Convert an arbitrary Binary Tree to a tree that holds Children Sum Property

Question: Given an arbitrary binary tree, convert it to a binary tree that holds Children Sum Property. You can only increment data values in any node (You cannot change structure of tree and cannot decrement value of any node).

For example, the below tree doesn't hold the children sum property, convert it to a tree that holds the property.

```
             50
           /     \
         /         \
       7             2
     / \             /\
   /     \          /   \
  3        5      1      30
```

Algorithm:
Traverse given tree in post order to convert it, i.e., first change left and right children to hold the children sum property then change the parent node.

Let difference between node's data and children sum be diff.

```
      diff = node's children sum - node's data
```

If diff is 0 then nothing needs to be done.

If diff > 0 (node's data is smaller than node's children sum) increment the node's data by diff.

If diff < 0 (node's data is greater than the node's children sum) then increment one child's data. We can choose to increment either left or right child if they both are not NULL. Let us always first increment the left child. Incrementing a child changes the subtree's children sum property so we need to change left subtree also. So we recursively increment the left child. If left child is empty then we recursively call increment() for right child. Let us run the algorithm for the given example. First convert the left subtree (increment 7 to 8).

```
             50
           /     \
         /         \
       8             2
     / \             /\
   /     \          /   \
  3        5      1      30
```

Then convert the right subtree (increment 2 to 31)

```
          50
        /    \
      /        \
    8            31
   / \           / \
 /     \       /     \
3       5    1       30
```

Now convert the root, we have to increment left subtree for converting the root.

```
          50
        /    \
      /        \
    19           31
   / \           /  \
 /     \       /      \
14      5     1       30
```

Please note the last step – we have incremented 8 to 19, and to fix the subtree we have incremented 3 to 14.

Implementation:

C

```
/* Program to convert an aribitary binary tree to
   a tree that holds children sum property */

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

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

/* This function is used to increment left subtree */
void increment(struct node* node, int diff);

/* Helper function that allocates a new node
 with the given data and NULL left and right
 pointers. */
struct node* newNode(int data);

/* This function changes a tree to to hold children sum
   property */
void convertTree(struct node* node)
{
  int left_data = 0,  right_data = 0, diff;

  /* If tree is empty or it's a leaf node then
     return true */
  if (node == NULL ||
     (node->left == NULL && node->right == NULL))
    return;
  else
  {
    /* convert left and right subtrees  */
    convertTree(node->left);
    convertTree(node->right);

    /* If left child is not present then 0 is used
       as data of left child */
    if (node->left != NULL)
      left_data = node->left->data;
    /* If right child is not present then 0 is used
      as data of right child */
    if (node->right != NULL)
      right_data = node->right->data;

    /* get the diff of node's data and children sum */
    diff = left_data + right_data - node->data;

    /* If node's children sum is greater than the node's data */
    if (diff > 0)
       node->data = node->data + diff;

    /* THIS IS TRICKY --> If node's data is greater than children sum,
      then increment subtree by diff */
    if (diff < 0)
      increment(node, -diff);  // -diff is used to make diff positive
  }
}

/* This function is used to increment subtree by diff */
void increment(struct node* node, int diff)
{
  /* IF left child is not NULL then increment it */
  if(node->left != NULL)
  {
    node->left->data = node->left->data + diff;

    // Recursively call to fix the descendants of node->left
    increment(node->left, diff);
  }
  else if (node->right != NULL) // Else increment right child
  {
    node->right->data = node->right->data + diff;

    // Recursively call to fix the descendants of node->right
    increment(node->right, diff);
  }
}

/* Given a binary tree, printInorder() prints out its
   inorder traversal*/
void printInorder(struct node* node)
{
  if (node == NULL)
    return;

  /* first recur on left child */
  printInorder(node->left);

  /* then print the data of node */
  printf("%d ", node->data);

  /* now recur on right child */
  printInorder(node->right);
}

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

/* Driver program to test above functions */
int main()
{
  struct node *root = newNode(50);
  root->left        = newNode(7);
  root->right       = newNode(2);
  root->left->left  = newNode(3);
  root->left->right = newNode(5);
  root->right->left  = newNode(1);
  root->right->right = newNode(30);

  printf("\n Inorder traversal before conversion ");
  printInorder(root);

  convertTree(root);

  printf("\n Inorder traversal after conversion ");
  printInorder(root);

  getchar();
  return 0;
}
```

Java

```
    // Java program to convert an arbitrary binary tree to a tree that holds
// children sum property

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

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

class BinaryTree
{
    Node root;
    /* This function changes a tree to to hold children sum
```

```
       property */

    void convertTree(Node node)
    {
        int left_data = 0, right_data = 0, diff;

        /* If tree is empty or it's a leaf node then
         return true */
        if (node == null
                || (node.left == null && node.right == null))
            return;
        else
        {
            /* convert left and right subtrees  */
            convertTree(node.left);
            convertTree(node.right);

            /* If left child is not present then 0 is used
             as data of left child */
            if (node.left != null)
                left_data = node.left.data;

            /* If right child is not present then 0 is used
             as data of right child */
            if (node.right != null)
                right_data = node.right.data;

            /* get the diff of node's data and children sum */
            diff = left_data + right_data - node.data;

            /* If node's children sum is greater than the node's data */
            if (diff > 0)
                node.data = node.data + diff;

            /* THIS IS TRICKY --> If node's data is greater than children
               sum, then increment subtree by diff */
            if (diff < 0)

                // -diff is used to make diff positive
                increment(node, -diff);
        }
    }

    /* This function is used to increment subtree by diff */
    void increment(Node node, int diff)
    {
        /* IF left child is not NULL then increment it */
        if (node.left != null)
        {
            node.left.data = node.left.data + diff;

            // Recursively call to fix the descendants of node->left
            increment(node.left, diff);
        }
        else if (node.right != null) // Else increment right child
        {
            node.right.data = node.right.data + diff;

            // Recursively call to fix the descendants of node->right
            increment(node.right, diff);
        }
    }

    /* Given a binary tree, printInorder() prints out its
     inorder traversal*/
    void printInorder(Node node)
    {
        if (node == null)
            return;

        /* first recur on left child */
        printInorder(node.left);

        /* then print the data of node */
        System.out.print(node.data + " ");

        /* now recur on right child */
        printInorder(node.right);
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(50);
        tree.root.left = new Node(7);
        tree.root.right = new Node(2);
        tree.root.left.left = new Node(3);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(1);
        tree.root.right.right = new Node(30);

        System.out.println("Inorder traversal before conversion is :");
        tree.printInorder(tree.root);

        tree.convertTree(tree.root);
        System.out.println("");

        System.out.println("Inorder traversal after conversion is :");
        tree.printInorder(tree.root);

    }
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

```
 Output :
Inorder traversal before conversion is :
3 7 5 50 1 2 30
Inorder traversal after conversion is :
14 19 5 50 1 31 30
```

Time Complexity: O(n^2), Worst case complexity is for a skewed tree such that nodes are in decreasing order from root to leaf.