```
template <typename T>
struct Node {
	T info;
	Node<T>* left;
	Node<T>* right;
};
```

```
int max(int a, int b) const
{
	return (a > b) ? a : b;
}
```

```
int height(Node<T>* root) const
 {
        if (root == nullptr) {
                return 0;
        }
        else {
                return 1 + max(height(root->left),
                                height(root->right));
        }
 }
```

```
template <typename T>
class BinaryTree {
public:
    BinaryTree();
    ~BinaryTree();
    void deleteTree();
    int height() const;
    void inorder(std::ostream& os = std::cout) const;
    void preorder(std::ostream& os = std::cout) const;
    void postorder(std::ostream& os = std::cout) const;
    void insert(const T& item);
    T* search(const T& item) const;
    int numNodes() const;
    int numLeafs() const;
private:
        Node<T>* root;
        int height(Node<T>* r) const;
        void deleteTree(Node<T>*& r);
        int max(int a, int b) const;
        void inorder(Node<T>* r, std::ostream& os) const;
        void preorder(Node<T>* r, std::ostream& os) const;
        void postorder(Node<T>* r, std::ostream& os) const;
        int numNodes(Node<T>* r) const;
        int numLeafs(Node<T>* r) const;
        void insert(Node<T>*& r, const T& item);
        T* search(Node<T>* r, const T& item) const;
};
```

```
template <typename T>
BinaryTree<T>::BinaryTree()
{
        root = nullptr;
}
template <typename T>
BinaryTree<T>::~BinaryTree()
{
         deleteTree(root);
}
```

```
template <typename T>
void BinaryTree<T>::deleteTree(Node<T>*& r)
{
	if (r != nullptr) {
		deleteTree(r->left);
		deleteTree(r->right);
		delete r;
		r = nullptr;
	}
}
```

```
template <typename T>
int BinaryTree<T>::height(Node<T>* r) const
{
        int result;
        if (r == 0) {
                result = 0;
        }
        else {
                result = 1 + max(height(r->left), height(r->right));
        }
        return result;
}
```

```
template <typename T>
void BinaryTree<T>::preorder(std::ostream& os) const
{
        preorder(root, os);
}
template <typename T>
void BinaryTree<T>::preorder(Node<T>* r, std::ostream& os)
const
{
        using namespace std;
        if (r != nullptr) {
                os << r->value << endl;
                preorder(r->left, os);
                preorder(r->right, os);
        }
}
```

```
template <typename T>
void BinaryTree<T>::inorder(std::ostream& os) const
{
	inorder(root, os);
}

template <typename T>
void BinaryTree<T>::inorder(Node<T>* r, std::ostream& os)
const
{
	using namespace std;
	if (r != nullptr) {
		inorder(r->left, os);
		os << r->value << endl;
		inorder(r->right, os);
	}
}
```

```
template <typename T>
int BinaryTree<T>::numNodes() const
{
	return numNodes(root);
}

template <typename T>
int BinaryTree<T>::numNodes(Node<T>* r) const
{
	using namespace std;
	int num = 0;
	if (r != nullptr) {
		num += 1 + numNodes(r->left) + numNodes(r->right);
	}
	return num;
}
```

```
template <typename T>
int BinaryTree<T>::numLeafs() const
{
	return numLeafs(root);
}
template <typename T>
int BinaryTree<T>::numLeafs(Node<T>* r) const
{
	using namespace std;
	int num = 0;
	if (r != nullptr) {
		if ((r->left == nullptr) && (r->right == nullptr)) {
			++num;
		}
		else {
			num = numLeafs(r->left) + numLeafs(r->right);
		}
	}
	return num;
}
```