十大经典算法排序总结对比

一张图概括：



##### 名词解释：

**n**: 数据规模  
**k**:“桶”的个数  
**In-place**: 占用常数内存，不占用额外内存  
**Out-place**: 占用额外内存  
**稳定性**：排序后2个相等键值的顺序和排序之前它们的顺序相同

**（1）对于评述算法优劣术语的说明**

**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；

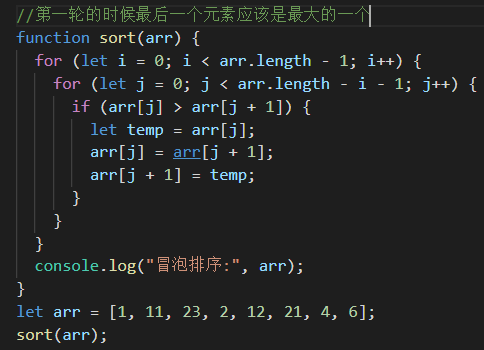
**内排序**：所有排序操作都在内存中完成；  
**外排序**：由于数据太大，因此把数据放在磁盘中，而排序通过磁盘和内存的数据传输才能进行；

**时间复杂度**: 一个算法执行所耗费的时间。  
**空间复杂度**: 运行完一个程序所需内存的大小。

## 冒泡排序（Bubble Sort）

##### 冒泡排序须知：

作为最简单的排序算法之一，冒泡排序给我的感觉就像Abandon在单词书里出现的感觉一样，每次都在第一页第一位，所以最熟悉。。。冒泡排序还有一种优化算法，就是立一个flag，当在一趟序列遍历中元素没有发生交换，则证明该序列已经有序。但这种改进对于提升性能来说并没有什么太大作用。。。



##### 什么时候最快（Best Cases）：

当输入的数据已经是正序时（都已经是正序了，我还要你冒泡排序有何用啊。。。。）

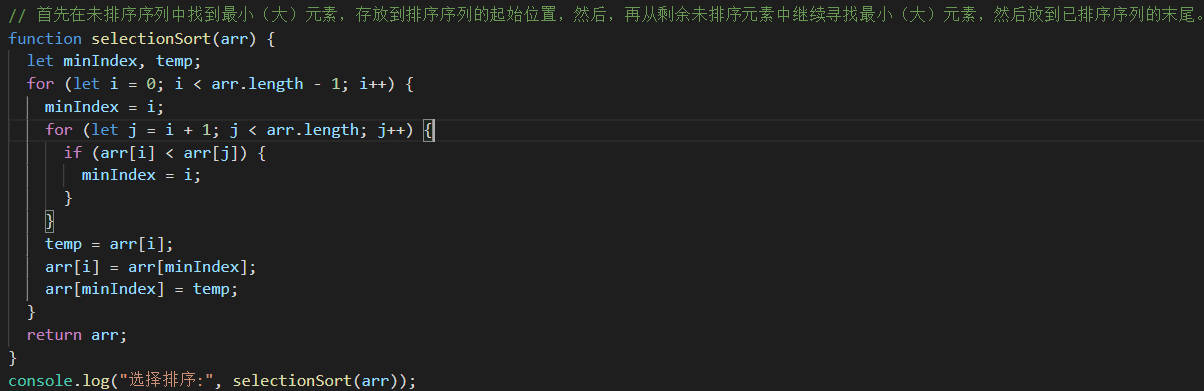
##### 什么时候最慢（Worst Cases）：

当输入的数据是反序时（写一个for循环反序输出数据不就行了，干嘛要用你冒泡排序呢，我是闲的吗。。。）

## 选择排序（Selection Sort）

##### 选择排序须知：

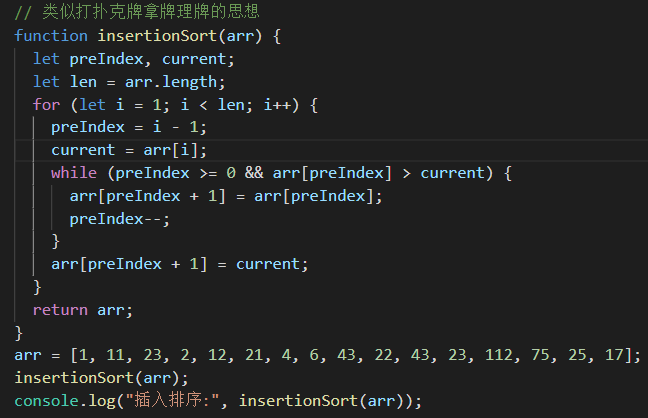
在时间复杂度上表现最稳定的排序算法之一，因为无论什么数据进去都是O(n²)的时间复杂度。。。所以用到它的时候，数据规模越小越好。唯一的好处可能就是不占用额外的内存空间了吧。



## 插入排序（Insertion Sort）

##### 插入排序须知：

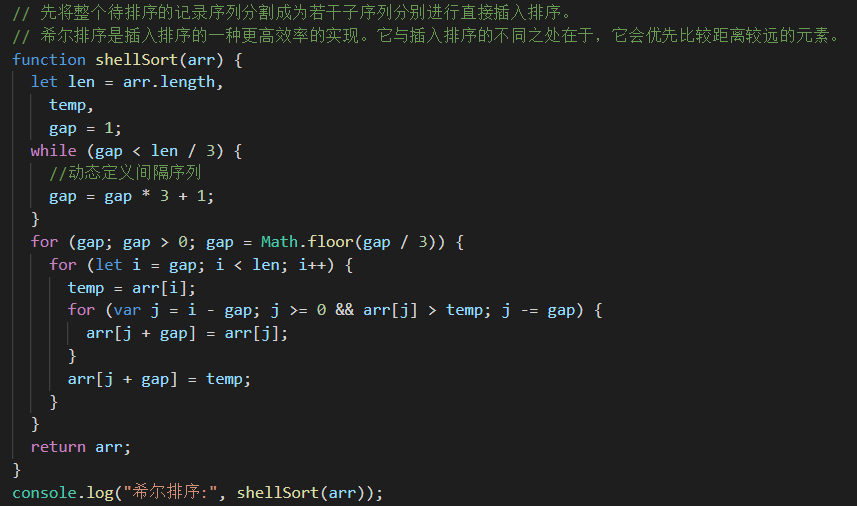
插入排序的代码实现虽然没有冒泡排序和选择排序那么简单粗暴，但它的原理应该是最容易理解的了，因为只要打过扑克牌的人都应该能够秒懂。当然，如果你说你打扑克牌摸牌的时候从来不按牌的大小整理牌，那估计这辈子你对插入排序的算法都不会产生任何兴趣了。。。



## 希尔排序（Shell Sort）

##### 希尔排序须知：

希尔排序是插入排序的一种更高效率的实现。它与插入排序的不同之处在于，它会优先比较距离较远的元素。希尔排序的核心在于间隔序列的设定。既可以提前设定好间隔序列，也可以动态的定义间隔序列。动态定义间隔序列的算法是《算法（第4版》的合著者Robert Sedgewick提出的。



## 归并排序（Merge Sort）

##### 归并排序须知：

作为一种典型的分而治之思想的算法应用，归并排序的实现由两种方法：

1. 自上而下的递归（所有递归的方法都可以用迭代重写，所以就有了第2种方法）
2. 自下而上的迭代  
   和选择排序一样，归并排序的性能不受输入数据的影响，但表现比选择排序好的多，因为始终都是O(n log n）的时间复杂度。代价是需要额外的内存空间。

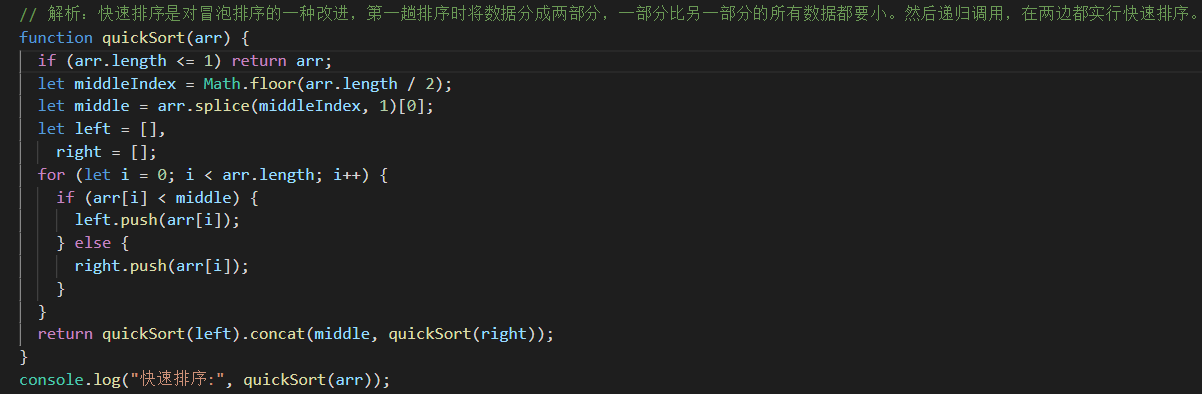


## 快速排序（Quick Sort）

##### 快速排序须知：

又是一种分而治之思想在排序算法上的典型应用。本质上来看，快速排序应该算是在冒泡排序基础上的递归分治法。  
快速排序的名字起的是简单粗暴，因为一听到这个名字你就知道它存在的意义，就是快，而且效率高! 它是处理大数据最快的排序算法之一了。虽然Worst Case的时间复杂度达到了O(n²)，但是人家就是优秀，在大多数情况下都比平均时间复杂度为O(n log n) 的排序算法表现要更好，可是这是为什么呢，我也不知道。。。好在我的强迫症又犯了，查了N多资料终于在《算法艺术与信息学竞赛》上找到了满意的答案：

快速排序的最坏运行情况是O(n²)，比如说顺序数列的快排。但它的平摊期望时间是O(n log n) ，且O(n log n)记号中隐含的常数因子很小，比复杂度稳定等于O(n log n)的归并排序要小很多。所以，对绝大多数顺序性较弱的随机数列而言，快速排序总是优于归并排序。

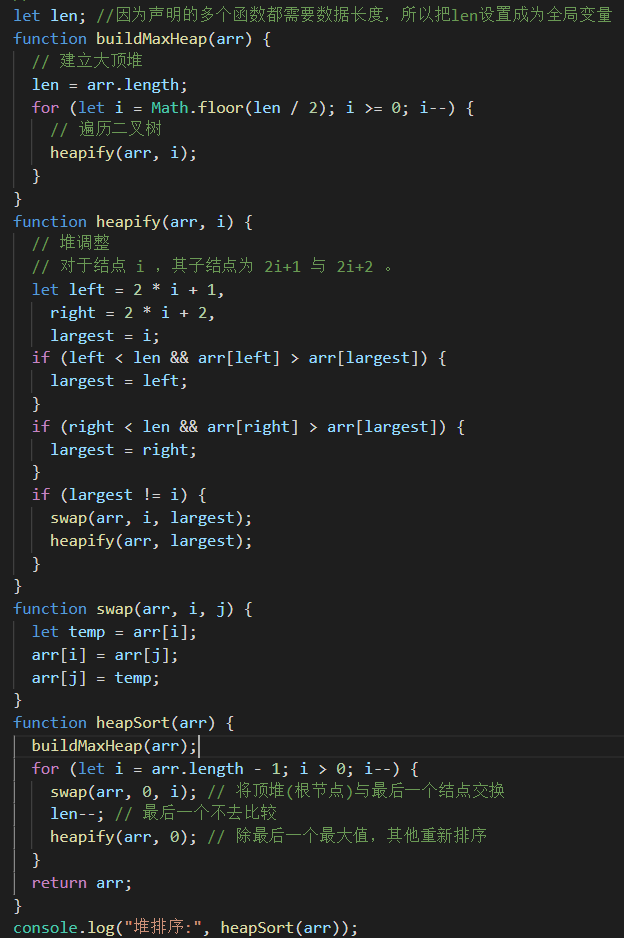


## 堆排序（Heap Sort）

##### 堆排序须知：

堆排序可以说是一种利用堆的概念来排序的选择排序。分为两种方法：

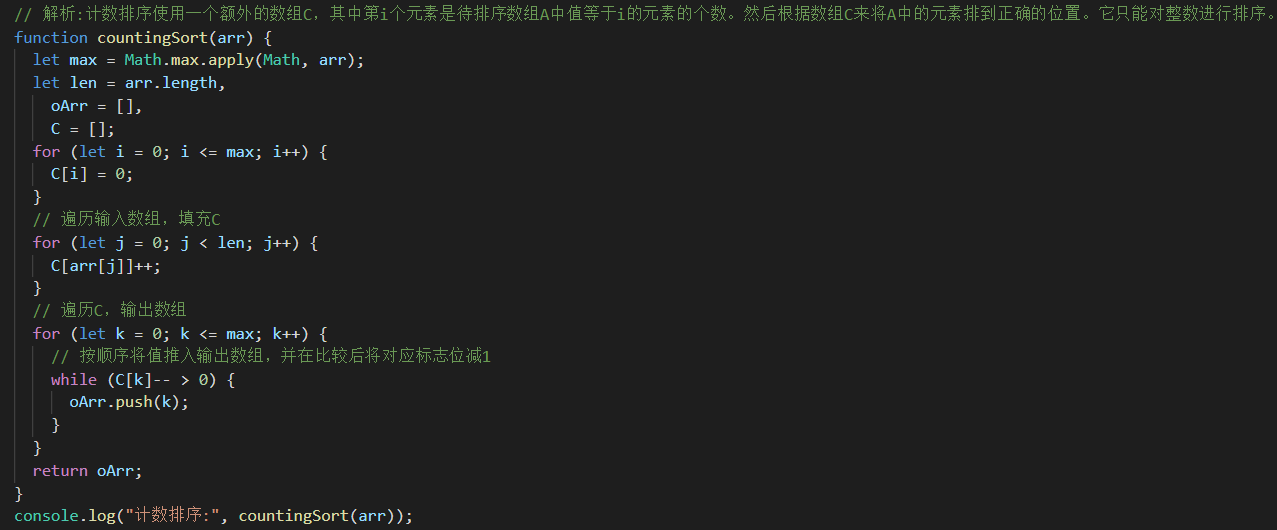
1. 大顶堆：每个节点的值都大于或等于其子节点的值，在堆排序算法中用于升序排列
2. 小顶堆：每个节点的值都小于或等于其子节点的值，在堆排序算法中用于降序排列



## 计数排序（Counting Sort）

##### 计数排序须知：

计数排序的核心在于将输入的数据值转化为键存储在额外开辟的数组空间中。  
作为一种线性时间复杂度的排序，计数排序要求输入的数据必须是有确定范围的整数。



## 桶排序（Bucket Sort）

##### 桶排序须知：

桶排序是计数排序的升级版。它利用了函数的映射关系，高效与否的关键就在于这个映射函数的确定。  
为了使桶排序更加高效，我们需要做到这两点：

1. 在额外空间充足的情况下，尽量增大桶的数量
2. 使用的映射函数能够将输入的N个数据均匀的分配到K个桶中

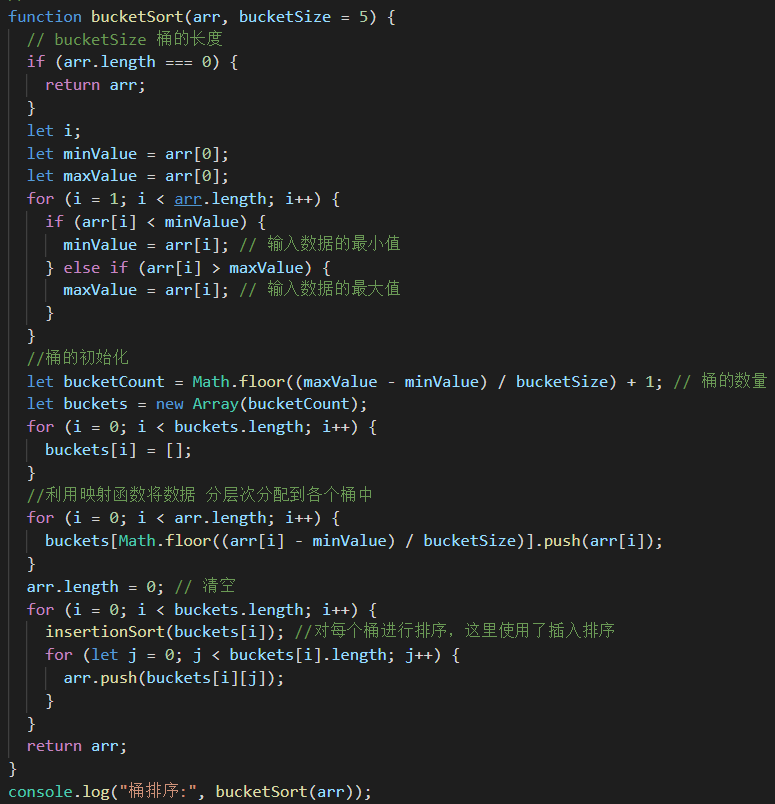
同时，对于桶中元素的排序，选择何种比较排序算法对于性能的影响至关重要。

##### 什么时候最快（Best Cases）：

当输入的数据可以均匀的分配到每一个桶中

##### 什么时候最慢（Worst Cases）：

当输入的数据被分配到了同一个桶中



## 基数排序（Radix Sort）

##### 基数排序须知：

基数排序有两种方法：

1. MSD 从高位开始进行排序
2. LSD 从低位开始进行排序

##### 基数排序 vs 计数排序 vs 桶排序

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

