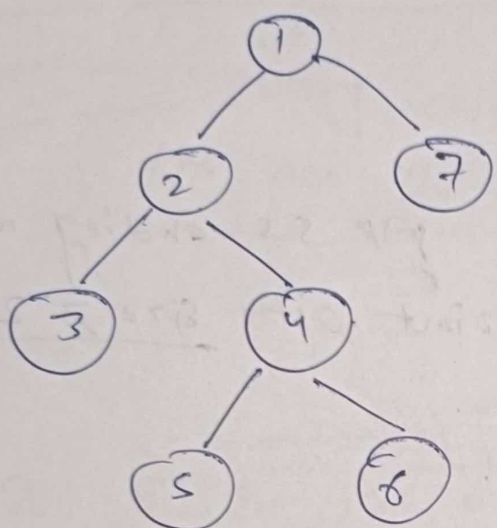
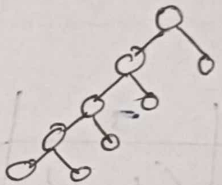
Iterative Preorder Traversal in Binary Tree



Preorder → 1 2 3 4 5 6 7

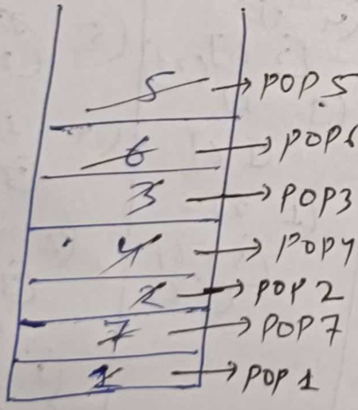
T.C. = O(n)

S.C. = O(n) ≃ O(H)
when tree like one right & more left



data structure use → stack

initially put root onto stack



Now in loop while stack is not empty do following ~~to~~ steps:

(i) pop the Node from stack top & print it

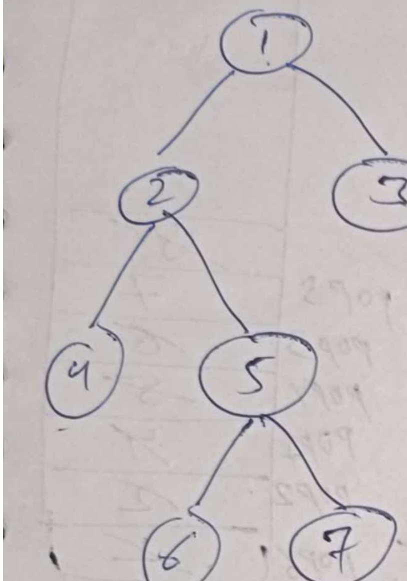
(ii) put right child onto stack

(iii) put left child onto stack.

1 2 3 4 5 6 7
preorder.

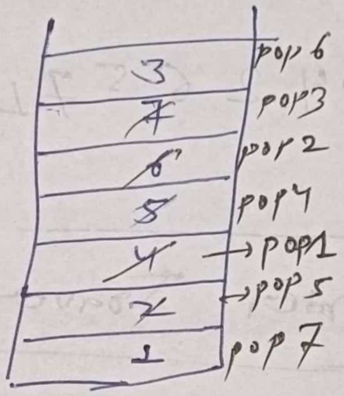
why we put first right & then left it is because in preorder we go for root left right that means after the root left part traversed i.e & in stack is LIFO Data structure & if we put left in last that means it stay on top of the stack.

Itrative Inorder Traversal for Binary Tree



Inorder → left Root right
↳ 4 2 6 5 7 1 3

In recursion how stack is working



ans → 4 2 6 5 1 3

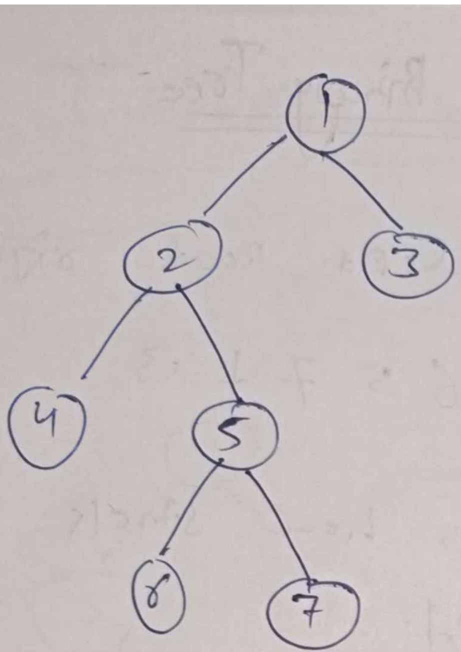
In itrative ~~[illegible]~~ inorder we maintain a stack data structure simillier to recursion that means

we take a node = root variable

~~and while no~~

and in every itration we follow these steps.

```
if( node != NULL)
{ st. push (node)
  node = node -> left;
}
else{
    if( st.empty()) break;
    node = st.top();
    st.pop();
    Print (node)
    node = node -> right;
}
```



nod= ~~1~~

~~2~~

~~4~~

~~null~~

~~null~~

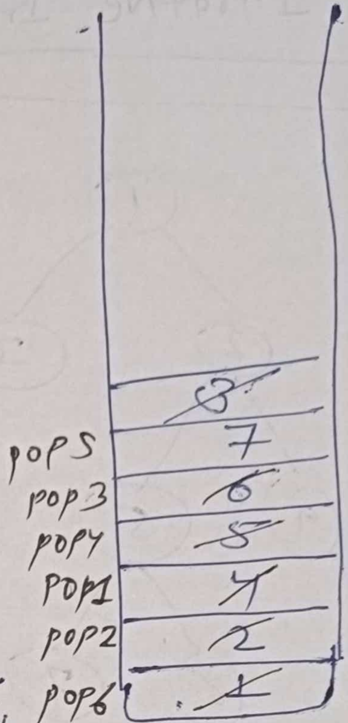
~~8~~

~~6~~

~~null~~

~~null~~

~~7~~ ~~null~~ ~~null~~



inorder → 4 2 6 5 7 1, 3 ~~3~~ ~~NULL~~ ~~NULL~~

Itrative Postorder Traversal

Approch 1 - using 2 stack

take 2 stack st1 & st2

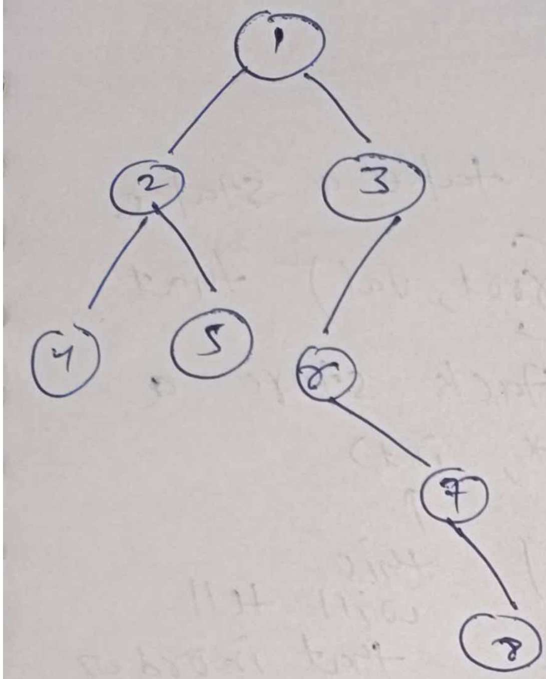
st1 initially store the root.

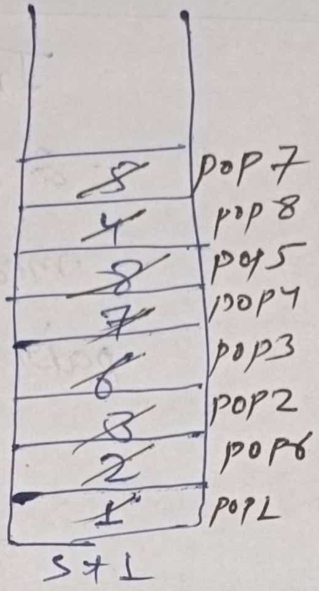
apply while loop & do following steps.

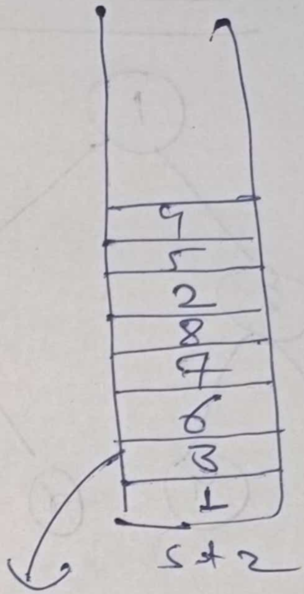
```
while ( !st1.empty() )
{
        step 1 → pop the top element from
                  st1.
                  and push this element
                  into st2,

        step 2 → push the left child of
                  the pop node if not NULL

        step 3 → push the right child of
                  the pop node if not
                  NULL
}
```

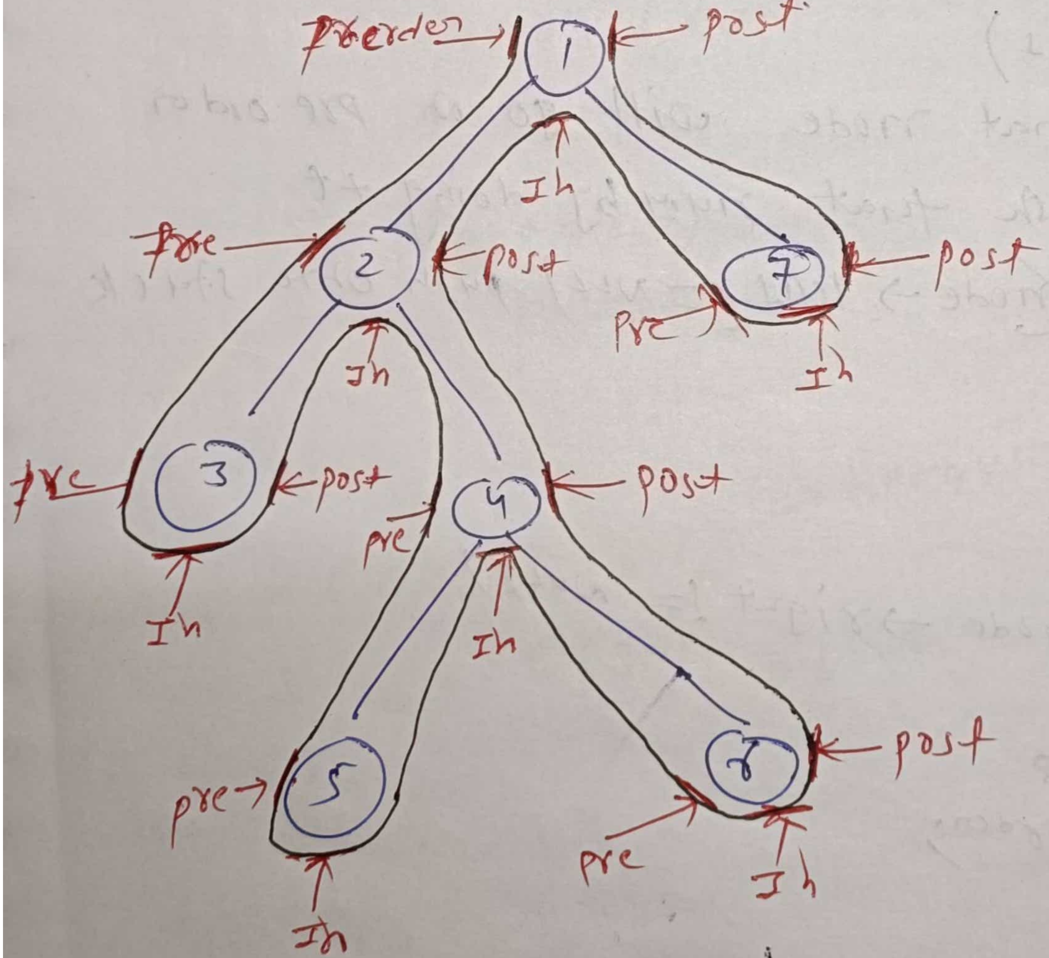




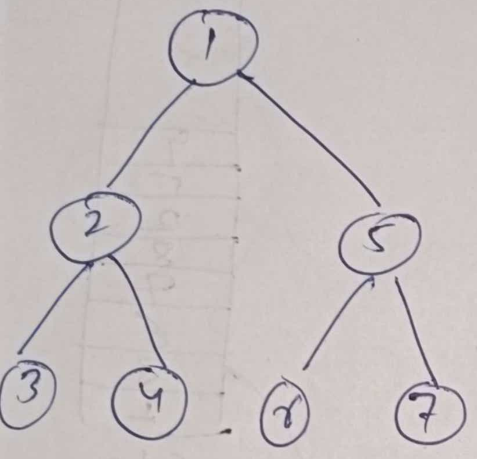


this stack
store the
postorder

4 5 2 8 7 6 3 1



Inorder / Preorder / postorder



Initially take a stake & inser (root, val) that means stack store a pair <Node*, int)

↑ pointing to node of tree

↑ this will tell that inorder pre order & post order

Rule :-

~~while you~~

if whenever you taking out an element from the stack & the num of the element is like

```
if ( num == 1)
{   then that node will go on pre order
    and push that num by doing ++
    if (node -> left != NULL) push onto stack
}
if num = 2
    inorder
    num ++
    push (node -> right != NULL)

if   num ==3
     postorder;
```

Rule for every itration

```
pair <Node*, int> = st.top()
    st.pop();
int num = pair. second;

if (num == 1)
{   preorder
    push ({pair.first, num++})
    push ({pair.first -> left != NULL})
}

if ( num = 2)
{   inorder
    push (____ num++)
    push (right -> child)
}

if (num == 3)
    postorder.
```

T.C. = O(3N) → because when you draw stack then you are visiting every node 3 times.

S.C. = O(3N) or O(4N)