“小萌，这两天没见，你都在忙什么呢？”

“我在整理老照片啊，本想按拍照日期排序，可照片太多了！头疼啊……”

“这个简单，可以用老师讲过的排序算法来实现。”

排序是使用计算机时经常进行的一种操作，其目的是将一组“无序”的数据调整为“有序”。例如，做早操时同学们要按照身高排好队列，单词表按照字母 a~z 排序，一个班级里学生名册可以按照姓名、学号排序等。

抛开算法细节不说，排序本质上就是根据某种规则来比较元素，当元素排序和规则冲突时，调整其顺序。在第 2 单元中，我们学习了如何衡量算法的计算成本，对于排序算法来讲，比较其优劣最直观的评价指标就是完成排序所需的比较次数。

当然，排序算法的执行效率和数据规模有关。很显然，“将 10 名同学按照身高排序”和“将全世界所有的城市按照人口数量排序”两个问题的难度不可同日而语。在数据量较小时，排序很简单。可随着数据量的增大，排序就需要消耗大量的计算资源。排序算法的应用可以有效提升排序操作的执行效率，而其中最常用的就是冒泡排序、选择排序和插入排序。

3.1 冒泡排序

1. 算法介绍

冒泡排序（Bubble Sort）是一种最简单的交换排序。它重复地走访过要排序的序列，一次比较两个元素，如果它们的顺序错误，就把它们交换过来。重复地进行以上操作直到没有元素再需要交换，此时序列已经排序完成。

“老师，为什么叫冒泡排序？这名字还挺可爱的！”

“我们在喝汽水时，瓶子中会有很多小气泡缓缓往上飘，这是因为小气泡是由二氧化碳组成的，它们比水要轻，所以气泡会上浮。冒泡排序的处理过程和它类似，算法中的每个元素都像气泡一样，根据规则（升序或降序）朝数列的尾部移动！”

我们通过一个例子来演示下算法的排序过程。

假设有一个序列：6 5 7 3 4，现在要将它们按照升序排列。

冒泡排序的第一轮排序过程如图 3-1 所示。

图 3-1　冒泡排序 - 升序（第一轮）

经过第一轮的比较，最大数 7 已经沉底，没有必要再参与后续处理过程。因此，在第二轮排序时可以排除已沉底的元素 7，对剩余元素（5，6，3，4）重复以上排序过程。

请大家自己试试看，在练习本上完成剩余元素的排序吧。

“老师，我排好了！总共经过了 4 轮排序，您看，34567，整整齐齐！”

“不错，所以我们可以知道，如果要对 n 个元素进行冒泡排序，那么完成最终排序就需要经过 $n-1$ 轮运算。”

现在来总结下冒泡排序的算法过程（以升序为例）。

步骤 1：比较相邻的元素。如果大的在前，就交换它们。

步骤 2：对每一对相邻元素做同样的操作，从开始第一对到结尾的最后一对，这样在最后的元素会是最大的数。

步骤 3：针对所有的元素重复以上的步骤，除了最后一个元素。

步骤 4：重复步骤 1~3，直到排序完成。

“老师，刚刚一直说的是升序，如果想降序排列呢？”

“这个简单，只要将比较的规则改一下就可以了。例如，将规则改为：比较两个数时，如果小的在前，那么就交换位置，反之则不交换。”

2. 代码实现

冒泡排序（升序）代码实现如下。

```
def bubbleSort(alist):
    # 每次需遍历元素个数递减
    for num in range(len(alist)-1, 0, -1):
        for i in range(num):
            if alist[i] > alist[i+1]:
                alist[i], alist[i+1] = alist[i+1], alist[i]
```

【问题 3-1】　请使用冒泡排序实现如下列表中元素的降序排列。

```
alist = [55, 26, 83, 17, 88, 34, 47, 59, 22]
```

3. 算法复杂度

利用第 2 单元所学知识来分析下冒泡排序的算法复杂度。对含有 n 个元素的序列排序需要 $n-1$ 轮。每一轮的比较次数如表 3-1 所示。

表 3-1　冒泡排序每一轮比较次数

轮　次	比较次数
1	$n-1$
2	$n-2$
…	…
$n-1$	1

通过观察可以发现，总的比较次数是前 $n-1$ 个数的和，即 $\frac{1}{2}n^2-\frac{1}{2}n$，所以该算法的时间复杂度为 $O(n^2)$。冒泡排序最好的情况是序列已经有序，这时不需要任何交换操作，只需一轮比较即可完成，此时的时间复杂度为 $O(n)$。相反，最坏的情况就是每一次比较都需要交换。冒泡排序不需要借助额外存储空间，其空间复杂度为 $O(1)$。

3.2 选 择 排 序

1. 算法介绍

选择排序与冒泡排序具有一定的相似性，二者的区别在于：

选择排序是根据排序规则（升序或降序）找到其中最大或最小的数，将其放到序列的末尾，而非冒泡排序中的“两两比较，随即交换”。

所以，选择排序在每轮排序中只做一次交换。为了便于理解，我们还是利用 3.1 节中的示例来演示选择排序的操作过程。

假设有一个序列：6 5 7 3 4，现在要将它们按照升序排列。

选择排序算法的排序过程如图 3-2 所示。

图 3-2　选择排序过程

“老师，我发现了，5 个数排序需要 4 轮完成！和冒泡排序时一样！”

“小萌说得对，和冒泡排序时一样，如果要对 n 个元素进行排序，那么完成最终排序就需要经过 $n-1$ 轮运算。”

选择排序的运算过程总结如下（以升序为例）。

步骤 1：从当前序列中找出最大的元素，将它与序列的队尾元素交换，如其为队尾元素，则在原位置不动。

步骤 2：针对所有的元素重复以上的步骤，除了最后一个。

步骤 3：重复步骤 1、步骤 2，直至排序完成。

请大家想一想，如果要使用选择排序实现降序排列，那么比较规则应该如何修改?

2. 代码实现

选择排序（升序）代码实现如下。

```
def selectionSort(alist):
    # 每次遍历元素个数递减
    for i in range(len(alist)-1, 0, -1):
        # 指定初始的最大值下标为 0
        max_idx = 0
        # 遍历剩余元素，依次与当前最大值比较
        for j in range(1, i+1):
            if alist[j] > alist[max_idx]:
                # 更新最大值下标
                max_idx = j
        # 将最大值与末尾元素交换
        alist[i], alist[max_idx] = alist[max_idx], alist[i]
```

【问题 3-2】 请使用选择排序实现如下列表中元素的降序排列。

```
alist = [55, 26, 83, 17, 88, 34, 47, 59, 22]
```

3. 算法复杂度

从算法原理可知，无论序列中元素是否有序，为了找出最大（最小）值，

每一轮中元素间的相互比较都是必不可少的，且其比较次数和冒泡排序算法的比较次数相同，所以不管在最好或最坏情况下，选择排序的时间复杂度都是 $O(n^2)$，但是选择排序每轮只交换一次，所以其运行速度比冒泡排序更快。选择排序无须借助额外存储空间，其空间复杂度为 $O(1)$。

3.3 直接插入排序

1. 算法介绍

在这一节中，我们再来学习插入排序。插入排序的衍生算法有很多，如直接插入排序、折半插入排序、希尔排序等，这里介绍较为基础的直接插入排序。对于少量元素的排序，它是一个有效的算法。

直接插入排序的基本思想是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。

“老师，直接插入排序是需要准备两个列表吗？一个有序，一个无序，这个算法描述有点复杂。”

“其实只用一个列表就可以了，下面我就用例子来解释一下！”

还是通过之前的例子来讲解直接插入排序的操作过程。

假设有一个数列：6 5 7 3 4，现在要将它们按照升序排列。

插入排序过程如图 3-3 所示。

直接插入排序的运算过程描述如下（以升序为例）。

步骤 1：首先将队首位置的元素视为“有序列表”中的第一个元素，其余元素视为“无序列表”。

步骤 2：将“无序列表”中的第一个元素 a 与“有序列表”中的元素从后向前依次进行比较。当“有序列表”中某个元素 b 比 a 大时，将元素 b 右移，

图 3-3　插入排序过程

继续向前进行比较，直到“有序列表”中没有比 a 大的元素时停止，将 a 插入；如果“有序列表”中没有比 a 大的元素，则将 a 插入到“有序列表”队尾；如果待插入的元素 a 与有序序列中的元素 b 相等，则将元素 a 插入到相等元素 b 的后面。

步骤 3：重复步骤 1、步骤 2，直至“无序列表”中元素为空，则排序完成。

如果要使用直接插入排序完成降序排列，那么算法要如何修改呢?

2. 代码实现

直接插入排序（升序）代码实现如下。

```
def insertSort(alist):
    # 将第一个元素视为有序，其余元素为无序，依次遍历无序元素
    for idx in range(1, len(alist)):
        # 记录当前待插入无序元素的值和下标
        current = alist[idx]
        position = idx
        # 从后向前依次与有序元素比较，如果该元素比待插入值大，将其后移，
```

```
        # 直到有序列表中没有比其大的元素时停止
        while position > 0 and alist[position-1] > current:
            alist[position] = alist[position-1]
            position = position - 1
        # 将待插入元素插入到当前位置
        alist[position] = current
```

【问题 3-3】 请使用直接插入排序实现如下列表中元素的降序排列。

```
alist = [55, 26, 83, 17, 88, 34, 47, 59, 22]
```

3. 算法复杂度

在直接插入排序中，最好的情况是待排序序列本身是有序的，只需将当前数与前一个数进行一次比较即可，此时共需要比较 $n-1$ 次，时间复杂度为 $O(n)$。

最坏的情况是待排序序列是逆序的，此时需要比较次数最多，总次数记为 $1+2+3+\cdots+n-1$，所以，直接插入排序最坏情况下的时间复杂度为 $O(n^2)$。

综上分析可知，直接插入排序的算法时间复杂度为 $O(n^2)$。直接插入排序不需要借助额外存储空间，其空间复杂度为 $O(1)$。

本单元涉及三种排序算法，它们的时间复杂度和空间复杂度情况汇总如表 3-2 所示。

表 3-2　三种排序算法的复杂度

排序方法	时间复杂度（平均）	时间复杂度（最好）	时间复杂度（最坏）	空间复杂度
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
直接插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$

“本单元我们主要学习了冒泡排序、选择排序、直接插入排序的算法思想和实现方法。希望大家能够根据应用场景和数据存储结构选择合适的算法完成排序，掌握各排序算法的时间复杂度和空间复杂度。本单元后面的习题，有助于检验大家的学习效果，抓紧做一下吧。”

习　题

1. 从未排序序列中挑选元素，并将其依次放入已排序序列的一端，这种排序方法称为（　　）。

A. 插入排序　　B. 冒泡排序　　C. 选择排序　　D. 堆排序

2. 对 n 个元素进行直接插入排序，完成排序所需要的轮数是（　　）。

A. n　　B. $n+1$　　C. $n-1$　　D. $2n$

3. 对 n 个元素进行冒泡排序，最好情况下的时间复杂度为（　　）。

A. $O(1)$　　B. $O(\log_2 n)$　　C. $O(n^2)$　　D. $O(n)$

4. 对 n 个元素进行直接插入排序，算法的空间复杂度为（　　）。

A. $O(1)$　　B. $O(\log_2 n)$　　C. $O(n^2)$　　D. $O(n\log_2 n)$

5. 设一组初始记录关键字序列 (5,2,6,3,8)，利用冒泡排序进行升序排序，且排序中从后向前进行比较，则第一趟冒泡排序的结果为（　　）。

A. 2，5，3，6，8　　B. 2，5，6，3，8

C. 2，3，5，6，8　　D. 2，3，6，5，8

6. 使用冒泡排序对包含 50 个元素的序列进行排序，在最坏情况下，比较次数是（　　）。

A. 150　　B. 1225　　C. 2450　　D. 2500

7. 有台计算机使用选择排序对 400 个数字排序花了 400ms，如果花费 1600ms，大概能够完成排序的数字个数是（　　）。

A. 1200　　B. 800　　C. 1600　　D. 3200

8. 有关选择排序的叙述中正确的是（　　）。

A. 每扫描一遍序列，只需要交换一次

B. 每扫描一遍序列，需要交换多次

C. 时间复杂度是 $O(n)$

D. 空间复杂度为 $O(n)$

9. 对序列（54，38，96，23，15，72，60，45，83）进行直接插入排序（升序），插入时由后向前比较，当把第八个元素 45 插入到有序表时，为找到插入位置需比较的次数为（　　）。

A. 4　　B. 6　　C. 5　　D. 3

10. 编写程序实现如下的功能。

（1）针对给定列表 alist = [55，26，83，17，88，34，47，59，22]，使用冒泡排序算法编写函数 bubbleSort 实现 alist 中元素的降序排列。

（2）调用 bubbleSort 函数，输出排序后的列表。

11. 编写程序实现如下的功能。

（1）针对给定列表 alist = [55，26，83，17，88，34，47，59，22]，使用选择排序算法编写函数 selectionSort 实现 alist 中元素的升序排列。

（2）调用 selectionSort 函数，输出排序后的列表。