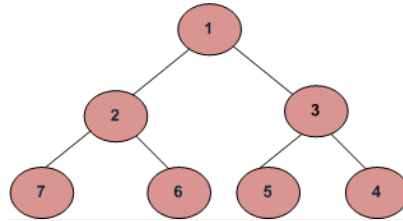


## Level order traversal in spiral form

Write a function to print spiral order traversal of a tree. For below tree, function should print 1, 2, 3, 4, 5, 6, 7.



### Method 1 (Recursive)

This problem can be seen as an extension of the [level order traversal](#) post.

To print the nodes in spiral order, nodes at different levels should be printed in alternating order. An additional Boolean variable *ltr* is used to change printing order of levels. If *ltr* is 1 then printGivenLevel() prints nodes from left to right else from right to left. Value of *ltr* is flipped in each iteration to change the order.

Function to print level order traversal of tree

```
printSpiral(tree)
bool ltr = 0;
for d = 1 to height(tree)
    printGivenLevel(tree, d, ltr);
    ltr ~= ltr /*flip ltr*/
```

Function to print all nodes at a given level

```
printGivenLevel(tree, level, ltr)
if tree is NULL then return;
if level is 1, then
    print(tree->data);
else if level greater than 1, then
    if(ltr)
        printGivenLevel(tree->left, level-1, ltr);
        printGivenLevel(tree->right, level-1, ltr);
    else
        printGivenLevel(tree->right, level-1, ltr);
        printGivenLevel(tree->left, level-1, ltr);
```

Following is C implementation of above algorithm.

## C

```
// C program for recursive level order traversal in spiral form
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

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

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

```

int height(struct node* node);
struct node* newNode(int data);

/* Function to print spiral traversal of a tree*/
void printSpiral(struct node* root)
{
    int h = height(root);
    int i;

    /*ltr -> Left to Right. If this variable is set,
    then the given level is traverseed from left to right. */
    bool ltr = false;
    for(i=1; i<=h; i++)
    {
        printGivenLevel(root, i, ltr);

        /*Revert ltr to traverse next level in opposite order*/
        ltr = !ltr;
    }
}

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

/* 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(7);
    root->left->right = newNode(6);
    root->right->left = newNode(5);
    root->right->right = newNode(4);
    printf("Spiral Order traversal of binary tree is \n");
    printSpiral(root);

    return 0;
}

```

## Java

```

// Java program for recursive level order traversal in spiral form

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
class Node
{
    int data;
    Node left, right;

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

class BinaryTree
{
    Node root;

    // Function to print the spiral traversal of tree
    void printSpiral(Node node)
    {
        int h = height(node);
        int i;

        /* ltr -> left to right. If this variable is set then the
           given label is transversed from left to right */
        boolean ltr = false;
        for (i = 1; i <= h; i++)
        {
            printGivenLevel(node, i, ltr);

            /*Revert ltr to traverse next level in opposite order*/
            ltr = !ltr;
        }
    }

    /* 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);
    }
}

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

/* Driver program to test the above functions */
public static void main(String[] args)
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(7);
    tree.root.left.right = new Node(6);
    tree.root.right.left = new Node(5);
    tree.root.right.right = new Node(4);
    System.out.println("Spiral order traversal of Binary Tree is ");
    tree.printSpiral(tree.root);
}

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

```

Output:

```

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

```

**Time Complexity:** Worst case time complexity of the above method is  $O(n^2)$ . Worst case occurs in case of skewed trees.

### Method 2 (Iterative)

We can print spiral order traversal in  **$O(n)$  time** and  $O(n)$  extra space. The idea is to use two stacks. We can use one stack for printing from left to right and other stack for printing from right to left. In every iteration, we have nodes of one level in one of the stacks. We print the nodes, and push nodes of next level in other stack.

### C++

```

// C++ implementation of a  $O(n)$  time method for spiral order traversal

```

```

#include <iostream>
#include <stack>
using namespace std;

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

void printSpiral(struct node *root)
{
    if (root == NULL) return;    // NULL check

    // Create two stacks to store alternate levels
    stack<struct node*> s1; // For levels to be printed from right to left
    stack<struct node*> s2; // For levels to be printed from left to right

    // Push first level to first stack 's1'
    s1.push(root);

    // Keep printing while any of the stacks has some nodes
    while (!s1.empty() || !s2.empty())
    {
        // Print nodes of current level from s1 and push nodes of
        // next level to s2
        while (!s1.empty())
        {
            struct node *temp = s1.top();
            s1.pop();
            cout << temp->data << " ";

            // Note that right is pushed before left
            if (temp->right)
                s2.push(temp->right);
            if (temp->left)
                s2.push(temp->left);
        }

        // Print nodes of current level from s2 and push nodes of
        // next level to s1
        while (!s2.empty())
        {
            struct node *temp = s2.top();
            s2.pop();
            cout << temp->data << " ";

            // Note that left is pushed before right
            if (temp->left)
                s1.push(temp->left);
            if (temp->right)
                s1.push(temp->right);
        }
    }
}

// A utility function to create a new node
struct node* newNode(int data)
{
    struct node* node = new struct node;
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

int main()
{
    struct node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);

```

```

// Create right child of root
root->left->left = newNode(7);
root->left->right = newNode(6);
root->right->left = newNode(5);
root->right->right = newNode(4);
cout << "Spiral Order traversal of binary tree is \n";
printSpiral(root);

return 0;
}

```

## Java

```

// Java implementation of an O(n) approach of level order
// traversal in spiral form

import java.util.*;

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

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

class BinaryTree
{
    static Node root;

    void printSpiral(Node node)
    {
        if (node == null)
            return;    // NULL check

        // Create two stacks to store alternate levels
        Stack<Node> s1 = new Stack<Node>(); // For levels to be printed from right to left
        Stack<Node> s2 = new Stack<Node>(); // For levels to be printed from left to right

        // Push first level to first stack 's1'
        s1.push(node);

        // Keep printing while any of the stacks has some nodes
        while (!s1.empty() || !s2.empty())
        {
            // Print nodes of current level from s1 and push nodes of
            // next level to s2
            while (!s1.empty())
            {
                Node temp = s1.peek();
                s1.pop();
                System.out.print(temp.data + " ");

                // Note that right is pushed before left
                if (temp.right != null)
                    s2.push(temp.right);

                if (temp.left != null)
                    s2.push(temp.left);
            }

            // Print nodes of current level from s2 and push nodes of
            // next level to s1
            while (!s2.empty())

```

```

        {
            Node temp = s2.peek();
            s2.pop();
            System.out.print(temp.data + " ");

            // Note that is left is pushed before right
            if (temp.left != null)
                s1.push(temp.left);
            if (temp.right != null)
                s1.push(temp.right);
        }
    }

    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(7);
        tree.root.left.right = new Node(6);
        tree.root.right.left = new Node(5);
        tree.root.right.right = new Node(4);
        System.out.println("Spiral Order traversal of Binary Tree is ");
        tree.printSpiral(root);
    }
}

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

```

Output:

```

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

```