```
/*
A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct
a tree and print the nodes. Find the time and space requirements of your method.
*/


#include <iostream>
using namespace std;

//Structre to create node
struct Node{
   int data;
   Node* right;
   Node* left;
};

//Function to create node
Node* create(int data){
   Node* temp=new Node();
   temp->data=data;
   temp->left=temp->right=NULL;
   return temp;
}

//Function to insert node
void insert(Node* &root,int data){
   if(root==NULL){
      root=create(data);         //say 5 root
   }                             //Next element n
   else if(root->data > data){    //if (5 > n)
      insert(root->left,data);    //n will go to left side
   }
   else{                         //else
   insert(root->right,data);      //n will go to right side
   }
}
//Preorder
void displayPre(Node* root){
   if(root!=NULL){
      cout<<root->data<<" ";     //PARENT
      displayPre(root->left);    //LEFT
      displayPre(root->right);   //RIGHT
   }

}

//Inorder
void displayIn(Node* root){
   if(root!=NULL){
      displayIn(root->left);     //LEFT
      cout<<root->data<<" ";     //PARENT
      displayIn(root->right);    //RIGHT
   }

}

//Postorder
```

```
void displayPost(Node* root){
   if(root!=NULL){
      displayPost(root->left);     //LEFT
      displayPost(root->right);    //RIGHT
      cout<<root->data<<" ";       //PARENT
   }

}

//Function to calculate Height
int height(Node* root){
   if(root==NULL){
      return 0;
   }
   else{
      int l_h=height(root->left);
      int r_h=height(root->right);
      if(l_h>=r_h){
         return l_h+1;
      }
      else{
         return r_h+1;
      }
   }
}

//Function to seach for value
void search(Node* root,int value){
   if(root!=NULL){
   if(root->data>value){
      search(root->left,value);

   }
   else if(root->data<value){
      search(root->right,value);
   }
   else if(root->data==value){
      cout<<"\nelement FOUND";
   }
   }
   else{
      cout<<"\nelement NOT found";
   }
}

//Function to find smallest element i.e display extreme left
void smallest(Node* root){
   if(root->left!=NULL){
      smallest(root->left);
   }
   else{
      cout<<"Smallest :: "<<root->data;
   }
}

//Function to display largest element i.e display extreme right
void largest(Node* root){
```

```
    if(root->right!=NULL){
        largest(root->right);
    }
    else{
        cout<<"\nlargest :: "<<root->data;
    }
}

//Function mirror the tree
void mirror(Node* root){
    if(root==NULL){
        return;
    }
    mirror(root->left);
    mirror(root->right);
    swap(root->left,root->right);
}

int main(){
    bool loop=1;
    Node * root=NULL;
    int ch,n,num;
    while(loop==1){
    //Menu
    cout<<"\n-----MENU-----"<<endl
        <<"1. create BST"<<endl
        <<"2. preorder"<<endl
        <<"3. inorder"<<endl
        <<"4. postorder"<<endl
        <<"5. height"<<endl
        <<"6. search"<<endl
        <<"7. smallest"<<endl
        <<"8. largest"<<endl
        <<"9. mirror"<<endl
        <<"10. exit"<<endl
        <<"ENTER :: ";
    cin>>ch;
    switch (ch)
    {
        case 1:
        {
            cout<<"\nEnter the number of elements :: ";
            cin>>n;

            cout<<"Enter the numbers :: ";
            for(int i=0;i<n;i++){
                cin>>num;
                insert(root,num);
            }
            break;
        }
        case 2:
        {
            cout<<"\nPRE ORDER : ";
            displayPre(root);
            break;
        }
```

```
      case 3:
      {
         cout<<"\nIN ORDER : ";
         displayIn(root);
         break;
      }
      case 4:
      {
         cout<<"\nPOST ORDER : ";
         displayPost(root);
         break;
      }
      case 5:
      {
         int h=height(root);
         cout<<"\nHeight of BST :: "<<h;
         break;
      }
      case 6:
      {
         int value;
         cout<<"Enter the element to search :: ";
         cin>>value;
         search(root, value);
         break;
      }
      case 7:
      {
         smallest(root);
         break;
      }
      case 8:
      {
         largest(root);
         break;
      }
      case 9:
      {
         cout<<"\nBEFORE MIRROR"
            <<"\nInorder :: ";
         displayIn(root);
         mirror(root);
         cout<<"\nAFTER MIRROR"
            <<"\nInorder :: ";
         displayIn(root);
         break;
      }
      case 10:
      {
         loop=0;
         break;
      }
   }
}
   return 0;
}
```