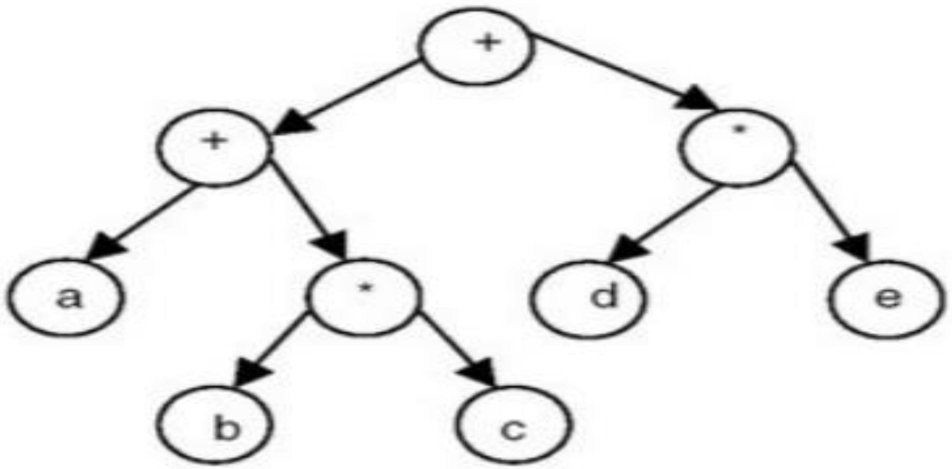
Algorithms & Data Structure

Kiran Waghmare

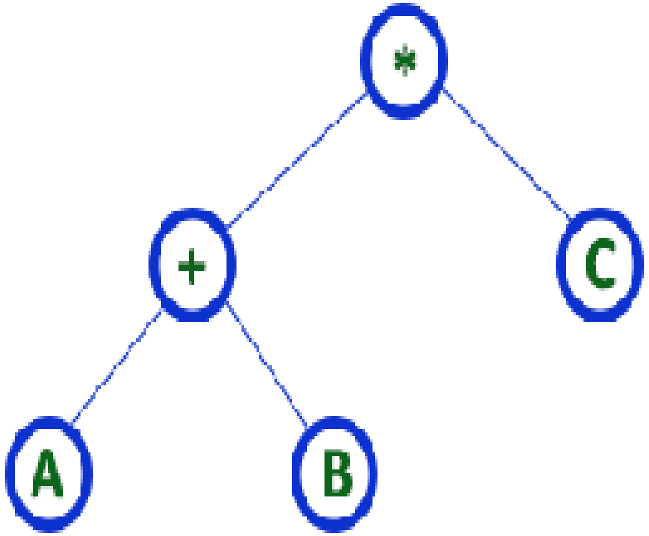
Expression Binary Tree Traversal

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.

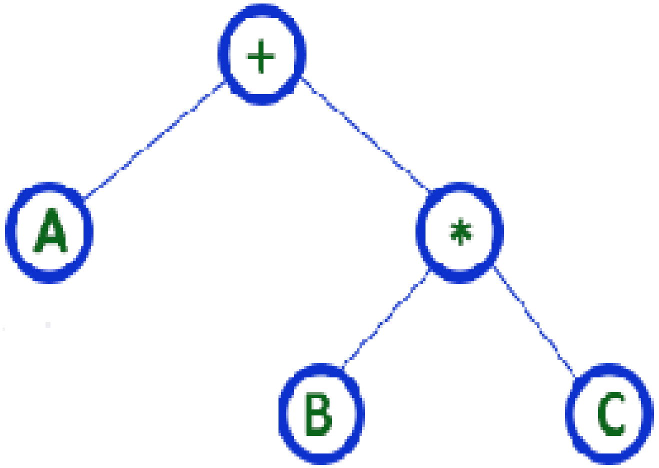


Inorder : a + b * c + d * e
postorder : abc*+de*+

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

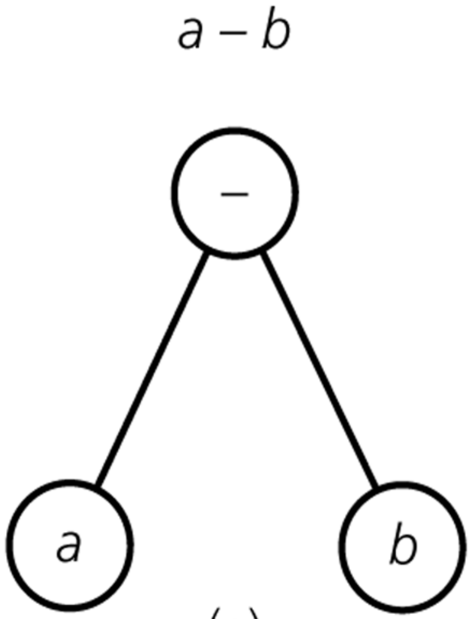


(A + B) * C

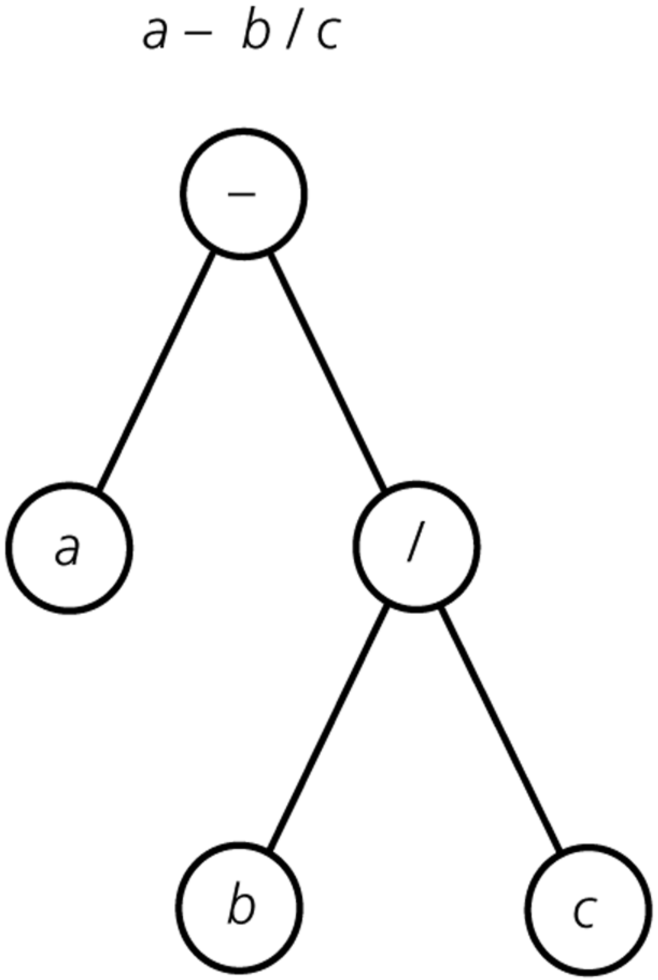


A + B * C

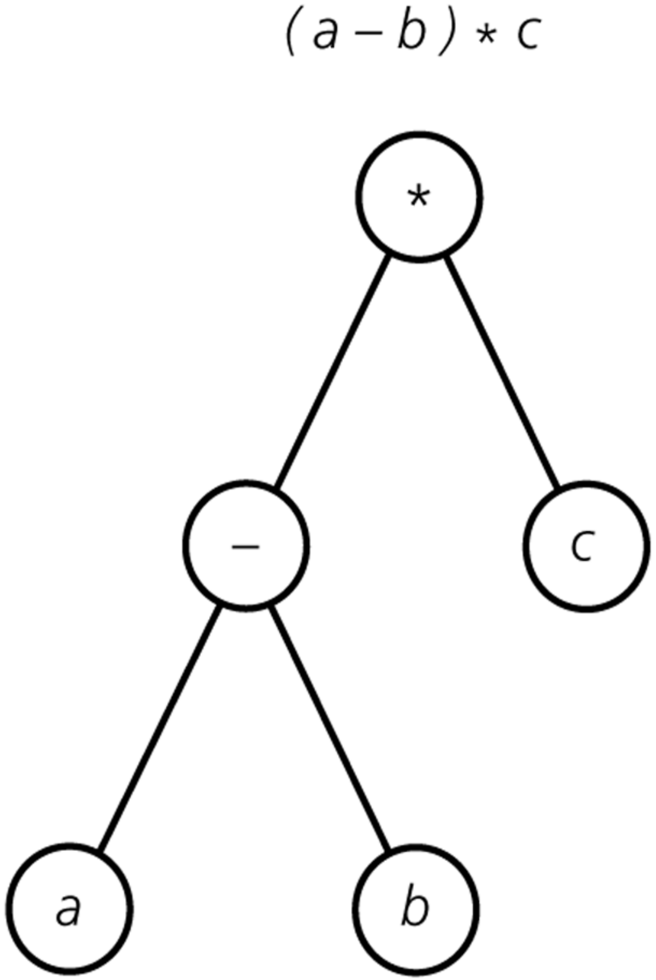
Binary Tree – Representing Algebraic Expressions



(a)



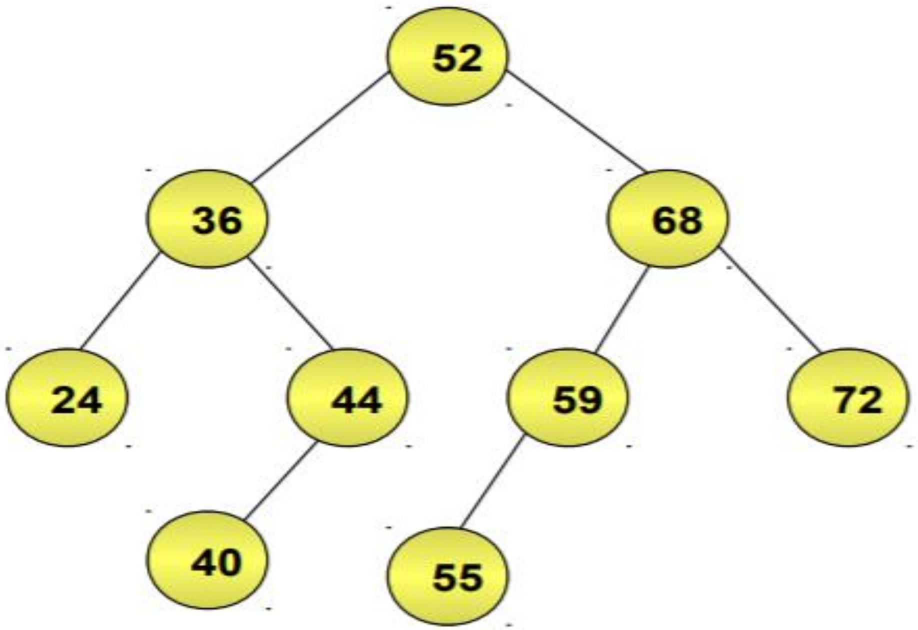
(b)



(c)

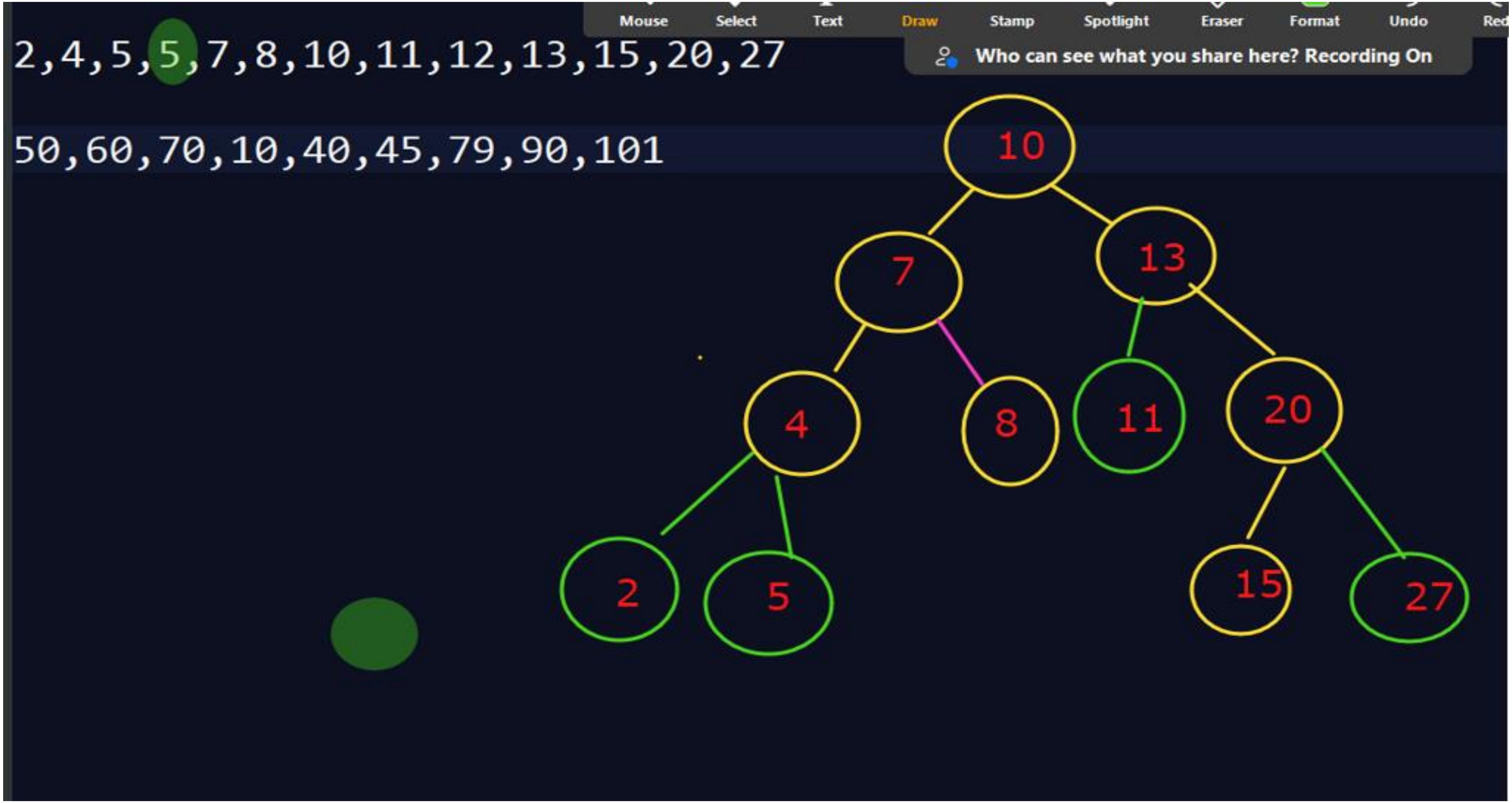
Binary Search Tree

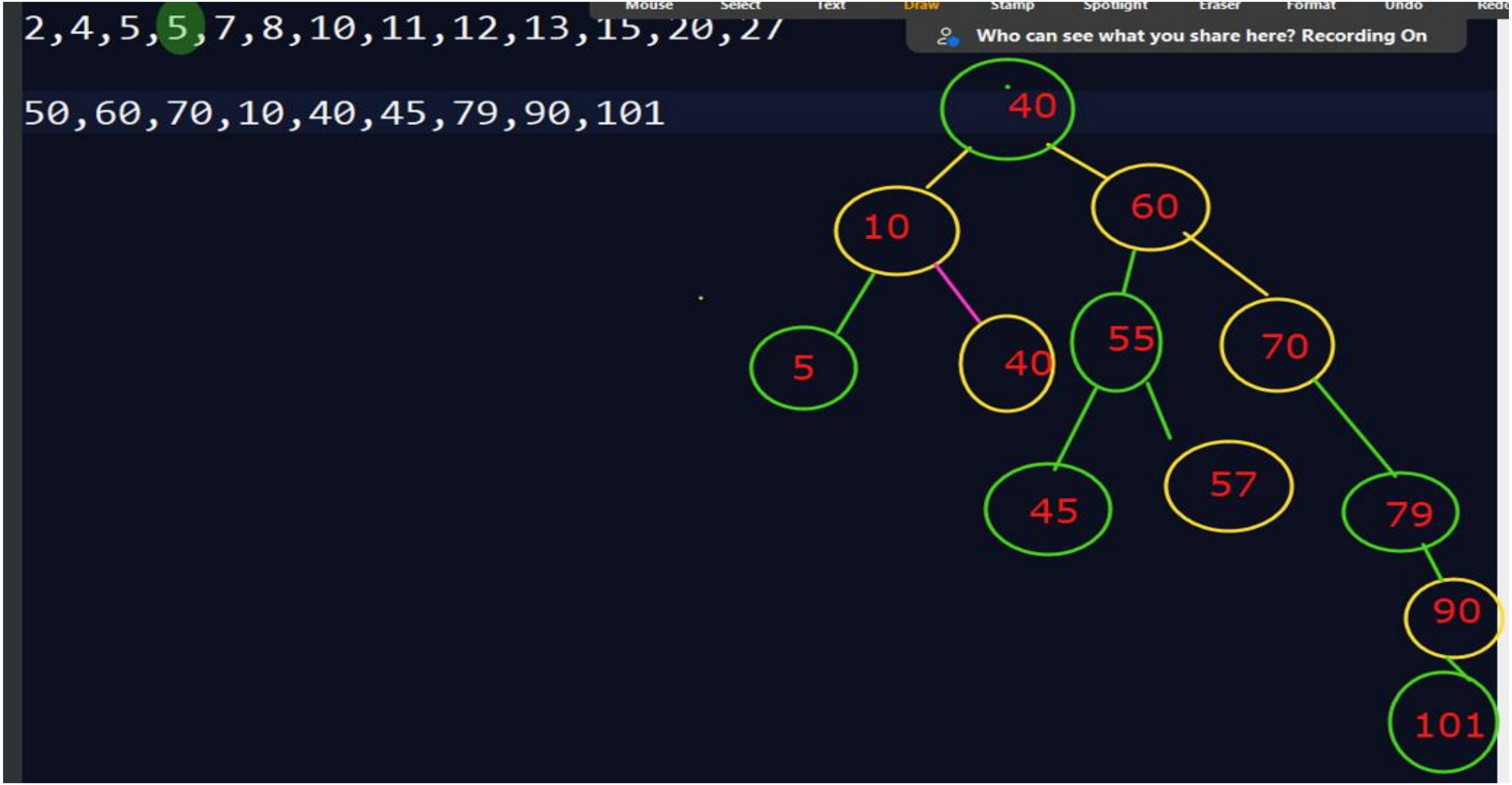
- Binary search tree is a binary tree in which every node satisfies the following conditions:
  - All values in the left subtree of a node are less than the value of the node.
  - All values in the right subtree of a node are greater than the value of the node.
- The following is an example of a binary search tree.



Operations on a Binary Search Tree

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


Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Who can see what you share here? Recording On
2,4,5,5,7,8,10,11,12,13,15,20,27
50,60,70,10,40,45,79,90,101
10
7
13
4
8
11
20
2
5
15
27

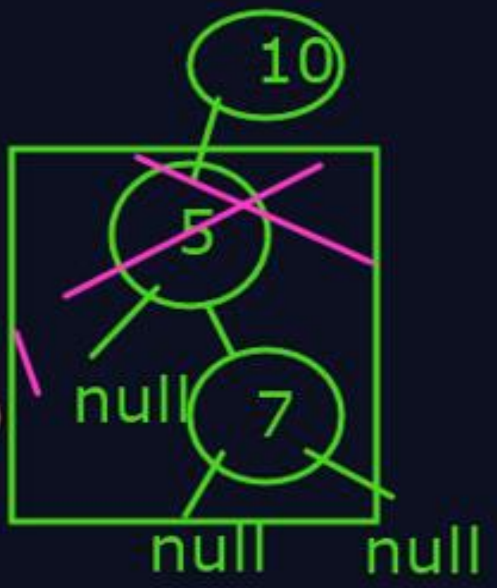

2,4,5,5,7,8,10,11,12,13,15,20,27
50,60,70,10,40,45,79,90,101
Who can see what you share here? Recording On
40
10
60
5
40
55
70
45
57
79
90
101

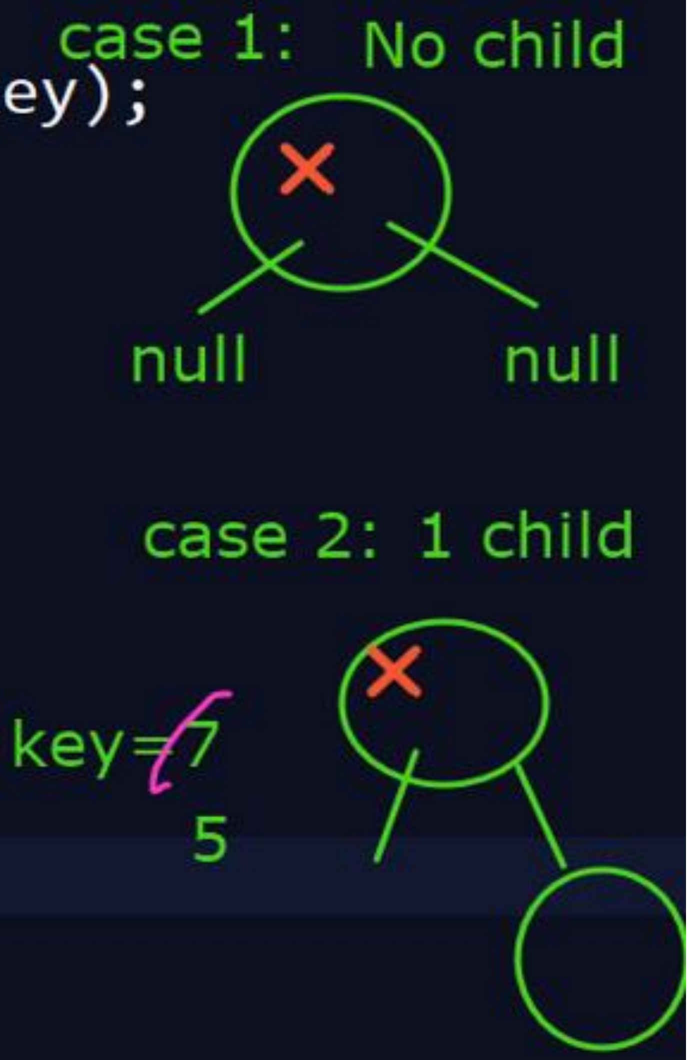
```
    if(key <= root.data)
        root.left = deletedata(root.left,key);
    else if(key > root.data)
        root.right = deletedata(root.right,key);


    //Code for case 1 & 2

    if(root.left == null)
        return root.right;
    else if(root.right == null)
        return root.left;

    //code
}
```





```
Node deletedata(Node root, int key)
{
    //Empty tree
    if(root == null)
        return root;

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

    else{

    //Code for case 1 & 2

    if(root.left == null)
        return root.right;
    else if(root.right == null)
        return root.left;
```

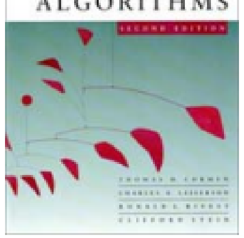
```
    root.data = minvalue(root.right)

    //deleting an element in Inorder and
    //replace it with successor node
    root.right = deletedata(root.right,root.data);
    }

    return root;

}

int minvalue(Node root)
{
    int x = root.data;
    while(root.left != null)
    {
        x =root.left.data;
        root = root.left;
    }
    return x;
```

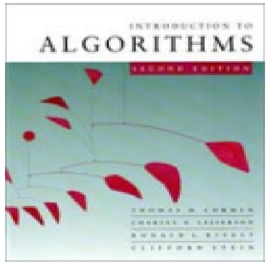


Balanced search trees

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

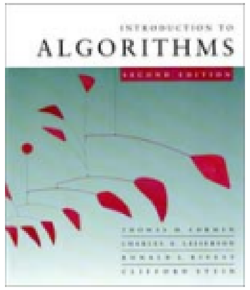


Red-black trees

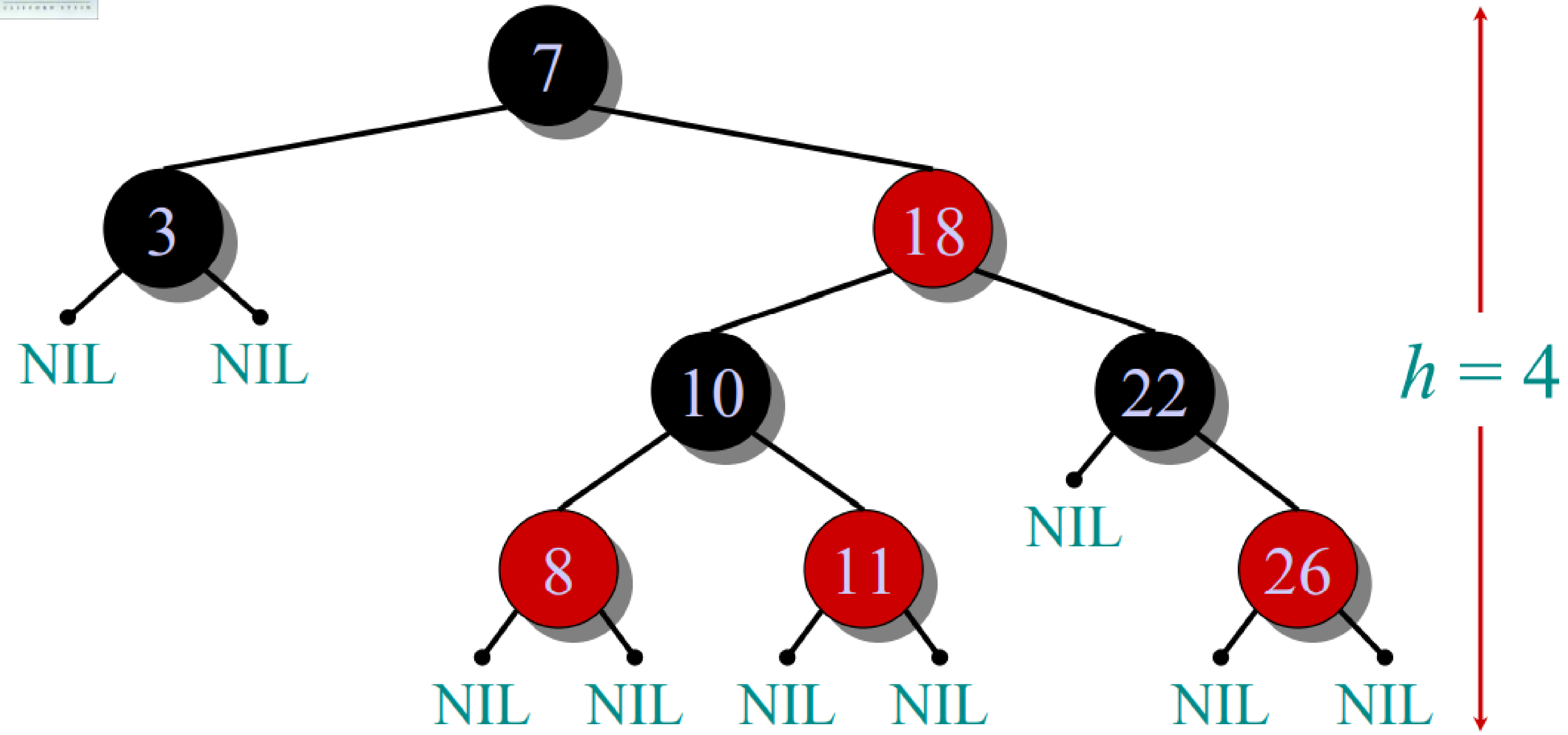
This data structure requires an extra one-bit color field in each node.

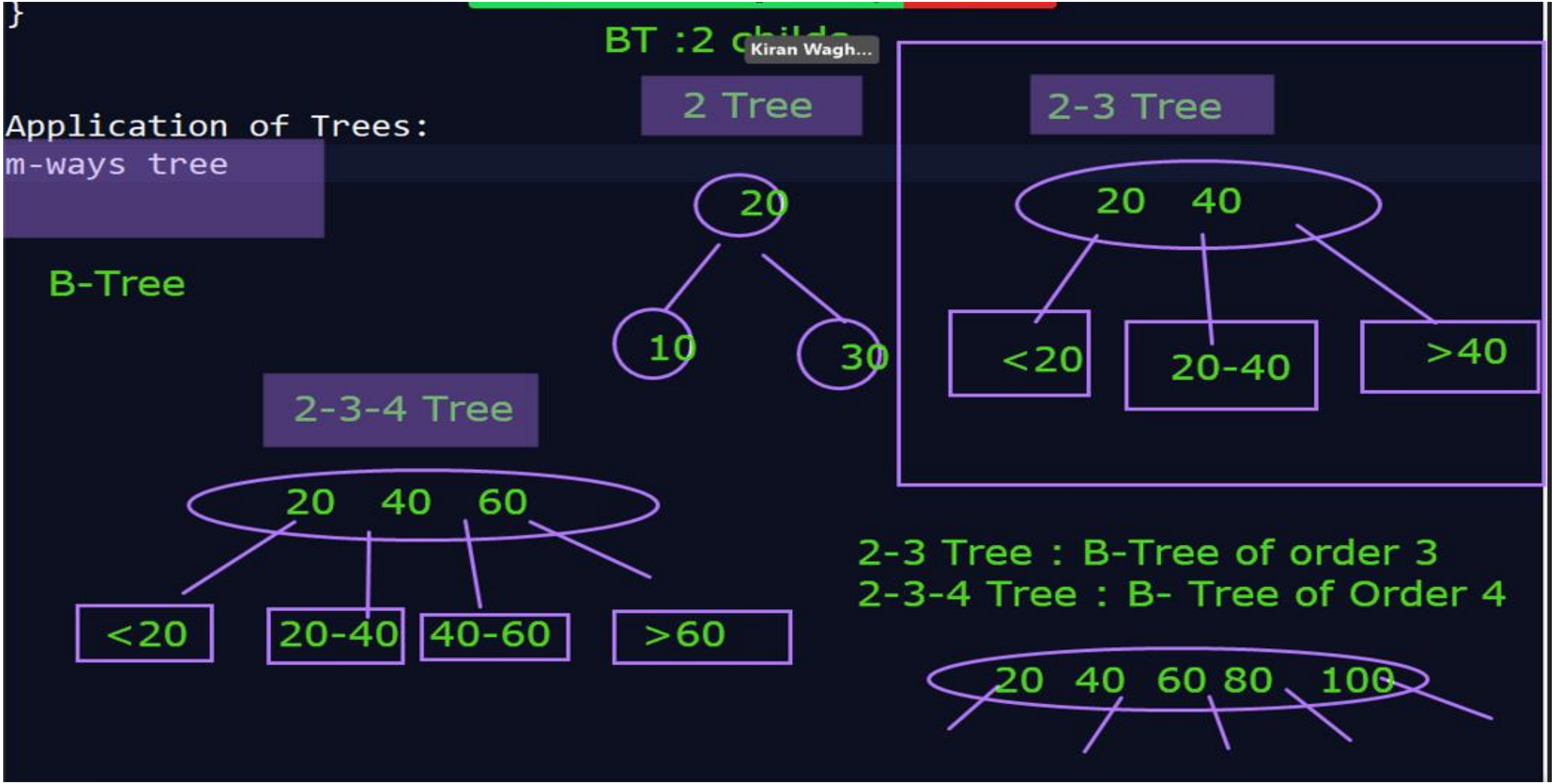
Red-black properties:

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).



Example of a red-black tree




}
BT :2
Kiran Wagh...
Application of Trees:
m-ways tree
B-Tree
2 Tree
20
10
30
2-3 Tree
20 40
<20
20-40
>40
2-3-4 Tree
20 40 60
<20
20-40
40-60
>60
2-3 Tree : B-Tree of order 3
2-3-4 Tree : B- Tree of Order 4
20 40 60 80 100

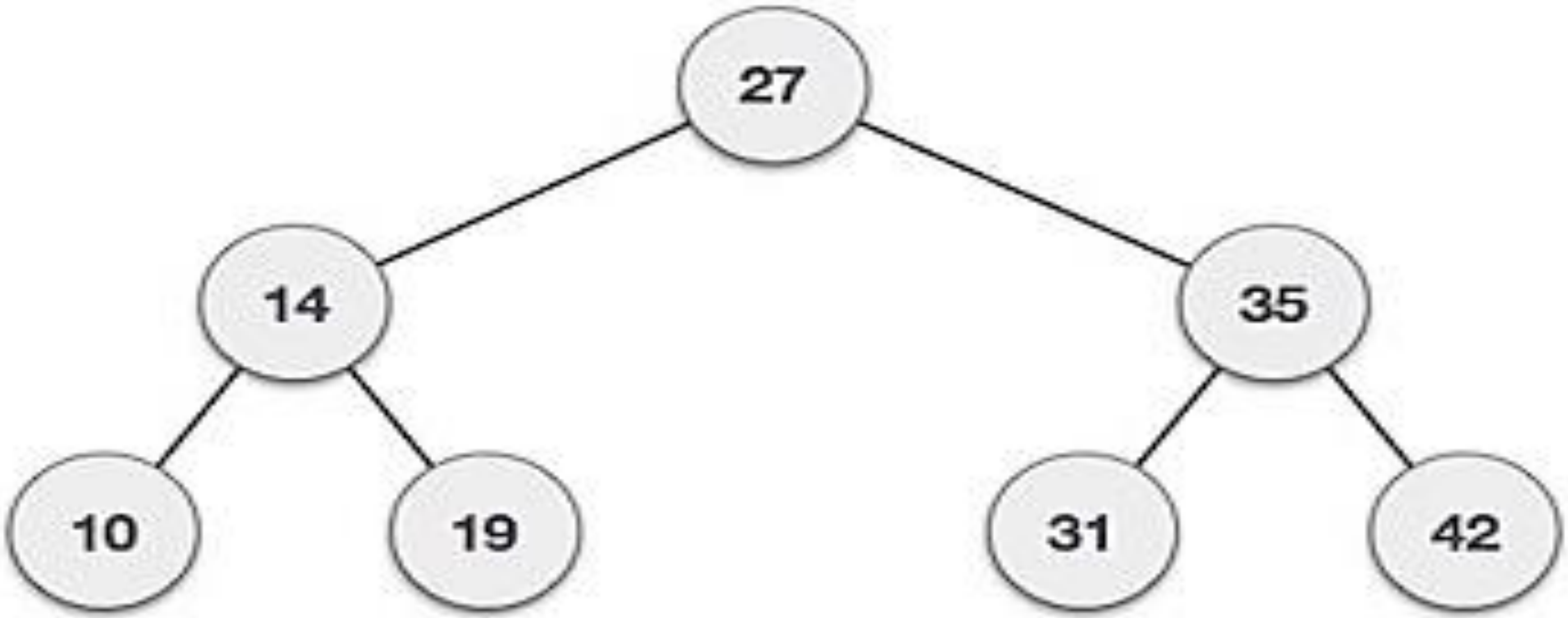
AVL Tree

AVL Tree (Adelson – Velskii – Landis)

- **Binary Tree:**
  - A binary search tree (BST) is a tree in which all nodes follows the below mentioned properties –
  - The left sub-tree of a node has key less than or equal to its parent node's key.
  - The right sub-tree of a node has key greater than or equal to its parent node's key.
- **Thus, a binary search tree (BST) divides all its sub-trees into two segments;**
- ***left* sub-tree and *right* sub-tree and can be defined as –**
- **left_subtree (keys) ≤ node (key) ≤ right_subtree (keys)**

Example:

Example



AVL Rotations

- To make itself balanced, an AVL tree may perform four kinds of rotations –

1. Left rotation
2. Right rotation
3. Left-Right rotation
4. Right-Left rotation

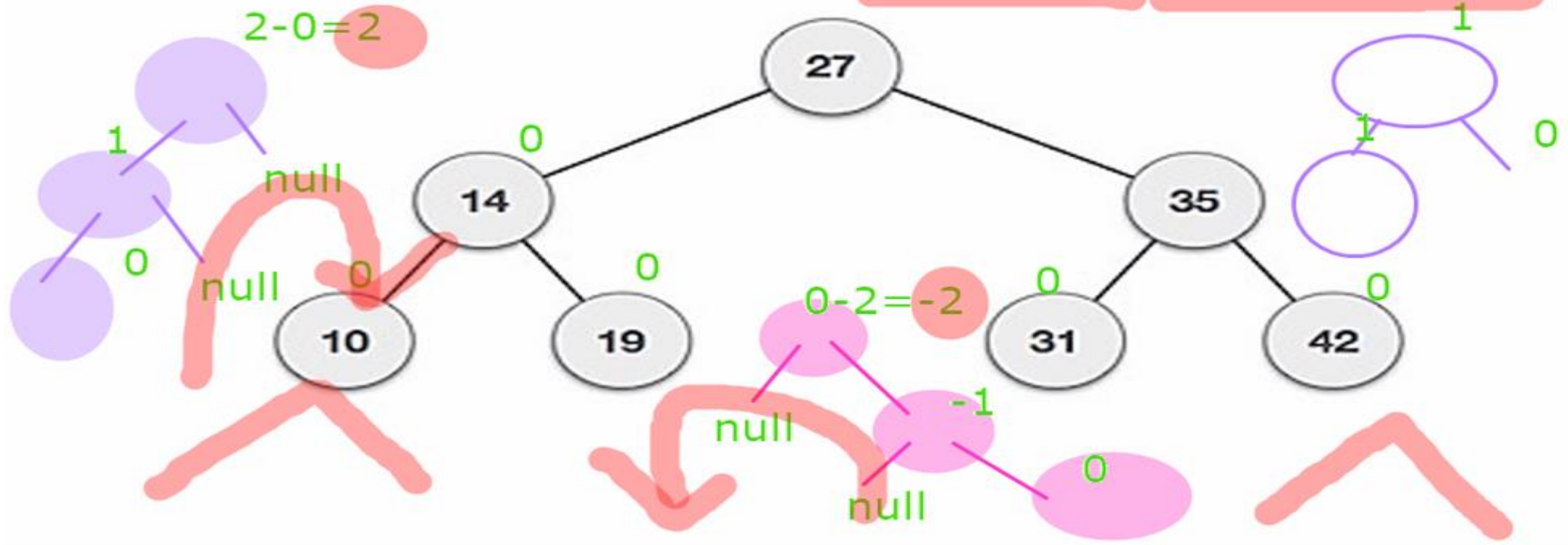
-
- First two rotations are single rotations and next two rotations are double rotations. Two have an unbalanced tree we at least need a tree of height 2. With this simple tree, let's understand them one by one.
-

Example

0,1<=2 Need for balance the tree

Balance Factor : height(LST) -height(RST)

Tree is not balance , then we apply rotations to balance that tree.



AVL Rotations

- To make itself balanced, an AVL tree may perform four kinds of rotations –

1. Left rotation
2. Right rotation
3. Left-Right rotation
4. Right-Left rotation

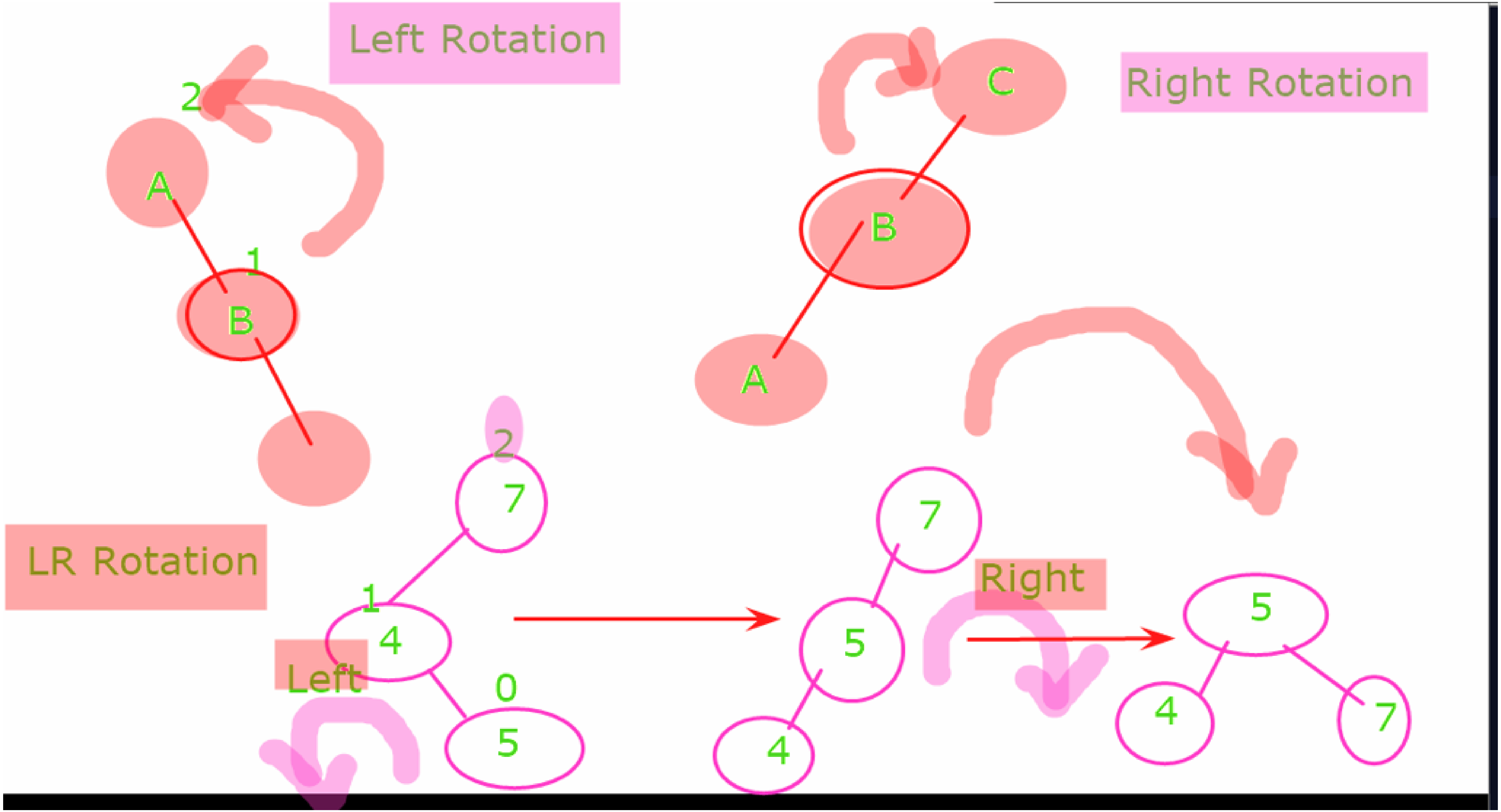
-

Left
Right => Single Rotation

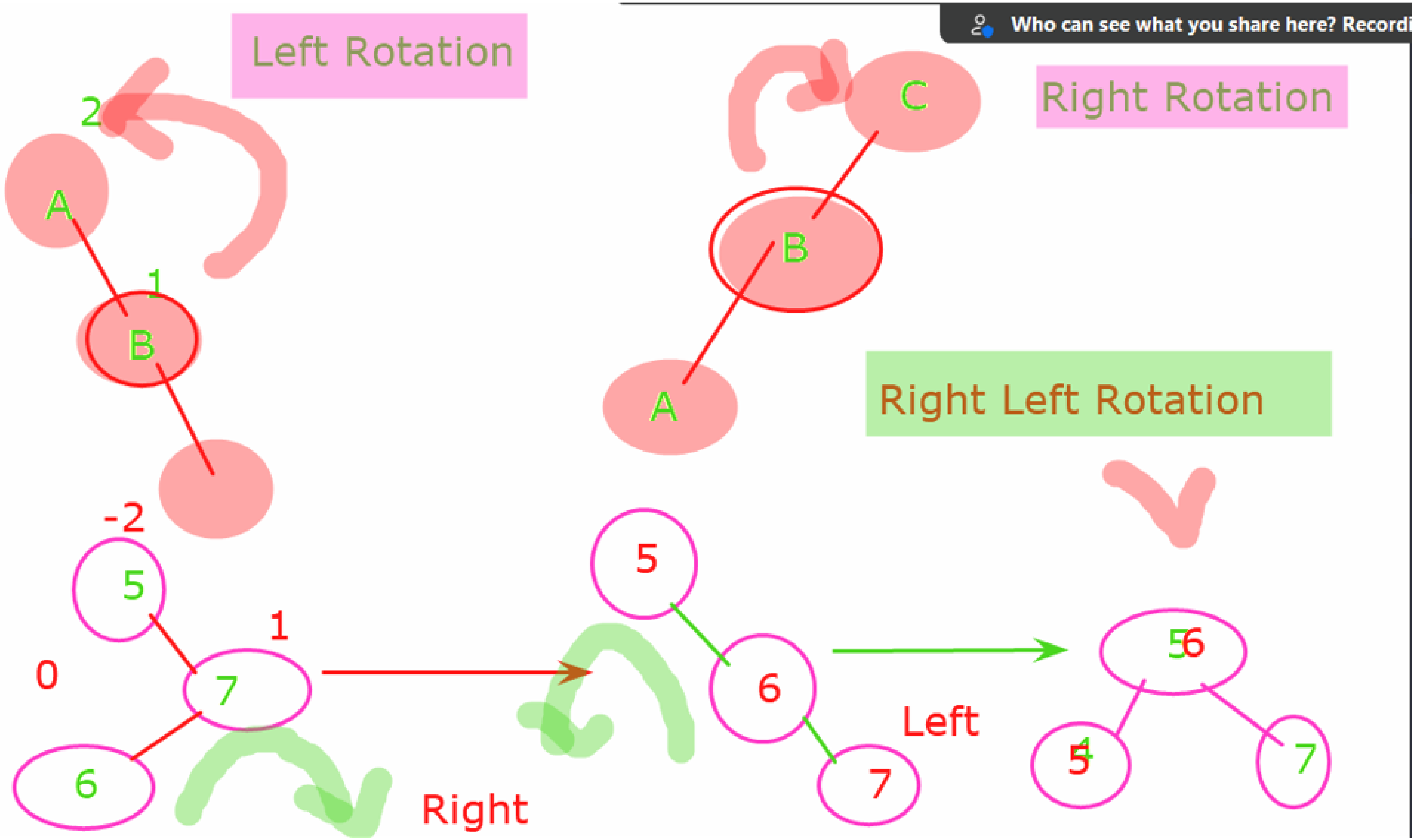
LR Rotation
RL Rotation => Double Rotation

- First two rotations are single rotations and next two rotations are double rotations. Two have an unbalanced tree we at least need a tree of height 2. With this simple tree, let's understand them one by one.

-



Left Rotation
2
A
1
B
C
Right Rotation
B
A
LR Rotation
2
7
1
4
Left
0
5
7
5
4
Right
5
4
7


Who can see what you share here? Recordi
Left Rotation
2
A
1
B
Right Rotation
C
B
A
Right Left Rotation
-2
5
1
0
7
6
Right
5
6
7
Left
56
54
7

Heap

Module I

Kiran Waghmare

Definition in Data Structure

- **Heap:**
  - A special form of complete binary tree that key value of each node is no smaller (larger) than the key value of its children (if any).
- **Max-Heap:**
  - root node has the largest key. A max tree is a tree in which the key value in each node is no smaller than the key values in its children. A max heap is a complete binary tree that is also a max tree.
- **Min-Heap:**
  - root node has the smallest key. A min tree is a tree in which the key value in each node is no larger than the key values in its children. A min heap is a complete binary tree that is also a min tree.
- **Complete Binary Tree:**
  - A complete binary tree is a binary tree in which every level, *except possibly the last*, **is completely filled, and all nodes are as far left as possible**

Heap

- **Definition in Data Structure**
  - **Heap:** A special form of complete binary tree that key value of each node is no smaller (larger) than the key value of its children (if any).

- **Max-Heap: root node has the largest key.**
  - A *max tree* is a tree in which the key value in each node is no smaller than the key values in its children. A *max heap* is a complete binary tree that is also a max tree.

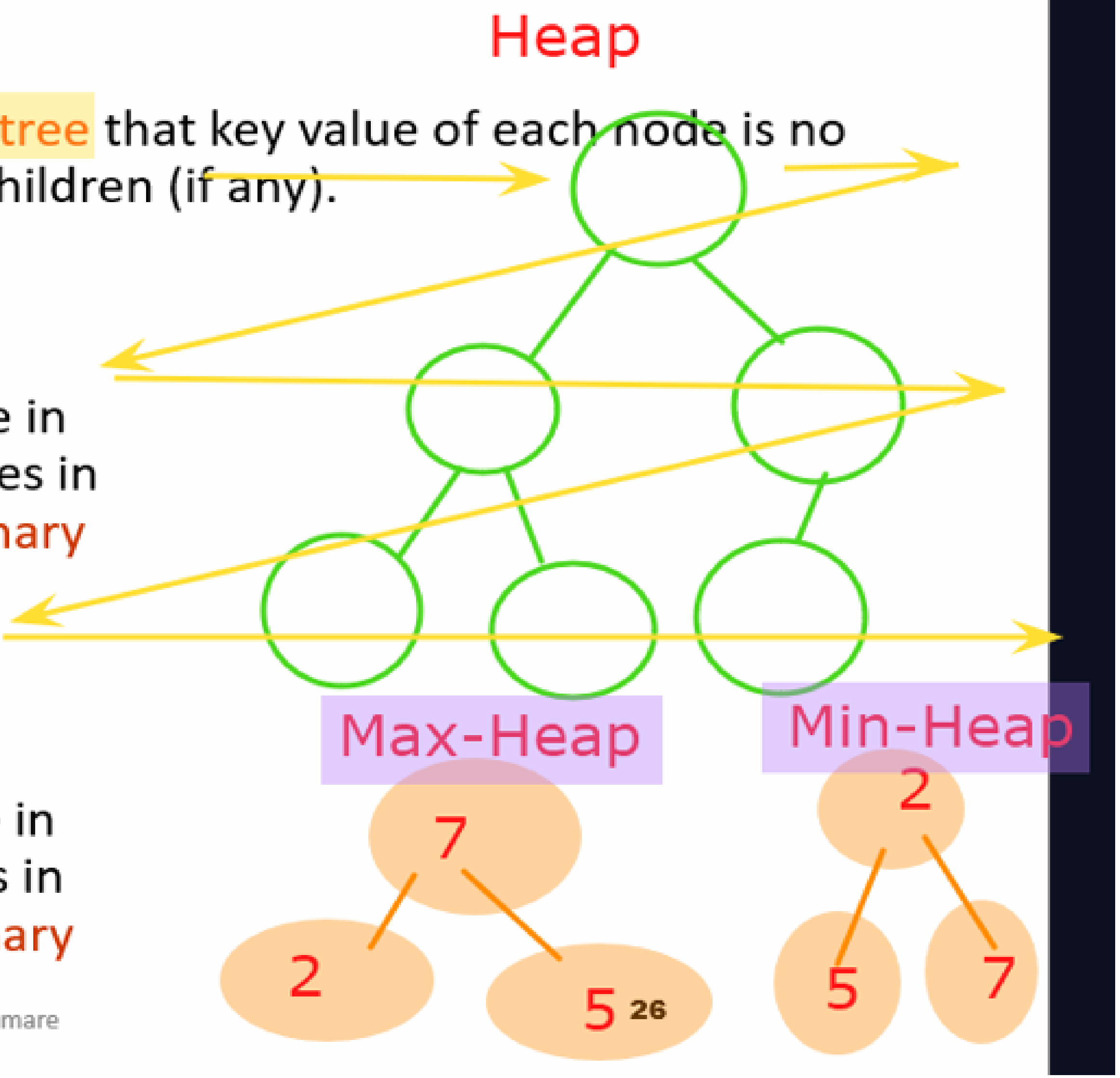
- **Min-Heap: root node has the smallest key.**
  - A *min tree* is a tree in which the key value in each node is no larger than the key values in its children. A *min heap* is a complete binary tree that is also a min tree.

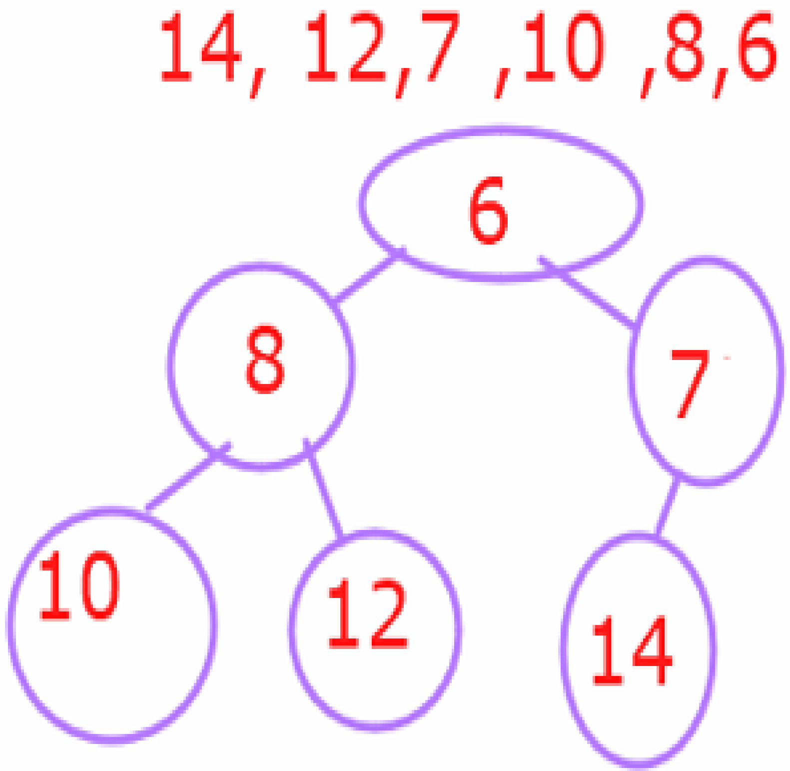
Heap

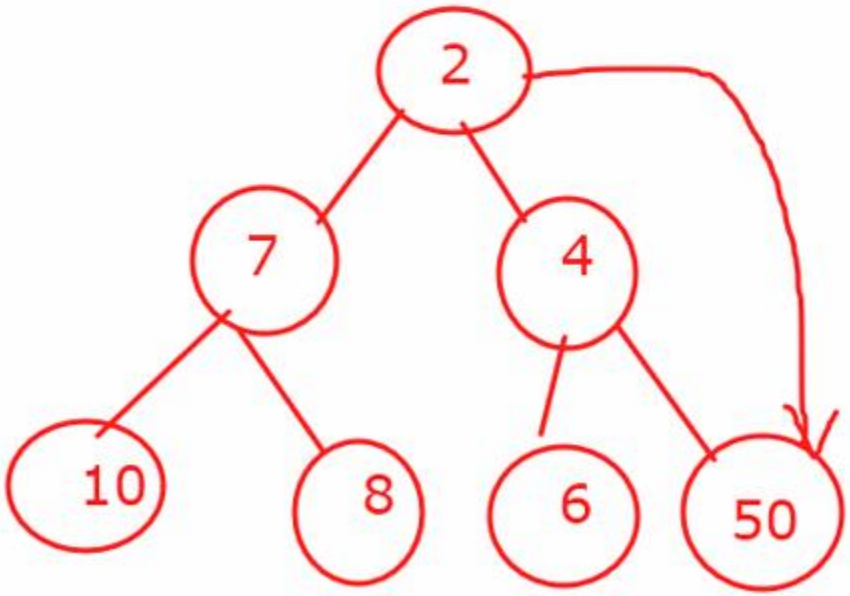
- **Definition in Data Structure**
  - **Heap:** A special form of complete binary tree that key value of each node is no smaller (larger) than the key value of its children (if any).

- **Max-Heap: root node has the largest key.**
  - A *max tree* is a tree in which the key value in each node is no smaller than the key values in its children. A *max heap* is a complete binary tree that is also a max tree.

- **Min-Heap: root node has the smallest key.**
  - A *min tree* is a tree in which the key value in each node is no larger than the key values in its children. A *min heap* is a complete binary tree that is also a min tree.




14, 12,7 ,10 ,8,6
6
8
7
10
12
14

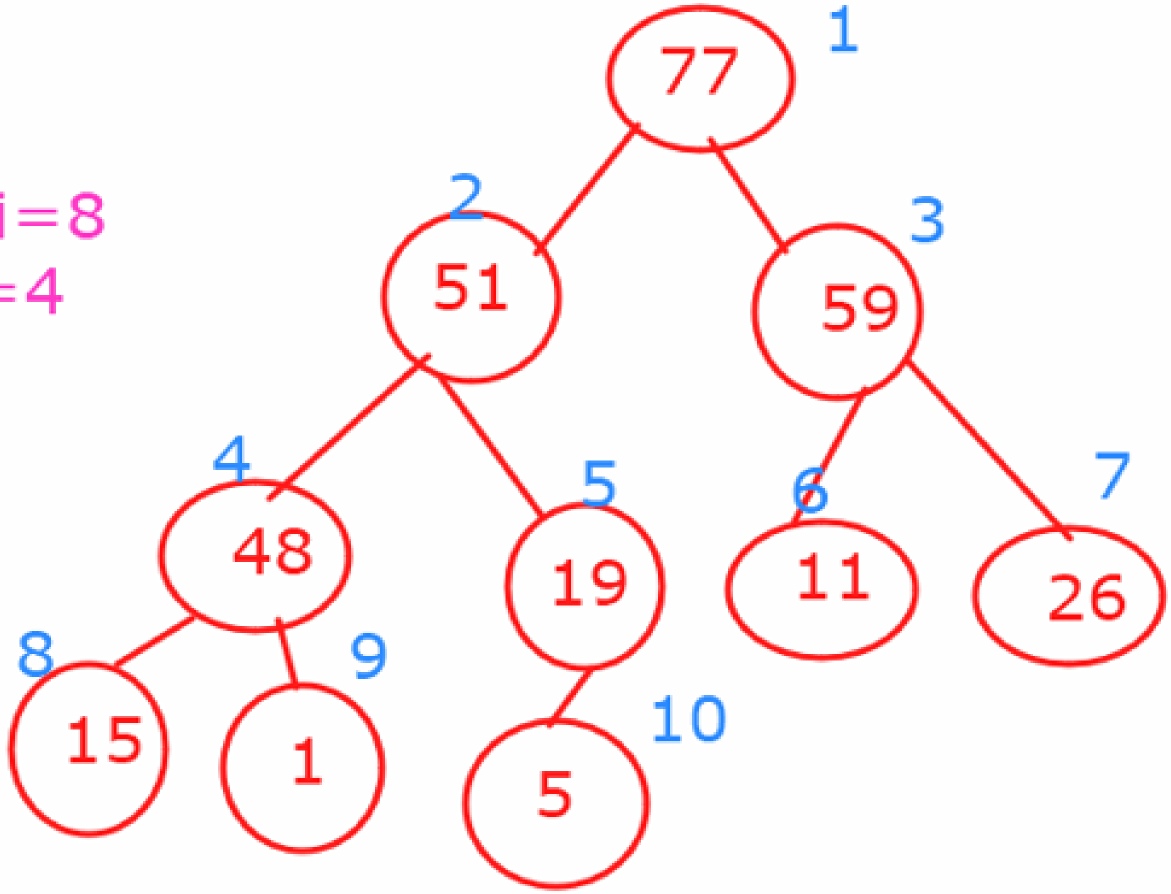

2
7
4
10
8
6
50

Heap Implementation:

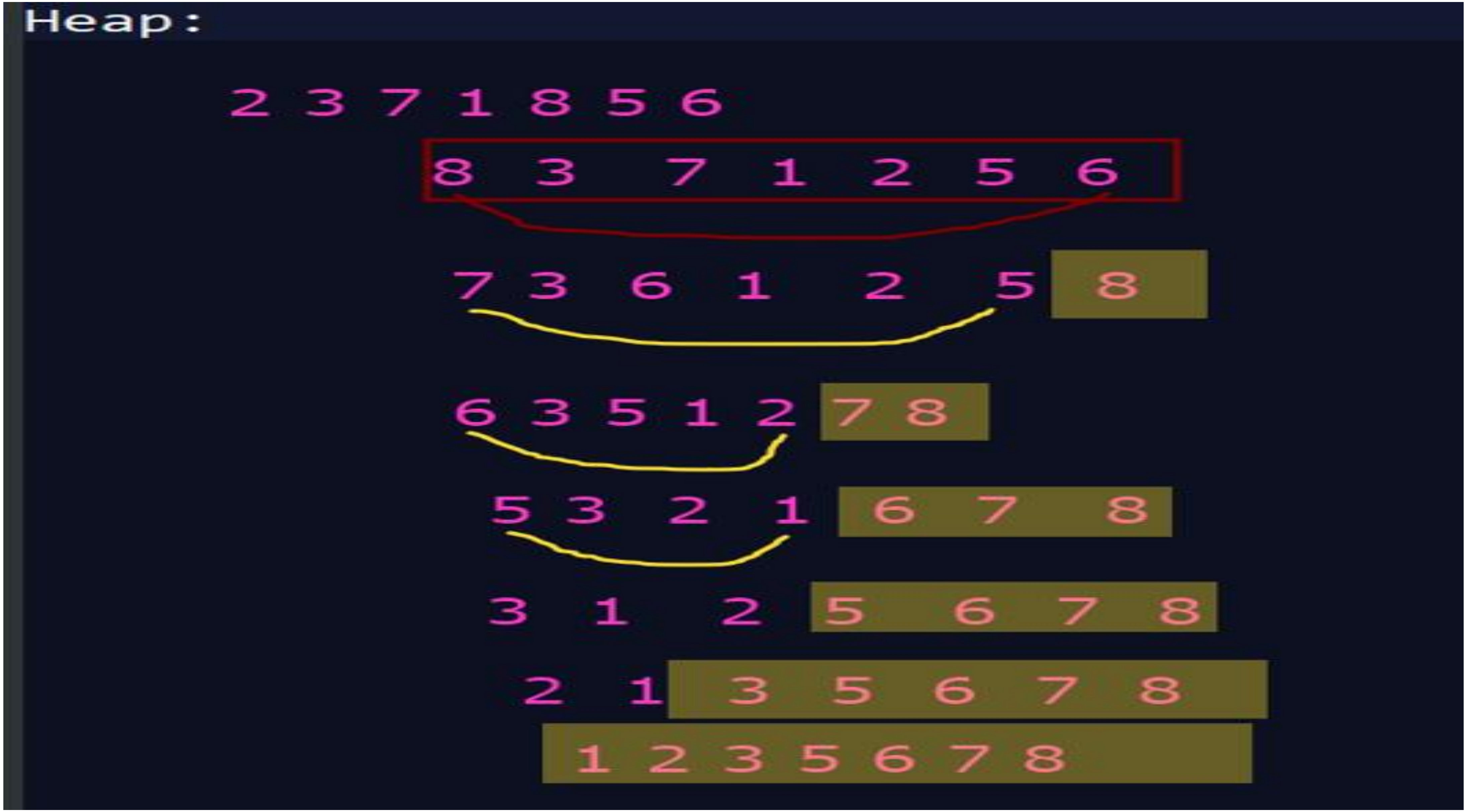
Root LC RC

LC: 2i
RC: 2i+1
Parent: i/2

2i=8
i=4



Value:	77	51	59	48	19	11	26	15	1	5
Index:	1	2	3	4	5	6	7	8	9	10


Heap:
2 3 7 1 8 5 6
8 3 7 1 2 5 6
7 3 6 1 2 5 8
6 3 5 1 2 7 8
5 3 2 1 6 7 8
3 1 2 5 6 7 8
2 1 3 5 6 7 8
1 2 3 5 6 7 8

Thanks