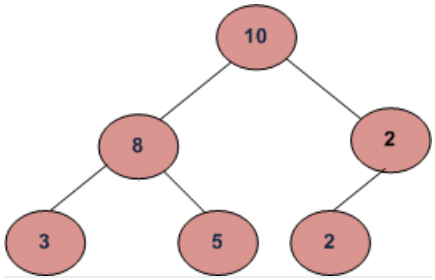
Check for Children Sum Property in a Binary Tree

Given a binary tree, write a function that returns true if the tree satisfies below property.

For every node, data value must be equal to sum of data values in left and right children. Consider data value as 0 for NULL children. Below tree is an example



Algorithm:

Traverse the given binary tree. For each node check (recursively) if the node and both its children satisfy the Children Sum Property, if so then return true else return false.

Implementation:

C

```
/* Program to check children sum property */
#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;
};

/* returns 1 if children sum property holds for the given
    node and both of its children*/
int isSumProperty(struct node* node)
{
  /* left_data is left child data and right_data is for right
     child data*/
  int left_data = 0,  right_data = 0;

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

    /* if the node and both of its children satisfy the
```

```
       property return 1 else 0*/
    if((node->data == left_data + right_data)&&
        isSumProperty(node->left) &&
        isSumProperty(node->right))
      return 1;
    else
      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(10);
  root->left         = newNode(8);
  root->right        = newNode(2);
  root->left->left   = newNode(3);
  root->left->right  = newNode(5);
  root->right->right = newNode(2);
  if(isSumProperty(root))
    printf("The given tree satisfies the children sum property ");
  else
    printf("The given tree does not satisfy the children sum property ");

  getchar();
  return 0;
}
```

Java

```
// Java program to check children sum property

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

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

class BinaryTree
{
    Node root;

    /* returns 1 if children sum property holds for the given
    node and both of its children*/
    int isSumProperty(Node node)
    {
```

```
        /* left_data is left child data and right_data is for right
           child data*/
        int left_data = 0, right_data = 0;

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

            /* 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;

            /* if the node and both of its children satisfy the
               property return 1 else 0*/
            if ((node.data == left_data + right_data)
                    && (isSumProperty(node.left)!=0)
                    && isSumProperty(node.right)!=0)
                return 1;
            else
                return 0;
        }
    }

    /* driver program to test the above functions */
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(8);
        tree.root.right = new Node(2);
        tree.root.left.left = new Node(3);
        tree.root.left.right = new Node(5);
        tree.root.right.right = new Node(2);
        if (tree.isSumProperty(tree.root) != 0)
            System.out.println("The given tree satisfies children"
                    + " sum property");
        else
            System.out.println("The given tree does not satisfy children"
                    + " sum property");
    }
}
```

Output:

```
The given tree satisfies the children sum property
```

Time Complexity: O(n), we are doing a complete traversal of the tree.

As an exercise, extend the above question for an n-ary tree.