Construct BST from given preorder traversal | Set 1

Given the preorder traversal of a binary search tree, construct the BST.

Examples:
Input: *{10, 5, 1, 7, 40, 50}*
Output: *10*
*/ *
5 40
*/ \ *
1 7 50

Naive approach: To solve the problem follow the below idea:
The first element of preorder traversal is always the root. We first construct the root. Then we find the index of the first element 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'(inclusive) and 'n-1' will be part of the right subtree. 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 follows.:*
10
*/ *
{5, 1, 7} {40, 50}

We recursively follow the above steps for subarrays {5, 1, 7} and {40, 50}, and get the complete tree.

Below is the implementation of the above approach:

C++

```
// C++ program for the above approach

#include <bits/stdc++.h>

using namespace std;
```

```
/* A binary tree node has data, pointer to left child
and a pointer to right child */
class node {
public:
    int data;
    node* left;
    node* right;
};

// A utility function to create a node
node* newNode(int data)
{
    node* temp = new node();

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct Full from pre[].
// preIndex is used to keep track of index in pre[].
node* constructTreeUtil(int pre[], int* preIndex, int low,
                        int high, int size)
{
    // Base case
    if (*preIndex >= size || low > high)
```

```
    return NULL;

// The first node in preorder traversal is root. So take
// the node at preIndex from pre[] and make it root, and
// increment preIndex
node* root = newNode(pre[*preIndex]);
*preIndex = *preIndex + 1;

// If the current subarray has only one element, no need
// to recur
if (low == high)
    return root;

// Search for the first element greater than root
int i;
for (i = low; i <= high; ++i)
    if (pre[i] > root->data)
        break;

// Use the index of element found in preorder to divide
// preorder array in two parts. Left subtree and right
// subtree
root->left = constructTreeUtil(pre, preIndex, *preIndex,
                               i - 1, size);
root->right
    = constructTreeUtil(pre, preIndex, i, high, size);
```

```
    return root;
}

// The main function to construct BST from given preorder
// traversal. This function mainly uses constructTreeUtil()
node* constructTree(int pre[], int size)
{
    int preIndex = 0;
    return constructTreeUtil(pre, &preIndex, 0, size - 1,
                             size);
}

// A utility function to print inorder traversal of a Binary
// Tree
void printInorder(node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    cout << node->data << " ";
    printInorder(node->right);
}

// Driver code
int main()
{
    int pre[] = { 10, 5, 1, 7, 40, 50 };
```

```
    int size = sizeof(pre) / sizeof(pre[0]);


      // Function call
    node* root = constructTree(pre, size);


    printInorder(root);


    return 0;
}
```

Output

```
Inorder traversal of the constructed tree:
1 5 7 10 40 50
```

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

Approach: To solve the problem follow the below idea:
Using the recursion concept and iterating through the array of the given elements we can generate the BST

Follow the below steps to solve the problem:

- Create a new Node for every value in the array
- Create a BST using these new Nodes and insert them according to the rules of the BST
- Print the inorder of the BST

Below is the implementation of the above approach:

C++

```
// C++ Program for the same approach

#include <bits/stdc++.h>
```

```
using namespace std;

/*Construct a BST from given pre-order traversal
for example if the given traversal is {10, 5, 1, 7, 40, 50},
then the output should be the root of the following tree.
     10
   /   \
  5     40
 /  \      \
1    7      50 */

class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int data)
    {
        this->data = data;
        this->left = this->right = NULL;
    }
};

static Node* node;

// This will create the BST
Node* createNode(Node* node, int data)
```

```
{
    if (node == NULL)
        node = new Node(data);

    if (node->data > data)
        node->left = createNode(node->left, data);
    if (node->data < data)
        node->right = createNode(node->right, data);

    return node;
}

// A wrapper function of createNode
void create(int data) { node = createNode(node, data); }
// A function to print BST in inorder
void inorderRec(Node* root)
{
    if (root != NULL) {
        inorderRec(root->left);
        cout << root->data << " ";
        inorderRec(root->right);
    }
}

// Driver code
int main()
{
```

```
    vector<int> nodeData = { 10, 5, 1, 7, 40, 50 };


    for (int i = 0; i < nodeData.size(); i++) {
        create(nodeData[i]);
    }
    inorderRec(node);
}
```

Output

```
1
5
7
10
40
50
```

Time Complexity: O(N * log N)
Auxiliary Space: O(N)

Efficient Approach: To solve the problem follow the below idea:
The trick is to set a range {min .. max} for every node.

Follow the below steps to solve the problem:

- Initialize the range as {INT_MIN .. INT_MAX}
- The first node will definitely be in range, so create a root node.
- To construct the left subtree, set the range as {INT_MIN ...root->data}.
- If a value is in the range {INT_MIN .. root->data}, the values are part of the left subtree.
- To construct the right subtree, set the range as {root->data..max .. INT_MAX}.

Below is the implementation of the above approach:

C++

```
// C++ program for the above approach
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left child
and a pointer to right child */
class node {
public:
    int data;
    node* left;
    node* right;
};

// A utility function to create a node
node* newNode(int data)
{
    node* temp = new node();

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct
// BST from pre[]. preIndex is used
// to keep track of index in pre[].
```

```
node* constructTreeUtil(int pre[], int* preIndex, int key,
                        int min, int max, int size)
{
    // Base case
    if (*preIndex >= size)
        return NULL;

    node* root = NULL;

    // If current element of pre[] is in range, then
    // only it is part of current subtree
    if (key > min && key < max) {
        // Allocate memory for root of this
        // subtree and increment *preIndex
        root = newNode(key);
        *preIndex = *preIndex + 1;

        if (*preIndex < size) {
            // Construct the subtree under root
            // All nodes which are in range
            // {min .. key} will go in left
            // subtree, and first such node
            // will be root of left subtree.
            root->left = constructTreeUtil(pre, preIndex,
                                           pre[*preIndex],
                                           min, key, size);
        }
```

```
        if (*preIndex < size) {
            // All nodes which are in range
            // {key..max} will go in right
            // subtree, and first such node
            // will be root of right subtree.
            root->right = constructTreeUtil(pre, preIndex,
                                            pre[*preIndex],
                                            key, max, size);
        }
    }

    return root;
}

// The main function to construct BST
// from given preorder traversal.
// This function mainly uses constructTreeUtil()
node* constructTree(int pre[], int size)
{
    int preIndex = 0;
    return constructTreeUtil(pre, &preIndex, pre[0],
                             INT_MIN, INT_MAX, size);
}

// A utility function to print inorder
// traversal of a Binary Tree
void printInorder(node* node)
```

```
{
    if (node == NULL)
        return;
    printInorder(node->left);
    cout << node->data << " ";
    printInorder(node->right);
}

// Driver code
int main()
{
    int pre[] = { 10, 5, 1, 7, 40, 50 };
    int size = sizeof(pre) / sizeof(pre[0]);

    // Function call
    node* root = constructTree(pre, size);

    printInorder(root);

    return 0;
}
```

Output

```
Inorder traversal of the constructed tree:
1 5 7 10 40 50
```

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