1008. Construct Binary Search Tree from Preorder Traversal

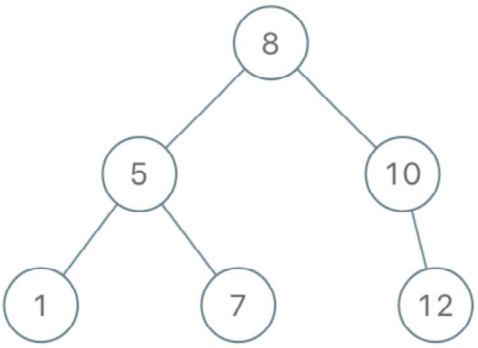
03 April 2022 08:12 AM

Given an array of integers preorder, which represents the **preorder 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.

A **binary search tree** is a binary tree where for every node, any descendant of `Node.left` has a value **strictly less than** `Node.val`, and any descendant of `Node.right` has a value **strictly greater than** `Node.val`.

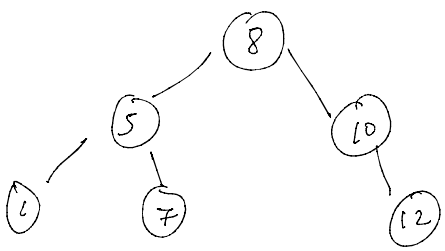
A **preorder traversal** of a binary tree displays the value of the node first, then traverses `Node.left`, then traverses `Node.right`.




```
Input: preorder = [8,5,1,7,10,12]
Output: [8,5,10,1,7,null,12]
```

Construct a BST from a pre-order traversal

pre-order → { 8 5 1 7 10 12 }



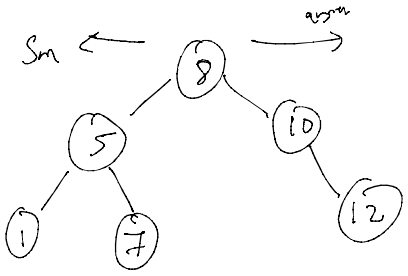
1) Brute force :

pre-order → { 8 5 1 7 10 12 }

↓

root left right

BST



T.C ⇒ O(N×N)

S.C ⇒ O(1)

Better Approach (using Inorder)

↳ Sorted

BST → Inorder → Sorted

preorder → {8, 5, 1, 7, 10, 12}

Inorder → {1, 5, 7, 8, 10, 12}

↙ Sorted only valid B.S.T

Whenever you have preorder and inorder you can create unique B.T.

↱ sorting

T.C ⇒ O(NlogN) + O(N)

↳ inorder

S.C ⇒ O(N)

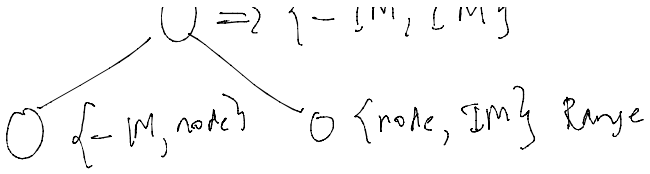
3) Efficient Approach

pre-order → { 8 5 1 7 10 12 }

-IM ⇒ Init MIN

IM ⇒ Init MAX

0 ⇒ {-IM, IM}

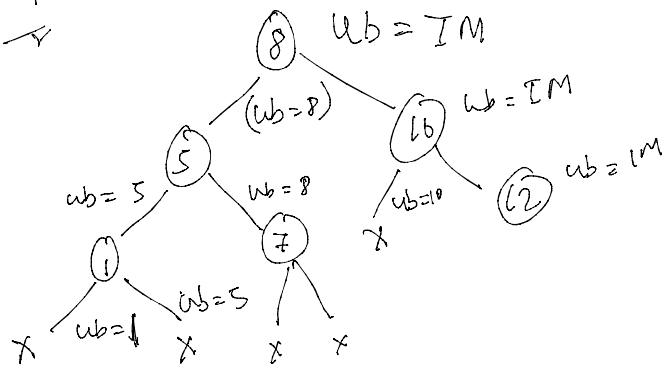


if you just carry upperbound it coill cOork, no need of lowerbound

pre-order → { 8 5 1 7 10 12 }

Root left Right

* Anything to left of 8 is less (ub=8)



(left, node.val) // for calling left

(right, bound) // for calling right

T.C ⇒ O(N)

S.C ⇒ O(1)

JAVA

```java
class Solution {
    public TreeNode bstFromPreorder(int[] preorder) {
        return bstFromPreorder(preorder, Integer.MAX_VALUE, new int[] {0});
    }

    public TreeNode bstFromPreorder(int[] preorder, int bound, int[] i){
        if(i[0] == preorder.length || preorder[i[0]] > bound) return null;

        TreeNode root = new TreeNode(preorder[i[0]++]);
        root.left = bstFromPreorder(preorder, root.val, i);
        root.right = bstFromPreorder(preorder, bound, i);
        return root;
    }
}
```

array of size 1 since it's pass by reference

C++

```cpp
class Solution {
public:
    TreeNode* bstFromPreorder(vector<int>& preorder) {
        int i = 0;
        return build(preorder, i, INT_MAX);
    }

    TreeNode* build(vector<int>& preorder, int& i, int bound) {
        if (i == preorder.size() || preorder[i] > bound) return NULL;
        TreeNode* root = new TreeNode(preorder[i++]);
        root->left = build(preorder, i, root->val);
        root->right = build(preorder, i, bound);
        return root;
    }
};
```