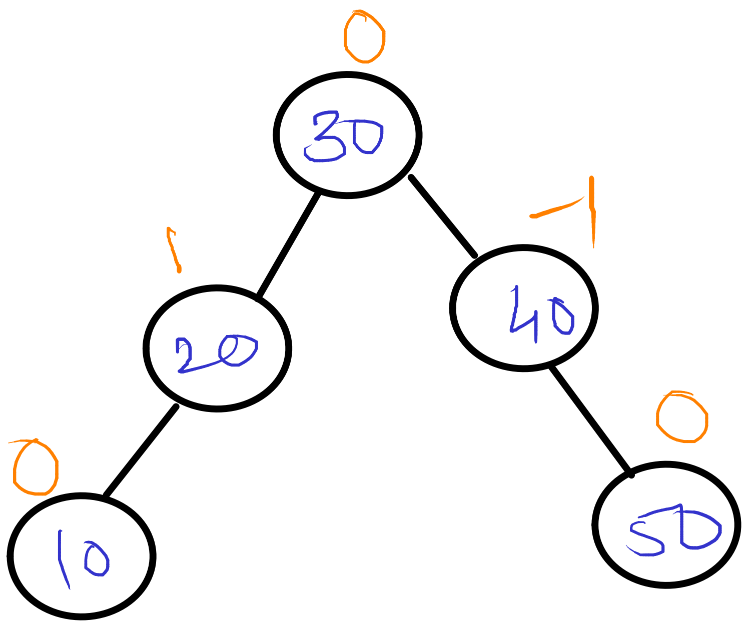
Skewed BST

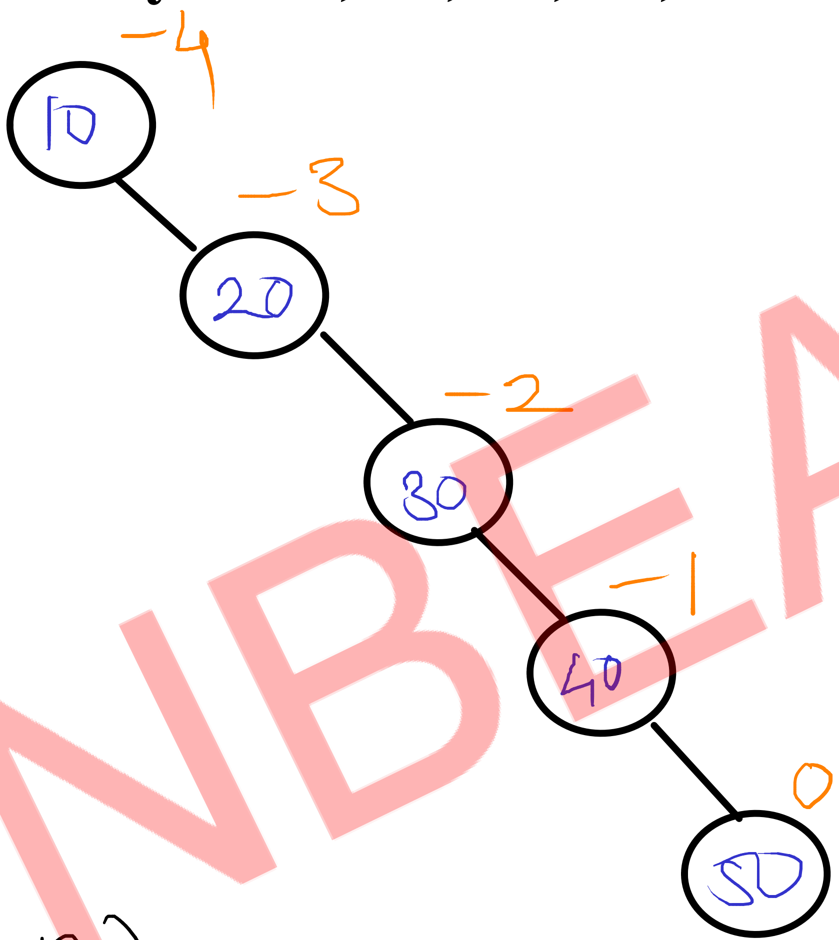
Keys : 30, 40, 20, 50, 10



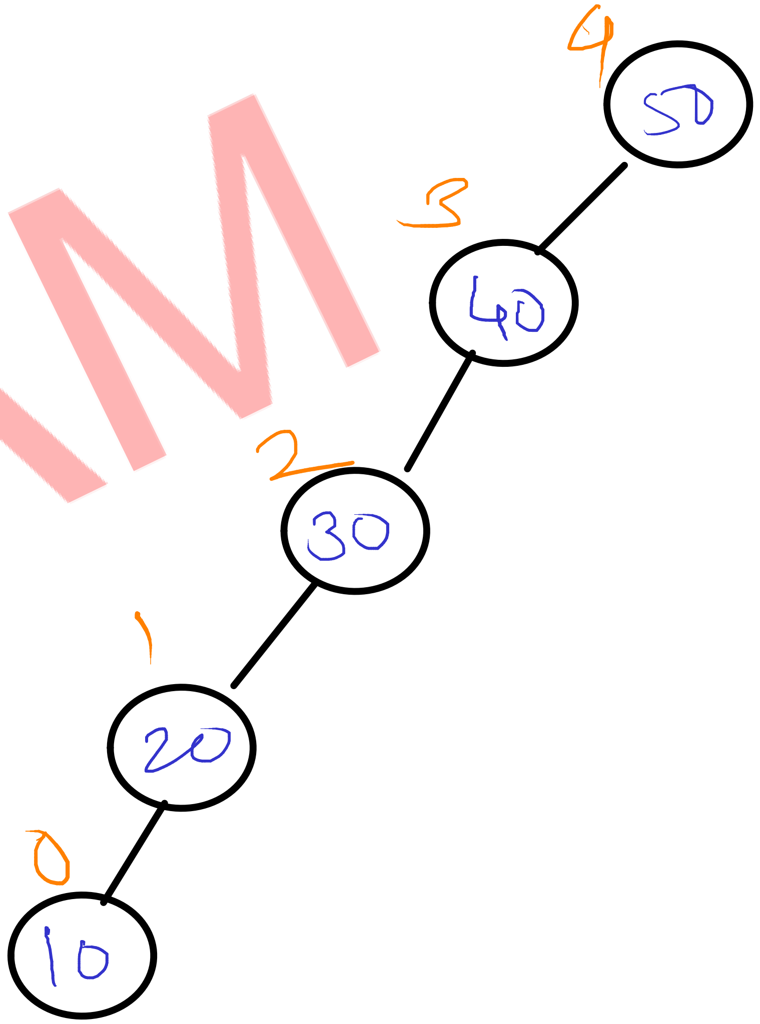
height = log n

T(n) = O(log n)

Keys : 10, 20, 30, 40, 50



Key : 50, 40, 30, 20, 10



height = n

T(n) = O(n)

- if tree is growing in only one direction, it is known as skewed BST

- if tree us growing in only left direction, it is known as left skewed BST

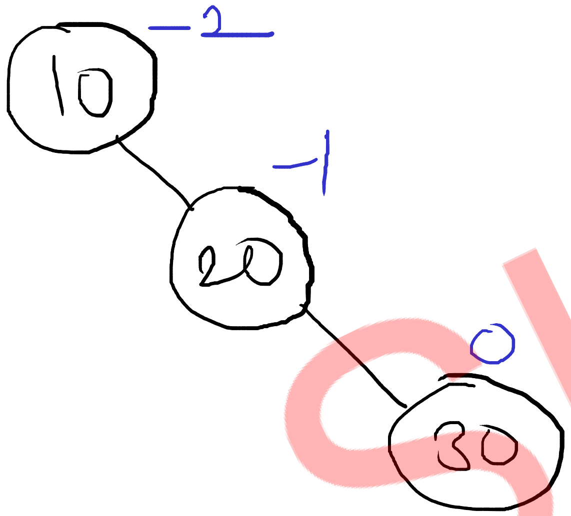
- if tree us growing in only right direction, it is known as right skewed BST

Balanced BST

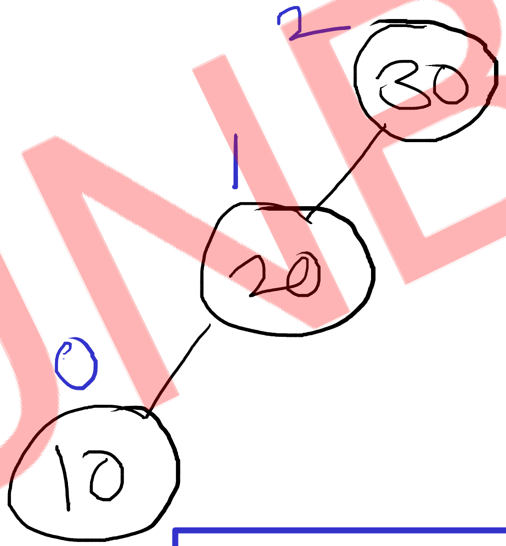
Balance Factor	=	height(left sub tree)	-	height(right sub tree)

- tres is balanced if balance factor of all the nodes is either -1, 0 or +1
- balance factor = {-1, 0, +1}

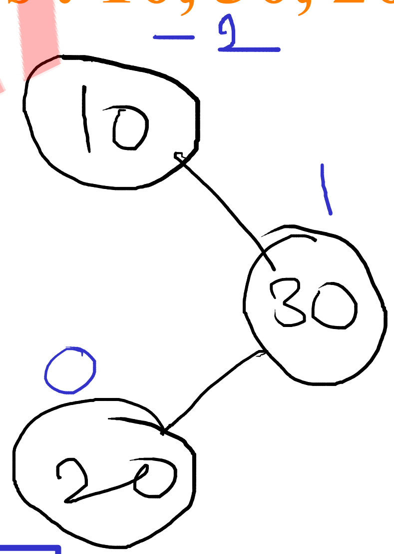
Keys : 10, 20 ,30



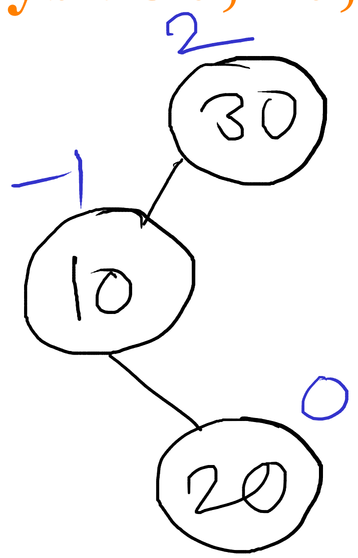
Keys : 30, 20, 10



Keys : 10, 30, 20

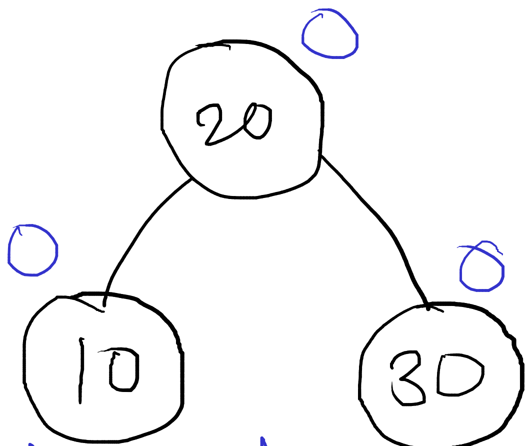


Keys : 30, 10, 20



Keys : 20, 10, 30

Keys : 20, 30, 10

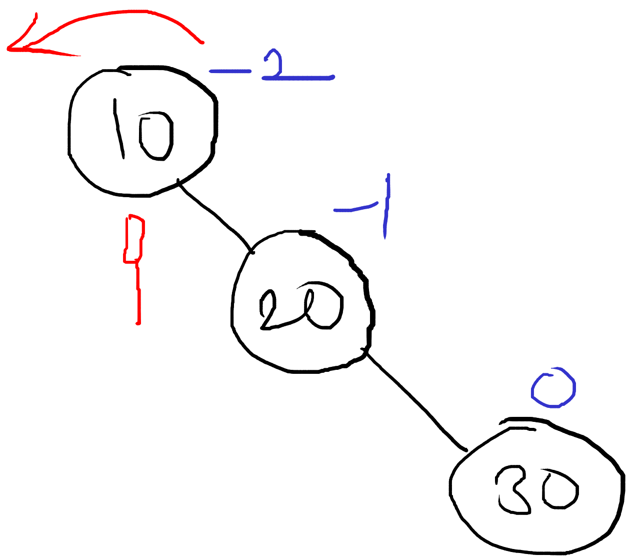


Balanced BST

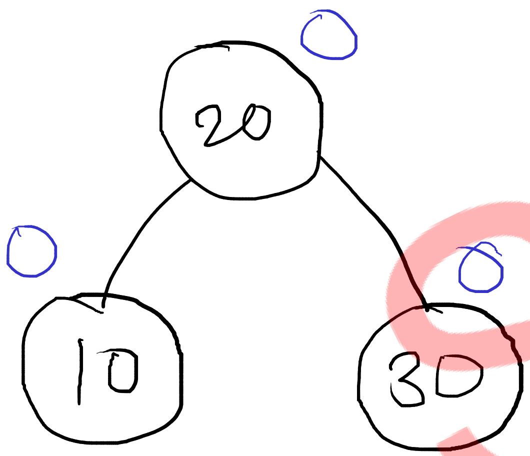
Rotations

RR Imbalance

Keys : 10, 20 ,30

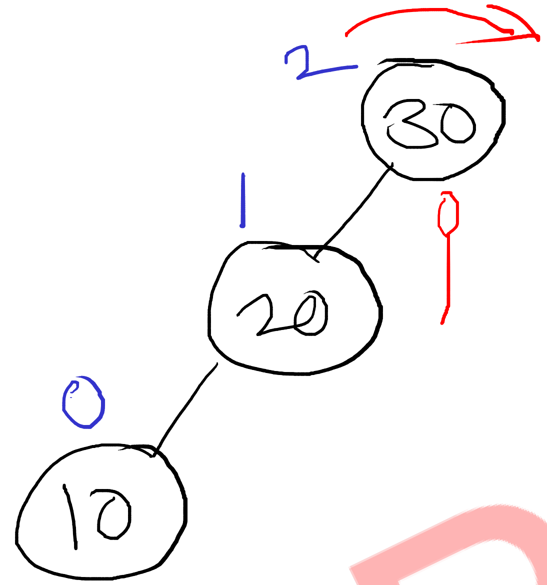


Left Rotation

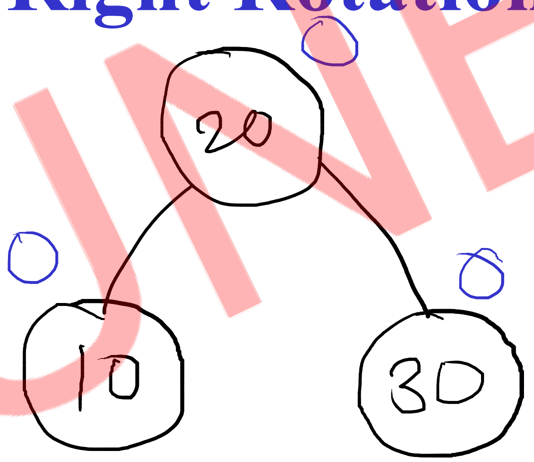


LL Imbalance

Keys : 30, 20, 10



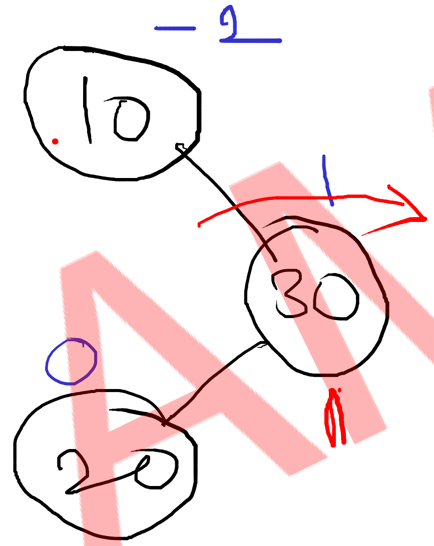
Right Rotation



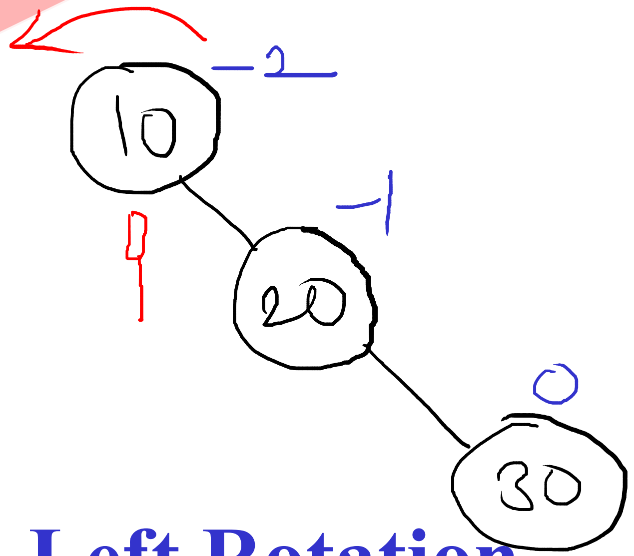
Single Rotation

RL Imbalance

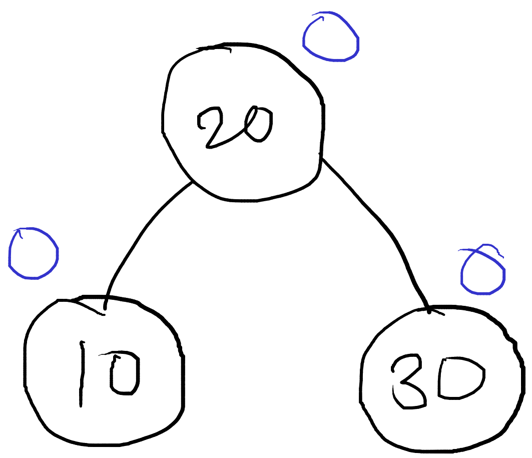
Keys : 10, 30, 20



Right Rotation

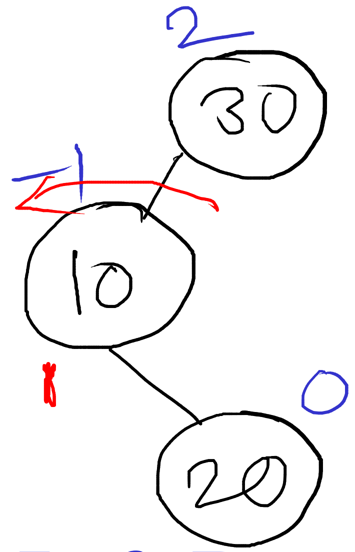


Left Rotation

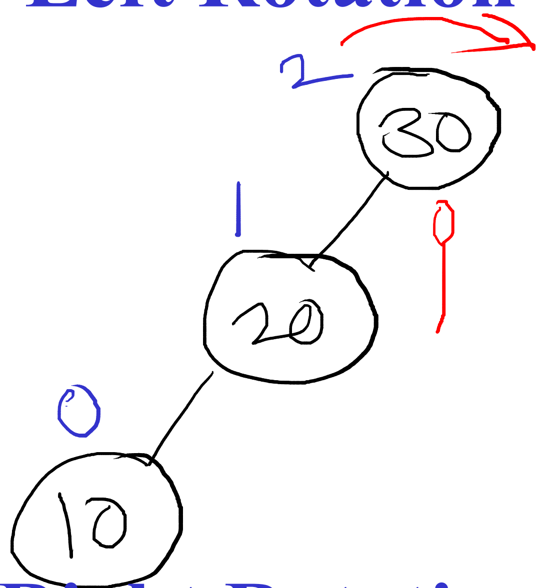


LR Imbalance

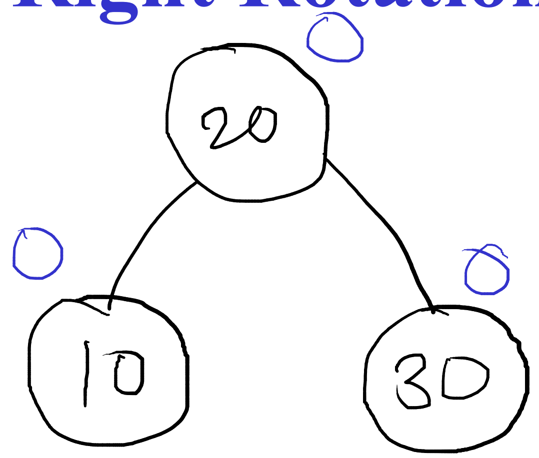
Keys : 30, 10, 20



Left Rotation



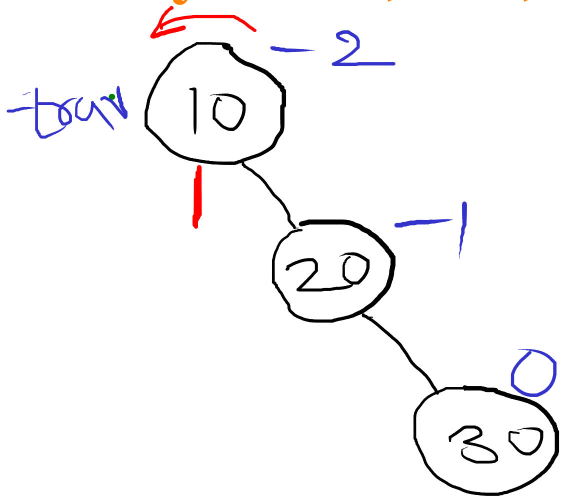
Right Rotation



Double Rotation

RR Imbalance

Keys : 10, 20 ,30



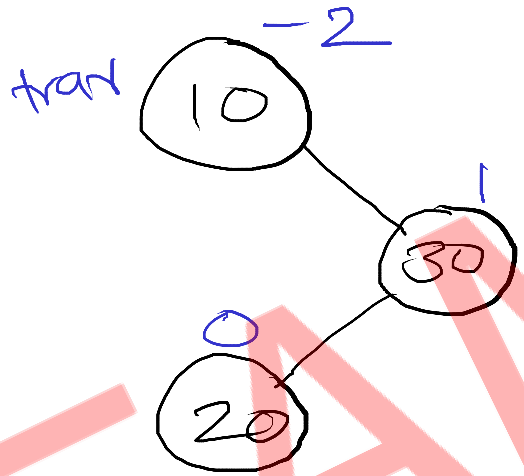
bf < -1

trav.right.data < value

(20 < 30)

RL Imbalance

Keys : 10, 30, 20



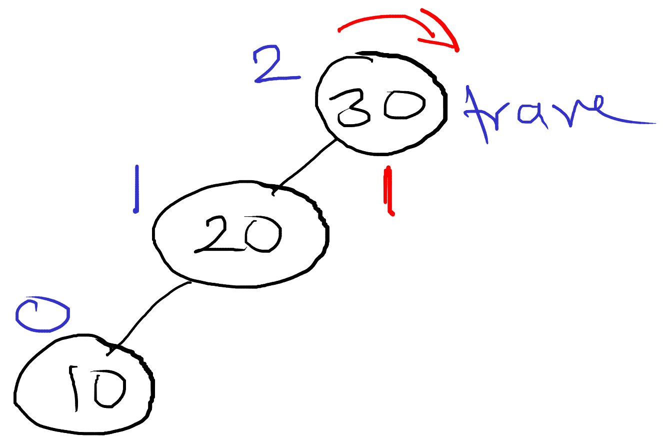
bf < -1

trav.right.data > value

(30 > 20)

LL Imbalance

Keys : 30, 20, 10



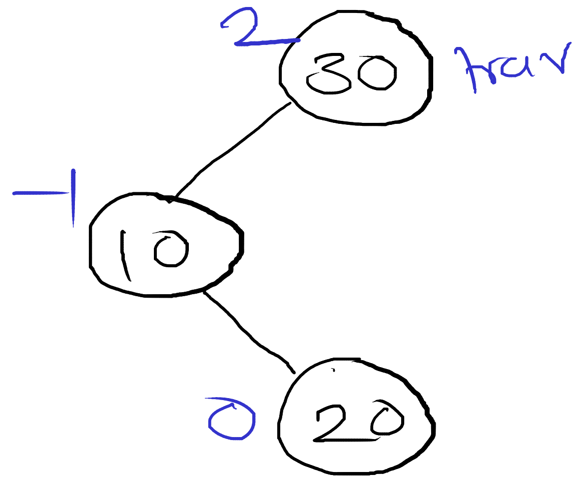
bf > 1

trav.left.data > value

(20 > 10)

LR Imbalance

Keys : 30, 10, 20

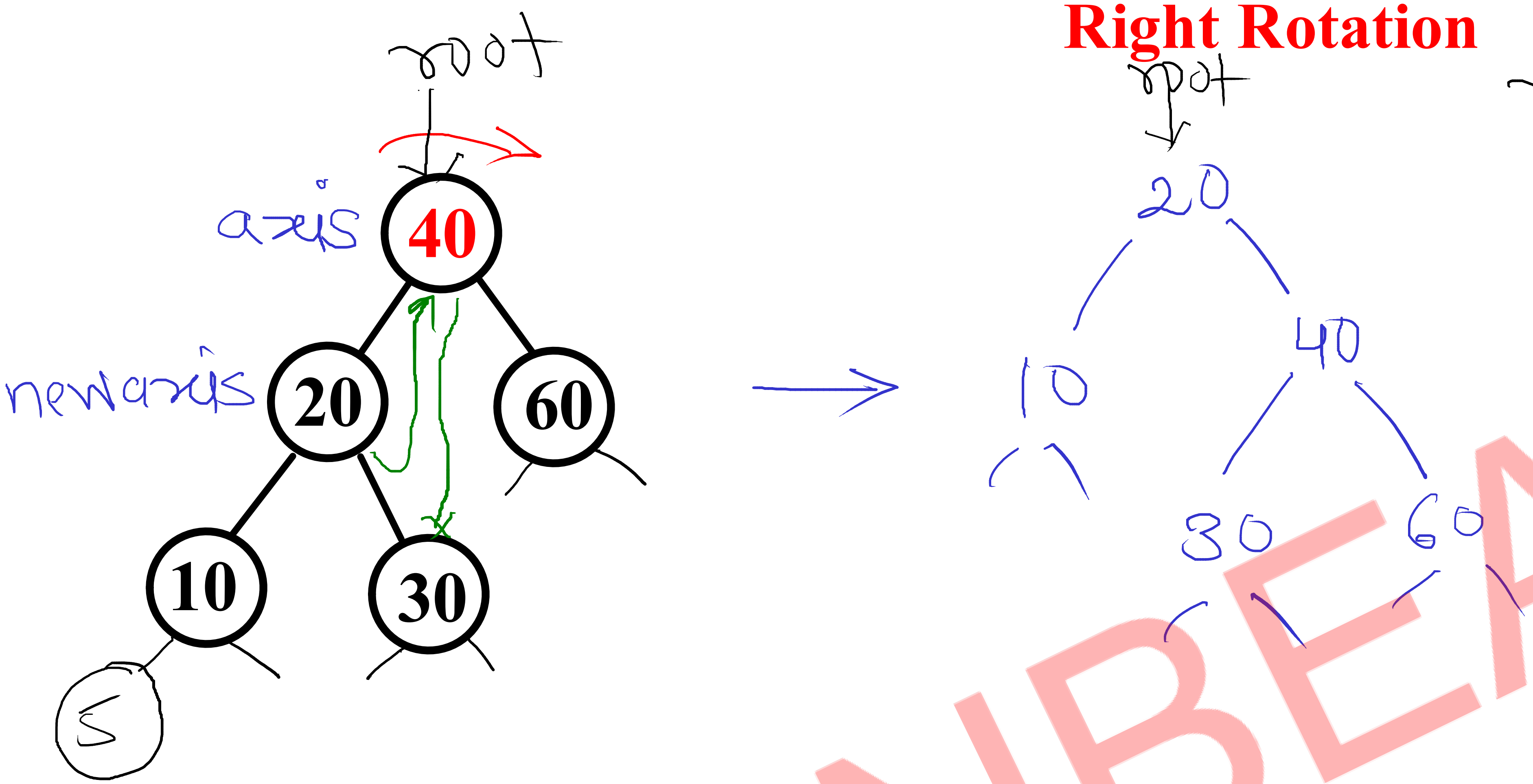


bf > 1

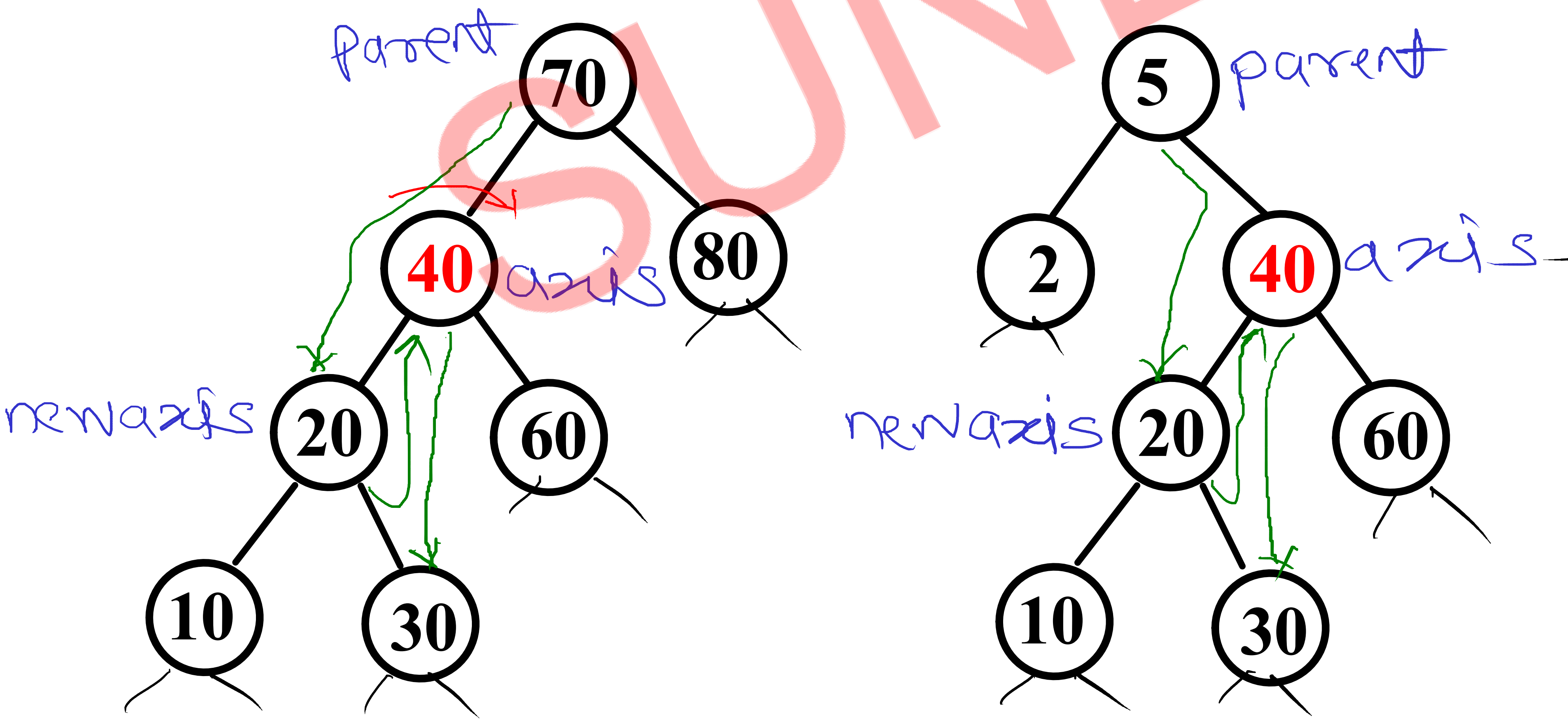
trav.left.data < value

(10 < 20)

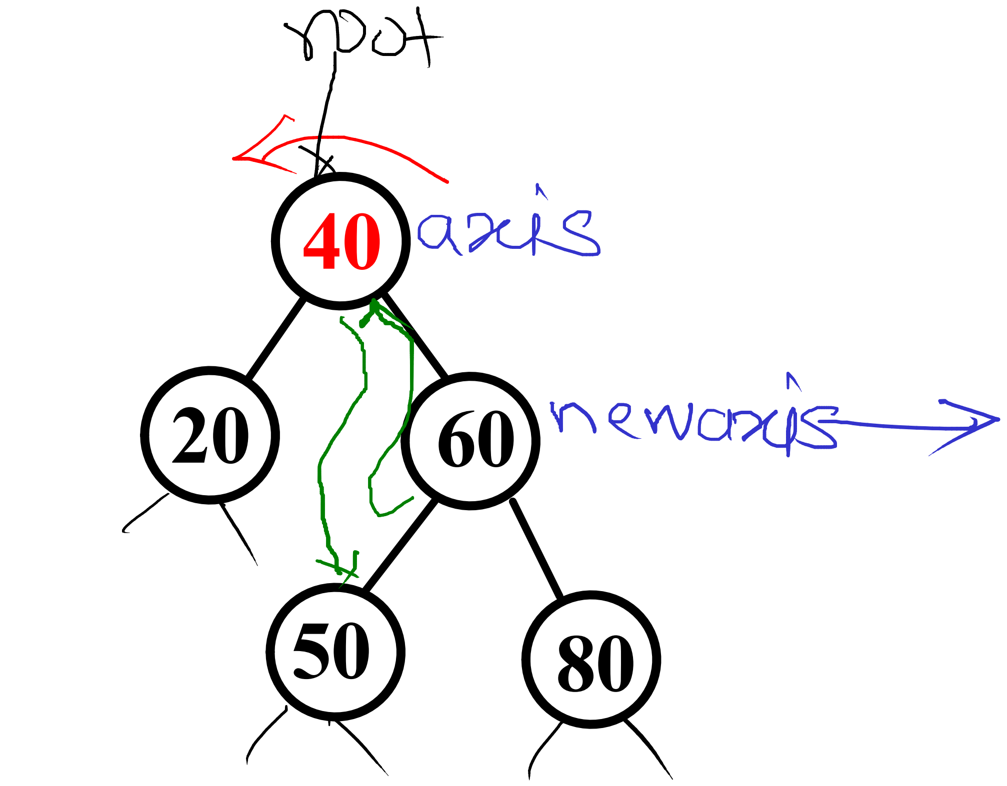
Right Rotation

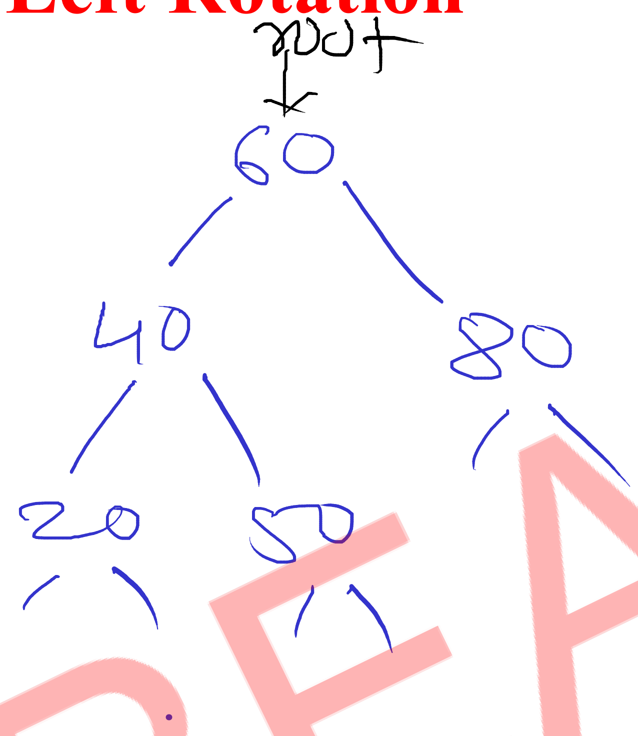



```
right_rotation(axis, parent){
    newaxis = axis.left;
    axis.left = newaxis.right;
    newaxis.right = axis;
    if(axis == root)
        root = newaxis
    else if(axis == parent.left)
        parent.left = newaxis;
    else if(axis == parent.right)
        parent.right = newaxis;
}
```

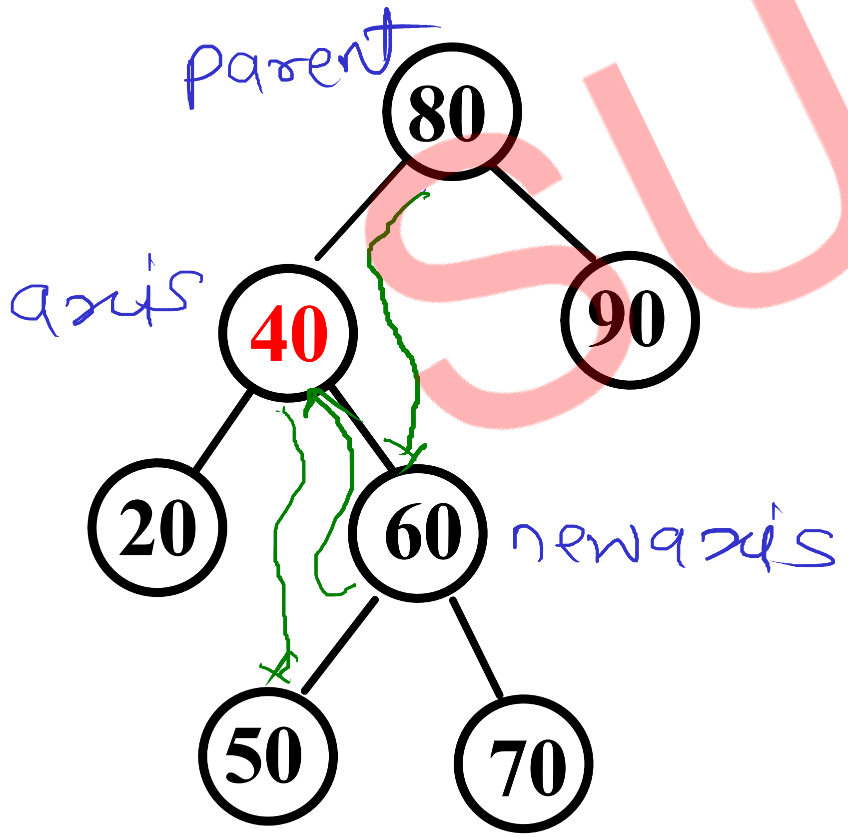


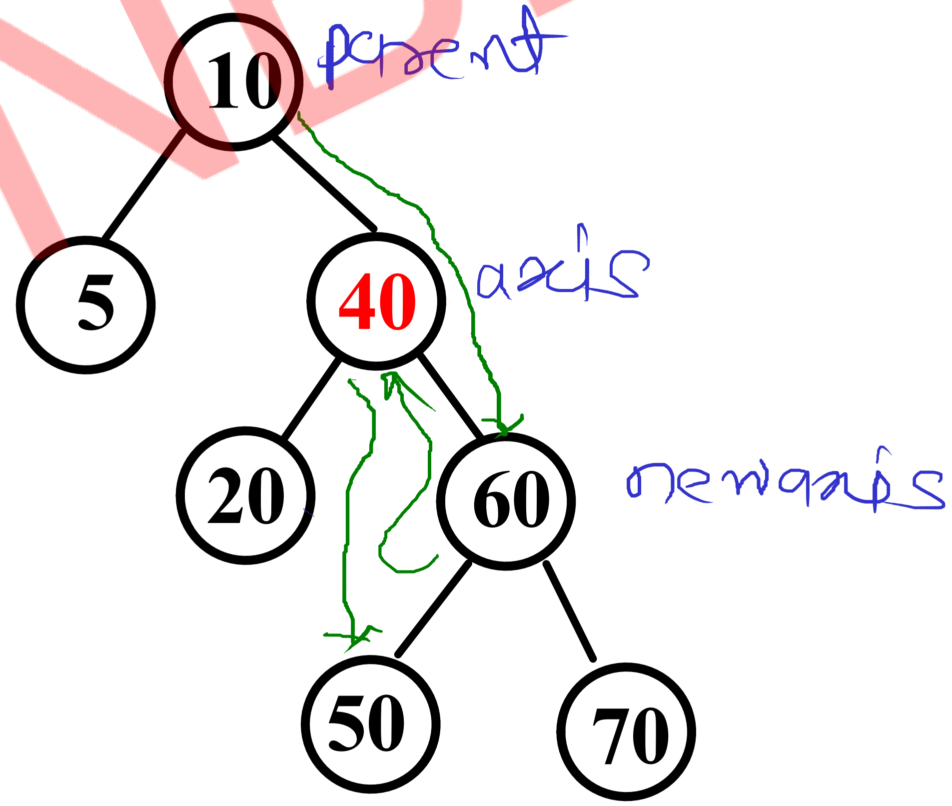
Left Rotation






```
left_rotation(axis, parent){
    newaxis = axis.right
    axis.right = newaxis.left
    newaxis.left = axis
    if(axis == root)
        root = newaxis;
    elseif(axis == parent.left)
        parent.left = newaxis;
    elseif(axis == parent.right)
        parent.right = newaxis;
}
```

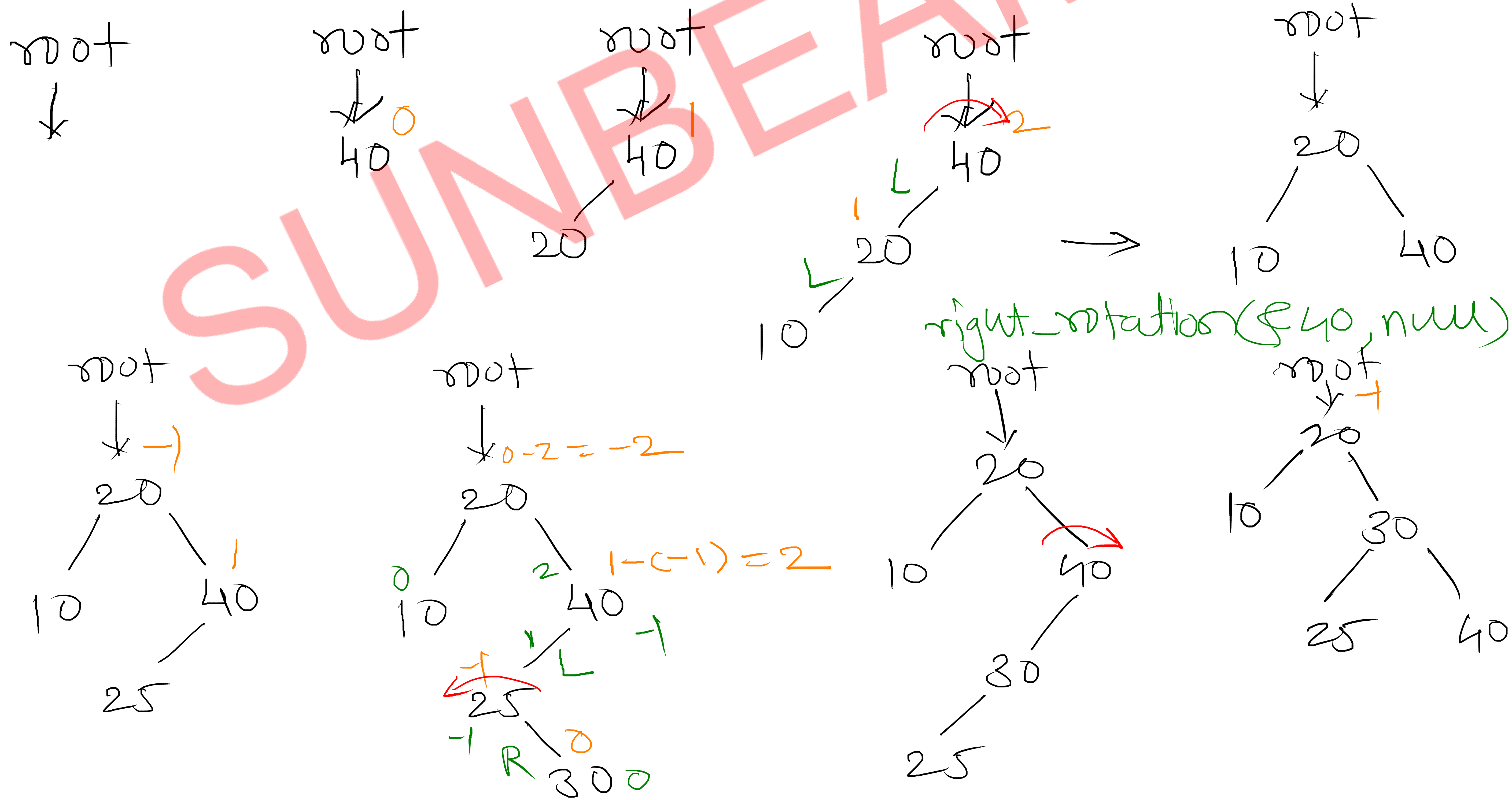


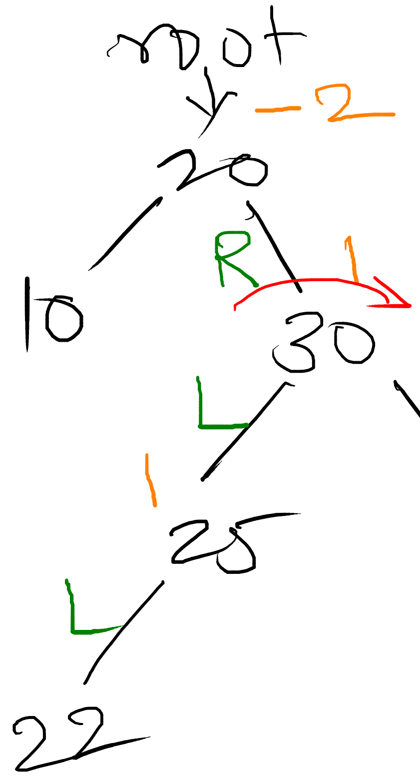


AVL Tree

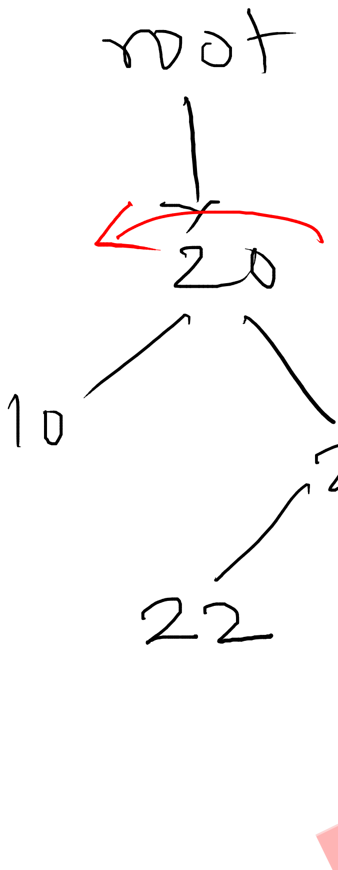
- Self balancing binary Search Tree
- on every insertion and deletion of node, tree is balanced
- All operation on AVL tree are perfromed in O(log n) time
- Balance factor of all nodes is either -1, 0 or +1

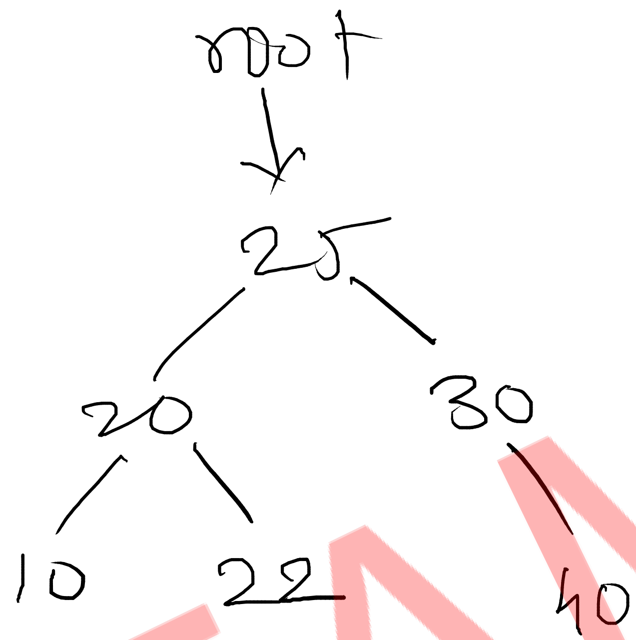
Keys : 40, 20, 10, 25, 30, 22, 50

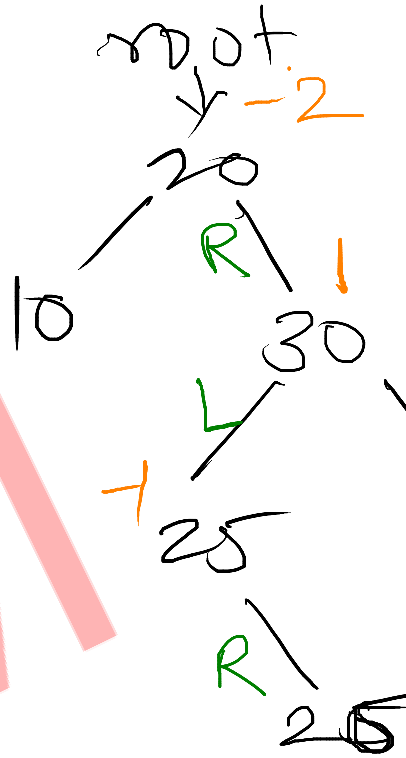



root
-2
20
10
R
1
30
L
1
25
40
L
22

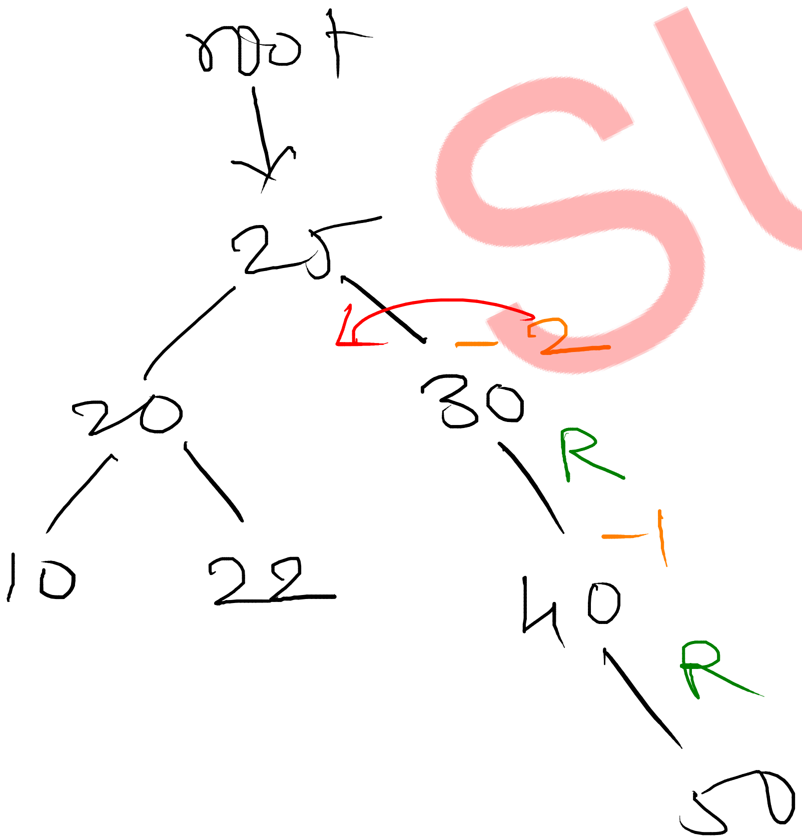
RLL

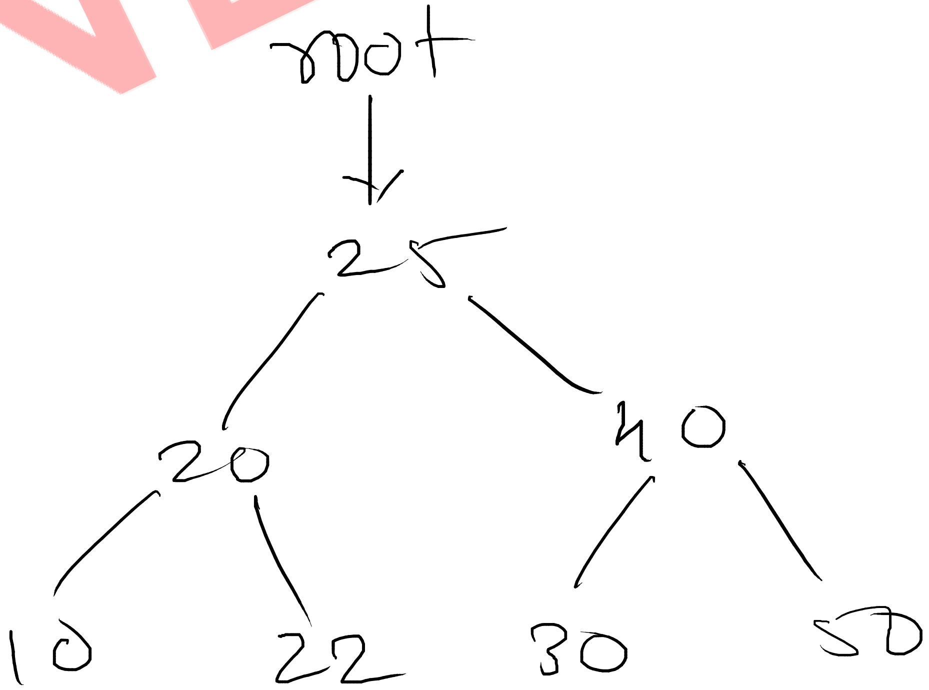

root
20
10
25
22
30
40


root
25
20
30
10
22
40


root
-2
20
10
R
1
30
L
-1
25
40
R
26

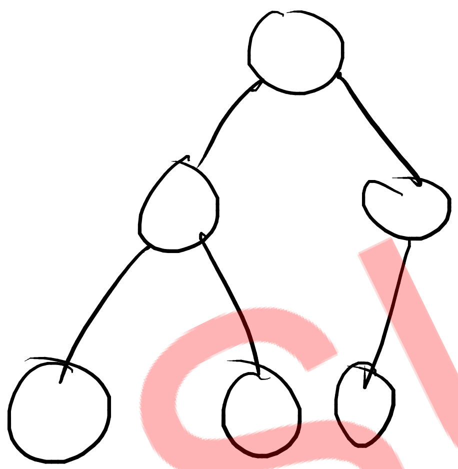
RLR

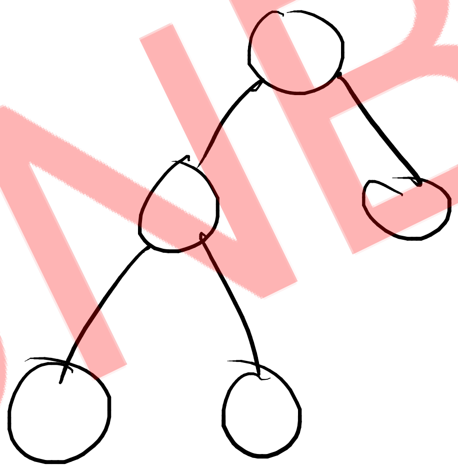

root
25
-2
20
30
R
10
22
-1
40
R
50

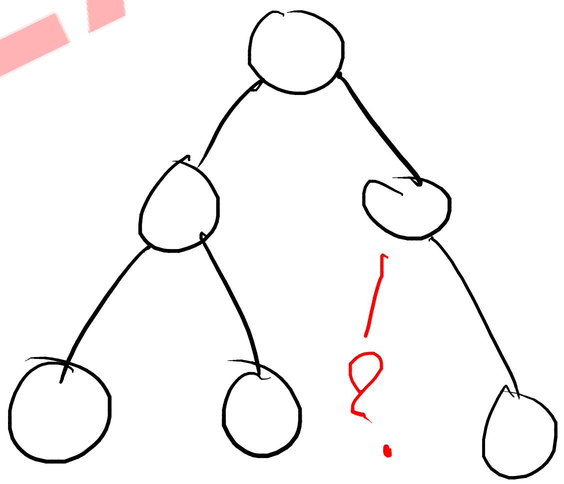

root
25
20
40
10
22
30
50

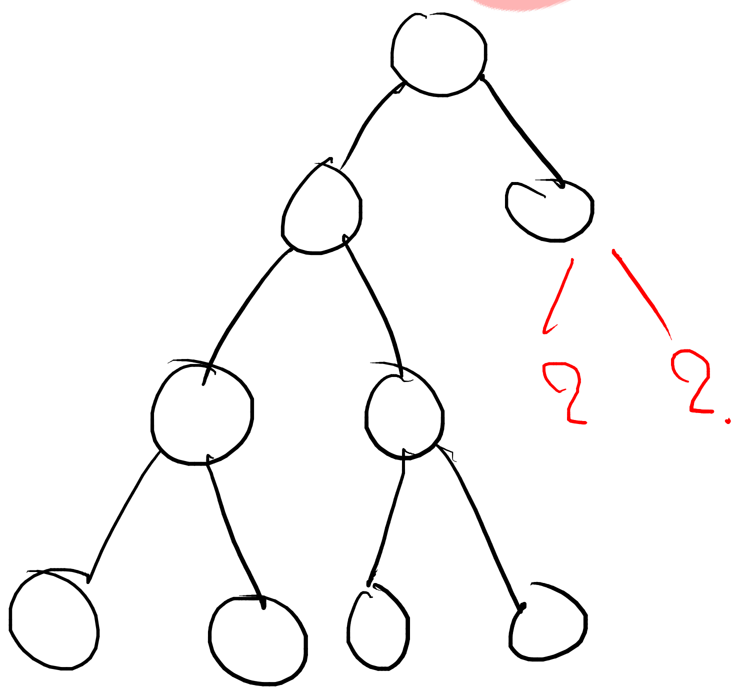
Almost Complete Binary Tree
(ACBT)

- this tree is filled level by level (from left to right)
- this tree should satisfy two condition
    1. all leaf nodes must be at level h or h-1
    2. nodes of last level should be left aligned as much as possible



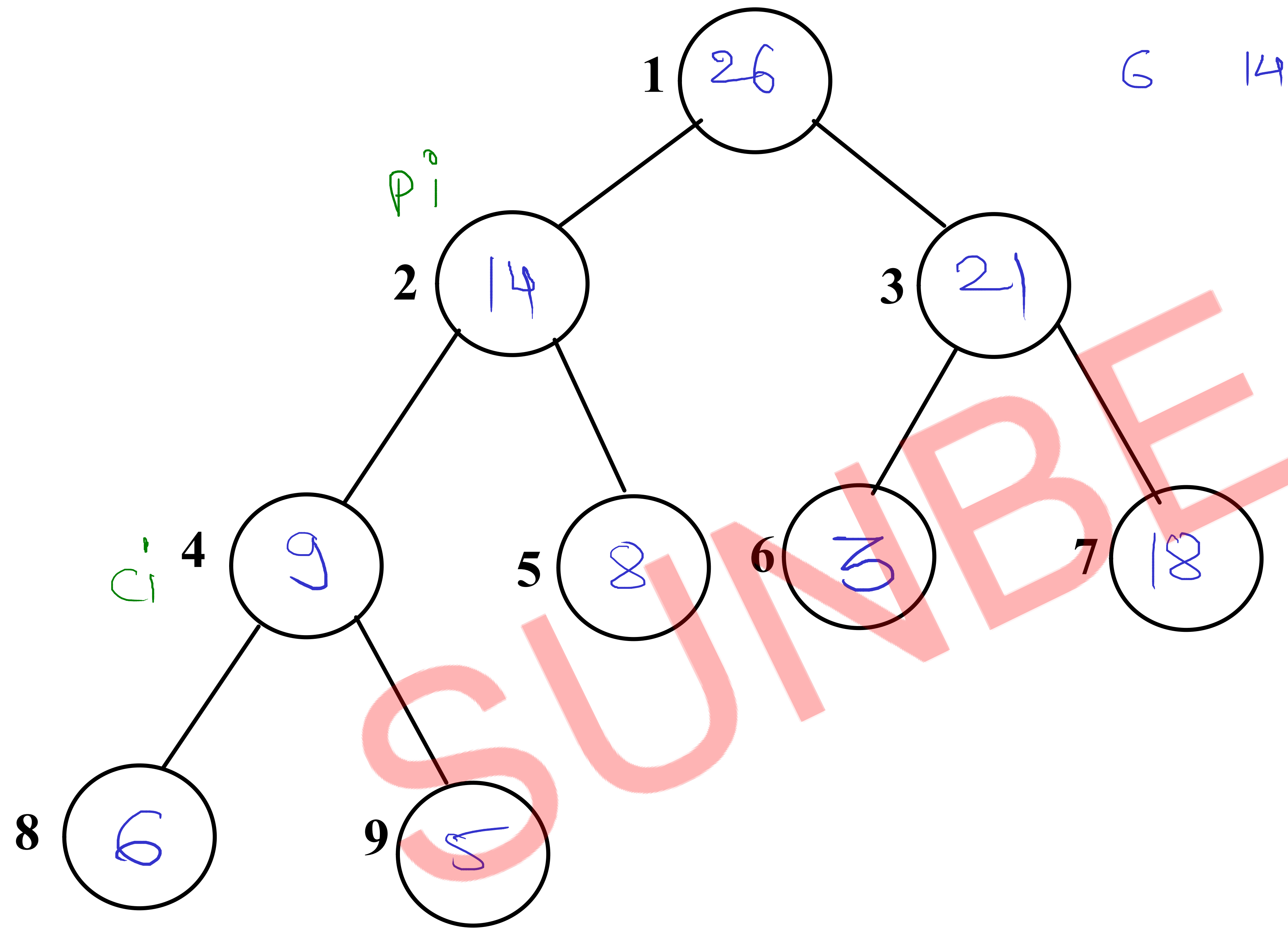






Heap - Create Heap

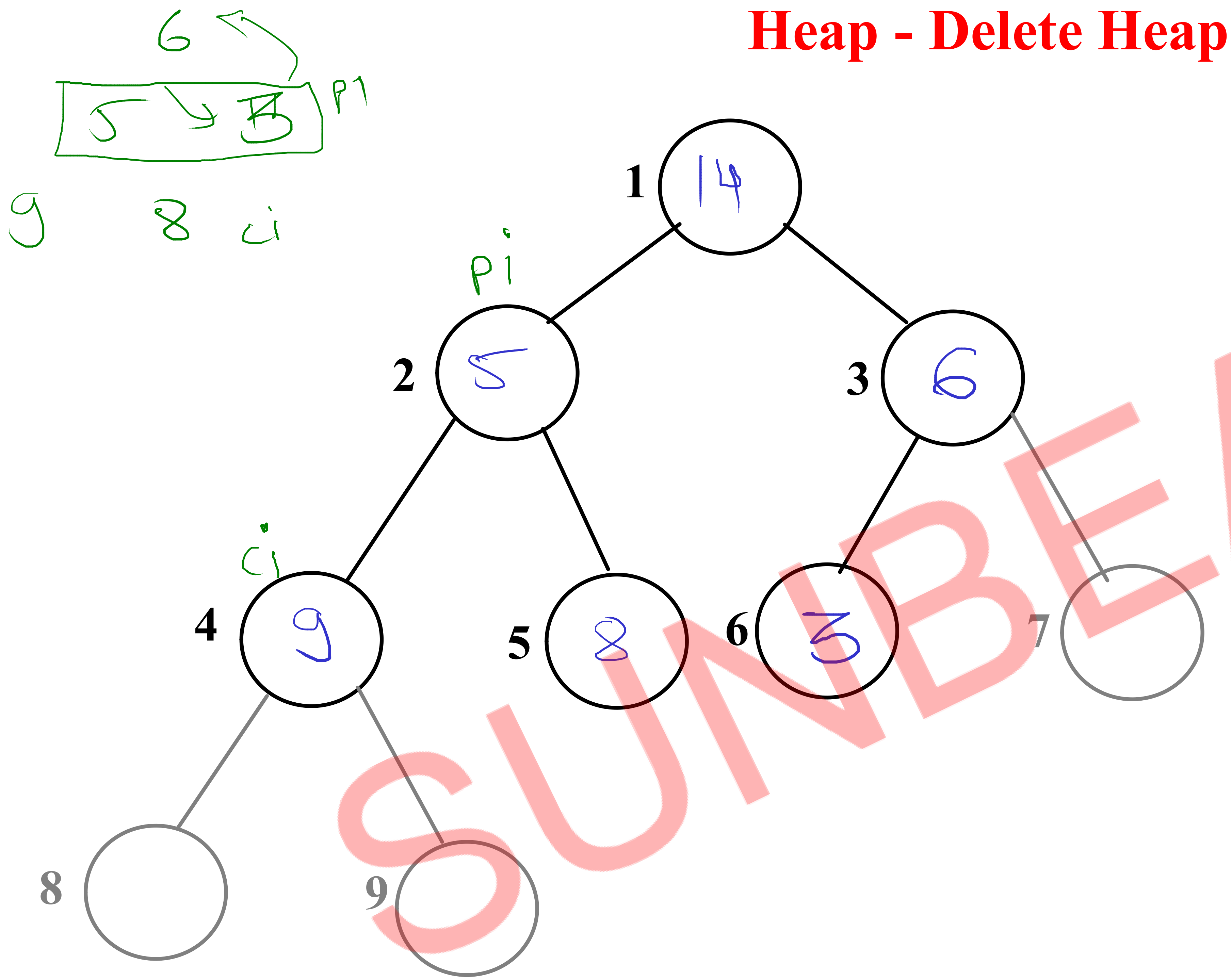
6 14 3 26 8 18 21 9 5



$$T(n) = O(\log n)$$
$$= O(h)$$

1	2	3	4	5	6	7	8	9
26	14	21	9	8	3	18	6	5

Heap - Delete Heap

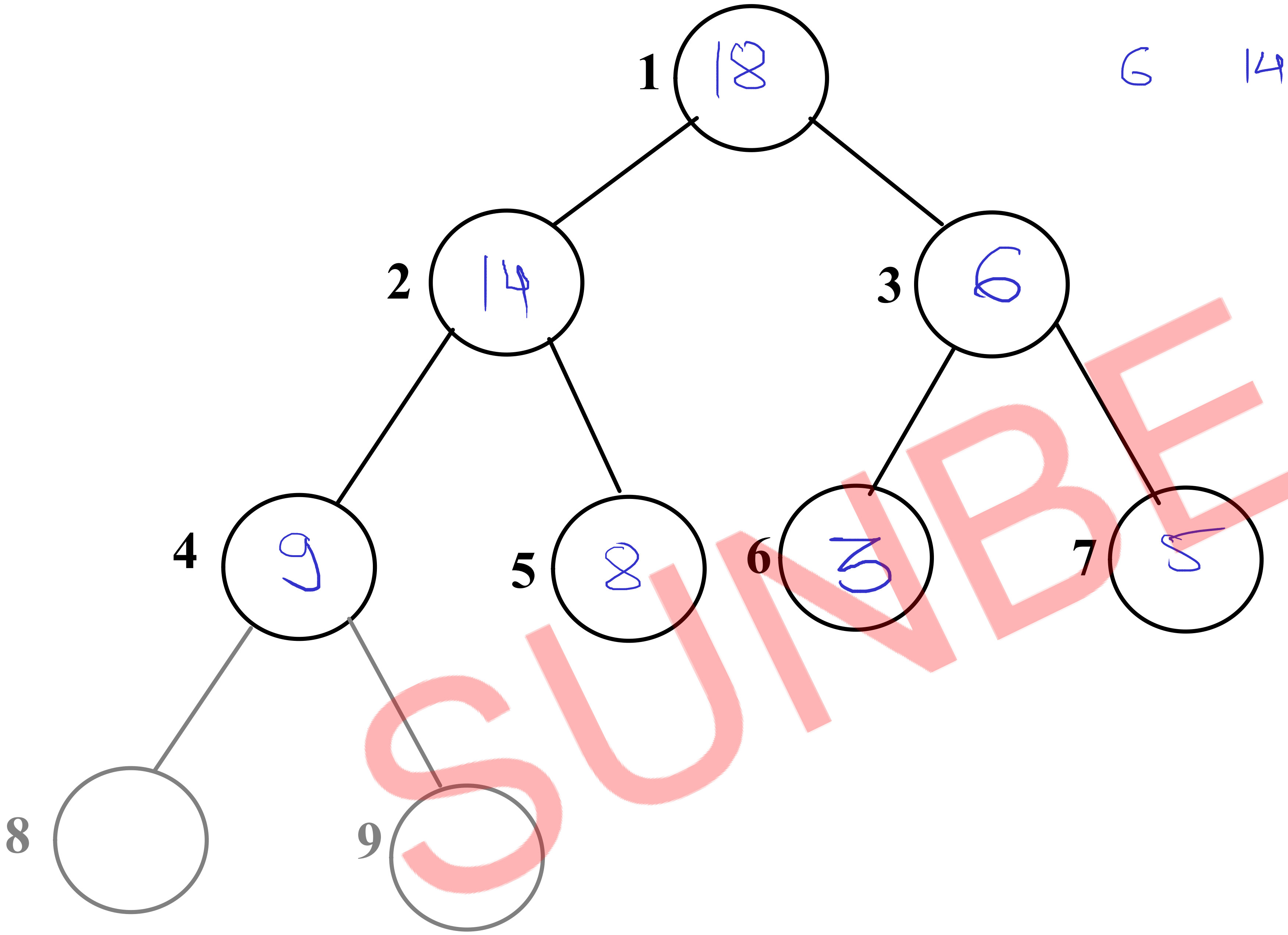


Max = 26
Max = 21
18

$T(n) = O(\log n)$
$= O(h)$

6	14	18	9	8	3	5		
1	2	3	4	5	6	7	8	9

Heap Sort



6 14 3 26 8 18 21 9 5

$$T(n) = O(\log n)$$
$$= O(h)$$

1	2	3	4	5	6	7	8	9
6	14	18	9	8	3	5	21	26