归并排序（Merge Sort）算法

杨其臻

Aug 29, 2025

排序算法是计算机科学中最基础且重要的研究领域之一，它不仅是编程入门的关键课题，更是评估算法效率与设计思想的经典案例。在众多排序算法中，归并排序凭借其稳定的 $O(n \log n)$ 时间复杂度以及典型的分治策略，成为了理论和实践中不可或缺的一部分。本文将引导您不仅实现归并排序，更深入理解其背后的分治哲学和运作机制。

1　归并排序的核心思想：分而治之

归并排序的核心是“分治”策略，这是一种将复杂问题分解为多个相同或相似的子问题，再递归解决子问题，最后合并子问题解以得到原问题解的方法。想象一下组织一场大型晚会：您不会试图一次性管理所有细节，而是将任务分解为场地、节目、餐饮等子任务，分别处理后再整合成果。归并排序正是如此运作，它通过递归将数组不断二分，直到每个子数组只含一个元素（自然有序），再通过合并操作将这些有序片段组装成更大的有序数组。
具体而言，归并排序遵循三步战略：分解、解决和合并。分解阶段将数组递归地分成两个尽可能等长的子数组；解决阶段递归排序这些子数组；合并阶段则将两个已排序的子数组合并为一个有序数组。整个算法的效率与正确性高度依赖于合并过程的实现。

2　深入剖析：合并两个有序数组的过程

合并两个有序数组是归并排序的灵魂所在。假设有两个已排序数组 A 和 B，目标是将它们合并为新的有序数组 C。这一过程通过三个指针高效完成：指针 i 遍历 A，指针 j 遍历 B，指针 k 指向 C 的当前写入位置。
合并从比较 A[i] 和 B[j] 开始，将较小者放入 C[k]，并移动相应指针。此过程循环直至任一数组被完全遍历，最后将另一数组的剩余元素直接追加至 C 末尾。这种策略确保了合并操作的时间复杂度为 $O(n)$，其中 n 是两个子数组的长度之和。
以下是一个 Python 的 merge 函数实现，它清晰展示了这一过程：

```
def merge(left, right):
    # 初始化结果列表和指针
    result = []
    i = j = 0
    # 循环比较并合并
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
```

```
            i += 1
        else:
            result.append(right[j])
            j += 1
    # 将剩余元素追加到结果中
    result.extend(left[i:])
    result.extend(right[j:])
    return result
```

这段代码首先比较左右数组的当前元素，选择较小者加入结果，并移动指针。循环结束后，任一数组的剩余部分直接被并入，保证了结果的完整性与有序性。

3 从思路到代码：完整的归并排序实现

基于分治思想与合并操作，归并排序的递归实现变得直观。主函数 merge_sort 首先处理递归终止条件 —— 当数组长度不大于 1 时直接返回（因为单元素数组自然有序）。否则，计算中点将数组分为左右两半，递归排序左右子数组，最后调用 merge 合并结果。

以下是 Python 的完整实现，代码注释与上述步骤一一对应：

```
def merge_sort(arr):
    # 递归终止条件：数组长度为 0 或 1
    if len(arr) <= 1:
        return arr
    # 找到中点，分解数组
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    # 合并排序后的子数组
    return merge(left, right)
```

递归深度为 $O(\log n)$，每层合并操作总时间为 $O(n)$，因此整体效率稳定。Java 的实现类似，但需处理类型声明和数组拷贝，此处略去以保持简洁。

4 复杂度分析：它为什么高效?

归并排序的时间复杂度分析可通过递归树直观理解。递归树高度为 $O(\log n)$，每层需处理 $O(n)$ 元素，故总时间为 $O(n \log n)$。这一效率在最好、最坏和平均情况下均保持一致，体现了算法的稳定性。

空间复杂度主要来自合并所需的临时数组，每层递归需 $O(n)$ 空间。由于递归深度为 $O(\log n)$，但同一时刻最大空间使用量为 $O(n)$，故归并排序的空间复杂度为 $O(n)$。与快速排序等原地排序算法相比，这是归并排序的主要缺点，但也换来了稳定性和可预测性。

归并排序的优点显著：时间复杂度稳定在 $O(n \log n)$，适用于大规模数据；它是一种稳定排序，即相等元素的相对顺序在排序后保持不变；此外，在处理链表结构时，归并排序可优化空间使用。缺点则包括需要 $O(n)$ 额外

空间，以及递归调用带来的开销，对于小规模数据，简单排序如插入排序可能更高效。

5 实战与应用

归并排序的思想远超排序本身。例如，求解逆序对问题可通过修改合并过程高效实现，统计在合并过程中右侧元素小于左侧元素的次数。现实中，归并排序的理念广泛应用于大数据处理（如 MapReduce 中的排序阶段）和编程语言基础库（如 Java 的 `Arrays.sort()` 和 Python 的 `sorted()` 在对象排序中使用变体 TimSort）。归并排序通过分治策略和高效合并，实现了稳定且高效的排序。理解其思想不仅助于掌握算法本身，更提升了解决复杂问题的能力。未来可探索优化方向，如对小数组使用插入排序减少递归开销，或实现迭代版本避免递归深度限制。

6 互动与思考

您能尝试用迭代方式实现归并排序吗？欢迎在评论区分享您的代码或提出疑问，共同探讨排序算法的更多奥秘。