偶尔也分析一个算法最好情形的性能。不过，通常这没有什么重要意义，因为它不代表典型的行为。平均情形性能常常反映典型的行为，而最坏情形的性能则代表对任何可能输入的性能的一种保证。还要注意，虽然在这一章我们分析的是Java程序，但所得到的界实际上是算法的界而不是程序的界。程序是算法以一种特殊编程语言的实现，程序设计语言的细节几乎总是不影响大 O 的答案。如果一个程序比算法分析提出的速度慢得多，那么可能存在低效率的实现。这在类似C++的语言中很普遍，比如，数组可能当作整体而被漫不经心地拷贝，而不是由引用来传递。不管怎么说，这在Java中也可能出现；在12.7节的最后两段有一个极其巧妙的例子来说明这个问题。因此，在后面各章我们将分析算法而不是分析程序。

一般说来，若无相反的指定，则所需要的量是最坏情况的运行时间。其原因之一是它对所有的输入提供了一个界限，包括特别坏的输入，而平均情况分析不提供这样的界。另一个原因是平均情况的界计算起来通常要困难得多。在某些情况下，“平均”的定义可能影响分析的结果。(例如，什么是下述问题的平均输入?)

作为一个例子，我们将在下一节考虑下述问题：

最大子序列和问题

给定(可能有负的)整数 $A_1, A_2, \cdots, A_N$，求 $\sum_{k=i}^{j} A_k$ 的最大值。(为方便起见，如果所有整数均为负数，则最大子序列和为0)。

例如：对于输入 $-2, 11, -4, 13, -5, -2$，答案为20(从 A_2 到 A_4)。

这个问题之所以有吸引力，主要是因为存在求解它的很多算法，而这些算法的性能又差异很大。我们将讨论求解该问题的四种算法。这四种算法在某台计算机上(究竟是哪一台具体的计算机并不重要)的运行时间如图2-2所示。

输入大小	算法时间			
	1 $O(N^3)$	2 $O(N^2)$	3 $O(N\log N)$	4 $O(N)$
$N=10$	0.000 009	0.000 004	0.000 006	0.000 003
$N=100$	0.002 580	0.000 109	0.000 045	0.000 006
$N=1\,000$	2.281 013	0.010 203	0.000 485	0.000 031
$N=10\,000$	NA	1.232 9	0.005 712	0.000 317
$N=100\,000$	NA	135	0.064 618	0.003 206

图2-2 计算最大子序列和的几种算法的运行时间(秒)

在表中有几个重要的情况值得注意。对于小量的输入，这些算法都在眨眼之间完成，因此如果只是小量输入的情形，那么花费大量的努力去设计聪明的算法恐怕就太不值得了。另一方面，近来对于重写那些不再合理的基于小输入量假设而在五年以前编写的程序确实存在巨大的市场。现在看来，这些程序太慢了，因为它们用的是一些低劣的算法。对于大量的输入，算法4显然是最好的选择(虽然算法3也是可用的)。

其次，表中所给出的时间不包括读入数据所需要的时间。对于算法4，仅仅从磁盘读入数据所用的时间很可能在数量级上比求解上述问题所需要的时间还要大。这是许多有效算法的典型特点。数据的读入一般是个瓶颈；一旦数据读入，问题就会迅速解决。但是，对于低效率的算法情况就不同了，它必然要占用大量的计算机资源。因此只要可能，使得算法足够有效而不至成为问题的瓶颈是非常重要的。

图2-3指出这四种算法运行时间的增长率。尽管该图只包含 N 从10到100的值，但是相对