**时间复杂度与空间复杂度**

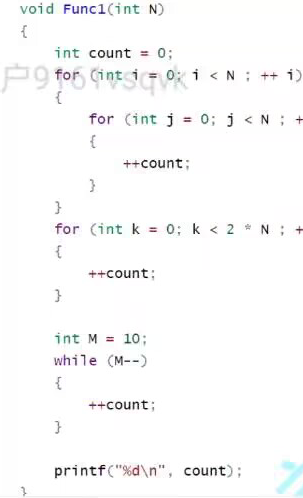
**算法效率：  
算法效率分为两种：第一种：时间效率 第二种：空间效率**

**现在比较看重的是时间效率**

**时间复杂度不是去计算一个算法真正跑了多少秒。**

**时间复杂度的概念**

**时间复杂度主要是算算法中的基本操作的执行次数，为算法的时间复杂度**

****

**对于上面的时间复杂度是否是N\*N+2\*N+10呢**

**我们会发现，随着N的增大，这个表达式中N^2对结果的影响是最大的**

**时间复杂度是一个估算，是去看表达式中影响最大的哪一项**

**时间复杂度用的是大O的渐进表示法，估算时间复杂度：O（N^2）**

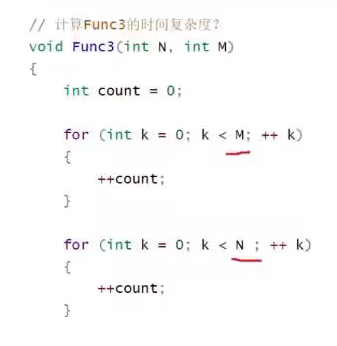
**因为其他两项对整个式子的影响是不大的，所以只看影响最大的一项。**

**推导大O阶方法：  
1.用常数1取代运算时间中的所有加法常数。**

**2.在修改后的运算次数函数中，只保留最高阶项**

**3.如果最高阶项存在且不是1，则去除去这个项目相乘的常数。得到的结果就是大O阶。**

**使用大O的渐进表示法以后，Func1的时间复杂度就是：O（N2）**

****

**对于上面函数的时间复杂度为O（M+N）**

**如果M远大于N时间复杂为O（M）**

**如果N和M相近的话时间复杂度就是O（M）或者O（N）**

**只要是确定的常数次时间复杂度就是O（1）**

**如果一个算法的时间复杂度是O(1)那就说明这个算法的运算次数是常数次**

**另外有些算法的时间复杂度存在最好、平均、最坏情况：  
最坏情况：在任意输入规模的最大运行次数（上界）**

**平均情况：任意输入规模的期望运行次数**

**最好情况：任意输入规模的最小运行次数**

**例如：在一个长度为N数组中搜索一个数据x**

**最坏情况：N次找到了**

**平均情况：N/2找到了**

**最好情况：一次找到了**

**当算法存在这三种情况的时候，看最坏的情况**

**如果算法时间复杂度是O(1)说明这个算法的效率是不变的**

**如果算法的时间复杂度是O(N)说明这个算法的效率随着字符串的长度变长我的效率也会线性的变长**

**冒泡排序的思想：**

**如果前一个大于后一个就交换，冒泡实际上就是挨着挨着往后走，把最大的排到最后面。**

**冒泡排序的时间复杂度是O(N^2)**

**因为对于冒泡排序第一趟冒N次最大的到了最右边，第二趟是的次数是N-1后面以此类推，所以这个是一个等差数列**

**所以影响最大的一项是N^2所以时间复杂度就是O(N^2)**

**不能认为两个循环嵌套在一起就是N^2这要根据实际情况进行分析具体要看程序**

**对于一个二分查找的时间复杂度，最坏的情况下是一个一个找，时间复杂度就是O(N)**

**算法优化：夹一个mid**

**最好的情况下：O(1)**

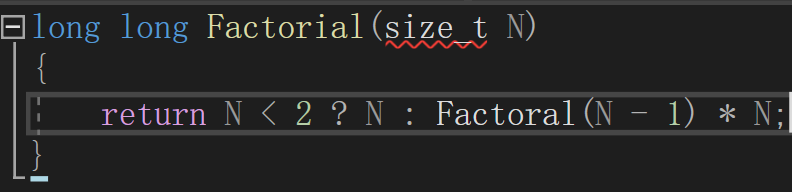
**这种算法也叫作折半查找。**

**对于时间复杂度最好是通过过程去分析**

**所以对于折半查找的话，假设找了X次，所以2^X=N**

**X=log2 N**

**算法的复杂度计算喜欢省略简写成logN，因为很多地方不好写底数（O(logN)）**

****

**时间复杂度**

**二分查找：N/2/2/2/2………==1;**

**N/2^n==1;**

**N=2^n;**

**n=logN;**

**有些书籍甚至写成了lgN,但是这个是不好的，不建议，容易和数学当中的混淆，如果是其他底数的，就正常写。**

**时间复杂度的意义：**

1. **复杂度（时间、空间），主要看时间**
2. **实现难度、简洁程度（可读性、可维护性）**

**二分查找**

**实践中，不太行**

1. **排序**
2. **有插入和删除才是最麻烦的**

**搜索树->红黑数 AVL树 B树系列 哈希**

**对于递归函数的调用的时间复杂度算法是将每次调用的运行次数加起来的总和就是他的时间复杂度**

**空间复杂度：**

**空间是可以重复利用的，时间是累加的，一去不返。**

**因为算法的需要额外开空间，才能算进空间复杂度**

**函数栈帧是依次向下建立的，上一个空间还给内存之后，下一个函数用的就是上一个函数的栈帧空间（前提是两个函数的实现是一样的）。**

**递归的前后两个空间也是一样的。**

**对于斐波那契数列的递归实现来说，空间复杂度就是O(N)。**

**因为第一个递归完了之后，第一个的递归深度是O(N)，但是第二个递归的空间是复用第一个的空间，所以总共的空间复杂度是O(N)**