# CS 2124: DATA STRUCTURES
# Spring 2024

**Topics:** AVL Trees and Segment Trees

# Topics

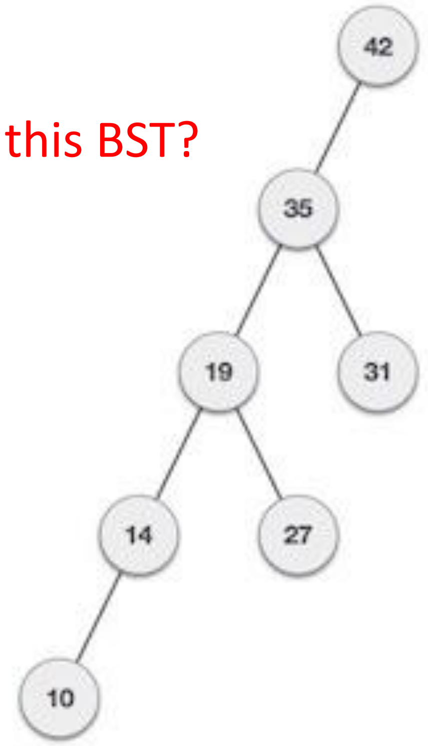
- AVL (Adelson, Velski & Landis) Trees
  - AVL Tree (Height vs Balance)
  - AVL Tree (Balance)
  - Rotations
    - Left rotation
    - Right rotation
    - Left-Right rotation
    - Right-Left rotation
  - AVL Tree (Insertion)
  - AVL Tree (Deletion)
- Segment Trees
  - Segment Trees (Array to Tree)
  - Segment Trees (Tree to Array)
  - Segment Trees (Applications)

# AVL (Adelson, Velski & Landis) Tree

- What if the input to binary search tree (BST) comes in a sorted (ascending or descending) manner?

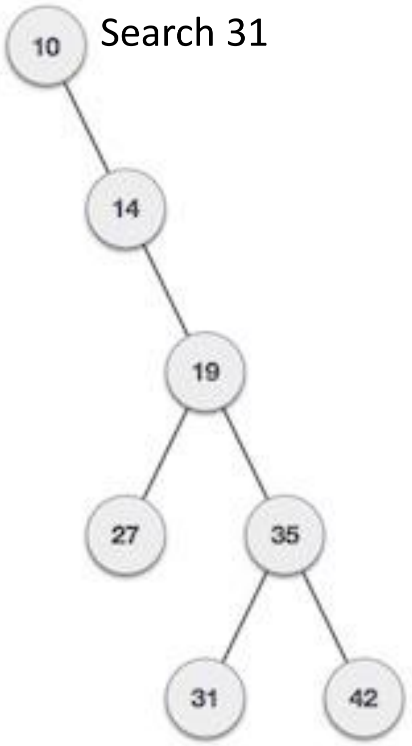
What is the problem with this BST?

- It is observed that BST's worst-case performance is closest to **linear search algorithms**, that is O(n).
- In real-time data, we cannot predict data pattern and their frequencies.
- So, a need arises to balance out the existing BST.




Search 10

If input 'appears' non-increasing manner




Search 31

If input 'appears' in non-decreasing manner

# AVL Tree

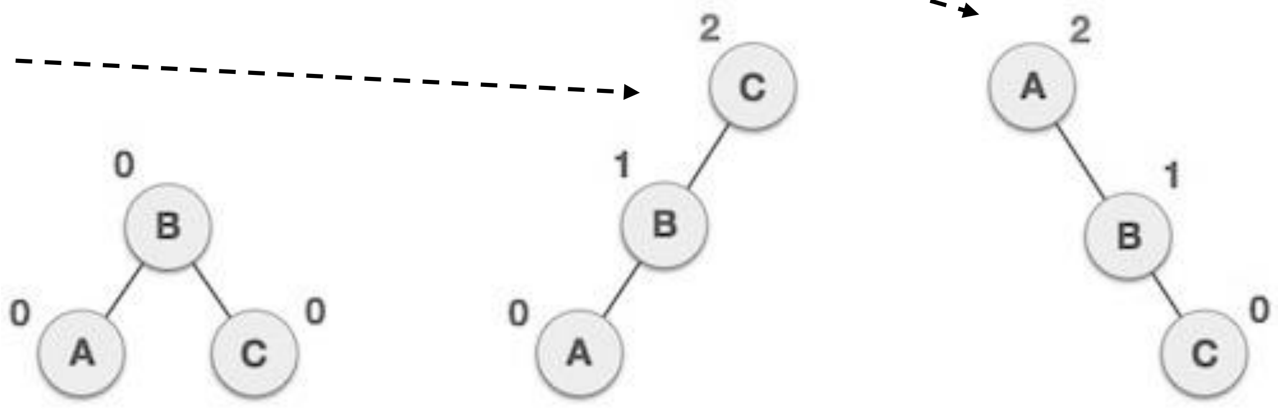
- AVL are height balancing binary search tree (BST). AVL trees have the property of dynamic self-balancing in addition to all the other properties exhibited by BST.
- AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1.
- This difference is called the Balance Factor.

In the third tree, the right subtree of A has height 2 and the left is missing, so it is 0, and the difference is 2 again.

In the second tree, the left subtree of C has height 2 and the right subtree has height 0, so the difference is 2.

AVL tree permits difference (balance factor) to be only 1.

***Balance-Factor** = height(left-subtree) − height(right-subtree)*




1.Balanced 2.Not balanced 3.Not balanced

# AVL Tree (Height vs Balance)

**Height (H):**

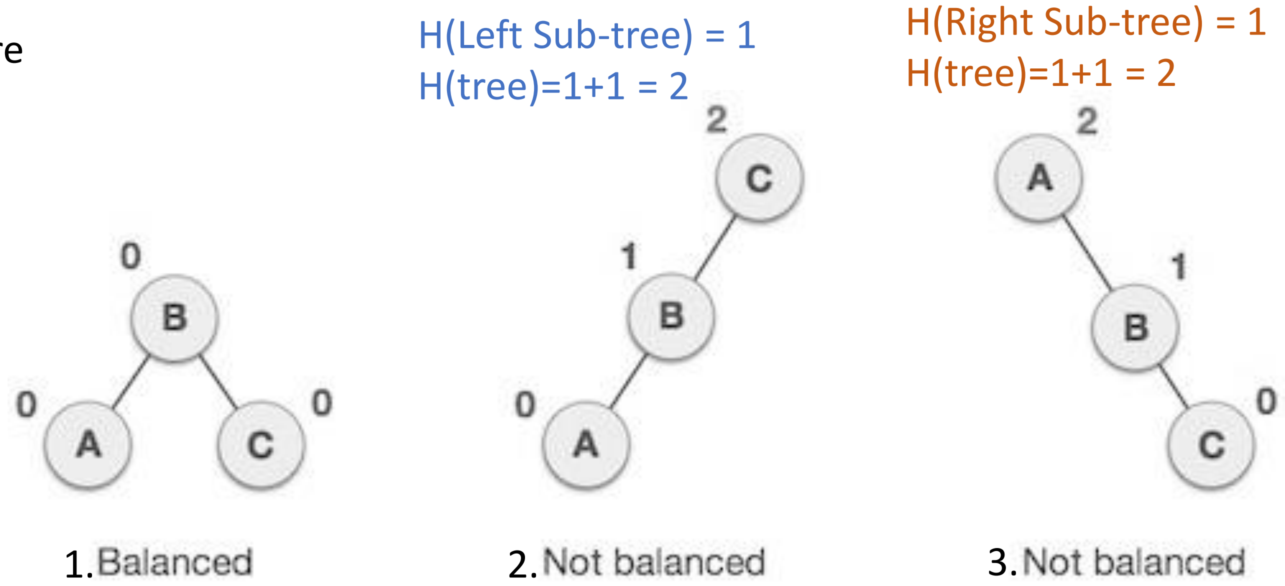
- H(null) = -1 (no nodes)
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

**Balance (B):**

- B(node) = H(Left Sub-Tree) - H(Right Sub-Tree)
  - If the difference in the height of left and right sub-trees is more than 1, the tree is balanced using some rotation techniques.

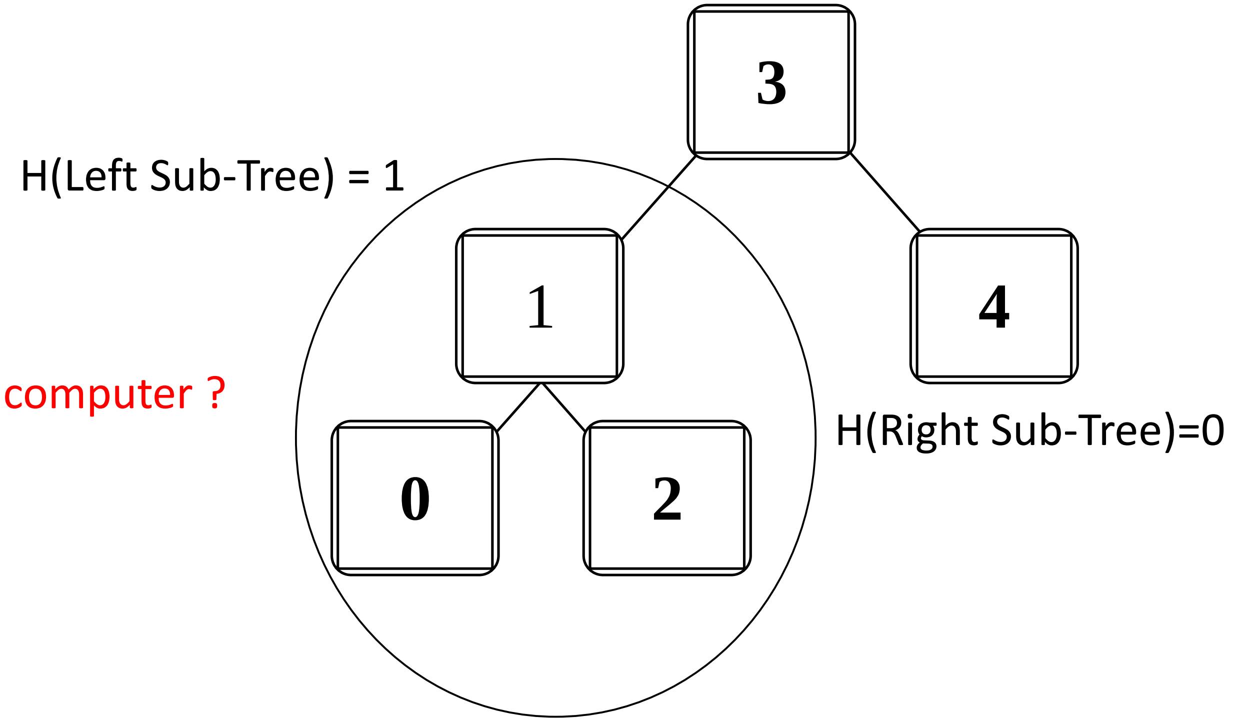
**AVL Tree** =$|B(node)| \leq 1$

- Maintain a threshold of 1 or less then 1
- This threshold is a parameterized, but standard is 1



# AVL Tree (Balance)

- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1




How are these computer ?

Height:

- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

# AVL Tree (Balance)

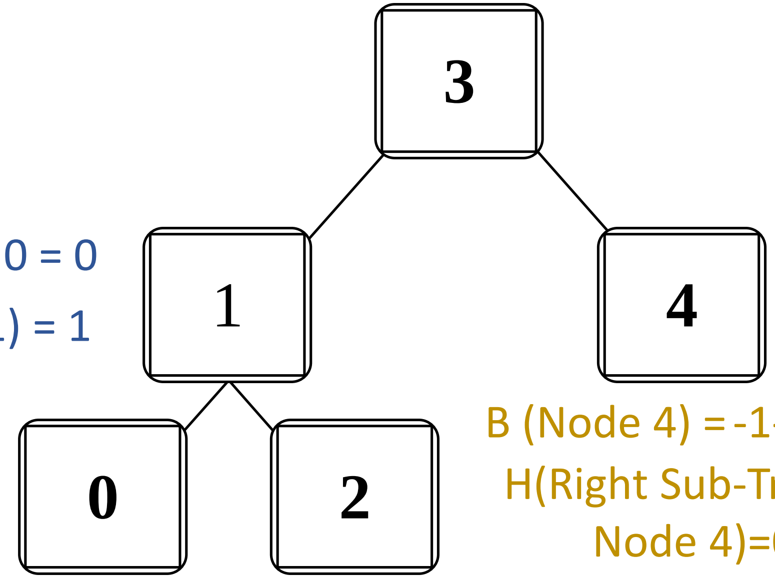
1. B (Node) = H(Left Sub-Tree) - H(Right Sub-Tree)

2. AVL Tree = $|B(node)| \leq 1$

3. H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

B (Node 3) = 1 – 0

B (Node 3) = 1




B (Node 1) = 0 - 0 = 0

H(Left Sub-Tree or Node 1) = 1

B (Node 4) = -1-(-1) = 0

H(Right Sub-Tree or Node 4)=0

B (Nodes 0, 2) = 0

B (Nodes 0) = -1 - (-1) = 0

B (Nodes 2) = -1 - (-1) = 0

Apparently this tree is Left Heavy

- Left Heavy = Positive Balance = + ve value
- Right Heavy = Negative Balance = - ve value

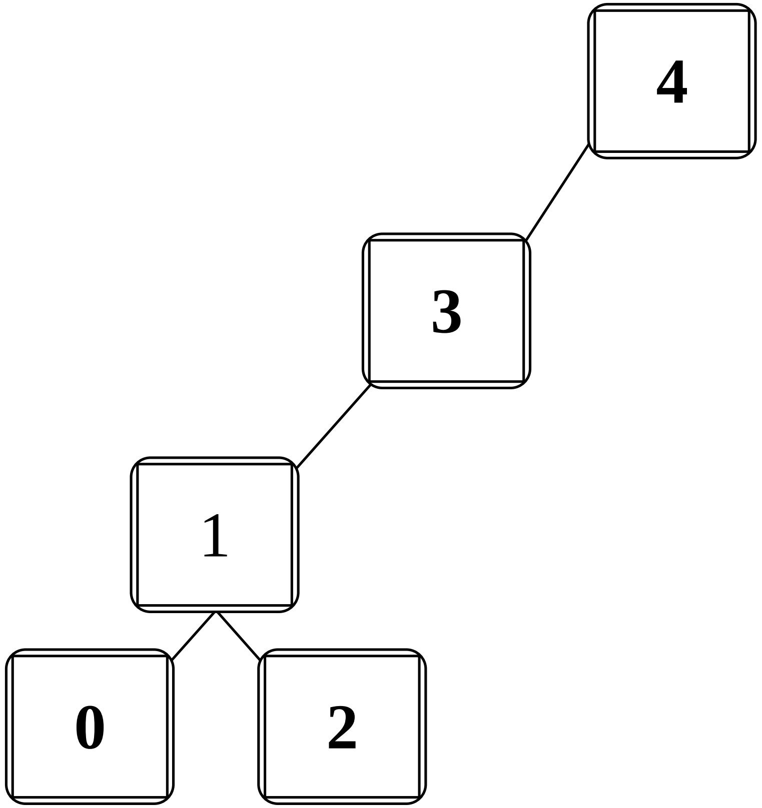
Height:

- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

# AVL Tree (Balance)

- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

B (Node 4) = 3




- Not an AVL Tree
- We will use **balancing** to make it an AVL tree

Height:

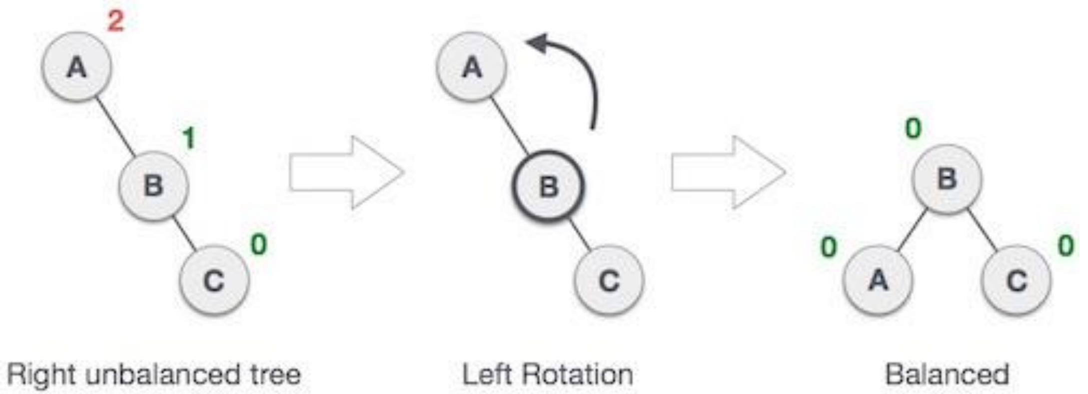
- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

# AVL Rotations

- To balance itself, an AVL tree may perform the following four kinds of rotations:
  1. Left rotation
  2. Right rotation
  3. Left-Right rotation
  4. Right-Left rotation
- The first two rotations are single rotations and the next two rotations are double rotations.

**1. Left Rotation:**

If a tree becomes unbalanced, when a node is inserted into the right of the right subtree, then we perform a single left rotation

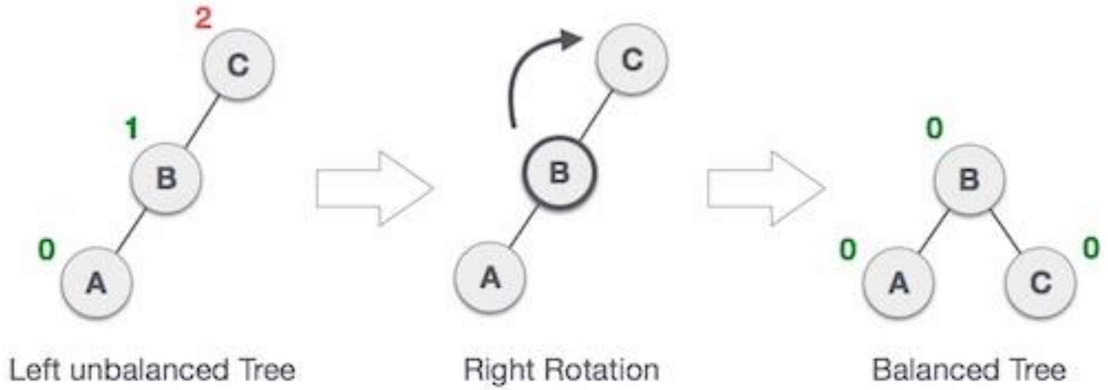


# AVL Rotations

- To balance itself, an AVL tree may perform the following four kinds of rotations:
  1. Left rotation
  2. Right rotation
  3. Left-Right rotation
  4. Right-Left rotation

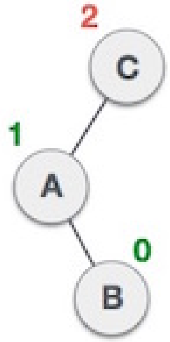
**2. Right Rotation**

- AVL tree may become unbalanced, if a node is inserted in the left of the left subtree. The tree then needs a right rotation.

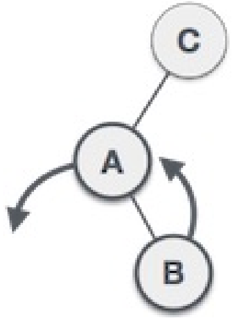


# AVL Rotations

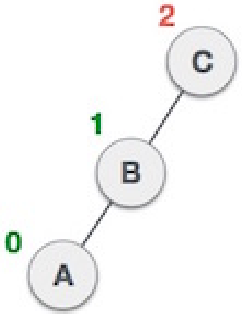
**3. Left-Right Rotation**: A left-right rotation is a combination of left rotation followed by right rotation.




a) A node has been inserted into the right of the left subtree. This makes C an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.

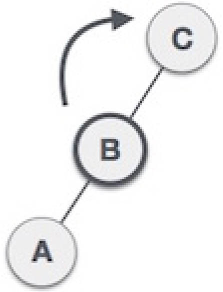



b) We first perform the left rotation on the left subtree of C. This makes A, the left subtree of B.

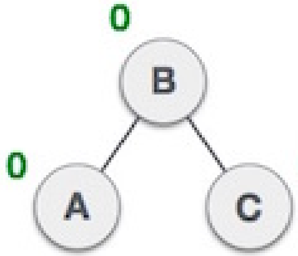



c) Node C is still unbalanced, however now, it is because of the left of the left-subtree.

d) We shall now right-rotate the tree, making B the new root node of this subtree. C now becomes the right subtree of its own left subtree.



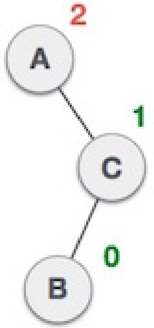

e) The tree is now balanced.



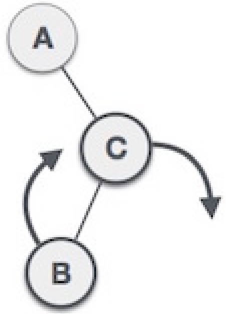
# AVL Rotations

**4. Right-Left Rotation**: It is a combination of right rotation followed by left rotation.

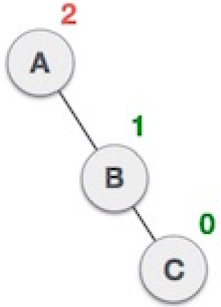
a) A node has been inserted into the left subtree of the right subtree. This makes A, an unbalanced node with balance factor 2.



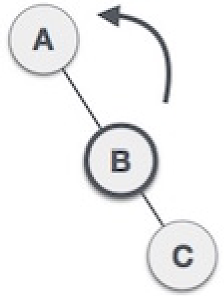

b) First, we perform the right rotation along C node, making C the right subtree of its own left subtree B. Now, B becomes the right subtree of A.



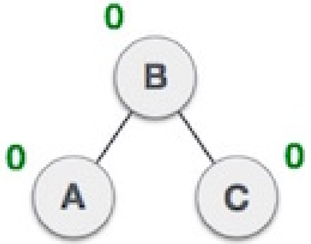

c) Node A is still unbalanced because of the right subtree of its right subtree and requires a left rotation.




d) A left rotation is performed by making B the new root node of the subtree. A becomes the left subtree of its right subtree B.

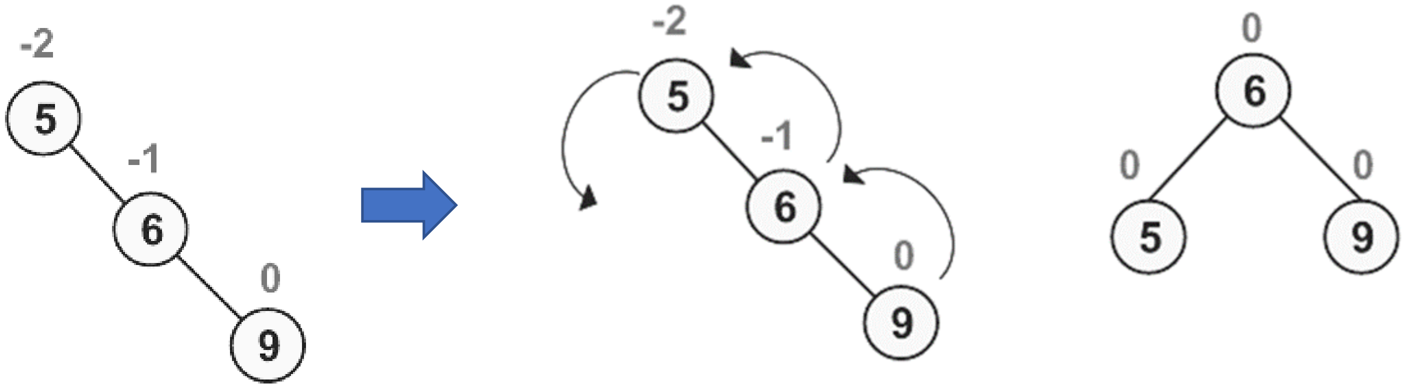



e) The tree is now balanced.



# AVL Rotations

- Left Heavy = Positive Balance = + ve value = right rotation
- Right Heavy = Negative Balance = - ve value = left rotation

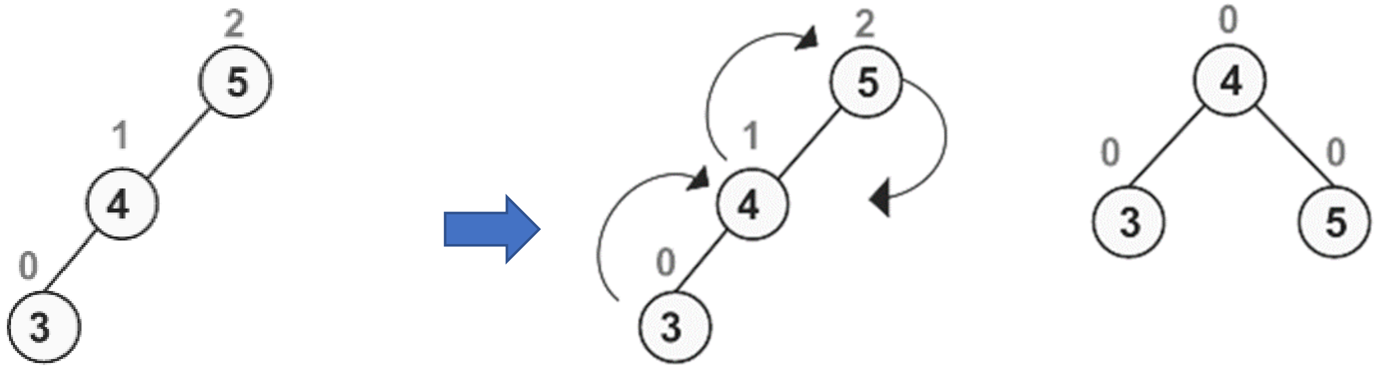



**The tree is not balanced** | **Moving one position left to balanced the tree** | **Tree is balanced after the left rotation**

- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

Height:

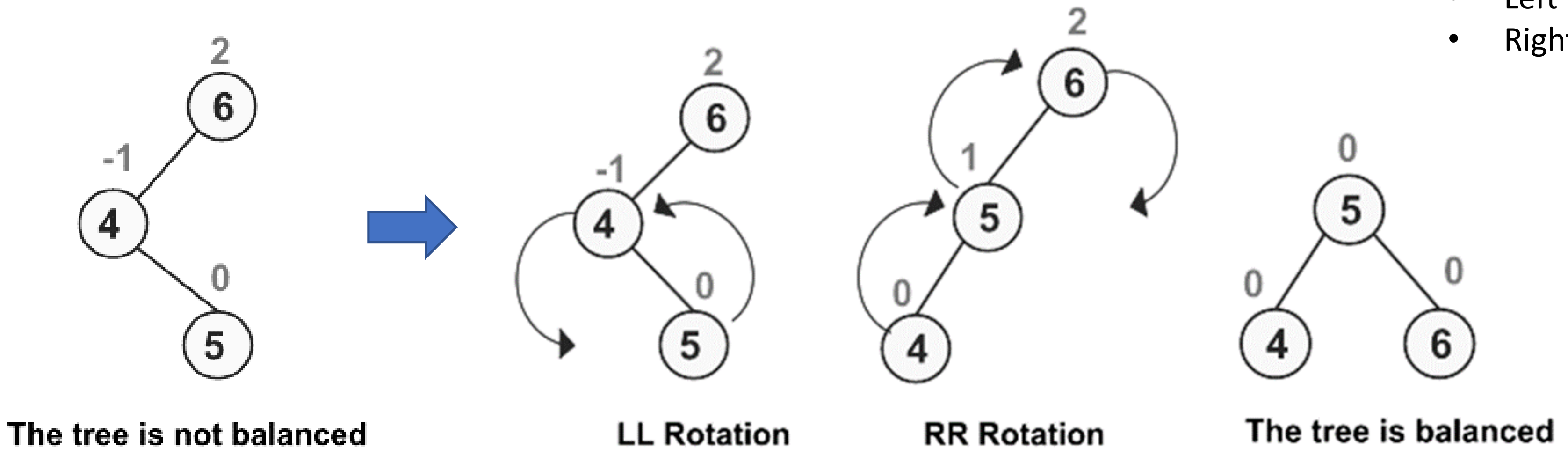
- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

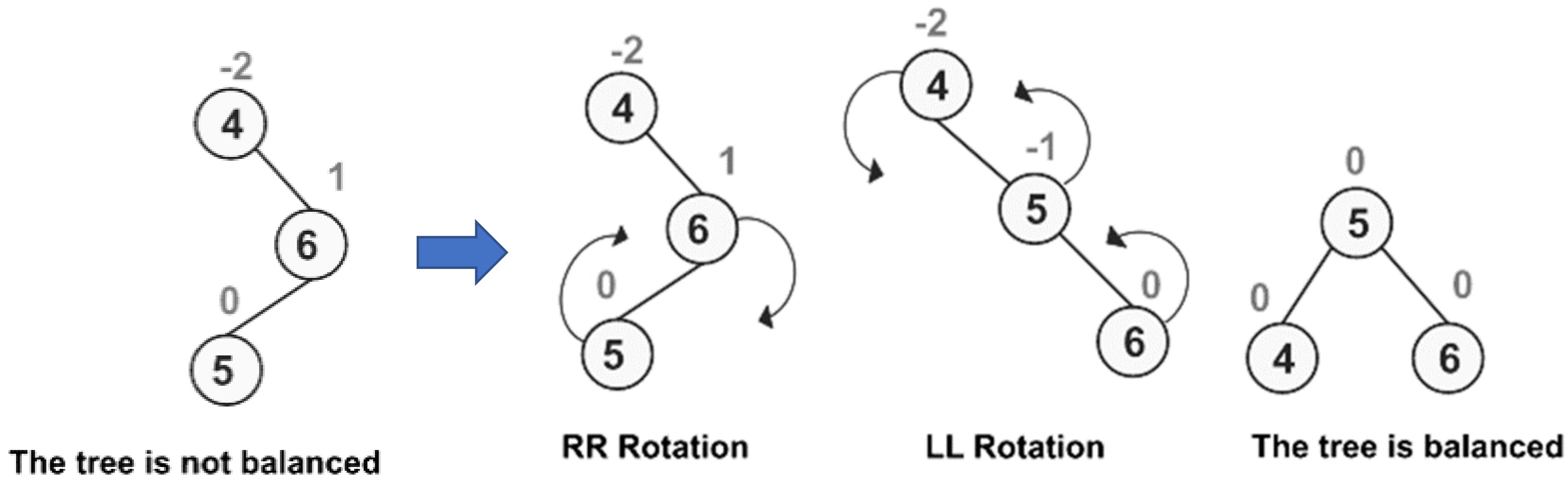



**The tree is not balanced** | **Moving one position right to balance the tree** | **Tree is balanced after the right rotation**

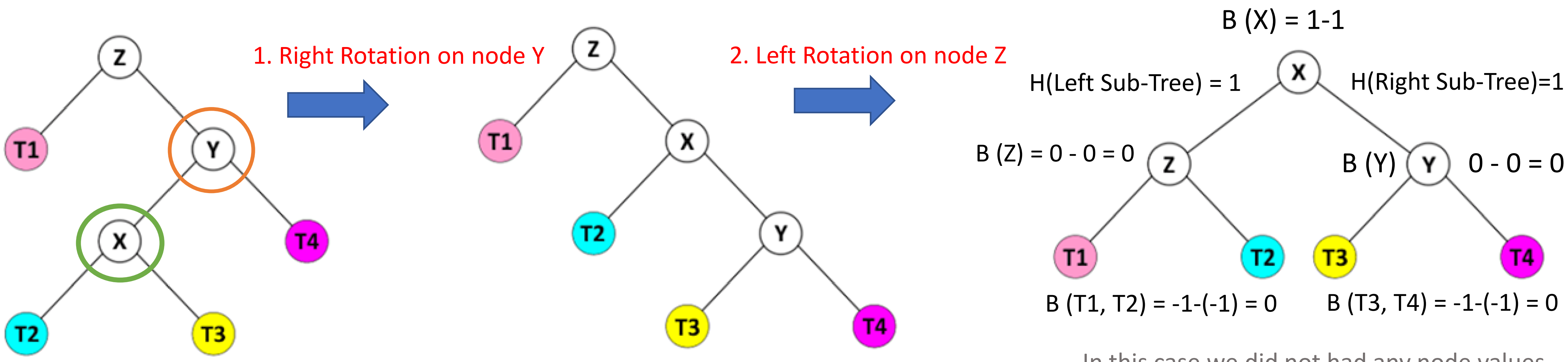
# AVL Rotations

- Left Heavy = Positive Balance = + ve value = right rotation
- Right Heavy = Negative Balance = - ve value = left rotation



# AVL Rotations




In this case we did not had any node values
In case of node values we need to maintain
BST property

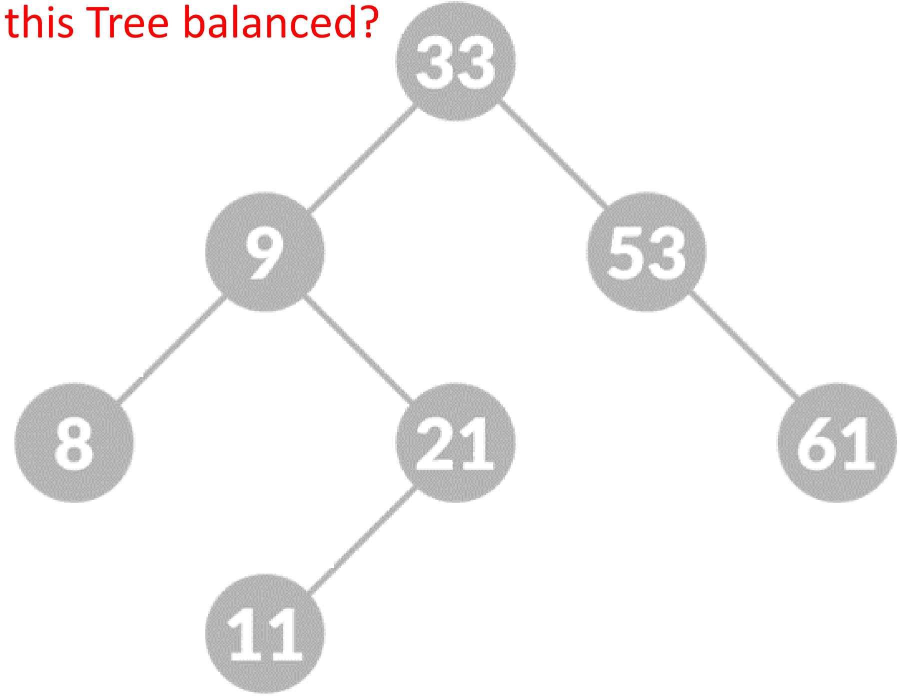
- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree = $|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

Height (H):

- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

# AVL Tree
## (Try to compute the balance by your self using formulas)

Is this Tree balanced?




- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

Height (H):
- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

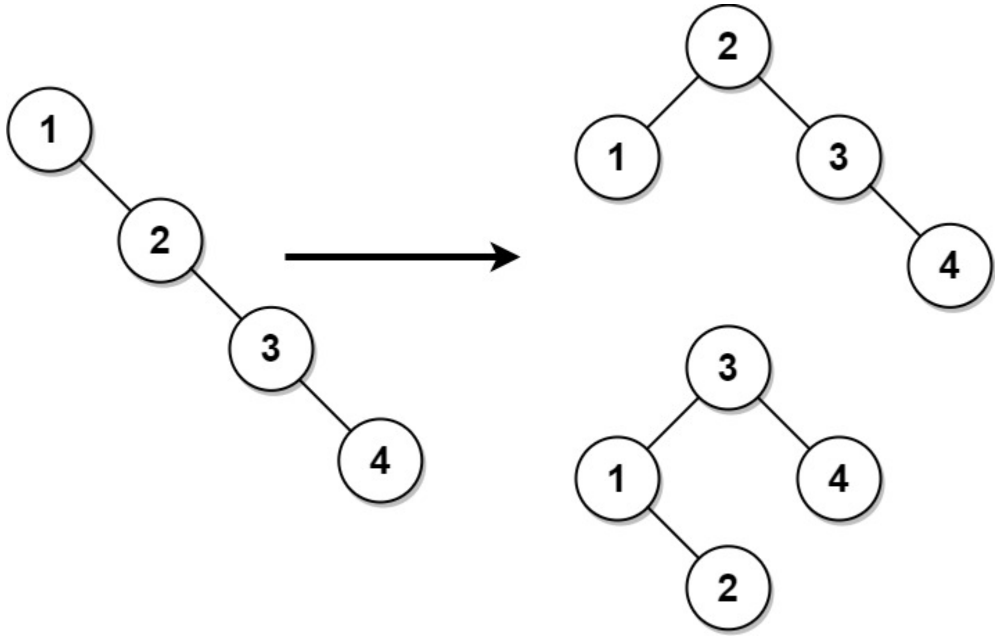
# AVL Rotations

- To balance itself, an AVL tree may perform the following four kinds of rotations:

- Right Heavy
  - Left rotation
  - Left-Right rotation

- Left Heavy
  - Right rotation
  - Right-Left rotation

Left Heavy = Positive Balance = + ve value
Right Heavy = Negative Balance = - ve value



# AVL Tree (Balance)

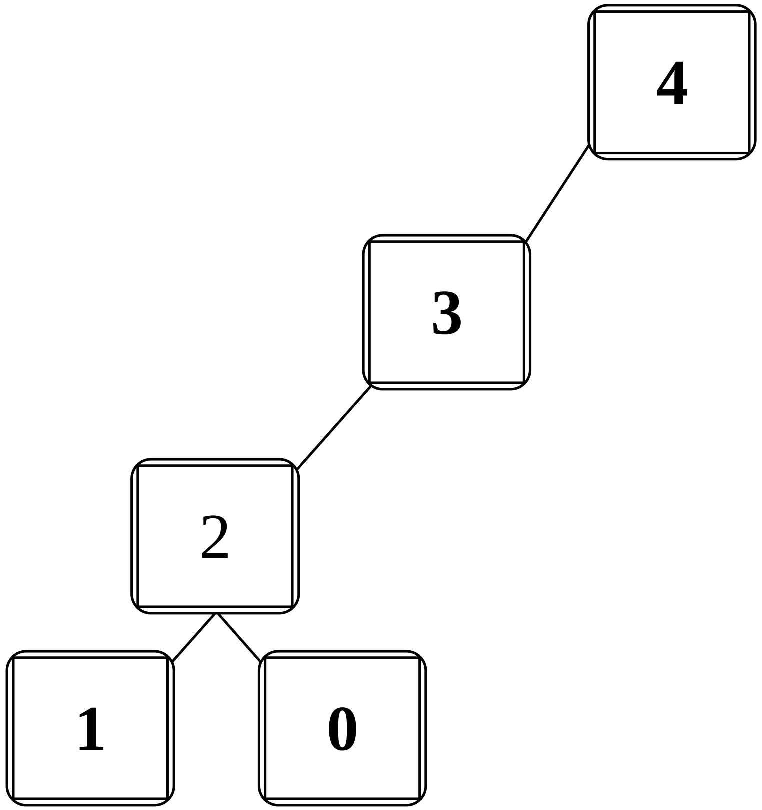
- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

- Not an AVL Tree
- We will use **balancing** to make it an AVL tree

- Left Heavy = Positive Balance = + ve value

Left Heavy
- Right rotation
- Right-Left rotation

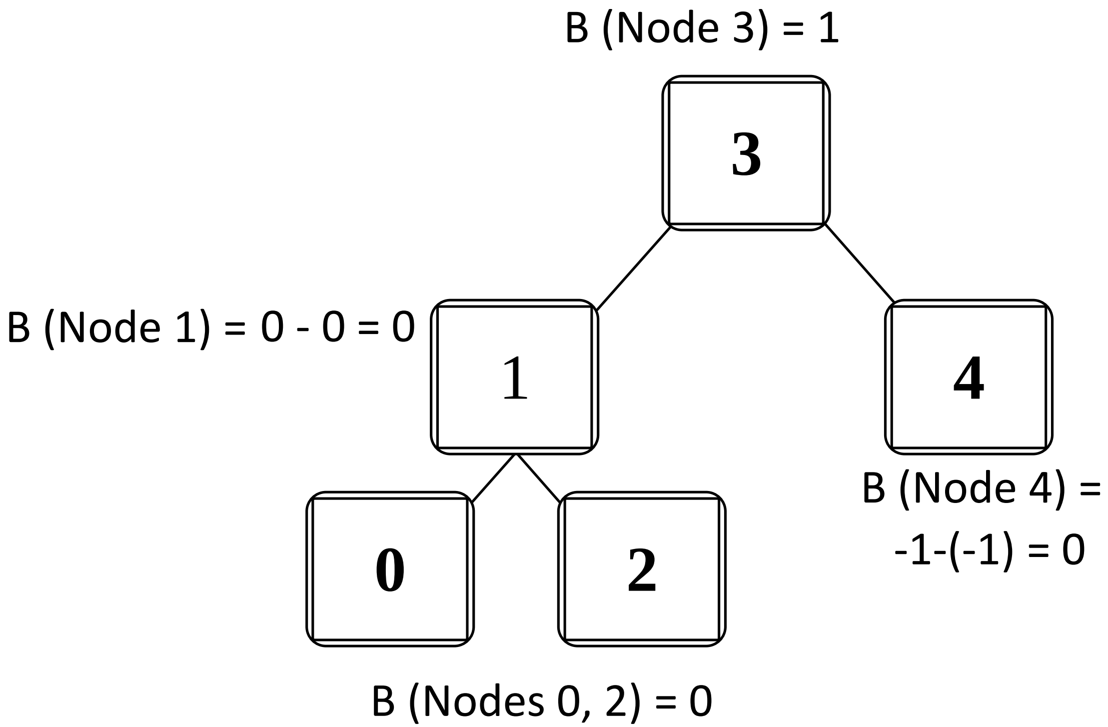
B (Node 4) = 3



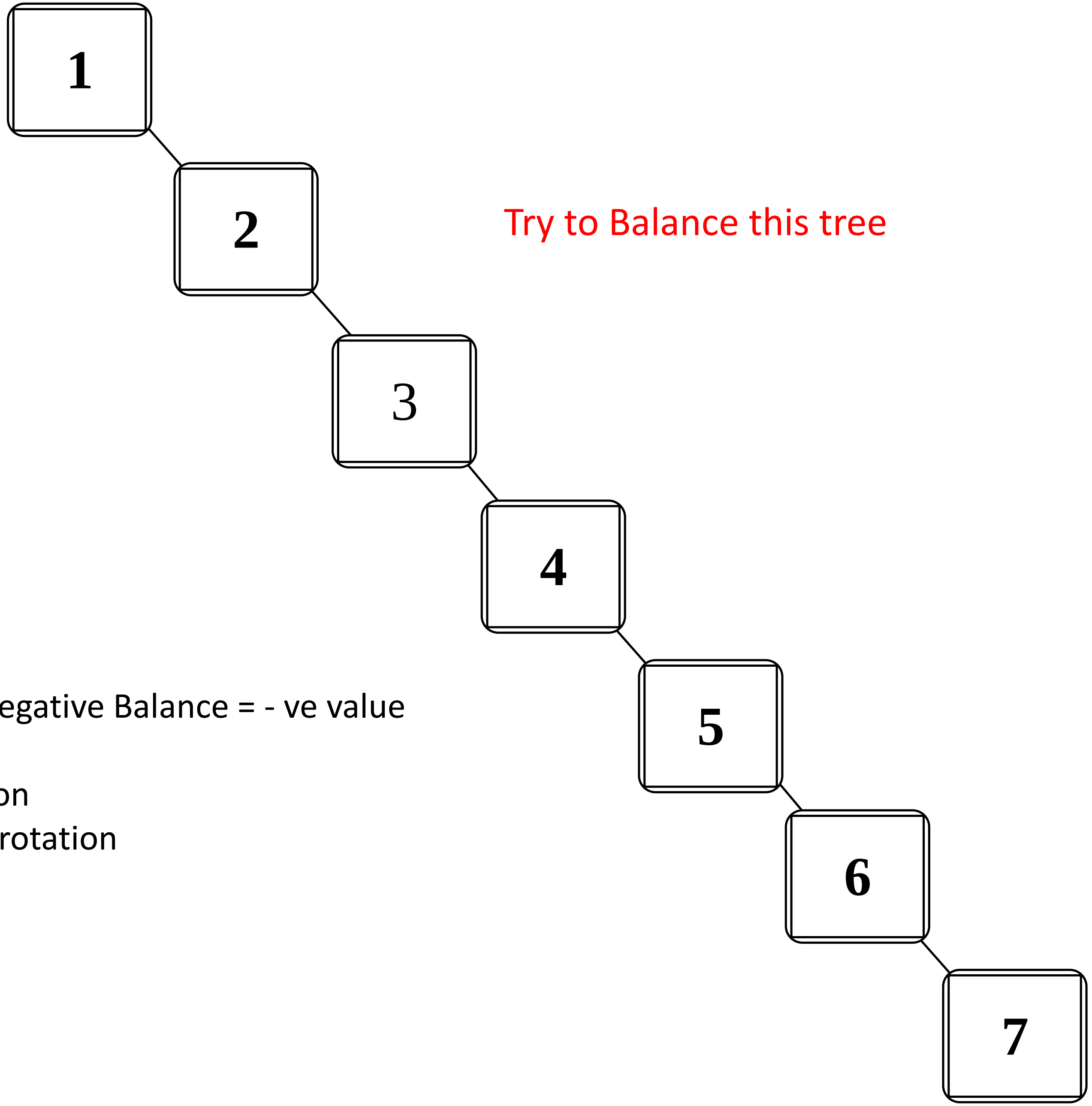
# AVL Tree (Balance)

- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1
- Not an AVL Tree
- We will use **balancing** to make it an AVL tree
- Left Heavy = Positive Balance = + ve value
- Left Heavy
  - Right rotation
  - Right-Left rotation

Height (H):
- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1



# AVL Tree (Balance)




Try to Balance this tree

- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree $=|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

- Not an AVL Tree
- We will use **balancing** to make it an AVL tree

- Left Heavy = Positive Balance = + ve value
- Left Heavy
    - Right rotation
    - Right-Left rotation

- Right Heavy = Negative Balance = - ve value
- Right Heavy
    - Left rotation
    - Left-Right rotation

Height:

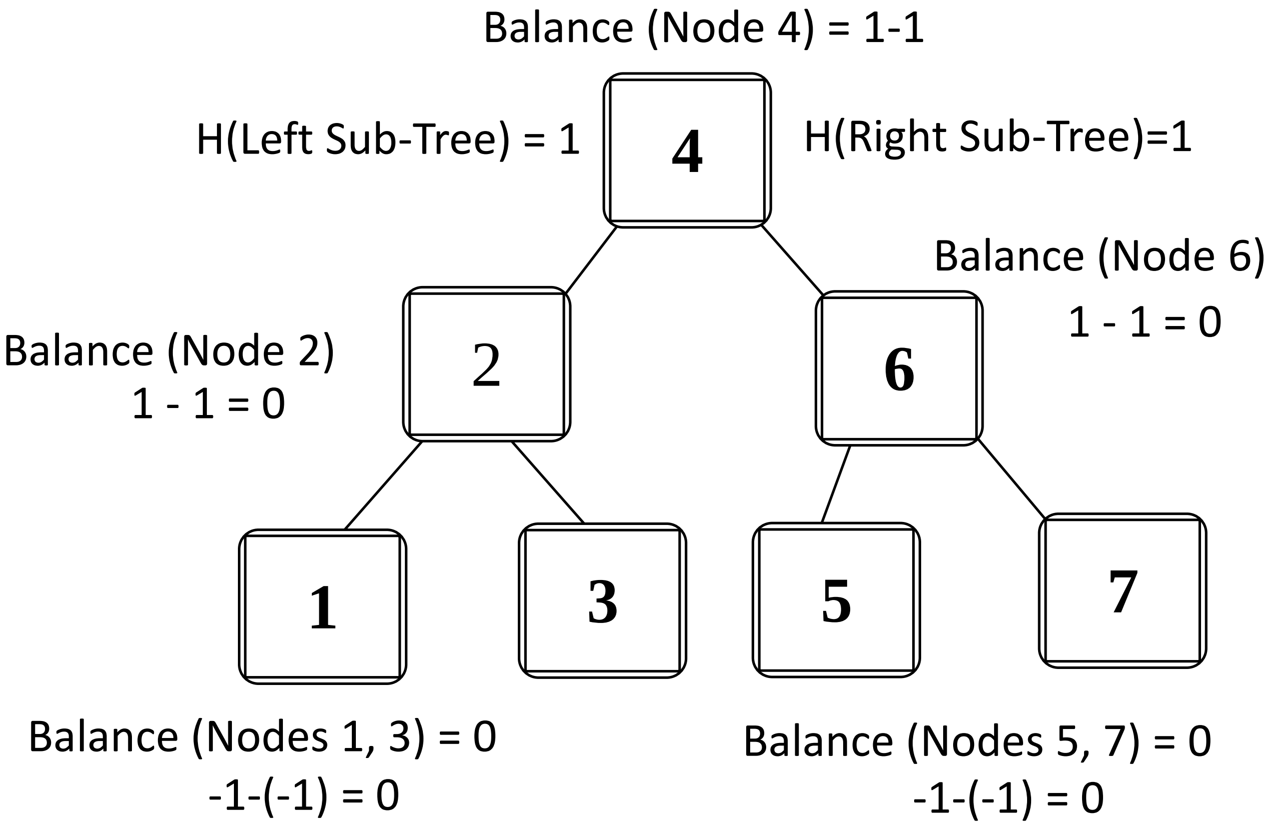
- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

# AVL Tree (Balance)

- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

- *To Balance:*

1. *First left rotation on node 4*
2. *Then right rotations on nodes 2 & 6*




Height:

- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1