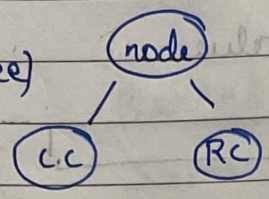
Atul Taneja

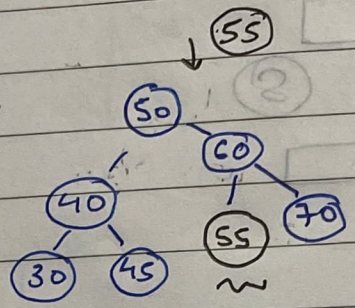
AVL Trees in Java

- BST → data of lc < node, data of Rc > data of node

(Binary Search tree)



- it reduces searching time



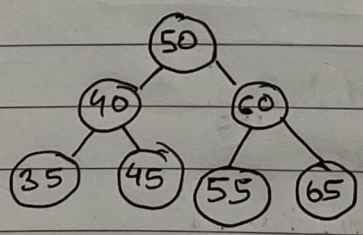
time → • O(h) (order of height)

↓

• O(log n) ← Perfectly Balanced BST

The limitation in BST is we cannot control the order in which data comes.

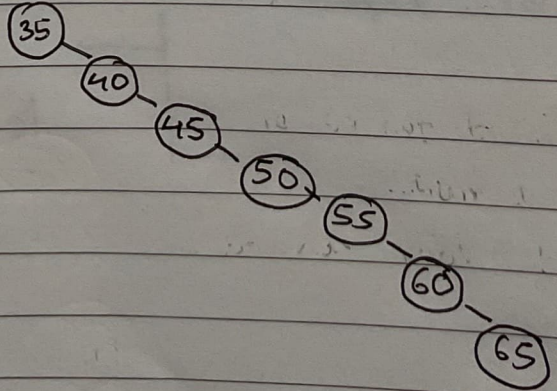
ex:- 50, 40, 60, 35, 45, 55, 65



h; (log n)

O(log n)

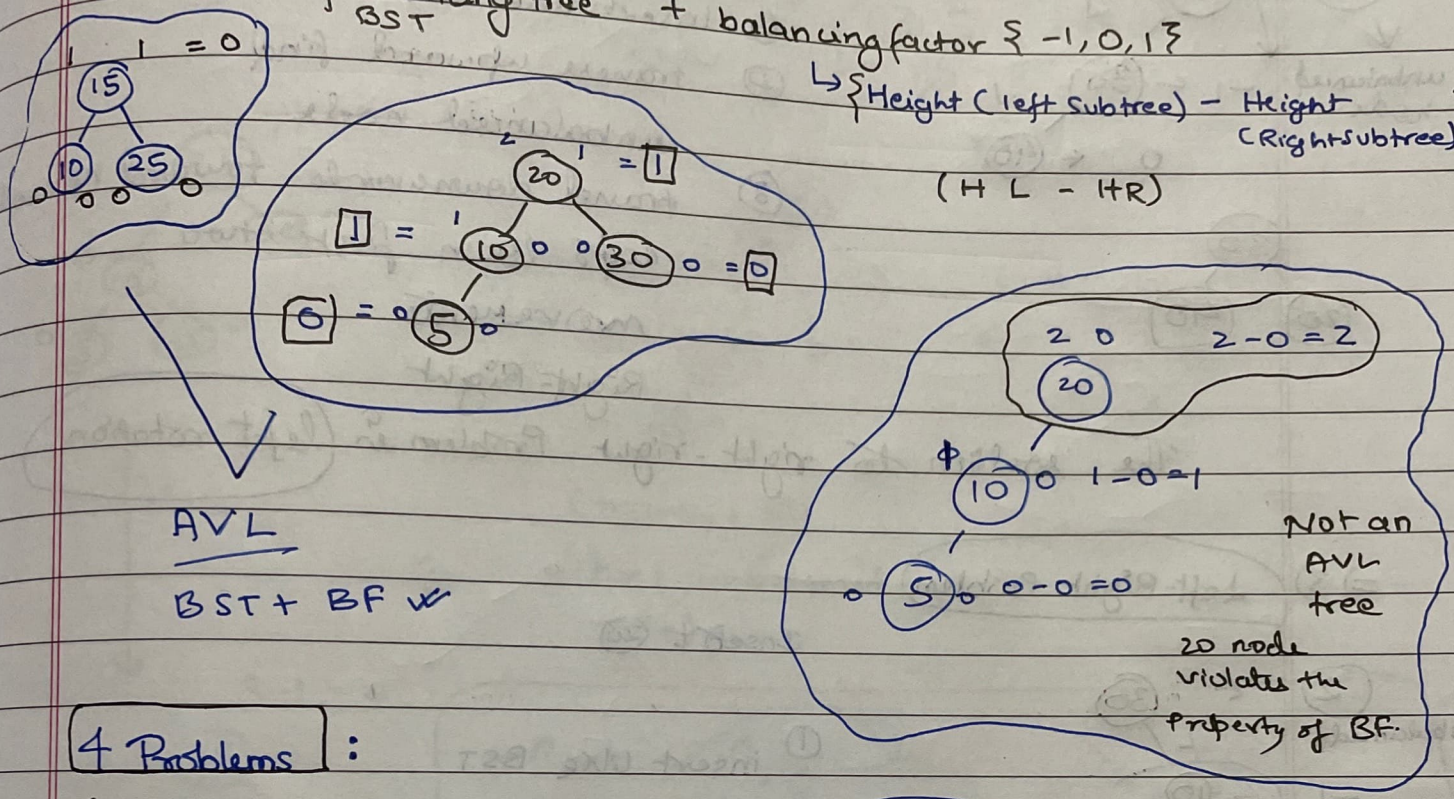
ex:- 35, 40, 45, 50, 55, 60, 65



h : O(n).

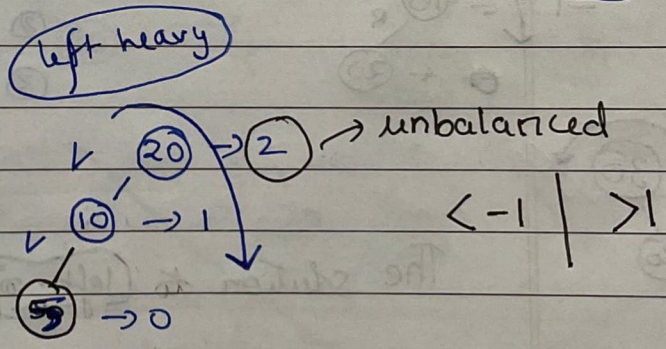
⊙ AVL trees inserts the value in such a way that tree remains balanced.

AVL : Self balancing tree + balancing factor {-1, 0, 1}
BST
↳ {Height (left subtree) − Height (Right subtree)}
(HL − HR)



4 Problems :
(That happen in case of AVL trees)

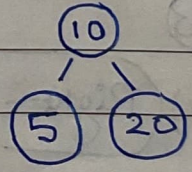
⇒ ① LL → left-left case.



→ insert ⑤

Steps

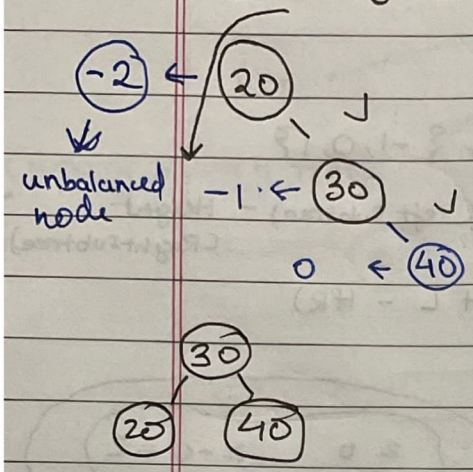
① insert the node in similar fashion the way you used to do in BST.
② traverse upwards till the time you find an unbalanced node.
③ traverse back towards the newly created node and note down the first two movements.



Left-left
Solution to L-L Problem is right rotation

we will rotate the node that has Problem in its balancing factor

⇒ 2) Right - Right Case.



Insert 40

Step

① Insert like BST

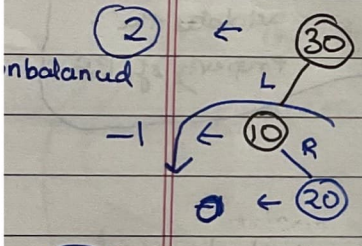
② traverse upward find unbalanced nodes.

③ travel downwards toward newnode ~~node~~ down first two movements

Right-Right

The solution to right-right Problem is left rotation

⇒ 3) Left-Right Problem

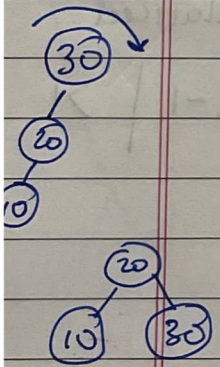


Insert 20

① insert like BST

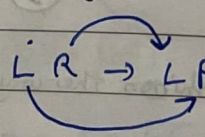
② traverse upwards find BF

③ travel downwards note two movment.

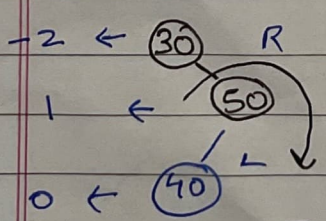


Left - Right

The solution to left-right Problem is left right rotation



⇒ 4) Right - left - Problem

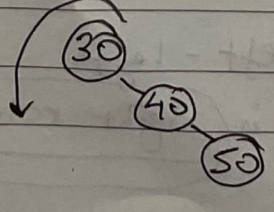


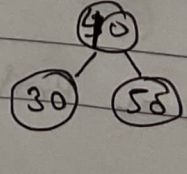
insert 40

The Solution to RL Problem is RL rotation

50 has Problem of left solved by Right Rotation

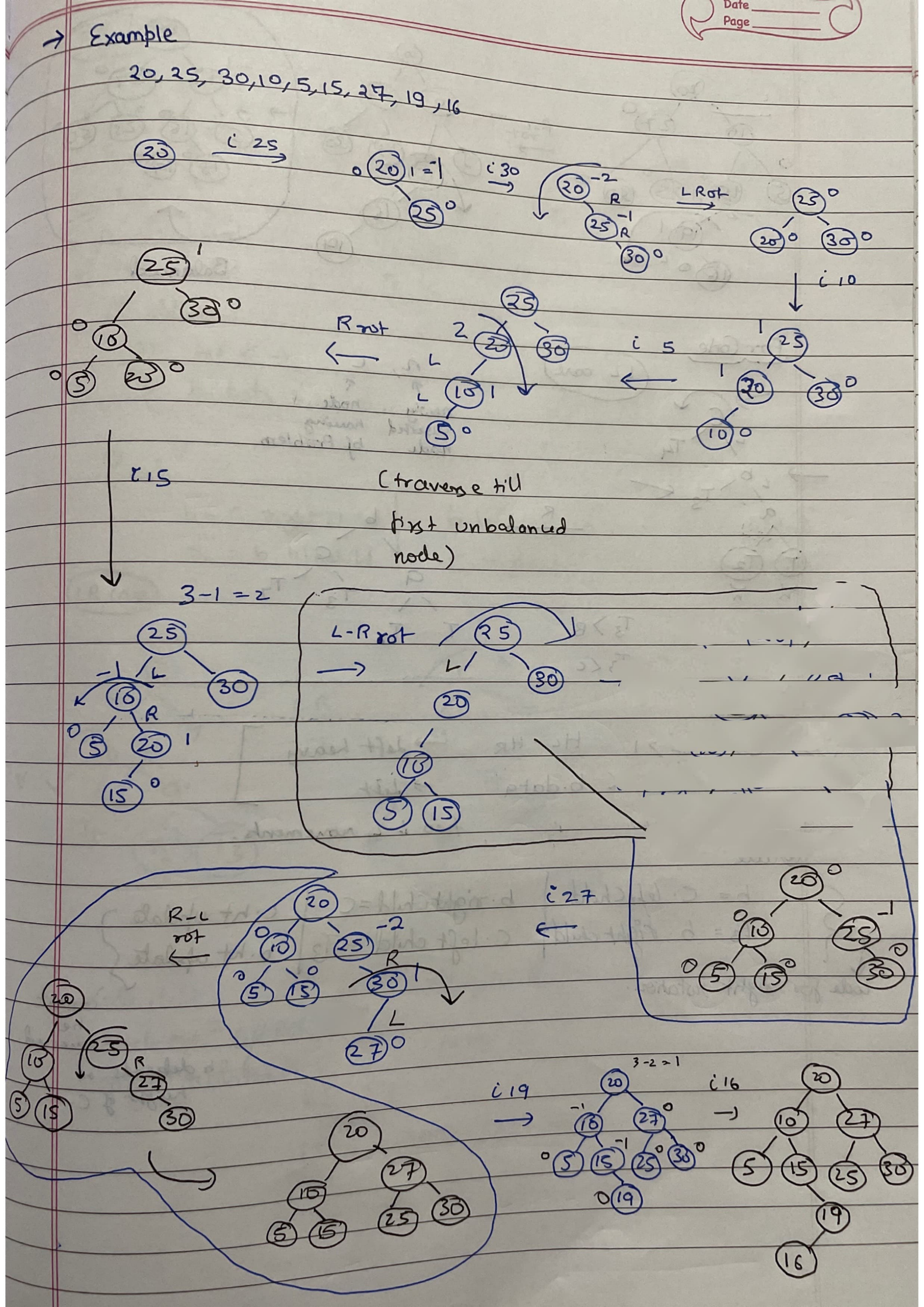
30 has Problem of Right solved by Left Rotation

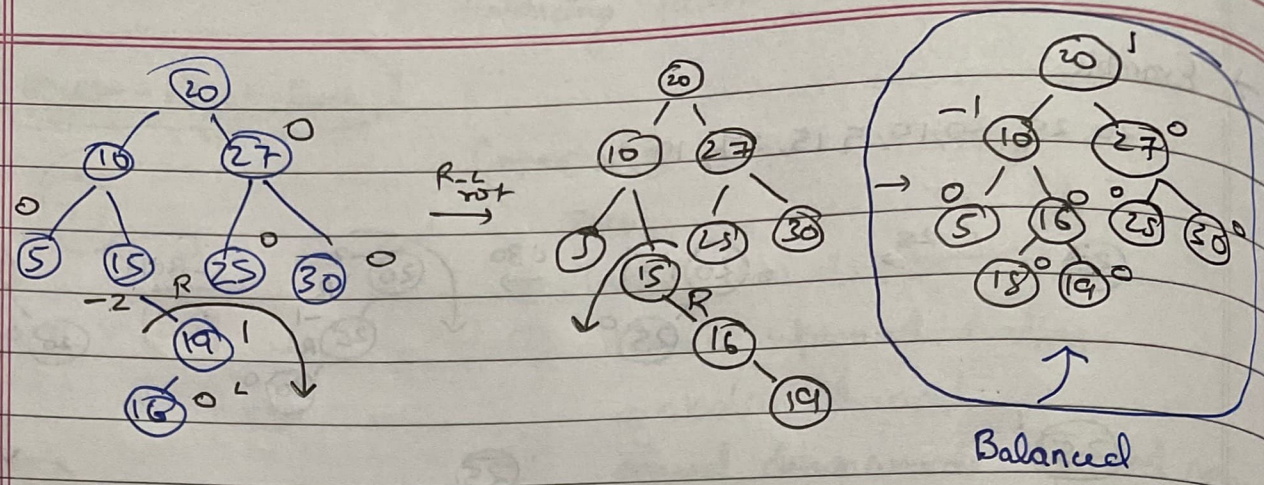




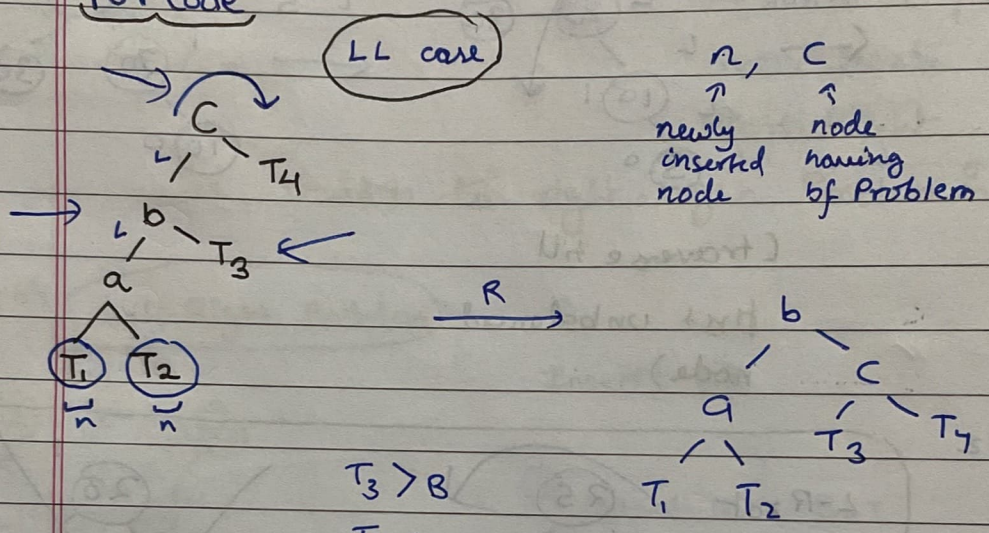
→ Example

20, 25, 30, 10, 5, 15, 27, 19, 16





For Code



n, C
↑ newly inserted node
↑ node having bf Problem

① detect ?

(a) bf > 1 $H_L - H_R$ → left heavy

(b) item < b.data → left

two L-L movements.

② Code

b = C.leftChild. | b.rightChild = C | C.ht update

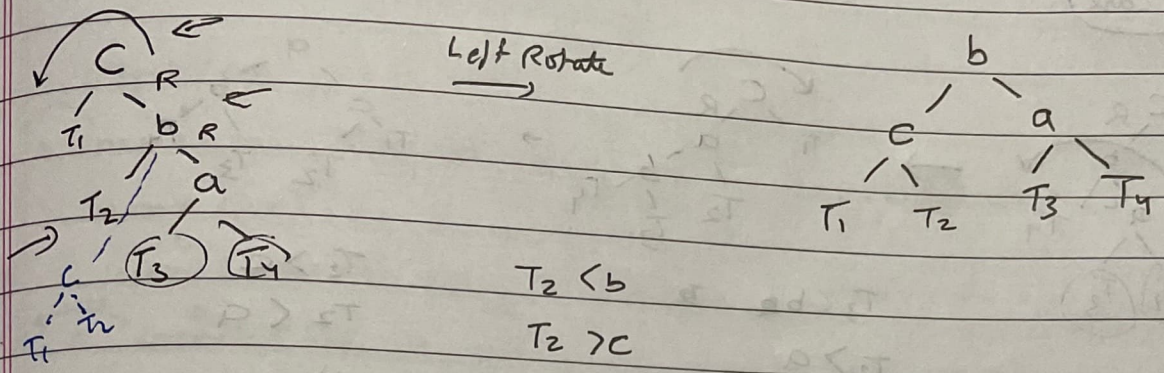
T_3 = b.right child | C.left child = T_3 | b.ht update

code for right rotation.

cannot be removed.

b depends on height of C.

R-R case



$T_2 < b$

$T_2 > c$

① detect ?

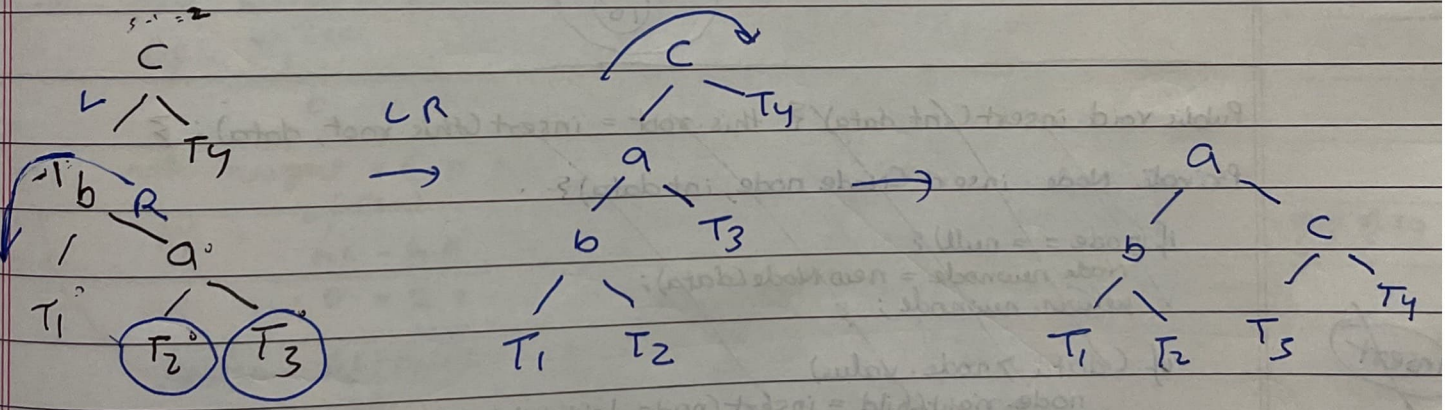
① $bf < -1$

② $b.data < item$

② code

$b = c.rightchild$	$b.leftchild = c$	c.ht update
$T_2 = b.leftchild$	$c.rightchild = T_2$	b.ht update.

LR case



$T_2 < a$

$T_2 > b$

$T_3 \geq a$

$T_3 < c$

node.left

= leftRotate(b)

rightrotate(( );

node.leftchild.

node

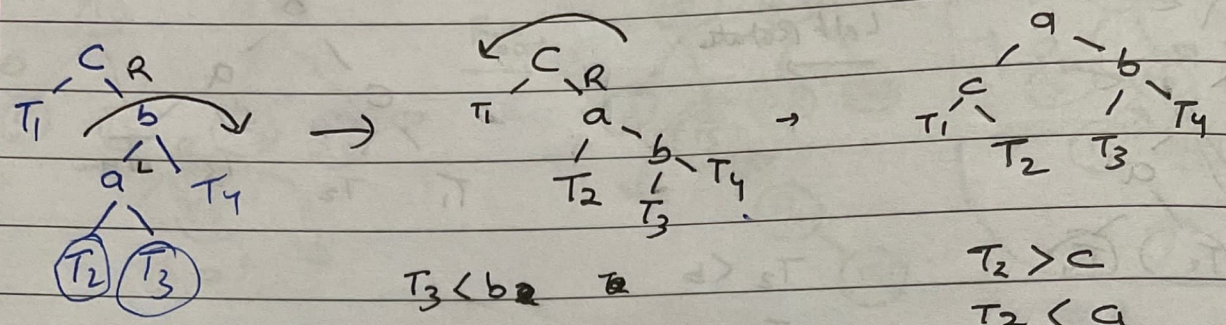
bf > 1

item > node.left.data

if(b<-1 && item < b.data) {

}

→ R-L case:



$T_3 < b$

$T_3 > a$

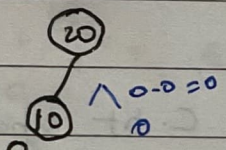
$T_2 > c$

$T_2 < a$

node.rightchild = rightrotate(node.rightchild)

return leftRotate(node).

→ (Q) why we have taken the height of newly created node as 1.



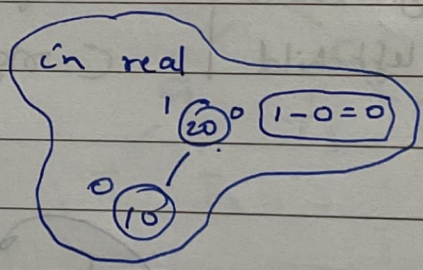
If taken 0

if node == null
ret 0;

1-0=1

bf = height(leftchild) - height(rightchild).

0 - 0 = 0



DRY RUN FOR INSERTION TILL 30

20, 25, 30, 10, 5, 15, 27, 19, 16

insert(20)

root = insert(root=null, 20) node == null newnode(20) return.

insert(25)

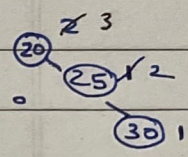
root = insert(root=20, 25) 25 > 20

node.rc = insert(node.rc, 25) node = null newnode(25) return.

node.height = M.Max(0,1)+1 = 1+1 = 2

bf = h(L) − h(R)
0 − 1 = −1

return node;



insert(30)

root = insert(root=20, 30) 30 > 20

node.rc = insert(node.rc, 30) node.rc = 25 != null 30 > 25

node.rc = insert(node.rc, 30) node = null newnode(30) return.

(h of 25) node.height = M.Max(0,1)+1 = 1+1 = 2

bf(25) = hL − hR
0 − 1 = −1

node.height = M.Max(0,2)+1 = 3

bf(20) = hL − hR
= 0 − 2 = −2

← h of Lc of 20 − h of Rc of 20
= 0 − 2

bf < −1 && data > node.right.value
30 > 25.

return leftRotate(node=20)

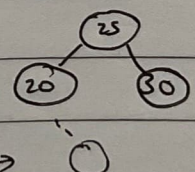
leftRotate(Node c = 20)

b = (Node c = 20.rightc) = 25

t2 = (b.leftchild) = null
25.Lc

b.Lc = (Node c = 20)
25.Lc

(Node c=20.rc) = t2 (=null) →



c.ht = M.Max(hL, hR)+1 = 1
0 0 +1

b.ht = M.Max(1, 1)+1 = 2

return b (Node 25)