列表与元组

车万翔

哈尔滨工业大学

一个例子

- 读取三个数字，并计算平均数

```
num1 = float(raw_input())
num2 = float(raw_input())
num3 = float(raw_input())
avg = (num1 + num2 + num3) / 3
```

- 如果是30个数呢？

```
num1 = float(raw_input())
num2 = float(raw_input())
num3 = float(raw_input())
...

avg = (num1 + num2 + num3 + ...) / 30
```

列表（List）

- 内建（built-in）数据结构（data structure），用来存储一系列元素（items）
- 如：lst = [5.4, 'hello', 2]

lst	5.4	‘hello’	2
index	0	1	2
index	-3	-2	-1

  - lst[0] is 5.4
  - lst[3] ERROR!
  - lst[1:3] is ['hello', 2]

列表与字符串

- **相同点**
  - 索引（[ ] 运算符）
  - 切片（[:]）
  - 拼接（ + ）和重复（ * ）
  - 成员（in 运算符）
  - 长度（len( ) 函数）
  - 循环（for ）
- **不同点**
  - 使用 [ ] 生成，元素之间用逗号分隔
  - 可以包含多种类型的对象；字符串只能是字符
  - 内容是可变的；字符串是不可变的

列表的方法

- **列表内容是可变的**
  - my_list[0] = 'a'
  - my_list[0 : 2] = [1.2, 3, 5.6]
  - my_list.append(), my_list.extend() #追加元素
  - my_list.insert() #任意位置插入元素
  - my_list.pop(), my_list.remove() #删除元素
  - my_list.sort() #排序
  - my_list.reverse() #逆序
- **更多文档**
  - http://docs.python.org/2/tutorial/datastructures.html#more-on-lists

回到第一个例子

- 读取30个数字，并计算平均数

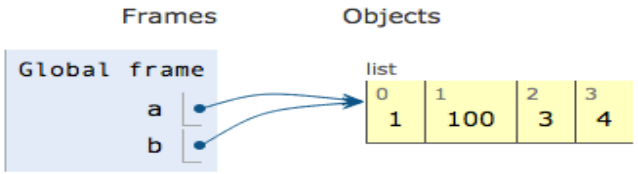
```
nums = []
for i in range(30):
    nums.append(float(raw_input()))
s = 0
for num in nums:
    s += num
avg = s / 30
print avg
```

- 内建 sum 函数
  - avg = sum(nums) / len(nums)
- 更多内建函数，如 max，min
  - http://docs.python.org/2/library/functions.html

列表赋值

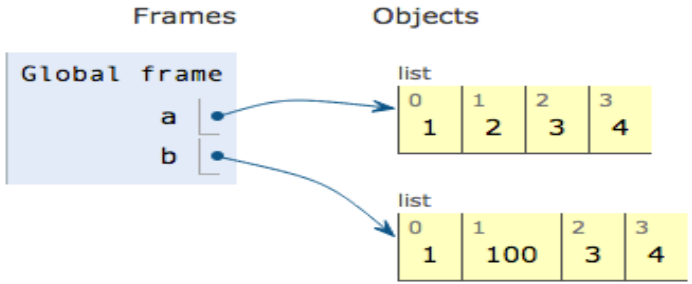
- **下列代码的执行结果是？**

```
a = [1, 2, 3, 4]
b = a
b[1] = 100
print a[1]
```



- **下面的呢？**

```
a = [1, 2, 3, 4]
b = a[:]
b[1] = 100
print a[1]
```



- **动态演示**
  - http://www.pythontutor.com/

列表作函数参数

- 如交换列表中两个元素的函数

```
def swap(a, b):
    tmp = a
    a = b
    b = tmp

x = 10
y = 20

swap(x, y)
print x, y
```

10, 20

VS.

```
def swap(lst, a, b):
    tmp = lst[a]
    lst[a] = lst[b]
    lst[b] = tmp

x = [10, 20, 30]

swap(x, 0, 1)
print x
```

示例：查找

- **在列表中查找一个值，并返回该值第一次出现的位置；如果该值不存在，则返回 -1**

```
def search(lst, x):
    for i in range(len(lst)):
        if lst[i] == x:
            return i
    return -1
```

- **list.index( ) 方法**
  - [1, 2, 2].index(2)
- **线性查找**
  - 最坏运行时间： $k_0n + k_1$

时间复杂度

- **量化一个算法的运行时间为输入长度的函数**
- **不需要显式的计算这些常数**
  - 如： 4n +10 和 100n +137 都与输入规长度正比
- **大*O*表示，只保留高阶项**
  - $4n + 4 = O(n)$
  - $137n + 271 = O(n)$
  - $n^2 + 3n + 4 = O(n^2)$
  - $2^n + n^3 = O(2^n)$
- **线性查找的时间复杂度为：*O*(n)**

时间复杂度

- **大O能告诉我们什么？**
  - 如果算法A的复杂度为O(n)，算法B的复杂度为$O(n^2)$，对于较大的输入，A总是比B快
  - 如果算法A的复杂度为O(n)，当输入规模翻倍时，运行时间也翻倍
- **大O不能告诉我们什么？**
  - 实际运行时间
    - $10^{100}n = O(n)$
    - $10^{-100}n = O(n)$
  - 对于小规模输入的行为
    - $n^3 = O(n^3)$
    - $10^6 = O(1)$

函数的增长率

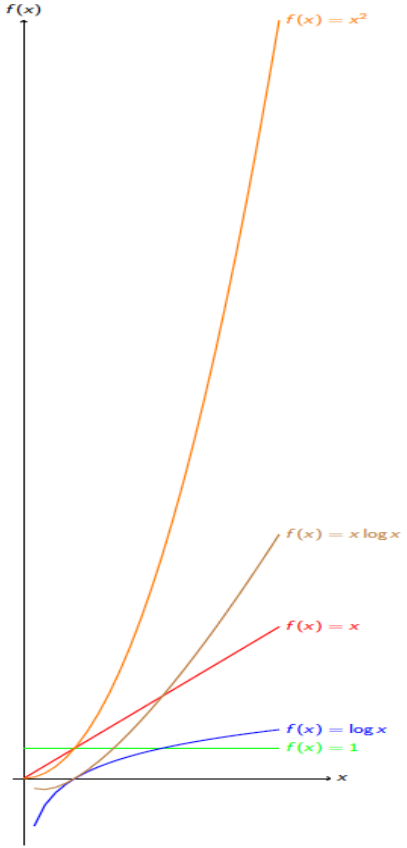


Table : 1 operation = 1 μs

Size	1	log n	n	$n \log n$	n^2	n^3
100	1μs	7μs	100μs	0.7ms	10ms	<1min
200	1μs	8μs	200μs	1.5ms	40ms	<1min
300	1μs	8μs	300μs	2.5ms	90ms	1min
400	1μs	9μs	400μs	3.5ms	160ms	2min
500	1μs	9μs	500μs	4.5ms	250ms	4min
600	1μs	9μs	600μs	5.5ms	360ms	6min
700	1μs	9μs	700μs	6.6ms	490ms	9min
800	1μs	10μs	800μs	7.7ms	640ms	12min
900	1μs	10μs	900μs	8.8ms	810ms	17min
1000	1μs	10μs	1000μs	10ms	1000ms	22min
1100	1μs	10μs	1100μs	11ms	1200ms	29min
1200	1μs	10μs	1200μs	12ms	1400ms	37min
1300	1μs	10μs	1300μs	13ms	1700ms	45min
1400	1μs	10μs	1400μs	15ms	2000ms	56min

二分查找

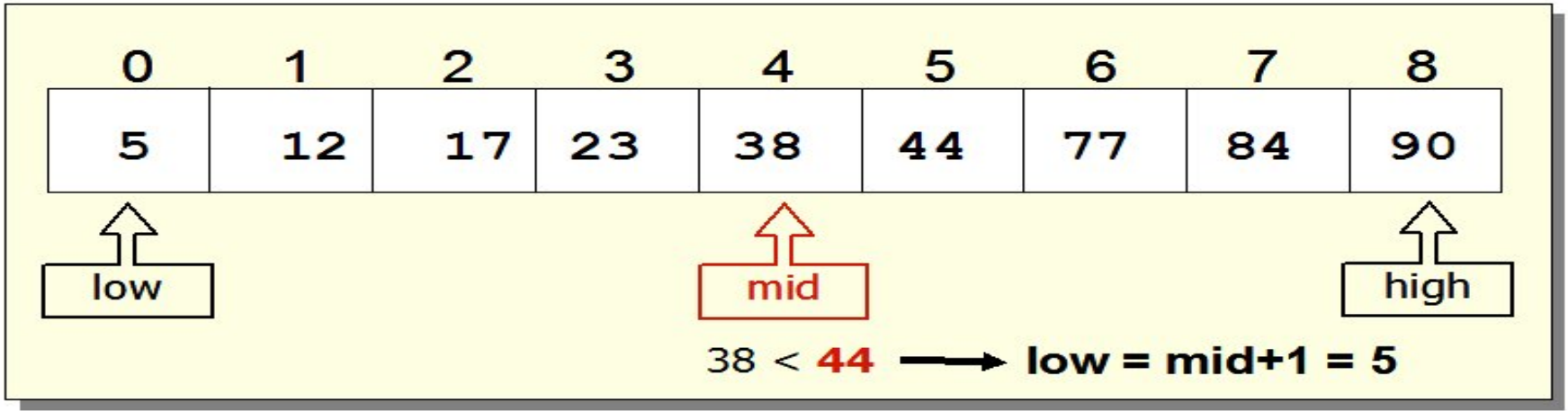
- **编写函数 bi_search，输入一个有序（由小到大）列表和一个值，如果该值在列表中，则返回相应的位置，否则返回 -1**
- **考虑以下三种情况**
  - 如果输入值**小于**列表中间的元素，则只需要在列表的**前**半部分继续查找
  - 如果输入值**大于**列表中间的元素，则只需要在列表的**后**半部分继续查找
  - 如果输入的值**等于**列表中间的元素，则返回该位置

二分查找示例

	low	high	mid
#1	0	8	4

search(44)

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

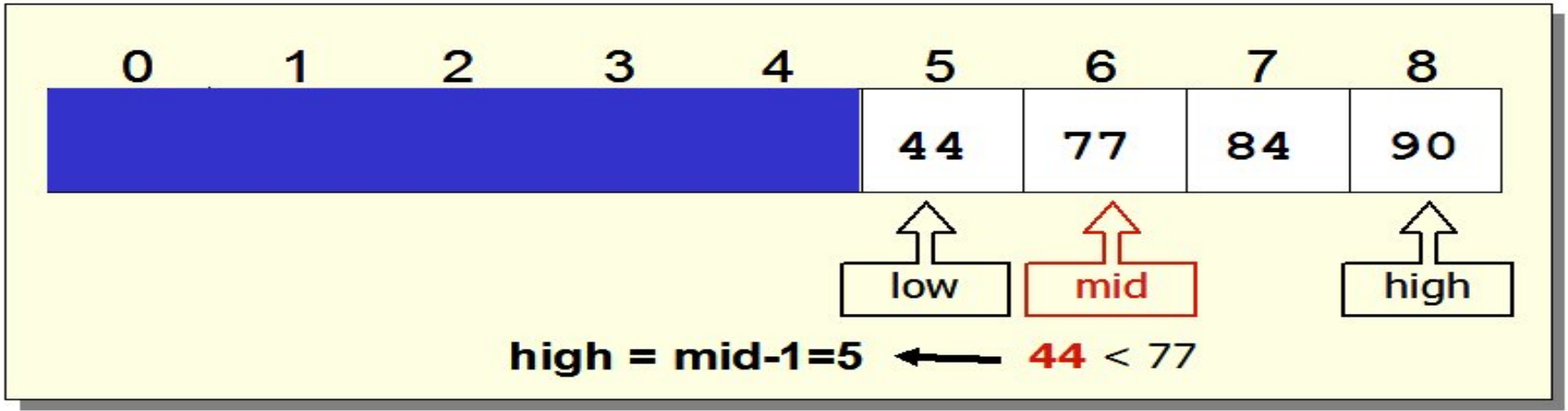


二分查找示例

	low	high	mid
#1	0	8	4
#2	5	8	6

search(44)

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

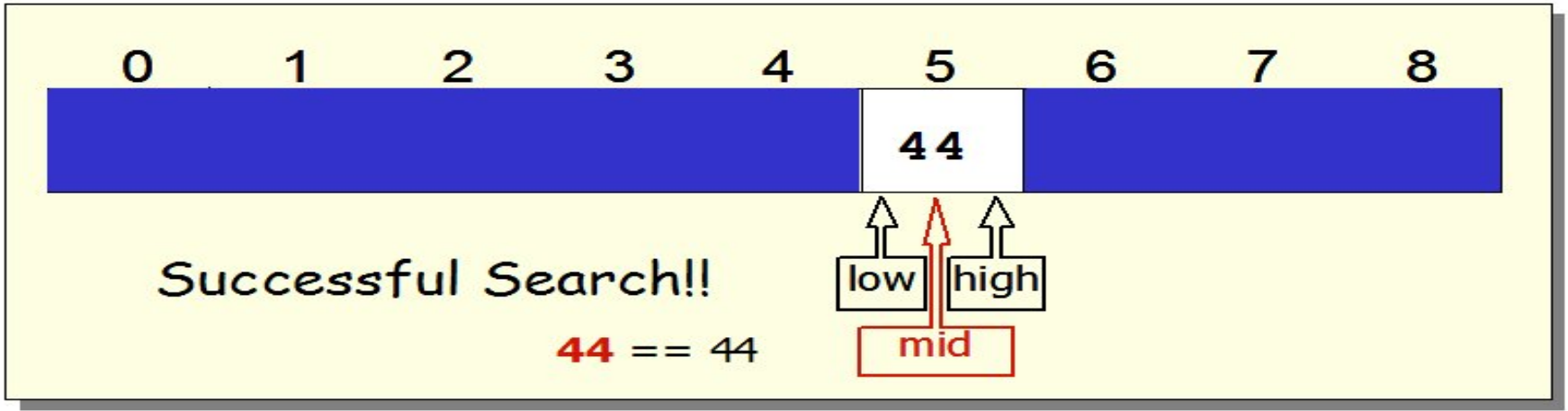


二分查找示例

	low	high	mid
#1	0	8	4
#2	5	8	6
#3	5	5	5

search(44)

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$



二分查找实现

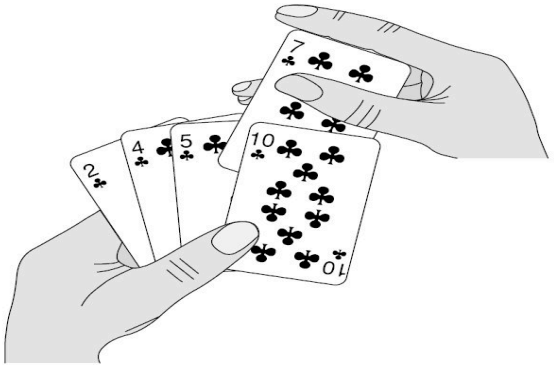

```
def bi_search(lst, v):
    low = 0
    up = len(lst) - 1

    while low <= up:
        mid = (low + up) / 2
        if lst[mid] < v:
            low = mid + 1
        elif lst[mid] == v:
            return mid
        else:
            up = mid - 1

    return -1
```

- 时间复杂度：$O(\log_2 n)$

排序（Sort）

- **将一个无序列表，按照某一顺序（由小到大或由大到小）排列**
- **是计算机科学中常见而且重要的任务**
- **有许多不同的算法，我们只介绍两种最简单直观的**
  - 选择排序（selection sort）
  - 冒泡排序（bubble sort）

选择排序（版本1）



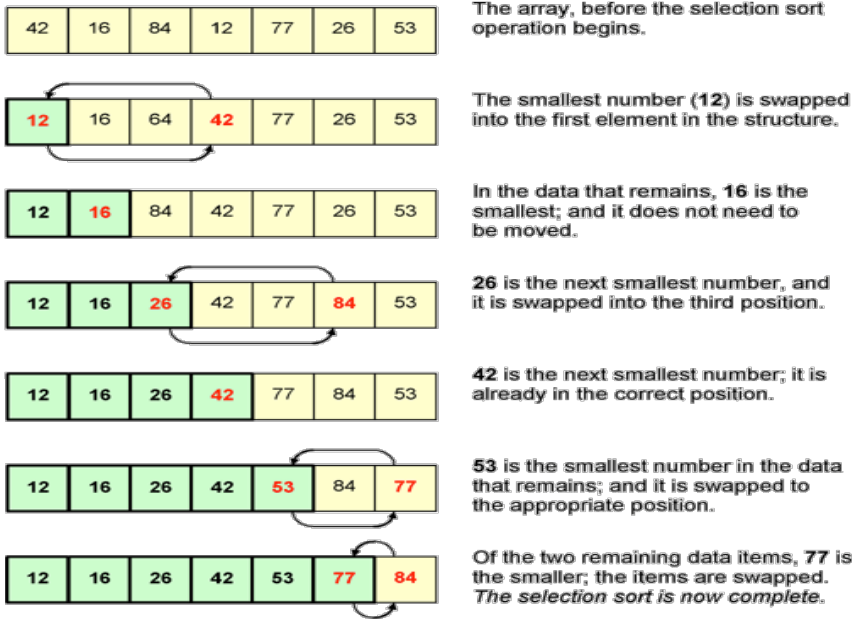
1. 找到最小的元素
2. 删除它，然后将其插入相应的位置
3. 对于剩余的元素，重复步骤 1 和 2

```
def selection_sort(lst):
    for i in range(len(lst) - 1):
        min_idx = i
        for j in range(i + 1, len(lst)):
            if lst[j] < lst[min_idx]:
                min_idx = j
        lst.insert(i, lst.pop(min_idx))

lst = [42, 16, 84, 12, 77, 26, 53]
print lst
selection_sort(lst)
print lst
```

选择排序（版本2）

1. 找到最小的元素
2. 和第一个元素**交换**
3. 对于剩余的元素，重复步骤 1 和 2

```
def selection_sort(lst):
    for i in range(len(lst) - 1):
        min_idx = i
        for j in range(i + 1, len(lst)):
            if lst[j] < lst[min_idx]:
                min_idx = j
        swap(lst, min_idx, i)

lst = [42, 16, 84, 12, 77, 26, 53]
print lst
selection_sort(lst)
print lst
```

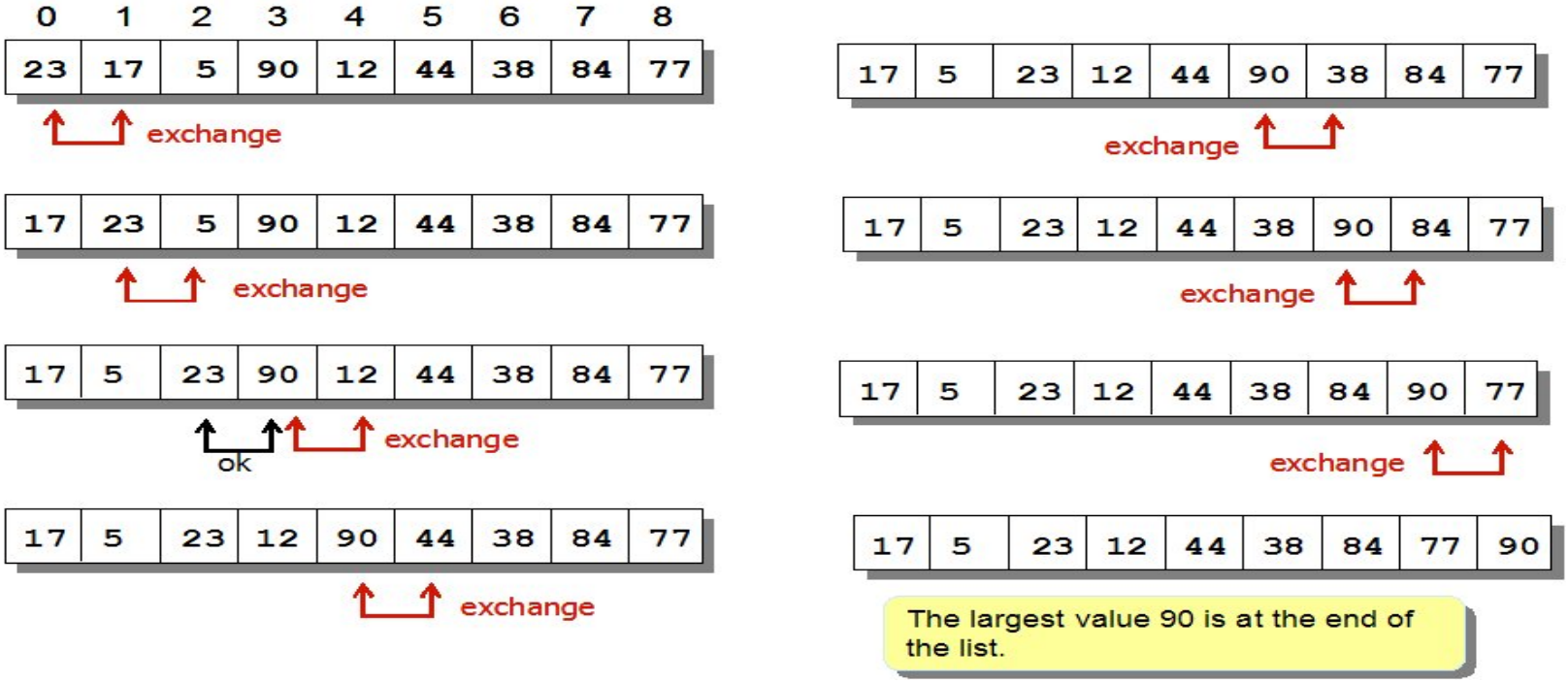


选择排序的时间复杂度

- **所需步骤**
  - 找到最小的元素需要 n 步
  - 找到剩余的最小元素需要 n-1 步
  - ……
- **总运行时间**
  - $n + (n - 1) + \ldots + 2 + 1$
- **时间复杂度**
  - $O(n^2)$

冒泡排序

- 与选择排序类似，但是每次遍历不止交换一次
- 每次遍历，将最大的值排在最后



冒泡排序

- **提示：一旦列表排好序，算法可以停止**

```
def bubble_sort(lst):
    exchanged = True
    top = len(lst) - 1
    while exchanged:
        exchanged = False
        for i in range(top):
            if lst[i] > lst[i+1]:
                swap(lst, i, i+1)
                exchanged = True
        top -= 1
```

- **时间复杂度**
  - $O(n^2)$，与选择排序相同，但是通常速度更快，为什么？

内建排序函数

- **sorted() 函数**

```
>>> a = [5, 2, 3, 1, 4]
>>> sorted(a)
[1, 2, 3, 4, 5]
```

- **list.sort() 方法**

```
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> a
[1, 2, 3, 4, 5]
```

- **算法：quicksort**
  - 时间复杂度：O($n$log$n$)

嵌套列表

- 如何存储如下的数据表？

	0	1	2	3
0	5	4	7	3
1	4	8	9	7
2	5	1	2	3

- 列表的列表
  - x = [[5,4,7,3], [4,8,9,7], [5,1,2,3]]
  - 访问第三行、第二列的元素： x[2][1]
  - 请问：len(x) 的结果是？
  - 如何获取列数？

嵌套列表示例

- 计算所有学生的平均分

```
students = [['Zhang', 84],
            ['Wang',  77],
            ['Li',    100],
            ['Zhao',  53]]
s = 0

for student in students:
    s += student[1]

print float(s) / len(students)
```

列表解析或推导（List Comprehension）

- 一种由原列表创建新列表的简洁方法
  - [表达式 for 变量 in 列表 if 条件]
- 如生成值为 $\{x^2 : x \in \{1 \ldots 9\}\}$ 的列表

```
lst = []

for x in range(1, 10):
    lst.append(x**2)

print lst
```

- 列表解析

```
lst = [x**2 for x in range(1, 10)]
```

列表解析示例

- **列表推导实现求平均分**
  - sum([x[1] for x in students]) / len(students)
- **使用列表解析对所输入数字 x 的因数求和**
  - 如：如果输入 6，应该显示12，即 1 + 2 + 3 + 6 = 12
  - sum([i for i in range(1, x + 1) if x % i == 0])

嵌套列表示例

- 按照成绩由高到低排序

```
students = [['Zhang', 84],
            ['Wang',  77],
            ['Li',    100],
            ['Zhao',  53]]

def f(a):
    return a[1]

students.sort(key = f, reverse = True)

print students
```

lambda 函数

- 定义匿名函数

```
>>> def f (x): return x**2
...
>>> print f(8)
64
>>>
>>> g = lambda x: x**2
>>>
>>> print g(8)
64
```

- lambda 函数实现按成绩排序

```
students.sort(key = lambda x: x[1], reverse=True)
```

元组

什么是元组（Tuple）？

- **元组即不可变（immutable）列表**
  - 除了可改变列表内容的方法外，其它方法均适用于元组
  - 因此，索引、切片、len( )、print等均可用
  - 但是，append、extend、del等不可用
- **使用，（可以加（））创建元组**

```
my_tuple = 1, 'a', 3.14, True
my_tuple = (1, 'a', 3.14, True)
```

- **为什么需要元组？**
  - 保证列表内容不被修改

元组赋值

- 交换两个值

```
temp = a
a = b
b = temp
```

- 或者

```
a, b = b, a
```

- 如切分一个邮件地址

```
name, domain = 'car@hit.edu.cn'.split('@')
```

函数和元组

- **函数只能有一个返回值**
  - 但是该值可以是一组值，如返回一个元组
- **如同时返回列表中的最大和最小值**

```
def max_min(lst):
    max = min = lst[0]
    for i in lst:
        if i > max:
            max = i
        if i < min:
            min = i
    return max, min
```

DSU模式

- **Decorate, Sort and Undecorate (DSU) 模式**
  - 装饰、排序和反装饰
- **如根据单词的长度对一个单词列表进行排序**

```
def sort_by_length(words):
    # decorate
    t = []
    for word in words:
        t.append((len(word), word))

    # sort
    t.sort(reverse = True)

    # undecorate
    res = []
    for length, word in t:
        res.append(word)

    return res
```

```
words.sort(key = lambda x: len(x), reverse = True)
```