**排序总结**

**排序算法稳定性**

假设在数列中存在a[i]=a[j]，若在排序之前，a[i]在a[j]前面；并且排序之后，a[i]仍然在a[j]前面。则这个排序算法是稳定的！



**一、冒泡排序**

**基本思想：**

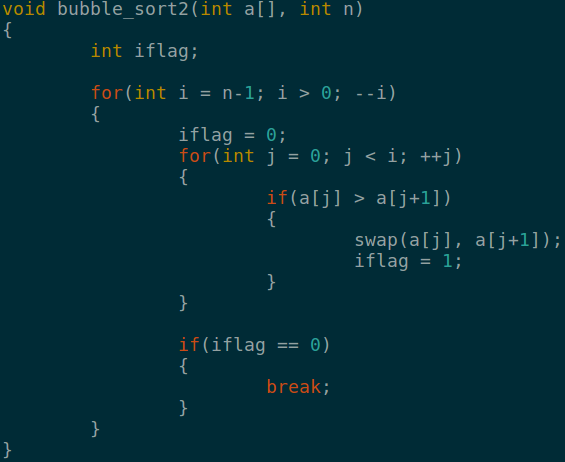
遍历若干次要排序的数列，每次遍历时，它都会从前往后依次的比较相邻两个数的大小； 如果前者比后者大，则交换它们的位置。这样，一次遍历后，最大的元素就在数列的末尾！

添加一个标记，如果一趟遍历中发生了交换，标记置为true，否则为false。如果某一趟遍历中没发生交换，就说明排序已经完成！

**时间复杂度：**0(N2)

**算法稳定性：**稳定

Screenshot from 2018-11-06 10-00-18



**二、插入排序**

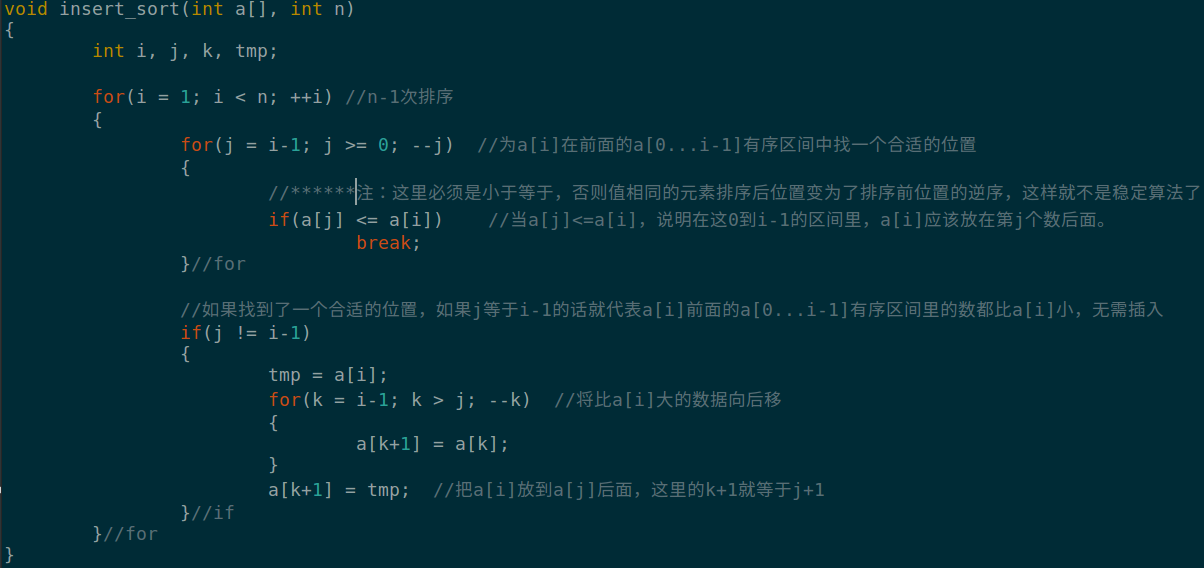
**1、直接插入排序**

**基本思想：**

把n个待排序的元素看成为一个有序表和一个无序表。开始时有序表中只包含1个元素，无序表中包含有n-1个元素，排序过程中每次从无序表中取出第一个元素，将它插入到有序表中的适当位置，使之成为新的有序表，重复n-1次可完成排序过程。

**时间复杂度：**O(N2)

**算法稳定性：**稳定



算法解释：从数列中第2个数开始a2，直到最后一个数，总共n-1次，每次在ai前面的i-1个数中从第i-1个数开始找，找到第一个aj（0<j<i）比ai小的数，一旦找到，把j+1到i-1这些个数向后移一个，然后把ai放到j+1的位置。

**注：**在查找那个a[j]的时候必须用的是小于等于！否则值相同的元素排序后位置变为了排序前位置的逆序，这样就不是稳定算法了

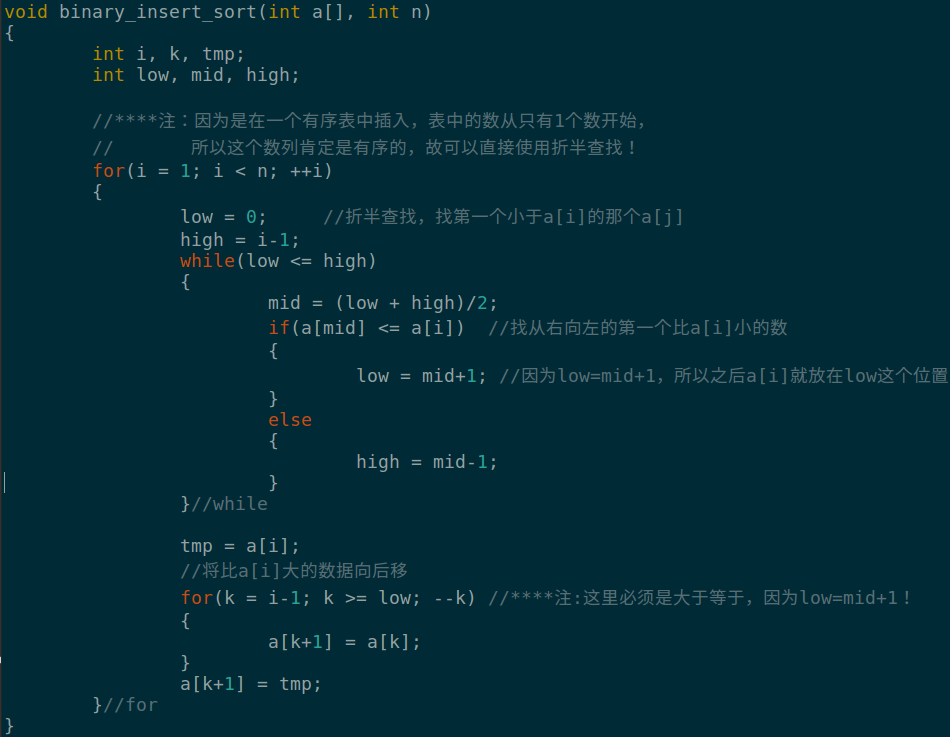
**2、折半插入排序**

**基本思想：**

把n个待排序的元素看成为一个有序表和一个无序表。开始时有序表中只包含1个元素，无序表中包含有n-1个元素，排序过程中每次从无序表中取出第一个元素，将它插入到有序表中的适当位置（查找这个位置的方法是采用折半查找的方法，即在a[0...i-1]这个有序区间里通过折半查找来满足a[j]<a[i]的那个a[j]），使之成为新的有序表，重复n-1次可完成排序过程。

当待排序序列中的记录数量n很大时，折半插入排序进行的关键字间的比较次数比直接插入排序少！

**时间复杂度：**O(N2)



1. **希尔排序(shell)**

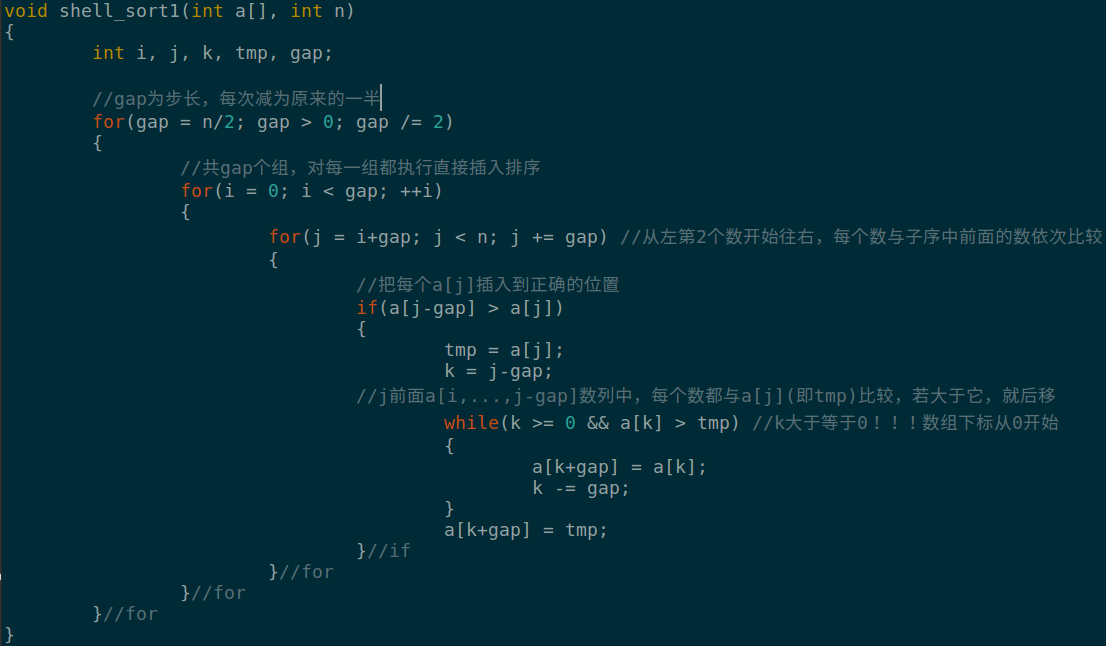
希尔排序(Shell Sort)是插入排序的一种，它是针对直接插入排序算法的改进，该方法又称缩小增量排序。希尔排序实质上是一种分组插入方法

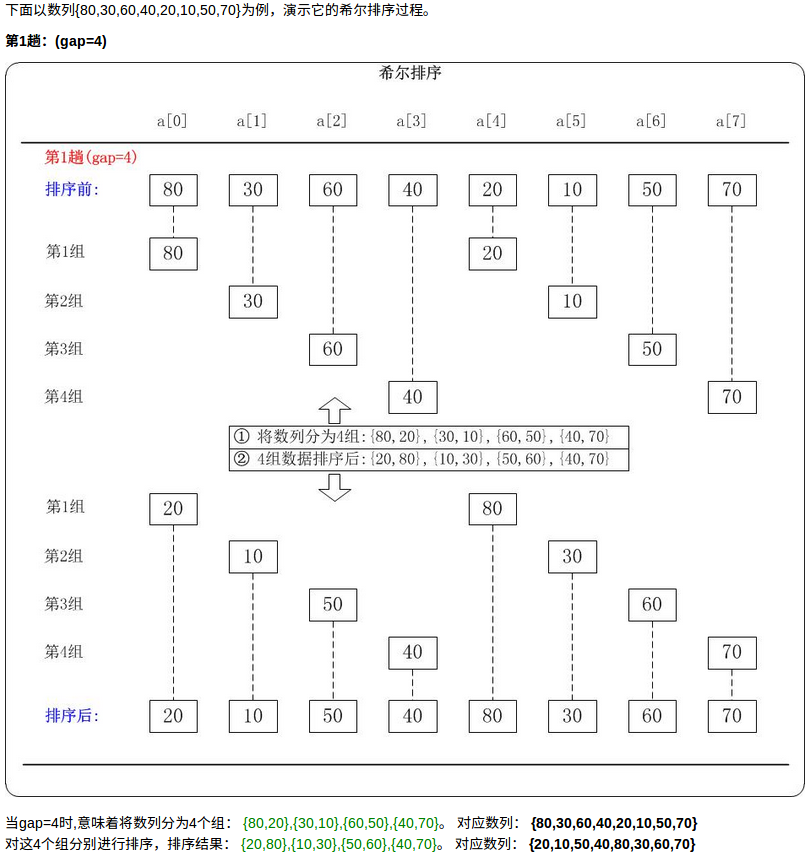
**基本思想：**

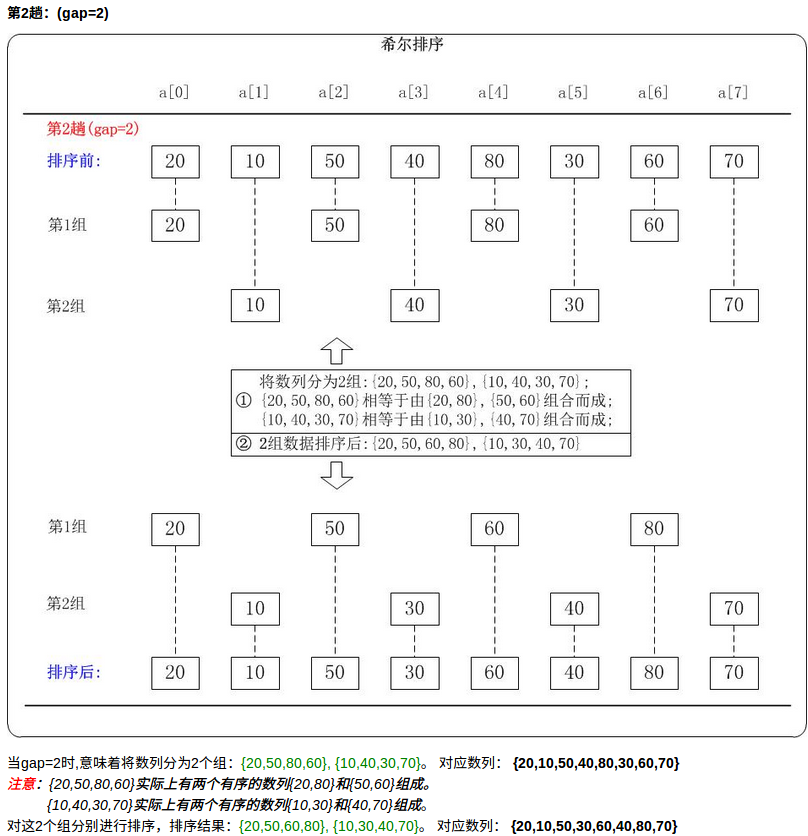
对于n个待排序的数列，取一个小于n的整数gap(gap被称为步长)将待排序元素分成若干个组子序列，所有距离为gap的倍数的记录放在同一个组中。然后，对各组内的元素进行直接插入排序。这一趟排序完成之后，每一个组的元素都是有序的。然后减小gap的值，并重复执行上述的分组和排序。重复这样的操作，当gap=1时，整个数列就是有序的。

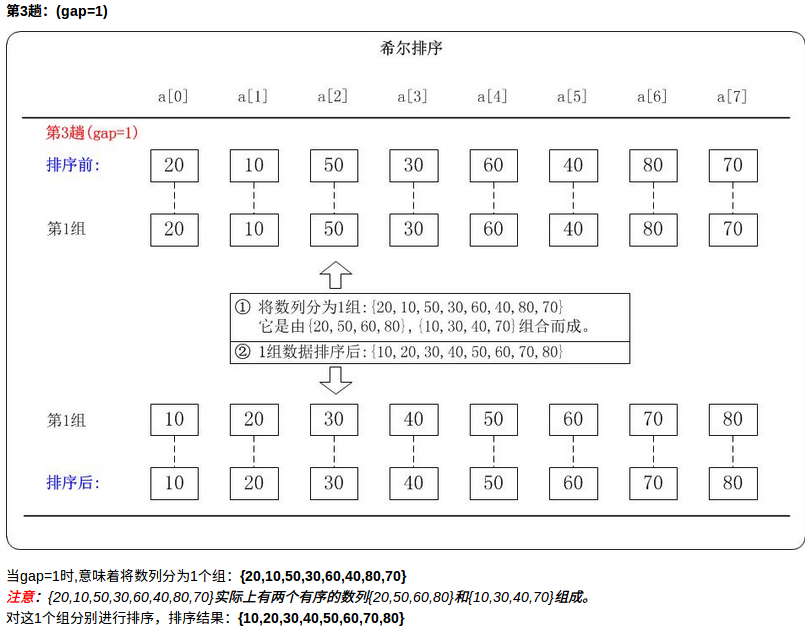
**时间复杂度：**希尔排序的时间复杂度与增量(即，步长gap)的选取有关。例如，当增量为1 时，希尔排序退化成了直接插入排序，此时的时间复杂度为O(N²)，而Hibbard 增量的希尔排序的时间复杂度为O(N3/2)。

**算法稳定性：**不稳定，对于相同的两个数，可能由于分在不同的组中而导致它们的顺序发生 变化。





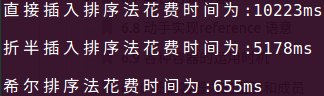




**总结：**

当数列比较大时，这三种插入排序的效率差别还是比较大的，最快的是希尔排序，其次是折半插入排序，最后是直接插入排序。

写了个测试的小程序，设定数列含有1000个随机数，用这三种插入排序算法分别进行排序，所用时间如下所示：(可能有点误差，随机数不一样)



**三、快速排序** (目前是最好的一种内部排序方法)

**基本思想：**

1. 先从数列中取出一个数作为基准数。
2. 分区过程，将比这个数大的数全放到它的右边，小于或等于它的数全放到它的左边。
3. 再对左右区间重复第二步，直到各区间只有一个数。

**快速排序：挖坑填数+分治法**

下面看一个实例：

以一个数组作为示例，取区间第一个数为基准数。

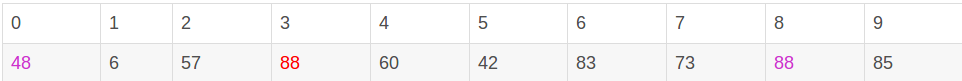


初始时，i=0；j=9；X=a[i]=72

由于已经将a[0]中的数据保存到X中，可以理解成在数组a[0]上挖了个坑，可以将其它数据填充到这来。

从j开始向前找一个比X小或等于X的数。当j=8，符合条件，将a[8]挖出再填到上一个坑a[0]中。a[0]=a[8]；i++；这样一个坑a[0]就被搞定了，但又形成了一个新坑a[8]，这怎么办了？简单，再找数字来填a[8]这个坑。这次从i开始向后找一个大于X的数，当i=3，符合条件，将a[3]挖出再填到上一个坑中a[8]=a[3]; j--。

数组变为：



i=3；j=7；X=72

再重复上面的步骤，**先从后向前找，再从前向后找。**

从j开始向前找，当j=5，符合条件，将a[5]挖出填到上一个坑中，a[3]=a[5]; i++;

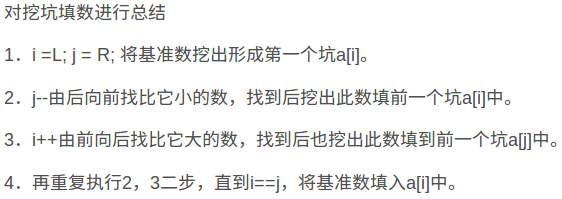
从i开始向后找，当i=5时，由于i==j 退出。

此时，i=j=5，而a[5]刚好又是上次挖的坑，因此将X填入a[5]。

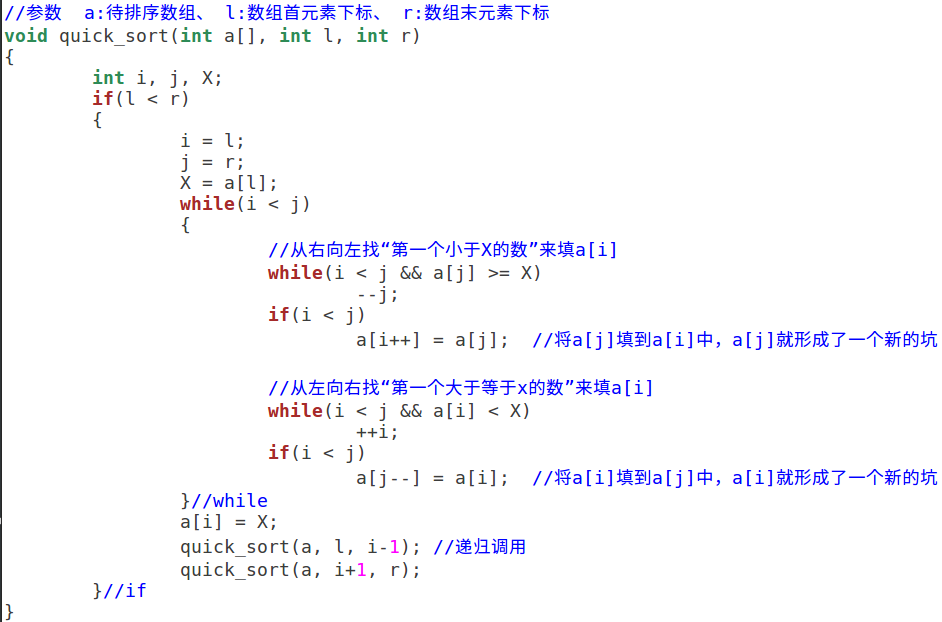
数组变为：



可以**看出a[5]前面的数组都小于它，a[5]后面的数字都大于它。**因此再对a[0...4]和a[6...9]这两个子区间**重复**上述步骤就可以了。



**算法如下：**



**时间复杂度：**

快速排序的时间复杂度在最坏情况下是O(N2)，平均的时间复杂度是O(N\*lgN)。

这句话很好理解：假设被排序的数列中有N个数。遍历一次的时间复杂度是O(N)，需要遍历多少次呐？至少lg(N+1)次，最多N次。

1. 为什么最少是lg(N+1)次？

快速排序是采用的分治法进行遍历的，我们将它看作一棵二叉树，它需要遍历的次数就是二叉树的深度，而根据完全二叉树的定义，它的深度至少是lg(N+1)。因此，快速排序的遍历次数最少是lg(N+1)次。

1. 为什么最多是N次？

这个应该非常简单，还是将快速排序看作一棵二叉树，它的深度最大是N。因此快速排序的遍历次数最多是N次。

**算法稳定性：不**稳定

**四、选择排序**

**1、简单选择排序**

**基本思想：**

首先在未排序的数列中找到最小(or最大)元素，然后将其存放到数列的起始位置；接着，再从剩余未排序的元素中继续寻找最小(or最大)元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。

**时间复杂度：**O(N2)

**算法稳定性：**不稳定。举个例子，序列5 8 5 2 9，我们知道第一遍选择第1个元素5会和2交换，那么原序列中2个5的相对前后顺序就被破坏了，所以选择排序不是一个稳定的排序算法。

