总开销是初始化1个单元时间，所有的测试为 $N+1$ 个单元时间，而所有的自增运算为 N 个单元时间，共 $2N+2$ 个时间单元。我们忽略调用方法和返回值的开销，得到总量是 $6N+4$ 个时间单元。因此，我们说该方法是 $O(N)$。

如果每次分析一个程序都要演示所有这些工作，那么这项任务很快就会变成不可行的负担。幸运的是，由于我们有了大 O 的结果，因此就存在许多可以采取的捷径并且不影响最后的结果。例如，第3行(每次执行时)显然是 $O(1)$语句，因此精确计算它究竟是2、3还是4个时间单元是愚蠢的；这无关紧要。第1行与 for 循环相比显然是不重要的，所以在这里花费时间也是不明智的。这使我们得到若干一般法则。

2.4.2 一般法则

法则1——for 循环

一个 for 循环的运行时间至多是该 for 循环内部那些语句(包括测试)的运行时间乘以迭代的次数。

法则2——嵌套的 for 循环

从里向外分析这些循环。在一组嵌套循环内部的一条语句总的运行时间为该语句的运行时间乘以该组所有的 for 循环的大小的乘积。

例如，下列程序片段为 $O(N^2)$：

```
for( i = 0; i < n; i++ )
    for( j = 0; j < n; j++ )
        k++;
```

法则3——顺序语句

将各个语句的运行时间求和即可(这意味着，其中的最大值就是所得的运行时间；见2.1节中的法则1(a))。

例如，下面的程序片段先是花费 $O(N)$，接着是 $O(N^2)$，因此总量也是 $O(N^2)$：

```
for( i = 0; i < n; i++ )
    a[ i ] = 0;
for( i = 0; i < n; i++ )
    for( j = 0; j < n; j++ )
        a[ i ] += a[ j ] + i + j;
```

法则4——if/else 语句

对于程序片段

```
if( condition )
    S1
else
    S2
```

一个 if/else 语句的运行时间从不超过判断的运行时间再加上 S1 和 S2 中运行时间长者的总的运行时间。

显然在某些情形下这么估计有些过头，但决不会估计过低。

其他的法则都是显然的，但是，分析的基本策略是从内部(或最深层部分)向外展开工作的。如果有方法调用，那么要首先分析这些调用。如果有递归过程，那么存在几种选择。若递归实际上只是被薄面纱遮住的 for 循环，则分析通常是很简单的。例如，下面的方法实际上就是一个简单的循环从而其运行时间为 $O(N)$：