**Experiment No. 6a**

**Title :** Implementation of Insertion in Binary Search Tree

**Problem Statement :** Implementing insert operation in binary search tree

Insert()

Inorder()

**Algorithm :**

**S1 :** Start

**S2 :** Create a class with data and left, right pointers to object from class created as private variables.

**S3 :** Create a constructor and which initializes data to 0 and pointers to NULL.

**S4 :** Define a parameterized constructor which when passed a parameter assigns pointers to NULL and value to data.

**S5 :** In insert function if there is no value in tree add the node by constructor else check if value is greater or lesser than the existing node and decide if node is right or left.

**S6 :** In inorder function print left node then parent node then right node which gives ascending order of tree.

**S7 :** Stop

**Code :**

#include<iostream>;

using namespace std;

class BST

{

int data;

BST \*left, \*right;

public:

// Default constructor.

BST();

// Parameterized constructor.

BST(int);

// Insert function.

BST\* Insert(BST \*, int);

// Inorder traversal.

void Inorder(BST \*);

};

// Default Constructor definition.

BST :: BST() : data(0), left(NULL), right(NULL){}

// Parameterized Constructor definition.

BST :: BST(int value)

{

data = value;

left = right = NULL;

}

// Insert function definition.

BST\* BST :: Insert(BST \*root, int value)

{

if(!root)

{

// Insert the first node, if root is NULL.

return new BST(value);

}

// Insert data.

if(value > root->data)

{

// Insert right node data, if the &#39;value&#39;

// to be inserted is greater than &#39;root&#39; node data.

// Process right nodes.

root->right = Insert(root->right, value);

}

else

{

// Insert left node data, if the &#39;value&#39;

// to be inserted is greater than &#39;root&#39; node data.

// Process left nodes.

root->left = Insert(root->left, value);

}

// Return &#39;root&#39; node, after insertion.

return root;

}

// Inorder traversal function.

// This gives data in sorted order.

void BST :: Inorder(BST \*root)

{

if(!root)

{

return;

}

Inorder(root->left);

cout << root->data << endl;

Inorder(root->right);

}

// Driver code

int main()

{

BST b, \*root = NULL;

cout<<"Binary tree :"<<endl;

root = b.Insert(root, 50);

b.Insert(root, 83);

b.Insert(root, 105);

b.Insert(root, 110);

b.Insert(root, 99);

b.Insert(root, 104);

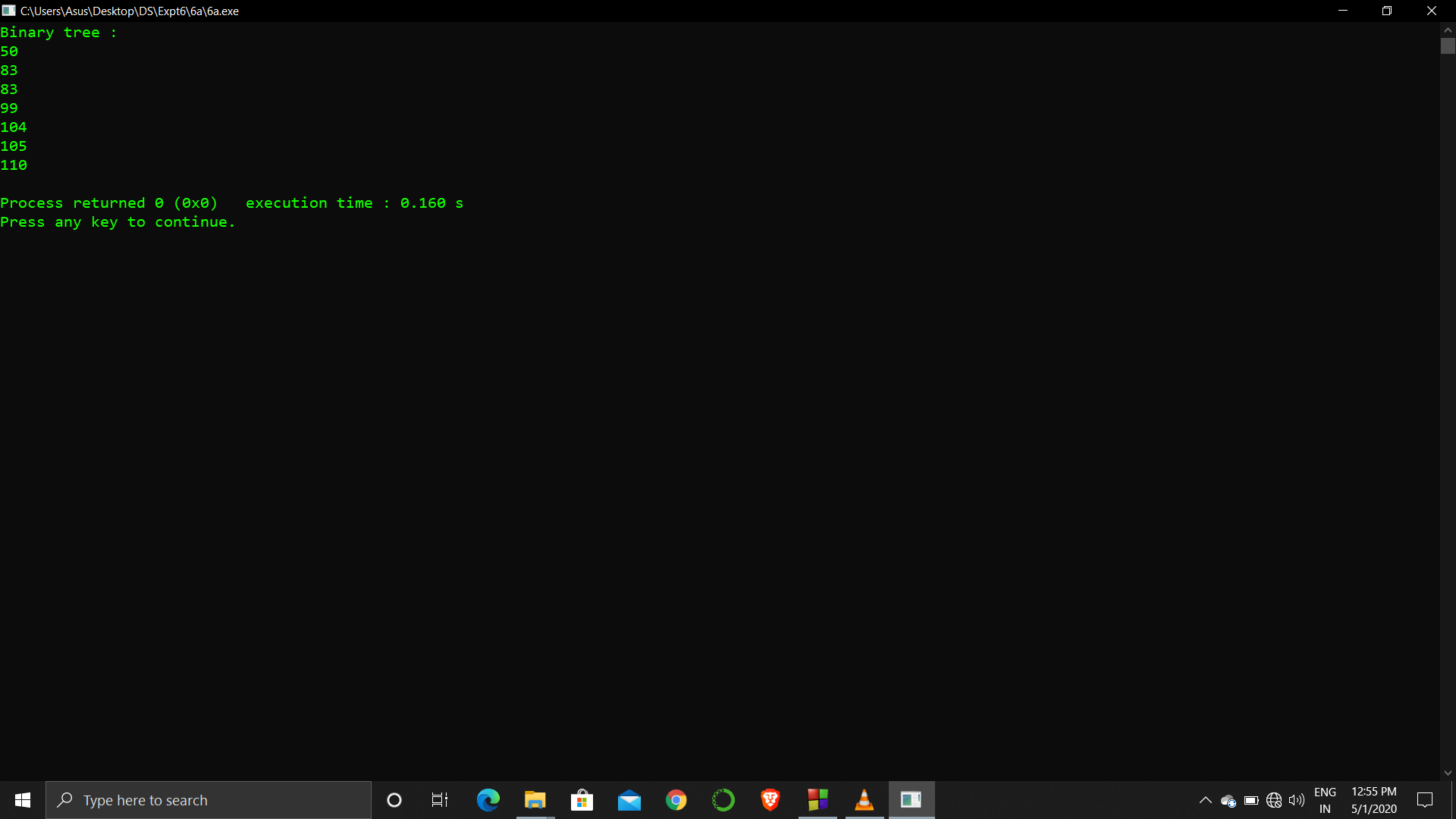
b.Insert(root, 83);

b.Inorder(root);

return 0;

}

**Output :**

****

**Analysis :**

In inorder function if the values in the tree are similar both are analyzed twice while printing which is a repetitive work

The time complexity increases as the nodes increase since to add a node we have to traverse until the leaf node on any one side.