Balanced Search Tree

Balanced Tree Structure ~~are~~ tree structure whose height is $O(\log n)$.

* The performance for the search, insert & delete operations of a search tree is $O(\log n)$

* One of the more popular balanced ~~Trees~~ known as AVL Tree. [Adelson-Velskey-Landies]

Definition :- An empty binary tree is an AVL tree. If T is a non-empty binary tree with T_L & T_R as its left & right subtrees, then T is an AVL tree if ① T_L & T_R are AVL trees & ② $|h_L - h_R| \leq 1$ where h_L & h_R are the heights of T_L & T_R respectively.

* AVL ~~tree~~ is a self-balanced ~~tree~~ binary search Tree.

* Every AVL search tree is a binary search tree but all the binary search Trees need not to be AVL trees.

Ex:-

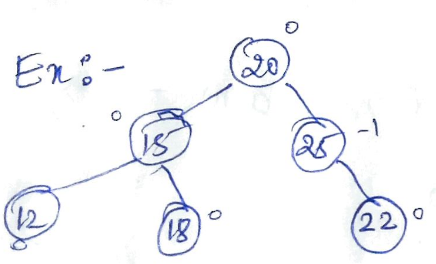


fig (a)

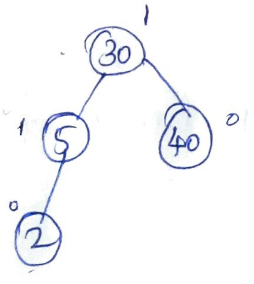


fig (b)

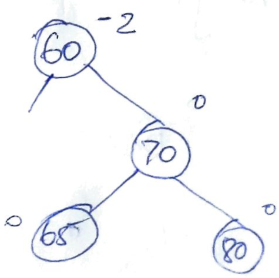


fig (c)

* AVL search Tree is a binary search Tree & that is also an AVL Tree.

* fig (a) & (b) trees are AVL trees & fig (c) is not

* Tree (a) is not an AVL search tree as it is not a binary search tree.

* AVL search tree represents a dictionary & perform each operation in logarithmic time.

* The height of an AVL tree with n elements or nodes is $O(\log n)$. It takes $O(\log n)$ time for search operation.
* Insert operation — $O(\log n)$ time.
* Delete —"— — $O(\log n)$ time

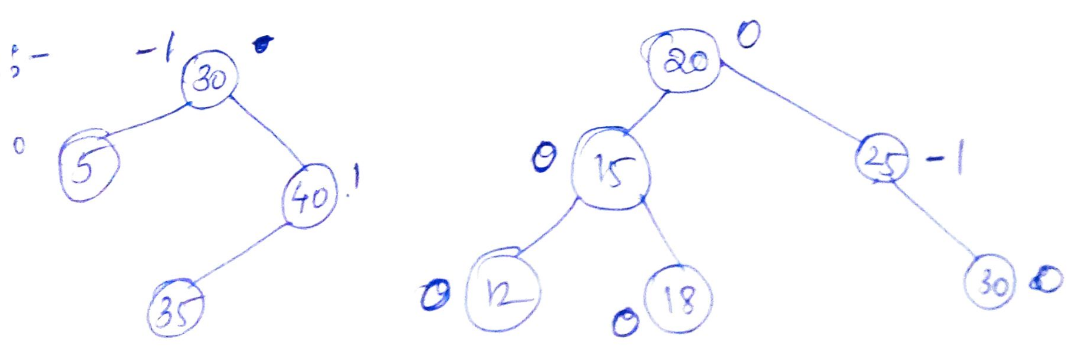
Representation of an AVL tree

* AVL trees are represented by using the linked representation scheme for binary trees.
* To facilitate insertion & deletion, a balance factor bf is associated with each node.
* The bf(x): of a node x is defined as,

height of left subtree of x - height of right ~~Such~~ subtree of x.

* The permissible balance factors are -1, 0, +1.

Ex:-



The number outside each node is its bf

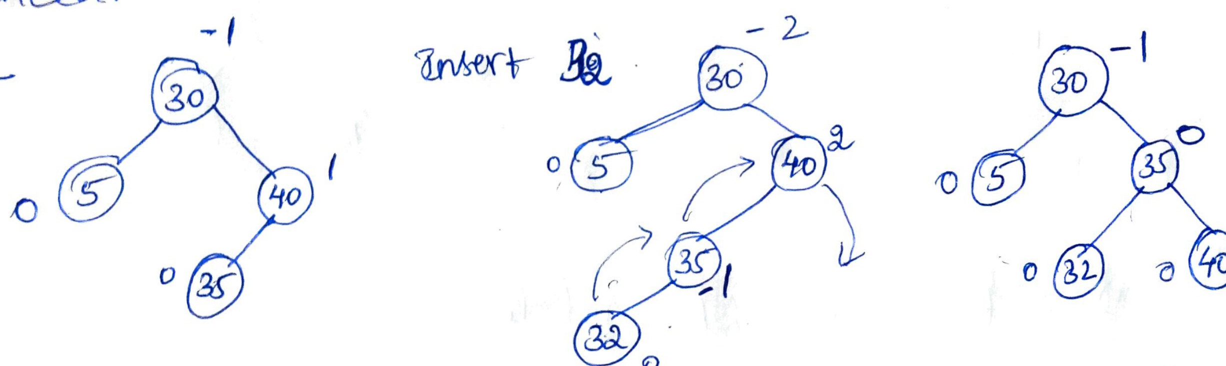
Searching an AVL search Tree

Searching an AVL tree is simillar to binary search Tree. Since the height of an AVL tree with n Elements is O(logn), the search time is O(logn)

2) Inserting into an AVL search Tree

* In AVL tree, after performing every operation like, insertion, & deletion, we need to check the balance factor (bf) of every node in the tree.
* If every node satisfies the bf condition then we conclude the operation, otherwise we must make it balanced.

Ex:-



imbalanced AVL Tree

rebalancing

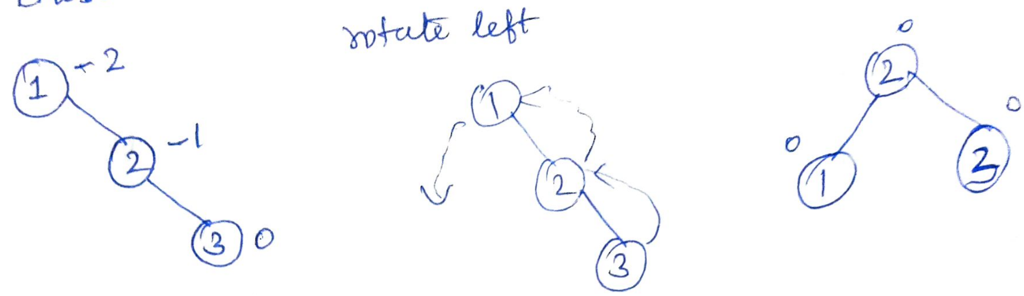
* Rotation operations make tree balanced whenever the tree is becoming imbalanced, due to any operation.
* There are 4 rotations & they are classified into two types.

- Rotation
  - Single Rotation
    - Left rotation (LL)
    - Right rotation (RR)
  - Double rotation
    - LR Rotation
    - RL rotation

(i) Single left rotation (LL)

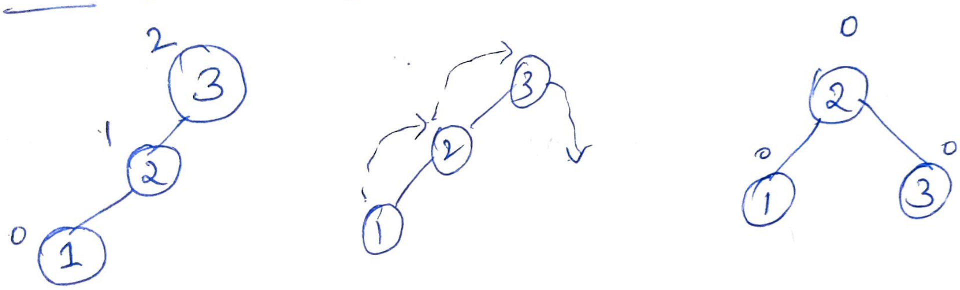
* In LL rotation every node moves one positio
to left from the current position.

Ex:- Insert 1, 2 & 3.



(ii) Single right rotation (RR)

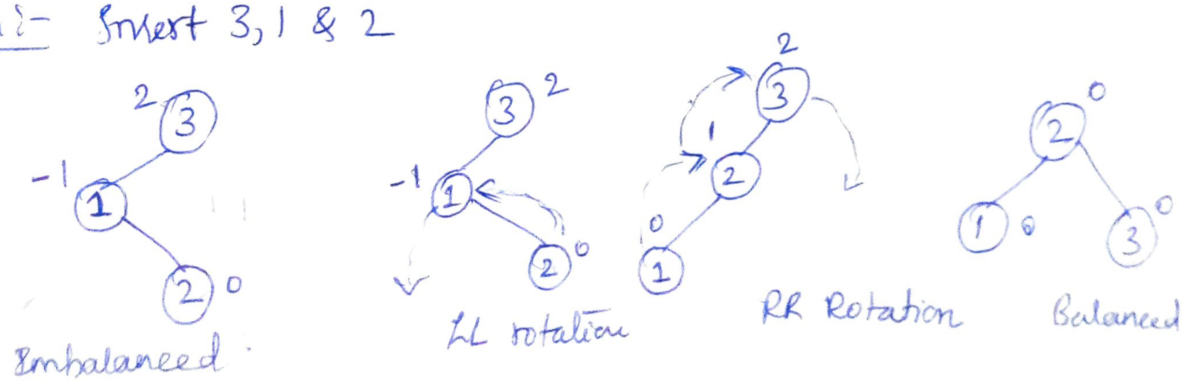
Ex:- Insert 3, 2 & 1



(iii) Left-right (LR) rotation.

* The LR rotation is combination of single left rotation followed by single right rotation.
* In LR rotation first every node moves one position to left then one position to right from the current position.

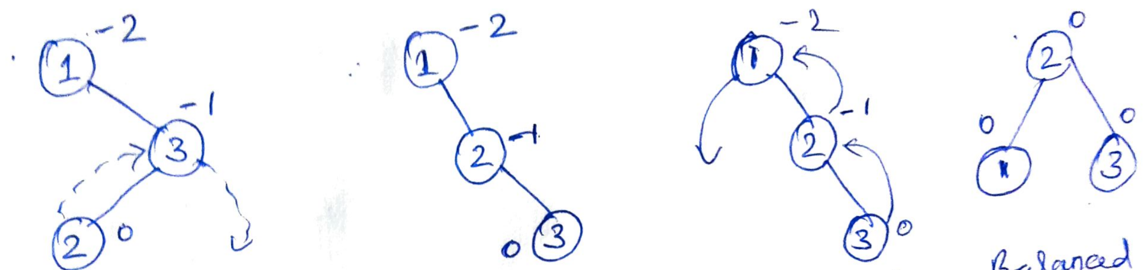
Ex:- Insert 3, 1 & 2



Right left rotation (RL)

* The RL rotation is combination of single right rotation followed by single left rotation.
* In RL, first every node moves one position to right then one position to left from the current position.

Ex:- Insert 1, 3 & 2



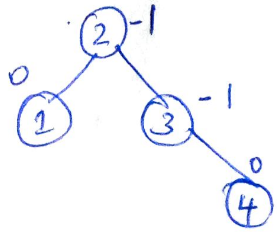
imbalanced Tree | RR rotation | LL rotation | Balanced

Example :- Construct an AVL tree by inserting numbers from 1 to 8.

* insert 1 | * insert 2 | * insert 3

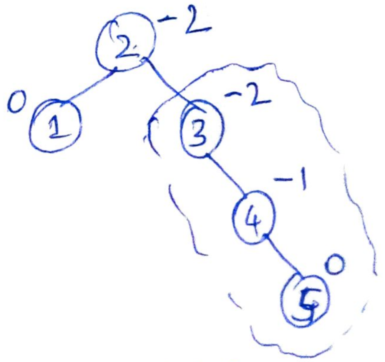
imbalanced | LL rotation | Balanced.

* Insert 4



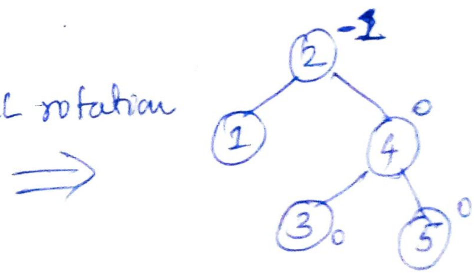
Balanced

* Insert 5



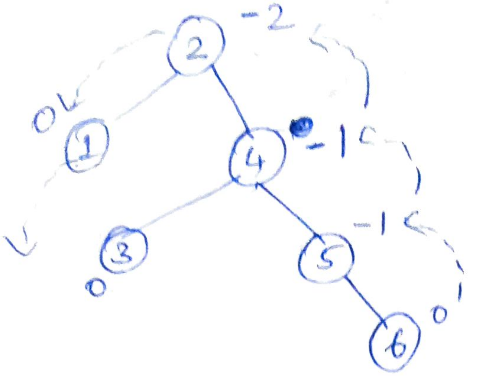
LL rotation at 3

LL rotation

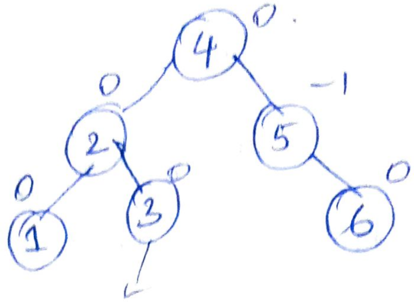


Balanced

* Insert 6.

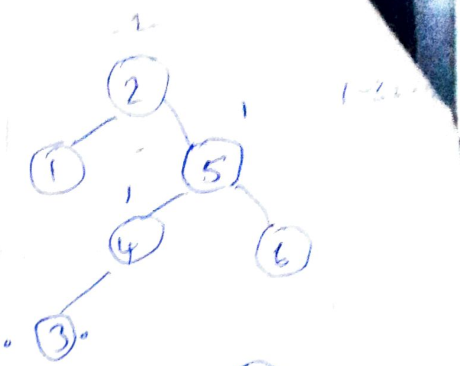


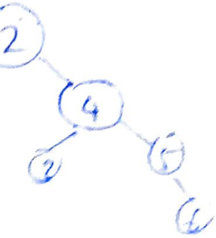
LL rotation at 2



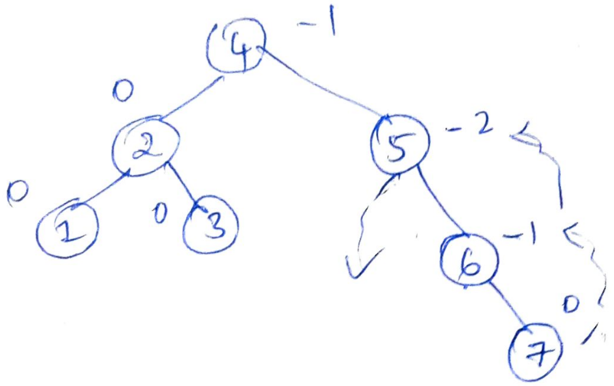
Becomes right child of 2

Balanced

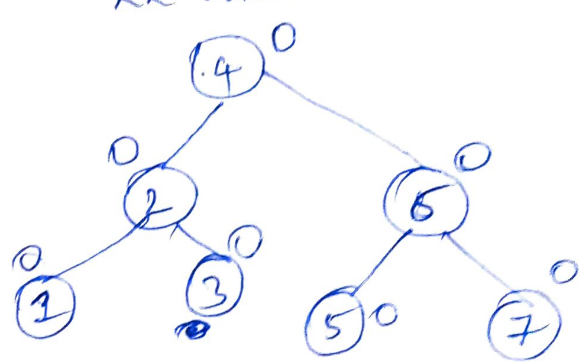




* Insert 7.

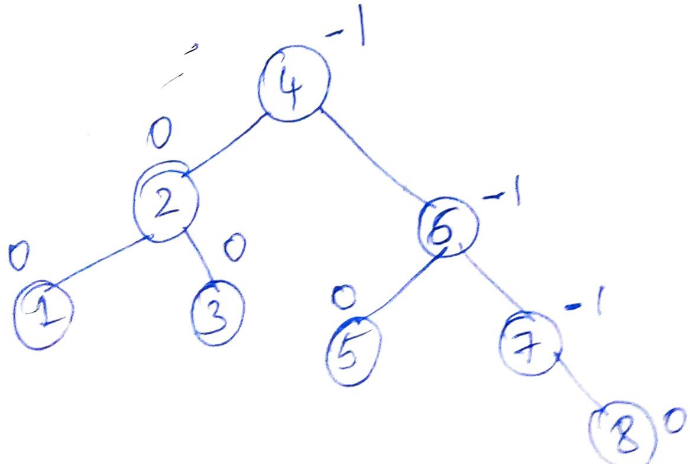


RL rotation



Balanced

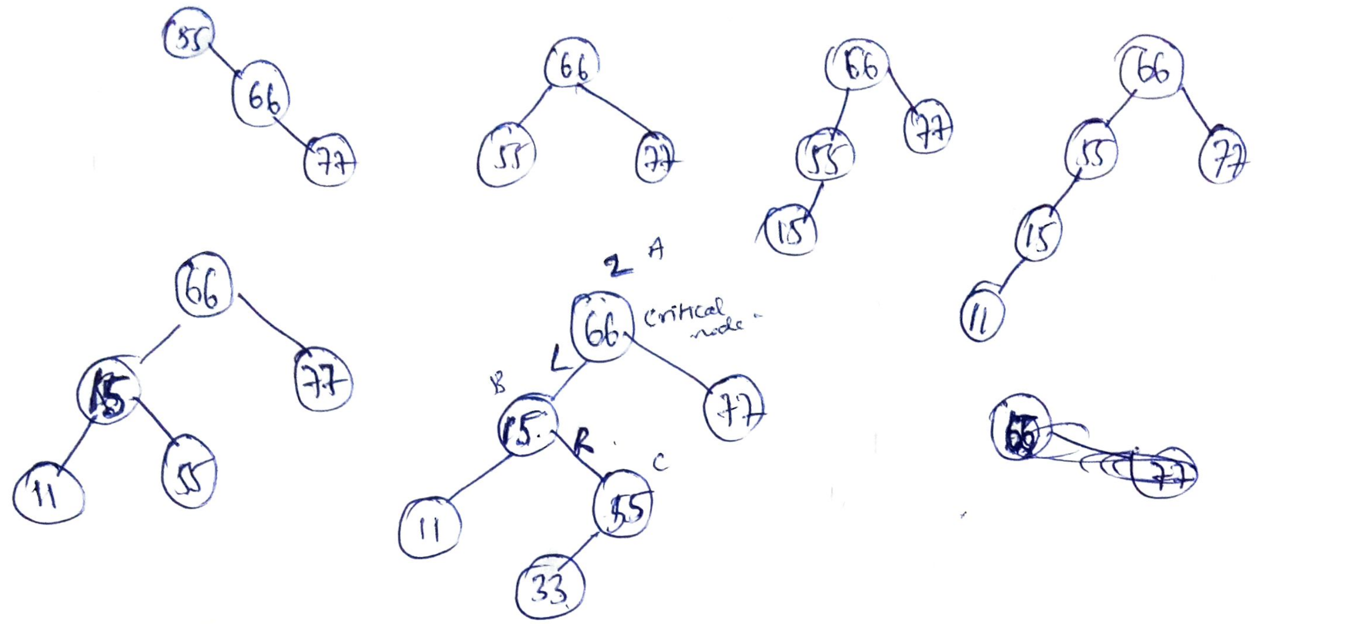
* Insert 8.



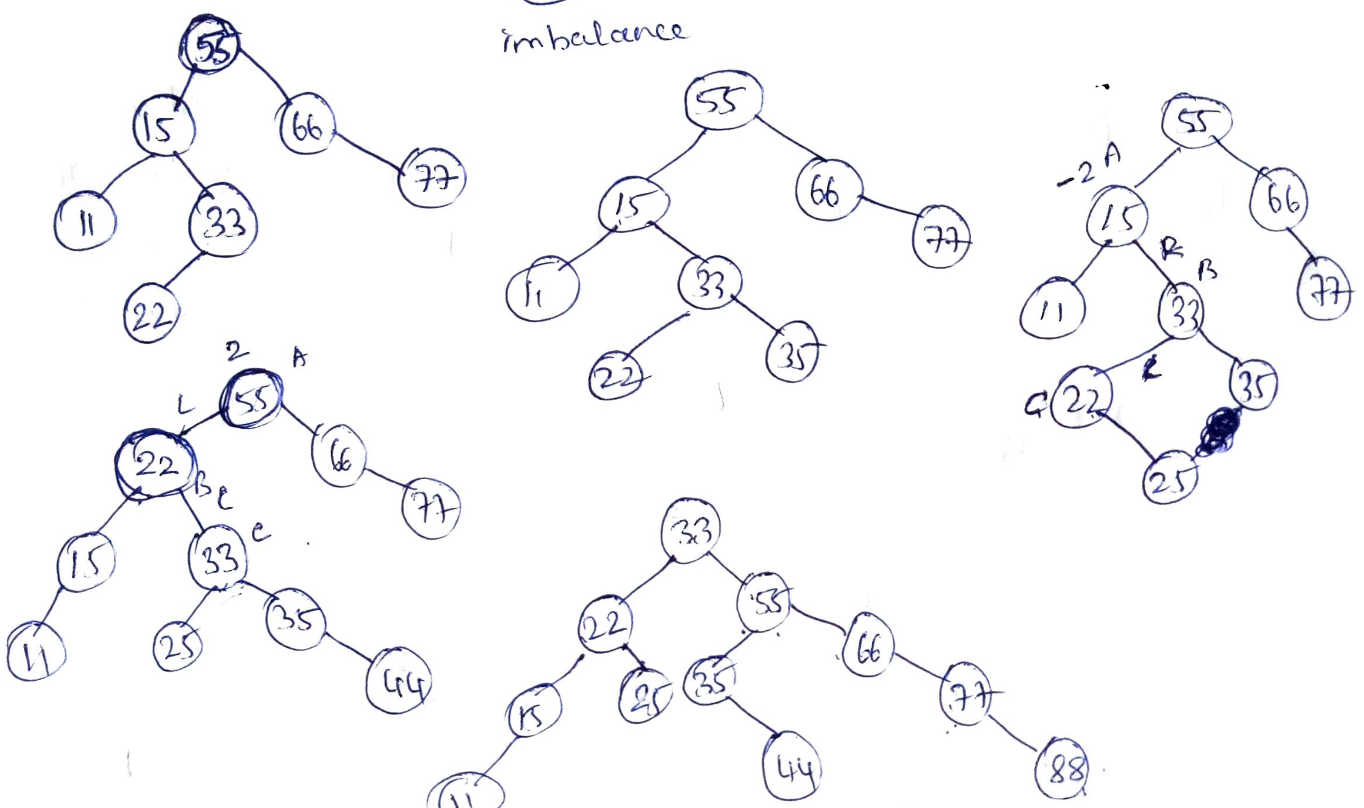
Balanced Tree.

Construct a AVL tree for the following numbers

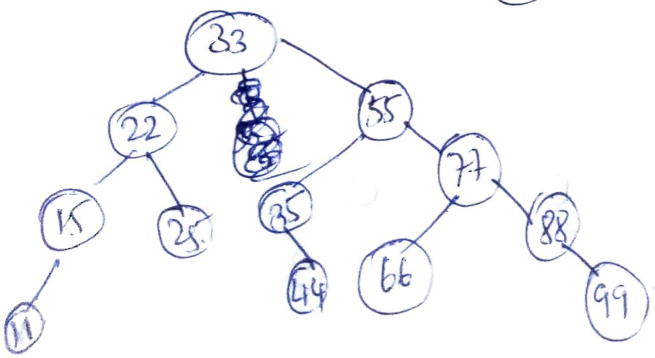
55 66 77 15 11 33 22 35 25 44 88 99



imbalance



RR rotation



* It takes ($O \log_2 n$) time for operation.

*

Final AVL tree