第七章 排序算法2

1. 插入排序

1.1 概述

插入排序算法类似于玩扑克时抓牌的过程，玩家每拿到一张牌都要插入到手中已有的牌里，使之 从小到大排好序
基本思想
将未排序的元素逐个插入到已排序序列的适当位置，直到排序完成

1. 首先将序列中的第一个元素视为已排序部分
2. 取下一个待排元素，从后向前扫描已排序序列
3. 直到找到第一个小于或等于所取元素的位置，将所取元素插入该位置的后面，并将插入点之后 的数据依次往后移动一个位置；
4. 重复步骤2、3，直到所有元素排序完成

- 参考代码

```
void insertionSort(int arr[], int n) {
    // 从序列的第二个元素开始遍历
    for (int i = 1; i < n; i++) {
        // 将当前元素存储在临时变量key中
        int key = arr[i];
        // 初始化一个索引 j，指向当前元素的前一个位置
        int j = i - 1;
        // 如果 j 大于等于0并且前面的元素比key大时执行循环
        while (j >= 0 && arr[j] > key) {
            // 将前面的元素后移一位
            arr[j + 1] = arr[j];
            // 索引 j 向前移动一位
            j--;
        }
        // 将 key 插入到正确的位置
        arr[j + 1] = key;
        cout << "元素" << key << "插入后的结果: ";
        for (int i = 0; i < n; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
}
```

1.2 插入排序性能分析

1. 在插入排序中，最好的情况是未排序数组已经有序，只需当前数跟前一个数比较一下就可以了，这时一共需要比较n-1次。 时间复杂度为 O(n) 。
2. 当输入的数组是逆序排列时，插入排序需要每次将当前元素与前面的所有元素进行比较，并且每次比较都需要进行元素的移动操作。在最环情况下，插入排序的时间复杂度为 O（n^2)。
3. 在平均情况下，插入排序的时间复杂度也为 O（n^2），因为无论输入的数组是什么顺序，都需要比较和移动元素来保持已排序部分有序
4. 插入排序的空间复杂度为 O(1），即它只需要使用常数级别的额外空间来存储临时变量，不随输入规模的增加而增加。 这是因为插入排序是在原数组上进行操作， 不需要额外的辅助空间来存储排序结果。 因此，插入排序是一种原地排序算法
5. 插入排序是稳定的，如果未排序的序列中存在两个或两个以上具有相同关键词的数据，排序后这些数据的相对次序保持不变

1.3 插入排序总结

1. 插入排序不适合对于数据量比较大的排序应用。 但是，如果需要排序的数据量很小，例如，量 级小于千，那么插入排序还是一个不错的选择。 尤其当数据基本有序时， 采用插入排序可以明显减少数据交换和数据移动次数， 进而提升排序效率 在 STL 的 sort 算法和 stdlib qsort 算法中，都将插入排序作为快速排序的补充， 用于少量元素的排序；
2. 直接插入排序的实现是一种稳定的排序算法