a)

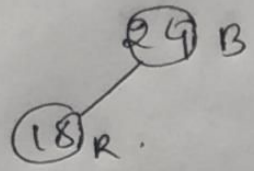
0 -50
1 -17
2 -12
3 -7
4
5 -35
6 -46
7 -45
8 -26
9 -55

b)

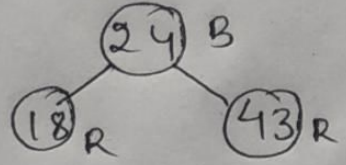
0 -50
1 -7
2 -2
3 -55
4 -26
5 -35
6 -46
7 -17
8
9 -45

2) insert 24

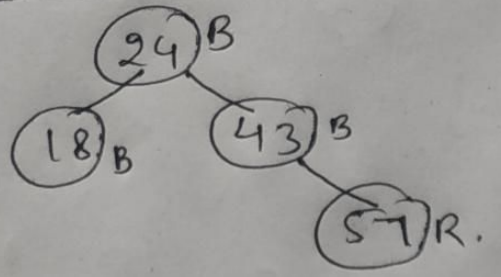
insert 18



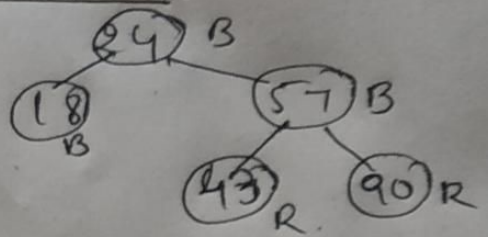
insert 43



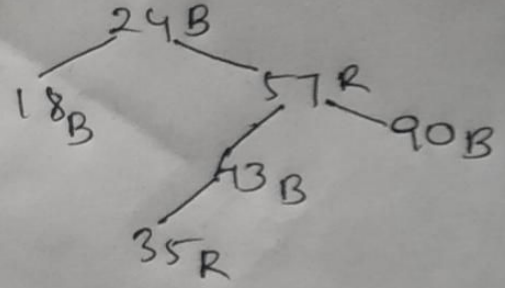
insert 57



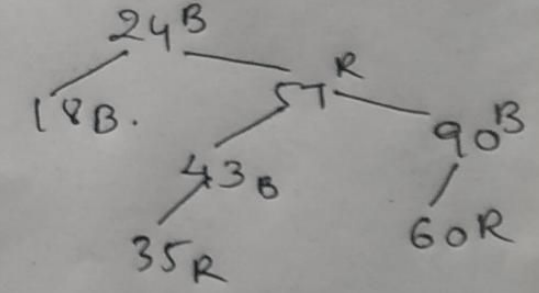
insert 90

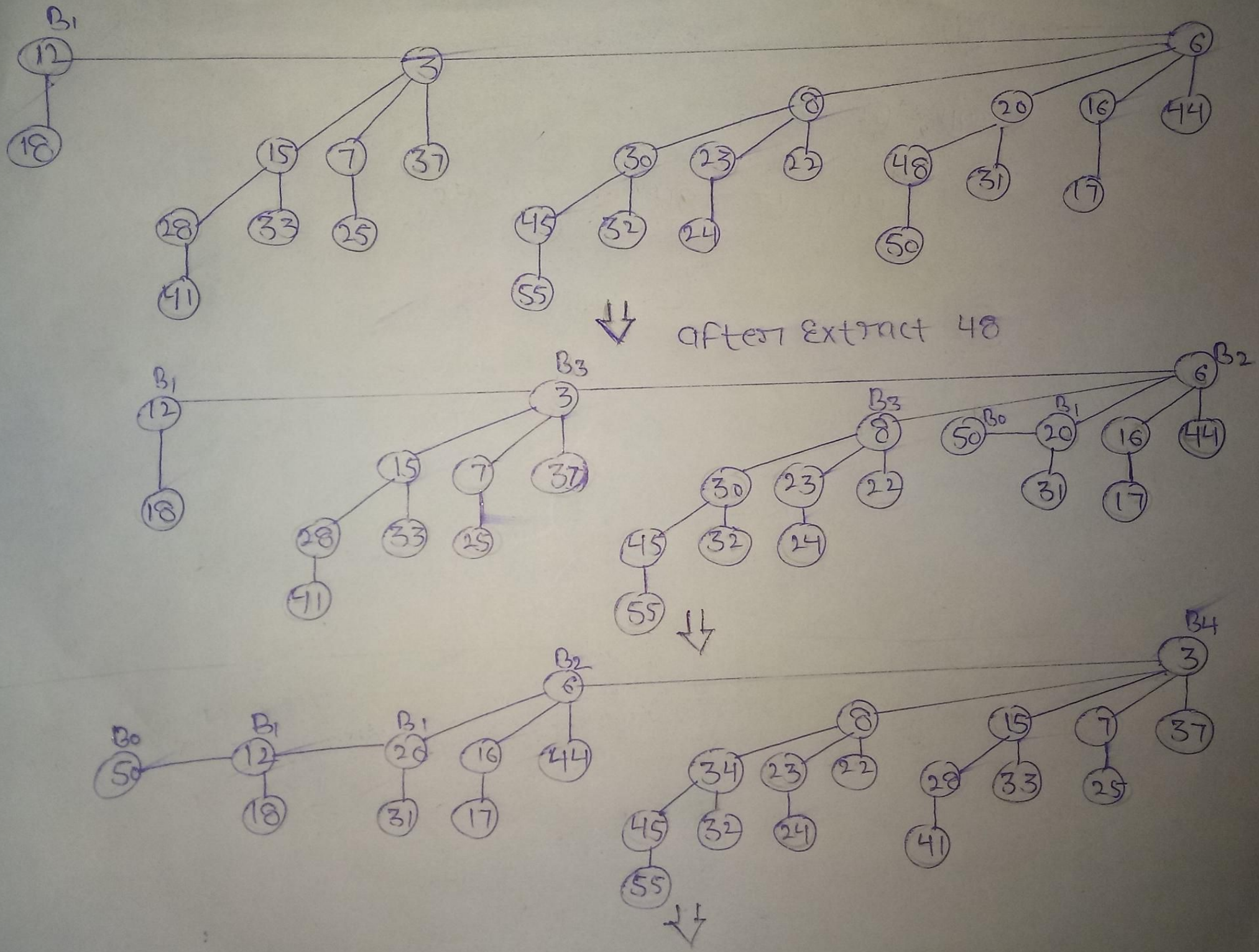


insert 35



insert 60




B1
12
3
6
18
15
7
37
8
20
16
44
28
33
25
30
23
22
48
31
17
41
45
32
24
50
55
after extract 48
B3
B2
B1
B3
B0
B1
B4
B2
B0
B1
B1

Answer :-

There are mainly two major parts in Huffman coding

1) Build a Huffman Tree from input characters.
2) Traverse the Huffman Tree & assign codes to characters.

Steps to build Huffman Tree

→ Input is an array of unique characters along with their frequency of occurences and output is Huffman Tree.

1) Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially the least frequency character is at root.

2) Extract two nodes with the minimum frequency from the min heap.

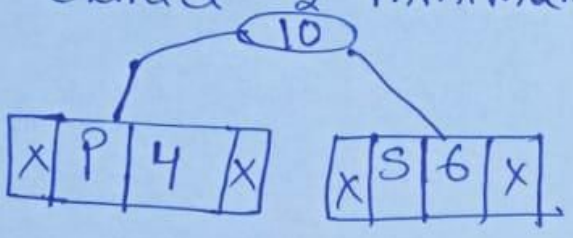
3) Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its ~~left~~ right child and the other extracted node as its right child. Add this node to the min heap.

4) Repeat steps #2 & #3 until the heap contains only one node. The remaining node is the root node & the tree is complete.

Character	Frequency
P	4
S	6
M	8
Y	9
K	12
E	14
A	16

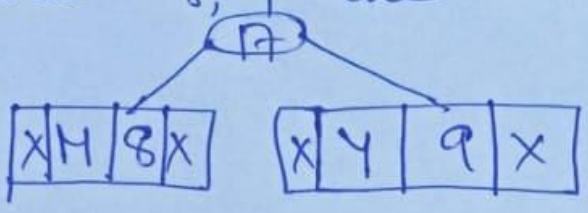
P	S	M	Y	K	E	A
4	6	8	9	12	14	16

→ Extract 2 minimum nodes



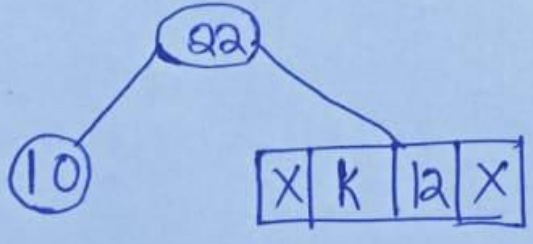
(10), 8, 9, 12, 14, 16

→ Now 8, 9 are 2 minimum nodes

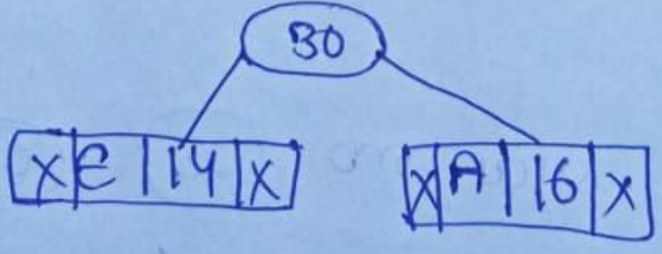


(10), (17), 12, 14, 16

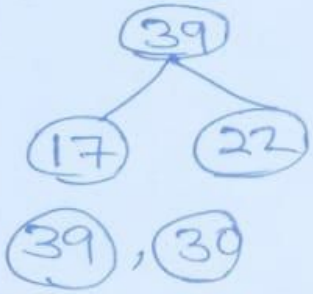
→ Now (10), 12 are minimum nodes



→ Now (17), (22), 14, 16 are minimum



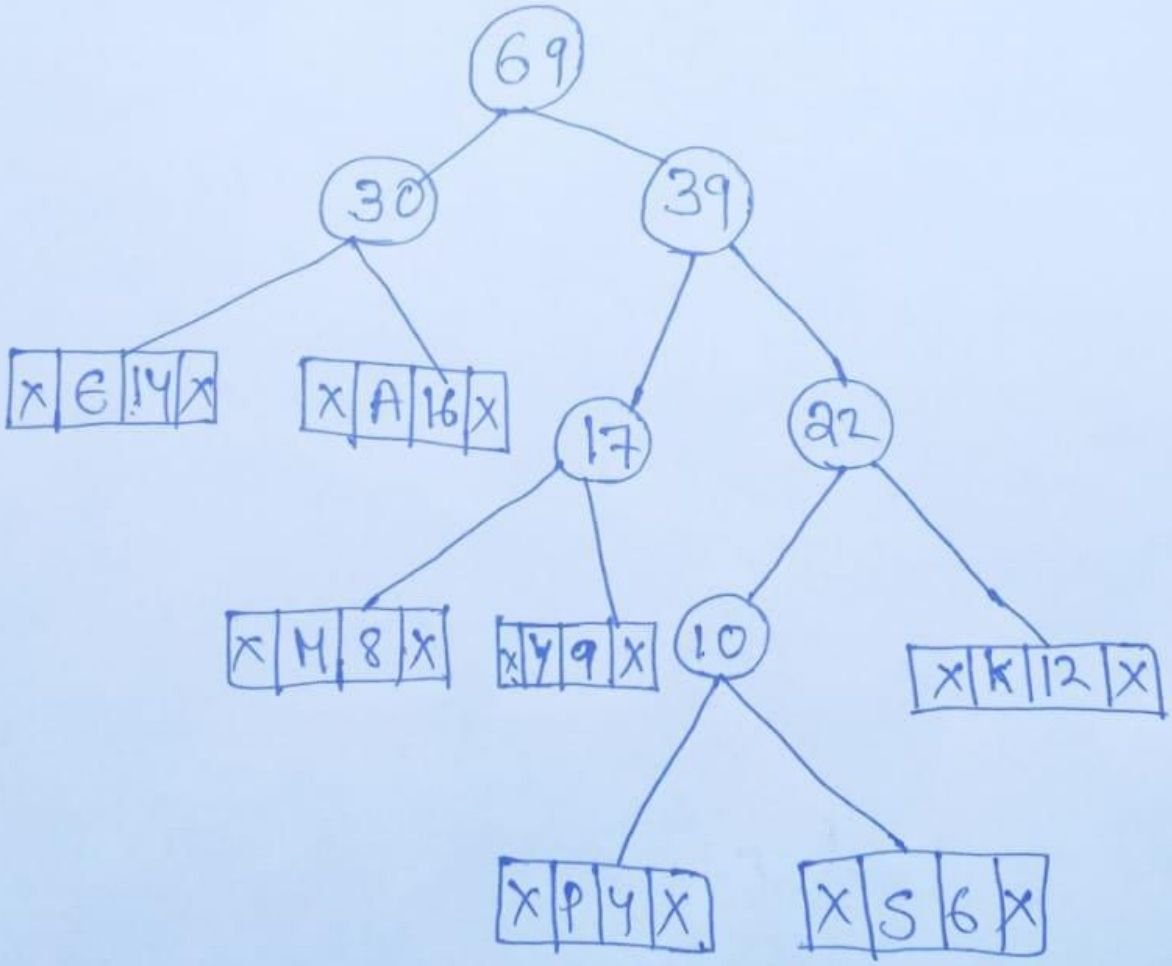
→ Now (17), (22), (30) are minimum nodes are (17) (22)



(39), (30)

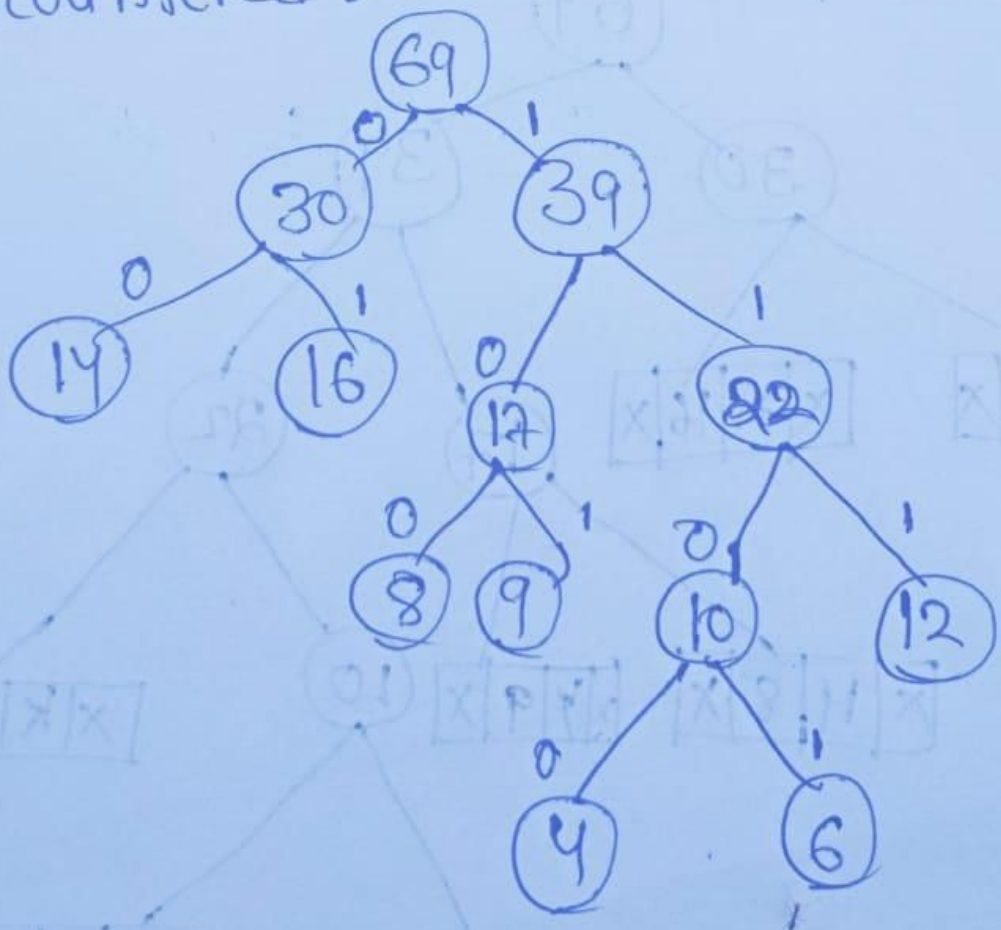
→ Now add (39), (39)

→ The final Huffman Tree is



Steps to print codes from Huffman Tree

Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child write 0 to the array. While moving to the right child write 1 to the array. Print the array when a leaf node is encountered.



Pre-Order Traversal: In this traversal,

1. Visit Root Node

2. Visit Left Subtree and

3. Visit Right Subtree

Post-Order Traversal: In this traversal,

1. Visit Left Subtree

2. Visit Right Subtree and

3. Visit Root Node

In-Order Traversal: In this traversal,

1. Visit Left Subtree

2. Visit Root Node and

3. Visit Right Subtree

Given Problem is:

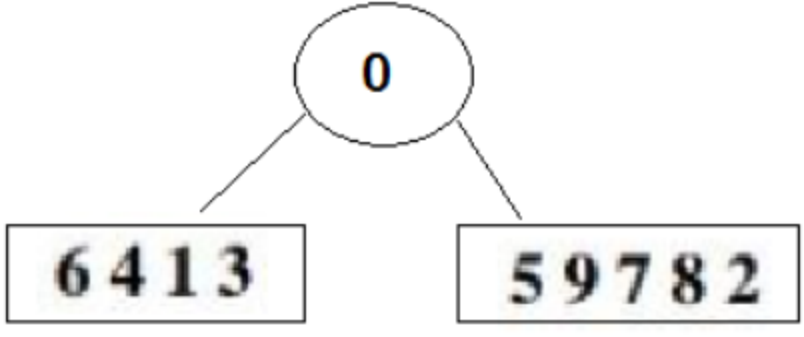
In-Order Traversal: **6 4 1 3 0 5 9 7 8 2**

Pre-Order Traversal: **0 1 4 6 3 2 5 7 9 8**

In Pre-order Traversal : Traversal order is Root, Left, Right

So, first number 0 is Root Node

From the list given In-Order traversal 0 is selected as Root and the list is divided like this according to its traversals order that is Left, Root and Right.

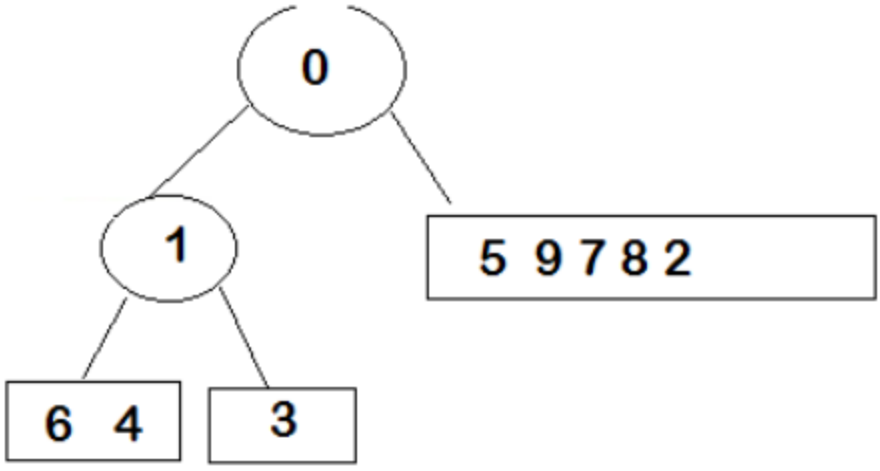


Now Considering List 1: 6 4 1 3

Now, finding root node among these numbers, we have check pre-order traversal list : 0 1 4 6 3 2 5 7 9 8

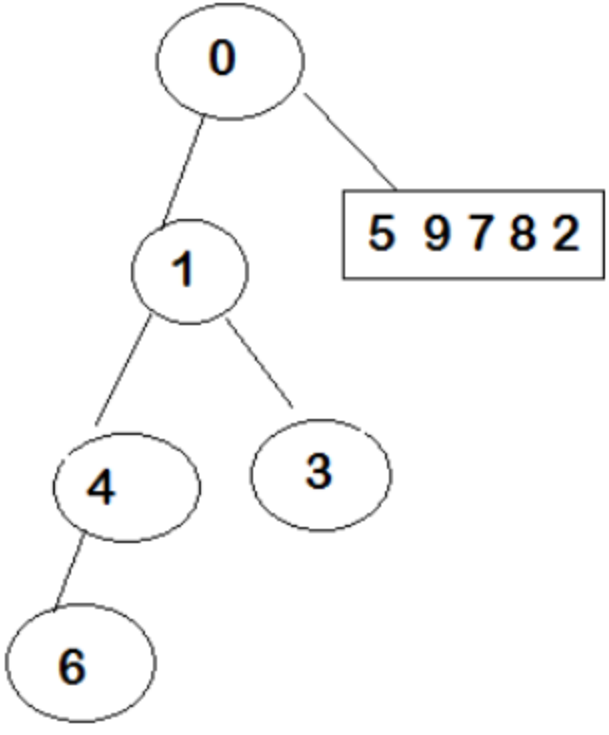
Among 6, 4, 1, 3 number 1 is visited first. So, number 1 becomes root.

So, list 6 4 1 3 is divided in this way: 6,4 becomes left sublist and 3 becomes right sublist.(1 is root Node). The tree becomes like this:



In the same considering 6, 4 list finding root element. For this check preorder traversal, Node which is visited first (among 6, 4) is treated as rooted. Here 4 is visited first in preorder traversal. 4 is root node. Left list is 6.

Now, tree is



Perform same procedure for right sublist also,

Right sublist is 5, 9, 7 , 8, 2. To find root check pre-order traversal, 2 is visited first among 5, 9, 7, 8, 2.

So, number 2 is root node. Left list is 5, 9, 7, 8.

Again, to find root from Left list 5, 9, 7, 8 check pre-order traversal, 5 is visited, which becomes root node.

So, number 5 is root node. Right List is 9, 7, 8.

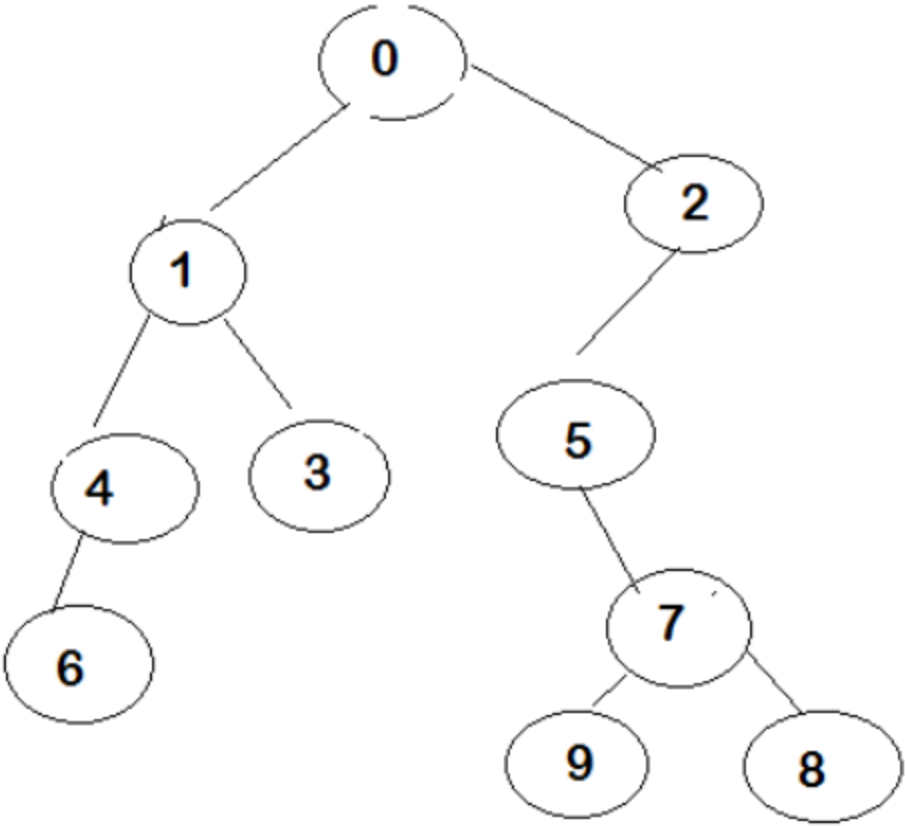
Again, to find root from Right list 9, 7, 8 check pre-order traversal 7 becomes root node, 9 left node and 8 right node.

The complete steps are given below:

Resultant binary tree formed is

In-Order Traversal: **6 4 1 3 0 5 9 7 8 2**

Pre-Order Traversal: **0 1 4 6 3 2 5 7 9 8**



Post order traversal is 6 4 3 1 9 8 7 5 2

(96) AVL Tree :-

50, 40, 25, 10, 15, 5, 45, 35, 20, 30

Rules :-

-> every tree has a root node (at the top)

-> the root node has zero, one or two child nodes.

-> and also child nodes has zero, one or two child nodes and so on

-> The each node has up to two children.

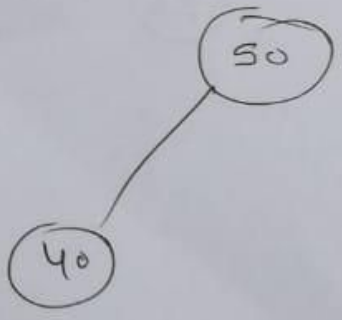
-> its left descendants are less than the current node, which is less than the right descendants.

L < root >= R

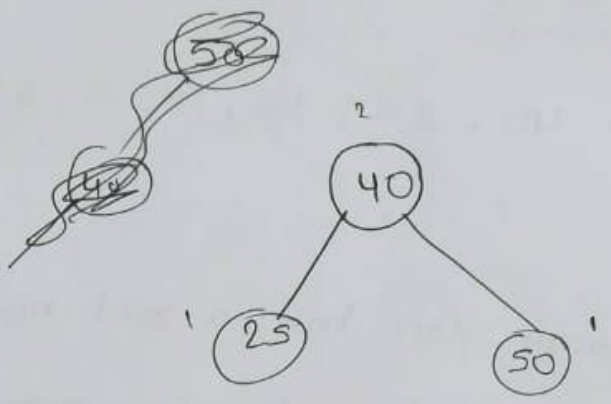
Step 1 :-

Insert 50 :-

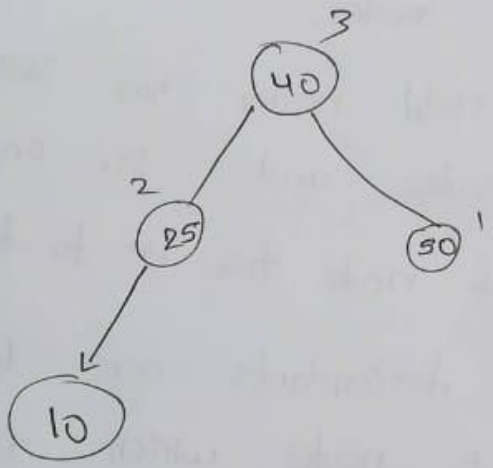
Insert 40 :-



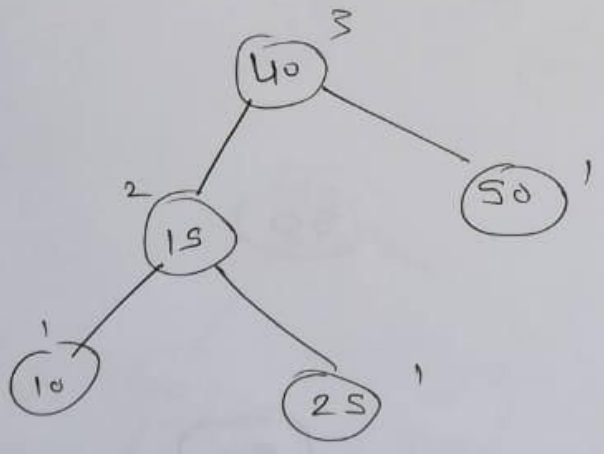
Insert 25:-



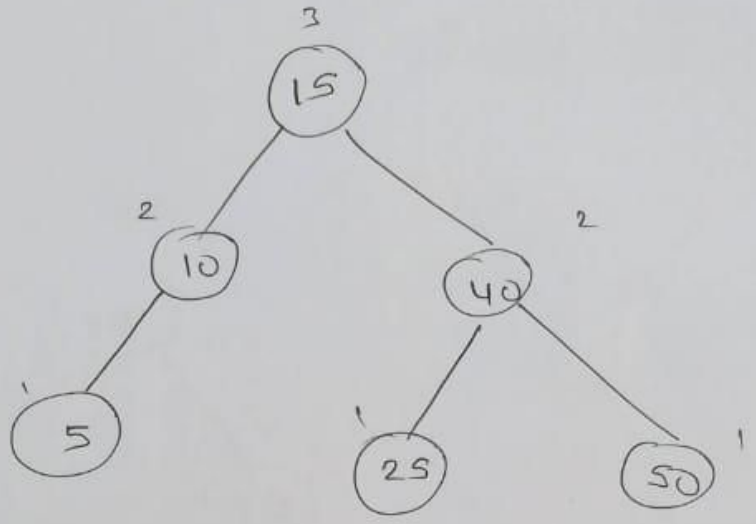
Insert 10:-



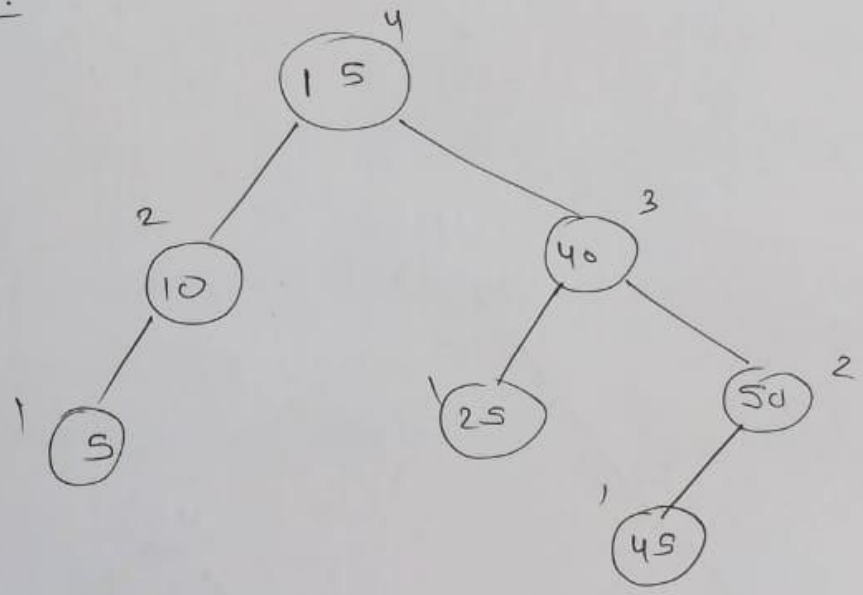
Insert 15:-



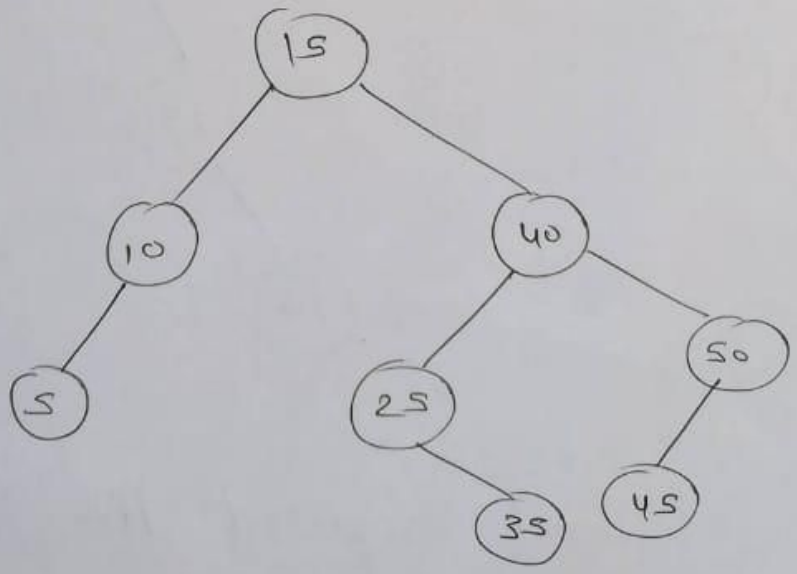
Insert 5 :-



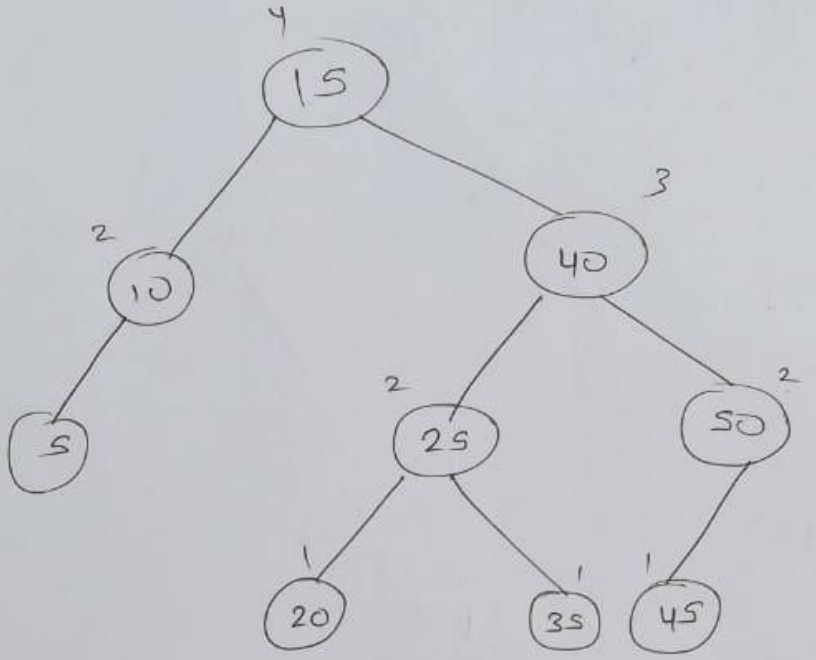
Insert 45 :-



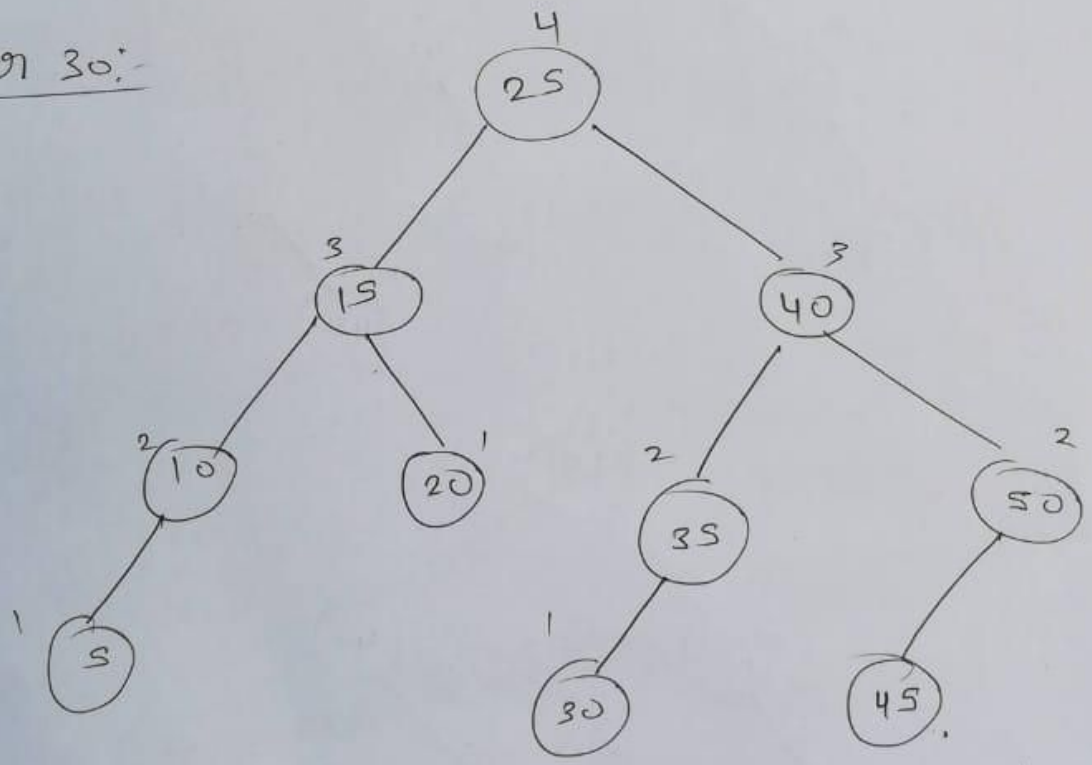
Insert 35 :-



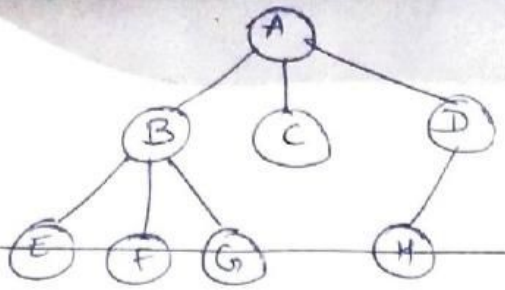
Insert 20:-



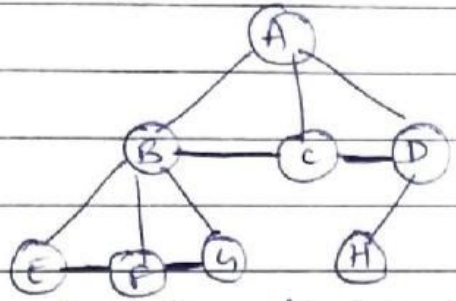
Insert 30:-



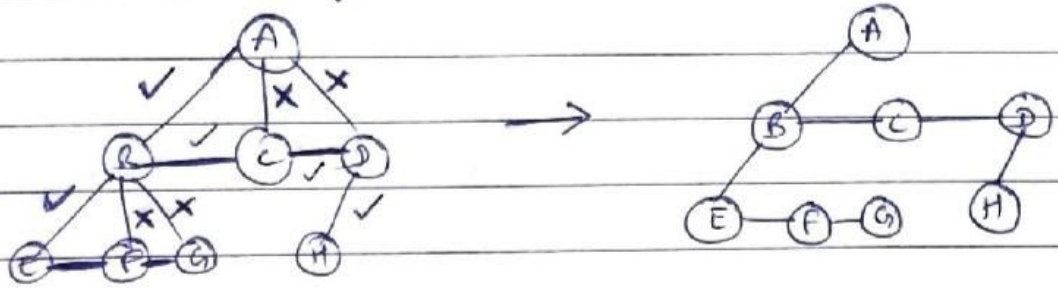
find required AVL Tree.



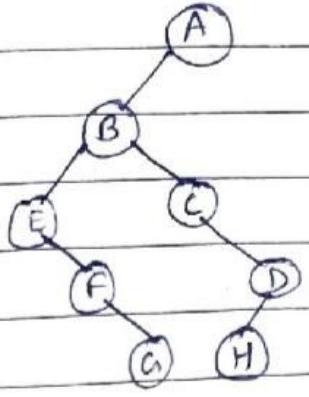
Step 1:- Insert edge connecting all siblings of the tree.



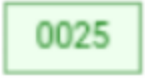
Step 2:- Delete all parent edges from parent to its children except the one to its leftmost child.



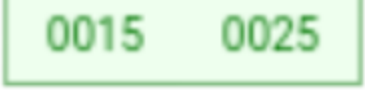
Step 3:- Rotate the resulting tree by 45° to differentiate b/w its left & right subtree.



Inserting 25



Inserting 15:



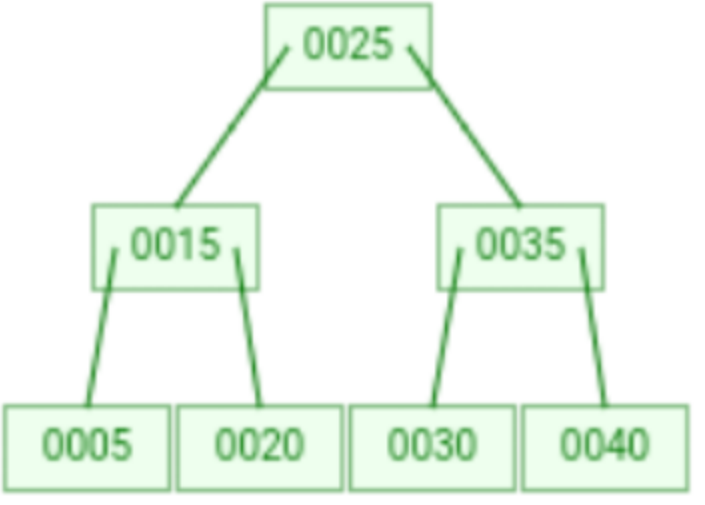
Inserting 5:

Inserting 30:

Inserting 20:

Inserting 35:

Inserting 40:



Deleting 30: