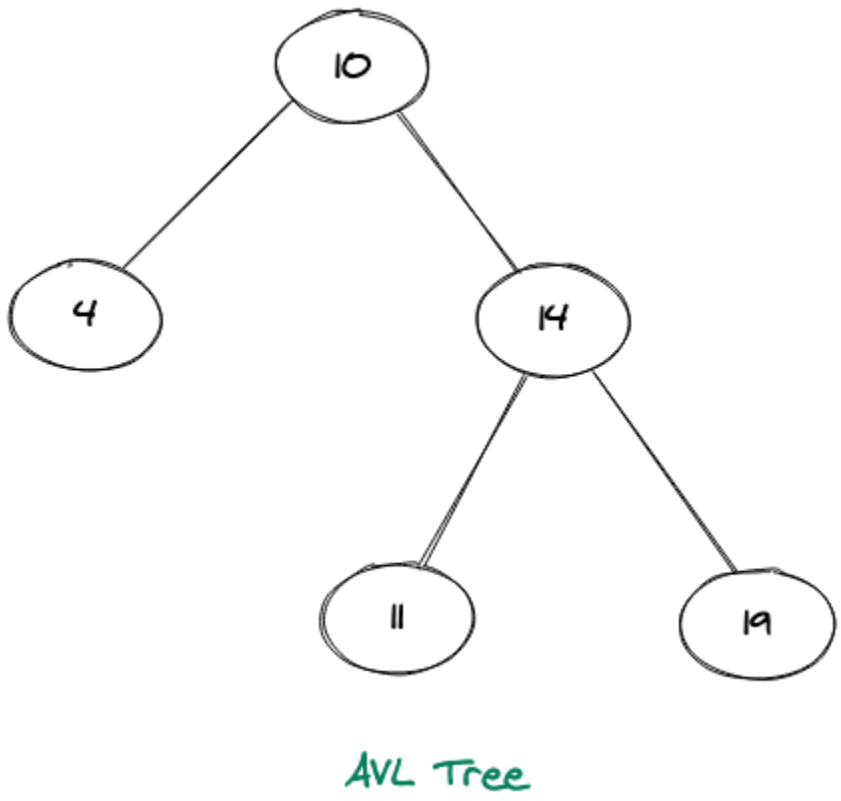
AVL Tree Data Structure

An AVL tree is another special tree that has several properties that makes it special. These are:

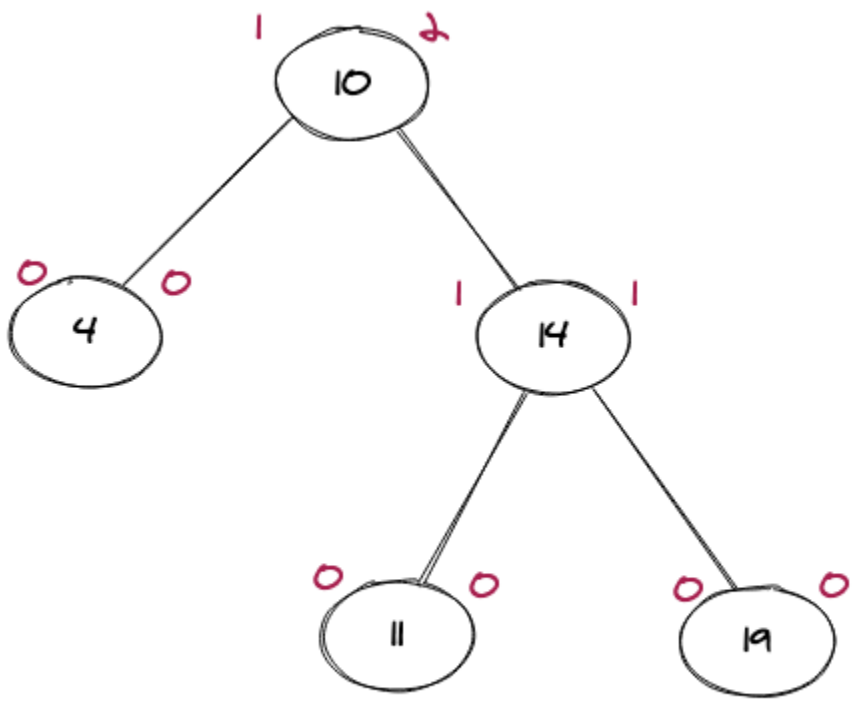
- It is a BST that is balanced in nature.
- It is self-balanced.
- It is not perfectly balanced.
- Every sub-tree is also an AVL Tree.

A pictorial representation of an AVL Tree is shown below:



AVL Tree

When explained, we can say that it is a Binary Search Tree that is height-balanced in nature. **Height balance** means that the difference between the left subtree height and the right subtree height for each node cannot be greater than 1(i.e **height(left) - height(right) <= 1**).



AVL Tree with Balance Factor

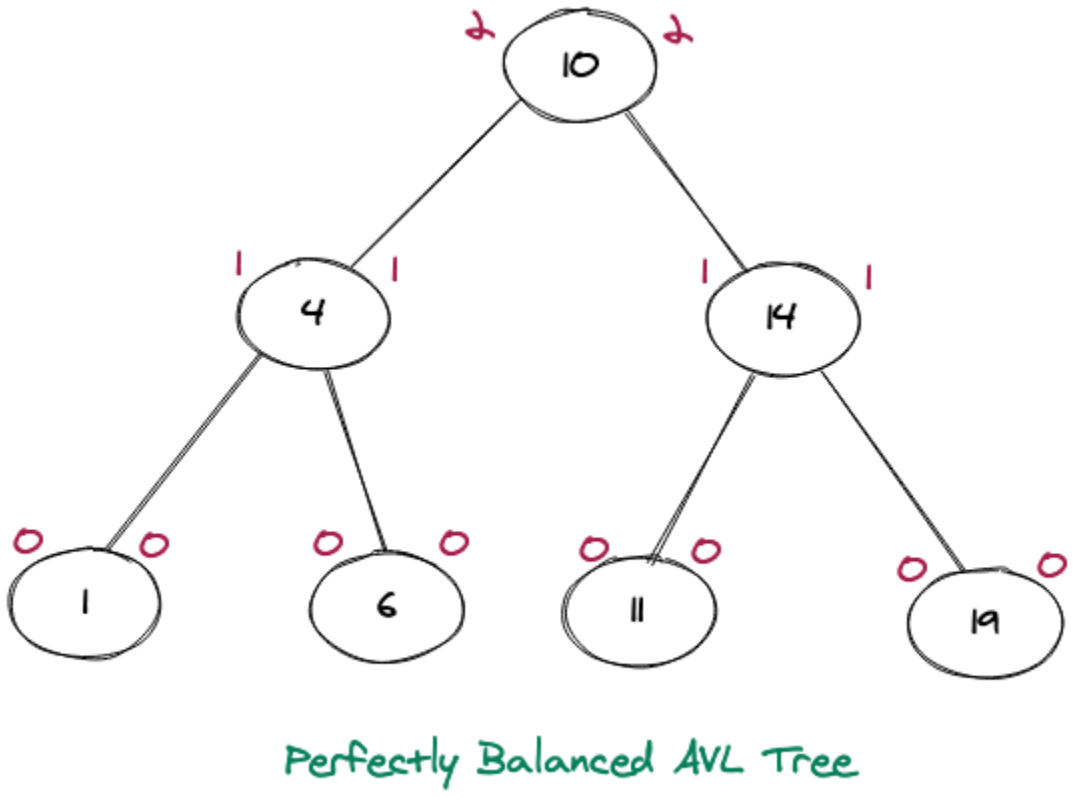
Notice that each node in the above BST contains two values, the left value denotes the height of the left subtree of that node and the right value denotes the height of the right subtree of that node, and it can be seen that at not a single node we have a value difference that is greater than 1, hence making it a balanced AVL Tree.

A **self-balanced tree** means that when we try to insert any element in this BST and if it violates the balancing factor of any node, then it dynamically rotates itself accordingly to make itself self-

balanced. It is also well known that AVL trees were the first well known dynamically balanced trees that were proposed.

It is not perfectly balanced, where the perfect balance implies the fact that the **height between the right subtree and the left subtree** of each node is equal to **0**.

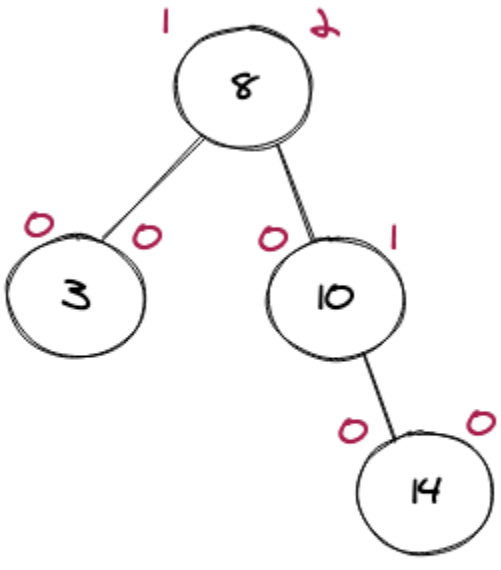
A pictorial representation of a Perfectly Balanced AVL Tree is shown below:



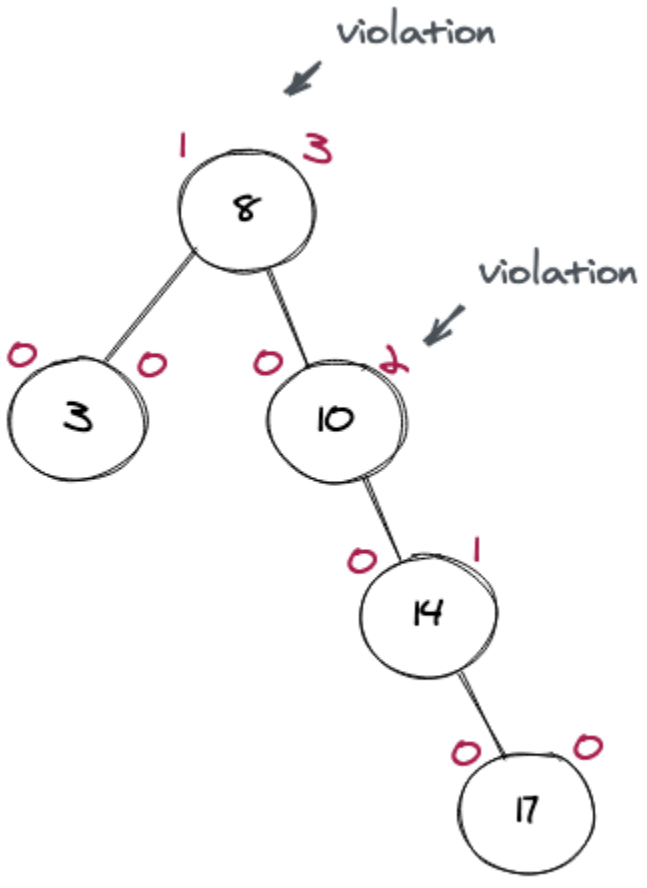
Perfectly Balanced AVL Tree

All the nodes of the above AVL Tree have the balance factor equal to 0(**height(left) - height(right) = 0**) making it a perfect AVL tree. It should also be noted that a perfect BST is the same as a perfect AVL tree. If we take a closer look at all the pictorial representations above, we can clearly see that each subtree of the AVL tree is also an AVL tree.

Now let's take a look at two more pictorial representations, where one of them is an AVL tree and one isn't.



AVL Tree



NOT an AVL Tree

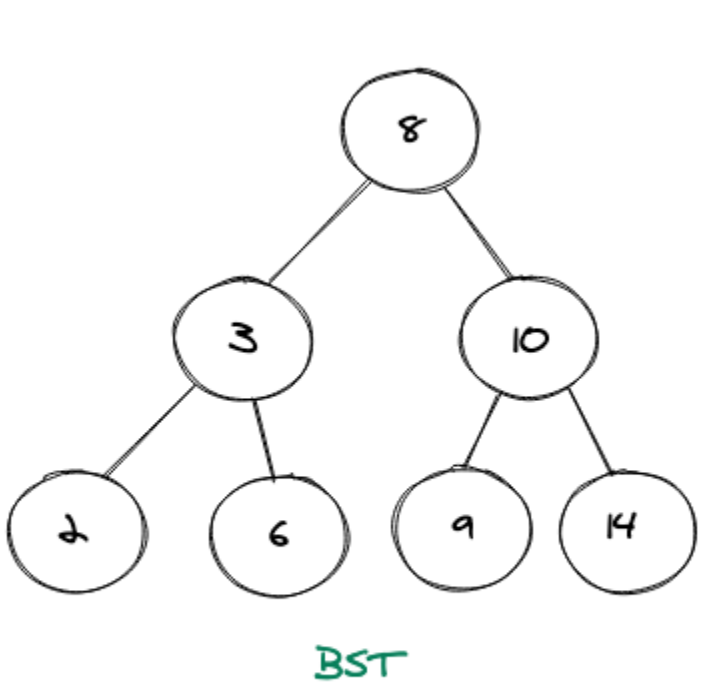
Though both the above trees are BST, only the left BST in the above representation is an AVL one as it height-balanced. In right BST, we have two violation nodes (i.e 8 and 10) where the balance

factor is more than 2 in both the cases. It should be noted that this tree can be rotated in a certain manner to make it an AVL tree.

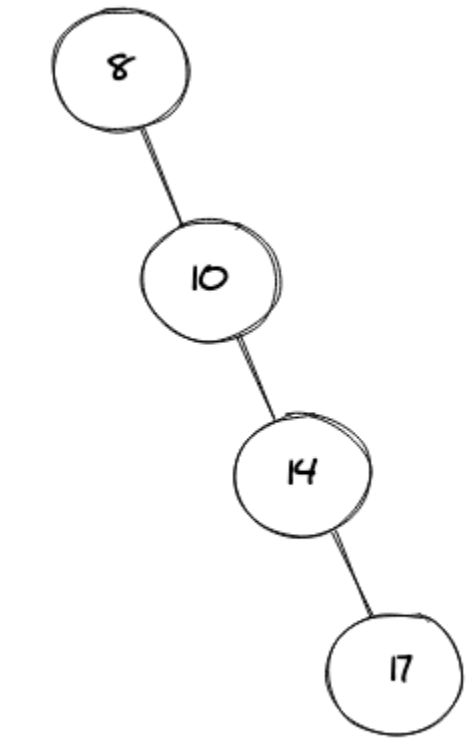
Why AVL Tree?

When we already had a similar data structure(i.e. BST's), why would we need a complex version of it?. The answer lies in the fact that BST has some limitations in certain scenarios that make some operations like searching, inserting costly (as costly as $O(n)$).

Consider the pictorial representation of two BST's below:



BST



Right Skewed BST

The height of the left BST is $O(\log n)$ and the height of the right BST is $O(n)$, thus the search operation time complexity for the left BST is $O(\log n)$ and for the right-skewed is $O(n)$ which is its worst case too. Hence, if we have such a skewed tree then there's no benefit of using a BST, as it is just like a linked list when it comes to time and space complexity.

That's why we needed a balanced BST's so that all the basic operations guarantee a time complexity of O(logN).

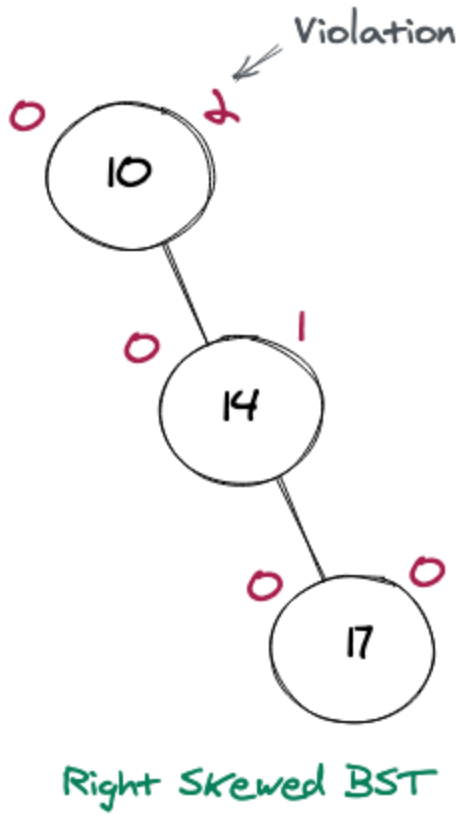
AVL Tree Rotations

If the AVL tree encounters any **violation of the balance factor** then it tries to do some rotations to make the tree balanced again. There are four rotations in total, we will look at each one of them. There mainly are:

- Left Rotation
- Right Rotation
- Left - Right Rotation
- Right - Left Rotation

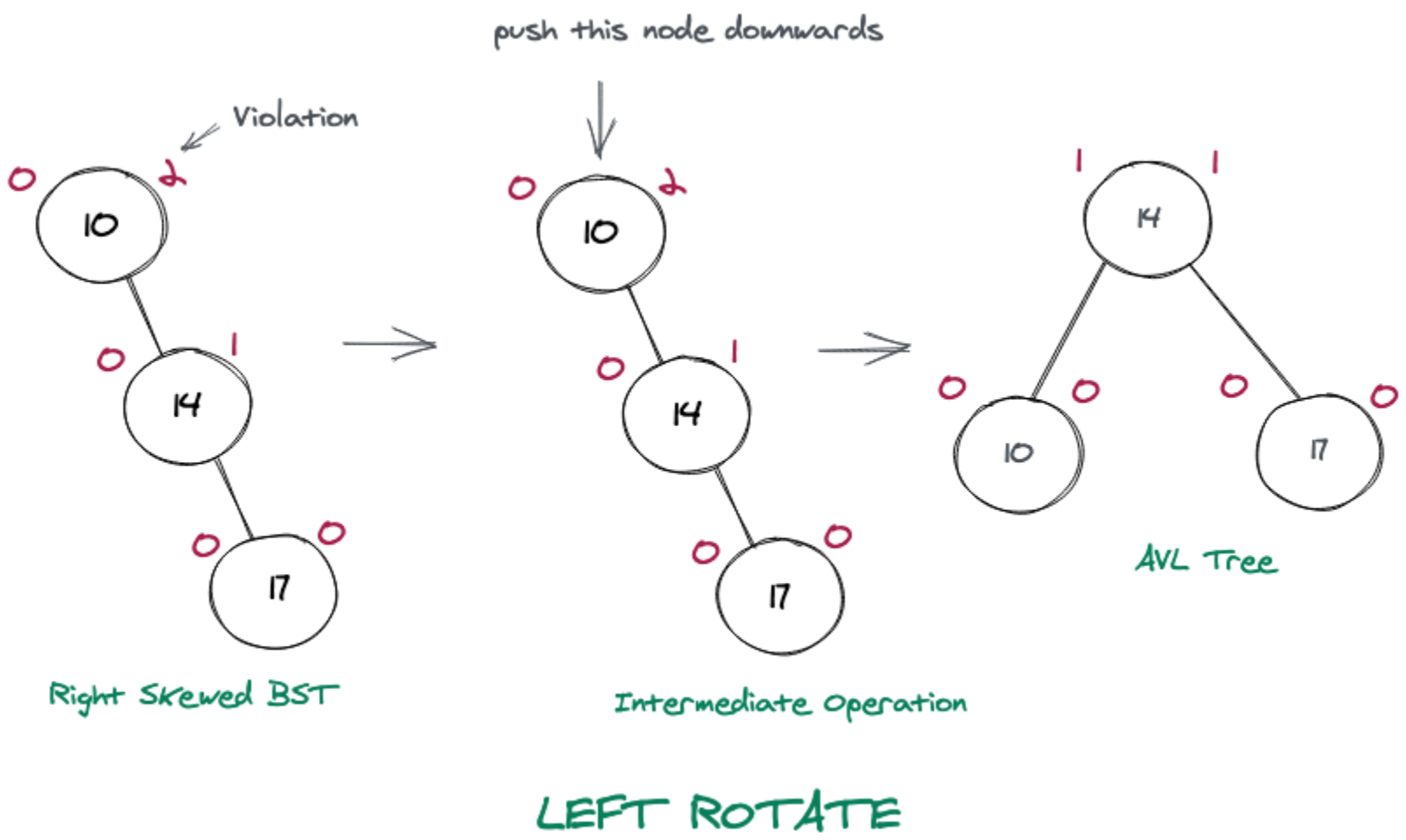
1. Left Rotation

This rotation is performed when the violation occurs in the left subtree of the left child of the violating node. Consider the pictorial representation of a BST below that is not balanced, in order to make it so, we will do some rotations.



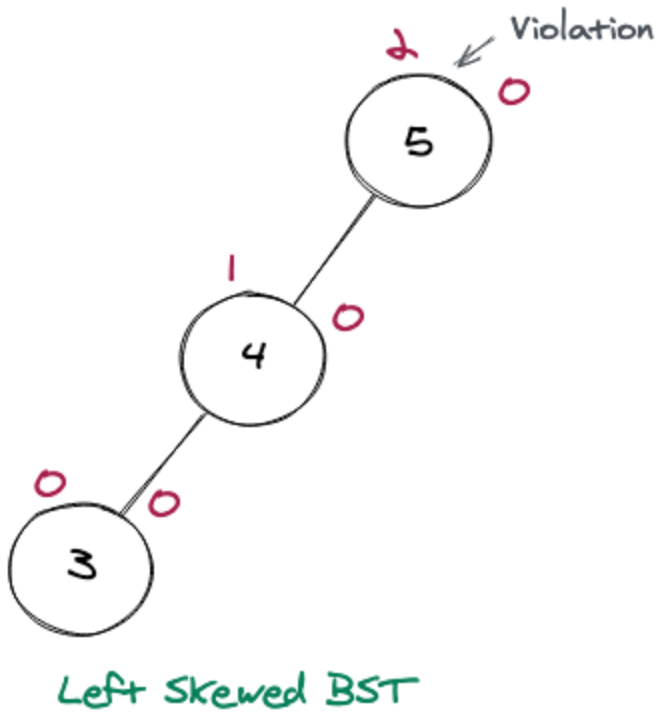
Clearly this is not an AVL tree, but consider the node where the violation occurs, we take this node as a pivot to do our rotations. The tree is heavy on the right and the node that causes the issue is 17 as if we remove that one, we would have a balanced BST (i.e. AVL). Since it is heavily skewed on the right, we will do a left-rotation to make it balanced.

Consider the pictorial representation of making the above right skewed tree into an AVL tree.



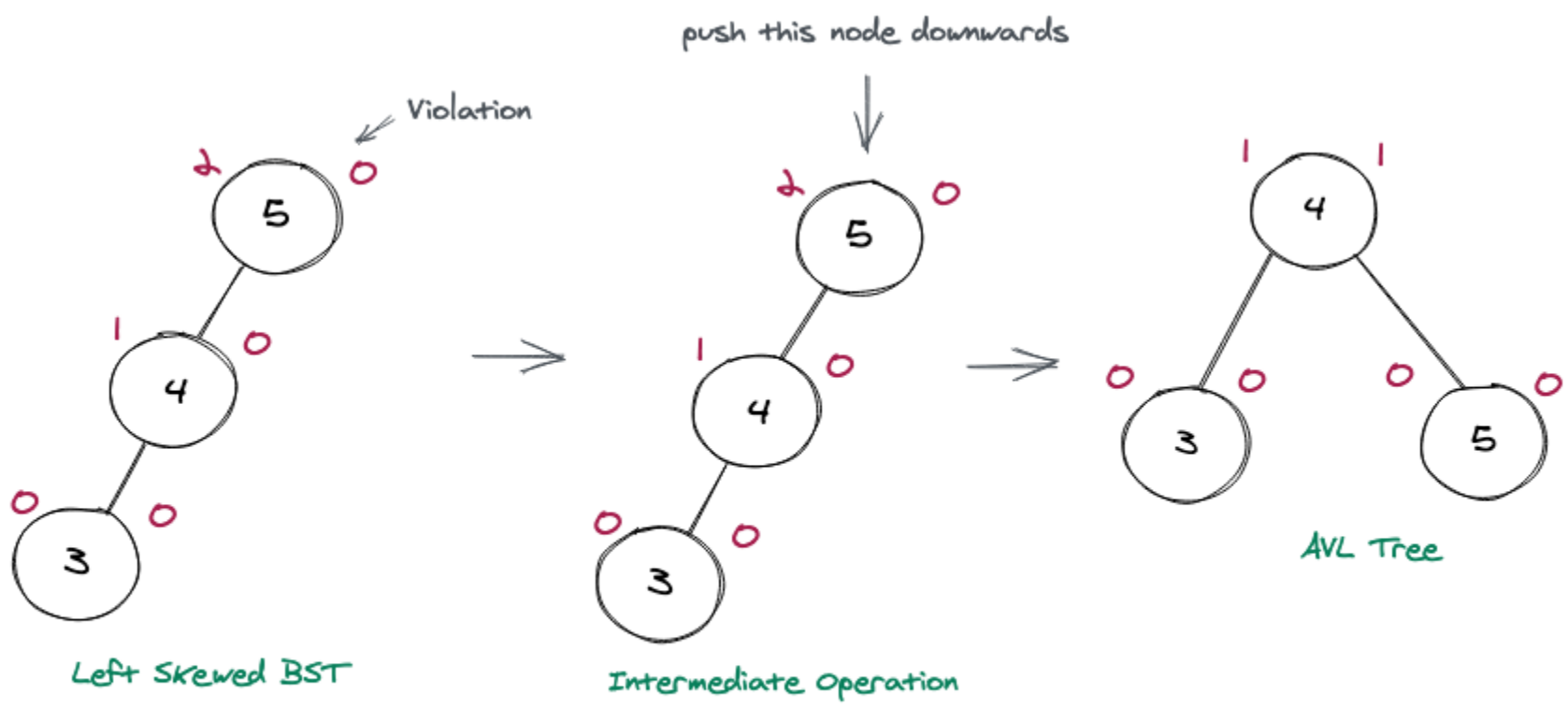
2. Right Rotation

This rotation is performed when the violation occurs in the right subtree of the right child of the violating node. Consider the pictorial representation of a BST below that is not balanced, in order to make it so, we will do some rotations.



Clearly this is not an AVL tree, but consider the node where the violation occurs, we take this node as a pivot to do our rotations. The tree is heavy on the left and the node that causes the issue is 3 as if we remove that one, we would have a balanced BST (i.e. AVL). Since it is heavily skewed on the left, we will do a right-rotation to make it balanced.

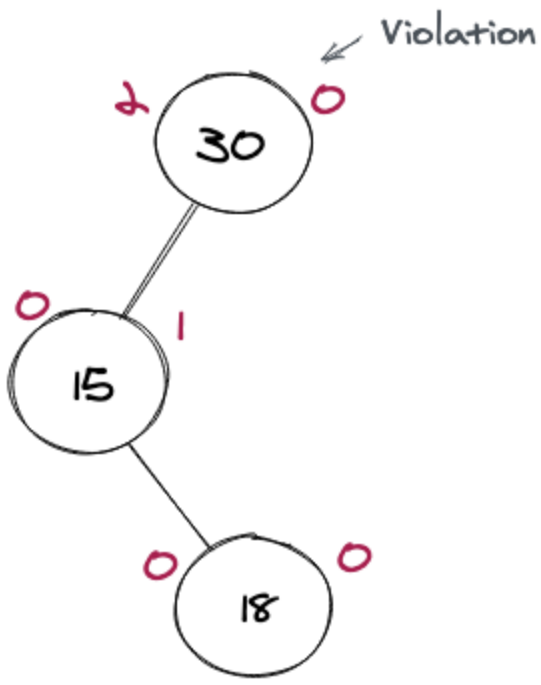
Consider the pictorial representation of making the above left-skewed tree into an AVL tree.



RIGHT ROTATE

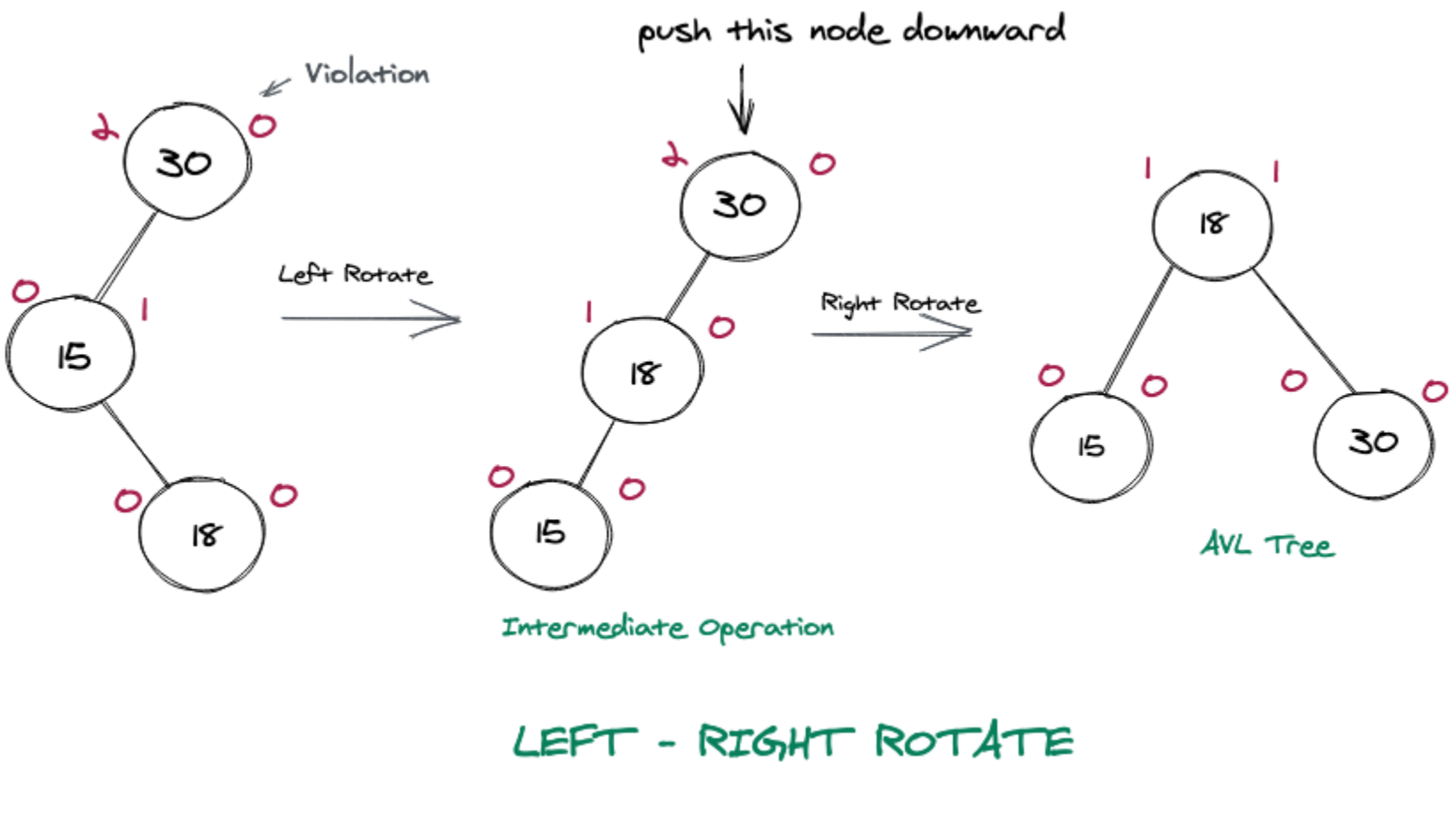
3. Left - Right Rotation

This rotation is performed when the violation occurs in the right subtree of the left child of the violating node. Consider the pictorial representation of a BST below that is not balanced, in order to make it so, we will do some rotations.



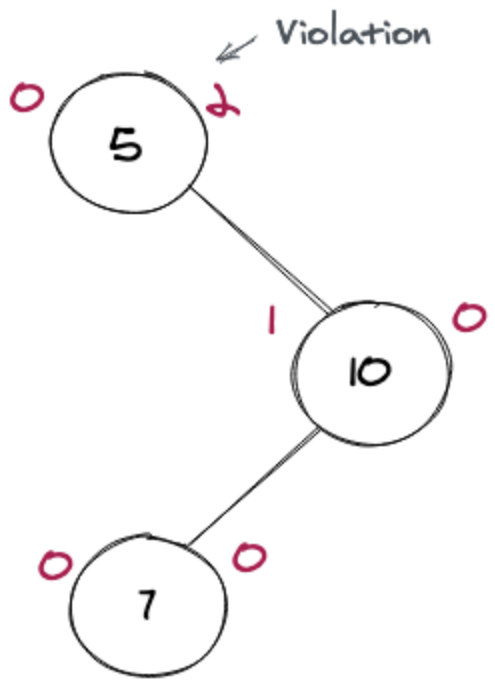
Clearly this is not an AVL tree, but consider the node where the violation occurs, we take this node as a pivot to do our rotations. The node that causes the issue is 18 as if we remove that one, we would have a balanced BST (i.e. AVL). Since the issue is because of the right child of the left subtree, we will do a Left-Right rotation to make it balanced.

Consider the pictorial representation of making the above tree into an AVL tree.



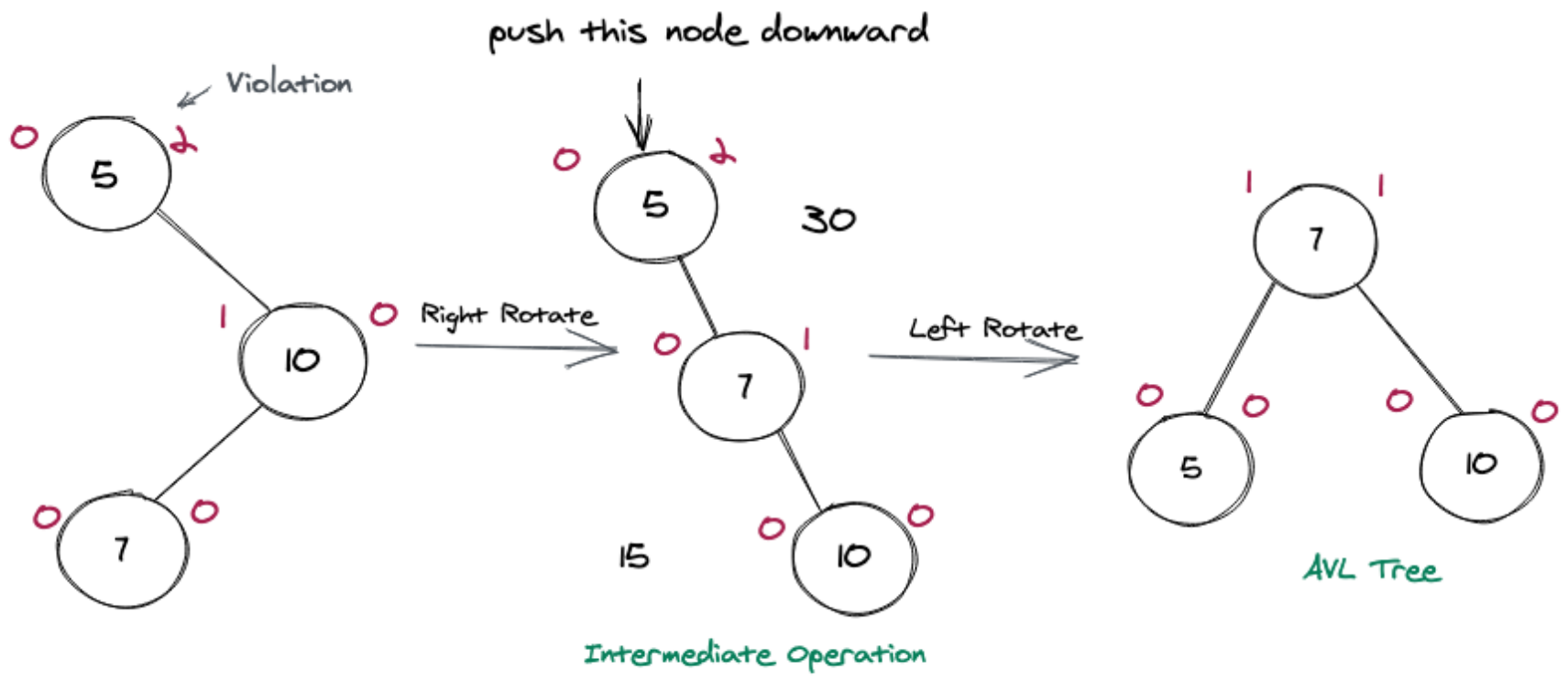
4. Right - Left Rotation

This rotation is performed when the violation occurs in the left subtree of the right child of the violating node. Consider the pictorial representation of a BST below that is not balanced, in order to make it so, we will do some rotations.



Clearly this is not an AVL tree, but consider the node where the violation occurs, we take this node as a pivot to do our rotations. The node that causes the issue is 7 as if we remove that one, we would have a balanced BST (i.e. AVL). Since the issue is because of the left child of the right subtree, we will do a Right-Left rotation to make it balanced.

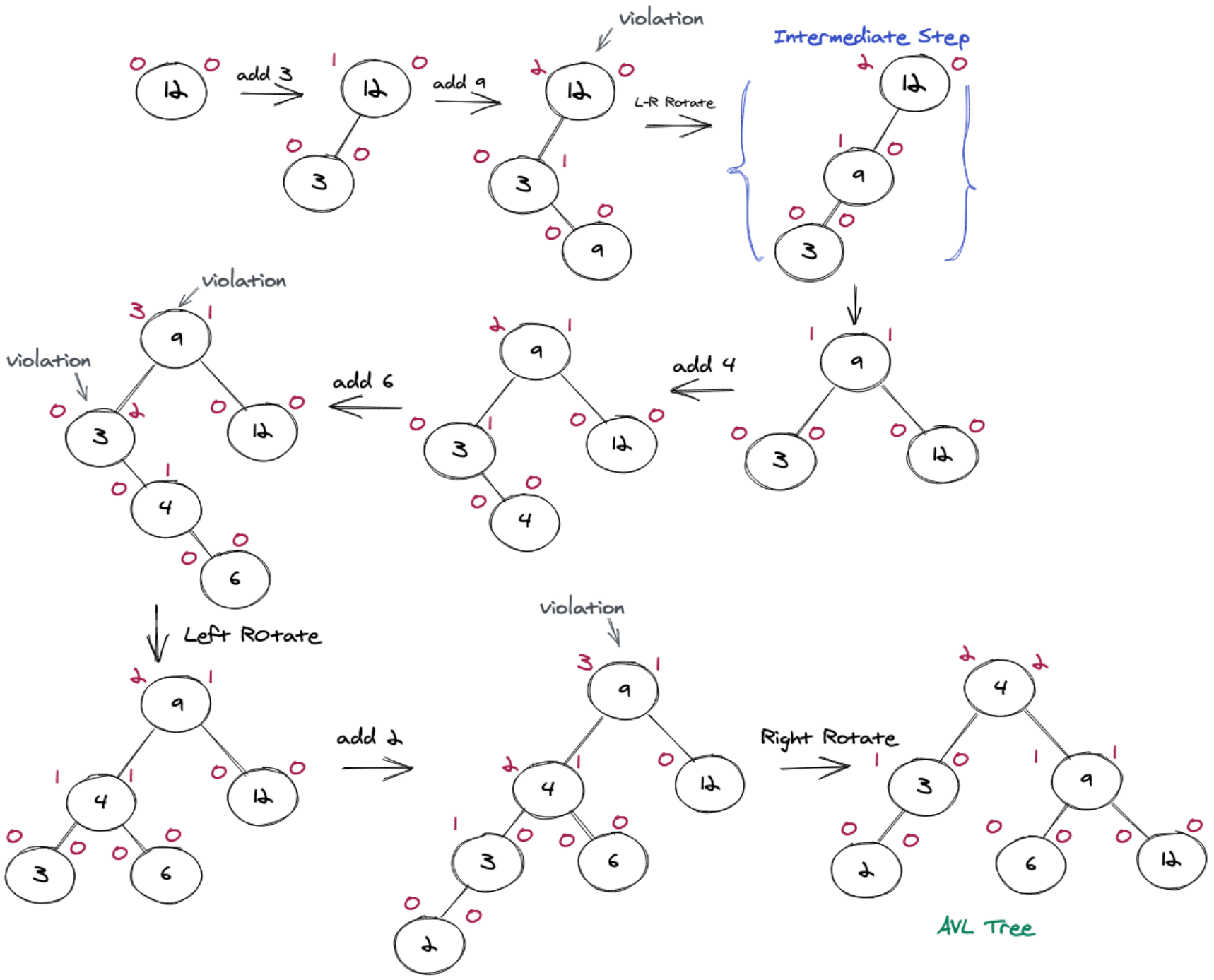
Consider the pictorial representation of making the above tree into an AVL tree.



AVL Tree Insertion

Let's take one array set of elements and build an AVL tree of these elements. Let, $\text{nums} = [12, 3, 9, 4, 6, 2]$

The complete step by step process of making an AVL tree (with rotations) is shown below.



INSERTING, IN AVL TREE

Key Points:

- An AVL tree with N number of nodes can have a minimum height of floor(logN) base 2.
- The height of an AVL tree with N number of nodes cannot exceed 1.44(logN) base 2.
- The maximum number of nodes in an AVL tree with height H can be : 2^H+1 - 1
- Minimum number of nodes with height h of an AVL tree can be represented as : N(h) = N(h-1) + N(h-2) + 1 for n>2 where N(0) = 1 and N(1) = 2.

Conclusions

- We learned what an AVL tree is and why we need it.
- Then we learned the different type of Rotations that are possible in AVL tree to make it a balanced one.
- Followed by the rotations, we also did an insertion example in an AVL Tree.
- Lastly, we talked about different key points that we should remember with AVL