Experiment No:6a

Title: Implementation of Insertion in Binary Search Tree in C.

Problem Statement: Implementation of Insertion in Binary Search Tree in C.

Algorithm:

1. The idea is to note that new keys are always inserted at the leaf node.
2. Take a temporary pointer named x and start from the root node to traverse the tree downwards to find the correct leaf node at which the key is to be inserted.
3. Also, keep track of the trailing pointer y to find the parent of the new node.
4. After getting the final values of x and y, assign x as the left child of y if x->key is less than y->key. Else, if x->key is greater than y->key, assign x as the right child of y.(If the tree is NULL then the new node is made the root node.)

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);

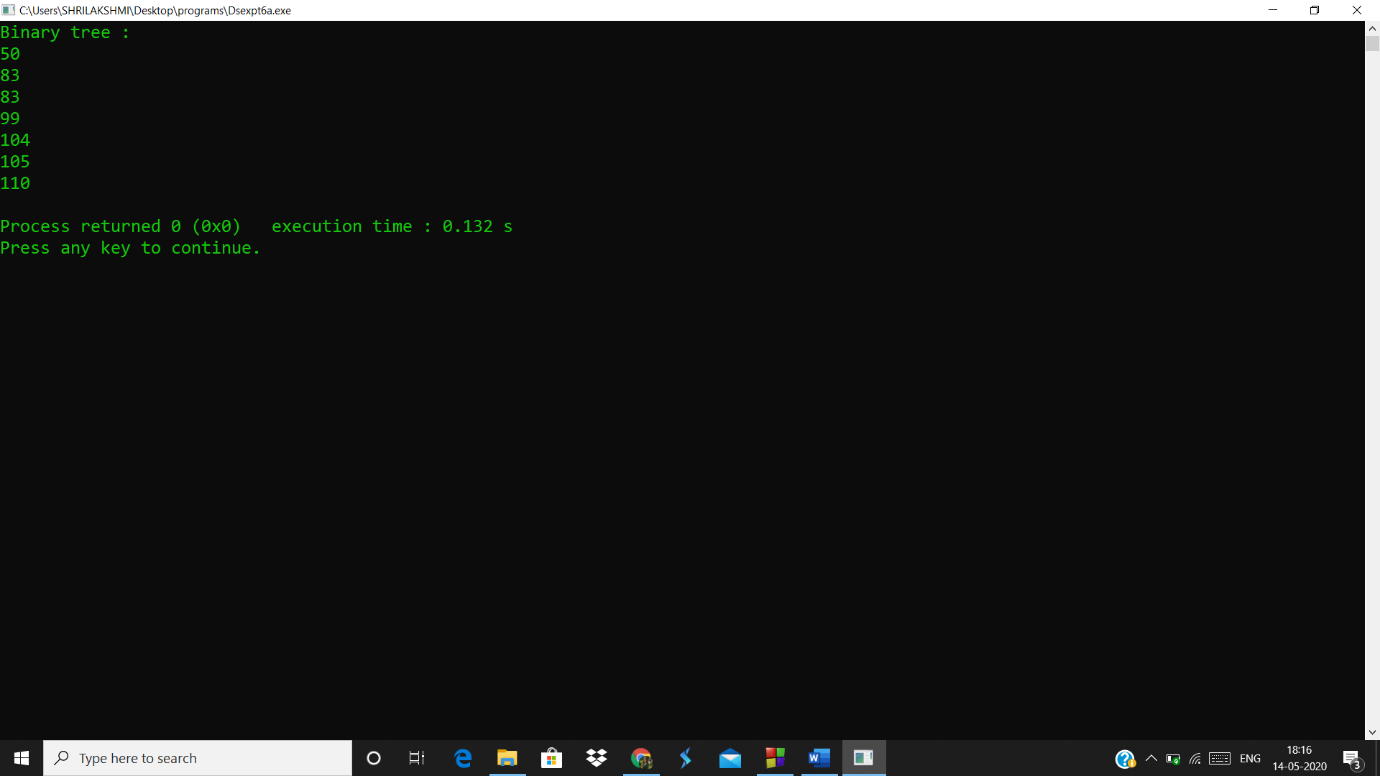
b.Insert(root, 83);

b.Inorder(root);

return 0;

}

Output:



Analysis(Limitations):

* 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.