```
Group A2
Title :- Construct A BST & Perform Its different Operations.

#include<iostream>

using namespace std;

struct BSTNode
{
    int data;
    BSTNode *left;
    BSTNode *right;
};
class BinSearch{
    BSTNode *root;
    public:
        BSTNode* Create();
        int FindMax(BSTNode *);
        int FindMin(BSTNode *);
        BSTNode* Insert(BSTNode *);
        void Inorder(BSTNode *);
        void PreOrder(BSTNode *);
        void PostOrder(BSTNode *);
        BSTNode* FindLogestPath(BSTNode *);
        BSTNode* Search(BSTNode *);
        int LongestPath(BSTNode *,int,int);
        int ToExecuteLongestPath(BSTNode *);
};
int BinSearch::LongestPath(BSTNode *rt,int Length,int MaxLength)
{
    if(rt==NULL){
        if(MaxLength<Length){
        MaxLength =Length;
        }
        else if(MaxLength==Length){
                return Length;
        }
    }
    //For Left Sub-Tree
    LongestPath(rt->left,Length=Length+1,MaxLength);
    //For Right Sub-Tree
    LongestPath(rt->right,Length=Length+1,MaxLength);
}
int BinSearch::ToExecuteLongestPath(BSTNode *rt)
{
    if(rt==NULL)
```

```
        cout<<"-1";

    int Maxlen =1;
    int TempLength=0;
    int sum=LongestPath(rt,TempLength,Maxlen);
    cout<<"\n\tLongestpath Is:"<<sum;
}
BSTNode* BinSearch::Search(BSTNode *rt){
    int Key;
    BSTNode *Temp;
    Temp= new BSTNode;
    cout<<"\nEnter the Key You Want To Search";
    cin>>Key;
    if(rt==NULL){
        return NULL;
    }
    else{
        Temp=rt;
        while(Temp!=NULL){
            if(Key>Temp->data){
                Temp=Temp->right;
            }
            else if(Key==Temp->data){
                    cout<<"\nElement Found"<<Key;
                    break;
            }
            else if(Key<Temp->data){
                Temp=Temp->left;
            }
        }
        return Temp;
    }

}
BSTNode* BinSearch::Insert(BSTNode *BRT){
    cout<<"\nHow Many node You Wants To Insert ";
    int a;
    cin>>a;
    while(a){
        //Initalization Of New Node
        BSTNode *NewNode,*NN1,*NN2;
        int db;
         //Asking User To Insert the Data
         cout<<"\nEnter the Data You Want To Insert(-1 For Abort)";
        cin>>db;
```

```
        if(db==-1){
            return NULL;
        }
        else{
            NewNode = new BSTNode;
            NewNode->data=db;
            NewNode->left=NULL;
            NewNode->right=NULL;
            //Insertion at proper Postion
            NN1=BRT;
            NN2=NULL;
            while(NN1!=NULL){
                    NN2=NN1;
                if(NewNode->data<NN1->data){
                    NN1=NN1->left;
                }
                else{
                    NN1=NN1->right;
                }
                //If Root Node is Empty Then Tree Will Be Empty
                if(NN2==NULL){
                    NN2=NewNode;
                }
                    //If the Data In New Node IS LEss the IS's Leaf Node then Ins
ert to the left Node
                else if(NewNode->data < NN2->data){
                    NN2->left=NewNode;
                }
                else{
                    NN2->right=NewNode;
                }
            }
        }
        a--;
    }
}
void BinSearch::Inorder(BSTNode *rt){
    if(rt!=NULL){
        Inorder(rt->left);
        cout<<"\n"<<rt->data;
        Inorder(rt->right);
    }
}
void BinSearch::PreOrder(BSTNode *rt){
    if(rt){
```

```
        cout<<"\n"<<rt->data;
            PreOrder(rt->left);
            PreOrder(rt->right);
    }
}
void BinSearch::PostOrder(BSTNode *rt){
    if(rt){
            PostOrder(rt->left);
            PostOrder(rt->right);
            cout<<"\n"<<rt->data;
    }
}
BSTNode* BinSearch::Create(){
    int db;
    root= new BSTNode;
    cout<<"\nEnter The Root Node Data";
    cin>>db;
    root->data=db;
    root->left=NULL;
    root->right=NULL;
    cout<<"\nRoot Node has Been Inserted"<<"\n"<<root->left<<"|"<<root-
>data<<"|"<<root->right;
    return root;
}
int BinSearch::FindMax(BSTNode *root){
    BSTNode *Frt;
    Frt=new BSTNode;
    Frt=root;
    while(Frt->right!=NULL){
        Frt=Frt->right;
    }
    cout<<"\n";
    return Frt->data;
}
int BinSearch::FindMin(BSTNode * root){
    BSTNode *Flt;
    Flt=new BSTNode;
    Flt=root;
    while(Flt->left!=NULL){
        Flt=Flt->left;
    }
    cout<<"\n";
    return Flt->data;
}
int main(){
```

```
    BSTNode *rt;
    char Answer;
    int Choice;
    BinSearch B1;
    do
    {
        cout<<"\n\t1.Create\n\t2.Insert\n\t3.Find_Minimum_Value\n\t4.Find_Maximum
_value\n\t5.Inorder_Traversal\n\t6.Preorder_Traversal\n\t7.Postorder_Travsesal\n\
t8.Search\n\t9.FindLongestPath";
        cout<<"\nEnter Your Choice";
        cin>>Choice;
        switch(Choice)
        {
            case 1:rt=B1.Create();
                    break;

            case 2:B1.Insert(rt);
                    break;

            case 3:cout<<B1.FindMin(rt);
                    break;

            case 4:cout<<B1.FindMax(rt);
                    break;

            case 5:B1.Inorder(rt);
                    break;

            case 6:B1.PreOrder(rt);
                    break;

            case 7:B1.PostOrder(rt);
                    break;

            case 8:B1.Search(rt);
                    break;

            case 9:B1.ToExecuteLongestPath(rt);
                    break;

            default:
                     break;
        }
        cout<<"\nContinue?";
        cin>>Answer;
```

```
    }while(Answer=='y');
    return 0;
}
```

Output:-

```
        1.Create
        2.Insert
        3.Find_Minimum_Value
        4.Find_Maximum_value
        5.Inorder_Traversal
        6.Preorder_Traversal
        7.Postorder_Travsesal
        8.Search
        9.FindLongestPath
Enter Your Choice1

Enter The Root Node Data50

Root Node has Been Inserted
0|50|0
Continue?y

        1.Create
        2.Insert
        3.Find_Minimum_Value
        4.Find_Maximum_value
        5.Inorder_Traversal
        6.Preorder_Traversal
        7.Postorder_Travsesal
        8.Search
        9.FindLongestPath
Enter Your Choice2

How Many node You Wants To Insert 2

Enter the Data You Want To Insert(-1 For Abort)10

Enter the Data You Want To Insert(-1 For Abort)60

Continue?y

        1.Create
        2.Insert
```

```
        3.Find_Minimum_Value
        4.Find_Maximum_value
        5.Inorder_Traversal
        6.Preorder_Traversal
        7.Postorder_Travsesal
        8.Search
        9.FindLongestPath
Enter Your Choice3

10
Continue?y

        1.Create
        2.Insert
        3.Find_Minimum_Value
        4.Find_Maximum_value
        5.Inorder_Traversal
        6.Preorder_Traversal
        7.Postorder_Travsesal
        8.Search
        9.FindLongestPath
Enter Your Choice4

60
Continue?y

        1.Create
        2.Insert
        3.Find_Minimum_Value
        4.Find_Maximum_value
        5.Inorder_Traversal
        6.Preorder_Traversal
        7.Postorder_Travsesal
        8.Search
        9.FindLongestPath
Enter Your Choice5

10
50
60
Continue?y

        1.Create
        2.Insert
        3.Find_Minimum_Value
```

```
        4.Find_Maximum_value
        5.Inorder_Traversal
        6.Preorder_Traversal
        7.Postorder_Travsesal
        8.Search
        9.FindLongestPath
Enter Your Choice6

50
10
60
Continue?y

        1.Create
        2.Insert
        3.Find_Minimum_Value
        4.Find_Maximum_value
        5.Inorder_Traversal
        6.Preorder_Traversal
        7.Postorder_Travsesal
        8.Search
        9.FindLongestPath
Enter Your Choice6

50
10
60
Continue?y

        1.Create
        2.Insert
        3.Find_Minimum_Value
        4.Find_Maximum_value
        5.Inorder_Traversal
        6.Preorder_Traversal
        7.Postorder_Travsesal
        8.Search
        9.FindLongestPath
Enter Your Choice7

10
60
50
Continue?y
```

```
        1.Create
        2.Insert
        3.Find_Minimum_Value
        4.Find_Maximum_value
        5.Inorder_Traversal
        6.Preorder_Traversal
        7.Postorder_Travsesal
        8.Search
        9.FindLongestPath
Enter Your Choice8

Enter the Key You Want To Search10

Element Found10
Continue?y

        1.Create
        2.Insert
        3.Find_Minimum_Value
        4.Find_Maximum_value
        5.Inorder_Traversal
        6.Preorder_Traversal
        7.Postorder_Travsesal
        8.Search
        9.FindLongestPath
```