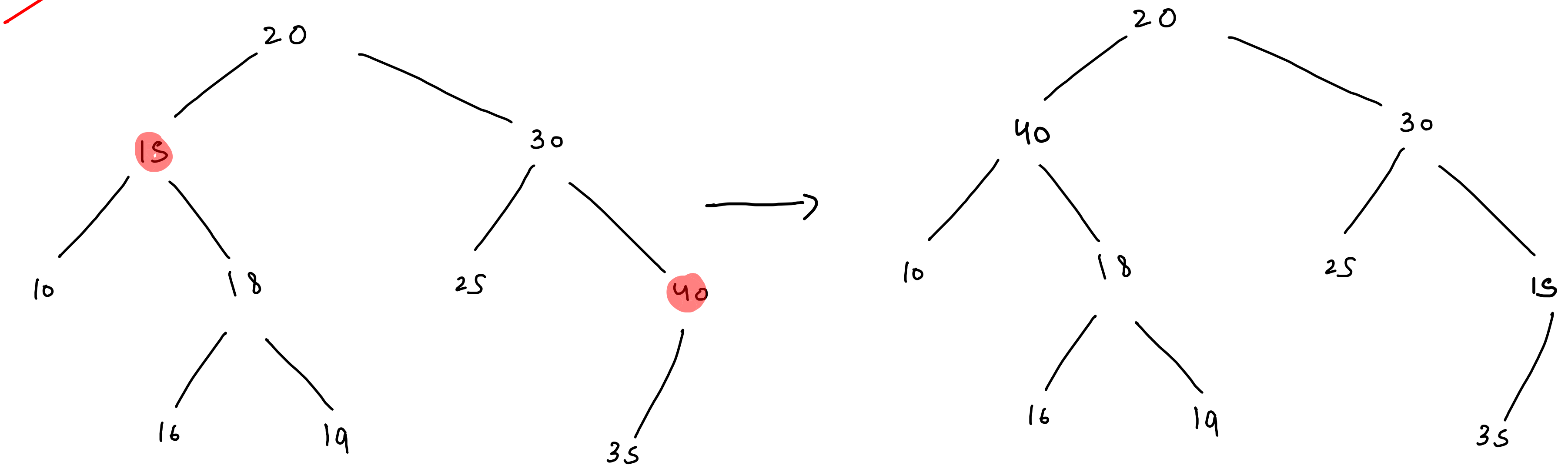
recover BST



Inorder: 10 15 16 18 19 20 25 30 35 40

P_1 C_1 P_2 C_2

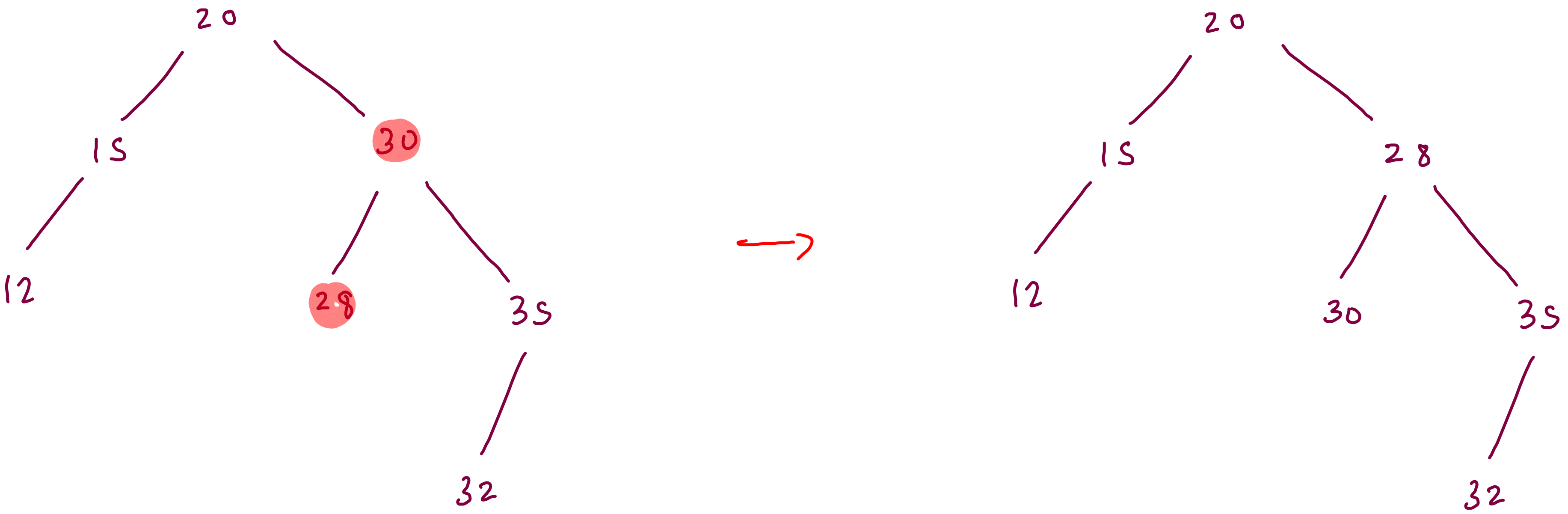
Inorder : 10 40 16 18 19 20 25 30 35 15

first = 40

second = 15

P > C

P_1, C_2



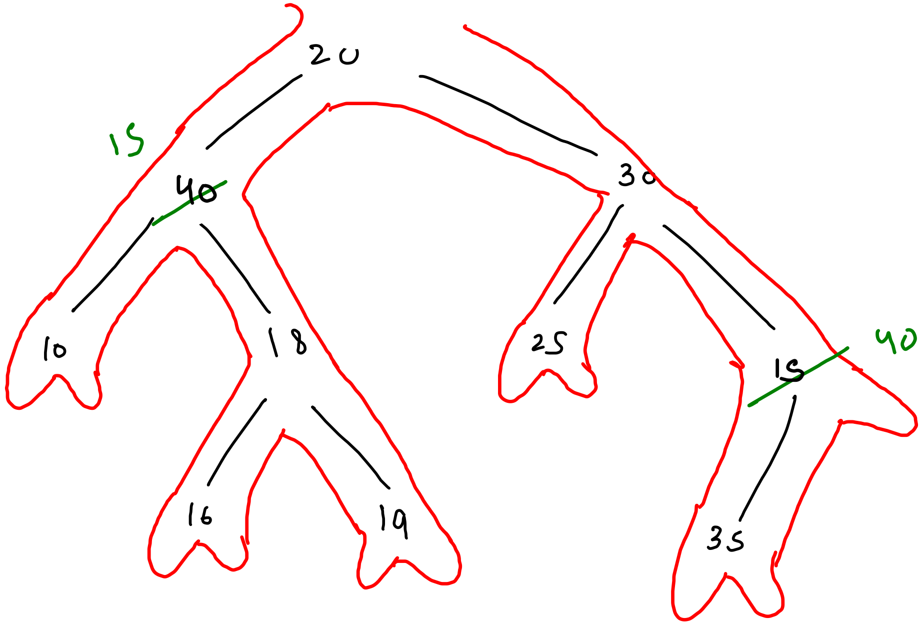
in: 12 15 20 28 30 32 35

in: 12 15 20 30 28 32 35

p c

first = 30

secn = 28



prev = ~~null~~ ~~10~~ ~~40~~ ~~16~~ ~~18~~ ~~19~~ ~~20~~ ~~25~~ ~~30~~ ~~35~~ 15

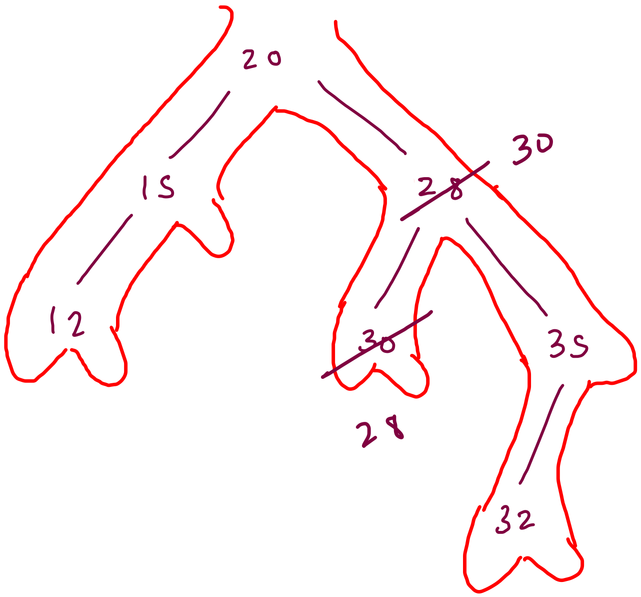
first = ~~null~~ 40

second = ~~null~~ ~~16~~ 15

in; 10 40 16 18 19 20 25 30 35 15

p c

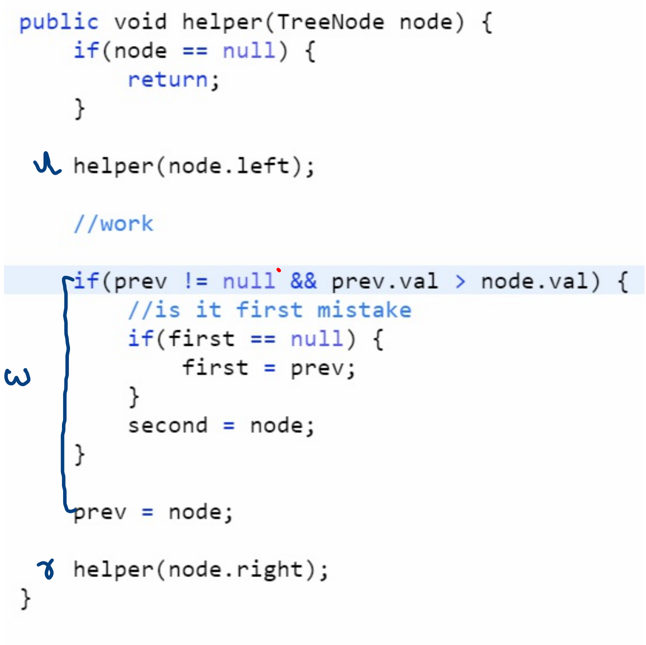
```
call (node.left);
[ check ;
[ prev = curr;

call (node.right);
```



prev = ~~n~~ ~~12~~ ~~15~~ ~~20~~ ~~28~~ ~~30~~ ~~32~~ 35

first = ~~n~~ 30

second = ~~n~~ 28



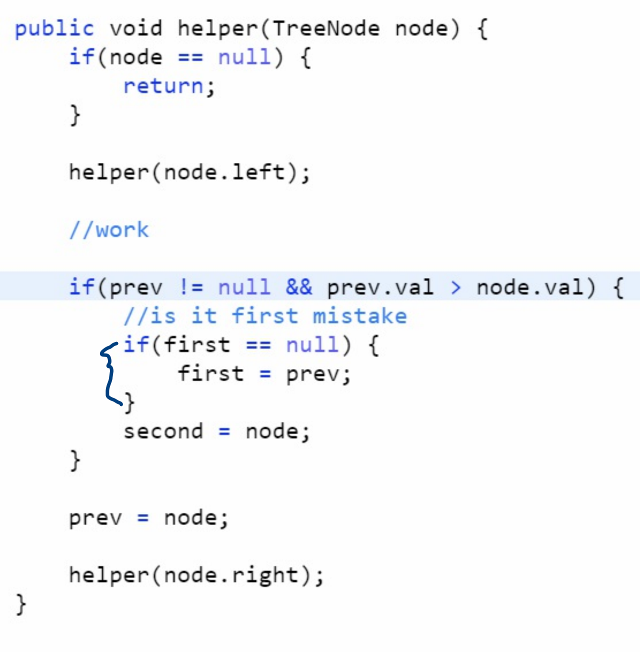
```
public void helper(TreeNode node) {
    if(node == null) {
        return;
    }

  l helper(node.left);

    //work

w   if(prev != null && prev.val > node.val) {
        //is it first mistake
        if(first == null) {
            first = prev;
        }
        second = node;
    }

    prev = node;

  r helper(node.right);
}
```

12 15 20 30 28 32 35

p c

in: 10 40 16 18 19 20 25 30 35 15

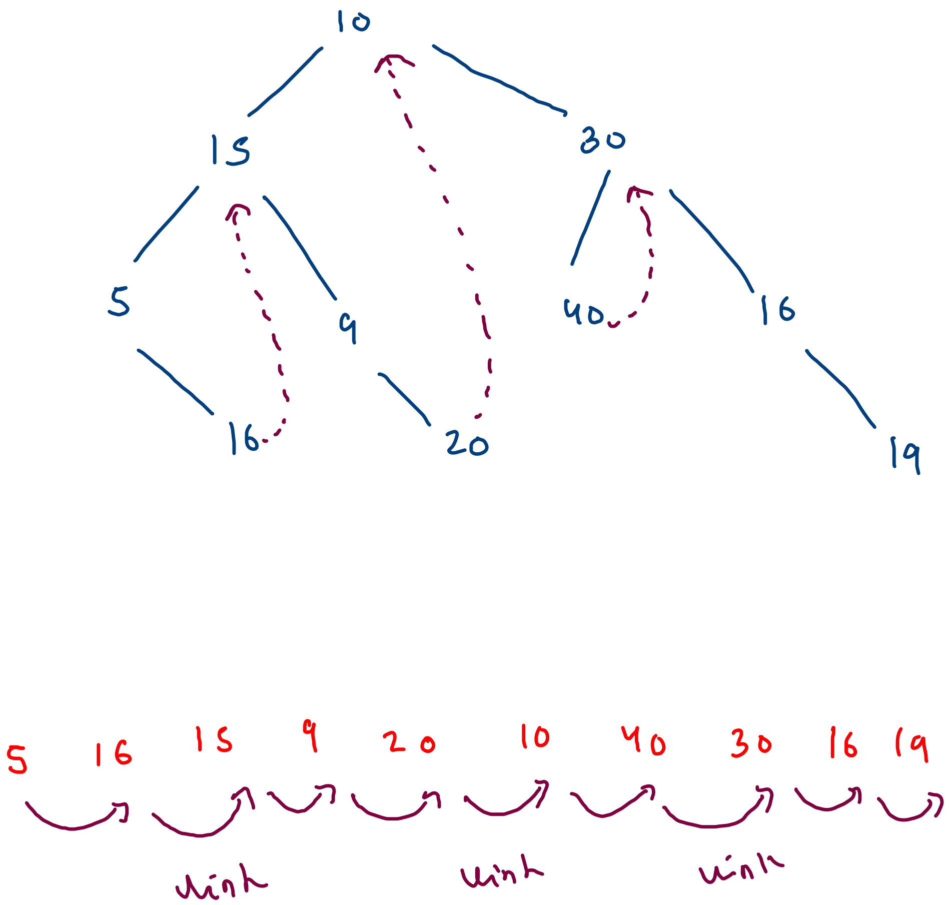
p c



```
public void helper(TreeNode node) {
    if(node == null) {
        return;
    }

    helper(node.left);

    //work

    if(prev != null && prev.val > node.val) {
        //is it first mistake
        if(first == null) {
            first = prev;
        }
        second = node;
    }

    prev = node;

    helper(node.right);
}
```

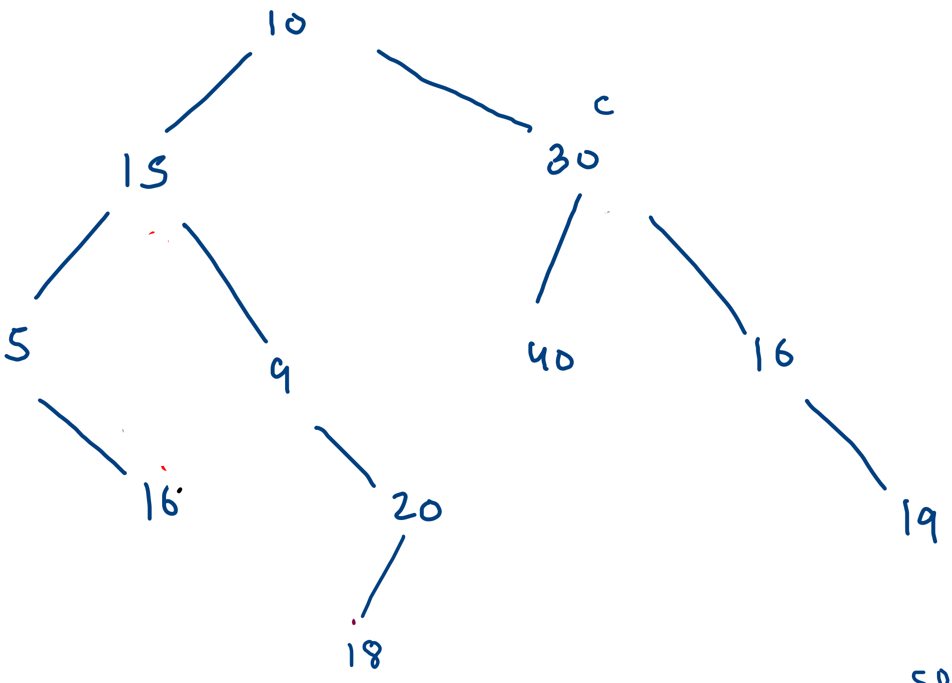
first = ~~n~~ 40

second = ~~n~~ ~~16~~ 15



Inorder

```
call ( node.left );
print ( node.val );
call ( node.right );
```



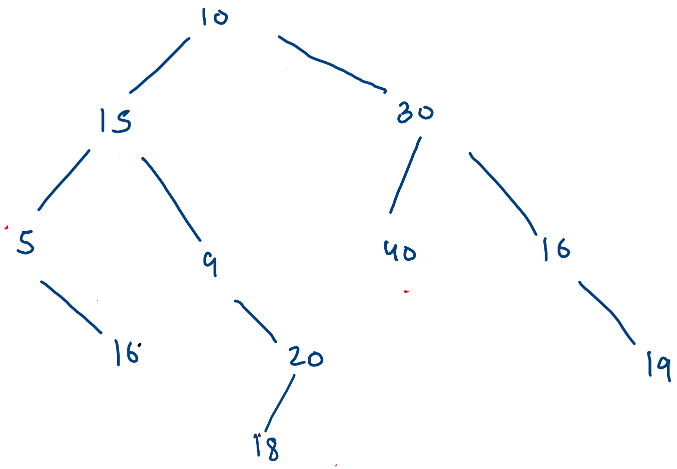
l n r
↓
curr

Inorder :

5 16 15 9 18 20 10

some steps

```
ln = curr.left;
rmn = rightmostNode(ln);
rmn.right = c ;
curr = curr.left;
```



5 16 15 9 18 20 10 40 30 16 19

L Cun R

```
cun = root ;

while (cun != null)    {

    ln = cun. left ;
    if (ln = = null) {
        syso( cun. data) ;
        cun = cun. right ;
    }

    else {
        rmn = rightmost (ln);
        if (rmn. right  = = null ) {
            rmn. right = cun ;
            cun = cun. left ;
        }
        else {
            rmn. right = null;
            syso (cun. data) ;
        }   cun = cun. right ;
    }

}
```

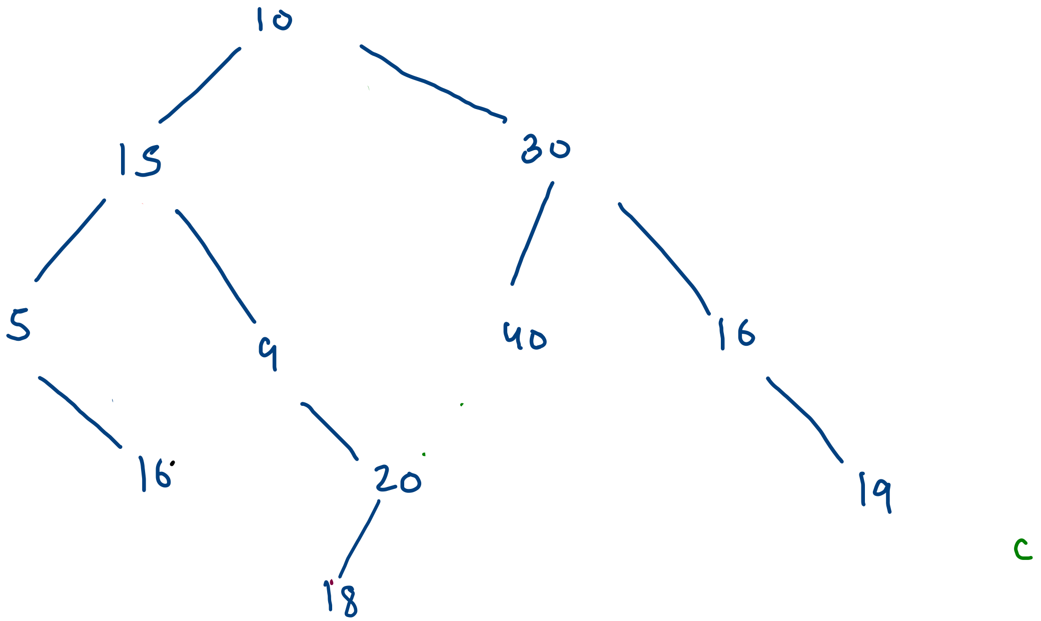
to create thread

if (node. right == null ||

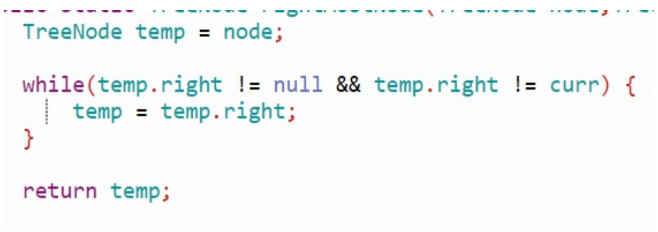
node . right == cur)

indication : left subtree is done

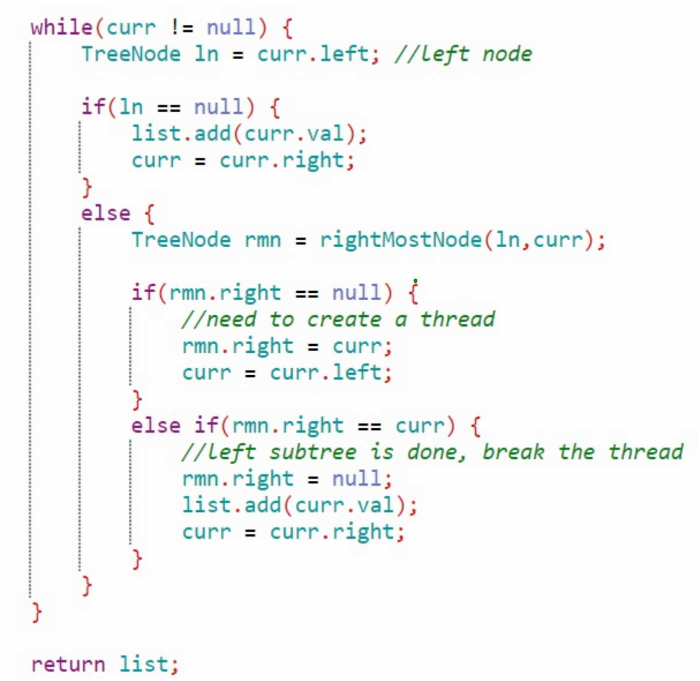
work : to break thread



5 16 15 9 18 20 10 40 30 16 19

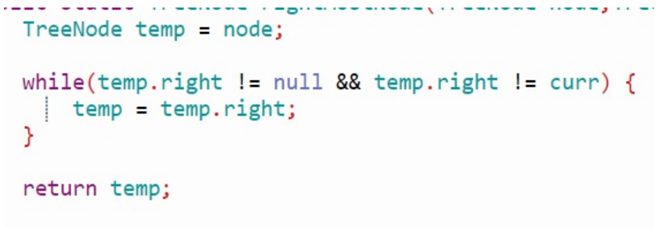


```
TreeNode temp = node;

while(temp.right != null && temp.right != curr) {
    temp = temp.right;
}

return temp;
```

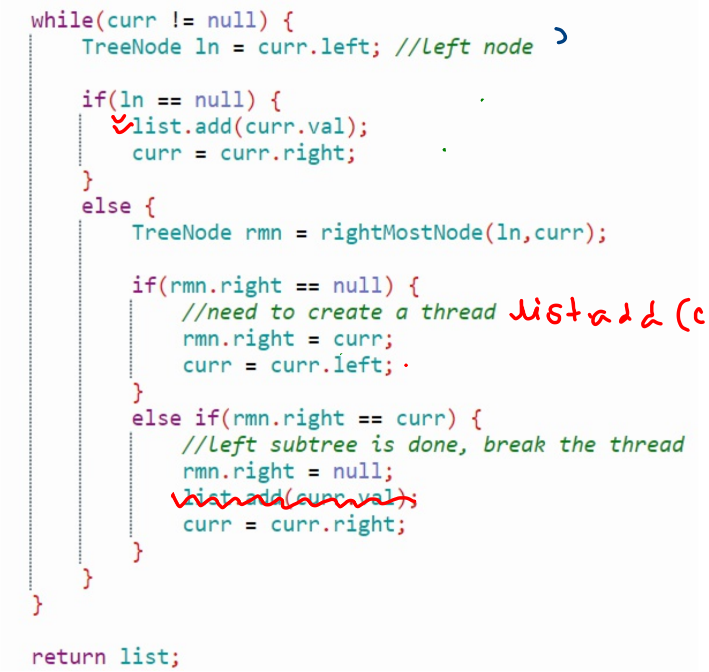


```
while(curr != null) {
    TreeNode ln = curr.left; //Left node

    if(ln == null) {
        list.add(curr.val);
        curr = curr.right;
    }
    else {
        TreeNode rmn = rightMostNode(ln,curr);

        if(rmn.right == null) {
            //need to create a thread
            rmn.right = curr;
            curr = curr.left;
        }
        else if(rmn.right == curr) {
            //Left subtree is done, break the thread
            rmn.right = null;
            list.add(curr.val);
            curr = curr.right;
        }
    }
}

return list;
```

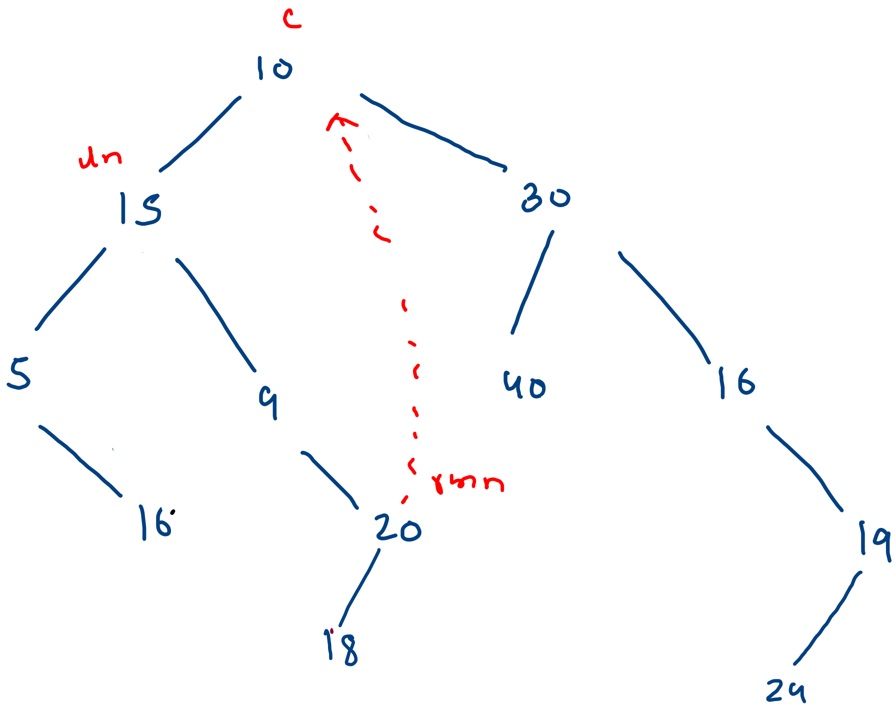
pre



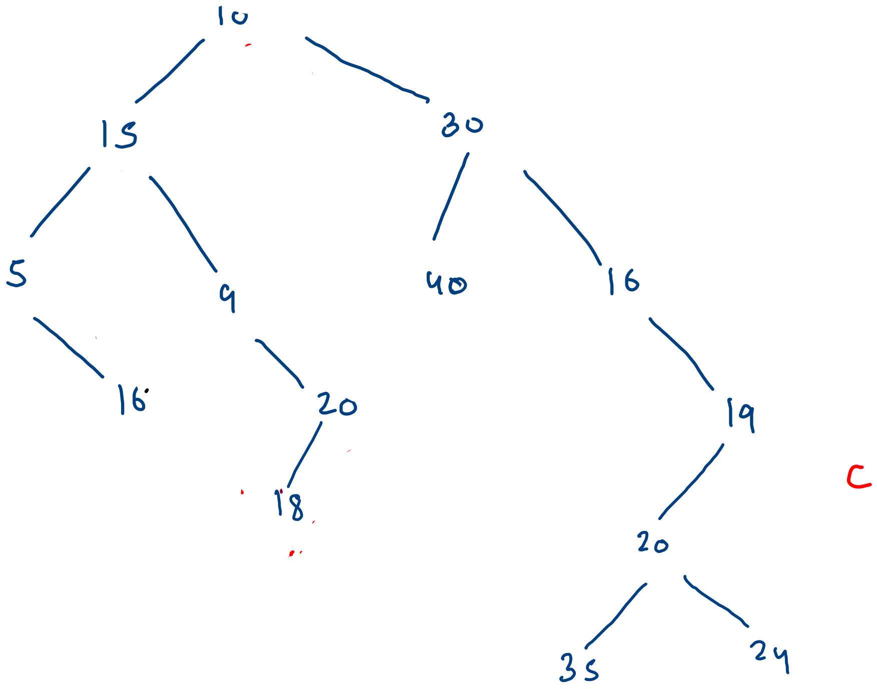
```
TreeNode temp = node;

while(temp.right != null && temp.right != curr) {
    temp = temp.right;
}

return temp;
```

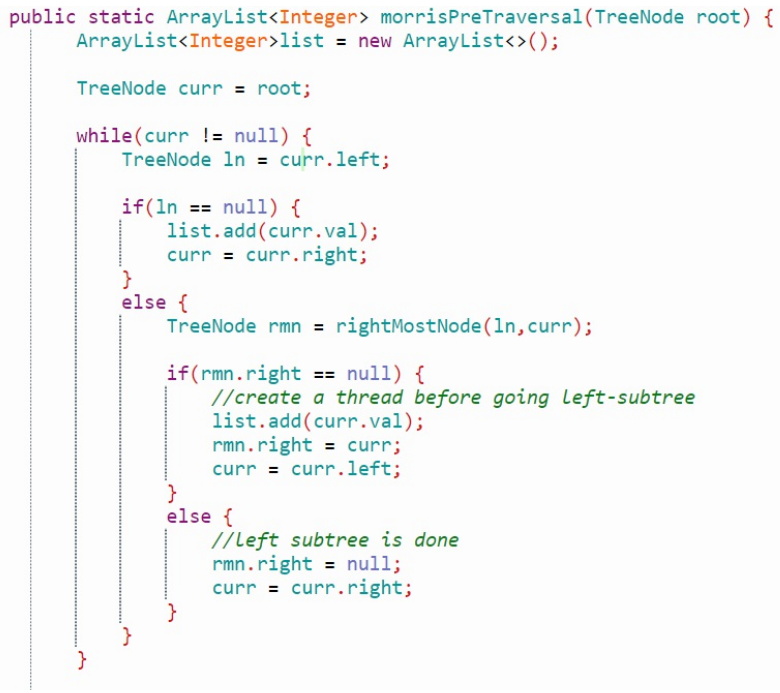


```
while(curr != null) {
    TreeNode ln = curr.left; //left node

    if(ln == null) {
        list.add(curr.val);
        curr = curr.right;
    }
    else {
        TreeNode rmn = rightMostNode(ln,curr);

        if(rmn.right == null) {
            //need to create a thread listadd(curr.val);
            rmn.right = curr;
            curr = curr.left;
        }
        else if(rmn.right == curr) {
            //left subtree is done, break the thread
            rmn.right = null;
            ~~list.add(curr.val);~~
            curr = curr.right;
        }
    }
}

return list;
```

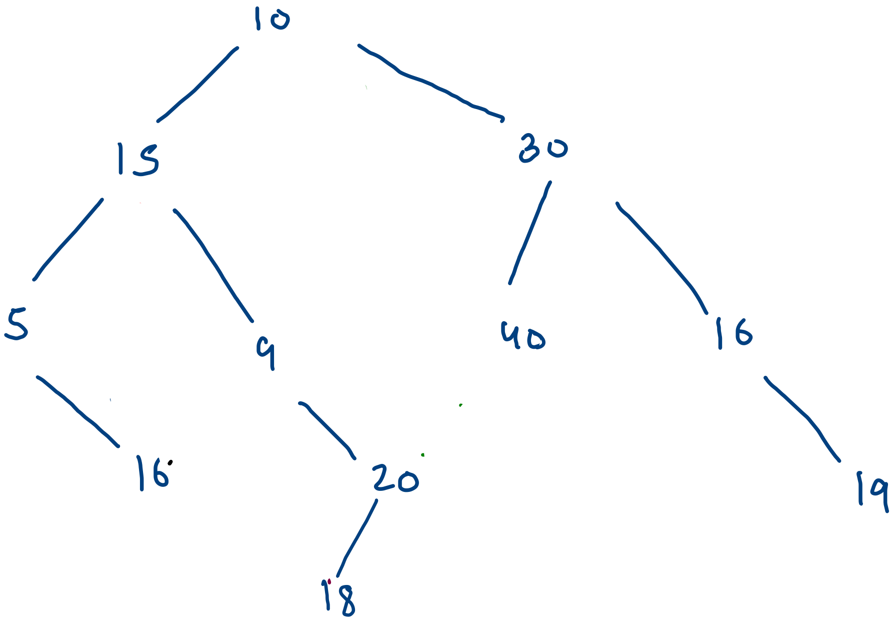


NLR




```
public static ArrayList<Integer> morrisPreTraversal(TreeNode root) {
    ArrayList<Integer>list = new ArrayList<>();

    TreeNode curr = root;

    while(curr != null) {
        TreeNode ln = curr.left;

        if(ln == null) {
            list.add(curr.val);
            curr = curr.right;
        }
        else {
            TreeNode rmn = rightMostNode(ln,curr);

            if(rmn.right == null) {
                //create a thread before going left-subtree
                list.add(curr.val);
                rmn.right = curr;
                curr = curr.left;
            }
            else {
                //left subtree is done
                rmn.right = null;
                curr = curr.right;
            }
        }
    }
```

pre: 10 15 5 16 9 20 18 30 40 16 19 20 35 24



Postorder

LRN

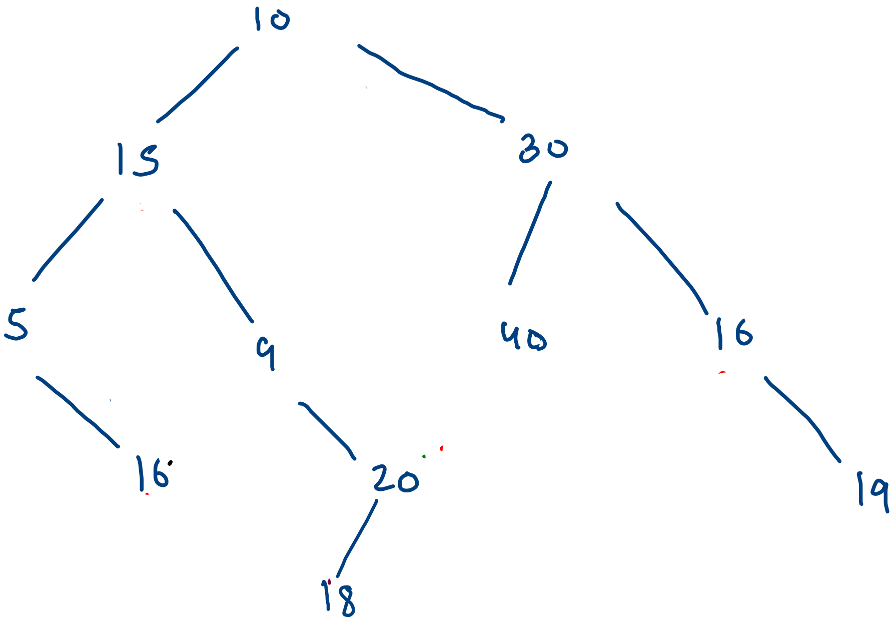
reverse euler preorder

NRL

```
print(node.val);
call(node.right);
call(node.left);
```

postorder = rev (reverse euler preorder)

= rev (NRL) = LRN

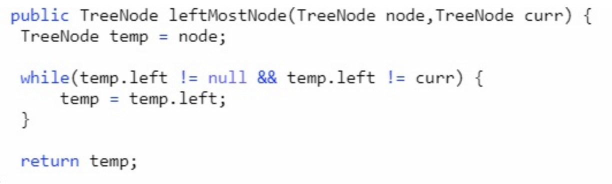


c.

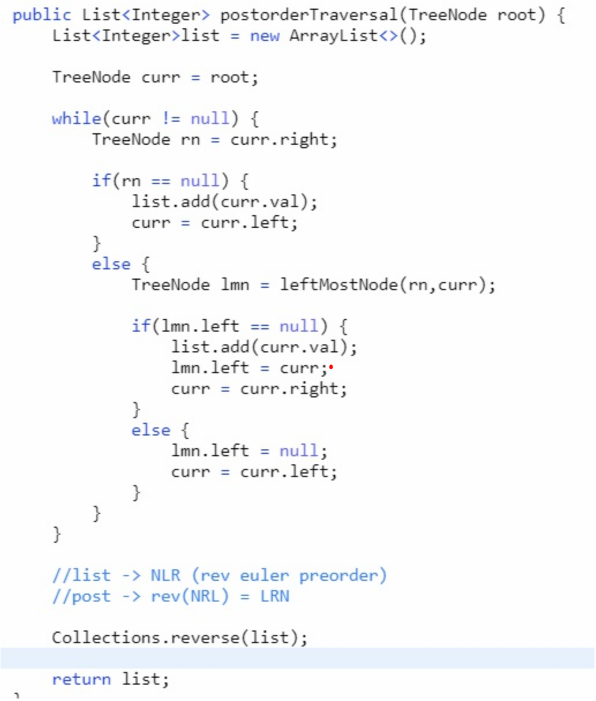
rev Euler preorder NRL

10 30 16 19 40 15 9 20 18 5 16

postorder. ✓ 16 5 18 20 9 15 40 19 16 30 10



```
public TreeNode leftMostNode(TreeNode node,TreeNode curr) {
 TreeNode temp = node;

 while(temp.left != null && temp.left != curr) {
     temp = temp.left;
 }

 return temp;
```



```
public List<Integer> postorderTraversal(TreeNode root) {
    List<Integer>list = new ArrayList<>();

    TreeNode curr = root;

    while(curr != null) {
        TreeNode rn = curr.right;

        if(rn == null) {
            list.add(curr.val);
            curr = curr.left;
        }
        else {
            TreeNode lmn = leftMostNode(rn,curr);

            if(lmn.left == null) {
                list.add(curr.val);
                lmn.left = curr;
                curr = curr.right;
            }
            else {
                lmn.left = null;
                curr = curr.left;
            }
        }
    }

    //list -> NLR (rev euler preorder)
    //post -> rev(NRL) = LRN

    Collections.reverse(list);

    return list;
```