There will be only one public function (apart from the constructor/destructor):

```csharp
    OperationTree Optimize (RelationalAlgebraTree tree)
```

RelationalAlgebraTree on the input is the output of the Parse method of QueryProcessor. It should perform an analysis of the tree using equality of relational algebra expressions and statistics to judge which operations are better to be performed first.

The output is an OperationTree object. Each node in the tree represents one operator (join, sort, select, project etc.) or an atomic value. Operator nodes have references to other nodes as input arguments to their operator. The tree contains a root node. Each node has an evaluation function that will evaluate it’s arguments and applies the assigned operator over them. Consequently, calling `operationTree.Root.Evaluate()` (or just `operationTree.Evaluate()`) will evaluate all operators while setting all smarterators according to optimized SmartSettings.

In C# syntax:

```csharp
class OperationTree {
    OperationTreeOperatorNode Root { get; set; }
    T Evaluate () { return Root?.Evaluate(); }
}

class OperationTreeOperatorNode<T> {
    T Evaluate ();
}

delegate Smarterator OperatorType (param object[] args);

class OperationTreeOperatorNode : OperationTreeNode<Smarterator> {
    SmartSettings SS { get; set; }
    OperatorType Operator { get; set; }
    OperationTreeOperatorNode<object> [] Arguments { get; set; } = new OperationTreeOperatorNode<object> [] ();
    Smarterator Evaluate () { return Operator (Arguments. Select (n => n?.Evaluate()).ToArray()).? Smarterize (SS); }
}

class OperationTreeValueNode<T> : OperationTreeNode<T> {
    T Value { get; set; }
    T Evaluate () { return Value; }
}
```