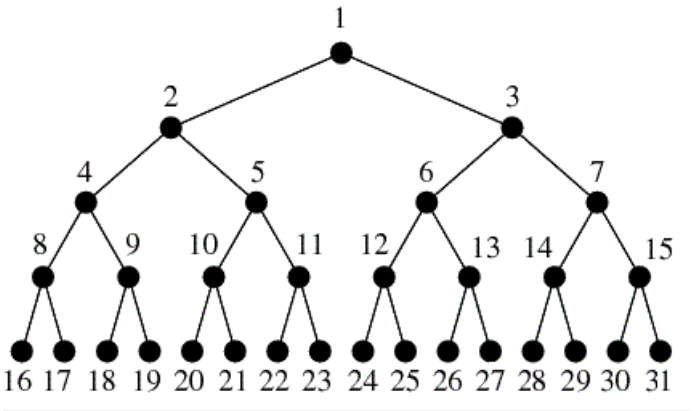
Perfect Binary Tree Specific Level Order Traversal

Given a Perfect Binary Tree like below:

(click on image to get a clear view)



Print the level order of nodes in following specific manner:

```
1 2 3 4 7 5 6 8 15 9 14 10 13 11 12 16 31 17 30 18 29 19 28 20 27 21 26  22 25 23 24
```

i.e. print nodes in level order but nodes should be from left and right side alternatively. Here 1st and 2nd levels are trivial.
While 3rd level: 4(left), 7(right), 5(left), 6(right) are printed.
While 4th level: 8(left), 15(right), 9(left), 14(right), .. are printed.
While 5th level: 16(left), 31(right), 17(left), 30(right), .. are printed.

We strongly recommend to minimize your browser and try this yourself first.

In standard Level Order Traversal, we enqueue root into a queue 1st, then we dequeue ONE node from queue, process (print) it, enqueue its children into queue. We keep doing this until queue is empty.

Approach 1:
We can do standard level order traversal here too but instead of printing nodes directly, we have to store nodes in current level in a temporary array or list 1st and then take nodes from alternate ends (left and right) and print nodes. Keep repeating this for all levels.
This approach takes more memory than standard traversal.

Approach 2:
The standard level order traversal idea will slightly change here. Instead of processing ONE node at a time, we will process TWO nodes at a time. And while pushing children into queue, the enqueue order will be: 1st node's left child, 2nd node's right child, 1st node's right child and 2nd node's left child.

C++

```
/* C++ program for special order traversal */
#include <iostream>
#include <queue>
using namespace std;

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

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
```

```
   given data and NULL left and right pointers. */
Node *newNode(int data)
{
    Node *node = new Node;
    node->data = data;
    node->right = node->left = NULL;
    return node;
}

/* Given a perfect binary tree, print its nodes in specific
   level order */
void printSpecificLevelOrder(Node *root)
{
    if (root == NULL)
        return;

    // Let us print root and next level first
    cout << root->data;

    // / Since it is perfect Binary Tree, right is not checked
    if (root->left != NULL)
      cout << " " << root->left->data << " " << root->right->data;

    // Do anything more if there are nodes at next level in
    // given perfect Binary Tree
    if (root->left->left == NULL)
        return;

    // Create a queue and enqueue left and right children of root
    queue <Node *> q;
    q.push(root->left);
    q.push(root->right);

    // We process two nodes at a time, so we need two variables
    // to store two front items of queue
    Node *first = NULL, *second = NULL;

    // traversal loop
    while (!q.empty())
    {
       // Pop two items from queue
       first = q.front();
       q.pop();
       second = q.front();
       q.pop();

       // Print children of first and second in reverse order
       cout << " " << first->left->data << " " << second->right->data;
       cout << " " << first->right->data << " " << second->left->data;

       // If first and second have grandchildren, enqueue them
       // in reverse order
       if (first->left->left != NULL)
       {
           q.push(first->left);
           q.push(second->right);
           q.push(first->right);
           q.push(second->left);
       }
    }
}

/* Driver program to test above functions*/
int main()
{
    //Perfect Binary Tree of Height 4
    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);

    root->left->left->left  = newNode(8);
    root->left->left->right  = newNode(9);
    root->left->right->left  = newNode(10);
    root->left->right->right  = newNode(11);
    root->right->left->left  = newNode(12);
    root->right->left->right  = newNode(13);
    root->right->right->left  = newNode(14);
    root->right->right->right  = newNode(15);

    root->left->left->left->left  = newNode(16);
    root->left->left->left->right  = newNode(17);
    root->left->left->right->left  = newNode(18);
    root->left->left->right->right  = newNode(19);
    root->left->right->left->left  = newNode(20);
    root->left->right->left->right  = newNode(21);
    root->left->right->right->left  = newNode(22);
    root->left->right->right->right  = newNode(23);
    root->right->left->left->left  = newNode(24);
    root->right->left->left->right  = newNode(25);
    root->right->left->right->left  = newNode(26);
    root->right->left->right->right  = newNode(27);
    root->right->right->left->left  = newNode(28);
    root->right->right->left->right  = newNode(29);
    root->right->right->right->left  = newNode(30);
    root->right->right->right->right  = newNode(31);

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

    return 0;
}
```

Java

```
// Java program for special level order traversal

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

/* Class containing left and right child of current
   node and key value*/
class Node
{
    int data;
    Node left, right;

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

class BinaryTree
{
    Node root;

    /* Given a perfect binary tree, print its nodes in specific
       level order */
    void printSpecificLevelOrder(Node node)
    {
        if (node == null)
            return;

        // Let us print root and next level first
        System.out.print(node.data);
```

```
        //  Since it is perfect Binary Tree, right is not checked
        if (node.left != null)
            System.out.print(" " + node.left.data + " " + node.right.data);

        // Do anything more if there are nodes at next level in
        // given perfect Binary Tree
        if (node.left.left == null)
            return;

        // Create a queue and enqueue left and right children of root
        Queue<Node> q = new LinkedList<Node>();
        q.add(node.left);
        q.add(node.right);

        // We process two nodes at a time, so we need two variables
        // to store two front items of queue
        Node first = null, second = null;

        // traversal loop
        while (!q.isEmpty())
        {
            // Pop two items from queue
            first = q.peek();
            q.remove();
            second = q.peek();
            q.remove();

            // Print children of first and second in reverse order
            System.out.print(" " + first.left.data + " " +second.right.data);
            System.out.print(" " + first.right.data + " " +second.left.data);

            // If first and second have grandchildren, enqueue them
            // in reverse order
            if (first.left.left != null)
            {
                q.add(first.left);
                q.add(second.right);
                q.add(first.right);
                q.add(second.left);
            }
        }
    }

    // Driver program to test for 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(4);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);

        tree.root.left.left.left = new Node(8);
        tree.root.left.left.right = new Node(9);
        tree.root.left.right.left = new Node(10);
        tree.root.left.right.right = new Node(11);
        tree.root.right.left.left = new Node(12);
        tree.root.right.left.right = new Node(13);
        tree.root.right.right.left = new Node(14);
        tree.root.right.right.right = new Node(15);

        tree.root.left.left.left.left = new Node(16);
        tree.root.left.left.left.right = new Node(17);
        tree.root.left.left.right.left = new Node(18);
        tree.root.left.left.right.right = new Node(19);
        tree.root.left.right.left.left = new Node(20);
        tree.root.left.right.left.right = new Node(21);
        tree.root.left.right.right.left = new Node(22);
        tree.root.left.right.right.right = new Node(23);
```

```
        tree.root.right.left.left.left = new Node(24);
        tree.root.right.left.left.right = new Node(25);
        tree.root.right.left.right.left = new Node(26);
        tree.root.right.left.right.right = new Node(27);
        tree.root.right.right.left.left = new Node(28);
        tree.root.right.right.left.right = new Node(29);
        tree.root.right.right.right.left = new Node(30);
        tree.root.right.right.right.right = new Node(31);

        System.out.println("Specific Level Order traversal of binary"
                                                              +"tree is ");
        tree.printSpecificLevelOrder(tree.root);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Python

```
# Python program for special order traversal

# A binary tree ndoe
class Node:
    # A constructor for making a new node
    def __init__(self, key):
        self.data = key
        self.left = None
        self.right = None

# Given a perfect binary tree print its node in
# specific order
def printSpecificLevelOrder(root):
    if root is None:
        return

    # Let us print root and next level first
    print root.data,

    # Since it is perfect Binary tree,
    # one of the node is needed to be checked
    if root.left is not None :
        print root.left.data,
        print root.right.data,

    # Do anythong more if there are nodes at next level
    # in given perfect Binary Tree
    if root.left.left is None:
        return

    # Create a queue and enqueue left and right
    # children of root
    q = []
    q.append(root.left)
    q.append(root.right)

    # We process two nodes at a time, so we need
    # two variables to stroe two front items of queue
    first = None
    second = None

    # Traversal loop
    while(len(q) > 0):

        # Pop two items from queue
        first = q.pop(0)
        second = q.pop(0)

        # Print children of first and second in reverse order
```

```
        print first.left.data,
        print second.right.data,
        print first.right.data,
        print second.left.data,

        # If first and second have grandchildren,
        # enqueue them in reverse order
        if first.left.left is not None:
            q.append(first.left)
            q.append(second.right)
            q.append(first.right)
            q.append(second.left)

# Driver program to test above function

# Perfect Binary Tree of Height 4
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)

root.left.left.left  = Node(8)
root.left.left.right  = Node(9)
root.left.right.left  = Node(10)
root.left.right.right  = Node(11)
root.right.left.left  = Node(12)
root.right.left.right  = Node(13)
root.right.right.left  = Node(14)
root.right.right.right  = Node(15)

root.left.left.left.left  = Node(16)
root.left.left.left.right  = Node(17)
root.left.left.right.left  = Node(18)
root.left.left.right.right  = Node(19)
root.left.right.left.left  = Node(20)
root.left.right.left.right  = Node(21)
root.left.right.right.left  = Node(22)
root.left.right.right.right  = Node(23)
root.right.left.left.left  = Node(24)
root.right.left.left.right  = Node(25)
root.right.left.right.left  = Node(26)
root.right.left.right.right  = Node(27)
root.right.right.left.left  = Node(28)
root.right.right.left.right  = Node(29)
root.right.right.right.left  = Node(30)
root.right.right.right.right  = Node(31)

print "Specific Level Order traversal of binary tree is"
printSpecificLevelOrder(root);

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

Output:

```
Specific Level Order traversal of binary tree is
1 2 3 4 7 5 6 8 15 9 14 10 13 11 12 16 31 17 30 18 29 19 28 20 27 21 26 22 25 23 24
```

Followup Questions:

1. The above code prints specific level order from TOP to BOTTOM. How will you do specific level order traversal from BOTTOM to TOP (Amazon Interview | Set 120 – Round 1 Last Problem)
2. What if tree is not perfect, but complete.
3. What if tree is neither perfect, nor complete. It can be any general binary tree.