Top-20 Training Program
(Data Structures)
Assignment-4

1) Tree Similarity: Write an efficient function that returns 1 if the given two binary trees are similar, and 0 otherwise. Two binary trees are similar if they are both empty, or both nonempty and have similar left and right subtrees. What are the time and space complexities of your solution?
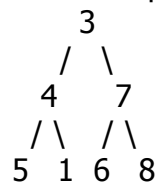Function prototype:
    boolean isSimilar(TreeNode t1, TreeNode t2)

2) Bottom-up level display: Write an efficient function that takes a binary tree as input and displays the elements of tree level by level, but from last level to first. What are the time and space complexities of your solution?
Function prototype:
    void BottomUpLevel(TreeNode t)

Input: 3
              /   \
            4   7
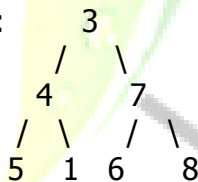   / \   / \
          5   1 6   8

Output: 5 1 6 8 4 7 3

3) Zig-Zag display: Write an efficient function to display given binary tree in zig-zag order. What are the time and space complexities of your solution?
Function prototype:
    void printTree(TreeNode root)

Input: 3
              /   \
            4   7
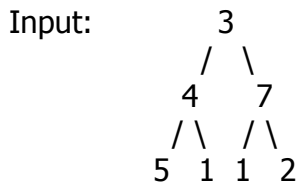   / \   / \
          5   1   6   8
Output: 3 7 4 5 1 6 8

4) Max Level Sum: Given a binary tree, write an efficient function that returns the maximum level sum. If tree is empty return 0. Assume that all values in binary tree are positive integers. What are the space and time complexities of your solution?

Function prototype:
        int MaxLevelSum(TreeNode t)
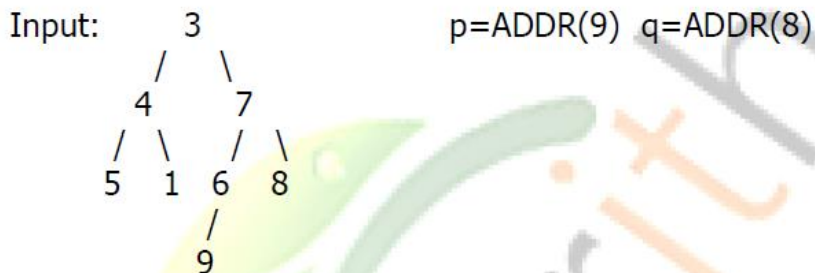Input: 3
              / \
             4   7
            / \ / \
           5 1 1 2

Return value: 11

5) LCA: Least Common Ancestor(LCA) in a tree is defined as the first node that comes common for both the given nodes while travelling towards the root. Write an efficient function "TreeNode FindLca(TreeNode t, TreeNode p, TreeNode q)" to find the least common ancestor of nodes p and q in a binary tree t. You are not allowed to modify the structure of tree node. What are the time and space complexities of your solution? Assume that p and q points to valid nodes in a given binary tree.

```
Input:      3              p=ADDR(9)  q=ADDR(8)
           / \
          4   7
         / \  / \
        5  1 6   8
             /
            9
Return value: ADDR(7)
```