Lecture : Trees-1

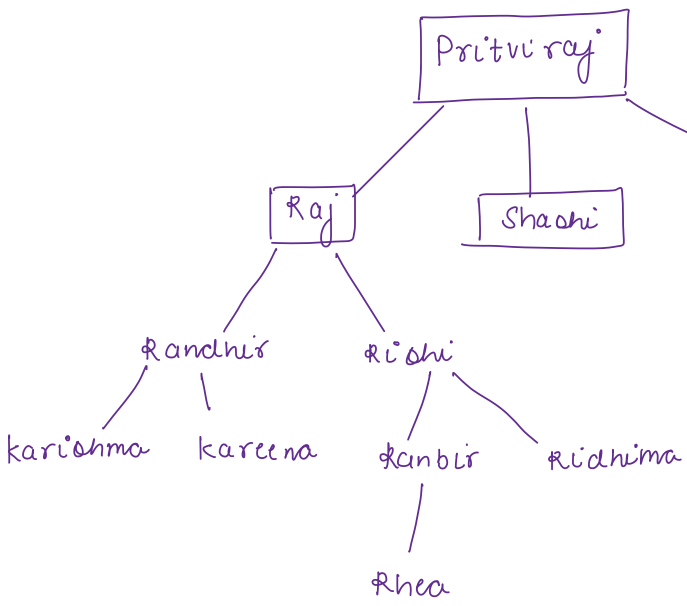
Agenda

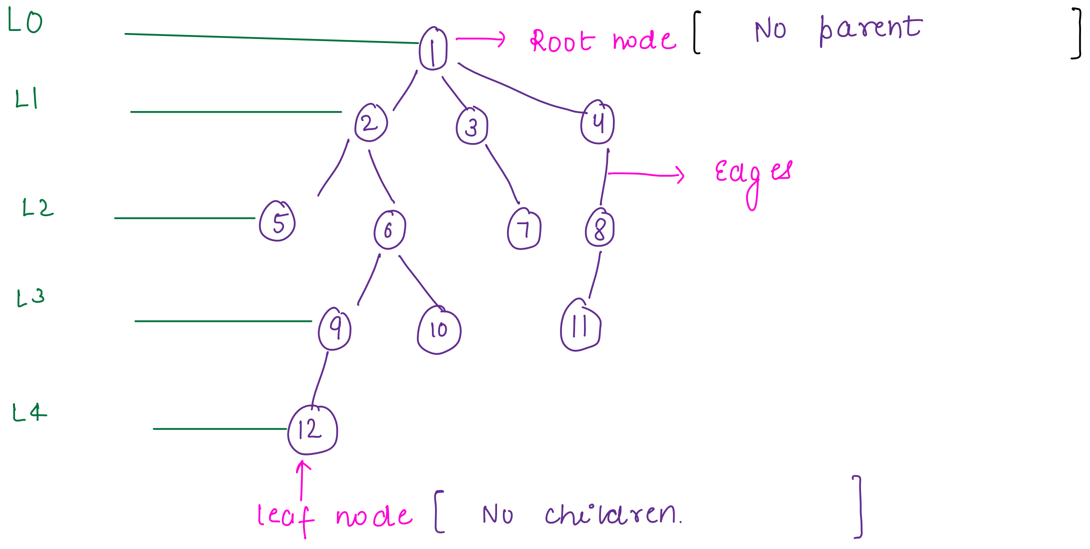
- Introduction
- Traversals
- Iterative traversal (Inorder)
- Construct tree from pre and in.

Linear data structures :- Arrays. stacks, queues, LL etc.

Non-linear / Heirarical data structures :- Trees.

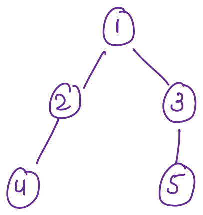
Ex: family tree, organisation structure.



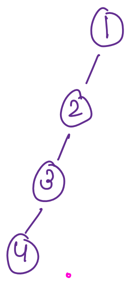


Binary trees:

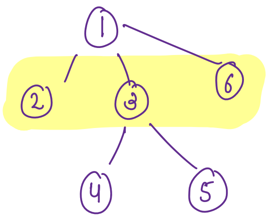
Max of 2 children. [0, 1, 2, $\cancel{3}$]



Valid



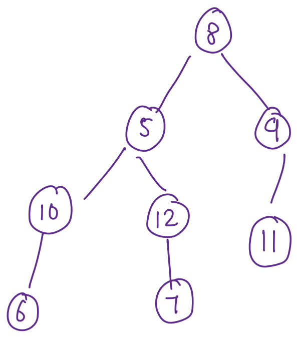
Valid



Invalid

Structure of tree

```
class Node {
    int val;
    Node left;
    Node right;
    Node( x) {
        val = x;
        left = null;
        right = null;
    }
}
```

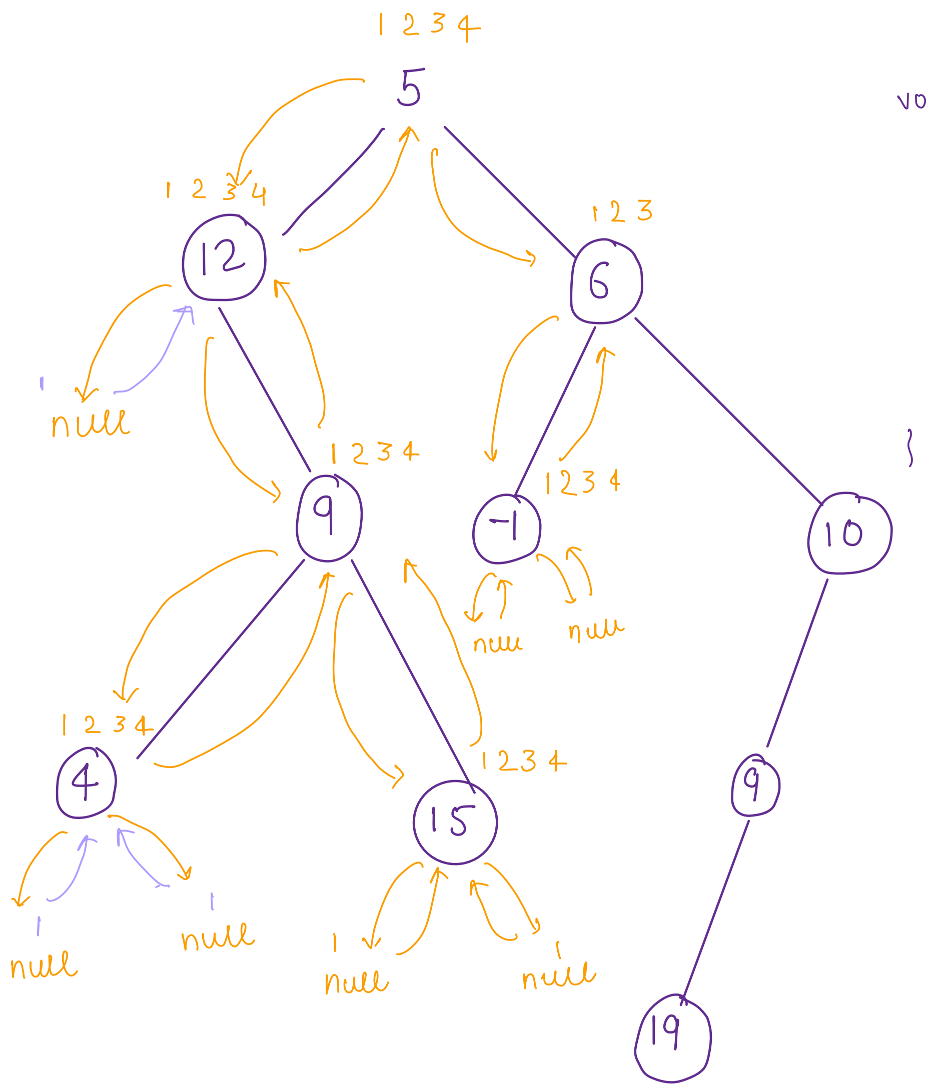


Tree traversals

1.	Inorder	L	D	R
2.	Preorder	D	L	R
3.	Postorder	L	R	D

Inorder

L D R



```
void inorder( root)  {
  1.  if (root == null) {
          return;
      }
  2.  inorder(root.left);
  3.  print (root.val);
  4.  inorder( root.right);
}
```

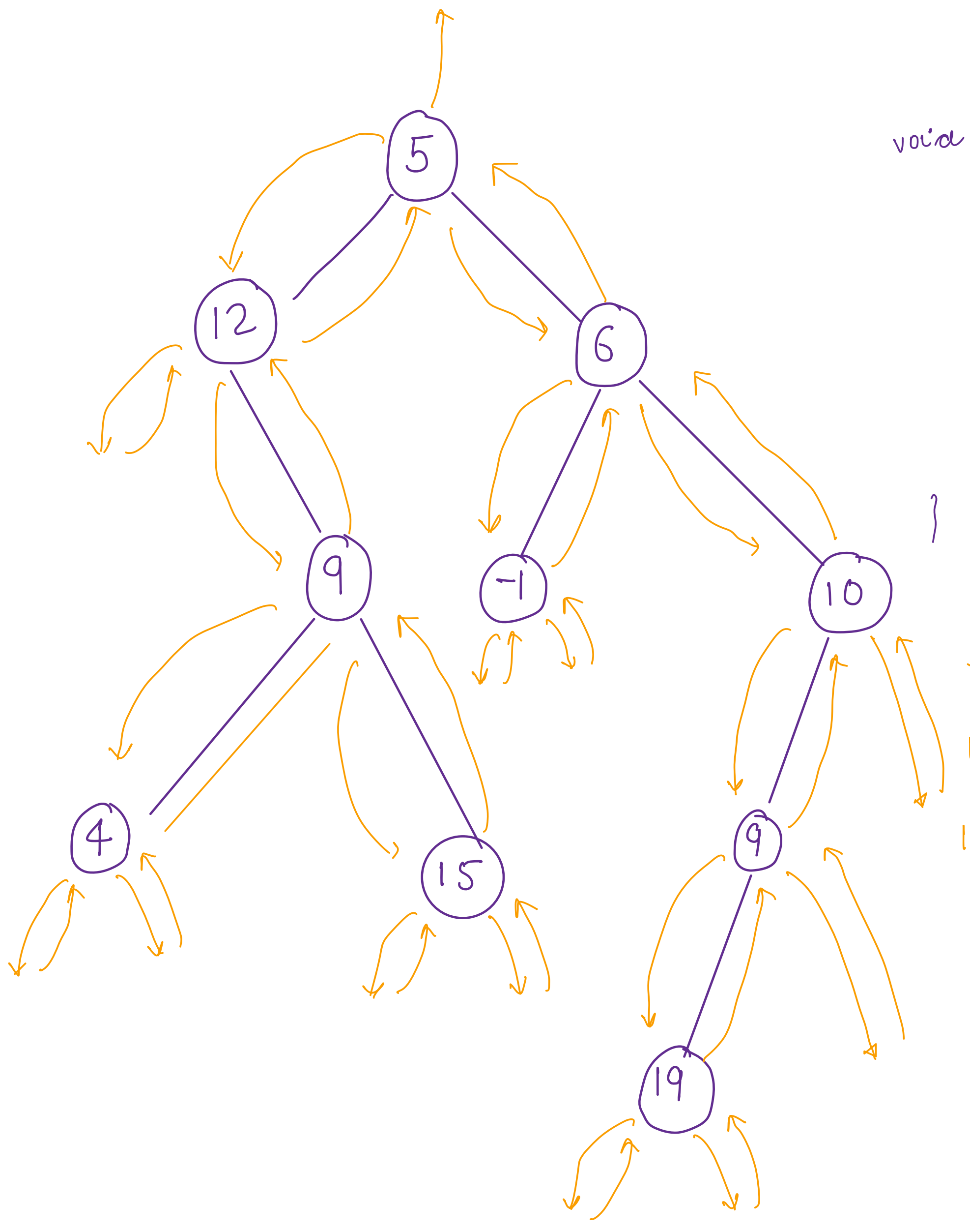
12	4	9	
15	5	-1	
6	19	9	10

TC: O(n)

SC: O(height)

Preorder

D L R.



```
void preorder(root){
    if(root==null){
        return;
    }
    print(root.data);
    preorder(root.left);
    preorder(root.right);
}
```

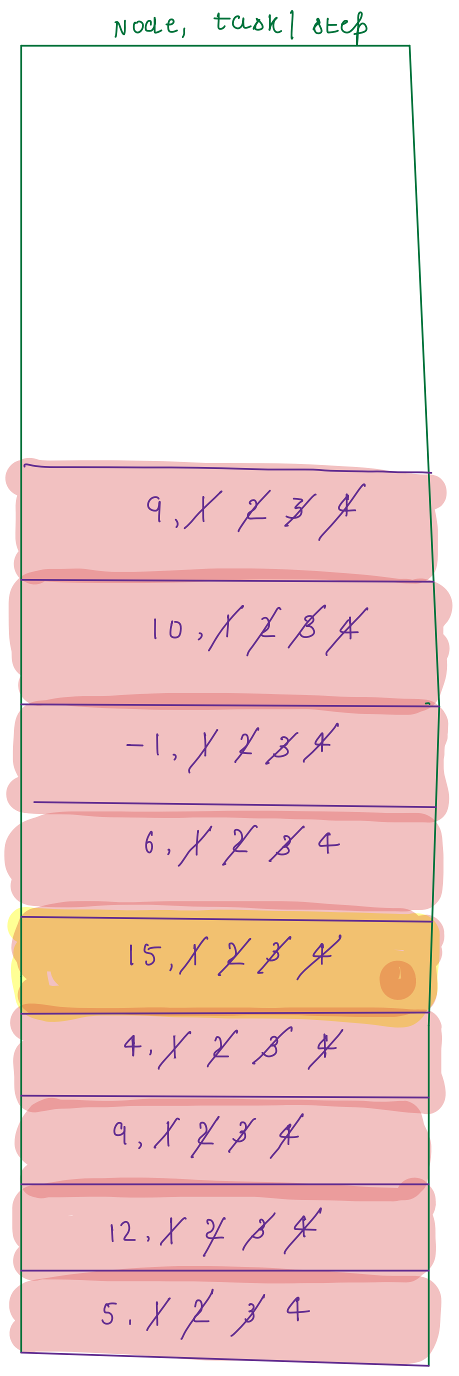
5	12	9	4
15	6	-1	
10	9	19	

Qu: Print inorder traversal of tree [Iterative] OCI ☆☆

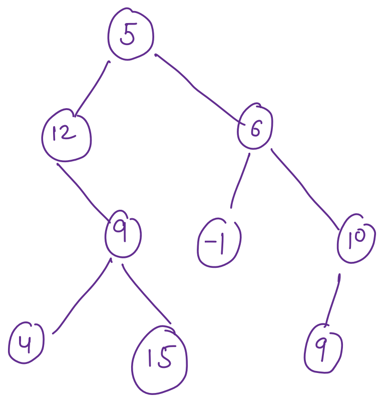
Steps :

1. Call left child
2. Print data
3. Call right child
4. Get out of stack | Return.

Stack



```
class Pair {
        Node node;
        int task;
        Pair ( x) {
            node = x;
            task = 1;
        }
}
```



12 4 9 15 5 -1 6 9 10

```
void inorder( Node root) {
        Stack < Pair> stack = new Stack<>();
        Pair p = new Pair (root);     //  (5, 1)
        stack. push (p);

        while( ! stack. is Empty()) {
              Pair  top  =  stack. peek();
              Node  curr  =  top. node;
              if ( top. task == 1) {
                    if( curr. left != null) {
                          p = new Node( curr.left);
                          stack. push(p);
                    }
                 top . task ++;
        } else if ( top. task == 2) {
                 print( curr. val);
                 top . task ++;
        } else if ( top. task == 3) {
                 if( curr. right != null) {
                       p = new Pair( curr. right);
                       stack. push( p);
                 }
                 top . task ++;
       } else {
              stack. pop();
       }
     }
}
```

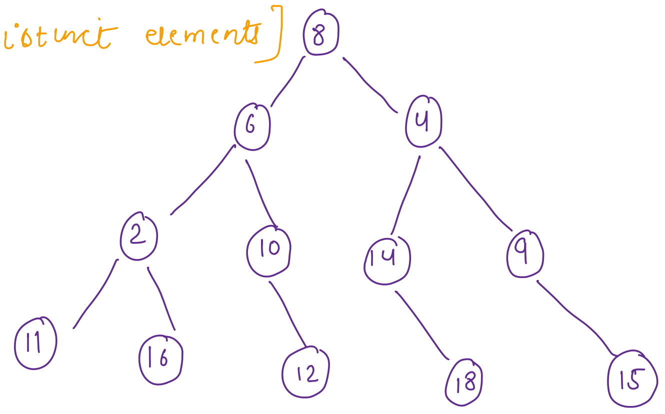
TC: O(n)

SC: O (height of tree)

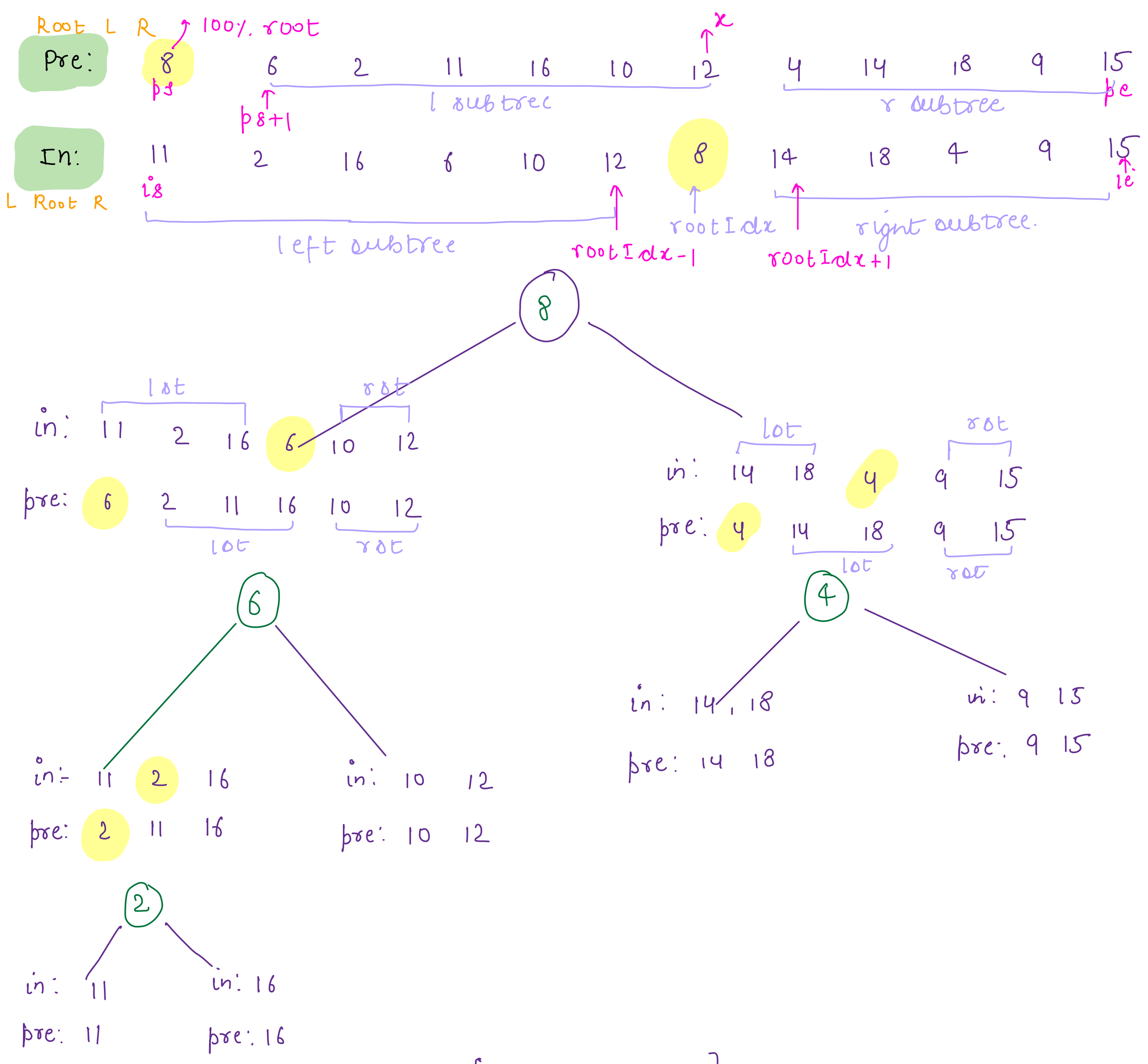
Break: 8 : 31 AM

Qu Given pre[] and in[]. create binary tree of it.

[Distinct elements]



Pre:	8	6	2	11	16	10	12	4	14	18	9	15
	↑ ps											↑ pe
In:	11	2	16	6	10	12	8	14	18	4	9	15
	↑ is											↑ ie



Elements in l subtree: $[\ i_s\ ,\ rootIdx-1\ ]$

$= rootIdx - \cancel{1} - i_s + \cancel{1}$

$= rootIdx - i_s$

$\downarrow$

elements in l subtree $= [\ p_s + 1\ ,\ x\ ]$

elements in lsubtree $= x - (p_s + 1) + 1.$

$= x - p_s \cancel{-1} \cancel{+1}$

$x = p_s + elementInLsubtree$

```
Node   create ( pre[], in[] ) {
       if (pre.length != in.length) {
            return null;
       }
       // compare all the el should be in pre & in be same.
    return helper ( pre, in, 0, pre.length-1, 0 , in.length-1);
}


Node  helper( pre[], in[], ps, pe, is, ie) {
      if ( ps > pe   ||  is > ie) {
             return null;
      }

    int   rootData  =  pre[ ps ];

    Node   root  =  new  Node( rootData);

    int   rootIdx =  search ( root , in , is , ie);

    int   elInLST  =  rootIdx - is;
    root.left  =  helper( pre , in , ps + 1, ps + elinLST ,  is,
                                           rootIdx - 1);

    root.right = helper( pre, in, ps + elinLST + 1 ,  pe,
                                  rootIdx + 1,  ie) ;


    return  root;
}
```

Thank you ☺

Doubts

$\langle \cancel{2}, 3, 8, 11, 1, 6 \rangle$

$\uparrow$

$O(n)$ $\qquad\qquad$ $O(1)$

$\langle 3, 8, 11, 1, 6 \rangle$

$$\begin{array}{c|cccccccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\text{pre} & \overset{}{8}_{\uparrow ps} & 6_{ps} & 2 & 11 & 16 & 10 & 12^{\uparrow pe} & 4 & 14 & 18 & 9 & 15_{\uparrow pe} \\
\text{in} & 11_{\uparrow is} & 2 & 16 & 6 & 10 & 12_{\uparrow ie} & 8 & 14 & 18 & 4 & 9 & 15_{\uparrow ie}
\end{array}$$

rootData = $\boxed{8}$

rootIdx = 6

elinLST = 6 − 0 = 6

ps = 1
pe = 6
is = 0
ie = 5
rootData = $\boxed{6}$
elinLST = 3
rootIdx = 3

ps = 2
pe = 1 + 3 = 4
is = 0
ie = 2
rootData = pre[ps] = $\boxed{2}$

```
                                   0     11    0    11
Node helper( pre[], in[], ps, pe, is, ie) {
  1. if ( ps > pe   ||   is > ie) {
             return null;
     }
  2. int rootData = pre[ps];

     Node root = new Node(rootData);

     int rootIdx = search(root, in, is, ie);

     int elInLST = rootIdx - is;
     root.left = helper(pre, in, ps+1, ps + elinLST, is,
                                           rootIdx-1);

     root.right = helper(pre, in, ps + elinLST+1, pe,
                                    rootIdx+1, ie);


     return root;
}
```