INPUT:

```
#include <iostream>
#define max 20
using namespace std;
class BST{
    struct node{
        int data;
        node *lc, *rc;
    }*root, *queue[max];
    int front,rear;
public:
    BST()   //constructor
    {
        root=NULL;
        front=rear=-1;
    }
    inline void create();
    void display();
    void minimum();
    void BFT(int);
    void Bsearch(int);
    void mirror();
    void enqueue(node *);
    node* dequeue();
    bool qempty();
    void clearQueue();
};
void BST::create()
{
    char ans='Y';
    do{
        node *temp, *curr;
        temp = new  node;
        temp->lc=NULL;
        temp->rc=NULL;
        cout<<"\nEnter an element: ";
        cin>>temp->data;
        if(root==NULL) //only executes first time
            root=temp;
        else
        {
            curr=root;
            while(curr!=NULL)
            {
            if((temp->data)<(curr->data))
            {
                if(curr->lc==NULL)
                {
                    curr->lc=temp;
                    break;
                }
                else
                    curr=curr->lc;
            }
            if((temp->data)>(curr->data))
            {
                if(curr->rc==NULL)
                {
                    curr->rc=temp;
                    break;
                }
                else
                    curr=curr->rc;
            }
            }
        }
        cout<<"\nDo you want to insert more? (y/n): ";
        cin>>ans;
    }while(ans=='y' || ans=='Y');
}
void BST::minimum()
{
    if(root!=NULL)
    {
        node *t;
        t=root;
        while(t->lc!=NULL)
            t=t->lc;
        cout<<"\nMinimum Value: "<<t->data;
    }
    else
        cout<<"\nBST Empty!";
}
bool BST::qempty()
{
     if(front==-1)
         return true;
     return false;
}
void BST::enqueue(node *t)
{
    if(rear==max-1)
        cout<<"\nQueue is full!";
    else
    {
        if(front==-1)
            front=rear=0;
        else
            rear++;
        queue[rear]=t;
    }
}
```

```
BST::node* BST::dequeue()
{
        node *t;
        if(!qempty())
        {
                t=queue[front];
                if(rear==front)
                        rear=front=-1;
                else
                        front++;
        }
        return t;
}
void BST::clearQueue()
{
        front=rear=-1;
}
void BST::BFT(int m)
{
        if(root==NULL)
                cout<<"Tree Empty!";
        else
        {
                int cnt=0;
                clearQueue();
                node *t=root;
                enqueue(t);
                enqueue(NULL);
                cout<<"Level Wise display: \n";
                while(!qempty())
                {
                        int tab=0;
                        t=dequeue();
                        if(t!=NULL)
                        {
                                cout<<t->data<<" ";
                                if(tab==2)
                                {
                                        cout<<"\t\t";
                                        tab=0;
                                }
                                else if(tab==1)
                                {
                                        cout<<"\t";
                                        tab=0;
                                }
                                if(t->lc!=NULL)
                                {
                                        enqueue(t->lc);
                                        tab++;
                                }
                if(t->rc!=NULL)
                {
                    enqueue(t->rc);
                    if(tab==1)
                        tab++;
                    else if(tab==0)
                        tab=2;
                }
            }
            else
            {
                if(!qempty())
                    enqueue(NULL);
                cnt++;
                cout<<endl;
            }
        }
        if(m==1)
        {
            cout<<"\nTotal nodes from root to farthest node(number of levels): "<<cnt;
        }
    }
}
void BST::Bsearch(int k)
{
    if(root==NULL)
        return;
    else
    {
        int cmp=0;
        node *t=root;
        int flag=0;
        while(t!=NULL)
        {
            cmp++;
            if(t->data==k)
            {
                flag=1;
                break;
            }
            else if(t->data<k)
                t=t->rc;
            else
                t=t->lc;
        }
        if(flag==0)
            cout<<"\nElement not present!";
        else
            cout<<"\nElement found in BST within "<<cmp<<" comparison(s).";
    }
}
```

```
void BST::mirror()
{
    if(root==NULL)
        return;
    else
    {
        cout<<"\nBST before swapping: ";
        BFT(0);
        clearQueue();
        node *t,*t1;
        t=root;
        enqueue(t);
        while(!qempty())
        {
            t=dequeue();
            t1=t->lc;
            t->lc=t->rc;
            t->rc=t1;
            if(t->lc!=NULL)
                enqueue(t->lc);
            if(t->rc!=NULL)
                enqueue(t->rc);
        }
        cout<<"\nAfter Swapping: ";
        BFT(0);
    }
}
void BST::display()
{
    int ch=0,flag=1, key;
    do{
    cout<<"\n\tDISPLAY MENU: ";
    cout<<"\n1.Create / Insert new node";
    cout<<"\n2.Breadth first Traversal";
    cout<<"\n3.Find number of nodes in longest path from root";
    cout<<"\n4.Minimum data value";
    cout<<"\n5.Swap the tree and display";
    cout<<"\n6.Search";
    cout<<"\n7.Exit";
    cout<<"\nEnter your choice: ";
    cin>>ch;
    switch(ch)
    {
    case 1: create();
            break;
    case 2: BFT(0);
            break;
    case 3:  BFT(1);
            break;
    case 4: minimum();
            break;
    case 5: mirror();
                break;
    case 6: cout<<"\nEnter value to be searched: ";
                cin>>key;
                Bsearch(key);
                break;
    case 7: cout<<"\nProgram exit";
                flag=0;
                break;
    default: cout<<"\nInvalid input.";
                 break;
    }
    if(flag==0)
        break;
    }while(true);
}
int main() {
    BST  obj;
    obj.display();
    return 0;
}
```

OUTPUT:

```
DISPLAY MENU:
1.Create / Insert new node
2.Breadth first Traversal
3.Find number of nodes in longest path from root
4.Minimum data value
5.Swap the tree and display
6.Search
7.Exit
Enter your choice: 1
Enter an element: 45
Do you want to insert more? (y/n): y
Enter an element: 53
Do you want to insert more? (y/n): y
Enter an element: 98
Do you want to insert more? (y/n): y
Enter an element: 32
Do you want to insert more? (y/n): y
Enter an element: 11
Do you want to insert more? (y/n): n
DISPLAY MENU:
1.Create / Insert new node
2.Breadth first Traversal
3.Find number of nodes in longest path from root
4.Minimum data value
5.Swap the tree and display
6.Search
7.Exit
Enter your choice: 2
```

```
Level Wise display:
45
32 53
11 98

            DISPLAY MENU:
1.Create / Insert new node
2.Breadth first Traversal
3.Find number of nodes in longest path from root
4.Minimum data value
5.Swap the tree and display
6.Search
7.Exit
Enter your choice: 3

Total nodes from root to farthest node(number of levels): 3

            DISPLAY MENU:
1.Create / Insert new node
2.Breadth first Traversal
3.Find number of nodes in longest path from root
4.Minimum data value
5.Swap the tree and display
6.Search
7.Exit
Enter your choice: 4
Minimum Value: 11

            DISPLAY MENU:
1.Create / Insert new node
2.Breadth first Traversal
3.Find number of nodes in longest path from root
4.Minimum data value
5.Swap the tree and display
6.Search
7.Exit
Enter your choice: 5
BST before swapping: Level Wise display:
45
32 53
11 98

After Swapping: Level Wise display:
45
53 32
98 11

            DISPLAY MENU:
1.Create / Insert new node
2.Breadth first Traversal
3.Find number of nodes in longest path from root
```

```
4.Minimum data value
5.Swap the tree and display
6.Search
7.Exit
Enter your choice: 6
Enter value to be searched: 43
Element not present!

        DISPLAY MENU:
1.Create / Insert new node
2.Breadth first Traversal
3.Find number of nodes in longest path from root
4.Minimum data value
5.Swap the tree and display
6.Search
7.Exit
Enter your choice: 7
Program exit
```