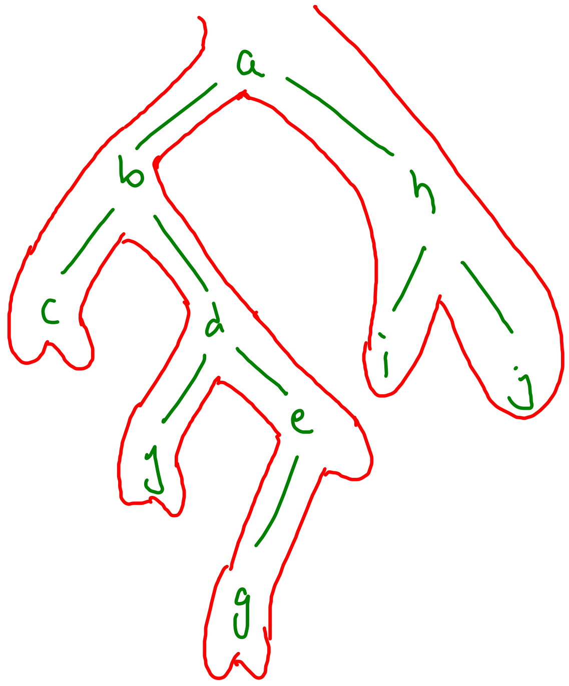
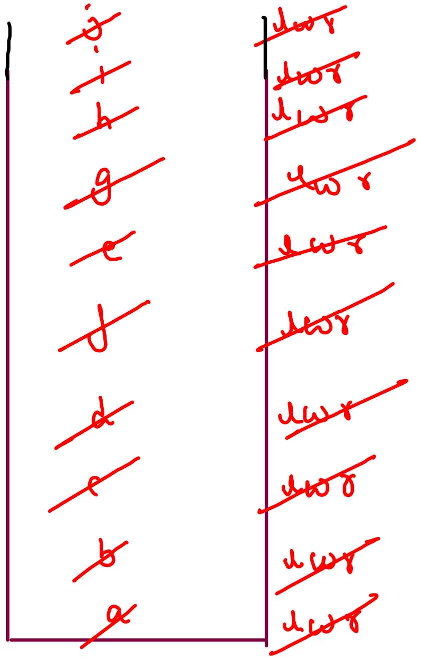
In Order Morris Traversal In Binarytree

travel the binary tree in o(1) space



space : o(h)

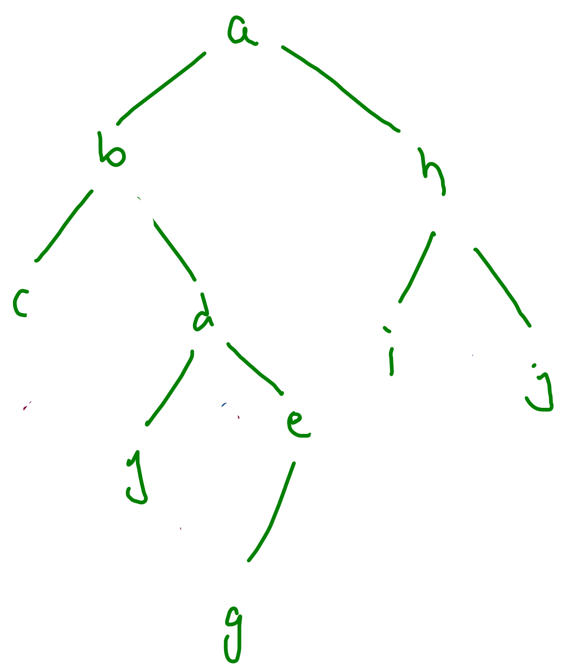


DFS

lwr

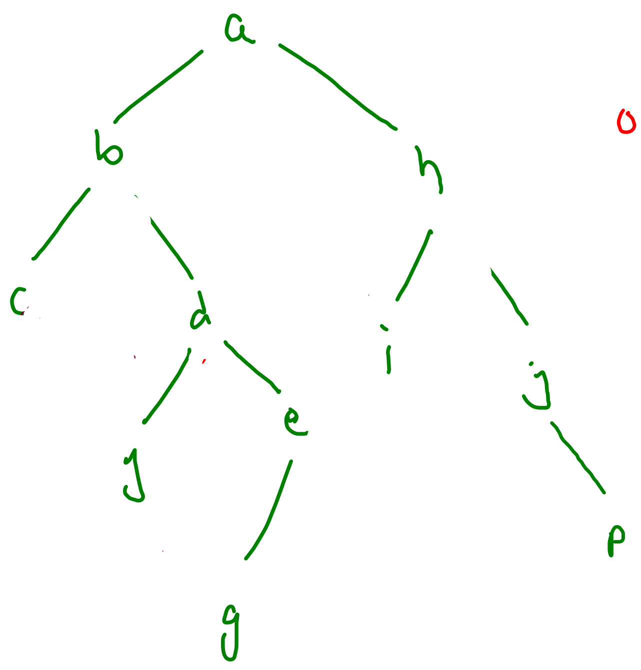
inorder :

c b j d g e a i h j



inorder : c b j d g e a i h j

```
while (curr != null) -->

    lc =   curr.left ;

    if (lc == null) {
        syso (curr.val) ;
        curr = curr.right ;
    }

    else {
        Node rmn = rightMostNode(lc,curr);
        if (rmn.right == null) {
            rmn.right = curr;
            curr = curr.left;
        }
        else if (rmn.right == curr) {
            // left subtree v.s
            syso (curr.val);
            rmn. right = null;
            curr = curr.right;
    }   }
}
```

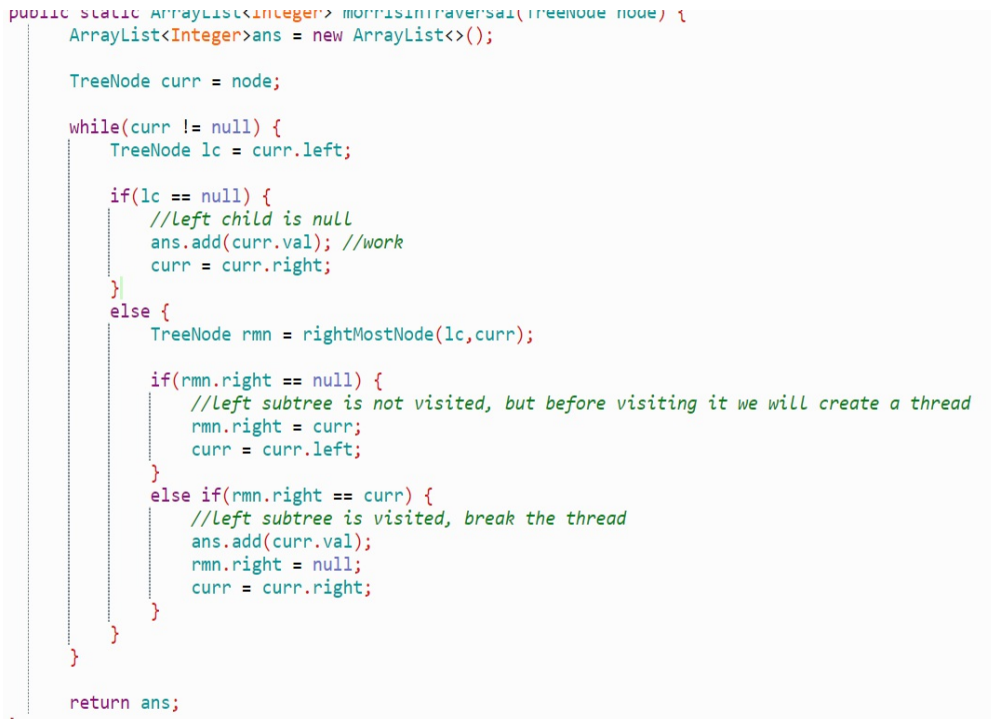


$O(3n)$: T

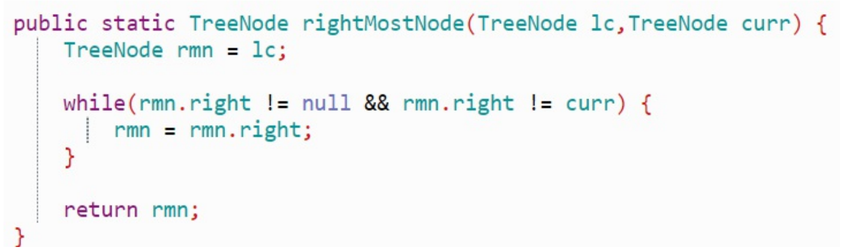
$O(1)$: S

c

ans: c b j d g e a i h j p



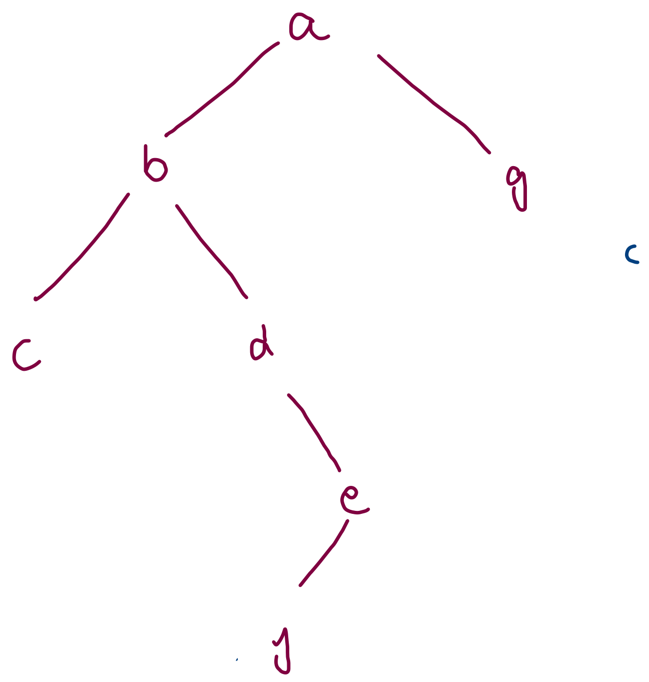
```java
public static ArrayList<Integer> morrisInTraversal(TreeNode node) {
    ArrayList<Integer>ans = new ArrayList<>();

    TreeNode curr = node;

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

        if(lc == null) {
            //left child is null
            ans.add(curr.val); //work
            curr = curr.right;
        }
        else {
            TreeNode rmn = rightMostNode(lc,curr);

            if(rmn.right == null) {
                //left subtree is not visited, but before visiting it we will create a thread
                rmn.right = curr;
                curr = curr.left;
            }
            else if(rmn.right == curr) {
                //left subtree is visited, break the thread
                ans.add(curr.val);
                rmn.right = null;
                curr = curr.right;
            }
        }
    }

    return ans;
```

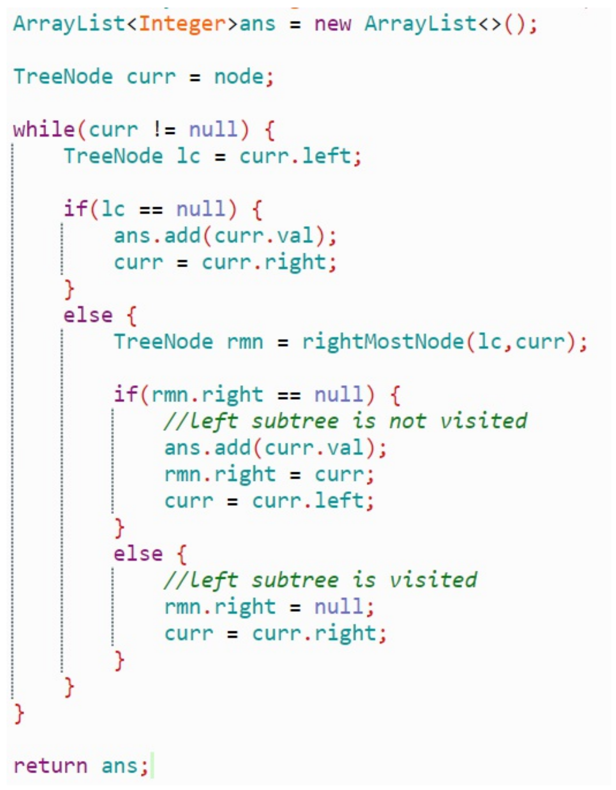


```java
public static TreeNode rightMostNode(TreeNode lc,TreeNode curr) {
    TreeNode rmn = lc;

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

    return rmn;
}
```

Preorder

WLR



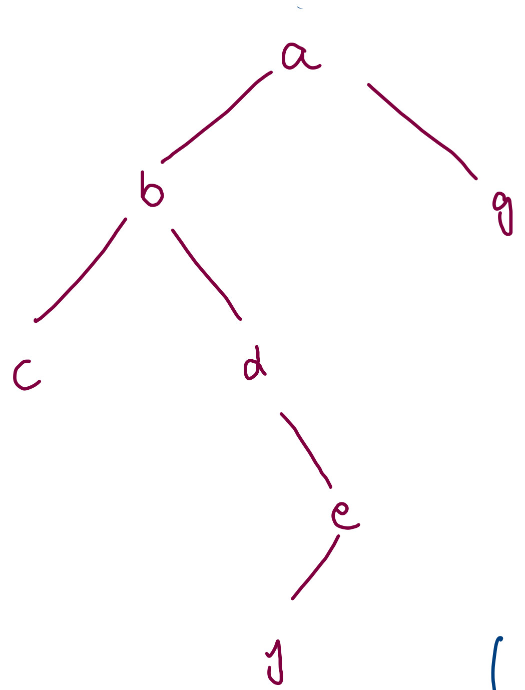


```
ArrayList<Integer>ans = new ArrayList<>();

TreeNode curr = node;

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

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

        if(rmn.right == null) {
            //left subtree is not visited
            ans.add(curr.val);
            rmn.right = curr;
            curr = curr.left;
        }
        else {
            //left subtree is visited
            rmn.right = null;
            curr = curr.right;
        }
    }
}

return ans;
```

Pre : a b c d e f g

Postorder

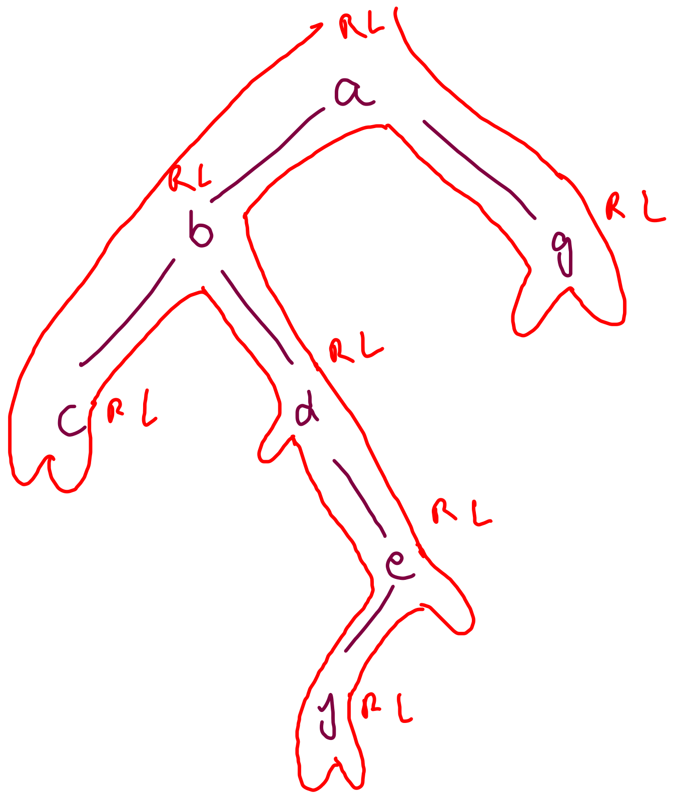
Morris : Inorder, Preorder



	Regular euler	reverse euler
Pre	NLR	NRL
In	LNR	RNL
Post	LRN	RLN

rev (reverse euler Pre (NRL)) $\rightarrow$ reg. euler postorder

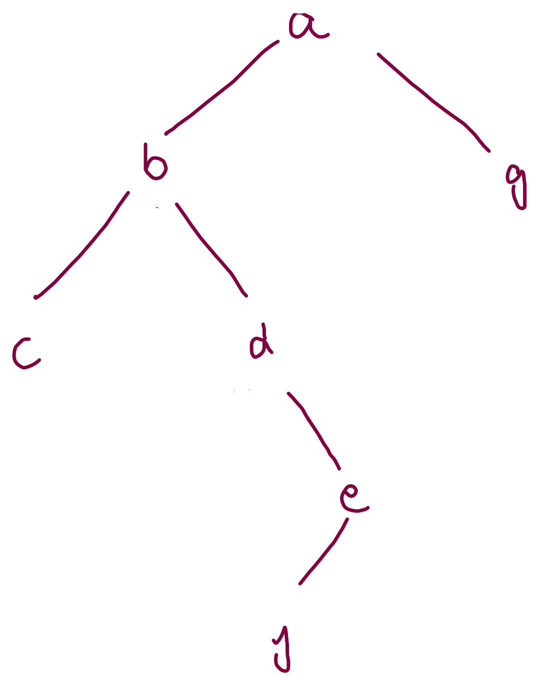
LRN



rev. euler preorder (N R L)

rev (a g b d e j c) → c j e d b g a

N R L

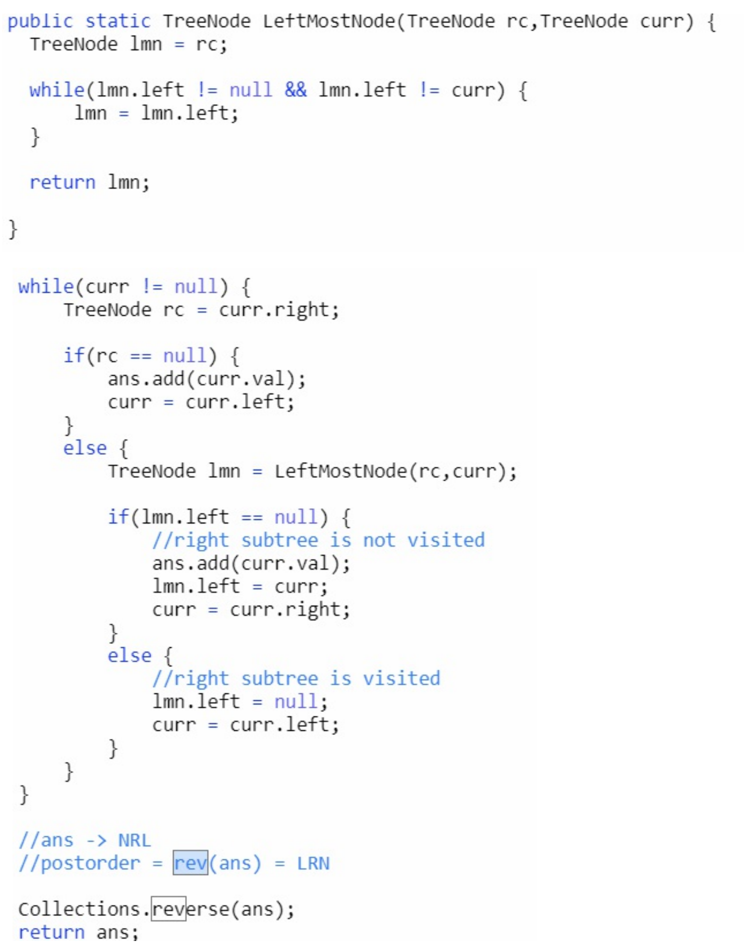


ans: a g b d e j c

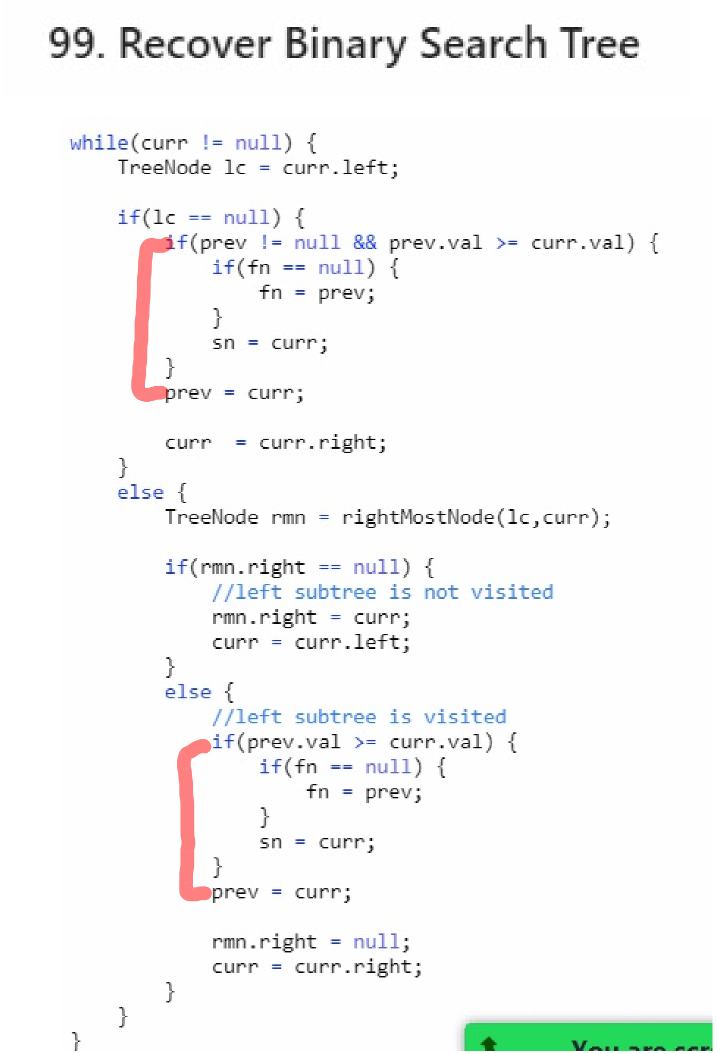
(NRL)

$\downarrow$ rev

c j e d b g a



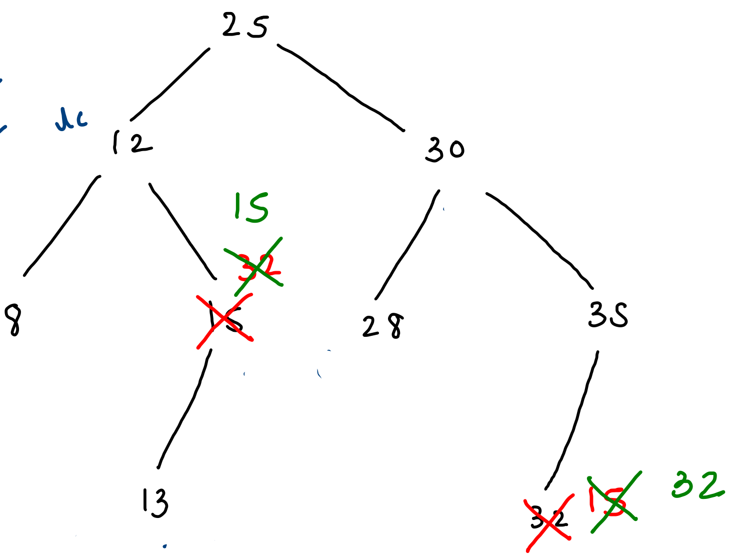
```
public static TreeNode LeftMostNode(TreeNode rc,TreeNode curr) {
  TreeNode lmn = rc;

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

  return lmn;

}

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

     if(rc == null) {
         ans.add(curr.val);
         curr = curr.left;
     }
     else {
         TreeNode lmn = LeftMostNode(rc,curr);

         if(lmn.left == null) {
             //right subtree is not visited
             ans.add(curr.val);
             lmn.left = curr;
             curr = curr.right;
         }
         else {
             //right subtree is visited
             lmn.left = null;
             curr = curr.left;
         }
     }
 }

 //ans -> NRL
 //postorder = rev(ans) = LRN

 Collections.reverse(ans);
 return ans;
```



99. Recover Binary Search Tree

```java
while(curr != null) {
    TreeNode lc = curr.left;

    if(lc == null) {
        if(prev != null && prev.val >= curr.val) {
            if(fn == null) {
                fn = prev;
            }
            sn = curr;
        }
        prev = curr;

        curr  = curr.right;
    }
    else {
        TreeNode rmn = rightMostNode(lc,curr);

        if(rmn.right == null) {
            //left subtree is not visited
            rmn.right = curr;
            curr = curr.left;
        }
        else {
            //left subtree is visited
            if(prev.val >= curr.val) {
                if(fn == null) {
                    fn = prev;
                }
                sn = curr;
            }
            prev = curr;

            rmn.right = null;
            curr = curr.right;
        }
    }
}
```

p = ~~n~~
~~8~~
~~12~~
~~13~~
~~32~~
~~25~~
~~28~~
~~30~~
~~15~~
35



Morris

(advance)

c

fn = ~~n~~ 32

sn = ~~n~~ ~~28~~ 15