基本的基数排序（Radix Sort）算法

叶家炜

Oct 11, 2025

在计算机科学中，排序算法是基础且重要的研究主题。想象一下，当我们面对一组数字，例如学生的学号「102」、「031」、「215」、「123」，如何高效地对它们进行排序？传统的基于比较的排序算法，如快速排序或归并排序，虽然通用性强，但其时间复杂度下界为 $O(n \log n)$。是否存在一种方法能够突破这一限制？答案是肯定的，基数排序（Radix Sort）作为一种非比较型整数排序算法，在某些条件下可以达到 $O(n \times k)$ 的线性时间复杂度，其中 k 是数字的最大位数。本文将带领读者深入理解基数排序的核心思想、完整流程、代码实现，并分析其优劣与应用场景。

1　为什么我们需要基数排序?

排序问题在现实世界中无处不在，从数据库查询到大数据处理，高效排序至关重要。基于比较的排序算法通过元素间的直接比较来确定顺序，但其性能受限于 $O(n \log n)$ 的理论下界。当数据量巨大时，这一限制可能成为瓶颈。基数排序则另辟蹊径，它不依赖于元素间的直接比较，而是利用数字的位结构进行排序，从而在特定场景下实现线性时间复杂度。例如，对于整数数据集，尤其是当数字范围相对集中且位数较小时，基数排序的性能优势显著。这不仅提升了效率，还拓宽了排序算法的应用边界。

基数排序的核心价值在于其非比较特性。它通过逐位处理元素，将排序问题分解为多个稳定的子排序过程。这种方法的灵感来源于日常生活中的分类逻辑，例如在整理文件时先按类别再按日期排序。基数排序正是将这种分层思想应用于数字排序，从而避免了直接比较的整体复杂度。理解基数排序不仅有助于掌握一种高效算法，还能深化对数据结构和算法设计的认识。

2　什么是基数排序?

基数排序中的「基数」指的是数字的进制基数。在十进制系统中，基数为 10，每一位数字的取值范围是 0 到 9。基数排序的基本思想是将待排序的整数视为由多个位组成的序列，然后从最低位到最高位依次对每一位进行排序。每一轮排序必须使用稳定的排序算法，以确保在后续排序中低位顺序不被破坏。

稳定性是基数排序成功的关键。例如，假设有两个数字 23 和 25，在对十位排序时，如果使用非稳定排序，可能会破坏个位已排好的顺序。稳定排序保证了相同高位的元素其低位的相对顺序不变。这类似于扑克牌排序：先按花色排序，再按数字排序，第二次排序时必须确保相同数字的牌其花色的顺序保持不变。基数排序通过这种逐位稳定的方式，最终实现整体有序。

基数排序适用于整数或可分解为位序列的数据类型。其核心在于将复杂排序问题简化为多个简单的子问题，每一子问题仅处理一个位上的数字。这种分解不仅降低了时间复杂度，还使得算法易于理解和实现。需要注意的是，基数排序通常从最低位开始（LSD，Least Significant Digit），但也可以从最高位开始（MSD，Most

Significant Digit），后者适用于递归实现。

3 算法流程详解：一步步拆解基数排序

基数排序的流程可以分为几个关键步骤。首先，需要确定待排序数组中的最大值，以计算最大位数 d。例如，对于数组 [170，45，75，90，2，802，24，66]，最大值为 802，其位数为 3，因此需要进行三轮排序。接下来，初始化一个过程数组用于存放中间结果。然后，从最低位（个位）开始，到最高位（百位）结束，依次执行分配和收集操作。

分配阶段，我们根据当前位的数字将元素分配到对应的桶中。桶的数量等于基数，在十进制中为 10 个（0 到 9）。收集阶段，则按照桶的顺序（从 0 到 9）将元素依次取出，放回过程数组。以数组 [170，45，75，90，2，802，24，66] 为例，第一轮按个位排序：170 的个位是 0，45 的个位是 5，75 的个位是 5，90 的个位是 0，2 的个位是 2，802 的个位是 2，24 的个位是 4，66 的个位是 6。分配后，桶 0 包含 170 和 90，桶 2 包含 2 和 802，桶 4 包含 24，桶 5 包含 45 和 75，桶 6 包含 66。收集后数组变为 [170，90，2，802，24，45，75，66]。

第二轮按十位排序：170 的十位是 7，90 的十位是 9，2 的十位是 0（视为 0），802 的十位是 0，24 的十位是 2，45 的十位是 4，75 的十位是 7，66 的十位是 6。分配后，桶 0 包含 2 和 802，桶 2 包含 24，桶 4 包含 45，桶 6 包含 66，桶 7 包含 170 和 75，桶 9 包含 90。收集后数组变为 [802，2，24，45，66，170，75，90]。第三轮按百位排序：802 的百位是 8，2 的百位是 0，24 的百位是 0，45 的百位是 0，66 的百位是 0，170 的百位是 1，75 的百位是 0，90 的百位是 0。分配后，桶 0 包含 2、24、45、66、75、90，桶 1 包含 170，桶 8 包含 802。收集后最终数组为 [2，24，45，66，75，90，170，802]，排序完成。

4 关键实现：使用计数排序作为子程序

在基数排序中，每一轮的排序子程序必须稳定且高效。计数排序（Counting Sort）是理想选择，因为当数据范围较小（0 到 9）时，其时间复杂度为 $O(n+k)$，且具有稳定性。计数排序通过统计每个数字的出现次数，并利用前缀和确定元素位置，从而保证排序的稳定性。

以下是基数排序的 Python 实现，附有详细注释。代码包括主函数 `radix_sort` 和子函数 `counting_sort_for_radix`，后者用于对特定位进行排序。

```
def radix_sort(arr):
    # 查找数组中的最大值，以确定最大位数
    max_val = max(arr)
    exp = 1 # 从个位开始
    # 循环直到处理完最高位
    while max_val // exp > 0:
        counting_sort_for_radix(arr, exp)
        exp *= 10 # 移动到下一位（十位、百位等）

def counting_sort_for_radix(arr, exp):
    n = len(arr)
    output = [0] * n # 输出数组，用于存放排序结果
```

```
    count = [0] * 10 # 计数数组，索引 0 到 9，对应当前位的数字
    # 统计当前位上每个数字的出现次数
    for i in range(n):
        index = (arr[i] // exp) % 10 # 提取当前位的数字
        count[index] += 1
    # 将计数数组转换为前缀和数组，以确定每个数字在输出数组中的最终位置
    for i in range(1, 10):
        count[i] += count[i - 1]
    # 逆向遍历原数组，将元素放置到输出数组的正确位置，保证稳定性
    for i in range(n - 1, -1, -1):
        index = (arr[i] // exp) % 10
        output[count[index] - 1] = arr[i] # 根据计数数组放置元素
        count[index] -= 1 # 更新计数，为相同数字的下一个元素预留位置
    # 将排序好的输出数组拷贝回原数组
    for i in range(n):
        arr[i] = output[i]
```

在 radix_sort 函数中，首先通过 max(arr) 获取最大值，并初始化 exp 为 1，表示从个位开始处理。循环条件 max_val // exp > 0 确保处理完所有位数。每次循环调用 counting_sort_for_radix 对当前位进行排序，然后 exp 乘以 10 以移动到更高位。

在 counting_sort_for_radix 函数中，首先初始化输出数组和计数数组。遍历原数组，使用 (arr[i] // exp) % 10 提取当前位的数字，并更新计数数组。接着，将计数数组转换为前缀和数组，这样 count[i] 就表示数字 i 及更小数字的累计次数，从而确定元素在输出数组中的位置。逆向遍历原数组是为了保证稳定性：当多个元素具有相同当前位数字时，后出现的元素在输出数组中会被放置在较后位置，从而维持原有顺序。最后，将输出数组拷贝回原数组，完成当前位的排序。

5 深度分析与探讨

基数排序的时间复杂度为 $O(d \times (n + k))$，其中 d 是最大位数，n 是元素个数，k 是基数（例如 10）。当 d 较小且 n 较大时，基数排序的性能优于基于比较的排序算法。空间复杂度为 $O(n + k)$，主要来自计数排序中的输出数组和计数数组。

基数排序的优点包括速度快（在特定条件下）、稳定性高，适用于整数排序。缺点则是只能处理整数或可分解为位的数据类型，需要额外空间，且当数字范围极大（d 很大）时，效率可能下降。与快速排序和归并排序相比，基数排序在整数排序场景下更具优势，但通用性较差。

基数排序的变体包括 MSD（最高位优先）基数排序，它从最高位开始排序，适用于递归实现，但可能更复杂。处理负数时，可以将数组分为正负两部分，负数取绝对值排序后再反转。基数排序还可扩展至字符串排序，通过逐字符处理实现字典序排序。

基数排序通过逐位稳定的排序方式，实现了整数的高效排序。其核心思想是将排序问题分解为多个子问题，每一子问题处理一个位上的数字。使用计数排序作为子程序，保证了算法的稳定性和性能。基数排序在特定场景下展现出线性时间复杂度的优势，为大数据处理和数据库索引等领域提供了实用工具。

6 互动与思考题

请尝试用您熟悉的编程语言实现上述基数排序算法，并用一个包含负数的数组测试其表现。思考一下，如果要用基数排序给日期（格式为 YYYYMMDD）排序，应该如何设计算法？欢迎在评论区分享您的疑问或应用场景！