NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

PROGRAM: SOFTWARE ENGINEERING

DATA STRUCTURES LAB

LAB TASK-09

SUBMITTED BY:

Name: Ahmed Ali

Roll No: 22P-9318

INSTRUCTOR NAME: Sir Saood Sarwar

A DEPARTMENT OF COMPUTER SCIENCE

Q1 CODE:

```
#include<iostream>
using namespace std;

class node
{
	private:
		int info;
		node *left;
		node *right;

	public:
		node(int info)
		{
			this->info=info;
			left=nullptr;
			right=nullptr;
		}

		friend class bst;
};

class bst
{
	private:
		node *root;

	public:
```

```
        bst()
        {
                root=nullptr;
        }




        //sir ka code insert ka

//      node *insert(node *n,int val)
//      {
//              if(n==nullptr)
//              {
//                      node *newnode = new node(val);
//                      if(n==nullptr)
//                      {
//                              n=newnode;
//                      }
//                      return newnode;
//              }
//
//              if(n->info==val)
//              {
//                      cout<<" already there..."<<endl;
//              }
//              else if(val<n->info)
//              {
//                      n->left=insert(n->left,val);
//              }
```

```
//                      else
//                      {
//                              n->right=insert(n->right,val);
//                      }
//                      return n;
//              }



        //previous semester code for insertion:

        node *insert(node *n,int val)
        {
                if(n==nullptr)
                {
                        return new node(val);
                }

                if(n->info==val)
                {
                        cout<<" "<<endl;
                }
                else if(val<n->info)
                {
                        n->left=insert(n->left,val);
                }
                else
                {
                        n->right=insert(n->right,val);
```

```
                }
                return n;
        }

        void insert(int val)
{
root=insert(root, val);
}

node *search(node *n, int val)
        {
if(n==nullptr)
                {
        return nullptr;
}
if(n->info==val)
                {
        return n;
}
else if(n->info>val)
                {
        return search(n->left,val);
}
else
                {
return search(n->right,val);
}
}
```

```
node *search(int val)
{
        return search(root, val);
        }

        void pre_order(node *n)
        {
        if(n==nullptr)
        {
                return;
                }
                cout<<n->info<<"\t";
                post_order(n->left);
                post_order(n->right);
        }


        void post_order(node *n)
        {
        if(n==nullptr)
        {
                return;
                }
                post_order(n->left);
                post_order(n->right);
                cout<<n->info<<"\t";
        }

        void inorder(node *n)
```

```
{
if(n==nullptr)
{
return;
}
post_order(n->left);
cout<<n->info<<"\t";
post_order(n->right);
}

void display_pre_order()
{
pre_order(root);
cout<<endl;
}

void display_post_order()
{
post_order(root);
cout<<endl;
}

void display_inorder()
{
inorder(root);
cout<<endl;
}

int find_min(node *n)
```

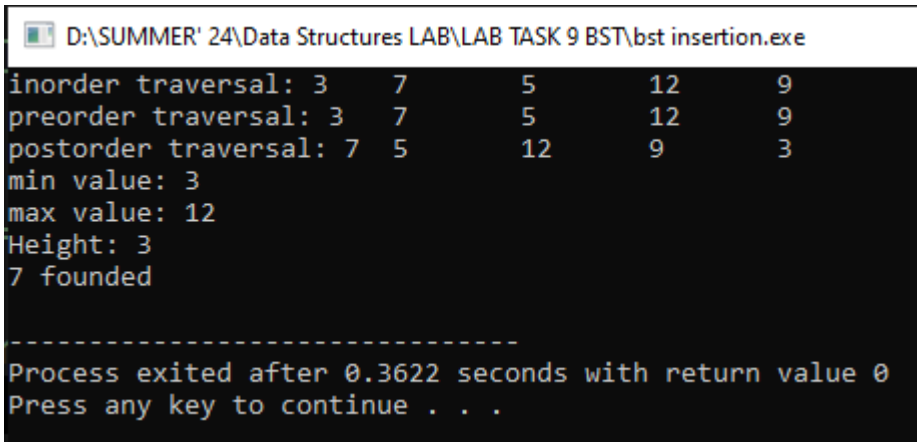
```
{
        if(n==nullptr)
        {
                return -1;
        }
        while(n->left != nullptr)
        {
                n=n->left;
        }
        return n->info;
}

int find_min()
{
        return find_min(root);
}


int find_max(node *n)
{
        if(n==nullptr)
        {
                return -1;
        }
        while(n->right!=nullptr)
        {
                n=n->right;
        }
        return n->info;
```

```
        }

        int find_max()
        {
                return find_max(root);
        }

        int height(node *n)
        {
  if(n==nullptr)
                {
    return -1;
  }
  int left_height=height(n->left);
  int right_height=height(n->right);
  if(left_height>right_height)
                {
    return left_height + 1;
  }
                else
                {
    return right_height + 1;
  }
}

int height()
        {
  return height(root);
}
```

```
};

int main()
{
    bst tree;
    tree.insert(3);
    tree.insert(9);
    tree.insert(5);
    tree.insert(12);
    tree.insert(7);

    cout<<"inorder traversal: ";
    tree.display_inorder();

    cout<<"preorder traversal: ";
    tree.display_pre_order();

    cout<<"postorder traversal: ";
    tree.display_post_order();

    cout<<"min value: "<<tree.find_min()<<endl;
    cout<<"max value: "<<tree.find_max()<<endl;
    cout<<"Height: "<<tree.height()<<endl;

    int search_val=7;
    if(tree.search(search_val))
            {
```

```
    cout<<search_val<<" founded"<<endl;
  }
        else
        {
    cout<<search_val<<" not founded"<<endl;
        }
  return 0;
}
```

Output-01:



Q2 CODE:

```
#include<iostream>
using namespace std;

class Node
{
```

```
        public:
          int data;
          Node *left;
                Node *right;
};

Node *new_node(int data)
{
  Node *node=new Node;
  node->data=data;
  node->left=nullptr;
        node->right=nullptr;
  return node;
}

int inorder(Node *root, int arr[], int index)
{
  if(root==nullptr)
        {
                return index;
        }
  index=inorder(root->left,arr,index);
  arr[index++]=root->data;
  index=inorder(root->right,arr,index);
  return index;
}

void common_nodes(int arr1[], int size1, int arr2[], int size2)
{
```

```
    int i=0;
            int j=0;


            while(i<size1 && j<size2)
            {
        if(arr1[i]==arr2[j])
                        {
            cout<<arr1[i]<<" ";
            i++;
            j++;
        }
                        else if(arr1[i]<arr2[j])
                        {
            i++;
        }
                        else
                        {
            j++;
        }
    }
}



Node *insert(Node *root,int data)
{
    if(root==nullptr)
            {
                        return new_node(data);
            }
```

```
        if(data<root->data)
        {
                root->left=insert(root->left,data);
        }
    else
        {
                root->right=insert(root->right,data);
        }
    return root;
}



Node *bst(int n)
{
    Node *root=nullptr;
    for(int i=0; i<n; ++i)
        {
      int data;
      cin>>data;
      root=insert(root,data);
    }
    return root;
}

int main()
{
    int n1, n2;
    cout<<"number of nodes to be entered in the first BST: "<<endl;
```

```
    cin>>n1;
    cout<<endl<<"Enter the nodes of the first BST: "<<endl;
    Node *root1=bst(n1);

    cout<<endl<<"number of nodes to be entered in the second BST: "<<endl;
    cin>>n2;
    cout<<"Enter the nodes of the second BST: "<<endl;
    Node *root2=bst(n2);

    int *arr1=new int[n1];
    int *arr2=new int[n2];
    int index1=0;
            int index2=0;

    index1=inorder(root1, arr1, index1);
    index2=inorder(root2, arr2, index2);

    cout<<"common nodes: ";
    common_nodes(arr1, n1, arr2, n2);

    delete []arr1;
    delete []arr2;

    return 0;
}
```

See Below For Output of Question 2:

Output-02:

D:\SUMMER' 24\Data Structures LAB\LAB TASK 9 BST\q2 intersection.exe

```
number of nodes to be entered in the first BST:
4

Enter the nodes of the first BST:
10
8
7
5

number of nodes to be entered in the second BST:
5
Enter the nodes of the second BST:
9
5
4
5
7
common nodes: 5 7
--------------------------------
Process exited after 23.04 seconds with return value 0
Press any key to continue . . .
```