Python算法之旅（第20期）

题目：分别对奇数和偶数进行升序排序

难度：3星 有趣：3星 有用：3星

分类：排序

描述：有一组正整数，要求分别对奇数和偶数进行升序排序，其中奇数在前，偶数在后。

要求运用各种简单排序算法思想分别完成本题

函数名： sort\_odd\_even(a)

参数表：a -- 待排序列表。

返回值：该方法没有返回值，但是会对列表的对象进行排序。

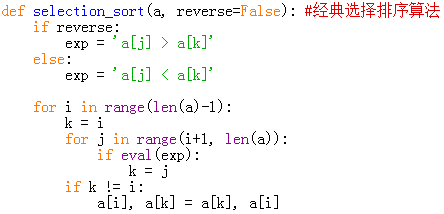
算法分析：

我们先来分析简单排序算法的原型。

选择排序：外层循环变量i指向当前待排序元素，也可以看做待排序区域的左边界，每趟排序的结果就是把待排序区域的最值与a[i]交换位置；内层循环变量j扫描待排序区域，寻找最值的下标。

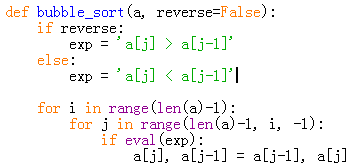
选择排序的特征是算法相当简洁，但是比较死板，哪怕排序对象是一个有序序列，也必须完成(n-1)轮选择，不能提前结束排序，没有多少优化空间。

最多是使用双向选择排序，以较高效率同时找出最大值最小值，从两端向中间减少待排序区域。



冒泡排序：外层循环变量i既可以看做冒泡的趟数，也可以看做待排序区域的左边界，每趟冒泡的结果就是把最值冒泡交换到a[i]处；内层循环变量j从待排序区域的右端依次扫描到左端，比较和交换（若有必要）相邻元素。

冒泡排序的特征是每趟冒泡，内层循环都要从右向左扫描整个待排序区域，其左边界不断右移，扫描范围不断减少；当一趟冒泡下来，没有发生任何交换，则说明排序已完成，即可对基本冒泡排序算法进行优化。



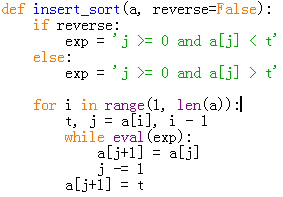
插入排序：外层循环变量i指向当前待排序元素，也可以看做待排序区域的左边界，

每趟排序的结果就是把a[i]插入到合适位置；内层循环变量j从已排序区域的右端向左扫描，比较和移动元素，直到找到插入位置。

插入排序的特征是虽然已排序区域逐渐增大，但内层循环变量j不一定要扫描整个区域，

我们可以使用对分查找快速定位插入位置。

此外，当整个序列“基本有序”时，a[i]距离它的插入位置通常不太远，往往能够较快地完成插入操作，这也是“希尔排序算法”之所以效率较高的原因。



选择排序：选择排序的特点是先确定最大值（最小值）将要放置的位置，再扫描待排序区域，

找到最大值（最小值）的下标，再将二者所指的元素交换位置。

现在要同时对奇数和偶数进行升序排列，只能把奇数的最小值定位在左端，

同时把偶数的最大值定位在右端，然后左右边界不断向中间移动，才能实现选择排序功能。

冒泡排序：算法分析：内层循环从右向左扫描待排序区域，

若a[j]是奇数，则将其左侧偶数和大于a[j]的奇数都交换到右侧；

否则只将其左侧大于a[j]的奇数交换到右侧。

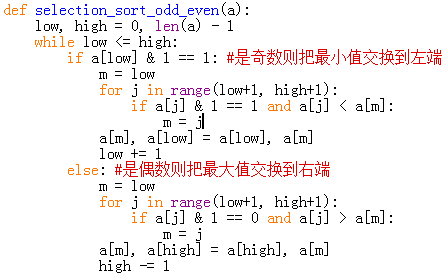
插入排序：将当前待排序元素存储在变量t，若t是奇数，则将偶数和大于t的奇数都右移；

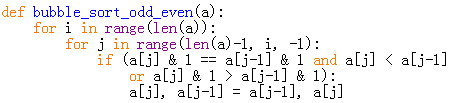
否则只将大于t的偶数右移。

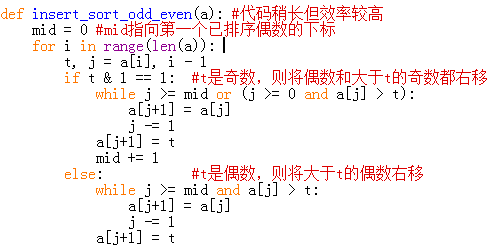
比较直观的算法上设置变量mid指向第一个已排序偶数的下标，这样可以判断奇偶数的边界；

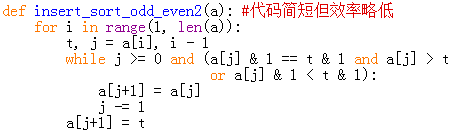
另一种方法是直接比较元素对2求余的结果（a[j] & 1相当于a[j] % 2），

根据余数大小可以判断二者的奇偶性。这样代码较短，但是由于求余运算过多，效率略低。





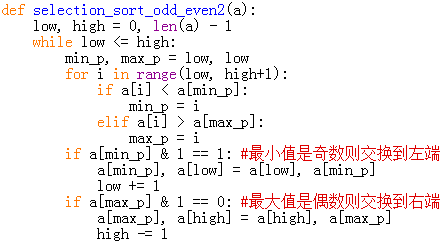




课后思考：

本题在使用选择排序算法思想时，是把奇数和偶数分开来扫描的，扫描过程中也只比较与a[low]奇偶性相同的元素。能不能在扫描时不考虑元素的奇偶性，先找出最大值和最小值，最后再根据最值的奇偶性来确定交换的位置呢？这样看起来代码要简短些，效率也似乎更高。

根据上述思考，我写了以下代码：



令人遗憾的是，这段代码有bug。那么，错误在什么地方？能否改正呢？

聪明的你不妨想一想，我们下期公布答案。

另外，如果你有更 Pythonic（优雅的、地道的、整洁的）代码，或者与本文不同的算法思路和代码实现，请你一定留言或联系我，让我们一起讨论，共同进步。