Preorder traversal of BST

Problem statement Send feedback

You have been given an array/list 'PREORDER' representing the preorder traversal of a BST with 'N' nodes. All the elements in the given array have distinct values.
Your task is to construct a binary search tree that matches the given preorder traversal.
A binary search tree (BST) is a binary tree data structure that has the following properties:

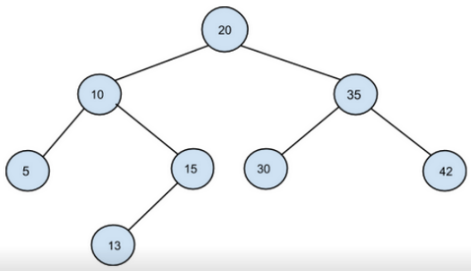
- The left subtree of a node contains only nodes with data less than the node's data.
- The right subtree of a node contains only nodes with data greater than the node's data.
- Both the left and right subtrees must also be binary search trees.

Note:

It is guaranteed that a BST can be always constructed from the given preorder traversal. Hence, the answer will always exist.

Example:

From PREORDER = [20, 10, 5, 15, 13, 35, 30, 42] , the following BST can be constructed:



→ Brute force :-

↳ Make a node.

↳ Use BST property to insert into the node.

Time Complexity : $O(N^2)$
Space Complexity : $O(N)$

→ Approach 2 :-

↳ We do have preorder traversal.

↳ Sort it to get inorder traversal

↳ Now, we can generate the tree from here.

Time Complexity :
Space Complexity :

→ Optimized Approach:-

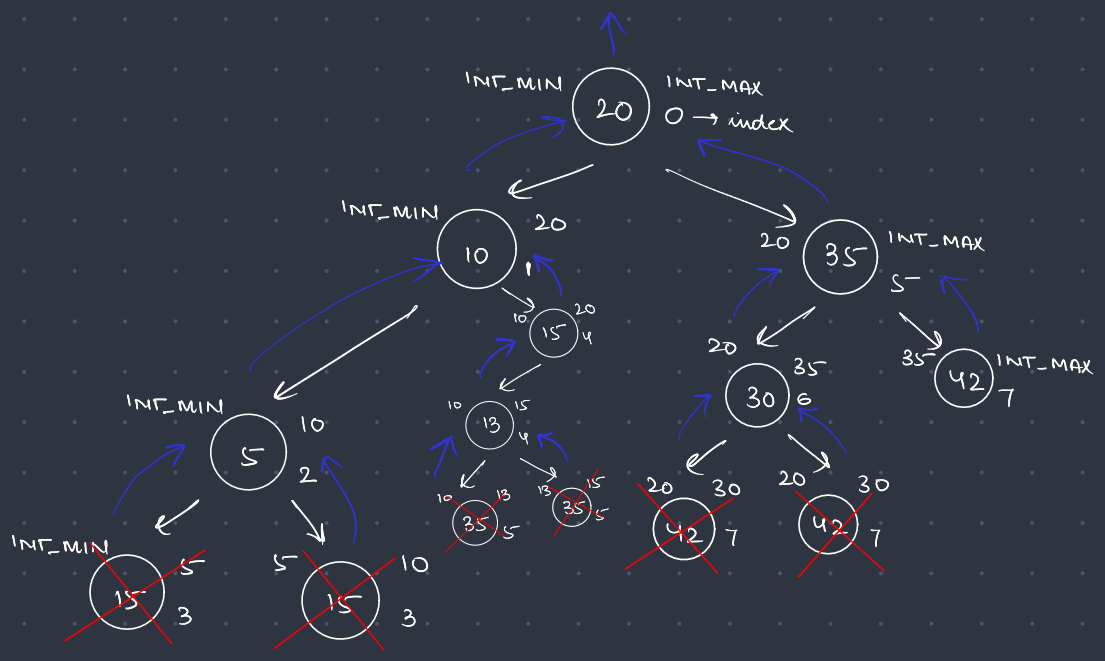
↳ We will use the Concept of isValidBST() here.

↳ for a particular node we will check if the root data < INT_MAX && root→data > MINI. If so, then we will make call for root→left & root→right.

↳ Else return NULL.

20	10	5	15	13	35	30	42

Recursion tree




```
BinaryTreeNode<int> *preToBST(vector<int> &preorder, int &index, int mini, int maxi) {
  if (index >= preorder.size())
    return NULL;
  if (preorder[index] < mini || preorder[index] > maxi)
    return NULL;

  BinaryTreeNode<int> *root = new BinaryTreeNode<int>(preorder[index++]);
  root->left = preToBST(preorder, index, mini, root->data);
  root->right = preToBST(preorder, index, root->data, maxi);

  return root;
}

BinaryTreeNode<int> *preorderToBST(vector<int> &preorder) {
  int mini = INT_MIN, maxi = INT_MAX;
  int index = 0;
  return preToBST(preorder, index, mini, maxi);
}
```

Time Complexity : O(N)

Space Complexity : O(N)