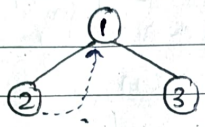
Date _____ Page _____

Day - 161

Trees - 9

* Morris Traversal:

⇉ Inorder:



⇉ Suppose, we have to find inorder.

⇉ For finding inorder, we have two methods that we know yet —

⇉ by using recursion &

⇉ iterative method. ⟶ T.C. → O(N) &
S.C → O(N).

⇉ Now, we have to solve or find inorder in O(1) S.C.

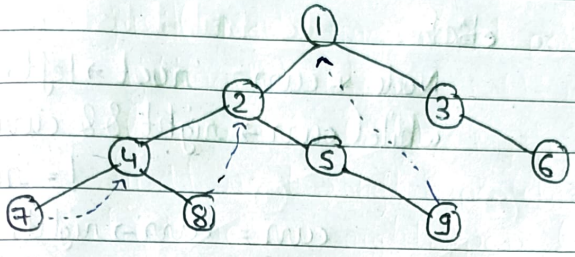
⇉ So, that time, we have used stack so that, we can come back.

⇉ Now, without using stack, we will have to know which node we have to come back.

⇉ For that, we will require a link.

⇉ So, when we are going to the left side then we tell the rightmost node of the left side to point to the curr. node.

Date: Page:



⇉ So, if left doesn't exist —
simply print value & then right.

⇉ If left exist,
⇉ Then first check if any link exist that means that part is traverse.
⇉ If not then traverse that part & make a link.
⇉ Also, if link exists then remove link & go to right part.

⇉ If the right side becomes null that means link doesn't exist.

Code

```
vector<int> inorder(Node *root){
    vector<int> ans;
    while(root){
        if(!root->left){
            ans.push_back(root->data);
            root= root->right;
        }
```

Date _____ Page _____

```
else{
    Node * curr = root->left;
    while(curr->right && curr->right
                        != root)
        curr = curr->right;
    if(curr->right == NULL){
        curr->right = root;
        root = root->left;
    }
    else{
        curr->right = NULL;
        ans.push_back(root->data);
        root = root->right;
    }
}
return ans;
}
```

T.C. → O(N) S.C. → O(1)

Pseudo Code

① Left doesn't exist
⇉ Note down the data.
⇉ Move to right.

② Left ~~root~~ part exist:
⇉ If left subtree not traverse —
→ Create the link.
→ Move to left.

Date _____ Page _____

⇉ else —

→ Remove the link.

→ Note down the data.

→ Move to right.

* Preorder:

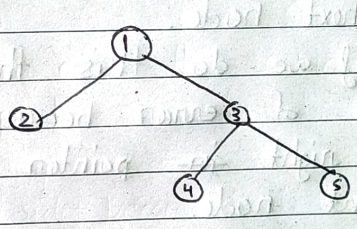
⇉ We can simply use the previous method.

⇉ When we are printing node in that part when the left part is traverse.

⇉ Here, we will print the node when the tree is not traverse.

* Postorder:

⇉ L R N



⇉ 2 4 5 3 1

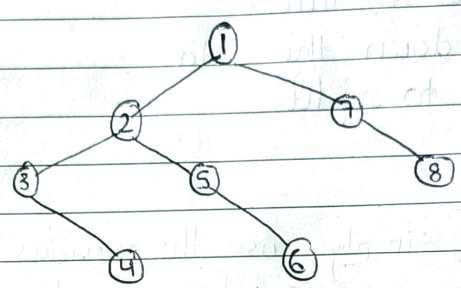
⇉ We will reverse the order i.e. N R L.

⇉ Then, we will reverse the ans and will get our required ans.

⇉ So, for solving this, we will use the previous approach but inspite of going to left side, we will go to right side.

Date _____ Page _____

* Flatten Binary to Linked List:



⇒ 1 2 3 4 5 6 7 8

⇒ If we have to make a singly LL of the Binary Tree.

⇒ We have to make right pointer point to next node.

⇒ So, if we do this then there is a chance of error because many node have right ~~in~~ pointer pointing to different node.

⇒ So, first, we will do that ~~we~~ if left exist then rightmost point to right of curr node.

⇒ Then make null of ~~right~~ node & point right of curr to left node.

Date _____ Page _____

Code
```
while(root){
  if left doesn't exist —
      root = root->right;
  else{ —
      Node * curr = root -> left;
      while( curr -> right)
          curr = curr -> right;
      curr -> right = root ->right;
      root -> right = root -> left;
      root -> left = NULL;
      root = root -> right;
  }
}
```