Q1)

// Java program for different tree traversals

/\* Class containing left and right child of current

node and key value\*/

class Node {

int key;

Node left, right;

public Node(int item)

{

key = item;

left = right = null;

}

}

class BinaryTree {

// Root of Binary Tree

Node root;

BinaryTree() { root = null; }

/\* Given a binary tree, print its nodes in inorder\*/

void printInorder(Node node)

{

if (node == null)

return;

/\* first recur on left child \*/

printInorder(node.left);

/\* then print the data of node \*/

System.out.print(node.key + " ");

/\* now recur on right child \*/

printInorder(node.right);

}

// Wrappers over above recursive functions

void printInorder() { printInorder(root); }

// Driver code

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);

// Function call

System.out.println(

"\nInorder traversal of binary tree is ");

tree.printInorder();

}

}

Q2)

// initialize indices

int indexA = n - 1; // index of the last element in A

int indexB = m - 1; // index of the last element in B

int indexMerged = n + m - 1; // index of the last element in the merged array

// merge A and B, starting from the end

while (indexB >= 0) {

if (indexA >= 0 && A[indexA] > B[indexB]) {

// move element from A to the end of the merged array

A[indexMerged] = A[indexA];

indexA--;

} else {

// move element from B to the end of the merged array

A[indexMerged] = B[indexB];

indexB--;

}

indexMerged--;

}