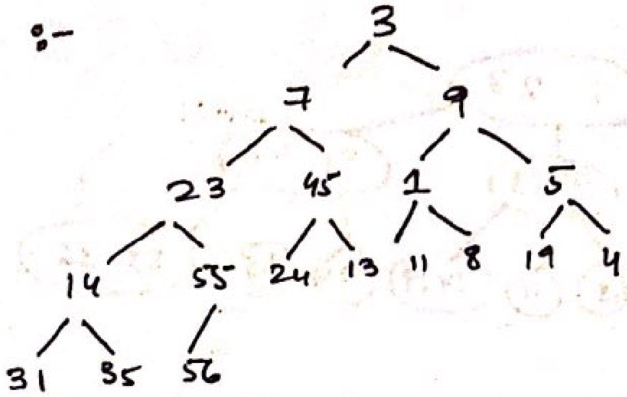
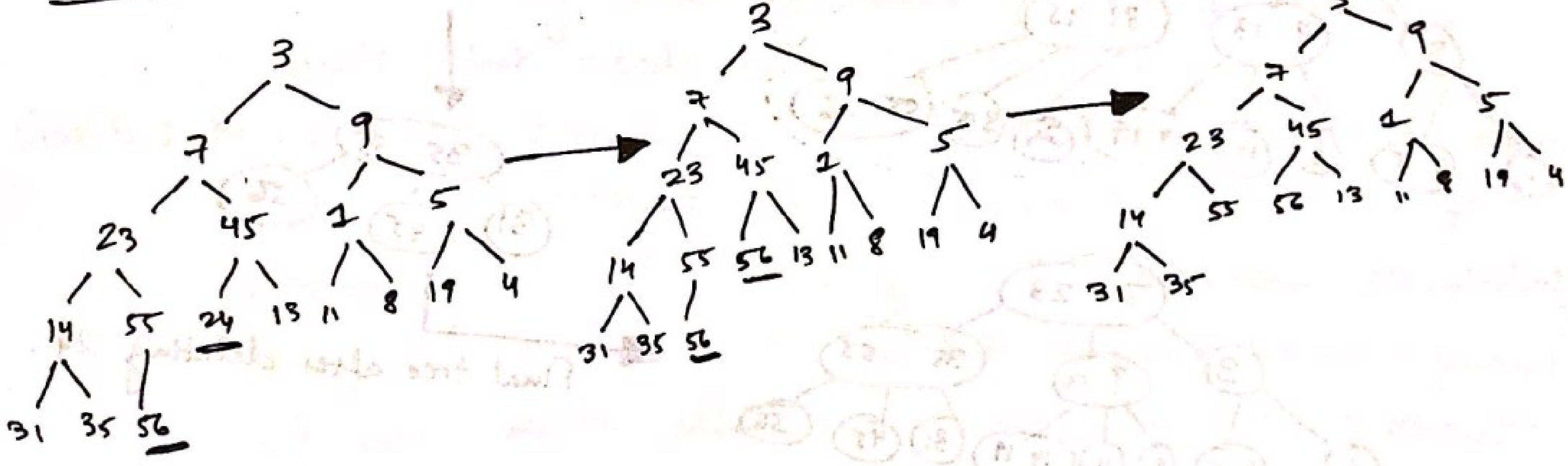
Binary Tree :-



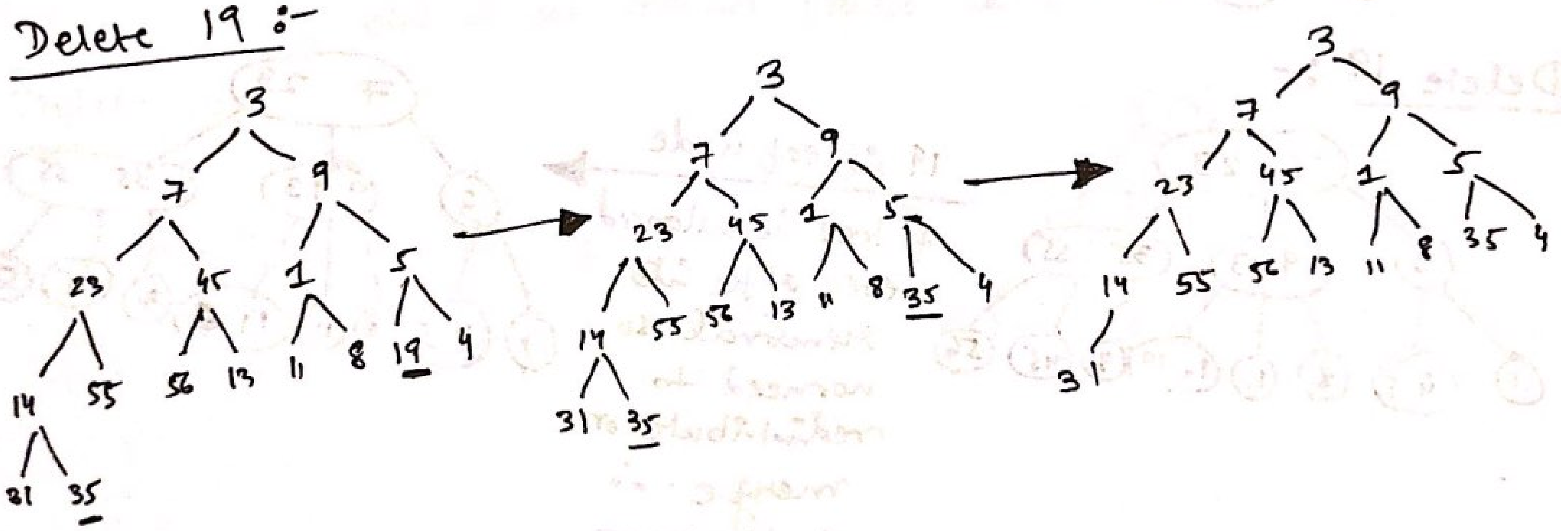
Delete Algo :-

1) Start at root node, find deepest rightmost node and node that we want to delete.
2) Replace deepest rightmost node's data with node to be deleted.
3) Then delete deepest rightmost node.

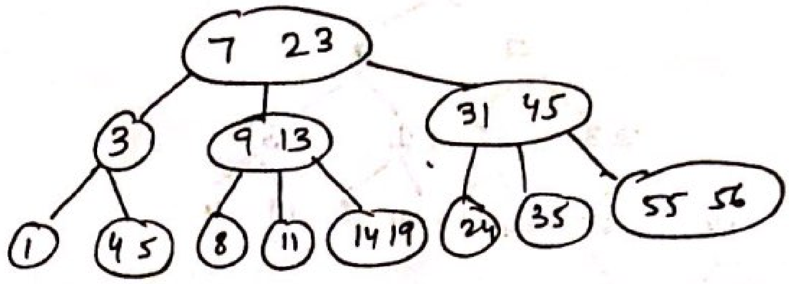
Delete 24 :-



Delete 19 :-



2-3 Tree :-

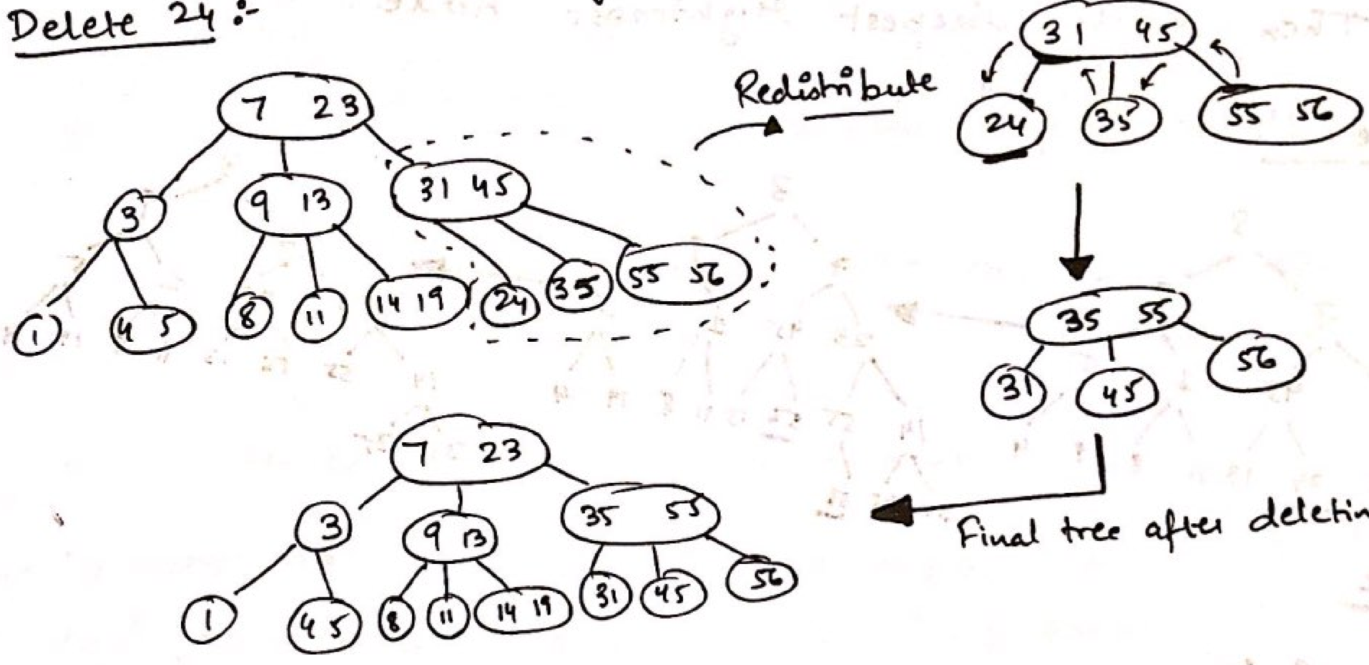


Delete Algorithm :-

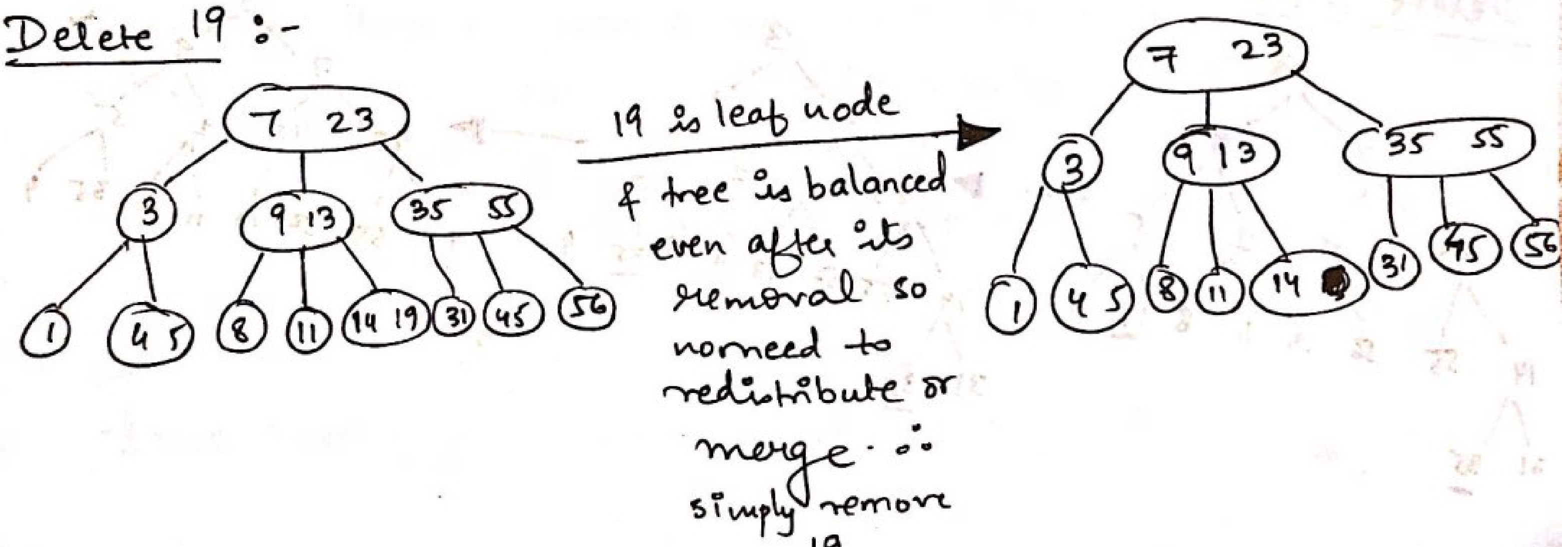
If not a leaf node ~~[illegible]~~.

1) Swap node to be deleted with its inorder successor ~~[illegible]~~.
2) Delete value from leaf & then merge nodes by deleting empty leaf & moving other value down or re-distribute nodes from siblings if it contains another item.

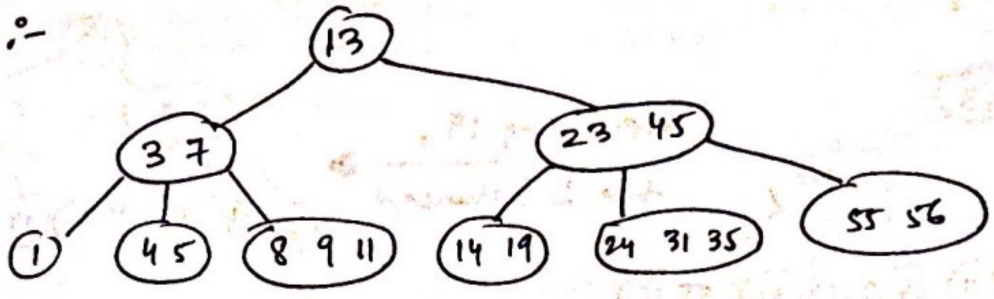
Delete 24 :-



Delete 19 :-


19 is leaf node
& tree is balanced even after its removal so noneed to redistribute or merge ∴ simply remove 19

2-3-4 Tree :-



Delete Algorithm :- Similar to 2-3 tree deletion algorithm.

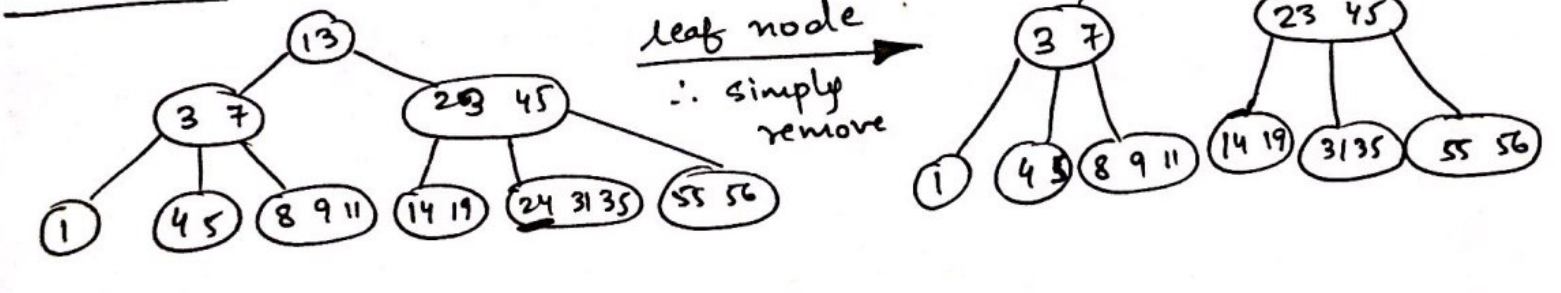
Case 1 : Node having more than 1 element, remove it.

Case 2 : Node has 1 element but immediate sibling has more than 1 element, then borrow 1 key from sibling & re-distribute to delete node that has to be removed

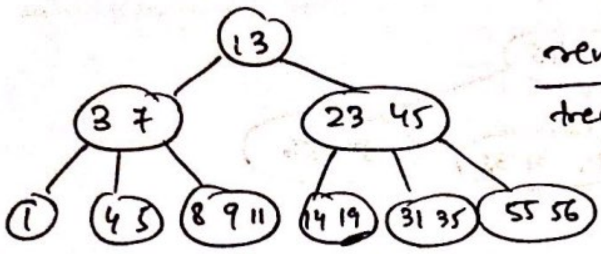
Case 3 : Node has 1 element but no immediate sibling have more than 1 element, then steal 1 element from parent (if it has more than 1 element), merge itself with sibling sharing same parent & remove node that needs to be deleted.

Case 4 : Node has 1 element, immediate siblings have 1 element and even parent has 1 element, then merge the parent with its siblings & grandparent recursively. If total elements is 3, then it should be new parent. If not, ~~do~~ re-distribute/rotate on its parent & delete as other cases above.

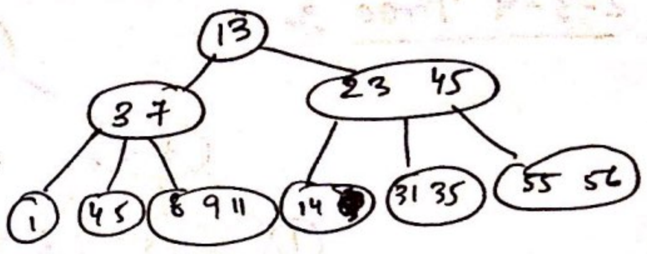
Delete 24 :-



Delete 19 :-



removing 19,
tree is balanced



13
3 7
23 45
1
4 5
8 9 11
14
31 35
55 56