**Algorithm Analysis and Data Structures**

**CS 5343.502(Spring 2020)**

**Assignment 2**

**Divya Birla**

**2021514344**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**QUESTION:**

 Implement insert and delete in Binary Search Trees.

    a. Insert at least 15 nodes in a Binary Search Tree. You may repeatedly call your insert function from the main. The numbers you use to insert should not be sorted.

    b. After inserting all the numbers, run in-order traverse to print the values in the tree nodes. Also draw your tree by hand to see which nodes are leaves and non-leaves.

    c. Now delete the value in the root node (it must be the value you inserted first.  Run the inorder traversal again.

    d. delete a node, use predecessor to replace it.  Use a node such that the predecessor is not a leaf node.  ( you can see that from the hand drawn tree ).  Run the inorder traversal again.

Here are the values which you can insert (in the order given here).  50, 40, 80, 20, 45, 60, 100, 70, 65, 42, 44, 30, 25, 35, 33

   a. delete 50 and run inorder traversal.

   b. delete 40, run inorder traversal

   c. delete 65, run inorder traversal.

   d. delete 35, run inorder traversal

2. We covered the left-straight-line case for AVL balance.  Write the other three cases - right-straight-line, left-zigzag and right-zigzag.  Submit this as a .txt file.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SOURCE CODE:**

**Question 1**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Course: CS 5343.502 – Spring 2020

\* Assignment <2>

\* Name: Divya Birla

\* E-mail: dxb190021@utdallas.edu

\* Submitted: <02/23/20>

The below program performs the following:

(a)Insertion of 15 Elements for Binary Search Tree

(b)In-order traversal of the BST (left child=>parent=>right child)

(c)Deletion at root node

(d)Replacement of deleted node by predecessor which is not a leaf node

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <cstddef>

#include <stdlib.h>

#include <stddef.h>

using namespace std;

// Definition of Nodes used in the linked list

class Node {

public:

int element;

Node\* left;

Node\* right;

};

//Creation of new node

Node \*newNode(int ele)

{

Node \*newnode = new Node;

newnode->element = ele;

newnode->left = newnode->right = NULL;

return newnode;

}

//Insertion of nodes in a Binary Search Tree

void BSTInsert(Node \*\* root, int ele)

{

if (\*root == NULL)

\*root = newNode(ele);

else if ((\*root)->element <= ele)

BSTInsert(&((\*root)->right), ele);

else if ((\*root)->element > ele)

BSTInsert(&((\*root)->left), ele);

}

Node \* BST(int \* a, int size)

{

Node \* root = NULL;

for (int i = 0; i < size; i++)

BSTInsert(&root, a[i]);

return root;

}

//Inorder BST traversal

void InorderTrav(Node\* refnode){

if (refnode == NULL)

return;

InorderTrav(refnode->left);

cout << refnode->element << " ";

InorderTrav(refnode->right);

}

//Find predecessor of current node

Node\* FindPred(Node\* pred) {

if (pred == NULL)

return NULL;

// search for the maximum value in left subtree for predecessor

while (pred->right != NULL)

{

pred = pred->right;

}

cout << "\n New predecessor = " << pred->element << endl;

return pred;

}

Node\* DeleteTreeNode(Node\* refnode, int ele) {

if (refnode == NULL) return refnode;

else if (ele < refnode->element)

refnode->left = DeleteTreeNode(refnode->left, ele);

else if (ele > refnode->element)

refnode->right = DeleteTreeNode(refnode->right, ele);

else {

//If refnode node to be deleted has no children

if (refnode->right == NULL && refnode->left == NULL)

{

delete refnode;

refnode = NULL;

}

//If refnode node to be deleted has only one child

else if (refnode->right == NULL)

{

Node\* temp = refnode;

refnode = refnode->left;

delete temp;

}

else if (refnode->left == NULL)

{

Node\* temp = refnode;

refnode = refnode->right;

delete temp;

}

//If refnode node to be deleted has two children

else

{

Node\* temp = FindPred(refnode->left);

refnode->element = temp->element;

refnode->left = DeleteTreeNode(refnode->left, temp->element);

}

}

return refnode;

}

int main()

{

Node\* parent = NULL;

int a[] = { 50, 40, 80, 20, 45, 60, 100, 70, 65, 42, 44, 30, 25, 35, 33 };

int n = sizeof(a) / sizeof(int);

//creating Binary Search Tree from array

parent = BST(a, n);

cout << "\n--------PROGRAM FOR INORDER TRAVERSAL IN BST AND DELETION --------\n";

cout << "\n\nInorder traversal of the tree is as follows:" << endl;

cout << " Parent node now = " << parent->element << endl;

cout << "\n\n";

InorderTrav(parent);

cout << "\n\n Deletion of root node 50 leads to the below tree" << endl;

parent=DeleteTreeNode(parent, 50);

cout << " Parent node = " << parent->element << endl;

cout << "\n\n";

InorderTrav(parent);

cout << "\n\nDeletion of node 40 leads to the below tree" << endl;

parent=DeleteTreeNode(parent, 40);

cout << " Parent node = " << parent->element << endl;

cout << "\n\n";

InorderTrav(parent);

cout << "\n\nDeletion of node 65 leads to the below tree" << endl;

parent = DeleteTreeNode(parent, 65);

cout << "\n Parent node = " << parent->element << endl;

cout << "\n\n";

InorderTrav(parent);

cout << "\n\nDeletion of node 35 leads to the below tree" << endl;

parent = DeleteTreeNode(parent, 35);

cout << " Parent node = " << parent->element << endl;

cout << "\n\n";

InorderTrav(parent);

cout << "\n\n";

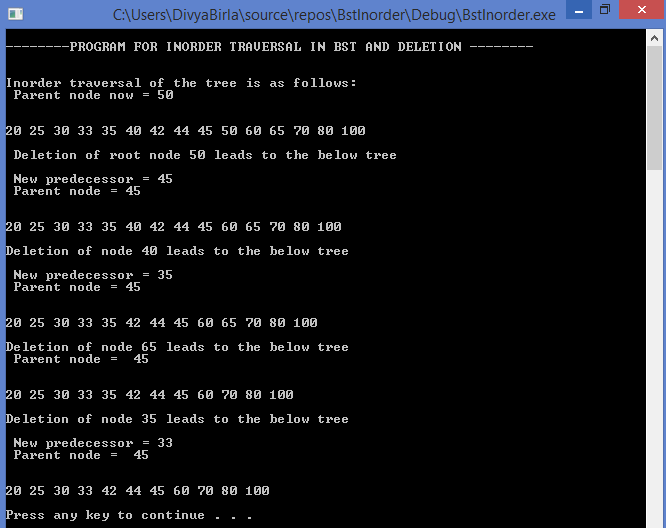
system("pause");

return 0;

}

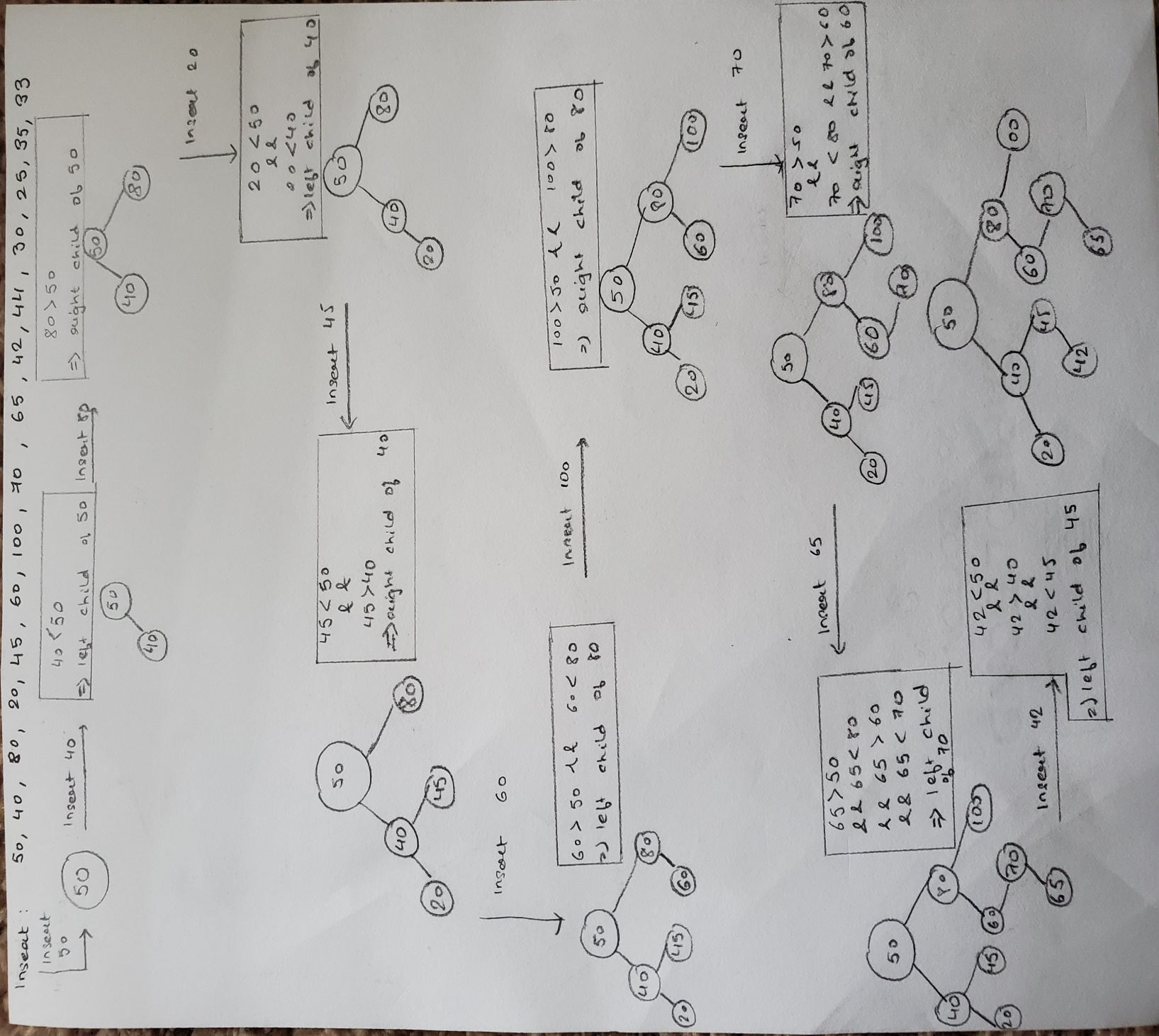
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

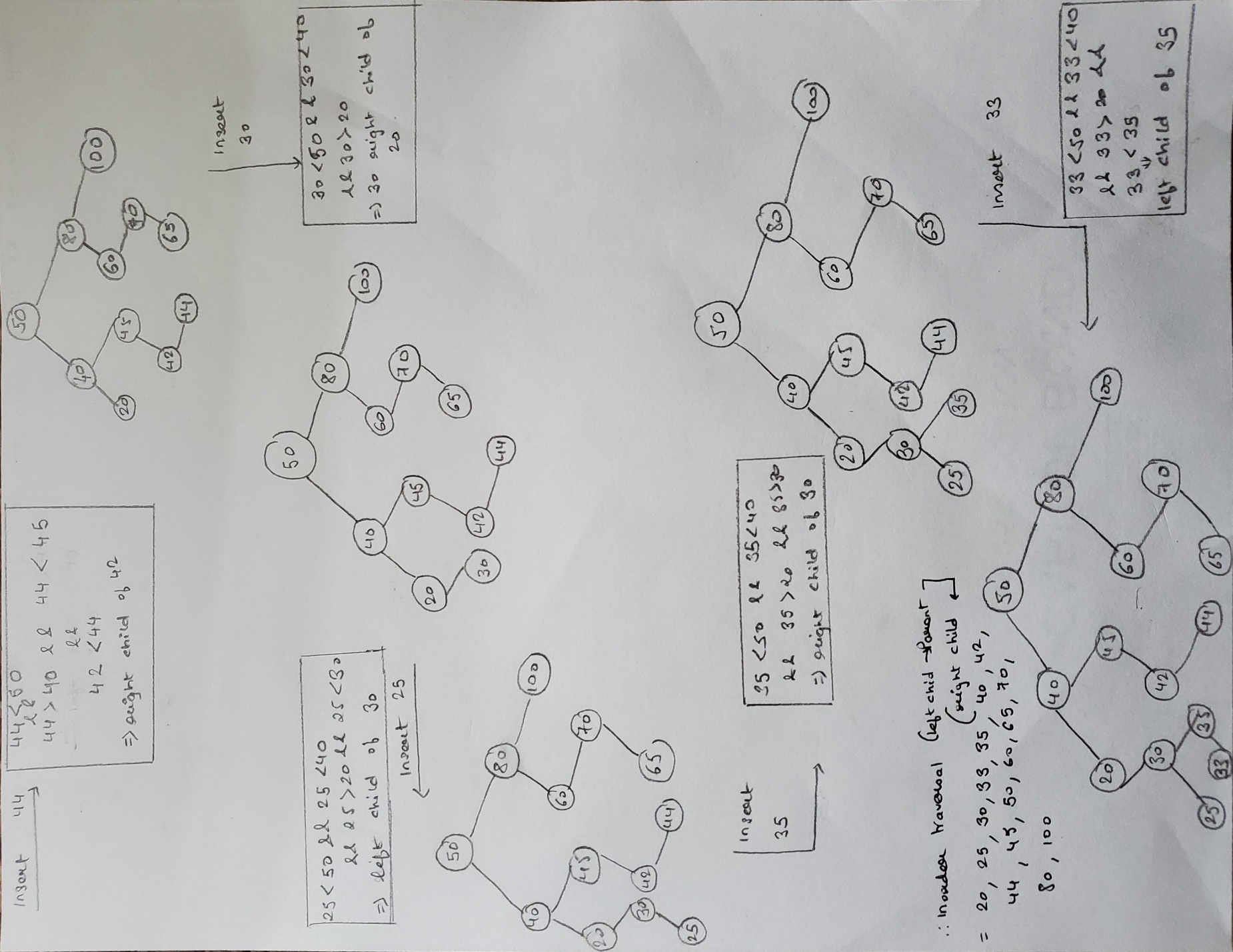
**OUTPUT:**



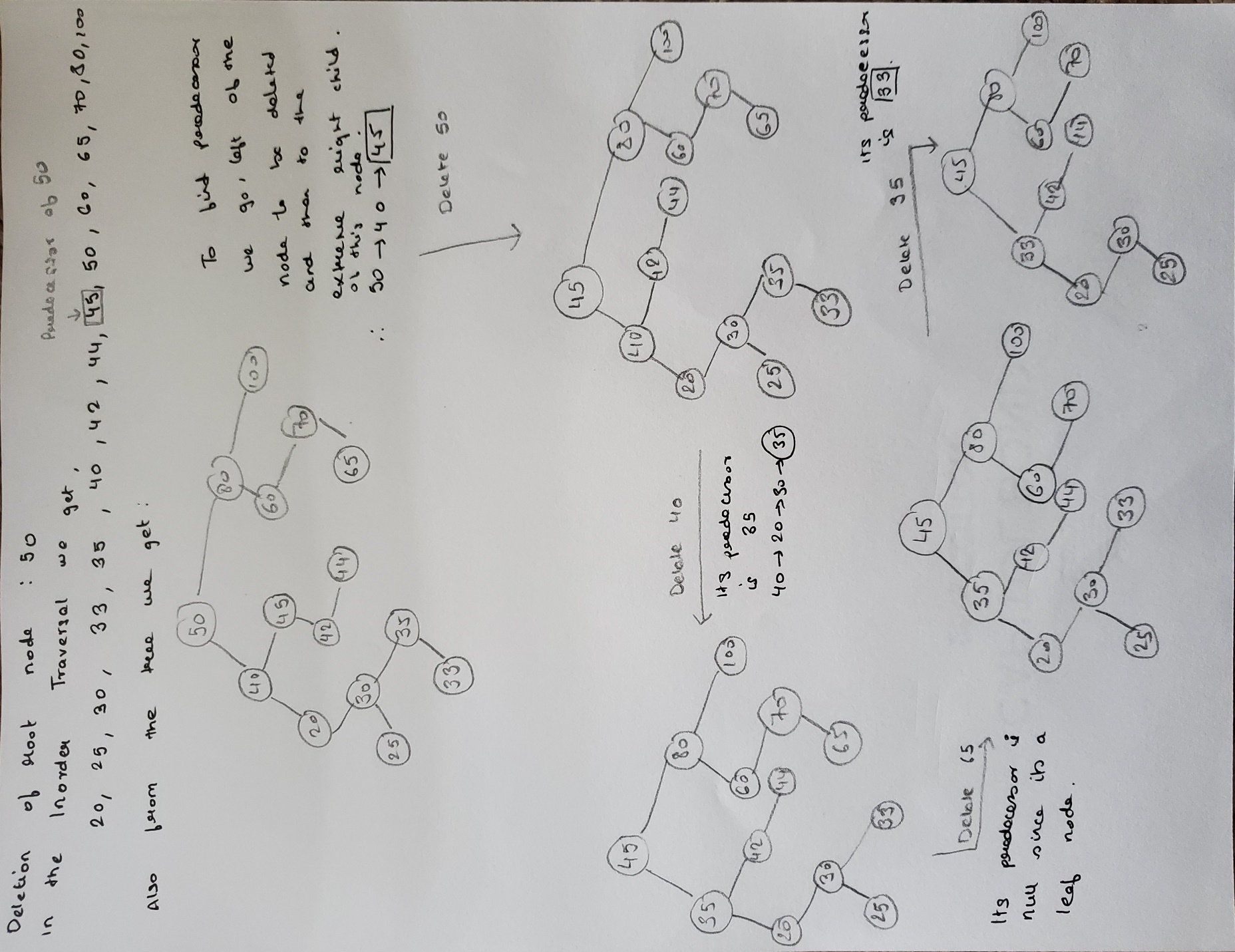
**TREE DRAWINGS:**

1.BST insertion and In-order Traversal





2.Deletion of 50 ,40,65 and 35



**PSEUDOCODE;**

**Question 2**

BalanceRightStLine(n1,n2,n3){

tmp= n1.parent;

n1.parent= n2;

n1.right=n2.left;

n2.left.parent=n1;

n2.parent=tmp;

n2.left=n1;

tmp2.right=n2;

}

BalanceLeftZigzag{

n3.parent=n1;

n2.right=n3.left;

n3.left.parent=n2;

n3.left=n2;

n2.parent=n3;

n1.left=n3;

BalanceLeftStLine(n1,n3,n2)

}

BalanceRightZigzag{

n3.parent=n1;

n2.left=n3.right;

n3.right.parent=n2;

n3.right=n2;

n2.parent=n3;

n1.right=n3;

BalanceRightStLine(n1,n3,n2)

}