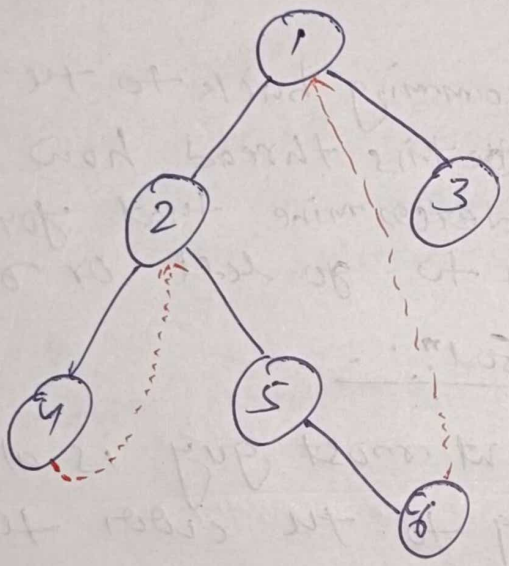
Morris Traversal

T.C = O(N)
S.C = O(1)

Morris Traversal uses the concept of threaded BT

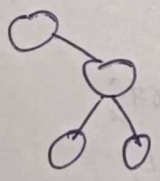
Inorder (left root right)

inorder - 4 2 5 6 1 3



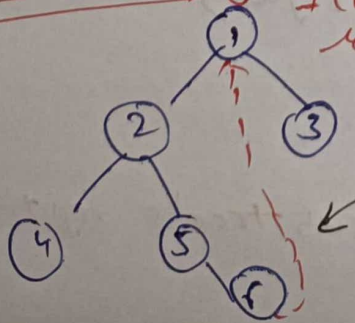
pattern → from the last node of any left subtree you go back to the root and after that you back to the right.

Case-1



```
if(left == NULL)
{
    inorder.push_back(root->val);
    root = root->right;
}
```

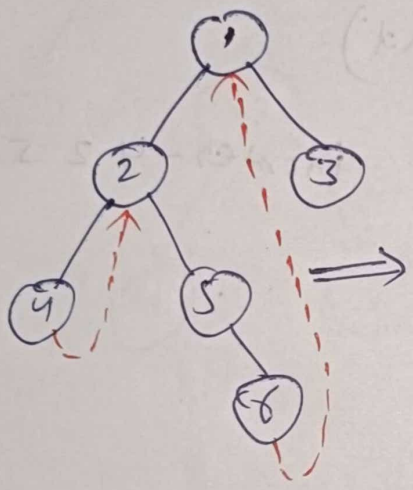
case-2



if you are standing at 1 (root) before moving to left you should make the thread/pointer BCZ if you once move to the 2 you can't come back at 1 without the thread.

Before going to the left subtree create a thread by the right most guy of the left subtree to the root

after that curr = curr → left.



⟹ after comming back to the 1 via this thread how you can determine that you have to go left or right.

Sol^n :-

if right most guy is already pointing to the curr then go the right of the curr & remove the thread.

Code

```
vector<int> inorder(TreeNode* root)
{
    vector<int> inorder
    TreeNode* curr = root;
    while (curr != NULL)
    {
        //case-1
        if(curr->left == NULL)
        {
            inorder.push_back(curr->val);
            curr = curr->right;
        }
        else
        {
            // go to the left subtree's right most guy for thread.
```

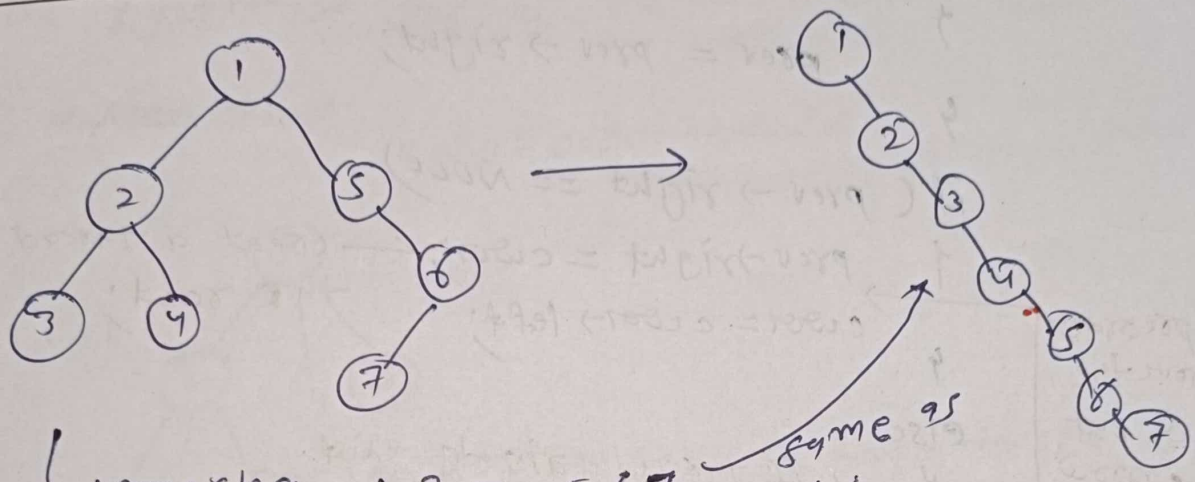
```
TreeNode* prev = curr->left;
while (prev->right && prev->right != curr)
{
    prev = prev->right;
}
if (prev->right == NULL)
{
    prev->right = curr;        <- creat a thread to the root;
    curr = curr->left;
}
else
{
    // if thread already exist.
    prev->right = NULL;
    inorder.push_back(curr->val);
    curr = curr->right;
}
}
return inorder;
}
```

for preorder remove the line & put the arrow location.

T.C. = O(N)
S.C. = O(1)

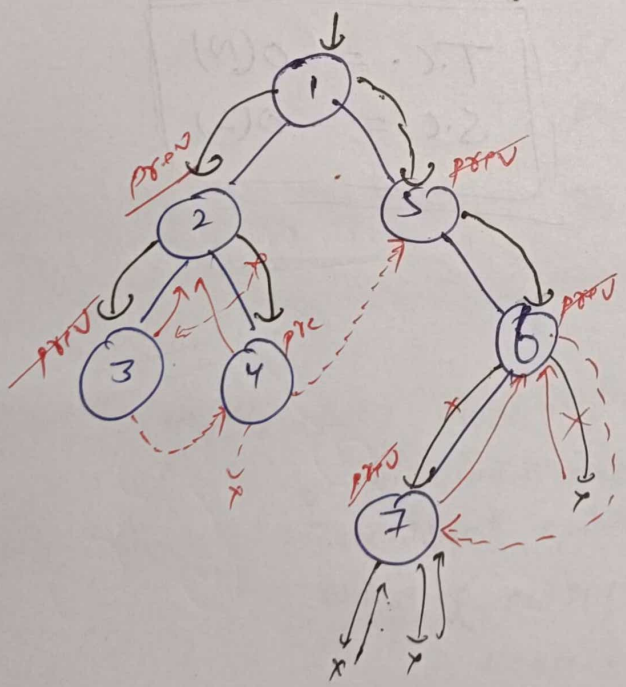
The inner while loop will run completely O(N) at most.

Flatten a Binary Tree to a Linked List



↳ preorder = 1 2 3 4 5 6 7

pre = NULL ~~1 2 3 4 5 6~~ 2



Code

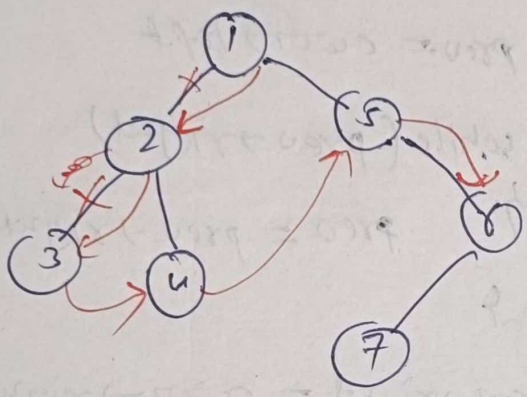
```
prev = NULL;
flatten (node)
{
    if (node == NULL)
        return;

    flatten (node → right)
    flatten (node → left)

    node → right = prev;
    node → left = NULL;
    prev = node
}
```

T.C. = O(N)

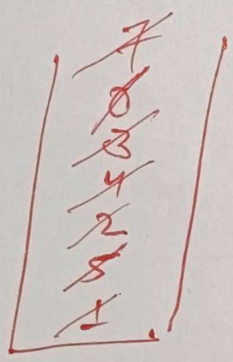
S.C. = O(N)

Approch-2

curr = 1 3 4 2 5 6 7

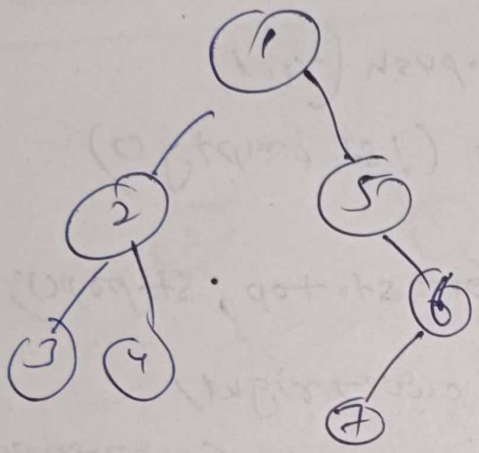



```
st.push(root)
while (!st.empty())
{
  curr = st.top(); st.pop();
  if(curr->right)
      st.push(curr->right)
  if(curr->left)
      st.push(curr->left)
  if(!st.empty())
      curr->right = st.top();
  curr->left = NULL;
}
```



T.C. = O(N)
S.C. = O(N)

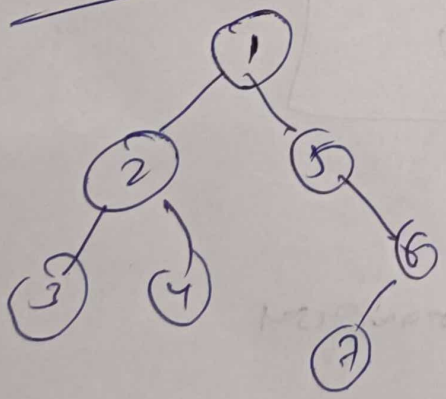
Now we use the morris traversal




```
curr = root;
while(curr != NULL)
{
if(curr->left != NULL)
{  prev = curr->left
   while(prev->right)
   {   prev = prev->right
   }
prev->right = curr->right;
curr->right = curr->left
}
curr = curr->right
}
```

T.C. = O(N)
S.C. = O(1)

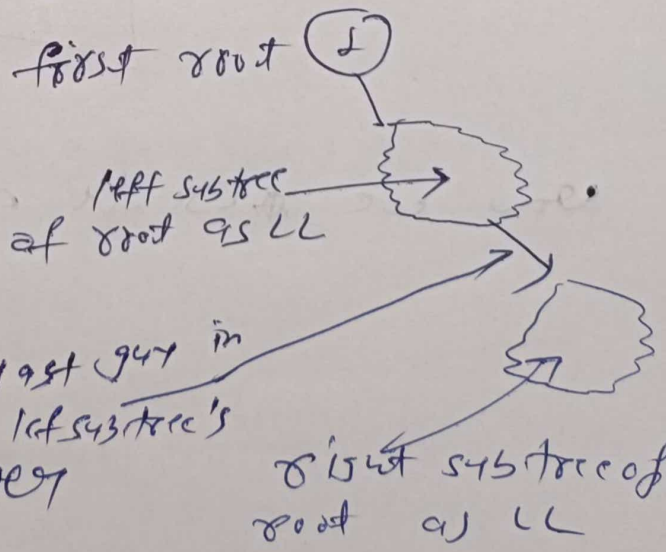
Thought process

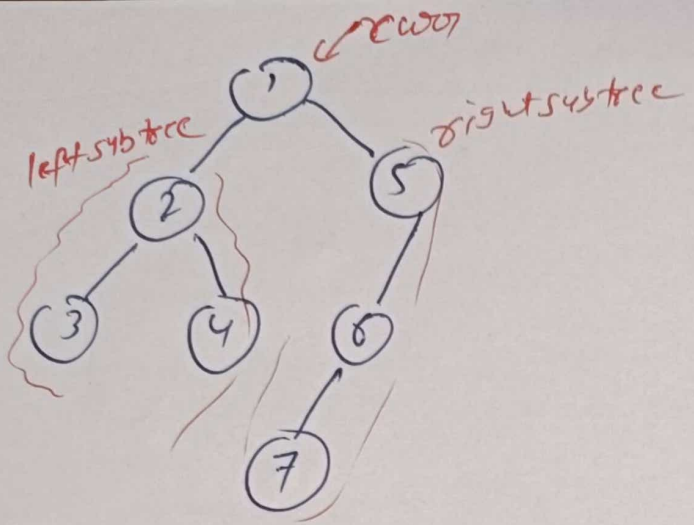


BT converted LL

(preorder same as

first root





leftsubtree के preorder का last guy find करो और उसे के right को curr के right से connect कर दो & curr के right को curr का left कर दो

```
if (curr->left != NULL)
{
    prev = curr->left
    while (prev->right)
        prev = prev->right

    prev->right = curr->right
    curr->right = curr->left
}
```