2. Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree –

i.Insert new node

ii.Find number of nodes in longest path

iii.Minimum data value found inthe tree

iv.Search a value

#include<iostream>

#include<math.h>

using namespace std;

struct Bstnode

/\*Null 70 null\*/

{

int data;

Bstnode \*left = NULL;

Bstnode \*right = NULL;

};

class Btree

{

int n;

int x;

int flag;

public:

Bstnode \* root;

Btree()

{

root = NULL;

}

Bstnode \*GetNewNode(int in\_data)

{

Bstnode \* ptr = new Bstnode();

ptr->data = in\_data;

ptr->left = NULL;

ptr->right = NULL;

return ptr;

}

Bstnode \*insert( Bstnode \*temp , int in\_data)

{

if( temp == NULL )

{

temp = GetNewNode(in\_data);

}

else if( temp->data > in\_data)

{

temp->left = insert(temp->left , in\_data);

}

else

{

temp->right = insert( temp->right , in\_data);

}

return temp;

}

void input()

{

cout<<"ENTER NUMBER OF ELEMENTS IN THE BST : ";

cin>>n;

for(int i = 0 ; i < n ; i++)

{

cout<<"NUMBER = ";

cin>>x;

root = insert(root , x);

}

}

int search(Bstnode \*temp ,int in\_data)

{

if( temp != NULL)

{

if(temp->data == in\_data)

{

cout<<":-- RECORD FOUND --:"<<endl;

return 1;

}

else if(in\_data < temp->data)

{

this->search(temp->left, in\_data);

}

else if(in\_data > temp->data)

{

this->search(temp->right , in\_data);

}

}

else

{

return 0;

}

}

void minvalue(Bstnode \*temp)

{

while(temp->left != NULL)

{

temp = temp->left;

}

cout<<"MINIMUM VALUE = "<<temp->data<<endl;

}

void display()

{

cout<<endl<<"--- INORDER TRAVERSAL ---"<<endl;

inorder(root);

cout<<endl;

cout<<endl<<"--- POSTORDER TRAVERSAL ---"<<endl;

postorder(root);

cout<<endl;

cout<<endl<<"--- PREORDER TRAVERSAL ---"<<endl;

preorder(root);

cout<<endl;

cout<<endl<<"--- Depth ---"<<endl;

int dep = depth(root);

cout<<dep;

cout<<endl;

}

void inorder(Bstnode \*temp)

{

if(temp != NULL)

{

inorder(temp->left);

cout<<temp->data<<" ";

inorder(temp->right);

}

}

void postorder(Bstnode \*temp)

{

if(temp != NULL)

{

postorder(temp->left);

postorder(temp->right);

cout<<temp->data<<" ";

}

}

void preorder(Bstnode \*temp)

{

if(temp != NULL)

{

cout<<temp->data<<" ";

preorder(temp->left);

preorder(temp->right);

}

}

//int depth(Bstnode \*temp)

/\*{

if(temp == NULL)

return 0;

cout<< (max((depth(temp->left)),(depth(temp->right))) +1);

}\*/

int depth(Bstnode \* temp)

{

if(temp == NULL)

return 0;

else

{

int left\_side;

int right\_side;

left\_side = depth(temp -> left);

right\_side = depth(temp -> right);

if(left\_side > right\_side)

{

return left\_side + 1;

}

else

return right\_side + 1;

}

}

};

int main()

{

int se;

Btree obj;

obj.input();

obj.display();

int a = 0;

cout<<"enter element to search:";

cin>>se;

a = obj.search(obj.root,se);

if( a == 0)

{

cout<<"ELEMENT NOT FOUND"<<endl;

}

else

cout<<"ELEMENT FOUND"<<endl;

obj.minvalue(obj.root);

obj.inorder(obj.root);

cout<<endl<<obj.depth(obj.root);

return 0;

}