选择排序

在排序区域内选最小（大）的数放在第一位

选择排序是对冒泡的改进，这种方法是对参加排序数组的所有元素中**找出最小(或最大)的元素，使它与第一个元素中数据相互交换位置。**然后在余下的 元素中找出最小(或最大)的数据的元素，与第二个元素中的数据交换位置。以此类推，直到所有元素成为一个有序的序列。

用k来记录最小元素的位置

对于n个元素的数组，用选择算法进行排序时，比较次 数与冒泡算法相同，但交换的次数比冒泡排序要少，因 此它具有较高的效率。

展示

找最大的，看看后面还有么，互换。

过度;从哪开始找，到哪结束，如何记录最大的元素的下标，方便对换

实现

3 4 6 7 3

第i趟排序开始时，设第i个位置上的数是当前最小数，用k来标记。 （第i趟排序时，已经有多少个元素在正确的位置）

让k位置上的数（d[k]）与i后面的数(d[j])逐个比较，当找到一个比k 位置上小的数（即d[k]>d[j]），用k记录j的值（k=j）。

当j到达最后一个数时，一趟比较结束，则k指向最小的数，即k记录 的是最小数的位置。

当i不等于k时，交换d[i]与d[k]的值.

3 4 1 2 3升序

第一趟

i = 1, j = 2, k = 1

a[1] = 3 和 a[2] = 4 比 , 3小 k = 1

a[1] = 3 和 a[3] = 1 比 , 1小 k = 3

a[k] a[j]

a[3] = 1 和 a[4] = 2 比, 1小 k = 3

a[3] = 1 和 a[5] = 3 比， 1小 k = 3

交换a[1] 和 a[k] 即a[3]

结果: 1, 4, 3, 2, 3　最小的数已经归位

第二趟

i = 2, j = 3, k = 2

。。。

code

1.先来一个大循环，控制几趟，每一趟都把一个元素放在正确的位置， 假设n个元素，那么需要几趟？ （n-1）

最后剩下一个元素就不用管了，他一定在正确的位置。 range(i, j) 应该是多少?

2.每一趟都在干啥？在比较，在找最小的元素。

每一趟比较的次数都少了，遍历的元素都少了， 为啥?

如何实现每一次遍历元素的个数都比上一次少一个?

找规律， 第一趟从第二个元素开始遍历（比较），到最后一个， 第二次从第三个元素开始遍历（比较）。。。

记录最小元素的位置， k应该放在哪儿?

找到最小元素的位置之后应该干什么？ 这段代码应该放在哪？每一趟换一次位置，所以应该放在控制趟数的for 里面

```
for i in range(0 ~ n-1)  #n-1趟
    k = i #k记录最小值的下标
    for j in range(i+1 ~ n) 第i趟排序时
        if 找到一个比k位置上小的元素，则：
            用k记录j的位置
    if i!=k，则：
        交换数据对
```

```
a = [3, 4, 1, 2, 0, 9, 10]
count = len(a)
for i in range(0, count-1): #这个for循环执行一次就把一个元素放到了正确的位置
    k = i
    for j in range(i+1, count):
        if a[k] > a[j]:
            k = j
    if k!= j:
        a[k], a[i] = a[i], a[k]
print(a)
```

题目：

18， 17， 23， 15， 16，选择排序，降序，互换的次数?

下列错误的是?

A相对而言，选择排序算法的效率比冒泡排序算法高

B冒泡排序算法和选择排序算法的都需要用到双循环结构

C对于n个无序数据，不管是冒泡排序还是选择排序，都要经过n - 1遍加工

D冒泡排序算法的程序实现一般要用到数组变量k，而选 择排序则不需要

插入排序

像是在打牌，

先将列表中的头两个元素按顺序排列（比如：升序）。

接着，每次将一个待排序的元素，按其大小插入前面 已经排好序的元素序列中，使序列依然有序，直到所有 待排序元素全部插入完成。

就像打牌，手里的牌都是排好序的，每次从牌堆里面拿一张牌，然后放到正确位置。

演示：

5 3 5 2 8

实现

a[0]	a[1]	a[2]	a[3]	a[4]
5	**3**	5	2	8

第一次插入，只要与第一个比较

1. 先将要插入的数a[1] 放入一个空的变量key;
2. 将key与前面已经排好序的比较，比如key < a[0] 成立，说明key要插入到 a[0]前面，将a[0]后移一个位置，放到a[1]中；key放入a[0]中

a[0]	a[1]	a[2]	a[3]	a[4]
3	5	**5**	2	8

第二次插入，第3个数插入到前面两个已排好的序列里

1. 先将要插入的数a[2] 放入一个空的变量key
2. 将key与前面已经排好序的比较，比如key<a[1]不成立，说明key要插入到a[1]后面，即a[2])中，将key放入a[2]中。

a[0]	a[1]	a[2]	a[3]	a[4]
3	5	5	**2**	8

第三次插入，第4个数插入到前面三个已排好的序列里

1. 先将要插入的数a[3] 放入一个空的变量key
2. 将key与前面已经排好序的比较

   比如key < a[2]成立， 说明key要插入到a[2]前面, **将a[2]后移一 个位置，放到a[3]中**;

   再比较前一个数key < a[1]成立，说明key要插入到a[1]前面，**将 a[1]后移一个位置，放到a[2]中**;

   再比较前一个数key < a[0]成立，说明key要插入到a[0]前面，**将 a[0]后移一个位置，放到a[1]中**;

   key放入a[0]中。

a[0]	a[1]	a[2]	a[3]	a[4]
2	3	5	5	8

。。。

code

第一个for循环控制几趟， 几趟？ （n-1），

每趟干了什么?

检查某一张牌应该插入前面哪一个位置，具体来说，就是依次比较，找到之后移动、插入。

第i趟结束，则前i+1个元素是已经排好序的。

每一趟有什么差别？ 随着趟数的增加，已经排好序的列表的长度也在增加， j 的初始值（从哪开始往前找合适的位置）也在增大。

与之前不同，在第二个for循环，比较的过程中，j的值是在依次减小的。

考虑可以使用 while 配合 j--

```
a = [5, 3, 5, 2, 8]
count = len(a)
for i in range(1, count):
    key = a[i]
    j = i - 1
    while j >= 0 and a[j] > key:
        a[j + 1] = a[j]
        j -= 1
        a[j + 1] = key
print(a)
```

快速排序

内容太多了，要一节课