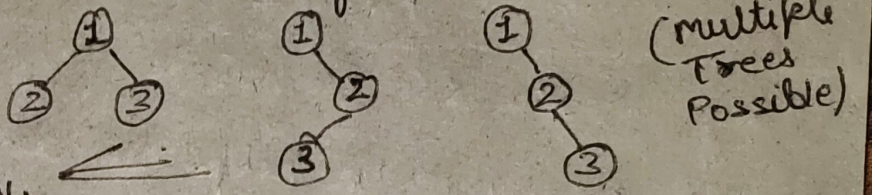
Concept: Constructing Tree From Inorder, Preorder & Postorder Traversals

Ques) Can we create a unique Binary Tree from Preorder ??

(NLR) Pre: | 1 | 2 | 3 |



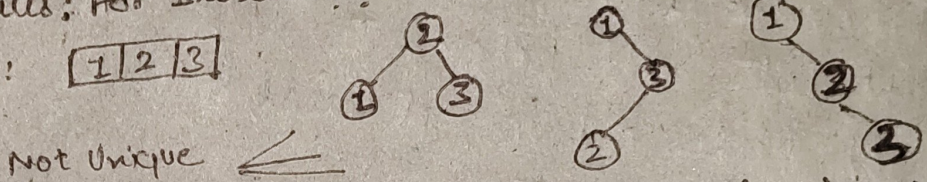
(multiple Trees Possible)

All these BTs have same Preorder Trav.

So, we can see here that: (We Can't Create Unique Tree From Only Preorder Traversal.)

Ques) Same Ques: for Inorder ??

(LNR) Inorder: | 1 | 2 | 3 |

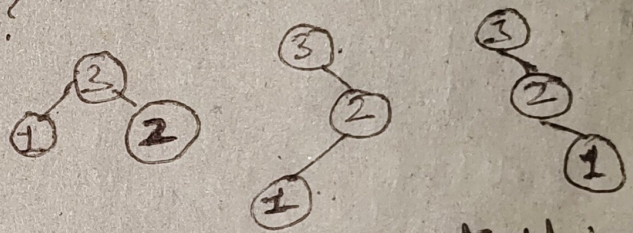


Not Unique

So, again we can say that: (We can't Create Unique Tree From Only Inorder Traversal.)

Ques) Same Ques: for Postorder ??

(LRN) Post order: | 1 | 2 | 3 |



Not Unique

So, again we can say tha (We Can't Create Unique Tree from Only Postorder Tr...

SO how can we construct Actual unique tree from given traversal ??

→ For that we need atleast two traversals for same tree then only we can create the actual unique tree.

Possible Pairs

✓	Inorder - Preorder	Only one Unique Tree Exist
✓	Inorder - Postorder	" " " "
×	Postorder - Preorder	Tree can't be created

Remember Inorder is OG!

Three possible pairs exist out of which only 2 can create a tree.

Inorder – Preorder ✓
Inorder – Postorder ✓
Preorder – Postorder ✗

Pair which contains Inorder trav. can make a unique original tree.

Why inorder is OCR? What is special??

Simple: B/z inorder helps us defining which node belong to left & which belongs to right. (very crucial thing)

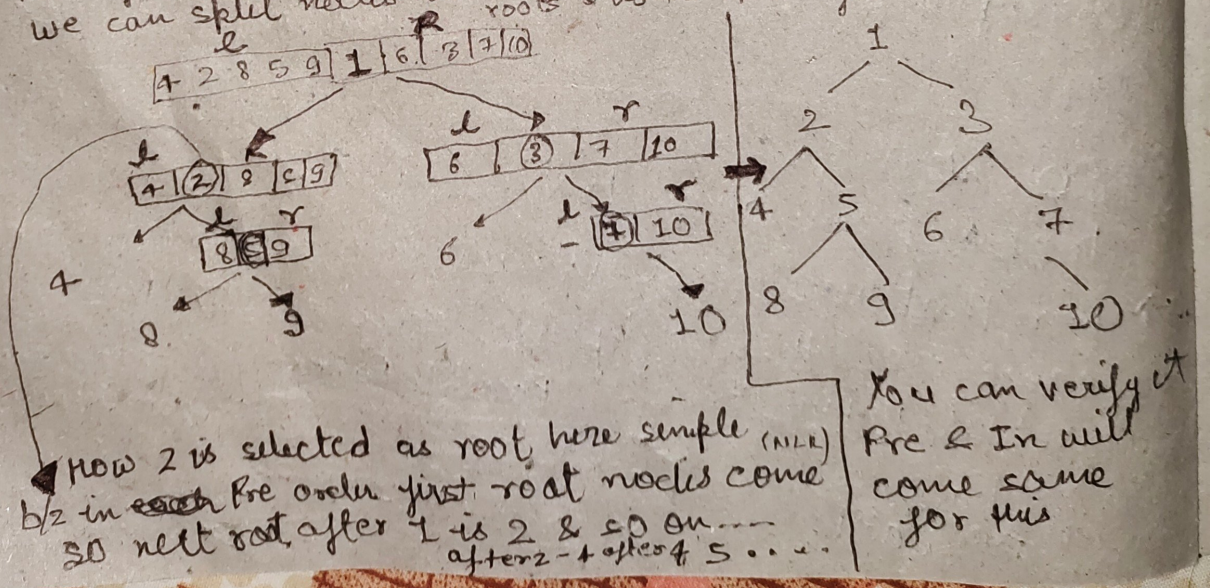
B/z it is: [LNR] or [Left Root Right]. If we have a Root node in inorder traversal. Then all nodes at left will belong to left subtree of root & all nodes at right will belong to right subtree of root.

Lets build tree from inorder and Post order traversal.

LNR In: 4 2 8 5 9 1 6 3 7 10
NLR Pre: 1 2 4 5 8 9 3 6 7 10
0 1 2 3 4 5 6 7 8 9

Rule No 1: First find root so 1 will be root b/z Preorder starts with root node.

Now find where this 1 exist in Inorder so that we can split nodes into left & right. (keep going for other roots & do the splitting)



Now 2 is selected as root, here simple (NLR) b/z in Pre order first root nodes come so next root after 1 is 2 & so on.... after 2 - 4 after 4 5 ....

You can verify it Pre & In will come same for this

But how can we put them into code:

We will use three pointers/variables:

1) idx → will represent or point to root node in Preorder traversal.

& using this root value we find where this root exist in inorder Traversal. (we call that idx "pos". (B/z it will help splitting nodes into Left & Right)

2) inStart : point to the starting of splitted array
(inorderStart) Initially 0;

3) inEnd : point to the end of splitted array
Initially n-1.

Basically these both help in slicing the inorder array:

Regarding Updation of variables:

left splitting → inStart to pos-1

right " → pos+1 to inEnd

when talking idx consider pre order array

what about idx ??

see when going root.left : idx is incrementing by 1
but when " root.right : idx will be updated differntly
b/z we have visited left nodes now we want idx point to right node so skip left nodes by updating idx

ex: [idx + (pos - inStart) + 1]

skipping left

see this in inorder array.

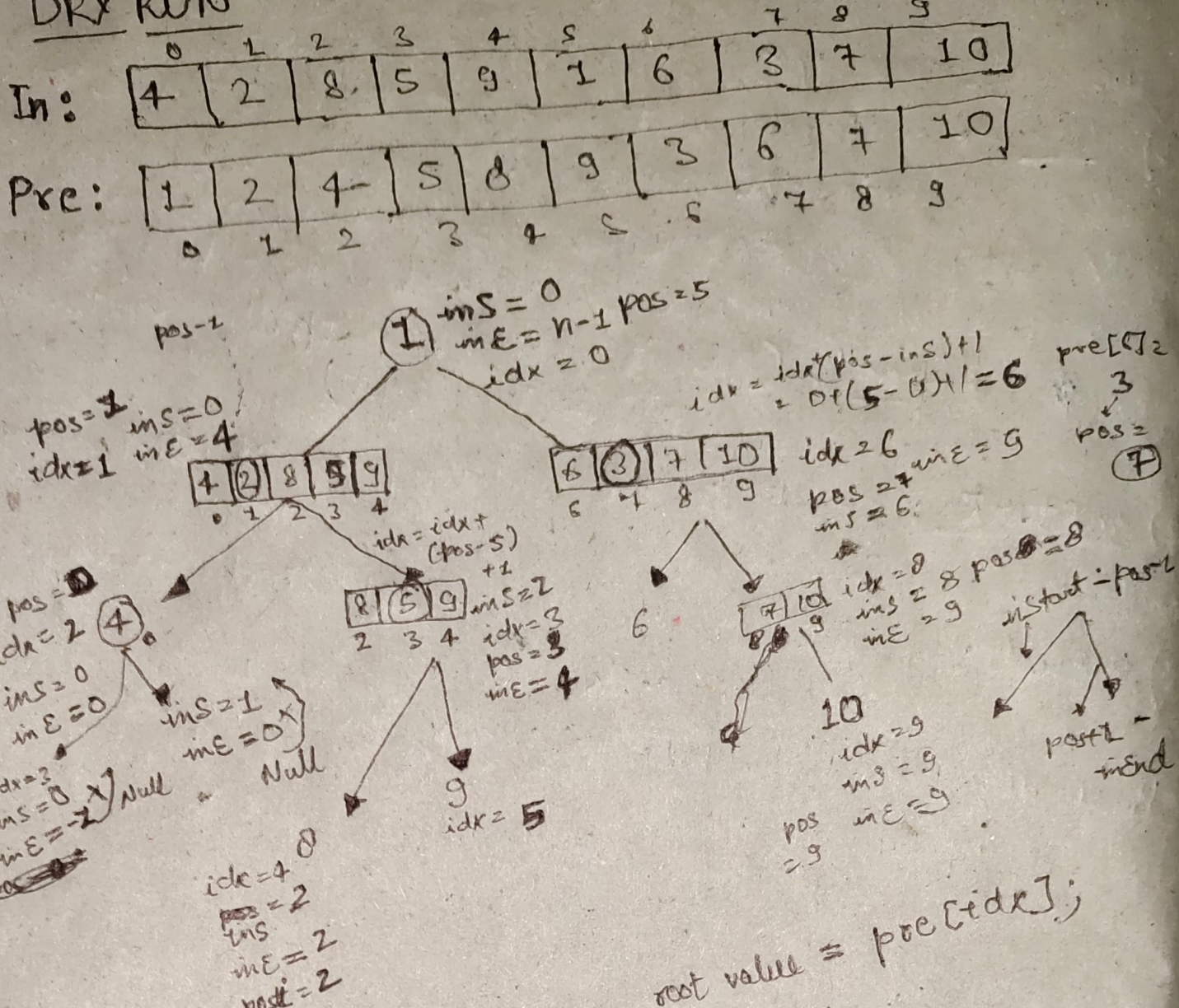
```
 */

/*
Steps:
1. Find the root node from preorder. (Idx will represent that)
2. Find the index of this root node (lets call it pos) in inorder so that
   we can split the nodes to left and right from root.
3. Left side nodes will be from inStart to pos-1 and right side nodes
   will be from pos+1 to inEnd.
   {inStart & inEnd are two pointers which tells that nodes from this
   to this belongs to left or right of the curr root.
   Initiall inStart = 0 & inEnd = n-1};
4. Note: for every left call left we will increase the idx pointer
         by 1 and for right call idx will be (skipping all left nodes)
         : idx+(pos-inStart)+1.
DRY RUN ON PEN AND PAPER EASY PEASY (DONE WHILE LEARNING ⭐)
*/
```

Code | Testcase | Test Result

Java | Auto

```java
class Solution {
    public static int findPos(int[] inorder, int value, int start, int end) {
        for(int i = start; i <= end; i++) {
            if(inorder[i] == value) return i;
        }
        return -1;
    }

    // "idx" here will point to root node of preorder & "pos" will point to root of inorder
    // time: O(n^2) space: O(n)
    public static TreeNode construct(int[] pre, int[] in, int idx, int inStart, int inEnd) {
        if(inStart > inEnd) return null;

        TreeNode root = new TreeNode(pre[idx]);
        int pos = findPos(in, pre[idx], inStart, inEnd); // finds the idx of root node from inorder

        // left
        root.left = construct(pre, in, idx+1, inStart, pos-1);
        // right
        root.right = construct(pre, in, idx+(pos-inStart)+1, pos+1, inEnd);
        return root;
    }

    public TreeNode buildTree(int[] preorder, int[] inorder) {
        return construct(preorder, inorder, 0, 0, inorder.length-1);
    }
```

DRY RUN



NOTE:

1. All node.val should be unique.
2. Only one unique tree is not possible to build from preorder and post order traversal because we don't know which node to put in last and which to right. So there can be multiple trees possible when building tree from pre order and post order.

Similarly We can Construct Tree from Inorder & Post Order Traversals.

Easy Peasy

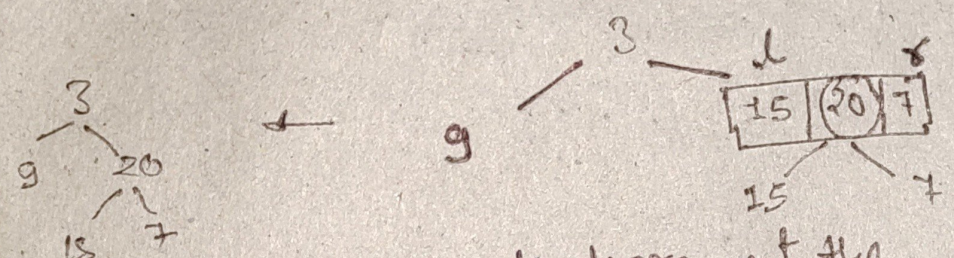
Few logical changes:

→ root node in PostOrder will be at end in postorder array b/c [L R N] (initially idx = (n-1))

Creating a Tree from given In & Post order Trav.

in : 9 3 15 20 7 — LNR

post : 9 15 7 20 3 — LRN



To move right decrement the idx by 1 simply

root.right

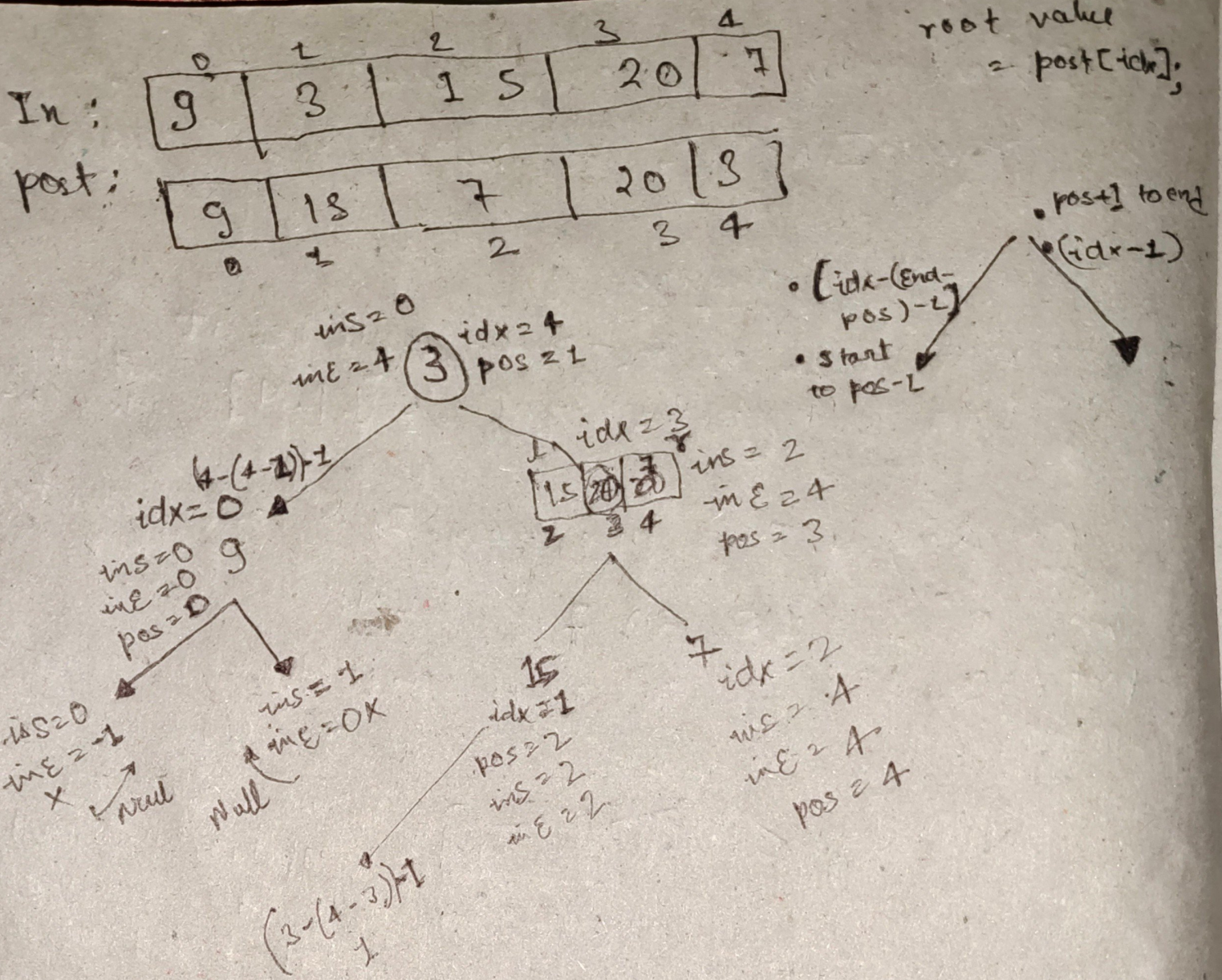
Let's see by DRY RUN

To move left we have to update the idx pointer so that it point to the ~~right~~ correct left node & for that we have to skip the right nodes first b/c in Postorder L R N Right will come first if moving from R to L

idx = idx - (inEnd - pos) - 1

```
/*
 If you have done Bulding tree from Preorder and Inorder this is exactly
 same with some logical change.

Steps:
1. Find the root node from postorder. (idx will represent that)
2. Find the index of this root node (lets call it pos) in inorder so that
   we can split the nodes to left and right from root.
3. Left side nodes will be from inStart to pos-1 and right side nodes
   will be from pos+1 to inEnd.
   {inStart & inEnd are two pointers which tells that nodes from this
   to this belongs to left or right of the curr root.
   Initially inStart = 0 & inEnd = n-1};
4. Note: for every left call idx will be (skipping all right nodes)
         : idx-(inEnd-pos)-1. as "LRN"
         ans for left calls we will just decrement the idx pointer
         by 1 b/z postorder me root ke just left side me right hota hai
DRY RUN ON PEN AND PAPER EASY PEASY (DONE WHILE LEARNING ⭐)
 */
```

```java
/
ass Solution {
  public static int findPos(int[] in, int value, int start, int end) {
      for(int i = start; i <= end; i++) {
          if(in[i] == value) return i;
      }
      return -1;
  }

  // "idx" is representing root from postorder and "pos" is representing root fro
  public static TreeNode construct(int[] in, int[] post, int idx, int inStart, in
      if(inStart > inEnd) return null;

      TreeNode root = new TreeNode(post[idx]);
      int pos = findPos(in, post[idx], inStart, inEnd);

      // left
      root.left = construct(in, post, idx-(inEnd-pos)-1, inStart, pos-1);
      // right
      root.right = construct(in, post, idx-1, pos+1, inEnd);
      return root;
  }
  public TreeNode buildTree(int[] inorder, int[] postorder) {
      int n = inorder.length;
      return construct(inorder, postorder, n-1, 0, n-1);
      // root in postorder will be at last idx thats why idx will get n-1 initial
  }
```


root value
= post[idx];
In:
post:
• post[] to end
• (idx-1)
• [idx-(end-pos)-2]
• start to pos-2
idx=4
pos=2
(4-(4-1))-1
idx=0
idx=3
pos=3
idx=2
pos=4
idx=1
pos=2
(3-(4-3))-1