### 常用排序

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **复杂度** | **说明** | **备注** |
| 冒泡排序 Bubble Sort | O(N\*N) | 将待排序的元素看作是竖着排列的“气泡”，较小的元素比较轻，从而要往上浮 |  |
| 插入排序  Insertion sort | O(N\*N) | 逐一取出元素，在已经排序的元素序列中从后向前扫描，放到适当的位置 | 起初，已经排序的元素序列为空 |
| 选择排序 | O(N\*N) | 首先在未排序序列中找到最小元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小元素，然后放到排序序列末尾。以此递归。 |  |
| 快速排序  Quick Sort | O(n \*log2(n)) | 先选择中间值，然后把比它小的放在左边，大的放在右边（具体的实现是从两边找，找到一对后交换）。然后对两边分别使用这个过程（递归）。 |  |
| 堆排序HeapSort | O(n \*log2(n)) | 利用堆（heaps）这种数据结构来构造的一种排序算法。堆是一个近似完全二叉树结构，并同时满足堆属性：即子节点的键值或索引总是小于（或者大于）它的父节点。 | 近似完全二叉树 |
| 希尔排序  SHELL | O(n1+￡)  0<￡<1 | 选择一个步长(Step) ,然后按间隔为步长的单元进行排序.递归,步长逐渐变小,直至为1. |  |
| 箱排序 Bin Sort | O(n) | 设置若干个箱子，把关键字等于 k 的记录全都装入到第k 个箱子里 ( 分配 ) ，然后按序号依次将各非空的箱子首尾连接起来 ( 收集 ) 。 | 分配排序的一种：通过" 分配 " 和 " 收集 " 过程来实现排序。 |
|  |  |  |

**快速排序（quick sort）**

设要排序的[数组](http://baike.baidu.com/view/209670.htm)是A[0]……A[N-1]，首先任意选取一个数据（通常选用数组的第一个数）作为关键数据，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。值得注意的是，快速排序不是一种稳定的[排序算法](http://baike.baidu.com/view/297739.htm)，也就是说，多个相同的值的相对位置也许会在算法结束时产生变动

*注：在待排序的文件中，若存在多个关键字相同的记录，经过排序后这些具有相同关键字的记录之间的相对次序保持不变，该排序方法是稳定的；若具有相同关键字的记录之间的相对次序发生改变，则称这种排序方法是不稳定的。*  
*要注意的是，排序算法的稳定性是针对所有输入实例而言的。即在所有可能的输入实例中，只要有一个实例使得算法不满足稳定性要求，则该排序算法就是不稳定的。*

**排序演示**

**示例**

假设用户输入了如下数组：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 |
| 数据 | 6 | 2 | 7 | 3 | 8 | 9 |

创建变量i=0（指向第一个数据）, j=5(指向最后一个数据), k=6([赋值](http://baike.baidu.com/view/1201049.htm)为第一个数据的值)。

我们要把所有比k小的数移动到k的左面，所以我们可以开始寻找比6小的数，从j开始，从右往左找，不断递减变量j的值，我们找到第一个下标3的数据比6小，于是把数据3移到下标0的位置，把下标0的数据6移到下标3，完成第一次比较：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 |
| 数据 | 3 | 2 | 7 | 6 | 8 | 9 |

i=0 j=3 k=6

接着，开始第二次比较，这次要变成找比k大的了，而且要从前往后找了。递加变量i，发现下标2的数据是第一个比k大的，于是用下标2的数据7和j指向的下标3的数据的6做交换，数据状态变成下表：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 |
| 数据 | 3 | 2 | 6 | 7 | 8 | 9 |

i=2 j=3 k=6

称上面两次比较为一个循环。

接着，再递减变量j，不断重复进行上面的循环比较。

在本例中，我们进行一次循环，就发现i和j“碰头”了：他们都指向了下标2。于是，第一遍比较结束。得到结果如下，凡是k(=6)左边的数都比它小，凡是k右边的数都比它大：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 |
| 数据 | 3 | 2 | 6 | 7 | 8 | 9 |

如果i和j没有碰头的话，就递加i找大的，还没有，就再递减j找小的，如此反复，不断循环。注意判断和寻找是同时进行的。

然后，对k两边的数据，再分组分别进行上述的过程，直到不能再分组为止。

注意：第一遍快速排序不会直接得到最终结果，只会把比k大和比k小的数分到k的两边。为了得到最后结果，需要再次对下标2两边的数组分别执行此步骤，然后再分解数组，直到数组不能再分解为止（只有一个数据），才能得到正确结果。

#### ****插入排序(Insertion Sort)****

**插入排序(Insertion Sort)的基本思想是：将列表分为2部分，左边为排序好的部分，右边为未排序的部分，循环整个列表，每次将一个待排序的记录，按其关键字大小插入到前面已经排好序的子序列中的适当位置，直到全部记录插入完成为止。**

****

插入排序非常类似于整扑克牌。

在开始摸牌时，左手是空的，牌面朝下放在桌上。接着，一次从桌上摸起一张牌，并将它插入到左手一把牌中的正确位置上。为了找到这张牌的正确位置，要将它与手中已有的牌从右到左地进行比较。无论什么时候，左手中的牌都是排好序的。

也许你没有意识到，但其实你的思考过程是这样的：现在抓到一张7，把它和手里的牌从右到左依次比较，7比10小，应该再往左插，7比5大，好，就插这里。为什么比较了10和5就可以确定7的位置？为什么不用再比较左边的4和2呢？因为这里有一个重要的前提：手里的牌已经是排好序的。现在我插了7之后，手里的牌仍然是排好序的，下次再抓到的牌还可以用这个方法插入。编程对一个数组进行插入排序也是同样道理，但和插入扑克牌有一点不同，不可能在两个相邻的存储单元之间再插入一个单元，因此要将插入点之后的数据依次往后移动一个单元。

#### 选择排序

The algorithm works by selecting the smallest unsorted item and then swapping it with the item in the next position to be filled.

The selection sort works as follows: you look through the entire array for the smallest element, once you find it you swap it (the smallest element) with the first element of the array. Then you look for the smallest element in the remaining array (an array without the first element) and swap it with the second element. Then you look for the smallest element in the remaining array (an array without first and second elements) and swap it with the third element, and so on. Here is an example,

#### 冒泡排序（Bubble Sort）

冒泡排序（Bubble Sort），是一种[计算机科学](http://baike.baidu.com/view/92404.htm)领域的较简单的[排序算法](http://baike.baidu.com/view/297739.htm)。

它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。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越大的元素会经由交换慢慢“浮”到数列的顶端，故名。