以下文章里，待排序数组是A[],A的长度是n,以从小到大为例，且i和j都代表数组下标。

# 排序算法大总结

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。(基于交换的排序算法默认相同关键字不交换)

## 直接选择排序：

很简单的排序，就是两个指针i和j，迭代i从0扫描到n-1, j就用来找区间[i+1,n)的最小值，如果这个最小值比A[i]小就和A[i]交换。

时间复杂度是稳定的o(n\*n),需要o(1)的额外空间 ,由于5 8 5 2 9这种数据，存在相同元素，但第一次交换后把它换到了后面，就使得两个5相对顺序改变了，所以它不稳定

## 冒泡排序：

很简单的排序，第一次迭代i范围是[0,n-2]，每次如果A[i]>A[i+1]就交换。

第二次迭代i范围是[0,n-3]，每次如果A[i]>A[i+1]就交换。

…

一直进行上面操作，最后一次是：迭代i范围是[0,0]，有一个优化是如果一次冒泡之后没有任何数被交换了，说明这就是有序数组了，不需要继续执行程序

冒泡排序就像一系列得泡泡从小到大飘起来，但每次高度有所降低。

时间复杂度是最坏核对平均都是o(n\*n), 但由于有优化，如果数组本身就是有序的，时间复杂度将是o(n)，空间复杂度O(1),冒泡排序在对比相同数字时不交换，所以永远不会改变相同关键字的相对顺序，是稳定的

## 直接插入排序：

很简单的排序，建立一个空的新数组B，迭代A数组每个元素，插入新数组B适当位置，但由于如果每次都插入在B数组最前面，就要把B后面的数向后移动，移位操作是o(n)复杂度。这里的插入实际可以在原数组进行，不需要B数组。

时间复杂度不稳定，最好情况o(n),平均和最坏是o(n\*n), 空间复杂度O(1)

插入排序每次循环的条件就是A[j]<A[i],所以永远不会改变相同关键字的相对顺序，是稳定的

插入排序在小规模数据下(指n是个位数),效率最好

## 堆排序：

堆排序是选择排序的优化，就是把A数组所有数放在一个堆里(堆参考另一片文章)，再一个个取出堆顶元素组成有序数组。堆取出元素要进行调整

此算法时间复杂度是稳定的o(n\*log(n)),如果单开一个数据结构存堆，空间复杂度是o(n),但若在原本给定数组上建立堆，空间复杂度就是O(1)

堆排序建堆再拿出来的过程可能会破坏稳定性

## 希尔排序：

是插入排序的优化，设置一个增量s=n/2;把数组分层s组，每组有n/s个数，意思如图所示。

时间复杂度是不稳定的，最好是o(n)，最坏是o(n\*sqrt(n)),平均还是o(n\*sqrt(n))，不需要额外空间

由于多次插入排序，我们知道一次插入排序是稳定的，不会改变相同元素的相对顺序，但在不同的插入排序过程中，相同的元素可能在各自的插入排序中移动，最后其稳定性就会被打乱，所以shell排序是不稳定的。



## 快速排序：

取数组任意一个数，把所有比它小的数左边一部分, 把所有比它大的数放在右边一部分，最后要得到左右的那个分界线，之后递归的对左右部分进行上述操作。递归到每层函数只有1个元素，作为结束。

此算法最好时机复杂度o(n\*logn),最坏时机复杂度o(n\*n),平均时机复杂度o(n\*logn)，需要递归的内存O(logn) 的空间

对于算法在某些极端数据下，可能退化成o(n\*n)。

可以有改进算法，每次随机取数，也可以对递归采取归并排序躲开。

## 归并排序：

很好的利用分治算法，把数组一半一半拆开，如gif所示[归并排序\归并排序.gif](归并排序/归并排序.gif)

时间复杂度是n\*log(n),最好是log(n)，需要额外o(n)的空间去储存每次归并时用的数组

## 计数排序：

很简单的非比较类型的排序，它只能对数字进行排序，开一个足够大小的数组叫做ind，初始化全部是0，能装下min(A)和max(A),遍历A的所有元素，把 ind[A[i]]++;

之后按下标顺序遍历ind数组，每次访问到一个不是0的ind[]元素就输出。这么做输出的序列是有序的。

这是一个空间换时间的算法，时间复杂度和空间复杂度都是o(max(A)- min(A));对于数组元素存在很大或者很小的值的情况，此算法不适用。

计数排序虽然是计数出个数，但也相当于按照顺序把数放进对应的索引里，每次取出数，是按照顺序取的(队列的先进先出)，永远不会改变相对位置，是稳定的排序

## 基数排序：

非比较类型的排序，它只能对有基数的数据进行排序。设所有数当中，位数最长的数有mlen位。初始化10个链表(其实就是图论里的邻接表)，首先遍历所有数，把个位是k的数放在编号为k的链表里。

然后第0个链表开始按顺序取出里面的数，放回原来数组。下一步，遍历所有数，把十位是k的数放在编号为k的链表里。 然后第0个链表开始按顺序取出里面的数，放回原来数组。….以此类推，直到把mlen位是k的数放在编号为k的链表里。

时间复杂度是o(n\*mlen)，一般情况下mlen不会很大，所以此算法很快。

基数排序可以处理有负数的情况，提前分类找出正数负数分别排序，最后再合并即可

空间复杂度是o(n)，就是用来储存邻接表

基数排序每次取出数，是按照顺序取的(队列的先进先出)，永远不会改变相对位置，是稳定的排序

# 排序算法性能分析

注意排序算法的稳定性和时间复杂度的稳定性不是一回事。

排序的稳定性：相同值的节点相对位置是否会发生改变。   
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。

时间复杂度 Time Complexity 用TC简写，空间复杂度Space Complexity简写SC

计数排序的N是指max(A)- min(A)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 选择 | 冒泡 | 插入 | 堆 | 希尔 |
| av(TC) | O(n^2） | O(n^2） | O(n^2） | O(nlogn) | O(n\*sqrt(n)) |
| min(TC) | O(n^2） | O(n） | O(n） | O(nlogn) | O(n) |
| max(TC) | O(n^2） | O(n^2） | O(n^2） | O(nlogn) | O(n\*sqrt(n)) |
| SC | O(1） | O(1） | O(1） | O(1) | O(1) |
| 稳定性 | False | Ture | Ture | false | false |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 快排 | 归并 | 计数 | 基数 |  |
| av(TC) | O(n^2） | O(nlogn) | O(N) | O(n\*mlen) |  |
| min(TC) | O(nlogn） | O(nlogn) | O(N) | O(n\*mlen) |  |
| max(TC) | O(nlogn) | O(nlogn) | O(N) | O(n\*mlen) |  |
| SC | O(logn) | O(n) | O(N) | O(n) |  |
| 稳定性 | False | ture | ture | true |  |