## 递归方法的时间效率

在选择合适的增长率函数时，第一步是对算法的主要操作进行计数。对于要评测的迭代示例，处理过程是简单的。这里我们使用更形式化的技术来估算迭代算法的时间需求，并由此来选择正确的增长率函数。

### countDown的时间效率

考虑countDown方法。从一个给定的整数倒数到1，这个问题的长度与所给的整数直接相关。因为使用*n*来表示问题长度，所以我们将countDown中的参数重命名为*n*，以简化我们的讨论。

public static void countDown(int n)

{

System.out.println(n);

if (n > 1)

countDown(n - 1);

} // end countDown

当*n*为1时，countDown显示1。这是基础情形，需要常数级的时间。当*n*>1时，方法执行println语句及进行比较时都需要常数级的时间。另外，它需要时间去解决由递归调用所表示的更小的问题。如果令*t*(*n*)表示countDown(n)的时间需求，则以上讨论的结果可写为

*t*(1) = 1

*t*(*n*) = 1 + *t*(*n* - 1) 对于 *n* > 1

表示*t*(*n*)的方程称为递推关系（recurrence relation），因为函数*t*的定义中又含有自身——即递推。我们需要一个不由自己定义自己的表达式来表示*t*(*n*)。找到这样表达式的一种办法是，挑选一个*n*值，写出*t*(*n*),*t*(*n*-1),等等的方程，直到到达*t*(1)。从这些方程，我们应该能猜出表示*t*(*n*)的合适的表达式。然后所需的就是证明我们是正确的。实际工作比听上去更简单些。

* 求解一个递推关系。

为求解前面关于*t*(*n*)的递推关系，从*n*=4开始。得到下面的方程序列：

*t*(4) = 1 + *t*(3)

*t*(3) = 1 + *t*(2)

*t*(2) = 1 + *t*(1) = 1 + 1 = 2

在*t*(3)的方程中，用2替代t(2)，得到

*t*(3) = 1 + 2 = 3

在*t*(4)的方程中，用3替代*t*(3)，得到

*t*(4) = 1 + 3 = 4

似乎得到

*t*(*n*) = *n* 对于*n* ≥ 1

我们可以从一个较大的*n*值开始，得到同样的结果，这让我们相信这是对的。但我们需要证明这个结果对于任意的*n*≥1都是对的。这不难做。

* 证明*t*(*n*) = *n*。

为证明对任意的*n*≥1有*t*(*n*)=*n*，我们从*t*(*n*)的递推关系开始，因为我们知道下列关系是成立的：

*t*(*n*) = 1 + *t*(*n* - 1) 对于*n* > 1

我们需要替换掉方程右侧的*t*(*n*-1)。如果当*n*>1时，有*t*(*n*-1)=*n*-1，则当*n*>1时下列关系是正确的：

*t*(*n*) = 1 + *n* - 1 = *n*

所以，如果我们能找到整数*k*，满足方程*t*(*k*)=*k*，则下一个整数也将满足它。用类似的推理过程，对于大于*k*的所有整数，方程都是正确的。因为已给条件*t*(1)=1，所以所有大于1的整数都满足方程。这个证明是归纳法证明（proof by induction）的例子。

最后，我们知道countDown的时间需求由函数*t*(*n*)=*n*给出。所以方法是O(*n*)的。