AIM: IMPLEMENT BINARY SEARCH TREE INSERTION AND TRAVERSAL, OPERATION ON BST, LARGEST NODE, SMALLEST NODE, COUNT NUMBER OF NODES

THEORY:

A binary search tree (BST), also called an ordered or sorted binary tree, is a rooted binary tree whose internal nodes each store a key greater than all the keys in the node's left subtree and less than those in its right subtree. A binary tree is a type of data structure for storing data such as numbers in an organized way. Binary search trees allow binary search for fast lookup, addition and removal of data items, and can be used to implement dynamic sets and lookup tables. The order of nodes in a BST means that each comparison skips about half of the remaining tree, so the whole lookup takes time proportional to the binary logarithm of the number of items stored in the tree. This is much better than the linear time required to find items by key in an (unsorted) array, but slower than the corresponding operations on hash tables.

A binary search tree is a rooted binary tree, whose internal nodes each store a key (and optionally, an associated value), and each has two distinguished sub-trees, commonly denoted left and right. The tree additionally satisfies the binary search property: the key in each node is greater than or equal to any key stored in the left sub-tree, and less than or equal to any key stored in the right sub-tree. The leaves (final nodes) of the tree contain no key and have no structure to distinguish them from one another.

Attributes of Binary Search Tree:

A BST is made of multiple nodes and consists of the following attributes:

- Nodes of the tree are represented in a parent-child relationship
- Each parent node can have zero child nodes or a maximum of two sub-nodes or subtrees on the left and right sides.
- Every sub-tree, also known as a binary search tree, has sub-branches on the right and left of themselves.
- All the nodes are linked with key-value pairs.
- The keys of the nodes present on the left subtree are smaller than the keys of their parent node
- Similarly, the left subtree nodes' keys have lesser values than their parent node's keys.

Operations on Binary Search Tree:

BST primarily offers the following three types of operations for your usage:

- Search: searches the element from the binary tree
- Insert: adds an element to the binary tree
- Delete: delete the element from a binary tree

Each operation has its own structure and method of execution/analysis, but the most complex of all is the Delete operation.

Search Operation:

Always initiate analyzing tree at the root node and then move further to either the right or left subtree of the root node depending upon the element to be located is either less or greater than the root.

Insert Operation:

This is a very straight forward operation. First, the root node is inserted, then the next value is compared with the root node. If the value is greater than root, it is added to the right subtree, and if it is lesser than the root, it is added to the left subtree.

Delete Operations:

Delete is the most advanced and complex among all other operations. There are multiple cases handled for deletion in the BST.

- Case 1- Node with zero children: this is the easiest situation, you just need to delete the node which has no further children on the right or left.
- Case 2 - Node with one child: once you delete the node, simply connect its child node with the parent node of the deleted value.
- Case 3 Node with two children: this is the most difficult situation, and it works on the following two rules
  - 3a - In Order Predecessor: you need to delete the node with two children and replace it with the largest value on the left-subtree of the deleted node
  - 3b - In Order Successor: you need to delete the node with two children and replace it with the largest value on the right-subtree of the deleted node

SOURCE CODE:

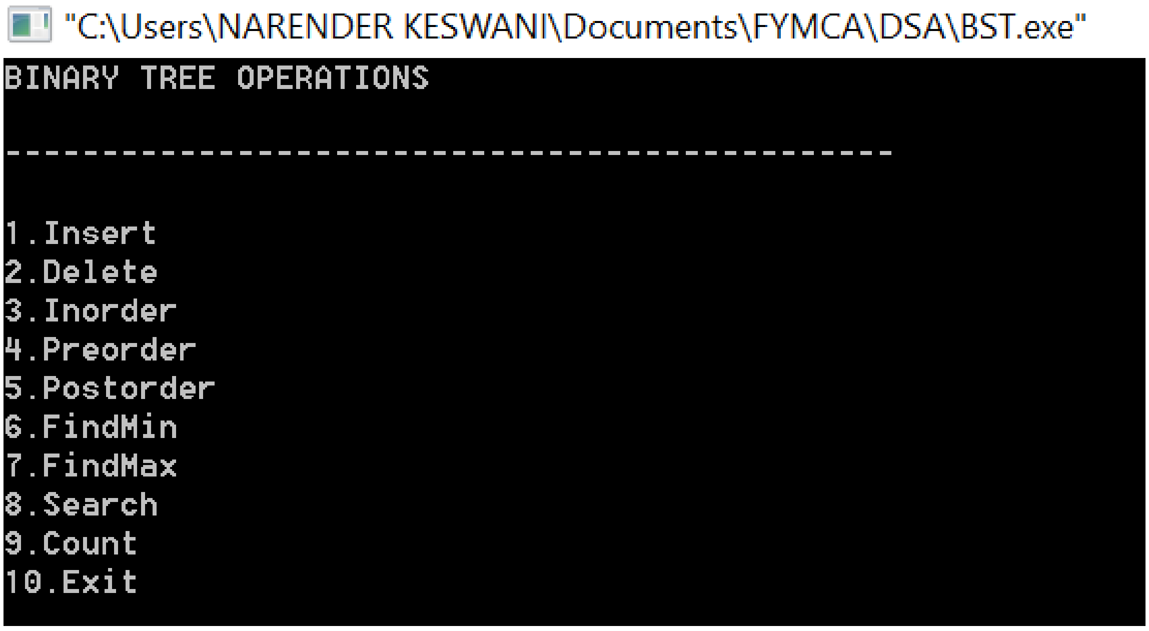
```
#include<iostream>
#include<stdlib.h>

using namespace std;

struct treeNode
{
   int data;
   treeNode *left;
   treeNode *right;
};
treeNode* FindMin(treeNode *node)
{
   if(node==NULL)
   {
      return NULL;
   }
   if(node->left)
      return FindMin(node->left);
   else
      return node;
}
treeNode* FindMax(treeNode *node)
{
   if(node==NULL)
   {
      return NULL;
   }
   if(node->right)
      return(FindMax(node->right));
   else
      return node;
}
treeNode *Insert(treeNode *node,int data)
{
   if(node==NULL)
   {
      treeNode *temp;
      temp=new treeNode;
```

```
    temp -> data = data;
    temp -> left = temp -> right = NULL;
    return temp;
  }
  if(data >(node->data))
  {
    node->right = Insert(node->right,data);
  }
  else if(data < (node->data))
  {
    node->left = Insert(node->left,data);
  }
  return node;
}
treeNode * Delet(treeNode *node, int data)
{
  treeNode *temp;
  if(node==NULL)
  {
    cout<<"Element Not Found";
  }
  else if(data < node->data)
  {
    node->left = Delet(node->left, data);
  }
  else if(data > node->data)
  {
    node->right = Delet(node->right, data);
  }
  else
  {

    if(node->right && node->left)
    {
      temp = FindMin(node->right);
      node -> data = temp->data;
      node -> right = Delet(node->right,temp->data);
    }
    else
    {
            temp = node;
            if(node->left == NULL)
               node = node->right;
            else if(node->right == NULL)
               node = node->left;
            free(temp);
         }
   }
   return node;
}
treeNode * Find(treeNode *node, int data)
{
   if(node==NULL)
   {
      return NULL;
   }
   if(data > node->data)
   {
      return Find(node->right,data);
   }
   else if(data < node->data)
   {
      return Find(node->left,data);
   }
   else
   {
      return node;
   }
}
void Inorder(treeNode *node)
{
   if(node==NULL)
   {
      return;
   }
   Inorder(node->left);
   cout<<node->data<<" ";
   Inorder(node->right);
}
void Preorder(treeNode *node)
{
   if(node==NULL)
   {
      return;
   }
   cout<<node->data<<" ";
   Preorder(node->left);
   Preorder(node->right);
}
void Postorder(treeNode *node)
{
   if(node==NULL)
   {
      return;
   }
   Postorder(node->left);
   Postorder(node->right);
   cout<<node->data<<" ";
}

int countNodes(treeNode *node)
{
   if(node==0)
      return 0;
   else
      return(countNodes(node->left)+countNodes(node->right)+1);
}
int main()
{
   treeNode *root = NULL,*temp;
   int ch;

   cout<<"BINARY TREE OPERATIONS"<<endl;
   cout<<"\n---------------------------------------------"<<endl;

cout<<"\n1.Insert\n2.Delete\n3.Inorder\n4.Preorder\n5.Postorder\n6.FindMin\n7.FindMax
\n8.Search\n9.Count\n10.Exit\n";
   do
   {
      cout<<"\nEnter the Choice: \n"<<endl;
      cin>>ch;
      switch(ch)
```

```
{
case 1:
   cout<<"\nEnter element to be insert:";
   cin>>ch;
   root = Insert(root, ch);
   cout<<"\nElements in BST are:";
   Inorder(root);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 2:
   cout<<"\nEnter element to be deleted:";
   cin>>ch;
   root = Delet(root,ch);
   cout<<"\nAfter deletion elements in BST are:";
   Inorder(root);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 3:
   cout<<"\nInorder Travesals is:";
   Inorder(root);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 4:
   cout<<"\nPreorder Traversals is:";
   Preorder(root);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 5:
   cout<<"\nPostorder Traversals is:";
   Postorder(root);
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 6:
   temp = FindMin(root);
   cout<<"\nMinimum element is :"<<temp->data;
   cout<<"\n---------------------------------------------"<<endl;
   break;
case 7:
   temp = FindMax(root);
   cout<<"\nMaximum element is :"<<temp->data;
   cout<<"\n---------------------------------------------"<<endl;
```

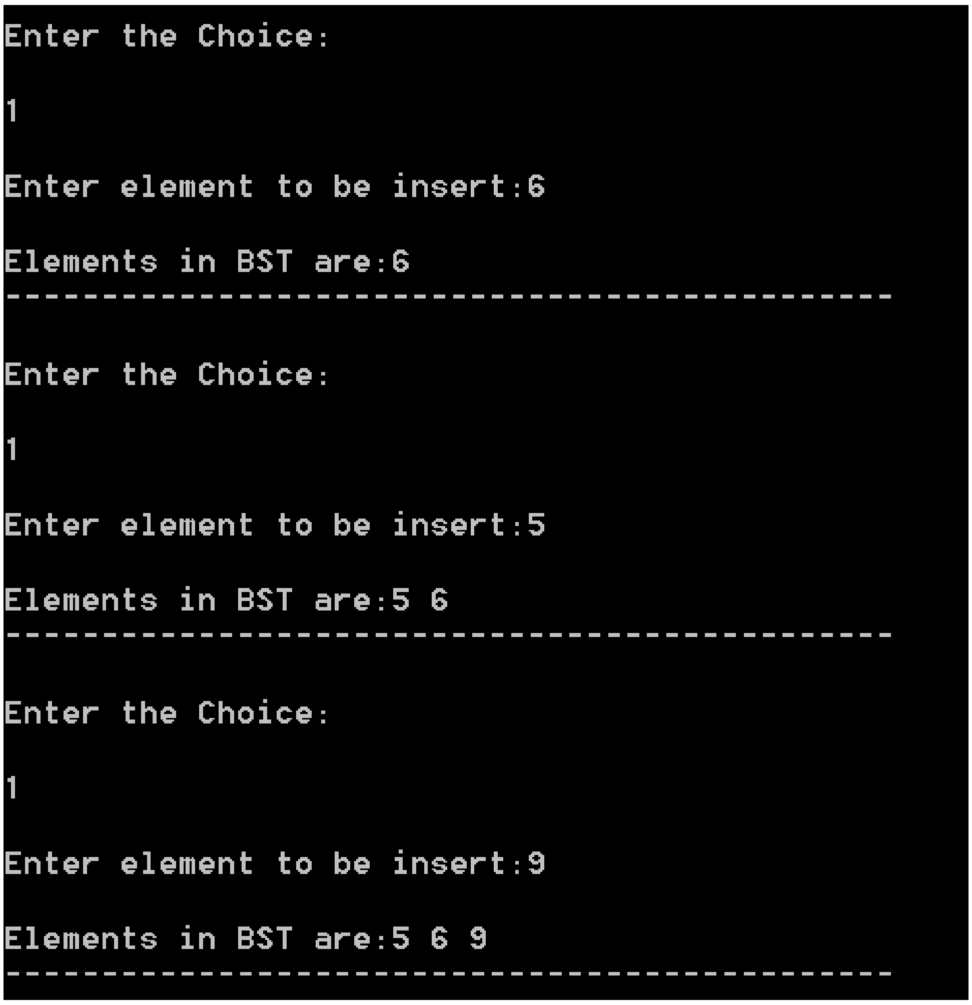
```
            break;
        case 8:
            cout<<"\nEnter element to be searched:";
            cin>>ch;
            temp = Find(root,ch);
            if(temp==NULL)
            {
                cout<<"Element is not found"<<endl;
            }
            else
            {
                cout<<"Element "<<temp->data<<" is Found\n";
            }
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 9:
            cout<<countNodes(root)<<endl;
            cout<<"\n---------------------------------------------"<<endl;
            break;
        case 10:
              exit(0);
            break;
        default:
            cout << "\nSelect Proper Option (1/2/3/4/5/6/7/8/9/10)" << endl;
            break;
        }
    }
    while(ch!=10);
    return 0;
}
```

OUTPUT:



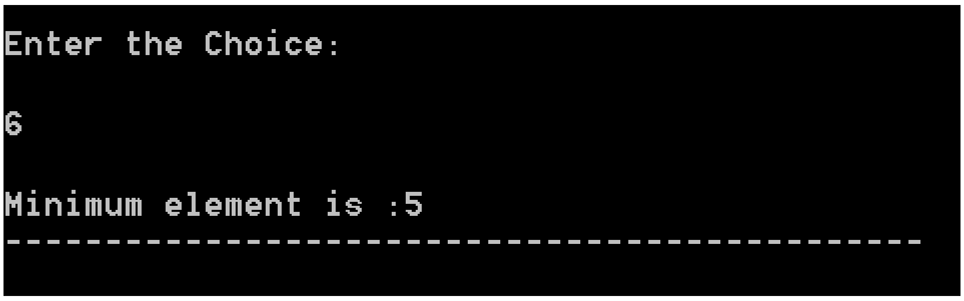
```
"C:\Users\NARENDER KESWANI\Documents\FYMCA\DSA\BST.exe"
BINARY TREE OPERATIONS

----------------------------------------------

1.Insert
2.Delete
3.Inorder
4.Preorder
5.Postorder
6.FindMin
7.FindMax
8.Search
9.Count
10.Exit
```

INSERT:



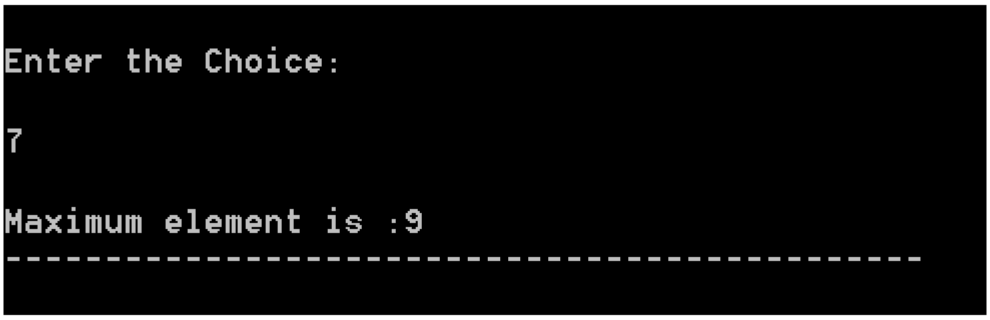
```
Enter the Choice:

1

Enter element to be insert:6

Elements in BST are:6
----------------------------------------------

Enter the Choice:

1

Enter element to be insert:5

Elements in BST are:5 6
----------------------------------------------

Enter the Choice:

1

Enter element to be insert:9

Elements in BST are:5 6 9
----------------------------------------------
```

FIND MIN:



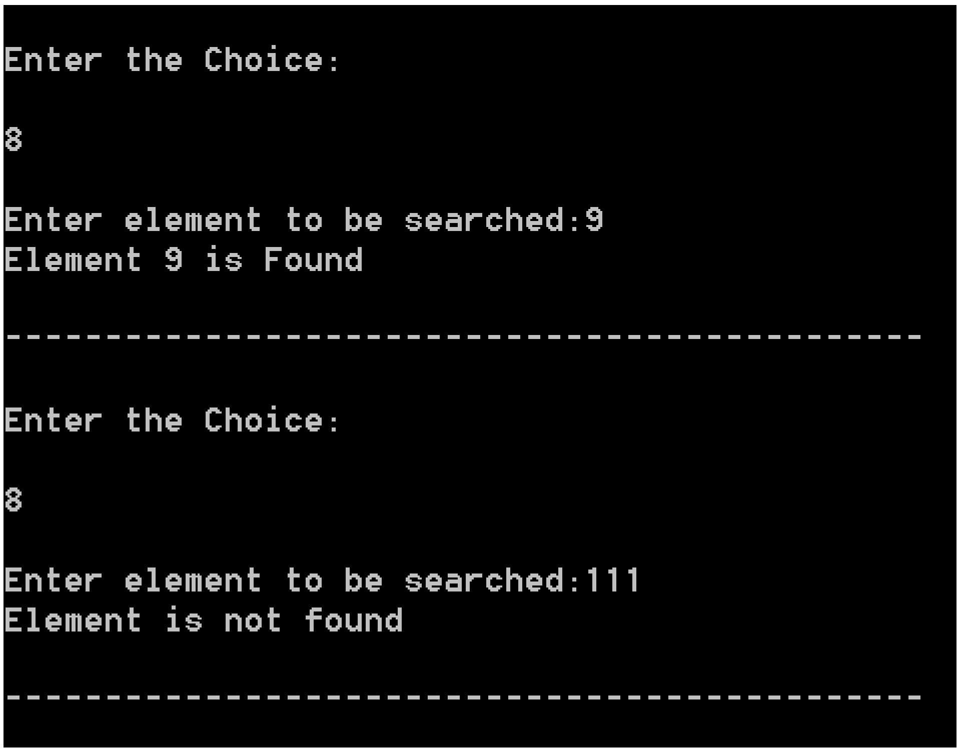
```
Enter the Choice:

6

Minimum element is :5
----------------------------------------------
```

FIND MAX:



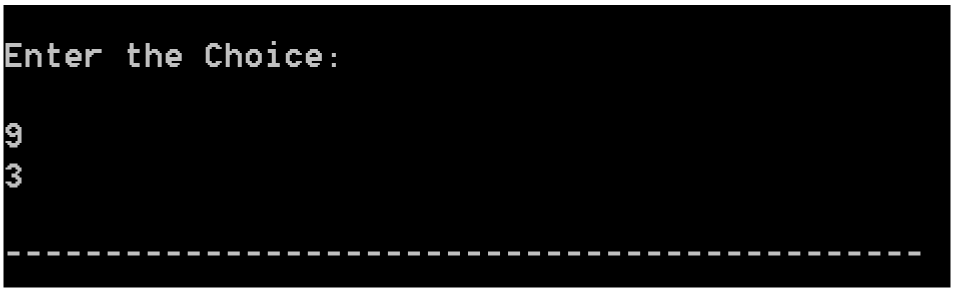
```
Enter the Choice:

7

Maximum element is :9
----------------------------------------------
```

SEARCH:



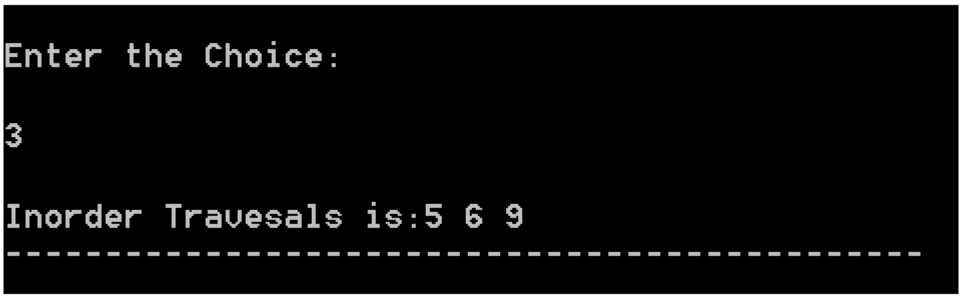
```
Enter the Choice:

8

Enter element to be searched:9
Element 9 is Found

----------------------------------------------

Enter the Choice:

8

Enter element to be searched:111
Element is not found

----------------------------------------------
```

COUNT:



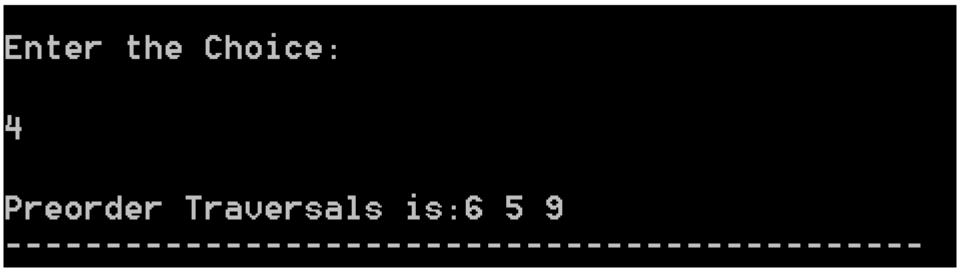
```
Enter the Choice:

9
3

--------------------------------------------------
```

INORDER:



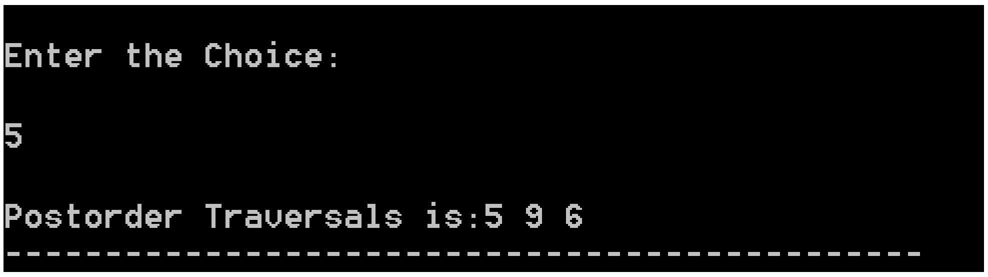
```
Enter the Choice:

3

Inorder Travesals is:5 6 9
--------------------------------------------------
```

PREORDER:



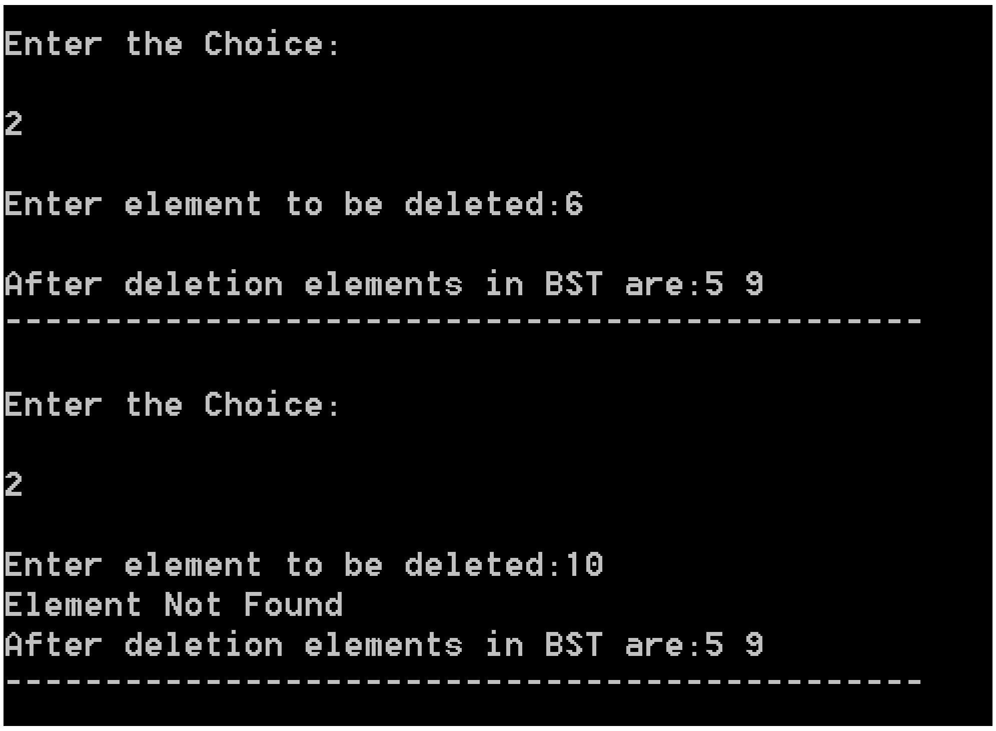
```
Enter the Choice:

4

Preorder Traversals is:6 5 9
--------------------------------------------------
```

POSTORDER:



```
Enter the Choice:

5

Postorder Traversals is:5 9 6
--------------------------------------------------
```

DELETE:



```
Enter the Choice:

2

Enter element to be deleted:6

After deletion elements in BST are:5 9
---------------------------------------------

Enter the Choice:

2

Enter element to be deleted:10
Element Not Found
After deletion elements in BST are:5 9
---------------------------------------------
```

CONCLUSION:

From this practical, I have learned about binary trees and their operations.