Assignment Questions 22

Question-1:
Given a Binary Tree (Bt), convert it to a Doubly Linked List(DLL). The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be the same as in Inorder for the given Binary Tree. The first node of Inorder traversal (leftmost node in BT) must be the head node of the DLL.
 code:-

```
class Node
{
	int data;
	Node left, right;

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

class BinaryTree
{
	Node root;
	Node head;
	static Node prev = null;
	void BinaryTree2DoubleLinkedList(Node root)
	{
		if (root == null)
			return;
		BinaryTree2DoubleLinkedList(root.left);
		if (prev == null)
			head = root;
		else
		{
			root.left = prev;
			prev.right = root;
		}
		prev = root;
		BinaryTree2DoubleLinkedList(root.right);
	}
	void printList(Node node)
	{
		while (node != null)
		{
			System.out.print(node.data + " ");
			node = node.right;
		}
	}
	public static void main(String[] args)
	{
		// Let us create the tree as shown in above diagram
		BinaryTree tree = new BinaryTree();
		tree.root = new Node(10);
		tree.root.left = new Node(12);
		tree.root.right = new Node(15);
		tree.root.left.left = new Node(25);
		tree.root.left.right = new Node(30);
```

```
            tree.root.right.left = new Node(36);
            tree.BinaryTree2DoubleLinkedList(tree.root);
            tree.printList(tree.head);

      }
}
```

Question-2
A Given a binary tree, the task is to flip the binary tree towards the right direction that is clockwise. See the below examples to see the transformation.
In the flip operation, the leftmost node becomes the root of the flipped tree and its parent becomes its right child and the right sibling becomes its left child and the same should be done for all left most nodes recursively.

code:-

```
import java.util.Queue;
import java.util.LinkedList;
public class FlipTree {
      public static Node flipBinaryTree(Node root)
      {
            if (root == null)
                  return root;
            if (root.left == null && root.right ==null)
                  return root;
            Node flippedRoot=flipBinaryTree(root.left);
            root.left.left=root.right;
            root.left.right=root;
            root.left=root.right=null;
            return flippedRoot;
      }
      public static void printLevelOrder(Node root)
      {
            // Base Case
            if(root==null)
                  return ;
            Queue<Node> q=new LinkedList<>();
            q.add(root);
            while(true)
            {
                  int nodeCount = q.size();
                  if (nodeCount == 0)
                        break;
                  while (nodeCount > 0)
                  {
                        Node node = q.remove();
                        System.out.print(node.data+" ");
                        if (node.left != null)
                              q.add(node.left);
                        if (node.right != null)
                              q.add(node.right);
                        nodeCount--;
                  }
                  System.out.println();
            }
      }

      public static void main(String args[]) {
            Node root=new Node(1);
            root.left=new Node(2);
            root.right=new Node(1);
            root.right.left = new Node(4);
            root.right.right = new Node(5);
            System.out.println("Level order traversal of given tree");
            printLevelOrder(root);

            root = flipBinaryTree(root);
            System.out.println("Level order traversal of flipped tree");
            printLevelOrder(root);
      }
}
class Node
{
      int data;
      Node left, right;
      Node(int data)
      {
            this.data=data;
      }
};
```

Question-3:
Given a binary tree, print all its root-to-leaf paths without using recursion. For example, consider the following Binary Tree.
Input:

```
        6
     /    \
    3      5
  /   \     \
 2     5     4
     /   \
    7     4
```

Output:
There are 4 leaves, hence 4 root to leaf paths -

```
  6->3->2
  6->3->5->7
  6->3->5->4
  6->5>4
```

code:-

```
class Node
{
      int data;
      Node left, right;

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

```
class BinaryTree
{
      Node root;
      void printPaths(Node node)
      {
            int path[] = new int[1000];
            printPathsRecur(node, path, 0);
      }
      void printPathsRecur(Node node, int path[], int pathLen)
      {
            if (node == null)
                  return;
            path[pathLen] = node.data;
            pathLen++;
            if (node.left == null && node.right == null)
                  printArray(path, pathLen);
            else
            {
                  printPathsRecur(node.left, path, pathLen);
                  printPathsRecur(node.right, path, pathLen);
            }
      }
      void printArray(int ints[], int len)
      {
            int i;
            for (i = 0; i < len; i++)
            {
                  System.out.print(ints[i] + " ");
            }
            System.out.println("");
      }
      public static void main(String args[])
      {
            BinaryTree tree = new BinaryTree();
            tree.root = new Node(10);
            tree.root.left = new Node(8);
            tree.root.right = new Node(2);
            tree.root.left.left = new Node(3);
            tree.root.left.right = new Node(5);
            tree.root.right.left = new Node(2);
            tree.printPaths(tree.root);
      }
}
```

Question-4:
Given Preorder, Inorder and Postorder traversals of some tree. Write a program to check if they all are of the same tree.
Examples:

Input :

```
        Inorder -> 4 2 5 1 3
        Preorder -> 1 2 4 5 3
        Postorder -> 4 5 2 3 1
```

Output :

Yes
Explanation :

All of the above three traversals are of
the same tree

```
                           1
                         /   \
                        2     3
                      /   \
                     4     5
```

Input :

```
        Inorder -> 4 2 5 1 3
        Preorder -> 1 5 4 2 3
        Postorder -> 4 1 2 3 5
```

Output :

No

code:-

```
import java.util.*;
class GfG {
      static int preIndex = 0;
static class Node
{
      int data;
      Node left, right;
}
static Node newNode(int data)
{
      Node temp = new Node();
      temp.data = data;
      temp.left = null;
      temp.right = null;
      return temp;
}
static int search(int arr[], int strt, int end, int value)
{
      for (int i = strt; i <= end; i++)
      {
            if(arr[i] == value)
                  return i;
      }
      return -1;
}
static Node buildTree(int in[], int pre[], int inStrt, int inEnd)
{

      if(inStrt > inEnd)
            return null;
      Node tNode = newNode(pre[preIndex++]);
      if (inStrt == inEnd)
            return tNode;
      int inIndex = search(in, inStrt, inEnd, tNode.data);
      tNode.left = buildTree(in, pre, inStrt, inIndex-1);
      tNode.right = buildTree(in, pre, inIndex+1, inEnd);

      return tNode;
}
static int checkPostorder(Node node, int postOrder[], int index)
{
      if (node == null)
            return index;
      index = checkPostorder(node.left,postOrder,index);
      index = checkPostorder(node.right,postOrder,index);
      if (node.data == postOrder[index])
            index++;
      else
            return -1;

      return index;
}
public static void main(String[] args)
{
      int inOrder[] = {4, 2, 5, 1, 3};
      int preOrder[] = {1, 2, 4, 5, 3};
      int postOrder[] = {4, 5, 2, 3, 1};

      int len = inOrder.length;
      Node root = buildTree(inOrder, preOrder, 0, len - 1);
      int index = checkPostorder(root,postOrder,0);
      if (index == len)
            System.out.println("Yes");
      else
            System.out.println("No");

}
}
```