Add all greater values to every node in a given BST

Given a **B**inary **S**earch **T**ree (BST), modify it so that all greater values in the given BST are added to every node. For example, consider the following BST.

```
              50
           /      \
         30        70
        /   \      /  \
      20    40    60   80

The above tree should be modified to following

              260
           /      \
         330        150
        /   \       /  \
      350   300    210   80
```

A **simple method** for solving this is to find sum of all greater values for every node. This method would take O(n^2) time.
We can do it **using a single traversal**. The idea is to use following BST property. If we do reverse Inorder traversal of BST, we get all nodes in decreasing order. We do reverse Inorder traversal and keep track of the sum of all nodes visited so far, we add this sum to every node.

```
// C program to add all greater values in every node of BST
#include<stdio.h>
#include<stdlib.h>

struct node
{
    int data;
    struct node *left, *right;
};

// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp =  (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

// Recursive function to add all greater values in every node
void modifyBSTUtil(struct node *root, int *sum)
{
    // Base Case
    if (root == NULL)  return;

    // Recur for right subtree
    modifyBSTUtil(root->right, sum);

    // Now *sum has sum of nodes in right subtree, add
    // root->data to sum and update root->data
    *sum = *sum + root->data;
    root->data = *sum;

    // Recur for left subtree
    modifyBSTUtil(root->left, sum);
}

// A wrapper over modifyBSTUtil()
void modifyBST(struct node *root)
{
    int sum = 0;
```

```
    int sum = 0;
    modifyBSTUtil(root, &sum);
}

// A utility function to do inorder traversal of BST
void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

/* A utility function to insert a new node with given data in BST */
struct node* insert(struct node* node, int data)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(data);

    /* Otherwise, recur down the tree */
    if (data <= node->data)
        node->left  = insert(node->left, data);
    else
        node->right = insert(node->right, data);

    /* return the (unchanged) node pointer */
    return node;
}

// Driver Program to test above functions
int main()
{
    /* Let us create following BST
              50
           /     \
          30      70
         /  \    /  \
       20   40  60   80 */
    struct node *root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    modifyBST(root);

    // print inoder tarversal of the modified BST
    inorder(root);

    return 0;
}
```

Output

```
350 330 300 260 210 150 80
```

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

As a side note, we can also use reverse Inorder traversal to find kth largest element in a BST.