冒泡排序-BubbleSort

- 比较相邻的元素。
  - 如果第一个比第二个大（升序），就交换他们两个。
  - 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。
  - 这步做完后，最后的元素会是最大的数。
- 针对所有的元素重复以上的步骤，除了最后一个。
- 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。
- 复杂度计算
  - 最优时间复杂度：O(n)（表示遍历一次发现没有任何可以交换的元素，排序结束。）
  - 最坏时间复杂度：O(n2)
  - 稳定性：稳定

- Python代码示例

```
def bubble_sort(alist):
    for j in range(len(alist)-1,0,-1):
        # j表示每次遍历需要比较的次数，是逐渐减小的
        for i in range(j):
            if alist[i] > alist[i+1]:
                alist[i], alist[i+1] = alist[i+1], alist[i]

li = [54,26,93,17,77,31,44,55,20]
bubble_sort(li)

print(li)
```

选择排序-SelectionSort

- 算法：
  - 首先在未排序序列中找到最小（大）元素，
  - 存放到排序序列的起始位置，
  - 然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，
    直到所有元素均排序完毕。
- 选择排序的主要优点与数据移动有关。
  - 如果某个元素位于正确的最终位置上，则它不会被移动。
  - 选择排序每次交换一对元素，它们当中至少有一个将被移到其最终位置上，
  - 因此对n个元素的表进行排序总共进行至多n-1次交换。

    - 所有的完全依靠交换去移动元素的排序方法中，选择排序属于非常好的一种。
- 性能分析
    - 最优时间复杂度：O(n2)
    - 最坏时间复杂度：O(n2)
    - 稳定性：不稳定（考虑升序每次选择最大的情况）

- 代码分析：

```
def selection_sort(alist):
    n = len(alist)
    # 需要进行n-1次选择操作
    for i in range(n-1):
        # 记录最小位置
        min_index = i
        # 从i+1位置到末尾选择出最小数据
        for j in range(i+1, n):
            if alist[j] < alist[min_index]:
                min_index = j
        # 如果选择出的数据不在正确位置，进行交换
        if min_index != i:
            alist[i], alist[min_index] = alist[min_index], alist[i]

alist = [54,226,93,17,77,31,44,55,20]
selection_sort(alist)
print(alist)
```

插入排序

- 插入排序（英语：Insertion Sort）是一种简单直观的排序算法。
- 算法：
    - 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。插入排序在实现上，在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，
      为最新元素提供插入空间。
- 性能分析
    - 最优时间复杂度：O(n)（升序排列，序列已经处于升序状态）
    - 最坏时间复杂度：O(n2)
    - 稳定性：稳定

- Python代码

```
def insert_sort(alist):
    # 从第二个位置，即下标为1的元素开始向前插入
    for i in range(1, len(alist)):
```

```
        # 从第i个元素开始向前比较，如果小于前一个元素，交换位置
        for j in range(i, 0, -1):
            if alist[j] < alist[j-1]:
                alist[j], alist[j-1] = alist[j-1], alist[j]

alist = [54,26,93,17,77,31,44,55,20]
insert_sort(alist)
print(alist)
```