

## In-place conversion of Sorted DLL to Balanced BST

Given a Doubly Linked List which has data members sorted in ascending order. Construct a **Balanced Binary Search Tree** which has same data members as the given Doubly Linked List. The tree must be constructed in-place (No new node should be allocated for tree conversion)

Examples:

Input: Doubly Linked List 1 <--> 2 <--> 3

Output: A Balanced BST

```

      2
     / \
    1   3
  
```

Input: Doubly Linked List 1 <--> 2 <--> 3 <--> 4 <--> 5 <--> 6 <--> 7

Output: A Balanced BST

```

      4
     / \
    2   6
   / \ / \
  1  3 4  7
  
```

Input: Doubly Linked List 1 <--> 2 <--> 3 <--> 4

Output: A Balanced BST

```

      3
     / \
    2   4
  
```

Google™ Custom Search



GeeksforGeeks



53,527 people like [GeeksforGeeks](#).



[Facebook](#)

[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

[Backtracking](#)

[Pattern Searching](#)

[Divide & Conquer](#)

[Mathematical Algorithms](#)

[Recursion](#)

[Geometric Algorithms](#)

```
/
1
```

Input: Doubly Linked List 1 <--> 2 <--> 3 <--> 4 <--> 5 <--> 6

Output: A Balanced BST

```
      4
     / \
    2   6
   / \ /
  1  3 5
```

The Doubly Linked List conversion is very much similar to [this Singly Linked List problem](#) and the method 1 is exactly same as the method 1 of [previous post](#). Method 2 is also almost same. The only difference in method 2 is, instead of allocating new nodes for BST, we reuse same DLL nodes. We use prev pointer as left and next pointer as right.

### 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:  $O(n \log n)$  where  $n$  is the number of nodes in Linked List.

### 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 they appear in Doubly 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 assign middle node to

## Fast XML Conversion Tool

 [adeptia.com/XML-Conversion](https://adeptia.com/XML-Conversion)

Convert Between Any Formats. Try Adeptia's ETL Suite For Free!



## Popular Posts

[All permutations of a given string](#)

[Memory Layout of C Programs](#)

[Understanding "extern" keyword in C](#)

[Median of two sorted arrays](#)

[Tree traversal without recursion and without stack!](#)

[Structure Member Alignment, Padding and Data Packing](#)

[Intersection point of two Linked Lists](#)

[Lowest Common Ancestor in a BST.](#)

[Check if a binary tree is BST or not](#)

[Sorted Linked List to Balanced BST](#)

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.

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

/* A Doubly Linked List node that will also be used as a tree node */
struct Node
{
    int data;

    // For tree, next pointer can be used as right subtree pointer
    struct Node* next;

    // For tree, prev pointer can be used as left subtree pointer
    struct Node* prev;
};

// A utility function to count nodes in a Linked List
int countNodes(struct Node *head);

struct Node* sortedListToBSTRecur(struct Node **head_ref, int n);

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

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

/* The main function that constructs balanced BST and returns root of
head_ref --> Pointer to pointer to head node of Doubly linked
n --> No. of nodes in the Doubly Linked List */
struct Node* sortedListToBSTRecur(struct Node **head_ref, int n)
{
    /* Base Case */
```

```

if (n <= 0)
    return NULL;

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

/* head_ref now refers to middle node, make middle node as root of
struct Node *root = *head_ref;

// Set pointer to left subtree
root->prev = 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) */
root->next = sortedListToBSTRecur(head_ref, n-n/2-1);

return root;
}

```

```

/* UTILITY FUNCTIONS */
/* A utility function that returns count of nodes in a given Linked Li
int countNodes(struct Node *head)
{
    int count = 0;
    struct Node *temp = head;
    while(temp)
    {
        temp = temp->next;
        count++;
    }
    return count;
}

```

```

/* Function to insert a node at the beginging of the Doubly Linked Lis
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

```

## Recent Comments

Abhi You live US or India?

Google (Mountain View) interview · 44 minutes ago

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 1 hour ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 1 hour ago

Sanjay Agarwal bool

tree::Root\_to\_leaf\_path\_given\_sum(tree...

Root to leaf path sum equal to a given number · 1 hour ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...


Find subarray with given sum · 2 hours ago

AdChoices 

► [Binary Tree](#)

► [Red Black Tree](#)

► [Convert BST](#)

AdChoices 

► [Convert DLL](#)

```
/* 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_ref);

/* change prev of head node to new node */
if((*head_ref) != NULL)
    (*head_ref)->prev = new_node ;

/* 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 Node *node)
{
    while(node!=NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

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

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

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

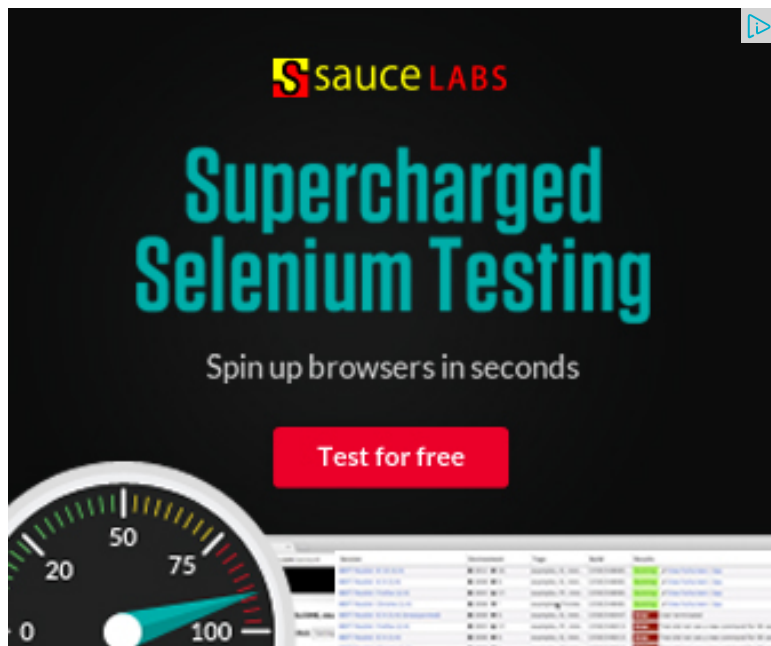
/* Convert List to BST */
struct Node *root = sortedListToBST(head);
printf("\n PreOrder Traversal of constructed BST ");
preOrder(root);

return 0;
}

```

Time Complexity:  $O(n)$

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Related Topics:

- Given a linked list, reverse alternate nodes and append at the end

- Pairwise swap elements of a given linked list by changing links
- Self Organizing List | Set 1 (Introduction)
- Merge a linked list into another linked list at alternate positions
- QuickSort on Singly Linked List
- Delete N nodes after M nodes of a linked list
- Design a stack with operations on middle element
- Swap Kth node from beginning with Kth node from end in a Linked List



1



Tweet

0



1

Writing code in comment? Please use [ideone.com](https://ideone.com) and share the link here.

29 Comments

GeeksforGeeks

Sort by Newest ▾



Join the discussion...



**Tinku** • 22 days ago

@GeeksforGeeks: In the Method2, when structure of type node has been defined, written as next pointer can be used as left subtree pointer. So please correct it, subtree and next pointer will point to right subtree.

1 ^ | ▾ • Reply • Share ›



**GeeksforGeeks** Mod → Tinku • 22 days ago

Thanks for pointing this out. We have updated the comments.

^ | ▾ • Reply • Share ›



**alien** • a month ago

Excellent

^ | ▾ • Reply • Share ›



**SwaS** • 7 months ago



How is method 1 different from method 2 ?

Both also work in the same fashion.. Using the recursion properly we can achieve the implementation of method 1 is not provided ..I think the author intends to update the value for current node before making recursive calls for left & right subtrees. Then to use recursion properly and return the root node at each level we don't need to update the value at each level. We get these values as sub-problems in deeper levels of recursion are  $O(n)$ .

^ | v • Reply • Share ›



**Arpit Baheti** • 7 months ago

For method 1:

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
/*
```

```
sample input:
```

```
9
```

```
10 5 14 1 8 7 11 16 13
```

```
*/
```

```
typedef struct tree
```

```
{
```

[see more](#)

^ | v • Reply • Share ›





**Apurv Kagrawal** · 9 months ago

I think there is a bug in this.

When you go to make the right subtree, it will change root to point to the next element in the list and then the root will be null. So you need to store the root in a temp variable before making the right subtree and then change the root to the temp variable.

After you return from this recursive call to right subtree, you cannot set root->right = sortedListToBSTRecur(head\_ref, n-n/2-1); because root has already been incremented. So I guess we need to store it in a temp and then change that temp?

^ | v · Reply · Share ›



**Ayush Jain** → Apurv Kagrawal · 7 months ago

Even if head\_ref is stored in a temp, it will not work because they point to the same list and once root->next is null, the list is also modified. exception. Correct me if I'm wrong.

^ | v · Reply · Share ›



**abhishek08aug** · a year ago

Intelligent :D

^ | v · Reply · Share ›



**ajinkya** · a year ago

```
#include <iostream>
#include <conio.h>
```

```
using namespace std;
```

```
struct DLinkedList
```

```
{
```

```
    int data ;
```

```
    DLinkedList *pPrev;
```

```
    DLinkedList *pNext;
```

```
};
```

```

DLinkedList* FindMiddleElement( DLinkedList *pStart , DLinkedList **p1 , DL:
{
    DLinkedList *pTemp = pStart , *pFast = NULL ;

    if(NULL == pTemp)
    {

```

[see more](#)

^ | v • Reply • Share ›



**Ankit Gupta** • 2 years ago

$O(N * \log N)$

[sourcecode language="java"]

```

public Node createBST(Node head, Node rear) {
    if (head == rear) {
        head.left = head.right = null;
        return head;
    }
    Node mid = getMidNode(head, rear);
    Node L = null;
    if (head != mid) {
        L = createBST(head, mid.left);
    }
    mid.left = L;
    Node R = null;
    if (rear != mid) {
        R = createBST(mid.right, rear);
    }
    mid.right = R;

```

[see more](#)

^ | v • Reply • Share ›



**BlackMath** · 2 years ago

Hi all,

I tried implementing the method 2 in java.  
I have written the same code i think.  
But things not working as they should be.  
My input list is {1->2->3->4->5->6}.

I am pasting my code here.  
Please could anyone tell me whats wrong.

```
/* Paste your code here (You may delete these lines if not writing code)
import java.util.*;

class DLLListNode
{
    int value;
    DLLListNode next;
    DLLListNode prev;
```

[see more](#)

^ | v · [Reply](#) · [Share](#) ›



**sasuke** → **BlackMath** · 2 years ago

Java is not pass by reference.the linked list head you are passing through you can do it with "ref" keyword in C#.i don't know if there are other alternatives  
In place code btw.you created new TNodes.

^ | v · [Reply](#) · [Share](#) ›



**archit** · 2 years ago

What about conversion of BST in to sorted DLL aur conversion of binary tree in the binary tree?

in the binary tree?

^ | v • Reply • Share ›



Avinash • 2 years ago

### Implementation of Method1

```
/* Paste your code here (You may delete these lines if not writing c)
struct node *ConvertTreefromDll(struct node *head)
{
    If (head==NULL || head->next == NULL) return head;
    struct node *p,*q, *temp;

    temp=FindMiddle(head);

    p=head;
    while(p->next!=temp) p=p->next;
    p->next=NULL;

    q=temp->next;
    temp->next=NULL;
    temp->prev=ConvertTreefromDll(head);
    temp->next=ConvertTreefromDll(q);

    return temp;
}
```

^ | v • Reply • Share ›



Pranav ➔ Avinash • 2 years ago

### Optimised method1

```
/* Paste your code here (You may delete these lines if not wri
struct node *ConvertTreefromDll(struct node *head)
```

Are you a developer? Try out the [HTML to PDF API](#)

```

{
If (head==NULL || head->next == NULL)
    return head;
struct node *q, *temp;
temp=FindMiddle(head);

temp->prev->next=NULL;

q=temp->next;
temp->next=NULL;
temp->prev=ConvertTreefromDll(head);
temp->next=ConvertTreefromDll(q);
return temp;
}
*/

```

^ | v • Reply • Share ›



**kartikaditya** • 2 years ago

```

Node* toBalancedBst(Node** node, int size) {
    if (size == 0) {
        return NULL;
    }

    // Distribute elements across left and right sub-trees
    int lsize = 0, rsize = 0;
    int assorted = 0, levelSize = 1;
    for (int i = size >> 1; i != 1; i = i >> 1) {
        lsize += levelSize;
        rsize += levelSize;
        levelSize = levelSize << 1;
    }
}

```

```
        assorted += levelSize;
    }
    if (size - 1 - assorted >= levelSize) {
        lsize += levelSize;
        rsize += size - 1 - assorted - levelSize;
    } else {
```

[see more](#)

^ | v • Reply • Share ›



**kartikaditya** • 2 years ago

[sourcecode language="C++"]

```
Node* toBalancedBst(Node** node, int size) {
    if (size == 0) {
        return NULL;
    }
```

```
// Distribute elements across left and right sub-trees
```

```
int lsize = 0, rsize = 0;
int assorted = 0, levelSize = 1;
for (int i = size >> 1; i != 1; i = i >> 1) {
    lsize += levelSize;
    rsize += levelSize;
    levelSize = levelSize << 1;
    assorted += levelSize;
}
if (size - 1 - assorted >= levelSize) {
    lsize += levelSize;
    rsize += size - 1 - assorted - levelSize;
```

[see more](#)

^ | v • Reply • Share ›



**vkjk89** · 2 years ago

Can somebody please explain about the time complexity of method1 ?  
How its  $O(n \log n)$  ?

^ | v · Reply · Share ›



**GeeksforGeeks** → vkjk89 · 2 years ago

See the explanation given on below link of a previous post:

<http://www.geeksforgeeks.org/a...>

^ | v · Reply · Share ›



**vkjk89** → GeeksforGeeks · 2 years ago

Hi ..

Thanks :)

Got it ..

^ | v · Reply · Share ›



**saurabh** · 2 years ago

\Round5 | Q2 : Suppose, Amazon have a Logging system Like This:

They Track all logs daily basis, stored in separate log file. Log contains a collection of Page ID. The length of Customer ID and Page ID is L1 and L2. Suppose We have a log file can be Order of  $n$ , where  $n$  be the number of customer.

In a most generalized situation, you can assume that a customer can visit the system any number of days.

We are interested to find out the number of Distinct customer visited at-least  $p$  times.

Propose a Data Structure to be use to solve this problem efficiently . Design a algorithm. Find out the complexity of the algorithm.

{Hints:- Double Hashing/ {Hashing + Tries/BST }}

^ | v · Reply · Share ›



**kartik** → saurabh · 2 years ago

@saurabh: Thanks for sharing the question. I have posted the question

^ | v · Reply · Share ›



**Dumanshu** · 2 years ago

somebody please explain by taking an example for method 2..

^ | v · Reply · Share ›



**kartik** → Dumanshu · 2 years ago

@Dumanshu: Please read the explanation provided before the C code not clear in code.

^ | v · Reply · Share ›



**Dumanshu** → kartik · 2 years ago

From the method 2:

After left subtree is constructed, we allocate memory for root and we recursively construct the right subtree and link it with root.

Don't you think its same as of method 1?

Please correct me.

Lets say I have list as 1,2,3,4,5,6

Method2:

First recursively construct left sub tree which would be

2

1 3

at the end of recursive calls. Same is for the right subtree like

5

6

4 is the root

how is it different from method 1?

^ | v · Reply · Share ›





**kartik** → Dumanshu • 2 years ago

Yes, it is similar to method 1. But in method 1, we first traverse the linked list. And this linear searching makes method 2, we link tree nodes in same order as they appear to construct the tree in  $O(n)$  time complexity only.

^ | v • Reply • Share ›



**Ayush** → kartik • a year ago

I still don't know how it's  $O(n)$  and not  $O(n \log n)$

I mean in  $n \log n$  one is finding the middle using some linear recursion on left and right subtree right?

so in  $O(n)$  one is finding the middle and then applying the recursion on left and right and then joining it with the root ....so where is the time taken?

1 ^ | v • Reply • Share ›



**Guest** → Ayush • 2 months ago

Every node in DLL is visited once so  $O(n)$ .

^ | v • Reply • Share ›



**Dumanshu** → kartik • 2 years ago

Yes.. thanks a lot! :)

```
/* Paste your code here (You may delete these lines) */
```

^ | v • Reply • Share ›

@geeksforgeeks, **Some rights reserved**

**Contact Us!**

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team