11.11 小结

1. 重点回顾

- 冒泡排序通过交换相邻元素来实现排序。通过添加一个标志位来实现提前返回，我们可以将冒泡排序的最佳时间复杂度优化到 $O(n)$ 。
- 插入排序每轮将未排序区间内的元素插入到已排序区间的正确位置，从而完成排序。虽然插入排序的时间复杂度为 $O(n^2)$ ，但由于单元操作相对较少，因此在小数据量的排序任务中非常受欢迎。
- 快速排序基于哨兵划分操作实现排序。在哨兵划分中，有可能每次都选取到最差的基准数，导致时间复杂度劣化至 $O(n^2)$ 。引入中位数基准数或随机基准数可以降低这种劣化的概率。尾递归方法可以有效地减少递归深度，将空间复杂度优化到 $O(\log n)$ 。
- 归并排序包括划分和合并两个阶段，典型地体现了分治策略。在归并排序中，排序数组需要创建辅助数组，空间复杂度为 $O(n)$ ；然而排序链表的空间复杂度可以优化至 $O(1)$ 。
- 桶排序包含三个步骤：数据分桶、桶内排序和合并结果。它同样体现了分治策略，适用于数据体量很大的情况。桶排序的关键在于对数据进行平均分配。
- 计数排序是桶排序的一个特例，它通过统计数据出现的次数来实现排序。计数排序适用于数据量大但数据范围有限的情况，并且要求数据能够转换为正整数。
- 基数排序通过逐位排序来实现数据排序，要求数据能够表示为固定位数的数字。
- 总的来说，我们希望找到一种排序算法，具有高效率、稳定、原地以及正向自适应性等优点。然而，正如其他数据结构和算法一样，没有一种排序算法能够同时满足所有这些条件。在实际应用中，我们需要根据数据的特性来选择合适的排序算法。
- 图 11-19 对比了主流排序算法的效率、稳定性、就地性和自适应性等。

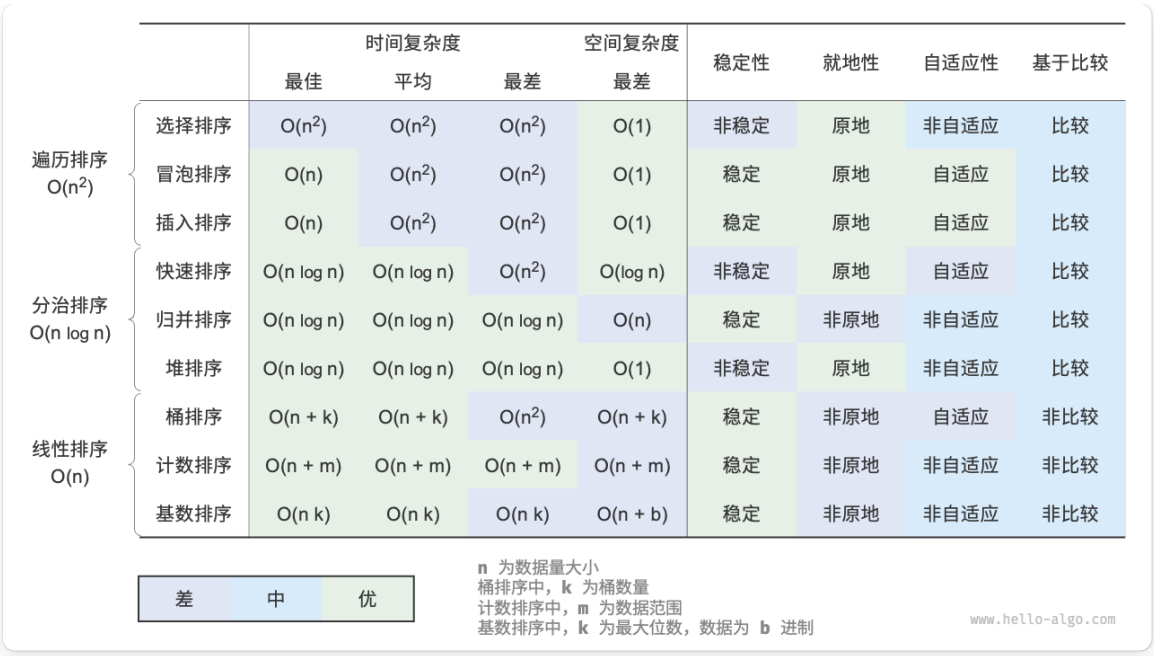

图 11-19 排序算法对比

2. Q & A

Q：排序算法稳定性在什么情况下是必需的？

在现实中，我们有可能基于对象的某个属性进行排序。例如，学生有姓名和身高两个属性，我们希望实现一个多级排序：先按照姓名进行排序，得到 `(A, 180) (B, 185) (C, 170) (D, 170)` ；再对身高进行排序。由于排序算法不稳定，因此可能得到 `(D, 170) (C, 170) (A, 180) (B, 185)` 。

可以发现，学生 D 和 C 的位置发生了交换，姓名的有序性被破坏了，而这是我们不希望看到的。

Q：哨兵划分中“从右往左查找”与“从左往右查找”的顺序可以交换吗？

不行，当我们以最左端元素为基准数时，必须先“从右往左查找”再“从左往右查找”。这个结论有些反直觉，我们来剖析一下原因。

哨兵划分 `partition()` 的最后一步是交换 `nums[left]` 和 `nums[i]` 。完成交换后，基准数左边的元素都 `<=` 基准数，**这就要求最后一步交换前 `nums[left] >= nums[i]` 必须成立**。假设我们先“从左往右查找”，那么如果找不到比基准数更大的元素，**则会在 `i == j` 时跳出循环，此时可能 `nums[j] == nums[i] > nums[left]`** 。也就是说，此时最后一步交换操作会把一个比基准数更大的元素交换至数组最左端，导致哨兵划分失败。

举个例子，给定数组 `[0, 0, 0, 0, 1]` ，如果先“从左向右查找”，哨兵划分后数组为 `[1, 0, 0, 0, 0]` ，这个结果是不正确的。

再深入思考一下，如果我们选择 `nums[right]` 为基准数，那么正好反过来，必须先“从左往右查找”。

Q：关于尾递归优化，为什么选短的数组能保证递归深度不超过 $\log n$ ？

递归深度就是当前未返回的递归方法的数量。每轮哨兵划分我们将原数组划分为两个子数组。在尾递归优化后，向下递归的子数组长度最大为原数组长度的一半。假设最差情况，一直为一半长度，那么最终的递归深度就是 $\log n$ 。

回顾原始的快速排序，我们有可能会连续地递归长度较大的数组，最差情况下为 n、$n - 1$、$\dots$、2、1 ，递归深度为 n 。尾递归优化可以避免这种情况出现。

Q：当数组中所有元素都相等时，快速排序的时间复杂度是 $O(n^2)$ 吗？该如何处理这种退化情况？

是的。对于这种情况，可以考虑通过哨兵划分将数组划分为三个部分：小于、等于、大于基准数。仅向下递归小于和大于的两部分。在该方法下，输入元素全部相等的数组，仅一轮哨兵划分即可完成排序。

Q：桶排序的最差时间复杂度为什么是 $O(n^2)$ ？

最差情况下，所有元素被分至同一个桶中。如果我们采用一个 $O(n^2)$ 算法来排序这些元素，则时间复杂度为 $O(n^2)$ 。

上一页 ← 11.10 基数排序

下一页 第 12 章 分治 →

欢迎在评论区留下你的见解、问题或建议