Viva Question with solution

Tree Data Structure

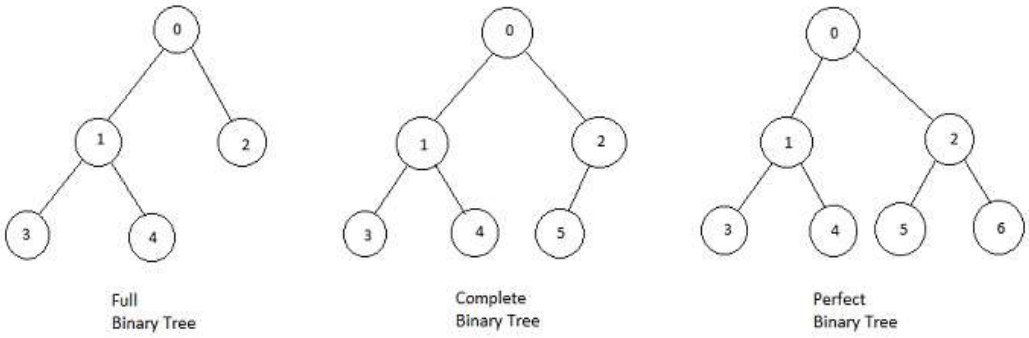
1. **Define Binary Tree**

Answer:

A normal tree has no restrictions on the number of children each node can have. A **binary tree** is made of nodes, where each node contains a "left" pointer, a "right" pointer, and a data element.

There are three different types of binary trees:

- **Full binary tree**: Every node other than leaf nodes has 2 child nodes.
- **Complete binary tree**: All levels are filled except possibly the last one, and all nodes are filled in as far left as possible.
- **Perfect binary tree**: All nodes have two children and all leaves are at the same level.



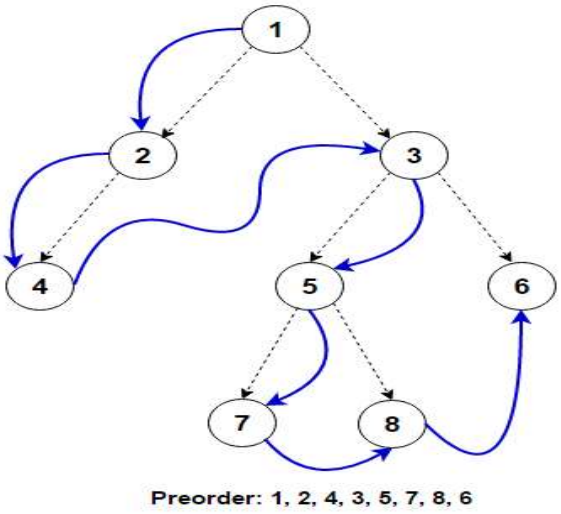
2. How to implement a *Tree* data-structure? Provide some code

3. How to implement *Pre-order Traversal* of *Binary Tree* using *Recursion*

Answer:

For traversing a (non-empty) binary tree in pre-order fashion, we must do these three things for every node N starting from root node of the tree:

- (N) Process N itself.
- (L) Recursively traverse its *left* subtree. When this step is finished we are back at N again.
- (R) Recursively traverse its *right* subtree. When this step is finished we are back at N again.

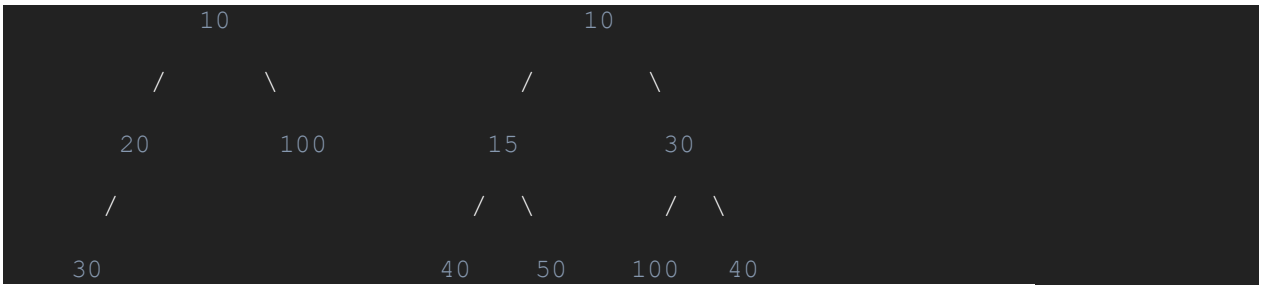


4. What is Binary Heap?

Answer:

A **Binary Heap** is a *Binary Tree* with following properties:

- It's a *complete* tree (all levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.
- A Binary Heap is either **Min Heap** or **Max Heap**. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.



5. What is Binary Search Tree?

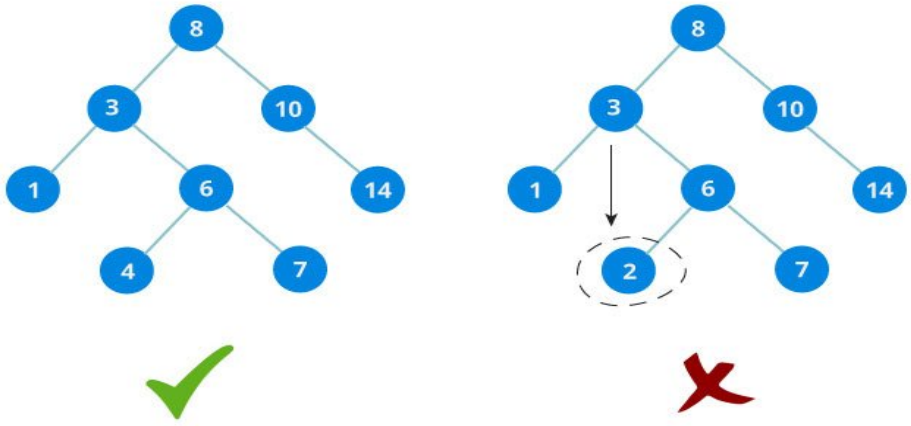
Answer:

Binary search tree is a data structure that quickly allows to maintain a *sorted list* of numbers.

- It is called a *binary tree* because each tree node has maximum of two children.
- It is called a *search tree* because it can be used to search for the presence of a number in `O(log n)` time.

The properties that separates a binary search tree from a regular binary tree are:

- All nodes of left subtree are less than root node
- All nodes of right subtree are more than root node
- Both subtrees of each node are also BSTs i.e. they have the above two properties



6. Classify Tree Traversal Algorithms. Provide some visual explanation.

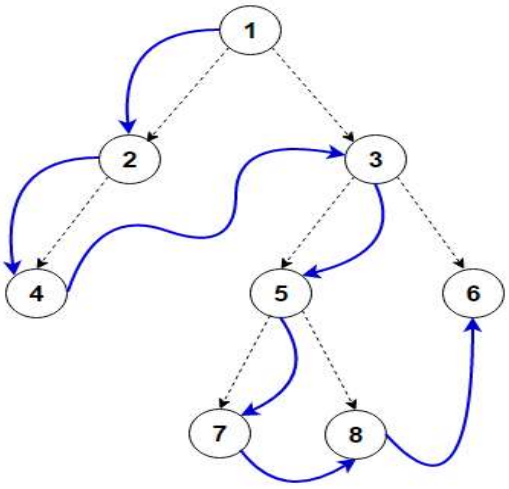
Answer:

Tree Traversal algorithms can be classified broadly in two categories:

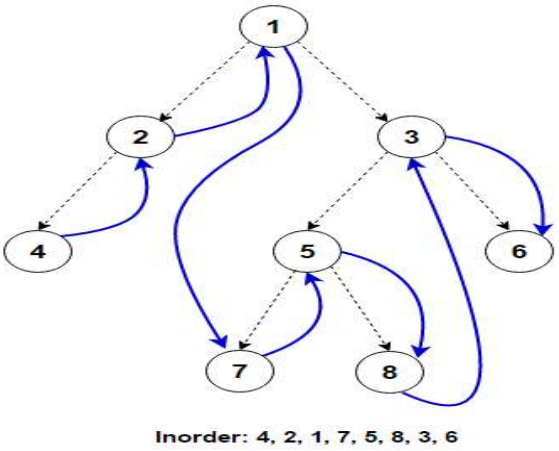
- Depth-First Search (DFS) Algorithms
- Breadth-First Search (BFS) Algorithms

Depth-First Search (DFS) Algorithms have three variants:

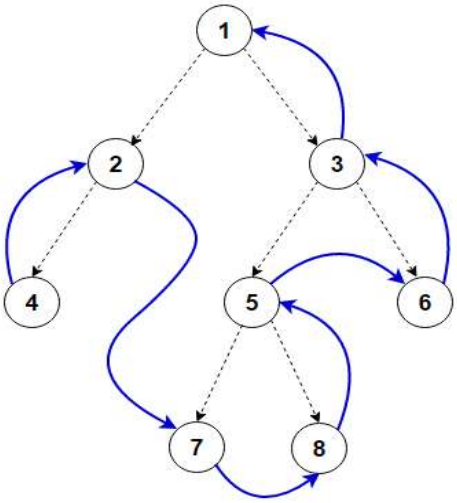
1. Preorder Traversal (current-left-right)— Visit the current node before visiting any nodes inside left or right subtrees.
2. Inorder Traversal (left-current-right)— Visit the current node after visiting all nodes inside left subtree but before visiting any node within the right subtree.
3. Postorder Traversal (left-right-current) — Visit the current node after visiting all the nodes of left and right subtrees.



Preorder: 1, 2, 4, 3, 5, 7, 8, 6



Inorder: 4, 2, 1, 7, 5, 8, 3, 6



Postorder: 4, 2, 7, 8, 5, 6, 3, 1

7. Explain the difference between Binary Tree and Binary Search Tree with an example?

Answer:

Binary tree: Tree where each node has up to two leaves

```
  1

 / \

2   3
```

Binary Search Tree (BST): Used for **searching**. A binary tree where the left child contains *only* nodes with values less than the parent node, and where the right child *only* contains nodes with values greater than or equal to the parent.

```
  2

 / \

1   3
```

8. What is AVL Tree?

Answer:

AVL trees are *height balancing binary search tree*. It is named after Adelson-Velsky and Landis, the inventors of the AVL tree. 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**. This allows us to search for an element in the AVL tree in `O(log n)`, where `n` is the number of elements in the tree.

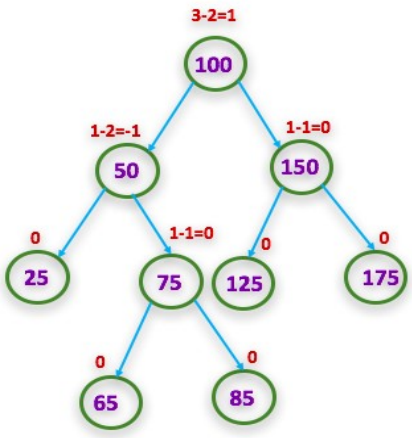
The **balance factor** of a node (N) in a binary tree is defined as the height difference:

```
BalanceFactor(N) = height(left_sutree(N)) - height(right_sutree(N))

// BalanceFactor(N) belongs to the set {-1,0,1}
```

If the balance factor doesn't equal -1,0, or 1 then our tree is unbalanced, and we need to perform certain operations (called **rotations**) to balance the tree. Specifically we need to do one or more of 4 tree **rotations**:

- Left Rotation,
- Right Rotation,
- Left Right Rotation,
- Right Left Rotation.



AVL Tree

9. What is Balanced Tree and why is that important?

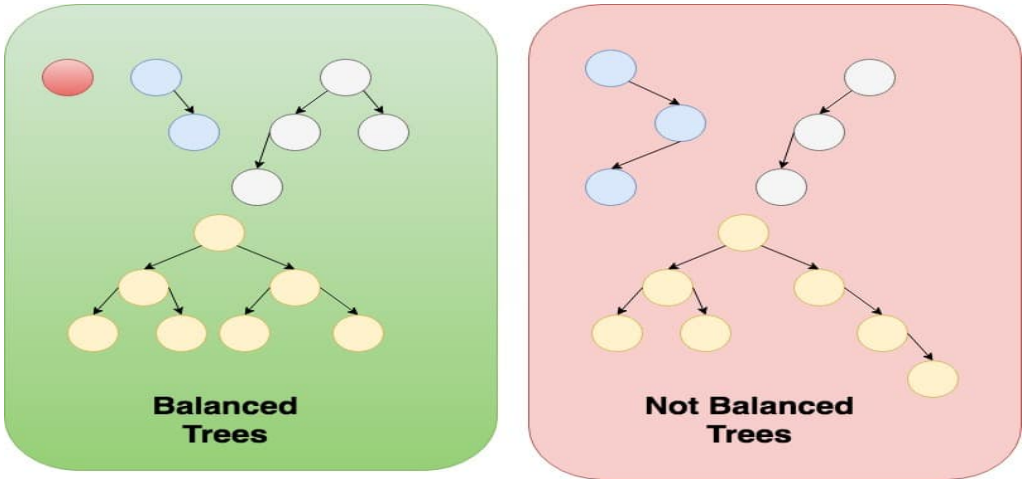
Answer:

A tree is said to be balanced only when all three conditions satisfy:

- The left and right subtrees height differ by at most one
- The left subtree is balanced
- The right subtree is balanced

Height-balancing requirement:

- A node in a tree is height-balanced if the heights of its subtrees differ by no more than 1.
- A tree is height-balanced if all of its nodes are height-balanced.



In a balanced BST, the height of the tree is `log n` where `n` is the number of elements in the tree. In the worst case and in an unbalanced BST, the height of the tree can be upto `n` which makes it same as a linked list. In the worst case each of the operations (lookup, insertion and deletion) takes time `O(n)` that shall be avoided.

Balanced BST maintains `h = O(log n)` so all operations run in `O(log n)` time.

10: Why do we want to use Binary Search Tree?

Answer:

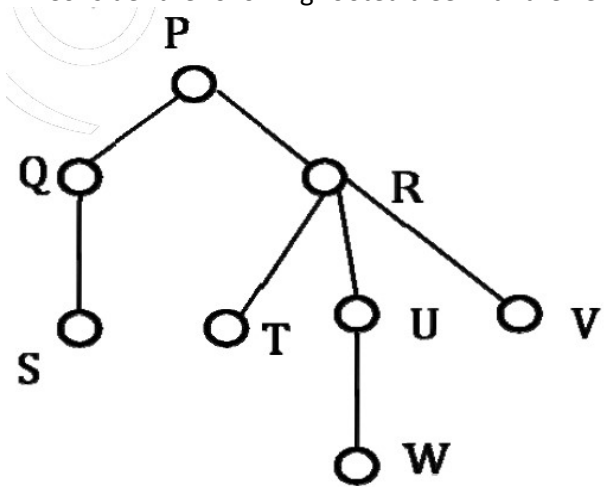
If we implement a **balanced binary search tree**, we can always keep the cost of `insert()`, `delete()`, `lookup()` to `O(log n)` where `n` is the number of nodes in the tree - so the benefit really is that lookups can be done in logarithmic time which matters a lot when n is large. This is much better than the *linear* time `O(n)` required to find items by key in an (unsorted) array, but slower than the corresponding operations on hash tables.

11. Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual ordering on natural numbers. What is the in-order traversal sequence of the resultant tree?

Answer:

0 1 2 3 4 5 6 7 8 9

12. Consider the following rooted tree with the vertex P labeled as root



What will be the order in which the nodes are visited during **in-order traversal**?

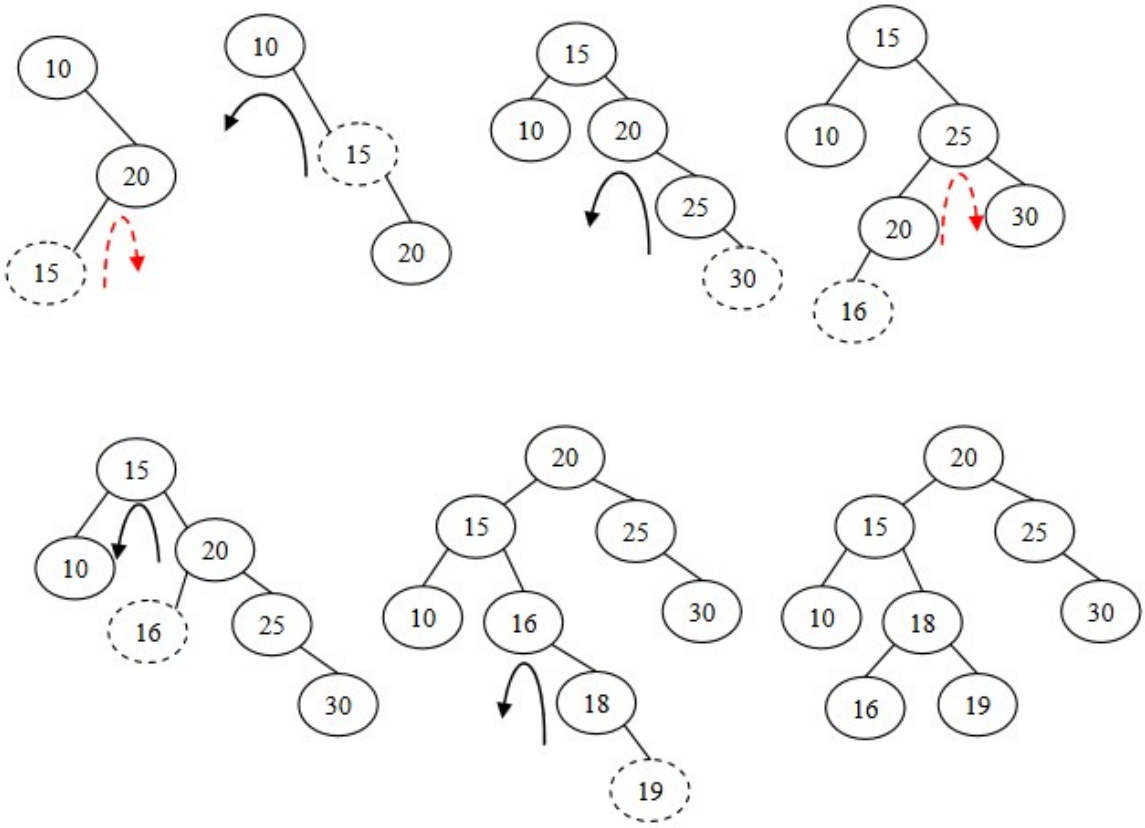
Answer:

SQPTRWUV

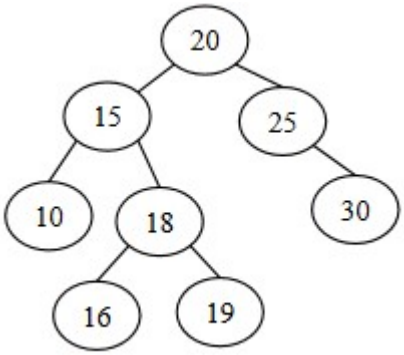
13. Insert the following sequence of elements into an AVL tree, starting with an empty tree: 10, 20, 15, 25, 30, 16, 18, 19.

Answer:

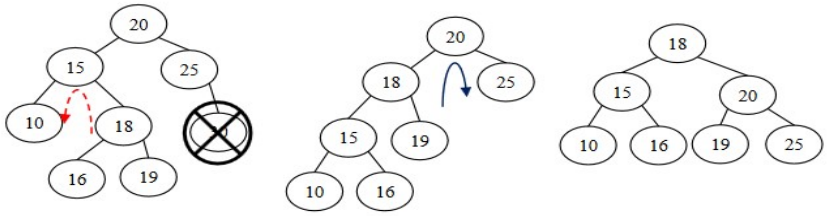
Red dashed line signifies first part of double rotate action.



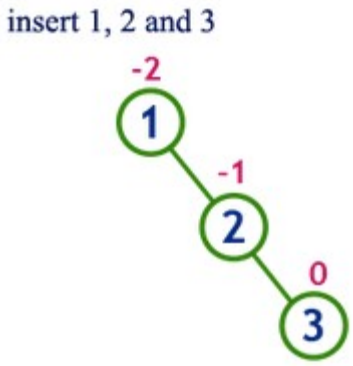
14. Delete 30 from given AVL tree?



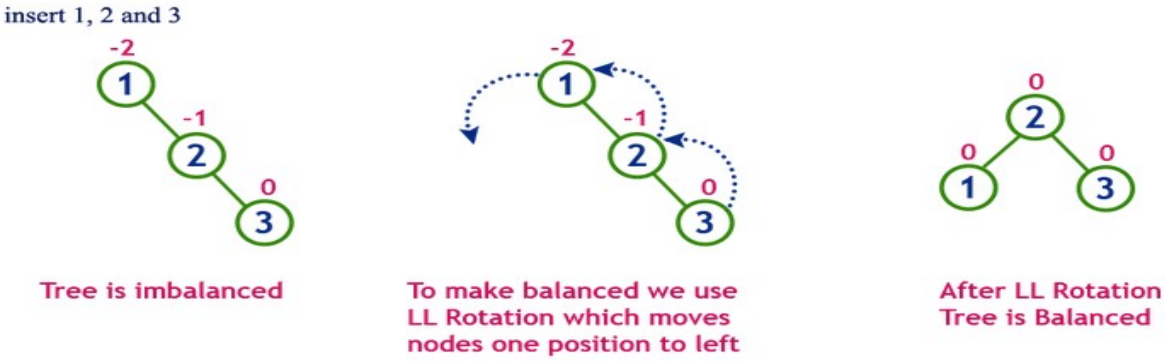
Answer:



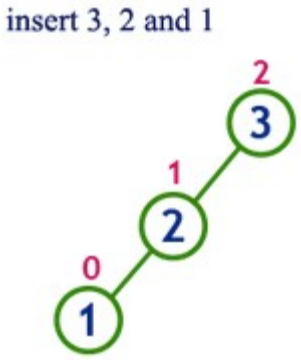
15. what will be the LL rotation of the following problem



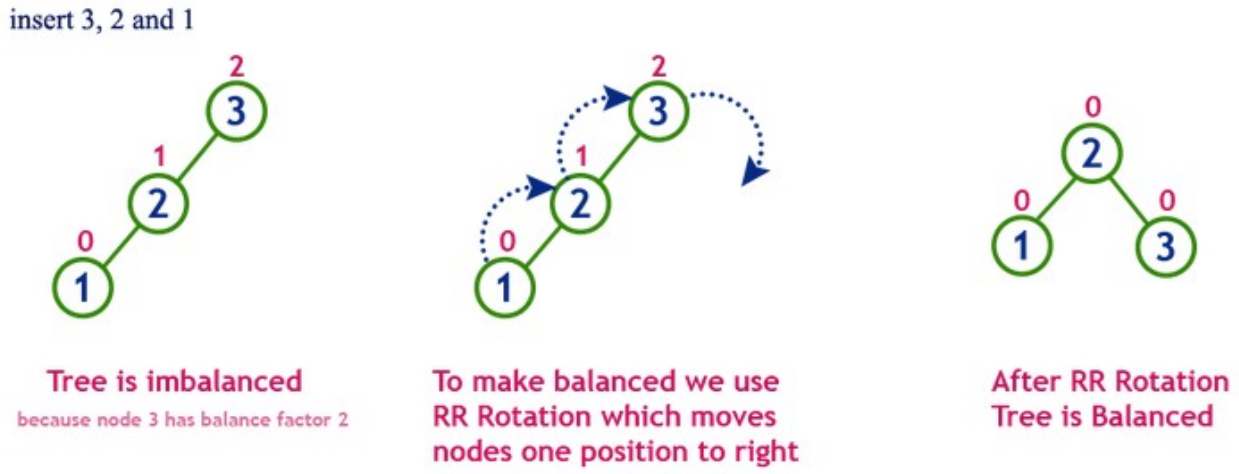
Answer:



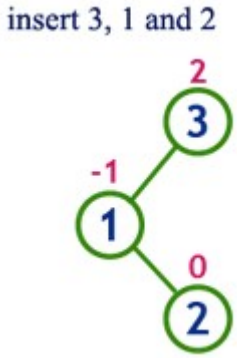
16. **what will be the RR rotation of the following problem**



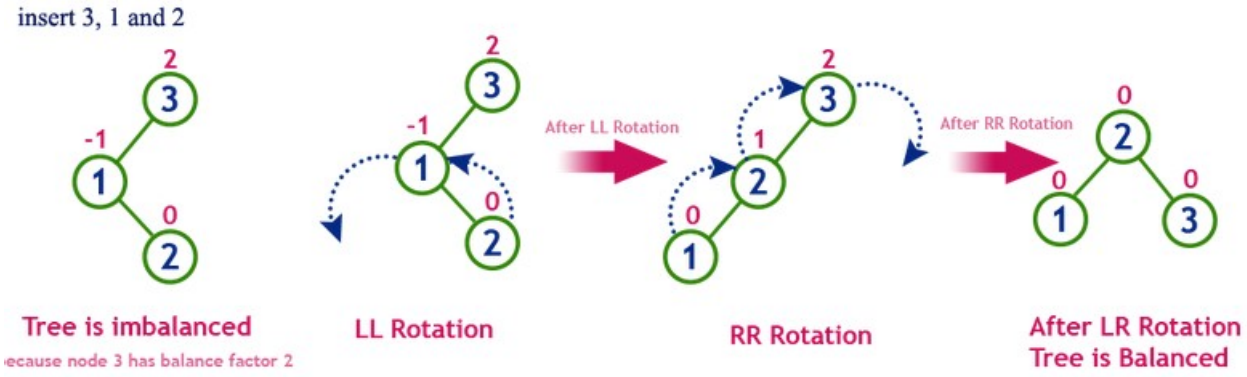
Answer:



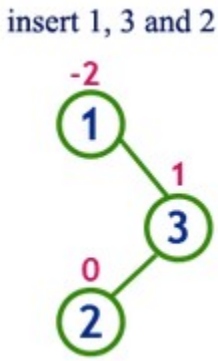
17. what will be the LR rotation of the following problem



Answer:



18. what will be the RL rotation of the following problem



Answer: