一下文章里，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。再有注意怕羞算法的稳定性和时间复杂度的稳定性不是一回事。

# 排序算法大总结

## 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

## 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]

时间复杂度是稳定的o(n\*n), 不需要额外空间(交换通过位运算得到)

## 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),平均还是o(n\*n),不需额外空间

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)),额外空间复杂度是o(n),用来储存堆

## 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



## 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),最坏时机复杂度o(n\*n),平均时机复杂度o(n\*logn)，除了递归的内存，不需要额外空间。

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)

时间复杂度是稳定的n\*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。

## 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