**目录**

[要点](https://www.cnblogs.com/jingmoxukong/p/4302718.html#%E8%A6%81%E7%82%B9)  
[算法思想](https://www.cnblogs.com/jingmoxukong/p/4302718.html#%E7%AE%97%E6%B3%95%E6%80%9D%E6%83%B3)  
[算法分析](https://www.cnblogs.com/jingmoxukong/p/4302718.html#%E7%AE%97%E6%B3%95%E5%88%86%E6%9E%90)  
[冒泡排序算法的性能](https://www.cnblogs.com/jingmoxukong/p/4302718.html#%E5%86%92%E6%B3%A1%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95%E7%9A%84%E6%80%A7%E8%83%BD)  
[时间复杂度](https://www.cnblogs.com/jingmoxukong/p/4302718.html#%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6)  
[算法稳定性](https://www.cnblogs.com/jingmoxukong/p/4302718.html#%E7%AE%97%E6%B3%95%E7%A8%B3%E5%AE%9A%E6%80%A7)  
[优化](https://www.cnblogs.com/jingmoxukong/p/4302718.html#%E4%BC%98%E5%8C%96)  
[完整参考代码](https://www.cnblogs.com/jingmoxukong/p/4302718.html#%E5%AE%8C%E6%95%B4%E5%8F%82%E8%80%83%E4%BB%A3%E7%A0%81)  
[JAVA版本](https://www.cnblogs.com/jingmoxukong/p/4302718.html#java%E7%89%88%E6%9C%AC)  
[参考资料](https://www.cnblogs.com/jingmoxukong/p/4302718.html#%E5%8F%82%E8%80%83%E8%B5%84%E6%96%99)  
[相关阅读](https://www.cnblogs.com/jingmoxukong/p/4302718.html#%E7%9B%B8%E5%85%B3%E9%98%85%E8%AF%BB)

**要点**

冒泡排序是一种**交换排序**。

什么是交换排序呢？

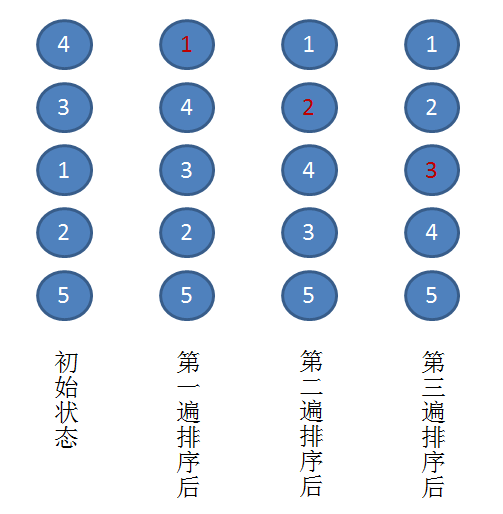
**交换排序**：两两比较待排序的关键字，并交换不满足次序要求的那对数，直到整个表都满足次序要求为止。

**算法思想**

它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，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小的元素会经由交换慢慢“浮”到数列的顶端，故名。

假设有一个大小为 N 的无序序列。冒泡排序就是要**每趟排序过程中通过两两比较，找到第 i 个小（大）的元素，将其往上排**。



**图-冒泡排序示例图**

以上图为例，演示一下冒泡排序的实际流程：

假设有一个无序序列  { 4. 3. 1. 2, 5 }

第一趟排序：通过两两比较，找到第一小的数值 1 ，将其放在序列的第一位。

第二趟排序：通过两两比较，找到第二小的数值 2 ，将其放在序列的第二位。

第三趟排序：通过两两比较，找到第三小的数值 3 ，将其放在序列的第三位。

至此，所有元素已经有序，排序结束。

要将以上流程转化为代码，我们需要像机器一样去思考，不然编译器可看不懂。

假设要对一个大小为 N 的无序序列进行升序排序（即从小到大）。

(1) 每趟排序过程中需要通过比较找到第 i 个小的元素。

所以，我们需要一个**外部循环**，从数组首端(下标 0) 开始，一直扫描到倒数第二个元素（即下标 N - 2) ，剩下最后一个元素，必然为最大。

(2) 假设是第 i 趟排序，可知，前 i-1 个元素已经有序。现在要找第 i 个元素，只需从数组末端开始，扫描到第 i 个元素，将它们两两比较即可。

所以，需要一个**内部循环**，从数组末端开始（下标 N - 1），扫描到 (下标 i + 1)。

**核心代码**

[复制代码](javascript:void(0);)

public void bubbleSort(int[] list) {  
    int temp = 0; // 用来交换的临时数  
   
    // 要遍历的次数  
    for (int i = 0; i < list.length - 1; i++) {  
        // 从后向前依次的比较相邻两个数的大小，遍历一次后，把数组中第i小的数放在第i个位置上  
        for (int j = list.length - 1; j > i; j--) {  
            // 比较相邻的元素，如果前面的数大于后面的数，则交换  
            if (list[j - 1] > list[j]) {  
                temp = list[j - 1];  
                list[j - 1] = list[j];  
                list[j] = temp;  
            }  
        }  
   
        System.out.format("第 %d 趟：\t", i);  
        printAll(list);  
    }  
}

[复制代码](javascript:void(0);)

**算法分析**

**冒泡排序算法的性能**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 排序类别 | 排序方法 | 时间复杂度 | | | 空间复杂度 | 稳定性 | 复杂性 |
| 平均情况 | 最坏情况 | 最好情况 |
| 交换排序 | 冒泡排序 | O(N2) | O(N2) | O(N) | O(1) | 稳定 | 简单 |

**时间复杂度**

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C和记录移动次数M均达到最小值：Cmin = N - 1, Mmin = 0。所以，冒泡排序**最好时间复杂度**为O(N)。  
若初始文件是**反序**的，需要进行 N -1 趟排序。每趟排序要进行 N - i 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：  
Cmax = N(N-1)/2 = O(N2)  
Mmax = 3N(N-1)/2 = O(N2)  
冒泡排序的**最坏时间复杂度**为O(N2)。  
因此，冒泡排序的**平均时间复杂度**为O(N2)。  
总结起来，其实就是一句话：当数据越接近正序时，冒泡排序性能越好。

**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**。

**优化**

对冒泡排序常见的改进方法是加入**标志性变量exchange**，用于标志某一趟排序过程中是否有数据交换。

如果进行**某一趟排序时并没有进行数据交换，则说明所有数据已经有序**，可立即结束排序，避免不必要的比较过程。

**核心代码**

[复制代码](javascript:void(0);)

// 对 bubbleSort 的优化算法  
public void bubbleSort\_2(int[] list) {  
    int temp = 0; // 用来交换的临时数  
    boolean bChange = false; // 交换标志  
   
    // 要遍历的次数  
    for (int i = 0; i < list.length - 1; i++) {  
        bChange = false;  
        // 从后向前依次的比较相邻两个数的大小，遍历一次后，把数组中第i小的数放在第i个位置上  
        for (int j = list.length - 1; j > i; j--) {  
            // 比较相邻的元素，如果前面的数大于后面的数，则交换  
            if (list[j - 1] > list[j]) {  
                temp = list[j - 1];  
                list[j - 1] = list[j];  
                list[j] = temp;  
                bChange = true;  
            }  
        }  
   
        // 如果标志为false，说明本轮遍历没有交换，已经是有序数列，可以结束排序  
        if (false == bChange)  
            break;  
   
        System.out.format("第 %d 趟：\t", i);  
        printAll(list);  
    }  
}

[复制代码](javascript:void(0);)

**完整参考代码**

**JAVA版本**

**代码实现**



[复制代码](javascript:void(0);)

 1 package notes.javase.algorithm.sort;  
 2    
 3 import java.util.Random;  
 4    
 5 public class BubbleSort {  
 6    
 7     public void bubbleSort(int[] list) {  
 8         int temp = 0; // 用来交换的临时数  
 9    
10         // 要遍历的次数  
11         for (int i = 0; i < list.length - 1; i++) {  
12             // 从后向前依次的比较相邻两个数的大小，遍历一次后，把数组中第i小的数放在第i个位置上  
13             for (int j = list.length - 1; j > i; j--) {  
14                 // 比较相邻的元素，如果前面的数大于后面的数，则交换  
15                 if (list[j - 1] > list[j]) {  
16                     temp = list[j - 1];  
17                     list[j - 1] = list[j];  
18                     list[j] = temp;  
19                 }  
20             }  
21    
22             System.out.format("第 %d 趟：\t", i);  
23             printAll(list);  
24         }  
25     }  
26    
27     // 对 bubbleSort 的优化算法  
28     public void bubbleSort\_2(int[] list) {  
29         int temp = 0; // 用来交换的临时数  
30         boolean bChange = false; // 交换标志  
31    
32         // 要遍历的次数  
33         for (int i = 0; i < list.length - 1; i++) {  
34             bChange = false;  
35             // 从后向前依次的比较相邻两个数的大小，遍历一次后，把数组中第i小的数放在第i个位置上  
36             for (int j = list.length - 1; j > i; j--) {  
37                 // 比较相邻的元素，如果前面的数大于后面的数，则交换  
38                 if (list[j - 1] > list[j]) {  
39                     temp = list[j - 1];  
40                     list[j - 1] = list[j];  
41                     list[j] = temp;  
42                     bChange = true;  
43                 }  
44             }  
45    
46             // 如果标志为false，说明本轮遍历没有交换，已经是有序数列，可以结束排序  
47             if (false == bChange)  
48                 break;  
49    
50             System.out.format("第 %d 趟：\t", i);  
51             printAll(list);  
52         }  
53     }  
54    
55     // 打印完整序列  
56     public void printAll(int[] list) {  
57         for (int value : list) {  
58             System.out.print(value + "\t");  
59         }  
60         System.out.println();  
61     }  
62    
63     public static void main(String[] args) {  
64         // 初始化一个随机序列  
65         final int MAX\_SIZE = 10;  
66         int[] array = new int[MAX\_SIZE];  
67         Random random = new Random();  
68         for (int i = 0; i < MAX\_SIZE; i++) {  
69             array[i] = random.nextInt(MAX\_SIZE);  
70         }  
71    
72         // 调用冒泡排序方法  
73         BubbleSort bubble = new BubbleSort();  
74         System.out.print("排序前:\t");  
75         bubble.printAll(array);  
76         // bubble.bubbleSort(array);  
77         bubble.bubbleSort\_2(array);  
78         System.out.print("排序后:\t");  
79         bubble.printAll(array);  
80     }  
81 }

[复制代码](javascript:void(0);)

**运行结果**

[复制代码](javascript:void(0);)

排序前:      2    9    9    7    1    9    0    2    6    8     
第 0 趟：    0    2    9    9    7    1    9    2    6    8     
第 1 趟：    0    1    2    9    9    7    2    9    6    8     
第 2 趟：    0    1    2    2    9    9    7    6    9    8     
第 3 趟：    0    1    2    2    6    9    9    7    8    9     
第 4 趟：    0    1    2    2    6    7    9    9    8    9     
第 5 趟：    0    1    2    2    6    7    8    9    9    9     
排序后:      0    1    2    2    6    7    8    9    9    9

[复制代码](javascript:void(0);)