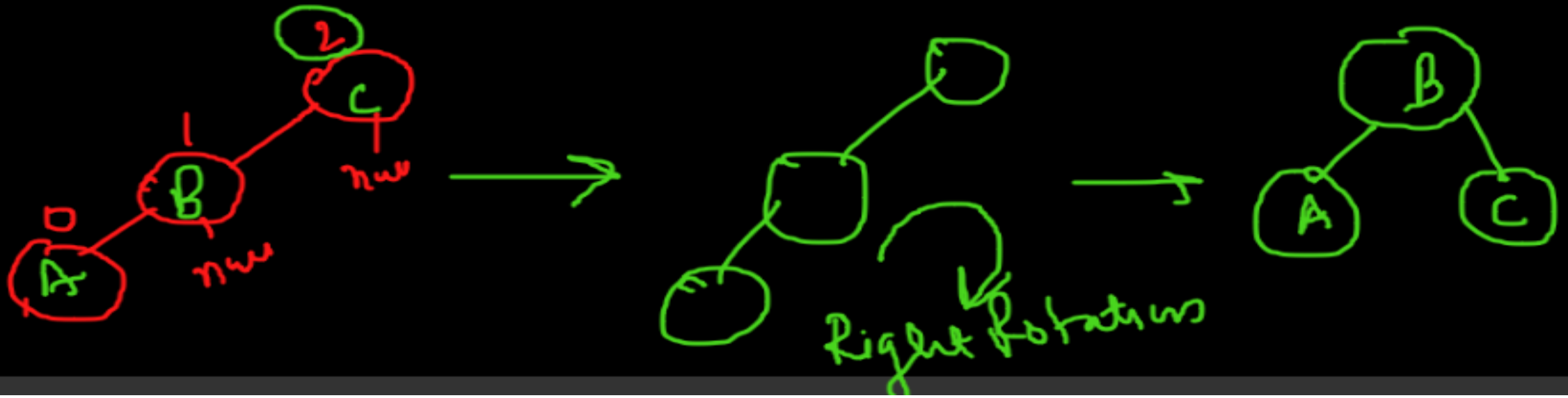


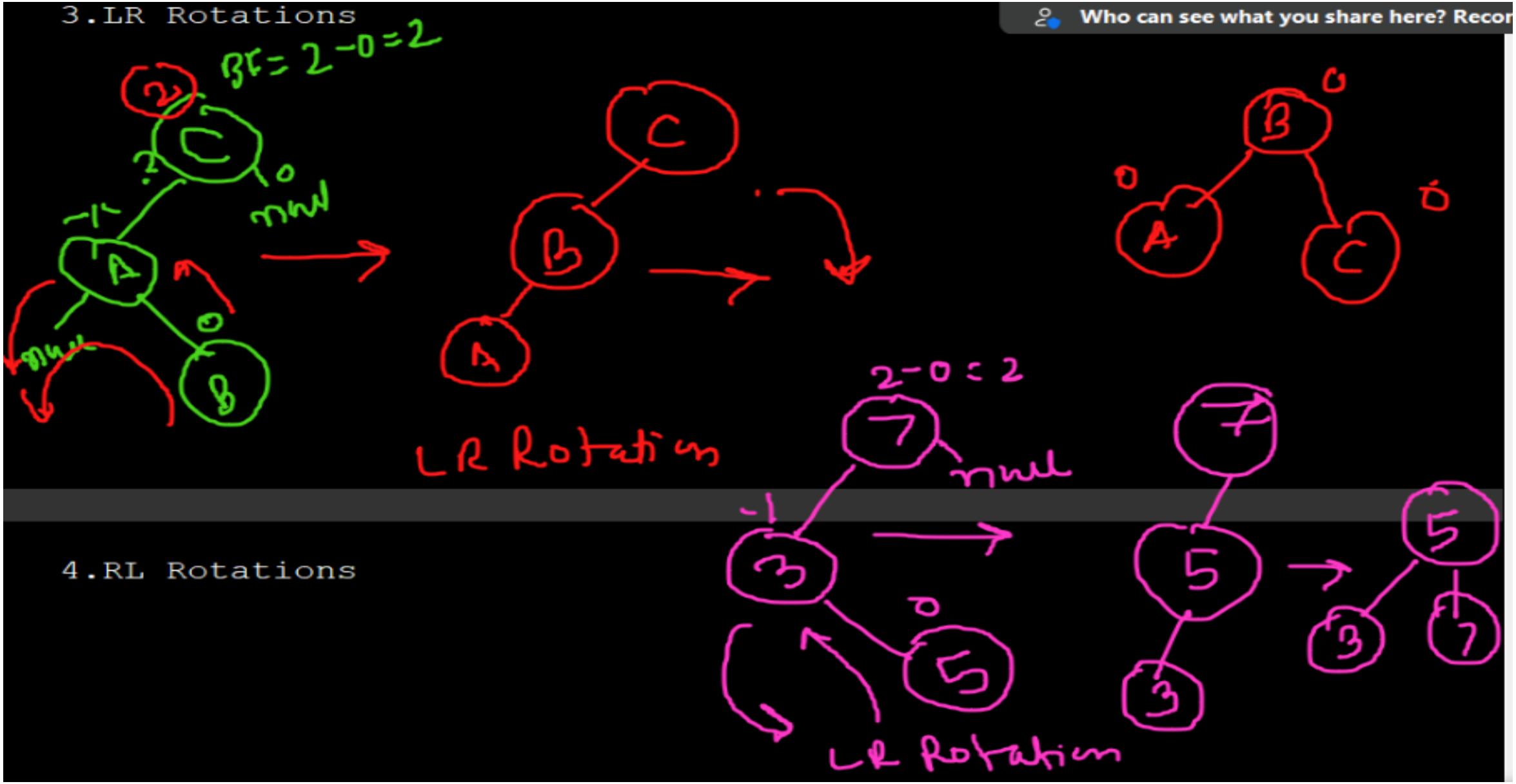
Solution: rotations:

1.Left Rotations



2.Right Rotations




3.LR Rotations
Who can see what you share here? Recor
BF= 2-0=2
2
C
-1
A
0
null
B
LR Rotation
C
B
A
B
0
A
C
0
0
2-0=2
7
null
-1
3
0
5
LR Rotation
7
5
3
5
3
7
4.RL Rotations

4.RL Rotations