数据结构与算法（Python）-07/第8周

北京大学 陈斌

2021.04.27

线下课堂

- 【H3】动态规划作业详解
- 本周内容小结：排序查找（上）
- 问题解答
- 慕课测验题讲解
- 【C1】开源硬件创意作品
- 【K07】课堂练习

【H4-1】作业详解：博物馆大盗

- 视频412详细解释了算法过程
- 最大价值表格m(i,W)

W →

i ↓

m	0	1	2	3	4	5	...
0	0	0	0	0	0	0	
1	0	0	3	3	3	3	
2	0	0	3	4	4	7	
3	0	0	3	4	8	8	
4	0	0	3	4	8	8	
5	0	0	3	4	8	8	

m(5,5)=m(4,5)=max(m(3,5), m(3,0)+8)

$$m(i,W)=\begin{cases}0 & \text{if } i=0\\ 0 & \text{if } W=0\\ m(i-1,W) & \text{if } w_i>W\\ \max\{m(i-1,W),v_i+m(i-1,W-w_i)\} & \text{otherwise}\end{cases}$$

* 装不下第i个
* 装或者不装/价值大者

【H4-1】作业详解：博物馆大盗

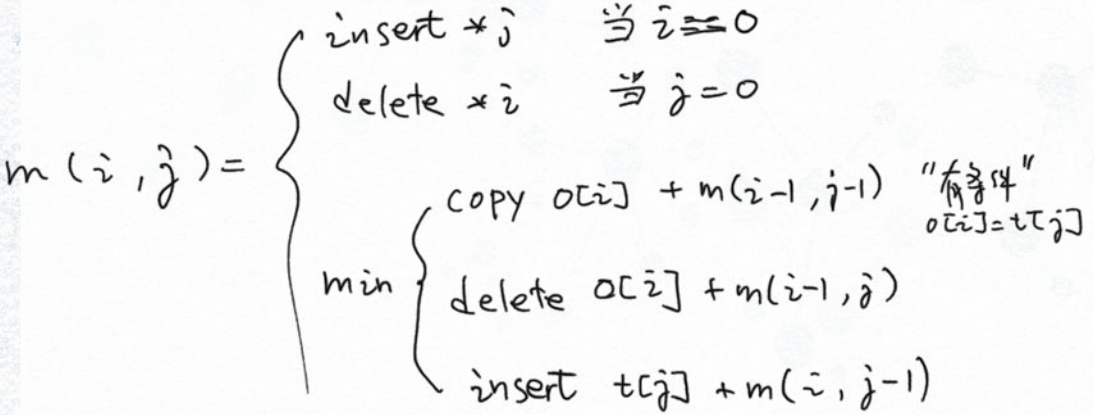
- 主要是输出选择的宝物
- 在m(i,W)中记录：[最大价值, 加了哪个宝物得到的最大价值]
- 小技巧：让宝物编码从1开始

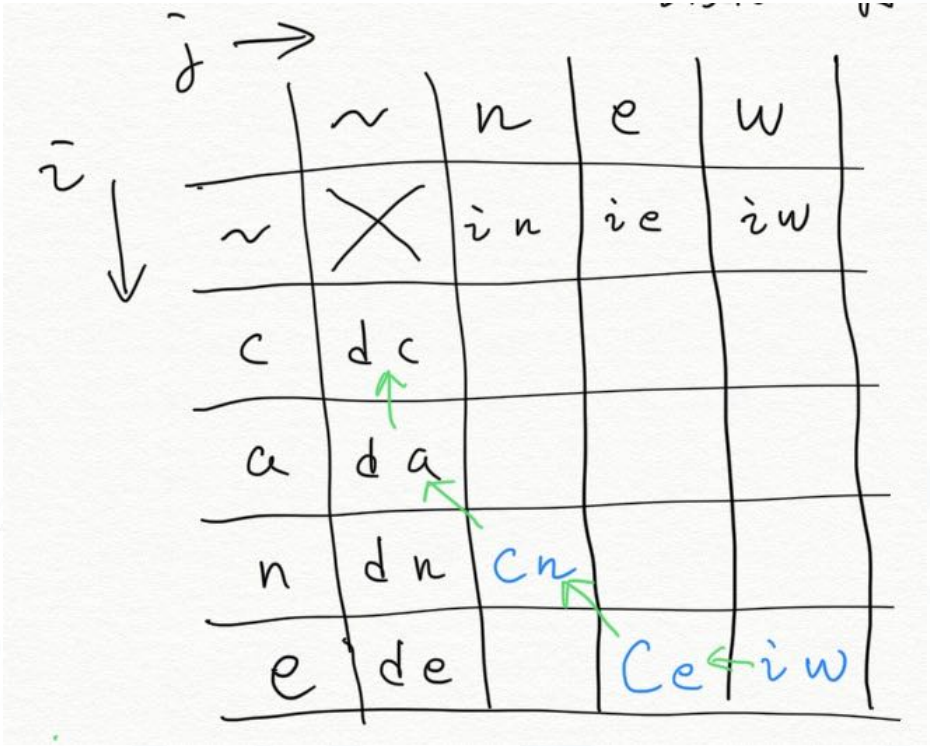
```
 9  def dpMuseumThief(treasureList, maxWeight):
10      maxValue = 0
11      chosenList = []
12
13      # 请在此编写你的代码（可删除pass语句）
14      tr = [None] + treasureList #让宝物编码从1开始
15
16      # 初始化二维表格m[(i, w)]，均为0,None
17      # 表示前i个宝物中，最大重量w的组合，所得到的最大价值
18      # 以及加了哪个宝物得到的这个最大价值
19      # 当i什么都不取，或w上限为0，价值均为0
20      m = {(i, w):[0, None] for i in range(len(tr))
21                            for w in range(maxWeight + 1)}
```

```
22      # 逐个填写二维表格
23      for i in range(1, len(tr)):
24          for w in range(1, maxWeight + 1):
25              if tr[i]['w'] > w:  # 装不下第i个宝物
26                  m[i, w][0]= m[i-1, w][0]
27                  m[i, w][1]= None # 不装宝物
28              else:
29                  # 不装第i个宝物，装第i个宝物，两种情况下最大价值
30                  if m[i-1, w][0]> m[i-1, w-tr[i]['w']][0] + tr[i]['v']:
31                      m[i, w][0]= m[i-1, w][0]
32                      m[i, w][1]= None # 不装宝物
33                  else:
34                      m[i, w][0]= m[i-1, w-tr[i]['w']][0] + tr[i]['v']
35                      m[i, w][1]= tr[i]
36
37      maxValue = m[i, w][0]
38      while w > 0:
39          if m[i, w][1] is not None: # 如果装了宝物，就输出
40              chosenList.insert(0, m[i, w][1])
41              w = w - m[i, w][1]['w']
42          i = i - 1
43
44      # 代码结束
45
46      return maxValue, chosenList
```

【H4-2】作业详解：单词最小编辑距离

- 实际上，这个问题跟博物馆大盗算法基本一样
- 构造最小距离表格m(i, j)





【H4-2】作业详解：单词最小编辑距离

- 在m(i,j)中记录：[最小距离, 使用了哪个操作得到的最小距离]
- 小技巧：让original、target字符编码从1开始

```
10 def dpWordEdit(original, target, oplist):
11     score = 0
12     operations = []
13 
14     # 请在此编写你的代码（可删除pass语句）
15 
16     # 初始化二维表格m[(i, j)]，均为0,None
17     # 表示源单词前i个字符子串，变为目标单词前j个字符子串，的最小编辑距离
18     # 以及最后进行了哪个操作得到的这个最小编辑距离
19     m = {(i, j):[0, None] for i in range(len(original)+1)
20                           for j in range(len(target)+1)}
21 
22     # 让字符的序号从1开始
23     original, target = "-" + original, "-" + target
24 
25     # 设定首行m[0,j]，全insert
26     for j in range(1, len(target)):
27         m[0, j] = [oplist["insert"] * j, "insert " + target[j]]
28     # 设定首列m[i,0]，全delete
29     for i in range(1, len(original)):
30         m[i, 0] = [oplist["delete"] * i, "delete " + original[i]]
```

```
    # 逐个填写二维表
    for i in range(1, len(original)):
        for j in range(1, len(target)):
            ops = [] # 可能的操作和对应的分数
            # copy操作，有条件
            if original[i] == target[j]:
                ops.append([m[i-1, j-1][0] + oplist["copy"], "copy " + original[i]])
            # delete操作
            ops.append([m[i-1,j][0] + oplist["delete"], "delete " + original[i]])
            # insert操作
            ops.append([m[i,j-1][0] + oplist["insert"], "insert " + target[j]])

            # 取分数最小的
            m[i,j] = min(ops, key=lambda x:x[0])

    score = m[i,j][0]
    while i> 0 or j> 0:
        op = m[i,j][1]
        operations.insert(0, op) # 倒序输出操作序列
        if "copy" in op:
            i, j = i - 1, j - 1
        elif "delete" in op:
            i = i - 1
        elif "insert" in op:
            j = j - 1

    # 代码结束

    return score, operations
```

关于输出所有的组合

所有的最大价值组合、所有的最小距离操作组合

```
if tr[i]['w'] > w:  # 装不下第i个宝物
    m[(i, w)][0]= m[(i-1, w)][0]
    m[(i, w)][1]= None # 不装宝物
else:
    # 不装第i个宝物，装第i个宝物，两种情况下最大价值
    if m[(i-1, w)][0]> m[(i-1, w-tr[i]['w'])][0] + tr[i]['v']:
        m[(i, w)][0]= m[(i-1, w)][0]
        m[(i, w)][1]= None # 不装宝物
    else:
        m[(i, w)][0]=
        m[(i, w)][1]=
```

```
ops = [] # 可能的操作和对应的分数
# copy操作，有条件
if original[i] == target[j]:
    ops.append([m[(i-1, j-1)][0] + oplist["copy"], "
# delete操作
ops.append([m[(i-1,j)][0] + oplist["delete"], "delet
# insert操作
ops.append([m[(i,j-1)][0] + oplist["insert"], "inser

# 取分数最小的
m[(i,j)] = min(ops, key=lambda x:x[0])
```

本周内容：排序与查找（上）

- 501 顺序查找算法及分析 9m41s
- 502 二分查找算法及分析 12m20s
- 503 冒泡和选择排序算法及分析 12m14s
- 504 插入排序算法及分析 7m06s
- 505 谢尔排序算法及分析 6m15s
- 506 归并排序算法及分析 9m13s
- 507 快速排序算法及分析 12m30s

各种排序的要点

- 冒泡排序：最稳定n^2；不需要额外空间；
- 选择排序：同冒泡，记录最大值，只做1次交换；

- 插入排序：平均n^2，最好n；不需要额外空间；
  从1项开始，逐步扩大排序子表，寻找“新项”插入位置
  在已排序子表中，从后往前，比对、移动所有比“新项”大的数据项
- 谢尔排序：在n、n^2之间；不需要额外空间；
  在子列表中采用插入排序gapInsertionSort
  从n/2倍增减小gap直到1，子列表个数直到1个，排好序

各种排序的要点

- **归并排序：稳定nlogn；需要1倍额外空间；**
  以“中间位置”作为分裂子列表的依据；

  递归->合并

- **快速排序：最好、平均nlogn；最差n^2；不需要额外空间；**
  以“中间值”作为分裂子列表的依据；

  分裂->递归

  数据的初始分布、取中间值的方法特别重要

- **各种算法的过程可视化**
  https://visualgo.net/zh/sorting

排序过程中的比较信息

一阶信息

ai，aj之间的大小比较，如ai<aj

冒泡、选择、插入、谢尔排序

每一轮都会将未排序的部分捋一遍，未排序部分每轮缩小一个数

时间复杂度在n^2级别

排序过程中的比较信息

二阶的隐含信息

ai，aj以及ai，ak之间大小的信息

如ai>aj，又有aj>ak，应该有ai>ak，ai和ak不需要再次比较

快速排序：中值就是aj，将把ai和ak分裂到不同子表，不会再次比较

归并排序：在合并过程中，比如ai属一个子表，aj/ak属另一个子表，也不会让ai和ak再次比较

未排序部分每轮缩小一半，时间复杂度在nlogn级别

具有先验信息的排序算法

- **基数排序**

  通过把整数表示为某个进制的符号表示，如十进制；

  我们已经预先知道了进制中数字符号之间的大小；

  基数队列之间是有固定的先后次序。

- **这样甚至不需要去直接比较，只需要按照每一位符号排到相应的队列里**

- **所以具有更好的排序性能，时间复杂度是$n \log_B N$级别，B是基数（如10），N是这组数中最大的数，就是最大的数有几位。**

什么是稳定的排序？

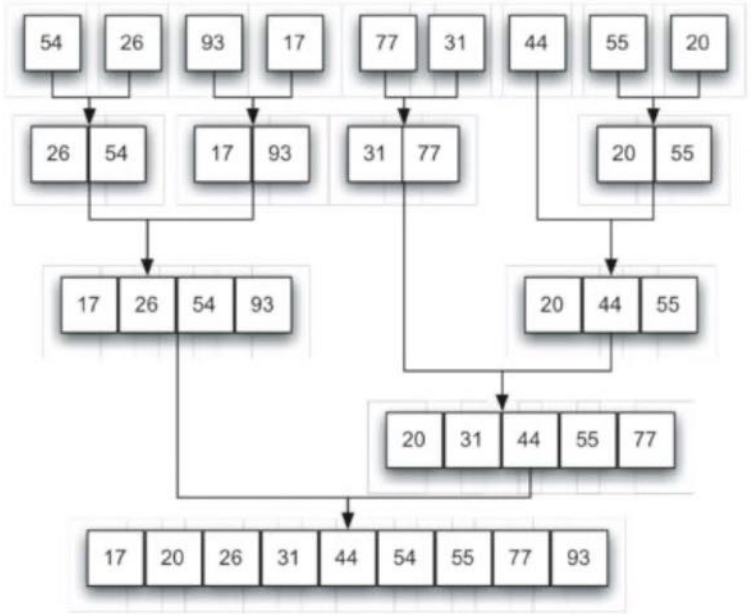
- 保证排序前2个相等数的前后位置，和排序后前后位置顺序相同

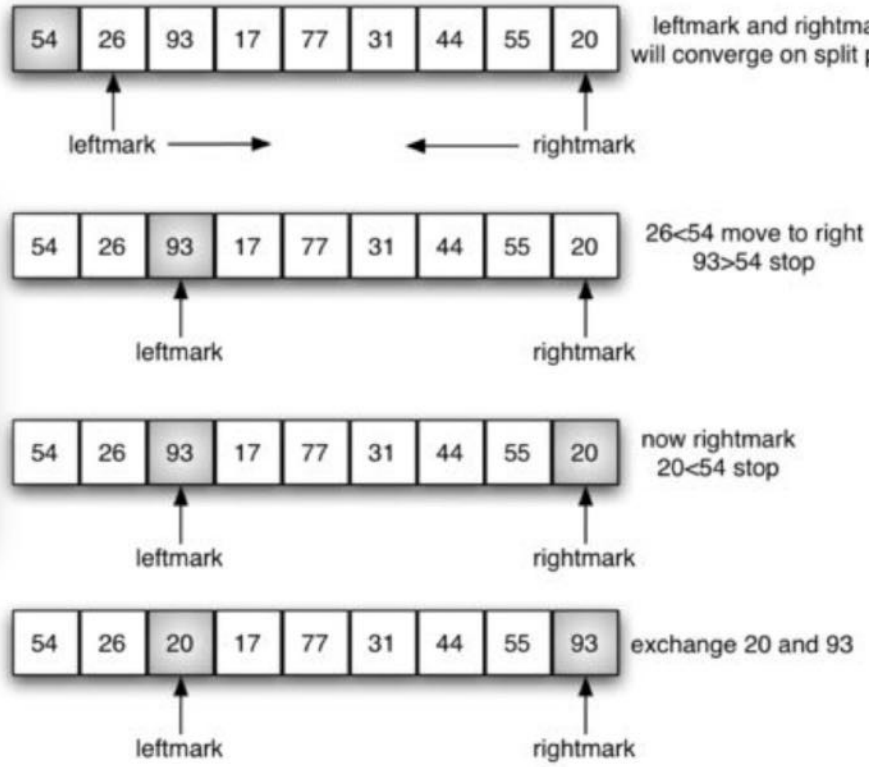
  稳定性的好处：支持多个键值排序；可以减少一些交换次数；

- 冒泡排序是**稳定**的：交换只发生在相邻两个数据项之间；
- 选择排序**不稳定**：可能会跨越多个数据项交换；
- 插入排序是**稳定**的：从后向前寻找插入位置，不改变相对；
- 谢尔排序**不稳定**：虽然以插入排序为基础，但多个子表是穿插的
- 基数排序是**稳定**的：相同数据项始终按顺序进入队列，出队列

什么是稳定的排序？

- 归并排序是**稳定**的：在合并的过程中，左半部始终在前面，也不破坏相同数据项的相对位置
- 快速排序**不稳定**：在右标左移的过程中，会把尾部的数，跳跃交换到前面，破坏相对位置





另一些奇奇怪怪的排序

睡眠排序（Sleep Sort）

把每个数对应一个线程，同时启动这些线程

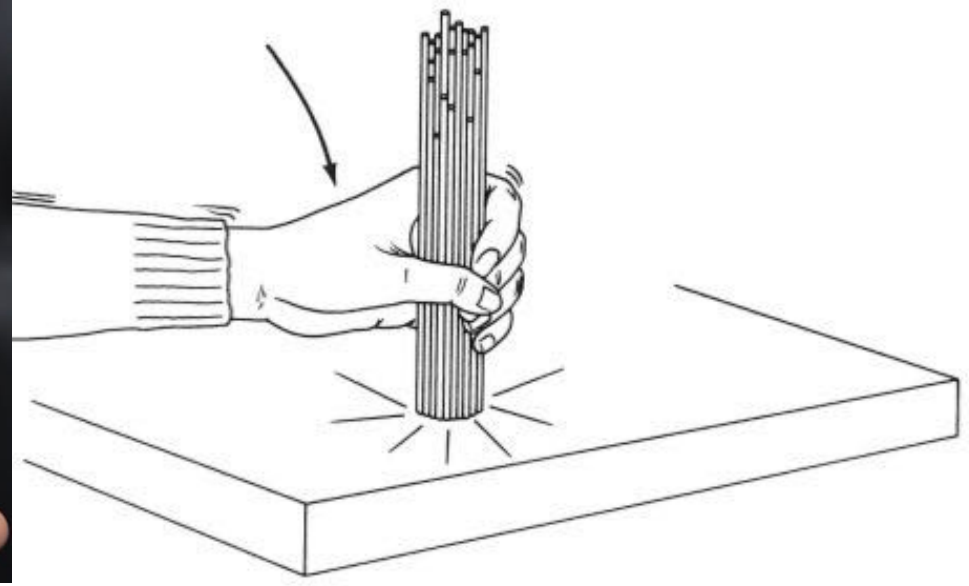
每个线程都是sleep对应数的时间，然后输出这个数

完成排序

```
1  import _thread
2  from time import sleep
3
4  items = [6, 2, 5, 3, 1, 7]
5
6  def sleep_sort(i):
7          sleep(i)
8          print(i)
9
10 [_thread.start_new_thread(sleep_sort, (i,)) for i in items]
```

另一些奇奇怪怪的排序

- 面条排序(Spaghetti Sort)
- 将需要排序的每个数对应到面条的长度，一墩……



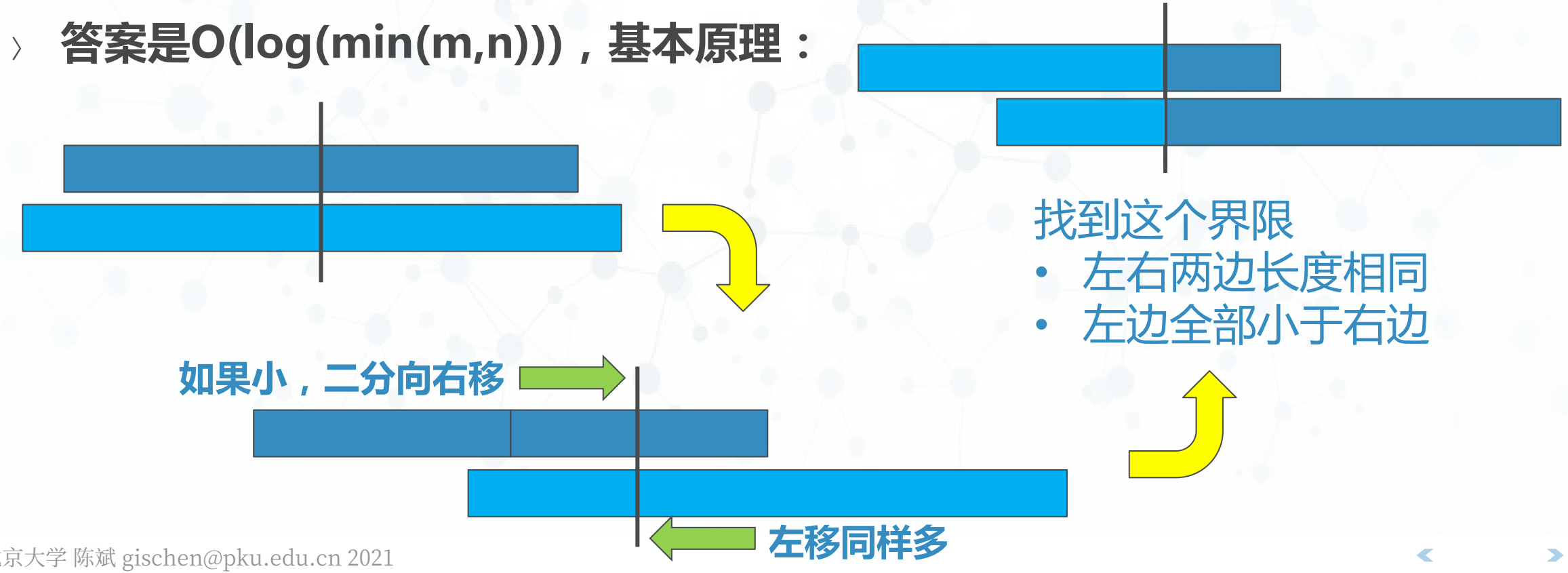
另一些奇奇怪怪的排序

猴子排序（Bogo Sort）

```python
from random import shuffle

def inorder(lst):
    for i in range(len(lst) - 1):
        if lst[i] > lst[i + 1]:
            return False
    return True

def bogo(x):
    while not inorder(x):
        shuffle(x)
    return x

l = bogo([1,3,2,4,5])
print(l)
```

问题解答：W07-寻找两个正序数组的中位数

- https://leetcode-cn.com/problems/median-of-two-sorted-arrays/solution/xun-zhao-liang-ge-you-xu-shu-zu-de-zhong-wei-s-114/
- 答案是O(log(min(m,n)))，基本原理：



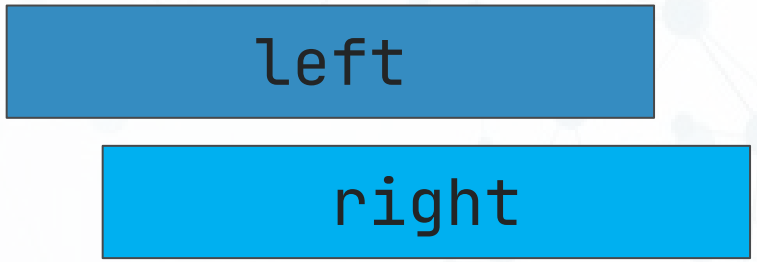
问题解答：W07-排好序的列表，能得到O(n)的算法

- 答案是冒泡、插入、归并排序
- 三者原理相同，都是可以在算法中通过**简单测试**来跳过不必要的比较

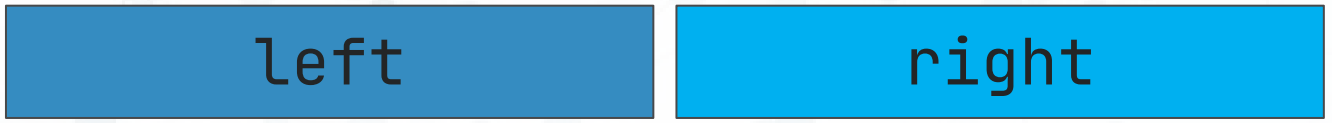
  冒泡：如果一趟中都没有交换次序发生，一趟就结束；

  插入：已排序子表，从右到左，与“新项”比对，如果一直都没有移动发生，一趟就结束；

  归并排序：如果归并的时候发现左边列表最大值小于右边列表最小值，就直接合并，无需比较。



一般情况归并



本题情况归并

问题解答：快速排序的左右标地位不对称？

› 🤔 ……为什么要对称？

- **分裂数据表的目标：找到“中值”的位置**
- **分裂数据表的手段**

设置左右标（left/rightmark）

左标向右移动，右标向左移动

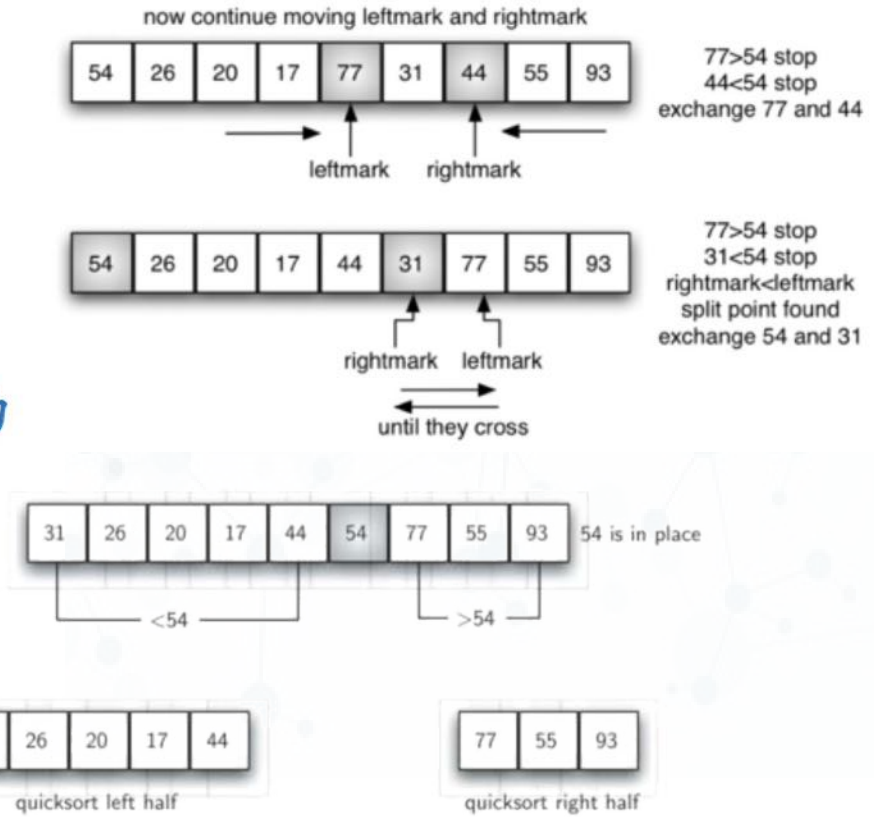
- 左标一直向右移动，碰到比中值大的就停止
- 右标一直向左移动，碰到比中值小的就停止
- 然后把左右标所指的数据项交换

继续移动，直到左标移到右标的右侧，停止移动

这时右标所指位置就是“中值”应处的位置

将中值和这个位置交换

分裂完成，左半部比中值小，右半部比中值大



问题解答

- **不是很清楚各类排序算法最优情况下的复杂度是怎么来的，特别是插入排序和归并**
- **插入排序：**

  最优情况就是：列表非常接近排好序，可以达到O(n)
- **归并排序：**

  如果不增加判断left[-1]和right[0]代码的话，所有情况都是稳定的O(nlogn)

  如果增加优化代码，那么最优情况同样是非常接近排好序，能达到O(n)

问题解答：各种排序算法复杂度大全

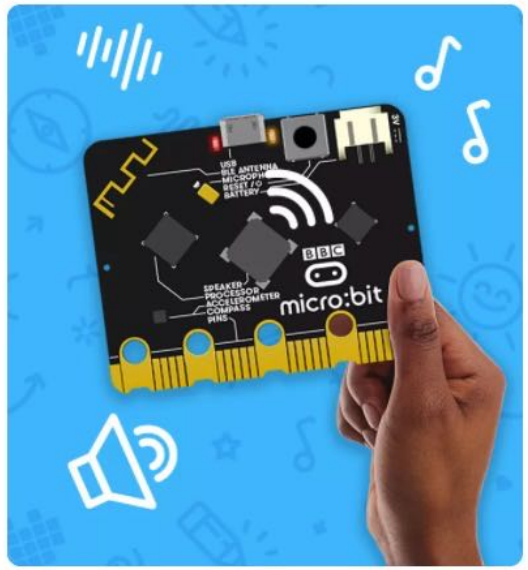
排序算法	平均时间复杂度	最坏时间复杂度	最好时间复杂度	空间复杂度	稳定性
冒泡排序	O(n²)	O(n²)	O(n)	O(1)	稳定
选择排序	O(n²)	O(n²)	O(n)	O(1)	不稳定
插入排序	O(n²)	O(n²)	O(n)	O(1)	稳定
快速排序	O(nlogn)	O(n²)	O(nlogn)	O(nlogn)	不稳定
堆排序	O(nlogn)	O(nlogn)	O(nlogn)	O(1)	不稳定
希尔排序	O(nlogn)	O(ns)	O(n)	O(1)	不稳定
归并排序	O(nlogn)	O(nlogn)	O(nlogn)	O(n)	稳定
计数排序	O(n+k)	O(n+k)	O(n+k)	O(n+k)	稳定
基数排序	O(N*M)	O(N*M)	O(N*M)	O(M)	稳定

【C1】Python开源硬件创意作品活动

- 报名：2人组；15个组；Python编程
- 截止5月16日，完成并提交作品的可保留micro:bit作为纪念；
- 班级投票，根据得票情况获得平时分的加分

The new BBC micro:bit

We're really excited to announce the launch of the latest BBC micro:bit.



报名

开发文档

- micropython文档
  http://docs.micropython.org/en/latest/

- micro:bit v2
  https://github.com/microbit-foundation/micropython-microbit-v2（micropython固件）
  https://microbit-micropython.readthedocs.io/en/v2-docs/（文档）

【K07】排序的课堂练习

1〉给定排序列表 [21, 1, 26, 45, 29, 28, 2, 9, 16, 49, 39, 27, 43, 34, 46, 40]，在归并排序的第3次递归调用时，排序的是哪个子表?

a) [16, 49, 39, 27, 43, 34, 46, 40]

b) [21,1]

c) [21, 1, 26, 45]

d) [21]

2〉排序数据同上，归并排序中，哪两个子表是最先归并的?

a) [21, 1] and [26, 45]

b) [1, 2, 9, 21, 26, 28, 29, 45] and [16, 27, 34, 39, 40, 43, 46, 49]

c) [21] and [1]

d) [9] and [16]

【K07】排序的课堂练习

3〉给定排序列表[14, 17, 13, 15, 19, 10, 3, 16, 9, 12]，快速排序在第2次分裂后，列表内容是：

a) [9, 3, 10, 13, 12]

b) [9, 3, 10, 13, 12, 14]

c) [9, 3, 10, 13, 12, 14, 17, 16, 15, 19]

d) [9, 3, 10, 13, 12, 14, 19, 16, 15, 17]

4〉给定排序列表 [1, 20, 11, 5, 2, 9, 16, 14, 13, 19]，快速排序如果采用“三点取样”法，得到的第1个“中值”是：

a) 1 b) 9 c) 16 d) 19

5〉下面哪些算法，即使在最坏情况下，复杂度还保证是O(nlog n)

a)谢尔排序 b)快速排序 c)归并排序 d)插入排序

下课！