Binary Tree Construction:

Root Node

Left subtree

Right subtree

PreOrder → 10 20 40 50 80 30 60 90 70

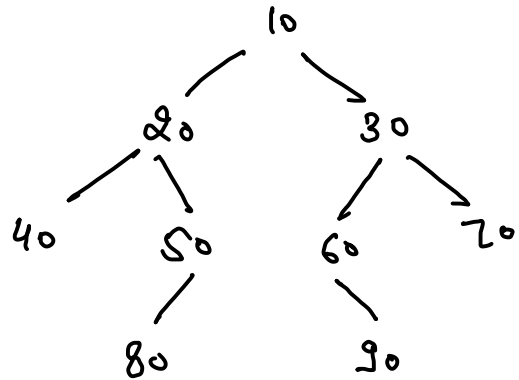
InOrder → 40 20 80 50 10 60 90 30 70

Left Subtree

Right

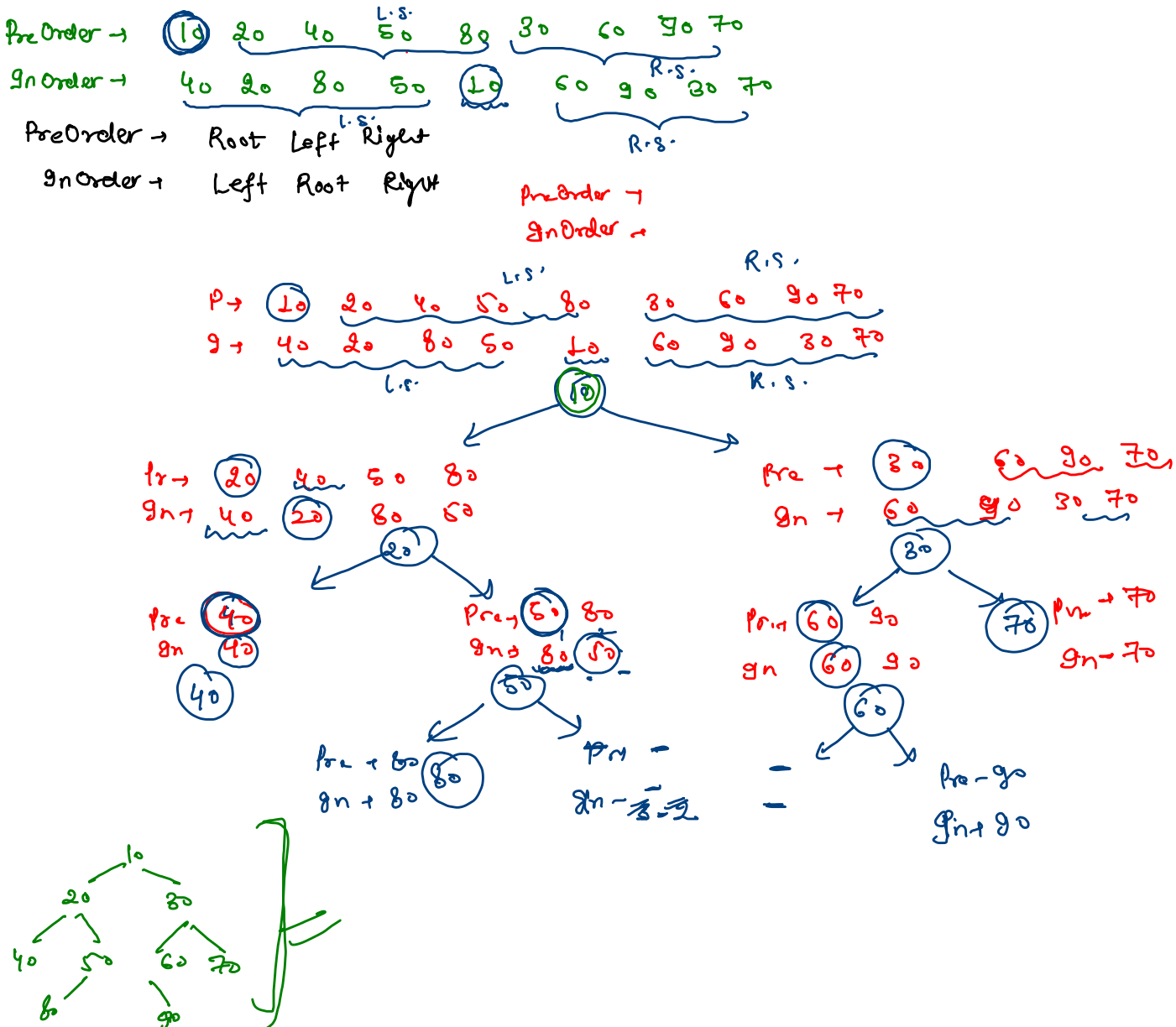
PreOrder → Root Node | Left Subtree | Right Subtree

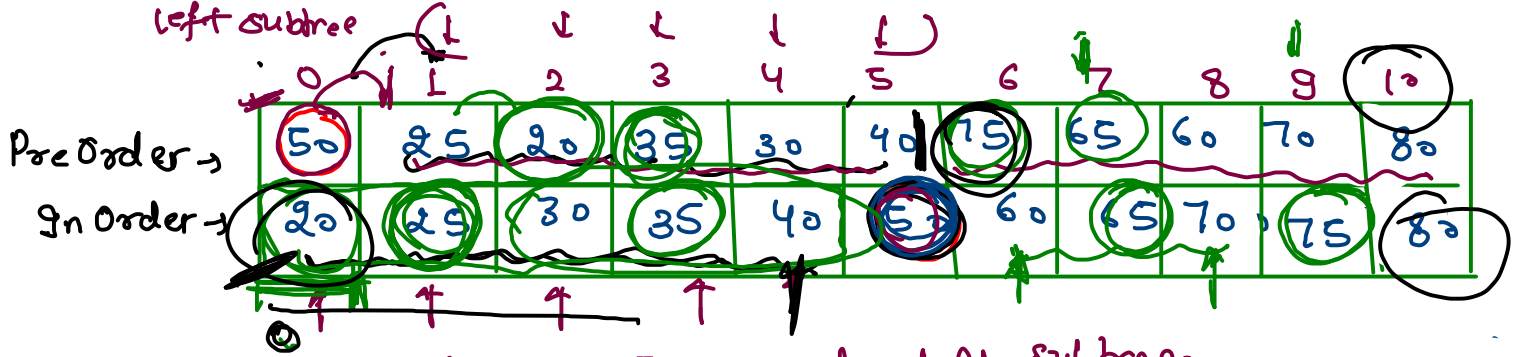
InOrder → Left Subtree | Root Node | Right Subtree



PreOrder, InOrder → single Possible Tree.

PostOrder, InOrder


PreOrder →
InOrder →
L.S.
R.S.
PreOrder → Root Left Right
InOrder → Left Root Right
PreOrder →
InOrder →
P →
I →
L.S.
R.S.
Ir →
In →
Pre →
In →
Pre →
In →
Pre →
In →
Pre - go
In - go



No. ele = 5 + element - left subtree.

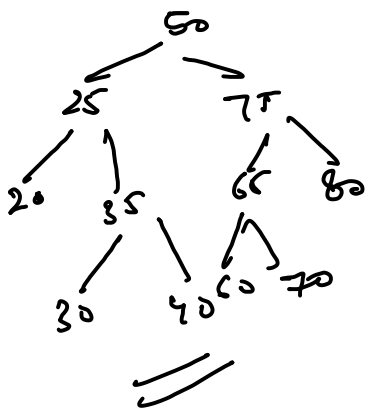
Pre-Starting Index

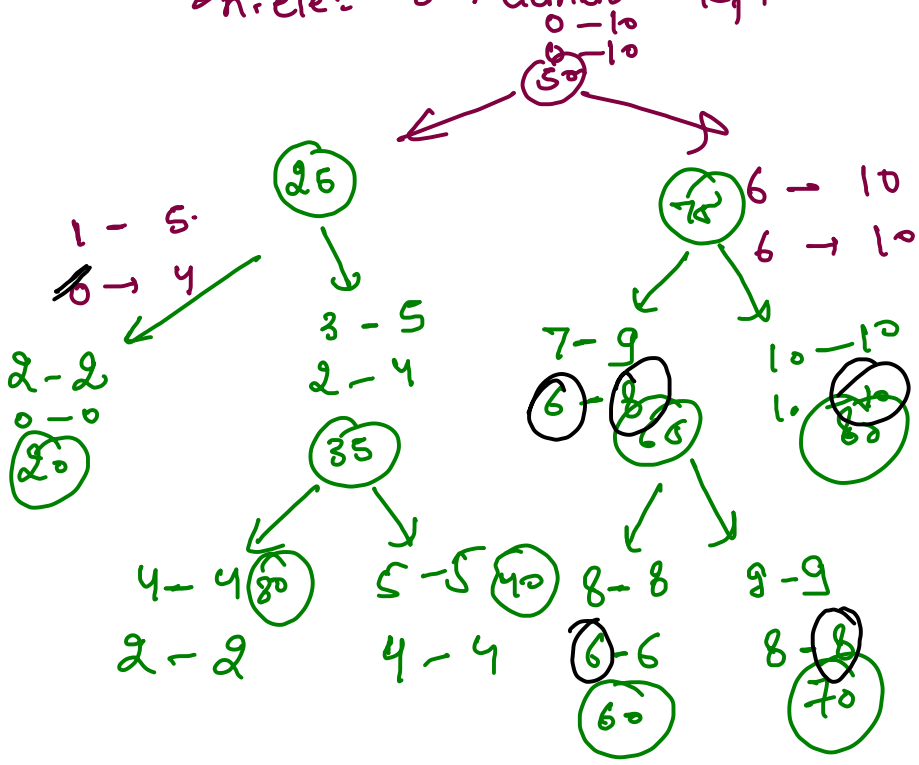
Pre Ending Index

In Starting Index

In Ending Index

Return → Node.





Steps:

1. Make Root Node
2. Find data Index in InOrder.
3. Split calls in left side. Right side.

```
// create root node
Treenode root = new TreeNode(pre[pst]);

root.left = construct(pre, in, pst + 1, ?, ist, ?);
root.right = construct(pre, in, ?, pend, ?, iend);
```

Root Left Right

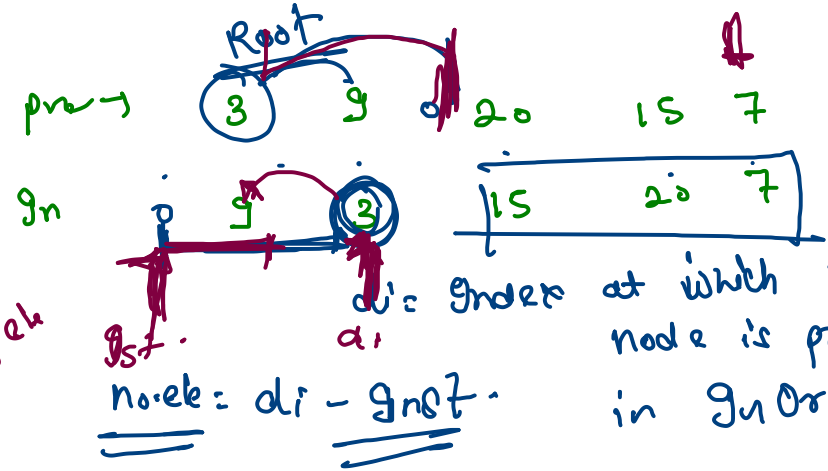
pend - ? → (pst + no. of ele.)

ind → ? di - 1

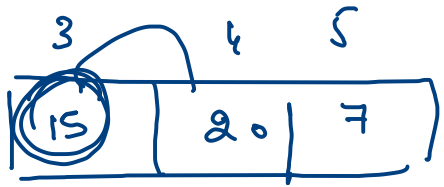
pst + no. of elt + 1

pst → ?

gn. start → di+1



count = di - gst



data: 20

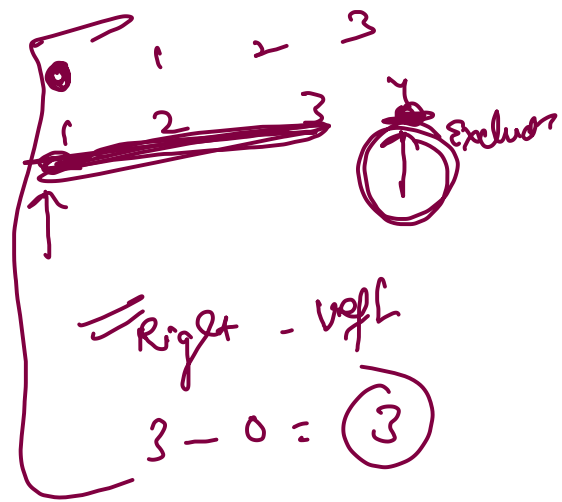
while (gnorder[di] != data)

di = gnst

di = 3 4

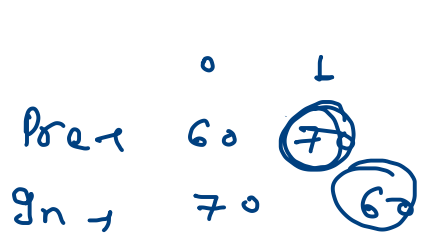
di++;

di = 4



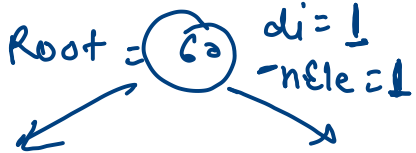
Right - left

3 - 0 = 3



pst = 0 gnst = 0

pend = 1 gnend = 1



pst > pend || gnst > gend

return NULL

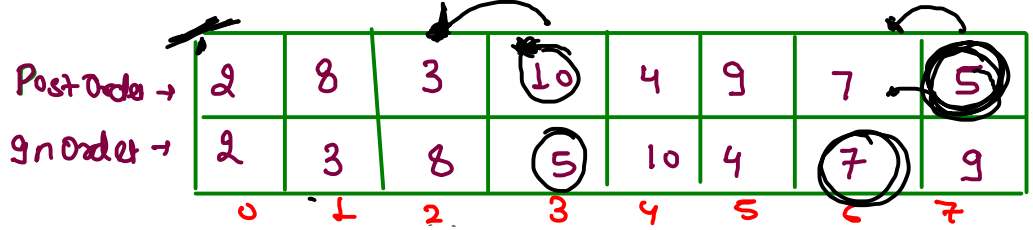
pst = 1 gnst = 0

pend = 1 gnEnd = 0

70

pst = 2 gnst = 2

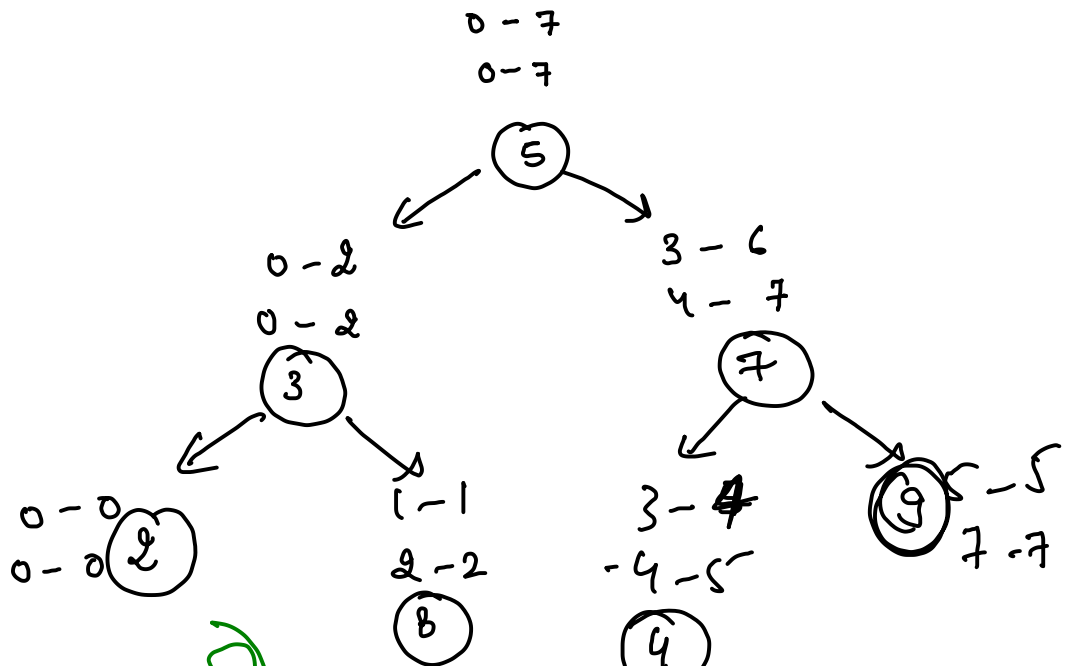
pend = 1 gend = 1

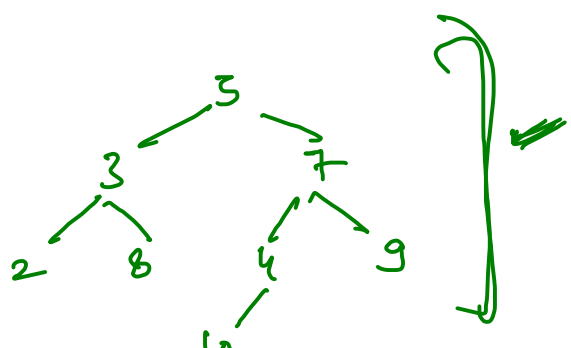


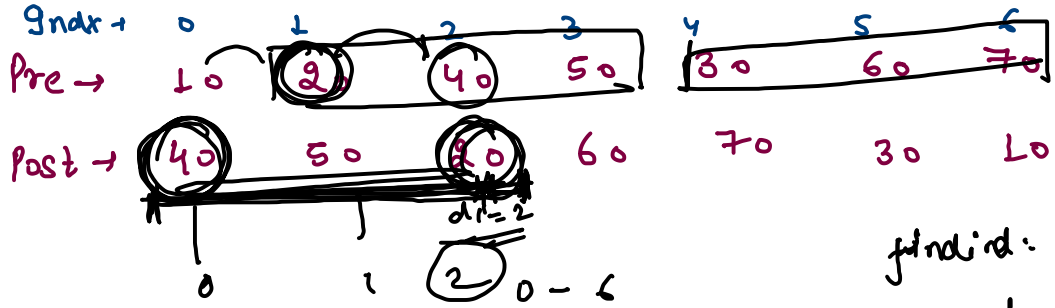
PostOrder → Left Right Root

Ind → Left Node Right

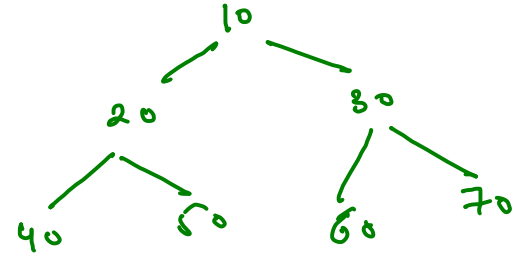
```
if(pst > pend || ist > iend)
        return null;

if(pst == pend || ist == iend) {
    return new TreeNode(post[pst]);
}

// create root node
TreeNode root = new TreeNode(post[pend]);

// di?
int di = ist;

while(in[di] != post[pend]) {
    di++;
}
// no. of ele?
int nEle = di - ist;


root.left = construct(post, in, pst, pst + nEle - 1, ist, di - 1);
root.right = construct(post, in, pst + nEle, pend - 1, di + 1, iend);
return root;
```







findInd: Pre[st+1] in Post Order.



, call →

left = pre → pst+1 ⟷ Pst+nEle

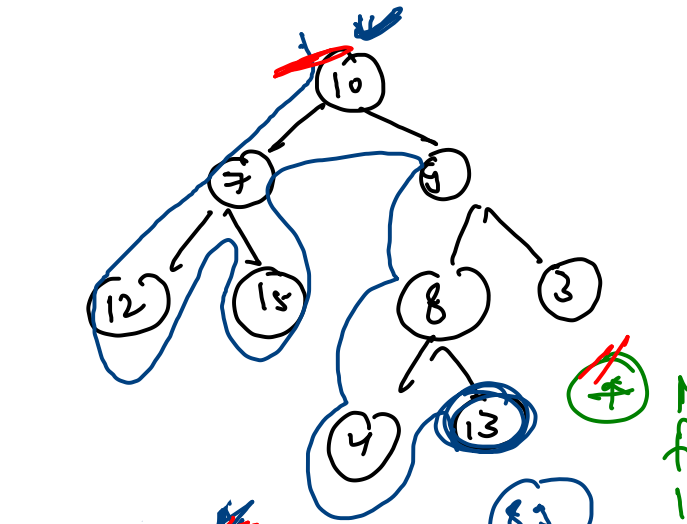
pst → post ⟷ di

nEle = di - pst + 1

Right Pre → pst+nEle ⟷ preEnd

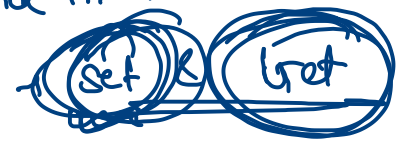
post → di+1 ⟷ postEnd-1

Binary Search Tree.

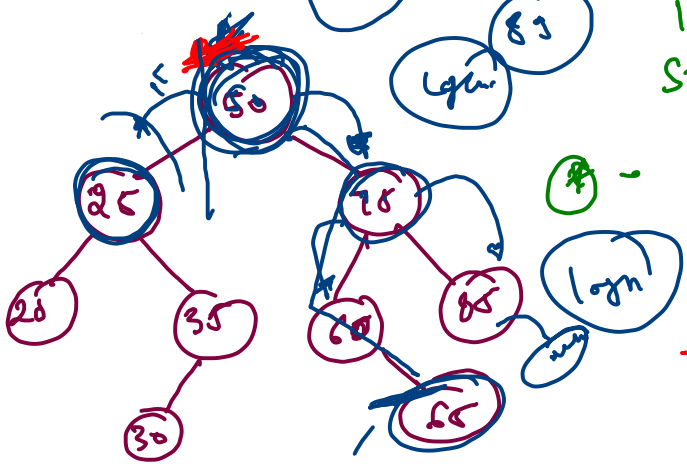


Data : 13

Find in B-Tree : O(h)



Max from left subtree



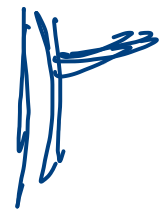
Array in BST

print it InOrder

Set of Rules for Binary Search Tree.

Root Node

Min from Right Subtree



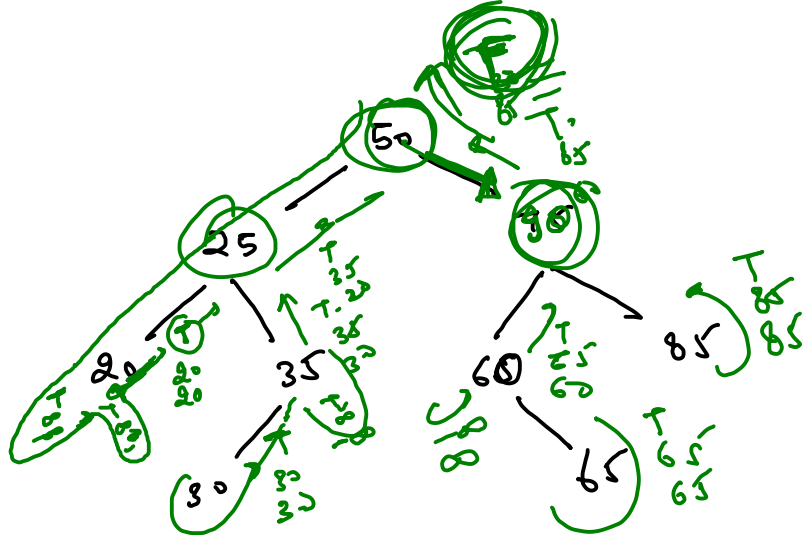
Feb.

- InOrder of BST → Sorted Order.

20 25 30 35 50 60 65 75 85

Sorted Order

```java
public static BSTPair isBST2(Node node) {
    if(node == null) {
        return new BSTPair();
    }

    BSTPair lp = isBST2(node.left);
    if(lp.isBST == false) return lp;

    BSTPair rp = isBST2(node.right);
    if(rp.isBST == false) return rp;

    BSTPair mp = new BSTPair();
    mp.isBST = lp.max < node.data && rp.min > node.data;
    mp.min = Math.min(node.data, Math.min(lp.min, rp.min));
    mp.max = Math.max(node.data, Math.max(lp.max, rp.max));

    return mp;
}
```



TreeSolver.