**1.4 算法分析**

**1.4.1 科学方法**

**1.4.2 观察**

我们重点研究如何更好的将问题规模和运行时间的关系量化。

编程：统计一个文件中所有和为0的三整数元组的数量。

使用stopwatch打印运行时间

产生一系列随机输入数组，在每一步中将数组长度加倍，并打印ThreeSum.count()处理每种输入规模所需的运行时间。

**1.4.3 数学模型**

**1.4.4 增长数量级的分类**

增长数量级：常数级别，对数级别，线性级别，线性对数级别，平方级别，立方级别和指数级别

执行最频繁的指令决定了程序执行的总时间，很多程序运行时间都只取决于其中的一小部分指令。

平方级别，立方级别和指数级别的算法对于大规模的问题是不可用的。

在本书中我们希望为各种基础问题找到对数级别，线性级别和线性对数级别的算法。

**1.4.5 设计更快的算法**

编程：改进2-sum算法。

改进2-sum的方法，当且仅当-a[i]存在于数组中时，a[i]存在于某个和为0的整数对之中。使用binarySearch的rank方法对-a[i]进行二分查找。

归并排序所需的时间和NlogN成正比，二分查找所需时间和logN成正比。

3-sum改进方法类似，改进后的总运行时间和N2\*logN成正比。

**1.4.6 倍率实验**

对于编写的每个程序，你都要能够回答这个基本问题：“该程序能在可接受的时间内处理这些数据吗？”。

**1.4.7 注意事项**

**1.4.8 处理对于输入的依赖**

**1.4.9 内存**

分析内存的使用比分析程序所需的运行时间要简单得多。

**1.4.10 展望**

在编程领域中，最常见的错误或许就是过于关注程序的性能。第二常见的错误或许是完全忽略了程序的性能。