数据结构与算法（Python）-04/0228

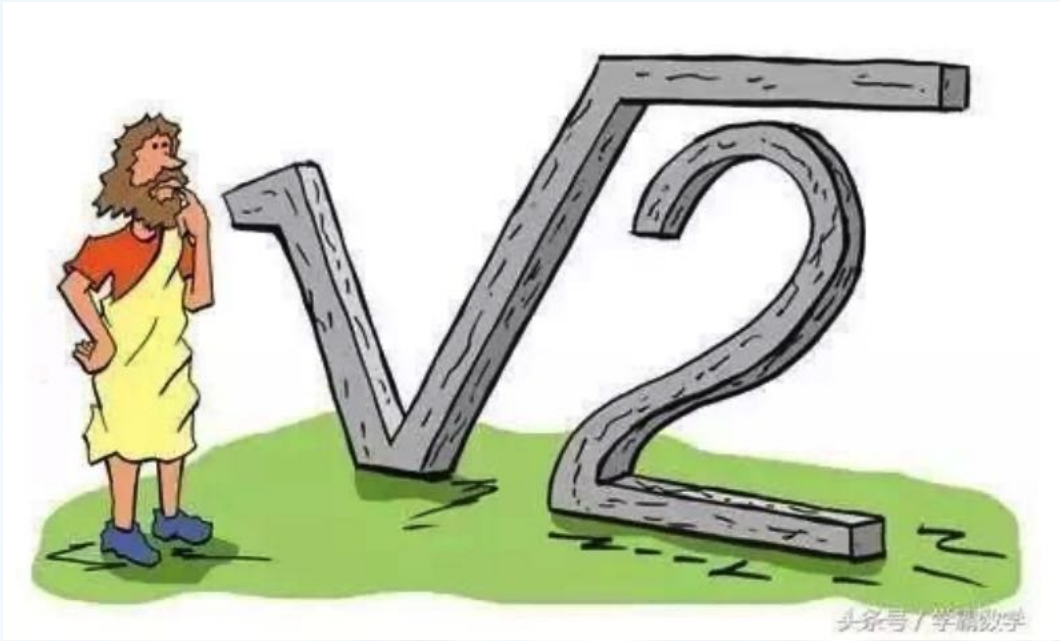
陈斌 gischen@pku.edu.cn 北京大学地球与空间科学学院

目录

- 两周的内容小结
  - W01：概述
  - W02：算法分析
- 关于H1作业
- 关于H2作业
- 问题解答

W01-概述

- 101 引子：数据时代
- 102 问题求解的计算之道
- 103 图灵机计算模型
- 104 算法和计算复杂性
- 105 突破计算极限
- 106 什么是抽象和实现
- 107 为什么研究数据结构与算法



W01-101-数据时代

信息时代就是数据的时代

- 人类在各个领域的生产生活
- 无时不刻在产生着巨量的数据

大数据Bigdata的5V特性

Volume（海量）

Velocity（高速增长）

Variety（多种类型）

Value（低价值密度）

Veracity（真实性）

W01-102-问题求解的计算之道

抽象的“计算”概念提出

❖ **基于有穷观点的能行方法**

由有限数量的明确有限指令构成；

指令执行在有限步骤后终止；

指令每次执行都总能得到唯一结果；

原则上可以由人单独采用纸笔完成，而不依靠其它辅助；

每条指令可以机械地被精确执行，而不需要智慧和灵感。

“计算”的特征

有限、明确、唯一、机械

有一些看似“随机”的算法，也还是符合“计算”的特征

W01-103 图灵机计算模型

图灵机的基本定义

❖ **图灵机由以下几部分构成**

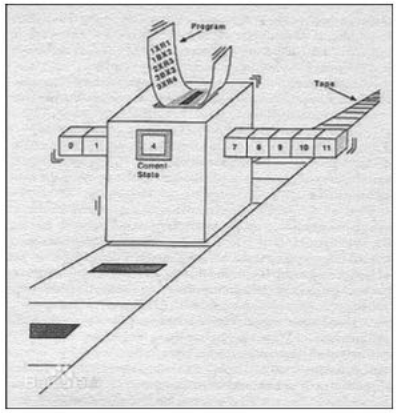
一条无限长的分格纸带，每格可以记录1个符号

一个读写头，可在纸带上左右移动，能读出和擦写格子的字符

一个状态寄存器，记录有限状态中的1个状态

一系列有限的控制规则：

- 某个状态，读入某个字符时：
- 要改写成什么字符
- 要如何移动读写头
- 要改变为什么状态



- **图灵机是计算模型的一种**

  理论证明，各种计算模型实际上都是等价的。

- **图灵机这个词，既指这个计算模型，又指具体的某一组规则**

- **存在死循环的图灵机**

  `<s0, B, B, s0, N>`

  或者：

  `<s0, B, B, s0, L>`

  或者更复杂的。

W01-104-算法和计算复杂性

计算复杂性

- “基于有穷观点的能行方法”的“可计算”概念

  仅仅涉及到问题的解决是否能在有限资源内（时间/空间）完成

  并不关心具体要花费多少计算步骤或多少存储空间

- 由于资源（时间/空间）相当有限，对于问题的解决需要考虑其可行性如何

- 人们发现各种不同的可计算问题，其难易程度是不一样的

  有些问题非常容易解决，如基本数值计算；

  有些问题的解决程度尚能令人满意，如表达式求值、排序等；

  有些问题的解决会爆炸性地吞噬资源，虽有解法，但没什么可行性，如哈密顿回路、货郎担问题等

W01-105-突破计算极限

- **超大规模分布式计算**
- **新型计算技术**

  光子计算、DNA计算、量子计算
- **分布式智慧——众包**

  众多人脑智力

  突破机械指令的限制

Foldit：游戏化众包蛋白质结构分析

- 多人在线游戏，众多**玩家**在给定一个目标蛋白的情况下，用各种氨基酸进行**组装**，最终**拼凑**出这个蛋白的完全体
- 玩家只需要掌握基本方块的拼插技巧，即可跟全世界众多玩家一起协同工作，攻克科研难题，有**60万**人玩过这个游戏

W01-106-什么是抽象和实现

抽象（Abstraction）

- 为了更好地处理**机器相关性**或**独立性**，引入了“抽象”的概念
- 用以从“**逻辑** Logical”或者“**物理** Physical”的不同层次上看待问题及解决方案

- 编程是通过一种程序设计语言，将抽象的**算法**实现为计算机**可以执行**的**代码**的过程

  没有算法，编程无从谈起

程序设计语言实现算法的基本机制

- 程序设计语言需要为算法的实现提供实现“**过程**”和“**数据**”的机制

  具体表现为“控制结构”和“数据类型”

- 程序设计语言均有**语句**对应控制结构

  顺序处理、分支选择、循环迭代

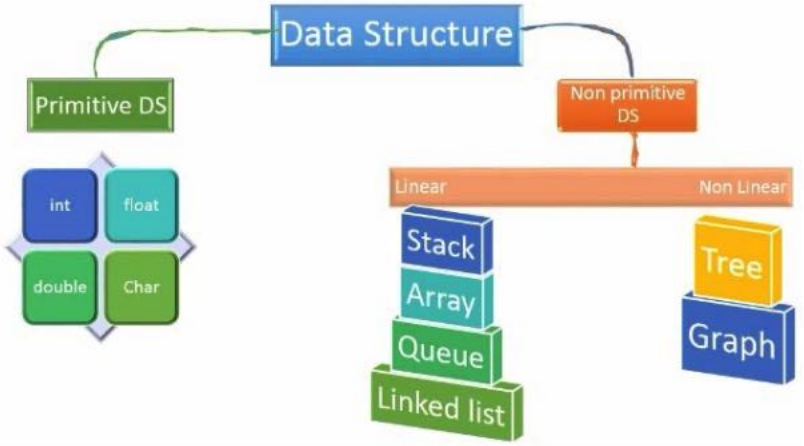
- 程序设计语言也提供最基本的**数据类型**来表示数据，如整数、字符等

  但对于复杂的问题而言，直接使用这些基本数据类型不利于算法的表达

W01-107 为什么研究数据结构与算法

ADT实现：数据结构Data Structure

- 对数据实现“逻辑”层次和“物理”层次的**分离**，可以定义复杂的数据模型来解决问题，而不需要立即考虑此模型如何实现



为什么要研究和学习算法

- 首先，学习各种不同问题的解决方案

  有助于我们在面对未知问题的时候，能够根据类似问题的解决方案来更好解决

- 其次，各种算法通常有较大差异

  我们可以通过算法分析技术来评判算法本身特性

  而不仅仅根据实现算法的程序在特定机器和特定数据上运行的表现来评判它

  即使同一个程序，在不同的运行环境和输入数据的情况下，其表现的差异可能也会很大

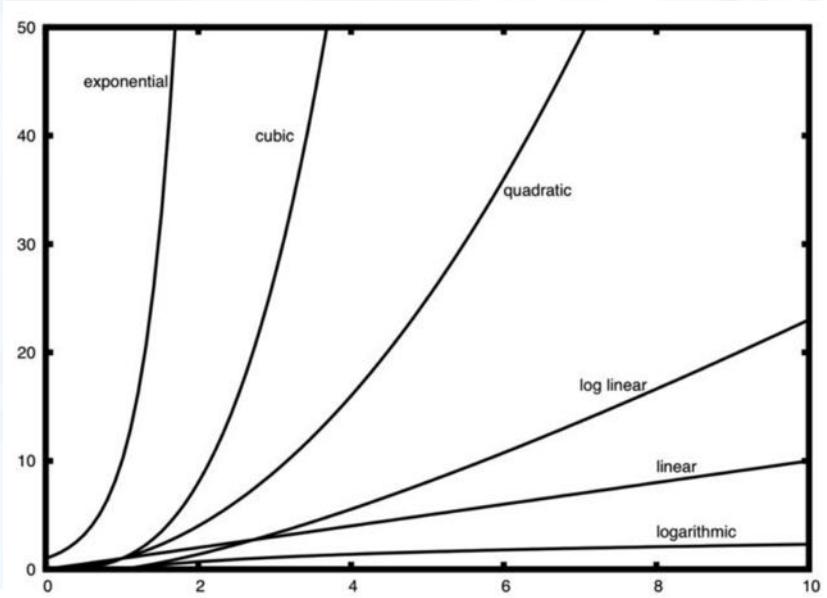
W02-算法分析

201 什么是算法分析

算法分析不是程序分析，更非代码分析，如何评判一个算法的性能？

202 大O表示法

时间复杂度的一种表示，当问题规模线性增长时，所需处理时间的增长趋势



203/4 “变位词”判断问题

通过“变位词”的四个算法，体验复杂度概念

205/6 Python数据类型的性能

Python内置的数据类型操作性能

W02-201-什么是算法分析

算法分析的概念

- 比较程序的“好坏”，有更多因素
  代码风格、可读性等等
- 我们主要感兴趣的是算法本身特性
- 算法分析主要就是从计算资源消耗的角度来评判和比较算法
  更高效利用计算资源，或者更少占用计算资源的算法，就是好算法

  从这个角度，前述两段程序实际上是基本相同的，它们都采用了一样的算法来解决累计求和问题

运行时间检测的分析

- 但关于运行时间的实际检测，有点问题
  关于编程语言和运行环境
- 同一个算法，采用不同的编程语言编写，放在不同的机器上运行，得到的运行时间会不一样，**有时候会大不一样**：
  比如把非迭代算法放在老旧机器上跑，甚至可能慢过新机器上的迭代算法
- 我们需要更好的方法来衡量算法运行时间
  这个指标与具体的机器、程序、运行时段都无关

W02-202-大O表示法

算法时间度量指标

- 一个算法所实施的操作数量或步骤数可作为独立于具体程序/机器的度量指标

  哪种操作跟算法的具体实现无关？

  需要一种通用的基本操作来作为运行步骤的计量单位

问题规模影响算法执行时间

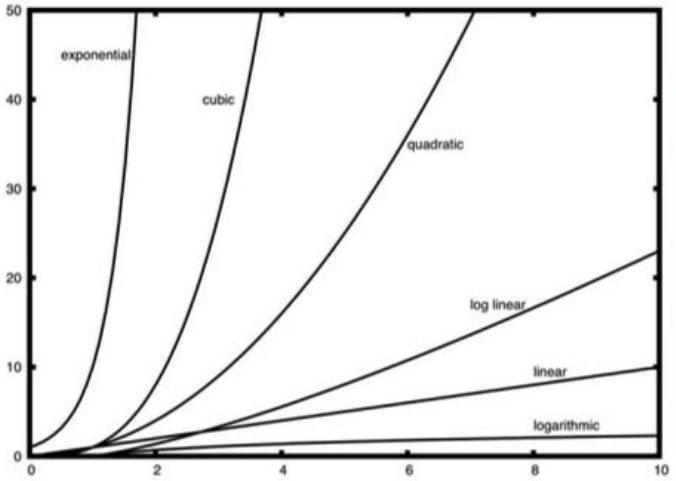
- 问题规模：影响算法执行时间的主要因素
- 在前n个整数累计求和的算法中，需要累计的**整数个数**合适作为**问题规模**的指标

  前100,000个整数求和对比前1,000个整数求和，算是同一问题的更大规模

- 算法分析的目标是要找出**问题规模**会怎么影响一个算法的**执行时间**

常见的大O数量级函数

- 通常当n较小时，难以确定其数量级
- 当n增长到较大时，容易看出其主要变化量级



$f(n)$	名称
1	常数
$Log(n)$	对数
n	线性
$n*Log(n)$	对数线性
n^2	平方
n^3	立方
2^n	指数

W02-202-大O表示法

世界上最早的算法：欧几里德算法

- 辗转相除法处理大数时非常高效
- 它需要的步骤不会超过较小数位数的5倍

加百利·拉梅(Gabriel Lame)于1844年证明了这个结论

并开创了计算复杂性理论。

- 请问这个算法的大O复杂度是多少？

W02-203/4 “变位词”判断问题

解法4：计数比较-算法分析

- 值得注意的是，本算法依赖于两个长度为26的计数器列表，来保存字符计数，这相比前3个算法需要更多的存储空间

  如果考虑由大字符集构成的词（如中文具有上万不同字符），还会需要更多存储空间。

- 牺牲存储空间来换取运行时间，或者相反，这种在**时间空间之间的取舍**和权衡，在选择问题解法的过程中经常会出现。

- 对于最好的O(n)算法，其实是有额外的空间代价

W02-205/6 Python数据类型的性能

用timeit来“验证”大O数量级

使用timeit模块对函数计时

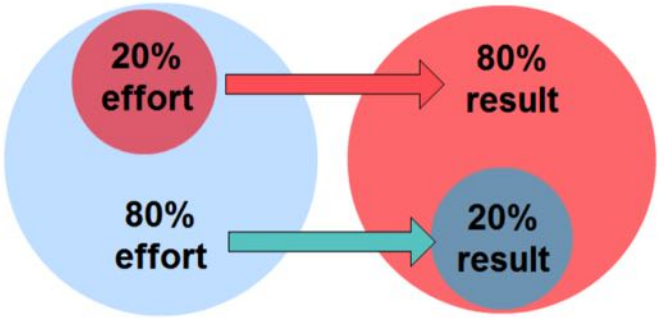
- 创建一个Timer对象，指定需要**反复运行**的语句和只需要**运行一次**的“安装语句”
- 然后调用这个对象的timeit方法，其中可以指定反复运行多少次

```
from timeit import Timer
t1= Timer("test1()", "from __main__ import test1")
print "concat %f seconds\n" % t1.timeit(number= 1000)

t2= Timer("test2()", "from __main__ import test2")
print "append %f seconds\n" % t2.timeit(number= 1000)

t3= Timer("test3()", "from __main__ import test3")
print "comprehension %f seconds\n" % t3.timeit(number= 1000)

t4= Timer("test4()", "from __main__ import test4")
print "list range %f seconds\n" % t4.timeit(number= 1000)
```

Python语言的实现中算法权衡

- 总的方案就是，让**最常用**的操作**性能最好**，牺牲不太常用的操作

80/20准则：80%的功能其使用率只有20%



【H1】关于计算的报告

数据结构与算法（Python）

【H2】（实验报告）算法计量与验证

- [H2.1]做计时实验，验证list的按索引取值确实是O(1)
- [H2.2]做计时实验，验证dict的get item和set item都是O(1)的
- [H2.3]做计时实验，比较list和dict的del操作符性能

  `del lst[i]/del dic[key]`
- [H2.4]做计时实验，通过对一些随机数列表排序，验证Python自带的list.sort的时间复杂度为O(n logn)

  随机数列表样例：`lst= [random.randrange(10**6) for n in range(10**4)]`
- 2020年3月4日（周三）18:00前提交

【H2】提交形式

- **实验报告：h2_学号_姓名_实验报告.pdf**
  例如：h2_1900000000_张三_实验报告.pdf
- **提交PDF文件一份**
- **实验报告包含代码、数据说明、运行结果说明和实验结果分析**
  要求每个计时实验均具备4个报告要素
  能用图表分析（可用Excel图表）说明最好
  自定格式，合成为单个电子文档，并转换为PDF格式上传到Canvas平台

关于【H2】作业：样例

“算法计量与验证”实验报告（H2）

徐玥 1800012616

一、【H2.1】验证 list 的按索引取值是 O(1)

1.实验源代码

实验 H2.1

```
1.  from timeit import Timer
2.  import random
3.
4.  #实验一：验证 list 的按索引取值是 O(1)
5.  def test1(lst,v):
6.      a=lst[v]
7.
8.  for i in range(10000,1000000,10000):
9.      lst=list(range(i))
10.     #随机得到索引序号
11.     v=random.randint(0,i-1)
12.     t=Timer("test1(lst,v)","from __main__ import test1,lst,v")
13.     print ("%f" % (t.timeit(number=1000)))
```

2.数据说明

List 的长度在 10,000 到 1,000,000 之间，公差为 10000，共 99 组数据。

3.运行实验说明

进行 99 次实验，每次生成不同长度的列表，随机索引序号，用 timeit 计时 1000 次重复实验的结果。

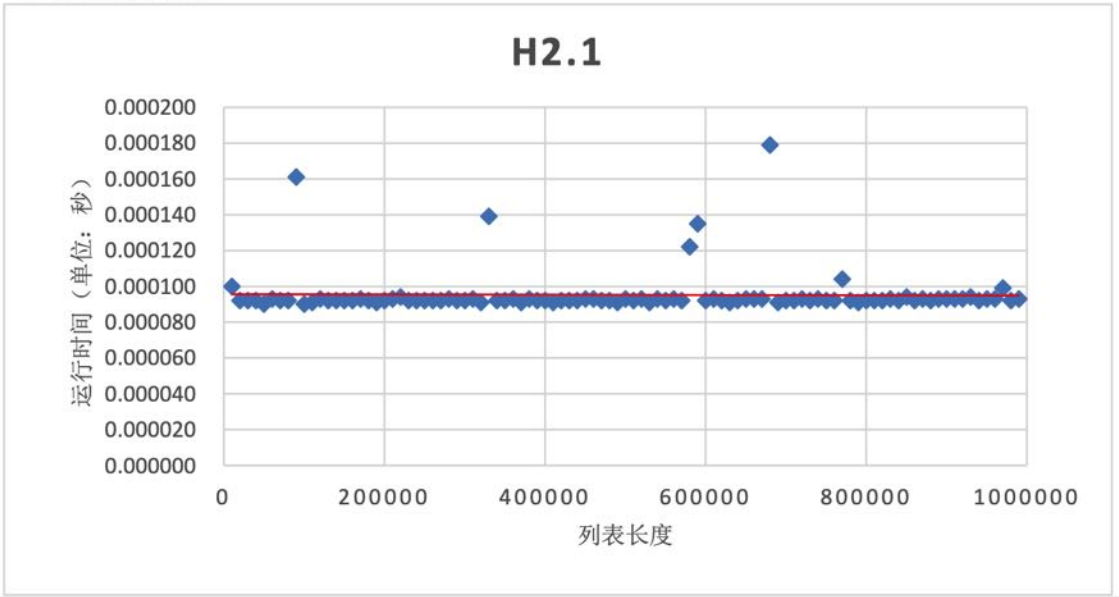
外层循环用于自动进行不同长度列表的实验，在计时函数之外循环生成列表、随机生成索引序号。为了尽量减少索引序号对时间结果的影响，索引序号在计时函数外随机生成。

4.实验结果及分析

a.理论分析与预测

根据 Python 官方的算法复杂度网站，list 的按索引取值是 O(1)。预测随着列表长度的增加，时间基本保持不变。

b.实验数据结果



c.结果分析

根据图像和回归曲线，除了少数波动数据外，算法基本符合 O(1)。

关于【H2】作业：样例

2019 年 3 月　　【H2】算法计量与验证（实验报告）　　Mar. 2019

【H2】算法计量与验证（实验报告）

作者：赵睿　　学号：1800012410　　指导老师：陈斌

（北京大学地球与空间科学学院 2018 级本科 5 班）

H2.1 做计时实验，验证 list 的按索引取值确实是 O(1)

H2.1.1 代码

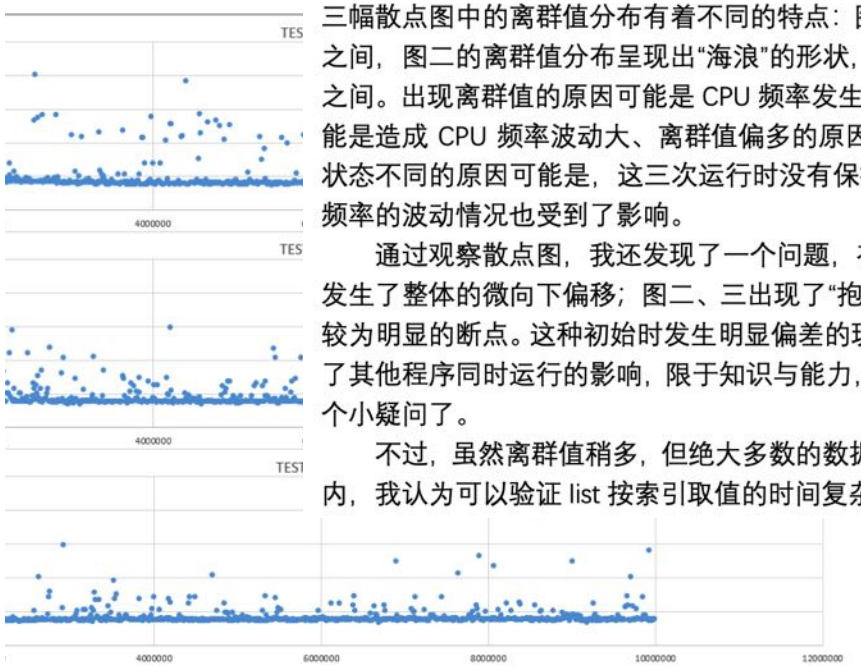
```
for i in range(100000,10000000,10000):
    alist=list(range(i))
    n=random.randint(0,i-1)
    t=Timer('a=alist[n]','from __main__ import alist,n')
    print('%f'%(t.timeit(number=1000)))
```

H2.1.2 数据说明

H2.1 一共做了三次实验，每次实验分别得到了 999 组数据，并制成了 3 张散点图。为保证数据的规模足够大，列表长度在 100000~10000000 的范围内取值，间隔为 10000。散点图及部分数据的截图如下：

	A	B	C	D	E	F	G	H
1	n值	TEST ONE		n值	TEST TWO		n值	TEST THREE
2	100000	0.00005		100000	0.000084		100000	0.000043
3	110000	0.000046		110000	0.000088		110000	0.00004
4	120000	0.000052		120000	0.000087		120000	0.000043
5	130000	0.000042		130000	0.000084		130000	0.00012
6	140000	0.000038		140000	0.000089		140000	0.000045
7	150000	0.000042		150000	0.000041		150000	0.000042
8	160000	0.000043		160000	0.000042		160000	0.000043

【H2】算法计量与



表（i 的取值范围为 100000~10000000，间隔为 10000）：

ge 为列表 alist 赋值；

标为 n 的值，计时；

执行 1000 次所用的时间。

H2.1.4 实验结果分析

可以明显地看出，三幅散点图中绝大多数的数据点都集中在 y=0.00004 这条直线附近，与列表长度 i 的大小无关。

但同时，散点图中出现了较多的离群值，有点离群值甚至达到了 0.0004 的 4 倍有余。三幅散点图中的离群值分布有着不同的特点：图一中的离群值主要分布在 0.00010~0.00015 之间，图二的离群值分布呈现出“海浪”的形状，图三的离群值主要分布在 0.00005~0.00010 之间。出现离群值的原因可能是 CPU 频率发生了波动，同时，笔记本 CPU 性能不太高也可能是造成 CPU 频率波动大、离群值偏多的原因。同时，我猜测三幅散点图的离群值的分布状态不同的原因可能是，这三次运行时没有保持后台其他程序的运行状况一致，导致 CPU 频率的波动情况也受到了影响。

通过观察散点图，我还发现了一个问题，在 i 取值较小（0~1000000）时：图一数据点发生了整体的微向下偏移；图二、三出现了“抱团飞出”的离群值，使原本的常数直线出现了较为明显的断点。这种初始时发生明显偏差的现象也出现在了一些其他实验的过程中。排除了其他程序同时运行的影响，限于知识与能力，我不知道如何解释这种现象，只能先留下一个小疑问了。

不过，虽然离群值稍多，但绝大多数的数据值都维持在了一个稳定的水平。在误差范围内，我认为可以验证 list 按索引取值的时间复杂度确实是 O(1)。

问题解答

- py的那一堆函数好难记的感觉……初学者莫得头绪，而且我的python要学到什么水平才够呢？

- 能不能简单介绍一下python的魔法方法，里面有比如__add __，__sub__，__delitem__等函数，感觉挺有意思的，对编程效率应该也有帮助。

问题解答

- 请问学习数据结构与算法时，Python和C会有什么差异呢？
- 是不是用C表达更直观，而Python调用方法更方便？
- Python当中的sort，reverse等方法，以及慕课W02中跳过的字典与列表的实质结构，是否会在之后的课堂中涉及？
- 还是说我们在自学python时就应该了解

问题解答

- 老师，能不能详细讲一下课程里Timer里面的那个

```
from __main__ import test
```

- 想听听您详细讲一下这个方法，我虽然用了pycharm的step over，但debug区域并没有对这个方法的详细反馈，所以还是不太懂

问题解答

- 想问一下老师：
- 1.直接赋值语句i=10这种是不是不包含计算只包含存储？
- 2.计算时间复杂度的时候为什么不考虑计算机计算乘除法的时间多于加减法的时间，这个有多大影响？比如说一开始那个用公式求和的程序，当n很大的时候计算n*(n-1)/2（一次减法，一次乘法一次除法）会不会比计算n次加法更久？

下课

- 别忘了周三晚18点的DDL