Preorder to BST

Given an array arr[] of N nodes representing preorder traversal of some BST. You have to build the **BST** from the given preorder traversal.

In Pre-Order traversal, **the root node is visited before the left child and right child nodes**.

Example 1:

Input:

N = 5

arr[] = {40,30,35,80,100}

Output: 35 30 100 80 40

Explanation: PreOrder: 40 30 35 80 100

Therefore, the BST will be:

```
      40
    /   \
   30    80
    \     \
    35    100
```

Hence, the postOrder traversal will

be: 35 30 100 80 40

Example 2:

Input:

N = 8

arr[] = {40,30,32,35,80,90,100,120}

Output: 35 32 30 120 100 90 80 40

Your Task:
You need to complete the given function and **return the root** of the tree. The driver code will then use this root to print the post order traversal.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(N).

Constraints:
1 <= N <= 10^3
1 <= arr[i] <= 10^4

```python
#User function Template for python3
class Node:
    def __init__(self, data=0):
        self.data = data
        self.left = None
        self.right = None

def constructBST(preorder, size, min_val, max_val, index):
    # Base case: If the entire preorder traversal is processed
    if index[0] == size:
        return None

    root = None
    current_value = preorder[index[0]]

    # If the current value is within the range defined by
min_val and max_val,
    # it can be the root of the subtree for this part of the
preorder traversal.
    if min_val < current_value < max_val:
        # Create the root node with the current value
        root = Node(current_value)
        index[0] += 1

        # Recursively build the left and right subtrees
        root.left = constructBST(preorder, size, min_val,
current_value, index)
        root.right = constructBST(preorder, size,
current_value, max_val, index)

    return root

def postOrderTraversal(root, result):
    if root is None:
        return
    # Visit left subtree
    postOrderTraversal(root.left, result)
    # Visit right subtree
    postOrderTraversal(root.right, result)
    # Visit root node
    result.append(root.data)

def Bst(pre, size) -> Node:
    index = [0]  # mutable index to track the current index in
the preorder traversal
    root = constructBST(pre, size, float('-inf'),
float('inf'), index)
    return root

# Function to print postorder traversal
def printPostOrder(root):
    result = []
    postOrderTraversal(root, result)
    print(" ".join(map(str, result)))


#contributed by RavinderSinghPB
class Node:

    def __init__(self, data=0):
        self.data = data
        self.left = None
        self.right = None


def postOrd(root):
    if not root:
        return
    postOrd(root.left)
    postOrd(root.right)
    print(root.data, end=' ')


if __name__ == '__main__':
    t = int(input())

    for _tcs in range(t):
        size = int(input())
        pre = [int(x) for x in input().split()]

        root = Bst(pre, size)

        postOrd(root)
        print()

# } Driver Code Ends
```