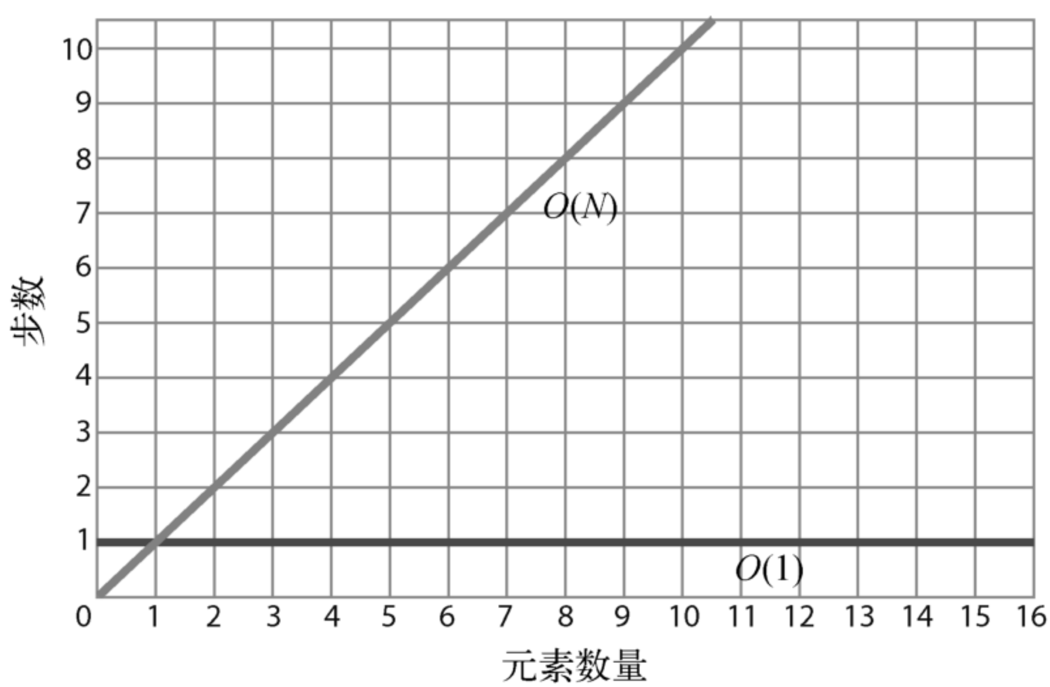
3 大O记法

3.1 大O：数步数

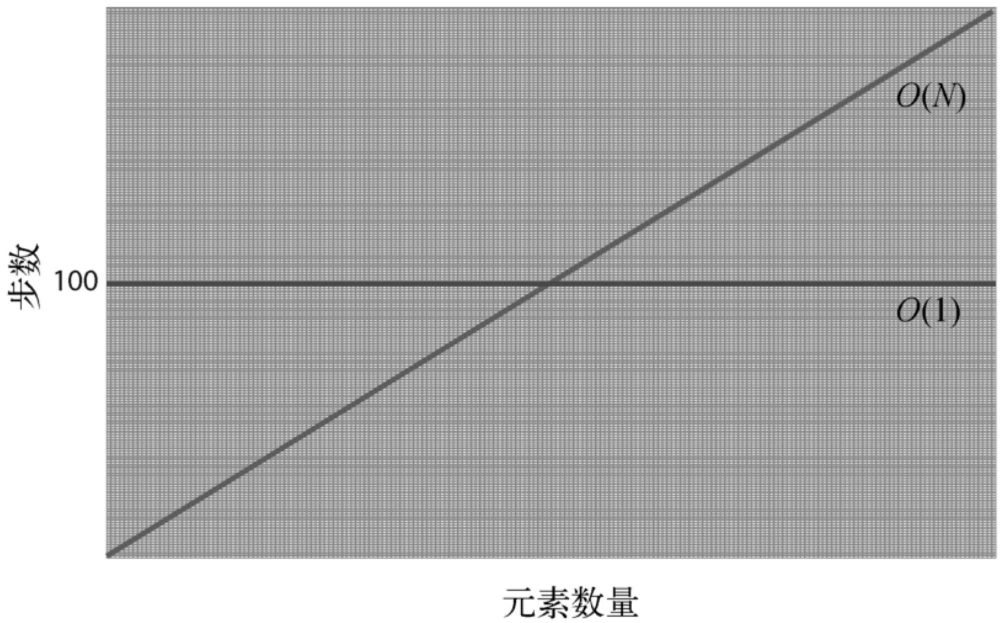
1. 大 O 不关注算法所用的时间，只关注其所用的**步数**
2. 数组不论多大，读取都只需 1 步 : **O(1)**
3. 对于N个元素的数组，线性查找需要花N步：**O(N)**

3.2 常数时间与线性时间

1. 当数据增长时，步数如何变化



2. O(N)也被称为**线性时间**
3. O(1)也被称为**常数时间。**用来表示**所有**数据增长但是步数不变的算法
4. 如果它的步数是恒定的，那么它还是比 O(N)更高效



3.3 同一算法，不同场景

1. 线性查找的最好情况是 O(1)，最坏情况是 O(N)
2. 大 O 记法一般都是指最坏情况

3.4 第三种算法

1. 二分查找 **O(log N)**
2. 归于此类的算法，它们的时间复杂度都叫作**对数时间**

3. 三种时间复杂度的对比

到这里我们所提过的 3 种时间复杂度，按照效率由高到低来排序的话，会是这样：

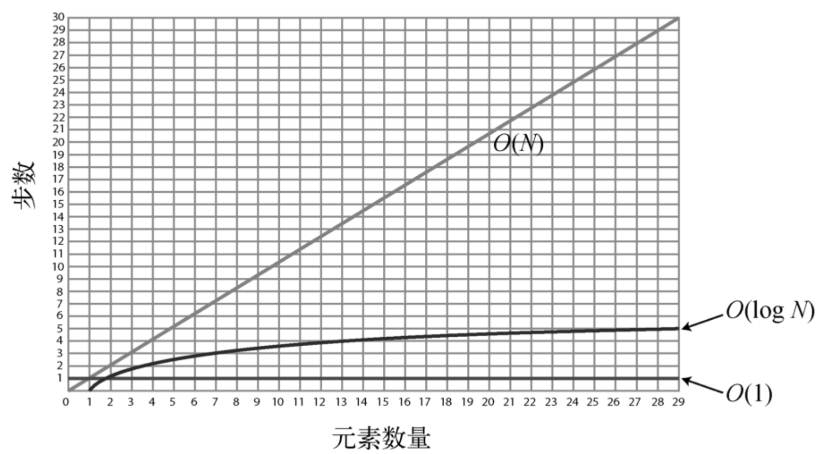
$O(1)$

$O(\log N)$

$O(N)$

下图为它们三者的对比。

注意 $O(\log N)$曲线的微弯，使其效率略差于 $O(1)$，却远胜于 $O(N)$。



3.5 对数

1. 一些对数计算

3.6 解释O(log N)

1. O(log N)算法的步数等于二分数据直至元素剩余 1 个的次数

3.7 实例

1. 一个Python实例-

   在该例子中的问题是print列表中的所有元素，算法是在for循环中使用print - O(N)

   ```
   things = ['apples', 'baboons', 'cribs', 'dulcimers']

   for thing in things:
       print "Here's a thing: %s" % thing
   ```

2. 另一个Python实例

   ```
   print 'Hello world!'
   # O(1)
   ```

3. 代码判断一个数字是否为质数

   ```
   def is_prime(number):
       for i in range(2, number):
           if number % i == 0:
               return False
       return True
   ```

4.