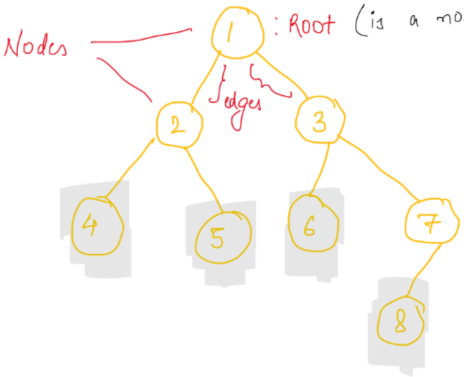
Trees 1

Introduction To Trees

→ Tree is non linear data structure

→ hierarchial.



: Root (is a node without parent)

- 6 and 7 are siblings
- 3 is parent of 7.
- 7 is child of 3
- 7, 3, 1 are ancestors of 8.
- 8 is descendant of 3
- Leaf Nodes : Nodes with no children.

- **Height of a node** :- Length of longest path from a node to its farthest descendant nodes.

  H (leaf node) = 0

  H (root node) = H (Tree)

  eg :- H (3) = 2

  H (8) = 0

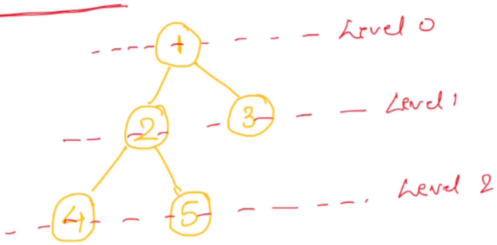
- **Depth of a node** :- Distance between the current node from the root node.

  eg :- Depth (7) = 2

  Depth (1) = 0

- **Subtree** :- Part of a tree.

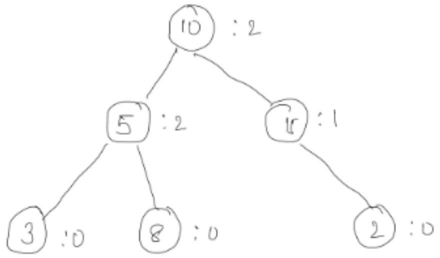
- **Levels of a Tree**



Binary Trees

→ Tree with all the nodes having <= 2 children.

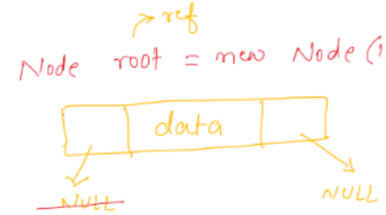
→ Bin + ary ⇒ 2- array tree.



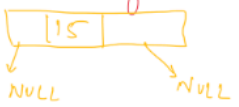
Node Structure :-

```
Class Node {
        int data;
        Node left;
        Node right;
        Node (int data) {
            this.data = data;
            this.left = Null;
            this.right = Null;
        }
}
```

Node root = new Node (10);

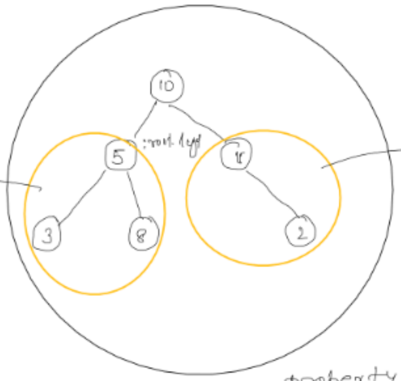


root.left = new Node (15)



- Seiralization and Deserialization is the way using which trees are created.

Property



left subtree for which root.left is the root node.

Right subtree for which root.right is the root node

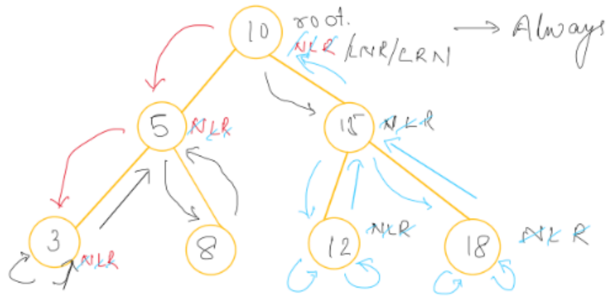
→ Because of this property recursion plays vital role in solving Trees problems.

Traversals In a Tree

1) Preorder :- Root - Left - Right

2) Inorder :- Left - Root - Right.

3) Postorder :- Left - Right - Root.



→ Always start from root.

1) Preorder : (NLR)

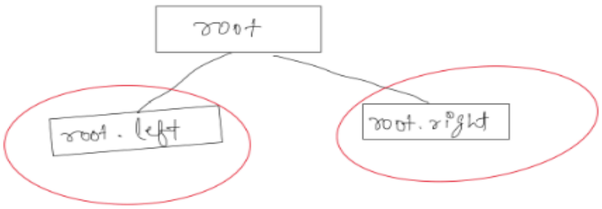
10 5 3 8 15 12 18

Iterative Inorder Traversal

1) Go to the left subtree & do inorder

2) Print (root.data)

3) Go to right subtree & do inorder.

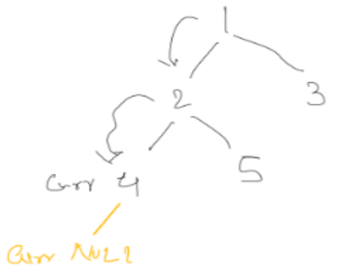


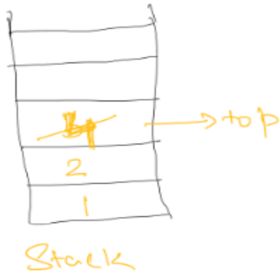
Pseudo Code:-

```
Void Inorder (Node root) {
    if(root == NULL)
        return;

    inorder (root.left);
    Print (root.data);
    inorder (root.right);
}
```

Iterative Inorder Traversal :-





```
while (Curr != NULL || !st.empty())
{
    if (curr != NULL)
        st.push (curr);
        Curr = Curr.left;
    } else {
        curr = st.top();
        st.pop();
        Print (curr.data);
        Curr = curr.right.
}
```

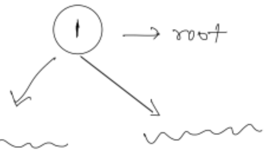
Construct Binary Tree from Inorder And Postorder

Inorder: [4 2 7 5 1 3 6] Return root of the tree.

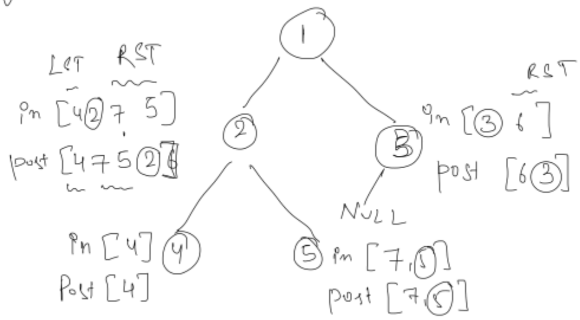
$\underbrace{4\ 2\ 7\ 5}_{LST}$ 1 $\underbrace{3\ 6}_{RST}$

Postorder: [4 7 5 2 6 3 1]

→ Identify the root node from the post order

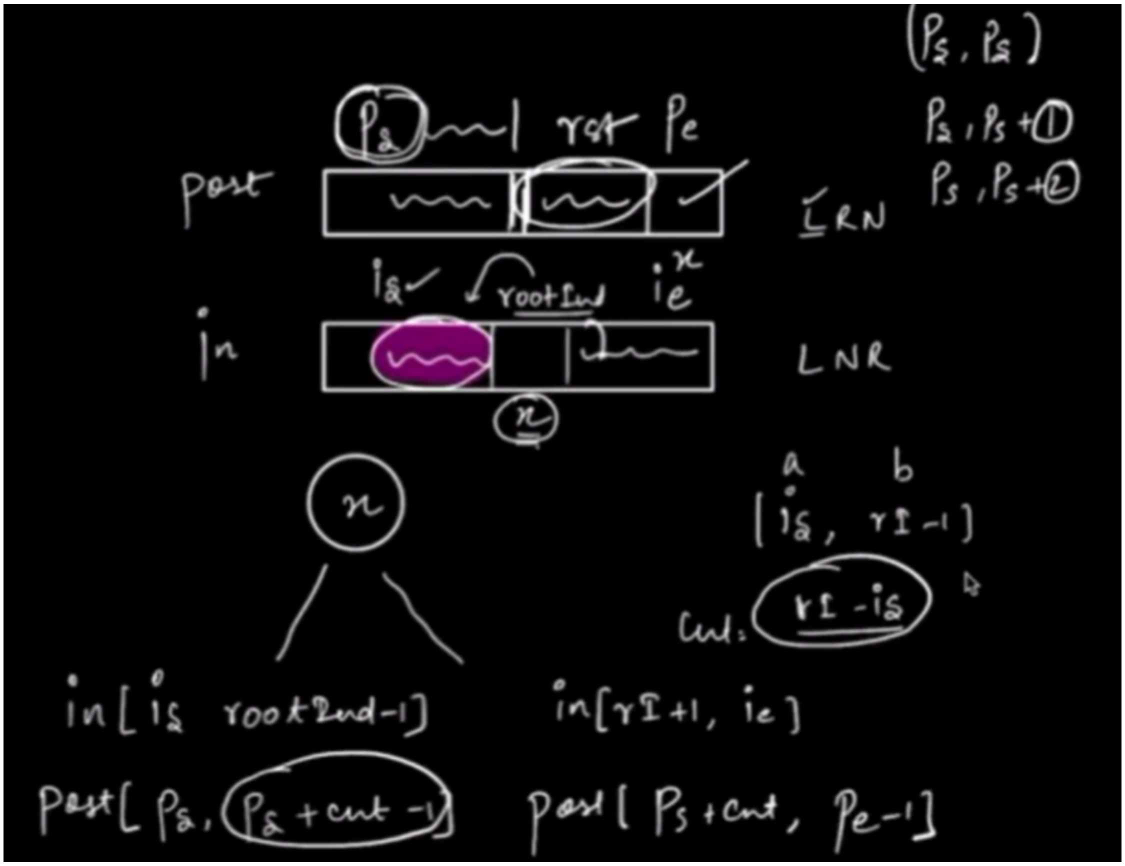


→ Identify LST & RST from Inorder array.



Pseudo Code:-

```
Node ConstructTree (in[], post[], start_i, end_i, start_p, end_p)
{
        if (start_p >= end_p) {return NULL;}
        int rootValue = post[pe]
        Node root = new Node (rootValue);
        int rootIndex = IndexOf (rootValue, inorder);
        int Cnt = rootIndex - start_i;
        root.left = ConstructTree (in, pre, start_i, rootIndex-1, start_p,
                                                              start_p+Cnt-1);
        root.right = ConstructTree (in, post, rootIndex+1, end_i, post_st Cnt
                                                              post_e-1);
        return root;
}
```



(ps, ps)

ps, ps+1

ps, ps+2

[is, rI-1]

cnt: rI - is

in[is rootInd-1]

in[rI+1, ie]

post[ps, ps+cnt-1]

post[ps+cnt, pe-1]