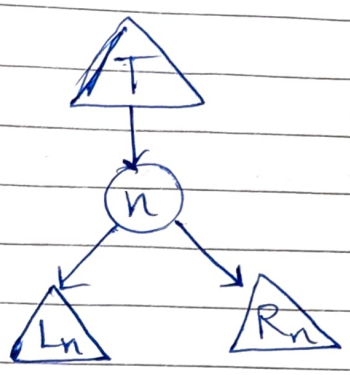
Aniket Gupta
2022073

Rush
PAGE NO.:
DATE: / /

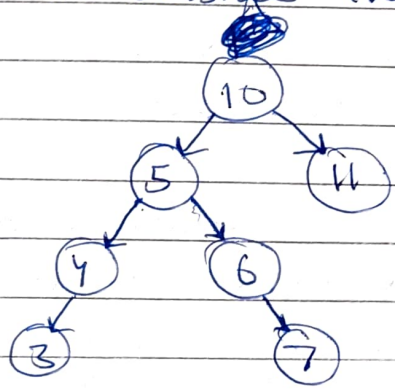
Homework-8

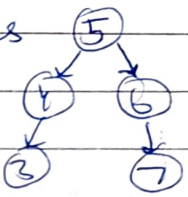
Q1.



→ given, node n is the part of an AVL tree and is imbalanced (left heavy)
→ L_n and R_n are AVL trees
→ left child of n is not heavy (L_n)

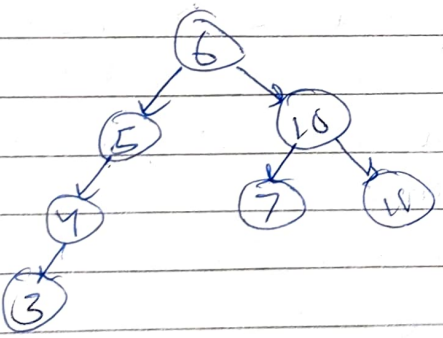
let us consider the following as example:



→ node 10 is imbalanced (left heavy)
→ subtrees  and (11) are AVL trees
→ left child of 10, i.e 5, is not heavy

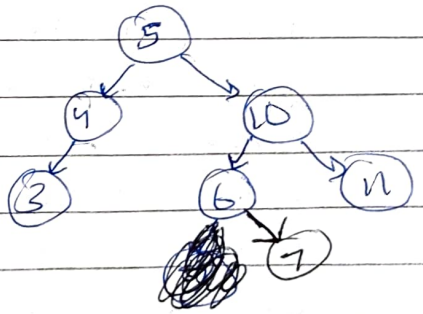
~~Performing LR rotation we get~~

LR rotation



node 5 is imbalanced, hence not an AVL tree

LL rotation



all nodes are perfectly balanced, hence it is an AVL tree

20

Q2. Given: every node in the binary tree has either zero children and n is the total no. of leaf nodes

To prove: total no. of internal nodes is n-1

Proof

Base case: if n=1, tree consists of only one node, i.e a leaf node
∴ there are no internal nodes
hence, the base case is satisfied

Induction hypothesis: let us assume that the statement is true for all trees with $n \leq k$ leaf nodes, where k is an arbitary positive integers.
∴ A tree with k leaf nodes has k-1 internal nodes.

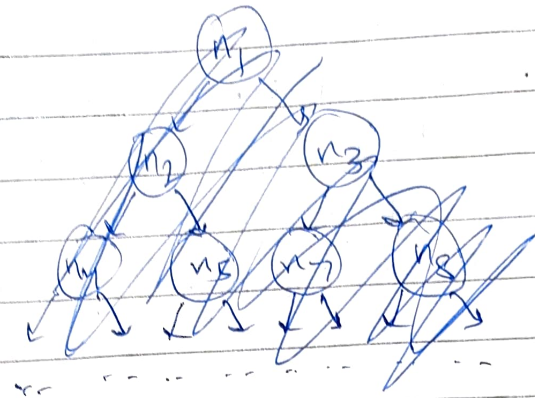
Induction step: we need to prove that the statement is true for a tree with k+1 leaf nodes

10

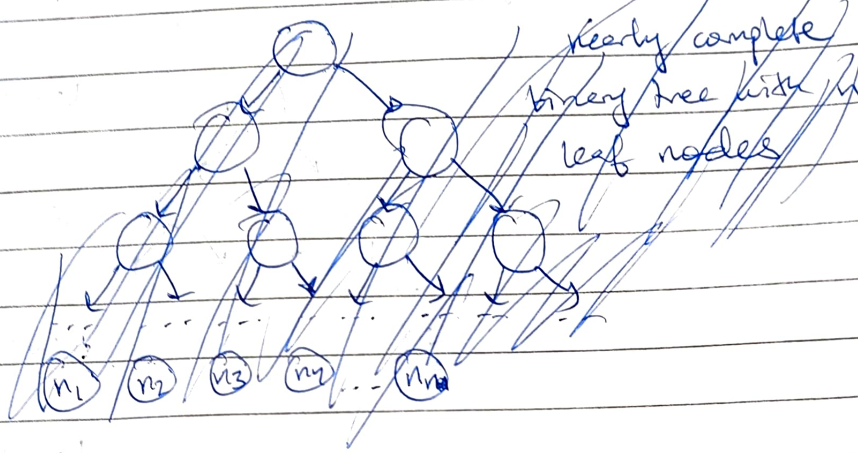
Let T be a tree with k+1 nodes. Consider a leaf node n whose parent is p. Now, we are given that every node in the binary tree has either zero or two children, hence p must contain another child, say m. Now, if we detach both m and n, we are left with a tree (say T') with k-1 internal nodes (since p now becomes a leaf node). Hence, using our induction hypothesis, T' would have k leaf nodes and p is one such leaf node. If we restore m and n, we again get T with k internal nodes. The no. of leaf nodes in T would be k+2-1 since m and n got added while p goes back to be an internal node. ∴ T has k+1 leaf nodes. Hence proved.

[illegible]

Q2.

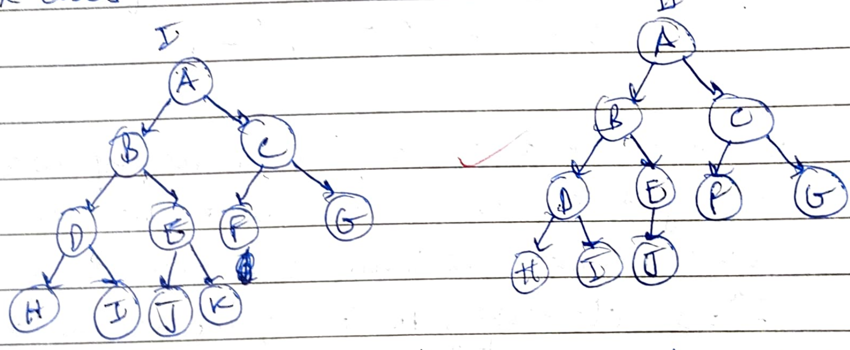


Q3.



let T be a ~~easy~~ neerly complete binary tree.

2 cases :



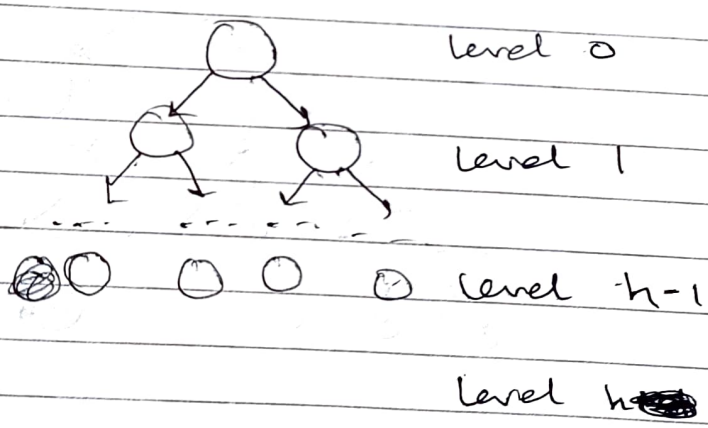
case I is like Q2. where every node has either 2 or 1 children, ∴ no. of internal nodes is always n-1 where n is no. of leaves.

P.T.O

In case II, if (J) is deleted, (E) changes from internal node to a leaf node. Hence, no. of leaves remain n and the tree resembles the tree in Q2. with n-1 internal nodes. As we restore (J), (E) changes from leaf to internal node. Hence, no. of leaves remain same, but internal nodes increase by 1. ∴ In this case, internal nodes are n.

10

Q4. a $\hat{}$ ^{max} heap of height 'h':



first, we would calculate no. of internal nodes
$= 2^0 + 2^1 + \dots + 2^{h-1} = 2^h - 1$

5

we know that a heap is a nearly complete binary tree, this means that the last level needs to have minimum one element and maximum 2^h elements. Number of internal nodes would remain same for both cases

∴ min. no. of nodes $= (2^h - 1) + 1 = \boxed{2^h}$

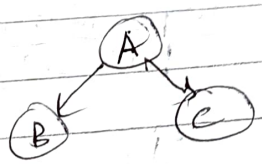
max. no. of nodes $= (2^h - 1) + 2^h = \boxed{2^{h+1} - 1}$

Rysh

PAGE NO.:
DATE: / /

25. → In preorder traversal, we first print the root node, then recursively travel the left subtree, and then recursively travel the right subtree

→ In inorder traversal, we recursively travel the left subtree, then print the root node, and then recursively travel the right subtree

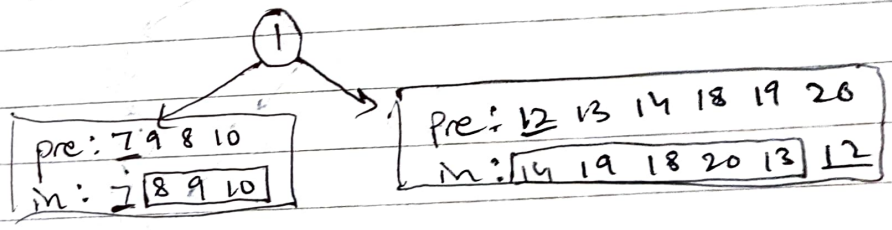


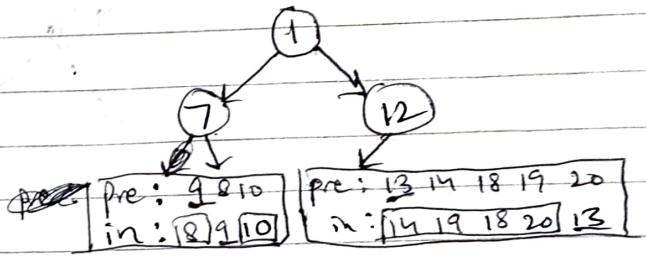
preorder : ABC → first node is root
inorder : BAC

left subtree | right subtree

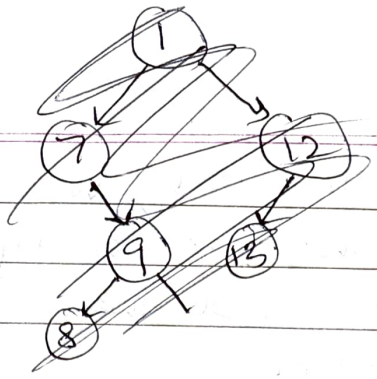
we can identify the root using preorder traversal and we can identify the left and right subtrees using inorder traversal

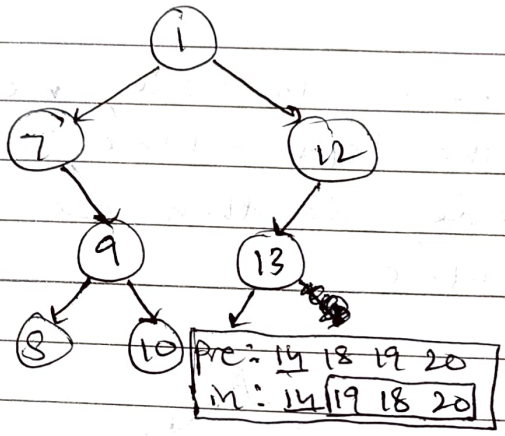
preorder : 1 | 7 9 8 10 | 12 13 14 18 19 20
inorder : 7 8 9 10 | 1 | 14 19 18 20 13 12



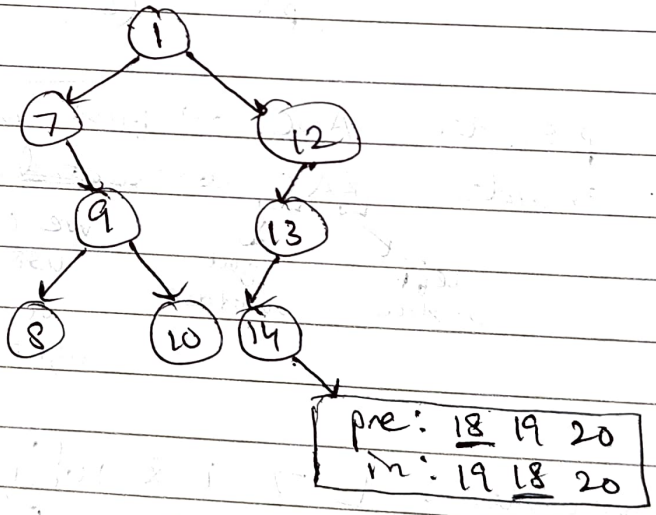


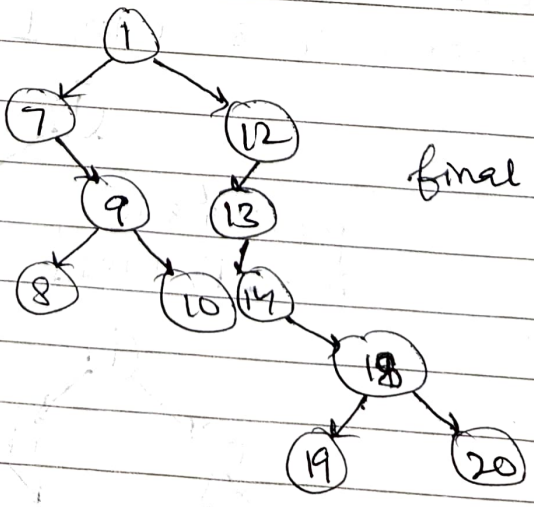
P.T.O





15





final binary tree