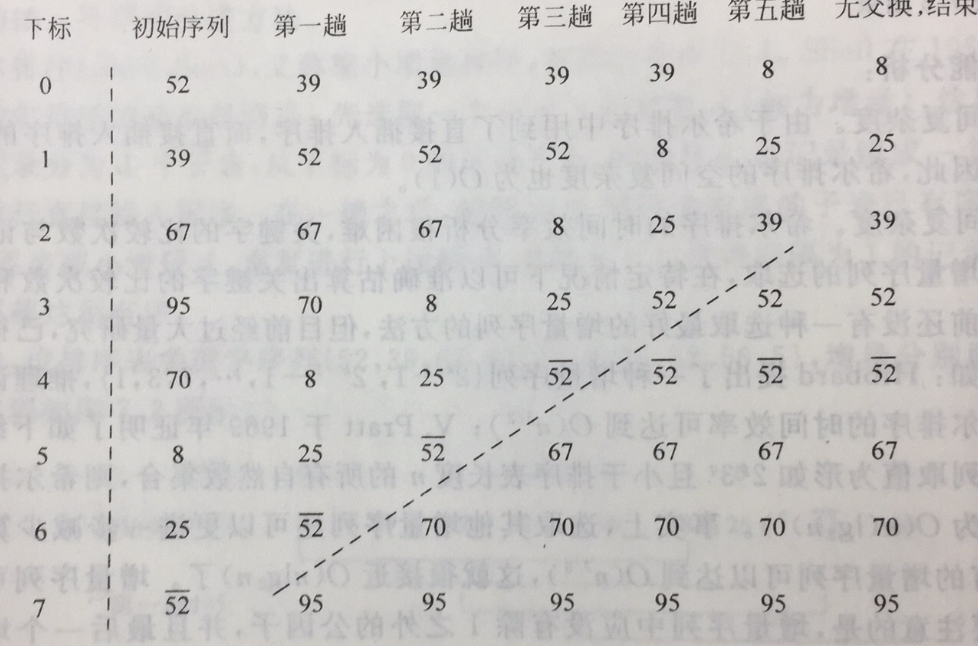
**交换排序的基本思想**

两两比较待排序记录的关键字，若两个记录的次序相反则交换两个记录，直到没有反序的记录为止。主要的方法有冒泡排序和快速排序。

**冒泡排序的基本思想**

将待排序的数组看成从上到下排放，把关键字值较小的记录看成比较轻的，关键字值较大的看成比较重的，较小关键字值的记录好像水中的气泡向上浮，较大的向下沉。

在第一趟中，从第0个记录到第n-1个记录，对两两相邻的关键字进行比较，若前一个关键字大于后一个关键字则交换，注意，这里是到n-1。每趟排序就能找出当前没有排好序的序列的最大的关键字，并移动到最后，在下趟排序就应该跳过这个关键字。n--。整个冒泡排序进行n-1趟刚好排序完。我们设立同时设立一个flag来记录每趟是否有交换，当该趟只要有一趟交换的时候flag=true，进行下趟排序，如果当前没有交换，flag=false，说明已经排好了，无需进行下次排序，直接跳出循环。



**步骤**

1.初始i=1，flag=true。

2.在无序序列中[0-(n-i)]中从头到尾两个相邻记录相比较，大的后移动，修改flag

3.判断flag，为true进行下趟排序，i++

4.重复2-3，直到i=length-1

**注意，这里设置标志位是要较少比较次数，如果在中间位置就已经排序完成，不设置标志位的话，会继续进行。如果设置了标志位，当排好的时候就停止。标志位设什么呢？当前排序，有交换的时候说明下趟排序可能没有排好，需要继续。当当前排序没有交换的时候，说明已经排好序了，不用再继续排序了，就跳出循环，输出结果。**

**代码**

public static int[] bubbleSortDown(int[] nums){//冒泡法排序，每次循环都从后到前排好一个关键字，下沉法

int i=1;

//用于记录当前没排好序的最后一个位置，length-i

//length-1，length-2，length-3 ... length-length=0

//因为采用的是大的关键字往后换

//每次都是找出当前最大的关键字放到没排好序的最后一个位置，然后//没排好序列的最后一个关键字位置-1

//也就是i++，当前没排好序的最后一个位置为length-i

boolean flag=true;//设立标志，初始为true，否则第一趟排序都没法排

while(i<nums.length&&flag) {

for (int j = 0; j < nums.length - i; j++) {//从前到后，因为每次nums.length-i是没排好序号的最后的位置

//也是当前最大关键字应该所在的位置

//所以应该遍历到nums.length-i的前一个位置，j<nums.length-i

//每次都是与后一个关键字比较

flag=false; //当没有交换的时候flag不会变成true

if (nums[j] > nums[j + 1]) { //用当前关键字和下一个关键字比较

int temp = nums[j]; //如果前面大，就和后边对换。

nums[j] = nums[j + 1];

nums[j + 1] = temp;

flag=true; //当有交换的时候flag变成true，进行下趟排序

//如果此趟排序没有任何交换，flag=false，说明排好序，跳出排序

}

}//这样每次从头到尾交换，就可以把当前最大的关键字换到没排好序的序列的最后边，也就是该在的位置。

i++;//排下一个关键字

}

return nums;

}

下沉法就是将大的关键字后移，必须从头开始

上浮法就是将小的关键字前移，必须从尾开始

public static int[] bubbleSortUp(int[] nums){ //上浮法

int i=0;

boolean flag=true; //是否修改的标志位

while(i<nums.length&&flag){

for(int j=nums.length-1;j>i;j--){

flag=false;

if(nums[j]<nums[j-1]){ //小的上浮

int temp=nums[j-1];

nums[j-1]=nums[j];

nums[j]=temp;

flag=true; //修改标志位

}

}

i++;

}

return nums;

}

**算法性能分析**

1.空间复杂度

只用了一个辅助单元，空间复杂度位O(1)

2.时间复杂度

最好情况是排序表已经有序，在第一趟比较过程中，一次交换都没有发生，所以执行一趟就结束了。比较次数为n-1，不需要移动记录。

最坏情况是排序表逆序，总共要进行n-1趟排序，第i趟排序进行比较的次数为

n-i，移动次数为3(n-i)，总的比较次数为，总的移动次数为.

因为冒泡排序算法的**时间复杂度为O(n2)**

**稳定性：稳定**