小结

本章对如何分析程序的复杂性给出一些提示。遗憾的是，它并不是完善的分析指南。简单的程序通常给出简单的分析，但是情况也并不总是如此。作为一个例子，在本书稍后我们将看到一个排序算法(希尔排序，第7章)和一个保持不相交集的算法(第8章)，它们大约都需要20行程序代码。希尔排序(Shellsort)的分析仍然不完善，而不相交集算法分析极其困难，需要许多页错综复杂的计算。不过，我们在这里遇到的大部分的分析都是简单的，它们涉及对循环的计数。

一类有趣的分析是下界分析，我们尚未接触到。在第7章我们将看到这方面的一个例子：证明任何仅通过使用比较来进行排序的算法在最坏的情形下只需要$\Omega(N \log N)$次比较。下界的证明一般是最困难的，因为它们不只适用求解某个问题的一个算法而是适用求解该问题的一类算法。

在本章结束前，我们指出此处描述的某些算法在实际生活中的应用。*gcd*算法和求幂算法应用在密码学中。特别地，400位数字的数自乘至一个大的幂次(通常为另一个400位数字的数)而在每乘一次后只有低400位左右的数字保留下来。由于这种计算需要处理400位数字的数，因此效率显然是非常重要的。求幂运算的直接相乘会需要大约10^{400}次乘法，而上面描述的算法在最坏情形下只需要大约2 600次乘法。

练习

2.1 按增长率排列下列函数：$N, \sqrt{N}, N^{1.5}, N^2, N \log N, N \log \log N, N \log^2 N, N \log(N^2), 2/N, 2^N, 2^{N/2}, 37, N^2 \log N, N^3$。指出哪些函数以相同的增长率增长。

2.2 设$T_1(N) = O(f(N))$和$T_2(N) = O(f(N))$。下列等式哪些成立？

a. $T_1(N) + T_2(N) = O(f(N))$

b. $T_1(N) - T_2(N) = o(f(N))$

c. $\dfrac{T_1(N)}{T_2(N)} = O(1)$

d. $T_1(N) = O(T_2(N))$

2.3 哪个函数增长得更快：$N \log N$，还是$N^{1+\varepsilon/\sqrt{\log N}}(\varepsilon > 0)$

2.4 证明对任意常数k，$\log^k N = o(N)$。

2.5 求两个函数$f(N)$和$g(N)$使得既不$f(N) = O(g(N))$，又不$g(N) = O(f(N))$。

2.6 在最近的一次法庭审理案件中，一位法官因蔑视罪传讯一个城市并命令第一天交纳罚金2美元，以后每天的罚金都要将上一天的罚金数额平方，直到该城市服从该法官的命令为止(即，罚金上升如下：$2, $4, $16, $256, $65 536, …)。

a. 在第N天罚金将是多少？

b. 使罚金达到D美元需要多少天？(大O的答案即可)

2.7 对于下列六个程序片段中的每一个：

a. 给出运行时间分析(使用大O)。

b. 用Java语言编程，并对N的若干具体值给出运行时间。

c. 用实际的运行时间与你所做的分析进行比较。