LABORATORY ASSIGNMENT 2

NAME-AKSHAT SRIVASTAV
REG.NO-19BCE0811

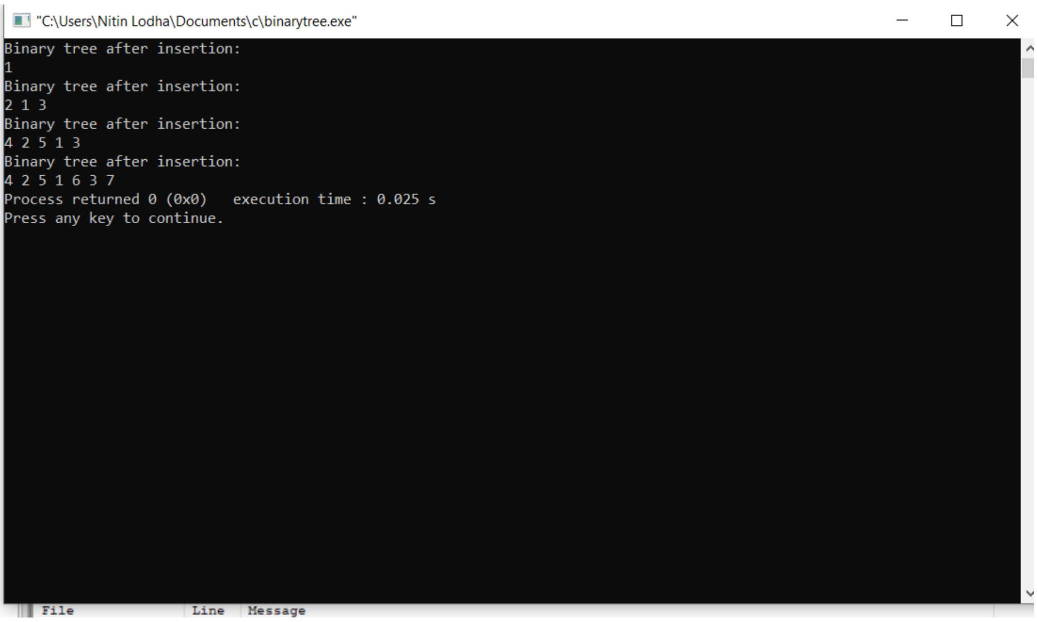
Write down a C/C++/C#/JAVA/Python/MATLAB/R Program to Implement Complete Binary Tree Using Linked List.

- **Program Input: A Series of Integer Numbers**
- **Program Output: Complete Binary Tree Representation of the Given Inputs**

General Instructions:

1) You have to submit your program in VTOP (submit it as a PDF file - don't upload
2) source files).
3) Last Date of Submission: 28th February 2020.
4) Please do include some COMMENTS into your code to make it easier to understand.
5) Your program is correct does not necessarily mean that you will get full marks.
6) Program written C/C++ language is preferable.
7) Please do not plagiarize someone else's work.

```c
//construction of a binary tree using linked list of a given value.say(1,2,3,4,5,6)
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

//Represent a node of binary tree
struct node{
    int data;
    struct node *left;
    struct node *right;
};

//Represent the root of binary tree
struct node *root = NULL;

//createNode() will create a new node
struct node* createNode(int data){
    //Create a new node
    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    //Assign data to newNode, set left and right child to NULL
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

//Represent a queue
struct queue
{
    int front, rear, size;
    struct node* *arr;
};

//createQueue() will create a queue
struct queue* createQueue()
{
    struct queue* newQueue = (struct queue*) malloc(sizeof( struct queue ));

    newQueue->front = -1;
    newQueue->rear = 0;
    newQueue->size = 0;

    newQueue->arr = (struct node**) malloc(100 * sizeof( struct node* ));

    return newQueue;
}

//Adds a node to queue
void enqueue(struct queue* queue, struct node *temp){
    queue->arr[queue->rear++] = temp;
    queue->size++;
}

//Deletes a node from queue
struct node *dequeue(struct queue* queue){
    queue->size--;
    return queue->arr[++queue->front];
}


//insertNode() will add new node to the binary tree
void insertNode(int data) {
    //Create a new node
    struct node *newNode = createNode(data);
    //Check whether tree is empty
    if(root == NULL){
        root = newNode;
        return;
    }
    else {
        //Queue will be used to keep track of nodes of tree level-wise
        struct queue* queue = createQueue();
        //Add root to the queue
        enqueue(queue, root);

        while(true) {
            struct node *node = dequeue(queue);
            //If node has both left and right child, add both the child to queue
            if(node->left != NULL && node->right != NULL) {
                enqueue(queue, node->left);
            enqueue(queue, node->right);
            }
            else {
                //If node has no left child, make newNode as left child
                if(node->left == NULL) {
                    node->left = newNode;
                enqueue(queue, node->left);
                }
                //If node has left child but no right child, make newNode as right child
                else {
                    node->right = newNode;
                enqueue(queue, node->right);
                }
                break;
            }
        }
    }
}

//inorder() will perform inorder traversal on binary search tree
void inorderTraversal(struct node *node) {
    //Check whether tree is empty
    if(root == NULL){
        printf("Tree is empty\n");
        return;
    }
    else {

        if(node->left != NULL)
            inorderTraversal(node->left);
        printf("%d ", node->data);
        if(node->right != NULL)
            inorderTraversal(node->right);

        }
    }

int main(){

    //Add nodes to the binary tree
    insertNode(1);
    //1 will become root node of the tree
    printf("Binary tree after insertion: \n");
    //Binary after inserting nodes
    inorderTraversal(root);

    insertNode(2);
    insertNode(3);
    //2 will become left child and 3 will become right child of root node 1
    printf("\nBinary tree after insertion: \n");
    //Binary after inserting nodes
    inorderTraversal(root);

    insertNode(4);
    insertNode(5);
    //4 will become left child and 5 will become right child of node 2
    printf("\nBinary tree after insertion: \n");
    //Binary after inserting nodes
    inorderTraversal(root);

    insertNode(6);
    insertNode(7);
    //6 will become left child and 7 will become right child of node 3
    printf("\nBinary tree after insertion: \n");
    //Binary after inserting nodes
    inorderTraversal(root);

    return 0;
}
```



```
Binary tree after insertion:
1
Binary tree after insertion:
2 1 3
Binary tree after insertion:
4 2 5 1 3
Binary tree after insertion:
4 2 5 1 6 3 7
Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.
```