冒泡排序（Bubble Sort）算法

杨其臻

Sep 06, 2025

想象一下，如何将一堆大小不一的泡泡按从小到大的顺序排列？最直观的方法是不是从左到右，让相邻的大泡泡和小泡泡交换位置，大的往右“浮”？这种像泡泡一样，通过重复地交换相邻元素来排序的算法，就是今天的主角 —— 冒泡排序。本文将带你从零开始，彻底弄懂冒泡排序的原理、实现、性能，并探讨其优化策略和应用场景，帮助读者不仅写出代码，更能透彻理解其背后的思想。

1　算法核心思想与工作原理

冒泡排序的核心思想基于重复遍历待排序列表，每次遍历时依次比较相邻的两个元素，如果它们的顺序错误（即前面的元素大于后面的元素），就交换它们的位置。每完成一轮完整的遍历，当前未排序部分中最大的元素就会“冒”到它最终的正确位置，即列表的末尾。这个过程类似于泡泡在水中上浮，因此得名冒泡排序。
为了更直观地理解，让我们以一个具体数组例子进行分步演示，例如数组 [5，3，8，6，4]。在第一轮遍历中，首先比较索引 0 和 1 的元素 5 和 3，由于 5 大于 3，顺序错误，因此交换位置，数组变为 [3，5，8，6，4]。接着比较索引 1 和 2 的元素 5 和 8，顺序正确，不交换。然后比较索引 2 和 3 的元素 8 和 6，顺序错误，交换位置，数组变为 [3，5，6，8，4]。最后比较索引 3 和 4 的元素 8 和 4，顺序错误，交换位置，数组变为 [3，5，6，4，8]。此时第一轮结束，最大值 8 已就位。后续轮次类似，每轮减少一个元素的比较范围，直到整个数组有序。

2　代码实现

以下是冒泡排序的基础版本（未优化）的 Python 实现。我们将提供代码并逐行解析其逻辑。

```
def bubble_sort_basic(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(0, n-1-i):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

在这段代码中，函数 bubble_sort_basic 接受一个列表 arr 作为输入。变量 n 存储列表的长度。外层循环 for i in range(n-1) 控制排序的轮数，由于每轮会将一个最大元素移动到末尾，因此总共需要 n-1 轮遍历。内层循环 for j in range(0, n-1-i) 负责在每轮中进行相邻元素的比较和交换，其中 n-1-i 表示每轮后未

排序部分的减少，因为末尾的 i 个元素已经有序。核心逻辑是 `if arr[j] > arr[j+1]` 判断，如果前一个元素大于后一个元素，则通过元组交换 `arr[j], arr[j+1] = arr[j+1], arr[j]` 实现位置交换。最终返回排序后的数组。

3 算法分析：透视其性能

冒泡排序的时间复杂度分析显示，在最坏情况下，即数组完全逆序时，需要进行 $\frac{n(n-1)}{2}$ 次比较和交换，因此时间复杂度为 $O(n^2)$。在平均情况下，由于元素顺序随机，时间复杂度同样为 $O(n^2)$。在最好情况下，如果数组已有序，基础版本仍会进行全部轮次的比较，但通过优化（如设置标志位），最好情况可提升至 $O(n)$。空间复杂度方面，冒泡排序是原地排序算法，只使用常数级额外空间用于临时交换，因此空间复杂度为 $O(1)$。稳定性方面，冒泡排序是稳定排序算法，因为只有当相邻元素大小严格大于时才交换，相等的元素不会交换，相对顺序得以保持。

4 优化策略：让冒泡排序“聪明”一点

基础版本的冒泡排序在数组已有序时仍会继续执行无用的遍历，效率低下。为此，我们引入优化方案：设置标志位（Flag）。思路是在一轮遍历开始前，初始化一个标志 swapped 为 `False`，如果本轮发生任何交换，则置为 `True`。一轮结束后检查标志位，如果为 `False`，说明数组已完全有序，可以提前终止算法。
以下是优化后的代码示例：

```
def bubble_sort_optimized(arr):
    n = len(arr)
    for i in range(n-1):
        swapped = False
        for j in range(0, n-1-i):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
```

在这段优化代码中，外层循环和内层循环的结构与基础版本相同，但增加了 swapped 变量来跟踪是否发生交换。每轮开始前，swapped 初始化为 `False`。在内层循环中，如果发生交换，swapped 被设置为 `True`。一轮结束后，通过 `if not swapped` 判断，如果没有交换发生，则使用 `break` 语句提前退出循环，从而优化最好情况下的性能，将时间复杂度从 $O(n^2)$ 提升到 $O(n)$。
冒泡排序的优点包括算法思想极其简单，易于理解和实现；它是原地排序算法，空间复杂度低；且具有稳定性，能保持相等元素的相对顺序。然而，其缺点也很明显：时间复杂度太高，效率低下，尤其不适合处理大规模数据。应用场景主要集中在教学目的，作为介绍排序算法概念的起点；或用于简单且数据量极小的任务；偶尔作为其他复杂算法的一小部分。但在实际项目开发和算法竞赛中，几乎永远不会使用冒泡排序，推荐选择更高效的算法如快速排序、归并排序或 Python 内置的 Timsort。

回顾冒泡排序的核心，它通过重复比较相邻元素并交换来实现排序，机制简单但效率有限。本文强调了其作为教学工具和简单场景算法的定位，帮助读者从原理到代码全面理解。未来，我们将探讨更高效的排序算法，如快速排序，以激发进一步学习的兴趣。

5 互动环节

挑战题部分，读者可以尝试用冒泡排序对字符串数组按字典序进行排序，或实现一个“下沉”版本的冒泡排序（即每轮将最小的元素移到最前面）。欢迎在评论区分享学习过程中遇到的问题或讨论其他基于比较的排序算法。