Convert a BST to a Binary Tree such that sum of all greater keys is added to every key

Given a Binary Search Tree (BST), convert it to a Binary Tree such that every key of the original BST is changed to key plus sum of all greater keys in BST.

Examples:

```
Input: Root of following BST
              5
            /   \
           2     13

Output: The given BST is converted to following Binary Tree
              18
            /   \
          20     13
```

Source: Convert a BST

Solution: Do reverse Inoorder traversal. Keep track of the sum of nodes visited so far. Let this sum be *sum*. For every node currently being visited, first add the key of this node to *sum*, i.e. *sum = sum + node->key*. Then change the key of current node to *sum*, i.e., *node->key = sum*. When a BST is being traversed in reverse Inorder, for every key currently being visited, all keys that are already visited are all greater keys.

C

```
// Program to change a BST to Binary Tree such that key of a node becomes
// original key plus sum of all greater keys in BST
#include <stdio.h>
#include <stdlib.h>

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

/* Helper function that allocates a new node with the given key and
   NULL left and right  pointers.*/
struct node* newNode(int key)
{
    struct node* node = (struct node*)malloc(sizeof(struct node));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    return (node);
}

// A recursive function that traverses the given BST in reverse inorder and
// for every key, adds all greater keys to it
void addGreaterUtil(struct node *root, int *sum_ptr)
{
    // Base Case
    if (root == NULL)
        return;

    // Recur for right subtree first so that sum of all greater
    // nodes is stored at sum_ptr
    addGreaterUtil(root->right, sum_ptr);

    // Update the value at sum_ptr
    *sum_ptr = *sum_ptr + root->key;
    *sum_ptr = *sum_ptr + root->key;

    // Update key of this node
    root->key = *sum_ptr;

    // Recur for left subtree so that the updated sum is added
    // to smaller nodes
    addGreaterUtil(root->left, sum_ptr);
}

// A wrapper over addGreaterUtil().  It initializes sum and calls
// addGreaterUtil() to recursivel upodate and use value of sum
void addGreater(struct node *root)
{
    int sum = 0;
    addGreaterUtil(root, &sum);
}

// A utility function to print inorder traversal of Binary Tree
void printInorder(struct node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    printf("%d ", node->key);
    printInorder(node->right);
}

// Driver program to test above function
int main()
{
    /* Create following BST
              5
            /   \
           2     13  */
    node *root = newNode(5);
    root->left = newNode(2);
    root->right = newNode(13);

    printf(" Inorder traversal of the given tree\n");
    printInorder(root);

    addGreater(root);

    printf("\n Inorder traversal of the modified tree\n");
    printInorder(root);

    return 0;
}
```

Java

```
// Java program to convert BST to binary tree such that sum of
// all greater keys is added to every key

class Node {

    int data;
    Node left, right;

    Node(int d) {
        data = d;
        left = right = null;
    }
}

class Sum {

    int sum = 0;
}
```

```
class BinaryTree {

    static Node root;
    Sum summ = new Sum();

    // A recursive function that traverses the given BST in reverse inorder and
    // for every key, adds all greater keys to it
    void addGreaterUtil(Node node, Sum sum_ptr) {

        // Base Case
        if (node == null) {
            return;
        }

        // Recur for right subtree first so that sum of all greater
        // nodes is stored at sum_ptr
        addGreaterUtil(node.right, sum_ptr);

        // Update the value at sum_ptr
        sum_ptr.sum = sum_ptr.sum + node.data;

        // Update key of this node
        node.data = sum_ptr.sum;

        // Recur for left subtree so that the updated sum is added
        // to smaller nodes
        addGreaterUtil(node.left, sum_ptr);
    }

    // A wrapper over addGreaterUtil().  It initializes sum and calls
    // addGreaterUtil() to recursivel upodate and use value of sum
    Node addGreater(Node node) {
        addGreaterUtil(node, summ);
        return node;
    }

    // A utility function to print inorder traversal of Binary Tree
    void printInorder(Node node) {
        if (node == null) {
            return;
        }
        printInorder(node.left);
        System.out.print(node.data + " ");
        printInorder(node.right);
    }

    // Driver program to test the above functions
    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(5);
        tree.root.left = new Node(2);
        tree.root.right = new Node(13);

        System.out.println("Inorder traversal of given tree ");
        tree.printInorder(root);
        Node node = tree.addGreater(root);
        System.out.println("");
        System.out.println("Inorder traversal of modified tree ");
        tree.printInorder(node);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Output:

```
 Inorder traversal of the given tree
2 5 13
 Inorder traversal of the modified tree
20 18 13
```

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