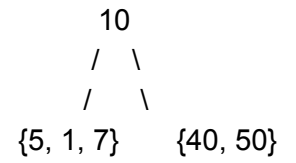
**Topic: Binary Search Tree**

**We are going to write the code to construct a binary search tree given postorder**

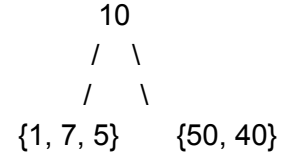
You are given a code skeleton that has a function that constructs a binary search tree from preorder traversal. **This will help you to understand how the creation of tree works recursively.**

Remember how constructing a Binary Search Tree from preorder works?
-The first element of preorder traversal is always root.
-We first construct the root.
-Then we find the index of the first element(from left to right) which is greater than the root. Let the index be 'i'. The values between root and 'i' will be part of the left subtree, and the values between 'i+1' and 'n-1' will be part of the right subtree(where n is the length of the array preorder)
-Divide the given pre[] at index "i" and recur for left and right sub-trees.
-For example, in {10, 5, 1, 7, 40, 50}, 10 is the first element, so we make it root. Now we look for the first element greater than 10, we find 40. So we know the structure of BST is as following.




You need to complete the function to construct a tree from postorder traversal.

Here is how constructing a tree from post-order would work.
-The last element of postorder traversal is always root.
- We first construct the root. Then we find the index of the last element ( when moving from left to right) which is smaller than the root. Let the index be 'i'. The values between 0 and 'i' are part of the left subtree, and the values between 'i+1' and 'n-2' are part of the right subtree(n is the length of the array postorder)
-Divide given post[] at index "i" and recur for left and right sub-trees.
-For example, in {1, 7, 5, 40, 50, 10}, 10 is the last element, so we make it root. Now we look for the last element smaller than 10, we find 5. So we know the structure of BST is as following.




**Once you have related pre-order code to the algorithm given, it would be easier to write code from post-order.**