# 冒泡排序

## 实现原理

所谓冒泡排序就是对一个长度为n的无序序列进行若干次遍历，每次遍历从序列初始位置开始，不断比较相邻的两个元素，根据比较结果调整元素位置，每次遍历将会有一个元素被移动到正确的位置：

第一次遍历结束，序列中最大的元素被移动到下标为n-1的位置；

第二次遍历结束，序列中倒数第二大的元素被移动到下标为n-2的位置；

第n-1次遍历结束，序列中倒数第二小的元素被移动到下标为1的位置；

第n次遍历没有必要比较和移动元素，因为此次序列已经有序。

## 注意点

1. 没有必要每次遍历都对整个序列进行遍历，因为在前几次的遍历已经使得序列末尾的若干个元素有序。
2. 可能在某次遍历结束后序列就已经有序了，比如下面实例中第4次遍历后序列已经有序了，有没有什么手段判断序列已经有序，从而省去后续的遍历呢？

## 实例

现有一个无序数组，共7个数：89 45 54 29 90 34 68。使用冒泡排序对这个序列进行升序排序。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 第1次遍历  89 45 54 29 90 34 68  45 89 54 29 90 34 68  45 54 89 29 90 34 68  45 54 29 89 90 34 68  45 54 29 89 90 34 68  45 54 29 89 34 90 68  45 54 29 89 34 68 90 | 第2次遍历  45 54 29 89 34 68 90  45 54 29 89 34 68 90  45 29 54 89 34 68 90  45 29 54 89 34 68 90  45 29 54 34 89 68 90  45 29 54 34 68 89 90 | 第3次遍历  45 29 54 34 68 89 90  29 45 54 34 68 89 90  29 45 54 34 68 89 90  29 45 34 54 68 89 90  29 45 34 54 68 89 90 | 第4次遍历  29 45 34 54 68 89 90  29 45 34 54 68 89 90  29 34 45 54 68 89 90  29 34 45 54 68 89 90 | 第5次遍历  29 34 45 54 68 89 90  29 34 45 54 68 89 90  29 34 45 54 68 89 90 | 第6次遍历  29 34 45 54 68 89 90  29 34 45 54 68 89 90 |

## 优化

蛮力法的应用有一个显著的特点，就是在经过适当的努力之后，可以对算法进行一定的改良，从而它的性能，但并不会减弱算法本身的时间复杂度。冒泡排序作为蛮力法的典型应用，自然也有这种特性。

观察上面实例中展示的排序过程，不难发现，在第4次遍历结束后，序列已经有序，后续的第5次、第6次遍历其实没有必要，极端情况下，序列一开始就是有序的，难道还要傻傻的进行后续的若干次遍历吗？

可以添加一个变量，记录某次遍历的过程中，是否存在元素交换，如果不存在则说明序列已经有序。

## 评价

**优点**：

1. 简单。
2. 稳定
3. 对于已经有序的数组，排序效率高。

**缺点**：

1. 效率低，时间复杂度为O(n2)，空间复杂度为O(1)。
2. 不能并发执行