**Experiment No. 6a**

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

**Problem Statement :** Write a C++ code to perform following operations on binary search tree

Insert(), Inorder()

**Algorithm:**

**Step 1:** Start

**Step 2:** Declare a class having variable data and pointers left and right.

**Step 3:** Create contractor to initialize data to 0 and pointers pointing to NULL.

**Step 4:** Create a insert() function which inserts value to BST

* Check if root node exist if no then put the value to root node
* If root node exist then depending on the data is greater than or less than that of root node put it to right or left respectively

**Step 5:** Create inorder() function which starts traversing from the left child and print till the right most child so that the output shows ascending order of the data entered.

**Step 6:** Create a instance of the class BST and insert elements into it using insert() operation on that object.

**Step 7:** Using inorder() function on root of BST object print the tree in ascending order.

**Step 8:** Stop

**Program:**

#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, 100);//insert root element

//insert elements

b.Insert(root, 31);

b.Insert(root, 01);

b.Insert(root, 14);

b.Insert(root, 22);

b.Insert(root, 63);

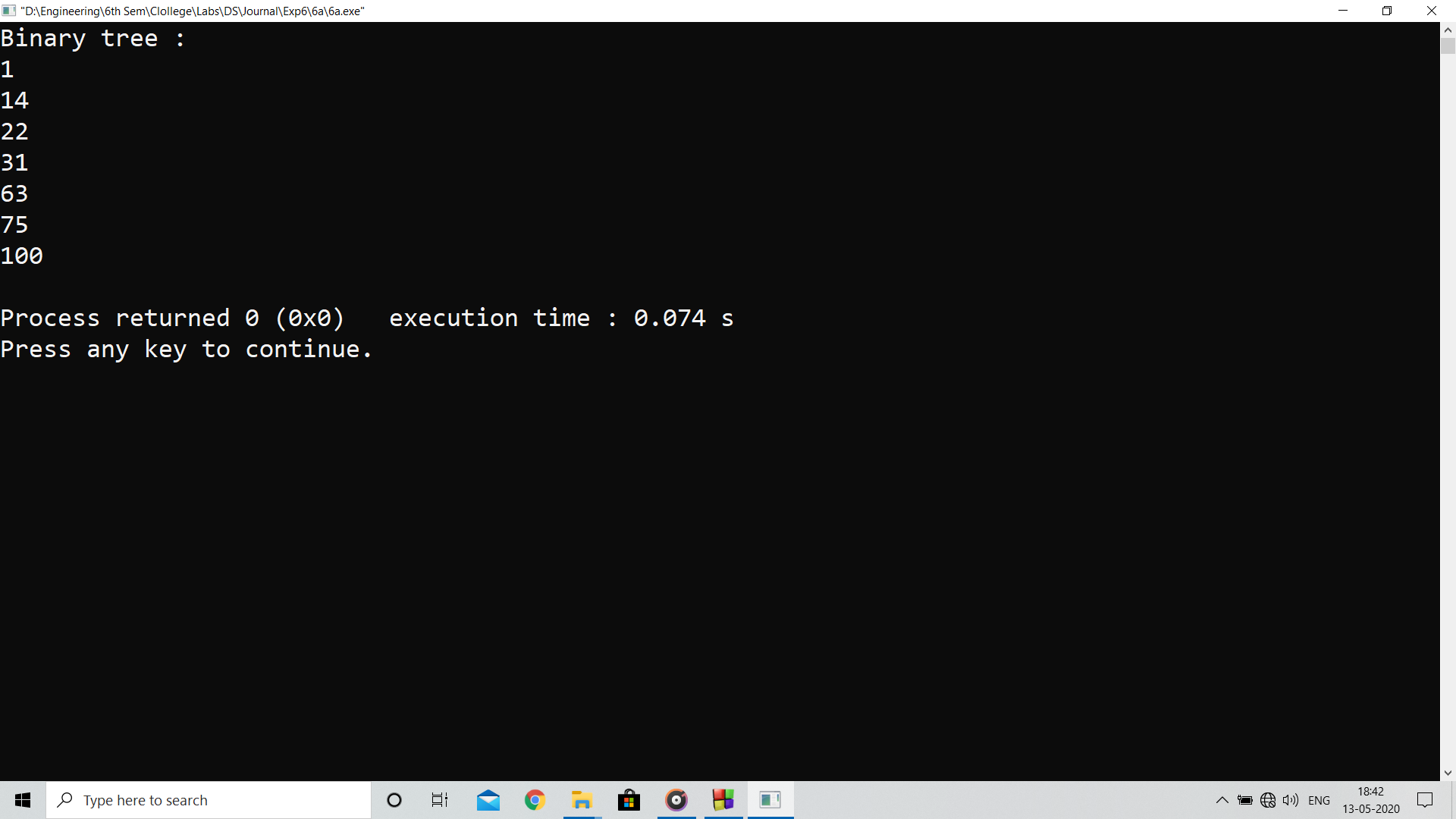
b.Insert(root, 75);

b.Inorder(root);

return 0;

}

**Output:**

****

**Analysis :**

Program implants Binary Tree which improves over searching by reducing the time by factor of half than normal search.

**Limitations :**

Inserting new element after many number of node is a costly process as insert() function travels through complete binary tree