## Sorted Linked List to Balanced BST

Given a Singly Linked List which has data members sorted in ascending order. Construct a Balanced Binary Search Tree which has same data members as the given Linked List.

Examples:

```
Input:  Linked List 1->2->3
Output: A Balanced BST
     2
   /  \
  1    3


Input: Linked List 1->2->3->4->5->6->7
Output: A Balanced BST
        4
      /   \
     2     6
   /  \   / \
  1   3  4   7

Input: Linked List 1->2->3->4
Output: A Balanced BST
      3
    /  \
   2    4
 /
1

Input:  Linked List 1->2->3->4->5->6
Output: A Balanced BST
      4
    /   \
   2     6
 /  \   /
1   3  5
```

**Method 1 (Simple)**

Following is a simple algorithm where we first find the middle node of list and make it root of the tree to be constructed.

```
1) Get the Middle of the linked list and make it root.
2) Recursively do same for left half and right half.
       a) Get the middle of left half and make it left child of the root
          created in step 1.
       b) Get the middle of right half and make it right child of the
          root created in step 1.
```

Time complexity: O(nLogn) where n is the number of nodes in Linked List.

See this forum thread for more details.

**Method 2 (Tricky)**

The method 1 constructs the tree from root to leaves. In this method, we construct from leaves to root. The idea is to insert nodes in BST in the same order as the appear in Linked List, so that the tree can be constructed in O(n) time complexity. We first count the number of nodes in the given Linked List. Let the count be n. After counting nodes, we take left n/2 nodes and recursively construct the left subtree. After left subtree is constructed, we allocate memory for root and link the left subtree with root. Finally, we recursively construct the right subtree and link it with root.

While constructing the BST, we also keep moving the list head pointer to next so that we have the appropriate pointer in each recursive call.

Following is C implementation of method 2. The main code which creates Balanced BST is highlighted.

## C

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

/* Link list node */
struct LNode
{
    int data;
    struct LNode* next;
};

/* A Binary Tree node */
struct TNode
{
    int data;
    struct TNode* left;
    struct TNode* right;
};

struct TNode* newNode(int data);
int countLNodes(struct LNode *head);
struct TNode* sortedListToBSTRecur(struct LNode **head_ref, int n);


/* This function counts the number of nodes in Linked List and then calls
   sortedListToBSTRecur() to construct BST */
struct TNode* sortedListToBST(struct LNode *head)
{
    /*Count the number of nodes in Linked List */
    int n = countLNodes(head);

    /* Construct BST */
    return sortedListToBSTRecur(&head, n);
}

/* The main function that constructs balanced BST and returns root of it.
       head_ref -->  Pointer to pointer to head node of linked list
       n  --> No. of nodes in Linked List */
struct TNode* sortedListToBSTRecur(struct LNode **head_ref, int n)
{
    /* Base Case */
    if (n <= 0)
        return NULL;

    /* Recursively construct the left subtree */
    struct TNode *left = sortedListToBSTRecur(head_ref, n/2);

    /* Allocate memory for root, and link the above constructed left
       subtree with root */
    struct TNode *root = newNode((*head_ref)->data);
    root->left = left;

    /* Change head pointer of Linked List for parent recursive calls */
    *head_ref = (*head_ref)->next;

    /* Recursively construct the right subtree and link it with root
      The number of nodes in right subtree  is total nodes - nodes in
      left subtree - 1 (for root) which is n-n/2-1*/
    root->right = sortedListToBSTRecur(head_ref, n-n/2-1);

    return root;
}



/* UTILITY FUNCTIONS */

/* A utility function that returns count of nodes in a given Linked List */
int countLNodes(struct LNode *head)
{
```

```
    int count = 0;
    struct LNode *temp = head;
    while(temp)
    {
        temp = temp->next;
        count++;
    }
    return count;
}

/* Function to insert a node at the beginging of the linked list */
void push(struct LNode** head_ref, int new_data)
{
    /* allocate node */
    struct LNode* new_node =
        (struct LNode*) malloc(sizeof(struct LNode));
    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct LNode *node)
{
    while(node!=NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

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

    return node;
}

/* A utility function to print preorder traversal of BST */
void preOrder(struct TNode* node)
{
    if (node == NULL)
        return;
    printf("%d ", node->data);
    preOrder(node->left);
    preOrder(node->right);
}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct LNode* head = NULL;

    /* Let us create a sorted linked list to test the functions
     Created linked list will be 1->2->3->4->5->6->7 */
    push(&head, 7);
    push(&head, 6);
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
```

```
    push(&head, 1);

    printf("\n Given Linked List ");
    printList(head);

    /* Convert List to BST */
    struct TNode *root = sortedListToBST(head);
    printf("\n PreOrder Traversal of constructed BST ");
    preOrder(root);

    return 0;
}
```

## Java

```
class LinkedList {

    /* head node of link list */
    static LNode head;

    /* Link list Node */
    class LNode
    {
        int data;
        LNode next, prev;

        LNode(int d)
        {
            data = d;
            next = prev = null;
        }
    }

    /* A Binary Tree Node */
    class TNode
    {
        int data;
        TNode left, right;

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

    /* This function counts the number of nodes in Linked List
       and then calls sortedListToBSTRecur() to construct BST */
    TNode sortedListToBST()
    {
        /*Count the number of nodes in Linked List */
        int n = countNodes(head);

        /* Construct BST */
        return sortedListToBSTRecur(n);
    }

    /* The main function that constructs balanced BST and
       returns root of it.
       n  --> No. of nodes in the Doubly Linked List */
    TNode sortedListToBSTRecur(int n)
    {
        /* Base Case */
        if (n <= 0)
            return null;

        /* Recursively construct the left subtree */
        TNode left = sortedListToBSTRecur(n / 2);

        /* head_ref now refers to middle node,
```

```
        make middle node as root of BST*/
    TNode root = new TNode(head.data);

    // Set pointer to left subtree
    root.left = left;

    /* Change head pointer of Linked List for parent
       recursive calls */
    head = head.next;

    /* Recursively construct the right subtree and link it
       with root. The number of nodes in right subtree  is
       total nodes - nodes in left subtree - 1 (for root) */
    root.right = sortedListToBSTRecur(n - n / 2 - 1);

    return root;
}

/* UTILITY FUNCTIONS */
/* A utility function that returns count of nodes in a
   given Linked List */
int countNodes(LNode head)
{
    int count = 0;
    LNode temp = head;
    while (temp != null)
    {
        temp = temp.next;
        count++;
    }
    return count;
}

/* Function to insert a node at the beginging of
   the Doubly Linked List */
void push(int new_data)
{
    /* allocate node */
    LNode new_node = new LNode(new_data);

    /* since we are adding at the begining,
       prev is always NULL */
    new_node.prev = null;

    /* link the old list off the new node */
    new_node.next = head;

    /* change prev of head node to new node */
    if (head != null)
        head.prev = new_node;

    /* move the head to point to the new node */
    head = new_node;
}

/* Function to print nodes in a given linked list */
void printList(LNode node)
{
    while (node != null)
    {
        System.out.print(node.data + " ");
        node = node.next;
    }
}

/* A utility function to print preorder traversal of BST */
void preOrder(TNode node)
{
    if (node == null)
        return;
    System.out.print(node.data + " ");
    preOrder(node.left);
    preOrder(node.right);
```

```
        preOrder(node.right);
    }

    /* Drier program to test above functions */
    public static void main(String[] args) {
        LinkedList llist = new LinkedList();

        /* Let us create a sorted linked list to test the functions
           Created linked list will be 7->6->5->4->3->2->1 */
        llist.push(7);
        llist.push(6);
        llist.push(5);
        llist.push(4);
        llist.push(3);
        llist.push(2);
        llist.push(1);

        System.out.println("Given Linked List ");
        llist.printList(head);

        /* Convert List to BST */
        TNode root = llist.sortedListToBST();
        System.out.println("");
        System.out.println("Pre-Order Traversal of constructed BST ");
        llist.preOrder(root);
    }
}

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

Time Complexity: O(n)