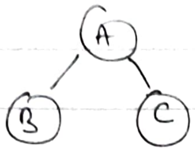
More on complexity later

★ We can use a type of binary search tech for searching an elem in bst. youvx

BST (Binary Search Tree).

→ 

Binary → 0,1,2 branches [Duplicate elements not allowed].

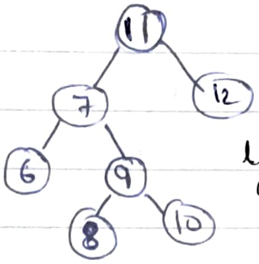
(C) → Always greater than (A)

(B) → Always less than (A)

B Case → O(1)
Avg → O(log₂n) = O(h)
W case → O(n)

Acc to the height.

Even the subtree of B will be < A.
vice versa for C.

Eg: – 

logical inst & deletion explained.

Coding will remain the same as in Binary Tree just data will change.

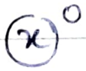
AVL Tree

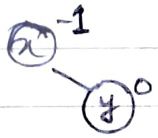
① → It is a BST. O(log₂n) = Always.

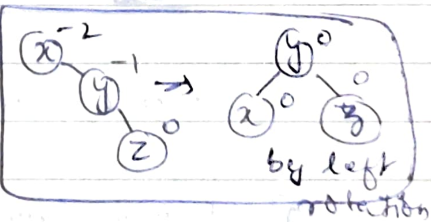
② → Height of left Subtr − Height of right subtr = {−1, 0, +1}.

This is known as balance ← factor.

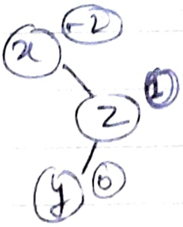
Balance factor = No. of nodes of a nodes. (Children)

Case 1 ⇒ 

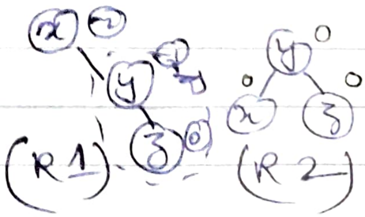
Case 2 

Case 3 

by left rotation

Case 4 ⇒ 

⇒ Simple 1 rotation not possible 2 rotations req ⇒



Case 5 ⇒ Vice Versa of case 4.

Code Remains same for it too

Rotations → LL, RR (single)
LR, RL (double)

Red - Black Trees

(1) ⇒ It is a BST.

(2) ⇒ It is a self balancing tree like AVL, but AVL requires many rotations.

In Red-Black only recolouring & max 2 rotation req.

(3) ⇒ Used when insertion & deletion operations are req frequently.
For Searching req the AVL trees. are best.

RBTr best

Props of RBTr

(1) ⇒ Every node is either Black OR Red.

(2) ⇒ A storage req for storing colour of tree also.

(3) ⇒ Root node is always Black.

(4) ⇒ Every leaf which is Nil is Black.

(5) ⇒ If node is Red, children are Black.

(6) ⇒ Every path from a node to any of it's descendent Nill node has same number of Black nodes.

A AVL tree is a subset of Red-Black Tree.

(7) Extreme (longest) branch should not be > than smallest branch,

Insertion in Red-Black Trees.

step

① ⇒ If tree is empty, create newnode as root node with colour black.

② ⇒ If not empty, create it as Red.

③ ⇒ If parent of newnode is black then exit.

③ ⇒ If parent of newnode is Red, then check colour of parent's sibling.

cases

a) If colour is black (or) null then do suitable rotation and recolour

b) if Red, then recolour & also check if Grand parent of newnode is not root node, if yes, then recolour the parent. of newnode.