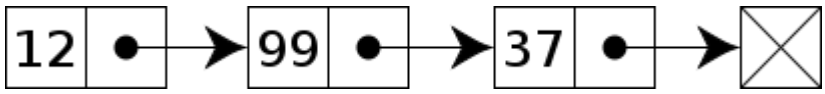
Convert Sorted List to Balanced Binary Search Tree (BST)

November 27, 2010 by 1337c0d3r 80 Replies

> Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

If you have not checked out my previous post: Convert Sorted Array to Balanced Binary Search Tree (BST), you should check it out now as this solution is built upon the previous solution.

Things get a little more complicated when you have a singly linked list instead of an array. Please note that in linked list, you no longer have random access to an element in $O(1)$ time.



Singly-linked lists contain nodes which have a data field as well as a *next* field, which points to the next node in the linked list.

Naive Solution:

A naive way is to apply the previous solution directly. In each recursive call, you would have to traverse half of the list's length to find the middle element. The run time complexity is clearly $O(N \lg N)$, where N is the total number of elements in the list. This is because each level of recursive call requires a total of $N/2$ traversal steps in the list, and there are a total of $\lg N$ number of levels (ie, the height of the balanced tree).

Hint:

How about inserting nodes following the list's order? If we can achieve this, we no longer need to find the middle element, as we are able to traverse the list while inserting nodes to the tree.

Best Solution:

As usual, the best solution requires you to think from another perspective. In other words, we no longer create nodes in the tree

using the top-down approach. We create nodes bottom-up, and assign them to its parents. The bottom-up approach enables us to access the list in its order while creating nodes.

Isn't the bottom-up approach neat? Each time you are stucked with the top-down approach, give bottom-up a try. Although bottom-up approach is not the most natural way we think, it is extremely helpful in some cases. However, you should prefer top-down instead of bottom-up in general, since the latter is more difficult to verify in correctness.

Below is the code for converting a singly linked list to a balanced BST. Please note that the algorithm requires the list's length to be passed in as the function's parameters. The list's length could be found in $O(N)$ time by traversing the entire list's once. The recursive calls traverse the list and create tree's nodes by the list's order, which also takes $O(N)$ time. Therefore, the overall run time complexity is still $O(N)$.

```
BinaryTree* sortedListToBST(ListNode *& list, int start, int end) {
  if (start > end) return NULL;
  // same as (start+end)/2, avoids overflow
  int mid = start + (end - start) / 2;
  BinaryTree *leftChild = sortedListToBST(list, start, mid-1);
  BinaryTree *parent = new BinaryTree(list->data);
  parent->left = leftChild;
  list = list->next;
  parent->right = sortedListToBST(list, mid+1, end);
  return parent;
}

BinaryTree* sortedListToBST(ListNode *head, int n) {
  return sortedListToBST(head, 0, n-1);
}
```

Rating: 4.6/**5** (159 votes cast)

Convert Sorted List to Balanced Binary Search Tree (BST), 4.6 out of 5 based on 159 ratings

← Convert Sorted Array to Balanced Binary Search Tree (BST) Convert Binary Search Tree (BST) to Sorted Doubly-Linked List →

80 thoughts on "Convert Sorted List to Balanced Binary Search Tree (BST)"

Anonymous
November 27, 2010 at 9:52 pm

Pretty darn smart and clean.
I wonder if there is an iterative version, since you have the size n, you can literally scratch out the tree.(knowing exactly where

nodes should be placed).

Reply ↓ 0

Rocky
November 25, 2012 at 6:19 pm

I think we could convert the recursive version to iterative version through using Stack.

Reply ↓ Report user 0

tk
October 17, 2014 at 8:35 pm

Yes, there is iterative solution. This solution is actually in-order traversing the balanced BST. As we have iterative solution for in-order traversal, we can solve this problem iteratively.

Reply ↓ +1

Holger Dürer
November 29, 2010 at 2:37 am

Now, is there a version as neat and tidy as this which does not require you to know the length of the list beforehand?

Reply ↓ -2

Anonymous
November 28, 2011 at 2:04 am

If there could have been one such algo in O(N) then why would avl tree insertion take O(NlogN) for inserting N elements (and suppose N is not known).
Hence we need to know the value of N in advance along with the data.
It is similar to buildHeap where build Heap takes O(N) if all numbers are given aprior . But in our case we have numbers in sorted order.

Reply ↓ +1

data
November 2, 2013 at 3:34 pm

As the algorithm in O(n), we only add a constant factor by iterating over the list once to get its length. Therefore, knowing the length or not is (mostly) irrelevant.

Reply ↓ **0**

abayley
November 29, 2010 at 3:16 am

I see that the *&list parameter is modified. Wondering if there is a solution that does not modify *&list e.g. for languages like Haskell?

Reply ↓ **0**

godheadatomic
November 29, 2010 at 8:00 am

An iterative approach would be to traverse the list in order, and at each node insert into your new binary tree and rebalance (rotate) if necessary. This would work for a list of unknown length.

Reply ↓ **0**

1337c0d3r
November 29, 2010 at 5:20 pm

@Anonymous:
The solution is highly recursive and definitely isn't a simple tail recursion, therefore converting it to an iterative one is definitely non-trivial.

However, I have an iterative solution in mind.

Since we know the size n, we already know the height of the tree, and how many leaves the tree has (ie, how many nodes at the bottom level).

We first create an empty tree with the correct structure, then do an iterative in-order traversal of the tree setting values into the nodes as we traverse the list. The overall complexity is still O(N).

Reply ↓ **+19**

ying
January 17, 2014 at 4:24 pm

Don’t need inorder traversal actually. We can create a pointer array which store pointer to every node. Then you only need N. Inorder Traversal best case is O(N), but larger than 1N.

Reply ↓ **-2**

tk
October 17, 2014 at 8:27 pm

Yes, this is a better perspective to reach the solution.

Reply ↓ **0**

1337c0d3r
November 29, 2010 at 5:23 pm

@Holger:
Without knowing the length of the list, you wouldn't know how the balanced tree structure is like, since you don't know its height and such. You have to re-balance the tree each time you insert an element, which complexity is O(log(N)).

Reply ↓ **0**

Xiaoguang Xu
December 14, 2013 at 10:44 pm

@1337c0d3r:
how about build a complete tree of height 1, then build bigger parent and bigger sibling if enough nodes in list. So we have a complete tree of height 2, and do it recursively. the most unbalanced one is a root with a complete tree as left child and NULL as right child. Still balanced.

Reply ↓ Report user **0**

Xiaoguang Xu
December 16, 2013 at 2:48 am

Wrong Answer:
Input: {-1,0,1,2}
Output: {2,0,#,-1,1}
Expected: {1,0,2,-1}

This is what i got from OJ, seems printed in level order.
I think the result should not be judged with exactly one answer.

0

Reply ↓ Report user

hpPlayer
October 23, 2015 at 6:31 pm

Bro, I don't know if you can still see this, but the problem requires a height balanced BST. Your output is not balanced

Reply ↓ **0**

1337c0d3r
November 29, 2010 at 5:26 pm

@abayley:
The list parameter is required to be passed as reference, as its pointer is changed as we traverse the list. If you couldn't use reference in a language, maybe a simple global variable will do?

Reply ↓ **0**

1337c0d3r
November 29, 2010 at 5:27 pm

@godheadatomic:
I assume your rebalance (rotate) operation would take O(log(N)) time. Therefore, your iterative approach's run time complexity is O(N log(N)).

Reply ↓ **0**

Matthew Leon Grinshpun
November 29, 2010 at 6:11 pm

@abayley and 1337c0d3r:

I have posted a Haskell solution as a gist: https://gist.github.com/720950

Once I get my new blog up and running I intend to write a bit about it, but I hope my style is fairly clear.

–Matthew Leon Grinshpun

Reply ↓ **0**

Anonymous

November 30, 2010 at 2:05 am

@1337c0d3r
About your response to @abayley, does the pointer have to be a reference? Since we are passing it as an argument, is it possible to just use a pointer?
I guess I'm just confused why it has to be passed by reference.

Reply ↓ **0**

Algoseekar
March 6, 2011 at 6:24 am

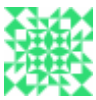
@ihas1337code will create BST or Binary Tree please reply asap…i think it will create binary tree try for linked list has data values as 1 2 3 4 5..& please let mem know

Reply ↓ **0**

hendry
March 10, 2011 at 9:09 am

I think there is a bug in line 9
parent->right = sortedListToBST(list, mid+1, end);
mid+1 and end is right for the function input, but list is already updated at line 5 and line 8, so the index mid+1 to end is no longer right here.

Reply ↓ **0**



Andliory
March 20, 2011 at 10:46 pm

I agree with 12L. maybe it's a bug

Reply ↓ **0**

sheen
March 22, 2011 at 7:39 am

I may be wrong. I think the implementation is incorrect.

list = list->next;

We should change this statement to while loop until we meet the mid one. Otherwise, there is no point we pass in start and end. And the tree wouldn't be balanced bst.

Correct me if I am wrong.

Reply ↓ 0

sheen
March 22, 2011 at 11:20 am

Please ignore the previous post I did. Now I understand. Very good solution.

Reply ↓ 0

algorist2011
March 31, 2011 at 4:02 am

Hi can you please explain the approach above…. this is Pretty recursive in nature algorithm. And i am not getting the approach.. Please examplify!! 🙂

Reply ↓ 0

someone
November 30, 2013 at 12:33 pm

Because the variable is passed as a reference, the multiple increments that sheen was expecting actually do happen in the recursion.

Reply ↓ 0

xTristan
April 4, 2011 at 9:24 am

This approach is elegant. Great! Thanks!

For languages without passing-by-reference arguments, this might be a bit trickier.

Would this simpler algo work:

Given you know the length of your list (the same assumption you are making for your algo. If not, an extra linear scan would suffice as well), create an array tree_nodes[len]. Note this is not extra space because in order to get the tree constructed you need that much space anyway.

Start from the beginning of your list, for each list node at position i, create a tree node and place it in tree_nodes[i].

After the entire list is scanned, you have an array converted from the linked list. From that everything is trivial.

I admit this is actually 2N because it requires a linear visit to the tree_nodes[len] array. We may be able to get rid of this if at the first loop we can figure out the parent index and child index of each node at constant time. Any thoughts?

pseudo code:

```
for (int i = 0; i next;
}

arrayToBST(tree_nodes, 0, length - 1);

Node arrayToBST (Node[] array, int start, int end) {
     if (start < end)
          return null;

     int mid = start + (end-start)/2;
     Node midNode = array[mid];
     midNode.left = arrayToBST(array, start, mid - 1);
     midNode.right = arrayToBST(array, mid+1, end);
     return midNode;
}
```

Reply ↓ +1

xTristan
April 4, 2011 at 9:26 am

the for loop above got messed up after submitting. re-attach:

for (int i = 0; i < length; i++)
tree_nodes[i] = new Node(head.data);
head = head.next;
}

Reply ↓ 0

speeddy
April 5, 2011 at 11:27 am

So you basically convert the sorted list to sorted array.

Then use the algorithm sorted array to bst?

Reply ↓ 0

xTristan
April 5, 2011 at 9:20 pm

Essentially yes, but not using the exact array algorithm. The first loop creates the tree nodes already, therefore, space-wise, you don't waste any space comparing to the algorithm in this post.

Time-complexity, yes, although both are linear, this approach requires an extra loop, which is bad. But on the other hand, recursion comes with its own price as well. Recursion requires pushing/popping on the call stack each time you recurse, which are not free operations.

Reply ↓ **+1**

Frank
November 15, 2013 at 5:57 pm

Brilliant! A lot cleaner than the recursive one.

0

ying xie
March 29, 2013 at 6:39 pm

I agree with you.
"parent" should be the mid node.

Reply ↓ **0**

Pingback: NEIL

Pingback: Convert a sorted doubly linked list to BST in place | NEIL

Raj
May 9, 2011 at 8:37 pm

Very smart idea and implementation, thanks!

Reply ↓ **0**

Greed
July 3, 2011 at 2:31 pm

Hey can you write code change a sorted doubly linked list to BST

Reply ↓ 0

wwwyhx
December 30, 2011 at 9:14 pm

Inplace makes sense

```
NODE* _inner_construct(NODE* pHead, int nLen)
{
if (NULL == pHead || nLen <= 0)
return NULL;

NODE* pMid = pHead;
for (int i = 0; i pRgt;

pMid->pLft = _inner_construct(pHead, nLen/2);
pMid->pRgt = _inner_construct(pMid->pRgt, nLen-1-nLen/2);

return pMid;
}
```

Reply ↓ -1

opoopo
May 23, 2012 at 9:34 am

Parameter start, end in 1337's code is not necessary, we can use len instead, Java code below:

```
public class Test {
public static Node list = null;
public static TreeNode sortedListToBST(int len) {
if (len <= 0) return null;
TreeNode leftChild = sortedListToBST(len/2);
TreeNode parent = new TreeNode(list.key);
parent.left = leftChild;
list = list.next;
parent.right = sortedListToBST(len-len/2-1);
return parent;
}
public static TreeNode sortedListToBST(Node head, int n) {
list = head;
return sortedListToBST(n);
```

}
}

Reply ↓ 0

Prateek Caire
August 14, 2012 at 9:22 am

Nice solution. However, unlike BST to Linked list solution where Binary tree nodes were used to represent a node in linked list, this solution creates new binary tree nodes. Thus O(N) space is required. i wonder if any solution exists without using extra space? One you already mentioned above, but that take O(NlogN).

Reply ↓ 0

Amit Kumar Swami
August 18, 2012 at 6:40 am

I think there is a bug in program. the line 5 should be placed after line 8. when list is updated. Otherwise left child will add same data as root.

Reply ↓ 0

LT
September 9, 2012 at 12:53 pm

I guess the way to reconstruct a balanced BST is similar to the way of traversing a tree. This question reconstruct the tree using “in-order” traverse.
So if we have another question like “Give a linked list, the numbers in front half size of list is increasing and in back half size of list in decreasing, convert the linked list to balanced BST”.
Can we just simply change answer of this question to “post order” traverse?

```
BinaryTree* sortedListToBST(ListNode *& list, int start, int end) {
if (start > end) return NULL;
// same as (start+end)/2, avoids overflow
int mid = start + (end – start) / 2;
BinaryTree *leftChild = sortedListToBST(list, start, mid-1);
BinaryTree *rightChild = sortedListToBST(list, mid+1, end);
BinaryTree *parent = new BinaryTree(list->data);
parent->left = leftChild;
parent->right = rightChild;
list = list->next;
```

```
return parent;
}

BinaryTree* sortedListToBST(ListNode *head, int n) {
return sortedListToBST(head, 0, n-1);
}
```

Reply ↓ Report user 0

SomeRandomDude
September 29, 2012 at 5:40 pm

Amazing. I had the O(nlogn) solution in mind only 😛

Reply ↓ 0

Pingback: Convert Sorted List to Balanced Binary Search Tree (BST) | This is how Cheng grows up

Benny
October 13, 2012 at 8:56 pm

Converting a Linked List to an Array is O(n).
Then use http://www.leetcode.com/2010/11/convert-sorted-array-into-balanced.html.

The solution would still be O(n).
Much easier and reuse code is always nice to iron out bugs. :)

Reply ↓ 0

itiswatitis
August 16, 2013 at 12:04 am

but 2n space

Reply ↓ Report user 0

sean
November 30, 2012 at 12:37 am

```
private TreeNode sortedListToBST(ref ListNode node, int start, int end)
{
if (start > end)
```

{
return null;
}

int mid = start + (end – start) >> 1;
var l = sortedListToBST(ref node, start, mid – 1);
var r = sortedListToBST(ref node, mid + 1, end);
TreeNode p = new TreeNode(node.Val);
p.LeftChild = l;
p.RightChild = r;
node = node.Next;
return p;
}

tried this in c#, got StackOverflow.

Reply ↓ 0

Pingback: Convert Sorted Array to Binary Search Tree – ZJustin@UMich

ja
February 15, 2013 at 10:49 pm

do we really need to know the length of the link list?

We can start from head. We know it should correspond to 2^0 = 1 in bst. we know we need to stop at 2^1 link-list node, then we know we need to stop at 2^2 link list node, 2^3 and so on….

Reply ↓ 0

Punit Patel
March 8, 2013 at 5:10 pm

Here is the iterative version

```
void balancedBST::insert(int d){
    // create new node
    TreeNodePtr temp = new TreeNode(d);
    if(!root)
        root = temp;
    else{
        TreeNodePtr p = root, ggp = 0, gp = 0;
        while(p->right != 0){
            ggp = gp;
            gp = p;
            p = p->right;
```

```
        }
        p->right = temp;
        // balance the tree
        if(gp && gp->left == 0){ // simple left rotation
            if(ggp)
                ggp->right = p;
            p->left = gp;
            if(root == p->left)
                root = p;
        } else if(ggp){
            ggp -> right = gp -> left;
            gp -> left = ggp;
            if(root == gp->left)
                root = gp;
        }
    }
}
```

Reply ↓ **0**

butlerain

April 7, 2013 at 3:03 pm

```
void buildBST(ListNode * pn, int size){
TreeNode * pt;
if(size==0){
return null;
}else if(size == 1){
pt->val=pn->val;
pt->left = null;
pt->right = null;
return pt;
}else {
ListNode * p=pn;
int middle = size/2;
for(int i = 0; inext;
}
pt->val=p->val;
pt->left = buildBST(pn, size/2);
pt->right = buildBST(p->next, size/2-1);
return pt;
}
}

void main(){
buildBST(head, N);
}
```

0

Reply ↓ Report user

Pingback: leetcode: Convert Sorted List to Binary Search Tree solution | 烟客旅人 sigmainfy

ted cheng
May 23, 2013 at 2:39 pm

Hi, why the following code can't pass the online judge? It's basically the same code as in the "Best Solution" section, just in Java. Thank you for any input.

```
public class Solution {
    public TreeNode sortedListToBST(ListNode head) {
        // Start typing your Java solution below
        // DO NOT write main() function
        int len = 0;
        ListNode node = head;
        while(node != null) {
            len++;
            node = node.next;
        }

        return sortedListToBSTHelper(head, 0, len - 1);
    }

    private TreeNode sortedListToBSTHelper(ListNode list, int start, int end){
        if(start > end) return null;
        int middle = start + (end - start) / 2;
        TreeNode leftNode = sortedListToBSTHelper(list, start, middle - 1);
        TreeNode parentNode = new TreeNode(list.val);
        parentNode.left = leftNode;
        list = list.next;
        parentNode.right = sortedListToBSTHelper(list, middle + 1, end);

        return parentNode;
    }
}
```

Reply ↓ Report user +2

yfcg
August 4, 2013 at 12:08 pm

what kind of error did you get? I think because in Java parameter are passed by reference, so the list didn't get changed. correct me if I'm wrong.

Reply ↓ Report user 0

lowiq3
September 27, 2014 at 4:30 pm

Java is pass by value of reference, the head which got moved is actually a copy, a leeway is to wrap it in an array

Reply ↓ +1

Toby
April 17, 2015 at 10:31 am

replace the line “ list = list.next; "

with the lines below:

```
if (list.next!=null) {
list.val = list.next.val;
list.next = list.next.next;
}
```

Reply ↓ +1

AJ
July 8, 2013 at 9:54 pm

Nice. Pretending to walk a tree in-order and building it while visiting!

Reply ↓ 0

lcn
August 20, 2013 at 1:22 pm

Solution for fixing Java’s no-argument-passed-with-pointer issue: place the head of the list in an array, and this gives us a pseudo “pointer”.

```
    public TreeNode sortedListToBST(ListNode head) {
        // Start typing your Java solution below
        // DO NOT write main() function
        int i=0;
        ListNode node = head;
        while (node!=null) {
            node = node.next;
            i++;
        }

        return buildBST(new ListNode[]{head}, 0, i);

    }
```

```
public TreeNode buildBST(ListNode[] head, int i, int j) {
    if (j==i) return null;

    int mid = (i+j)/2;

    TreeNode left = buildBST(head, i, mid);
    TreeNode root = new TreeNode(head[0].val);
    head[0] = head[0].next;
    TreeNode right = buildBST(head, mid+1, j);
    root.left = left;
    root.right = right;

    return root;
}
```

Reply ↓ Report user +4

imbaqy
August 26, 2013 at 7:28 pm

Why not using a member variable here. Then you do not even need to pass in the head.

Reply ↓ 0

lowiq3
September 27, 2014 at 4:26 pm

not thread safe

Reply ↓ +1

zach
August 27, 2013 at 2:13 pm

cool! To understand this algorithm, just remember that the function
BinaryTree* sortedListToBST(ListNode *& list, int start, int end) actually does the following two main things:
1. It creates a balanced BST for nodes from “start” to “end”
2. During the process in creating the BST, it also moves the “list” pointer to point at the node after the “end” node.

Reply ↓ +6

Feng
January 10, 2014 at 9:33 am

impressive!

Reply ↓ Report user 0

LK

January 16, 2014 at 6:15 am

```
public class SortedListToBalancedBST {

    private static class TreeNode
    {
        private int data;
        private TreeNode left;
        private TreeNode right;

        public TreeNode(int data)
        {
            this.data = data;
            left = null;
            right = null;
        }
    }

    private static class LinklistNode
    {
        private int data;
        private LinklistNode next;

        public LinklistNode(int data)
        {
            this.data = data;
            next = null;
        }
    }

    private static LinklistNode createSortedLinkList(int n) {
        if (n < 1) {
            return null;
        }
        LinklistNode head = new LinklistNode(1);
        LinklistNode tmp = head;
        for (int i = 2; i  endindex) {
            return null;
        }

        int midIndex = startIndex + ((endindex - startIndex)/2);
        TreeNode leftNode = createBSTFromSortedLinklist(startIndex, midIndex - 1);
        TreeNode rootNode = new TreeNode(node.data);
        rootNode.left = leftNode;
        node = node.next;
        TreeNode rightNode = createBSTFromSortedLinklist(midIndex + 1, endindex);
        rootNode.right = rightNode;
        return rootNode;
    }

    private static void inOrderTraversal(TreeNode root) {
        if (root == null) {
            return;
        }
```

```
        inOrderTraversal(root.left);
        System.out.print(root.data + " ");
        inOrderTraversal(root.right);
    }
}
```

Reply ↓ -1

LK
January 16, 2014 at 6:17 am

You need to double click on code to view it

Reply ↓ 0

jason
March 20, 2014 at 4:23 pm

Here is a proof that this algorithm is correct
http://blog.flexdms.com/2014/03/sorted-list-to-height-balanced-binary.html

Reply ↓ 0

Pingback: Fun code to Convert Sorted Array to Binary Search Tree | Changhaz's Codeplay

kiran
April 30, 2014 at 8:42 am

Is the code provided for sortedListToBST correct ? anybody tested this ? I tried testing this and I get a wrong output and a wrong tree was built.

Below is the code that i used.

```
#include
#include


typedef struct single_link_list {
int i;
struct single_link_list *next;
} slist;
```

```
typedef struct binarytree {
int i;
struct binarytree *left;
struct binarytree *right;
} bst;

addelemt(slist **sl, int i)
{
slist *tmp;

if (*sl == NULL)
{
*sl = malloc(sizeof(slist *));
(*sl)->i = i;
(*sl)->next = NULL;
} else {
tmp = malloc(sizeof(slist *));
tmp->i = i;
tmp->next = *sl;
*sl = tmp;
}
}

printlist(slist *sl)
{
slist *tmp;
tmp = sl;
printf("\n printing the list\n");
while (tmp != NULL)
{
printf("%d ", tmp->i);
tmp = tmp->next;
}
}

bst* sortedListToBST(slist *list, int start, int end) {
printf("\n Entered for %d", list->i);

if (start > end) return NULL;

// same as (start+end)/2, avoids overflow
int mid = start + (end - start) / 2;

bst *leftChild = sortedListToBST(list, start, mid-1);
bst *parent = malloc(sizeof(bst));
```

```
parent->i = list->i;
printf("\n creating root for %d", list->i);
parent->left = leftChild;
list = list->next;
parent->right = sortedListToBST(list, mid+1, end);
return parent;
}

preorder_printtree(bst *tree)
{
if (tree != NULL) {
printf(" %d", tree->i);
preorder_printtree(tree->left);
preorder_printtree(tree->right);
}
return;
}

inorder_printtree(bst *tree)
{
if (tree != NULL) {
inorder_printtree(tree->left);
printf(" %d ", tree->i);
inorder_printtree(tree->right);
}
return;
}

main()
{
slist *list1 = NULL;
slist *list2 = NULL;
int carry = 0;
bst *tree;

//addelemt(&list1, 9);
addelemt(&list1, 8);
addelemt(&list1, 7);
addelemt(&list1, 6);
addelemt(&list1, 5);
addelemt(&list1, 4);
addelemt(&list1, 3);
addelemt(&list1, 2);
```

```
addelemt(&list1, 1);
printlist(list1);

tree = sortedListToBST(list1, 0, 7);
//preorder_printtree(tree);
inorder_printtree(tree);

}

expeted output: 1 2 3 4 5 6 7 8
actual output: 3 2 3 1 3 2 3 4
```

Reply ↓ 0

kiran
April 30, 2014 at 9:21 am

My above code has a bug. Its a wrong code. Ignore the query. Thanks

Reply ↓ 0

Pingback: DS conversion | GO 2 Computer Science and Trading Technologies

pyed
September 10, 2014 at 12:23 am

I spend a long time on this code, it’s very tricky but it’s correct actually

Reply ↓ 0

networkguy88
September 28, 2014 at 10:47 am

Since you’re traversing the whole list to find the length, what if you created a hash that mapped a nodes place in the list to the node itself (also requiring N work)? In this example:

hash[0] = 12’s node object
hash[1] = 99’s node object
hash[2] = 33’s node object

The hash would allow you to re-use your sorted array to balanced BST algorithm.

Reply ↓ 0

Pingback: Important Data Structure And Algorithms To Prepare For Interview | Ankit Anand

Kevin__14
December 11, 2014 at 6:07 am

WHY the complexity is O(n), not nlog(n)?
Cause in sortedListToBST( ), we have the deduction T(n) = 2T(n/2) + O(1),
the time complexity should be nlog(n).

Reply ↓ Report user 0

Pingback: [LeetCode] Convert Sorted List to Binary Search Tree, Solution | 水中的鱼(镜像)

Xiaohui Liu
January 30, 2015 at 11:56 am

A slightly improved implementation, hopeful it's easier to grasp.

```
// create a balanced BST using @len elements starting from @head & move @head forward by @len
TreeNode *sortedListToBSTHelper(ListNode *&head, int len) {
    if (0 == len)   return NULL;

    auto left = sortedListToBSTHelper(head, len / 2);
    auto root = new TreeNode(head->val);
    root->left = left;
    head = head->next;
    root->right = sortedListToBSTHelper(head, (len - 1) / 2);
    return root;
}

TreeNode *sortedListToBST(ListNode *head) {
    int n = length(head);
    return sortedListToBSTHelper(head, n);
}
```

Reply ↓ 0

Pingback: Leetcode - Convert Sorted List to Binary Search Tree - Timshel

Pingback: Convert Sorted List to Binary Search Tree | Wenxiang's Evolutionary World

Maha

June 22, 2015 at 5:44 am

Can somebody provide a python version of this code! I find it difficult to comprehend in C and Java

Reply ↓ 0

Pingback: Convert Sorted List to Binary Search Tree | dev4future

Pingback: [LeetCode] Convert Sorted List to Binary Search Tree | thorcsblog

Pingback: Convert Sorted List to Binary Search Tree-IT技术

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

To embed your code, please use <code>your code here</code>.

You may use the <code> tag to embed your code.

Name *

Email *

Website

CAPTCHA Code

*

Post Comment