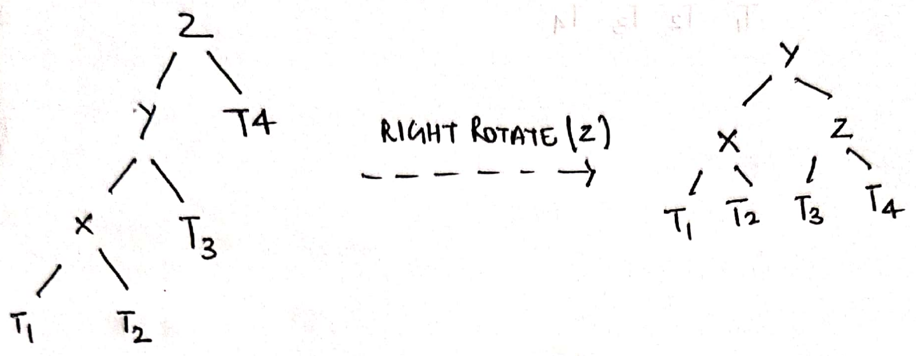
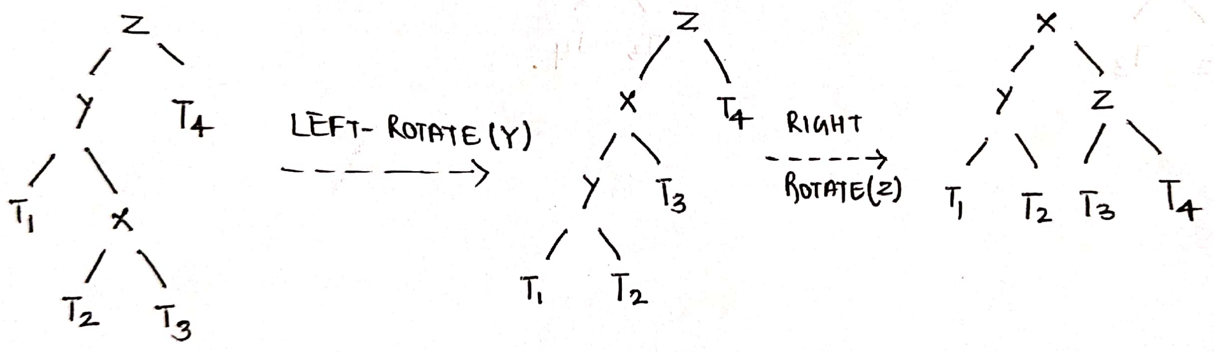
AVL TREE

① Left-Left Case.

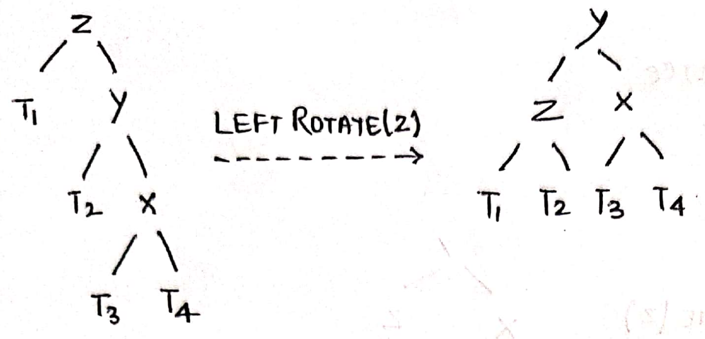
T₁, T₂, T₃ and T₄ are sub-tree



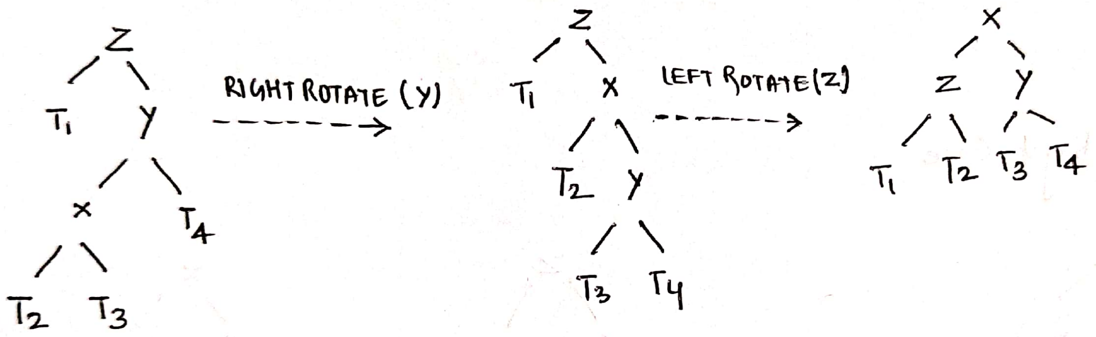
(2) Left-Right Case



(3). RIGHT-RIGHT CASE

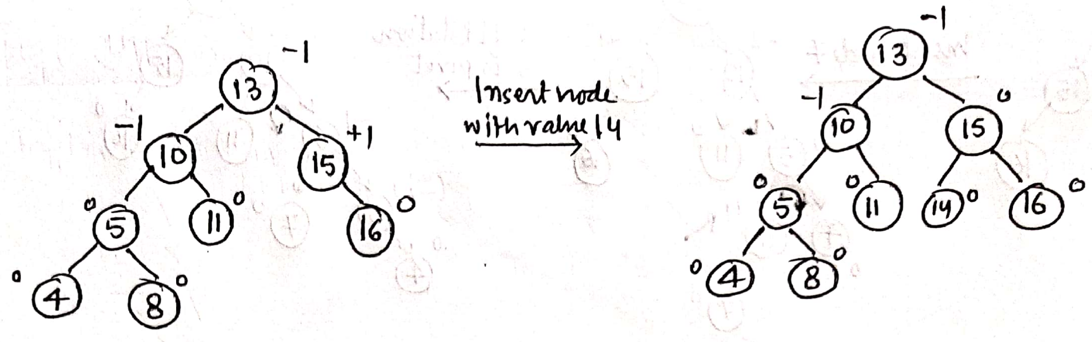


(4). RIGHT-LEFT CASE

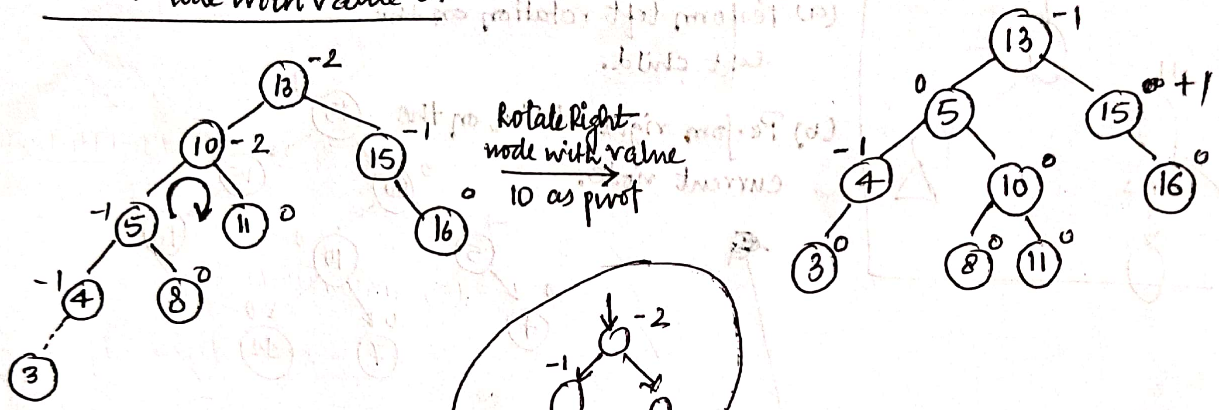


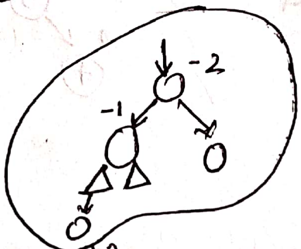
INSERTION IN AVL TREE

a.

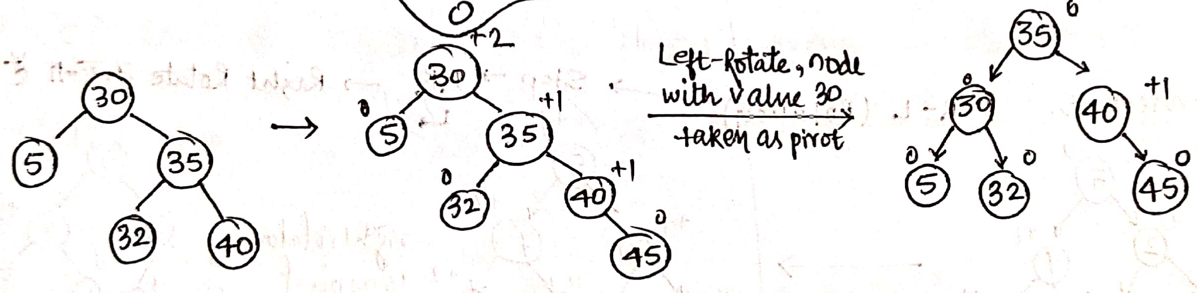


b. Insert node with value 3.

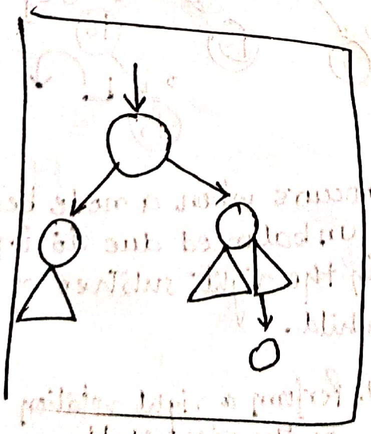




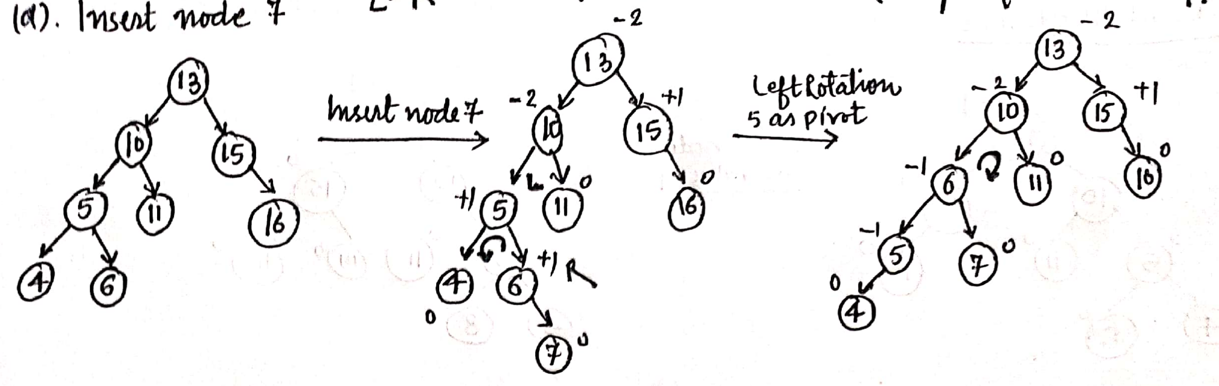
(c.)



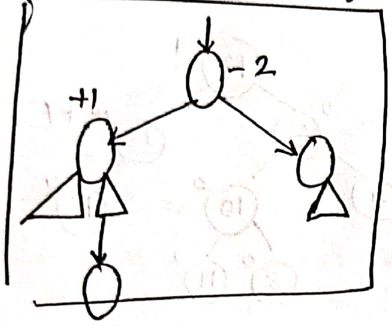
Insert 45



(d). Insert node 7 L-R → Left Rotate पहले फिर Right Rotate बाद में।

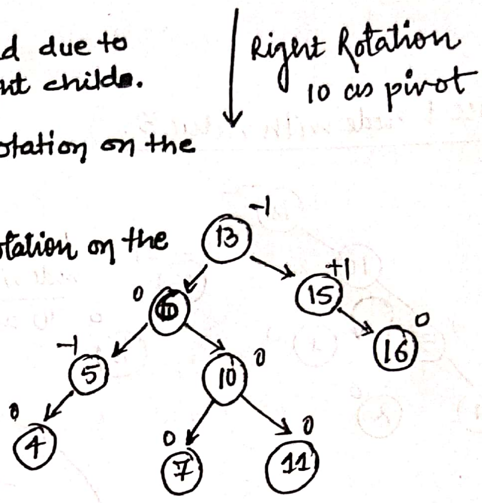


→ occurs when a node becomes unbalanced due to insertion in the left subtree of the right child.

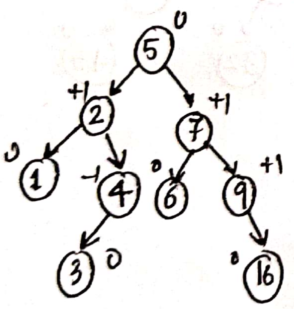


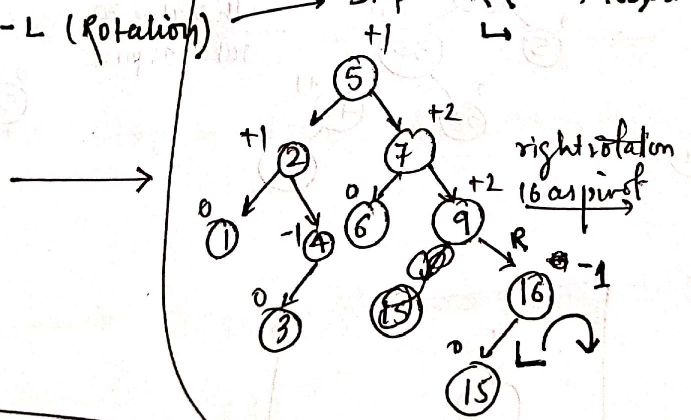
(a) Perform left rotation on the left child.

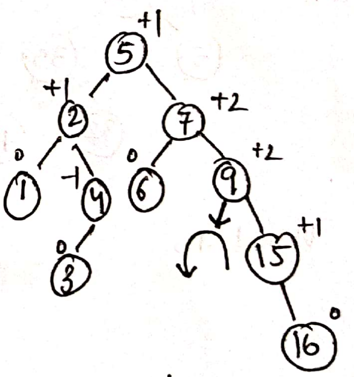
(b) Perform right rotation on the current node.

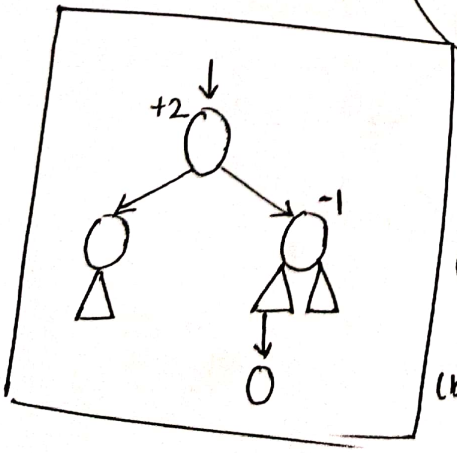


(e). Insert 15. R-L (Rotation) → Step → RR ↳ → Right Rotate करना है





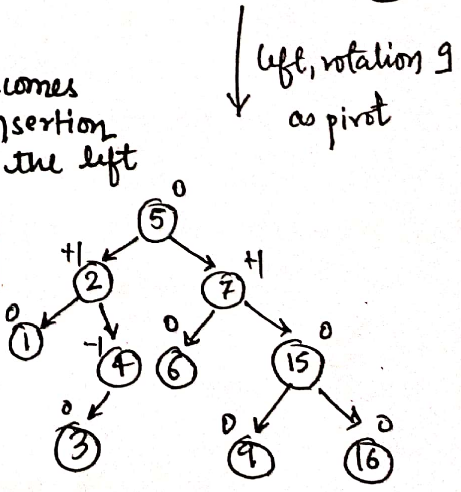




→ occurs when a node becomes unbalanced due to insertion in the right subtree of the left child.

(a). Perform a right rotation on the right child.

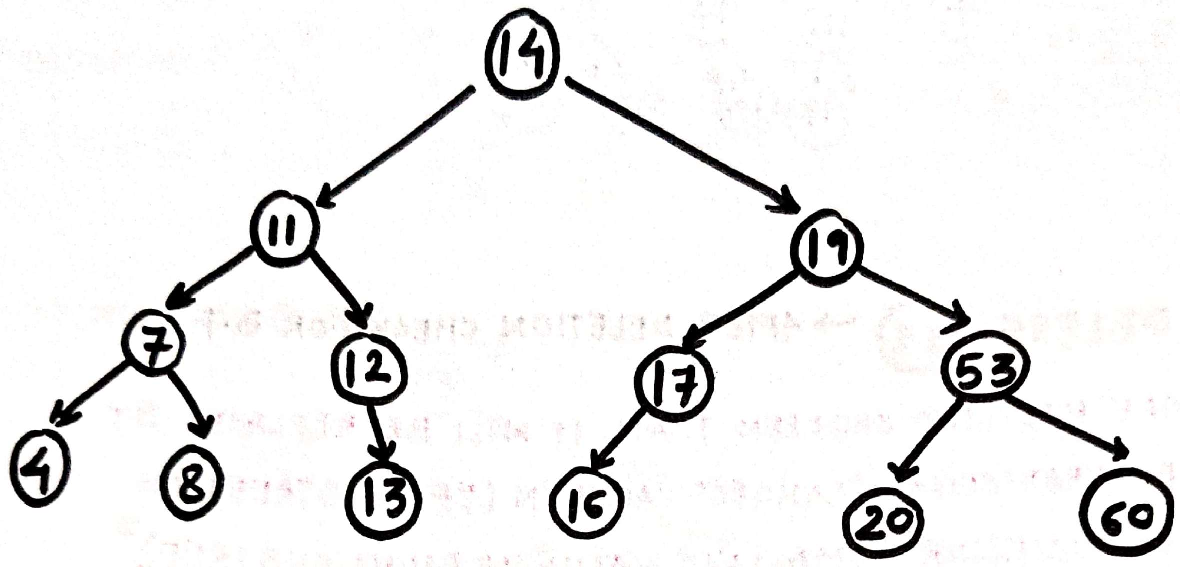
(b). Perform a left rotation on the current node.



DELETION IS SAME AS BINARY SEARCH TREE.

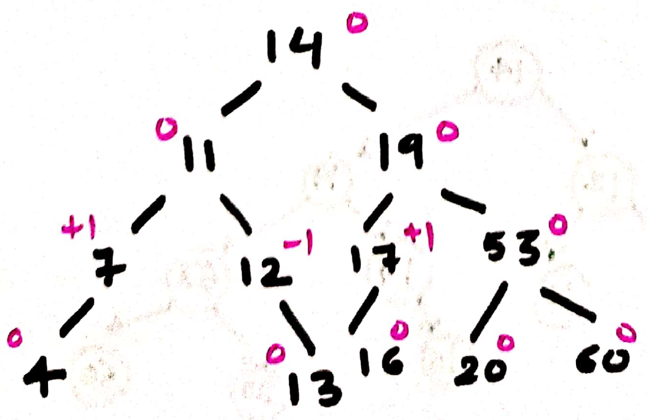
Deletion in AVL Tree.

8, 7, 11, 14, 17.



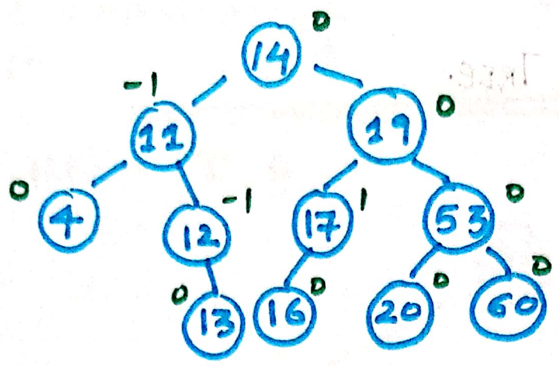
ONLY CHANGE FROM BST DELETION – IS THAT YOU HAVE TO CHECK FOR BALANCE FACTOR.

• FIND ⑧ ⟶ Delete it → Leaf Node.



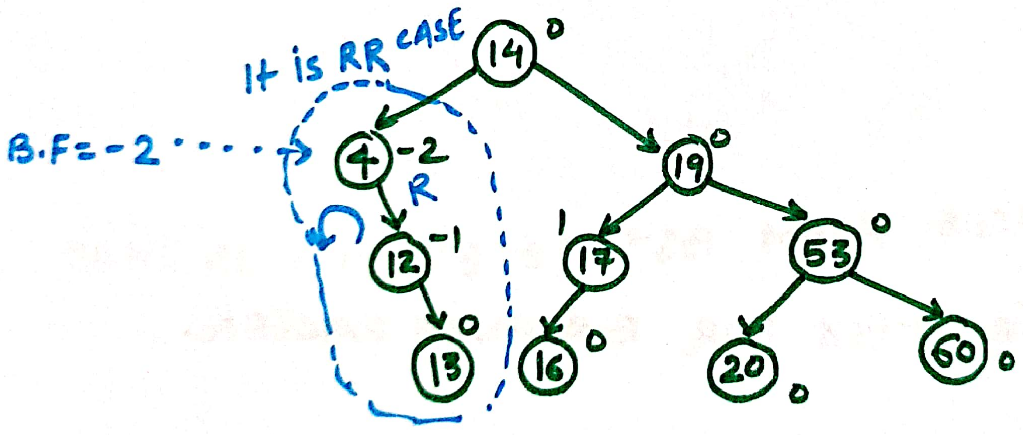
NOW CHECK FOR B.F OF EACH NODE AGAIN

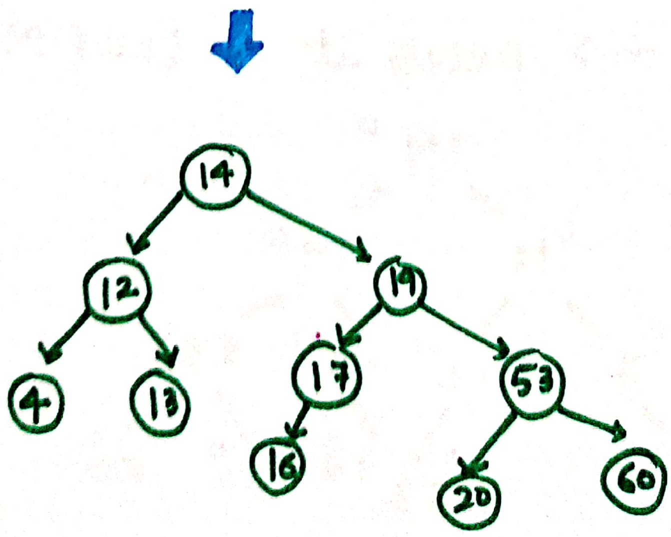
• NOW DELETE ⑦ → CHECK FOR B.F after deletion of ⑦.



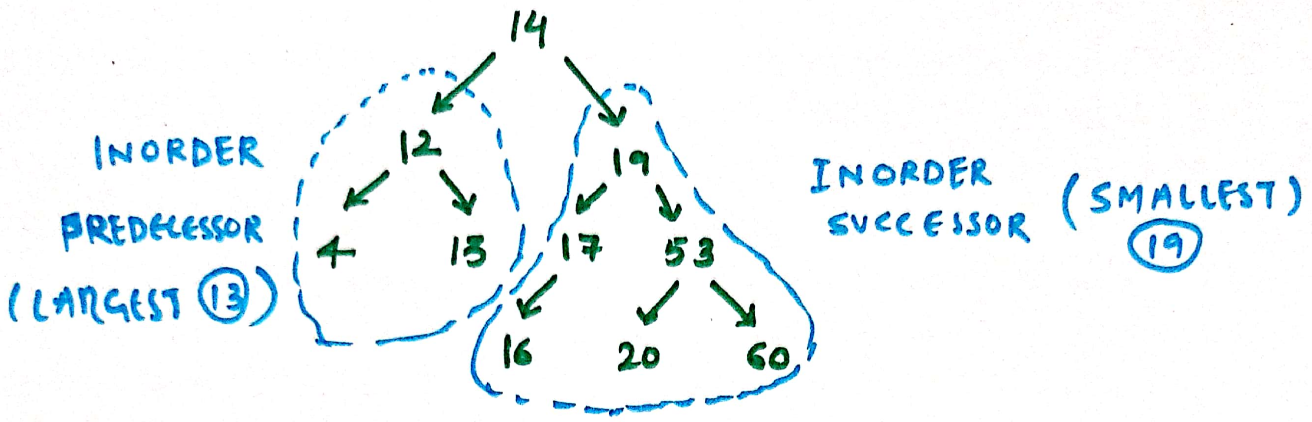
• NOW DELETE ⑪ → AFTER DELETION CHECK FOR B.F.

IF NODE HAS TWO CHILDREN THAN, IT WILL BE REPLACED BY INORDER PREDECESSOR (LARGEST VALUE IN LEFT SUBTREE) OR INORDER SUCCESSOR (SMALLEST VALUE IN RIGHT SUBTREE).



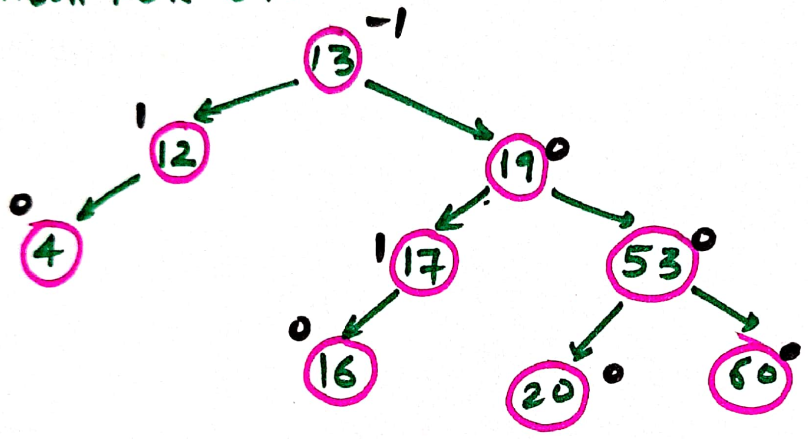


· NOW DELETE $\textcircled{14}$ NODE



14
12
4
13
19
17
53
16
20
60
INORDER PREDECESSOR (LARGEST 13)
INORDER SUCCESSOR (SMALLEST) 19

NOW CHECK FOR B.F.



13 -1
12 1
4 0
19 0
17 1
53 0
16 0
20 0
60 0