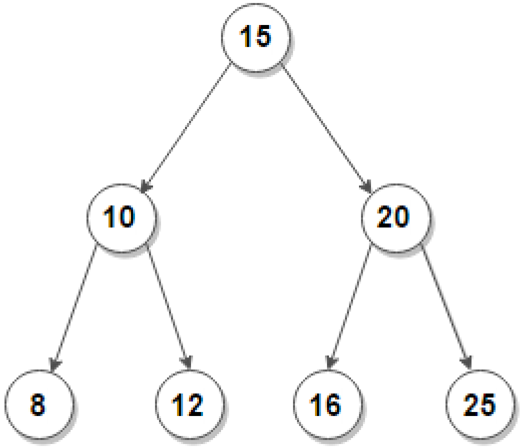
Build a Binary Search Tree from a preorder sequence

Given a distinct sequence of keys representing the preorder sequence of a binary search tree (BST), construct a BST from it.

For example, the following BST corresponds to the preorder traversal { 15, 10, 8, 12, 20, 16, 25 }.



Practice this problem

We can easily build a BST for a given preorder sequence by recursively repeating the following steps for all keys in it:

1. Construct the root node of BST, which would be the first key in the preorder sequence.
2. Find index `i` of the first key in the preorder sequence, which is greater than the root node.
3. Recur for the left subtree with keys in the preorder sequence that appears before the `i'th` index (excluding the first index).
4. Recur for the right subtree with keys in the preorder sequence that appears after the `i'th` index (including the `i'th` index).

Let's consider the preorder traversal `{15, 10, 8, 12, 20, 16, 25}` to make the context more clear.

1. The first item in the preorder sequence 15 becomes the root node.
2. Since 20 is the first key in the preorder sequence, which greater than the root node, the left subtree consists of keys `{10, 8, 12}` and the right subtree consists of keys `{20, 16, 25}`.
3. To construct the complete BST, recursively repeat the above steps for preorder sequence `{10, 8, 12}` and `{20, 16, 25}`.

The algorithm can be implemented as follows in C, Java, and Python:

C Java Python

```c
#include <stdio.h>
#include <stdlib.h>

// Data structure to store a binary tree node
struct Node
{
    int key;
    struct Node *left, *right;
};

// Function to create a new binary tree node having a given key
struct Node* newNode(int key)
{
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = node->right = NULL;

    return node;
}

// Recursive function to perform inorder traversal on a given binary tree
void inorder(struct Node* root)
{
    if (root == NULL) {
        return;
    }

    inorder(root->left);
    printf("%d ", root->key);
    inorder(root->right);
}

// Recursive function to build a BST from a preorder sequence.
struct Node* constructBST(int preorder[], int start, int end)
{
```

```
        return NULL;
    }

    // Construct the root node of the subtree formed by keys of the
    // preorder sequence in range `[start, end]`
    struct Node* node = newNode(preorder[start]);

    // search the index of the first element in the current range of preorder
    // sequence larger than the root node's value
    int i;
    for (i = start; i <= end; i++)
    {
        if (preorder[i] > node->key) {
            break;
        }
    }

    // recursively construct the left subtree
    node->left = constructBST(preorder, start + 1, i - 1);

    // recursively construct the right subtree
    node->right = constructBST(preorder, i, end);

    // return current node
    return node;
}

int main(void)
{
    /* Construct the following BST
              15
            /    \
           /      \
          10       20
         /  \     /  \
        /    \   /    \
       8     12 16    25
    */

```

```
    // construct the BST
    struct Node* root = constructBST(preorder, 0, n - 1);

    // print the BST
    printf("Inorder traversal of BST is ");

    // inorder on the BST always returns a sorted sequence
    inorder(root);

    return 0;
}
```

Download Run Code

Output:

```
Inorder traversal of BST is 8 10 12 15 16 20 25
```

The time complexity of the above solution is $O(n^2)$, where `n` is the size of the BST, and requires space proportional to the tree's height for the call stack. We can reduce the time complexity to $O(n)$ by following a different approach that doesn't involve searching for an index that separates the left and right subtree keys in a preorder sequence:

We know that each node has a key that is greater than all keys present in its left subtree, but less than the keys present in the right subtree of a BST. The idea to pass the information regarding the valid range of keys for the current root node and its children in the recursion itself.

We start by setting the range as `[-INFINITY, INFINITY]` for the root node. It means that the root node and any of its children can have keys ranging between `-INFINITY` and `INFINITY`. Like the previous approach, construct BST's root node from the first item in the preorder sequence. Suppose the root node has value `x`, recur for the right subtree with range `(x, INFINITY)` and recur for the left subtree with range `[-INFINITY, x)`. To construct the complete BST, recursively set the range for each recursive call and return if the next element in preorder traversal is out of the valid range.

Following is the C++, Java, and Python program that demonstrates it:

C++ Java Python

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

// Data structure to store a binary tree node
struct Node
{
    int data;
    Node* left = nullptr, *right = nullptr;

    Node() {}
    Node(int data): data(data) {}
};

// Function to print the inorder traversal on a given binary tree
void inorder(Node* root)
{
    if (root == nullptr) {
        return;
```

```
    cout << root->data << ' ';
    inorder(root->right);
}

// Recursive function to build a BST from a preorder sequence.
Node* buildTree(vector<int> const &preorder, int &pIndex,
                int min, int max)
{
    // Base case
    if (pIndex == preorder.size()) {
        return nullptr;
    }

    // Return if the next element of preorder traversal is not in the valid range
    int val = preorder[pIndex];
    if (val < min || val > max) {
        return nullptr;
    }

    // Construct the root node and increment `pIndex`
    Node* root = new Node(val);
    pIndex++;

    // Since all elements in the left subtree of a BST must be less
    // than the root node's value, set range as `[min, val-1]` and recur
    root->left = buildTree(preorder, pIndex, min, val - 1);

    // Since all elements in the right subtree of a BST must be greater
    // than the root node's value, set range as `[val+1…max]` and recur
    root->right = buildTree(preorder, pIndex, val + 1, max);

    return root;
}

// Build a BST from a preorder sequence
Node* buildTree(vector<int> const &preorder)
{
    // start from the root node (the first element in a preorder sequence)
    int pIndex = 0;
```

```
}

int main()
{
    /* Construct the following BST
              15
            /    \
           /      \
          10        20
         /  \     /  \
        /    \   /    \
       8     12 16     25
    */

    // preorder traversal of BST
    vector<int> preorder = { 15, 10, 8, 12, 20, 16, 25 };

    // construct the BST
    Node* root = buildTree(preorder);

    // print the BST
    cout << "Inorder traversal of BST is ";

    // inorder on the BST always returns a sorted sequence
    inorder(root);

    return 0;
}
```

Download Run Code

Output:

```
Inorder traversal of BST is 8 10 12 15 16 20 25
```

BST

Depth-first search, Hard, Recursive