



数据结构与算法 (Python版)

插入排序算法及分析

陈斌 北京大学 gischen@pku.edu.cn

插入排序Insertion Sort

- ❖ 插入排序时间复杂度仍然是 $O(n^2)$ ，但算法思路与冒泡排序、选择排序不同
- ❖ 插入排序维持一个已排好序的子列表，其位置始终在列表的前部，然后逐步扩大这个子列表直到全表



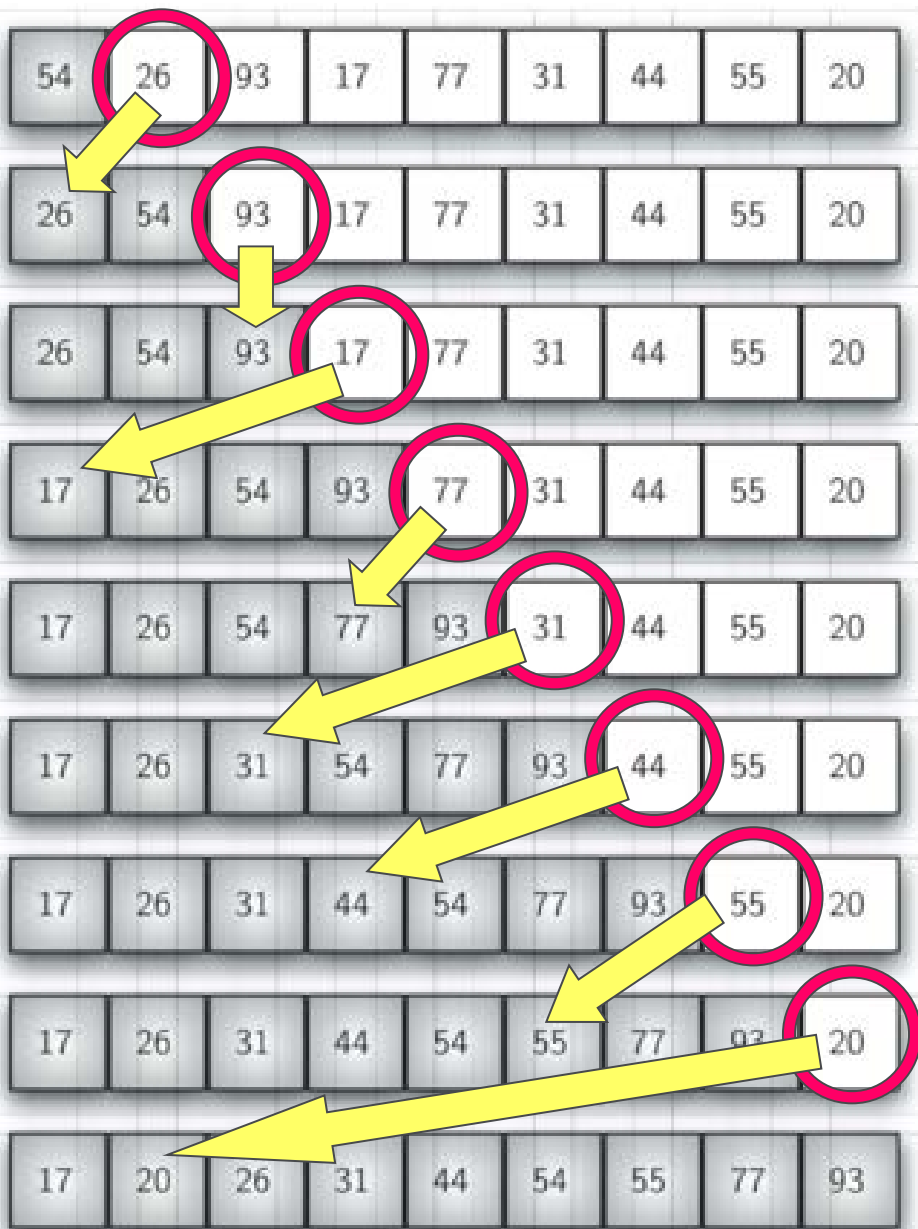
插入排序Insertion Sort

- ❖ 第1趟，子列表仅包含第1个数据项，将第2个数据项作为“新项”插入到子列表的合适位置中，这样已排序的子列表就包含了2个数据项
- ❖ 第2趟，再继续将第3个数据项跟前2个数据项比对，并移动比自身大的数据项，空出位置来，以便加入到子列表中
- ❖ 经过 $n-1$ 趟比对和插入，子列表扩展到全表，排序完成

插入排序Insertion Sort

- ❖ 插入排序的比对主要用来寻找“新项”的插入位置
- ❖ 最差情况是每趟都与子列表中所有项进行比对，总比对次数与冒泡排序相同，数量级仍是 $O(n^2)$
- ❖ 最好情况，列表已经排好序的时候，每趟仅需1次比对，总次数是 $O(n)$

Assume 54 is a sorted
list of 1 item



inserted 26

inserted 93

inserted 17

inserted 77

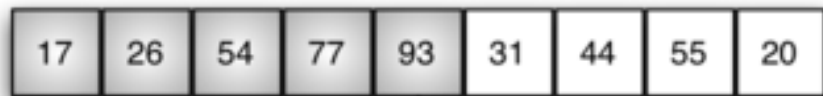
inserted 31

inserted 44

inserted 55

inserted 20

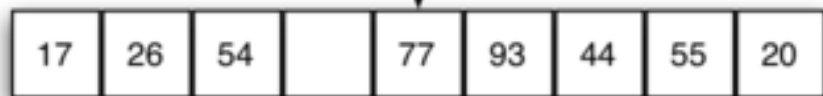
插入排序：思路



Need to insert 31
back into the sorted list



93>31 so shift it
to the right



77>31 so shift it
to the right



54>31 so shift it
to the right



26<31 so insert 31
in this position

比对, 移动所有
比“新项”大的数据项

插入排序：代码

```
def insertionSort(alist):  
    for index in range(1, len(alist)):  
  
        currentvalue = alist[index] ← 新项/插入项  
        position = index  
  
        while position > 0 and alist[position-1] > currentvalue:  
            alist[position] = alist[position-1] ← 比对、移动  
            position = position - 1  
  
        alist[position] = currentvalue ← 插入新项
```

由于移动操作仅包含1次赋值，是交换操作的1/3，所以插入排序性能会较好一些。

