

# 排序（上）：为什么插入排序比冒泡排序更受欢迎？

作者：[someone26671](#)

原文链接：<https://ld246.com/article/1547637519482>

来源网站：[链滴](#)

许可协议：[署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

<p><strong>本文章是从极客时间抄写的，仅仅是看不懂，抄了一遍，分享给大家。<br>© 版权归极客邦科技所有</strong></p>

<hr>

<p>排序对于任何一个程序员来说，可能都不会陌生。你学的第一个算法，可能就是排序。大部分编程语言中，也都提供了排序函数。在平常的项目中，我们也经常会用到排序。排序非常重要，所以我会多一点时间来详细讲一讲经典的排序算法。</p>

<p>排序算法太多了，有很多可能你连名字都没听说过，比如猴子排序、睡眠排序、面条排序等。我讲众多排序算法中的一小撮，也是最经典的、最常用的：冒泡排序、插入排序、选择排序、归并排序、快速排序、计数排序、基数排序、桶排序。我按照时间复杂度把他们分成了三类，分三节课讲解。</p>

<p></p>

<p>在这问题去学习，是最有效的学习方法。所以按照惯例，我还是先给你出一个思考题：<strong>插入排序和冒泡排序的时间复杂度相同，都是  $O(n^2)$ ，在实际的软件开发里，为什么我更倾向于使用插入排序算法而不是冒泡排序算法呢？</strong></p>

<p>你可以思考一两分钟，带着这个问题，我们开始今天的内容！</p>

<p><em><strong>如何分一个“排序算法”？</strong></em></p>

<p>学习排序算法，我们除了学习它的算法原理、代码原理之外，更重要的是要学会如何评价、分析个排序算法。那分析一个排序算法，要从哪几个方面入手呢？</p>

<h2 id="排序算法的执行效率">排序算法的执行效率</h2>

<h2 id="1-最好情况-最坏情况-平均情况时间复杂度">1.最好情况、最坏情况、平均情况时间复杂度</h2>

<p>我们在分析排序算法的时间复杂度时，要分别给出最好情况、最坏情况、平均情况下的时间复杂度。除此之外，你还要说出最好、最坏时间复杂度对应的要排序的原始数据是什么样的。</p>

<p>为什么要区分这三种时间复杂度呢？第一，有些排序算法会区分，为了好对比，所以我们最好都一下区分。第二，对于要排序的数据，有的接近有序，有的完全无序。有序度不同的数据，对于排序执行时间肯定是有影响的，我们要知道排序算法在不同数据下的性能表现。</p>

<h2 id="2-时间复杂度的系数-常数-低阶">2.时间复杂度的系数、常数、低阶</h2>

<p>我们知道，时间复杂度反映的是数据规模  $n$  很大的时候的一个增长趋势，所以它表示的时候会忽略系数、常数、低阶。但是实际的软件开发中，我们排序的可能是 10 个、100 个、1000 个这样规模小的数据，所以，在对同一阶时间复杂度的排序算法性能对比的时候，我们就要把系数、常数、低阶考虑进来。</p>

<h2 id="3-比较次数和交换-或移动-次数">3.比较次数和交换（或移动）次数</h2>

<p>这一节和下一节讲的都是基于比较的排序算法。基于比较的排序算法的执行过程，会涉及两种操作，一种是元素比较大小，另一种是元素交换或移动。所以，如果我们在分析排序算法的执行效率的时候，应该把比较次数和交换（或移动）次数也考虑进去。</p>

<h2 id="排序算法的内存消耗">排序算法的内存消耗</h2>

<p>我们前面讲过，算法的内存消耗可以通过空间复杂度来衡量，排序算法也不例外。不过，针对排序算法的空间复杂度，我们还引入了一个新的概念，<strong>原地排序（Sorted in place）</strong>。原地排序算法，就是特指空间复杂度是  $O(1)$  的排序算法。我们今天讲的三种排序算法，都是原地排序算法。</p>

<h2 id="排序算法的稳定性">排序算法的稳定性</h2>

<p>仅仅用执行效率和内存消耗来衡量排序算法的好还是不够的。针对排序算法，我们还有一个重要度量指标，<strong>稳定性</strong>。这个概念是说，如果待排序的序列中存在值相等的元素，过排序之后，相等元素之间原有的先后顺序不变。</p>

<p>我通过一个例子来解释一下。比如我们有一组数据 2,9,3,4,8,3，按照大小排序之后就是 2,3,3,4,5,8。</p>

<p>这组数据里有两个 3。经过某种排序算法排序之后，如果两个 3 的前后顺序没有改变，那我们把这种排序算法叫作<strong>稳定的排序算法</strong>；如果前后顺序发生变化，那对应的排序算法就叫作<strong>不稳定的排序算法</strong>。</p>

<p>你可能要问了，两个 3 哪个在前，哪个在后有什么关系啊，稳不稳定又有什么关系呢？为什么要察排序算法的稳定性呢？</p>

<p>很多数据结构和算法课程，在讲排序的时候，都是用整数来举例，但真正软件开发中，我们要排

的往往不是单纯的整数，而是一组对象，我们需要按照对象的某个 key 来排序。

比如所，我们现在要给电商交易系统中的“订单”排序。订单有两个属性，一个是下单时间，一个是订单金额。如果我们现在有 10 万条订单数据，我们希望按照金额从小到大对订单数据排序。于金额相同的订单，我们希望按照下单时间从早到晚有序。对于这样一个排序需求，我们怎么做呢

最先想到的方法是：我们先按照金额对订单数据进行排序，然后，再遍历排序之后的订单数据，于每个金额相同的小区间再按照下单时间排序。这种排序思路理解起来不难，但是实现起来会很复杂

借助稳定排序算法，这个问题可以非常简洁地解决。解决思路是这样的：我们先按照下单时间给单排序，主义是按照下单时间，不是金额。排序完成之后，我们用稳定排序算法，按照订单金额重新序。两遍排序之后，我们得到的订单数据就是按照金额从小到大排序，金额相同的订单按照下单时间早到晚排序的。为什么呢？

**稳定排序算法可以保持金额相同的两个对象，在排序之后的前后顺序不变**第一次排序之后，所有的订单按照下单时间从早到晚有序了。在第二次排序中，我们用的是稳定的排算法，所以经过第二次排序之后，相同金额的订单仍然保持下单时间从早到晚有序。



## 冒泡排序 (Bubble Sort)


我们从冒泡排序开始，学习今天的三种排序算法。

冒泡排序值会操作相邻的两个数据。每次冒泡操作都会对相邻的两个元素进行比较，看是否满足小关系要求。如果不满足就让它俩互换。一次冒泡会让至少一个元素移动到它应该在的位置，重复 n，就完成了 n 个数据的排序工作。

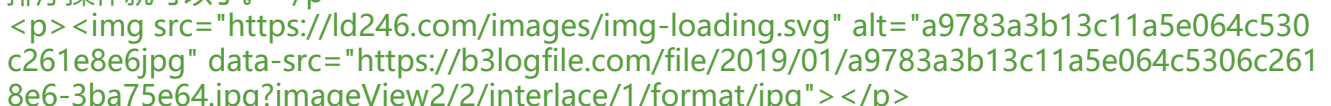
我用一个例子，但你看下冒泡排序的整个过程。我们要对一组数据 4,5,6,4,3,2,1 从小到大进行排。第一次冒泡操作的详细过程就是这样：



可以看出，经过一次冒泡操作之后，6 个元素已经存储在正确的位置上。要想完成所有数据的排，我们只要进行 6 次这样的冒泡操作就行了。



实际上，刚讲的冒泡排序还可以优化。当某次冒泡操作已经没有数据交换时，说明已经达到完全序，不用再继续执行后续的冒泡操作。我这里还有另外一个例子，这里面给 6 个元素排序，只需要 4 排序操作就可以了。



冒泡排序算法的原理比较容易理解，具体的代码我贴到下面，你可以结合着代码来看我前面讲的理。

```
 //冒泡排序，a表示数组，n表示数组大小
public void bubbleSort(int [] a,int n){
    if(n &lt;=1 ) return;
    for(int i = 0;i &lt; n; ++i){
        //提前退出
        泡循环的标志位
        boolean flag = false;
```

```

</span></span><span class="highlight-line"><span class="highlight-cl">
    &lt; n - i - 1; ++j){
</span></span><span class="highlight-line"><span class="highlight-cl">
a[j+1]}){//交换
</span></span><span class="highlight-line"><span class="highlight-cl">
    = a[j];
</span></span><span class="highlight-line"><span class="highlight-cl">
    a[j] = a[
+1];
</span></span><span class="highlight-line"><span class="highlight-cl">
    a[j+1]
tmp;
</span></span><span class="highlight-line"><span class="highlight-cl">
    flag = t
ue; //表示有数据交换
</span></span><span class="highlight-line"><span class="highlight-cl">
    }
</span></span><span class="highlight-line"><span class="highlight-cl">
    }
</span></span><span class="highlight-line"><span class="highlight-cl">
    if(!flag) bre
k; //没有数据交换，提前退出
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl"> }
</span></span></code></pre>

```

<p>现在，结合刚才我分析排序算法的三个方面，我有三个问题要问你。</p>

<p><strong>第一，冒泡排序是原地排序算法吗？</strong></p>

<p>冒泡的过程只涉及相邻数据的交换操作，只需要常量级别的临时空间，所以它的空间复杂度为  $O(1)$ ，是一个原地排序算法。</p>

<p><strong>第二，冒泡排序是稳定的排序算法吗？</strong></p>

<p>在冒泡排序中，只有交换才可以改变两个元素的前后顺序。为了保证冒泡排序算法的稳定性，当相邻的两个元素大小相等的时候，我们不做交换，相同大小的数字在排序前后不会改变顺序，所以冒泡排序是稳定的排序算法。</p>

<p><strong>第三，冒泡排序的时间复杂度是多少？</strong></p>

<p>最好情况下，要排序的数据已经是有序的了，我们只需要进行一次冒泡操作，就可以结束了，所最好情况时间复杂度是  $O(n)$ 。而最坏的情况是，要排序的数据刚好是倒序排列的，我们需要进行  $n$  次冒泡操作，所以最坏情况时间复杂度为  $O(n^2)$ 。</p>

<p><img alt="https://ld246.com/images/img-loading.svg" data-bbox="115 531 300 548"/> <img alt="https://b3logfile.com/file/2019/01/fe107c06da8b290fb78fcce4f774c0f.jpg" data-bbox="115 548 300 565"/> <img alt="https://b3logfile.com/file/2019/01/fe107c06da8b290fb78fcce4f6774c0f1a1ba134.jpg?imageView2/2/interlace/1/format/jpg" data-bbox="115 565 300 582"/></p>

<p>最好、最坏情况下的时间复杂度很容易分析，那平均情况下的时间复杂度是多少呢？我们前面讲，平均时间复杂度就是加权平均期望时间复杂度，分析的时候要结合概率论的知识。</p>

<p>对于包含几个数据的数组，这  $n$  个数据就有  $n!$  种排列方式。不同的排列方式，冒泡排序执行的时间肯定不同。比如我们前面举的那两个例子，其中一个要进行 6 次冒泡，而另一个需要 4 次。如果用概率论方法定量分析平均时间复杂度，设计的数学推理和计算就会很复杂。我这里还有一种思路，通过“有序度”和“逆序度”这两个概念来进行分析。</p>

<p><strong>有序度</strong>是数组中具有有序关系的元素的个数。有序元素对数学表达式表示是这样：</p>

```

<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight-cl">
有序元素对： a[i] &lt;= a[j],如果 i &lt; j.
</span></span></code></pre>

```

<p><img alt="https://ld246.com/images/img-loading.svg" data-bbox="115 747 300 764"/> <img alt="https://b3logfile.com/file/2019/01/a1ef4cc1999d6bd0af08d841ee55220.jpg" data-bbox="115 764 300 781"/> <img alt="https://b3logfile.com/file/2019/01/a1ef4cc1999d6bd0af08d8417ee5520-1bf4cf75.jpg?imageView2/2/interlace/1/format/jpg" data-bbox="115 781 300 798"/></p>

<p>同理，对于一个倒序排序的数组，比如 6,5,4,3,2,1，有序度是 0；对于一个完全有序的数组，比如 1,2,3,4,5,6，有序度就是  $n*(n-1)/2$ ，也就是 15。我们把这种完全有序的的数的有序度叫做<strong>满有序度</strong>。</p>

<p>逆序度的定义正好跟有序度相反（默认从小到大为有序），我想你应该已经想到了。关于逆序度我就不举例子了。你可以对照我讲的有序度的例子自己看下。</p>

```

<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight-cl">

```

cl"> 逆序元素:  $a[i] > a[j]$ , 如果  $i < j$ .

```
</span> </span> </code> </pre>
```

<p>关于这三个概念, 我们还可以得到一个公式: **逆序度 = 满有序度 - 有序度**</strong> 我们排序的过程就是一种增加有序度, 减少逆序度的过程, 最后达到满有序度, 就说明排序完成了。</p>

<p>我还是拿前面的那个冒泡排序的例子来说明。要排序的数组的初始状态是 4,5,6,3,2,1, 其中, 有元素对有(4,5)(4,6)(5,6), 所以有序度是 3。  $n = 6$ , 所以排序完成之后终态的满有序度为  $n*(n-1)/2 = 5$ 。</p>

<p></p>

<p>冒泡排序包含两个操作原子, **比较**和**交换**。每交换一, 有序度就加 1。不管算法怎么改进, 交换次数总是确定的, 即为**逆序度**, 也就是  $n*(n-1)/2 - \text{初始有序度}$ 。此例中就是  $15-3=12$ , 要进行 12 次交换操作。</p>

<p>对于包含  $n$  个数据的数组进行冒泡排序, 平均交换次数是多少呢? 最坏情况下, 初始状态的有序是 0, 所以要进行  $n*(n-1)/2$  次交换。最好情况下, 初始状态的有序度是  $n*(n-1)/2$ , 就不需要进行交换。我们可以取个中间值  $n*(n-1)/4$ , 来表示初始有序度既不是很高也不是很低的平均情况。</p>

<p>换句话说, 平均情况下, 需要  $n*(n-1)/4$  次交换操作, 比较操作肯定要比较操作多, 而复杂度的限是  $O(n^2)$ , 所以平均情况下的时间复杂度就是  $O(n^2)$ 。</p>

<p>这个平均时间复杂度推导过程其实并不严格, 但是很多时候很实用, 毕竟概率论的定量分析太复杂, 不太好用。等我们讲到快排的时候, 我还会再次用到这种“不严格”的方法来分析平均时间复杂度。</p>

## 插入排序 (Insertion Sort)

<p>我们先来看一个问题。一个有序的数组, 我们往里面添加一个新的数据后, 如何继续保持数据有序的? 很简单, 我们只要遍历数组, 找到数据应该插入的位置将其插入即可。</p>

<p></p>

<p>这是一个动态排序的过程, 即动态地往有序集合中添加数据, 我们可以通过这种方法保持集合中数据一直有序。而对于一组静态数据, 我们也可以借鉴上面讲的插入方法, 来进行排序, 于是就有了插入排序算法。</p>

<p>那**插入排序**具体是如何借助上面的思想来实现排序的呢?</p>

<p>首先, 我们将数组中的数据分为两个区间, **已排序区间**和**未排序区间**。初始已排序区间只有一个元素, 就是数组的第一个元素。插入算法的核心思想是取未排序区间中的元素, 在已排序区间中找到合适的插入位置将其插入, 并保证已排序区间数据一直有序。重复这个过程, 知道未排序区间中元素为空, 算法结束。</p>

<p>如图所示, 要排序的数据是 4,5,6,1,3,2, 其中左侧为已排序区间, 右侧是未排序区间。</p>

<p></p>

<p>插入排序也包含两种操作, 一种是**元素的比较**, 一种是**元素的移动**。当我们需要将一个数据  $a$  插入到已排序区间是, 需要拿  $a$  与已排序区间的元素一次比较, 找到合适的插入位置。找到插入点之后, 我们还需要将插入点之后的元素顺序往后移动一位, 这样才能腾出位置给元素  $a$  插入。</p>

<p>对于不同的查找插入点方法 (从头到尾、从尾到头), 元素的比较次数是有区别的。但对于一个定的初始序列, 移动操作的次数总是固定的, 就等于逆序度。</p>

<p>为什么说移动次数就等于逆序度呢? 我拿刚才的例子画了一个图标, 你一看就明白了。满有序度  $n*(n-1)/2=15$ , 初始序列的有序度是 5, 所以逆序度是 10。插入排序中, 数据移动的个数总和也等于  $10=3+3+4$ 。</p>

<p></p>

<p>插入排序的原理也很简单吗? 我也将代码实现贴在这里, 你可以结合着代码再看下。</p>

```
<pre> <code class="highlight-chroma"> <span class="highlight-line"> <span class="highlight
```

```

cl"> // 插入排序, a表示数组, n表示数组大小
</span></span><span class="highlight-line"><span class="highlight-cl"> public void insert
onSort(int[] a,int n){
</span></span><span class="highlight-line"><span class="highlight-cl">     if(n &lt;= 1) ret
rn;
</span></span><span class="highlight-line"><span class="highlight-cl">     for(int i = 1; i &
t; n;++i){
</span></span><span class="highlight-line"><span class="highlight-cl">         int value = a[
];
</span></span><span class="highlight-line"><span class="highlight-cl">         int j = i - 1;
</span></span><span class="highlight-line"><span class="highlight-cl">         //查找插入
位置
</span></span><span class="highlight-line"><span class="highlight-cl">         for(j &gt;= 0
--j){
</span></span><span class="highlight-line"><span class="highlight-cl">             if(a[j] &gt;
value){
</span></span><span class="highlight-line"><span class="highlight-cl">                 a[j+1] =
a[j]; //数据移动
</span></span><span class="highlight-line"><span class="highlight-cl">             }else{
</span></span><span class="highlight-line"><span class="highlight-cl">                 break;
</span></span><span class="highlight-line"><span class="highlight-cl">             }
</span></span><span class="highlight-line"><span class="highlight-cl">             a[j+1] =val
e; //插入数据
</span></span><span class="highlight-line"><span class="highlight-cl">         }
</span></span><span class="highlight-line"><span class="highlight-cl">     }
</span></span></code></pre>

```

<p>现在, 我们来看点稍微复杂的东西。我这里还是有三个问题要问你。</p>

<p>第一, 插入排序是原地排序算法吗? </p>

<p>从实现过程可以很明显地看出, 插入排序算法的运行并不需要额外的存储空间, 所以空间复杂度  $O(1)$ , 也就是说, 这是一个原地排序算法。</p>

<p>第二, 插入排序是稳定的排序算法吗? </p>

<p>在插入排序中, 对于值相同的元素, 我们可以选择将后面出现的元素, 插入到前面出现元素的后面, 这样就可以保持原有的前后顺序不变, 所以插入排序是稳定的排序算法。</p>

<p>第三, 插入排序的时间复杂度是多少? </p>

<p>如果要排序的数据已经是有序的, 我们并不需要搬移任何数据。如果我们从尾到头在有序数据组里面查找插入位置, 每次只需要比较一个数据就能确定插入的位置。所以这种情况下, 最好是时间复杂度为  $O(n)$ 。注意, 这里是**从尾到头遍历已经有序的数据**。</p>

<p>如果数组是倒序的, 每次插入都相当于在数组的第一个位置插入新的数据, 所以需要移动大量的数据, 所以最坏情况时间复杂度为  $O(n^2)$ 。</p>

<p>还记得我们在数组中插入一个数据的平均时间复杂度是多少吗? 没错, 是  $O(n)$ 。所以, 对于插入排序来说, 每次插入操作都相当于在数组中插入一个数据, 循环执行  $n$  次插入操作, 所以平均时间复杂度为  $O(n^2)$ 。</p>

## 选择排序 (Selection Sort) </h2>

<p>选择排序算法的实现思路有点类似插入排序, 也分已排序区间和未排序区间。但是选择排序每次会从未排序区间找到最小的元素, 将其放到已排序区间的末尾。</p>

<p></p>

<p>照例, 也有是哪个问题需要你思考, 不过前面两种排序算法我已经分析得很详细了, 这里就直接布答案了。</p>

<p>首先, 选择排序空间复杂度为  $O(1)$ , 是一种原地排序算法。选择排序的最好情况时间复杂度、坏情况和平均情况时间复杂度都为  $O(n^2)$ 。你可以自己来分析看看。</p>

<p>那选择排序是稳定的排序算法吗？这个问题我着重开说一下。</p>

<p>答案是否定的，选择排序是一种不稳定的排序算法。从我前面的那张图中，你可以看出来，选择排序每次都要找剩余未排序元素中的最小值，并和前面的元素交换位置，这样破坏了稳定性。</p>

<p>比如 5,8,5,2,9 这样一组数据，使用选择排序算法来排序的话，第一次找到最小元素 2，与第一个交换位置，那第一个 5 和中间的 5 顺序就变了，所以就不稳定了。正是因此，相对于冒泡排序和插入排序，选择排序就稍微逊色了。</p>

<h2 id="解答开篇">解答开篇</h2>

<p>基本知识都讲完了，我们来看开篇的问题：冒泡排序和插入排序的时间复杂度都是  $O(n^2)$ ，都是原地排序算法，为什么插入排序要比冒泡排序更受欢迎的？</p>

<p>我们前面分析冒泡排序和插入排序的时候讲到，冒泡排序不管怎么优化，元素交换的次数是一个定值，是原始数据的逆序度。插入排序是同样的，不管怎么优化，元素移动的次数也等于原始数据的序度。</p>

<p>但是，从代码实现上来看，冒泡排序的数据交换要比插入排序的数据移动要复杂，冒泡排序需要 3 个赋值操作，而插入排序只需要 1 个。我们来看这段操作：</p>

```
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight-cl"> //冒泡排序中数据的交换操作
</span></span><span class="highlight-line"><span class="highlight-cl"> if(a[j] > a[j+1]){
//交换
</span></span><span class="highlight-line"><span class="highlight-cl">     int tmp = a[j];
</span></span><span class="highlight-line"><span class="highlight-cl">     a[j] = a[j+1];
</span></span><span class="highlight-line"><span class="highlight-cl">     a[j+1] = tmp;
</span></span><span class="highlight-line"><span class="highlight-cl">     flag = true;
</span></span><span class="highlight-line"><span class="highlight-cl"> }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl"> //插入排序中数据
移动操作
</span></span><span class="highlight-line"><span class="highlight-cl"> if(a[j] > value){
</span></span><span class="highlight-line"><span class="highlight-cl">     a[j+1] = a[j]; /
数据移动
</span></span><span class="highlight-line"><span class="highlight-cl"> }else{
</span></span><span class="highlight-line"><span class="highlight-cl">     break;
</span></span><span class="highlight-line"><span class="highlight-cl"> }
</span></span></code></pre>
```

<p>我们把执行一个赋值语句的时间粗略地计为单位时间 (unit\_time)，然后分别用冒泡排序和插入排序对同一个逆序度是 K 的数组进行排序。用冒泡排序，需要 K 次交换操作，每次需要 3 个赋值语句，所以交换操作总耗时就是  $3 * K$  单位时间。而插入排序中数据移动只需要 K 个但闻时间。</p>

<p>这个只是我们非常理论的分析，为了试验，针对上面的冒泡排序和插入排序的 Java 代码，我写一个性能对比测试程序，随机生成 10000 个数组，每个数组中包含 200 个数据，然后在我的机器上别用冒泡和插入排序算法来排序，冒泡排序算法大约 700ms 才能执行完成，而插入排序只需要 100m 左右就能搞定！</p>

<p>所以，虽然冒泡排序和插入排序在时间复杂度上是一样的，都是  $O(n^2)$ ，但是如我们希望把性能优化做到极致，那肯定首选插入排序。插入排序的算法思路也很大的优化空间，我们是讲了最基础的一种。如果你对插入排序的优化感兴趣，可以自行学习一下

<h2 id="内容小结">内容小结</h2>

<p>要想分析、评价一个排序算法，需要从执行效率、内存消耗和稳定性三个方面来看。因此，这一，我带你分析了三种时间复杂度是  $O(n^2)$  的排序算法，冒泡排序、插入排序、选择排。你需要重点掌握的是他们的分析方法。</p>

<p></p>

<p>这三种时间复杂度为  $O(n^2)$  的排序算法中，冒泡排序、选择排序，可能就纯粹停

在理论的层面了，学习的目的也只是为了开拓思维，实际开发中应用并不多，但是插入排序还是挺有的。后面讲排序优化的时候，我会讲到，有些编程语言中的排序函数的实现原理会用到插入排序算法

今天讲的这三种排序算法，实现代码都非常简单，对于小规模数据的排序，用起来非常搞笑。但大规模数据排序的时候，这个时间复杂度还是稍微有点高，所以我们更倾向于用下一节要讲的时间复杂度为  $O(n \log n)$  的排序算法

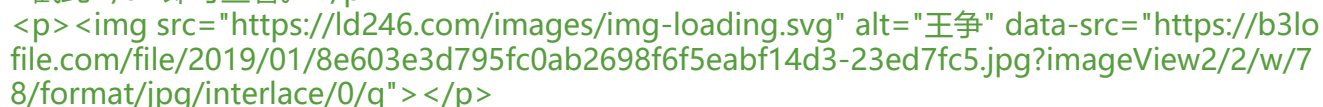
## 课后思考

我们讲过，特定算法是依赖特定的数据结构的。我们今天讲的几种排序算法，都是基于数组实现的。如果数据存储存储在链表中，这三种排序算法还能工作吗？如果能，那相应的时间、空间复杂度又是多少呢？

欢迎留言和我分享，我会第一时间给你反馈。

---

我已将本节内容相关的详细代码更新到 GitHub，[戳此](https://ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2Fwangzheng0822%2Falgo)即可查看。

王争