[sort]20180323各种排序算法复杂度表和稳定性分析

1- 主要讨论一下稳定性。

常见排序算法的复杂度分析如下：

序号	类别	排序方法	时间复杂度			空间复杂度	稳定性	复杂度
			平均情况	最好情况	最坏情况	辅助存储		
1	插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定	简单
2		shell排序	$O(n^{1.3})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定	较复杂
3	选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定	简单
4		堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定	较复杂
5	交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定	简单
6		快速排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	不稳定	较复杂
7		归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	稳定	较复杂
8		基数排序	$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定	较复杂
		注：基数排序的复杂度中，r代表关键字的基数，d代表长度，n代表关键字的个数						

1- 稳定与不稳定性

假定在待排序的记录序列中，存在多个具有相同的关键字的记录，若经过排序，这些记录的相对次序保持不变，即在原序列中，ri=rj，且ri在rj之前，而在排序后的序列中，ri仍在rj之前，则称这种排序算法是稳定的；否则称为不稳定的。【相同元素的次序不变】

判断方法

对于不稳定的排序算法，只要举出一个实例，即可说明它的不稳定性；而对于稳定的排序算法，必须对算法进行分析从而得到稳定的特性。需要注意的是，排序算法是否为稳定的是由具体算法决定的，不稳定的算法在某种条件下可以变为稳定的算法，而稳定的算法在某种条件下也可以变为不稳定的算法。

例如，对于如下起泡排序算法，原本是稳定的排序算法，如果将记录交换的条件改成`r[j]>=r[j+1]`，则两个相等的记录就会交换位置，从而变成不稳定的算法。

```
void BubbleSort(int r[ ], int n){
  exchange=n; //第一趟起泡排序的范围是r[1]到r[n]
  while (exchange) //仅当上一趟排序有记录交换才进行本趟排序
  {
    bound=exchange; exchange=0 ;
    for (j=1; j if (r[j]>r[j+1]) {
      r[j]↔r[j+1] ;
      exchange=j ; //记录每一次发生记录交换的位置
    }
  }
}
```

再如，快速排序原本是不稳定的排序方法，但***若待排序记录中只有一组具有相同关键码的记录，而选择的轴值恰好是这组相同关键码中的一个，此时的快速排序就是稳定的***

2- 常见排序算法的稳定性

堆排序、快速排序、希尔排序、直接选择排序不是稳定的排序算法，而基数排序、冒泡排序、直接插入排序、折半插入排序、归并排序是稳定的排序算法。

说一下稳定性的好处。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，第一个键排序的结果可以为第二个键排序所用。基数排序就 是这样，先按低位排序，逐次按高位排序，低位相同的元素其顺序在高位也相同时是不会改变的。【相同的元素，其位置不用交换】

(1)冒泡排序

冒泡排序就是把小的元素往前调或者把大的元素往后调。比较是相邻的两个元素比较，交换也发生在这两个元素之间。所以，如果两个元素相等，我想你是**不会再无 聊地把他们俩交换一下的**；如果两个相等的元素没有相邻，那么即使通过前面的两两交换把两个相邻起来，这时候也不会交换，所以相同元素的前后顺序并没有改 变，所以冒泡排序是一种稳定排序算法。

(2)选择排序

选择排序是给每个位置选择当前元素最小的，比如给第一个位置选择最小的，在剩余元素里面给第二个元素选择第二小的，依次类推，直到第n-1个元素，第n个 元素不用选择了，因为只剩下它一个最大的元素了。那么，在一趟选择，如果当前元素比一个元素小，而该小的元素又出现在一个和当前元素相等的元素后面，那么 交换后稳定性就被破坏了。比较拗口，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。

(3)插入排序

插入排序是在一个已经有序的小序列的基础上，一次插入一个元素。当然，刚开始这个有序的小序列只有1个元素，就是第一个元素。**比较是从有序序列的末尾开始**，也就是想要插入的元素和已经有序的最大者开始比起，如果比它大则直接插入在其后面，否则一直往前找直到找到它该插入的位置***。如果碰见一个和插入元素相 等的，那么插入元素把想插入的元素放在相等元素的后面。所以，相等元素的前后顺序没有改变，从原无序序列出去的顺序就是排好序后的顺序，所以插入排序是稳 定的。***

(4)快速排序

快速排序有两个方向，左边的i下标一直往右走，当a[i] <= a[center_index]，其中center_index是中枢元素的数组下标，一般取为数组第0个元素。而右边的j下标一直往左走，当a[j] > a[center_index]。如果i和j都走不动了，i <= j, 交换a[i]和a[j],重复上面的过程，直到i>j。 交换a[j]和a[center_index]，完成一趟快速排序。在中枢元素和a[j]交换的时候，很有可能把前面的元素的稳定性打乱，比如序列为 5 3 3 4 3 8 9 10 11， 现在**中枢元素5和3(第5个元素，下标从1开始计)交换就会把元素3的稳定性打乱，所以快速排序是一个不稳定的排序算法，不稳定发生在中枢元素和a[j] 交换的时刻。**

(5)归并排序

归并排序是把序列**递归地分成短序列，递归出口是短序列只有1个元素(认为直接有序)或者2个序列(1次比较和交换),然后把各个有序的段序列合并成一个有序的长序列，不断合并直到原序列全部排好序。**可以发现，在1个或2个元素时，1个元素不会交换，2个元素如果大小相等也没有人故意交换，这不会破坏稳定 性。那么，**在短的有序序列合并的过程中，稳定是是否受到破坏？没有，合并过程中我们可以保证如果两个当前元素相等时，我们把处在前面的序列的元素保存在结果序列的前面，这样就保证了稳定性。**所以，归并排序也是稳定的排序算法。

(6)基数排序

基数排序是**按照低位先排序，然后收集；再按照高位排序，然后再收集；依次类推，直到最高位。**有时候有些属性是有优先级顺序的，先按低优先级排序，再按高优先级排序，最后的次序就是高优先级高的在前，高优先级相同的低优先级高的在前。**基数排序基于分别排序，分别收集，所以其是稳定的排序算法。**

(7)希尔排序(shell)

希尔排序是按照不同步长对元素进行插入排序，当**刚开始元素很无序的时候，步长最大，所以插入排序的元素个数很少，速度很快；当元素基本有序了，步长很小， 插入排序对于有序的序列效率很高。所以，希尔排序的时间复杂度会比o(n^2)好一些。**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。**

(8)堆排序

我们知道堆的结构是**节点i的孩子为2*i*和2i+1节点，大顶堆要求父节点大于等于其2个子节点，小顶堆要求父节点小于等于其2个子节点。**在一个长为n 的序列，堆排序的过程是**从第n/2开始和其子节点共3个值选择最大**(大顶堆)或者最小(小顶堆),这3个元素之间的选择当然不会破坏稳定性。但当为`n/2-1, n/2-2, ...1`这些个父节点选择元素时，就会破坏稳定性。有可能第`n/2`个父节点交换把后面一个元素交换过去了，而第`n/2-1`个父节点把后面一个相同的元素没有交换，那么这2个相同的元素之间的稳定性就被破坏了。所以，**堆排序不是稳定的排序算法。**

综上，得出结论: 选择排序、快速排序、希尔排序、堆排序不是稳定的排序算法，而**冒泡排序、插入排序、归并排序和基数排序**是稳定的排序算法。

2- 时间复杂度

n称为问题的规模，当n不断变化时，时间频度T(n)也会不断变化。但有时我们想知道它变化时呈现什么规律。为此，我们引入时间复杂度概念。 一般情况下，算法中基本操作重复执行的次数是问题规模n的某个函数，用T(n)表示，若有某个辅助函数f(n),使得当n趋近于无穷大时，T (n)/f(n)的极限值为不等于零的常数，则称f(n)是T(n)的同数量级函数。记作T(n)=O(f(n)),称O(f(n)) 为算法的渐进时间复杂度，简称时间复杂度。

1- 若算法中语句执行次数为一个常数，则时间复杂度为O(1)

3- 空间复杂度

空间复杂度(Space Complexity)是对一个算法在运行过程中临时占用存储空间大小的量度。

包含三个方面：

一个算法在计算机存储器上所占用的存储空间 =

1- 存储算法**本身**所占用的存储空间 【静态】

写较短的算法

2- 算法的输入输出数据所占用的存储空间【IO】

通过参数表由调用函数传递而来的，它不随本算法的不同而改变。

3- 算法在**运行**过程中临时占用的存储空间【运行时】

算法在运行过程中临时占用的存储空间随算法的不同而异，有的算法只需要占用少量的临时工作单元，而且不随问题规模的大小而改变，我们称这种算法是"就地/"进行的，是节省存储的算法，如这一节介绍过的几个算法都是如此；有的算法需要占用的临时工作单元数与解决问题的规模n有关，它随着n的增大而增大，当n较大时，将占用较多的存储单元

- 当一个算法的空间复杂度为一个常量，即不随被处理数据量n的大小而改变时，可表示为O(1)
- 当一个算法的空间复杂度与以2为底的n的对数成正比时，可表示为`0(1og2 n)`；
- 当一个算法的空I司复杂度与n成线性比例关系时，可表示为0(n).

1. 若形参为**数组**，则只需要为它分配一个存储由实参传送来的一个**地址指针的空间**，即一个机器字长空间；
2. 若形参为**引用方式**，则也只需要为其**分配存储一个地址的空间**，用它来存储对应实参变量的地址，以便由系统自动引用实参变量。