- 极限是∞：这意味着 $g(N)=o(f(N))$。
- 极限摆动：二者无关(在本书中将不会发生这种情形)。

使用这种方法几乎总能够算出相对增长率，不过有些复杂化。通常，两个函数 $f(N)$ 和 $g(N)$ 间的关系用简单的代数方法就能得到。例如，如果 $f(N)=N\log(N)$ 和 $g(N)=N^{1.5}$，那么为了确定 $f(N)$ 和 $g(N)$ 哪个增长得更快，实际上就是确定 $\log N$ 和 $N^{0.5}$ 哪个增长更快。这与确定 $\log^2 N$ 和 N 哪个增长更快是一样的，而后者是个简单的问题，因为我们已经知道，N 的增长要快于 log 的任意的幂。因此，$g(N)$ 的增长快于 $f(N)$ 的增长。

另外，在风格上还应注意：不要写成 $f(N)\leqslant O(g(N))$，因为定义已经隐含有不等式了。写成 $f(N)\geqslant O(g(N))$ 是错误的，它没有意义。

作为所执行的典型类型分析的例子，考虑在互连网上下载文件的问题。设有初始 3 s 的延迟(来建立连接)，此后下载以 1.5K(B)/s 的速度进行。可以推出，如果文件为 N 个 KB，那么下载时间由公式 $T(N)=N/1.5+3$ 表示。这是一个**线性函数**(linear function)。注意，下载一个 1 500 K 的文件所用时间(1 003 s)近似(但不是精确)地为下载 750 K 文件所用时间(503 s)的两倍。这是典型的线性函数。还要注意，如果连接的速度快两倍，那么两种时间都要减少，但 1 500 K 文件的下载仍然花费大约下载 750 K 文件的时间的两倍。这是线性时间算法的典型特点，这就是为什么我们写 $T(N)=O(N)$ 而忽略常数因子的原因。(虽然使用大 Θ 会更精确，但是一般给出的是大 O 答案。)

还要看到，这种行为不是对所有的算法都成立。对于 1.1 节描述的第一个选择算法，运行时间由执行一次排序所花费的时间来控制。对诸如所提出的冒泡排序这样的简单排序算法，当输入量增加到两倍的时候，则对大量输入的运行时间增加到 4 倍。这是因为这些算法不是线性的，我们将看到，当讨论排序时，普通的排序算法是 $O(N^2)$，或叫做二次的。

2.2 模型

为了在正式的构架中分析算法，我们需要一个计算模型。我们的模型基本上是一台标准的计算机，在机器中指令被顺序地执行。该模型有一个标准的简单指令系统，如加法、乘法、比较和赋值等。但不同于实际计算机情况的是，模型机做任一件简单的工作都恰好花费一个时间单位。为了合理起见，我们将假设模型像一台现代计算机那样有固定大小(比如 32 位)的整数并且不存在如矩阵求逆或排序这种想像的操作，它们显然不能在一个时间单位内完成。我们还假设模型机有无限的内存。

显然，这个模型有些缺点。很明显，在现实生活中不是所有的运算都恰好花费相同的时间。特别在我们的模型中，一次磁盘读入按一次加法记时，虽然加法一般要快几个数量级。还有，由于假设有无限的内存，我们再不用担心缺页中断，而它可能是个实际问题，特别是对一些高效的算法。

2.3 要分析的问题

通常，要分析的最重要的资源就是运行时间。有几个因素影响着程序的运行时间。有些因素(如所使用的编译器和计算机)显然超出了任何理论模型的范畴，因此，虽然它们是重要的，但是我们在这里还是不能考虑它们。剩下的主要因素则是所使用的算法以及对该算法的输入。

典型的情形是，输入的大小是主要的考虑方面。我们定义两个函数 $T_{avg}(N)$ 和 $T_{worst}(N)$，分别为算法对于输入量 N 所花费的平均运行时间和最坏情况的运行时间。显然，$T_{avg}(N)\leqslant T_{worst}(N)$。如果存在多于一个的输入，那么这些函数可以有多于一个的变量。