**Experiment Number: 6a**

**Title:** Implementation of Insertion in Binary Search Tree Recursively Using C++.

**Problem Statement:** Write a C++ program to demonstrate the insertion in BST using recursion.

**Algorithm:**

1. Create a class called BST.
2. Inside class declare a variable called data to hold the node value.
3. Declare the two pointers left and right that point to the left and right child respectively.
4. Declare the default constructor which initializes the data variable to 0, left pointer and right pointer to NULL.
5. Declare the parameterized constructor which takes root as argument and inserts the root node and initializes the left and right pointers to NULL.
6. Declare and define the function called insert which takes the node pointer and value as argument and returns the modified node pointer.
7. This function first checks the value of the pointer.

If NULL, then it inserts the first node by calling the function parameterized constructor and stops.

If Not NULL, then compare the value of the variable “value” with the value at the pointer.

If value is greater than the value at pointer then call the same function but with the arguments right pointer and value recursively so that the value is inserted at the right side.

If value is not greater than the value at pointer then call the same function but with the arguments left pointer and value recursively so that the value is inserted at the left side.

1. Return the pointer.
2. Declare and define the function called inorder which takes root pointer as argument.
3. This function first checks the value of the root pointer if NULL then it does nothing and stops. If Not NULL then it calls the same function with argument left pointer recursively, prints the root node value and then it calls the same function with argument right pointer recursively.
4. In main function call the insert function as many times as you wish.
5. Call the inorder function to display the BST in order.

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 'value'

// to be inserted is greater than 'root' node data.

// Process right nodes.

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

}

else

{

// Insert left node data, if the 'value'

// to be inserted is greater than 'root' node data.

// Process left nodes.

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

}

// Return 'root' 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, 70);

b.Insert(root, 60);

b.Insert(root, 90);

b.Insert(root, 20);

b.Insert(root, 70);

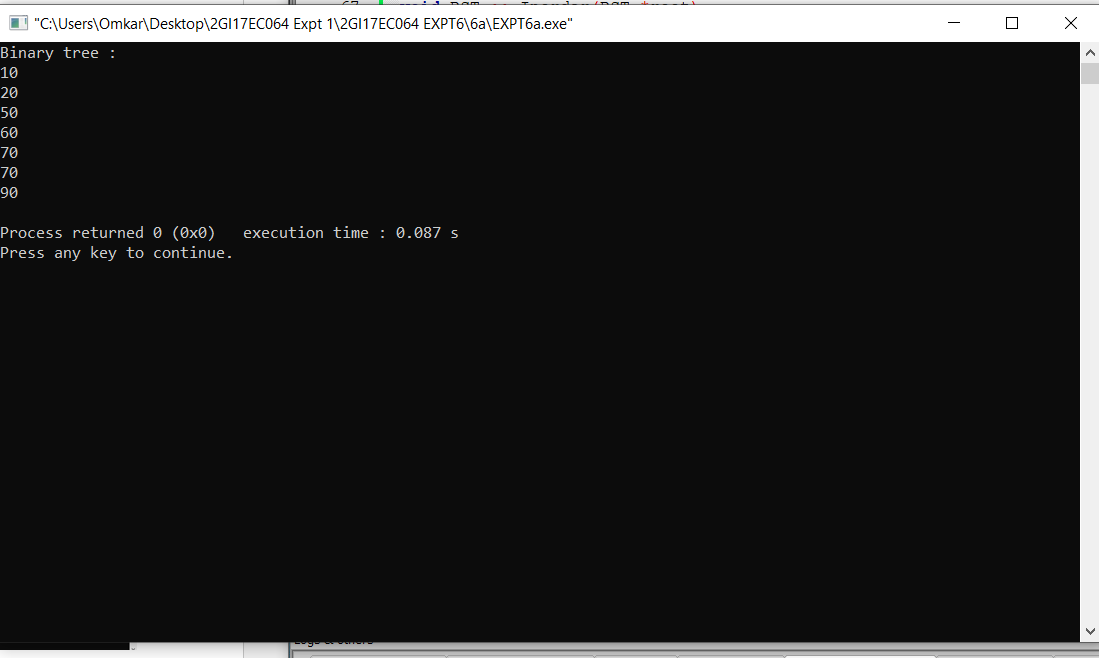
b.Insert(root, 10);

b.Inorder(root);

return 0;

}

**Output:**



**ANALYSIS (LIMITATIONS):**

The disadvantage is that it takes O(logn) time to modify the list (balanced trees take longer - this is for the baseline) and to retrieve elements with a known location. These can be done in constant time in some other data structures. Heaps are another common type of binary tree.