冒泡排序算法-BubbleSort

# 冒泡排序简述（Bubble Sort）

排序分成两种，从大到小和从小到大，这里默认是**从小到大排序**。（从小到大冒泡排序分为两种，**从后往前排序**和**从前往后排序）**。一般采用**从前往后**排序。

冒泡排序重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换为止，也就是说该数列已经排序完成。这个算法的名字由来是**因为越大的元素会经由交换慢慢“浮”到数列的顶端**，故名。

# 算法原理：（从前往后，从小到大排序）

## 设数组有N个元素，共需要N-1轮。每一轮所需要的比较次数依次从N-1到1；因此外循环i最好设置为从N-1到1，便于内循环参数设置；（若外循环i从N-1到1，则内循环j需要从0到i-1即可；若外循环i从0到N-1，内循环j需要从0到N-1-i）；（显然第一种好些）

## 第一轮，从前往后，依次比较相邻的两个元素，共有N-1对比较。如果第一个元素比第二个大，就交换他们两个，否则不交换。完成第一轮，数组最后一个元素一定是最大值，后续就不需要对最后一个元素作比较了。

## 然后，开始第二轮，第二轮需要N-2次比较，完成第二轮，数组倒数第二个元素为第二大值；第三轮，需要N-3次比较；第N-1轮，需要1次比较即可。每一轮比较都会把最大的值排到最后。

## 经过N-1轮之后，所有的元素已经排好。

# Java代码实现：

外循环i从N-1到1：

**public** **static** **void** bubbleSort(**int** a[]) {

**int** N = a.length;

**int** temp = 0;

**for (int i = N - 1; i > 0; i--) {**

**// 外循环i从N-1到1**

**for (int j = 0; j < i; j++) {**

**// 内循环j从0到i-1**

**if** (a[j] > a[j + 1]) {

temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

}

}

}

}

掌握好冒泡排序的关键是：**控制嵌套循环的意义，外循环控制轮数，N个数据，需要N-1轮，for(int i=N-1;i>0;i--)；内循环控制每轮的比较次数，第i轮需要比较i-1次，for(int j=0;j<i;j++)。**

外循环i从0到N-2：（当然i也可以从1到N-2，那么j就从0到N-i）

**public** **static** **void** bubbleSort(**int** a[]){

**int** N = a.length;

**int** temp = 0;

**for(int i = 0;i < N-1;i++){**

// 外循环控制轮数

**for(int j = 0;j < N-i-1;j++){**

// 内循环控制每轮比较的次数，逐渐减少至0

**if**(a[j] > a[j+1]){

temp = a[j];

a[j] = a[j+1];

a[j+1] = temp;

}

}

}

}

掌握好冒泡排序的关键是：**控制嵌套循环的意义，外循环控制轮数，N个数据，需要N-1轮，for(int i=0;i<N-1;i++)；内循环控制每轮的比较次数，第i轮需要比较N-1-i次，for(int j=0;j<N-1-i;j++)。**

这种方式推荐使用，意义更加明确。因为每一轮的比较次数肯定是**从N-1递减为1**，所有外循环就是**从N-1到1**，内循环直接从0到i-1即可。

# 冒泡排序算法优化

可以知道冒泡排序的核心是**两两对比进行交换**。如果有一个无序数列（2，1，3，4，5，6，7，8，9，10），按照上面的代码，从第一次循环交换后的操作，可以说都是没必要的。（经过一轮操作就可以完成排序任务。）**所以，这些操作就是我们需要优化的地方**。

**那么如何优化？**

通过观察可以看到，造成没必要的操作主要原因是**后面8个数的顺序都已经是有序**。所以，我们**可以通过设置一个标记变量，标记数列中的数是否在循环结束前就已经排好序。**

**优化后的代码：**

**public** **static** **void** bubbleSortOptimization2(**int** a[]) {

**int** N = a.length;

**int** temp = 0;

**int flag = 1; //设置标记变量**

**for** (**int** i = N - 1; i > 0&&flag != 0; i--) {

**flag = 0;** //只要flag在下一次外循环条件检测的时候值为0，就说明已经排好序，不用继续循环

**for** (**int** j = 0; j < i; j++) {

**if** (a[j] > a[j + 1]) {

**flag = 1; //如果有交换，就将标记变量赋1**

temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

}

}

}

}

在最好的情况下，冒泡排序的时间复杂度为O（N）。

即1,2,3,4，5，7,8,9,10 只需要比较9次，不需要交换。

下面讨论几种情况：

1,2,3,4,5,6,7,8,9,10 需要1轮比较，交换0次。

2,1,3,4,5,6,7,8,9,10 需要两轮比较，需要9+8=17次比较，交换1次。

1,2,3,4,6,5,7,8,9,10 需要2轮比较，交换1次。

2,3,4,5,6,7,8,9,1,10 需要9轮比较，交换8次。（第一轮中交换1次）

2,3,4,5,6,7,8,9,10,1 需要9轮比较，交换9次。（第一轮中交换1次）

只要在本轮中发生了数据交换，下一轮就必须进行；只有在本轮中没有发生数据交换，才能确保数据已经排序完毕。**虽然第2、3中情况，第一轮就可以完成排序，第二轮必须确定已经排序完毕。不能通过设置交换次数大于等于2，来作为设置flag=1的标志，因为第4、5中情况，在每一轮中都是交换一次。**

# 时间复杂度

在优化之前，**比较次数**是固定的，都是；移动次数最优下为0，最坏下为。

若文件的初始状态是正序的，一趟扫描即可完成排序。所需的关键字比较次数 https://gss0.bdstatic.com/-4o3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D12/sign=5e6ca6622c738bd4c021b633a08bb91b/b3b7d0a20cf431ad4f379a864936acaf2fdd98ac.jpg 和记录移动次数 https://gss1.bdstatic.com/-vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D16/sign=c765e07cd5ca7bcb797bc329bf09e8ac/48540923dd54564e27333d86b1de9c82d1584f3f.jpg 均达到最小值： https://gss3.bdstatic.com/7Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D84/sign=c4f5a2db0b23dd542573aa6cd0090824/78310a55b319ebc4b8a3a8268026cffc1f1716b1.jpg ， https://gss3.bdstatic.com/7Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D66/sign=1745c99097cad1c8d4bbff217e3e11cd/b21c8701a18b87d6007472a0050828381f30fd5f.jpg 。

所以，冒泡排序最好的[时间复杂度](https://baike.baidu.com/item/%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6)为 https://gss3.bdstatic.com/7Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D33/sign=feddd1e257fbb2fb302b5e114e4a94f7/b58f8c5494eef01f9d1ac17ae2fe9925bc317d07.jpg 。

若初始文件是反序的，需要进行 https://gss3.bdstatic.com/7Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D32/sign=411c7ed7d309b3deefbfe26acdbf0492/8b82b9014a90f60364df45f83b12b31bb051ed26.jpg 趟排序。每趟排序要进行 https://gss3.bdstatic.com/7Po3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D30/sign=3af139940cf41bd5de53eef450da752f/503d269759ee3d6d1214597441166d224e4adebb.jpg 次关键字的比较(1≤i≤n-1)，且每次比较都必须移动记录三次来达到交换记录位置。在这种情况下，比较和移动次数均达到最大值：

https://gss1.bdstatic.com/-vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D171/sign=f38fc3a1bb389b503cffe455b434e5f1/838ba61ea8d3fd1f846a7b85324e251f95ca5f2a.jpg

https://gss0.bdstatic.com/94o3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D186/sign=41bfe6dab119ebc4c4787291b427cf79/7af40ad162d9f2d36fa2fd50abec8a136227ccda.jpg

冒泡排序的**最坏时间复杂度**为 https://gss1.bdstatic.com/-vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D40/sign=17b28fa860d0f703e2b294dc09fa9d5c/e4dde71190ef76c63a4e22949f16fdfaae5167f5.jpg 。

综上，因此冒泡排序总的平均时间复杂度为 https://gss1.bdstatic.com/-vo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D40/sign=17b28fa860d0f703e2b294dc09fa9d5c/e4dde71190ef76c63a4e22949f16fdfaae5167f5.jpg 。

# 算法稳定性

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**。