基本的插入排序（Insertion Sort）算法

黄京

Sep 01, 2025

在编程和算法学习中，排序算法是基础且至关重要的部分。插入排序作为一种简单直观的算法，常常是初学者入门的第一选择。想象一下，您在整理一副扑克牌时，通常会一张一张地拿起牌，并将其插入到手中有序牌堆的适当位置。这个过程正是插入排序的核心思想。通过学习插入排序，您不仅能理解排序的基本概念，还能为后续学习更复杂的算法打下坚实基础。本文将带领您深入理解插入排序的原理，亲手实现它，并分析其性能特点。

1　算法核心思想：像整理扑克牌一样排序

插入排序的灵感来源于日常生活中的卡片整理过程。算法将待排序的数组分为两个部分：已排序区间和未排序区间。初始时，已排序区间只包含第一个元素，其余元素都属于未排序区间。然后，算法逐个从未排序区间取出元素，并将其插入到已排序区间的正确位置，通过从后向前扫描已排序区间来找到插入点。这个过程重复进行，直到未排序区间为空，数组完全有序。这种分而治之的视角使得算法易于理解和实现。

2　分步拆解：可视化排序过程

让我们通过一个具体数组示例来可视化插入排序的过程。考虑数组 [5，2，4，6，1，3]。初始状态时，已排序区间包含第一个元素 5，未排序区间包含其余元素。第一轮处理未排序区间的第一个元素 2，将其与已排序区间的 5 比较，由于 2 小于 5，我们将 5 向后移动，并将 2 插入到正确位置，得到 [2，5，4，6，1，3]。第二轮处理元素 4，它比 5 小但比 2 大，因此插入到 2 和 5 之间，数组变为 [2，4，5，6，1，3]。第三轮处理 6，它比已排序区间的所有元素都大，因此直接留在末尾，数组为 [2，4，5，6，1，3]。第四轮处理 1，这是一个关键步骤，因为它需要多次比较和移动：从后向前扫描，1 比 6、5、4、2 都小，因此这些元素依次向后移动，最后 1 插入到开头，数组变为 [1，2，4，5，6，3]。第五轮处理 3，它插入到 2 和 4 之间，最终得到有序数组 [1，2，3，4，5，6]。这个过程清晰地展示了算法如何逐步构建有序序列。

3　算法实现：手把手编码

我们将使用 Python 语言来实现插入排序，因为其语法简洁，易于理解。以下是基础版本的代码：

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
```

```
        arr[j + 1] = arr[j]
        j -= 1
    arr[j + 1] = key
return arr
```

现在，让我们逐行解析这段代码的逻辑。外层循环 `for i in range(1, len(arr))` 从索引 1 开始遍历数组，因为索引 0 的元素被视为初始已排序区间。变量 `i` 代表当前待处理元素的索引。 Inside the loop, we assign `key = arr[i]`, which is the element we are about to insert into the sorted region. Then, we set `j = i - 1` to point to the last element of the sorted region. The inner loop `while j >= 0 and key < arr[j]` is where the actual comparison and shifting happen: as long as `j` is within bounds and `key` is less than the element at `j`, we shift `arr[j]` to the right by assigning `arr[j + 1] = arr[j]`, and decrement `j` to move backwards. Once the loop exits, we have found the correct position for `key`, and we insert it with `arr[j + 1] = key`. This process ensures that each element is placed in its proper place in the sorted region.

4 算法分析：优点、缺点与适用场景

插入排序的时间复杂度分析显示，在最坏情况下，当数组完全逆序时，每个新元素都需要比较和移动所有已排序元素，导致时间复杂度为 $O(n^2)$。在最好情况下，如果数组已经有序，每个元素只需比较一次，时间复杂度为 $O(n)$，这是一个显著的优点。平均情况下，时间复杂度仍为 $O(n^2)$。空间复杂度方面，算法只使用常数级别的额外变量，如 `key` 和 `j`，因此是原地排序，空间复杂度为 $O(1)$。稳定性方面，插入排序是稳定排序，因为当遇到相等元素时，内层循环条件 `key < arr[j]` 不成立，循环停止，`key` 被插入到相等元素的后面，保持了相对顺序。优点包括实现简单、对于小规模或基本有序数据高效、空间效率高和稳定性。缺点是在大规模无序数据上效率低下。适用场景主要包括数据量较小（例如 n ⩽ 50）、数据基本有序，或作为高级排序算法（如快速排序）的子过程来处理小区间。

5 优化方向

一个常见的优化方向是使用二分查找来改进插入排序。思路是在已排序区间中使用二分查找快速定位插入位置，将查找时间从 $O(n)$ 降低到 $O(\log(n))$。然而，由于元素移动操作仍然是 $O(n)$，整体时间复杂度保持为 $O(n^2)$，但常数因子减小，在实际应用中可能带来性能提升。这可以作为进一步学习的主题。
通过本文，我们深入探讨了插入排序算法的核心思想、实现步骤和性能分析。插入排序通过构建有序序列并逐个插入元素来实现排序，其简单性和对小规模数据的效率使其成为算法学习的重要起点。理解插入排序有助于培养算法思维，并为学习更高级算法（如希尔排序或归并排序）奠定基础。

6 互动与练习

为了巩固学习，我鼓励您尝试一个挑战问题：不用临时变量 `key` 来实现插入排序，并思考这可能带来的问题，例如数据覆盖或逻辑错误。请在评论区分享您的代码或疑问，我很乐意与您交流。如果您想深入学习，推荐阅读关于其他排序算法的文章，如冒泡排序或选择排序。