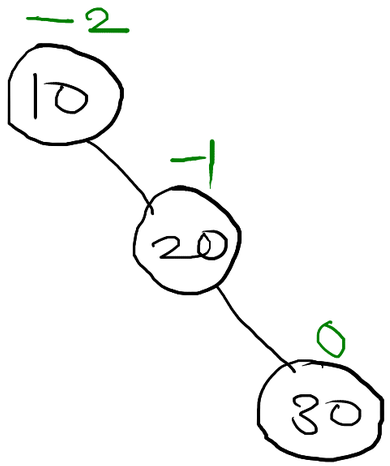
Balanced BST

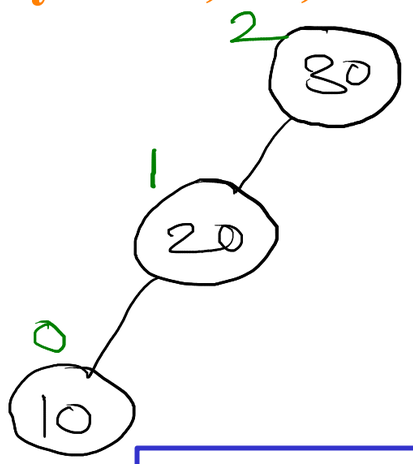
Balance Factor = height(left sub tree) - height(right sub tree)

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

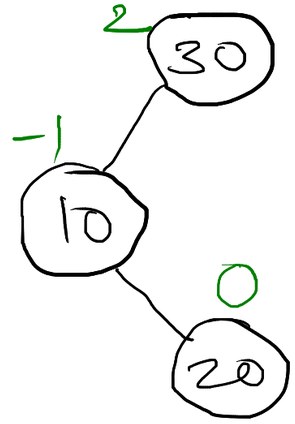
Keys : 10, 20 ,30



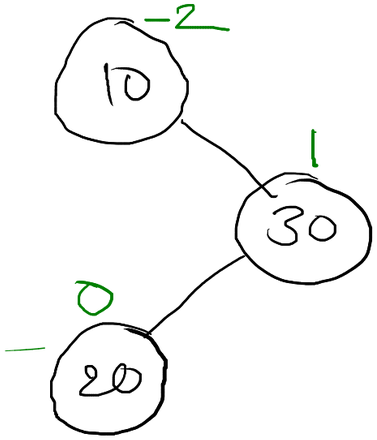
Keys : 30, 20, 10



Keys : 30, 10, 20

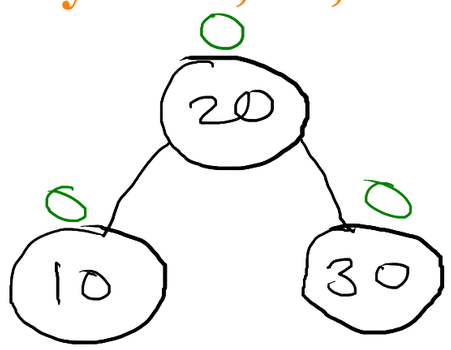


Keys : 10, 30, 20



Keys : 20, 10, 30

Keys : 20, 30, 10

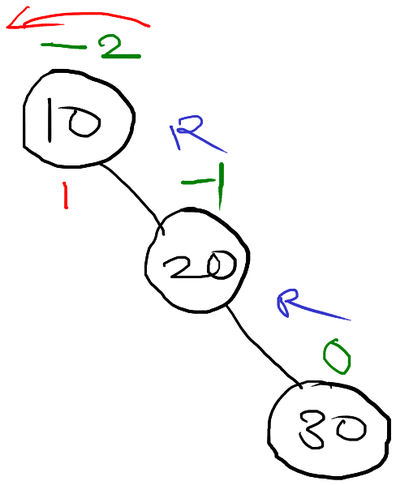


Balanced BST

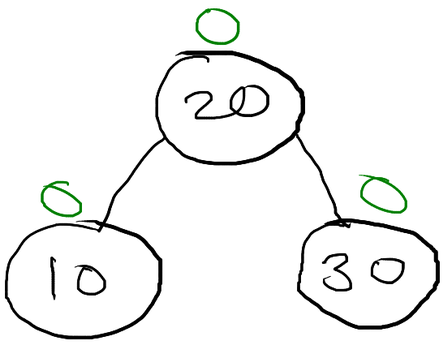
Rotations

RR Imbalance

Keys : 10, 20 ,30

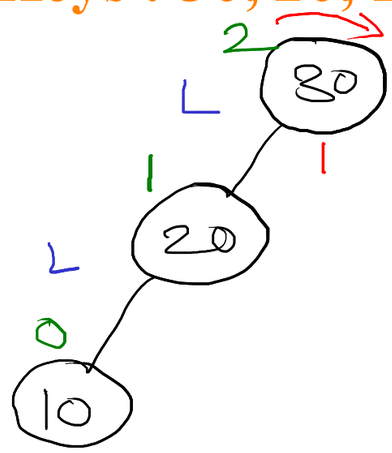


Left Rotation

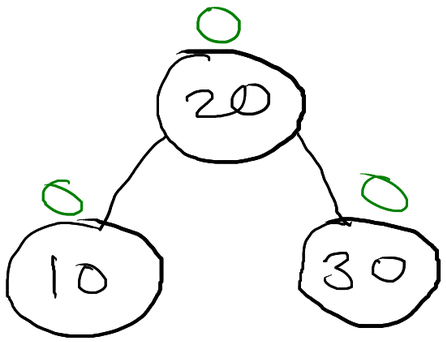


LL Imbalance

Keys : 30, 20, 10



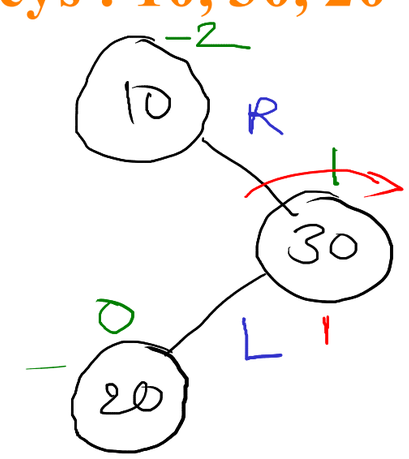
Right Rotation



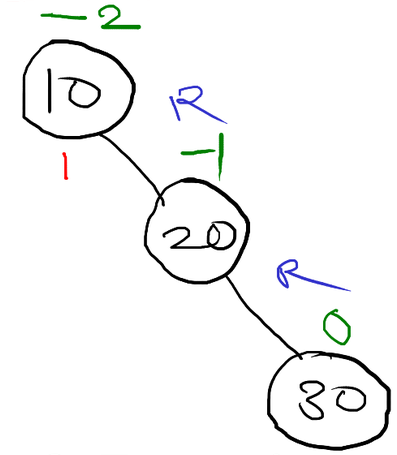
Single Rotation

RL Imbalance

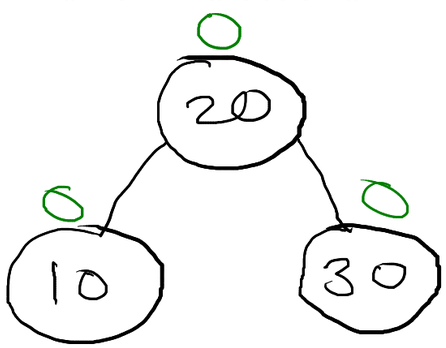
Keys : 10, 30, 20



Right Rotation

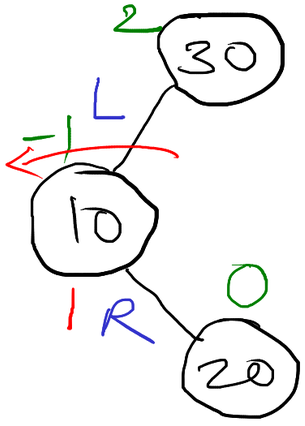


Left Rotation

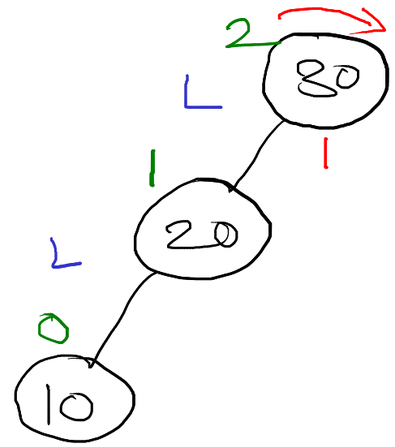


LR Imbalance

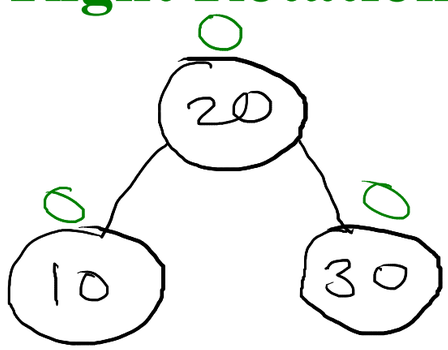
Keys : 30, 10, 20



Left Rotation



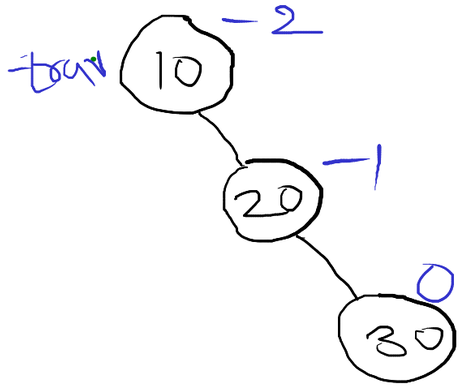
Right Rotation



Double Rotation

RR Imbalance

Keys : 10, 20 ,30

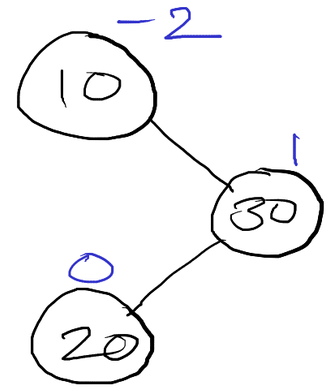


bf < -1

value > trav->right->data

RL Imbalance

Keys : 10, 30, 20

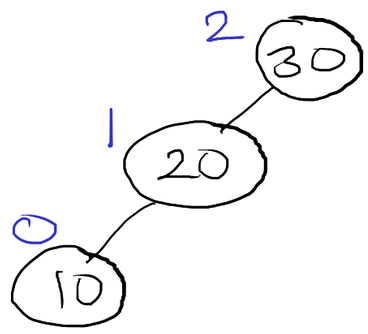


bf < -1

value < trav->right->data

LL Imbalance

Keys : 30, 20, 10

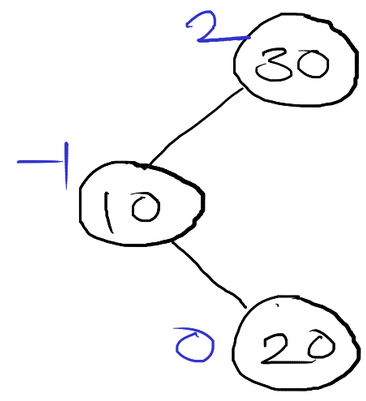


bf > 1

value < trav->left->data

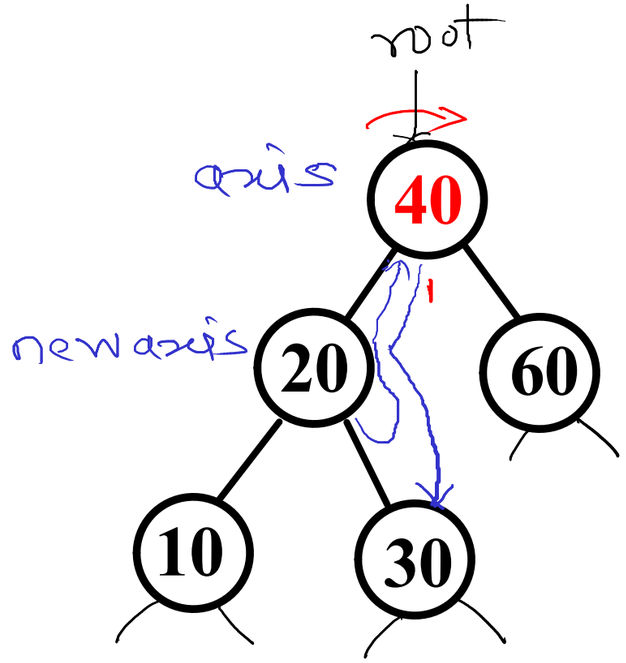
LR Imbalance

Keys : 30, 10, 20

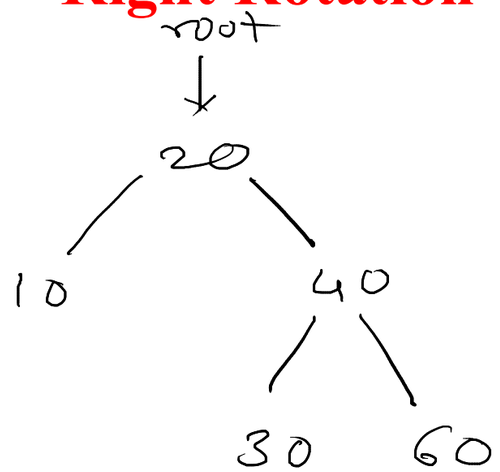


bf > 1

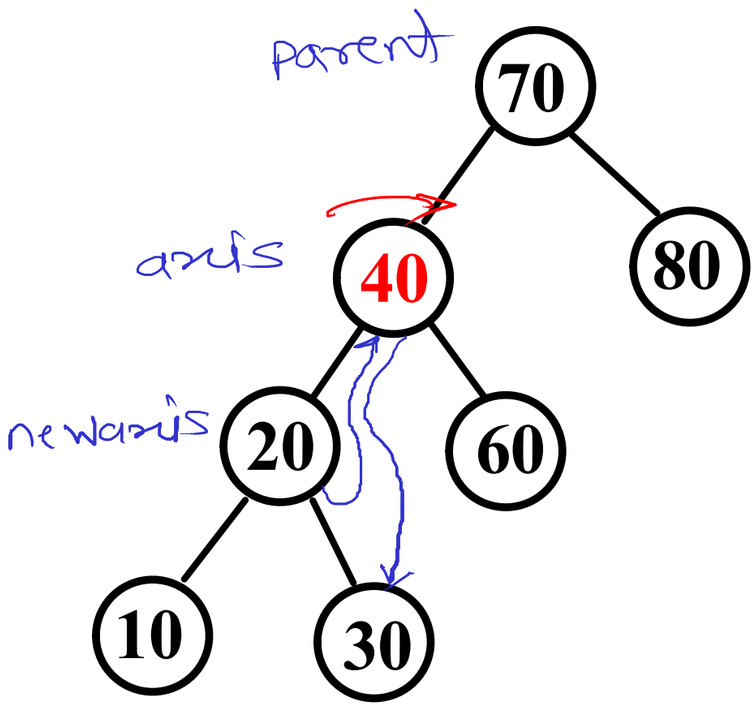
value > trav->left->data

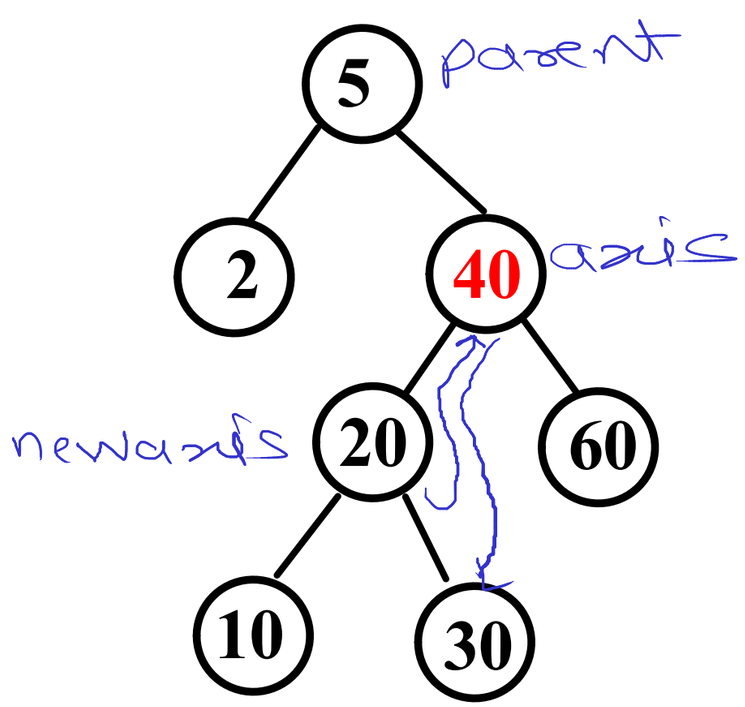


Right Rotation

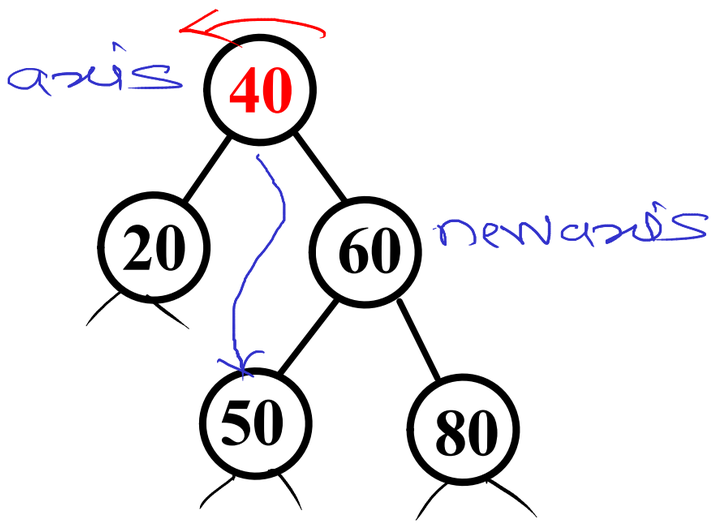


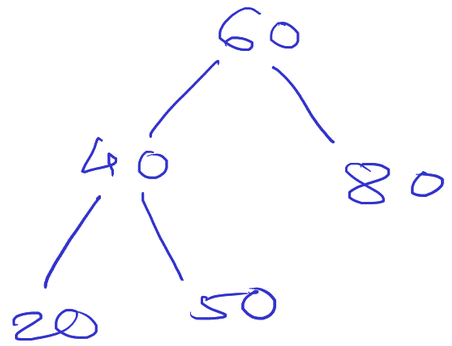

```
newaxis = axis->left
axis->left = newaxis->right
newaxis->right = axis
if(axis == root)
    root = newaxis;
elseif(axis == parent->left)
    parent->left = newaxis;
elseif(axis == parent->right)
    parent->right = newaxis;
```



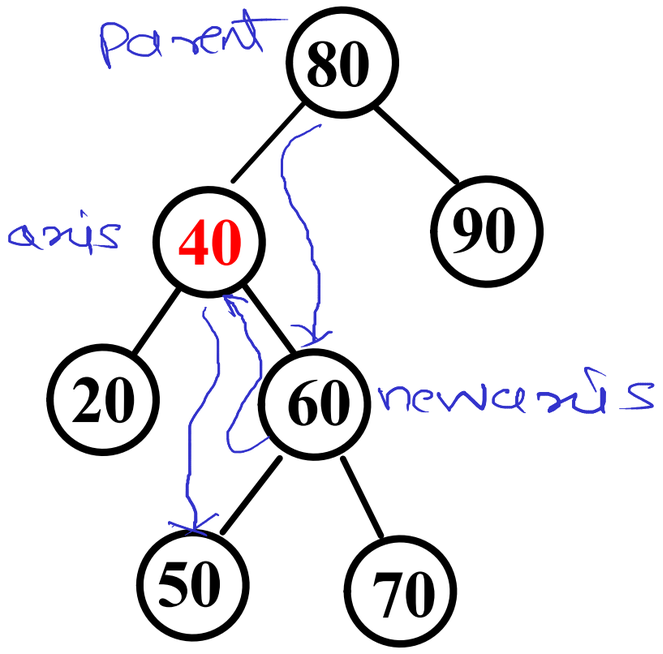


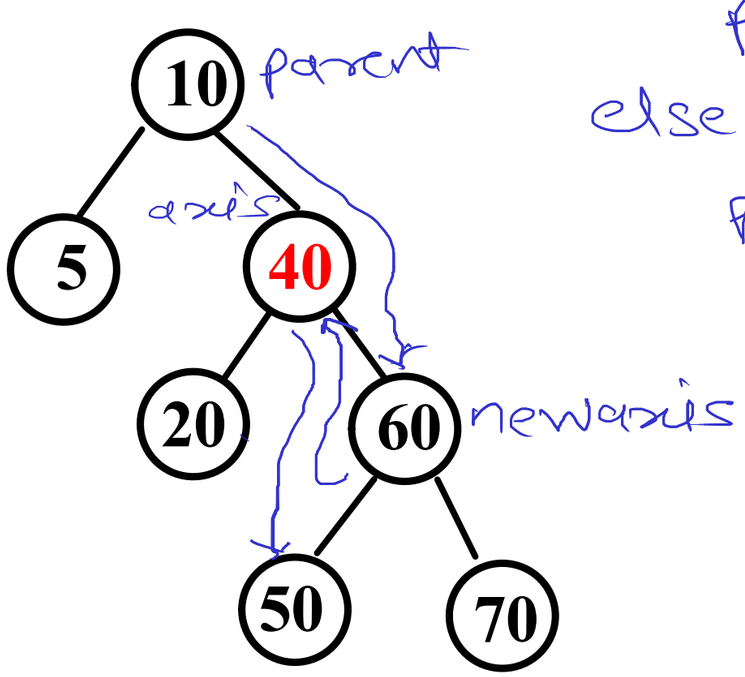
Left Rotation






```
newaxis = axis->right
axis->right = newaxis->left
newaxis->left = axis
if(axis == root)
    root = newaxis;
else if(axis == parent->left)
    parent->left = newaxis;
else if(axis == parent->right)
    parent->right = newaxis;
```

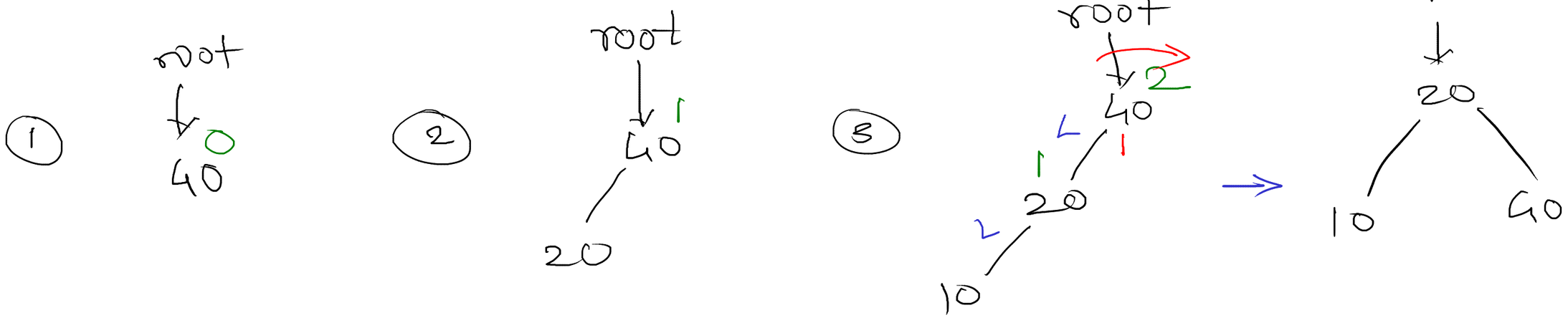


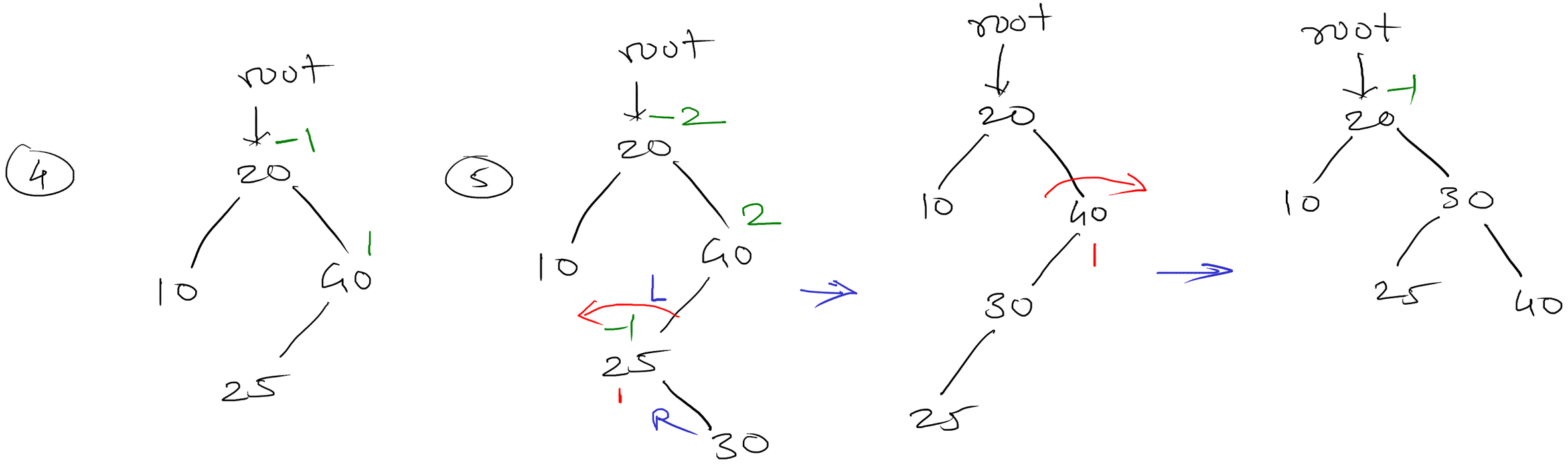


AVL Tree

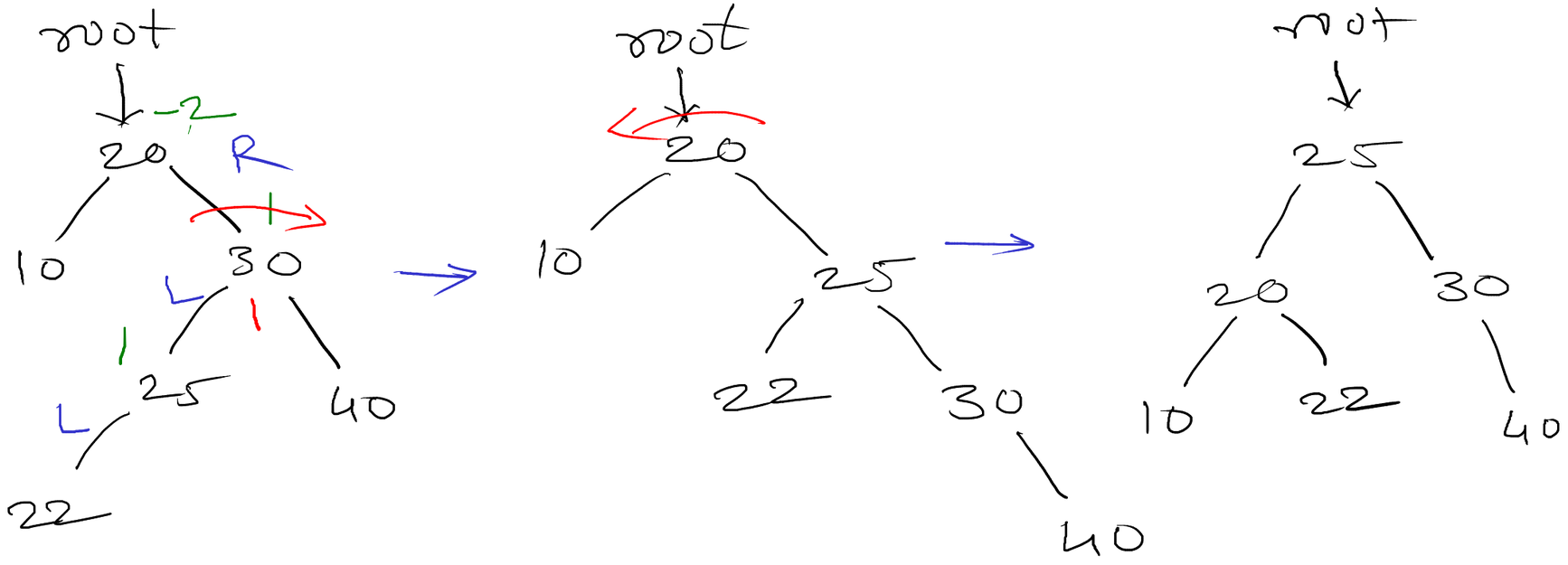
- Self balancing binary Search Tree
- on every insertion and deletion of node, tree is balanced
- All operation on AVL tree are perfromed in O(log n) time
- Balance factor of all nodes is either -1, 0 or +1

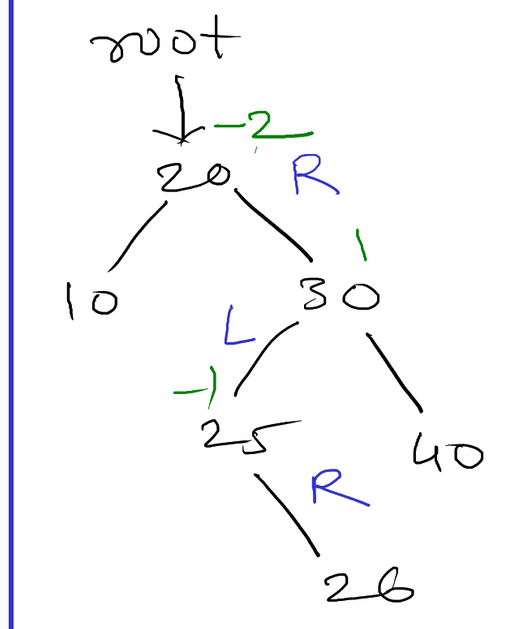
Keys : 40, 20, 10, 25, 30, 22, 50

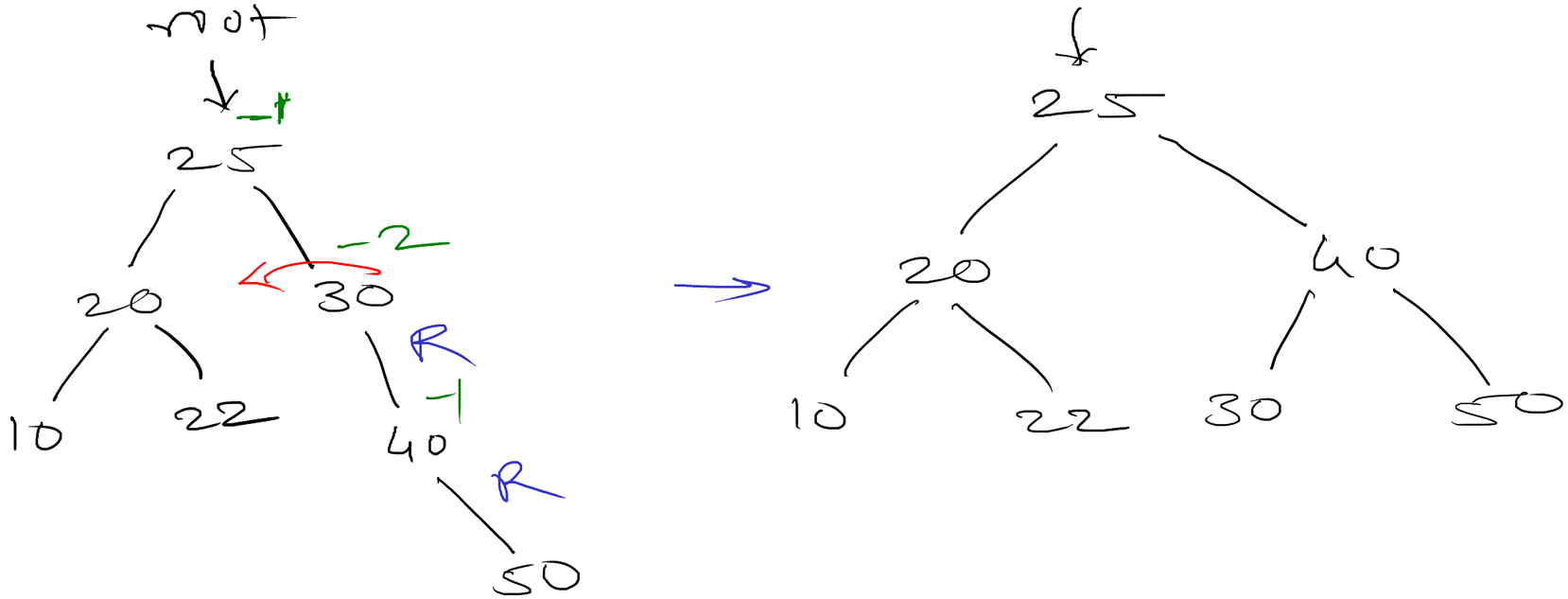




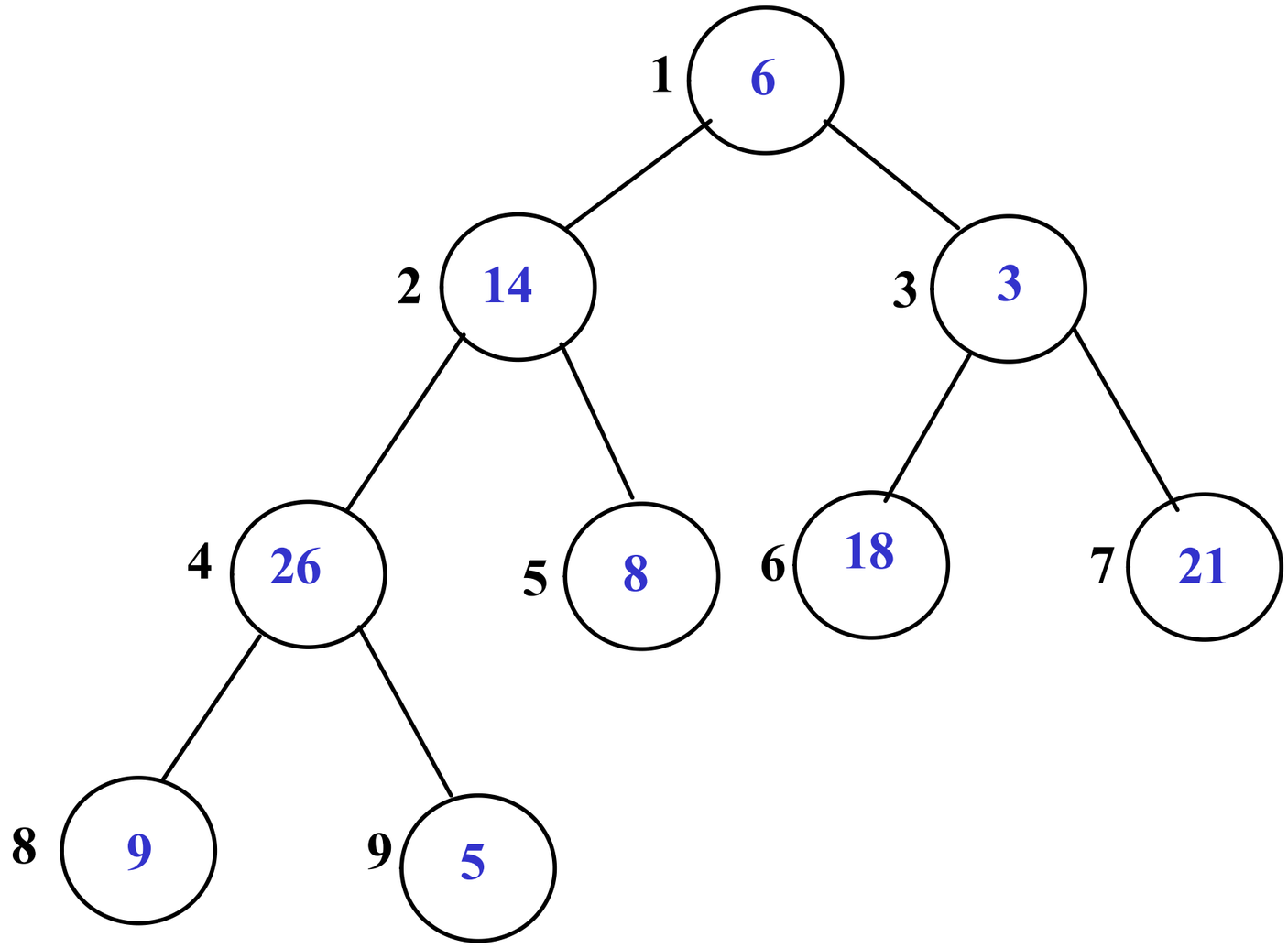
(6)


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


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


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

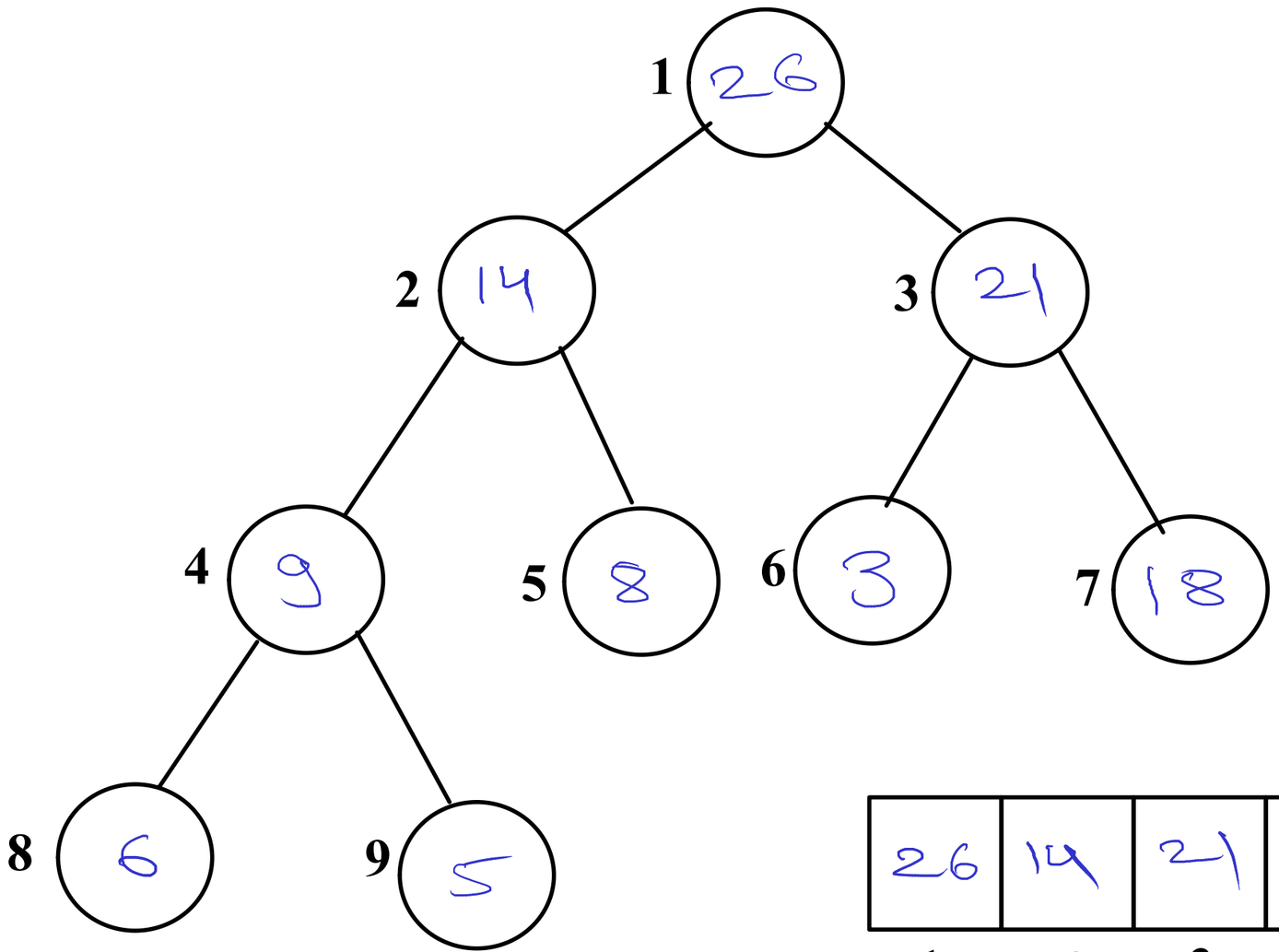
Almost Complete Binary Tree



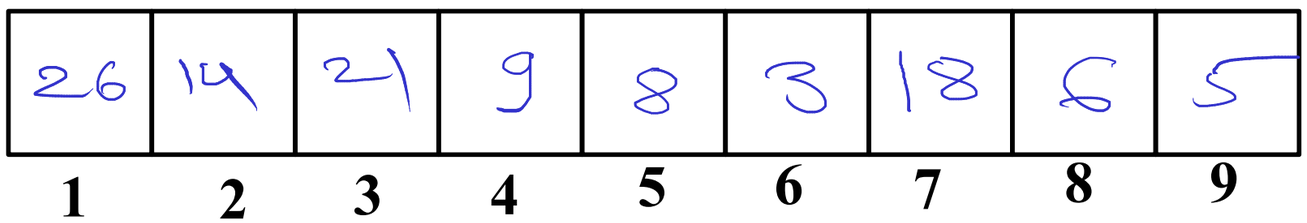
6	14	3	26	8	18	21	9	5
1	2	3	4	5	6	7	8	9

Create Max Heap

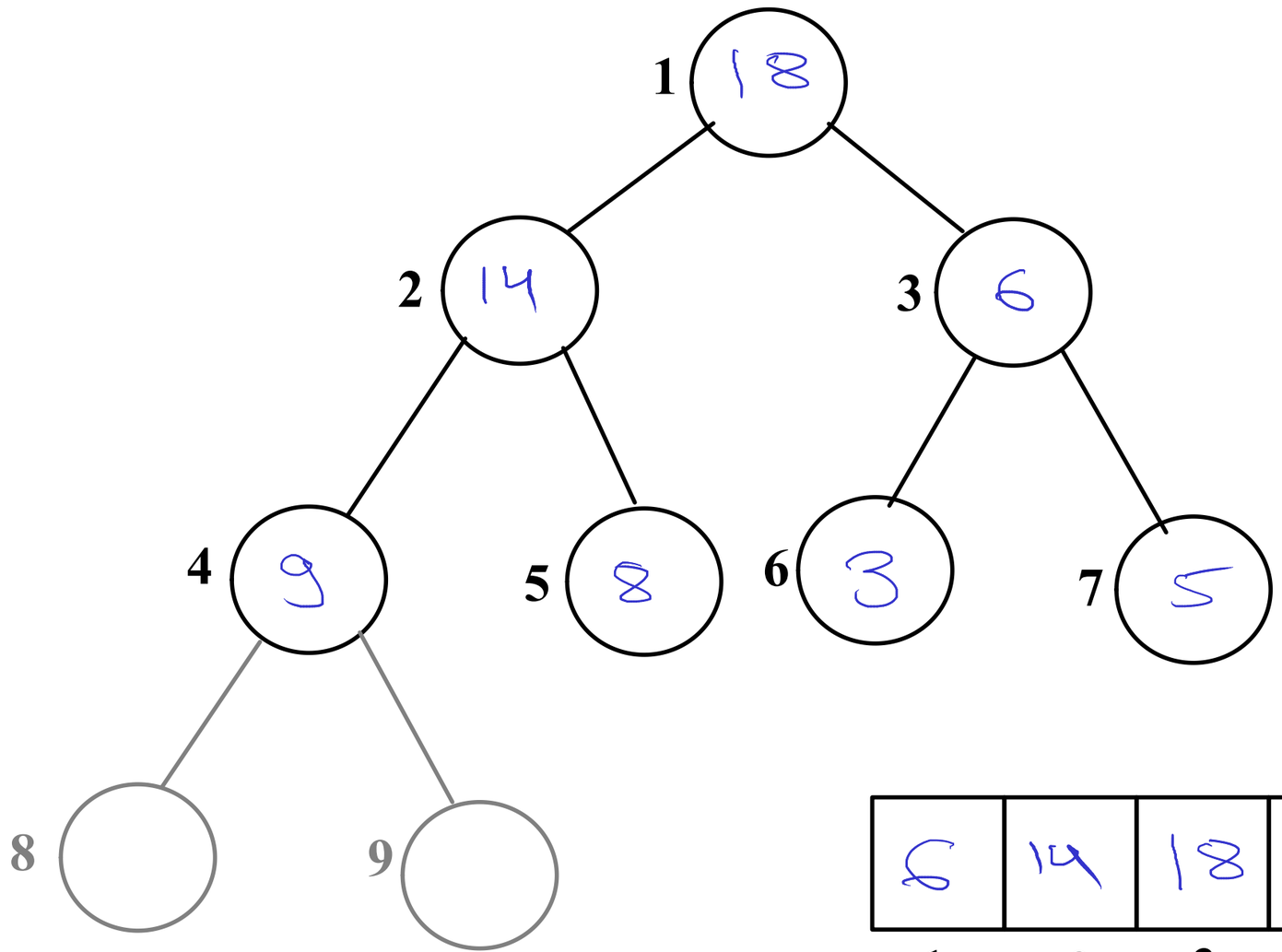
6 14 3 26 8 18 21 9 5



Add/insert:
$T(n) = O(\log n)$



Delete Max Heap



Delete/remove:

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

Max = 26

Max = 21