Check if a given Binary Tree is SumTree

Write a function that returns true if the given Binary Tree is SumTree else false. A SumTree is a Binary Tree where the value of a node is equal to sum of the nodes present in its left subtree and right subtree. An empty tree is SumTree and sum of an empty tree can be considered as 0. A leaf node is also considered as SumTree.

Following is an example of SumTree.

```
          26
        /   \
      10     3
    /    \     \
  4      6      3
```

Method 1 (Simple)

Get the sum of nodes in left subtree and right subtree. Check if the sum calculated is equal to root's data. Also, recursively check if the left and right subtrees are SumTrees.

C

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

/* A binary tree node has data, left child and right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* A utility function to get the sum of values in tree with root
  as root */
int sum(struct node *root)
{
   if(root == NULL)
     return 0;
   return sum(root->left) + root->data + sum(root->right);
}

/* returns 1 if sum property holds for the given
    node and both of its children */
int isSumTree(struct node* node)
{
    int ls, rs;

    /* If node is NULL or it's a leaf node then
       return true */
    if(node == NULL ||
            (node->left == NULL && node->right == NULL))
        return 1;

   /* Get sum of nodes in left and right subtrees */
   ls = sum(node->left);
   rs = sum(node->right);

   /* if the node and both of its children satisfy the
       property return 1 else 0*/
    if((node->data == ls + rs)&&
            isSumTree(node->left) &&
            isSumTree(node->right))
        return 1;
```

```
   return 0;
}

/*
 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 function */
int main()
{
    struct node *root  = newNode(26);
    root->left         = newNode(10);
    root->right        = newNode(3);
    root->left->left   = newNode(4);
    root->left->right  = newNode(6);
    root->right->right = newNode(3);
    if(isSumTree(root))
        printf("The given tree is a SumTree ");
    else
        printf("The given tree is not a SumTree ");

    getchar();
    return 0;
}
```

Java

```
// Java program to check if Binary tree is sum tree or not

/* A binary tree node has data, left child and right child */
class Node
{
    int data;
    Node left, right, nextRight;

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

class BinaryTree
{
    Node root;

    /* A utility function to get the sum of values in tree with root
     as root */
    int sum(Node node)
    {
        if (node == null)
            return 0;
        return sum(node.left) + node.data + sum(node.right);
    }

    /* returns 1 if sum property holds for the given
       node and both of its children */
    int isSumTree(Node node)
    {
        int ls, rs;

        /* If node is NULL or it's a leaf node then
           return true */
        if ((node == null) || (node.left == null && node.right == null))
            return 1;

        /* Get sum of nodes in left and right subtrees */
        ls = sum(node.left);
        rs = sum(node.right);

        /* if the node and both of its children satisfy the
           property return 1 else 0*/
        if ((node.data == ls + rs) && (isSumTree(node.left) != 0)
                && (isSumTree(node.right)) != 0)
            return 1;

        return 0;
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(26);
        tree.root.left = new Node(10);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(6);
        tree.root.right.right = new Node(3);

        if (tree.isSumTree(tree.root) != 0)
            System.out.println("The given tree is a sum tree");
        else
            System.out.println("The given tree is not a sum tree");
    }
}

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

Output:

```
The given tree is a SumTree
```

Time Complexity: O(n^2) in worst case. Worst case occurs for a skewed tree.

Method 2 (Tricky)
The Method 1 uses sum() to get the sum of nodes in left and right subtrees. The method 2 uses following rules to get the sum directly.
1) If the node is a leaf node then sum of subtree rooted with this node is equal to value of this node.
2) If the node is not a leaf node then sum of subtree rooted with this node is twice the value of this node (Assuming that the tree rooted with this node is SumTree).

C

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

/* A binary tree node has data, left child and right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* Utillity function to check if the given node is leaf or not */
int isLeaf(struct node *node)
{
    if(node == NULL)
        return 0;
    if(node->left == NULL && node->right == NULL)
        return 1;
    return 0;
}

/* returns 1 if SumTree property holds for the given
    tree */
int isSumTree(struct node* node)
{
    int ls; // for sum of nodes in left subtree
    int rs; // for sum of nodes in right subtree

    /* If node is NULL or it's a leaf node then
       return true */
    if(node == NULL || isLeaf(node))
        return 1;

    if( isSumTree(node->left) && isSumTree(node->right))
    {
        // Get the sum of nodes in left subtree
        if(node->left == NULL)
            ls = 0;
        else if(isLeaf(node->left))
            ls = node->left->data;
        else
            ls = 2*(node->left->data);

        // Get the sum of nodes in right subtree
        if(node->right == NULL)
            rs = 0;
        else if(isLeaf(node->right))
            rs = node->right->data;
        else
            rs = 2*(node->right->data);

        /* If root's data is equal to sum of nodes in left
           and right subtrees then return 1 else return 0*/
        return(node->data == ls + rs);
    }
```

```
    }

    return 0;
}

/* 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 function */
int main()
{
    struct node *root  = newNode(26);
    root->left         = newNode(10);
    root->right        = newNode(3);
    root->left->left   = newNode(4);
    root->left->right  = newNode(6);
    root->right->right = newNode(3);
    if(isSumTree(root))
        printf("The given tree is a SumTree ");
    else
        printf("The given tree is not a SumTree ");

    getchar();
    return 0;
}
```

Java

```
// Java program to check if Binary tree is sum tree or not


/* A binary tree node has data, left child and right child */
class Node
{
    int data;
    Node left, right, nextRight;

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

class BinaryTree
{
    Node root;

    /* Utility function to check if the given node is leaf or not */
    int isLeaf(Node node)
    {
        if (node == null)
            return 0;
        if (node.left == null && node.right == null)
            return 1;
        return 0;
    }

    /* returns 1 if SumTree property holds for the given
       tree */
```

```
    int isSumTree(Node node)
    {
        int ls; // for sum of nodes in left subtree
        int rs; // for sum of nodes in right subtree

        /* If node is NULL or it's a leaf node then
         return true */
        if (node == null || isLeaf(node) == 1)
            return 1;

        if (isSumTree(node.left) != 0 && isSumTree(node.right) != 0)
        {
            // Get the sum of nodes in left subtree
            if (node.left == null)
                ls = 0;
            else if (isLeaf(node.left) != 0)
                ls = node.left.data;
            else
                ls = 2 * (node.left.data);

            // Get the sum of nodes in right subtree
            if (node.right == null)
                rs = 0;
            else if (isLeaf(node.right) != 0)
                rs = node.right.data;
            else
                rs = 2 * (node.right.data);

            /* If root's data is equal to sum of nodes in left
               and right subtrees then return 1 else return 0*/
            if ((node.data == rs + ls))
                return 1;
            else
                return 0;
        }

        return 0;
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(26);
        tree.root.left = new Node(10);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(6);
        tree.root.right.right = new Node(3);

        if (tree.isSumTree(tree.root) != 0)
            System.out.println("The given tree is a sum tree");
        else
            System.out.println("The given tree is not a sum tree");
    }
}

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

Output:

```
The given tree is a sum tree
```

Time Complexity: O(n).