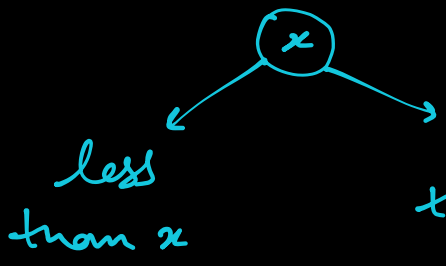
AVL Trees

https://youtu.be/jDM6_TnYIqE

Binary Search trees



N → no. of nodes

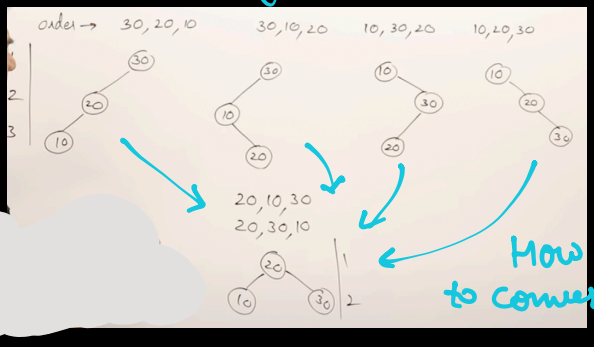
Searching → start from root and go left & right to find → O(N) → max height of BST

inserting → go left and right and insert at right place → O(N)

deleting → find the node then change the value to the value of the node on RHS and so on until the leave node → O(N) + O(N) = O(N)

Balancing a binary Search Tree

→ min height BST ⇒ Aim



→ Rotations → always done on 3 nodes

→ 4 types of rotations

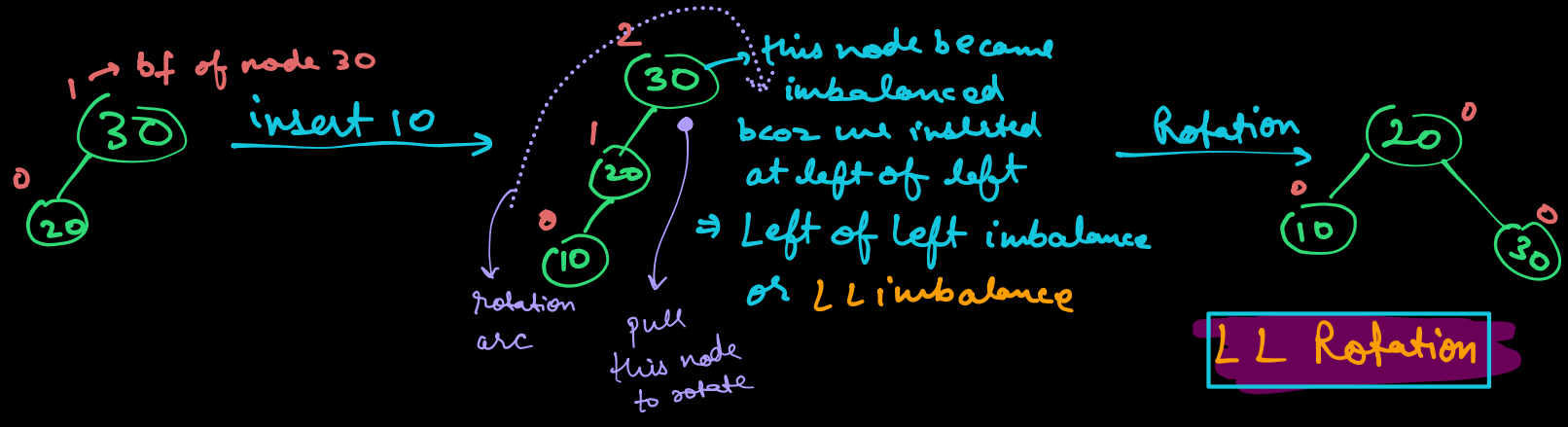
Balance factor = ht. (left subtree) − ht (right subtree)

$bf = h_l - h_r = \{-1, 0, 1\}$ ⇒ if AVL tree (i.e. balanced BT)

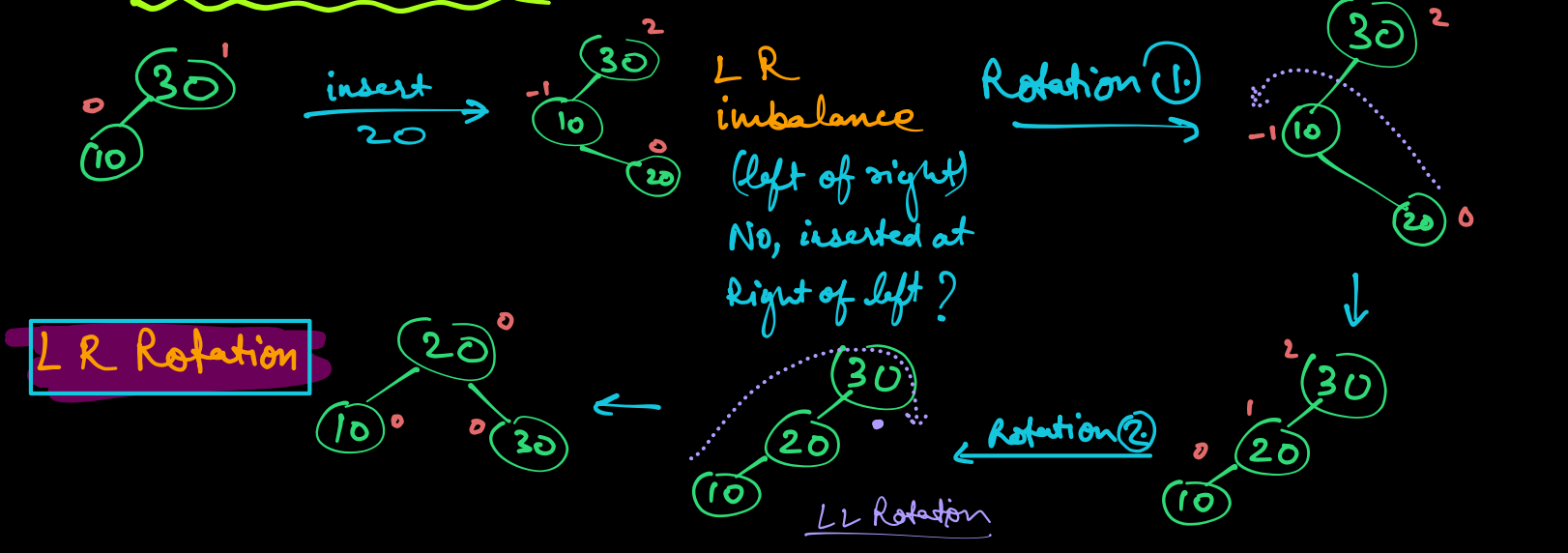
i.e. $|bf| = |h_l - h_r| \leq 1$

Rotations

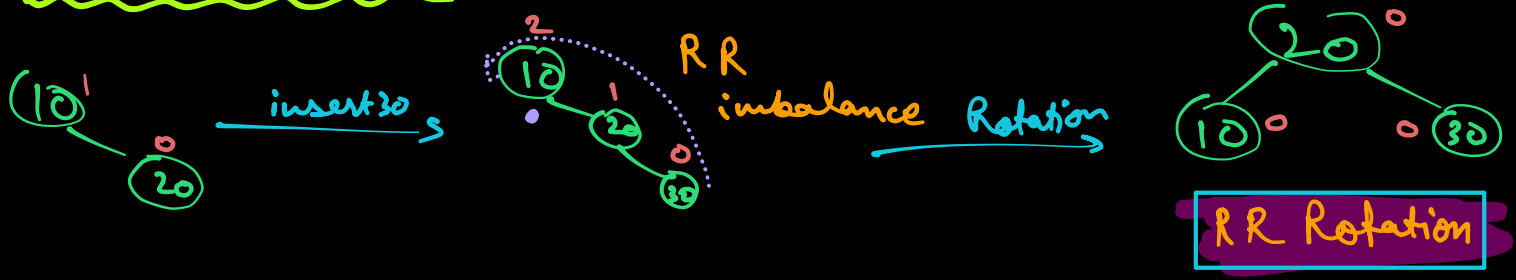
① LL Rotation



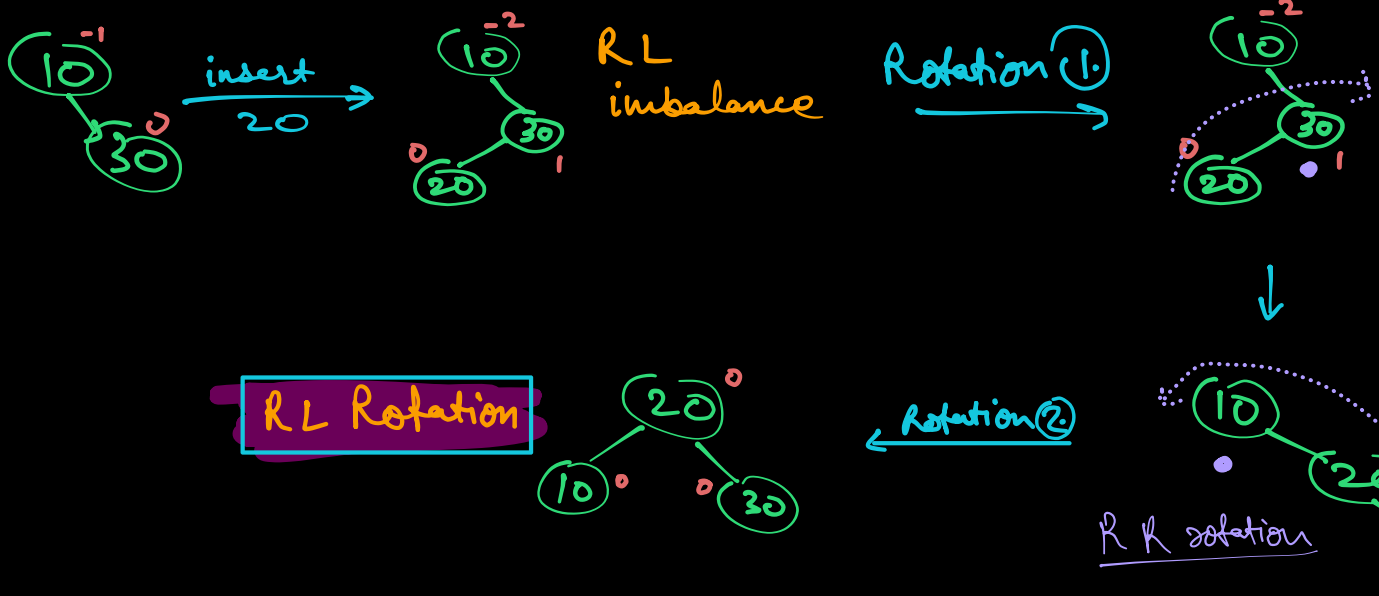
② LR Rotation



③ RR Rotation

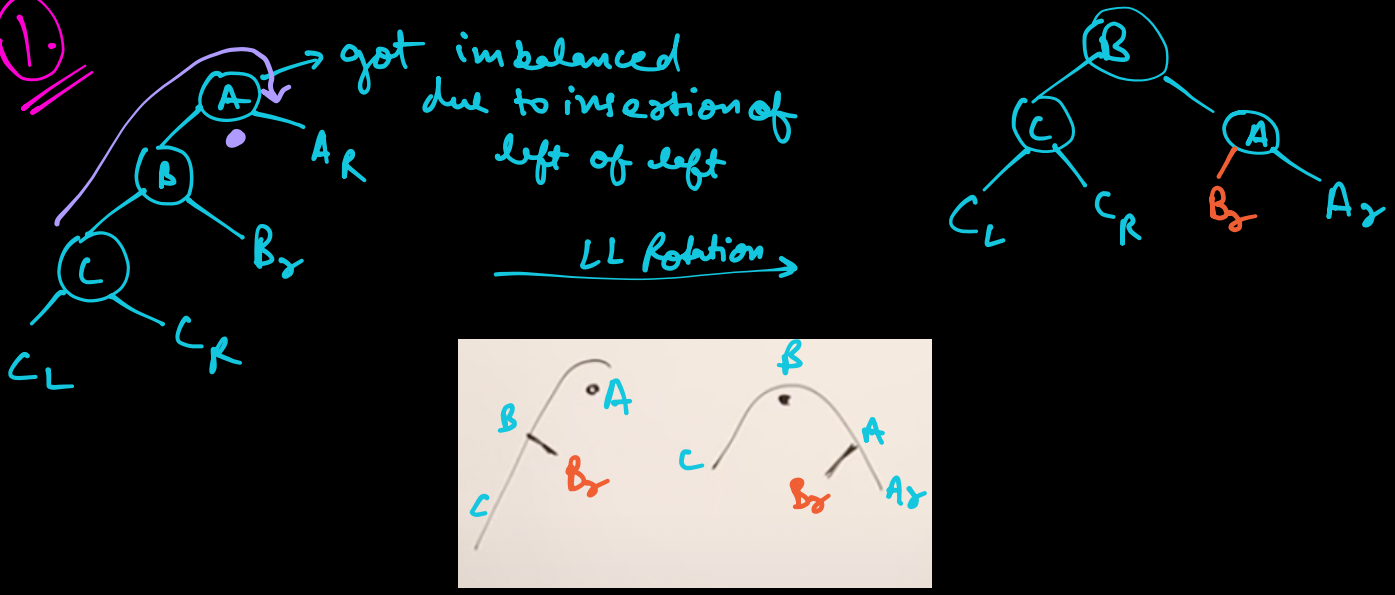


4. RL - Rotation

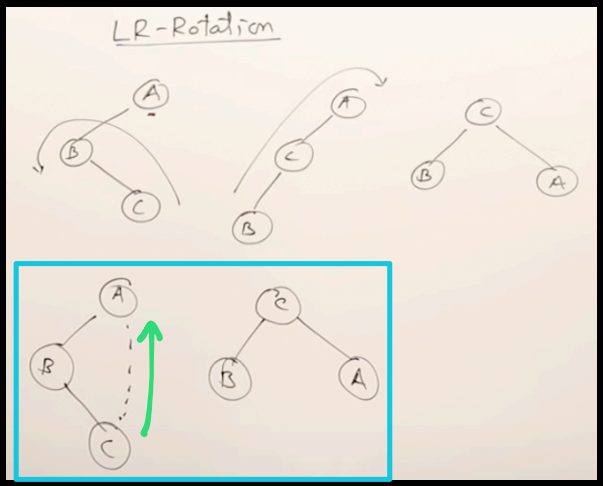


Special Cases

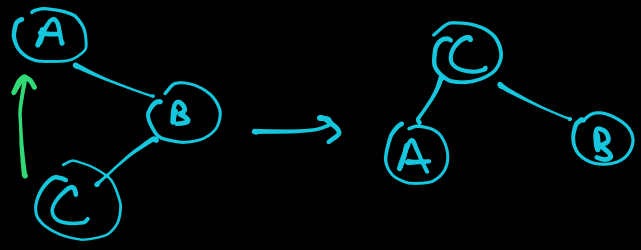
1.



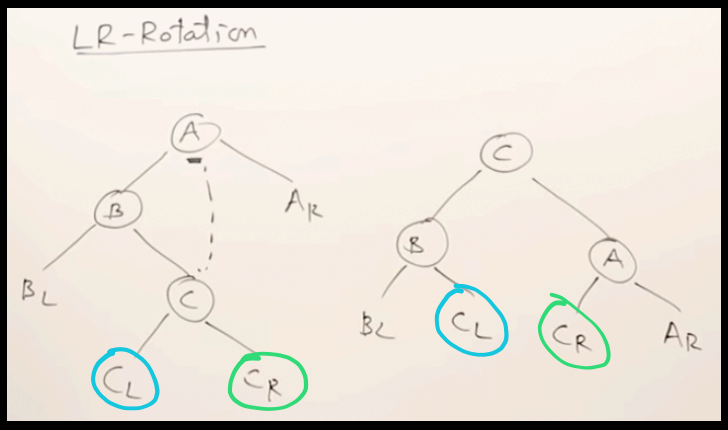
2.



RL Rotations

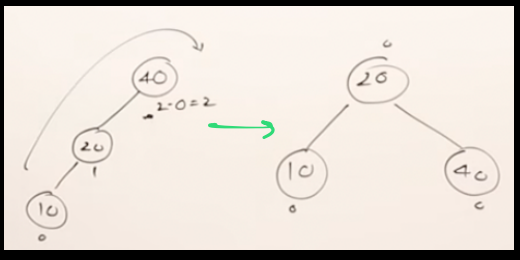


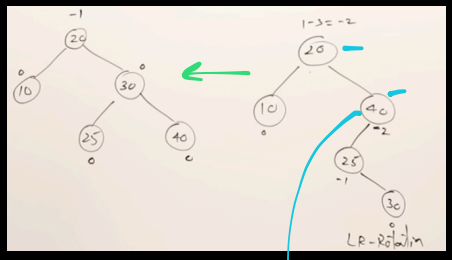
3

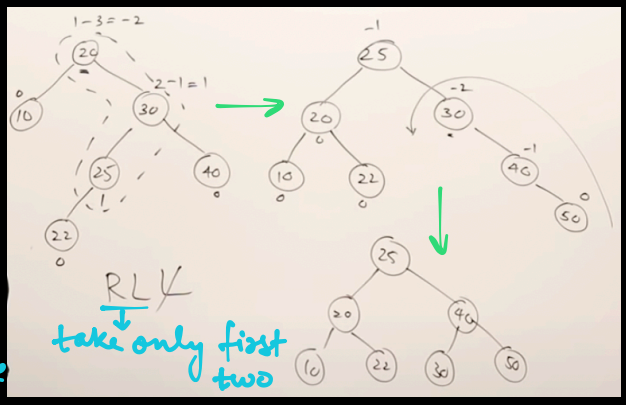


AVL Trees

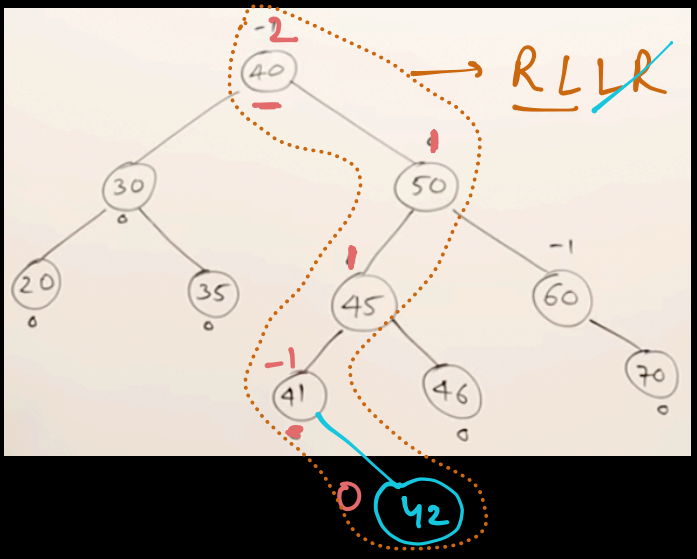
Keys:— 40, 20, 10, 25, 30, 22, 50

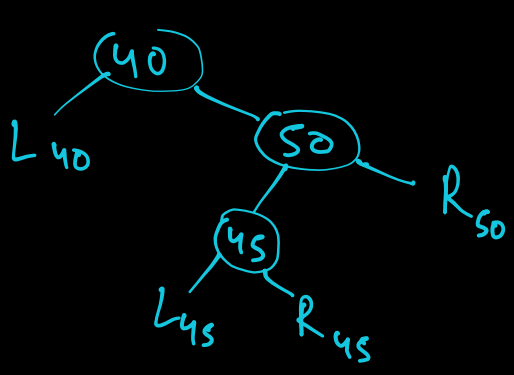


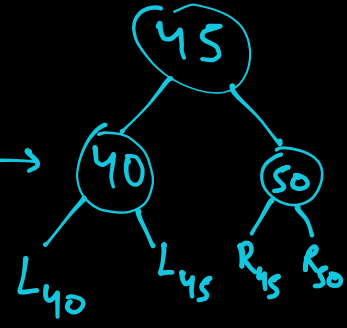


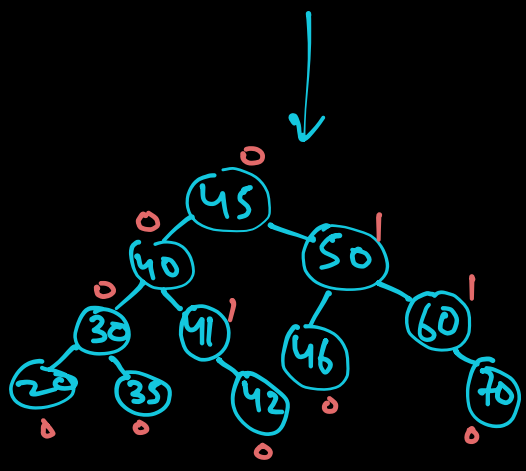


first balance this node, node 20 will automatically get balanced.









At most the height of an AVL tree $= \frac{1.44 \log N}{\downarrow ?}$

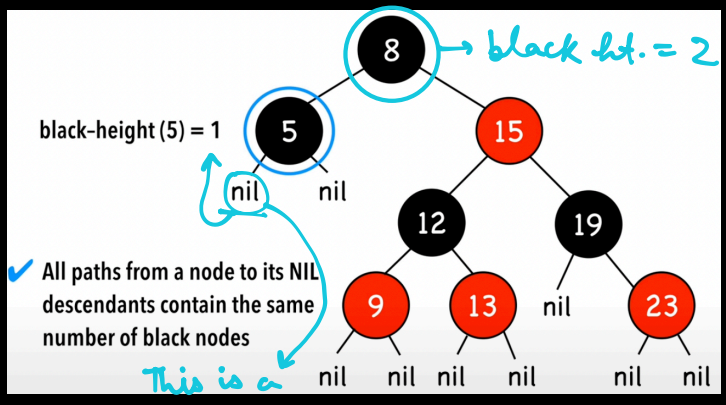
Searching in AVL tree = O(logn)

To avoid a lot of rotations in an AVL tree we use Red-black trees.

https://youtube.com/playlist?list=PL9xmBV_5YoZNqDI8qfOZgzbqahCUmUEin

red-black tree

1. A node is either red or black.
2. The root and leaves (NIL) are black.
3. If a node is red, then its children are black.
4. All paths from a node to its NIL descendants contain the same number of black nodes.



extra notes

1. Nodes require one storage bit to keep track of color.
2. The longest path (root to farthest NIL) is no more than twice the length of the shortest path (root to nearest NIL).
   - Shortest path: all black nodes
   - Longest path: alternating red and black

time complexity

Search	O(log n)
Insert	O(log n)
Remove	O(log n)

rotation

Space complexity = O(N)

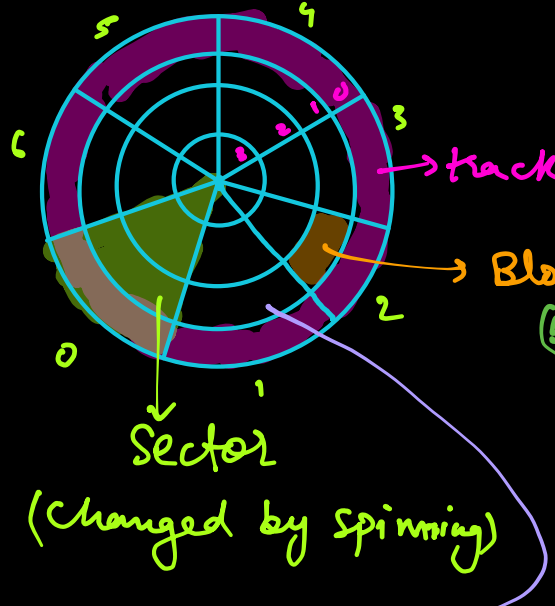
B Trees

B-Trees & B+-Trees

1. Disk structure
2. How Data is Stored on Disk
3. What is Indexing
4. What is Multilevel Indexing
5. M-way Search Trees
6. B-Trees
7. Insertion & Deletion - B-Trees
8. B+-Trees

block address = (track, sector size)
for study purpose, block size=512 bytes

Disk Structure



Block add. = (Track no., Sector no.)

* Data is stored on the disk in terms of blocks.

→ organising the data in M.M. is called 'Data Structures'

* Organising the data efficiently on disk for easy utilisation is DBMS.