.vscode\dsa\Binary Tree\BinarySearchTree.cpp

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

class node
{
    public :
    int data ;
    node* left ;
    node* right ;
    // constructor
    node(int data)
    {
        this->data = data ;
        this->left = NULL ;
        this->right = NULL ;
    }
};

void levelOrderTraversal(node* root)
{
    queue<node*>q ;
    q.push(root) ;
    q.push(NULL) ;
    while(!q.empty())
    {
        node* temp = q.front() ;
        q.pop() ;
        if(temp == NULL)
        {
            cout<<endl ;
            if(!q.empty())
            {
                q.push(NULL) ;
            }
        }

        else
        {
            cout<<temp->data<<" " ;
            if(temp->left != NULL)
            {
                q.push(temp->left) ;
            }
            if(temp->right != NULL)
            {
                q.push(temp->right) ;
            }
        }
    }
}
```

```cpp

node* insertIntoBST(node* root, int data) // Time Complexity - O(logn)
{
    // base case
    if(root == NULL)
    {
        root = new node(data) ;
        return root ;
    }

    if(data > root->data)
    {
        root->right = insertIntoBST(root->right,data) ;
    }
    else{
        root->left = insertIntoBST(root->left,data) ;
    }

    return root ;
}

node* minVal(node* root)
{
    node* temp = root ;
    while(temp->left != NULL)
    {
        temp = temp->left ;
    }
    return temp ;
}

node* maxVal(node* root)
{
    node* temp = root ;
    while(temp->right != NULL)
    {
        temp = temp->right ;
    }
    return temp ;
}

node* deleteFromBST(node* root, int val) // time complexity - O(n)
{
    // base case
    if(root == NULL)
    {
        return root ;
    }
    if(root->data == val)
    {
        // 0 child
        if(root->left == NULL && root->right == NULL)
        {
            delete root ;
```

```
106             return NULL ;
107         }
108         // 1 child
109         // left child
110         if(root->left != NULL && root->right == NULL)
111         {
112             node* temp = root->left ;
113             delete root ;
114             return temp ;
115         }
116         //right child
117         if(root->left == NULL && root->right != NULL)
118         {
119             node* temp = root->right ;
120             delete root ;
121             return root->right ;
122         }
123 
124 
125         // 2 child
126         if(root->left != NULL && root->right != NULL)
127         {
128             int mini = minVal(root->right)->data ; // find minimum value node from right
129             root->data = mini ; // copy minimum value data into root node
130             root->right = deleteFromBST(root->right,mini) ;
131             return root ;
132         }
133 
134     }
135     else if(root->data > val)
136     { // left part mai jao
137         root->left = deleteFromBST(root->left,val) ;
138     }
139     else{ // right part mai jao
140         root->right = deleteFromBST(root->right,val) ;
141     }
142 }
143 
144 void takeInput(node* &root)
145 {
146     int data ;
147     cin>>data ;
148     while(data != -1)
149     {
150         root = insertIntoBST(root,data) ;
151         cin>>data ;
152     }
153 }
154 
155 int main()
156 {
157     node* root = NULL ;
158     cout<<"enter data to create BST - "<<endl;
159     takeInput(root) ;
```

```cpp

    cout<<"printing the BST"<<endl ;
    levelOrderTraversal(root) ;

    cout<<"min value is = "<<minVal(root)->data<<endl;
    cout<<"max value is = "<<maxVal(root)->data<<endl;

    root = deleteFromBST(root,30) ;

    return 0 ;
}

// NOTE : inorder traversal of BST is sorted


```