排序算法总结

1.冒泡排序

思想：每次比较两个相邻的元素，如果他们的顺序错误，就把他们交换过来。n个元素排n-1趟，每一趟会有一个元素回到正确的位置，每一趟待排序列都比前一趟少一个，直到待排序列中只剩下一个元素，整个序列其他元素的位置都正确，待排序列中的这个元素的位置也一定是正确的。

code

```
a = [12, 35, 99, 18, 76]
count = len(a)
for i in range(0, count - 1):
    for j in range(0, count - 1 -i):
        if a[j] < a[j+1]:
            a[j], a[j+1] = a[j+1], a[j]
print(a)
```

2.选择排序

思想：冒泡排序的改进，减少了交换的次数，一趟只换一次。在每个待排序列中，每次找出最小的（最大的）与第一个元素进行交换。n个元素要排n-1趟，每趟有一个元素回归到正确的位置。每一趟待排序列都比上一趟少一个元素，直到待排序列中只有一个元素。

code

```
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 #记录最大（最小）元素的位置
    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)
```

3.插入排序

思想：像是打牌一样，先摸一张牌，后面每从牌堆里摸一张牌就把它放在正确的位置。手里的牌就是排好序的，牌堆里的牌就是未排好序的，n个元素要排n-1趟，每一趟都有一个元素回归到正确的位置，每一趟未排序的序列比前一趟少一个，直到未排序的序列中只有一个元素。

code

```
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)
```

4.快速排序

思想：在待排序的序列中以第一个数为基准数，将小于基准数的元素都放在基准数的左边，将大于基准数的数都放在基准数的右边。将序列分正两个小序列，再分别对小序列进行同样的操作，直到每个小序列的元素为1。

code

```
def quicksort(l, left, right):
    if left > right: #序列中至少有一个元素
        return
    i = left #哨兵
    j = right
    while i!=j:
        while l[j] >= l[left] and i < j: #哨兵不能相遇
            j-=1
        while l[i] <= l[left] and i < j:
            i+=1
        if i < j: #哨兵还没碰面
            l[i], l[j] = l[j], l[i]
    l[left], l[i] = l[i], l[left]


    quicksort(l, left, i -1)
    quicksort(l, i+1, right)

list1 = [6, 1, 2, 7, 9, 3, 4, 5, 10, 8]
quicksort(list1, 0, len(list1)-1)
print(list1)
```

要求：

问你那个排序算法第几趟干了啥，要能回答的上来。

查找

顺序查找

思想：这是最简单的查找方式，从第一个数据开始，按顺序逐个将数据与给定的数据(查找键)进行比较，若某个数据和查找键相等，则查找成功，输出所查数据的位置；反之，输出未找到。

code

```
lst = [32, 17, 56, 25, 26, 89, 65, 12]
key = 26 #要查找的元素
b = -1 #要查找元素的索引
m= len(lst) #列表长度
for i in range(0, m):
    if lst[i] == key:
        b = i
        break
if b == -1: #-1代表元素未查找到
    print("要查找的元素[" + str(key) + "]不在列表lst中。")
else:
    print("要查找的元素[" + str(key) + "]的索引是：" + str(b))
```

小试牛刀

1．为找自己第一次上幼儿园时的照片，小张同学依次翻开自己的多本相 册来逐张查找。这种查找方法为( )

A．无序查找 B．顺序查找 C．对分查找 D．随机查找

在数组23、41、54、26、84、52、65、21中查找数字52，采用从后 往前顺序查找，需要查找的次数是( )

A．2次 B．3次 C．7次 D．1次

对分查找

对分查找又称二分查找，是一种高效的查找方法。对分查找的前提是，被查找的数据序列是有序的(升序或降序)。

思想：对分查找的基本思想是在有序的数列中，首先将要查找 的数据与有序数列内处于中间位置的数据进行比较，如 果两者相等，则查找成功；否则就根据数据的有序性， 再确定该数据的范围应该在数列的前半部分还是后半部 分；在新确定的缩少范围内，继续按上述方法进行查找， 直到找到要查找的数据，即查找成功，如果要查找的数 据不存在，即查找不成功。

有没有像是猜数字游戏。。。。 大了，小了。。。

程序实现:

若key为查找键，数组a存放n个已按升序排序的元素。在使用对分查找时， 把查找范围[i，j]的中间位置上的数据a[m]与查找键key进行比较，结果必然 是如下三种情况之一：

1.若key<a[m],查找键小于中点a[m]处的数据。由a中的数据的递增性，可以 确定：在(m，j)内不可能存在值为key的数据，必须在新的范围(i，m - 1)中继续查找.

2.key = a[m]，找到了需要的数据

3.key>a[m],，由与(1)相同的理由，必须在新的范围(m + 1，j)中继续查找。 这样，除了出现情况(2)，在通过一次比较后，新的查找范围将不超过上次查 找范围的一半.

中间位置数据a[m]的下标m的计算方法：m = (i + j)//2或m = int((i + j)/2)

(1)由于比较次数难以确定，所以用while语句来实现循环；

(2)在while循环体中用If语句来判断查找是否成功；

(3)若查找成功则输出查找结果，并结束循环(break)；

(4)若查找不成功，则判断查找键在数组的左半区间还是右半区间，从而 缩小范围

```
lst = [12, 17, 23, 25, 26, 35, 47, 68, 76, 88, 96]
key = 25
n = len(lst)
i, j = 0, n - 1
b = -1
while i < j:
    m = (i + j) // 2
    if key == lst[m]:
        b = m #找到了我们要找的数，赋值给b
        break #找到key，退出循环
    elif key > lst[m]:
        i = m + 1
    else:
        j= m - 1
if b == -1: #-1代表元素为查找到
    print("要查找的元素[" + str(key) + "]不在列表lst中。")
else:
    print("要查找的元素[" + str(key) + "]的索引是：" + str(b))

递归：
def binsearch_dg(ll, low, hight, key):
    mid = (low + hight)//2
    if low > hight:
        return 0
    if key == ll[mid]:
        return mid
    elif key < ll[mid]:
```

```
        return binsearch_dg(ll, low, mid-1-1, key)
    else:
        return binsearch_dg(ll, mid + 1, hight-1, key)
ll = [12, 17, 23, 25, 26, 35, 47, 68, 76, 88, 96]
n = binsearch_dg(ll, 0, len(ll)-1, 68)
print(n)
```

小试牛刀

1．下列有关查找的说法，正确的是( )

A．顺序查找时，被查找的数据必须有序 B．对分查找时，被查找的数据不一定有序 C．顺序查找总能找到要查找的关键字 D．一般情况下，对分查找的效率较高

2．某列表有7个元素，依次为19、28、30、35、39、42、48。若采用对 分查找法在该列表中查找元素48，需要查找的次数是( ) A．1 B．2 C．3 D．4