# 排序算法概述

排序就是将一组对象按照某种逻辑顺序重新排列的过程。比如，订单按照日期排序的——这种排序很可能使用了某种排序算法。在计算时代早期，大家普遍认为30% 的计算周期都用在了排序上。如果今天这个比例降低了，可能的原因之一是如今的排序算法更加高效，而并非排序的重要性降低了。现在计算机的广泛使用使得数据无处不在，而整理数据的第一步通常就是进行排序。几乎所有的计算机系统都实现了各种排序算法以供系统和用户使用。

即使你只是使用标准库中的排序函数，学习排序算法仍然有三大实际意义：

*  IT从业人员必备技能，也是互联网公司面试的必考点；
*  类似的技术也能有效解决其他类型的问题；
*  排序算法常常是我们解决其他问题的第一步。

排序在商业数据处理和现代科学计算中有着重要的地位，它能够应用于事物处理、组合优化、天体物理学、分子动力学、语言学、基因组学、天气预报和很多其他领域。其中一种排序算法（快速排序）甚至被誉为20 世纪科学和工程领域的十大算法之一。

数据结构和算法中，关于排序有十大算法，包括冒泡排序，简单选择排序，简单插入排序，归并排序，堆排序，快速排序、希尔排序、计数排序，基数排序，桶排序。

一般在面试中最常考的是快速排序和归并排序，并且经常有面试官要求现场写出这两种排序的代码。对这两种排序的代码一定要信手拈来才行。对于其他排序可能会要求比较各自的优劣、各种算法的思想及其使用场景，还有要知道算法的时间和空间复杂度。

通常查找和排序算法的考察是面试的开始，如果这些问题回答不好，估计面试官都没有继续面试下去的兴趣都没了。所以想开个好头就要把常见的排序算法思想及其特点要熟练掌握，有必要时要熟练写出代码。我们将由易到难学习这十种算法。

# 冒泡排序

冒泡排序是一种简单的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果它们的顺序错误就把它们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为相关的元素会经由交换慢慢“浮”到数列的顶端。

基本思路：

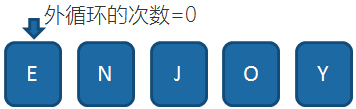
1、比较相邻的元素。如果第一个比第二个大(小)，就交换它们两个；

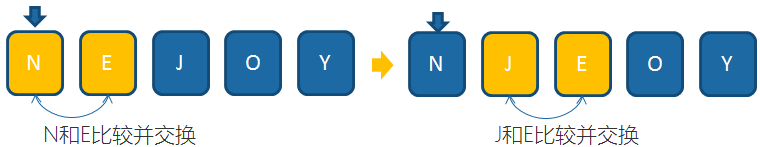
2、对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对，这样在最后的元素应该会是最大(小)的数；

3、针对所有的元素重复以上的步骤，除了最后一个；

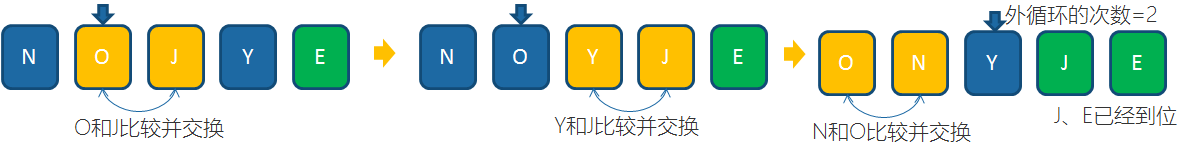
重复步骤1~2，直到排序完成。

## 冒泡降序示例：

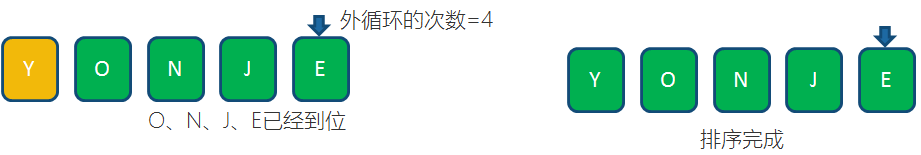
原始数组 











# 简单选择排序

选择排序的思想其实和冒泡排序有点类似，都是在一次排序后把最小的元素放到最前面。但是过程不同，冒泡排序是通过相邻的比较和交换。而选择排序是通过对整体的选择。

~~举个例子，对5,3,8,6,4这个无序序列进行简单选择排序，首先要选择5以外的最小数来和5交换，也就是选择3和5交换，一次排序后就变成了3,5,8,6,4.对剩下的序列继续进行选择和交换，最终就会得到一个有序序列。~~其实选择排序可以看成冒泡排序的优化，因为其目的相同，只是选择排序只有在确定了最小数的前提下才进行交换，大大减少了交换的次数。

**具体步骤：**

首先，找到数组中最大（小）的那个元素；

其次，将它和数组的第一个元素交换位置（如果第一个元素就是最大（小）元素那么它就和自己交换）；

再次，在剩下的元素中找到最大（小）的元素，将它与数组的第二个元素交换位置。如此往复，直到将整个数组排序。

## 简单选择排序（降序）示例

原始数组： 

，Y是数组中最大的元素

，Y和数组的第一个元素E交换位置

，在剩下的数组中继续寻找最大的元素，找到了O

 O和数组的第二个元素N交换位置

如此重复，最后形成降序后数组：

# 简单插入排序

插入排序不是通过交换位置而是通过比较找到合适的位置插入元素来达到排序的目的的。相信大家都有过打扑克牌的经历，特别是牌数较大的。在分牌时可能要整理自己的牌，牌多的时候怎么整理呢？就是拿到一张牌，找到一个合适的位置插入。这个原理其实和插入排序是一样的。

~~举个例子，我们将要收到5,3,4,,8,6这几张牌，我们先收到5这张牌，毫无疑问，这张牌的位置是正确的，没必要整理。接着收到了牌3，然后3要插到5前面，把5后移一位，变成3,5。接着又收到了牌4，现在我们会怎么做？把4插入到5前面，把5后移一位。~~

**具体步骤：**

* 对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。
* 为了给要插入的元素腾出空间，我们需要将插入位置之后的已排序元素在都向右移动一位。

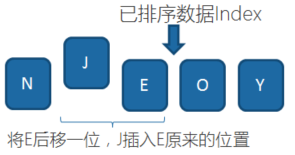
插入排序所需的时间取决于输入中元素的初始顺序。例如，对一个很大且其中的元素已经有序（或接近有序）的数组进行排序将会比对随机顺序的数组或是逆序数组进行排序要快得多。

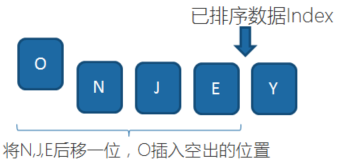
总的来说，插入排序对于部分有序的数组十分高效，也很适合小规模数组。

## 简单插入排序（降序）示例

原始数组：  我们总是认为原始数组的第一个元素已经是有序的了，于是从第二个元素开始进行排序。

第二个元素是N，比E大，所以将E后移一位，N插入E原来的位置

第三个元素是J，在已经排序的序列N、E中，J比N小，N不动，继续比较，J比E大，所以将E后移一位，J插入E原来的位置

第四个元素是O，在已经排序的序列N、J、E中，O比N大，O应该插在N的前面，所以将N、J、E全部后移一位，O插入N原来的位置。

如此重复，最后形成降序后数组：

# 希尔排序

一种基于插入排序的快速的排序算法(请大家先学习插入排序，了解基本的插入排序的思想。对于大规模乱序数组插入排序很慢，因为元素只能一点一点地从数组的一端移动到另一端。例如，如果主键最小的元素正好在数组的尽头，要将它挪到正确的位置就需要N-1 次移动。

希尔排序为了加快速度简单地改进了插入排序，也称为缩小增量排序，同时该算法是冲破O(n^2）的第一批算法之一。

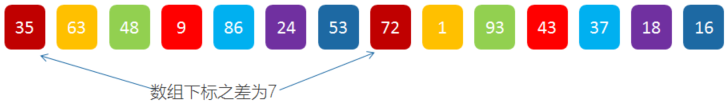
希尔排序是把待排序数组按一定增量的分组，对每组使用直接插入排序算法排序；然后缩小增量继续分组排序，随着增量逐渐减少，每组包含的元素越来越多，当增量减至 1 时，整个数组恰被分成一组，排序便完成了。这个不断缩小的增量，就构成了一个增量序列。

## 希尔排序（降序）示例

我们选择增量的计算方式为：gap=数组的长度/2，缩小增量继续以gap = gap/2的方式，于是形成的增量序列为{7,3,1}。



1、第一个增量为7，则原始数组被分为7组，比如{35,72}为一组，{63,1}为一组，组内的每个元素之间数组下标之差为7，这7组分别进行插入排序



组内插入排序之后：



2、第二个增量为3，则第一次排序后数组被分为3组，比如{72,43,53,48,18}为一组，组内的每个元素之间数组下标之差为3，这3组分别进行插入排序



组内插入排序之后：



3、

第三个增量为1，则第二次排序后数组被分为1组，进行插入排序，形成最后的排序数组



## 希尔排序中的增量序列

在先前较大的增量下每个子序列的规模都不大,用直接插入排序效率都较高，尽管在随后的增量递减分组中子序列越来越大,由于整个序列的有序性也越来越明显,则排序效率依然较高。

从理论上说，只要一个数组是递减的，并且最后一个值是1，都可以作为增量序列使用。有没有一个步长序列,使得排序过程中所需的比较和移动次数相对较少,并且无论待排序列记录数有多少,算法的时间复杂度都能渐近最佳呢？但是目前从数学上来说，无法证明某个序列是“最好的”。

**常用的增量序列有：**

希尔增量序列 ：{N/2, (N / 2)/2, ..., 1}，其中N为原始数组的长度，这是最常用的序列，但却不是最好的

Hibbard序列：{2^k-1, ..., 3,1}

Sedgewick序列：{... , 109 , 41 , 19 , 5，1} 表达式为

# 归并排序

归并排序是建立在归并操作上的一种有效的排序算法。该算法是采用分治法的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。

若将两个有序表合并成一个有序表，称为2-路归并，与之对应的还有多路归并。

对于给定的一组数据，利用递归与分治技术将数据序列划分成为越来越小的半子表，在对半子表排序后，再用递归方法将排好序的半子表合并成为越来越大的有序序列。

为了提升性能，有时我们在半子表的个数小于某个数（比如15）的情况下，对半子表的排序采用其他排序算法，比如插入排序。

## 归并排序（降序）示例



先讲数组划分为左右两个子表：

，

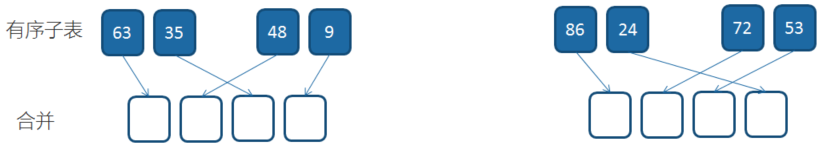
然后继续左右两个子表拆分：



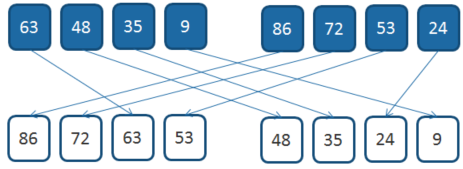
对最后的拆分的子表，两两进行排序



对有序的子表进行排序和比较合并



对合并后的子表继续比较合并



# 快速排序

快速排序被誉为20 世纪科学和工程领域的十大算法之一。快速排序（Quicksort）是对冒泡排序的一种改进，也是采用分治法的一个典型的应用。

首先任意选取一个数据（比如数组的第一个数）作为关键数据，我们称为基准数，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序，也称为分区（partition）操作。在实际实现时，一般会在原数组上直接操作。

通过一趟快速排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

为了提升性能，有时我们在分割后独立的两部分的个数小于某个数（比如15）的情况下，会采用其他排序算法，比如插入排序。

## 快速排序原理



1、选择数组的第一个数35为基准，切分数组，将所有比35大的都放到35前面，所有比35小的，放到35后面，进行快速排序



于是数组变成了：



2、对35左边的子数组，我们以63为基准数，进行快速排序，于是左边数组变为：



对35右边的子数组，我们以9为基准数，进行快速排序，于是左边数组变为：

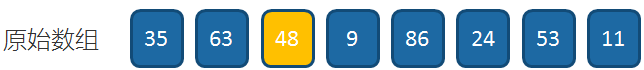


3、继续快速排序下去，最终形成有序数组

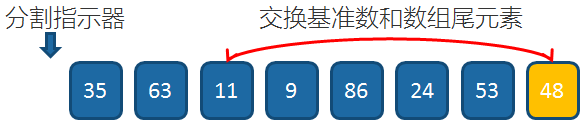


以上是快速排序的一个基本原理，看起来仿佛需要额外的数组进行辅助，但其实在实现的时候，并不需要，只要借助一个分割指示器就可以了。

## 快速排序（升序）实现图示



随机选择数组的一个数，比如48为基准，切分数组 同时引入一个分割指示器，这个分割指示器初始值是数组头元素下标减一，这里就是-1。同时交换基准数和数组尾元素 。



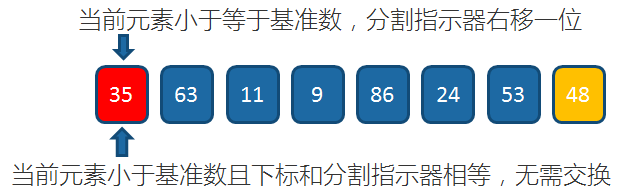
进行数组的遍历，将数组中的元素和基准数进行比较，为了满足所有比基准数小的数都放到基准数前面，所有比基准数大的数都放到基准数后面，我们需要遵循着这样的规则：

当前元素小于等于基准数时，分割指示器右移一位 ，当前元素下标小于等于分割指示器时当前元素保持不动，当前元素下标大于分割指示器时，当前元素和分割指示器所指元素交换；

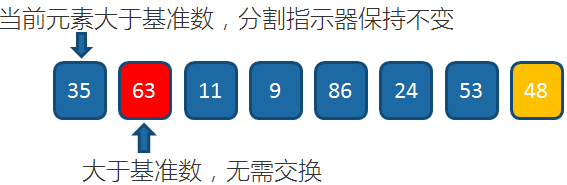
当前元素大于等于基准数，分割指示器保持不变 ，元素也无需交换。

所以接下来随着数组的遍历，其中元素的变动情况是：

1、



2、



3、

，交换63和11

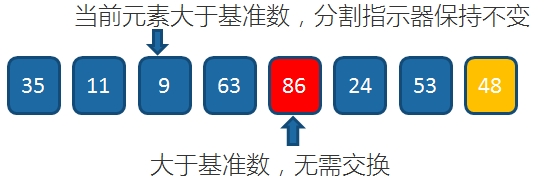
4、



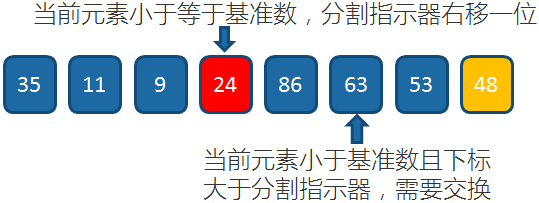
5、

，交换63和9

6、



7、

分割指示器指向了63，当前元素是24，于是交换63和24

8、



9、

，分割指示器指向了86，当前元素是48，于是交换86和48

10、

完成了本次分区操作



11、所有比基准数48小的数都已到它左边，所有比基准数48大的数都已到它右边。将左边和右边再按快速排序继续排序下去，就可完成最终的排序。

## 快速排序中的基准数

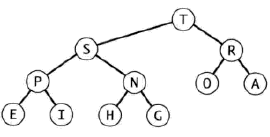
基准的选取：最优的情况是基准值刚好取在无序区的中间，这样能够最大效率地让两边排序，同时最大地减少递归划分的次数，但是一般很难做到最优。基准的选取一般有三种方式，选取数组的第一个元素，选取数组的最后一个元素，以及选取第一个、最后一个以及中间的元素的中位数（如4 5 6 7, 第一个4, 最后一个7, 中间的为5, 这三个数的中位数为５, 所以选择5作为基准）。

Dual-Pivot快排：两个基准数的快速排序算法，其实就是用两个基准数, 把整个数组分成三份来进行快速排序，在这种新的算法下面，比经典快排从实验来看节省了10%的时间。

# 堆排序

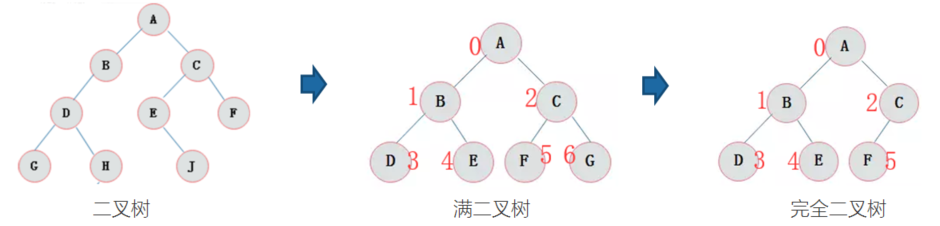
许多应用程序都需要处理有序的元素，但不一定要求他们全部有序，或者不一定要一次就将他们排序，很多时候，我们每次只需要操作数据中的最大元素（最小元素），那么有一种基于二叉堆的数据结构可以提供支持。

所谓二叉堆，是一个完全二叉树的结构，同时满足堆的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。在一个二叉堆中，根节点总是最大（或者最小）节点。

，这就是一个典型的二叉堆。

堆排序算法就是抓住了这一特点，每次都取堆顶的元素，然后将剩余的元素重新调整为最大（最小）堆，依次类推，最终得到排序的序列。

## 补充知识：完全二叉树



**二叉树：**是每个结点最多有两个子树的树结构。通常子树被称作“左子树”（left subtree）和“右子树”（right subtree）

**满二叉树：**除最后一层无任何子[节点](https://baike.baidu.com/item/%E8%8A%82%E7%82%B9/865052)外，每一层上的所有结点都有两个子结点二叉树。

**完全二叉树：**是由[满二叉树](https://baike.baidu.com/item/%E6%BB%A1%E4%BA%8C%E5%8F%89%E6%A0%91)而引出来的，如果我们将一棵满二叉树由上到下，由左至右，每个结点都用数字编号，另外一个二叉树也同样由上到下，由左至右，每个结点都用数字编号，二叉树中的每个结点都可以在满二叉树中一一对应，我们称这个二叉树为完全二叉树。所以一棵满二叉树一定是个完全二叉树，而完全二叉树不是满二叉树。

完全二叉树的数组表示法

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **E** | **F** |
| 0 | 1 | 2 | 3 | 4 | 5 |

A:0 B:1 C:2 B=2\*0+1 C=2\*(0+1)

B:1 D:3 E:4 D=2\*1+1 E = 2\*(1+1)

由此可以退出两个重要的推论：

**推论1**：对于位置为K的结点 左子结点=2\*k+1 右子结点=2\*(k+1)

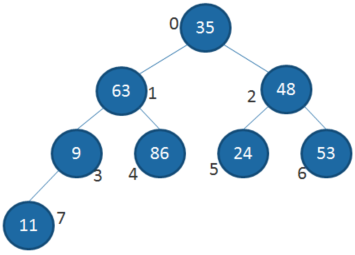
验证：C:2 2\*2+1=5 2\*(k+1)=6

**推论2**：最后一个非叶节点的位置为 (N/2)-1，N为数组长度。

## 堆排序（降序）示例



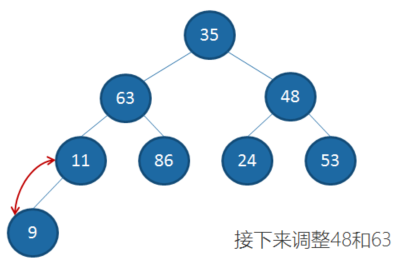
将该数组视为一个完全二叉树 ，则是：

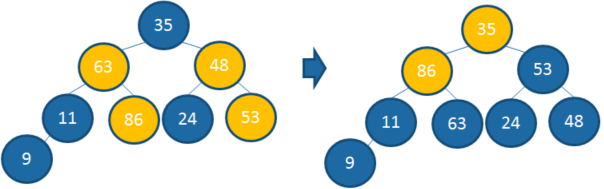


### 堆的初始化

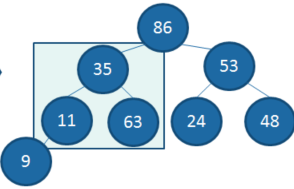
很明显，这个二叉树不符合最大堆的定义，需要初始化为最大堆，从**最后一个非叶节点**开始，从下到上，从右到左调整

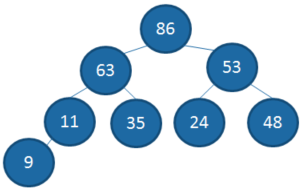
最后一个非叶节点在8/2-1=3的位置，也就是值为9的元素，将它和自己的叶节点进行比较并交换





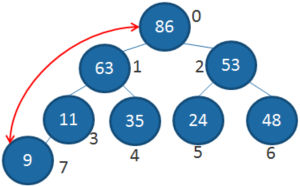
48和63调整到位后，进而调整根节点35，

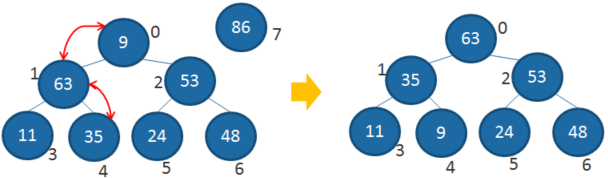
将35和它的子结点86交换，此时86变成根结点，35则变成子结点。很明显35和11、63组成的树不符合二叉堆的定义，此时需要再次调整35的位置：

此时就完成了堆的初始化，最大的数已经成为了根节点 。

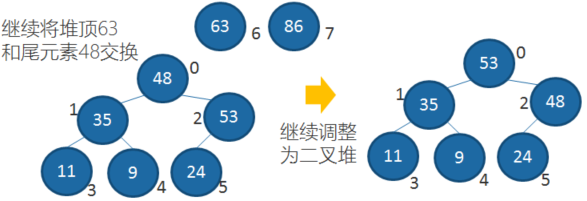
### 正式开始堆排序的过程

此时将堆顶的86和尾元素9交换

，86现在处于数组下标为7的位置，不再将86视为二叉树的一部分。9处于根结点，很明显，此时需要调整元素的位置 使之重新变成二叉堆



继续将堆顶63和尾元素48交换，63现在处于数组下标为6的位置，不再将63视为二叉树的一部分。48处于根结点，很明显，此时需要调整元素的位置 使之重新变成二叉堆



经过反复将堆顶元素和尾元素交换，并调整二叉堆的过程，最后数据变为



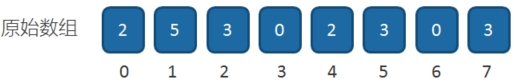
**如果需要进行降序，改用最小堆即可。**

# 计数排序

计数排序是一个排序时不比较元素大小的排序算法。

计数排序对一定范围内的整数排序时候的速度非常快，一般快于其他排序算法。但计数排序局限性比较大，只限于对整数进行排序，而且待排序元素值分布较连续、跨度小的情况。

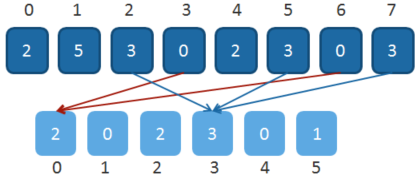
如果一个数组里所有元素都是整数，而且都在0-K以内。那对于数组里每个元素来说，如果能知道数组里有多少项小于或等于该元素，就能准确地给出该元素在排序后的数组的位置。

比如有个数组

对于这个数组来说，元素5之前有8个元素小于等于5（含5本身），因此排序后5所在的位置肯定是7.只要构造一个（5+1）大小的数组，里面存下所有对应A中每个元素之前的元素个数，就能在线性时间内完成排序。

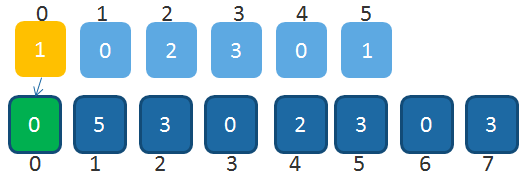
## 计数排序（升序）示例

1、初始化一个大小为（5+1）的计数数组（所有元素初始值为0），遍历整个原始数组，将原始数组中每个元素值对应计数数组下标中的元素大小+1

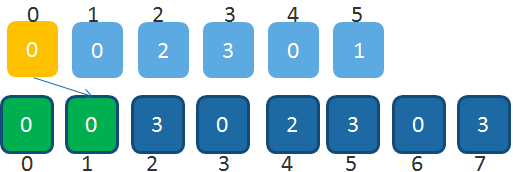


比如遍历数组的时候，发现原始数组有2个0，则在计数数组的下标为0的元素改为2，原始数组有3个3，则在计数数组的下标为3的元素改为3，

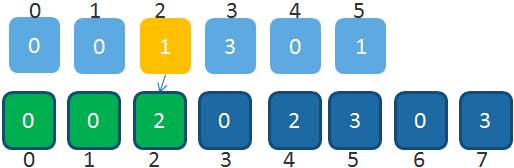
2、遍历计数数组，先访问第0个元素。第0个元素，下标为0，值为2，那么就应该将原始数组中第0个位置改写为0。



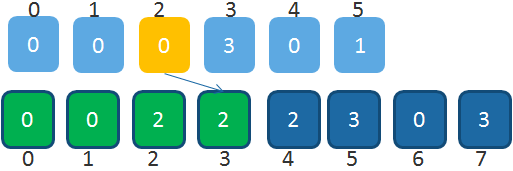
3、计数数组第0个元素，值为由2变为1，但是还没有归零，因此，计数数组访问索引不进行移动，而原始数组的访问索引应该向前移动一位，到了第1个位置，于是我们将原始数组第1个位置也改写为0。



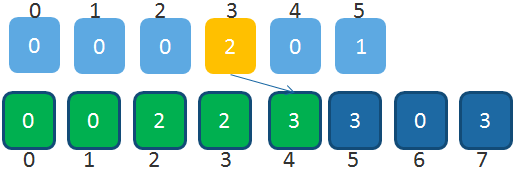
4、计数数组第0个元素，值已经归零，所以，计数数组访问索引向前移动一位，到了第1个位置，但是这个位置的元素值为0，无需处理，所以访问索引继续向前移动，到了第2个位置。这个位置上，值为2，此时原始数组的访问索引又向前移动了一位，到了第2个位置上，于是我们将原始数组的第二个位置改写为2。

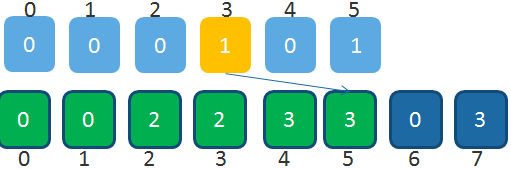


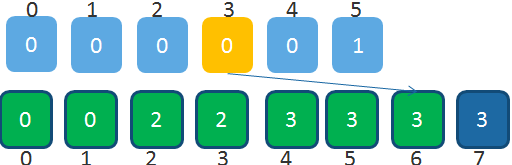
5、计数数组第2个元素，值为由2变为1，但是还没有归零，因此，计数数组访问索引不进行移动，而原始数组的访问索引应该向前移动一位，到了第3个位置，于是我们将原始数组第3个位置也改写为2。



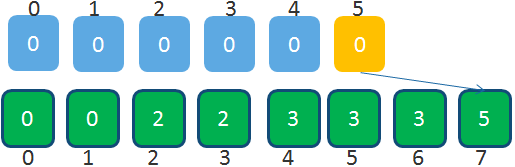
6、计数数组第2个元素，值已经归零，所以，计数数组访问索引向前移动一位，到了第3个位置，。这个位置上，值为3，按照我们前面所说的方法，原始数组的第4,5,6个元素都应该改写为3，同时计数数组中的第3个元素，值变为3.







7、计数数组到了第4个位置，元素值为0，无需处理，所以访问索引继续向前移动，到了第5个位置。这个位置上，值为1，此时原始数组的访问索引已经来到了第7个位置，这个位置的元素值改写为5。至此完成了排序。



## 注意点

实际应用中我们会同时找出数组中的max和min，主要是为了尽量节省空间。试想[1003, 1001, 1030, 1050]这样的数据要排序，真的需要建立长度为1050 + 1的数组吗？我们只需要长度为1050 - 1003 + 1= 48的数组（先不考虑额外+1的长度），就能囊括从最小到最大元素之间的所有元素了。

如果待排序数组的元素值跨度很大，比如[99999, 1, 2]，为三个元素排序要使用99999 - 1 + 1的空间，实在是浪费。所以计数排序适用于待排序元素值分布较连续、跨度小的情况。

# 桶排序

桶排序 (Bucket sort)的工作的原理：假设输入数据服从均匀分布，利用某种函数的映射关系将数据分到有限数量的桶里，每个桶再分别排序（有可能再使用别的排序算法或是以递归方式继续使用桶排序）。

基本步骤是：

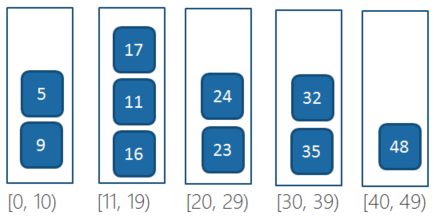
* 根据输入建立适当个数的桶，每个桶可以存放某个范围内的元素；
* 将落在特定范围内的所有元素放入对应的桶中；
* 对每个非空的桶中元素进行排序，可以选择通用的排序方法，比如插入、快排；
* 按照划分的范围顺序，将桶中的元素依次取出。排序完成。

桶排序利用函数的映射关系，减少了几乎所有的比较工作。实际上，桶排序的f(k)值的计算，其作用就相当于快排中划分，已经把大量数据分割成了基本有序的数据块(桶)。然后只需要对桶中的少量数据做先进的比较排序即可。

## 桶排序（降序）示例



我们可以建立5个桶，每个桶按范围顺序依次是[0, 10)、[10, 20)......[40, 49)，注意是左闭右开区间，对于待排序数组，5,9会被放到[0, 10)这个桶中，.......，48会被放到[40, 49)这个桶中



对这5个桶中的元素分别排序。 依次取出5个桶中元素，得到排序后的序列。

# 基数排序

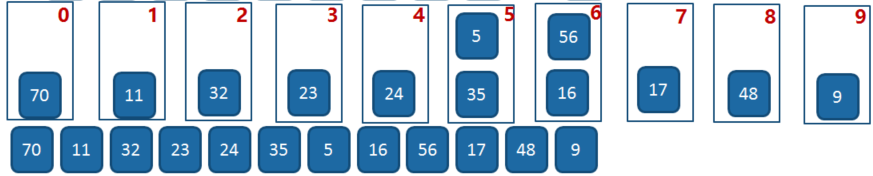
常见的数据元素一般是由若干位组成的，比如字符串由若干字符组成，整数由若干位0~9数字组成。基数排序按照从右往左的顺序，依次将每一位都当做一次关键字，然后按照该关键字对数组排序，同时每一轮排序都基于上轮排序后的结果；当我们将所有的位排序后，整个数组就达到有序状态。比如对于数字2985，从右往左就是先以个位为关键字进行排序，然后是十位、百位、千位，总共需要四轮。基数排序不是基于比较的算法。

基数是什么意思？对于十进制整数，每一位都只可能是0~9中的某一个，总共10种可能。那10就是它的基，同理二进制数字的基为2；对于字符串，如果它使用的是8位的扩展ASCII字符集，那么它的基就是256。

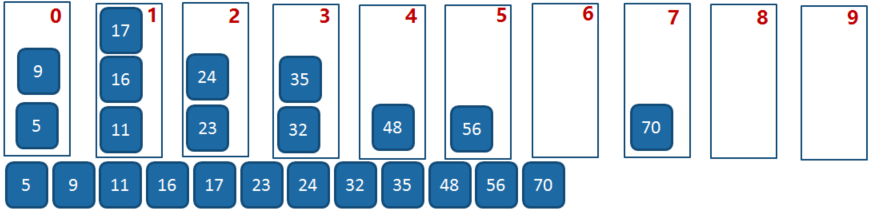
## 基数排序（降序）示例



首先按个位排序



然后再按十位进行排序



## 基数排序 vs 计数排序 vs 桶排序

基数排序有两种方法：

* MSD 从高位开始进行排序
* LSD 从低位开始进行排序

这三种排序算法都利用了桶的概念，但对桶的使用方法上有明显差异：

* 基数排序：根据键值的每位数字来分配桶
* 计数排序：每个桶只存储单一键值
* 桶排序：每个桶存储一定范围的数值

# 外部排序

有时，待排序的文件很大，计算机内存不能容纳整个文件，这时候对文件就不能使用内部排序了（我们一般的排序都是在内存中做的，所以称之为内部排序，而外部排序是指待排序的内容不能在内存中一下子完成，它需要做内外存的内容交换），外部排序常采用的排序方法也是归并排序，这种归并方法由两个不同的阶段组成：

1、采用适当的内部排序方法对输入文件的每个片段进行排序，将排好序的片段（成为归并段）写到外部存储器中（通常由一个可用的磁盘作为临时缓冲区），这样临时缓冲区中的每个归并段的内容是有序的。

2、利用归并算法，归并第一阶段生成的归并段，直到只剩下一个归并段为止。

例如要对外存中4500个记录进行归并，而内存大小只能容纳750个记录，在第一阶段，我们可以每次读取750个记录进行排序，这样可以分六次读取，进行排序，可以得到六个有序的归并段

每个归并段的大小是750个记录，记住，这些归并段已经全部写到临时缓冲区（由一个可用的磁盘充当）内了，这是第一步的排序结果。

完成第二步该怎么做呢？这时候归并算法就有用处了，算法描述如下：

1、将内存空间划分为三份，每份大小250个记录，其中两个用作输入缓冲区，另外一个用作输出缓冲区。首先对Segment\_1和Segment\_2进行归并，先从每个归并段中读取250个记录到输入缓冲区，对其归并，归并结果放到输出缓冲区，当输出缓冲区满后，将其写道临时缓冲区内，如果某个输入缓冲区空了，则从相应的归并段中再读取250个记录进行继续归并，反复以上步骤，直至Segment\_1和Segment\_2全都排好序，形成一个大小为1500的记录，然后对Segment\_3和Segment\_4、Segment\_5和Segment\_6进行同样的操作。

2、对归并好的大小为1500的记录进行如同步骤1一样的操作，进行继续排序，直至最后形成大小为4500的归并段，至此，排序结束。

# 排序算法总结



## 名词解释

### 算法的稳定性

**稳定：**如果a原本在b前面，而a=b，排序之后a仍然在b的前面；

**不稳定：**如果a原本在b的前面，而a=b，排序之后a可能会出现在b的后面；

排序算法如果是稳定的，那么从一个键上排序，然后再从另一个键上排序，前一个键排序的结果可以为后一个键排序所用。

### 复杂度

人总是贪婪的，在做一件事的时候，我们总是期望着可以付出最少的时间、精力或者金钱来获得最大的回报，这个类比到算法上也同样适用，那就是花最少的时间和最少的存储做成最棒的解决办法，所以好的算法应该具备时效高和存储低的特点。这里的「时效」是指时间效率，也就是算法的执行时间，对于同一个问题的多种不同解决算法，执行时间越短的算法效率越高，越长的效率越低；「存储」是指算法在执行的时候需要的存储空间，主要是指算法程序运行的时候所占用的内存空间。

从我们日常的经验就可以得知，一般来说，如果处理一件事情，这件事情越大，那么我们处理所花费的时间和精力就越多，在算法中同样也如此，所以我们在讨论算法复杂度时，往往需要考虑这个待处理数据的大小，我们把待处理数据的大小称之为数据的规模大小。讨论一个算法的复杂度，往往也就是寻找程序的运行时间是如何随着问题规模的变化而变化。

而有的时候算法的运行时间还取决于数据本身分布性质而不仅仅是「问题的规模大小」。对于这样的算法，我们把它们的执行情况分为「最优情况」、「最坏情况」和「平均情况」来讨论。

某个特定的数据集能让算法的执行情况极好，这就是最「最好情况」，而另一个不同的数据会让算法的执行情况变得极差，这就是「最坏情况」。不过在大多数情况下，算法的执行情况都介于这两种极端情况之间，也就是「平均情况」。

对于「最优情况」，没有什么大的价值，因为它没有提供什么有用信息，反应的只是最乐观最理想的情况，没有参考价值。「平均情况」是对算法的一个全面评价，因为它完整全面的反映了这个算法的性质，但从另一方面来说，这种衡量并没有什么保证，并不是每个运算都能在这种情况内完成。而对于「最坏情况」，它提供了一种保证，这个保证运行时间将不会再坏了，所以一般我们所算的时间复杂度是最坏情况下的时间复杂度，这和我们平时做事要考虑到最坏的情况是一个道理。。

#### 时间复杂度：

一个算法执行所耗费的时间。

#### 空间复杂度：

对一个算法在运行过程中临时占用存储空间大小的量度。

#### 常见复杂度

在各种不同算法中，若算法中语句执行次数(占用空间)为一个常数，则复杂度为O(1)；

当一个算法的复杂度与以2为底的n的对数成正比时，可表示为O(log n)；当一个算法的复杂度与n成线性比例关系时，可表示为O (n)，依次类推。

**由小到大：**O(1) < O(logn) < O(n) < O(nlogn) < O(n^2) < O(n^3) < O(2^n)

## 排序算法时间复杂度助记

冒泡、选择、插入排序需要两个for循环，每次只关注一个元素，平均时间复杂度为O()（外循环找元素O(n)，内循环找位置O(n)）

快速、归并、希尔、堆基于分治思想，log以2为底，平均时间复杂度往往和O(nlogn)（外循环找元素O(n)，内循环找位置O(logn)）相关

基数排序时间复杂度为O（N\*M），其中N为数据个数，M为数据位数

## 快速排序的优势

从平均时间来看，快速排序是效率最高的：

快速排序中平均时间复杂度O(nlog n)，这个公式中隐含的常数因子很小，比归并排序的O(nlog n)中的要小很多，所以大多数情况下，快速排序总是优于合并排序的。

而堆排序的平均时间复杂度也是O(nlog n)，但是堆排序存在着重建堆的过程，它把根节点移除后，把最后的叶子结点拿上来，是为了重建堆，但是，拿上的值是要比它的两个叶子结点要差很多的，它要比较很多次，才能回到合适的位置。堆排序就会有很多的时间耗在堆调整上。

虽然快速排序的最坏情况为排序规模（n）的平方关系，但是这种最坏情况取决于每次选择的基准， 对于这种情况，已经提出了很多优化的方法，比如三取样划分和Dual-Pivot快排。

同时，当排序规模较小时，划分的平衡性容易被打破，而且频繁的方法调用超过了O(nlog n)为O()省出的时间，所以一般排序规模较小时，会改用插入排序或者其他排序算法。