数据结构与算法

Data Structure and Algorithm

I. 简介

授课人：Kevin Feng
翻译：　王 落 桐

特别感谢！

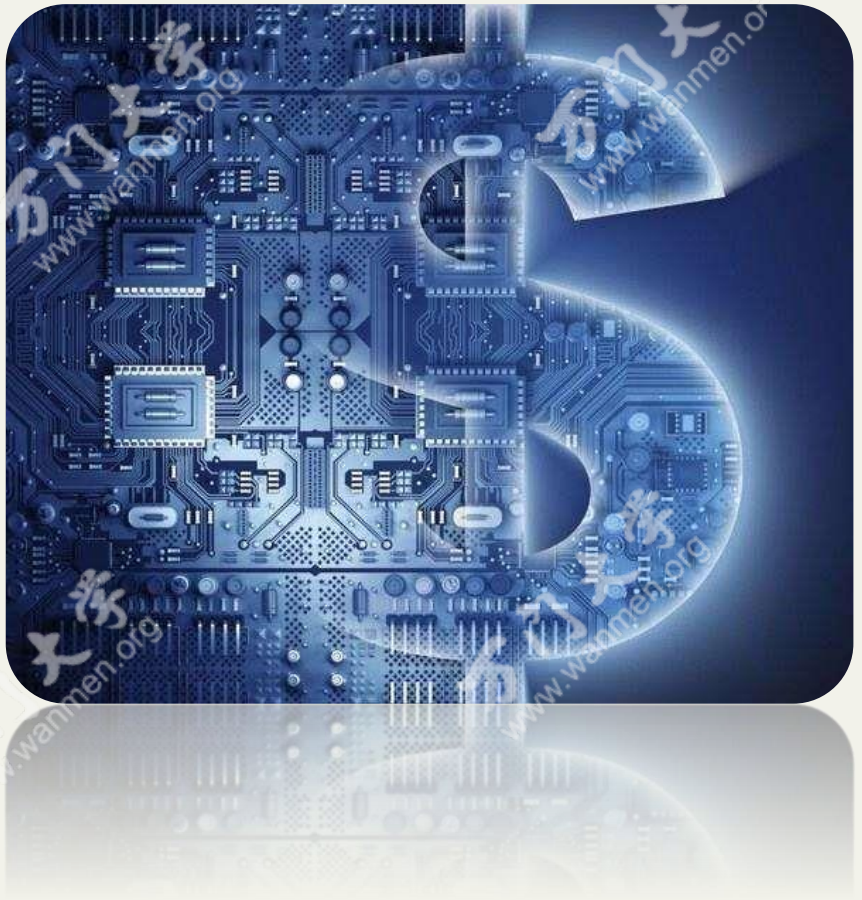
- 梁少华
- 潘婧
- 孙兴
- 王落桐
- 赵其辰
- 赵伟明

按拼音顺序排列,排名不分先后

课程总览

- 自我介绍
- 课程简介
- 职业规划
- 面试简介

关于我 | Kevin Feng



- ✓ 四大银行7年金融量化分析师
- ✓ 带领创业公司团队开发企业级应用
- ✓ 计算机科学博士学位
- ✓ Capital One 上海高级软件工程师

课程简介 Course Introduction

2 个月高强度学习IT面试必考知识

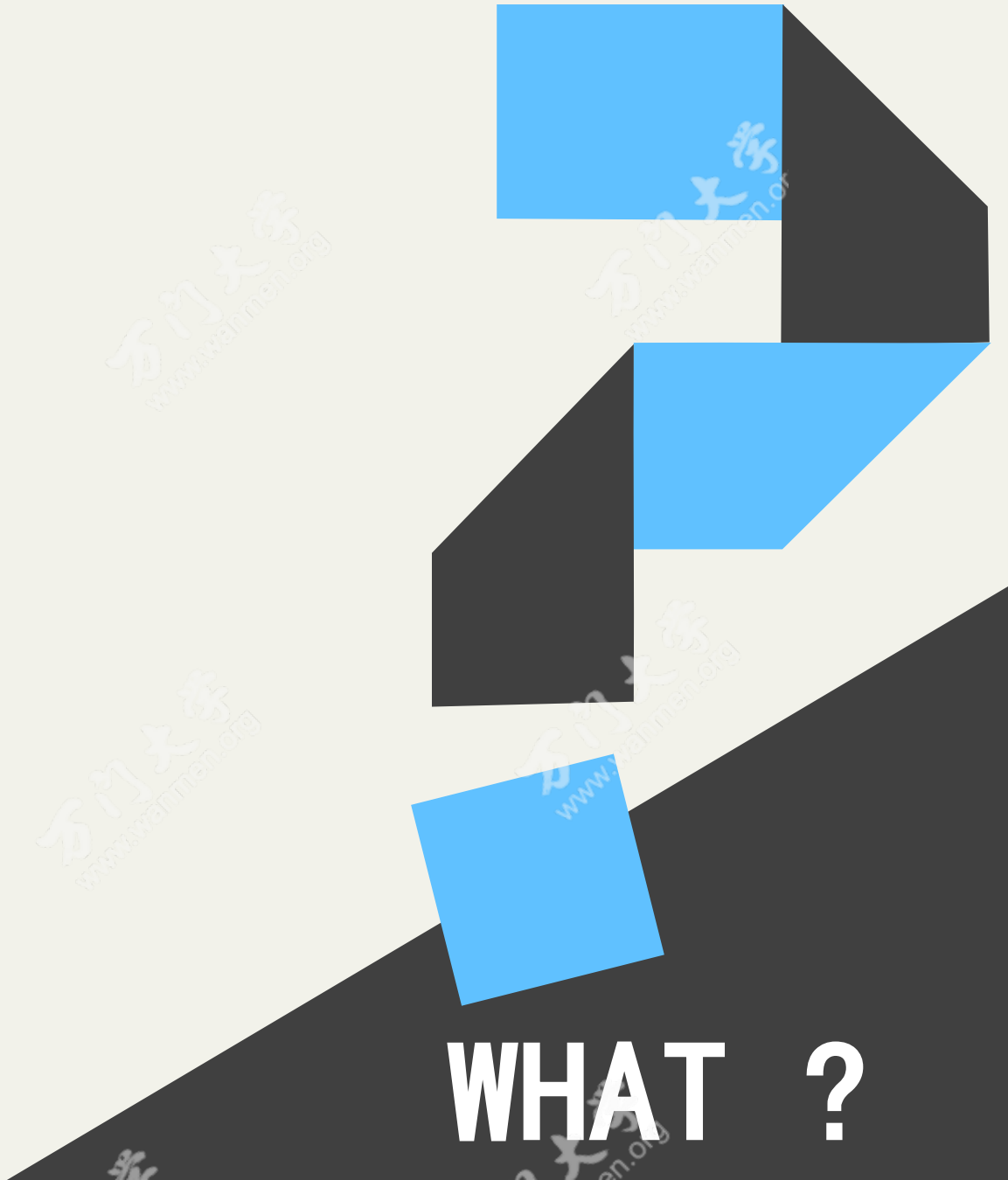
48 课时系统讲解数据结构和算法

300 道顶尖科技公司技术面试真题

3 大火热职位：
- 数据科学家：容易
- 数据工程师：容易到中等难度
- 软件工程师：中等难度到有挑战

4 大面试考察方向：基础 + 数据结构 + 算法 +项目

70% FLAG面试来自计算机语言，数据结构和算法



课程安排 | Course Agenda

Week 01	1	【Algorithm 01】 Introduction
	2	【Algorithm 02】 ArrayList
Week 02	3	【Algorithm 03】 Recursion
	4	【Algorithm 04】 Search and Sort
Week 03	5	【Algorithm 05】 Binary Search
	6	【Algorithm 06】 Divide and Conquer I
Week 04	7	【Algorithm 07】 Divide and Conquer II
	8	【Algorithm 08】 Linked List
Week 05	9	【Algorithm 09】 Linked List II
	10	【Algorithm 10】 Stack & Queue
Week 06	11	【Algorithm 11】 Stack & Queue II
	12	【Algorithm 12】 Hashtable
Week 07	13	【Algorithm 13】 Hashtable II
	14	【Algorithm 14】 Tree
Week 08	15	【Algorithm 15】 Tree II
	16	【Algorithm 16】 Heap
Week 09	17	【Algorithm 17】 Graph
	18	【Algorithm 18】 Graph II - DFS / BFS
Week 10	19	【Algorithm 19】 Graph III - Dijkstra
	20	【Algorithm 20】 Graph IV - Union Find
Week 11	21	【Algorithm 21】 Two Pointers
	22	【Algorithm 22】 Sliding Windows
Week 12	23	【Algorithm 23】 DP I
	24	【Algorithm 24】 DP II
Week 13	25	【Algorithm 25】 DP III
	26	【Algorithm 26】 Bit Manipulate/Math
Week 14	27	【Algorithm 27】 Greedy
	28	【Algorithm 28】 String I
Week 15	29	【Algorithm 29】 String II
	30	【Algorithm 30】 Mock Interview

职业生涯路线 | Career Path Insight

- B

level	最高	中间值	最低	股票
T1,	140000	106000	70000	
T2	180000	136000	90000	
T3	260000	194000	130000	
T4	360000	266000	180000	
T5	450000	331000	230000	30W-40W
T6	600000	426000	290000	35W-55万
T7	700000	572000	390000	45W-70万
T8	1000000	736000	500000	70W-100万
T9	1300000	980000	660000	90W-120W

- A

级别	薪资	股票（4年拿完）
P5	15W-25W	无
P6	20W-35W	无
P7	30W-50W	2400股
P8	45W-80W	6400股
P9	80W-100W	16000股
P10		

- T

级别		基本定义	薪酬范围
T1 应届生	1.1	低端岗位	10k+
	1.2	本科生	
	1.3	研究生	
T2	2.1	博士生	10K，14w
	2.2	工程师	15K，21w
	2.3	工程师	20K，28w
T3 高级	3.1	资深工程师	23K以上，32w以上
	3.2	技术专家	50-70w
	3.3	总监级	60w、70w
T4 专家		总监以上，基本没有	大于100万，另外100万股票/期权

软件工程师面试

- 数据结构（70%–80%）
- 系统设计
- 简历 （课程，项目，经验，等等） （10%）
- 计算机语言（不限，Java，Python，C++，等）
- 行为面试 （10%）
- 项目相关：操作系统，网络，等

数据工程师面试

- 数据结构和算法
- 编程语言
- 大数据通用处理平台
- 分布式存储与计算
- 并发程序设计
- 使用 Hadoop，Spark，Kafka，Hive 等工具
- 数据分析/数据仓库（SQL类）
- 开发ETL/数据流水线（data pipelines）
- 数据可视化

数据科学家面试

- 数学，统计
- SQL
- 数据建模
- 机器学习

Linear Regression, Logistics Regression, Clustering, Decision Tree, Time Series,

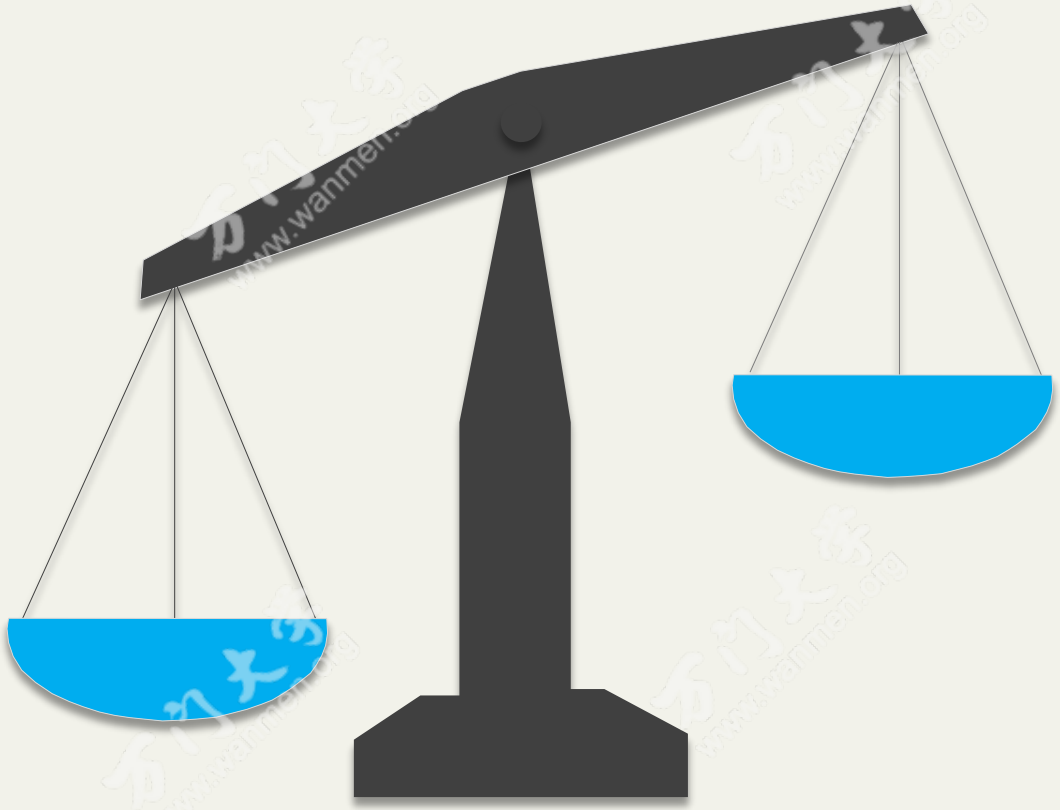
Random Number, Monte-Carlo, Bayesian, Bayes, SVM, KNN, etc.

- 算法
- 商业智能的 dashboards
- 会读paper，会写research proposal
- R / Python / MATLAB / SAS
- 数据可视化

划重点

算法的要求

- 软件工程师：高
- 数据工程师：中
- 数据科学家：一般



语言的要求

- 软件工程师：高
- 数据工程师：高
- 数据科学家：中

基础知识

- 数据结构简介
- 算法简介
- 如何学习
- 算法时间复杂度分析

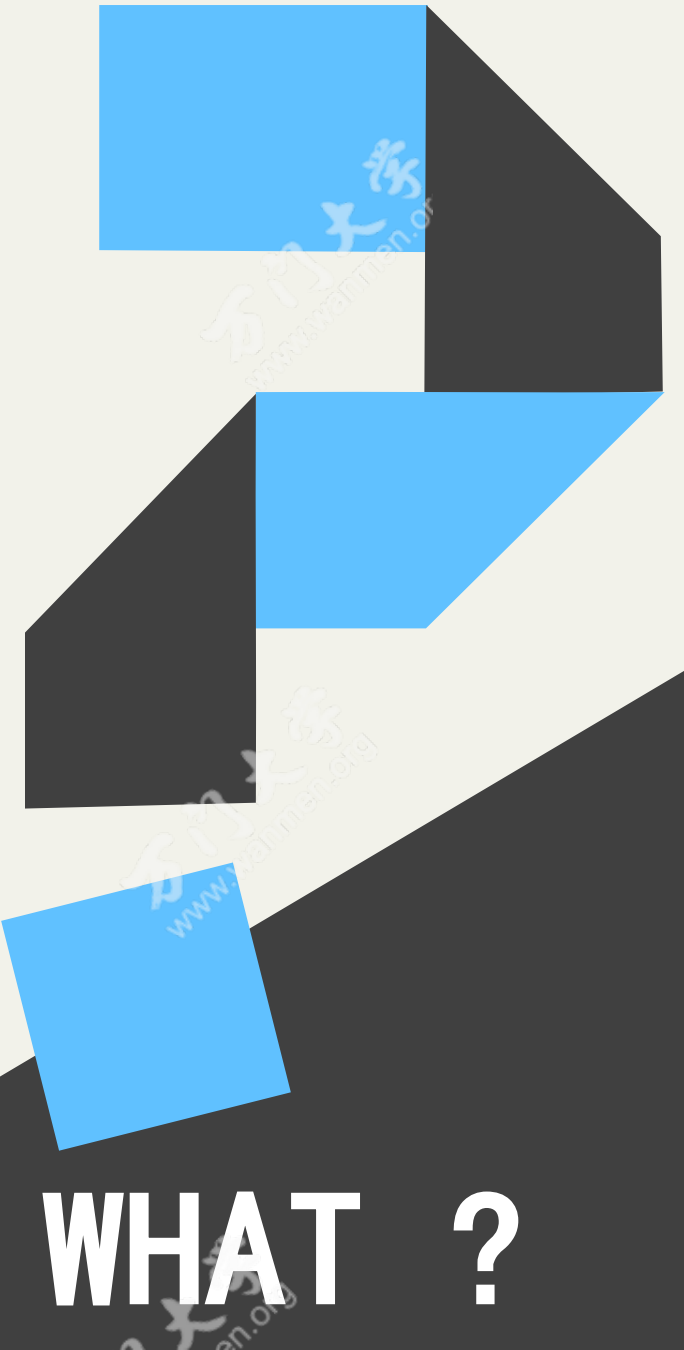
CONTENTS

目录

- 什么是数据结构与算法
- 为什么要使用算法
- 一个简单的例子
- 如何准备面试
- 数学回顾
- 简单数据结构实例：Array List

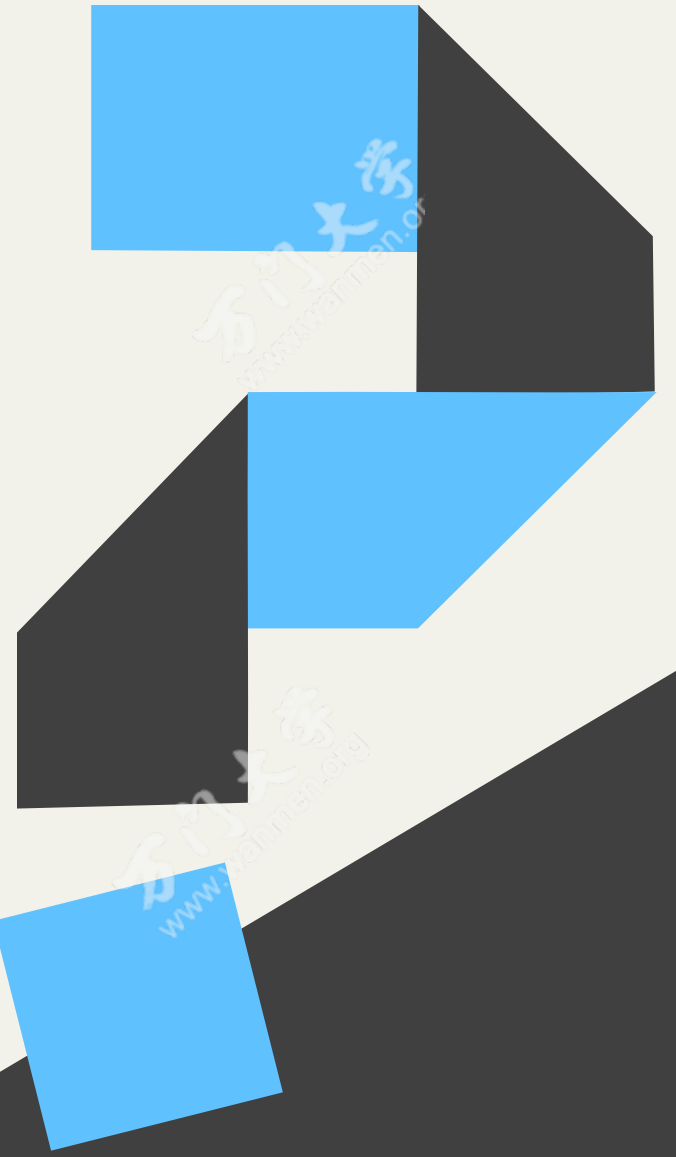
数据结构 | Data Structure

- 数据结构：数据结构是一种特定的计算机存储、组织数据的方式。其宗旨是使计算机能够高效的使用数据。
  - 越强大的计算机 ➔ 越复杂的数据结构
- 抽象的数据类型 (ADT): 数列, 列表, 树, 表格.....
  - 对于某一类型的数据或者某一个数据集的描述以及对该数据的各种操作
- ADTs 拥有干净的接口，其具体的实施细节是封装起来的



算法 | Algorithm

- 算法：有限时间内解决问题的一系列清晰指令
- 效率
  - 空间
  - 时间
- 目标
  - 能够识别程序要求的功能以解决当前任务
  - 设计能够高效解决此任务的数据结构与算法
  - 评价该方案的效率和正确性



WHAT ？

简单的例子

- ✓ 找到丢失的数字

  现在你手上有n-1个数字，这些数字的范围是［1 ， n］且这n-1个数字中没有重复的数字。由上述条件可知：你手上的数字丢失了一个。请编写一段高效的找到该缺失数字的代码。

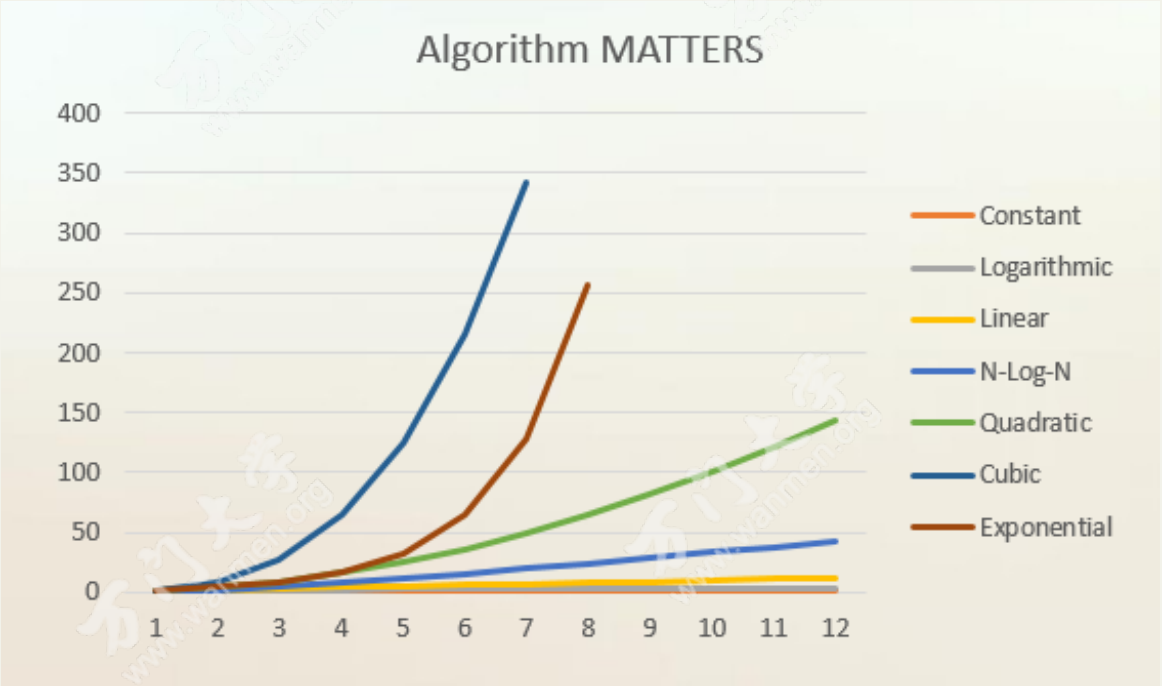
- ✓ 找到亚马逊中前K个最经常被搜索的物品
  - 这可是亚马逊！那得有多少个商品？
  - 老板着急要答案
  - 如何能做实时显示呢？比如说每5分钟？

如何学习数据结构和算法？

- 基础知识
- 编程语言
- 如何练习代码
- 推广和总结

算法分析

- 运行时间（时间复杂度）
  - 运行时间会随着输入的大小如何变化
  - 最好的情况：运行时间的上限（最少运行时间）
    由最简单的输入决定
    提供了所有输入的最终优化目标
  - 最差的情况:运行时间的下限（最多运行时间）
    由最复杂的输入决定
    提供了所有输入的保障时间
  - 平均情况：随机输入的运行时间的期望
    需要建立随机输入的模型
    是一种评价算法表现的方法
  - 平均情况时间通常很难测定
  - 我们通常情况下关注最差情况下的运行时间



理论分析

- 归纳算法的本质，而不是采用流程模式
- 建立运行函数关于输入大小 (n)的函数
- 考虑各种可能的输入
- 在硬件和软件上分别考虑评估算法的速度
- 伪代码
  - 找到一个数组中的最大数字

Algorithm *arrayMax*(*A*, *n*)	# operations
currentMax ← *A*[0]	2
for *i* ← 1 **to** *n* − 1 **do**	2*n*
if *A*[*i*] > *currentMax* **then**	2(*n* − 1)
currentMax ← *A*[*i*]	2(*n* − 1)
{ increment counter *i* }	2(*n* − 1)
return *currentMax*	1
Total	8*n* − 3

- 算法基本操作
  - 评估表达式
  - 赋值给变量
  - 数组索引
  - 调用方法
  - 从一个方法中得到返回值

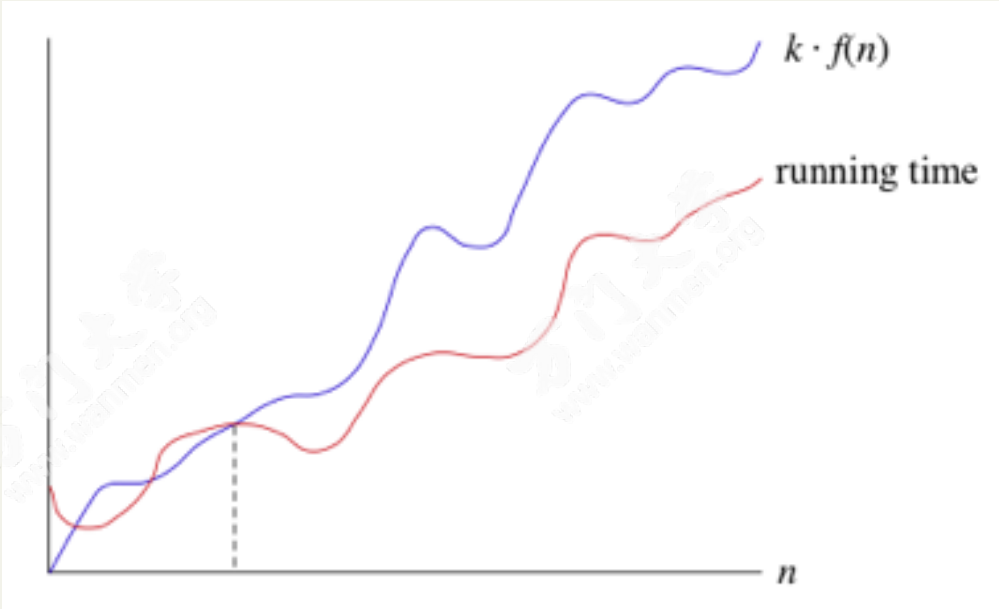
- 通过检查伪代码，我们可以找到被算法执行的基本操作的最大数字，也就是找到关于输入大小的函数
  - 找出算法中的基本语句
  - 计算基本语句的执行次数的数量级
  - 用大O记号表示算法的时间性能

评估算法运行时间

- 算法 arrayMax 在最差情况下执行了 8n − 3 个基本操作。定义:
  - a = 最快的基本操作所需要的运行时间
  - b = 最慢的基本操作所需要的运行时间
- 令 T（n）为 arrayMax 的运行时间。 则应该有:
  - a （8n − 3）⩽ T（n）⩽ b （8n − 3）
- 因此，运行时间 T（n）由两条线性函数所划定其范围

近似记法

- Big - O 记法
  - 一般习惯用 Θ (n) 记法来 渐进界定算法运行时间的常数函数边界。有时，我们只希望这个常数函数代表算法运行时间的上界。
  - 尽管在最差情况下二分法搜索的运行时间为 Θ (lgn)，认为在所有情况下二分法搜索的搜索时间为 Θ (lgn) 是错误的。
  - 二分法搜索的时间从来不会超过 Θ (lgn)，多数情况下其搜索时间都会少于Θ (lgn)

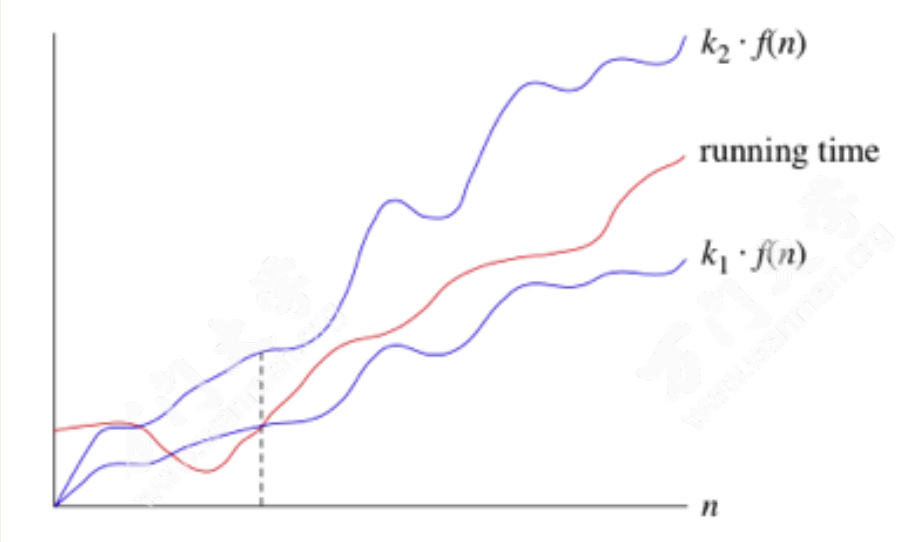


时间复杂度

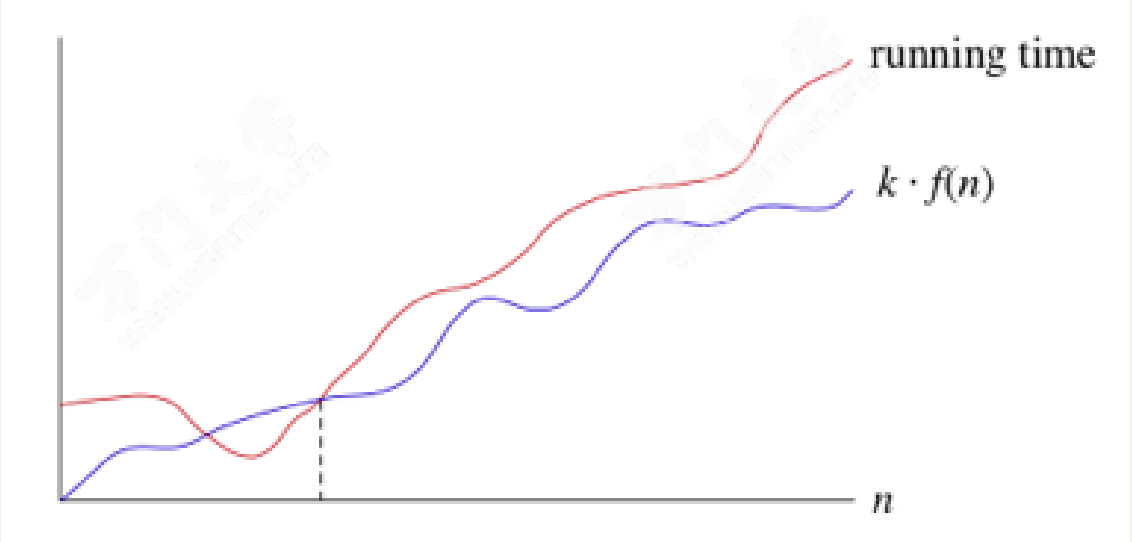
- 一般情况下，算法中基本操作重复执行的次数是问题规模n的某个函数，用T(n)表示，若有某个辅助函数f(n)，使得当n趋近于无穷大时，T(n)/f(n)的极限值为不等于零的常数，则称f(n)是T(n)的同数量级函数。 记作T(n)=O(f(n)),称O(f(n)) 为算法的渐进**时间复杂度**，简称**时间复杂度。**
- **T(n) = O(f(n))** 表示存在一个常数C，使得在当n趋于正无穷时总有 T(n) ≤ C * f(n)。简单来说，就是T(n)在n趋于正无穷时最大也就跟f(n)差不多大。也就是说当n趋于正无穷时**T(n)**的上界是**C * f(n)**。

近似记法

- Big-θ （Big-Theta）记法
  - 当特别说明运行时间为 Θ (n)时：
    当n变得很大的时候，算法的运行时间介于$k_1 \cdot n$ 和$k_2 \cdot n$之间，其中k_1和k_2 是常数。



- Big-Ω （Big-Omega）记法
  - 有的算法我们只能描述其至少要运行多少时间而无法给出其运行时间的上限时，采用Big-Ω 记法。
  - 如果算法的运行时间是Ω (f (n)), 则对于足够大的n来说，运行的时间至少是k·f (n),其中k为常数。



记法总结

notation	provides	example	shorthand for	used to
Big Theta	asymptotic order of growth	$\Theta(N^2)$	$\frac{1}{2} N^2$ $10 N^2$ $5 N^2 + 22 N \log N + 3N$ ⋮	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	$10 N^2$ $100 N$ $22 N \log N + 3 N$ ⋮	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2} N^2$ N^5 $N^3 + 22 N \log N + 3 N$	develop lower bounds

P 与 NP

- Polynomial
  - 多项式类问题
  - O(log$_2 n$)、O(n)、 O(nlog$_2 n$)、O(n^2)和O(n^3)

- Non-Deterministic Polynomial
  - 非确定多项式问题
  - O(2^n)和O(n!)

- **我们的课，只讨论P的问题。**

- P = NP ?
  - It asks whether every problem whose solution can be quickly verified (technically, verified in polynomial time) can also be solved quickly (again, in polynomial time).
  - $ 1,000,000
  - 3-SAT

主项定理（Master Theorem）

设 $a \geq 1$ 和 $b > 1$ 为常数，设 $f(n)$为一函数，$T(n)$由递归式

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

其中$\frac{n}{b}$指$\left\lfloor\frac{n}{b}\right\rfloor$和$\left\lceil\frac{n}{b}\right\rceil$，可以证明，略去上下去整不会对结果造成影响。那么 $T(n)$可能有如下的渐进界

(1)若 $f(n) < n^{\log_b^a}$，且是多项式的小于。即

$\exists\, \varepsilon > 0$，有 $f(n) = O\left(n^{\log_b^a - \varepsilon}\right)$，则 $T(n) = \Theta\left(n^{\log_b^a}\right)$

(2)若 $f(n) = n^{\log_b^a}$，则 $T(n) = \Theta\left(n^{\log_b^a}\log n\right)$

(3)若 $f(n) > n^{\log_b^a}$，且是多项式的大于。即

$\exists\, \varepsilon > 0$，有 $f(n) = \Omega\left(n^{\log_b^a + \varepsilon}\right)$，且对$\forall\, c < 1$ 与所有足够大的 n，有 $af\left(\frac{n}{b}\right) \leq cf(n)$，则 $T(n) = \Theta\left(f(n)\right)$

$T(n) = 3T(n/2) + n^2 \Longrightarrow T(n) = \Theta(n^2)$ (Case 3)

$T(n) = 4T(n/2) + n^2 \Longrightarrow T(n) = \Theta(n^2 \log n)$ (Case 2)

$T(n) = T(n/2) + 2^n \Longrightarrow \Theta(2^n)$ (Case 3)

$T(n) = 16T(n/4) + n \Longrightarrow T(n) = \Theta(n^2)$ (Case 1)

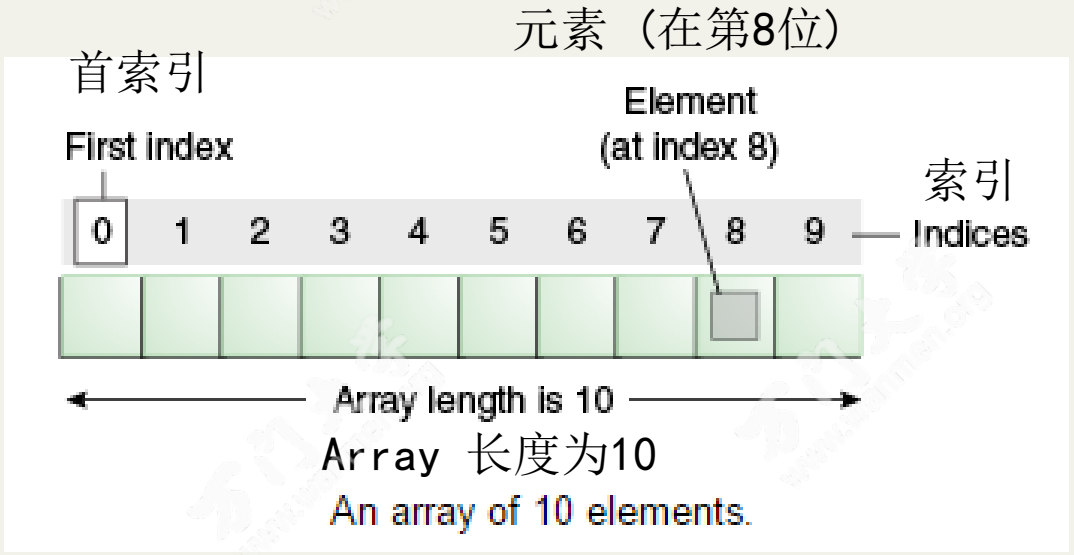
$T(n) = 2T(n/2) + n \log n \Longrightarrow T(n) = n \log^2 n$ (Case 2)

实战

- Array简介
- Array应用场景举例
- 家庭作业
- 动手写代码

Array（数组）

- 盛有单一类型固定数量值的容器类
  - 以0开始的索引
  - 数组长度
  - 内存表示：连续的
  - 边界检查
- Array 分析
  - 读取
  - 尾部增加删除
  - 中间增加删除
  - 改变形状
  - 按位查找
  - 按值查找
  - 打印输出所有元素



10个元素的Array

Array（数组）应用1

- 计算一元二次方程解

  $aX^2 + bX + c \geq 0$

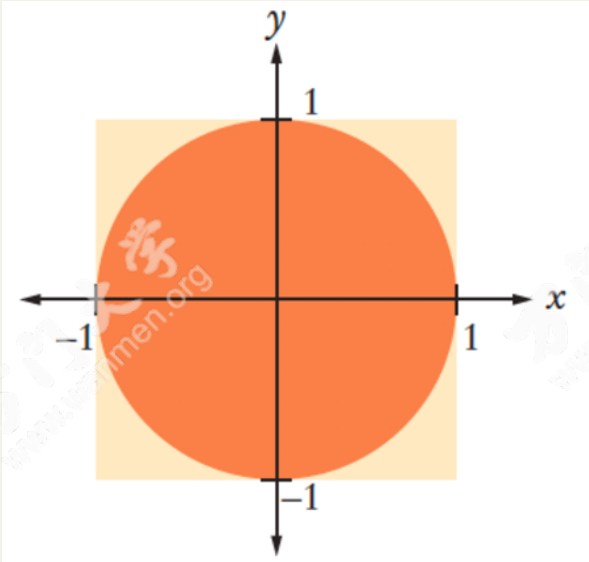
  - $b^2 - 4ac \geq 0$, return two results
  - $b^2 - 4ac < 0$, raise error

- 歌唱比赛
  - 一个歌唱比赛的歌手打分，我们设计一个程序帮助现场去掉一个最低分和一个最高分，再计算一个平均分

Array（数组）应用2

- 计算 π 值
  - $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots,$
  - (模拟蒙特卡洛)Monte Carlo Simulation

    $X^2 + Y^2 = 1$



Array（数组）应用3

- 打印99乘法表

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12 4*4=16
1*5=5  2*5=10 3*5=15 4*5=20 5*5=25
1*6=6  2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7  2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8  2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9  2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
```

Array（数组）应用4

- 洗牌问题
  - 用Array编写洗牌程序
  - 如何测试？
- 卡牌收集问题
  - 假设你有一副洗好了的牌，你现在一张一张的进行翻牌，请问你要翻多少张才能够凑齐一个花色？假定一副牌有N种类型，那么你至少需要翻多少张才能凑齐一个类型？

Array（数组）应用5

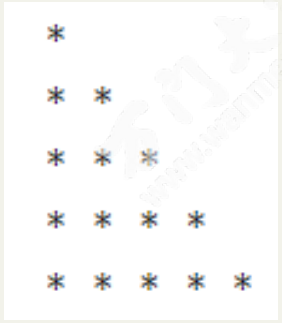
- 计算素数（质数）
  - 计算素数函数π（N）代表素数的个数小于等于N
  - 举个例子：　π（17）=7，因为前7个素数为2，3，5，7，11，13，17
- 证明哥德巴赫猜想
  - 1742年，哥德巴赫提出了著名的哥德巴赫猜想。即：任一大于2的偶数都可写成两个质数之和。比如说16=3+13。试着编码写出程序：只包含N作为参数并且给出N为两个质数之和的表达式。哥德巴赫猜想至今没有被证明，但是目前已知其在N小于10^{14}的时候都是成立的。

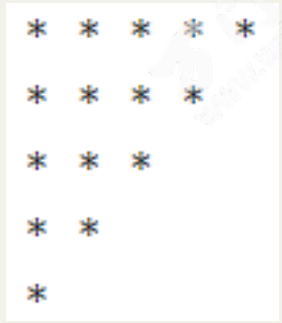
Array（数组）应用6

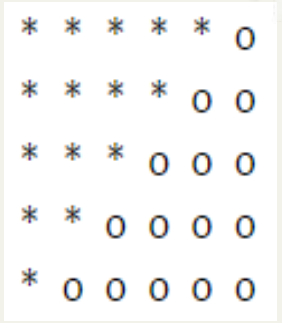
- 1-bit 和 2-bit的字符
  - 现有两个字符，第一个字符用1-bit的0表示，第二个字符可以用2-bit表示（10或11）
  - 现在给出一个以0结尾的字符串，试分析其最后一个字符是否是1-bit表示的字符。
  - Eg1：bits = [1,0,0]，输出：True。（解释：该字符串的唯一编码方式是10，0）
  - Eg2：bits = [1,1,1,0]，输出：False。（解释：该字符串的唯一编码方式是11，10）

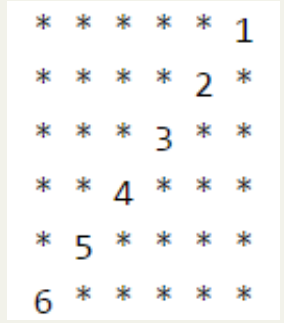
Array （数组）作业1

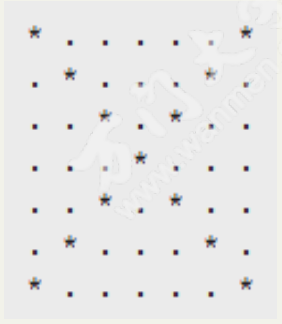
- 打印下图：

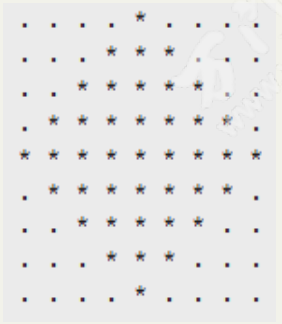


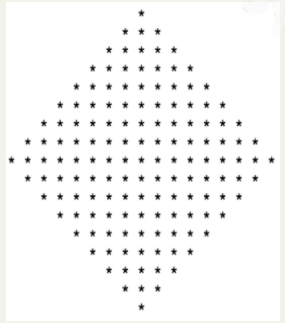












Array （数组）作业2

- 素数间隔：
  - 试编写程序：输入N，输出：2-N之间的无素数最长连续数字数列
- 构造Hadamard 矩阵：

阿达马矩阵最初的构造的例子是由[[詹姆斯·约瑟夫·西尔维斯特]]给出的。假设"H"是一个"n"阶的阿达马矩阵，则下面的矩阵

$$\begin{bmatrix} H & H \\ H & -H \end{bmatrix}$$

给出一个2n阶的阿达马矩阵。连续使用这个方法，我们可以给出下面的一系列矩阵：

$$H_1 = \begin{bmatrix} 1 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Array （数组）作业3

- 螺旋矩阵
  - 给定一个二维矩阵，给出其螺旋排列形式

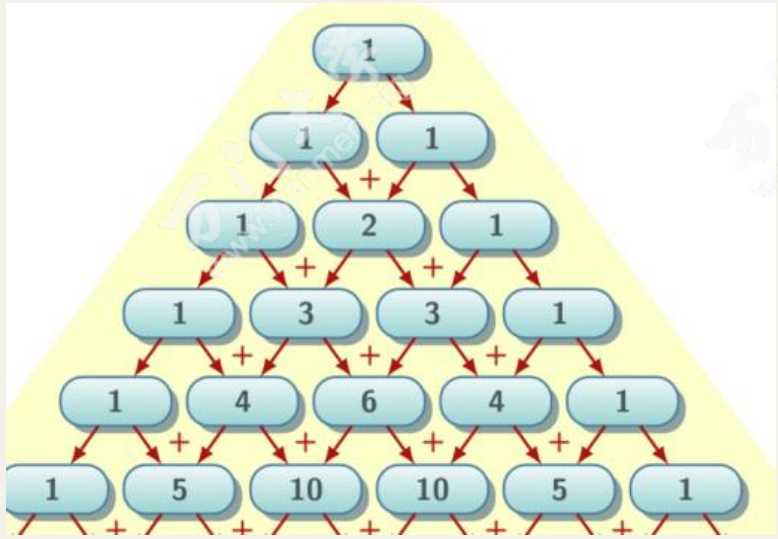
```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16

1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

- 对角矩阵

1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8
5	6	7	8	9

- 帕斯卡三角

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1



编程作业1

- 三门问题（Monty Hall problem)亦称为蒙特霍问题或蒙提霍尔悖论，大致出自美国的电视游戏节目Let's Make a Deal。问题名字来自该节目的主持人蒙提·霍尔（Monty Hall)。
- 假定你参与了竞猜节目看见三扇关闭了的门，已知其中一扇的后面有一辆汽车，选中后面有车的那扇门可赢得该汽车，另外两扇门后面则各藏有一只山羊。当你选定了一扇门，但未去开启它的时候，节目主持人开启剩下两扇门的其中一扇，露出其中一只山羊。主持人其后问你要不要换另一扇仍然关上的门。你是否选择更改自己原始的答案？
- 用数学方法解决上述问题
- 编写一段Python代码来仿真该过程

编程作业2

- 新生儿问题：1月1日清晨婴儿房中有3名男婴和若干名女婴，1月2号婴儿房中又新增了一名婴儿，1月3号时护士从婴儿房中随机抱出了一名婴儿，该婴儿为男婴。求1月2号新增的婴儿为男婴的概率？
- 用数学方法解决上述问题
- 编写一段Python代码来仿真该过程

编程作业3

- 掷骰子喝酒问题：开始时有编号1-6的6个空杯子，开始掷骰子，如果点数对应的杯子是空的，那么将此空杯满上。如果点数对应的杯子是满的，你则需要喝掉它。如果有5杯都是满的，你恰好选中了空杯的点数，这个时候你需要喝掉6杯酒。游戏结束。求该游戏结束轮次的数学期望。
- 用数学方法解决上述问题
- 编写一段Python代码来仿真该过程

数据结构与算法

Data Structure and Algorithm

I. 简介结束

授课人：Kevin Feng
翻译：　王 落 桐

总结

Week 01	1	【Algorithm 01】	Introduction
	2	【Algorithm 02】	ArrayList
Week 02	3	【Algorithm 03】	Recursion
	4	【Algorithm 04】	Search and Sort
Week 03	5	【Algorithm 05】	Binary Search
	6	【Algorithm 06】	Divide and Conquer I
Week 04	7	【Algorithm 07】	Divide and Conquer II
	8	【Algorithm 08】	Linked List
Week 05	9	【Algorithm 09】	Linked List II
	10	【Algorithm 10】	Stack & Queue
Week 06	11	【Algorithm 11】	Stack & Queue II
	12	【Algorithm 12】	Hashtable
Week 07	13	【Algorithm 13】	Hashtable II
	14	【Algorithm 14】	Tree
Week 08	15	【Algorithm 15】	Tree II
	16	【Algorithm 16】	Heap
Week 09	17	【Algorithm 17】	Graph
	18	【Algorithm 18】	Graph II - DFS / BFS
Week 10	19	【Algorithm 19】	Graph III - Dijkstra
	20	【Algorithm 20】	Graph IV - Union Find
Week 11	21	【Algorithm 21】	Two Pointers
	22	【Algorithm 22】	Sliding Windows
Week 12	23	【Algorithm 23】	DP I
	24	【Algorithm 24】	DP II
Week 13	25	【Algorithm 25】	DP III
	26	【Algorithm 26】	Bit Manipulate/Math
Week 14	27	【Algorithm 27】	Greedy
	28	【Algorithm 28】	String I
Week 15	29	【Algorithm 29】	String II
	30	【Algorithm 30】	Mock Interview

Summarization – Data Structure I

- Linked List　(链表)
  - Basic Knowledge　(基础知识)
  - Two Pointers / Runner Technique　(双向指针)
  - Reverse　(颠倒)
  - Sort　(排序)
  - Combine with Other Data Structures　(与其他数据结构合并)
- Stack, Queue and Deque　(栈，队列，双端队列)
  - Basic Knowledge　(基础知识)
  - Stack / Queue Construction　(栈/队列的构建)
  - Stack / Queue Basic Application　(栈/队列的基础应用)
  - Index Stack *　(索引栈)
  - Calculator　(计算器)
  - BFS, DFS　(广度优先搜索，深度优先搜索)

Summarization – Data Structure II

- Hashtable （哈希表）
  - Basic Knowledge （基础知识）
  - Hash Function and Hashcode （哈希函数以及哈希编码）
  - Collision （冲突）
  - Open Addressing （开放寻址）
  - Separate Chain （独立链表）
  - equals
  - Rehash （重新配置）
  - Double Hash （双重哈希）
  - LinkedHash （哈希链表）
  - TreeMap （树状匹配）
  - Rolling Hash (*) （滚动哈希）
  - Customized Hash Object （自定义哈希类）
  - DFS, BFS （广度优先搜索，深度优先搜索）
  - Count Sort, Array as Hash, int as Hash （计算排序，哈希数组，哈希整数）

Summarization – Data Structure III

- Tree （树）
  - Basic Knowledge （基础知识）
  - Recursion （递归）
  - Iteration （迭代）
  - Traversal （遍历）
  - DFS, BFS （广度优先搜索，深度优先搜索）
  - Balanced Tree （*） （平衡树）
- Heap （堆）
  - Basic Knowledge （基础知识）
  - Top K and Counting （最大值与计数）
  - Sort （排序）
  - Streaming （流）

Summarization – Data Structure IV

- Graph （表）
  - Basic Knowledge （基础知识）
  - Implementation （实现）
  - DFS, BFS （广度优先搜索，深度优先搜索）
  - Recursion vs Iteration （递归和迭代）
  - Shortest Path （最短路径）
  - Topology Sort （拓扑排序）
  - Union Find （并集）

Summarization – Data Structure V

- Array, ArrayList, String （数组，数列，字符串）
  - Sort （排序）
  - Search （搜索）
  - Recursion （递归）
  - Binary Search （二分法搜索）
  - Divide and Conquer （分而治之）
  - Two Pointers （双向指针）
  - Sliding Windows （滑动窗口）
  - Dynamic Programming （动态规划）
  - Greedy（贪心算法）
  - Brute Force （暴力破解）

Summarization – Algorithm

- Divide and Conquer （分而治之）
- Dynamic Programming （动态规划）
  - 1-D
  - 2-D
  - 3-D
- Greedy （贪心算法）
- Bit Manipulation （位操作）
- Multi-Solution Questions （多解决方案问题）