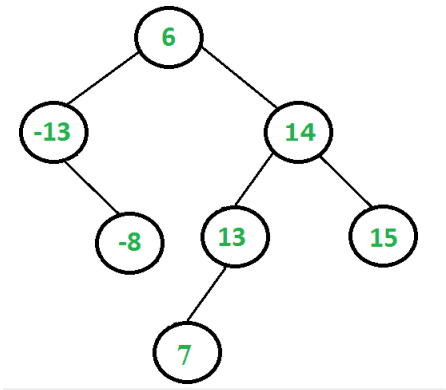
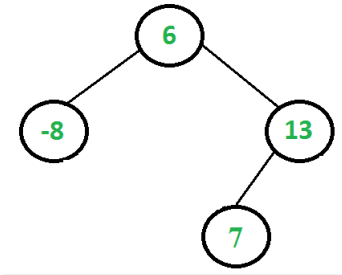
Remove BST keys outside the given range

Given a Binary Search Tree (BST) and a range [min, max], remove all keys which are outside the given range. The modified tree should also be BST. For example, consider the following BST and range [-10, 13].



The given tree should be changed to following. Note that all keys outside the range [-10, 13] are removed and modified tree is BST.



There are two possible cases for every node.
1) Node's key is outside the given range. This case has two sub-cases.
.....**a)** Node's key is smaller than the min value.
.....**b)** Node's key is greater that the max value.
2) Node's key is in range.

We don't need to do anything for case 2. In case 1, we need to remove the node and change root of sub-tree rooted with this node.
The idea is to fix the tree in Postorder fashion. When we visit a node, we make sure that its left and right sub-trees are already fixed. In case 1.a), we simply remove root and return right sub-tree as new root. In case 1.b), we remove root and return left sub-tree as new root.

Following is C++ implementation of the above approach.

```
// A C++ program to remove BST keys outside the given range
#include<stdio.h>
#include <iostream>

using namespace std;

// A BST node has key, and left and right pointers
struct node
{
    int key;
    struct node *left;
    struct node *right;
};

// Resmoves all nodes having value outside the given range and returns the root
// of modified tree
node* removeOutsideRange(node *root, int min, int max)
{
   // Base Case
   if (root == NULL)
      return NULL;
```

```
    return NULL;

   // First fix the left and right subtrees of root
   root->left =  removeOutsideRange(root->left, min, max);
   root->right =  removeOutsideRange(root->right, min, max);

   // Now fix the root.  There are 2 possible cases for toot
   // 1.a) Root's key is smaller than min value (root is not in range)
   if (root->key < min)
   {
       node *rChild = root->right;
       delete root;
       return rChild;
   }
   // 1.b) Root's key is greater than max value (root is not in range)
   if (root->key > max)
   {
       node *lChild = root->left;
       delete root;
       return lChild;
   }
   // 2. Root is in range
   return root;
}

// A utility function to create a new BST node with key as given num
node* newNode(int num)
{
    node* temp = new node;
    temp->key = num;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to insert a given key to BST
node* insert(node* root, int key)
{
    if (root == NULL)
       return newNode(key);
    if (root->key > key)
       root->left = insert(root->left, key);
    else
       root->right = insert(root->right, key);
    return root;
}

// Utility function to traverse the binary tree after conversion
void inorderTraversal(node* root)
{
    if (root)
    {
        inorderTraversal( root->left );
        cout << root->key << " ";
        inorderTraversal( root->right );
    }
}

// Driver program to test above functions
int main()
{
    node* root = NULL;
    root = insert(root, 6);
    root = insert(root, -13);
    root = insert(root, 14);
    root = insert(root, -8);
    root = insert(root, 15);
    root = insert(root, 13);
    root = insert(root, 7);

    cout << "Inorder traversal of the given tree is: ";
    inorderTraversal(root);

    root = removeOutsideRange(root, -10, 13);
```

```
    cout << "\nInorder traversal of the modified tree is: ";
    inorderTraversal(root);

    return 0;
}
```

Output:

```
Inorder traversal of the given tree is: -13 -8 6 7 13 14 15
Inorder traversal of the modified tree is: -8 6 7 13
```

Time Complexity: O(n) where n is the number of nodes in given BST.