0.1 记号定义

定义 1： 如果存在正的常数 c 和 n_0 使得，对于任何 $N \geqslant n_0$ 时总有 $T(N) \leqslant cf(N)$，那么就使用记号 $T(N) = O(f(N))$。

定义 2： 如果存在正的常数 c 和 n_0 使得，对于任何 $N \geqslant n_0$ 时总有 $T(N) \geqslant cg(N)$，那么就使用记号 $T(N) = \Omega(g(N))$。

定义 3: 如果我们有关系式 $T(N) = O(h(N))$，并且同时还具有关系式 $T(N) = \Omega(h(N))$，那么我们使用记号 $T(N) = \Theta(h(N))$。

定义 4: 如果我们具有关系 $T(N) = O(p(N))$，但是不具有关系 $T(N) = \Theta(p(N))$，那么我们使用记号 $T(N) = o(p(N))$。

此处的记号定义非常类似于数字的比较，我们可以很简单地将定义 1 类比于数字比较的小于或等于，意即如果我们有关系 $T(N) = O(f(N))$，那么我们可以“认为” $T(N) \leqslant f(N)$，同样地，对于定义 2，我们类比于关系小于或等于；显然定义三就类比于等于，而定义 4 可以类比于小于。注意，此处的出现的 $T(N), f(N), g(N), p(N)$ 等，表明是一些关于 N 的函数，而 N 的取值一般限于整数。在算法分析中，我们会使用这些记号来度量算法之间的增长关系，即比较算法的运算时间的复杂程度。

这里的定义，直观地看来，与序列极限的定义十分类似；事实上，它们分别就是序列极限之比的另一种表述方式。我们来观察这样的极限表达式

$$\lim_{n\to\infty} \frac{T(N)}{f(N)} = r$$

如果 $r = 0$，那么我们可以断言 $T(N) = o(f(N))$，如果 $r = c(0 < c < \infty)$，那么我们可以得到 $T(N) = \Theta(f(N))$，如果 $r = 0$，我们有 $f(N) = o(T(N))$。

0.2 T(N) 函数的意义

在算法分析中，我们不可能精确地估计算法需要执行的时间，这是因为，首先我们无法预计该算法会以什么样的编程语言形式，在什么样的计算机上运行；即便是我们在同一台计算机上进行编译，编译器的发行商甚至版本的不同，也会造成效率不一的情况。所以，精确预计算法的运行时间是不可能的，但是，一个算法可能的运行时间快慢确实是可以量化估计的，但不是以确定时间的形式，我们会代之以其他类似的形式——基本运算次

数——来估计算法的运行效率。$T(N)$ 函数，就是这样一种函数，对于一个输入规摸 N，我们将会得到怎样量级的基本运算次数。

举例说明，如果 $T(N) = \Theta(N^3)$，此时，如果我们的输入规摸是 300，那么算法就需要 N^3 量级的运算次数，但是算法可能不一定就恰好进行 300^3 次基本运算，可能是它的某个常数倍，或者还有更多的出入，但是，当输入规摸 N 增加到非常大时，算法所需的基本运算次数会越来越趋近于 N^3。