1、判断一棵二叉树是不是完全二叉树的算法设计：

算法思想：完全二叉树的特点是，如果一个节点有右子树，那么它必定有左子树，并且所有的叶子节点都在最后两层。因此，我们可以通过层次遍历二叉树，当遇到一个节点，如果它没有左子树但是有右子树，那么它一定不是完全二叉树；如果它有左子树没有右子树，或者它是叶子节点，那么之后遍历到的所有节点都必须是叶子节点，否则它就不是完全二叉树。

伪代码：

```

function isCompleteTree(root) {

if root is null, return true

queue = new Queue()

queue.enqueue(root)

leaf = false

while queue is not empty {

node = queue.dequeue()

left = node.left

right = node.right

if (right is not null and left is null) or (leaf and (left is not null or right is not null)), return false

if left is not null, queue.enqueue(left)

if right is not null, queue.enqueue(right)

else leaf = true

}

return true

}

```

2、软件系统功能结构图的问题：

（1）统计软件系统需要实现的页面总数的算法设计：

算法思想：这是一个统计树的叶子节点数量的问题，可以通过深度优先搜索或者广度优先搜索来解决。

伪代码：

```

function countPages(root) {

if root is null, return 0

if root is a leaf node, return 1

return countPages(root.left) + countPages(root.right)

}

```

（2）统计系统所有页面的总开发时间的算法设计：

算法思想：这是一个求树的带权路径长度的问题，可以通过深度优先搜索来解决。

伪代码：

```

function totalDevTime(root, depth) {

if root is null, return 0

if root is a leaf node, return depth \* root.code

return totalDevTime(root.left, depth+1) + totalDevTime(root.right, depth+1)

}

```

3、树状索引结构中，查找性能与树结构的关系：

在树状索引结构中，查找性能主要与树的高度有关。一般来说，树的高度越低，查找性能越高。这是因为查找过程即为在索引树中寻找一条从树根到叶子节点的路径，路径的长度即为树的高度。因此，B树和B+树等平衡树结构被广泛用于数据库索引，它们可以保证树的高度尽可能低，从而提高查找性能。同时，B+树的所有关键字都在叶子节点，非叶子节点仅用来索引，这使得在B+树中查找每个关键字的路径长度都相等，查找性能更稳定。