1. Divide-And-Conquer 14A1

Quicksort是基于**分治**策略，Mergesort也是，从每个递归实例的计算流程来看，二者有何**区别**？

2. Recursion Base 14A1

本节给出了递归式Quicksort的基本框架，递归将会一直持续至“**ho - lo < 2**”，也就是序列缩减至仅含**单个元素**。与此前类似的问题同理，实践中总体效率**更高**的一种选择是，适当地**提前**终止递归，并改用某种**朴素**的方式继续处理。

a) 为**确定**何时可以提前终止递归，除了实验测量对比，还有什么办法？为此需要参考系统的哪些**指标**？

b) 就Quicksort而言，递归提前终止后，可以继续采用什么**朴素**的排序算法？

3. LUG 14A2

在本节介绍的LUG版partition()的主循环内，有两个前后并列的循环。紧接于这两个循环之后，都有一次可能的移动操作，它们具体执行与否，都取决于是否满足“lo < hi”。

a) 如果**省略**这两句判断，会有有何**不妥**？

b) 这两句判断毕竟显得有些**冗余**，你觉得可以如何**消除**它们？

4. Sedgewick's Trick 14A3

本节已证明，将quickSort()改为**迭代**版并采用“**小任务优先**”策略，可保证$\mathcal{O}(\log n)$的空间复杂度。

a) 试采用其它方法，证明这个结论；

b) 何时会达到**上界**$\mathcal{O}(\log n)$？

c) **最好**情况呢？**何时**最好？

5. Randomization 14A3

a) 通过**随机**选取轴点，为何“可以”**降低**Quicksort最坏情况出现的概率？

b) 在输入数据是**理想**随机的情况下，这种办法**的确**有效吗？

6. Median Of Three 14A3

本节指出，只要随机选取**三个样本**元素，并将**居中者**选作轴点，便可很好地降低坏情况出现的概率。

a) 若将划分之后子序列长度之比超过**9:1**视作**坏情况**，反之为**好情况**，则好情况、坏情况的**概率**各是多少？

b) 若采用**三者取中**策略呢？

c) 一般地，若将好情况、坏情况的**分界线**定在$0 < \lambda < 1$，采用**三者取中**策略后，**坏情况**的概率会是多大？

7. Cumulative Binomials 14A4

本节通过严格的分析，证明了快速排序算法的**递归深度**不超过$\mathcal{O}(\log n)$是**高概率**（w.h.p）事件。

其中最关键的一步，是对二项式系数前k项总和的**估计**：$\sum_{i=0}^{k}\binom{N}{i} \leq (eN/k)^k$。试证明这个不等式。

8. Expected Recursion Depth 14A4

然而，尽管讲义中的上述分析结论还算**可信**，但毕竟还非常地**不紧**。

试**改进**目前的估计（甚至彻底**改用**新的数学方法），给出**更紧**的上界。

9. Backward Analysis 14A5

本节通过两种方法，殊途同归地证明了快速排序所做的比较操作，期望次数为$1.386 \cdot n\log n$。其中，后一种

方法是将时间**颠倒**过来，考查输出（**已排序**）序列中任意一对元素，**倒推**出它们在**此前**的计算过程中相互比较的概率。讲义中指出，应用这种方法的前提是，算法的输出是**确定**的而且与具体采用何种算法**无关**。

a) 如果这一前提不能满足，这一分析方法为何**不再**可行？

b) 试举出一个这样的计算问题**实例**。

10. Expected # Moves 14A5

本节指出，Quicksort**移动**元素的次数，**期望**地不会超过**比较**操作次数的**一半**。试就此给出理由。

11. Stability 14A6

本节介绍的DUP版快速划分算法partition()，在输入中有**大量**元素**相等**时，可以将quickSort()的运行时间控制在$\mathcal{O}(n\log n)$，避免接近甚至达到$\mathcal{O}(n^2)$。为什么说，作为代价，算法的**稳定性**会进一步退化？（我们这里所谓的两个元素**"相等"**，是指它们被**判等器**判定为相等，但其内容未必**相同**。）

12. Duplication 14A6

为进一步提高quickSort()在存在**相等**元素时的处理效率，你还能想到什么办法？具体地，我们希望：

a) 相等元素**越多**，运行时间**越接近**于$\mathcal{O}(n)$；

b) **不含任何**相等元素时，算法效率不仅在**渐近**意义上与DUP版一样，而且**常系数**也几乎一样；

c) **略有几个**相等元素时，算法效率不仅在**渐近**意义上与DUP版一样，而且**常系数**也几乎一样。

13. Mode 14B1

若在众数的定义从“占一半**以上**”改为“占**至少**一半”，majCandidate()算法应该如何调整？

14. Median 14B2

试将本节以**递归**形式实现的median()算法，改写为**迭代**形式。

15. QuickSelect 14B3

QuickSelect算法的**主循环**，**期望**地会迭代多少步？

16. LinearSelect 14B4

在依照“中位数的中位数”**划分**出子集L、E、G之后，LinearSelect为什么必然**减除掉**至少**25%**的元素？

17. Cache 14C1

对于每一个增量h_k，Shellsort都是以**齐头并进**的方式来实现h_k列各自的**插入排序**。这类似于日常牌戏的发牌环节：尽管每位玩家都是在做自己的**插入排序**，但纵观桌上的所有玩家，则是**轮值**地向前推进。

a) 为什么说，相对于依次**逐列**完成插入排序的**串行**方式，上述方式效率**更高**？

b) 试通过**实测**对比，验证这一结论。

18. Shell Sequence 14C2

我们看到Shell序列的问题在于，除了$h_1 = 1$，增量序列中其余所有的各项都是**偶数**。

试证明：只要$h_1 = 1$以外的所有项有一个非平凡的**公因子**，Shellsort在**最坏**情况下都需要$\Omega(n^2)$时间。

19. Theorem K 14C2

Knuth指出，任何已**g-有序**的序列在经过**h-排序**之后，不仅自然地**h-有序**，而且依然保持**g-有序**。

试证明：该命题只要对任何**互素**的g和h成立，便对**任何**g和h均成立。

20. PS Sequence **14C3**

不难看出，PS序列中的每一项，都会被另一项**整除。**

那么，该序列为何没有如Shell序列那样，在**最坏**情况下需要$\Omega(n^2)$时间？

21. PS Sequence **14C3**

本节引述了前人的结论：采用PS序列的Shellsort，**平均**运行时间仅为$\mathcal{O}(n^{5/4})$。

试通过你自己的**实测**，**证实**或**推翻**这一结论。

22. Pratt Sequence **14C4**

从渐近复杂度的角度来看，Pratt序列$\mathcal{O}(n \log^2 n)$的时间性能，在我们介绍的所有增量序列中是**最好**的。

那么从**实际**计算效率的角度，这个结论是否成立？为什么？

23. Sedgewick Sequence **14C5**

Sedgewick已证明，采用他的增量序列，Shellsort的**期望**运行时间为$\mathcal{O}(n^{7/6})$。那么，问题的**规模**n需要**大到**何等程度，时间复杂度为$\mathcal{O}(n \log n)$的排序算法（比如Mergesort）方能在**实际**中显现出**优势**？