**Experiment No. 6c**

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

**Problem Statement :** Write a C++ code to implement search operation in binary search tree which includes following functions

Insert()

Search()

**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 a function of class Search() in which keep traversing through the tree with maintaining the depth value as

* If the value to be searched is more than node then traverse to right sub tree and vice versa
* This should continue till the Null address
* And if element found return the depth value

**Step 6:** Instantiate a object of the tree and then put values

**Step 7:** Input the value to be searched and call Search() function with that and display the results

**Step 8:** Stop

**Program :**

#include<iostream>

using namespace std;

struct node {

int d;

node \*left;

node \*right;

};

node\* CreateNode(int d)

{

node \*newnode = new node;

newnode->d = d;

newnode->left = NULL;

newnode->right = NULL;

return newnode;

}

node\* InsertIntoTree(node\* root, int d)

{

node \*temp = CreateNode(d);

node \*t = new node;

t = root;

if(root == NULL)

root = temp;

else

{

while(t != NULL)

{

if(t->d < d)

{

if(t->right == NULL)

{

t->right = temp;

break;

}

t = t->right;

}

else if(t->d > d)

{

if(t->left == NULL)

{

t->left = temp;

break;

}

t = t->left;

}

}

}

return root;

}

void Search(node \*root, int d)

{

int depth = 0;

node \*temp = new node;

temp = root;

while(temp != NULL)

{

depth++;

if(temp->d == d)

{

cout<<"\nitem found at depth: "<<depth;

return;

}

else if(temp->d > d)

temp = temp->left;

else

temp = temp->right;

}

cout<<"\n item not found";

return;

}

int main()

{

char ch;

int n, i, a[10] = {91, 23, 35, 24, 72, 37, 32, 14, 134, 86};

node \*root = new node;

root = NULL;

for (i = 0; i < 10; i++)

root = InsertIntoTree(root, a[i]);

up:

cout<<"\nEnter the Element to be searched: ";

cin>>n;

Search(root, n);

cout<<"\n\n\tDo you want to search more...enter choice(y/n)?";

cin>>ch;

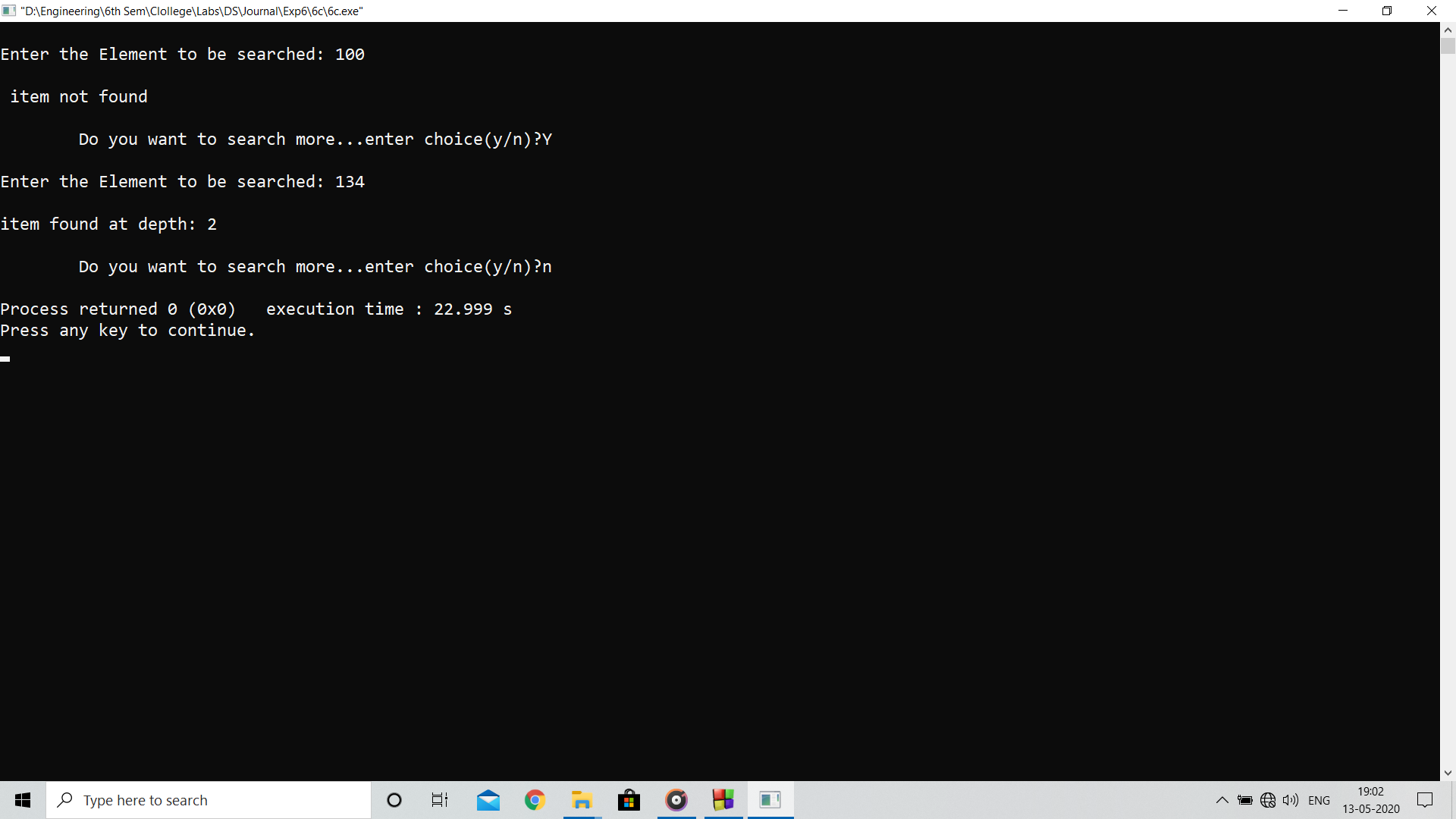
if(ch == 'y' || ch == 'Y')

goto up;

return 0;

}

**Output:**

****

**Analysis :**

Binary tree search is more efficient than regular search as here each at node the probability of traversal is halved thus the time required also graduallyreduces