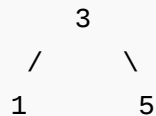


## Merge two BSTs with limited extra space

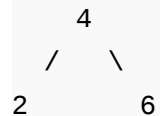
Given two Binary Search Trees(BST), print the elements of both BSTs in sorted form. The expected time complexity is  $O(m+n)$  where  $m$  is the number of nodes in first tree and  $n$  is the number of nodes in second tree. Maximum allowed auxiliary space is  $O(\text{height of the first tree} + \text{height of the second tree})$ .

Examples:

First BST

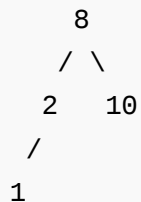


Second BST



Output: 1 2 3 4 5 6

First BST



Second BST



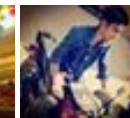
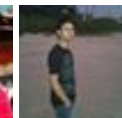
Google™ Custom Search



GeeksforGeeks



52,731 people like GeeksforGeeks.



Interview Experiences

Advanced Data Structures

Dynamic Programming

Greedy Algorithms

Backtracking

Pattern Searching

Divide & Conquer

Mathematical Algorithms

Recursion

Geometric Algorithms

```

    /
   3
  /
 0

```

Output: 0 1 2 3 5 8 10

Source: [Google interview question](#)

A similar question has been discussed earlier. Let us first discuss already discussed methods of the [previous post](#) which was for Balanced BSTs. The method 1 can be applied here also, but the time complexity will be  $O(n^2)$  in worst case. The method 2 can also be applied here, but the extra space required will be  $O(n)$  which violates the constraint given in this question. Method 3 can be applied here but the step 3 of method 3 can't be done in  $O(n)$  for an unbalanced BST.

Thanks to [Kumar](#) for suggesting the following solution.

The idea is to use [iterative inorder traversal](#). We use two auxiliary stacks for two BSTs. Since we need to print the elements in sorted form, whenever we get a smaller element from any of the trees, we print it. If the element is greater, then we push it back to stack for the next iteration.

```

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

// Structure of a BST Node
struct node
{
    int data;
    struct node *left;
    struct node *right;
};

//..... START OF STACK RELATED STUFF.....
// A stack node
struct snode
{
    struct node *t;
    struct snode *next;
};

// Function to add an elemt k to stack
void push(struct snode **s, struct node *k)
{

```



## 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](#)

```

    struct snode *tmp = (struct snode *) malloc(sizeof(struct snode));

    //perform memory check here
    tmp->t = k;
    tmp->next = *s;
    (*s) = tmp;
}

// Function to pop an element t from stack
struct node *pop(struct snode **s)
{
    struct node *t;
    struct snode *st;
    st=*s;
    (*s) = (*s)->next;
    t = st->t;
    free(st);
    return t;
}

// Function to check whether the stack is empty or not
int isEmpty(struct snode *s)
{
    if (s == NULL )
        return 1;

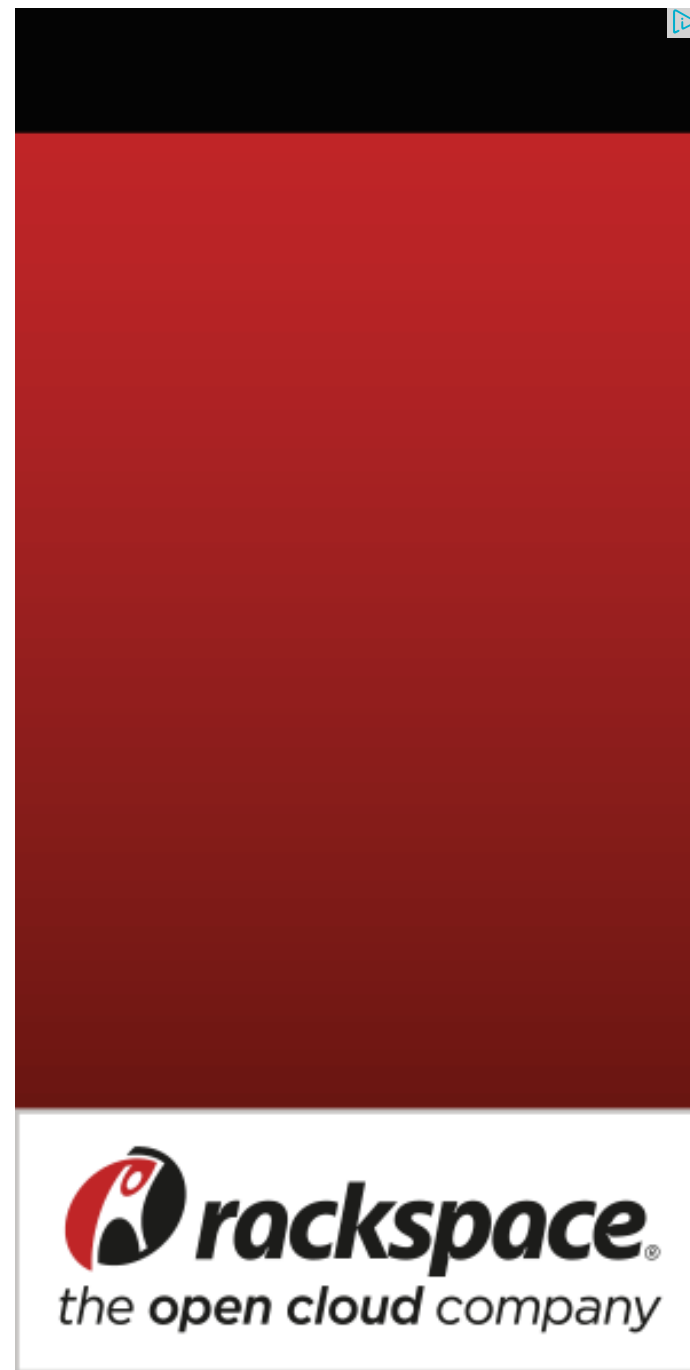
    return 0;
}

//..... END OF STACK RELATED STUFF.....

/* Utility function to create a new Binary Tree node */
struct node* newNode (int data)
{
    struct node *temp = new struct node;
    temp->data = data;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}

/* A utility function to print Inorder traversal of a Binary Tree */
void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);

```



```
printf("%d ", root->data);
inorder(root->right);
}
```

```
}

// The function to print data of two BSTs in sorted order
void merge(struct node *root1, struct node *root2)
{
    // s1 is stack to hold nodes of first BST
    struct snode *s1 = NULL;

    // Current node of first BST
    struct node *current1 = root1;

    // s2 is stack to hold nodes of second BST
    struct snode *s2 = NULL;

    // Current node of second BST
    struct node *current2 = root2;

    // If first BST is empty, then output is inorder
    // traversal of second BST
    if (root1 == NULL)
    {
        inorder(root2);
        return;
    }
    // If second BST is empty, then output is inorder
    // traversal of first BST
    if (root2 == NULL)
    {
        inorder(root1);
        return ;
    }

    // Run the loop while there are nodes not yet printed.
    // The nodes may be in stack(explored, but not printed)
    // or may be not yet explored
    while (current1 != NULL || !isEmpty(s1) ||
           current2 != NULL || !isEmpty(s2))
    {
        // Following steps follow iterative Inorder Traversal
        if (current1 != NULL || current2 != NULL )
        {
            // Reach the leftmost node of both BSTs and push ancestors
            // leftmost nodes to stack s1 and s2 respectively
            if (current1 != NULL)
```

## Recent Comments

affizerv Your example has two 4s on row 3, that's why it...

[Backtracking | Set 7 \(Sudoku\)](#) · 37 minutes ago

**RVM** Can someone please elaborate this Qs from above...

[Flipkart Interview | Set 6](#) · 57 minutes ago

**Vishal Gupta** I talked about as an Interviewer in general,...

[Software Engineering Lab, Samsung Interview | Set 2](#) · 57 minutes ago


**@meya** Working solution for question 2 of 4f2f round....

[Amazon Interview | Set 53 \(For SDE-1\)](#) · 1 hour ago  
sandeep void rearrange(struct node \*head)  
{...

[Given a linked list, reverse alternate nodes and append at the end](#) · 3 hours ago

Neha I think that is what it should return as, in...

[Find depth of the deepest odd level leaf node](#) · 3 hours ago

AdChoices 

[► Binary Tree](#)

[► Java Tree](#)

[► Merge Java](#)

```

        {
            push(&s1, current1);
            current1 = current1->left;
        }
        if (current2 != NULL)
        {
            push(&s2, current2);
            current2 = current2->left;
        }
    }
    else
    {
        // If we reach a NULL node and either of the stacks is empty
        // then one tree is exhausted, print the other tree
        if (isEmpty(s1))
        {
            while (!isEmpty(s2))
            {
                current2 = pop (&s2);
                current2->left = NULL;
                inorder(current2);
            }
            return ;
        }
        if (isEmpty(s2))
        {
            while (!isEmpty(s1))
            {
                current1 = pop (&s1);
                current1->left = NULL;
                inorder(current1);
            }
            return ;
        }

        // Pop an element from both stacks and compare the
        // popped elements
        current1 = pop(&s1);
        current2 = pop(&s2);

        // If element of first tree is smaller, then print it
        // and push the right subtree. If the element is larger,
        // then we push it back to the corresponding stack.
        if (current1->data < current2->data)
        {
            printf("%d ", current1->data);


```

AdChoices 

[► Merge Data](#)

[► Java Array](#)

[► Extra Space](#)

AdChoices 

[► C++ Merge List](#)

[► Tree Root](#)

[► Tree Block](#)

```

        current1 = current1->right;
        push(&s2, current2);
        current2 = NULL;
    }
    else
    {
        printf("%d ", current2->data);
        current2 = current2->right;
        push(&s1, current1);
        current1 = NULL;
    }
}
}

/* Driver program to test above functions */
int main()
{
    struct node *root1 = NULL, *root2 = NULL;

    /* Let us create the following tree as first tree
           3
        /  \
       1    5
    */
    root1 = newNode(3);
    root1->left = newNode(1);
    root1->right = newNode(5);

    /* Let us create the following tree as second tree
           4
        /  \
       2    6
    */
    root2 = newNode(4);
    root2->left = newNode(2);
    root2->right = newNode(6);

    // Print sorted nodes of both trees
    merge(root1, root2);

    return 0;
}

```

Time Complexity:  $O(m+n)$

Auxiliary Space:  $O(\text{height of the first tree} + \text{height of the second tree})$

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



## Related Topics:

- [Print a Binary Tree in Vertical Order | Set 2 \(HashMap based Method\)](#)
- [Print Right View of a Binary Tree](#)
- [Red-Black Tree | Set 3 \(Delete\)](#)
- [Construct a tree from Inorder and Level order traversals](#)
- [Print all nodes at distance k from a given node](#)
- [Print a Binary Tree in Vertical Order | Set 1](#)
- [Interval Tree](#)
- [Check if a given Binary Tree is height balanced like a Red-Black Tree](#)



0

Tweet

0



0

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

**25 Comments**

**GeeksforGeeks**

Sort by Newest ▼

[open in browser](#) [PRO version](#)

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

[pdfcrowd.com](#)



Join the discussion...



**AlienOnEarth** • 4 days ago

The question is to just print merged trees or actually merging them? According that we only need to print 2 merged trees. Not actually merging them.

^ | v • Reply • Share ›



**AlienOnEarth** • 19 days ago

@Geeksforgeeks

I have another recursive solution with  $O(m+n)$  time and  $O(\log(m+n))$  space cor

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

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

```
// Structure of a BST Node
```

```
struct node
```

```
{
```

```
int data;
```

```
struct node *left;
```

```
struct node *right;
```

```
};
```

see more

^ | v • Reply • Share ›





**Guest** • 4 months ago

why are we calling inorder(current1) in while loop  
if (isEmpty(s2))

```
{  
while (!isEmpty(s1))  
{  
current1 = pop (&s1);  
current1->left = NULL;  
inorder(current1);  
}  
return ;  
}
```

2 ^ | v • Reply • Share ›



**powerhu** • 8 months ago

C# Code:

```
public static void AddLeftNodes(Node node, Stack<node> s)  
  
{  
  
while (node != null)  
  
{  
  
s.Push(node);  
  
node = node.left;  
  
}  
  
}
```

{

[see more](#)

^ | v • [Reply](#) • [Share](#) ›



**Amit Bgl** • 9 months ago

wow code :D

^ | v • [Reply](#) • [Share](#) ›



**bateesh** • 10 months ago

@GeeksforGeeks

Why Cant we apply 3rd method of previous post.Its 3rd step is just a merging number of nodes in the list.It will be done in  $O(n)$  regardless of balanced/unbalanced DLL.The problem can be with step 1 where we convert the BST to DLL.Can you why 3rd method of previous post is not feasible here to do in  $O(m+n)$ .

^ | v • [Reply](#) • [Share](#) ›



**saket** → **bateesh** • 9 months ago

Yes I agree with you. 3rd method of previous post is feasible and it does

```
/* Paste your code here (You may delete these lines if not write)
```

^ | v • [Reply](#) • [Share](#) ›



**abhishek08aug** • a year ago

Intelligent :D

^ | v • [Reply](#) • [Share](#) ›



**Arunkumar** • a year ago

Convert the two BST's to DLL inplace then insert the contents of second list in  
convert it back to bst

/\* Paste your code here (You may **delete** these lines **if not** writing c

^ | v • Reply • Share ›



**Rajneesh** • a year ago

/\* Paste your code here (You may **delete** these lines **if not** writing c

^ | v • Reply • Share ›



**rahul sundar** • a year ago

The easiest approach is

- Store in order traversal of 2 trees in 2 desperate arrays
- Since the arrays are already sorted, print the smaller elements from both of t possible in order of m or n , whichever is greater.

The last step ,mentioned above is just a problem of merging 2 sorted arrays!

/\* Paste your code here (You may **delete** these lines **if not** writing c

^ | v • Reply • Share ›



**Priso** → rahul sundar • a year ago

But, Maximum allowed auxiliary space is  $O(\text{height of the first tree} + \text{height of the second tree})$ .  
auxiliary space required would be more i.e.,  $O(m+n)$

/\* Paste your code here (You may **delete** these lines **if not** wr

1 ^ | v • Reply • Share ›



**sachin** • 2 years ago

why the method 3 of previous post will not work in  $O(n)$  for unbalanced bst?



**yc** • 2 years ago

Here is a pure iterative algorithm to merge 2 trees. No stack, not recursion. It is a tweak of the Morris algorithm (but restores it after done), the code guards against the situation when the 2 trees are simply copy trees before calling.

```
[sourcecode language="C++"]
int getNext(node* &cur,bool& set ){
int k=0;
set=false;
while(cur){
if(cur->left){
node *p=cur->left;
while(p->right && (p->right != cur))
p=p->right;
if(!p->right){
p->right=cur;
cur=cur->left;
}else{
```

[see more](#)

^ | v • Reply • Share ›



**titan** • 2 years ago

Providing a slightly better coded version of the method that has been provided

```
void sortedprint(tree *first,tree *second)
{
    stack *s1=NULL,*s2=NULL;
    tree *one,*two;
    while(!isempty(s1)||!isempty(s2)||first||second)
```

```

{
    while(first!=NULL)
    {
        s1=push(first,s1);
        first=first->left;
    }
    while(second!=NULL)
    {
        s2=push(second,s2);
        second=second->left;
    }
}

```

[see more](#)

1 ^ | v • Reply • Share ›



**srikanth** • 2 years ago

it is better to use newNode function without any arguments.a

^ | v • Reply • Share ›



**Doom** • 2 years ago

Given a node pointer of any node in a BST, we can write a method which returns the next element in the BST. Now start with the two BST's root nodes and compare them. Follow the same MergeSort. To get the next element call the getNextInorderNode(currNode). With O(1) auxiliary space. Any flaws in this solution?

```

/* Paste your code here (You may delete these lines if not writing code)

```

^ | v • Reply • Share ›



**Doom** ➔ Doom • 2 years ago

Please ignore my last post. It won't work.

^ | v • Reply • Share ›



**kamal** • 2 years ago

```
public static void inordermodif(Node root1,Node root2){
    if(root1!=null && root2!=null){
        inorderdisplay(root1.left);
        inorderdisplay(root2.left);
        if(root1.data<=root2.data){
            System.out.println(root1.data);
            System.out.println(root2.data);
        }
        else{
            System.out.println(root2.data);
            System.out.println(root1.data);
        }
        inorderdisplay(root1.right);
        inorderdisplay(root2.right);
    }
}
```

^ | v • Reply • Share ›



**kamal** → kamal • 2 years ago

sorry for the last post. It doesn't work!

^ | v • Reply • Share ›



**Ila** • 2 years ago

I think you should check if current-> right is not null before pushing it onto the s

^ | v • Reply • Share ›



ok got it.

^ | v • Reply • Share ›



**Kiran** • 2 years ago

In your comment inside main:

Let us create the following tree as second tree

2

/ \

4 6

This isn't a BST.

```
/* Paste your code here (You may delete these lines if not writing cor
```

^ | v • Reply • Share ›



**GeeksforGeeks** → **Kiran** • 2 years ago

There was a typo in comment. Thanks for poitning this out. We have fi

^ | v • Reply • Share ›



**rahul** • 2 years ago

Recently found this site, great work, hope i will be able to crack some big com

^ | v • Reply • Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

**Contact Us!**

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