1. Computational Tools　01A1

a) 在01A1节中我们重温了欧几里得的“尺规计算机”及对应的“算法”，你还能想到哪些计算及对应的工具?

b) 其中对应的算法都是以什么形式描述的？与C++语言有何相似与不同之处?

2. Finiteness　01A2

01A2告诉我们**程序**未必就是**算法**，比如，我们时常遇到的“死循环”就不满足**有穷性**。

a) 回顾你的编程经历，还记得自己写过的这类程序吗?

b) 为尽可能避免这类问题，编程及调试过程中你有什么经验?

c) 能否设计一个算法，来**自动检测**任何程序中是否存在死循环，从而**彻底根除**?

3. Failstone　01A2

01A2节介绍的Hailstone序列，至今仍不能证明，对任何自然数n都是长度有限的。现考查如下函数:

$$failstone(p,q) = \begin{cases} 1 & (p = q) \\ 1 + failstone(p-q, p) & (p > q) \\ 1 + failstone(2p, q) & (p < q \; and \; q \; is \; odd) \\ 1 + failstone(p, q/2) & (p < q \; and \; q \; is \; even) \end{cases}$$

a) 试编写一个程序，对任何自然数p和q，计算出$failstone(p,q)$（假定字宽足够，不致溢出）;

b) 是否对任何p和q，都有$failstone(p,q) < \infty$？换言之，你的程序能否称作算法?

4. Geometric Distribution　01A2

a) 在01A2节中我们介绍了“几何分布”，你还见过哪些随机过程也符合这种分布?

b) 一般地，当其中单次尝试成功的概率$p \neq 1/2$时，期望的尝试次数应该是多少？试给出一般性公式并记住。

5. Algorithm Performance　01B1

本节我们以“最小三角形面积”问题为例说明了，即便对于规模接近甚至完全相同的问题实例，有些算法的性能也会差异极大。现在再考查“整数素因子分解”问题的蛮力算法：对任何待分解的整数n，依次用2、3、5、7、11、...做整除测试；每当发现一个素因子d，便输出d，同时将n替换为n/d；直到n等于1。

a) 当n大致为10^100时，该算法最坏情况下需要做多少次整除测试?

b) 最好情况呢?

6. Turing Machine　01B2

a) 对照01B2节讲义及对应的演示，了解图灵机的结构组成及工作过程;

b) 阅读increase()算法，掌握其原理;

c) 尝试编写一个decrease()算法，实现对二进制数字串（至少2位）的**递减**。

7. RAM　01B3

a) 对照01B3节讲义及对应的演示，了解RAM模型的结构组成及工作过程;

b) 阅读ceiling()算法，掌握其原理;

c) 实现multiply()算法，实现整数**乘法**，并估计其时间复杂度;

d) 尝试编写一个floor()算法，实现**向下**取整的除法，并估计其时间复杂度;

e) 图灵机与RAM之间有哪些区别?

f) 从编程语言的角度看，RAM与C++有哪些区别？

8. Small-o 01C1

01C1节介绍过“大$\mathcal{O}$”和“大Ω”记号，请查阅资料自学了解“小o”和“小ω”记号的含义及作用。

9. Fundamental Calculations 01C2

讲义上指出，我们将RAM中所有基本操作的时间成本都视作$\mathcal{O}(1)$，比如加法、减法。那么，乘法、除法呢？

10. Polynomial Logarithm 01C2

复杂度函数如果满足$T(n) = \mathcal{O}(f(n))$，是否必有$\log(T(n)) = \mathcal{O}(\log(f(n)))$？反过来呢？

11. Exponential Complexity 01C3

01C3节通过2-Subset这个问题来说明，**指数复杂度**的问题及算法普遍存在。你还能再举出哪些这类例子？

12. Master Theorem 01C3

01E2节介绍了“大师定理”的使用方法，你能证明这个定理吗？

13. Fractional Series 01D1

级数之于计算复杂度，无非是用来估计计算的成本，比如运行时间。具体来说，这类成本在RAM等计算模型中，都统一度量为基本操作的**次数**。你应该注意到，我们温习的级数中不少都是**分数**形式（比如调和级数），它们对于度量本身为**整数**的操作次数有什么作用呢？

14. Harmonic Series 01D1

温习此前在微积分课堂学习的**调和级数**，确认$\sum_{k=1}^{n} 1/k = \mathcal{O}(\log n)$。

15. Stirling Approximation 01D1

温习此前在微积分课堂学习的Stirling近似，确认$\log(n!) = \mathcal{O}(n \log n)$。

16. Iteration 01D2

《习题解析》1-32例举了多种典型的迭代、调用模式，试独立地估计它们的渐近复杂度，并与答案对照。

17. Back-Of-Envelope 01D3

32位、64位最大的无符号整数，表示为10进制是多少位？

18. Back-Of-Envelope 01D3

a) 查阅PA-book，了解并熟悉实验平台OJ的硬件配置及编译选项；

b) 在你开始着手做每一道PA题之前，参照所设定的时限估计出可行算法的时间复杂度**上界**。

19. Storage Cost Of Recursive Algorithms 01E1 + 01E2

01E1、01E2节分别基于**减治**、**分治**的策略，给出了两种递归的sum()算法，并分析了时间复杂度。

所谓**空间复杂度**，是指除去**输入数据**本身后，计算过程所需的空间量。

a) 上述算法的空间复杂度分别是多少？

b) 原本朴素的迭代算法呢？

c) 一般地，递归算法的空间复杂度与哪些因素有关？其中最主要的有哪些？

d) 同一算法的时间、空间复杂度之间，有什么联系？

20. Greatest Slice 01E3

在01E3节，我们介绍了效率由低到高的四种总和最大区间算法。

a） 如果不仅是算出最大总和，还需要具体地给出对应区间的（起始）位置，则应对算法作何调整？

b） 在最大总和同时出现于多个区间的情况下，应如何约定以消除这类歧义？

c） 针对你所做的约定，算法还需要作何调整？

d） 调整后算法的时间、空间复杂度，相对于与原算法是否有所增加？能否保持不变？

21. Memoization 01F1

翻出你以前编写过的一两个递归程序，试着借助**记忆化**技巧降低其复杂度。

22. Longest Common Subsequence 01F2

01F2节采用不同策略，分别给出了几种**最长公共子序列**的算法。

a） 如果不仅是算出LCS的长度，还需要具体地其与两个输入序列的对应关系，则应对算法作何调整？

b） 调整后算法的时间、空间复杂度，相对于与原算法是否有所增加？能否保持不变？

23. Code Reading

a） 下载本课程提供的示例代码包，在Visual Studio中通过dsacpp.sln打开整个解决方案；

b） 找出本章所涉及的项目，分别编译运行，观察运行结果，并细读对应的代码。

24. Demo

a） 下载算法演示包，并参照网络学堂公告中所介绍的方法，学会使用excel和applet类型的演示；

b） 找到本章所涉及的演示，对照讲义经常把玩，反复推敲。