```
#include<iostream>
using namespace std;
struct node {
      int  data;
      node *leftChild;
      node *rightChild;
}*b1,*b2;
/////// create a node in a BT function
node* CreateNode(int N) {
      node *new_node=new node;
      new_node->data = N;
      new_node->leftChild = NULL;
      new_node->rightChild = NULL;
      return(new_node);
}
/////// Is empty function
bool IsEmpty(node *p) {
      if (p==NULL) return true;
      else         return false;
}

/////// DFS functions
void preorder(node *p) {
      if (!IsEmpty(p)) {
            cout<<p->data<<" ";
            preorder(p->leftChild);
            preorder(p->rightChild);
      }
}
void inorder(node *p) {
      if (!IsEmpty(p)) {
            inorder(p->leftChild);
            cout<<p->data<<" ";
            inorder(p->rightChild);
      }
}
void postorder(node *p) {
      if (!IsEmpty(p)) {
            postorder(p->leftChild);
            postorder(p->rightChild);
            cout<<p->data<<" ";
      }
}
/////// find a node in a BT function recursive version
bool search(node *p, int N) {
      if (!IsEmpty(p))
            if (p->data < N)       return search(p->rightChild, N);
            else if (p->data > N)  return search(p->leftChild, N);
            else return true;
      return false;
}
/////// find a node in a BT function iterative version
/*bool search(node *p, int N) {
      while(!IsEmpty(p)) {
            if (p->data < N)  p=p->rightChild;
            else if (p->data > N)  p=p->leftChild;
            else return true;
      }
      return false;
}*/
/////// insert in a BT function
node* insert(node *p, int N) {
      if (IsEmpty(p))
            return CreateNode(N);
      if (p->data > N)
            p->leftChild = insert(p->leftChild, N);
      else if (p->data < N)
            p->rightChild = insert(p->rightChild, N);
      return p;
}
/////// height function
int height(node* p) {
      int l,r;
      if (IsEmpty(p))
            return 0;
      else {
            l =1+height(p->leftChild);
            r =1+height(p->rightChild);
            if (l > r)
                  return l;
            else
                  return r;
      }
}
/////// level order search functions
void CurrentLevel(node* p, int level) {
      if (IsEmpty(p))
            return;
      if (level == 1)
            cout<<p->data<<" ";
      else if (level > 1) {
            CurrentLevel(p->leftChild, level-1);
            CurrentLevel(p->rightChild, level-1);
      }
}
void LevelOrder(node* p) {
      for (int i=1; i<=height(p); i++)
            CurrentLevel(p, i);
}
/////// delete funcion in BST
node* minimum(node* p) {
      node* browser = p;
      while (browser && browser->leftChild != NULL)
            browser = browser->leftChild;

      return browser;
}
node *deleteNode(node *p, int N) {
      if (IsEmpty(p)) return p;
      if (N < p->data)
            p->leftChild = deleteNode(p->leftChild, N);
      else if (N > p->data)
            p->rightChild = deleteNode(p->rightChild, N);
      else {
            if ( p->leftChild == NULL) {
                  node *browser = p->rightChild;
                  delete p;
                  return browser;
            } else if ( p->rightChild == NULL) {
                  node *browser = p->leftChild;
                  delete p;
                  return browser;
            }
            node *browser = minimum(p->rightChild);
            p->data = browser->data;
            p->rightChild = deleteNode(p->rightChild, browser->data);
      }
      return p;
}
/////// Completeness test function
bool isComplete(node *p,int index,int numberOFnodes) {

      if (p == NULL)
            return true;
      if (index >= numberOFnodes)
            return false;
      bool R;
      R=isComplete(p->leftChild, 2*index + 1,numberOFnodes) &&
isComplete(p->rightChild, 2*index + 2,numberOFnodes);
      return R;
}
/////// sum function
int sum(node *p) {
      if (!IsEmpty(p))
            return p->data+sum(p->leftChild)+sum(p->rightChild);
}
/////// LCA function
node *lca(node *p, int n1, int n2) {
      if (p == NULL) return NULL;
      // If both n1 and n2 are smaller than root, then LCA lies in left
      if (p->data > n1 && p->data > n2)
            return lca(p->leftChild, n1, n2);
      // If both n1 and n2 are greater than root, then LCA lies in right
      if (p->data < n1 && p->data < n2)
            return lca(p->rightChild, n1, n2);
      return p;
}
/////// EXAM1
int test1=0;int test2=0;//global variables
void swap(int *x,int *y)
{     int z=*x; *x=*y; *y=z;    }
void exam1(node *p,int x, int y) {
	if (!IsEmpty(p)) {
		if (p->data==x && test1==0) {
			b1=p; test1=1;
			if (test2==1) {
				swap(&b1->data,&b2->data);
				return;
			}
		}
		if (p->data==y && test2==0) {
			b2=p;test2=1;
			if (test1==1) {
				swap(b1->data,b2->data);;
				return;
			}
		}
		exam1(p->leftChild,x,y);
		exam1(p->rightChild,x,y);
	}
}
/////// main function
int main() {

	node *root =NULL;
	int size,a,x,y;
	/*node *node2,*node3,*node4,*node5,*node6,*node7;
	root=CreateNode(10);
	node2=CreateNode(10);
	node3=CreateNode(10);
	node4=CreateNode(3);
	node5=CreateNode(7);
	node6=CreateNode(8);
	node7=CreateNode(8);
	root->leftChild=node2;
	root->rightChild=node3;
	node2->leftChild=node4;
	node2->rightChild=node5;
	node5->leftChild=node6;
	node5->rightChild=node7;*/
	cout<<"W're about to construct a Binary Search Tree"<<endl;
	cout<<"How many nodes it contains?"<<endl;
	cin>>size;
	cout<<"OK! the BST contains "<<size<<" nodes, let us fill the
tree"<<endl;
	cout<<"Enter the value of the root\n";
	cin>>a;
	root=insert(root,a);
	for(int i=1; i<size; i++) {
		cout<<"Enter a non duplicated value\n";
		cin>>a;
		root=insert(root,a);
	}
	cout<<"Done! the BST is filled"<<endl;
	cout<<"The tree height is ="<<height(root)<<endl;
menu:
      cout<<"\nMake a choice:\n1-Preorder display\n2-Inorder display\n3-
Postorder display\n4-Level order display\n5-Completness test\n6-Search a
node\n7-Delete a node\n8-Exit\n";
      do cin>>x;
      while(x<1 || x>8);
      switch(x) {
            case 1:
                  preorder(root);
                  break;
            case 2:
                  //exam1(root,10,8);
                  inorder(root);
                  break;
            case 3:
                  postorder(root);
                  break;
            case 4:
                  LevelOrder(root);
                  break;
            case 5:
                  if (isComplete(root,0,size))cout<<"\nComplete tree!\n";
                  else cout<<"\nThe tree is not complete!\n";
                  break;
            case 6:
                  cout<<"\nsearch an existing node, enter a value\n";
                  cin>>y;
                  if (search(root,y))cout<<"found";
                  else cout<<"NOT found";
                  break;
            case 7:
                  cout<<"\nEnter a value to be deleted";
                  cin>>y;
                  root=deleteNode(root,y);
                  size--;

      }

      if(x!=8) goto menu;
      cout<<sum(root);
      return 0;
}
```