

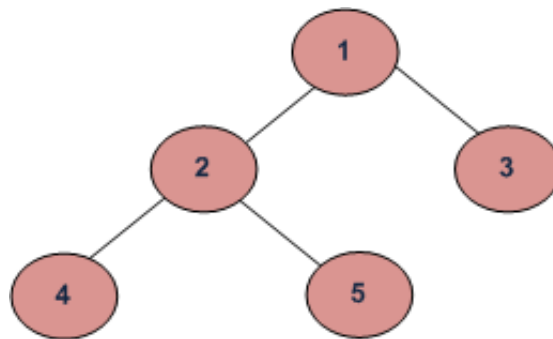
# GeeksforGeeks

A computer science portal for geeks

Placements Practice GATE CS IDE Q&A  
GeeksQuiz

## Reverse Level Order Traversal

We have discussed [level order traversal](#) of a post in previous post. The idea is to print last level first, then second last level, and so on. Like Level order traversal, every level is printed from left to right.



*Example Tree*

Reverse Level order traversal of the above tree is "4 5 2 3 1".

Both methods for [normal level order traversal](#) can be easily modified to do reverse level order traversal.

### METHOD 1 (Recursive function to print a given level)

We can easily modify the method 1 of the normal [level order traversal](#). In method 1, we have a method `printGivenLevel()` which prints a given level number. The only thing we need to change is, instead of calling `printGivenLevel()` from first level to last level, we call it from last level to first level.

## C

```
// A recursive C program to print REVERSE level order traversal
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left and right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};
```

```

/*Function prototypes*/
void printGivenLevel(struct node* root, int level);
int height(struct node* node);
struct node* newNode(int data);

/* Function to print REVERSE level order traversal a tree*/
void reverseLevelOrder(struct node* root)
{
    int h = height(root);
    int i;
    for (i=h; i>=1; i--) //THE ONLY LINE DIFFERENT FROM NORMAL LEVEL ORDER
        printGivenLevel(root, i);
}

/* Print nodes at a given level */
void printGivenLevel(struct node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d ", root->data);
    else if (level > 1)
    {
        printGivenLevel(root->left, level-1);
        printGivenLevel(root->right, level-1);
    }
}

/* Compute the "height" of a tree -- the number of
nodes along the longest path from the root node
down to the farthest leaf node.*/
int height(struct node* node)
{
    if (node==NULL)
        return 0;
    else
    {
        /* compute the height of each subtree */
        int lheight = height(node->left);
        int rheight = height(node->right);

        /* use the larger one */
        if (lheight > rheight)
            return(lheight+1);
        else return(rheight+1);
    }
}

/* 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 functions*/
int main()
{

```

```

struct node *root = newNode(1);
root->left      = newNode(2);
root->right     = newNode(3);
root->left->left = newNode(4);
root->left->right = newNode(5);

printf("Level Order traversal of binary tree is \n");
reverseLevelOrder(root);

return 0;
}

```

[Run on IDE](#)

## Java

// A recursive java program to print reverse level order traversal

// A binary tree node

```

class Node {

    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right;
    }
}

class BinaryTree {

    static Node root;

    /* Function to print REVERSE level order traversal a tree*/
    void reverseLevelOrder(Node node) {
        int h = height(node);
        int i;
        for (i = h; i >= 1; i--)

            //THE ONLY LINE DIFFERENT FROM NORMAL LEVEL ORDER
            {
                printGivenLevel(node, i);
            }

        /* Print nodes at a given level */
        void printGivenLevel(Node node, int level) {
            if (node == null) {
                return;
            }
            if (level == 1) {
                System.out.print(node.data + " ");
            } else if (level > 1) {
                printGivenLevel(node.left, level - 1);
                printGivenLevel(node.right, level - 1);
            }
        }

        /* Compute the "height" of a tree -- the number of
        nodes along the longest path from the root node
        down to the farthest leaf node.*/
    }
}

```

```

int height(Node node) {
    if (node == null) {
        return 0;
    } else {
        /* compute the height of each subtree */
        int lheight = height(node.left);
        int rheight = height(node.right);

        /* use the larger one */
        if (lheight > rheight) {
            return (lheight + 1);
        } else {
            return (rheight + 1);
        }
    }
}

// Driver program to test above functions
public static void main(String args[]) {

    BinaryTreeNode tree = new BinaryTreeNode();

    // Let us create trees shown in above diagram
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

    System.out.println("Level Order traversal of binary tree is : ");
    tree.reverseLevelOrder(root);

}
}

// This code has been contributed by Mayank Jaiswal

```

[Run on IDE](#)

## Python

# A recursive Python program to print REVERSE level order traversal

# A binary tree node

```
class Node:
```

# Constructor to create a new node

```
def __init__(self, data):
```

```
    self.data = data
```

```
    self.left = None
```

```
    self.right = None
```

# Function to print reverse level order traversal

```
def reverseLevelOrder(root):
```

```
    h = height(root)
```

```
    for i in reversed(range(1, h+1)):
```

```
        printGivenLevel(root, i)
```

# Print nodes at a given level

```
def printGivenLevel(root, level):
```

```
    if root is None:
```

```
        return
    if level == 1 :
        print root.data,

    elif level > 1:
        printGivenLevel(root.left, level-1)
        printGivenLevel(root.right, level-1)

# Compute the height of a tree-- the number of
# nodes along the longest path from the root node
# down to the farthest leaf node
def height(node):
    if node is None:
        return 0
    else:

        # Compute the height of each subtree
        lheight = height(node.left)
        rheight = height(node.right)

        # Use the larger one
        if lheight > rheight :
            return lheight + 1
        else:
            return rheight + 1

# Driver program to test above function
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)

print "Level Order traversal of binary tree is"
reverseLevelOrder(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

[Run on IDE](#)

#### Output:

```
Level Order traversal of binary tree is
4 5 2 3 1
```

**Time Complexity:** The worst case time complexity of this method is  $O(n^2)$ . For a skewed tree, `printGivenLevel()` takes  $O(n)$  time where  $n$  is the number of nodes in the skewed tree. So time complexity of `printLevelOrder()` is  $O(n) + O(n-1) + O(n-2) + \dots + O(1)$  which is  $O(n^2)$ .

## METHOD 2 (Using Queue and Stack)

The method 2 of [normal level order traversal](#) can also be easily modified to print level order traversal in reverse order. The idea is to use a stack to get the reverse level order. If we do normal level order traversal and instead of printing a node, push the node to a stack and then print contents of stack, we get "5 4 3 2 1" for above

example tree, but output should be "4 5 2 3 1". So to get the correct sequence (left to right at every level), we process children of a node in reverse order, we first push the right subtree to stack, then left subtree.

## C++

```
// A C++ program to print REVERSE level order traversal using stack and queue
// This approach is adopted from following link
// http://tech-queries.blogspot.in/2008/12/level-order-tree-traversal-in-reverse.html
#include <iostream>
#include <stack>
#include <queue>
using namespace std;

/* A binary tree node has data, pointer to left and right children */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* Given a binary tree, print its nodes in reverse level order */
void reverseLevelOrder(struct node* root)
{
    stack<node*> S;
    queue<node*> Q;
    Q.push(root);

    // Do something like normal level order traversal order. Following are the
    // differences with normal level order traversal
    // 1) Instead of printing a node, we push the node to stack
    // 2) Right subtree is visited before left subtree
    while (Q.empty() == false)
    {
        /* Dequeue node and make it root */
        root = Q.front();
        Q.pop();
        S.push(root);

        /* Enqueue right child */
        if (root->right)
            Q.push(root->right); // NOTE: RIGHT CHILD IS ENQUEUED BEFORE LEFT

        /* Enqueue left child */
        if (root->left)
            Q.push(root->left);
    }

    // Now pop all items from stack one by one and print them
    while (S.empty() == false)
    {
        root = S.top();
        cout << root->data << " ";
        S.pop();
    }
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
node* newNode(int data)
{
    node* temp = new node;
```

```

temp->data = data;
temp->left = NULL;
temp->right = NULL;

return (temp);
}

/* Driver program to test above functions*/
int main()
{
    struct node *root = newNode(1);
    root->left      = newNode(2);
    root->right     = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);

    cout << "Level Order traversal of binary tree is \n";
    reverseLevelOrder(root);

    return 0;
}

```

[Run on IDE](#)

## Java

```

// A recursive java program to print reverse level order traversal
// using stack and queue

import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

// A binary tree node
class Node {
    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right;
    }
}

class BinaryTree {

    static Node root;

    /* Given a binary tree, print its nodes in reverse level order */
    void reverseLevelOrder(Node node) {
        Stack<Node> S = new Stack();
        Queue<Node> Q = new LinkedList();
        Q.add(node);

        // Do something like normal level order traversal order. Following are the
        // differences with normal level order traversal
        // 1) Instead of printing a node, we push the node to stack
        // 2) Right subtree is visited before left subtree
        while (Q.isEmpty() == false) {

```

```

        /* Dequeue node and make it root */
        node = Q.peek();
        Q.remove();
        S.push(node);

        /* Enqueue right child */
        if (node.right != null) {
            Q.add(node.right); // NOTE: RIGHT CHILD IS ENQUEUED BEFORE LEFT
        }
        /* Enqueue left child */
        if (node.left != null) {
            Q.add(node.left);
        }
    }

    // Now pop all items from stack one by one and print them
    while (S.empty() == false) {
        node = S.peek();
        System.out.print(node.data + " ");
        S.pop();
    }
}

// Driver program to test above functions
public static void main(String args[]) {

    BinaryTree tree = new BinaryTree();

    // Let us create trees shown in above diagram
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);
    tree.root.right.left = new Node(6);
    tree.root.right.right = new Node(7);

    System.out.println("Level Order traversal of binary tree is : ");
    tree.reverseLevelOrder(root);
}
}

// This code has been contributed by Mayank Jaiswal

```

Run on IDE

## Python

# Python program to print REVERSE level order traversal using  
# stack and queue

# A binary tree node

**class** Node:

    # Constructor to create a new node

```

def __init__(self, data):
    self.data = data
    self.left = None
    self.right = None

```



```
# Given a binary tree, print its nodes in reverse level order
def reverseLevelOrder(root):
    S = []
    Q = []
    Q.append(root)

    # Do something like normal level order traversal order.
    # Following are the differences with normal level order
    # traversal:
    # 1) Instead of printing a node, we push the node to stack
    # 2) Right subtree is visited before left subtree
    while(len(Q) > 0):

        # Dequeue node and make it root
        root = Q.pop(0)
        S.append(root)

        # Enqueue right child
        if (root.right):
            Q.append(root.right)

        # Enqueue left child
        if (root.left):
            Q.append(root.left)

    # Now pop all items from stack one by one and print them
    while(len(S) > 0):
        root = S.pop()
        print root.data,

# Driver program to test above function
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)

print "Level Order traversal of binary tree is"
reverseLevelOrder(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

[Run on IDE](#)

Output:

```
Level Order traversal of binary tree is
4 5 6 7 2 3 1
```

*Time Complexity:* O(n) where n is number of nodes in the binary tree.

Please write comments if you find any bug in the above programs/algorithms or other ways to solve the same problem.



41 Comments Category: Trees

## Related Posts:

- [Check if removing an edge can divide a Binary Tree in two halves](#)
- [Check sum of Covered and Uncovered nodes of Binary Tree](#)
- [Lowest Common Ancestor in a Binary Tree | Set 2 \(Using Parent Pointer\)](#)
- [Construct a Binary Search Tree from given postorder](#)
- [BFS vs DFS for Binary Tree](#)
- [Maximum difference between node and its ancestor in Binary Tree](#)
- [Inorder Non-threaded Binary Tree Traversal without Recursion or Stack](#)
- [Check if leaf traversal of two Binary Trees is same?](#)

([Login](#) to Rate and Mark)

**2.4** Average Difficulty : **2.4/5.0**  
Based on **5** vote(s)

☐

Add to TODO List

☐

Mark as DONE

Like Share 42 people like this.

Writing code in comment? Please use [code.geeksforgeeks.org](http://code.geeksforgeeks.org), generate link and share the link here.

@geeksforgeeks, [Some rights reserved](#)

[Contact Us!](#)

[About Us!](#)

[Advertise with us!](#)