目录

[排序概述： 1](#_Toc514337734)

[冒泡排序： 2](#_Toc514337735)

[插入排序： 2](#_Toc514337736)

[选择排序： 2](#_Toc514337737)

[Shell排序——递减增量排序 3](#_Toc514337738)

[快速排序 6](#_Toc514337739)

## 排序概述：

* 冒泡排序：——多轮，每次两两比较
* 插入排序——假设数据分为两部分，前面的部分默认是有序的，将后面的序列每次选择一个插入前面的序列。只有插入的方法可以使用二分或者是简单的比较（交换），或者是移动（比如说我找到一个位置之后，将后面的序列整体的向右进行移动）。
* 简单选择排序——每次都选择当前序列中最小的元素，和第i的元素进行位置交换。相对于冒泡来说，略有提升。
* 希尔排序：是快排和插入排序的一个结合，其实就是不断的分组，分组的组数越来越多，组内的元素越来越少，（一个循环是不断的缩小组件间距，直到每个组里都是一个元素）。另外一点就是：对于当前的间距，按照一个固定的次序进行一个距离的缩减。【实现比较复杂】
* 递归的归并排序——一开始是N个序列，每个序列中的元素都是一个，两两归并，得到一个长度为N的序列为止。【2路归并排序】
* 非递归的归并排序——不适用递归方式
* 堆排序——首先明确堆：堆是完全二叉树并且——每个节点的值都大于孩子的节点的值是大顶堆，然后节点的值都是小于自己的孩子的堆是小顶堆。

具体的方式：将一个待排序的序列构造为一个大顶堆，序列的最大值就是堆顶。将堆顶移走，剩下的东西就是一个新的堆，得到上面序列的第二大值。

* 快排：【可以递归】

将要排序的序列分成两部分，一部分的所有数据都比第二个部分要小。然后按照这个方法对于两部分进行快速排序。整个序列变成一个有序的序列。



* 桶排序——

## 冒泡排序：

每次从左到右边，比较并且交换顺序，把大的交换到后面，每次都能排好一个单位。每次可以确保将前M个元素的最大值移动到最右边。

由于我们要重复执行n次冒泡，每次冒泡要执行n次比较。

## 插入排序：

将序列看做是两个序列，然后假设前面的序列已经排好序了，当然一开始就是一个元素，肯定是排好序了。

然后从后面序列中挑出来一个元素，插入前面的序列中。（当然插入的位置我觉得可以使用二分查找加速，毕竟已经有序）。【重点：前面的序列插入后仍然有序】【浪费的时间——重点在元素的比较。】

两种比较位置的方法——逐个比较位置，并且进行交换。——或者是：二分查找到对应的索引位置，然后移动该索引之后的元素到指定的位置。

## 选择排序：

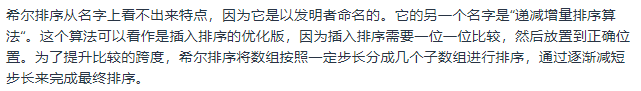
选择当前序列中的最大元素放在第一个位置，并且不断减少队列长度（可以递归）

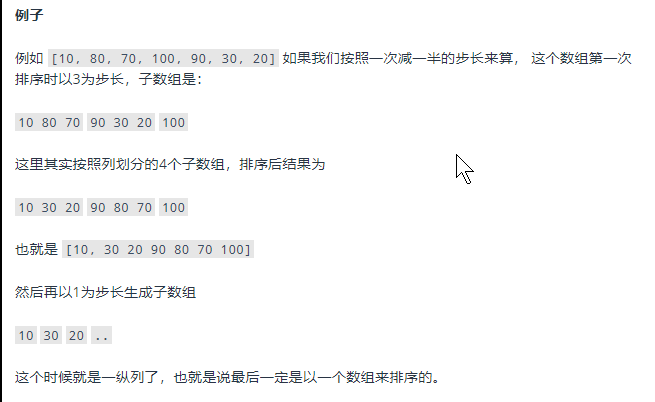
## Shell排序——递减增量排序

关键的一点——Shell增量要选择多少

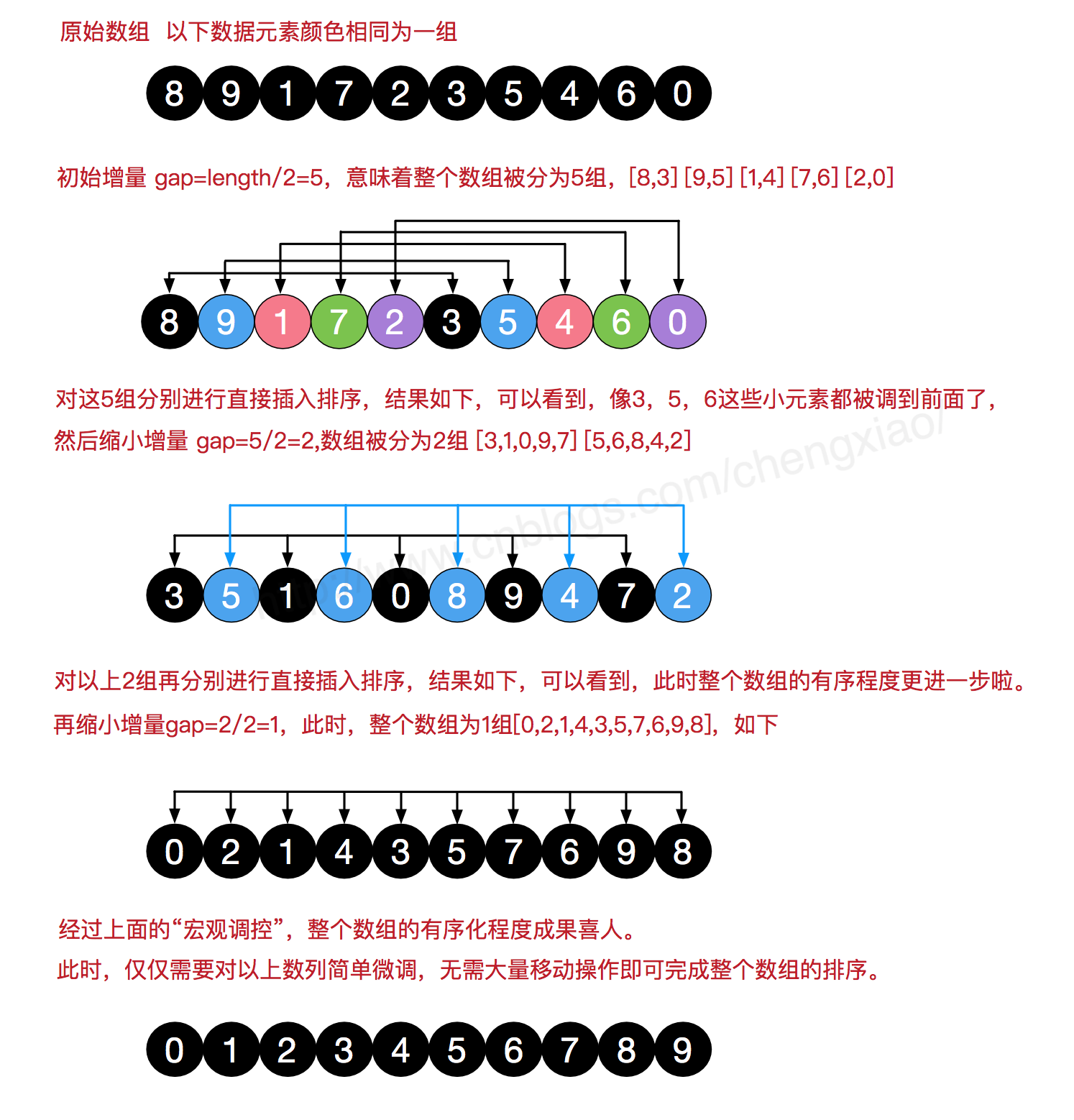
确定了增量之后，就是不断的改变步长，不断的按照步长分组，在组内进行一个直接插入排序。

插入排序是需要一位一位的进行比较，然后shell排序，是按照一定的步长，将数组分成若干个子数组进行操作。 比如这次的步长为5，那就将5 10 15 20 25这几个元素进行排序。然后是1 6 11 16 21 26，然后就是缩短步长，1 5 9 13 17这样的排序。最后缩短到步长为1的情况。【似乎是不对的】





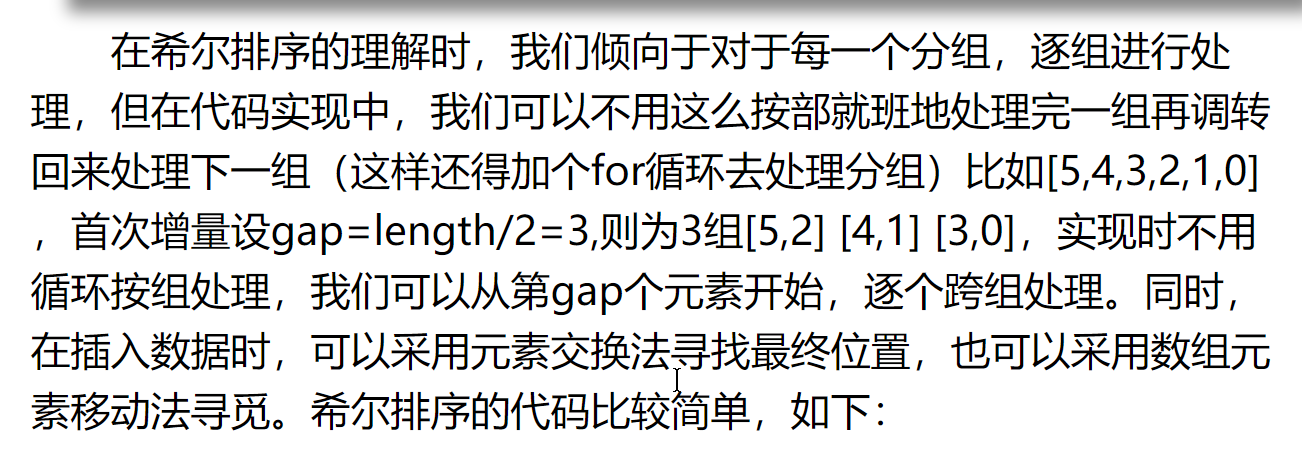
还不是很懂诶。。。



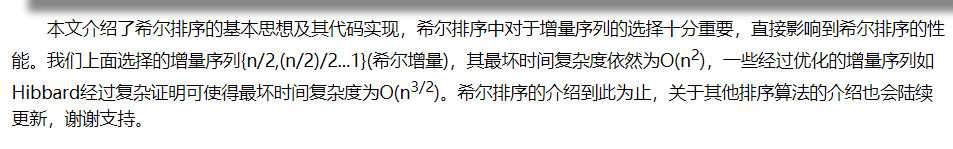
希尔排序是第一个突破了N2的时间复杂度的排序，然后通过使用宏观上的有序和微观上的微调。对于一个原来需要调换N-1次才能到头边的数据，shell排序大大减少了排序移动的次数。

然后关于希尔增量也有很多的不同的版本，但是大家推荐的都是用1/2进行递增的方式。

我记得自己原来看过，希尔排序是快速排序和插入排序的一个综合，也就是说，首先进行分组，然后进行一个插入排序。



也就是从i位置开始，i位置的元素的比较情况是和i-gap的位置，甚至是i-gap-gap位置的元素进行比较的。



Emm怎么说呢，就是从你把i赋值为gap之后，让从i开始递增，然后呢，一开始这个序列里面，只有一个元素，然后等i到了i+gap的时候，这个逻辑上的序列里面就是两个元素了。这两个元素进行插入排序，然后关于插入排序也是有两种方法，一个是说，首先确定元素的位置，然后进行移动，一个是说，只要不满足顺序，就进行交换。

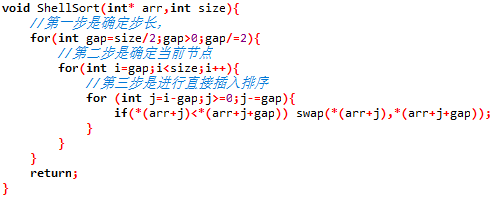
其实二者的效果是一样的，在你进行完了这两个元素的组的比较之后，等到i到了i+gap+gap的地步，这个祖里面就是三个元素了，这三个元素进行插入排序。

一次类推。

理论上应该进行一个组一个组的直接插入排序，但是经过这样的操作也是没有差别的。

反倒是一个组一个组的进行比较，需要增加for循环。

最重要的还是Shell增量，这一点是核心。也是要很多的数学证明的。Shell增量会极大的影响排序·的效率和结果。



## 快速排序

**举个栗子：对5,3,8,6,4这个无序序列进行快速排序，思路是右指针找比基准数小的，左指针找比基准数大的，交换之。**

**5,3,8,6,4 用5作为比较的基准，最终会把5小的移动到5的左边，比5大的移动到5的右边。**

**5,3,8,6,4 首先设置i,j两个指针分别指向两端，j指针先扫描（思考一下为什么？）4比5小停止。然后i扫描，8比5大停止。交换i,j指向的的元素。**

**5,3,4,6,8 然后j指针再扫描，这时j扫描4时两指针相遇。停止。然后交换4和基准数。**

**4,3,5,6,8 一次划分后达到了左边比5小，右边比5大的目的。之后对左右子序列递归排序，最终得到有序序列。当左指针等于右指针的时候，交换这个点位和轴位的位置。**

**上面留下来了一个问题为什么一定要j指针先动呢？首先这也不是绝对的，这取决于基准数的位置，因为在最后两个指针相遇的时候，要交换基准数到相遇的位置。一般选取第一个数作为基准数，那么就是在左边，所以最后相遇的数要和基准数交换，那么相遇的数一定要比基准数小。所以j指针先移动才能先找到比基准数小的数。**

**快速排序是不稳定的，其时间平均时间复杂度是O(nlgn)。**

**总结快速排序的思想：冒泡+二分+递归分治。**

**“轴在中间，两边往中间走”版本**

**核心就是petition的划分，也就是如何选择一个轴，然后将小于轴的放在左边，将大于轴的放在右边。**

**左指针先走，走到中间的那个petition为止，然后右指针，如果出现了小于轴的数字，交换轴和右指针指向的数字。然后左指针继续走。左指针如果发现了比轴大的，也是和轴进行交换。**

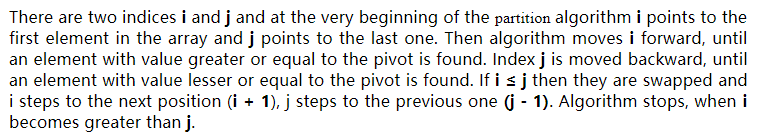
**“算法导论”版本——**

Quicksort主要包含以下3步：

1. 从数组中取出一个元素，叫做主元（pivot）
2. 重排序数组，使得所有小于pivot的元素在它前面，所有大于pivot的元素在它后面，等于pivot的元素放在哪面都行。这样的划分以后，pivot的位置已经排好了。这个过程叫做partition操作
3. 递归地应用步骤2到小于pivot的子数组和大于pivot的子数组

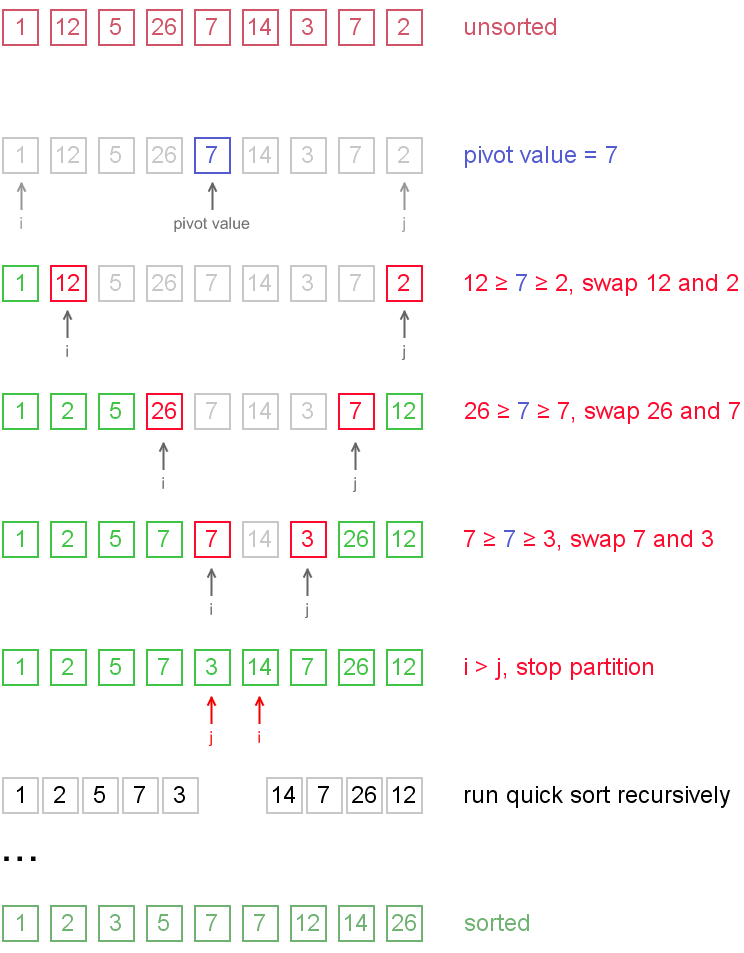
**网络上的版本——**

**我们一方面要实现，小于的在左边，大于的在右边，一方面又要减少空间的使用，所以最佳的方式就是交换proxy的位置。**



**注意：上述的话中，是交换i和j的位置。交换的条件是：i<=j,然后才会发生交换。交换的之后，i++，而j--。算法终止的条件是，i>=j。在算法结束之后，所有比proxy小的，都在i的左边，所有比proxy大的，都在j的右边。**

**算法复杂度：平均是nlogn，但是也会有n2的可能。**



**注：排序的稳定性和不稳定性——相等元素的相对位置，会不会被改变。**

## 归并排序Merge Sort

## 堆排序 Heap Sort