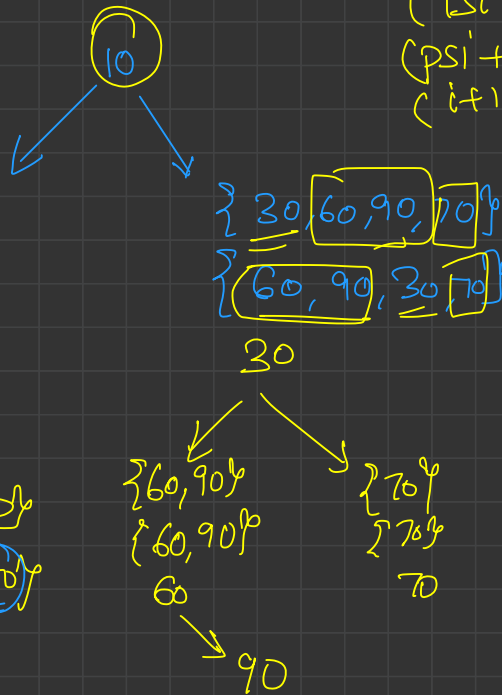
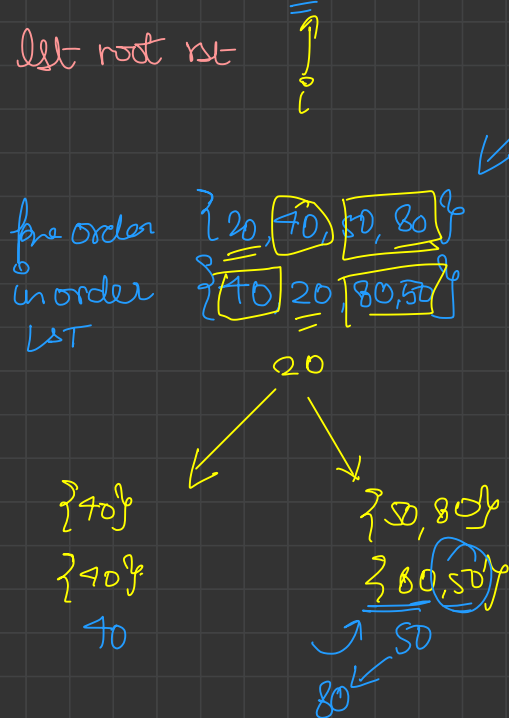




# Construct a Binary Tree from Preorder & Inorder

→ root Lst rst  
 preOrder: 10, 20, 40, 50, 80, 30, 60, 90, 70  
 inOrder: 40, 20, 80, 50, 10, 60, 90, 30, 70  
 → 1st root rst



unt = 4

→ (psi+1, psi+unt) LST  
 (isi, i-1) RST  
 (psi+unt+1, pei) RST  
 (i+1, pei) RST

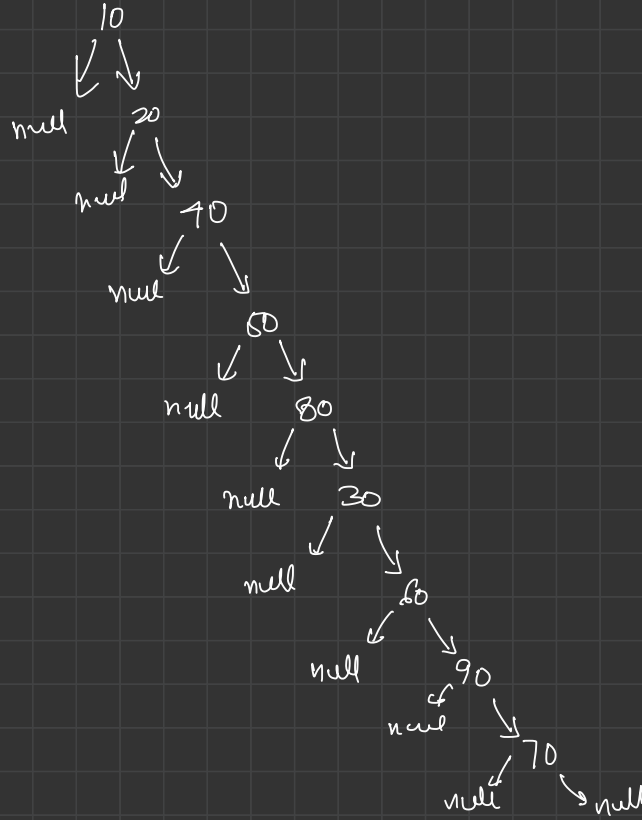
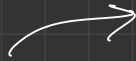
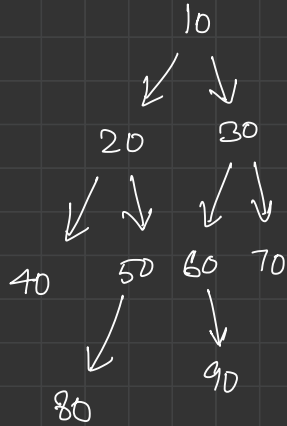
Build correct tree

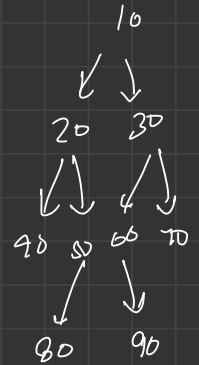
construct( int[] pre, int[] ino )

→ faith : It constructs tree from both the array given

# Flatten the Binary Tree

{ in place }

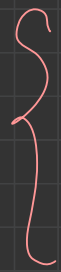
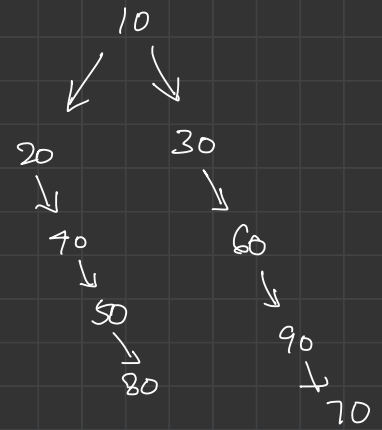
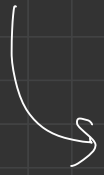




void flatten (Node root)

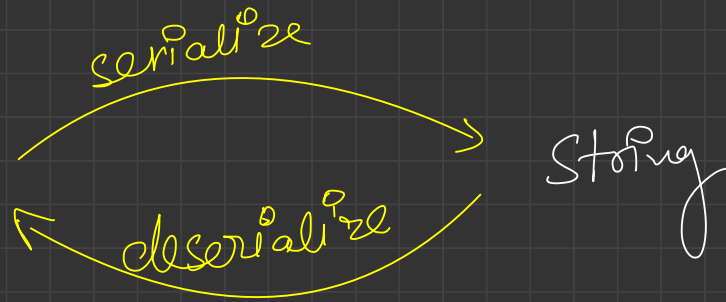
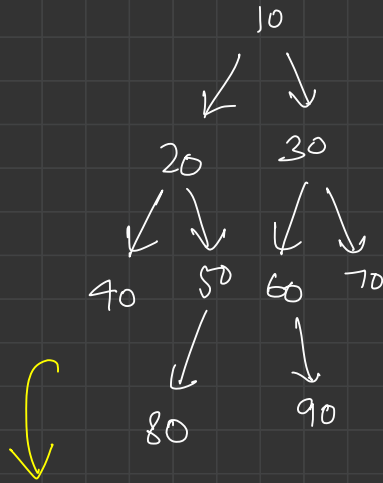


faith: It will flatten the Binary Tree



- ① flatten LST
- ② flatten RST
- ③ rearranging to a whole flattened tree

# Serialize and deserialize a Binary Tree

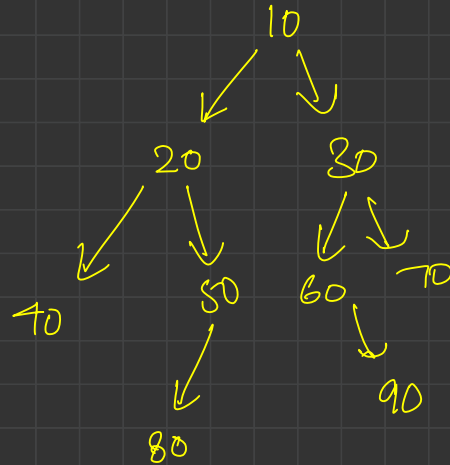


pre = <sup>ee</sup> 10, 20, 40, null, null, 50, 80, null, null, null, 30, 60, null, 90, null, null, 70, null, null <sup>99</sup>

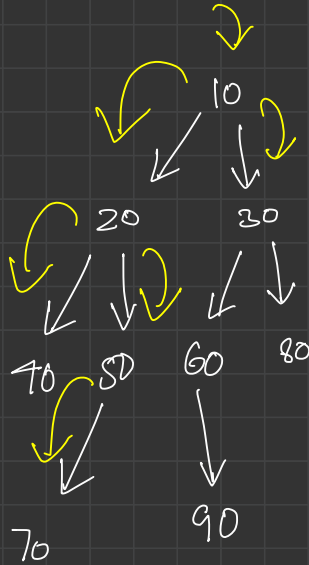
split(',')

pre = [10, 20, 40, null, null, 50, 80, null, null, null, 30, 60, null, 90, null, null, 70, null, null]

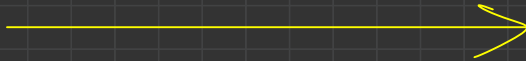
~~10~~ ~~20~~ ~~40~~ ~~50~~ ~~80~~ ~~30~~ ~~60~~ ~~90~~ ~~70~~



## Iterative pre order traversal

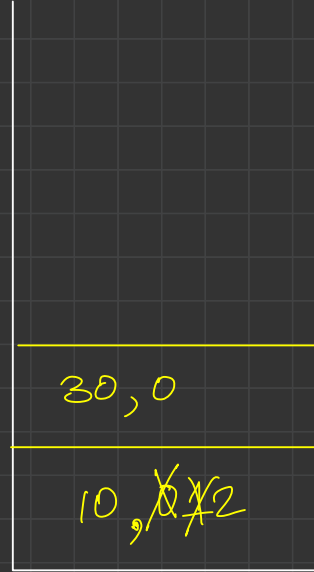


10, 20, 40, 50, 70, 30



call No.

0 → call left side }  
1 → call right side }  
2 → remove



Stack