Convert a given tree to its Sum Tree

Given a Binary Tree where each node has positive and negative values. Convert this to a tree where each node contains the sum of the left and right sub trees in the original tree. The values of leaf nodes are changed to 0.

For example, the following tree

```
                  10
               /      \
      -2        6
           /   \      /  \
  8     -4    7    5
```

should be changed to

```
                 20(4-2+12+6)
               /      \
    4(8-4)      12(7+5)
           /   \      /  \
  0      0    0    0
```

Solution:
Do a traversal of the given tree. In the traversal, store the old value of the current node, recursively call for left and right subtrees and change the value of current node as sum of the values returned by the recursive calls. Finally return the sum of new value and value (which is sum of values in the subtree rooted with this node).

C

```
#include<stdio.h>

/* A tree node structure */
struct node
{
  int data;
  struct node *left;
  struct node *right;
};

// Convert a given tree to a tree where every node contains sum of values of
// nodes in left and right subtrees in the original tree
int toSumTree(struct node *node)
{
    // Base case
    if(node == NULL)
      return 0;

    // Store the old value
    int old_val = node->data;

    // Recursively call for left and right subtrees and store the sum as
    // new value of this node
    node->data = toSumTree(node->left) + toSumTree(node->right);

    // Return the sum of values of nodes in left and right subtrees and
    // old_value of this node
    return node->data + old_val;
}

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

/* Utility function to create a new Binary Tree node */
struct node* newNode(int data)
{
  struct node *temp = new struct node;
  temp->data = data;
  temp->left = NULL;
  temp->right = NULL;

  return temp;
}

/* Driver function to test above functions */
int main()
{
  struct node *root = NULL;
  int x;

  /* Constructing tree given in the above figure */
  root = newNode(10);
  root->left = newNode(-2);
  root->right = newNode(6);
  root->left->left = newNode(8);
  root->left->right = newNode(-4);
  root->right->left = newNode(7);
  root->right->right = newNode(5);

  toSumTree(root);

  // Print inorder traversal of the converted tree to test result of toSumTree()
  printf("Inorder Traversal of the resultant tree is: \n");
  printInorder(root);

  getchar();
  return 0;
}
```

Java

```
// Java program to convert a tree into its sum tree

// A binary tree node
class Node
{
    int data;
    Node left, right;

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

class BinaryTree
{
    Node root;

    // Convert a given tree to a tree where every node contains sum of
    // values of nodes in left and right subtrees in the original tree
    int toSumTree(Node node)
    {
        // Base case
        if (node == null)
            return 0;
            return 0;

        // Store the old value
        int old_val = node.data;

        // Recursively call for left and right subtrees and store the sum
        // as new value of this node
        node.data = toSumTree(node.left) + toSumTree(node.right);

        // Return the sum of values of nodes in left and right subtrees
        // and old_value of this node
        return node.data + old_val;
    }

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

    /* Driver function to test above functions */
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();

        /* Constructing tree given in the above figure */
        tree.root = new Node(10);
        tree.root.left = new Node(-2);
        tree.root.right = new Node(6);
        tree.root.left.left = new Node(8);
        tree.root.left.right = new Node(-4);
        tree.root.right.left = new Node(7);
        tree.root.right.right = new Node(5);

        tree.toSumTree(tree.root);

        // Print inorder traversal of the converted tree to test result
        // of toSumTree()
        System.out.println("Inorder Traversal of the resultant tree is:");
        tree.printInorder(tree.root);
    }
}

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

Output:

```
Inorder Traversal of the resultant tree is:
0 4 0 20 0 12 0
```

Time Complexity: The solution involves a simple traversal of the given tree. So the time complexity is O(n) where n is the number of nodes in the given Binary Tree.