深入理解并实现基本的希尔排序（Shell Sort）算法

黄梓淳

Sep 08, 2025

在计算机科学中，排序算法是基础且重要的主题。插入排序作为一种简单的排序方法，对于小规模或基本有序的数组表现出较高的效率，因为它通过逐个比较和移动元素来工作。然而，插入排序的局限性在于处理大规模乱序数组时，其时间复杂度为 $O(n^2)$，这主要是由于元素只能一位一位地移动，导致性能低下。为了解决这一问题，Donald Shell 在 1959 年提出了希尔排序，这是一种对插入排序的改进版本。希尔排序的核心思想是允许元素进行「长距离跳跃」，通过预处理使数组更快达到基本有序状态，从而显著提升排序效率。简而言之，希尔排序可以视为插入排序的「预排序」优化版。

1　希尔排序的核心思想与工作原理

希尔排序的工作原理基于一个关键概念：增量（Gap）。增量用于划分子序列，例如，当增量为 gap 时，数组会被分成多个子序列，每个子序列由间隔为 gap 的元素组成。具体来说，对于一个数组，当 gap=5 时，索引为 0、5、10 等的元素形成一个子序列，索引为 1、6、11 等的元素形成另一个子序列，以此类推。算法步骤包括：首先选择一个增量序列（如初始 gap 为 n/2），然后按当前增量将数组分割成子序列，并对每个子序列进行插入排序；之后缩小增量（如 gap/2），重复这个过程，直到增量为 1；最后，对整个数组进行一次插入排序，此时数组已基本有序，插入排序效率很高。

为了更直观地理解，让我们以一个具体数组为例进行演示。考虑数组 [9，8，7，6，5，4，3，2，1]，初始长度 n=9，因此初始 gap 为 4（n/2 取整）。首先，以 gap=4 划分子序列：子序列 1 包括索引 0、4、8 的元素（9, 5, 1），子序列 2 包括索引 1、5 的元素（8, 4），子序列 3 包括索引 2、6 的元素（7, 3），子序列 4 包括索引 3、7 的元素（6, 2）。对每个子序列进行插入排序后，数组变为 [1，4，3，2，5，8，7，6，9]。接下来，缩小 gap 到 2，重新划分子序列并排序，数组进一步优化。最后，gap=1 时进行最终插入排序，得到有序数组 [1，2，3，4，5，6，7，8，9]。这个过程展示了希尔排序如何通过大步距移动元素来加速排序。

2　关键问题：增量序列的选择

增量序列的选择对希尔排序的性能有显著影响。不同的序列会导致不同的时间复杂度。希尔原始序列使用简单的除法策略，如 gap = n/2, n/4, ..., 1，虽然易于理解和实现，但在最坏情况下时间复杂度仍为 $O(n^2)$，因此不是最优选择。为了提高性能，研究者提出了其他序列，例如 Hibbard 序列，其形式为 1, 3, 7, 15, ..., $2^k - 1$，这种序列可以将最坏情况时间复杂度优化到 $O(n^{3/2})$。另一个著名的是 Sedgewick 序列，它通过更复杂的计算提供更好的平均性能。在本文中，为了教学和实现简单，我们将使用希尔原始序列，即 gap 初始为 n/2，然后逐步缩小到 1。

3 代码实现

我们选择 Python 作为实现语言，因为其代码简洁易懂，适合教学目的。以下是希尔排序的 Python 实现，附有详细注释。

```
def shell_sort(arr):
    n = len(arr)
    # 初始增量 gap 设置为数组长度的一半
    gap = n // 2

    # 循环直到 gap 大于 0
    while gap > 0:
        # 从第 gap 个元素开始，对各个子序列进行插入排序
        for i in range(gap, n):
            temp = arr[i] # 当前需要插入的元素
            j = i
            # 在子序列中，进行插入排序：将比 temp 大的元素向后移动 gap 位
            while j >= gap and arr[j - gap] > temp:
                arr[j] = arr[j - gap]
                j -= gap
            # 将 temp 插入到正确位置
            arr[j] = temp
        # 缩小增量
        gap //= 2
    return arr
```

现在，让我们逐行解析这段代码。函数 shell_sort 接受一个数组 arr 作为输入。首先，获取数组长度 n，并设置初始 gap 为 n // 2，即整数除法的一半。外层 while 循环控制增量的变化，只要 gap 大于 0 就继续执行。内层 for 循环从索引 gap 开始遍历数组，这是为了处理每个子序列的元素。变量 temp 存储当前需要插入的元素值，j 初始化为当前索引 i。内层 while 循环执行插入排序的核心操作：如果 j 大于等于 gap 且前一个子序列元素（索引 j - gap）大于 temp，则将较大的元素向后移动 gap 位，并递减 j 以继续比较。循环结束后，将 temp 插入到正确位置 j。完成内层循环后，缩小 gap 为原来的一半，重复过程直到 gap 为 0。最终返回排序后的数组。这段代码实现了希尔排序的基本逻辑，通过分组插入排序来优化性能。

4 算法分析

希尔排序的时间复杂度分析较为复杂，因为它依赖于所选的增量序列。使用希尔原始序列时，最坏情况时间复杂度为 $O(n^2)$，这与插入排序相同，但平均情况通常优于插入排序。如果使用优化序列如 Hibbard 序列，最坏情况可改善到 $O(n^{3/2})$，最佳情况甚至可以达到 $O(n \log n)$。总体而言，平均时间复杂度大约在 $O(n^{13/10})$ 到 $O(n^2)$ 之间，具体取决于数据分布和序列选择。空间复杂度方面，希尔排序是原地排序算法，只需要常数级的额

外空间（即 $O(1)$），因此非常高效 in terms of memory usage。然而，希尔排序是不稳定的排序算法，原因在于相同的元素可能在子序列排序过程中被打乱相对顺序，例如，如果两个相同值的元素位于不同的子序列，它们的顺序可能因移动而改变。

希尔排序的优点包括：它是插入排序的高效改进版，算法简单易于实现，对于中等大小的数组表现良好，且空间复杂度低（$O(1)$）。这些特点使其在资源受限的环境中有一定应用价值。缺点则在于时间复杂度依赖于增量序列，分析复杂，在最坏情况下性能可能不佳，而且不如快速排序、堆排序或归并排序等高级算法快，因此通常被视为教学算法而非生产环境的首选。

总之，希尔排序的核心思想是通过分组插入排序和逐步缩小增量来优化排序过程。它适用于中等规模、非性能临界的数据排序，例如在嵌入式系统或内存受限的环境中，由于其简单的实现和低空间开销，可能被采用。然而，在大多数现代应用中，更高效的算法如快速排序或归并排序更受青睐。希尔排序更多地作为一种教学工具，帮助我们理解算法改进的思路。

5 互动与思考题

读者可以尝试修改代码，使用不同的增量序列（如 `gap // 2 + 1`）并观察性能差异，这有助于深入理解增量序列的影响。同时，思考一个问题：「希尔排序为什么是不稳定的？你能举出一个例子吗？」例如，考虑数组 [3, 2, 2, 1] 使用 gap=2 进行排序，可能打乱相同元素的顺序。对于扩展阅读，推荐了解其他排序算法如快速排序和归并排序，以 broaden your knowledge。