CS-2001
DATA STRUCTURE

Dr. Hashim Yasin

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

AVL TREE

AVL Trees

To maintain the height balanced property of the AVL tree after insertion or deletion, it is necessary to perform a *transformation* on the tree so that,

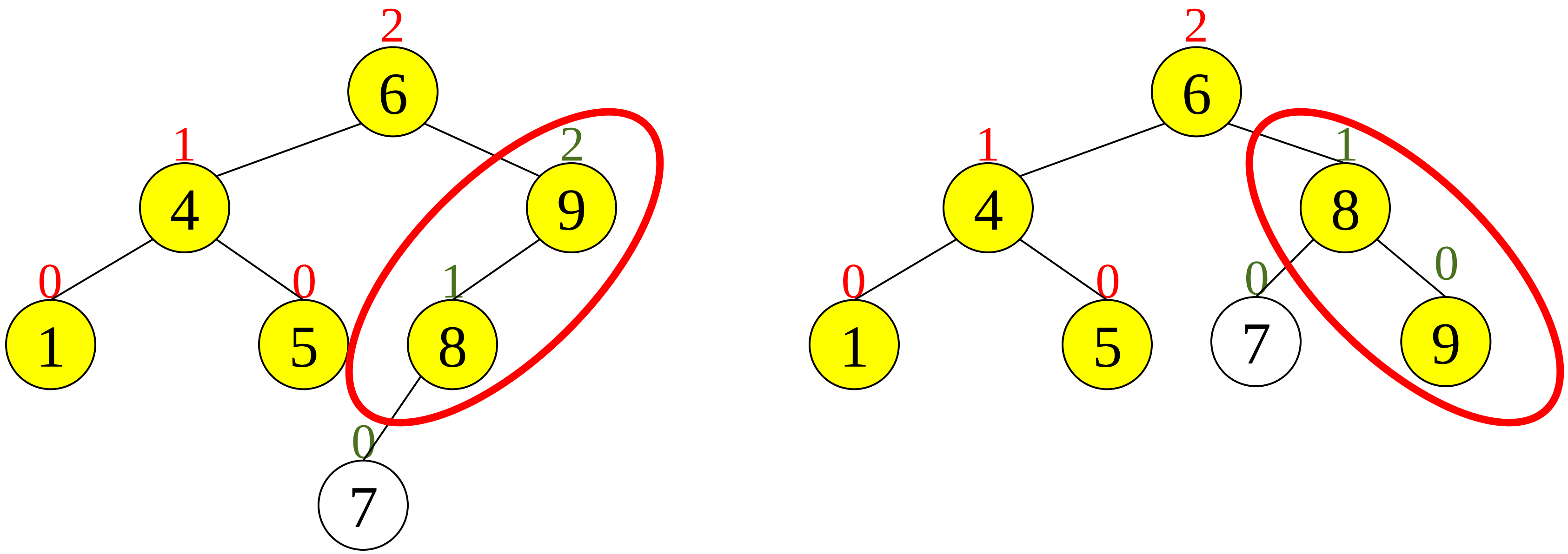
(1) the *in-order traversal of the transformed tree is the same as for the original tree* (i.e., the new tree remains a binary search tree).

(2) the tree after transformation is height-balanced.

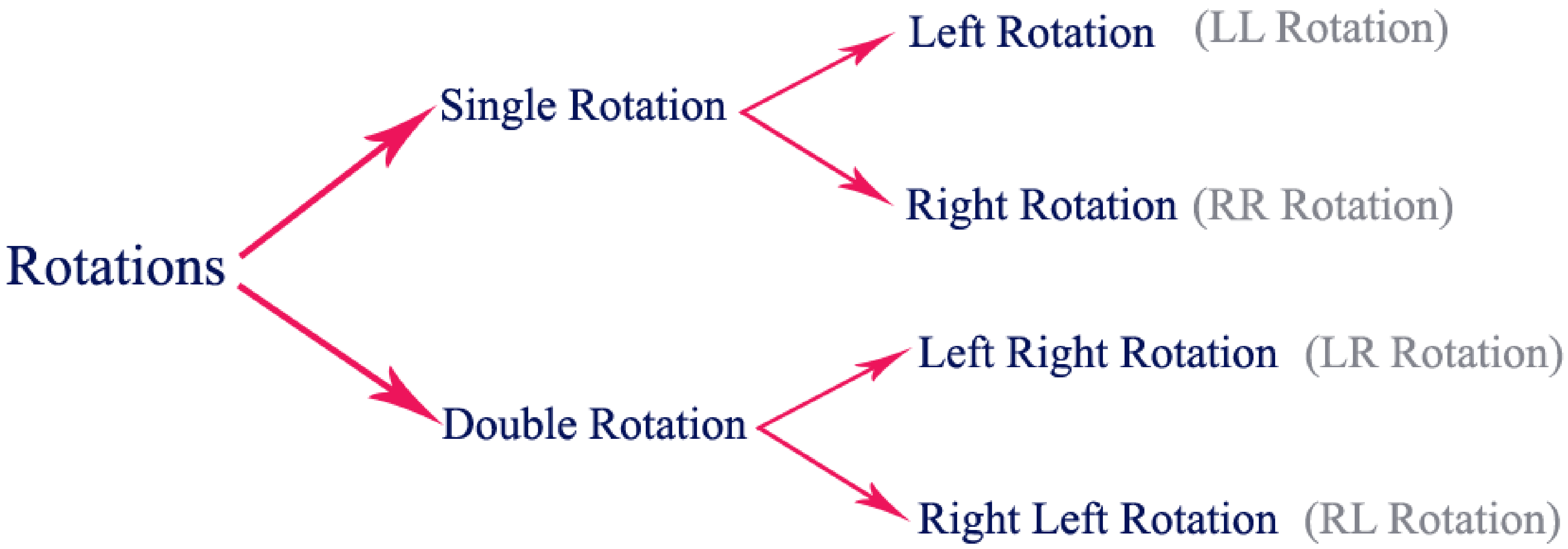
Insertion in AVL Trees

- Insert operation may cause balance factor to become 2 or –2 for some node
  - only nodes on the path from insertion point to root node have possibly changed in height
  - Follow the path up to the root, find the first node (i.e., deepest) whose new balance violates the AVL condition. Call this node *a*
  - If a new balance factor (the difference h_{left}-h_{right}) is 2 or –2, adjust tree by *rotation* around the node

AVL Tree



AVL Tree … Rotations

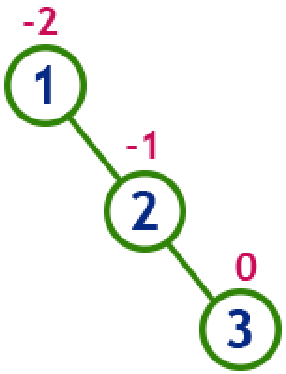


Rotations
Single Rotation
Double Rotation
Left Rotation (LL Rotation)
Right Rotation (RR Rotation)
Left Right Rotation (LR Rotation)
Right Left Rotation (RL Rotation)

LL Rotation

- In LL Rotation, every node moves *one position to **left** from the current position*.

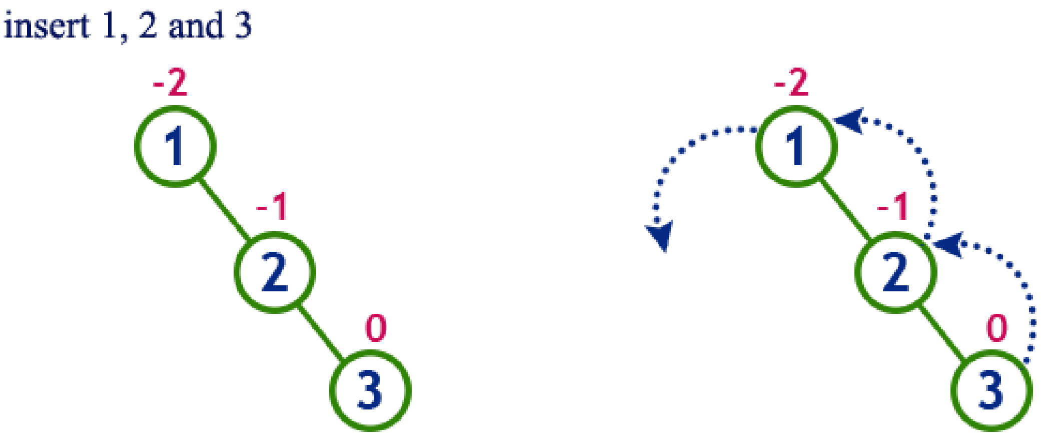
insert 1, 2 and 3



Tree is imbalanced

LL Rotation

- In LL Rotation, every node moves *one position to **left** from the current position*.

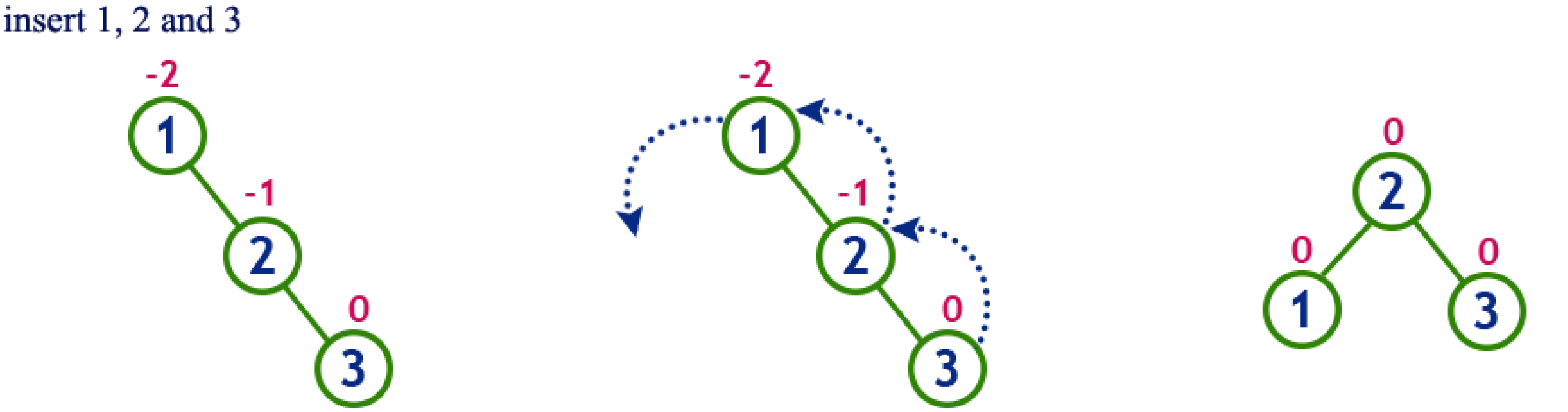


Tree is imbalanced

To make balanced we use LL Rotation which moves nodes one position to left

LL Rotation

- In LL Rotation, every node moves *one position to **left** from the current position*.



Tree is imbalanced

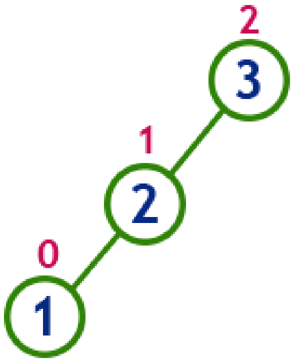
To make balanced we use LL Rotation which moves nodes one position to left

After LL Rotation Tree is Balanced

RR Rotation

- In RR Rotation, every node moves *one position to* ***right*** *from the current position*.

insert 3, 2 and 1



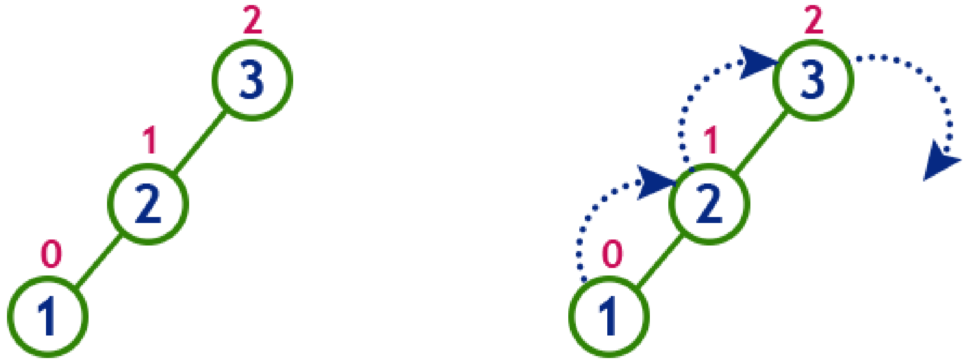
Tree is imbalanced

because node 3 has balance factor 2

RR Rotation

- In RR Rotation, every node moves *one position to* ***right*** *from the current position*.

insert 3, 2 and 1



Tree is imbalanced

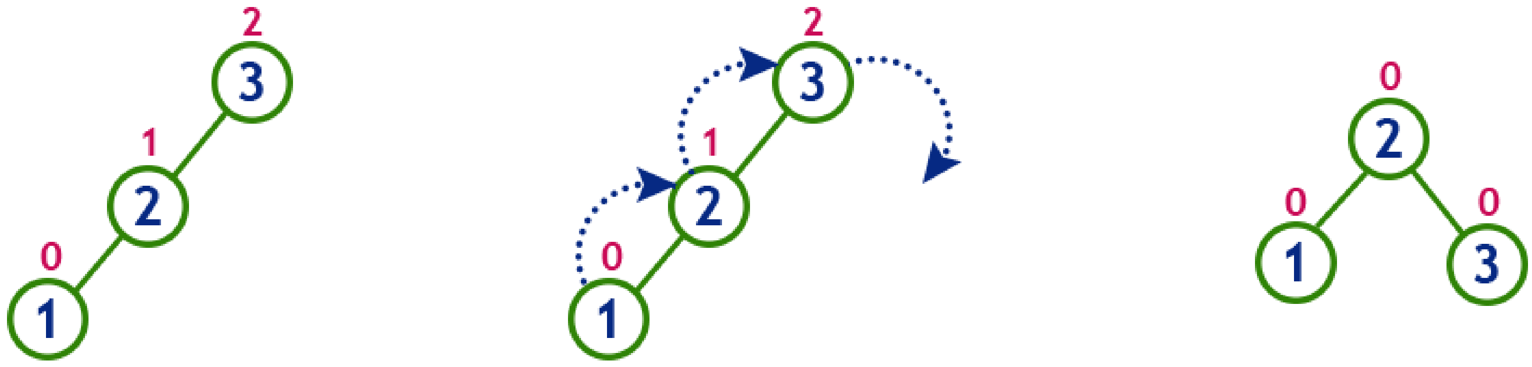
because node 3 has balance factor 2

To make balanced we use RR Rotation which moves nodes one position to right

RR Rotation

- In RR Rotation, every node moves *one position to* ***right*** *from the current position*.

insert 3, 2 and 1



Tree is imbalanced

because node 3 has balance factor 2

To make balanced we use RR Rotation which moves nodes one position to right

After RR Rotation Tree is Balanced

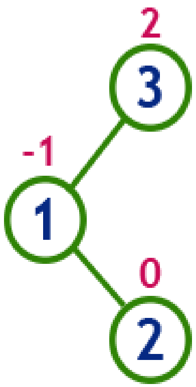
LR Rotation

- The LR Rotation is a *sequence of single left rotation followed by a single right rotation*.

- In LR Rotation, at first,
  - every node moves one position to the left and
  - one position to right from the current position.

LR Rotation

insert 3, 1 and 2

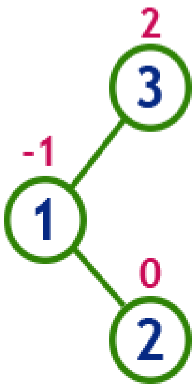


Tree is imbalanced

because node 3 has balance factor 2

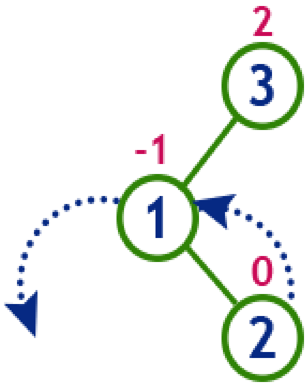
LR Rotation

insert 3, 1 and 2



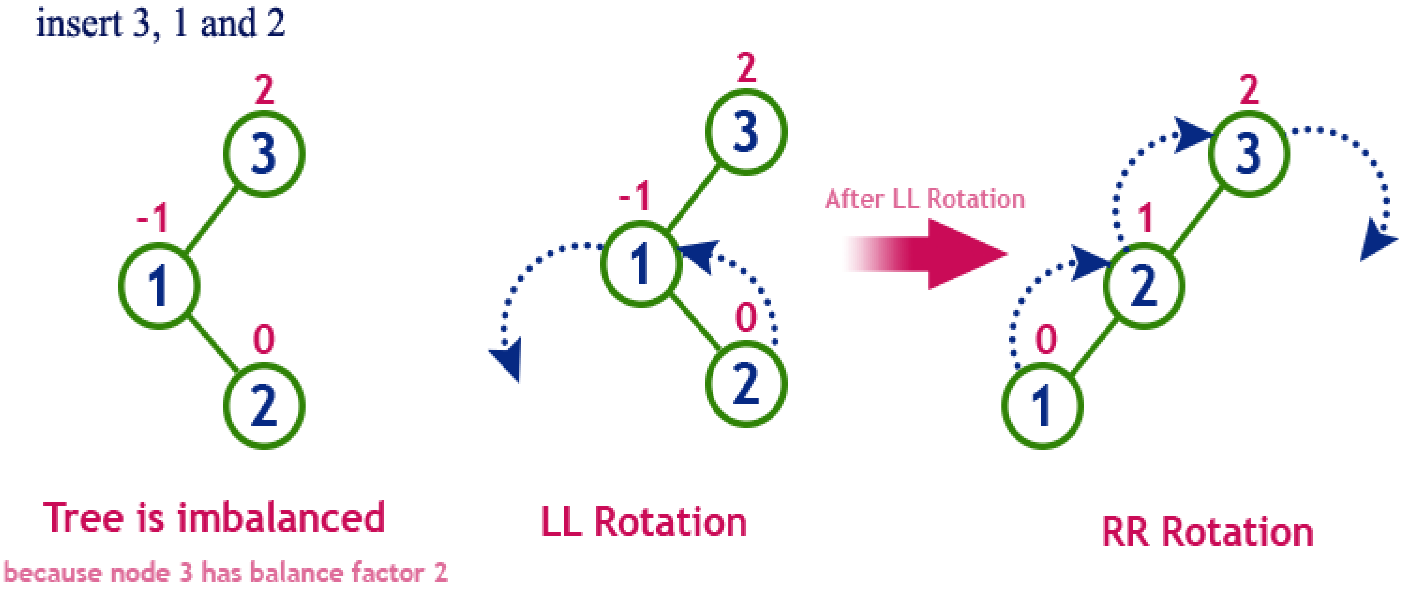
Tree is imbalanced

because node 3 has balance factor 2

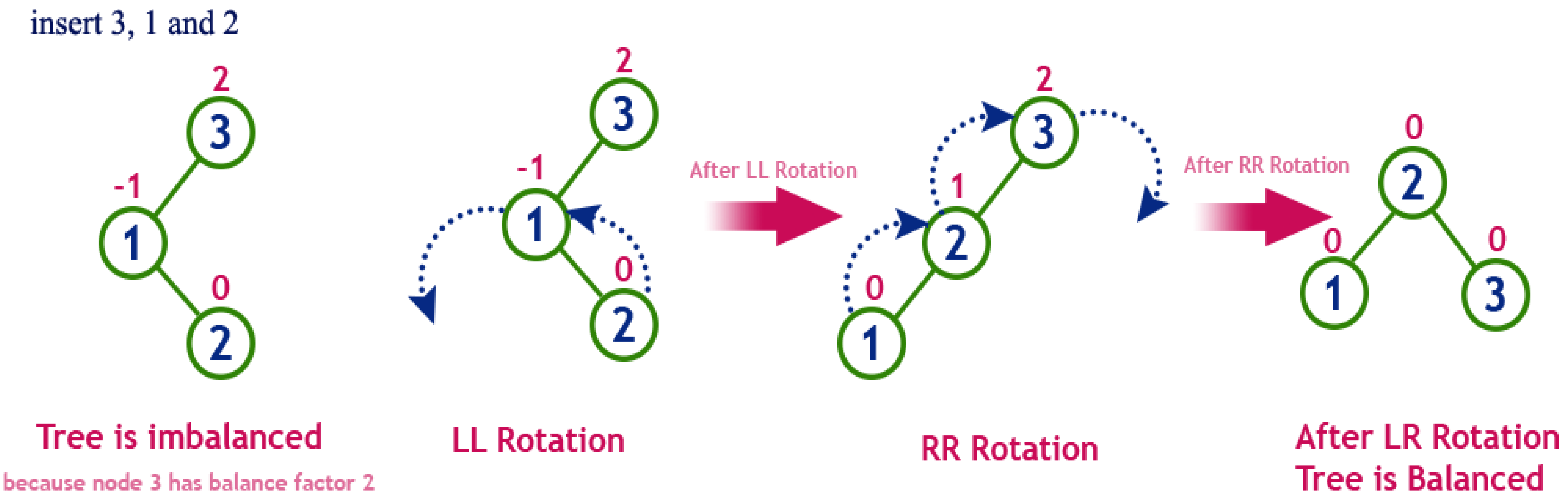


LL Rotation

LR Rotation


insert 3, 1 and 2
2
3
-1
1
0
2
Tree is imbalanced
because node 3 has balance factor 2
2
3
-1
1
0
2
LL Rotation
After LL Rotation
2
3
1
2
0
1
RR Rotation

LR Rotation


insert 3, 1 and 2
2
3
-1
1
0
2
2
3
-1
1
0
2
After LL Rotation
2
3
1
2
0
1
After RR Rotation
0
2
0
1
0
3
Tree is imbalanced
because node 3 has balance factor 2
LL Rotation
RR Rotation
After LR Rotation
Tree is Balanced

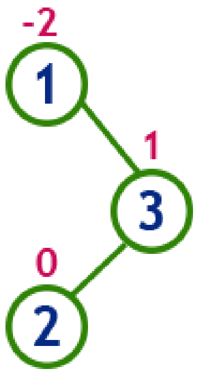
RL Rotation

- The RL Rotation is *sequence of single right rotation followed by single left rotation*.

- In RL Rotation, at first
  - every node moves one position to right and
  - one position to left from the current position.

RL Rotation

insert 1, 3 and 2

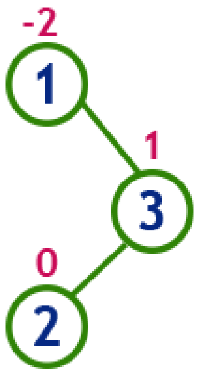


Tree is imbalanced

because node 1 has balance factor -2

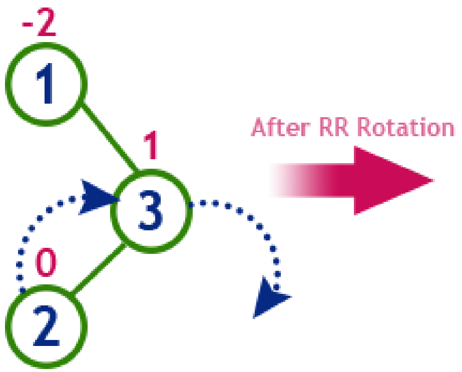
RL Rotation

insert 1, 3 and 2



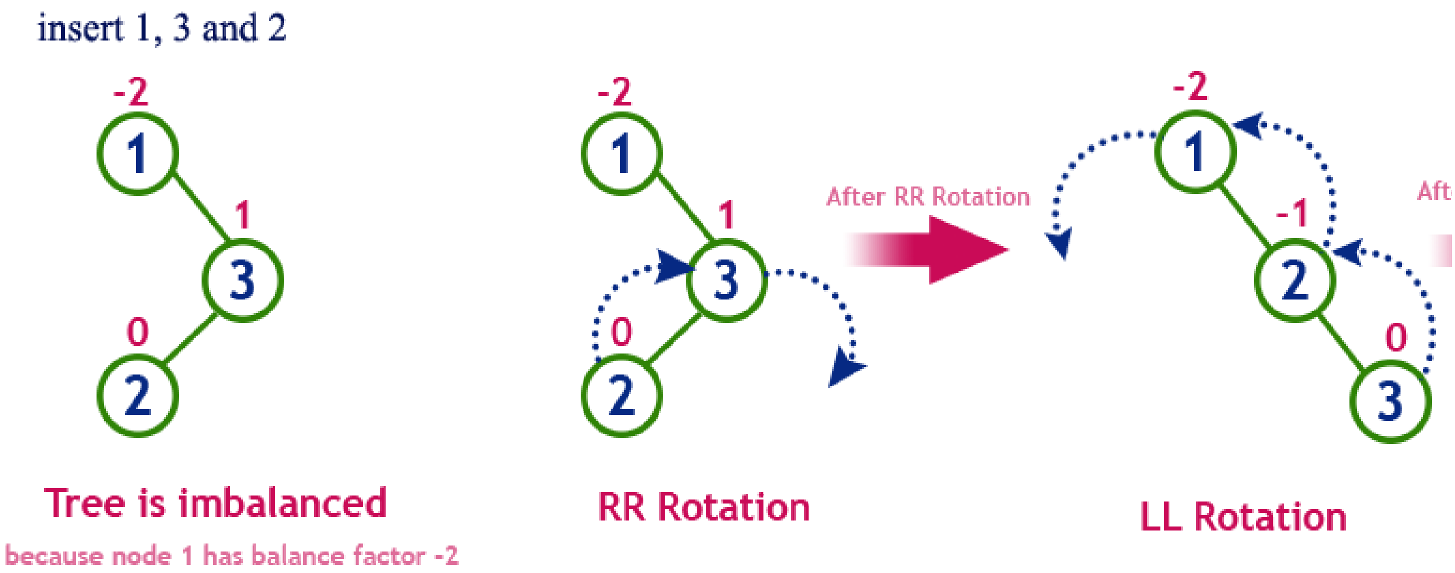
Tree is imbalanced

because node 1 has balance factor -2



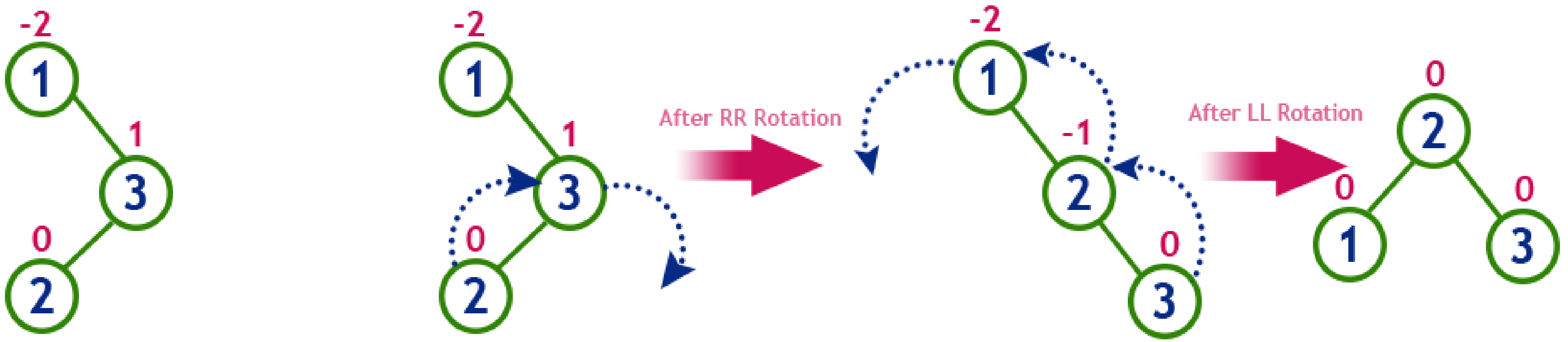
RR Rotation

RL Rotation


insert 1, 3 and 2
-2
1
1
3
0
2
Tree is imbalanced
because node 1 has balance factor -2
-2
1
1
3
0
2
RR Rotation
After RR Rotation
-2
1
-1
2
0
3
LL Rotation
Aft

RL Rotation

insert 1, 3 and 2



Tree is imbalanced

because node 1 has balance factor -2

RR Rotation

LL Rotation

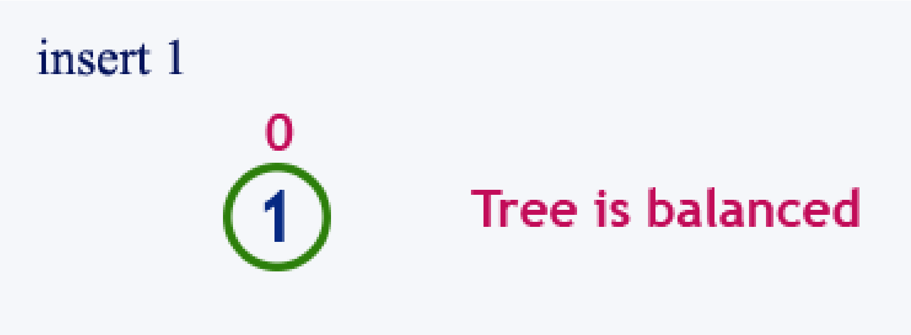
After RL Rotation
Tree is Balanced

EXAMPLE

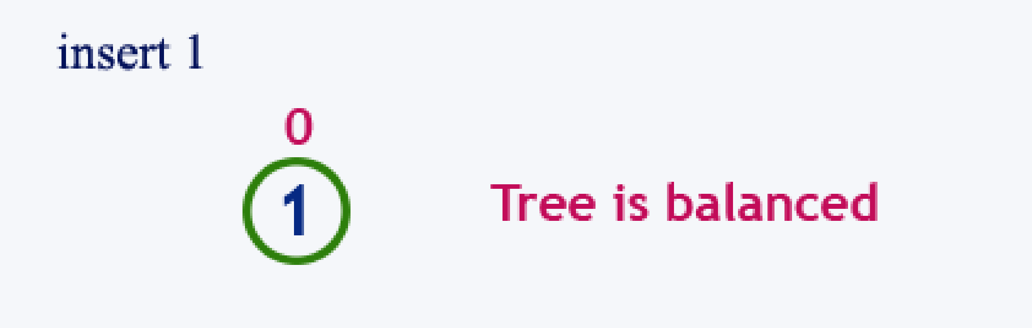
Example

Construct an AVL Tree by inserting numbers from 1 to 8.

Example

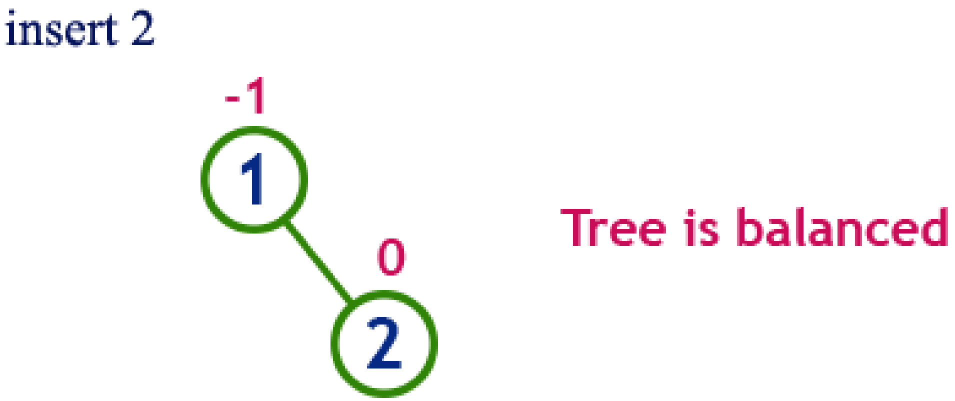


Example



insert 1

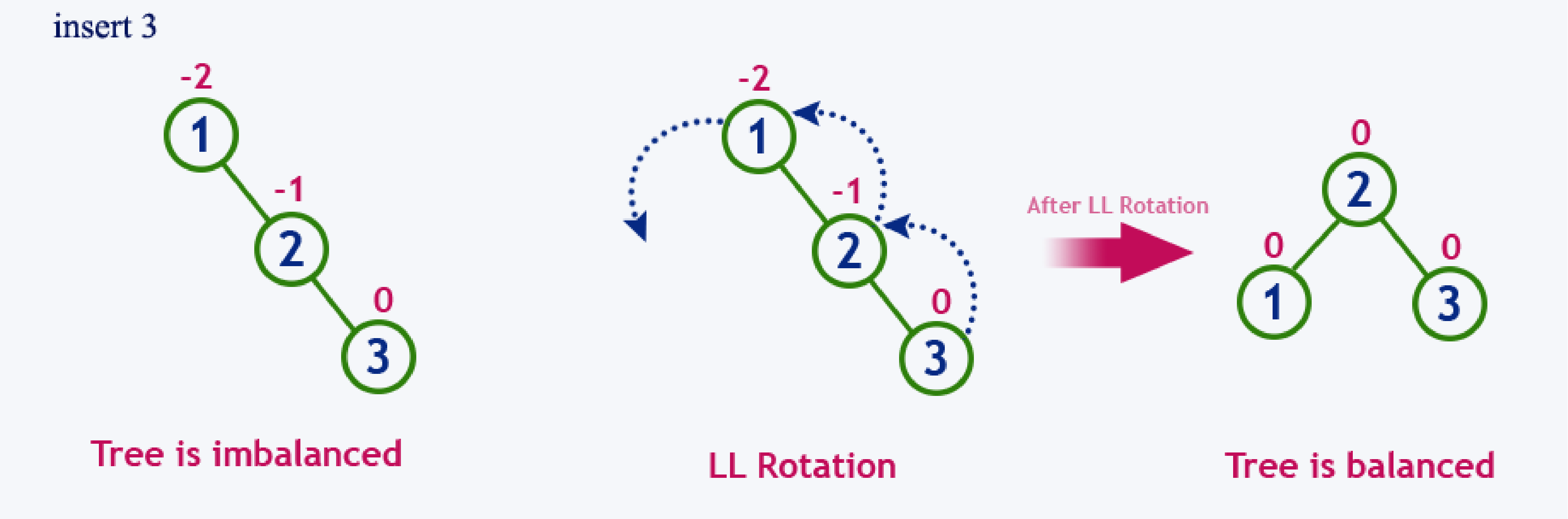
Tree is balanced



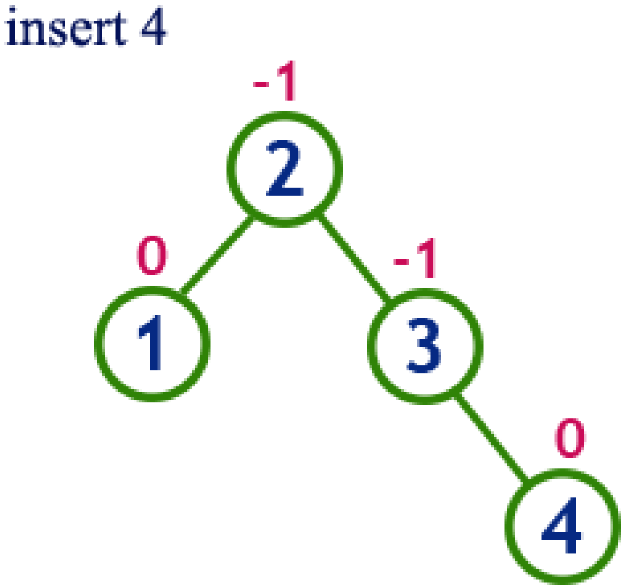
insert 2

Tree is balanced

Example

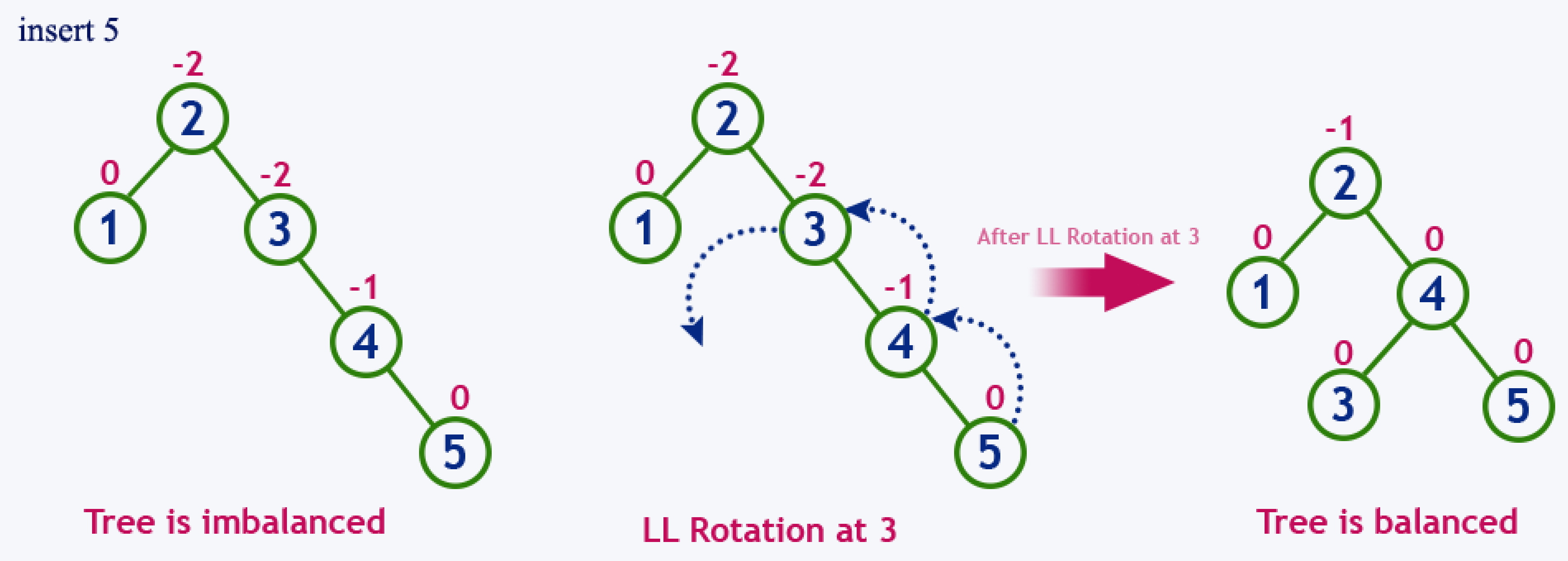

insert 3
-2
1
-1
2
0
3
Tree is imbalanced
-2
1
-1
2
0
3
LL Rotation
After LL Rotation
0
2
0
1
0
3
Tree is balanced

Example

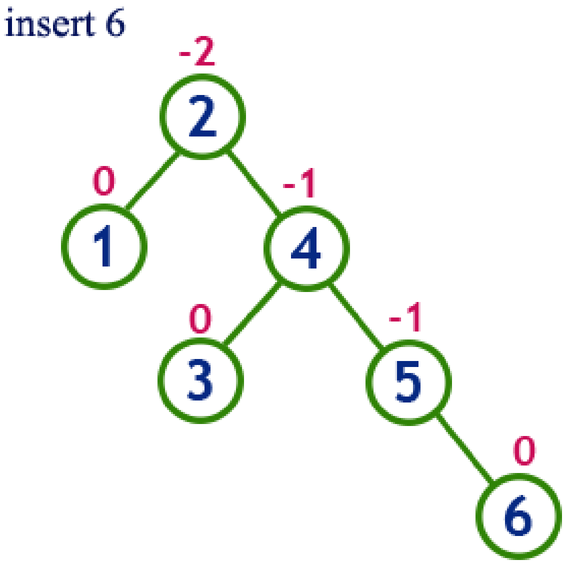


Tree is balanced

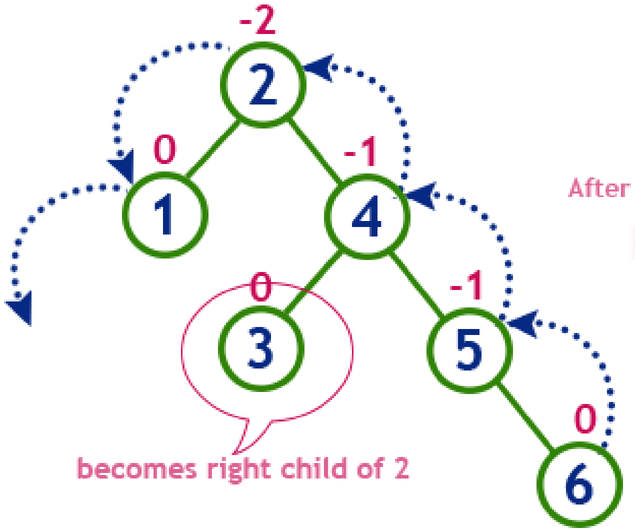
Example


insert 5
-2
2
0
1
-2
3
-1
4
0
5
Tree is imbalanced
-2
2
0
1
-2
3
-1
4
0
5
LL Rotation at 3
After LL Rotation at 3
-1
2
0
1
0
4
0
3
0
5
Tree is balanced

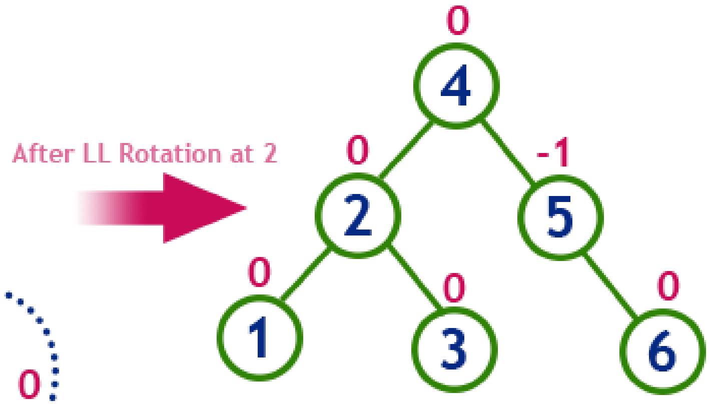
Example



Tree is imbalanced

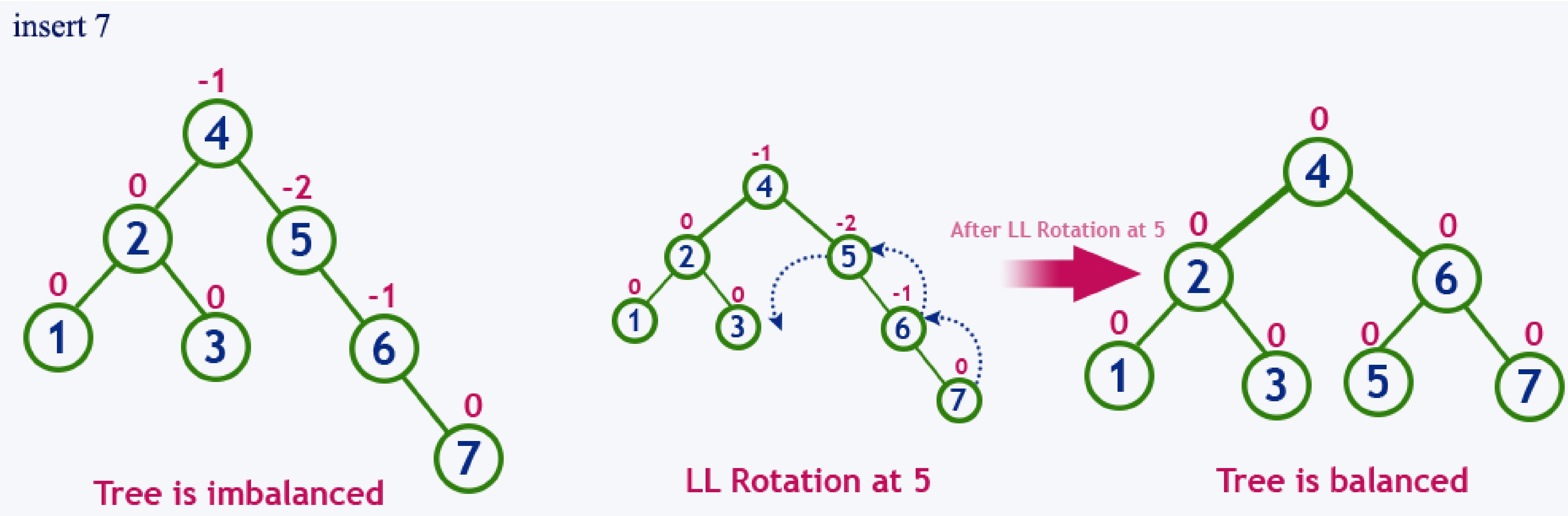


LL Rotation at 2

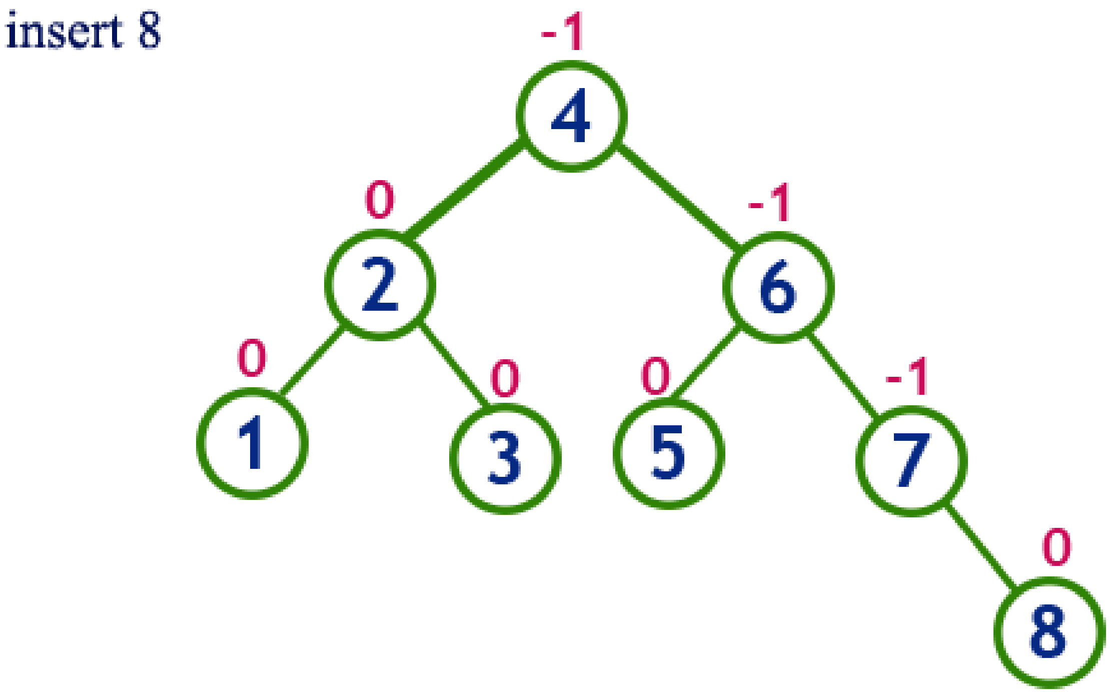


Tree is balanced

Example


insert 7
-1
4
0
2
-2
5
0
1
0
3
-1
6
0
7
Tree is imbalanced
-1
4
0
2
-2
5
0
1
0
3
-1
6
0
7
LL Rotation at 5
After LL Rotation at 5
0
4
0
2
0
6
0
1
0
3
0
5
0
7
Tree is balanced

Example



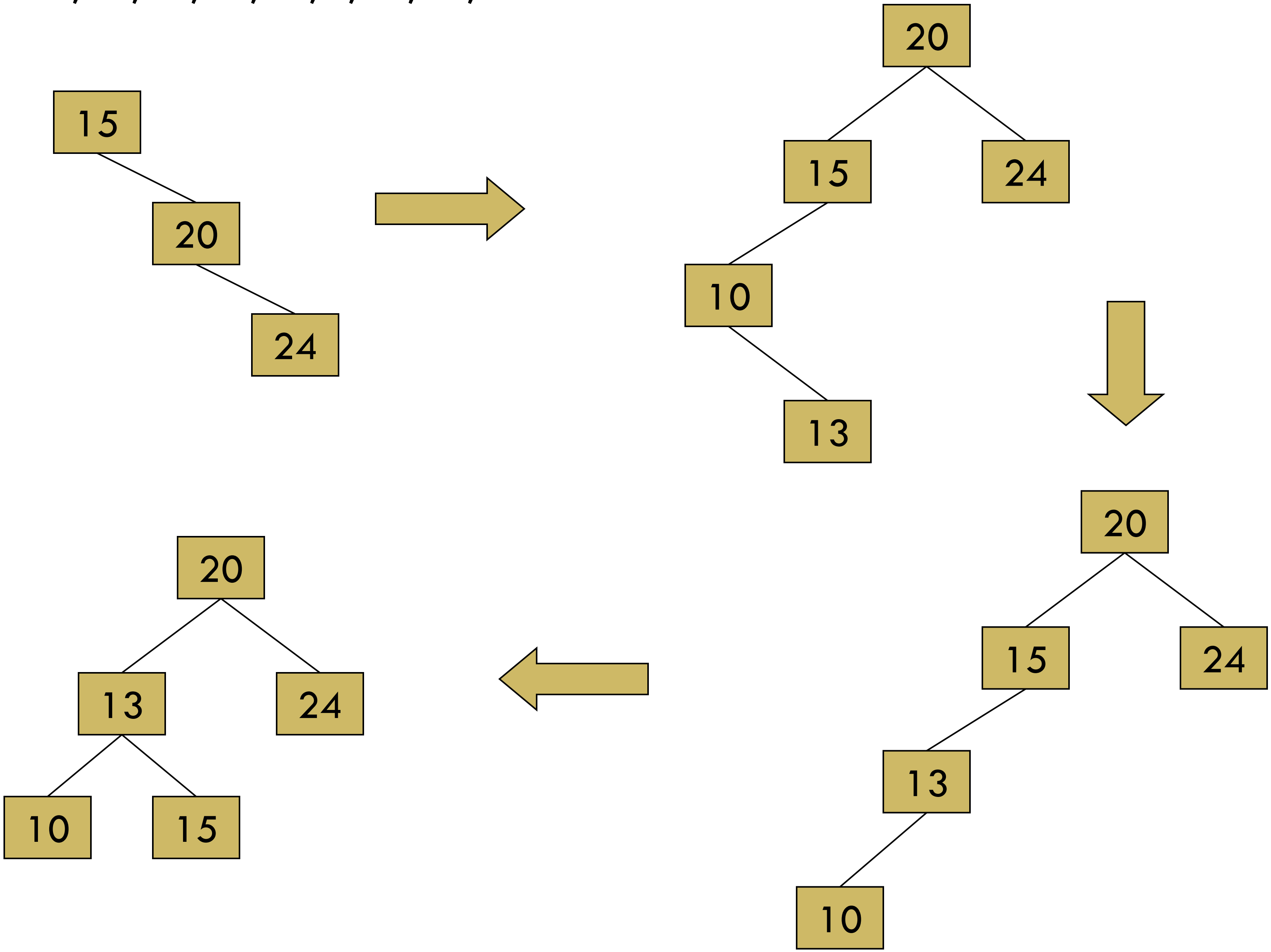
Tree is balanced

In Class Exercise

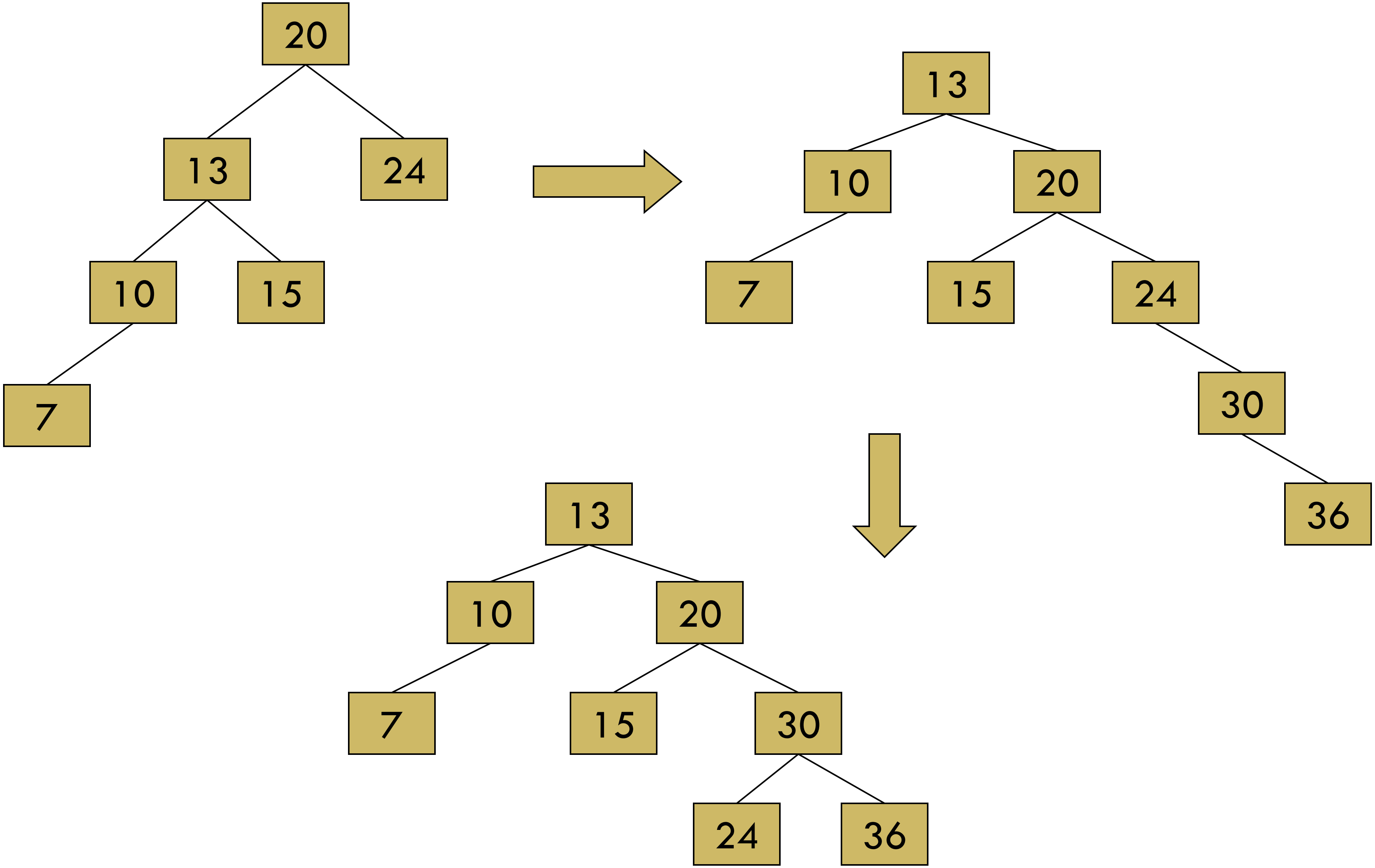
- Build an AVL tree with the following values:

  15, 20, 24, 10, 13, 7, 30, 36, 25

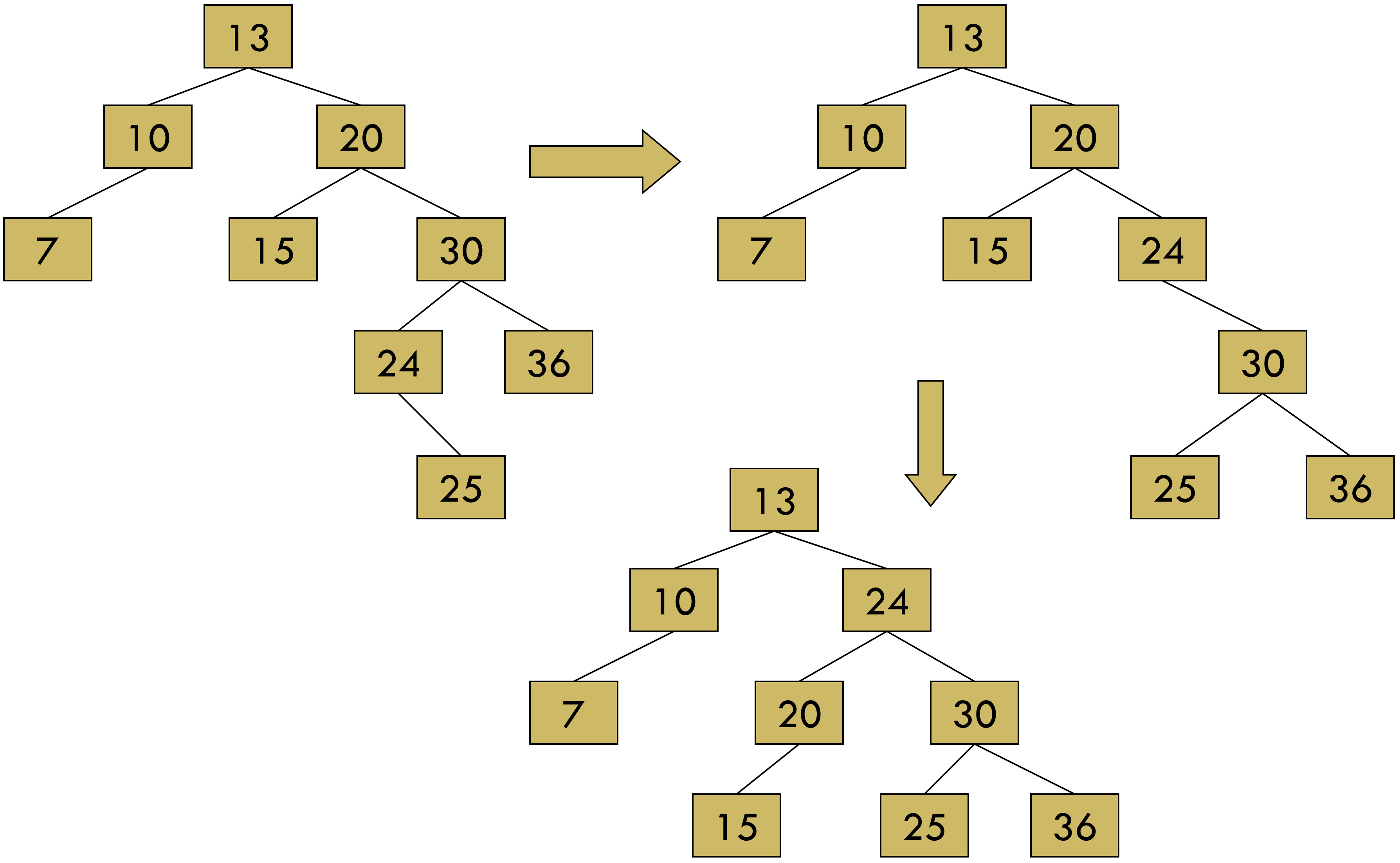
15, 20, 24, 10, 13, 7, 30, 36, 25



15, 20, 24, 10, 13, 7, 30, 36, 25



15, 20, 24, 10, 13, 7, 30, 36, 25



Reading Materials

- Schaum’s Outlines: Chapter # 7
- D. S. Malik: Chapter # 11
- Nell Dale: Chapter # 8
- Allen Weiss: Chapter # 4
- Tenebaum: Chapter # 5