1. Associative Array 09A1

a) 试分别在Java、Python、Perl、Ruby等语言中运行本节所给的示例代码，并理解Associative Array的功能及用法。

b) 你还知道哪些编程语言支持Associative Array？

c) 如果这些语言同时还提供支持二分查找的常规有序向量，试就二者的查找**速度**做一实测对比。

2. Map vs. Dictionary 09A1

就ADT接口的形式而言，Map与Dictionary没有区别，但在是否允许不同词条（的关键码key）**相等**方面，二者却截然不同。在我们的讲义及多数文献中前者禁止，而后者则允许。有趣而又令我们困扰的是，在个别文献中，二者的定义有可能会**颠倒**过来。你是否读到过这类文献？

3. Call-By-Object 09A2

除了讲义所举的实例，日常生活中还有哪些场合采用了**Call-By-Object**的方式？

4. Perfect Hashing 09A3

试查阅相关资料，自学**完美散列**的构造方法。

5. Collision Probability 09A3

a) 试收集一组常用的**英文**词汇，针对典型的散列算法，统计其中彼此冲突的词汇。

b) 改成**中文**词汇，冲突的情况有何变化？

6. Birthday Paradox 09A3

a) 将N个随机的词条存入长度为M的散列表，至多出现x组冲突的概率p(x)是多少？试从数学上就此做一估算。

b) 你的估算是否与课上所建议的策略一致：**完全**杜绝冲突并不现实，我们必须乐于容忍**些许**的冲突？

7. Prime M 09B1

讲义指出，之所以需要将散列表长度M取作**素数**，乃是因为实际应用中的数据**远非**随机。

a) 试分别选择素数、合数表长，用来自**伪随机数发生器**的数据集做一测试，对冲突情况做一统计；

b) 改用来自**真实应用**的数据集，同样完成实测统计，并与上述结果做一对比。

8. Adversary 09B1

本节从**对抗攻击**的角度，指出了**除余法**的两个**弱点**，并进而针对性地改进为**MAD法**。

你还见过哪些算法，也可以采用这种方法来做分析？

9. Distributed Hash Table 09B1

尽管在一般情况下，散列函数应该使原本临近的词条在散列表中尽可能**散开**，但在分布式散列表（DHT）之类的特殊场合，我们却可能需要反其道而行之，需要邻近的词条保持一定的空间**临近性**（Localoty）。试查阅相关资料，了解背后的原因。

10. shuffle With rand() 09B2

尽管从理论上看，shuffle()算法通过对rand()的n次调用，可以得到n个元素的一个理想随机的序列，但实际上该算法可能产生的序列**远远少于**n!种。

a） 试对这一现象做出解释。

b） 你能想到什么弥补的办法？

11. hashCode() 09B3

课上指出，实际应用中的串对象**远非**随机，比如每个字符往往按各自的**频度**出现，因此如果只是对各字符做简单的累加，将会引发大量的冲突。

a） 试通过实测，验证上述判断。具体地，你可以从任一特定应用中收集一组真实的字符串，并通过简单累加得出它们的hashCode，然后对串长与hashCode的相关性做一**回归分析**。

b） 继续实验，改用多项式法来计算hashCode后，这个相关性会有什么变化？

12. Open Hashing By Separate Chaining 09C1

课上指出，采用基于**独立链**的开放散列策略，每一组同义词尽管**逻辑**上都会排成一个List，但它们之间的**物理**存储位置往往相距很远，即便逻辑上紧邻的同义词也是如此。试通过实验来验证这些结论。

13. Close Hashing By Linear Probing 09C2

课上指出，采用基于线性试探的封闭散列策略，在发生冲突时即便需要多次试探，也会因为试探的位置在**物理上**是连续的，使得系统的**缓存机制**得以充分发挥，由此所得收益之大，完全足以抵消多次试探的不足。试通过实验来验证这些结论。

14. Lazy Remove 09C3

采用封闭散列策略时，散列表中的每个词条除了记录本身的信息，还会同时成为若干条试探链上**不可或缺**的一环。懒惰删除法尽管巧妙地回避了**修复**试探链的复杂操作，但随着懒惰删除标志位的增加，毕竟后续的查找将会因此付出**更多**时间。这种额外的时间成本会持续增加，直到下一次重散列。

除了懒惰删除，你能否构想出其它办法，在摘除一个词条之后，相对**简便**地修复其所属的所有试探链？

15. Rehashing To 4N ~ 2M 09C4

封闭散列策略中，当装填因子超过预设的**上限**，便会做**重散列**。在示例代码中该上限取作**50%**，新的表长则取作**4*N**而不是**2*M**。如果取作**2*M**，会有什么不同？有何不妥？

16. Rehashing By put() 09C4

示例代码中实现的**重散列**算法，通过一趟遍历，借助**put()**接口将所有词条**逐一**地转移到新表。是否可以改用**memcpy()**之类的方式，**整体**地完成搬迁？

17. $\lceil \mathcal{M}/2 \rceil$ 09C5

讲义中证明了，在采用**平方试探**策略的散列表中，每条试探链的前$\lceil \mathcal{M}/2 \rceil$个位置都必然互异。

a） 试确认，如果再考查**更长**的区间，试探位置之间将会出现冲突；

b） 关于这些冲突，你能发现什么规律？

18. Two-Square Theorem 09C6

从字面来看，Fermat双平方定理只是给出了$\mathcal{M} = 4k + 1$型素数表长可能出现的**一种**冲突，即：

存在$0 < a < b \leq (M-1)/2$，使得$\mathcal{M} = a^2 + b^2$

然而实际上，我们还需留意**更多**可能的冲突，比如：

是否存在$0 < c < d \le (M-1)/2$，使得$\mathcal{M} \mid c^2 + d^2$ ？

a) 试证明：若素数$\mathcal{M} = 2m+1 = a^2 + b^2$，则对任何$0 < c \le m$，都存在$0 < d \le m$，使得$\mathcal{M} \mid c^2 + d^2$ ；

b) 试证明：$\mathcal{M} = 4k+3$型素数不可能是两个数的平方和。

19. Stability Of Bucketsort　　09D1

课上指出，桶排序算法可处理**相等**的元素，并给出**稳定**的输出。那么具体地，Distribution与Collection这两个阶段应分别选择何种**操作次序**和**方向**，以通过相互的配合来保证**稳定性**？

20. Minimum Gap　　09D2

本节借助**分桶**的技巧，给出了一个MaxGap的**线性**算法。对称地，我们也可以考查MinGap问题：

* 任给一组实数，找出它们之间的**最小**差 *

那么，MinGap是否也**存在**线性算法？试**给出**这样的一个算法，或者**证明**不可能有。

21. Radixsort　　09E1

a) 本节给出了基数排序的一种具体实现，试阅读对应的**示例代码**，理解其原理与技巧；

b) 某趟分拣之后若前缀、后缀没有变化，是否可以**随即终止**算法？

c) 什么情况下可以省略针对后续各位的分拣，**提前终止**算法？

d) 试确认，当前的版本在分拣过程中需反复调用remove()、insert()来**移动**节点，效率不高；

e) 试为List增加一个move(p,t)接口，**更高效地**将p直接移至t之前。

22. Integer Sorter　　09E2

本节给出的整数排序算法，首先要对每个整数做**进制转换**，为此需要花费$\mathcal{O}(d \cdot n)$时间。

a) 如果省略这一步，直接将每个数视作为**二进制**的（它们在计算机中本来就如此），然后调用09E1节示例代码中所实现的基数排序算法，总体的时间**成本**将有什么变化？

b) 在目前通用的系统架构中，新方法相对于原方法的**利弊**如何？

23. Countingsort　　09F

本节介绍的Countingsort算法，思路与Bucketsort本质上是相同的：

前者为每一组雷同元素预留的一个**区段**，等价于后者使用的一个**桶**

a) 首先通过一趟扫描**统计**出个区段的长短，为后续的计算提供了什么**便利**？

b) 试对照讲义中实例，编码实现该算法；

讲义的实例中，第二趟扫描是**自后向前**进行的，各区段也是**自后向前**填充的。实际上，这两个次序也可时改成**从前向后**的。

c) 为此，算法还需要做哪些调整？

d) 调整后，时间复杂度能否保持不变？

e) 试按新的思路编码实现该算法。

24. Geometric Distribution　　09G1

本节指出，按照Pugh的约定，Skiplist中各塔的高度服从**几何分布**，每座塔的**期望高度**均为$\mathcal{O}(1)$。

试动手推算，或查阅资料确认，塔高的**方差**是多少？

25. Horizontal Hops 09G2

本节的**重点**在于证明：`Skiplist::search()`的过程中，沿每一层**横向**列表都只会**期望**地跳转$\mathcal{O}(1)$步。为得出这一结论，**关键**在于这样一个事实：每一步水平跳转的**目的地**，都是某座塔的**顶部。**

a） 试证明这一事实；

b） 基于该事实进一步证明：沿任一横向列表跳转的步数，亦符合**几何分布。**

26. Failed Trials 09G2

`Skiplist::search()`沿着每一层**横向**列表的跳转，最终都会以一次**失败**的尝试（含对**哨兵+∞**的尝试）而结束。在该算法的时间复杂度中，为何“没有”体现这部分时间消耗？

27. Sentinels 09G3

在`Skiplist`的示例代码中，每层横向列表都配置了一对**哨兵**（-∞和+∞）。

得益于这些哨兵，算法的实现在哪些方面变得更加**简捷**了？