增长率还是很明显的。虽然 $O(N\log N)$算法的图看起来是线性的，但是用直尺的边(或是一张纸)容易验证它并不是直线。虽然 $O(N)$算法的图看似直线，但这只是因为对于小的 N 值其中的常数项大于线性项。图 2-4 显示了对于更大值的性能。该图明显地表明，对于即使是适度大小的输入量低效算法依然是多么的无用。

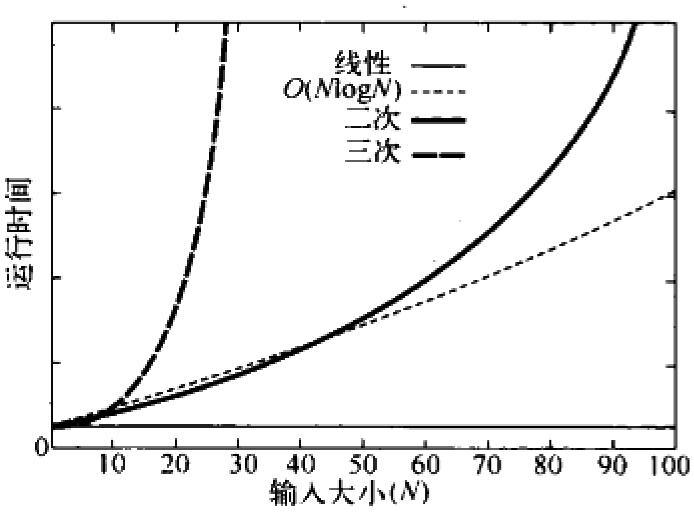

图 2-3 各种计算最大子序列和算法(N 和时间之间)的图

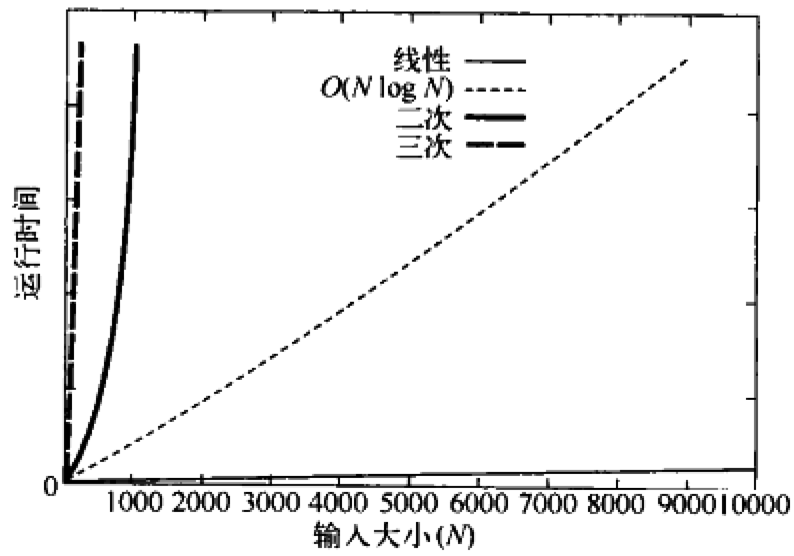

图 2-4 各种计算最大子序列和的算法(N 和时间之间)图

2.4 运行时间计算

有几种方法估计一个程序的运行时间。前面的表是凭经验得到的。如果认为两个程序花费大致相同的时间，要确定哪个程序更快的最好方法很可能就是将它们编码并运行！

一般地，存在几种算法思想，而我们总愿意尽早除去那些不好的算法思想，因此，通常需要分析算法。不仅如此，进行分析的能力常常提供对于设计有效算法的洞察能力。一般说来，分析还能准确地确定瓶颈，这些地方值得仔细编码。

为了简化分析，我们将采纳如下的约定：不存在特定的时间单位。因此，我们抛弃一些前导的常数。我们还将抛弃低阶项，从而要做的就是计算大 O 运行时间。由于大 O 是一个上界，因此我们必须仔细，绝不要低估程序的运行时间。实际上，分析的结果为程序在一定的时间范围内能够终止运行提供了保障。程序可能提前结束，但绝不可能错后。

2.4.1 一个简单的例子

这里是计算 $\sum_{i=1}^{N} i^3$ 的一个简单的程序片段：

```
    public static int sum( int n )
    {
        int partialSum;

1       partialSum = 0;
2       for( int i = 1; i <= n; i++ )
3           partialSum += i * i * i;
4       return partialSum;
    }
```

对这个程序段的分析是简单的。所有的声明均不计时间。第 1 行和第 4 行各占一个时间单元。第 3 行每执行一次占用 4 个时间单元(两次乘法，一次加法和一次赋值)，而执行 N 次共占用 $4N$ 个时间单元。第 2 行在初始化 i、测试 $i \leqslant N$ 和对 i 的自增运算隐含着开销。所有这些的