Construct Binary Search Tree from the Postorder Traversal [CodeStudio](https://www.codingninjas.com/studio/problems/construct-bst-from-post-order_2674162?leftPanelTab=0)

Given an array of integers postorder, which represents the **postorder traversal** of a BST (i.e., **binary search tree**), construct the tree and return *its root*.

It is **guaranteed** that there is always possible to find a binary search tree with the given requirements for the given test cases.

Example:

The Input Postorder Traversal: [2,4,3,7,6,5]

Output:

5

/ \

3 6

/ \ \

2 4 7

**Approach 1: Build a Binary Search Tree (BST) from a given postorder traversal using a brute-force approach**

* **Function Purpose**: Build a binary search tree (BST) from a given postorder traversal using a brute-force approach.
* **Explanation**:
  + Create a new node for the root with the last element of the postorder traversal.
  + Iterate through the remaining elements in the postorder traversal to insert them into the BST.
  + Traverse the BST to find the appropriate position for each new node.
* **Time Complexity**: **O(N^2) in the worst case, where N is the number of elements in the postorder traversal.**
* **Space Complexity: O(N) to store the constructed BST.**

**Approach 2: Build a Binary Search Tree (BST) from a given postorder traversal using an optimized approach**

* **Function Purpose**: Build a binary search tree (BST) from a given postorder traversal using an optimized approach.
* **Explanation**:
  + Initialize the minimum and maximum values for elements in the BST.
  + Initialize an index for the postorder traversal.
  + Use a helper function to create the BST by recursively updating the minimum and maximum values and decrementing the index.
* **Time Complexity**: **O(N), where N is the number of elements in the postorder traversal.**
* **Space Complexity: O(N) to store the constructed BST.**

**Conclusion**:

* Both approaches construct a binary search tree from a given postorder traversal.
* **The optimized range-based approach is more efficient in terms of time complexity and is recommended for larger postorders**, as it has a time complexity of O(N), which is better than the brute-force approach's O(N^2).