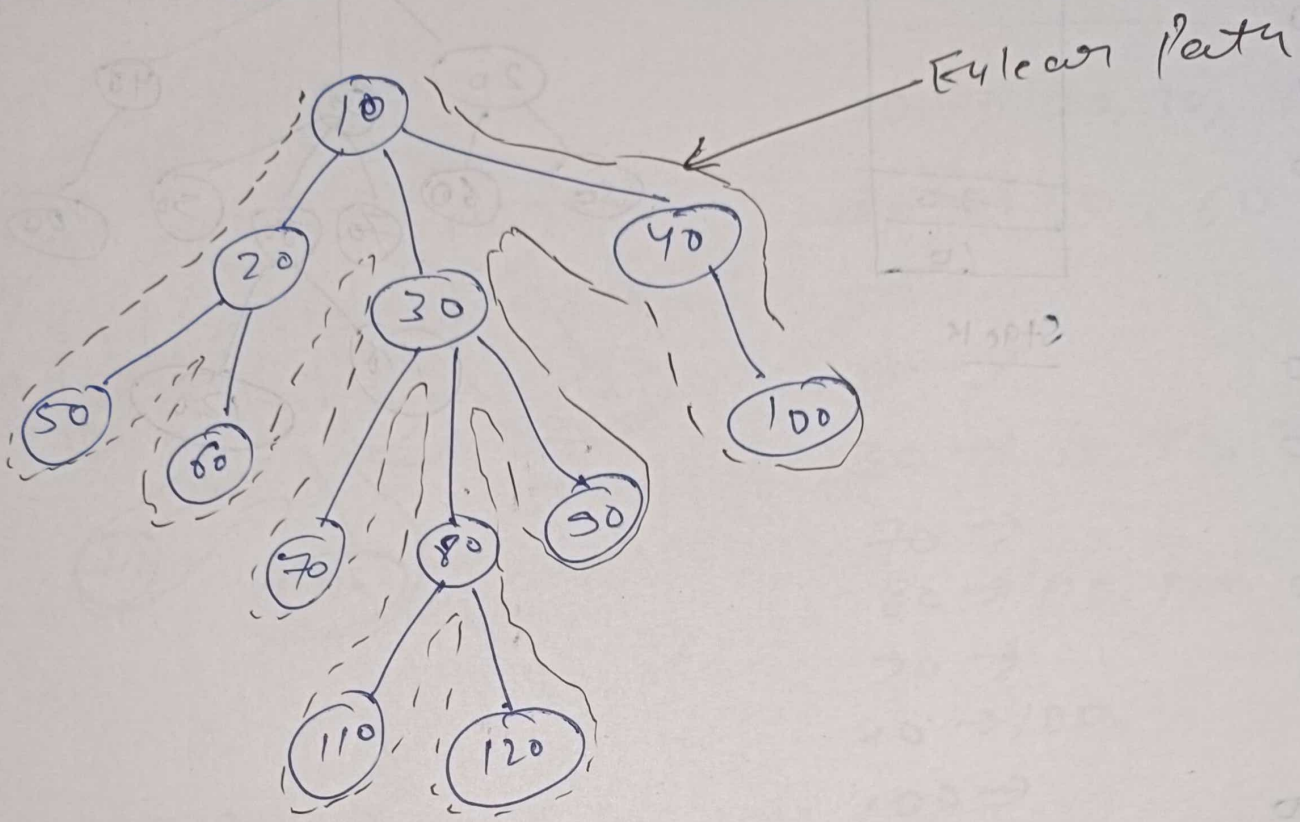
Generic Tree

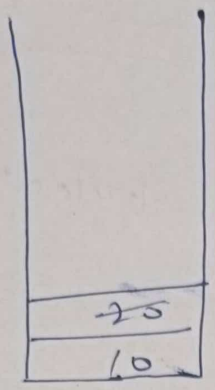


i/p according eulear path →

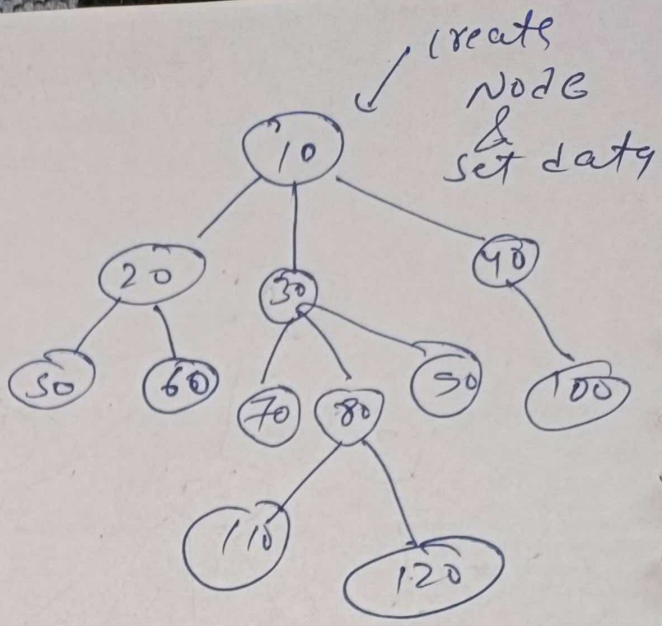
10
20
50
-1
end 50
60
-1
end 60
end 20
-1
30
70
-1
80
110
-1
120
-1
90
-1
-1
40
100
-1
-1
-1

Pass 10

10
20
50
-1
60
-1
-1
30
70
-1
80
110
-1
120
-1
-1
90
-1
-1
40
100
-1
-1
-1



Stack



Create Node & set data

(0th) pass data from i/p array

(any) if i/p data == -1, pop from stack

Steps :-

(i) ~~pass data from i/p array~~

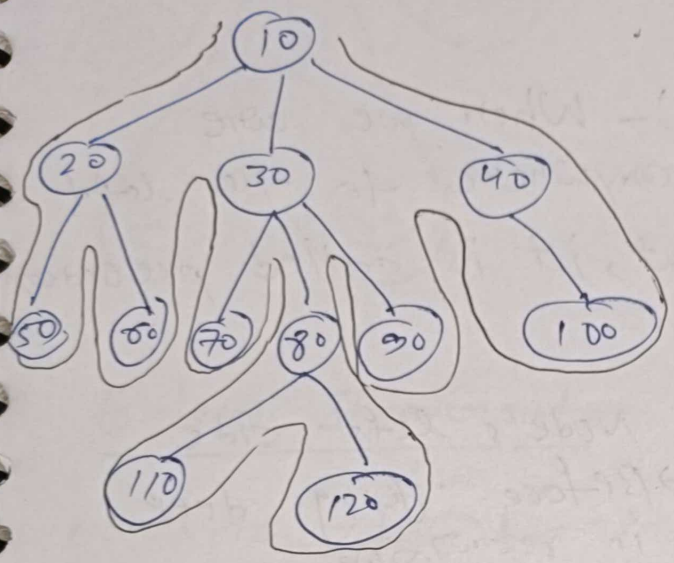
(ii) create Node & set data

(iii) Now push that Node into stack

if stack empty
then set that Node as root & push into stack

not empty
↓
Make the top of the stack as parent of the new Node.
& push into stack

Display a Generic Tree

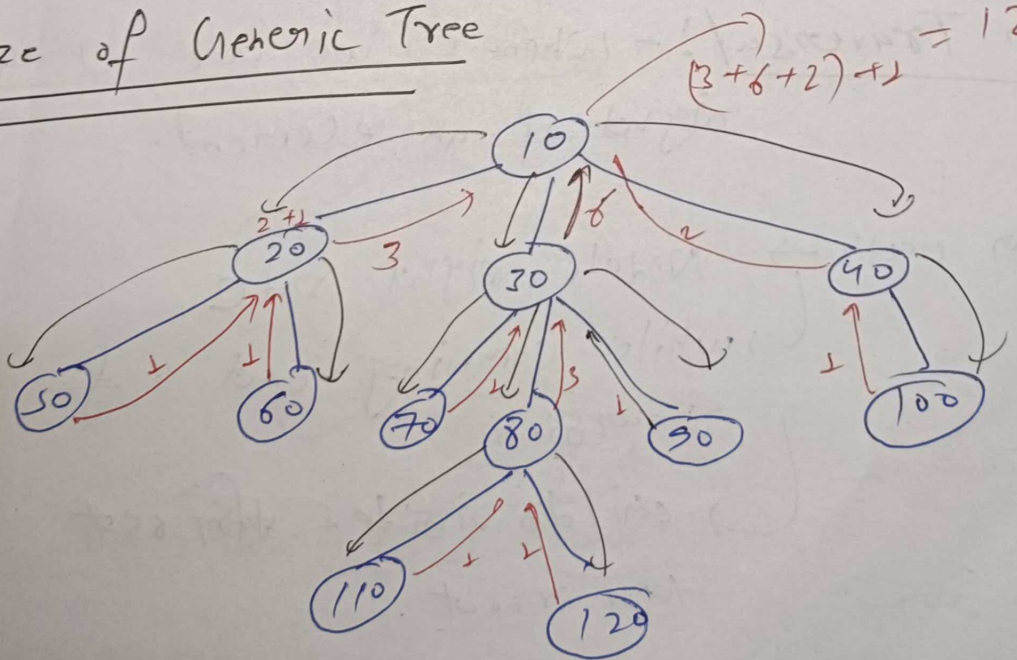


o/p in Display

10 → 20, 30, 40
20 → 50, 60
50 →
60 →
30 → 70, 80, 90
70 →
80 → 110, 120
90 →
40 → 100
100 →

d (10)
Means → s(10) → self of 10
→ d (20) → display of 20
→ d (30) → " " 30
→ d (40) → " " 40

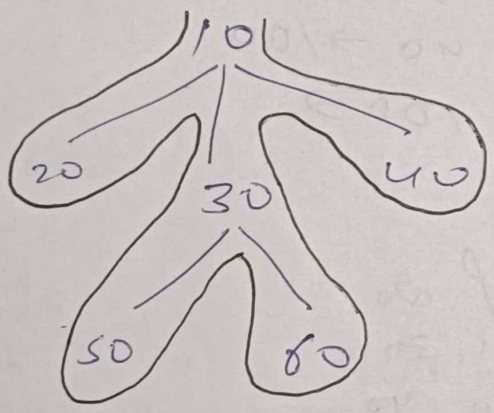
Size of Generic Tree



Traversal in Tree

Pre-Order Traversal :- When we are anywhere to the left of an element, it is called preorder traversal

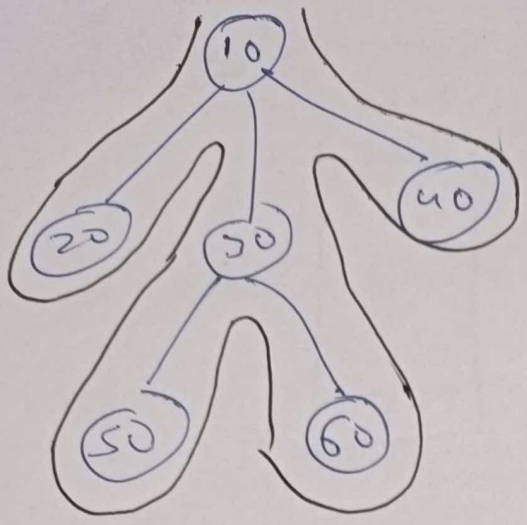
⇒ In euler path → Node's left side
- → Before going deep in recursion
- → root visited first then child



pre-order → 10
20
30
50
60
40

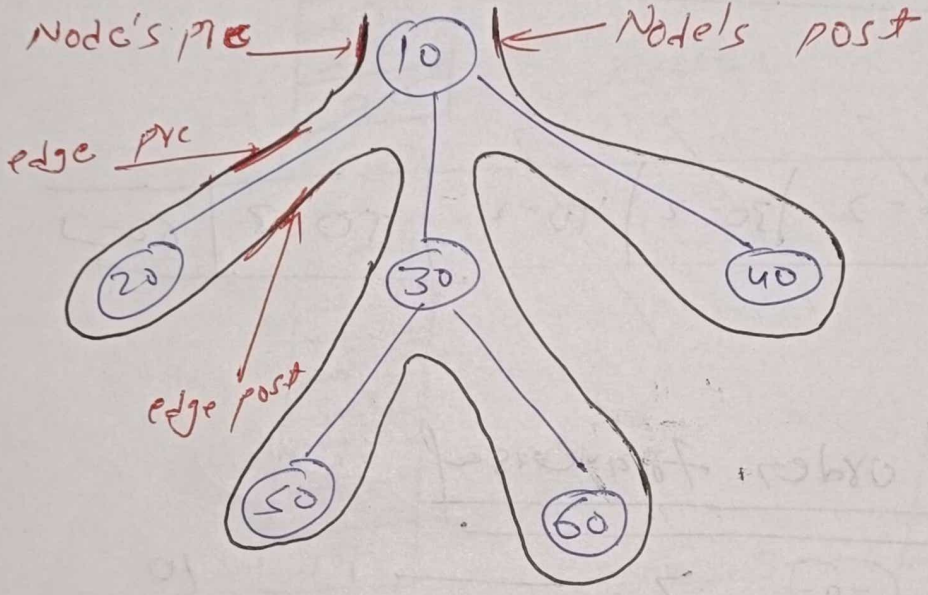
Post-order Traversal :- Whenever we are at the right of an element.

⇒ In euler path → Node's right side
- → while coming out of recursion
- → child is visited first then root

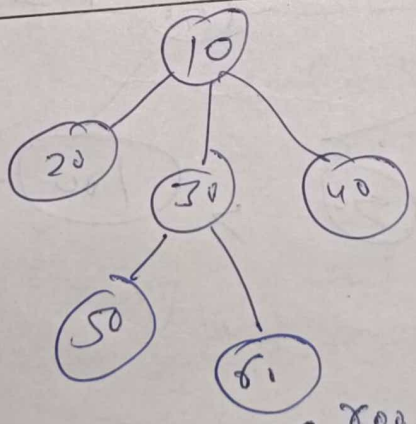


postorder traversal → 20 50 60 30 40 10

Euler Traversal



Level order traversal



root

Algorithm → queue → 10

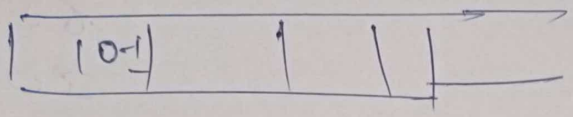
q → till queue is not empty

p → print

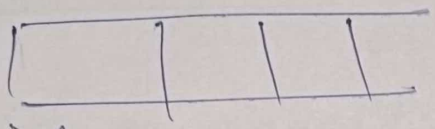
q → add children

q → remove

why this will work



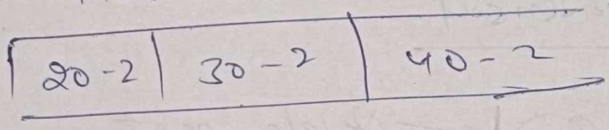
remove



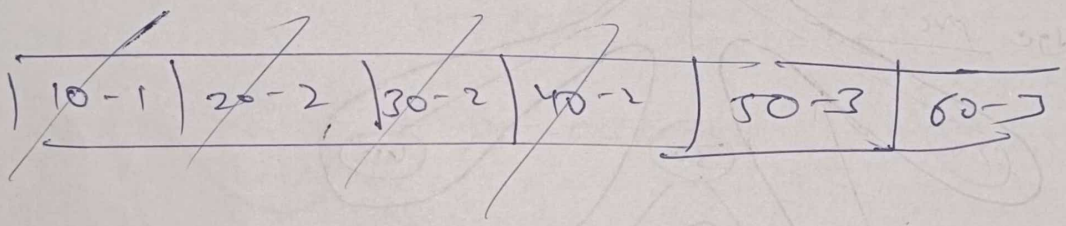
print

10

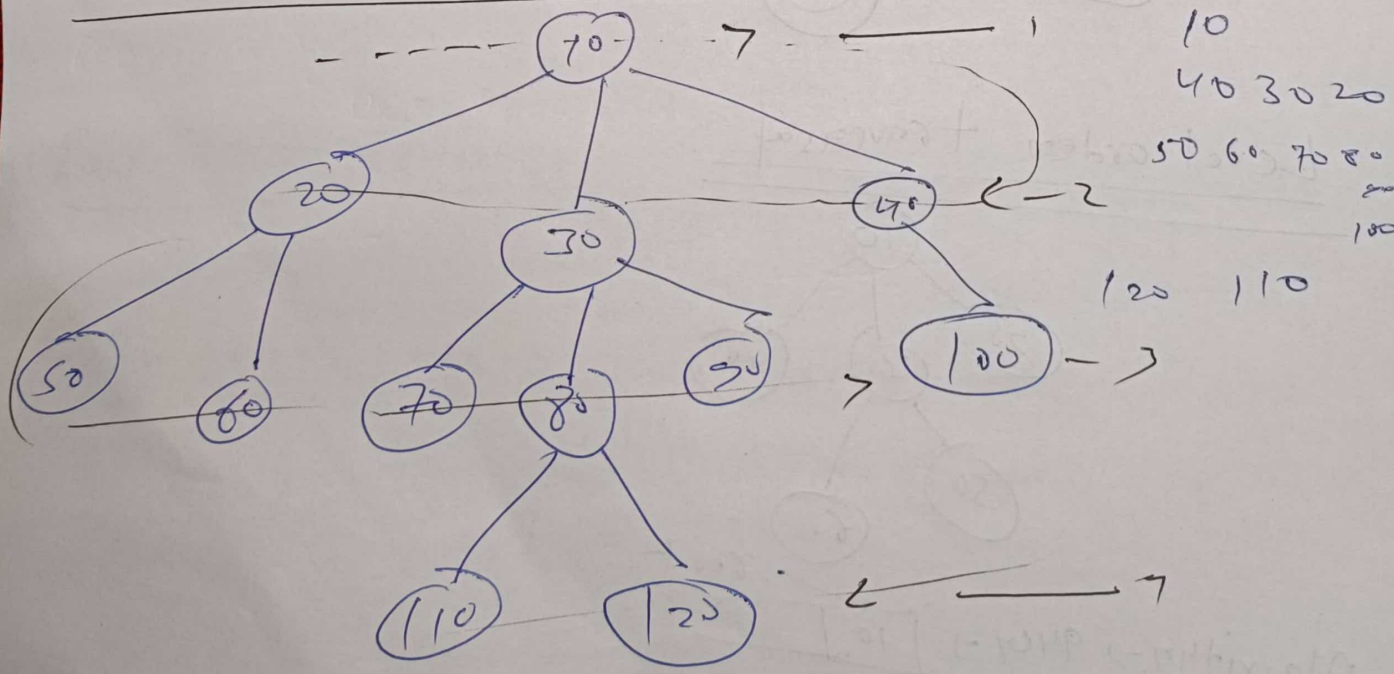
add children



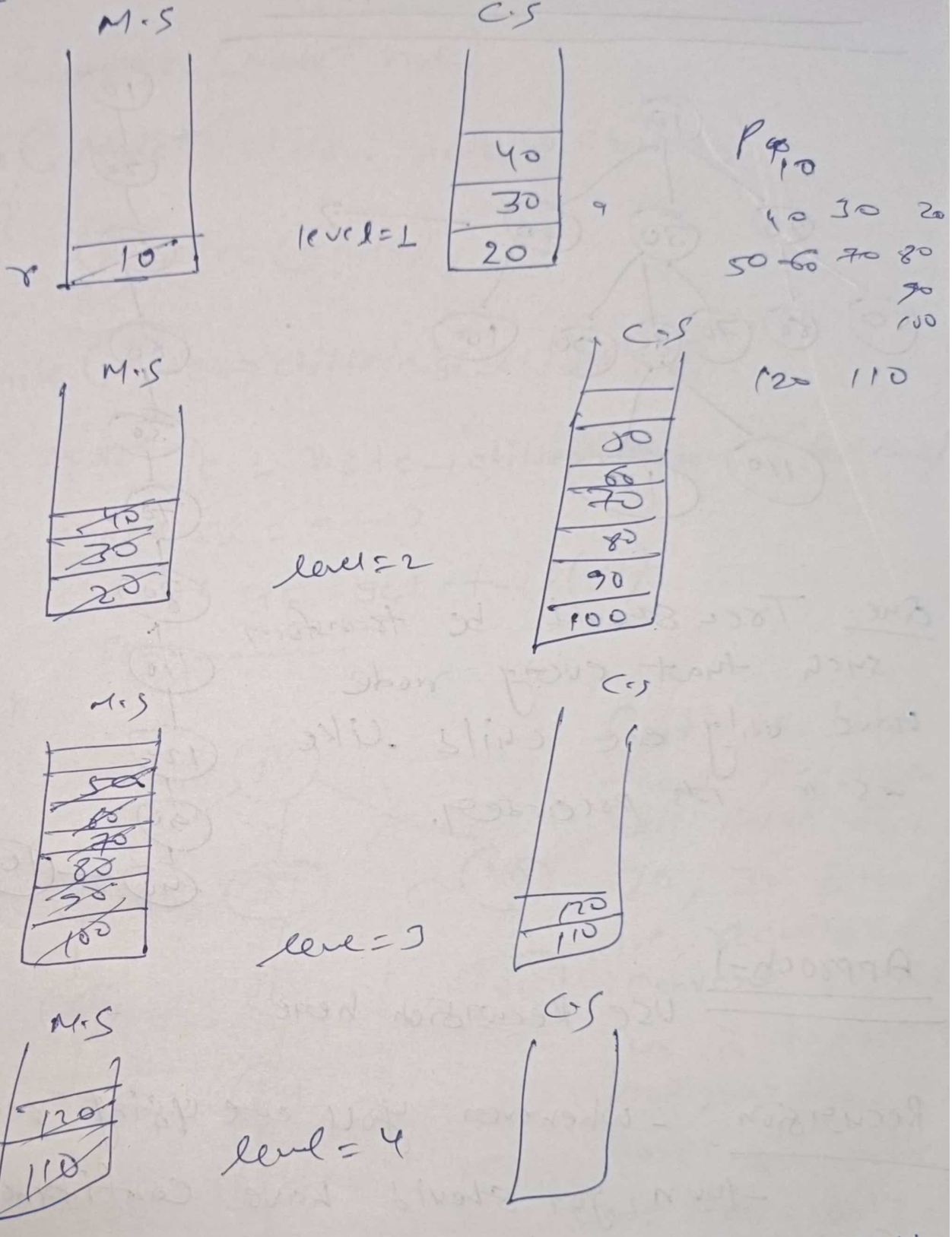
or



zig-zag Level order traversal

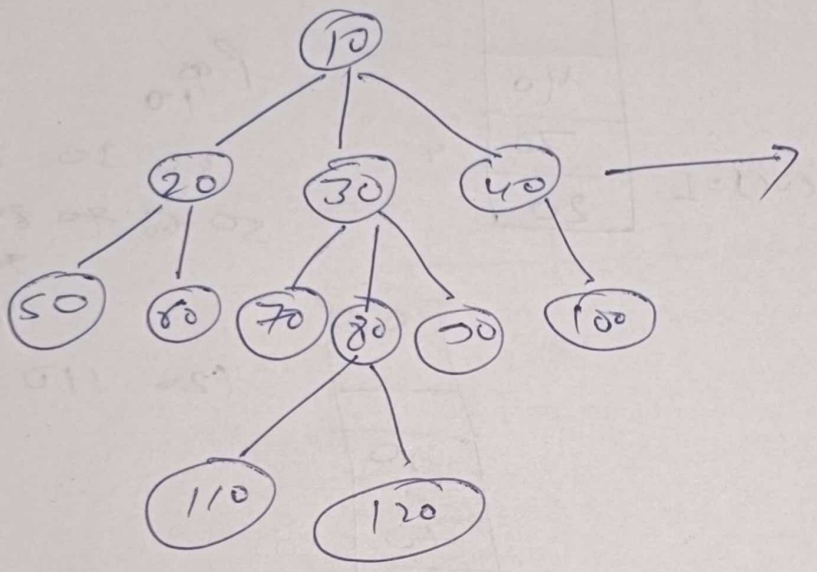


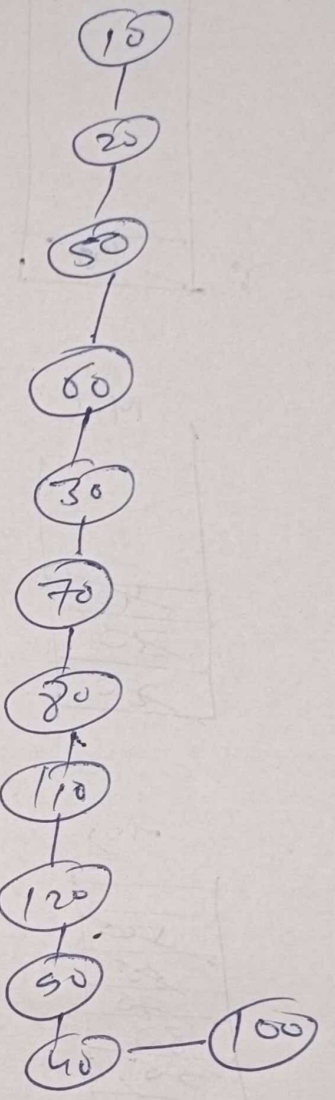
Q 29



if we are going in this direction → then add child in also this direction into stack

Linearize a Generic Tree





Que. Tree should be transform such that every node have only one child like as in its preorder.

Approch-1

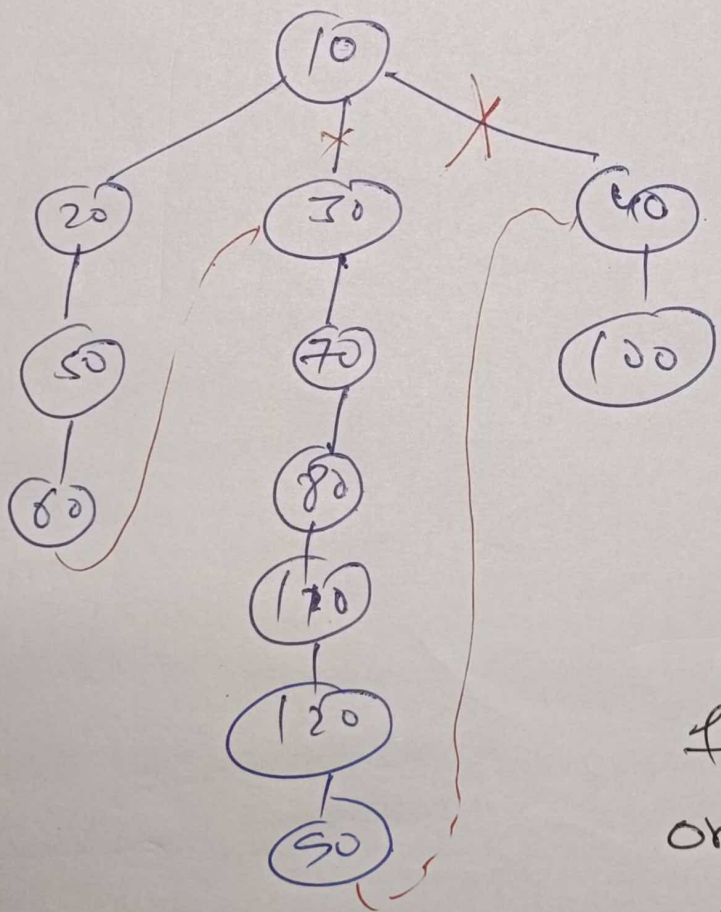
Use Recursion here

Recursion :- Whenever you are using recursion then you should have confidance or faith such that recursion solve the problem & you have to solve only one sub problem like in this question

Suppose after using recursion like

```
void linearize(Node* node)
{
    for(Node* child : node->children)
    {
        linearize(child);
    }
    while(node->children.size()>1)
    {
        Node* lc = node->children.removeLastChild();
        Node* sl = ?
        Node* sf = getHead(sl)
        sl.next = sf;
    }
}
```

recursion



faith is like that Recursion will solve for 20, 30 & 40 i.e. recursion convert 20, 30 & 40 in linear and now you have to write code for only for 10 or only for one sub problem.