Anany Levitin 著《算法设计与分析基础(第3版)》(Introduction to the Design and Analysis of Algorithms 3rd Edition)部分习题、提示和答案(1~4章)

来源： https://bookzhou.com

以下是 A. Levitin 所著的《算法设计与分析基础》第三版的习题、提示和答案。对至少部分学生来说可能具有挑战性的问题用▷标记；对大多数学生来说可能具有难度的问题用▶标记。

第 1 章

习题 1.1

1． 研究一下 al-Khorezmi(也叫 al-Khwarizmi，即阿尔·花剌子模)这个人，“算法”(algorithm)一词就起源于他的名字。通过研究还会发现，“算法”和“代数”(algebra)这两个词的起源是相同的。

2． 如果告诉你，设立美国专利体系的基本目的是促进“有用的技艺”(useful arts)，那么你认为算法在这个国家能申请到专利吗？是否应该允许算法申请专利？

3． **a．**按照算法要求的精确性写出你从学校到家的驾车导航路线。
b．按照算法要求的精确性写出你最喜欢的一道菜的烹饪方法。

4． 设计一个计算$\lfloor\sqrt{n}\rfloor$的算法，n 是任意正整数。除了赋值和比较运算，该算法只能用到基本的四则运算。

5． 设计一个算法，针对已排好序的两个列表，找出所有共同的元素。例如，针对列表 2, 5, 5, 5 和 2, 2, 3, 5, 5, 7，应输出 2, 5, 5。如给定的两个列表的长度分别为 m 和 n，你设计的算法的最大比较次数是多少？

6． **a．**用欧几里得算法计算 gcd(31415, 14142)。
b．用欧几里得算法计算 gcd(31415, 14142)时，速度是检查 min{m, n}和 gcd(m, n)间连续整数的算法的多少倍？请估算一下。

7． ▷证明等式 gcd(m, n) = gcd(n, m mod n)对每一对正整数(m, n)都成立。

8． 对于第一个数小于第二个数的一对整数，欧几里得算法将会如何处理？该算法在

处理这种输入的过程中，上述情况最多会发生几次？

9． **a．**对于所有 $m \geqslant 1$，$n \leqslant 10$ 的输入，欧几里得算法最少要做几次除法？

b．对于所有 $m \geqslant 1$，$n \leqslant 10$ 的输入，欧几里得算法最多要做几次除法？

10． **a．**在欧几里得的书里，欧几里得算法用的不是整数除法，而是减法。请用伪代码描述这个版本的欧几里得算法。

b．▶欧几里得游戏(参见[Bog]) 一开始，板上写有两个不相等的正整数。两个玩家交替写数字。每一次，当前玩家都必须在板上写一个正整数，它是板上任意两个数字之差，而且这个数字必须是新的；换言之，不能与板上任何一个已有的数字重复。当玩家写不出新数字时，他就输了。请问，你是选择先行动还是后行动？

11．扩展欧几里得算法 该算法不仅能求出两个正整数 m 和 n 的最大公约数 d，还能求出两个整数 x 和 y(不一定为正)，使得 $mx + ny = d$。

a．查询扩展欧几里得算法的描述(参见[KnuI])，任选一种编程语言实现它。

b．修改你的程序，对丢番图(Diophantine)方程 $ax + by = c$ 求解，系数 a，b，c 为任意整数。

12．▷储物柜的门 走廊上有 n 个储物柜，从 1 到 n 依次编号。最初所有储物柜都关着。我们从储物柜前通过 n 次，每次都从 1 号柜开始。第 i 次通过时($i = 1, 2, \cdots, n$)切换 i 的整数倍号储物柜的状态：储物柜关着就打开；储物柜开着就关上。例如，假定有 6 个关着的储物柜，第一次通过后，每个储物柜都会开着；第二次通过则只切换偶数的储物柜(2 号、4 号、6 号)，所以第二次通过后，偶数的储物柜关闭，奇数的储物柜仍然打开；第三次通过，关闭 3 号储物柜(第一次通过时打开)，打开 6 号储物柜的门(第二次通过时关闭) ……以此类推。最后一次通过后，哪些储物柜开着，哪些门关着？有多少开着的储物柜？

习题 1.1 提示

1． 在网上查找会更快一些，但去图书馆也是有帮助的。

2． 其实，支持两方面的论据都存在。但有一个非常确实的原则和本题是相关的：科学事实和数学表达式是不能取得专利的。(你认为这是什么原因？)那么，这是不是排除了为所有算法授予专利呢？

3． 可以假设我们不是为机器，而是在为人写算法。但是，仍然要确保我们的表述不存在明显的歧义。克努特拿小甜饼食谱和算法做了一个有趣的比较([KnuI]，p. 6)。

4． 基于$\lfloor\sqrt{n}\rfloor$的定义，这个问题有一个相当简单直接的算法。

5． 试着设计一个最多做 mn 次比较的算法。

6． **a．**就遵循课本中给出的欧几里得算法。

b．比较一下这两个算法所做的除法次数。

7． 证明如果d能整除 m 和 n(也就是说，存在正整数 s 和 t，使 $m = sd$，$n = td$)，那么它也能整除n和 $r = m \bmod n$，反之亦然。利用公式 $m = qn + r$ ($0 \leqslant r < n$)以及这样一个事实：如果d能整除u和v，那么它也能整除 $u + v$ 和$u - v$。(为什么？)

8． 对于两个任意选定的整数$m < n$，执行算法的一次迭代。

9． a 的答案可以立即给出；b 则需要检查每一对$1 < m < n \leq 10$的数字，才能给出答案。

10．a． 利用方程

$$gcd\,(m,n) = gcd\,(m-n,\ n)，\text{其中}m \geq n > 0$$

b．关键是要计算出，从初始对 m,n 开始，能在板上写出的各不相同的整数总共有多少个。应利用该问题和 a 部分的关系。考虑一些较小的例子，尤其是那些$n = 1$和 $n = 2$ 的例子，也应该有所帮助。

11．对于某些系数来说，这个问题显然无解。

12．手动跟踪该算法(例如$n = 10$)并研究算法的输出有助于回答这两个问题。

习题 1.1 答案

1. Al-Khwarizmi(公元 9 世纪)是一位伟大的阿拉伯学者.公元 830 年，他写了一本有关代数的书《Hisab al-jabr wal-muqabalah》。史学家一直以来对此书的标题的适当翻译的意见不一，al-jabr 原为恢复平衡的意思，在这里指的是一项这种代数运算——移项完成后，等式两端又恢复平衡(al-jabr 也表示接骨师使断骨复原的意思)。wal-muqabalah 意指某种面对面而立的事实，在这里指的是集项这种代数运算。所以书名可译为《移项和集项的科学》，但通常习惯译作《积分和方程计算法》。这本书转成欧文，书名逐渐简化后，就被直接译成了《代数学》，代数学(Algebra)一词即由此书而来。书中阐述了解一次和二次方程的基本方法及二次方根的计算公式(即：$x2+10x=39$)，明确提出了代数、已知数、未知数、根、移项、集项、无理数等一系列概念，并载有例题 800 多道，提供了代数计算方法，把代数学发展成为一门与几何学相提并论的独立学科。此外，印度数码(1～9、0)也藉其著作传入西方，欧洲人称为阿拉伯数字。

   至于“算法”(Algorithm)一词，则来自 Al-Khwarizmi 姓氏的拉丁语翻译(即 Algoritmi，参见[KnuI，pp.1-2]和[Knu96，pp. 88-92，114])。
2. 这个法律问题还有待解决。目前的法律状况将数学算法和其他算法区分开来，前者不能申请专利，后者如果作为计算机程序实现，则可以申请专利(例如，[Cha00])。
3. n/a
4. 一个不依赖$\lfloor\sqrt{n}\rfloor$的近似值的简单算法是检查连续正整数的平方，直到结果首次大于 n。此时，答案就是这个正整数的上一个正整数，即它的直接前趋(immediate predecessor)。注意：使用牛顿法可以得到一个更快的算法来解决这个问题(参见 11.4 节和 12.4 节)。
5. 先创建一个公共元素列表，并初始化为空。从两个给定的已排好序的列表的第一个元素开始，重复以下操作，直到其中一个列表用完所有元素(没有下一个元素)：比较两个列表的当前元素；如果相等，将该元素添加到公共元素列表，然后两个列表都移动到下一个元素(如果有的话)；如果不相等，就移动到参与比较的两个元素中较小的那个之后。最大比较次数会在对没有共同元素的两个列表进行比较时

发生；例如，其中一个列表包含 m 个正奇数，另一个列表包含 n 个正偶数，则比较次数为 $m + n - 1$。

6. a. gcd(31415, 14142) = gcd(14142, 3131) = gcd(3131, 1618) = gcd(1618, 1513) = gcd(1513, 105) = gcd(1513, 105) = gcd(105, 43) = gcd(43, 19) = gcd(19, 5) = gcd(5, 4) = gcd(4, 1) = gcd(1, 0) = 1

   b. 为了回答这个问题，需比较这两种算法在给定输入上所做的除法次数。欧几里德算法的除法次数是 11(见 a 部分)。连续整数检查算法在其 14142 次迭代中的每次除法数量都是 1 和 2；因此，乘法总次数介于 1·14142 到 2·14142 之间。而欧几里德的算法速度介于 1·14142/11 ≈ 1300 到 2·14142/11 ≈ 2600 倍之间。

7. 首先证明如果 d 能整除两个整数 u 和 v，它也能整除 $u + v$ 和 $u - v$。根据除法定义，存在整数 s 和 t，使 $u = sd$，且 $v = td$。因此，

$$u \pm v = sd \pm td = (s \pm t)d$$

   即 d 能整除 $u + v$ 和 $u - v$。

   再证明如果 d 能整除 u，它也能被 u 的任意整数倍数 ku 整除。由于 d 能整除 u，$u = sd$，因此，

$$ku = k(sd) = (ks)d$$

   即 d 能整除 ku。

   现在来证明本题的断言。对于任何一对正整数 m 和 n，如果 d 能同时整除 m 和 n，它也能同时整除 n 和 $r = m \bmod n = m - qn$。类似地，如果 d 能同时整除 n 和 $r = m \bmod n = m - qn$，它也能同时整除 $m = r + qn$ 和 n。所以，两对整数(m, n)和(n, r)具有相同的非空公因数集合，其中包括集合中最大的元素，即 $\gcd(m, n) = \gcd(n, r)$。

8. 对于任何一对整数(m, n)，如果$0 \le m < n$，欧几里得算法直接在第一次迭代时交换两个数字：

$$\gcd(m, n) = \gcd(n, m)$$

   这是因为如果 $m < n$，那么 $m \bmod n = m$。这种交换之所以只发生一次，是因为既然 $\gcd(m, n) = \gcd(n, m \bmod n)$，那么在算法每次迭代之后，在新的整数对中，第一个数(n)必然大于第二个数$(m \bmod n)$。

9. a. 对于任何任何一对整数(m, n)，如果$m \ge n \ge 1$，而且 m 是 n 的倍数，欧几里得算法只做一次除法；这是两个正整数做的最少除法次数。

   b. 答案是做 5 次除法，这是在用欧几里得算法计算 gcd(5, 8)时发生的。检查算法对$1 < m < n \le 10$范围内的每一对输入所做的除法次数，用不了多少时间就能得到这个答案。

   注意，更具一般性的结论(参见[KnuII, p. 360])是对于任何一对输入(m, n)，其中$0 \le n < N$，欧几里得算法计算 $\gcd(m, n)$所需的最大除法次数是$\lfloor \log_{\emptyset}(3 - \emptyset)\, N \rfloor$；其中，$\emptyset = (1 + \sqrt{5})/2$。

10. a. 以下是一个非递归版本：

    算法 *Euclid2*(m, n)

    //基于减法，使用欧几里得算法计算 $\gcd(m, n)$

    //输入：两个不全为 0 的非负整数 m，n

//输出：m，n 的最大公约数

while $n \neq 0$ **do**

if m<n $swap(m, n)$

$m \leftarrow$ m − n

return m

b. 不难证明，可以写在板上的整数是由欧几里德算法的减法版本所产生的整数，而且只有这些整数。虽然它们出现在板上的顺序可能不同，但总数始终一样，即 m / gcd(m, n)；其中 m 是初始两个数字中最大的那个。注意，这个总数可能等于初始的两个整数。所以，总的行动次数是 m / gcd(m, n) – 2。结论是，如果 m / gcd(m, n)为奇数，你选择先行动；如果为偶数，则选择后行动。

11. n/a

12. 由于最初所有门都是关着的，所以在最后一次通过后，只有在发生了奇数次切换的储物柜才会开着。只有在 j 能整除 i 的时候，门 $i(1 \leq i \leq n)$才会在第 $j(1 \leq j \leq n)$次通过时发生切换。所以，门 i 的除数有多少个，就会被切换多少次。注意，如果 j 能整除 i，即 $i = jk$，则 k 也能整除 i。所以，除非 i 是一个完全平方数(例如，假定 i = 16，4 就没有另一个除数可以配对)，否则 i 的所有除数都能成对。例如，假定 i =12，那么 1 和 12、2 和 6 以及 3 和 4 都能成对。这意味着 i 只有在是一个完全平方数的时候(即 $i = j^2$)，它的除数数量才是一个奇数。所以，只有编号为完全平方数的门才会在最后一次通过后打开。不超过 n 的这种位置的总数是$\lfloor\sqrt{n}\rfloor$：这些位置的编号是 1~$\lfloor\sqrt{n}\rfloor$(含)的正整数的平方。

习题 1.2

1．古代谜题 农夫带着一只狼、一只羊和一棵白菜来到河边。他需要用船把它们带到河对岸。然而，这艘船只能容下农夫本人和另外一样东西(要么是狼，要么是羊，要么是白菜)。农夫不在场的话，狼会吃掉羊，羊也会吃掉白菜。请为农夫解决这个问题，或证明它无解(注意：农夫是素食主义者，但不喜欢白菜，所以他不能通过吃掉羊和白菜来解决该问题。更不用说狼是受保护动物)。

2．现代谜题 四个人要过一座摇摇欲坠的桥，他们都从同一侧开始。你有 17 分钟的时间让他们都过到另一边。现在是晚上，他们有一支手电。一次最多只能有两个人过桥。任何过桥的人，无论是一个人还是两个人，都必须带着手电。手电必须跟着人走，也就是说不能扔过去。甲过桥要用 1 分钟，乙要用 2 分钟，丙要用 5 分钟，丁要用 10 分钟。两个人一起走时，速度以较慢的那个人为准。(注意：网上传言，这是西雅图某知名软件公司的面试题。)

3． 已知三角形边长是正数 a，b，c，下面哪个公式可以作为计算三角形面积的算法？

a． $S = \sqrt{p(p-a)(p-b)(p-c)}$，其中 $p = (a + b + c) / 2$

b． $S = \frac{1}{2}bc \sin A$，其中 A 是 b 边和 c 边的夹角

c． $S = \frac{1}{2}ah_a$，h_a 是 a 边上的高

4． 用伪代码写一个算法来求方程 $ax^2 + bx + c = 0$ 的实根，a，b，c 是任意实系数。(假定平方根函数 sqrt(x)可用)。

5． 描述将十进制正整数转换为二进制的标准算法。

a． 用文字描述。

b． 用伪代码描述。

6． 写出你最喜欢用的 ATM 在提款时所用的算法(可依据喜好选用文字或伪代码描述)。

7． **a．** π能精确求解吗？

b． 该问题存在多少实例？

c． 在网上查找该问题的算法。

8． 列出一个你已知有多种算法的问题(计算最大公约数除外)。哪个算法更简单？哪个更有效率？

9． 考虑下面这个算法，它求的是数值数组中大小最接近的两个元素的差。

算法 MinDistance(A[0..n – 1])

//输入：数字数组 A[0..n – 1]

//输出：数组中两个大小相差最少的元素的差值

$dmin \leftarrow \infty$

for $i \leftarrow 0$ **to** $n - 1$ **do**

for $j \leftarrow 0$ **to** $n - 1$ **do**

if $i \neq j$ **and** $|A[i] - A[j]| < dmin$

$dmin \leftarrow |A[i] - A[j]|$

return $dmin$

尽可能改进该算法(如有必要，可完全替换该算法；否则请改进)。

10． 匈裔美籍数学家乔治·波利亚(George Polya，1887—1985)的《*How To Solve It*》(参见[Pol57])是关于问题求解最有影响的书籍之一。波利亚将他的观点总结为 4 点。请到网上查找这个总结，或最好直接从他的书中找，然后将其与 1.2 节概括的方法进行比较。它们之间有什么共同之处？有什么不同之处？

习题 1.2 提示

1． 农夫必须多次穿越这条河，而且第一次只有一种可行的做法。

2． 和第 1 题中的古代谜题不同，求解谜题的第一步并不是显而易见的。

3． 这里的主要问题是可能会概念不清。

4． 对于系数的所有可能值，该算法都应该正确处理，包括 0 在内。

5． 大家很可能在某一门编程课的导论部分学过该算法了。否则可以选择是自己设计这样的一个算法，还是去找找现成的。

6． 可以做一次实地考察来更新自己的记忆。

7． a 部分较难，虽然它的答案(18 世纪 60 年代由德国数学家约翰·朗伯特发现)是众所周知的。相比之下，b 部分就简单多了。

8． 大家很可能知道两种或更多对数字数组进行排序的算法。

9． 可以减少最内层循环的执行次数，让该循环运行更快(至少对于某些输入是如此)，或者更有效的方法是，重新设计一个更快的算法。

习题 1.2 答案

1. 用 P，w，g 和 c 分别代表农夫(peasant)、狼(wolf)、羊(goat)和白菜(cabbage)。以下是解决这个问题的两个主要序列之一：

	P g	g	Pwg	w	Pw c	w c	Pwgc
Pwgc	w c	Pw c	c	P gc	g	P g	

注意：本书后面还会回顾这个问题(参见 6.6 节)。

2. 假设 1、2、5、10 是代表过桥人的标签，*f* 代表手电的位置，括号中的数字是经过的总时间。下面这一连串行动可以解决这个问题：

	f,1,2	2	*f*,2,5,10	5,10	*f*,1,2,5,10
(0)	(2)	(3)	(13)	(15)	(17)
f,1,2,5,10	5,10	*f*,1,5,10	1	*f*,1,2	

3. a. 如假设知道如何计算一个任意正数的平方根，该公式可被视为一种算法。

   b. 这里的困难在于计算 $\sin A$。由于公式中没有提到如何计算，所以不应视为一种算法。即使像刚才对待平方根函数那样，假设知道如何计算一个给定角度的正弦(有几种算法能做到这一点，但当然只是近似的)，它仍然不应被视为算法，因为真正的问题在于，该公式还没有说明如何计算角度 A。

   c. 公式没有说如何计算 h_a。

4. **算法** *Quadratic*(*a*, *b*, *c*)

   //该算法求方程 $ax^2 + bx + c = 0$ 的实根

   //输入：实系数

   //输出：方程的实根，或者报告无实根

   if $a \neq 0$
   $D \leftarrow b * b - 4 * a * c$
   if $D > 0$
   $temp \leftarrow 2 * a$
   $x1 \leftarrow (-b + sqrt(D))/temp$
   $x2 \leftarrow (-b - sqrt(D))/temp$
   return $x1, x2$
   else if $D = 0$ **return** $-b/(2 * a)$
   else return '*no real roots*'
   else //$a = 0$
   if $b \neq 0$ **return** $-c/b$
   else //$a = b = 0$
   if $c = 0$ **return** '*all real numbers*'
   else return '*no real roots*'

   注意：11.4 节提供了该问题的一个更现实的算法。

5. a. 给定的数字 *n* 除以 2：余数 r_n(0 或 1)是二进制形式的下一个数位(从右向左)。用上一次除法的商来代替 *n*。重复上述操作，直到 *n* 变成 0。

   b. **算法** *Binary*(n)

   //该算法实现十进制正整数转换为二进制的标准方法

   //输入：一个十进制正整数

   //输出：*n* 的二进制数位列表：$b_k\ b_{k-1}...b_1\ b_0$

   $k \leftarrow 0$

   while $n \neq 0$

   $b_k \leftarrow n \bmod 2$

   $n \leftarrow \lfloor n/2 \rfloor$

   $k \leftarrow k + 1$

6. n/a

7. a. π作为无理数只能近似求解。

   b. 作为这个问题的一个实例，很自然地就是以一定精度计算π值，即包含 *n* 个正确的小数位。基于这一解释，该问题有无限多的实例。

   c. n/a

8. n/a

9. 以下改进版本只考虑同一对元素一次，避免在最内层循环中重新计算相同的表达式。

   算法 *MinDistance2*(*A*[0..*n* − 1])

   //输入：数字数组 *A*[0..*n* − 1]

   //输出：两个元素的最小差值

   $dmin \leftarrow \infty$

   for $i \leftarrow 0$ **to** $n - 2$ **do**

   for $j \leftarrow i + 1$ **to** $n - 1$ **do**

   $temp \leftarrow |A[i] - A[j]|$

   if $temp < dmin$

   $dmin \leftarrow temp$

   return $dmin$

10. 波利亚的四步解题法是：

    1. 理解问题
    2. 形成解题思路
    3. 执行
    4. 回顾/检查

习题 1.3

1． 考虑这样一个排序算法：对于待排序数组中的每一个元素，统计小于它的元素个数。然后，利用这个信息，将各个元素放到有序数组的相应位置。

算法 *ComparisonCountingSort*(*A*[0..*n* – 1])

//通过比较计数对数组进行排序

//输入：可排序数组 *A*[0..*n* – 1]

```
//输出：数组 S[0..n – 1]，A 的元素在其中按照非降序排列
for i ← 0 to n – 1 do
    Count[i] ← 0
for i ← 0 to n – 2 do
    for j ← i + 1 to n – 1 do
        if A[i] < A[j]
            Count[j] ← Count[j] + 1
        else Count[i] ← Count[i] + 1
for i ← 0 to n – 1 do
    S[Count[i]] ← A[i]
return S
```

a．应用该算法对列表“60, 35, 81, 98, 14, 47”进行排序。

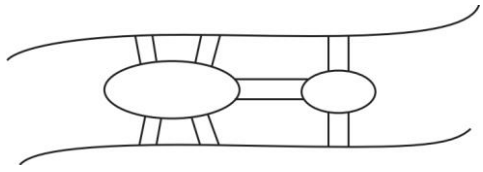
b．该算法是稳定(stable)的吗？

c．该算法是原地(in-place)的吗？

2． 写出你所知道的有关查找问题的算法名称。简要描述每个算法。如果不知道这样的算法，正好借此机会自己设计一个。

3． 为字符串匹配问题设计一个简单的算法。

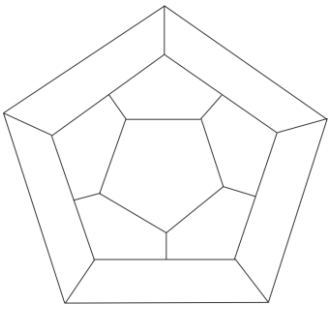
4． **柯尼斯堡七桥问题** 大家公认，图论诞生于柯尼斯堡七桥问题(Seven Bridges of Königsberg)。瑞士出生的伟大数学家莱昂哈德·欧拉 (Leonhard Euler, 1707—1783)解决了这个问题。 问题是在所有桥都只能走一遍的前提下，如何才能在一次步行中把所有桥都走遍？以下是河流及其两个岛和七座桥的草图。



a．将这个问题描述成一个图问题。

b．该问题有解吗？如果认为有解，请画出步行路线图；如果认为无解，请解释你的理由，并说明为了使这种步行路线成为可能，最少需要增加几座新桥。

5． **环游世界游戏**(Icosian) 在欧拉的发现(参见第 4 题)一个世纪以后，著名的爱尔兰数学家威廉·哈密顿(William Hamilton，1805—1865)创造了另一个著名的问题，它被称为环游世界游戏。这个游戏是在一块圆形的木板上玩的，板上刻的图如下所示：



寻找**哈密顿回路**(Hamiltonian circuit)——在回到起点之前，这一路径能访问该图的所有顶点，并且只访问一次。

6． 考虑以下问题：假设身处华盛顿特区和伦敦那样发达的地铁系统中，为地铁乘客设计一个算法，找出从一个指定车站到另一个车站的最优路径。

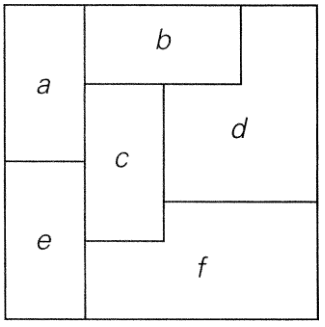
a .该问题的定义有些模糊，但现实生活中的问题往往就是这样的。对于这个问题来说，有什么合理的标准可以用来定义“最优”路径？

b . 如何用图对该问题进行建模？

7． a . 用组合对象的术语重新描述旅行商问题。

b . 用组合对象的术语重新描述图着色问题。

8． 考虑以下地图。



a . 解释如何利用图着色问题对该地图着色，使相邻区域颜色不同。

b . 利用 a 的答案，用最少数量的颜色对该地图着色。

9． 为以下问题设计一个算法：给定平面直角坐标系(笛卡尔坐标系)上的一组 n 个点，确定它们是否都位于同一圆周上。

10． 写程序输入两条线段 P_1Q_1 和 P_2Q_2 的端点的(x, y)坐标，判断两条线段是否有交点。

习题 1.3 提示

1． 对于给定的输入跟踪该算法。利用本节给出的稳定和在位的定义。

2． 如果一个查找算法都想不起来，你应该设计一个简单的查找算法。(要抵制住从后文中找一个类似算法的诱惑。)

3． 本书后文会介绍这个算法，但自己设计一个也不难。

4． 如果在前面的课程中没有遇到过这个问题，可以在网上或者在离散数学的教材中寻找答案。实际上，这个答案出奇地简单。

5． 对于一个任意的图，我们没有对这个问题求解的通用算法。但这个特定的图是存在哈密顿回路的，而且不难找到。(需要大家找出其中的一条。)

6． **a．** 假设自己处于乘客的位置，问问自己会采用哪种标准来评判“最优”路径。然后考虑一下可能有不同需求的其他人。

b． 用图来表示这个问题是非常直接明了的。尽管如此，还是要考虑一下那些可以换乘的车站。

7． **a．** 在旅行商问题中的旅程是怎样的？

b． 可以很自然地把颜色相同的顶点作为同一个子集中的元素。

8． 考虑一个图，它的顶点代表地图的区域。你必须自己决定边代表什么。

9． 假设所讨论的圆周存在，先求出它的圆心。还有，不要忘了对$n \leq 2$的情况专门给出答案。

10． 注意不要遗漏这个问题的任何特例。

习题 1.3 答案

1. a. 通过比较计数对 60, 35, 81, 98, 14, 47 进行排序的过程是：

数组 $A[0..5]$		60	35	81	98	14	47
初始	$Count[]$	0	0	0	0	0	0
第$i=0$次之后	$Count[]$	3	0	1	1	0	0
第$i=1$次之后	$Count[]$		1	2	2	0	1
第$i=2$次之后	$Count[]$			4	3	0	1
第$i=3$次之后	$Count[]$				5	0	1
第$i=4$次之后	$Count[]$					0	2
最终状态	$Count[]$	3	1	4	5	0	2
数组$S[0..5]$		14	35	47	60	81	98

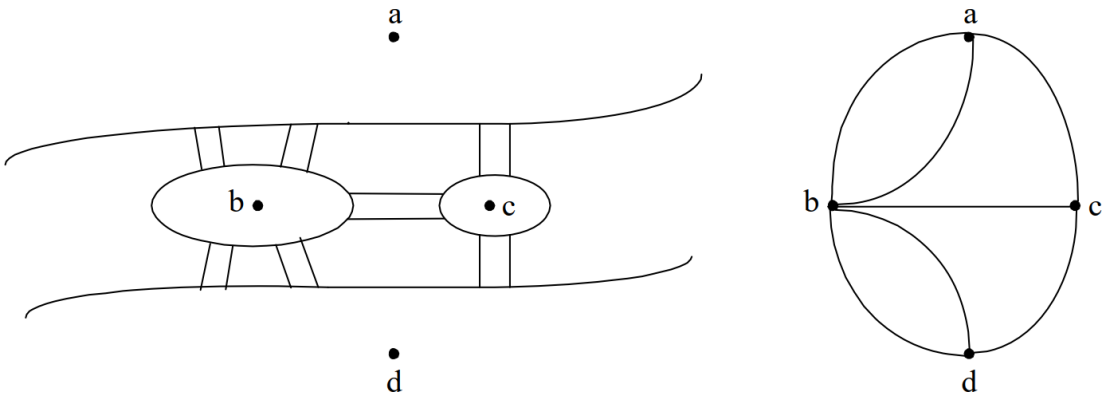
   b. 该算法是不稳定的。例如，作为反例，考虑到它应用于 1'，1"的结果

   c. 该算法不是原地的，因其使用了大小为 *n* 的两个额外的数组：*Count* 和 *S*。

2. 有许多答案，但大多数学生应熟悉顺序查找、二分查找、二叉树查找，可能还包括编程入门课程中的散列查找。

3. 将模式与文本的开头对齐。从左向右比较模式和文本的相应字符，直到所有模式字符都被匹配(此时可以停止——搜索成功)，或者算法用完文本的所有字符(此时可以停止——搜索不成功)，或者遇到一对不匹配的字符。在后一种情况下，将模式右移一个位置，继续进行比较。

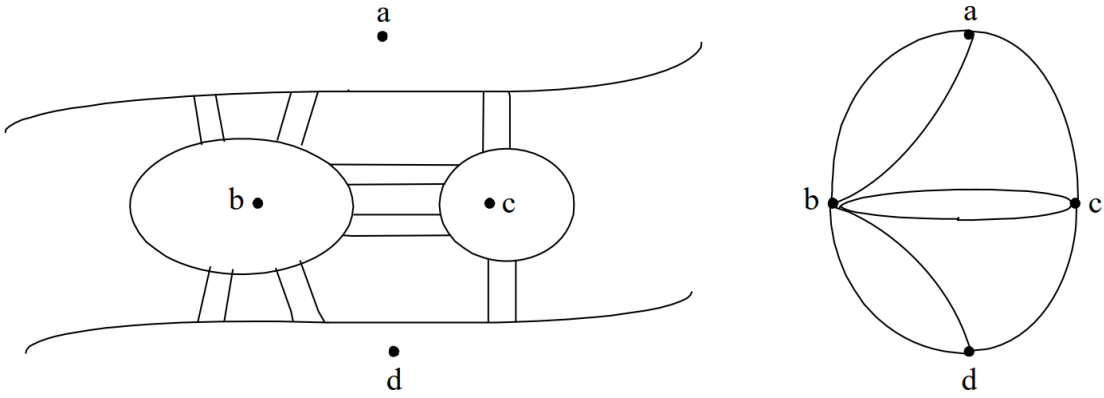
4. a. 如果用顶点表示河岸和两个岛中的每一个，用边表示桥，将得到下图：



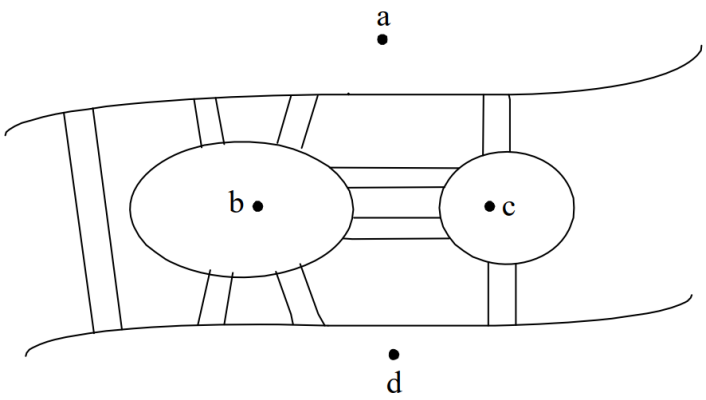
这实际是一个多重图(multigraph)，而不是一个图(graph)，因为同一对顶点之间有不止一条边。但这对目前的问题来说并不重要。问题在于，这个多重图中是否存在一条路径(即相邻顶点的序列)，该路径正好穿越所有边一次，并返回一个起始顶点；这样的路径称为**欧拉路径**(Eulerian path)。

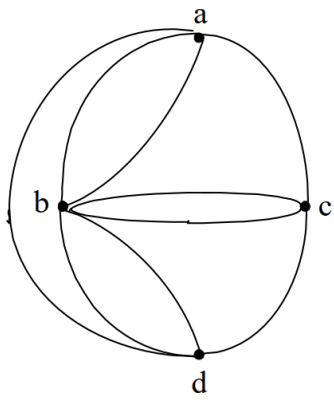
b. 欧拉证明，当且仅当所有顶点的度数为偶数时，在一个连接的(多重)图中，才存在欧拉回路。其中，顶点的“度数”被定义为它作为端点的边的数量。另外，在一个连接的(多重)图中，当且仅当存在两个奇数度的顶点，才存在一条欧拉路径；这样的路径必须从这两个顶点中的一个开始，在另一个结束。所以，对于谜题中的多重图，既不存在欧拉回路，也不存在欧拉路径，因其四个顶点的度数都是奇数。要获得一个欧拉路径，多重图中的两个顶点必须变成偶数。这可以通过增加一座新的桥梁来实现，这座桥连接与现有桥相同的地点。例如，在两个岛之间的一座新桥将使以下步行路线成为可能：

$$a-b-c-a-b-d-c-b-d$$



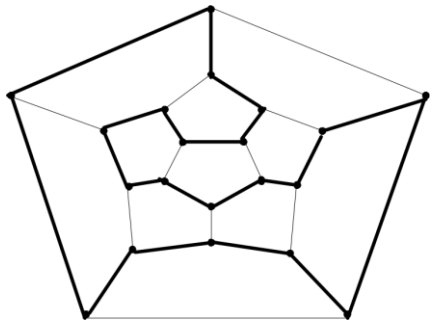
如果想步行回起点，则相应的多重图中的所有顶点必须是偶数度。由于一个新桥/边改变了两个顶点的奇偶性，所以至少需要两个新的桥/边。例如，下面展示了这样的一个“增强”：





这样一来，至少可以这样走：$a-b-c-a-b-d-c-b-d-a$(还有其他走法)。

5. 下图标出了一个哈密顿回路：



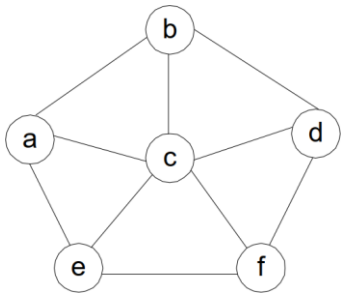
6. a. 至少存在三个“合理”标准：最快，途经地铁站最少，以及最少换乘次数。注意，第一个标准需要了解地铁一站路需要多少时间和换乘所需时间，其他两个标准不需要这些信息。

   b. 一个自然的方法是模拟地铁线路图，用顶点表示车站，如相应的车站之间有一条火车线路，则两个顶点用一条边连接。如需考虑换乘时间(例如，某个站可能在不止一条线路上)，这个站应该用多个顶点表示。

7. a. 找到 *n* 个给定城市的一个排列，其中连续城市之间的距离之和加上其最后一个和第一个城市之间的距离尽可能小。

   b. 将图的所有顶点划分为最小数量的不相交子集，使同一子集的顶点之间不存在连接的边。

8. a. 创建一个图，其顶点代表地图的区域，只有在对应的区域存在公共边界的前提下(所以不能着相同的颜色)，才用边来连接两个顶点。下面是和该地图对应的图：



解这个图的图着色问题，就能用最少的颜色完成地图着色。

b. 在不失一般性的前提下，可将颜色 1 和 2 分别分配给顶点 c 和 a。这迫使我们对其余顶点进行以下颜色分配：3 分配给 b，2 分配给 d，3 分配给 f，4 分配给 e。因此，该地图所需的最小颜色数是 4。

注意：众所周知，*任何*地图都能用四种或更少颜色来着色。这个问题称为“四色问题”，一个多世纪以来一直没有得到解决。直到 1976 年，美国数学家凯尼斯 •阿佩尔和沃夫冈 • 哈肯通过数学论证和大量借助计算机，最终得到了一个完全的证明。

9. 如果 $n=2$，答案肯定为“是”；所以，我们可以假定 $n \geq 3$。从给定的集合中选择三个点 P_1，P_2 和 P_3。写端点为 P_1 和 P_2 的线段的垂直平分线 $l1$ 的方程，这是与 P_1 和 P_2 等距的点的位置。写端点为 P_2 和 P_3 的线段的垂直平分线 l_2 的方程，这是与 P_2 和 P_3 等距的点的位置。通过求解两个未知数 x 和 y 的两个方程组，找出线段 l_1 和 l_2 的交点 P 的坐标(x, y)。(如果该方程组没有解，则返回“否”：这样的圆周不存在)。计算从 P 到每个点 $P_i, i=3, 4..., n$ 的距离(或更好，计算平方距离！)，并检查它们是否都相同：如果是，返回“是”，否则返回“否”。

10. n/a

习题 1.4

1． 描述应如何实现以下数组操作，使得所花的时间不依赖于数组长度 n。

a．删除数组的第 i 个元素$(1 \leqslant i \leqslant n)$。

b．删除有序数组的第 i 个元素(当然，数组余下的部分必须保持有序)。

2． 假定要解决包含 n 个数字的列表的查找问题，应如何利用该列表已排好序这一事实？针对以下情况分别解答：

a．列表表示成数组。

b．列表表示成链表。

3． **a．**给出一个空栈在依次进行以下每个操作之后的状态(push 是入栈，pop 是出栈)：

push(a)，push(b)，pop，push(c)，push(d)，pop

b．给出一个空队列在依次进行以下每个操作之后的状态(enqueue 是入队，dequeue 是出队)：

enqueue(a)，enqueue(b)，dequeue，enqueue(c)，enqueue(d)，dequeue

4．a．*A* 是一个无向图的邻接矩阵。请说明当邻接矩阵具有何种特征时意味着图具有以下特性：

i．图是完全图。

ii．图具有自环(loop)，即某些顶点具有指向自己的边。

iii．图具有一个孤立顶点，即没有边和该顶点相连。

b．换成用邻接链表来表示图，回答同样的问题。

5．详细描述将自由树转换为以该树给定顶点为根的有根树的一个算法。

6．证明以下关于 *n* 个顶点的一个二叉树的高度的不等式：

$$\lfloor \log_2 n \rfloor \le h \le n - 1$$

7．请指出如何用以下数据结构来实现 ADT 优先队列：

a．(无序)数组。

b．有序数组。

c．二叉查找树。

8．如何实现一个相对较小，长度为 *n*、所有元素都唯一的字典 (例如美国 50 个州的州名)。详细说明如何实现每种字典操作。

9．指出对于以下每种应用来说最适合的数据结构：

a．按已知的优先顺序接听电话。

b．按订单接收顺序向客户发货。

c．实现一个能计算简单算术表达式的计算器。

10．变位词 设计一个算法来检查两个单词是否为变位词(anagrams)。换言之，能否通过改变一个单词的字母顺序(不区分大小写)来得到另一个单词。例如，单词 tea 和 eat 是变位词，Unclear 和 Nuclear 也是。

习题 1.4 提示

1．a．充分利用数组是无序的这样一个事实。

b．我们实现 1.1 节的一个算法时使用过这个技巧。

2．a．大家肯定应该听说过，对于一个有序数组来说，有一个特别高效的算法。

b．加快不成功查找的速度。

3．a．push(*x*)将*x*放在栈顶，*pop*则从栈顶将元素删除。

b．enqueue(*x*)将*x*加入队尾，dequeue 则从队头将元素删除。

4．对于所讨论的图的特性和所涉及的数据结构，使用它们的定义就好。

5．有两种著名的算法可以对这个问题求解。第一种使用栈，第二种使用队列。虽然本书后面会讨论这两种算法，但不要错过自己发现它们的机会。

6．不等式$h \le n - 1$可立即从高度的定义得出。下界不等式则来自于$2\text{^}(h + 1) - 1 \ge n$；为了证明这个不等式，可考虑一棵高度为$h$的二叉树可能拥有的最大节点数。

7． 应指出优先队列的三种操作中，每一种操作是如何实现的。

8． 由于涉及插入和删除，所以使用由字典元素构成的一个数组(有序或无序)可能不是最佳实现。

9． 为了回答其中的某个问题，需要先了解后缀表达式的概念。(如果还不熟悉，请在网上查找相关信息。)

10． 该问题有若干种算法。记住，在单词中，同一个字母可能多次出现。

习题 1.4 答案

1. a. 用最后一个元素替换第 i 个元素，数组大小减 1。

   b. 用一个不可能是数组元素值的特殊符号替换第 i 个元素(例如，正数数组可使用 0)，以标记第 i 个位置为空(这种方法有时称为“懒惰删除”)。

2. a. 使用折半查找(或称二分查找；如果还不熟悉该算法，请参见 4.4 节)。

   b. 在有序链表中查找时，遇到大于或等于搜索键的元素就停止。

3. a.

$push(a)$	$push(b)$	b	pop	$push(c)$	c	$push(d)$	d	c	pop	c
a		a	a		a			a		a

   b.

$enqueue(a)$	$enqueue(b)$	$dequeue$	$enqueue(c)$	$enqueue(d)$	$dequeue$
a	ab	b	bc	bcd	cd

4. a. 用邻接矩阵来表示：

   i. 在一个图的邻接矩阵中，当且仅当除主对角线上的元素外的所有元素都等于 1 时，该图是完全图。换言之，对每个$1 \le i,j \le n, i \ne j$来说，$A[i,j] = 1$。

   ii. 在一个图的邻接矩阵中，当且仅当其主对角线上有一个等于 1 的元素时，该图有一个自环。换言之，对于某个$1 \le i \le n$来说，$A[i,i] = 1$

   iii. 在一个(无向、无自环)图的邻接矩阵中，当且仅当有一个全零的行时，它才有一个孤立的顶点。

   b. 用邻接链表来表示：

   i. 当且仅当每个链表都包含图的其他所有顶点，该图是完全图。

   ii. 当且仅一个邻接链表包含定义列表的顶点，该图具有自环。

   iii. 当且仅当一个邻接链表是空的，(无向，无环)图具有一个孤立的顶点。

5. 第一个算法的工作方式如下。标记一个顶点作为树的根，使其成为要构建的树的根，初始化只包含该顶点的一个栈。重复以下操作，直到栈变空：如果有一个未标记的顶点与栈顶的顶点相邻，则标记前者，在树中将其子女节点连接到当前栈顶所对应的顶点，然后将其入栈；否则，对栈顶的顶点做一次出栈操作

   第二种算法的工作方式如下。标记一个顶点作为树的根，使其成为要构建的树的根，初始化只包含该顶点的一个队列。重复以下操作，直到队列变空：如果有未标记的顶点与队头的顶点相邻，则标记所有这些顶点，在树中把它们作为子女节点连接到当前队头所对应的顶点，然后将它们入队；然后对队列做一次出队操作(移除队头顶点)。

6. 由于高度 h 定义为从树根到叶子的最长简单路径的长度，因此这样的路径将包含不超过 n 个顶点，即不超过树中的顶点总数。因此，$h \leq n-1$。

高度为 h 且顶点数最多的二叉树称为完全树，在其总共 $h+1$ 层中，每一层都填充了尽可能多的顶点数。这种树的总顶点数是 $\sum_{l=0}^{h} 2^l = 2^{h+1}-1$。

因此，对于任何具有 n 个顶点和高度为 h 的二叉树：

$$2^{h+1} - 1 \geq n$$

即：

$$2^{h+1} \geq n+1$$

或者，取两边以 2 为底的对数，并考虑到 $h+1$ 是一个整数：

$$h+1 \geq \lceil \log_2 (n+1) \rceil$$

由于 $\lceil \log_2 (n+1) \rceil = \lfloor \log_2 n \rfloor + 1$(参见附录 A)，最后得到：

$$h+1 \geq \lfloor \log_2 n \rfloor + 1 \text{ 或 } h \geq \lfloor \log_2 n \rfloor$$

7. a. 插入可通过在数组最后一个元素之后添加新项来实现。找到最大的元素需要对数组进行一次标准扫描。删除最大元素 $A[i]$可通过将其与最后一个元素交换并使数组大小减 1 来实现。

b. 假设代表优先队列的数组 $A[0...n-1]$按升序排序。插入一个值为 v 的新项可通过扫描有序数组来完成；换言之，从左向右，直到某个元素 $A[j] \geqslant v$ 或者到达数组末尾。(4.4 节讨论的折半搜索是一种更快的寻找新元素插入位置的算法)。在前一种情况下，新元素被插入 $A[j]$之前，先将 $A[n-1],...,A[j]$右移一个位置即可。在后一种情况下，新元素被简单地附加到数组最后一个元素之后。为了找到最大元素，返回有序数组最后一个元素的值即可。为了删除最大元素，数组大小减 1 即可。

c. 插入新元素使用在二叉查找树中插入新元素的标准算法来完成。换言之，根据新键是小于还是大于根键，将其插入左子树或右子树(以递归的方式)。找到最大元素需要找到二叉树最右侧的元素。换言之，从根开始，一直沿右子树的链条，直至到达一个没有右子树的顶点。该顶点的键就是最大元素。删除它可通过使其父的右指针指向被删除顶点的左子来实现；如果最右侧的顶点没有左子，该指针变成“空”。最后，如果最右侧的顶点没有父(也就是说，它恰好是树的根)，其左子成为新的根；如果没有左子，整个树就会变成空。

8. 使用一个位向量(bit vector)，即包含 n 个二进制位的一个数组。其中，如果基础集合的第 i 个元素在字典中，则第 i 位为 1；否则为 0。查找、插入和删除操作需要检查或更改该数组中的一个二进制位。

9. 使用：(a)优先队列；(b)队列；(c)栈(以及逆波兰记法——一种不用括号表示算术表达式的巧妙方法，通常在数据结构课程中学习)。

10. 最直截了当的方案是在第二个单词中查找第一个单词的每一个连续字母。如查找成功，就删除第二个单词中第一个出现的该字母；否则停止。

另一个方案是对每个单词的字母进行排序，然后执行一次简单的并行扫描来比较它们。

还可以生成和比较给定单词的“字母向量”(letter vectors)：

$V_w[i] =$ 字母表第 i 个字母在单词 w 中出现的次数。这样一个向量可通过将其所

有分量初始化为 0，然后扫描单词并在向量中递增对应的字母计数来生成。

第 2 章

习题 2.1

1． 对于下列每种算法，指出(i)它的合理的输入规模度量标准；(ii)它的基本操作；(iii)对于相同规模的输入，其基本操作的次数是否会有所不同。

a． 计算n个数之和。

b． 计算$n!$。

c． 找出包含n个数字的列表中的最大元素。

d． 欧几里得算法。

e． 埃拉托色尼筛选法。

f． 两个n位十进制整数相乘的笔算算法。

2． **a．**考虑一个基于定义的、对两个$n \times n$矩阵相加的算法。它的基本操作是什么？以矩阵阶n的函数来表示的基本操作需要执行多少次？以输入矩阵的元素总量的函数来表示的基本操作又需要执行多少次？

b. 对一个基于定义的矩阵乘法算法回答同样的问题。

3． 考虑顺序查找算法的一个变化形式，它返回给定查找键在列表中的出现次数。它的效率和经典顺序查找算法的效率有区别吗？

 **4．** **a. 选择手套** 抽屉里有 22 只手套：5 双红手套、4 双黄手套和 2 双绿手套。你摸黑挑选手套，而且只能在选好后才能检查它们的颜色。在最优情况下，最少选几只手套就能找到一双匹配的手套？最差情况呢？

 **b. 丢失的袜子** 假设在洗了 5 双各不相同的袜子以后，你发现有两只袜子找不到了。当然，你希望留下数量最多的完整袜子。所以，在最佳情况下，你会留下 4 双完整的袜子，而在最坏的情况下，只会有 3 双完整的袜子。假定 10 只袜子中，每只袜子丢失的概率都相同，找出最佳和最差情况下的发生概率。在平均情况下，你能够指望留下几双袜子呢？

5． **a.** ▷证明公式(2.1)，它代表十进制正整数用二进制表示时的位数。

b. ▷证明正整数 n 用二进制表示时的另一个位数计算公式：

$$b = \lceil log_2(n+1) \rceil$$

c. 如果想知道十进制数位的数量，类似的公式是什么样的？

d. 基于我们的分析框架，无论用二进制位数还是十进制位数来度量问题规模n，都不会影响分析结果。请说明一下原因。

6． 如何对任意排序算法进行增强，使其在最优情况下的键比较次数正好等于$n-1$

(n当然是列表的大小)。你认为这个方法对于任何排序算法都是一个有效的补充吗？

7． 高斯消去法是一个经典的算法，用于对包含n个未知数和n个线性方程的联立方程组求解。乘法是这种算法的基本操作，对于这样的方程组，该算法大约需要做$\frac{1}{3}n^3$次乘法运算。

a．用高斯消去法解一个 1000 个方程的方程组比解一个 500 个方程的方程组要多运行多少时间？

b．你打算买一台运行速度是当前所用机器 1000 倍的计算机。假设两台机器都运行相同的时间，新计算机可以求解的方程组规模和老计算机相比有什么变化？

8． 对于下列每种函数，请指出当参数值增加到 4 倍时，函数值会改变多少。

a．$log_2 n$　　**b．**$\sqrt{n}$　　**c．**n　　**d．**n^2　　**e．**n^3　　**f．**2^n

9． 请指出下面每一对函数中，第一个函数的增长量级(在输入规模以常数倍增长的情况下)比第二个函数的增长量级大还是小，还是二者相同。

a．$n(n+1)$和$2000n^2$　　**b．**$100n^2$和$0.01n^3$

c．$log_2 n$和$\ln n$　　**d．**$log_2^2 n$和$log_2 n^2$

e．2^{n-1}和2^n　　**f．**$(n-1)!$和$n!$

10．国际象棋的发明

a．传说国际象棋是许多世纪前由印度西北部的一个智者发明的。他将该发明献给了国王，国王很喜欢，许诺可以给智者任何想要的奖赏。智者要求以这种方式给他一些粮食：棋盘的第一个方格内只放 1 粒麦粒，第二格 2 粒，第三格 4 粒，第四格 8 粒，依次类推，直到 64 个方格全部放满。假定数 1 粒麦粒需要 1 秒钟时间，那么数完所有格子的麦粒需要多长时间？

b．如果每个格子的麦粒数是前一个格子的麦粒数加 2，那么数完所有格子的麦粒需要多长时间？

习题 2.1 提示

1． 虽然其中某些问题可能会有不同的回答，但这个问题的确就像看上去的那么简单。不过，不要忘了我们是如何度量一个整数的大小的。

2． **a．** 矩阵的和是这样定义的：它的元素等于给定矩阵的相应元素的和。

b． 矩阵的乘法需要两种操作：乘法和加法。应将哪一种操作作为基本操作？为什么？

3． 对于相同规模的不同输入，该算法的效率会有差异吗？

4． **a．**手套和袜子不同，它们是分左右手的。

b．只有两种本质上不同的结果。数一数得到每种结果各自有多少种不同的情况。

5． **a．**首先证明，如果一个十进制整数n在它的二进制表示中包含b位，那么

$$2\text{^}(b-1) \le n < 2\text{^}b$$

然后对这个不等式中的项以 2 为底求对数。

b．和公式(2.1)的证明是类似的。

c．公式基本一样，只是要根据基数的不同做些微的调整。

d．我们是如何改变对数的底的？

6．插入一个验证，判断问题是否已经解决。

7．本节研究过一个类似的问题。

8．既可以使用 $f(4n)$和 $f(n)$的差，也可以使用它们的比率，这依赖于哪种方法得到的答案更简洁。(如果可能，试着给出一个不依赖于n的答案。)

9．如有必要，对所讨论的函数进行化简。挑选出一个带乘数常量的，能够定义它们的增长量级的项(将在下一节中讨论回答这个问题的正式方法。然而，即使不知道这个方法，也应该能回答这个问题)。

10．a．使用公式$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$。

b．使用前 n 个奇数和的公式或者等差数列求和公式。

习题 2.1 答案

1. 答案如下：

   a. (i) n；(ii) 两个数相加；(iii) 否。

   b. (i) n 的大小(magnitude)，即它的二进制形式的位数；(ii) 两个整数相乘；(iii) 否。

   c. (i) n；(ii) 比较两个数字；(iii) 否(就标准列表扫描算法而言)。

   d. (i) 要么是两个输入数中较大的那个的大小(magnitude)，要么是两个输入数中较小的那个的大小，要么是两个输入数的大小之和；(ii) 模除；(iii)是。

   e. (i) n 的大小(magnitude)，即它的二进制形式的位数；(ii) 从剩余候选质数列表中消去一个数；(iii) 否

   f. (i) n；(ii) 两个数位相乘；(iii) 否

2. a. 基本操作是两个数字的加法运算。要进行 n^2 次(对要计算的矩阵中的 n^2 个元素各进行一次)。由于两个给定矩阵中的元素总数为 $N = 2n^2$，所以加法的总数也可以表示为 $n^2 = N/2$。

   b. 由于在大多数计算机上，乘法比加法更耗时，所以在考虑标准矩阵乘法算法的基本操作时，乘法是更好的选择。两个 $n \times n$ 矩阵的乘积的 n^2 个元素中的每一个都被计算为大小为 n 的两个向量的标量(点)积，这需要 n 次乘法。所以乘法总次数为：

$$n \cdot n^2 = n^3 = (N/2)^{3/2}$$

3. 该算法对每个大小为n的输入都要进行n次键比较，而对于经典的顺序查找算法，这个数字可能在 n 到 1 之间变化。

4. a. 最优情况显然是两只。最差情况是 12 只：比单手手套的数量(5+4+2)多一只。

   b. 只有两种结果：两只丢失的袜子是一双(最优情况)和不是一双(最差情况)。不同情况的总数(选择丢失袜子的方式)是$\binom{10}{2} = 45$。最佳情况的数量是 5；所以它的概率是$\frac{5}{45} = \frac{1}{9}$。最差情况的数量是$45 - 5 = 40$，概率是$\frac{40}{45} = \frac{8}{9}$。

5. 在其二进制形式中具有 b 个二进制位的最小正整数是$\underbrace{10\ldots0}_{b-1}$，即$2^{b-1}$；在其二进制形式中具有 b 个二进制位的最大正整数是$\underbrace{11\ldots0}_{b-1}$，即$2^{b-1}+2^{b-2}+\cdots+1=2^b-1$。

所以：

$$2^{b-1} \le n \le 2^b$$

相当于：

$$\log_2 2^{b-1} \le \log_2 n \le \log_2 2^b$$

或者：

$$b-1 \le \log_2 n < b$$

这些不等式意味着$b-1$是不小于$\log_2 n$的最小整数。换言之，根据向下取整(floor)函数的定义：

$$b = \lceil \log_2(n+1) \rceil$$

c. $B = \lfloor \log_{10} n \rfloor + 1 = \lceil \log_{10}(n+1) \rceil$

d. $b = \lfloor \log_2 n \rfloor + 1 \approx \log_2 n = \log_2 10 \log_{10} n \approx (\log_2 10)B$，其中$B = \lfloor \log_{10} n \rfloor + 1$。

也就是说，对于大的 n 值，在输入规模以常数倍增长的情况下，这两个输入规模指标大致相等。

6. 在应用一个排序算法之前，先比较其输入的相邻元素：如果对于每一个 $i = 0,\ldots,n-2$，$a_i \le a_i+1$，就直接停止。通常，这并不是一个值得增加的内容，因为除了非常特殊的输入，它在所有输入上都会减慢算法的速度。注意，一些排序算法(特别是冒泡排序和插入排序，分别在 3.1 节和 4.1 节讨论)在算法主体中内在地包含了这个测试。

7. a. $\frac{T(2n)}{T(n)} \approx \frac{c_M \frac{1}{3}(2n)^3}{c_M \frac{1}{3} n^3} = 8$，其中 C_M 是一次乘法的时间。

b. 在旧计算机上解决 n 阶系统的运行时间和在新计算机上解决 N 阶系统的运行时间可分别估计为$T_{old}(n) \approx C_M \frac{1}{3} n^3$和$T_{new}(N) \approx 10^{-3} C_M \frac{1}{3} N^3$。其中，$C_M$ 是旧计算机上一次乘法的时间。在等式$T_{old}(n) = T_{new}(N)$中，用这些估计值替换$T_{old}(n)$和$T_{new}(N)$，可得$C_M \frac{1}{3} n^3 \approx 10^{-3} C_M \frac{1}{3} N^3$或$\frac{N}{n} \approx 10$。

8.

a. $\log_2 4n - \log_2 n = (\log_2 4 + \log_2 n) - \log_2 n = 2$

b. $\frac{\sqrt{4n}}{\sqrt{n}} = 2$

c. $\frac{4n}{n} = 4$

d. $\frac{(4n)^2}{n^2} = 4^2$

e. $\frac{(4n)^3}{n^3} = 4^3$

f. $\frac{2^{4n}}{2^n} = 2^{3n} = (2^n)^3$

9. a. 在输入规模以常数倍增长的情况下，$n(n+1) \approx n^2$ 与 $2000n^2$ 具有相同的增长量级(平方)。

b. $100n^2$ (平方)具有比 $0.01n^3$ (立方)更低的增长量级。

c. 更改对数的底可通过以下公式完成：

$$log_a n = log_a b \, log_b n$$

因此，在输入规模以常数倍增长的情况下，所有对数函数具有相同的增长量级。

d. $log_2^2 n = \log_2 n \log_2 n$，而$log_2 n^2 = 2 \log n$。所以，$log_2^2 n$具有比$log_2 n^2$更高的增长量级。

e. 在输入规模以常数倍增长的情况下，$2^{n-1} = \frac{1}{2}2^n$具有与2^n一样的增长量级。

f. $(n-1)!$的增长量级低于 $n! = (n-1)!$。

10. a. 智者最后得到的麦粒总数为：

$$\sum_{i=1}^{64} 2^{i-1} = \sum_{j=0}^{63} 2^j = 2^{64} - 1 \approx 1.8 \cdot 10^{19}$$

(这比在整个地球都种上谷物所能收获的麦粒都要多许多倍)。如果数一粒麦粒需要一秒钟，需要 5850 亿年才能数完，比我们这个星球估计的年龄多 100 多倍。

b. 这种情况下的麦粒总量等于：

$$1 + 3 + ... + (2 \cdot 64 - 1) = 64^2$$

同样以一秒一粒的速度数麦粒，只需 1 小时 14 分钟。

习题 2.2

1． 从O，Ω和Θ中选择最合适的符号，指出顺序查找算法的时间效率类别 (参见2.1 节)。

a． 在最差情况下

b． 在最优情况下

c． 在平均情况下

2． 用 O，Ω和Θ的非正式定义来判断下列断言是真还是假。

a． $n(n+1)/2 \in O(n^3)$　　**b．** $n(n+1)/2 \in O(n^2)$

c． $n(n+1)/2 \in \Theta(n^3)$　　**d．** $n(n+1)/2 \in \Omega(n)$

3． 对于下列每一种函数，指出它们属于哪一种$\Theta(g(n))$类别 (尽量使用最简单的$g(n)$)，并进行证明。

a． $(n^2+1)^{10}$　　**b．** $\sqrt{10n^2+7n+3}$

c． $2n \, lg(n+2)^2 + (n+2)^2 \, lg \frac{n}{2}$　　**d．** $2^{n+1} + 3^{n-1}$

e． $\lfloor log_2 n \rfloor$

4． **a．** 表 2.1 包含一些在算法分析中常用的几个函数的值。这些值表明，下列函数的增长量级按升序排列：

$log \, n$，n，$n \, log \, n$，n^2，n^3，2^n，$n!$

这些函数值是否以一种数学的确定性来证明这个事实？

b． 请证明，这些函数的增长量级确实以这种方式升序排列。

5． 根据下列函数的增长量级按照从低到高的顺序对它们进行排序。

$(n-2)!$，$5\,lg(n+100)^{10}$，2^{2n}，$0.001n^4+3n^3+1$，$ln^2 n$，$\sqrt[3]{n}$，3^n

6．a .证明当$a_k>0$时，任何多项式$p(n)=a_k n^k+a_{k-1}n^{k-1}+\cdots+a_0$属于集合$\Theta(n^k)$。

b . 请证明，对于不同的底 a >0，指数函数 a^n 具有不同的增长量级。

7． 对下列断言进行证明(使用相关符号的定义)或者证伪(举出一个特定的反例)：

a . 如果$t(n)\in O(g(n))$，则$g(n)\in\Omega(t(n))$。

b . $\alpha>0$时，$\Theta(\alpha g(n))=\Theta(g(n))$。

c . $\Theta(g(n))=O(g(n))\cap\Omega(g(n))$。

d .对于任意两个定义在非负整数集合上的非负函数$t(n)$和$g(n)$，要么$t(n)\in O(g(n))$，要么$t(n)\in\Omega(g(n))$，要么两者都成立。

8． ▷证明本节的定理对于下列符号也成立：

a . Ω符号　　**b .** Θ符号

9． 本节曾提到，可用基于数组预排序的两部分算法检查数组中的元素是否全都唯一。

a .如果该预排序算法的时间效率属于$\Theta(n\,log\,n)$，那么整个算法的时间效率类别是怎样的呢？

b . 如果用于预排序的排序算法需要用到一个大小为n的额外数组，那么整个算法的空间效率类别是怎样的呢？

10．对于含有 n 个实数的有限非空集合 S，其**范围**(range)被定义为 S 中最大元素与最小元素的差。对于 S 的下列定义，用自然语言描述计算 S 的范围的一个算法，并用最合适的符号(O，Ω或者Θ)表示这些算法的时间效率类别。

a . 一个未排序的数组

b . 一个排序的数组

c . 一个排序的单链表

d . 一个二叉查找树

11．更轻或者更重？

你有 $n>2$ 个外观相似的硬币和一个没有砝码的天平。其中一枚为假币，但不知道它比真币重还是轻。所有真币都一样重。设计一个Θ (1)的算法来确定假币比真币重还是轻。

12．▷墙上的门　你面前是一堵朝两个方向无限延伸的墙。墙上有一扇门，但你并不知道门离你有多远，也不知道门位于哪个方向。只有走到门面前才能看到它。假设从初始位置到门要走n步(事先不知道n的大小)，请设计一个算法，使你最多走 $O(n)$步就能遇到门。([Par95])

习题 2.2 提示

1． 对该算法相应的基本操作进行计数(参见 2.1 节)，并使用 O，Θ和Ω的定义。

2． 先建立$n(n+1)/2$的增长量级，然后使用 O，Θ和Ω的非正式定义(本节给出过类似的例子)。

3． 对给定的函数进行化简。挑选出一个能够定义它们的增长量级的项。

4． **a．**仔细检查相关的定义。

b．对于列表中每一对相邻的函数，求它们比率的极限。

5． 先化简某些给定函数，再用表 2.2 中列出的函数来框定(anchor)每个给定函数，然后计算相应的极限，来证明它们的最终位置。

6． **a．**为了证明这个断言，既可以求极限，也可以使用数学归纳法。

b．计算$\lim_{n\to\infty} a_1^n/a_2^n$。

7． 用相应的定义证明 a，b，c 的正确性。为 d 构造一个反例(例如，可以构造两个函数，当自变量为奇数和偶数时，函数的行为是不同的)。

8． 对 a 部分的证明和 2.2 节对定理断言的证明是类似的。当然，需要用不同的不等式来限制和的下界。

9． 按照第一次提到该算法时课本中所使用的分析方案。

10．可以用直接算法求解 4 个问题，其中一个用 O 表示时间效率类型，另三个则用Θ表示。

11．这个问题能够用两次称重解决。

12．应该从初始位置交替地朝左和朝右走，直到遇到这扇门。

习题 2.2 答案

1. a. 由于$C_{worst}(n)=n$，所以$C_{worst}(n)\in\Theta(n)$。

   b. 由于$C_{best}(n)=1$，所以$C_{best}(1)\in\Theta(1)$。

   c. 由于$C_{avg}(n)=\frac{p(n+1)}{2}+n(1-p)=\left(1-\frac{p}{2}\right)+\frac{p}{2}$，其中$0\le p\le 1$，所以$C_{avg}(n)\in\Theta(n)$。

2. 由于$n(n+1)/2\approx n^2/2$是平方函数，所以：

   a. $n(n+1)/2\in O(n^3)$为真　　b. $n(n+1)/2\in O(n^2)$为真

   c. $n(n+1)/2\in\Theta(n^3)$为假　　d. $n(n+1)/2\in\Omega(n)$为真

3. a. 非正式地说，$(n^2+1)^{10}\approx(n^2)^{10}=n^{20}\in\Theta(n^{20})$。正式地说，

   $$\lim_{n\to\infty}\frac{(n^2+1)^{10}}{n^{20}}=\lim_{n\to\infty}\frac{(n^2+1)^{10}}{(n^2)^{10}}=\lim_{n\to\infty}\left(\frac{n^2+1}{n^2}\right)^{10}==\lim_{n\to\infty}\left(1+\frac{1}{n^2}\right)^{10}=1$$

   所以$(n^2+1)^{10}\in\Theta(n^{20})$。

   注意，另一个证明可基于二项式和练习 6a 的断言。

   b. 非正式地说，$\sqrt{10n^2+7n+3}\approx\sqrt{10n^2}=\sqrt{10}n\in\Theta(n)$，正式地说，

   $$\lim_{n\to\infty}\frac{\sqrt{10n^2+7n+3}}{n}=\lim_{n\to\infty}\sqrt{\frac{10n^2+7n+3}{n^2}}=\lim_{n\to\infty}\sqrt{10+\frac{7}{n}+\frac{3}{n^2}}=\sqrt{10}$$

   所以$\sqrt{10n^2+7n+3}\in\Theta(n)$。

   c. $2n\lg(n+2)^2+(n+2)^2\lg\frac{n}{2}=2n2\lg(n+2)+(n+2)^2(\lg n-1)\in\Theta(n\lg n)+\Theta(n^2\lg n)=\Theta(n^2\lg n)$

   d. $2^{n+1}+3^{n-1}=2^n2+3^n\frac{1}{3}\in\Theta(2^n)+\Theta(3^n)=\Theta(3^n)$

   e. 非正式地说，$\lfloor\log_2 n\rfloor\approx\log_2 n\in\Theta(\log n)$。正式地说，使用不等式$x-1<\lfloor x\rfloor\le$

x(参见附录 A)，我们获得一个上界：

$$\lfloor \log_2 n \rfloor \le \log_2 n$$

和一个下界：

$$\lfloor \log_2 n \rfloor > \log_2 n - 1 \ge \log_2 n - \frac{1}{2}\log_2 n \text{ (针对每个} n \ge 4) = \frac{1}{2}\log_2 n$$

因此，$\lfloor \log_2 n \rfloor \in \Theta(\log_2 n) = \Theta(\log n)$。

4. a. 增长量级和相应的符号 *O*、Ω 和 Θ 处理的是函数在 *n* 趋于无穷大时的渐近行为。因此，在 *n* 值的有限范围内，没有任何函数的具体数值(虽然表面上如此)能以数学的确定性来建立其增长量级。

b. $\lim_{n\to\infty} \frac{\log_2 n}{n} = \lim_{n\to\infty} \frac{(\log_2 n)'}{(n)'} = \lim_{n\to\infty} \frac{\frac{1}{n}\log_2 e}{1} = \log_2 e \lim_{n\to\infty} \frac{1}{n} = 0$

$\lim_{n\to\infty} \frac{n}{n\log_2 n} = \lim_{n\to\infty} \frac{1}{\log_2 n} = 0$

$\lim_{n\to\infty} \frac{n\log_2 n}{n^2} = \lim_{n\to\infty} \frac{\log_2 n}{n} = (\text{see the first limit of this exercise}) = 0$

$\lim_{n\to\infty} \frac{n^2}{n^3} = \lim_{n\to\infty} \frac{1}{n} = 0$

$\lim_{n\to\infty} \frac{n^3}{2^n} = \lim_{n\to\infty} \frac{(n^3)'}{(2^n)'} = \lim_{n\to\infty} \frac{3n^2}{2^n \ln 2} = \frac{3}{\ln 2}\lim_{n\to\infty} \frac{n^2}{2^n} = \frac{3}{\ln 2}\lim_{n\to\infty} \frac{(n^2)'}{(2^n)'}$
$= \frac{3}{\ln 2}\lim_{n\to\infty} \frac{2n}{2^n \ln 2} = \frac{6}{\ln^2 2}\lim_{n\to\infty} \frac{n}{2^n} = \frac{6}{\ln^2 2}\lim_{n\to\infty} \frac{(n)'}{(2^n)'}$
$= \frac{6}{\ln^2 2}\lim_{n\to\infty} \frac{1}{2^n \ln 2} = \frac{6}{\ln^3 2}\lim_{n\to\infty} \frac{1}{2^n} = 0$

$\lim_{n\to\infty} \frac{2^n}{n!} =$ (参见本节例3) 0

5.

$(n-2)! \in \Theta((n-2)!)$, $5\lg(n+100)^{10} = 50\lg(n+100) \in \Theta(\log n)$, $2^{2n} = (2^2)^n \in \Theta(4^n)$, $0.001n^4 + 3n^3 + 1 \in \Theta(n^4)$, $\ln^2 n \in \Theta(\log^2 n)$, $\sqrt[3]{n} \in \Theta(n^{\frac{1}{3}})$, $3^n \in \Theta(3^n)$

这些函数按增长量级从低到高排序如下：

$$5\lg(n+100)^{10},\quad \ln^2 n,\quad \sqrt[3]{n},\quad 0.001n^4 + 3n^3 + 1,\quad 3^n,\quad 2^{2n},\quad (n-2)!$$

6.

a. $\lim_{n\to\infty} \frac{p(n)}{n^k} = \lim_{n\to\infty} \frac{a_k n^k + a_{k-1}n^{k-1} + \ldots + a_0}{n^k} = \lim_{n\to\infty} (a_k + \frac{a_{k-1}}{n} + \ldots + \frac{a_0}{n^k})$
$= a_k > 0$。

因此，$p(n) \in \Theta(n^k)$。

b.

$$\lim_{n\to\infty} \frac{a_1^n}{a_2^n} = \lim_{n\to\infty} \left(\frac{a_1}{a_2}\right)^n = \begin{cases} 0 & \text{if } a_1 < a_2 \quad \Leftrightarrow \quad a_1^n \in o(a_2^n) \\ 1 & \text{if } a_1 = a_2 \quad \Leftrightarrow \quad a_1^n \in \Theta(a_2^n) \\ \infty & \text{if } a_1 > a_2 \quad \Leftrightarrow \quad a_2^n \in o(a_1^n) \end{cases}$$

7. a. 这个断言应该是正确的，因为它指出，如果 *t*(*n*)的增长量级小于或等于 *g*(*n*)的增长量级，那么 *g*(*n*)的增长量级就大于或等于 *t*(*n*)的增长量级。正式证明也很直截了当：

$$t(n) \le cg(n) \quad \text{对于所有} n \ge n_0, \text{其中} c > 0,$$

这意味着：

$$(\frac{1}{c})t(n) \le g(n) \quad \text{对于所有 } n \ge n_0$$

b. 断言$\Theta(\alpha g(n)) = \Theta(g(n))$应该为真，因为$\alpha g(n)$和$g(n)$唯一的区别就是一个正数常量倍数。所以，根据$\Theta$的定义，两者肯定具有相同的增长量级。需正式证明$\Theta(\alpha g(n)) \in \Theta(g(n))$，且$\Theta(g(n)) \in \Theta(\alpha g(n))$。设$f(n) \in \Theta(\alpha g(n))$，我们要证明$f(n) \in \Theta(g(n))$。已知：

$$f(n) \leq c\alpha g(n) \quad \text{对于所有} n \geq n_0 \ (\text{其中 } c > 0)$$

可以重写为：

$$f(n) \leq c_1 g(n) \quad \text{对于所有} n \geq n_0 \ (\text{其中 } c_1 = c\alpha > 0)$$

即$f(n) \in \Theta(g(n))$。

再设$f(n) \in \Theta(g(n))$；现在要证明在$\alpha > 0$的情况下，$f(n) \in \Theta(\alpha g(n))$。如果$f(n) \in \Theta\big(g(n)\big)$，那么：

$$f(n) \leq cg(n) \quad \text{对于所有} n \geq n_0 \ (\text{其中 } c > 0)$$

所以：

$$f(n) \leq \frac{c}{\alpha} ag(n) = c_1 \alpha g(n) \quad \text{对于所有} n \geq n_0 \ (\text{其中 } c_1 = \frac{c}{\alpha} > 0)$$

即$f(n) \in \Theta(\alpha g(n))$。

c. 这个断言明显是正确的(它相当于这个断言：当且仅当 $a \leqslant b$ 而且 $a \geqslant b$ 的时候，$a = b$)。要正式证明$\Theta(g(n)) \subseteq O(g(n)) \cap \Omega(g(n))$，而且$O(g(n)) \cap \Omega(g(n)) \subseteq \Theta(g(n))$，这本来就是 O，Ω和Θ的定义。

d. 这个断言为假。下面这对函数可作为反例：

$$t(n) = \begin{cases} n & \text{如果} n \text{为偶数} \\ n^2 & \text{如果} n \text{为奇数} \end{cases} \quad \text{和} \quad g(n) = \begin{cases} n^2 & \text{如果} n \text{为偶数} \\ n & \text{如果} n \text{为奇数} \end{cases}$$

8.

a. 需要证明如果$t_1(n) \in \Omega(g_1(n))$，而且$t_2(n) \in \Omega(g_2(n))$，那么$t_1(n) + t_2(n) \in \Omega(\max\{g_1(n), g_2(n)\})$。

证明 由于$t_1(n) \in \Omega(g_1(n))$，必然存在某个正数常量$c_1$和某个非负整数$n_1$，使得：

$$t_1(n) \geq c_1 g_1(n) \quad \text{对于所有} n \geq n_1$$

由于$t_2(n) \in \Omega(g_2(n))$，必然存在某个正常量$c_2$和某个非负整数$n_2$，使得：

$$t_2(n) \geq c_2 g_2(n) \quad \text{对于所有} n \geq n_2$$

设 $c = \min\{c1, c2\}$，而且 $n \geqslant \max\{n1, n2\}$，以便可以使用这两个不等式。将上面的两个不等式相加，可以得到以下结果：

$$\begin{aligned} t_1(n) + t_2(n) &\geq c_1 g_1(n) + c_2 g_2(n) \\ &\geq cg_1(n) + cg_2(n) = c[g_1(n) + g_2(n)] \\ &\geq c \max\{g_1(n), g_2(n)\} \end{aligned}$$

因此，$t_1(n) + t_2(n) \in \Omega(\max\{g_1(n), g_2(n)\})$，$O$ 定义所要求的常数c和n_0分别为$\min\{c_1, c_2\}$和$\max\{n_1, n_2\}$。

b. 可直接利用现成的证明：课本中已证明的定理(O 部分)、本练习(a)所证明的断言(Ω 部分)以及 Θ 的定义(练习 7c)。

9. a. 由于算法的排序部分的运行时间仍将主导第二部分的运行时间，所以是前者决定了整个算法的时间效率。正式地说，它遵循以下等式：

$$\Theta(n \log n) + O(n) = \Theta(n \log n)$$

其有效性很容易以与本节定理相同的方式证明。

b. 由于算法的第二部分不使用额外的空间，空间效率类别将由第一(排序)部分的空间效率类别决定。所以，答案是 $\Theta(n)$。

10. a. 扫描数组，找到其元素中的最大值和最小值，然后计算它们之间的差值。该算法的时间效率为 $\Theta(n)$ 。注意：虽然可通过大约 $1.5n$ 次比较找到 n 个元素的一个数组中的最大值和最小值(参见习题 2.3 的题 5 和习题 5.1 的题 2 的答案)，但这自然改变不了线性效率类别。

b. 对于一个已排序的数组，可以简单地计算其第一个和最后一个元素的差值：$A[n - 1] - A[0]$。时间效率类别显然是 $\Theta(1)$。

c. 最小元素在链表的第一个节点，因此它的值可以在恒定时间内得到。最大元素在最后一个节点，只有遍历整个链表才能到达，这需要线性时间。计算这两个值之间的差值需要恒定的时间。所以，时间效率类别为 $\Theta(n)$。

d. 二叉查找树中最小(最大)的元素在最左边(最右边)的节点。要到达它，需要从根开始，沿着左子(右子)指针链，直到到达一个左子(右子)指针为空的节点。取决于树的结构，这个节点链的长度可以在 1 到 n 个节点之间。所以，到达最后一个节点的时间属于 $O(n)$类别。整个算法的运行时间也是线性的：$O(n)+O(n)+ \Theta(1) = O(n)$。

11. 这个谜题可通过两次称重来解决。首先，如果 n 是奇数，就拿一个硬币，如果 n 是偶数，就拿两个硬币。然后把剩下的偶数硬币均分成两组，把它们放在天平的两个盘子里。如果重量相同，所有这些硬币都是真的，假币就在拿出来的硬币中。所以，提前拿出来的一个或两个硬币可与同等数量的真币进行称重：如果前者重量较轻，则假币较轻，否则，假币较重。如果第一次称重发现重量不一，则取较轻的一组，如果其中的硬币数量是奇数，则在其中加入最初拿出来的硬币之一(肯定是真币)。将所有这些硬币均分成两组并称重。如果重量相同，所有这些硬币都是真的，因此假币更重；否则，其中包含假币，假币更轻。

注意：这个谜题提供了一个非常罕见的例子，即无论问题的实例(这里是指硬币的数量)有多大，都可以用相同数量的基本操作(即两次称重)来解决。当然，如果考虑将一枚硬币放在天平上作为算法的基本操作，那么算法的效率将是 $\Theta(n)$，而不是 $\Theta(1)$。

12. 这里的关键思路是交替向右和向左走，每次都以指数方式远离初始位置。这个思路的一个简单实现是采取以下步骤，直到在途中找到门为止：对于 $i = 0,1,...$ 向右走2^i步，回到初始位置，向左走2^i，再回到初始位置。设$2^{k-1} \le n \le 2^k$，这个算法找到门所需的步数可以这样估算：

$$\sum_{i=0}^{k-1} 4 \cdot 2^i + 3 \cdot 2^k = 4(2^k - 1) + 3 \cdot 2^k < 7 \cdot 2^k = 14 \cdot 2^{k-1} < 14n$$

因此，算法产生的步数属于 $O(n)$类别。(注意：用更好的算法改进乘法常量并不困难。)

习题 2.3

1． 计算下列求和表达式的值。

a． $1 + 3 + 5 + 7 + \cdots + 999$

b． $2+4+8+16+\cdots+1024$

c． $\sum_{i=3}^{n+1} 1$　　**d．** $\sum_{i=3}^{n+1} i$　　**e．** $\sum_{i=0}^{n-1} i(i+1)$

f． $\sum_{j=1}^{n} 3^{j+1}$　　**g．** $\sum_{i=1}^{n}\sum_{j=1}^{n} ij$　　**h．** $\sum_{i=1}^{n} 1/i(i+1)$

2． 计算下列求和的增长量级。用$\Theta(g(n))$的形式给出，其中$g(n)$要尽可能简单。

a． $\sum_{i=0}^{n-1}(i^2+1)^2$　　**b．** $\sum_{i=2}^{n-1} lg\ i^2$

c． $\sum_{i=1}^{n}(i+1)2^{i-1}$　　**d．** $\sum_{i=0}^{n-1}\sum_{j=0}^{i-1}(i+j)$

3． $x_1, x_2, \cdots, x_n$ 的采样方差n可以这样计算：

$$\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}，\ 其中\bar{x}=\frac{\sum_{i=1}^{n} x_i}{n}$$

或者

$$\frac{\sum_{i=1}^{n} x_i^2-\frac{(\sum_{i=1}^{n} x_i)^2}{n}}{n-1}$$

根据每个公式，对求方差中需要用到的除法、乘法和加/减运算(加法和减法常常被捆绑在一起)的次数进行计算和比较。

4． 考虑下面的算法。

算法 *Mystery*(*n*)
```
//输入：非负整数 n
S ← 0
for i ← 1 to n do
    S ← S + i * i
return S
```

a．该算法求的是什么？

b．它的基本操作是什么？

c．该基本操作执行了多少次？

d．该算法的效率属于哪一类？

e．对该算法进行改进，或设计一个更好的算法，然后指出它们的效率类别。如果做不到，请证明确实做不到。

5． 考虑下面的算法。

算法 *Secret*(*A*[0..*n* – 1])
```
//输入：包含 n 个实数的数组 A[0..n – 1]
minval ← A[0]; maxval ← A[0]
for i ← 1 to n – 1 do
    if A[i] < minval
        minval ← A[i]
    if A[i] > maxval
        maxval ← A[i]
return maxval – minval
```

基于本算法回答第 4 题的 a～e 小题。

6． 考虑下面的算法。

算法 *Enigma*(*A*[0..*n* – 1, 0..*n* – 1])
```
//输入：一个实数矩阵 A[0..n – 1, 0..n – 1]
for i ← 0 to n – 2 do
    for j ← i + 1 to n – 1 do
```

if $A[i, j] \neq A[j, i]$
return false
return true

基于本算法回答第 4 题的 a～e 小题。

7． 通过减少算法所做的加法次数来改进矩阵乘法算法的实现(参见例 3)。这种变化会为算法的效率带来什么影响？

8． 针对“储物柜的门”谜题中所有门的开关总次数(习题 1.1 的第 12 题)，确定其渐近增长量级。

9． 证明下面的公式：

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

可以使用数学归纳法，也可以像 10 岁的高斯(1777—1855)一样，用洞察力来解决该问题。这个小学生长大以后成为有史以来最伟大的数学家之一。

10．心算术 如下图所示，在一个 10×10 的表格中，用相同数字填满对角线，心算表格中所有数字之和([Cra07]，问题 1.33)。

1	2	3			⋯			9	10
2	3						9	10	11
3						9	10	11	
					9	10	11		
				9	10	11			
⋮			9	10	11				⋮
		9	10	11					
	9	10	11						17
9	10	11						17	18
10	11				⋯		17	18	19

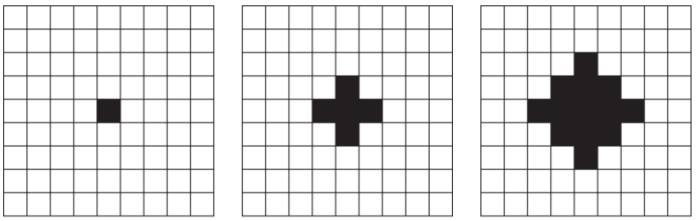
11． 以下是某重要算法的一个版本，本书后面会研究该算法：

算法 *GE*(***A***[0..*n* – 1, 0..*n*])
//输入：一个 *n* 行 *n* + 1 列的实数矩阵 ***A***[0..*n* – 1, 0..*n*]
for *i* ← 0 **to** *n* – 2 **do**
for *j* ← *i* + 1 **to** *n* – 1 **do**
for *k* ← *n* **downto** *i* **do**
A[*j*, *k*] ← *A*[*j*, *k*]–*A*[*i*, *k*]**A*[*j*, *i*]/*A*[*i*, *i*]

a． ▷该算法的时间效率类别是什么？

b． ▷该算法在效率方面有什么重要缺陷？如何弥补以提高该算法的运行速度？

12．冯·诺依曼邻居问题 考虑以下算法：从一个 1×1 的方格开始，在它的 *n* 次迭代中，每次迭代都在周围加一圈方格，在第 *n* 次迭代的时候有多少个挨在一起的方格？([Gar99]，根据元胞自动机理论，答案等于 *n* 阶冯·诺依曼邻居中的元胞数。)下图给出了 *n* = 0, 1, 2 时的结果。



13．书页编号 在一本 1000 页的书中，找出全部页码所需的十进制数位总数。假设从 1 开始连续编号。

习题 2.3 提示

1． 使用附录 A 列出的通用求和公式和法则。在应用这些公式之前，可能需要做一些简单的代数运算。

2． 在附录 A 给出的求和式中，找出和目标求和式类似的公式，试着将后者转化为前者。注意，在建立求和式的增长量级之前，并不一定要求出它的闭合公式。

3． 根据题目中的公式计算即可。

4． **a．**如有必要，可以跟踪该算法，得到它对于 n 的一些较小值的输出(例如，$n = 1, 2, 3$)。

b．我们针对课本中讨论的例子问过同样的问题。其中有一个和本题尤其相关。

c．遵循本节描述的方案。

d．如果是作为 n 的函数，能从 c 的答案立即得到答案。你可能还要以 n 的二进制位数为函数的输入值来给出答案。(为什么？)

e．你没有在某些地方遇到过这种求和吗？

5． **a．**如有必要，可以跟踪该算法，得到它对于 n 的一些较小值的输出(例如，$n = 1, 2, 3$)。

b．我们针对课本中讨论的例子问过同样的问题。其中有一个和本题尤其相关。

c．既可遵循本节描述的方案，建立一个求和式并求解，也可以直接回答问题。(两种方法都试试。)

d．答案从 c 的答案中立即可知。

e．该算法是不是在每次迭代时都必须做两次比较？这个思路可以进一步展开，以得到一个更显著的改进——试着对包含 4 个元素的一个数组做做看，再把结论一般化。但是，我们是不是可以指望得到一个比线性效率更好的算法呢？

6． **a．**元素 $A[i, j]$和 $A[j, i]$相对于矩阵的主对角线是对称的。

b．这里只有一个候选对象。

c．研究最坏情况即可。

d．从 c 的答案立即可知。

e．拿该算法解决的问题和该算法的解题方法做比较。

7． 计算 n 个数的和需要做 $n-1$ 次加法。该算法在计算乘积矩阵的每一个元素时，需

要做多少次加法？

8． 为门的开关次数建立一个求和式，然后用附录 A 的公式计算它的渐近增长量级。

9． 对于归纳法中一般性步骤的证明，请使用公式：

$$\sum_{i=1}^{n} i = \sum_{i=1}^{n-1} i + n$$

少年高斯在计算$1+2+\cdots+99+100$的和时，注意到这个式子可以按照 50 对数字的和来计算，其中每一对的和都相等。

10．这个问题其实来自于一本华尔街面试问题集锦，至少有两种不同的方法求解该问题。

11．a．建立一个求和式的难度并不大。然而，在应用标准求和公式和法则时，却要比前面的例子付出更多的努力。

b．优化该算法的最内层循环。

12．针对该算法 n 次迭代以后的方块数建立一个求和式，然后对它化简以得到一个闭合形式的解。

13．推导一个数位总数的公式，该公式是页码数 n 的一个函数，其中$1 \le n \le 1000$。将函数的定义域按照几个自然区间分段较为便捷。

习题 2.3 答案

1.

a. $1+3+5+7+\ldots+999 = \sum_{i=1}^{500}(2i\text{-}1) = \sum_{i=1}^{500} 2i\text{-}\sum_{i=1}^{500} 1 = 2\frac{500*501}{2}\text{-}500 = 250\,000$

也可使用奇数整数求和公式：

$$\sum_{i=1}^{500}(2i\text{-}1) = 500^2 = 250\,000$$

或者使用等差数列求和公式，其中$a_1 = 1$，$a_n = 999$，且$n = 500$：

$$\frac{(a_1+a_n)n}{2} = \frac{(1+999)500}{2} = 250\,000$$

b. $2+4+8+16+\ldots+1\,024 = \sum_{i=1}^{10} 2^i = \sum_{i=0}^{10} 2^i - 1 = (2^{11}-1)-1 = 2\,046$

或者使用几何数列求和公式，其中 $a = 2$，$q = 2$，且 $n = 9$：

$$a\frac{q^{n+1}-1}{q-1} = 2\frac{2^{10}-1}{2-1} = 2\,046$$

c. $\sum_{i=3}^{n+1} 1 = (n+1) - 3 + 1 = n - 1$

d. $\sum_{i=3}^{n+1} i = \sum_{i=0}^{n+1} i - \sum_{i=0}^{2} i = \frac{(n+1)(n+2)}{2} - 3 = \frac{n^2+3n-4}{2}$

e. $\sum_{i=0}^{n-1} i(i+1) = \sum_{i=0}^{n-1} (i^2+i) = \sum_{i=0}^{n-1} i^2 + \sum_{i=0}^{n-1} i = \frac{(n-1)n(2n-1)}{6} + \frac{(n-1)n}{2}$

$= \frac{(n^2-1)n}{3}$

f. $\sum_{j=1}^{n} 3^{j+1} = 3\sum_{j=1}^{n} 3^j = 3[\sum_{j=0}^{n} 3^j - 1] = 3[\frac{3^{n+1}-1}{3-1} - 1] = \frac{3^{n+2}-9}{2}$

g. $\sum_{i=1}^{n}\sum_{j=1}^{n} ij = \sum_{i=1}^{n} i \sum_{j=1}^{n} j = \sum_{i=1}^{n} i\frac{n(n+1)}{2} = \frac{n(n+1)}{2}\sum_{i=1}^{n} i = \frac{n(n+1)}{2}\frac{n(n+1)}{2}$

$= \frac{n^2(n+1)^2}{4}$

h. $\sum_{i=1}^{n} 1/i(i+1) = \sum_{i=1}^{n}(\frac{1}{i} - \frac{1}{i+1})$

$= (\frac{1}{1} - \frac{1}{2}) + (\frac{1}{2} - \frac{1}{3}) + \ldots + (\frac{1}{n-1} - \frac{1}{n}) + (\frac{1}{n} - \frac{1}{n+1}) = 1 - \frac{1}{n+1} = \frac{n}{n+1}$

2.

a. $\sum_{i=0}^{n-1} (i^2+1)^2 = \sum_{i=0}^{n-1} (i^4 + 2i^2 + 1) = \sum_{i=0}^{n-1} i^4 + 2\sum_{i=0}^{n-1} i^2 + \sum_{i=0}^{n-1} 1$

$\in \Theta(n^5) + \Theta(n^3) + \Theta(n) = \Theta(n^5)$ (or just $\sum_{i=0}^{n-1} (i^2+1)^2 \approx \sum_{i=0}^{n-1} i^4 \in \Theta(n^5)$)

b. $\sum_{i=2}^{n-1} \log_2 i^2 = \sum_{i=2}^{n-1} 2\log_2 i = 2\sum_{i=2}^{n-1} \log_2 i = 2\sum_{i=1}^{n} \log_2 i - 2\log_2 n$

$\in 2\Theta(n\log n) - \Theta(\log n) = \Theta(n\log n)$

c. $\sum_{i=1}^{n} (i+1)2^{i-1} = \sum_{i=1}^{n} i2^{i-1} + \sum_{i=1}^{n} 2^{i-1} = \frac{1}{2}\sum_{i=1}^{n} i2^i + \sum_{j=0}^{n-1} 2^j$

$\in \Theta(n2^n) + \Theta(2^n) = \Theta(n2^n)$ (or $\sum_{i=1}^{n} (i+1)2^{i-1} \approx \frac{1}{2}\sum_{i=1}^{n} i2^i \in \Theta(n2^n)$)

d. $\sum_{i=0}^{n-1}\sum_{j=0}^{i-1} (i+j) = \sum_{i=0}^{n-1} [\sum_{j=0}^{i-1} i + \sum_{j=0}^{i-1} j] = \sum_{i=0}^{n-1} [i^2 + \frac{(i-1)i)}{2}] = \sum_{i=0}^{n-1} [\frac{3}{2}i^2 - \frac{1}{2}i]$

$= \frac{3}{2}\sum_{i=0}^{n-1} i^2 - \frac{1}{2}\sum_{i=0}^{n-1} i \in \Theta(n^3) - \Theta(n^2) = \Theta(n^3)$

3. 对于第一个公式：$D(n) = 2, M(n) = n, \mathrm{A}(n) + S(n) = [(n-1) + (n-1)] + (n-1) = 3n - 1$。

对于第二个公式：$D(n) = 2$，$M(n) = n + 1$，$\mathrm{A}(n) + S(n) = [(n-1) + (n-1)] + 2 = 2n$。

4. a. 计算的是 $S(n) = \sum_{i=1}^{n} i^2$ 。

b. 基本操作是乘法(或者，如乘法和加法需要相同的时间，则为两者中的任何一个)。

c. $C(n) = \sum_{i=1}^{n} 1 = n$ 。

d. $C(n) = n \in \Theta(n)$。由于二进制位数$b = \lfloor \log_2 n \rfloor + 1 \approx \log_2 n$，即$n \approx 2^b$，所以

$C(n) \approx 2^b \in \Theta(2^b)$。

e. 使用以下公式在Θ(1)时间内求和(假设无论操作数有多大，算术运算的时间都保持恒定)：

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

5. a. 计算的是“范围”(range)，即数组最大和最小元素的差值。

b. 基本操作是元素比较。

c. $C(n) = \sum_{i=1}^{n-1} 2 = 2(n-1)$。

d. $\Theta(n)$。

e. 针对某些输入(但不是最差情况)的明显改进是将两个 if 语句替换成下面这个：

if A[i] < *minval* *minval* ← *A*[*i*]

else if *A*[*i*] > *maxval* *maxval* ← *A*[*i*]

另一个更微妙和实质性的改进基于以下观察结果：对迄今为止看到的最小值和最大值进行更新时，相较于针对每个元素进行，针对一对连续两个元素进行更有效。如果先对两个这样的元素进行比较，则更新只需再进行两次比较，每对元素总共三次比较。注意，分治算法也可获得同样的改进(参见习题 5.1 的题 2)

6. a. 如果输入矩阵是对称的，算法返回 true；否则返回 false。

b. 基本操作是对两个矩阵元素的比较。

c.

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1)$$

$$= \sum_{i=0}^{n-2} (n-1-i) = (n-1) + (n-2) + \ldots + 1 = \frac{(n-1)n}{2}$$

d. 平方：

$$C_{worst}(n) \in \Theta(n^2) \text{ (or } C(n) \in O(n^2))$$

e. 该算法已经最优了，因为任何解决这个问题的算法，在最差情况下，都必须比较矩阵上三角部分的$(n-1)n/2$ 个元素和下三角部分对称的元素，这正是该算法所做的。

7. 用以下代码替换 *j* 循环的主体：

$C[i,j] \leftarrow A[i,0] * B[0,j]$
for $k \leftarrow 1$ **to** $n-1$ **do**
$C[i,j] \leftarrow C[i,j] + A[i,k] * B[k,j]$

这使加法数量从n^3减至$n^3 - n^2$，但乘法数量仍为n^3。算法效率类别还是立方。

8. 设 $T(n)$是所有门的开关总次数。问题陈述意味着：

$$T(n) = \sum_{i=1}^{n} \lfloor n/i \rfloor$$

由于$x - 1 < \lfloor x \rfloor \leq x$，而且$\sum_{i=1}^{n} 1/i \approx \ln n + \gamma$，其中$\gamma = 0.5772$ ...(参见附录 A)，所以：

$$T(n) \leq \sum_{i=1}^{n} n/i = n \sum_{i=1}^{n} 1/i \approx n(\ln n + \gamma) \in \Theta(n \log n)$$

类似地：

$$T(n) > \sum_{i=1}^{n}(n/i - 1) = n\sum_{i=1}^{n} 1/i - \sum_{i=1}^{n} 1 \approx n(\ln n + \gamma) - n \in \Theta(n \log n)$$

这意味着$T(n) \in \Theta(n\ log\ n)$。

9. 下面用数学归纳法证明针对每个正整数n：

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

(i) 基本步骤：对于$n = 1$：

$$\sum_{i=1}^{n} i = \sum_{i=1}^{1} i = 1，且\ \frac{n(n+1)}{2}\Big|_{n=1} = \frac{1(1+1)}{2} = 1$$

(ii) 归纳步骤：假设对于正整数n：

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

我们需证明：

$$\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$$

证明过程如下：

$$\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{n(n+1)+2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$$

少年高斯进行以下求和时：

$$1 + 2 + \ldots + 99 + 100$$

发现它可以通过求 50 个数对之和来实现，每一对的和都是 101：

$$1 + 100 = 2 + 99 = \ldots = 50 + 51 = 101$$

所以，总和等于$50 \times 101 = 5050$(据说当时是老师布置的一个作业)。高斯的想法很容易推广到一个任意的 n，具体就是以下两个等式相加：

$$S(n) = 1 + 2 + \cdots + (n-1) + n$$
$$S(n) = n + (n-1) + \cdots + 2 + 1$$

得：

$$2S(n) = (n+1)n$$

即：

$$S(n) = \frac{n(n+1)}{2}$$

10. 目标是计算(心算)下表所有数字之和：

1	2	3			⋯			9	10
2	3						9	10	11
3						9	10	11	
					9	10	11		
				9	10	11			
⋮			9	10	11				⋮
		9	10	11					
	9	10	11						17
9	10	11						17	18
10	11				⋯		17	18	19

第一种方法是基于这样的观察：相对于连接左下角和右上角的对角线，对称方格中任意两个数字之和都是 20：1+19，2+18，2+18……等等。所以，由于有(10 × 10－10)/2 = 45 个这样的对子(对角线上的方块数要从方块总数中减去)，所以该对角线外的数字之和等于 20×45=900。对角线上的数字之和是 10×10=100，所以总和是 900 + 100 = 1000。

第二种方法是逐行(或逐列)计算总和。根据公式(S2)，第一行的数字之和等于 10×11/2=55。第二行的数字之和是 55 + 10，因为每个数字都比上面一行的数字大 1。其他各行的情况也是如此。因此，总和等于 55 +(55 + 10)+(55 + 20)+...+ (55+90) = 55×10 +(10+20+...+90)= 55×10+10×(1+2+...+9)=55×10 + 10×45 = 1000。

注意，第一种方法使用的是卡尔 •高斯 10 岁的时候用来求前一百个整数之和的技巧(练习 2.3 的第 9 题)。本题第二个解也使用了这个公式(实际使用了两次)。

11. a. 该算法所做的乘法次数 $M(n)$和除法次数 $D(n)$由同一个和给出：

$$\begin{aligned} M(n) &= D(n) = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}\sum_{k=i}^{n} 1 = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(n-i+1) = \\ &= \sum_{i=0}^{n-2}(n-i+1)(n-1-(i+1)+1) = \sum_{i=0}^{n-2}(n-i+1)(n-i-1) \\ &= (n+1)(n-1)+n(n-2)+\ldots+3*1 \\ &= \sum_{j=1}^{n-1}(j+2)j = \sum_{j=1}^{n-1}j^2+\sum_{j=1}^{n-1}2j = \frac{(n-1)n(2n-1)}{6}+2\frac{(n-1)n}{2} \\ &= \frac{n(n-1)(2n+5)}{6} \approx \frac{1}{3}n^3 \in \Theta(n^3) \end{aligned}$$

b. 效率低下的原因是算法最内层循环对比率 $A[j, i] / A[i, i]$进行重复求值。该比率实际并不随循环变量 k 而变。所以，在进入这个循环之前，只需计算一次这个循环不变式(loop invariant)：temp ← $A[j, i] / A[i, i]$；然后将最内层循环更改为：

$$A[j,k] \leftarrow A[j,k] - A[i,k] * temp$$

这一更改消除了算法中最昂贵的操作，即最内层循环中的除法。通过这种更改所获得的运行时间收益可以估算如下：

$$\frac{T_{old}(n)}{T_{new}(n)} \approx \frac{c_M\frac{1}{3}n^3 + c_D\frac{1}{3}n^3}{c_M\frac{1}{3}n^3} = \frac{c_M + c_D}{c_M} = \frac{c_D}{c_M} + 1$$

其中C_D和C_M分别为一次除法和一次乘法的时间。

12. 答案通过直接求和可得：

$$2\sum_{i=1}^{n}(2i-1) + (2n+1) = 2n^2 + 2n + 1$$

(也可通过注意到范围为 n 的冯 •诺依曼邻居的交替对角线上的方格分别构成两个大小为 $n + 1$ 和 n 的正方形来得到闭合解。)

13. 设 $D(n)$是前 n 个正整数(页码)中的十进制位的总数。前 9 个数是一位数，所以 $D(n) = n$，其中 $1 \leqslant n \leqslant 9$。因此：

$$D(n) = 9 + 2(n-9) \text{ ，其中} 10 \leq n \leq 99$$

在这个范围中，$D(n)$的最大值是 $D(99) = 189$。另外，已知有 900 个三位十进制数，因此：

$$D(n) = 189 + 3(n - 99)\ ，\ 其中 100 \le n \le 999$$

在这个范围中，$D(n)$的最大值是 $D(999) = 2889$。加上第 1000 页的 4 位数，我们得到 D(1000) = 2893。

习题 2.4

1． 解下列递推关系。

a． $x(n) = x(n-1) + 5$，其中$n > 1$，$x(1) = 0$

b． $x(n) = 3x(n-1)$，其中$n > 1$，$x(1) = 4$

c． $x(n) = x(n-1) + n$，其中$n > 0$，$x(0) = 0$

d． $x(n) = x(n/2) + n$，其中$n > 1$，$x(1) = 1$ (对于 $n = 2^k$ 的情况求解)

e． $x(n) = x(n/3) + 1$，其中$n > 1$，$x(1) = 1$ (对于 $n = 3^k$ 的情况求解)

2． 对于计算$n!$的递归算法 $F(n)$，建立其递归调用次数的递推关系并求解。

3． 考虑下列递归算法，该算法用来计算前n个立方的和：$S(n) = 1^3 + 2^3 + \cdots + n^3$。

```
算法  S(n)
      //输入：正整数 n
      //输出：前 n 个立方的和
      if n = 1 return 1
      else return S(n – 1)+n*n*n
```

a． 建立该算法的基本操作执行次数的递推关系并求解。

b． 如果将这个算法和直截了当的非递归算法比较，你做何评价？

4． 考虑下面的递归算法。

```
算法  Q(n)
      //输入：正整数 n
      if n = 1 return 1
      else return Q(n – 1)+2*n – 1
```

a． 建立该函数值的递推关系并求解，以确定该算法计算的是什么。

b． 建立该算法所做的乘法运算次数的递推关系并求解。

c． 建立该算法所做的加减运算次数的递推关系并求解。

5．　汉诺塔谜题

a． 汉诺塔谜题最早由法国数学家卢卡斯于 19 世纪 90 年代提出，当时的版本是这样的：当 64 个盘子被从梵塔上移走时，世界末日也就来临了。如果祭司一分钟移动一个盘子(假设该祭司不吃不睡，而且长生不老)，请估计移走全部盘子一共需要多少年？

b． 在该算法中，要移动第i大的盘子$(1 \le i \le n)$一共需要移动多少步？

c． 为汉诺塔谜题设计一个非递归的算法，并用你熟悉的语言实现。

6．　▷受限汉诺塔　考虑以下版本的汉诺塔谜题：有 n 个盘子需要从柱子 A 借助

柱子 *B* 搬到柱子 *C*，且任何移动要么把一个盘子搬到柱子 *B*，要么从柱子 *B* 上搬走一个盘子。(还是禁止把较大的盘子放在较小的盘子上面。)设计一个递归算法解决这个问题，并且确定实现它所需要的移动次数。

7． ▷**a．**请证明，对于一个任意十进制正整数 *n* 来说，递归算法 *BinRec*(*n*)所做的加法运算的精确次数是$\lfloor log_2 n \rfloor$。

b．对于该算法的非递归版本，建立其所做的加法运算次数的递推关系并求解(参见 2.3 节，例 4)。

8． **a．**请基于公式 $2^n = 2^{n-1} + 2^{n-1}$ 设计一个递归算法。当n是任意非负整数时，该算法能够计算 2^n 的值。

b．建立该算法所做的加法运算次数的递推关系并求解。

c．为该算法构造一棵递归调用树，然后计算它所做的递归调用的次数。

d．对于该问题的求解来说，这是一个好算法吗？

9． 考虑下面的递归算法。

```
算法 Riddle(A[0..n − 1])
    //输入：包含 n 个实数的数组 A[0..n − 1]
    if n = 1 return A[0]
    else temp ← Riddle(A[0..n − 2])
        if temp ≤ A[n − 1] return temp
        else return A[n − 1]
```

a．该算法计算的是什么？

b．建立该算法所做的基本操作次数的递推关系并求解。

10．考虑下面的算法，它检查一个由邻接矩阵表示的图是不是完全图。

```
算法 GraphComplete(A[0..n − 1, 0..n − 1])
    //输入：一个无向图 G 的邻接矩阵 A[0..n − 1, 0..n − 1]
    //输出：如果 G 是完全图，返回 1，否则返回 0
    if n = 1 return 1 //按照定义，一个顶点的图是完全图
    else
        if not GraphComplete(A[0..n − 2, 0..n − 2]) return 0
        else for j←0 to n − 2 do
            if A[n − 1, j]) = 0 return 0
        return 1
```

这个算法最坏情况下的效率类型是什么？

11．一个n阶方阵

$$A = \begin{bmatrix} a_{0,0} \cdots a_{0,n-1} \\ a_{1,0} \cdots a_{1,n-1} \\ \vdots \\ a_{n-1,0} \cdots a_{n-1,n-1} \end{bmatrix}$$

的行列式记作 det $\boldsymbol{A}$。当$n = 1$时，我们可以把它定义为$a_{0,0}$；当$n > 1$时，则定义为下面的递推关系：

$$det \quad A = \sum_{j=0}^{n-1} s_j a_{0,j} \, det \, A_j$$

当 j 为偶数时，s_j 取+1；当 j 为奇数时，s_j 取−1。$a_{0,j}$ 是位于第 0 行和第 j 列的元素，A_j 是把第 0 行和第 j 列从矩阵A中删除后获得的$n-1$阶方阵。

a．假设有一个实现该定义的算法，建立该算法的乘法运算次数的递推关系并求解。

b．不对该递推关系求解，你认为它的增长量级和$n!$相比会有什么结论？

 **12．重温冯·诺依曼邻居问题** 建立一个递推关系并求解，以计算 n 阶冯·诺依曼邻居的细胞数(参见习题 2.3 的第 12 题)。

 **13．煎饼** 煎饼 有 n 个饼需要用一个小平底锅来煎，但锅中一次只能放 2 个饼。每个饼都需要两面煎，不管是煎一个饼还是同时煎两个饼，煎好饼的一面都需用时 1 分钟。假设要在最短时间内完成该任务，考虑下列递归算法。如果 $n \leqslant 2$，一个饼单独煎并翻面，两个饼则同时煎并翻面。如果 $n > 2$，两个饼同时煎并翻面，然后对余下 $n-2$ 个饼递归地应用同样的过程。

a．给出该算法煎 n 个饼所需要时间的递推关系并求解。

b．对于任意 $n > 0$ 个饼，该算法完成任务的时间并不是最少的，为什么？

c．给出一个在最少时间内完成煎饼任务的正确递归算法。

 **14．▷名人问题** n 个人中的名人是指这样一个人：他不认识别人，但别人都认识他。任务就是找出这样一个名人，但只能通过询问“你认识他/她吗？”这种问题来完成。设计一个高效算法，找出该名人或者确定这群人中没有名人。你的算法在最坏的情况下需要提问多少次？

习题 2.4 提示

1．其中每一个递推关系都可以用反向替换法求解。

2．本节已建立了该算法的乘法次数的递推关系，也求过解。题目中的递推关系和乘法次数的递推关系基本上是相等的。

3． **a．**这个问题类似于计算 $n!$ 的递归算法的基本操作执行次数的递推关系。

b．写一个非递归算法的伪代码并确定它的效率。

4． **a．**注意，这里要求的是函数值的递推关系，而不是算法操作次数的递推关系。就按照问题中的伪代码建立递推式。用前向替换法是很容易对这个递推式求解的(参见附录 B)。

b．这个问题和我们讨论过的一个问题非常类似。

c．应该把对 n 递减时所做的减法也包含进来。

5． **a．**利用本节推导出的盘子移动次数的公式。

b．对三个盘子的情况求解该问题，研究一下每个盘子的移动次数。将这个观察结果推而广之，并对于 n 个盘子的一般情况，证明这个结论的正确性。

6．所求算法及其分析方法可参见该谜题的经典版本。因为额外的约束，需要求解的小规模实例超过 2 个。

7． **a．**根据 n 是奇数还是偶数两种情况分别讨论。并证明，对于这两种情况，$\lfloor \log_2 n \rfloor$

都能满足递推关系和初始条件。

b．参考该算法的伪代码即可。

8． **a．**就使用公式 $2^n = 2^{n-1} + 2^{n-1}$，不要对它化简。不要忘了给出一个递归停止条件。

b．本节研究过一个类似的算法。

c．本节研究过一个类似的问题。

d．算法本身的效率类型不好并不意味着该算法一定不好。例如，尽管汉诺塔谜题的经典算法是指数级效率，但它仍然是最优的。所以，如果要声称一个算法不好，就要拿出一个更好的算法来。

9． **a．**对于 $n = 1$ 和 $n = 2$ 的情况跟踪该算法应该会有所帮助。

b．这和本节讨论的一个例子非常相似。

10．求解基本操作的数量有两个办法：一是解一个递推关系式，二是直接计算最坏情况下算法需要检查的邻接矩阵的元素个数。

11． **a．**利用定义中的公式，得到该算法的乘法次数的递推关系。

b．研究该递推关系的右边部分。计算 $M(n)$的前几个值也会有所帮助。

12．利用邻居的对称性，得出该算法在第 n 次迭代时加入方块数的一个简单公式。

13．煎三个饼的最少时间不到 4 分钟。

14．首先求解一个简单版本，假设必须存在一个名人。

习题 2.4 答案

1.

a. $x(n) = x(n-1) + 5$，其中$n > 1, \quad x(1) = 0$

$$\begin{aligned} x(n) &= x(n-1) + 5 \\ &= [x(n-2) + 5] + 5 = x(n-2) + 5 \cdot 2 \\ &= [x(n-3) + 5] + 5 \cdot 2 = x(n-3) + 5 \cdot 3 \\ &= \ldots \\ &= x(n-i) + 5 \cdot i \\ &= \ldots \\ &= x(1) + 5 \cdot (n-1) = 5(n-1) \end{aligned}$$

注：也可使用 n 项等差数列求和公式来求解：

$$x(n) = x(1) + d(n-1) = 0 + 5(n-1) = 5(n-1)$$

b. $x(n) = 3x(n-1)$，其中 $n > 1,\ x(1) = 4$

$$\begin{aligned} x(n) &= 3x(n-1) \\ &= 3[3x(n-2)] = 3^2x(n-2) \\ &= 3^2[3x(n-3)] = 3^3x(n-3) \\ &= \ldots \\ &= 3^i x(n-i) \\ &= \ldots \\ &= 3^{n-1}x(1) = 4 \cdot 3^{n-1} \end{aligned}$$

注：也可使用 n 项等比数列求和公式来求解：

$$x(n) = x(1)q^{n-1} = 4 \cdot 3^{n-1}$$

c. $x(n) = x(n-1) + n$，其中$n > 0, \quad x(0) = 0$

$$\begin{aligned} x(n) &= x(n-1) + n \\ &= [x(n-2) + (n-1)] + n = x(n-2) + (n-1) + n \\ &= [x(n-3) + (n-2)] + (n-1) + n = x(n-3) + (n-2) + (n-1) + n \\ &= \ldots \\ &= x(n-i) + (n-i+1) + (n-i+2) + \cdots + n \\ &= \ldots \\ &= x(0) + 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \end{aligned}$$

d. $x(n) = x(n/2) + n \quad \text{for } n > 1, \quad x(1) = 1$ (对于$n = 2^k$的情况)

$$\begin{aligned} x(2^k) &= x(2^{k-1}) + 2^k \\ &= [x(2^{k-2}) + 2^{k-1}] + 2^k = x(2^{k-2}) + 2^{k-1} + 2^k \\ &= [x(2^{k-3}) + 2^{k-2}] + 2^{k-1} + 2^k = x(2^{k-3}) + 2^{k-2} + 2^{k-1} + 2^k \\ &= \ldots \\ &= x(2^{k-i}) + 2^{k-i+1} + 2^{k-i+2} + \cdots + 2^k \\ &= \ldots \\ &= x(2^{k-k}) + 2^1 + 2^2 + \cdots + 2^k = 1 + 2^1 + 2^2 + \cdots + 2^k \\ &= 2^{k+1} - 1 = 2 \cdot 2^k - 1 = 2n - 1 \end{aligned}$$

e. $x(n) = x(n/3) + 1 \quad \text{for } n > 1, \quad x(1) = 1$ (对于 $n = 3^k$)的情况

$$\begin{aligned} x(3^k) &= x(3^{k-1}) + 1 \\ &= [x(3^{k-2}) + 1] + 1 = x(3^{k-2}) + 2 \\ &= [x(3^{k-3}) + 1] + 2 = x(3^{k-3}) + 3 \\ &= \ldots \\ &= x(3^{k-i}) + i \\ &= \ldots \\ &= x(3^{k-k}) + k = x(1) + k = 1 + \log_3 n \end{aligned}$$

2. $C(n) = C(n-1) + 1$，$C(0) = 1$(当$n = 0$时，只有一次调用，但无乘法)。

$$\begin{aligned} C(n) &= C(n-1) + 1 = [C(n-2) + 1] + 1 = C(n-2) + 2 = \ldots \\ &= C(n-i) + i = \ldots = C(0) + n = 1 + n \end{aligned}$$

3. a. 设 $M(n)$是算法执行的乘法总次数。我们得到以下递推式：

$$M(n) = M(n-1) + 2, \quad M(1) = 0$$

用反向替换法来求解：

$$\begin{aligned} M(n) &= M(n-1) + 2 \\ &= [M(n-2) + 2] + 2 = M(n-2) + 2 + 2 \\ &= [M(n-3) + 2] + 2 + 2 = M(n-3) + 2 + 2 + 2 \\ &= \ldots \\ &= M(n-i) + 2i \\ &= \ldots \\ &= M(1) + 2(n-1) = 2(n-1) \end{aligned}$$

b. 非递归版本的伪代码如下所示：

算法 *NonrecS*(*n*)

//非递归地计算前 *n* 个立方之和

//输入：正整数 *n*

//输出：前 *n* 个立方之和

$S \leftarrow 1$

for $i \leftarrow 2$ **to** *n* **do**

$S \leftarrow S + i * i * i$

return *S*

该算法做的乘法次数是：

$$\sum_{i=2}^{n} 2 = 2\sum_{i=2}^{n} 1 = 2(n-1)$$

这与递归版本的结果一模一样，非递归版本还不会产生递归版本和栈相关的时间和空间开销。

4. a. $Q(n) = Q(n-1) + 2n - 1$，其中 $n > 1, \quad Q(1) = 1$

计算前几项得到以下结果：

$$\begin{aligned} Q(2) &= Q(1) + 2 \cdot 2 - 1 = 1 + 2 \cdot 2 - 1 = 4 \\ Q(3) &= Q(2) + 2 \cdot 3 - 1 = 4 + 2 \cdot 3 - 1 = 9 \\ Q(4) &= Q(3) + 2 \cdot 4 - 1 = 9 + 2 \cdot 4 - 1 = 16 \end{aligned}$$

所以，似乎$Q(n) = n^2$？将这个公式代入递归方程和初始条件来检验这一假设。左手边得出$Q(n) = n^2$，右手边则得出：

$$Q(n-1) + 2n - 1 = (n-1)^2 + 2n - 1 = n^2$$

初始化条件立即就能验证：$Q(1) = 1^2 = 1$。

b. 当$n > 1$时，$M(n) = M(n-1) + 1$，$M(1) = 1$。通过反向替换法(这与阶乘的例子几乎完全一样，参见本节例 1)或应用 *n* 项等差数列公式来求得：$M(n) = n - 1$。

c. 设 *C*(*n*)是算法所做的加减法次数。在 $n > 1$ 的情况下，*C*(*n*)的递推式是 *C*(*n*)= *C*(*n*−1) + 3，*C*(1) = 0。通过反向替换法或应用 *n* 项等差数列公式来求得$C(n) = 3(n-1)$。注意，如果不把递减 *n* 所需的减法计算在内，那么递归式是：在 $n > 1$ 的情况下，*C*(*n*)= *C*(*n*−1) + 2，*C*(1) = 0。它的解是$C(n) = 2(n-1)$。

5. a. 走的步数由以下公式给出：$M(n) = 2^n - 1$。即：

$$\frac{2^{64} - 1}{60 \cdot 24 \cdot 365} \approx 3.5 \cdot 10^{13}\text{年}$$

(我们这个宇宙的年龄估计为$13 \cdot 10^9$年。)

b. 注意，对于第 i 个盘子的每一次移动，该算法首先将所有比它小的盘的塔移到另一个木桩上(这需要移动一次第(i+1)个盘)，然后，在第 i 个盘子移动之后，这个较小的塔被移到它的上面(这又需要移动一次第(i+1)个盘)。因此，对于第 i 个盘子的每一次移动，该算法都正好要将第(i+1)个盘子移动两次。由于在 i = 1 的时候，移动次数等于 1，所以有以下关于第 i 个盘子移动次数的递推式：

在$1 < i < n$的情况下，$m(i+1) = 2m(i)$；$m(1) = 1$

在 i=1，2，...，n 的情况下，它的解是$m(i) = 2^{i-1}$(得到这个公式的最简单方法是使用等比数列的通项公式)。请注意，这个答案与总移动次数的公式完美吻合：

$$M(n) = \sum_{i=1}^{n} m(i) = \sum_{i=1}^{n} 2^{i-1} = 1 + 2 + \cdots + 2^{n-1} = 2^n - 1$$

c. 非递归版本请自行完成。下面给出一个用 C++实现的递归版本：

```cpp
#include <iostream>
using namespace std;

void move_rings(int n, int src, int dest, int other);   // 移动多个盘
void move_a_ring(int src, int dest);                    // 移动一个盘
int i = 1; // 统计移动次数

int main()
{

    int n = 4; // 假定有4个盘子
    move_rings(n, 1, 3, 2); // 从塔1移至塔3，塔2是过渡塔
    return 0;
}

void move_rings(int n, int src, int dest, int other) {
    // 终止条件
    if (n == 1) {
        move_a_ring(src, dest);
    }
    else {
        move_rings(n - 1, src, other, dest);    // 递归调用1
        move_a_ring(src, dest);                 // 移动一个盘子
        move_rings(n - 1, other, dest, src);    // 递归调用2
    }
}

// 移动一个盘子
void move_a_ring(int src, int dest) {
    cout << "第" << i << "次移动，" << "从塔" << src << "移至塔"
        << dest << endl;
    i++;
}
```

6. 如果 n = 1，先将一个盘子从柱子 A 移到柱子 B，再从柱子 B 移到柱子 C，如果 n > 1，执行以下操作：

递归地将顶部 n - 1 个盘子从柱子 A 通过柱子 B 移到柱子 C

将盘子从柱子 A 移至柱子 B

递归地将 n - 1 个盘子从柱子 C 通过柱子 B 移到柱子 A

将盘子从柱子 B 移至柱子 C

递归地将 n - 1 个盘子从柱子 A 通过柱子 B 移到柱子 C。

移动次数 $M(n)$的递推式是：

在$n > 1$ 的情况下，$\mathrm{M}(n) = 3M(n-1) + 2$；$m(1) = 2$

用反向替换法来求解：

$$
\begin{aligned}
M(n) &= 3M(n-1)+2 \\
&= 3[3M(n-2)+2]+2 = 3^2M(n-2)+3\cdot 2+2 \\
&= 3^2[3M(n-3)+2]+3\cdot 2+2 = 3^3M(n-3)+3^2\cdot 2+3\cdot 2+2 \\
&= \ldots \\
&= 3^iM(n-i)+2(3^{i-1}+3^{i-2}+\cdots+1) = 3^iM(n-i)+3^i-1 \\
&= \ldots \\
&= 3^{n-1}M(1)+3^{n-1}-1 = 3^{n-1}\cdot 2+3^{n-1}-1 = 3^n-1
\end{aligned}
$$

7. a. 可通过替换来验证$A(n) = \lfloor log_2 n \rfloor$满足加法次数的递推式：

在$n > 1$ 的情况下，$A(n) = A(\lfloor n/2 \rfloor) + 1$

设 n 为偶数，即 $n = 2k$。

左手边是：

$$A(n) = \lfloor \log_2 n \rfloor = \lfloor \log_2 2k \rfloor = \lfloor \log_2 2 + \log_2 k \rfloor = (1 + \lfloor \log_2 k \rfloor) = \lfloor \log_2 k \rfloor + 1$$

右手边是：

$$A(\lfloor n/2 \rfloor) + 1 = A(\lfloor 2k/2 \rfloor) + 1 = A(k) + 1 = \lfloor \log_2 k \rfloor + 1$$

设 n 为奇数，即 $n = 2k + 1$。

左手边是：

$$A(n) = \lfloor \log_2 n \rfloor = \lfloor \log_2(2k+1) \rfloor = \text{using } \lfloor \log_2 x \rfloor = \lceil \log_2(x+1) \rceil - 1$$
$$\lceil \log_2(2k+2) \rceil - 1 = \lceil \log_2 2(k+1) \rceil - 1$$

右手边是：

$$A(\lfloor n/2 \rfloor) + 1 = A(\lfloor 2k/2 \rfloor) + 1 = A(k) + 1 = \lfloor \log_2 k \rfloor + 1$$

初始条件可立即得到验证：$A(1) = \lfloor log_2\, 1 \rfloor = 0$。

b. 非递归版本加法次数的递推关系和递归版本完全一样：

在$n > 1$ 的情况下，$A(n) = A(\lfloor n/2 \rfloor) + 1$；$A(1) = 0$

它的解是：$A(n) = \lfloor log_2\, n \rfloor + 1$。

8. a.

算法 *Power*(n)

//用公式 $2^n = 2^{n-1} + 2^{n-1}$ 递归地计算2^n

//输入：非负整数 n

//输出：返回2^n

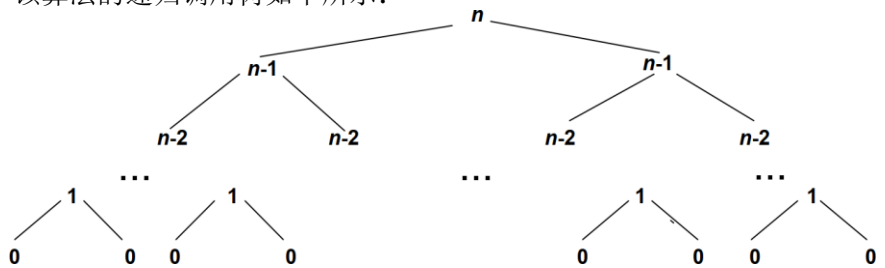
if $n = 0$ **return** 1

else return *Power*(n - 1) + *Power*(n - 1)

b. $A(n) = 2A(n-1) + 1,\ A(0) = 0$

$$\begin{aligned}
A(n) &= 2A(n-1) + 1 \\
&= 2[2A(n-2) + 1] + 1 = 2^2 A(n-2) + 2 + 1 \\
&= 2^2[2A(n-3) + 1] + 2 + 1 = 2^3 A(n-3) + 2^2 + 2 + 1 \\
&= \ldots \\
&= 2^i A(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 1 \\
&= \ldots \\
&= 2^n A(0) + 2^{n-1} + 2^{n-2} + \cdots + 1 = 2^{n-1} + 2^{n-2} + \cdots + 1 = 2^n - 1
\end{aligned}$$

c. 该算法的递归调用树如下所示：



注意，和汉诺塔谜题的类似树相比，它多了一个比较层级。

d. 这是一个非常糟糕的算法，因为它比简单地将一个累加器乘以 2n 倍的算法要差得多，更不用说本书后面讨论的更高效的算法。即使只允许加法，将 2 加2^{n-1}次也比这个算法好。

9. a. 该算法计算给定数组中最小元素的值。

b. 键比较次数的递推关系是：

在$n > 1$的情况下，$C(n) = C(n-1) + 1$；$C(1) = 0$

用反向替换法求解：$C(n) = n - 1$。

10. 设$C_w(n)$是最坏情况下(完全图)检查邻接矩阵元素的次数。我们对$C_w(n)$有以下递推关系：

在$n > 1$的情况下，$C_w(n) = C_w(n-1) + n - 1$；$C_w(1) = 0$

用反向替换法求解：

$$\begin{aligned} C_w(n) &= C_w(n-1) + n - 1 \\ &= [C_w(n-2) + n - 2] + n - 1 \\ &= [C_w(n-3) + n - 3] + n - 2 + n - 1 \\ &= \ldots \\ &= C_w(n-i) + (n-i) + (n-i+1) + \cdots + (n-1) \\ &= \ldots \\ &= C_w(1) + 1 + 2 + \cdots + (n-1) = 0 + (n-1)n/2 = (n-1)n/2 \end{aligned}$$

也可通过观察在最差情况下，算法会检查图的邻接矩阵主对角线下方的所有元素，从而获得结果。

11. a. 设$M(n)$为算法根据公式$\det A = \sum_{j=0}^{n-1} s_j a_{0,j}\, det\, A_j$所执行的乘法次数。如果拿掉对$S_j$(它的值不过是$\pm 1$)的乘法运算，那么：

$$M(n) = \sum_{j=0}^{n-1} (M(n-1) + 1)$$

即：

在$n > 1$的情况下，$M(n) = n(M(n-1) + 1)$；$M(1) = 0$

b. 由于$M(n) = nM(n-1) + n$，序列$M(n)$增长到无穷大的速度至少与$F(n) = nF(n-1)$定义的阶乘函数一样。

12. 对于冯·诺依曼邻居的四个对称边，在第n次迭代时，每个边所增加的方格数都等于n。所以，我们得到$S(n)$的以下递归关系，即第n次迭代后邻居中的总方格数。

在$n > 0$的情况下，$S(n) = S(n-1) + 4n$；$S(0) = 1$

用反向替换法求解：

$$\begin{aligned} S(n) &= S(n-1) + 4n \\ &= [S(n-2) + 4(n-1)] + 4n = S(n-2) + 4(n-1) + 4n \\ &= [S(n-3) + 4(n-2)] + 4(n-1) + 4n = S(n-3) + 4(n-2) + 4(n-1) + 4n \\ &= \ldots \\ &= S(n-i) + 4(n-i+1) + 4(n-i+2) + \cdots + 4n \\ &= \ldots \\ &= S(0) + 4 \cdot 1 + 4 \cdot 2 + \cdots + 4n = 1 + 4(1 + 2 + \cdots + n) \\ &= 1 + 4n(n+1)/2 = 2n^2 + 2n + 1 \end{aligned}$$

13. a. 设$T(n)$是按所给算法煎 n 个饼所需的分钟数，那么$T(n)$有以下递推关系：

在$n > 2$的情况下，$T(n) = T(n-2) + 2$；$T(1) = 2$，$T(2) = 2$

它的解是：针对每个大于 0 的偶数 n，$T(n) = n$；针对每个大于 0 的奇数 n，$T(n) = n + 1$。这既可以用反向替换法求得，也可利用等差数列通项公式。

b. 对于任何大于 1 的奇数 n，该算法都无法在最短时间内煎 n 个饼。特别是，煎

3 个饼需要 $T(3)=4$ 分钟，而一个人完全可用 3 分钟完成。首先，煎饼 1 和 2 的第一面。然后，煎饼 1 的第二面和饼 3 的第一面。最后，煎饼 2 和 3 的第二面。

c. 如果 $n \leqslant 2$，就煎饼的的每一面(如果 $n=2$，两个饼一起煎)。如果 $n=3$，按照 b 部分问题的答案，用 3 分钟煎好。如果 $n>3$，把两个饼放在一起煎每一面，然后用同样的算法煎剩余 $n-2$ 个饼。现在，煎 n 个饼所需分钟数的递推关系如下：

在$n>3$的情况下，$T(n)=T(n-2)+2$；$T(1)=2$，$T(2)=2$，$T(3)=3$

对于每个大于 1 的 n，该算法需要 n 分钟来完成这个工作。这才是可能的最少时间，因为 n 个饼有 $2n$ 面要煎，而任何算法在一分钟内都不可能煎超过两面。对于 $n=1$ 的情况，该算法显然也是最优的，需要两分钟来煎一个饼的两面。

注：$n=3$ 的情况是一个著名的谜题，它至少可以追溯到 1943 年。它对于任意 n 的算法版本包括在 A.Levitin 和 M.Levitin 所著的《算法谜题》(Algorithmic Puzzles)中，牛津大学出版社，2011 年，第 16 题。

14. 这个问题可通过一个递归算法来解决。事实上，只要问一个问题，就可将能成为名人的人数减少 1，对剩下的$n-1$人进行递归求解，然后通过不超过两个问题验证返回的解。下面是这个算法的详细说明。

如果 $n=1$，返回该人作为名人。如果 $n>1$，则按以下步骤进行：

第 1 步：从给定的组中选择两人，例如 A 和 B，问 A 是否认识 B，如果 A 认识 B，就把 A 从剩下可能成为名人的人中移除；如果 A 不认识 B，就把 B 从这个组中移除。

第 2 步：对于剩下的$n-1$个可能成为名人的人，递归地解决这个问题。

第 3 步：如果第 2 步返回的解表明在$n-1$人的组中没有名人，那么更大的n人的组中也不可能有名人。如果第 2 步确定了 A 或 B 以外的一个人是名人，例如 C，就问 C 是否认识第 1 步中移除的人。如果答案是否定的，就问第 1 步移除的人是否认识 C。如果第二个问题的答案是肯定的，返回 C 作为名人；否则返回“没有名人”。如果第 2 步确定 B 是名人，就问 B 是否认识 A。如果答案是否定的，返回 B 作为名人；否则返回“没有名人”。如果第 2 步确定 A 是名人，就问 B 是否认识 A。如果答案是肯定的，返回 A 作为名人；否则返回“没有名人”。

设$Q(n)$是最差情况下需要提问的次数，它的递推式是：

在$n>2$的情况下，$Q(n)=Q(n-1)+3$；$Q(2)=2$，$Q(1)=0$

它的解是：

在$n>1$的情况下，$Q(n)=2+3(n-2)$；$Q(1)=0$

注：关于这个问题的讨论，包括这个算法的 Pascal 风格的伪代码实现，可参见 Udi Manber 所著的《算法引论》(Introduction to Algorithms: A Creative Approach)，Addison-Wesley，1989。

习题 2.5

1． 去找一个专门讨论斐波那契数应用的网站，做一番研究。

 **2．斐波那契兔子问题** 一个人把一对兔子用围墙围住。如果最初的一对兔子(一雌一雄)是新生的，并且所有的兔子在出生后的第一个月都不能繁殖，但是在之后的每个月末都能够生出一对(一雌一雄)兔子，那么一年后围墙里将会有多少对兔子？

 **3．爬梯子** 假设每一步可以爬一级或两级梯子，爬一部n级梯子一共可以用几种方法？(例如，三级的梯子可用三种不同的方法爬：1-1-1，1-2 和 2-1。)

4. 斐波那契数列前 n 项(即在 $F(0)$，$F(1)$，$F(2)$，…，$F(n-1)$中)有多少偶数？给出对于所有 $n > 0$ 都成立的闭合公式。

5． 用直接替换法验证，函数$\frac{1}{\sqrt{5}}(\varphi^n - \hat{\varphi}^n)$在$n > 1$时，满足递归式(2.6)；在 $n = 0$ 和 1 时，满足初始条件(2.7)。

6． Java 的基本数据类型 int 和 long 的最大值分别是 $2^{31} - 1$ 和 $2^{63} - 1$。当n最小为多少时，第n个斐波那契数能够使下面的类型溢出？

a．int 类型 **b．**long 类型

7． 考虑基于定义的计算第n个斐波那契数的递归算法 $F(n)$，假设 $C(n)$和 $Z(n)$分别是 $F(n)$调用 $F(1)$和 $F(0)$的次数。请证明：

a．$C(n) = F(n)$ **b．**$Z(n) = F(n-1)$

8． 改进算法Fib，使它只需要$\Theta(1)$的额外空间。

9． 证明等式：

当$n \geq 1$时，$\begin{bmatrix} F(n-1) & F(n) \\ F(n) & F(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n$

10． ▷当两个连续的斐波那契数$F(n)$*和*$F(n-1)$作为欧几里得算法的输入时，该算法需要做多少次模除运算？

 **11．分解斐波那契矩形** 给定一个矩形，它的边长是两个连续的斐波那契数。设计一个算法来把它分解为正方形，且具有相同尺寸的正方形不超过两个。你的算法的时间效率类型是什么？

12．选择一种语言，计算第n个斐波那契数的最后 5 个数位。要求分别实现下列两种算法：(a)基于定义的递归算法 $F(n)$，(b)基于定义的循环算法 Fib(n)。做一个实验，看看在你的计算机上，这两个程序一分钟内能处理的最大的n是多少。

习题 2.5 提示

1．试试搜索引擎。

2．为 n 个月后的兔子数建立一个方程，该月兔子数取决于前几个月的兔子数。

3．有若干种解题方法。其中最优雅的一种可以把问题和这一节的主题联系起来。

4． 首先写出前 10 个斐波那契数的值，然后寻找明显的模式。

5． 把 ϕ^n 和 $\hat{\phi}^n$ 分别代入递推方程中会更简单。为什么这样就足够了呢？

6． 用 $F(n)$的近似公式计算超过给定数字的 n 值。

7． 建立 $C(n)$和 $Z(n)$的递推关系，当然，也要建立合适的初始条件。

8． 在该算法的每次迭代时，需要的全部信息只不过是最后两个连续的斐波那契数的值。利用这个事实对算法进行修改。

9． 用数学归纳法证明。

10． 先考虑一个较小的例子，例如计算 gcd (13, 8)。

11． 利用矩形维数的特别性质。

12． 一个整数 N 的最后 k 位数可以用 $N \bmod 10^k$ 来计算。在算法的每一次操作时都对 10^k 求模(参见附录 A)可避免斐波那契数的指数级增长。还要注意，2.6 节会专门对算法的经验分析做全面的讨论。

习题 2.5 答案

1. n/a
2. 设 $R(n)$为 n 月末的兔子对数。显然，$R(0)=1$，$R(1)=1$。对于每个大于 1 的 n，兔子对数 $R(n)$等于 $n-1$ 月末的兔子对数 $R(n-1)$加 n 月末出生的兔子对数；根据问题的假设，这等于 $R(n-2)$，即 $n-2$ 月末的兔子对数。因此，我们得到以下递推关系：

   在$n>1$ 的情况下，$R(n)=R(n-1)+R(n-2)$；$R(0)=1$，$R(1)=1$

   下表给出了这个递推关系所定义的序列的前 13 项的值，称为**斐波那契数**。

n	0	1	2	3	4	5	6	7	8	9	10	11	12
$R(n)$	1	1	2	3	5	8	13	21	34	55	89	144	233

   注意，$R(n)$与典型的斐波那契数列略有不同，后者由相同的递推方程 $F(n) = F(n-1)+F(n-2)$定义，但初始条件不同，即 $F(0)=0$ 和 $F(1)=1$。很明显，对于 $n\geq 0$，$R(n) = F(n+1)$。

   注：这个问题被比萨的列奥纳多(又名斐波纳契)列入他 1202 年的《Liber Abaci》(计算之书)中，他在书中主张使用印度传来的阿拉伯数字。
3. 设 $W(n)$为爬一个 n 级梯子的不同方式的数量。其中 $W(n-1)$种方式从爬一级梯子开始，$W(n-2)$种方式从两级梯子开始，那么：

   在$n\geq 3$ 的情况下，$W(n)=W(n-1)+W(n-2)$；$W(1)=1$，$W(2)=2$

   无论是“从头开始”解这个递推关系，还是敏锐地注意到它的解比典型的斐波那契序列 $F(n)$领先一步，都可以得到在$n\geq 1$ 的情况下，$W(n)=F(n+1)$。
4. 考虑到 $F(0) = 0$ 和 $F(1) = 1$，而且序列中后续每个元素的规则是 $F(n) = F(n-1) + F(n-2)$，很容易看到斐波那契数形成了以下模式：

   偶，奇，奇，偶，奇，奇，.....

   所以，前 n 个斐波那契数中的偶数个数可通过公式$\lceil n/3 \rceil$得到
5. 将φ^n代入方程的左侧，由于φ是特征方程$r^2-r-1=0$的根之一，所以得到$F(n)-F(n-1)-F(n-2)=\varphi^n-\varphi^{n-1}-\varphi^{n-2}=\varphi^{n-2}(\varphi^2-\varphi-1)=0$。对$\hat{\varphi}^n$的验证同理。由于方程$F(n)-F(n-1)-F(n-2)=0$是同质和线性的，所以它的解$\varphi^n$和$\hat{\varphi}^n$的任何线性组合，即任何形式的$\alpha\varphi^n+\beta\hat{\varphi}^n$序列，也是$F(n)-F(n-$

1) $-F(n-2)=0$的解，尤其是在斐波那契数列$\frac{1}{\sqrt{5}}\varphi^n-\frac{1}{\sqrt{5}}\hat{\varphi}^n$的情况下。两个初始条件都是以相当直截了当的方式确定的。

6. a. 问题是要找出使$F(n)>2^{31}-1$的最小 n 值。使用取整到最接近整数的公式$F(n)=\frac{1}{\sqrt{5}}\varphi^n$，我们得到以下不等式(近似)：

$$\frac{1}{\sqrt{5}}\phi^n>2^{31}-1 \quad 或 \quad \phi^n>\sqrt{5}(2^{31}-1)$$

两侧都取自然对数后，得到：

$$n>\frac{\ln(\sqrt{5}(2^{31}-1))}{\ln\phi}\approx 46.3$$

所以，答案是 $n = 47$。

b. 类似地，需要找出使$F(n)>2^{63}-1$的最小的 n，即：

$$\frac{1}{\sqrt{5}}\phi^n>2^{63}-1, \quad 或 \quad \phi^n>\sqrt{5}(2^{63}-1)$$

两侧都取自然对数，得到：

$$n>\frac{\ln(\sqrt{5}(2^{63}-1))}{\ln\phi}\approx 92.4$$

所以，答案是 $n = 93$。

7. 由于 $F(n)$是通过公式 $F(n)=F(n-1)+F(n-2)$递归计算的，所以 $C(n)$和 $Z(n)$的递归方程和 $F(n)$一样。C(n)和 $Z(n)$初始条件分别是：

$$C(0)=0, \quad C(1)=1 \quad 和 \quad Z(0)=1, \quad Z(1)=0$$

由于 $C(n)$和 $F(n)$的递归方程和初始条件都一样，所以 $C(n) = F(n)$。另外，也很容易看出 $Z(n) = F(n-1)$的断言是成立的，因为 $Z(n)$数列的形式是：

$$1,\ 0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ \ldots,$$

也就是说，它与斐波那契数向右移动一个位置的情况相同。要进行正式证明，可检查数列 F(n − 1)(其中 F(−1)被定义为 1)是否满足以下递归关系来正式证明：

在$n>1$ 的情况下，$Z(n)=Z(n-1)+Z(n-2)$；$Z(0)=1$，$Z(1)=0$

它也可以通过数学归纳法或推导出 $Z(n)$的一个精确公式来证明，并证明这个公式与 $F(n)$的精确公式中用 $n-1$ 代替 n 的值相同。

8.

算法 *Fib2*(*n*)

//只用两个变量计算第 *n* 个斐波那契数

//输入：一个非负整数 *n*

//输出：第 *n* 个斐波那契数

$u \leftarrow 0$; $v \leftarrow 1$

for $i \leftarrow 2$ **to** n **do**

  $v \leftarrow v + u$

  $u \leftarrow v - u$

if $n = 0$ **return** 0

else return v

9. (i) 基于斐波那契数列的定义，在 n =1 的时候，等式成立。

(ii) 假设对于正整数 n：

$$\begin{bmatrix} F(n-1) & F(n) \\ F(n) & F(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n$$

那么我们需要证明：

$$\begin{bmatrix} F(n) & F(n+1) \\ F(n+1) & F(n+2) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n+1}$$

证明过程如下所示：

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n+1} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n}$$

$$= \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} F(n-1) & F(n) \\ F(n) & F(n+1) \end{bmatrix} = \begin{bmatrix} F(n) & F(n+1) \\ F(n+1) & F(n+2) \end{bmatrix}$$

10. 主要的观察结果是，欧几里德算法用另一对连续的斐波那契数代替两个连续的斐波那契数作为其输入，即：

$$\text{对于每个} n \geq 4，\ \gcd(F(n), F(n-1)) = \gcd(F(n-1), F(n-2))$$

对于每个$n \geq 4$，由于$F(n-2) < F(n-1)$，即：

$$F(n) = F(n-1) + F(n-2) < 2F(n-1)$$

因此，对于每个$n \geq 4$，$F(n)$除以 $F(n-1)$的商和余分别为 1 和 $F(n)-F(n-1)=F(n-2)$。这正是我们开头所断言的。这又导致了除法次数 $D(n)$的以下递推式：

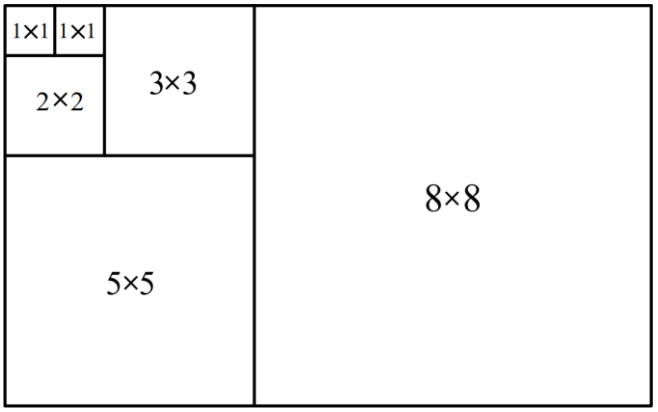
$$\text{对于每个} n \geq 4，\ D(n) = D(n-1) + 1；\ D(3) = 1$$

其初始条件 $D(3)=1$ 是通过对输入对 $F(3)$，$F(2)$，即 2,1 的算法追踪得到的。这个递归式的解是：

$$\text{对于每个} n \geq 3，\ D(n) = n - 2$$

(很容易就知道 $D(2)=1$， $D(1)=0$。)

11. 给定一个边长为 $F(n)$和 $F(n+1)$的矩形，问题可以通过以下递归算法求解。如果 n = 1，问题已经解决了，因为矩形是一个 1×1 的正方形。如果 $n>1$，将矩形分解为 $F(n)\times F(n)$的正方形和边长为 $F(n-1)$和 $F(n)$的矩形，然后用同样的算法将后者分解。下面用 8×13 的矩形来演示该算法。



由于该算法将边长为 $F(n)$和 $F(n+1)$的矩形分解为 n 个正方形——这可以通过解递推式 $S(n)=S(n-1)+1$，$S(1)=1$ 来正式得到，所以其时间效率属于$\Theta(n)$类别。

12. n/a

习题 2.6

1． 考虑一个著名的排序算法(本书后文会进一步研究)，其中插入了一个计数器来对键的比较次数进行计数。

```
算法  SortAnalysis(A[0..n − 1])
      //输入：包含 n 个可排序元素的一个数组 A[0..n − 1]
      //输出：键的比较总次数
      count ← 0
      for i ← 1 to n − 1 do
          v ← A[i]
          j ← i − 1
          while j > 0 and A[j] > v do
                count ← count + 1
                A[j + 1] ← A[j]
                j ← j − 1
          A[j + 1] ← v
      return count
```

比较计数器是否在正确的位置插入？如果认为正确，请证明；如果认为错误，请改正。

2． **a．**对于 20 个大小分别为 1 000，2 000，3 000，…，20 000 的随机数组，运行问题 1 中的程序，程序中应正确插入用于键比较计数的一个或多个计数器。

b．分析所得到的数据，建立一个对该算法平均效率的假设。

c．请估计，如果用相同的算法对一个规模为 25 000 的随机数组进行排序，预期的键比较次数应该是多少。

3． 以毫秒作为程序运行时间的度量单位，把题 2 再做一遍。

4． 基于以下基本操作运行次数的经验观察，猜测该算法的可能效率类型。

规模	1 000	2 000	3 000	4 000	5 000	6 000	7 000	8 000	9 000	10 000
次数	11 966	24 303	39 992	53 010	67 272	78 692	91 274	113 063	129 799	140 538

5． 如何变换坐标使一个对数散点图看上去像一个线性散点图？

6． 我们如何区分属于$\Theta(lglg\ n)$算法的散点图和属于$\Theta(lg\ n)$算法的散点图？

7． **a．**基于经验方法，用欧几里得算法计算 gcd(m, n)时，最多要执行多少次除法？其中，$1 \le n \le m \le 100$。

b．k是任意确定的正整数，如果欧几里得算法计算 gcd(m, n)时要进行k次除法运算，请基于经验方法求最小的整数对m, n，其中$1 \le n \le m \le 100$。

8． 欧几里得算法在输入规模为n时的平均效率，是根据算法执行的平均除法次数$D_{avg}(n)$来度量的，$D_{avg}(n)$是 gcd $(n,1)$，gcd $(n, 2)$，…，gcd(n, n)的除法次数的平均值。例如，

$$D_{avg}(5) = \frac{1}{5}(1 + 2 + 3 + 2 + 1) = 1.8$$

画一个 $D_{avg}(n)$的散点图，并指出该算法可能的效率类型。

9． 做一个实验，确定“埃拉托色尼筛选法”的效率类型(参见 1.1 节)。

10． 1.1 节展示了三种计算 gcd (*m*, *n*)的算法，做一个基于时间记录的实验，确定它们的效率类型。

习题 2.6 提示

1． 它是否对所有规模为 2 的数组都返回了正确的比较次数？

2． 先对较小的数组调试你的比较计数和输入随机生成部分。

3． 在一台速度比较快的台式机上，至少对于较小的样本来说，我们测得的时间可能为 0。2.6 节给出了一个克服这个困难的窍门。

4． 当输入规模翻番时，看看计数值增长得有多快。

5． 本节也讨论过一个类似的问题。

6． 对于 $n = 2^k$ 比较函数 lg lg *n* 和 lg *n* 的值。

7． 在该算法的程序实现中插入一个除法计数器，并对指定范围的输入对运行该程序。

8． 得到 *n* 在某个范围(例如，10^2 到 10^4，或者 10^2 到 10^5)的随机值的经验数据，并画出这些得到的数据。(我们应该对坐标系中的坐标轴使用不同的比例尺。)

习题 2.6 答案

1. 若 $A[j] > v$ 失败，该算法不会对这个比较进行计数(while 循环主体不会执行)。如果语言不支持短路求值，即使 while 条件的第一个判断式失败，第二个判断式也会求值，那么应该在 while 语句之前或 while 语句结束之后简单地将计数递增 1。如果支持短路求值，第二个判断式在第一个判断式返回 false 后不会继续求值，就应紧接着 while 语句的末尾添加下面这一行：

$$\textbf{if } j \geq 0 \quad count \leftarrow count + 1$$

2. a. 我们应该期待非常接近$n^2/4$的数字(对随机数组进行插入排序所做的键比较的理论近似次数)。

b. $C(n)/n^2$的比率接近于一个常数，表明平均效率为$\Theta(n^2)$。观察到当数组大小翻倍时，键的比较次数变成四倍，也能得出同样的结论。

c. $C(10\,000)$可估算为$10\,000^2/4$或$4C(5\,000)$。

3. 参见题 2 的答案。但要注意，在精确性和稳定性方面，计时数据天生就比计数数据差。

4. 该数据表现出一种 $n \lg n$ 算法的行为。

5. 如果 $M(n) \approx c \log n$，那么变换$n = a^k (a > 1)$将得到$M(a^k) \approx (c \log a)k$。

6. 函数$lglg\ n$的增长速度比缓慢增长的函数$lg\ n$慢得多。另外，如果通过替换$n = 2^k$来变换这些图，前者的图看起来是对数，而后者的图看起来是线性。

7. a. 9($m = 89$，$n = 55$)

b. 针对每个$k \geq 2$，两个连续的斐波那契数——$m = F_{k+2}$，$n = F_{k+1}$——是需要k

次比较的最小的一对整数($m \geq n > 0$)(这是 G.Lamé建立的一个著名的理论事实，参见[KnuII])。对于$k = 1$，答案是F_{k+1}和F_k，两者均等于 1。

8. 实验应证实了已知的理论结果：欧几里德算法的平均效率是$\Theta(\lg n)$。对于 D.Knuth 研究的稍有不同的指标$T(n)$，$T(n) \approx \frac{12ln2}{\pi^2} \ln n \approx 0.843 \ln n$(参见[KnuII]，4.5.3 节)。

9. n/a

10. n/a

第 3 章

习题 3.1

1. **a .** 请举出不能作为蛮力法的一个算法。

b . 请举出不能用蛮力法解决的一个问题。

2. **a .** 有一个计算 a^n 的蛮力算法，请用 n 的函数来表示它的效率。如果用 n 的二进制位数的函数来表示呢？

b . 如果要计算 $a^n \bmod m$，其中 $a > 1$，并且 n 是一个大于 0 的大整数，如何才能处理好 a^n 值过大的问题？

3. 对于习题 2.3 中 4，5，6 题的每一个算法，请指出它们是不是基于蛮力法的。

4. **a .** 设计一个蛮力算法，对于给定的 x_0，计算下列多项式的值：

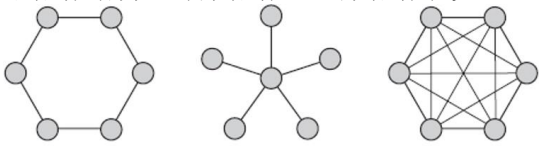
$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

并确定该算法的最差效率类别。

b . 如果你设计的算法属于$\Theta(n^2)$，请为该问题设计一个线性的算法。

c . 对于该问题来说，能不能设计出一个比线性效率还要好的算法呢？

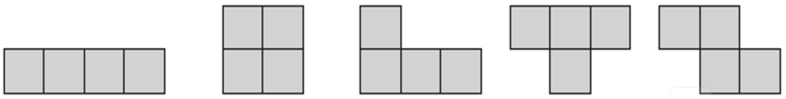
5. 网络拓扑图可以展示计算机、打印机和其他的设备如何通过网络进行互连。以下给出了三种常见的网络拓扑结构：环形拓扑、星形拓扑和完全互连拓扑。



环形　　星形　　完全互连

现在给出一个布尔矩阵 $\boldsymbol{A}[0..n-1, 0..n-1]$，其中 $n > 3$，它用邻接矩阵来表示以上某种拓扑结构所对应的图。能否确定邻接矩阵表示的是哪一种拓扑结构(如果存在的话)？设计一种蛮力算法来解决这个问题并指出它的时间效率类型。

6. **四格拼板** 四格拼板由 4 个 1× 1 的正方形组成。下面是 5 种类型的四格拼板：



直线拼板　　方形拼板　　L 形拼板　　T 形拼板　　Z 形拼板

分别利用以下四格拼板，看看是否有可能在不重叠的情况下完全覆盖一个 8 × 8 的棋盘。

a. 直线拼板　　**b.** 方形拼板

c. L 形拼板　　**d.** T 形拼板

e. Z 形拼板

7. 一摞假币　有 n 摞硬币，每摞包含 n 个硬币，硬币外观完全相同。其中一摞硬币全是假币，而其他摞全是真币。每个真币重 10 克。每个假币重 11 克。你有一架天平，可对任意数量硬币精确称重。

a . 设计一个蛮力算法识别那摞假币，并且确定该算法最坏情况的效率类型。

b . 要识别出那摞假币，至少需要称重多少次？

8 . 应用选择排序将序列 E, X, A, M, P, L, E 按照字母顺序排序。

9 . 选择排序稳定吗？(1.3 节给出了稳定的排序算法的定义。)

10 . 如果对链表实现选择排序，能不能获得和数组版本相同的$\Theta(n^2)$效率？

11 . 应用冒泡排序将序列 E, X, A, M, P, L, E 按照字母顺序排序。

12 . a . 请证明，如果冒泡排序对列表比较一遍之后没有交换元素的位置，那么这个表已经排好序了，算法也可以停止了。

b . 结合所做的改进，为冒泡排序写一段伪代码。

c . 请证明改进版本的最差效率也是平方级的。

13 . 冒泡排序稳定吗？

14 . 交替放置的碟子　我们有数量为 2n 的一排碟子，n 黑 n 白交替放置：黑、白、黑、白……现在要把黑碟子都放在右边，白碟子都放在左边，但只允许通过互换相邻碟子的位置来实现。为该谜题写个算法，并确定该算法需要执行的换位次数。([Gar99])

习题 3.1 提示

1 . a . 想一想，哪个算法的效率和(或者)复杂性给我们留下了深刻的印象。但这两个特点都不是蛮力算法的特征。

b . 令人吃惊的是，这并不是一个容易回答的问题。这种例子的一个比较好的来源是数学问题(包括你你在中学和大学课程中学到的)。

2 . a . 第一个问题差不多已经在本节中回答了。将答案用二进制位数的函数来表示时，可以使用这两种度量标准的关系公式。

b . 我们是如何计算(ab) mod m 的？

3 . 如果已经做过这些题，会很有帮助。

4．a．最直截了当的算法就是把 x_0 直接代入公式中，它是平方级的。

b．分析一下平方算法做了哪些不必要的操作，从而使我们得出一个更好的算法(线性的)。

c．一个 n 次多项式有多少个系数？一个算法要计算 n 次多项式在任意点的值，能不能不对所有的系数做处理？

5．对于这三种网络拓扑，算法应该分别检查矩阵的什么特征？

6．其中四个问题的答案是“是”。

7．a. 对求解的问题应用蛮力法。

b. 用一次称重即可解决问题。

8．就按照算法来处理给定的输入。(本节中，对另一个输入也这么做过。)

9．虽然大多数基本排序算法都是稳定的，但不要急于给出答案。可以参考 1.3 小节对稳定性给出的一个一般性评论，不过它介绍的稳定性的定义也是有所帮助的。

10．一般来说，如果需要不按照顺序来访问列表元素，实现链表算法时会遇到麻烦。

11．就按照算法来处理给定的输入。(参见本节的另一个例子。)

12．a．当且仅当所有邻接元素的顺序都正确时，列表是有序的。为什么？

b．加入一个布尔标志来记录是否做过交换。

c．先给出一个最差输入。

13．冒泡排序会改变输入中两个相等元素的顺序吗？

14．考虑一下，把这个谜题作为一个排序问题，是不是会得出一个最简单和高效的解法？

习题 3.1 答案

1. a. 本书之前提到了欧几里德算法和将整数转换为二进制形式的标准算法，这些都不算是蛮力法。当然，其他章节还有许多例子。

   b. 非线性方程求解或计算定积分是任何算法都不能精确求解的问题的例子(特殊情况除外)。

2. a. $M(n) = n \approx 2^b$，其中$M(n)$是蛮力算法在计算a^n时所做的乘法次数，b是n的二进制形式中的二进制位数。所以，效率用n的函数来表示是线性的，用b的函数来表示是指数级的。

   b. 所有乘数都模除m，即：

   $$a^i \bmod m = (a^{i-1} \bmod m \cdot a \bmod m) \bmod m，\text{其中} i = 1, ..., n$$

3. 问题 4(计算$\sum_1^n i^2$)：是

   问题 5(计算数组中值的“范围”)：是

   问题 6(验证矩阵是否对称)：是

4. a. 下面是一个最直截了当的版本的伪代码：

算法 *BruteForcePolynomialEvaluation*(*P*[0..*n*], *x*)

//用“从最高项到最低项”的蛮力算法，求多项式 *P* 在给定点 *x* 的值

//输入：n 次多项式的系数数组 $P[0..n]$(从低到高存储)，以及一个数字 x
//输出：多项式在 x 点处的值
$p \leftarrow 0.0$
for $i \leftarrow n$ **downto** 0 **do**
    $power \leftarrow 1$
    for $j \leftarrow 1$ **to** i **do**
        $power \leftarrow power * x$
    $p \leftarrow p + P[i] * power$
return p

我们将用多项式的次数 n 来度量输入规模。这个算法的基本操作是两个数的乘法；乘法次数 $M(n)$只取决于多项式的次数。

虽然求这个算法的乘法总次数并不难，但可以只统计算法最内层循环中的乘法次数来求得算法的效率类别：

$$M(n) = \sum_{i=0}^{n} \sum_{j=1}^{i} 1 = \sum_{i=0}^{n} i == \frac{n(n+1)}{2} \in \Theta(n^2)$$

b. 上述算法的效率非常低：我们一次又一次地重新计算 x 的幂，好像它们之间没有任何关系。所以，最明显的改进是更高效地计算连续幂。如果从最高项到最低项，可以使用x^i来计算x^{i-1}，但这需要除法，而且需要对 $x = 0$ 进行特殊处理。第二个方案是使用乘法而非除法，并且不需要对 $x = 0$ 进行特殊处理，所以它既高效又简洁。算法如下所示：

算法 *BruteForcePolynomialEvaluation*($P[0..n], x$)
//用“从最低项到最高项”的蛮力算法，求多项式 P 在给定点 x 的值
//输入：一个 n 次多项式的系数数组 $P[0..n]$(从低到高存储)，以及一个数字 x
//输出：多项式在 x 点的值
$p \leftarrow P[0]; \quad power \leftarrow 1$
for $i \leftarrow 1$ **to** n **do**
    $power \leftarrow power * x$
    $p \leftarrow p + P[i] * power$
return p

所做的乘法次数是：

$$M(n) = \sum_{i=1}^{n} 2 = 2n$$

(同时加法次数是 n)；换言之，我们得到一个线性算法。

注：6.5 节讨论的霍纳法则只需要 n 次乘法(和 n 次加法)来解决这个问题。

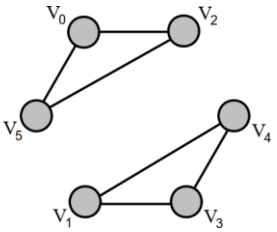
c. 不能，因为对n次多项式在任意点x处的值进行求值的算法都必须处理其所有$n+1$个系数。(注意，即使当$x = 1$时，$p(x) = a_n + a_{n-1} + \cdots + a_1 + a_0$，也需要不折不扣地为任意$a_n, a_{n-1}, \cdots, a_0$至少执行$n$次加法。)

5. 为简单起见，我们分别检查这三种拓扑结构中的每一种。

环形拓扑图的邻接矩阵肯定是对称的，其每一行肯定正好有两个 1，而且都不在主对角线上，以避免自环(loop)。这些必要条件并不充分，因为它们并不能保证图

的连通性，如下例所示。

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$



以下蛮力算法跟踪给定矩阵中的 1，它表示有两条边与一个顶点相连：一条是进入该顶点的边，另一条是离开该顶点的边，确保回路只在访问了图的其他所有顶点后关闭于起始顶点。

先扫描行 0，验证它在我们记为j_1和$j_{n-1}(0 < j_1 < j_{n-1})$的列中是否正好有两个 1。如果不是，则停止：该矩阵不是环形拓扑图的邻接矩阵。如果是，就将列 0 标记为一个已访问的顶点的列，然后继续处理行j_1。$(0 - j_1$是被遍历的回路中的第一条边)。核实$A[j_1, 0] = 1$，并找到该行唯一剩下值为 1 的未标记$j_2 \neq j_1$。如果这是不可能的，则停止；否则，将列j_1标记为一个已访问的顶点的列，并继续处理行j_2。$(j_1 - j_2$是被遍历的回路中的第二条边。)以这种方式继续，直到需要处理和唯一剩下的未标记顶点对应的行。这必然是行j_{n-1}，其中j_{n-1}是在算法的第一次迭代中找到的。行j_{n-1}的两个 1 必然在列j_{n-2}(从它对应的顶点到达顶点j_{n-1})和列 0(以关闭回路)中。

该算法的时间效率为$O(n^2)$，因为在最坏情况下，它要检查$n \times n$矩阵的所有元素。

注意，

不难证明，当且仅当一个图的所有顶点的度(degree，相邻顶点的数量)为 2，同时无环，而且连通，那么该图就具有环形拓扑结构。所以，为了解决这个问题，可以直接对第一个条件进行检查，并通过两种标准算法之一来检查图的连通性：深度优先查找或广度优先查找，这在本书的 3.5 节中有讨论。上述算法实际上模仿了这两种算法中的第一种。

星形拓扑图的邻接矩阵只包含 0，只有除$A[j_0, j_0,]$(主对角线上的元素)之外的某些行j_0和列j_0全都包含 1。所以，蛮力算法可以先扫描行 0，检查它是否除列 0 之外都包含 1，或者是否在某列$j_0 > 0$上包含单个 1。

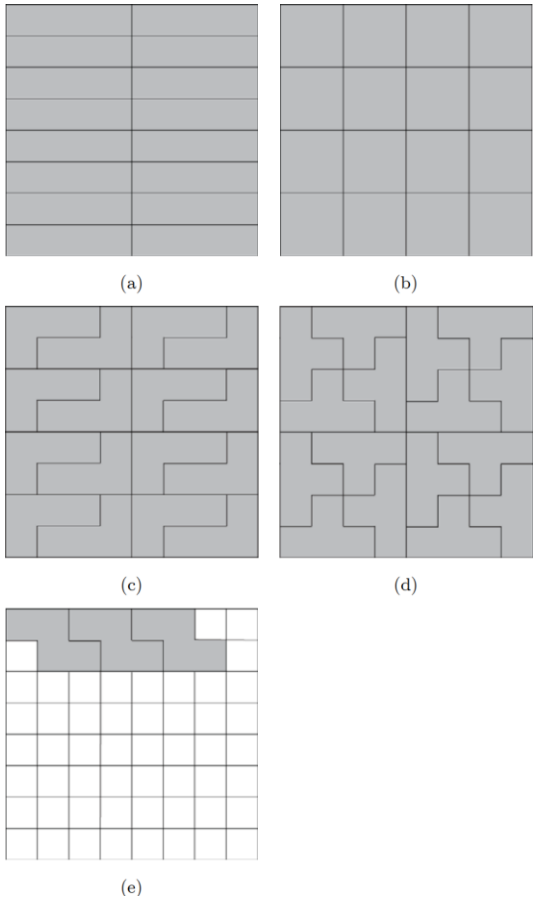
如果是前一种情况，后续行$i = 1,2, \dots, n - 1$肯定在列 0 包含单个 1。换言之，对于$j = 1, \dots, n - 1$，$A[i, 0] = 1$且$A[i, j] = 0$。

如果是后一种情况，针对后续每一行$i = 1,2, \dots, n - 1$，除了$i = j_0$之外，都检查该行是否具有与行 0 同样的性质。换言之，对于$j = 0,1, , \dots, n - 1(j \neq j_0)$，$A[i, j_0] = 1$且$A[i, j] = 0$。而行$j_0$除了主对角线上的元素之外肯定只包含 1。换言之，对于$j = 0,1, , \dots, n - 1(j \neq j_0)$，$A[j_0, j_0] = 0$且$A[j_0, j] = 1$。显然，该算法的时间效率也是$O(n^2)$。

最后，对于具有完全互联(网状)拓扑结构的图，它的邻接矩阵除了主对角线上的 0，其他都是 1。扫描矩阵的行(或列)来进行核实，也需要$O(n^2)$时间。

6. 下面是 8×8 棋盘上的直线拼板、方形拼板、L 形拼板和 T 形拼板的平局。用 Z 形拼板覆盖 8×8 的棋盘是不可能的。事实上，用这样的拼板来覆盖棋盘的一个角，

就必须沿棋盘边继续放两个拼板，这样就不可能覆盖第一行剩下的两个正方形。



使用以下形状的拼板覆盖棋盘：(a)直线；(b)方形；(c)L 形；(d)T 形；(e)Z 形(这个做不到)

注意：用四格拼板(和其他类型的多格拼板，或称“多格骨牌”)来拼图的问题由其发明人 S.W.Golomb 在其专著“Polyominoes: Puzzles, Patterns, Problems, and Packings”(Princeton University Press，1994)进行了讨论，并在第二版中进行了修订和扩充。

7. a. 对每一摞硬币进行编号，从 1 到 n。从第一摞开始，重复以下操作。如果当前这摞硬币不是最后一摞，从其中任取一枚硬币并称重：如果重 11 克，这一摞包含假币，算法停止；如果重 10 克，继续下一摞。如果到达最后一摞，它一定是有假币的那一摞，它的硬币都不需要称量。在最坏情况下(假币在最后一摞)，该算法将进行 $n-1$ 次称重，这使它的时间效率属于$\Theta(n)$类别。

b. 一次称重即可解决。对每一摞硬币进行编号，从 1 到 n。从第一摞取一枚硬币，从第二摞取两个硬币……以此类推，直到从最后一摞取所有硬币。将所有这些硬币放到一起称量。这个重量与 $10n(n+1)/2=5n(n+1)$——即$(1+2+...+n)=n(n+1)/2$ 枚真币的重量——的差值即为参与称重的假币的数量，也就是假币那一摞的编号。例

如，假定 n = 10，所选硬币重 553 克，那么有 3 个硬币是假的，所以第三摞含有假币。

注：一次称重就发现哪一摞是假币是一个著名的谜题，它被列入许多脑筋急转弯问题集。

8.

$$
\begin{array}{l|llllll}
E & X & \mathbf{A} & M & P & L & E \\
A & X & \mathbf{E} & M & P & L & E \\
A & E & X & M & P & L & \mathbf{E} \\
A & E & E & M & P & \mathbf{L} & X \\
A & E & E & L & P & \mathbf{M} & X \\
A & E & E & L & M & \mathbf{P} & X \\
A & E & E & L & M & P & X
\end{array}
$$

9. 选择排序是不稳定的。在交换互不相邻的元素的过程中，该算法可能颠倒相等元素的顺序。列表$2', 2'', 1$就是这样一个例子。

10. 是的。这两种操作——寻找最小元素和交换它——用链表可以和数组一样高效地完成。

11.

$$E, X, A, M, P, L, E$$

$$
\begin{array}{lllllllllll}
E & \overset{?}{\leftrightarrow} & X & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E \\
E & & A & & X & \overset{?}{\leftrightarrow} & M & & P & & L & & E \\
E & & A & & M & & X & \overset{?}{\leftrightarrow} & P & & L & & E \\
E & & A & & M & & P & & X & \overset{?}{\leftrightarrow} & L & & E \\
E & & A & & M & & P & & L & & X & \overset{?}{\leftrightarrow} & E \\
E & & A & & M & & P & & L & & E & & |X \\
E & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E & & \\
A & & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & P & \overset{?}{\leftrightarrow} & L & & E & & \\
A & & E & & M & & L & & P & \overset{?}{\leftrightarrow} & E & & \\
A & & E & & M & & L & & E & & |P & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & L & & E & & & & \\
A & & E & & L & & M & \overset{?}{\leftrightarrow} & E & & & & \\
A & & E & & L & & E & & |M & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & \overset{?}{\leftrightarrow} & E & & & & & & \\
A & & E & & E & & |L & & & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & & & & & &
\end{array}
$$

算法在这里就可以停止了(参见下一题)。

12. a. 冒泡排序的的第 i $(0 \leqslant i \leqslant n-2)$遍比较可用下图表示：

$$A_0, ..., A_j \overset{?}{\leftrightarrow} A_{j+1}, ..., A_{n-i-1} \leq | \underbrace{A_{n-i} \leq \cdots \leq A_{n-1}}_{\text{已经位于它们最终的位置上}}$$

如果这一遍没有发生交换，那么：

$$A_0 \leq A_1 \leq \cdots \leq A_j \leq A_{j+1} \leq \cdots \leq A_{n-i-1}$$

其中，位置 $n-i$ 到 $n-1$ 处的较大(更准确地说是“非较小”)元素已经在之前的迭

代中排好序了。

b. 以下是冒泡排序改进版本的伪代码：

算法 *BetterBubbleSort* (*A*[0..*n* – 1])

//该算法用改进的冒泡排序法对数组 *A*[0..*n* – 1]进行排序

//输入：一个可排序数组 *A*[0..*n* – 1]

//输出：升序排列的数组 *A*[0..*n* – 1]

```
count ← n − 1 //要比较的相邻对的数量
sflag ← true  //交换标志
while sflag do
    sflag ← false
    for j ← 0 to count−1 do
        if A[j + 1] < A[j]
            swap A[j] and A[j + 1]
            sflag ← true
    count ← count −1
```

c. 最坏情况下，输入的数组是严格按降序排列的。对于这种输入，改进后的版本和原始版本所做的比较次数一样，即平方级的效率。

13. 冒泡排序是稳定的。因为只要 $A[j+1] < A[j]$，它就只交换相邻元素。

14. 下面是这个问题的一个简单而高效(事实上是最佳)的算法：从第一个到最后一个碟子，将其与左侧 $i(1 \leqslant i \leqslant n)$个碟子中的每一个进行交换。该算法的第 i 次迭代可以用下图来说明，其中 1 和 0 分别对应黑碟子和白碟子。

$$\underbrace{00..0}_{i-1}\underbrace{11..1}_{i-1}\mathbf{1}0..10 \quad \Rightarrow \quad \underbrace{00..0}_{i}\mathbf{0}\underbrace{11..1}_{i}10..10$$

总的交换次数是$\sum_{i=1}^{n} i = n(n+1)/2$。

这个问题也可通过模仿冒泡排序在对分别代表黑白碟子的 1 和 0 所构成的数组进行排序时的交换来解决。

习题 3.2

1． 求哨兵版的顺序查找算法的比较次数：

a．在最差情况下。

b．在平均情况下。假设成功查找的概率是 $p(0 \leqslant p \leqslant 1)$。

2． 如 2.1 节所述，顺序查找算法的平均键比较次数(没有用哨兵，并且输入满足标准假设)由下式给出：

$$C_{avg}(n) = \frac{p(n+1)}{2} + n(1-p)$$

其中，p是成功查找的概率。如果n已知，$p(0 \leqslant p \leqslant 1)$为何值时 $C_{avg}(n)$的值最大？p 为何值时 $C_{avg}(n)$的值最小？

3． **装置测试** 某公司想要测试一种装置从 n 层总部大楼最高哪一层坠地时不会摔坏。公司有两个一模一样的装置来进行测试。两个中的任何一个受损，它将无法修复，试验不得不用剩下的那个装置完成。设计一个具有最佳效率类型的算法

来帮助该公司解决问题。

4． 如果要在下面的文本中查找模式“GANDHI”(甘地)，请确定蛮力算法将要执行的字符比较的次数。

THERE_IS_MORE_TO_LIFE_THAN_INCREASING_ITS_SPEED

假设查找之前已知文本的长度(本例是 47 个字符)。

5． 用蛮力字符串匹配算法在由 1000 个 0 组成的二进制文本中查找下列模式需要做多少次比较(包括成功的和不成功的)?

a．00001　　**b．**10000　　**c．**01010

6． 给出一个长度为n的文本和长度为m的模式构成的实例，它是蛮力字符串匹配算法的一个最差输入。请确切指出，对于这样的输入需要做多少次字符比较运算。

7． 在解决字符串匹配问题时，将模式和文本中的字符从右向左比较会不会比从左向右比较更有优势呢?

8． 有这样一个问题：在一段给定的文本中查找以 A 开始、以 B 结尾的子串的数量(例如，在 CABAAXBYA 中有 4 个这样的子串)。

a．为该问题设计一个蛮力算法并确定它的效率类型。

b．为该问题设计一个更高效的算法。([Gin04])

9． 为蛮力字符串匹配算法写一段可视化程序。

10．填字游戏　“填字”(称为 word find 或 word search)是在美国流行的一种游戏，它要求游戏者从一张填满字母的正方形表中，找出包含在一个给定集合中的所有词。这些词可以竖着读(向上或向下)，横着读(从左或从右)，或者沿 45°对角线斜着读(4 个方向都可以)，但这些词必须是由表格中邻接的连续单元格构成。遇到表格的边界时可以环绕，但方向不得改变，也不能折来折去。表格中的同一单元格可以出现在不同的词中，但在任一词中，同一单元格不得出现一次以上。为该游戏设计一个计算机程序。

11．海战游戏　基于蛮力模式匹配，在计算机上编程实现“海战”游戏。游戏的规则如下：游戏中有两个对手(在这里，分别是玩家和计算机)，游戏是在两块完全相同的棋盘(10×10 的方格)上进行的，两个对手分别在各自的棋盘上放置他们的舰艇，当然对手是看不见的。每一个对手都有 5 艘舰艇：一艘驱逐舰(2 格)、一艘潜艇(3 格)、一艘巡洋舰(3 格)、一艘战列舰(4 格)和一艘航空母舰(5 格)。每艘舰艇都在棋盘上占据一定数量的格子。每艘舰艇既可以竖着放，也可以横着放，但任意两艘舰艇不能互相接触。游戏的玩法是双方轮流“轰炸”对方的舰艇。每次轰炸的结果是击中还是未击中都会显示出来。如果击中的话，该玩家就可以继续攻击，直到击不中为止。游戏的目标是赶在对手之前把他所有的舰艇都击沉。要击沉一艘舰艇，该舰艇的所有格子都必须被命中。

习题 3.2 提示

1． 对 2.1 节算法版本的分析进行修改。
2． 作为 p 的函数，C_{avg} 是哪种类型的函数？
3． 先求解只用一个小仪器的较简单问题。然后为使用两个小仪器的版本设计一个算法，该算法的效率要比线性更好。
4． 圣雄甘地这段话的含义要比这道题更发人深省。
5． 对于每个输入，只要做一次迭代就可以给出回答问题所需的全部信息。
6． 只要找一个二进制文本和模式的例子就足够了。
7． 令人惊讶的是，该问题的答案是肯定的。
8． **a．**对于文本中 A 的一个给定匹配，应该查找什么样的子串？
b．对于文本中 B 的一个给定匹配，应该查找什么样的子串？
9． 该可视化程序既可以用位串也可以用自然文本。鼓励大家在给定的文本中查出给定模式的所有匹配。
10．彻底测试你的程序，对于那些斜着读，并在表格边缘折行的词要尤其注意。
11．例如，一个(极其)蛮力的算法只是从棋盘的某个角轰击相邻的可行方格。你能不能给出一个更好的策略呢？(为了研究不同策略的相对效率，可以让两个程序相互战斗。)你的策略是不是比轰击对手棋盘上随机方格的策略更好？

习题 3.2 答案

1. a. $C_{worst}(n) = n + 1$。
   b. $C_{avg}(n) = \frac{(2-p)(n+1)}{2}$。按照和 2.1 节几乎完全相同的分析方式，我们得到：

$$\begin{aligned} C_{avg}(n) &= [1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}] + (n+1) \cdot (1-p) \\ &= \frac{p}{n}[1 + 2 + \ldots + i + \cdots + n] + (n+1)(1-p) \\ &= \frac{p}{n}\frac{n(n+1)}{2} + (n+1)(1-p) = \frac{(2-p)(n+1)}{2} \end{aligned}$$

2. 表达式
   $\frac{p(n+1)}{2} + n(1-p) = p\frac{n+1}{2} + n - np = n - p(n - \frac{n+1}{2}) = n - \frac{n-1}{2}p$
   是 p 的线性函数。由于 p 的系数在 n>1 时为负，所以函数在 0≤p≤1 的区间内从 n 严格递减到(n+1)/2。因此，p=0 和 p=1 分别是它在这个区间内的最大和最小点。(当然，这也是我们应该期待的答案：当搜索成功的概率为 0 时，平均比较次数应该是最大的；而当搜索成功的概率为 1 时，它应该是最小的)。
3. 分别在楼层$\lceil\sqrt{n}\rceil$，$2\lceil\sqrt{n}\rceil$……扔下第一个装置，直至在楼层$i\lceil\sqrt{n}\rceil$扔下装置时发生损坏，或直到顶楼都没有摔坏。如果是前一种情况，目标楼层比$(i-1)\lceil\sqrt{n}\rceil$高，又比$i\lceil\sqrt{n}\rceil$低。所以，分别在楼层$(i-1)\lceil\sqrt{n}\rceil$+1，$(i-1)\lceil\sqrt{n}\rceil$+2……扔下第二个装置，直到装置落地时摔坏的楼层。这个楼层之前的那一个楼层就是我们的答案(坠地不会摔坏的最高楼层)。第一轮尝试时，如果没有任何一次坠地导致装置损坏，那么

目标楼层肯定高于$i\lceil\sqrt{n}\rceil$，即该轮尝试中最后一个尝试的楼层。所以，连续检查楼层$i\lceil\sqrt{n}\rceil$+1，$i\lceil\sqrt{n}\rceil+2\cdots\cdots$，直到装置摔坏，或者到达最后一楼。用两个装置做试验时的坠落次数不超过$\lceil\sqrt{n}\rceil+\lceil\sqrt{n}\rceil$，所以效率类型是$O\lceil\sqrt{n}\rceil$。

4. 43 次比较。

该算法将进行 47−6+1=42 次尝试。在第一次尝试中，模式中的 G 与文本的第一个 T 对齐；在最后一次尝试中，它与最后一个空格(文本中显示为下划线)对齐。除了一次尝试之外，算法每次尝试都会进行一次不成功的比较。在特殊的那一次尝试中，即模式中的 G 直接与文本中的 G 对齐，会做两次比较。所以，字符比较总次数是$41\times1+1\times2=43$。

5. a. 对于模式 00001，该算法将在每次尝试中进行 4 次成功和一次不成功的比较，然后将该模式右移一个位置。

```
0 0 0 0 0 0                                   0 0 0 0 0
0 0 0 0 1
  0 0 0 0 1
              etc.
                                              0 0 0 0 1
```

字符比较总次数是$C=5\times996=4980$次。

b. 对于模式 10000，该算法将在每次尝试中进行一次不成功的比较，然后将该模式右移一个位置。

```
0 0 0 0 0 0                                   0 0 0 0 0
1 0 0 0 0
  1 0 0 0 0
              etc.
                                              1 0 0 0 0
```

字符比较总次数是$C=1\times996=996$次。

c. 对于模式 01010，该算法将在每次尝试中进行一次成功的比较和一次不成功的比较，然后将该模式右移一个位置。

```
0 0 0 0 0 0                                   0 0 0 0 0
0 1 0 1 0
  0 1 0 1 0
              etc.
                                              0 1 0 1 0
```

字符比较总次数是$C=2\times996=1992$次。

6. 由n个零和模式$\underbrace{0\ldots01}_{m-1}$构成的文本是最差输入的一个例子。算法将对这样的输入进行$m(n-m+1)$次字符比较。

7. 对模式和文本字符进行从右到左的比较，可以在发现不匹配后进行更大幅度的模式移位。这正是 7.2 节要讨论的两种字符串匹配算法的基本思路。(作为一个具体的例子，可考虑在一千个 0 的文本中搜索 1111 这个模式)。

8. a. 注意，在文本中某个位置$i(0\le i<n-1)$以 A 开头的目标子串的数量等于该位置右侧 B 的数量。这导致了以下的简单算法。将目标子串的计数初始化为 0。从左向右扫描文本，对除最后一个字符的每个字符做如下处理：如果遇到一个 A，就统计它后面所有 B 的数量，把这个数字加到目标子串计数中。扫描结束后，返

回该计数的最终值。

最坏的情况是文本全由 A 构成，此时的字符比较次数是：

$$n + (n-1) + \cdots + 2 = n(n+1)/2 - 1 \in \Theta(n^2)$$

b. 注意，在文本中某个位置$i(0 < i \leq n-1)$以 B 结尾的目标子串的数量等于该位置左边的 A 的数量。这就导致了以下算法。

将目标子串的计数和遇到的 A 的计数初始化为 0。 从左到右扫描文本，直到文本用完，并做以下处理。如果遇到 A，则递增 A 的计数；如果遇到 B，则将 A 的当前计数加到目标子串计数中。在文本用完后，返回目标子串计数的最终值。

由于该算法只遍历一次给定文本，在每个字符上花费的是恒定的时间，所以该算法是线性的。

9. n/a

10. n/a

11. n/a

习题 3.3

1. 假设在 *BruteForceClosestPoints* 算法中，最内层循环中 sqrt 的运行时间是其他运算的 10 倍，并且其他运算的运算时间相同。如果根据 3.3 节所讨论的方法对算法进行改进，试估算这个算法速度会提升多少。

2 . 对于实线上的n个点 $x_1, x_2, \cdots, x_n$ 的最近对问题，能不能设计出一个比基于蛮力策略的算法更快的算法呢？

3 . 假设$x_1 < x_2 < \cdots < x_n$是实数，它们分别代表坐落在一条直路上的n个村庄的坐标。我们需要在其中一个村庄建一所邮局。

a . 设计一个高效的算法，用来求出邮局的位置，使得各村庄和该邮局之间的平均距离最小。

b . 设计一个高效的算法，用来求出邮局的位置，使得各村庄和该邮局之间的最大距离最小。

4 . **a .** ▷笛卡尔平面上的两个点 $p_1 = (x_1, y_1)$，$p_2 = (x_2, y_2)$之间的距离有多种不同的定义方式。其中，有一种**曼哈顿距离**(Manhattan distance)是这样定义的：

$$d_M(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$$

证明 d_M 满足所有距离函数都肯定满足的下列公理：

i . 对于任意两个点 p_1 和 p_2，$d_M(p_1, p_2) \geqslant 0$，当且仅当 $p_1 = p_2$ 时，$d_M(p_1, p_2) = 0$。

ii . $d_M(p_1, p_2) = d_M(p_2, p_1)$。

iii . 对于任意 p_1，p_2 和 p_3，$d_M(p_1, p_2) \leqslant d_M(p_1, p_3) + d_M(p_3, p_2)$。

b . 在笛卡尔平面上，画出所有与原点(0, 0)的曼哈顿距离等于 1 的点。再对欧几里得距离也做一遍。

c . ▷判断正误：最近对问题的解不依赖于我们用的是哪种度量标准——d_E(欧几里得)还是 d_M(曼哈顿)。

5. 两个等长字符串之间的**汉明距离**(Hamming distance)被定义为：在两个字符串中，相应位置字符不同的个数。这个定义是以理查德·汉明(1915—1998)的名字命名的，他是美国著名的科学家和工程师，他在其关于错误检测码和纠错码的开创性论文里介绍了这个定义。

a．此汉明距离是否满足在问题 4 中列出的关于距离度量的三个公理？

b．假设最近点对问题中的点用一个含 m 个字符的字符串来表示，并且字符串之间的距离通过汉明距离来度量。给出运用蛮力法来解决该最近对问题的时间效率类型。

6． ▷**奇数派游戏**　在操场(欧几里得平面)上有$n \geq 3$个人，每人都有唯一一位最近的邻居。他们手上都拿着一块奶油派，收到信号后，都会把派扔向他最近的邻居。假设n是奇数，而且每个人都扔得很准。请判断对错：每次至少有一个人不会被奶油派击中。([Car79])

7． 最近对问题也可以以k维空间的形式出现，k维空间中的两个点$p' = (x_1', \cdots, x_k')$和$p'' = (x_1'', \cdots, x_k'')$的欧几里得距离是这样定义的：

$$d(p', p'') = \sqrt{\sum_{s=1}^{k} (x_s' - x_s'')^2}$$

对k维空间中的最近对问题，蛮力算法的效率类型是怎样的？

8． 求下列集合的凸包，并指出它们的极点(如果有的话)。

a．线段

b．正方形

c．正方形的边界

d．直线

9． 对于一个平面上$n > 1$个点的集合，设计一个线性效率的算法来求出其凸包的两个极点。

10．为了解决两个以上的点共线问题，需要对凸包问题的蛮力算法做怎样的改动？

11．写一个程序，实现凸包问题的蛮力算法。

12．考虑下面这个线性规划问题的一个小规模的实例：

假定：

$$x + y \leq 4$$
$$x + 3y \leq 6$$
$$x \geq 0,\ y \geq 0$$

求$3x + 5y$的最大值。

a．在笛卡儿平面上，画出该问题的**可行区域**(feasible region)，或者满足问题中所有约束的点的集合。

b．找出该区域的极点。

c．用下面的定理解该最优问题：可行区域有界、非空的线性规划问题总有解，这个解一定位于可行区域的某个极点上。

习题 3.3 提示

1． 可考虑两种不同的答案：在算法的内部循环中不考虑比较和赋值，或反之。

2． 可在 $O(n \log n)$的时间内对 n 个实数排序。

3． **a．**针对 $n=2$ 和 $n=3$来求解这个问题，可能会给你关键的启示。

b．如果邮局不必位于某个给定的村庄，你打算把邮局放在什么地方呢？

4． **a．**使用绝对值的基本特性检验 i 到 iii 的要求。

b．对于曼哈顿距离来说，习题中要求的点是由方程 $|x-0|+|y-0|=1$ 定义的。可以先在坐标系的第一象限描绘这些点(也就是 x，$y \geqslant 0$ 的点)，然后利用它们的对称性描绘其余的点。

c．这个断言是错误的。例如，可以选择 $p_1(0, 0)$，$p_2(1, 0)$，然后寻找 p_3 来作为一个反例。

5． **a．**证明汉明距离确实满足距离度量的三个公理。

b．你的答案应该包括两个参数。

6． 正确。用数学归纳法证明。

7． 我们的答案应该是两个参数的函数：n 和 k。本节已经解决了这个问题的一个特殊情况($k=2$时)。

8． 复习一下本节给出的例子。

9． 凸包的某些极点要比其他极点容易找到。

10．如果集合中的其他点也位于穿越 p_i 和 p_j 的直线上，其中哪些点还要保留下来，用于后面的处理？

11．这个程序应该对于 n 个不同点的任意集合都有效，包括多点共线的集合。

12． **a．**满足不等式 $ax+by \leqslant c$ 的点集合，就是位于直线 $ax+by=c$ 一侧的半平面，包括该直线本身的所有点。画出每一个不等式的半平面，然后找出它们的交集。

b．极点就是 a 中得到的多边形的顶点。

c．计算目标函数在极点上的值并进行比较。

习题 3.3 答案

1. 如果只考虑计算两点之间欧氏距离与计算其平方所涉及的算术运算，将得到以下时间比率的估计(包括算法最内部的循环和整个算法)：$(2 + 3 + 10)/(2 + 3) = 3$

   如果再考虑比较和赋值操作，得到的时间比估计为$(2 + 3 + 10 + 2)/(2 + 3 + 2) \approx 2.4$。

2. 将数字按升序排序，计算有序列表中相邻数字之间的差值，并找出差值最小的一个。如果排序在$O(n\ log\ n)$时间内完成，则整个算法的时间效率类型是：

$$O(n \log n) + \Theta(n) + \Theta(n) = O(n \log n)$$

3. a. 如果将邮局设在位置x_i，它和所有点$x_1 < x_2 < \ldots < x_n$之间的平均距离由公式

$\frac{1}{n}\sum_{j=1}^{n}|x_j-x_i|$给出。由于点的数量$n$保持不变，所以可以忽略多个$\frac{1}{n}$，并使$\sum_{j=1}^{n}|x_j-x_i|$最小化。我们必须分别考虑$n$为偶数和奇数的情况。

如果n是偶数，首先考虑$n=2$的情况。对于这个区间的任何一点x(包括端点)，$|x_1-x|+|x_2-x|$等于x_2-x_1，即端点在x_1和x_2的区间的长度。而且对于这个区间之外的任何一点x，它都大于x_2-x_1。这意味着对于任何偶数n，当x属于$[x_1,x_n]\supset[x_2,x_{n-1}]\supset\ldots\supset[x_{n/2},x_{n/2+1}]$的每个区间时，

$$\sum_{j=1}^{n}|x_j-x|=[|x_1-x|+|x_n-x|]+[|x_2-x|+|x_{n-1}-x|]+\cdots+[|x_{n/2}-x|+|x_{n/2+1}-x|]$$

最小。如果x是给定的点(指定的村庄)之一，那么无论$x_{n/2}$还是$x_{n/2+1}$都能解决该问题

如果n是奇数，那么在$x=x_{\lceil n/2\rceil}$时(在这一点，它的左边给定点的数量等于右边的给定点的数量)，$\sum_{j=1}^{n}|x_j-x|$最小。

注意，$x_{\lceil n/2\rceil}$这个点——第$\lceil n/2\rceil$个最小的数称为中位数——也能解决n为偶数时的问题。对于作为数组来实现的一个有序列表，中位数可以在$\Theta(1)$时间内找到，返回数组的第$\lceil n/2\rceil$个元素即可。(5.6 节对计算中位数的算法进行了更常规的讨论)。

b. 假设点$x_1,x_2,\ldots,x_n$按升序给出，答案是离$m=(x_1+x_n)/2$最近的点x_i，m是x_1和x_n之间的中间点(如邮局不一定要建在给定的村庄，中间点就是明显的解)。事实上，如果将邮局建在m左边的任何位置x_i，那么从一个村庄到邮局的最长距离将是x_n-x_i；这个距离对于这些点中最右边的那个来说是最小的。如果将邮局建在m右边的任何位置x_i，那么从一个村庄到邮局的最长距离将是x_i-x_1；这个距离对于这些点中最左边的那个来说是最小的。

算法 *PostOffice1*(P)

//输入：$n(n\geq 2)$个点$x_1,x_2,\ldots,x_n$的列表P，按升序排列

//输出：在所有点$x_1,x_2,\ldots,x_n$中，使$\underbrace{max}_{1\leq j\leq n}|x_j-x_i|$最小的点$x_i$

$m\leftarrow(x_1+x_n)/2$
$i\leftarrow 1$
while $x_i<m$ **do**
    $i\leftarrow i+1$
if $x_i-x_1<x_n-x_{i-1}$
    return x_i
else return x_{i-1}

该算法的时间效率类型为$O(n)$。

4. a. 对于$d_M(p_1,p_2)=|x_1-x_2|+|y_1-y_2|$，我们这样证明距离公式的公理：

(i) $d_M(p_1,p_2)=|x_1-x_2|+|y_1-y_2|\geq 0$；当且仅当$x_1=x_2$并且$y_1=y_2$时(即$P_1$和$P_2$重合)，$d_M(p_1,p_2)=0$。

(ii) $d_M(p_1,p_2)=|x_1-x_2|+|y_1-y_2|=|x_2-x_1|+|y_2-y_1|=d_M(p_2,p_1)$

(iii) $d_M(p_1,p_2)=|x_1-x_2|+|y_1-y_2|$
$=|(x_1-x_3)+(x_3-x_2)|+|(y_1-y_3)+(y_3-y_2)|$
$\leq|x_1-x_3|+|x_3-x_2|+|y_1-y_3|+|y_3-y_2|=d(p_1,p_3)+d(p_3,p_2)$

b. 对于曼哈顿距离，目标点由以下方程定义：

$$|x-0|+|y-0|=1,\ \text{即}|x|+|y|=1$$

这个方程的图是正方形的边界，其顶点分别是(1，0)，(0，1)，(−1，0)和(−1，−1)。

对于欧几里得距离，目标点由以下方程定义：

$$\sqrt{(x-0)^2+(y-0)^2}=1,\ \text{即}x^2+y^2=1$$

这个方程的图是半径为 1 的圆周，圆心是(0,0)。

c. 错。考虑$p_1(0,0), p_2(1,0)$和$p_3\left(\frac{1}{2},\frac{3}{4}\right)$这三个点：

$$d_E(p_1,p_2)=1 \quad \text{且} \quad d_E(p_3,p_1)=d_E(p_3,p_2)=\sqrt{\left(\frac{1}{2}\right)^2+\left(\frac{3}{4}\right)^2}<1$$

所以，对于欧几里得距离，两个最近的点要么是p_1和p_3，要么是p_2和p_3。而对于曼哈顿距离：

$$d_M(p_1,p_2)=1 \quad \text{且} \quad d_M(p_3,p_1)=d_M(p_3,p_2)=\frac{1}{2}+\frac{3}{4}=\frac{5}{4}>1$$

所以，对于曼哈顿距离，两个最近的点是p_1和p_2。

5. a. 汉明距离显然满足前两个公理，所以只有第三个公理(三角不等式)需要证明。可通过对字符串长度m的数学归纳来得证。如果$m=1$，不等式$d_H(S_1,S_2)\le d_H(S_1,S_3)+d_H(S_3,S_2)$对任何包含单字符的字符串$S_1$，$S_2$和$S_3$都成立：如果$S_1=S_2$，左手边等于 0；如果$S_1\ne S_2$，左手边等于 1，右手边大于或等于 1，因为$S_3$不可能与$S_1$和$S_2$都相同。 对于归纳步骤，假设三角不等式对任何三个长度为m的字符串都成立，并考虑三个任意的字符串$S_i=S_i'c_i$，其中$i=1$，2，3，而且S_i'是长度为m的字符串，c_i是它们的最后一个字符。那么

$$\begin{aligned} d_H(S_1,S_2) &= d_H(S_1',S_2')+d_H(c_1,c_2)\\ &\le d_H(S_1',S_3')+d_H(S_3',S_2')+d_H(c_1,c_3)+d_H(c_3,c_2)\\ &= [d_H(S_1',S_3')+d_H(c_1,c_3)]+[d_H(S_3',S_2')+d_H(c_3,c_2)]\\ &= d_H(S_1,S_3)+d_H(S_3,S_2) \end{aligned}$$

b. 由于该算法的基本操作是比较长度为m的字符串中的两个字符，最坏情况下的时间效率类型是$\Theta(mn^2)$。

6. 可通过数学归纳法证明总有至少一个人不会被派击中。基础步骤很简单：如果$n=3$，两个成对距离最小的人(最近对)互相投掷，第三个人向其中一个人投掷(更接近的那个)。所以，这第三个人将“不受伤害”。

对于归纳步骤，假设该断言对于奇数$n\ge 3$的人来说成立，然后考虑$n+2$个人的情况。同样，找到两人之间距离最小的一对(最近对)互相投掷。

考虑以下两种可能的情况。如果剩下n个人都相互投掷，根据归纳假设，至少有一个人保持“不受伤害”。如果剩下n个人中至少有一个投向最近对中的一个人，那么在剩下的$n-1$个人中，最多只有$n-1$个派被用来相互投掷，所以必然至少有一个人保持“不受伤害”，因为没有足够的派来击中该组中的所有人。这样就完成了证明。

注：该问题来自 L.Carmony 的论文“Odd pie fights”，*Mathematics Teacher*, vol. 72,

no. 1, 1979, 61—64。

7. 开方数是：

$$C(n,k) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{s=1}^{k}1 = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}k = k\sum_{i=1}^{n-1}(n-i)$$
$$= k[(n-1)+(n-2)+\cdots+1] = \frac{k(n-1)n}{2} \in \Theta(kn^2)$$

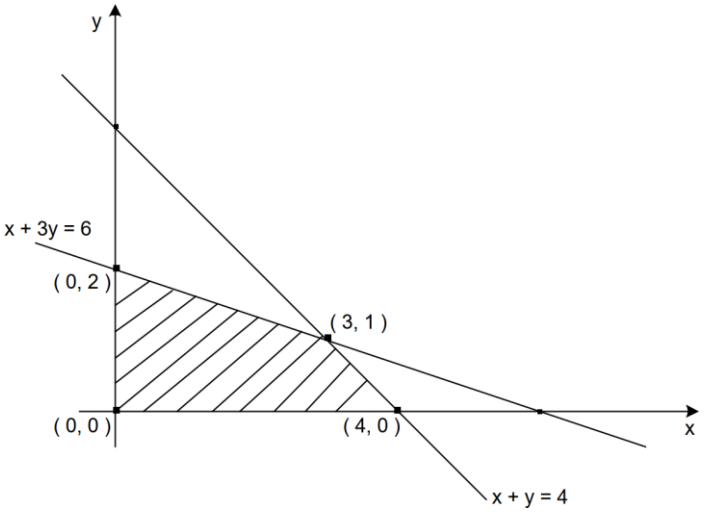
8. a. 线段的凸包是线段本身；其极点是线段的端点。
   b. 正方形的凸包是正方形本身；其极点是正方形的四个顶点。
   c. 正方形的边界的凸包是由该边界内和边界本身的点组成的区域；其极点是正方形的四个顶点。
   d. 直线的凸包就是直线本身。它没有任何极点。

9. 找到x坐标最小的那个点；如果有几个这样的点，找到其中y坐标最小的那个。类似地，找到x坐标最大的那个点；如果有几个这样的点，找到其中y坐标最大的那个。(注意，在给定点的列表中，在一次遍历中寻找最小和最大x坐标会更有效率。虽然这样不会改变算法的线性效率类别，但可以将比较总次数减少至$1.5n$左右)

10. 如果在通过p_i和p_j的直线上存在给定集合中的其他点(该集合的其他所有点都位于直线的同一侧)，那么凸包边界的一个线段的端点将位于集合在直线上最远的两个点。直线上其他点所有都可在后续处理中剔除。

11. n/a

12. a. 下面是目标可行区域的一个草图：



   b. 极点是：$(0,0)$，$(4,0)$，$(3,1)$和$(0,2)$。

   c.

极 点	$3x+5y$ 的值
(0,0)	0
(4,0)	12
(3,1)	14
(0,2)	10

所以，最优解是(3，1)，$3x + 5y$的最大值等于 14。(注：该线性规划问题的实例将在 10.1 节进一步讨论)。

习题 3.4

1． **a．**假设每一条旅行路线都能在固定时间内生成，对于书中描述的旅行商问题的穷举查找算法来说，它的效率类型是怎样的？

b．如果该算法的实现运行在一台每秒能做 10 亿次加法的计算机上，请估计在下述时间中，该算法能处理的城市个数。

i．1 小时　　**ii．**24 小时　　**iii．**1 年　　**iv．**100 年

2． 概要描述一个解决哈密顿回路问题的穷举查找算法。

3． 概要描述一个算法，判断一个用邻接矩阵表示的连通图是否存在欧拉回路。该算法的效率类型是怎样的？

4． 应用穷举查找，把书中开了一个头的分配问题的实例补充完整。

5． 给出一个分配问题的例子，在它的最优解中，不包含其成本矩阵的最小元素。

6． 请考虑**划分问题**(partition problem)：给定n个正整数，把它们划分为元素之和相同但不相交的两个子集。(当然，这种问题并不总是有解的。)为该问题设计一个穷举查找算法。应尽量减少该算法需要生成的子集的数量。

7． 请考虑**完备子图问题**(clique problem)：给定一个图 G 和一个正整数k，确定该图是否包含一个大小为 k 的完备子图，也就是说，一个具有k个节点的完全子图。为该问题设计一个穷举查找算法。

8． 解释一下如何对排序问题应用穷举查找，并确定这种算法的效率类型。

9． **八皇后问题**　这是一个经典游戏：在一个 8×8 的棋盘上，摆放 8 个皇后使任意两个皇后不在同一行、同一列或同一斜线上。对下列各种情形，各有多少种不同的摆放方法？

a．任意两个皇后不在同一格。

b．任意两个皇后不在同一行。

c．任意两个皇后不在同一行或同一列。

同时估计在每秒能检查 100 亿个位置的计算机上，穷举查找方法针对上述各种情形找到问题的所有解需要多少时间。

10． **幻方问题**　n阶幻方是把从 1 到 n^2 的整数填入一个n阶方阵，每个整数只出现一次，使得每一行、每一列、每一条主对角线上各数之和都相等。

a．证明：如果n阶幻方存在的话，所讨论的这个和一定等于 $n(n^2+1)/2$。

b．设计一个穷举算法，生成阶数为n的所有幻方。

c．在互联网或在图书馆查找一个更好的生成幻方的算法。

d．实现这两个算法——穷举查找算法以及在互联网上找到的算法。在自己的计算机上做试验，确定每种算法在一分钟之内能找到的最大幻方阶数 n

11．字母算术　有一种称为**密码算术**(cryptarithm)的算式谜题，它的算式(例如加法算式)中，所有数字都被字母代替。如果该算式中的单词是有意义的，那么这种算式被称为**字母算术题**(alphametic，也称为覆面算)。最著名的字母算术是由大名鼎鼎的英国谜题大师亨利·E. 杜德尼(Henry E. Dudeney，1857—1930)给出的：

$$\begin{array}{r} \text{S E N D} \\ +\ \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

这里有两个前提假设：第一，字母和十进制数字之间是一一对应关系，也就是说，每个字母只代表一个数字，而且不同的字母代表不同的数字；第二，数字 0 不出现在任何数的最左边。求解一个字母算术意味着找到每个字母代表的是哪个数字。请注意，解可能并不是唯一的，不同人的解可能并不相同。

a．写一个程序用穷举查找解密码算术谜题。假设给定的算式是两个单词的加法算式。

b．杜德尼的谜题发表于 1924 年，用你认为合理的方法解该谜题。

习题 3.4 提示

1．a．找出该算法的基本操作并计算它的执行次数。

b．对于给定的每一段时间，计算不会超时的 n 的最大值。

2．旅行商问题和哈密顿回路问题有多少差别？

3．有一个著名的充要条件可以判断连通图中是否存在欧拉回路，我们的算法应该检验这个条件是否成立。

4．生成余下的 $4!-6=18$ 次分配，计算它们的成本，找出其中成本最小的一种方案。

5．所给出的反例的规模要尽可能小。

6．将这个问题换一种说法，使得每次尝试划分时，不需要检查两个子集的元素和，而只需要检查一个子集的元素和即可。

7．遵循完备子图和穷举查找算法的定义。

8．尝试给定元素的所有序列。

9．使用初级排列组合的通用公式。

10．a．用两种不同的方法将魔方的所有元素相加。

b．在这里，我们需要生成什么样的组合对象？

11．a．测试时，可以从网上搜集一些字母算式。

b．1924 年还没有电子计算机，解题时也无法去网上搜索。

习题 3.4 答案

1. a. $\Theta(n!)$

   对于每个旅程($n+1$个城市的序列)，需要做n次加法来计算旅程的长度。因此，加法总次数$A(n)$是所考虑的旅程总数的n倍，即$n*\frac{1}{2}(n-1)!=\frac{1}{2}n!\in\Theta(n!)$。

   b. (i)$n_{max}=16$；(ii) $n_{max}=17$；(iii) $n_{max}=19$；(iv) $n_{max}=21$。

   基于 a 部分的答案，我们必须找到n的最大值，使：

$$\frac{1}{2}n!10^{-10}\leq t$$

   其中，t是可用的时间(以秒为单位)。所以，对于$t=1$ 小时 $=3.6*10^3$秒。

我们得到以下不等式：

$$n!\leq 2*10^{10}t=7.2*10^{13}$$

   这个不等式成立的最大n值是 16(因为$16!\approx 2.1*10^{13}$，而$17!\approx 3.6*10^{14}$)。

   对于其他给定的t值，可用同样的方式得出答案。

2. 寻找哈密顿回路的问题与旅行商问题非常相似。生成n个顶点的排列组合，以某个顶点(例如第一个顶点)为起点和终点，并检查当前排列组合中每一对连续的顶点是否由一条边连接。如果是，当前排列组合就代表一个哈密尔顿电路，否则就需要生成下一个排列组合。

3. 当且仅当连通图的所有顶点的度(degree，相邻节点的数量)都为偶数时，它就存在一个欧拉回路。算法应检查这个条件，直到遇到一个奇数顶点(那么欧拉回路不存在)或者所有顶点都是偶数(那么欧拉回路一定存在)。对于一个由$n\times n$邻接矩阵表示的图(无环)，顶点的度是该顶点对应行中的 1 的数量。因此，计算其度数将花费$\Theta(n)$时间，检查其是否为偶数将花费$\Theta(1)$时间，并且将在1到n次之间完成。所以，该算法的效率将是$\Theta(n^2)$。

4. 本章正文(3.4.3 节)生成了以下分配方式：

$$C=\begin{bmatrix}9&2&7&8\\6&4&3&7\\5&8&1&8\\7&6&9&4\end{bmatrix}\quad\begin{matrix}<1,\ 2,\ 3,\ 4>\\<1,\ 2,\ 4,\ 3>\\<1,\ 3,\ 2,\ 4>\\<1,\ 3,\ 4,\ 2>\\<1,\ 4,\ 2,\ 3>\\<1,\ 4,\ 3,\ 2>\end{matrix}\quad\begin{matrix}成本=9+4+1+4=18\\成本=9+4+8+9=30\\成本=9+3+8+4=24\\成本=9+3+8+6=26\\成本=9+7+8+9=33\\成本=9+7+1+6=23\end{matrix}$$

   等

   剩下的是：

2, 1, 3, 4　　成本 = 2+6+1+4 = 13
2, 1, 4, 3　　成本 = 2+6+8+9 = 25
2, 3, 1, 4　　成本 = 2+3+5+4 = 14
2, 3, 4, 1　　成本 = 2+3+8+7 = 20
2, 4, 1, 3　　成本 = 2+7+5+9 = 23
2, 4, 3, 1　　成本 = 2+7+1+7 = 17

3, 1, 2, 4　　成本 = 7+6+8+4 = 25
3, 1, 4, 2　　成本 = 7+6+8+6 = 27
3, 2, 1, 4　　成本 = 7+4+5+4 = 20
3, 2, 4, 1　　成本 = 7+4+8+7 = 26
3, 4, 1, 2　　成本 = 7+7+5+6 = 25
3, 4, 2, 1　　成本 = 7+7+8+7 = 29

4, 1, 2, 3　　成本 = 8+6+8+9 = 31
4, 1, 3, 2　　成本 = 8+6+1+6 = 21
4, 2, 1, 3　　成本 = 8+4+5+9 = 26
4, 2, 3, 1　　成本 = 8+4+1+7 = 20
4, 3, 1, 2　　成本 = 8+3+5+6 = 22
4, 3, 2, 1　　成本 = 8+3+8+7 = 26

最优解是：任务 2 分配给人员 1，任务 1 分配给人员 2，任务 3 分配给人员 3，任务 4 则分配给人员 4，此时的总成本(最小值)为 13。

5. 下面是一个非常简单的例子：

$$\begin{bmatrix} 1 & 2 \\ 2 & 9 \end{bmatrix}$$

6. 首先计算给定数字的总和S。如S是奇数，则停止，因为这个问题无解。如S是偶数，则生成子集，直到遇到一个元素之和等于$S/2$的子集，或者没有更多子集了。注意，只生成元素数量不超过$n/2$的子集就足够了。
7. 生成一个 k 个顶点的子集，并检查子集中的每一对顶点是否由一条边连接。如果是，则停止(该子集是完备子图)；否则，生成下一个子集。
8. 生成给定元素的一个排列，通过比较其连续元素的值，检查它们是否按要求排序。如果是，则停止；否则，生成下一个排列。由于n项总共有$n!$种排列，而检查一个排列最多需要$n-1$次比较，所以该算法的效率类型为$O(n!\,(n-1)) = O((n+1)!)$。
9. 在 8×8 的棋盘上，八皇后不同位置的数量等于：
   a. 如果没有两个皇后同一格，$C(64,8) = 4\,426\,165\,368$。
   b. 如果没有两个皇后同一行，$8^8 = 16\,777\,216$。
   c. 如果没有两个皇后同一行或同一列，$8! = 40\,320$。
10. a. 设s是$n \times n$幻方(平常所说的“魔方”是幻方的立体版本)中每一行的数字之和。将第 1 行到第n行的所有数字相加，得到以下等式：

$$sn = 1+2+\cdots+n^2, \text{即 } sn = \frac{n^2(n^2+1)}{2}, \text{或者 } s = \frac{n(n^2+1)}{2}$$

   b. 将$n \times n$矩阵中的位置从 1 到n^2编号。生成数字 1~n^2的一个排列，把它们放到

矩阵中相应的位置上。然后，针对矩阵的每一行、每一列以及两条主对角线的每一个位置检查幻方的相等性(已在 a 部分中证明)。

c. n/a

d. n/a

11. a. 由于字母与数字肯定一一对应，而且只有 10 个不同的十进制数字，所以穷举查找需要检查$P(10，k) = 10!/(10-k)!$种可能的替换，其中k是输入中各不相同的字母的数量。(单词首字母不能为 0，这一要求可进一步减少该数量)。所以，对于今天的计算机来说，一个程序的运行时间应该处于相当合理的区间。注意，程序不需要为每个数字位置检查两种情况——有“进 1”和无“进 1”，而只需检查基于十进制数字系统定义的一个等式。以杜德尼的字母表为例，这个等式是 1000(S+M) + 100(E+O) + 10(N+R) + (D+E) = 10000M + 1000O + 100N + 10E + Y。

b. 下面是这个经典问题的一个不依赖计算机的解法。首先，注意 M 肯定为 1。(S 和 M 都不大于 9，所以它们的和，即使从“百位”进了 1，也必然小于 20)。接着，我们不得不研究该问题的一些细节来进一步思考怎么解题。两个加数最左边的数字意味着两种可能性之一：要么是 S + M = 10 + O(如果没有从百位进 1)，要么是 1 + S + M = 10 + O(如果从百位进 1)。首先看看这两种可能性中的前一种。由于 M=1，S≤9，O≥0，方程 S + 1 = 10 + O 只有一个解：S = 9，O = 0。这样就剩下了：

$$\begin{array}{r} \text{E N D} \\ +\,\text{O R E} \\ \hline \text{N E Y} \end{array}$$

由于当前处理的不从百位进 1 的情况，而且 E 和 N 肯定各不相同，所以唯一的可能性是从十位进 1：1 + E = N，而且要么 N + R = 10 + E(如果个位没有进 1)，要么 1 + N + R = 10 + E(如果从个位进了 1)。第一种组合导致一个冲突：将 N = 1 + E 代入 N + R = 10 + E，我们得到 R = 9，由于 S 已经代表 9 了，所以出现冲突。第二种组合是 1 + E = N 且 1 + N + R = 10 + E。将第一个等式代入第二个后，我们得到 R = 8。注意，现在唯一剩下未分配的数字只有 2、3、4、5、6 和 7。最后，对于个位数，我们得到等式 D + E = 10 + Y(因为要从个位进 1)。但 10 + Y≥12(因为最小的未分配数字是 2)，同时 D + E ≤ 12(因为最大的两个未分配数字是 6 和 7，且 E < N)。所以，D + E = 10 + Y = 12。换言之，Y = 2，D + E = 12。现在，唯一一对仍未分配的数字是 5 和 7，加起来是 12，它们必须分别分配给 E 和 D，因为如果反过来的话(E=7，D=5)，那么 N = E + 1 = 8，这个数字已被 R 所代表。这个谜题的最终解是：

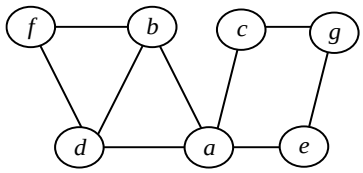
$$\begin{array}{r} 9\,5\,6\,7 \\ +\,1\,0\,8\,5 \\ \hline 1\,0\,6\,5\,2 \end{array}$$

但这是唯一解吗？要回答这个问题，应研究百位到千位的进位可能性(见上文)。如果从百位进 1，那么 1 + S + M = 10 + O。由于 M = 1，所以 S = 8 + O。但 S ≤ 9，而 8 + O ≥ 10(因为 O≥2)。所以，最后一个方程在我们的领域中无解。这就证明

了该谜题没有其他解。

习题 3.5

1． 考虑下图。



a．写出表示这个图的邻接矩阵和邻接链表(假设矩阵的行和列以及邻接链表中的顶点都是按照顶点标签的字母顺序排列的)。

b．从节点a开始用深度优先查找来遍历图，按照字母顺序选择未访问的顶点，并构造相应的深度优先查找树。给出顶点第一次被访问到(压入遍历栈)的顺序以及这些顶点变为终点(出栈)的顺序。

2． 如果定义一个稀疏图，它的$|E| \in O(|V|)$，对于这样的图来说，哪一种 DFS 实现的时间效率更好，是使用邻接矩阵的实现，还是使用邻接链表的实现？

3． 假设 G 是一个有n个顶点和m条边的图，下列说法是对还是错？

a．它的所有 DFS 森林(对应于不同顶点开始的遍历)包含相同数量的树。

b．它的所有 DFS 森林包含相同数量的树向边和回边。

4． 使用广度优先查找对第 1 题中的图进行遍历，并构造相应的广度优先查找树。从顶点a开始遍历，在遇到多个可选顶点时，根据字母顺序来选择。

5． 请证明，在无向图中，BFS 树的交叉边要么连接同层的顶点，要么连接 BFS 树中两个相邻层的顶点。

6． **a．**请解释一下如何使用广度优先查找来检查图的无环性。

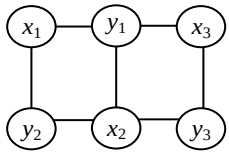
b．在 DFS 遍历和 BFS 遍历中，是不是某种方法总是比另一种方法更快地找到回路？如果你认为是，请指出哪一种遍历更快并解释原因；如果回答否，请给出支持你的观点的两个例子。

7． 请说明如何使用以下两种方法求得一个图的连通分量：

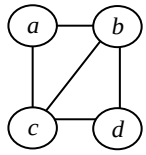
a．深度优先查找

b．广度优先查找

8． 如果图中的顶点可以分为两个不相交的子集X和Y，使得每条连接X中顶点的边都连接着Y中的顶点，这样的图是**二分**(bipartite)**图**。也可以这样说：如果只用两种颜色对顶点着色，就能使得每条边上的两个顶点不同色，这样的图是二分图，也称为**二色**(2-colorable)**图**。例如，图(i)是二分图，图(ii)则不是。



(i)



(ii)

a．设计一个基于 DFS 的算法来检查一个图是不是二分图。

b．设计一个基于 BFS 的算法来检查一个图是不是二分图。

9． 设计一个程序，对于一个给定的图，它能够输出

a．每一个连通分量的顶点；

b．图的回路，或者返回一个消息表明图是无环的。

10． 可用一个代表起点的顶点、一个代表终点的顶点、若干个代表死胡同和通道的顶点对迷宫进行建模，迷宫中的通道不止一条，我们必须求出连接起点和终点的迷宫道路。

a．为下面的迷宫构造一个图。

b．如果发现自己身处一个迷宫，你会选择 DFS 遍历还是 BFS 遍历？为什么？

11．三壶问题 西蒙·丹尼斯·泊松(Siméon Denis Poisson，1781—1840)是著名的法国数学家和物理学家。据说在他遇到某个古老的谜题之后，就开始对数学感兴趣了，这个谜题是这样的：给定一个装满水的 8 品脱壶以及两个容量分别为 5 品脱和 3 品脱的空壶，如何通过完全灌满或者倒空这些壶从而使得某个壶精确地装有 4 品脱的水？用广度优先查找来求解这个谜题。

习题 3.5 提示

1． **a．**使用 1.4 节给出的邻接矩阵和邻接链表的定义。

b．按照课本中对另一个图执行 DFS 遍历的同样方式进行(参见图 3.10)。

2． 比较对稀疏图进行 DFS 遍历的两个不同版本的效率类型。

3． **a．**这样的树的数量等于多少？

b．先对连通图回答这个问题。

4． 按照课本中对另一个图执行 BFS 遍历的同样方式进行(参见图 3.11)。

5． 可以利用这样一个事实：一个顶点在 BFS 树中的层数代表从根到该顶点的最短路径(最少边)中的边数。

6． **a．**BFS 森林的哪种特性能够指出图中存在一个回路(这个答案和 DFS 森林的答案类似)?

b．答案是否。找两个支持此答案的例子。

7． 事实是，对于这两种遍历来说，当且仅当新的顶点和前面访问的顶点邻接时，它才能被访问到。当这两种遍历停止(也就是说，栈或队列为空)时，什么样的顶点已经被访问过了?

8． 对 a 和 b 分别使用一个 DFS 森林和一个 BFS 森林。

9． 使用 DFS 森林或者 BFS 森林。

10． **a．**遵循问题定义中的指导。

b．两种遍历都尝试一遍，会让你迅速得出结论。

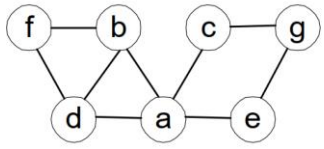
11．直接应用 BFS，无需画出表示谜题状态的草图。

习题 3.5 答案

1. a. 下页是该图的邻接矩阵和邻接链表：

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \qquad \begin{array}{|c|l} \hline a & \rightarrow b \rightarrow c \rightarrow d \rightarrow e \\ \hline b & \rightarrow a \rightarrow d \rightarrow f \\ \hline c & \rightarrow a \rightarrow g \\ \hline d & \rightarrow a \rightarrow b \rightarrow f \\ \hline e & \rightarrow a \rightarrow g \\ \hline f & \rightarrow b \rightarrow d \\ \hline g & \rightarrow c \rightarrow e \\ \hline \end{array}$$

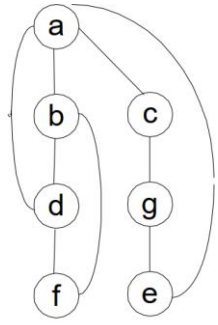
b. 如下所示：(i) 图；(ii) 遍历栈。第一个下标表示顶点的访问顺序，即入栈顺序，第二个下标表示它成为终点(dead end)的顺序，即出栈顺序；(iii)DFS 树(树向边用实线表示，树回边用虚线表示)。



(i)

$f_{4,1}$ $e_{7,4}$
$d_{3,2}$ $g_{6,5}$
$b_{2,3}$ $c_{5,6}$
$a_{1,7}$

(ii)



(iii)

2. DFS 的邻接矩阵的时间效率是$\Theta(|V|^2)$，邻接链表的时间效率是$\Theta(|V| + |E|)$。如$|E| \in O(|V|)$，前者还是$\Theta(|V|^2)$，后者则变成$\Theta(|V|)$。所以，对于稀疏图来说，DFS 的邻接链表版本比邻接矩阵版本更高效。

3. a. DFS 树的数量等于图的连通分量的数量。所以，对于图的所有 DFS 遍历，这个数量都是相同的。

b. 对于有$|V|$个顶点的连通(无向)图，DFS 树中的向边数量$|E^{tree}|$是$|V| - 1$。所以，

回边数量$|E^{back}|$是向边总数减去树向边数，即$|E|-(|V|-1)\ =\ |E|-|V|+1$。因此，它将独立于对同一个图的一次特定的 DFS 遍历。这一观察结果可扩展到具有$|C|$个连通分量的任意图，将这一推理应用于其每个连通分量：

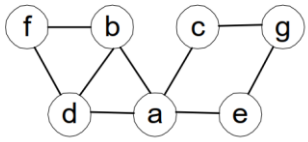
$$|E^{(tree)}| = \sum_{c=1}^{|C|} |E_c^{(tree)}| = \sum_{c=1}^{|C|} (|V_c| - 1) = \sum_{c=1}^{|C|} |V_c| - \sum_{c=1}^{|C|} 1 = |V| - |C|$$

而且：

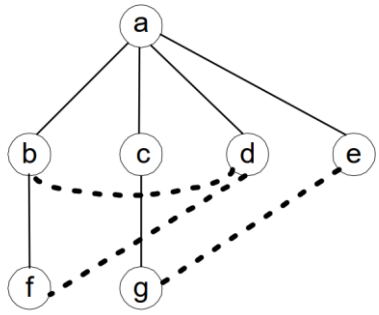
$$|E^{(back)}| = |E| - |E^{(tree)}| = |E| - (|V| - |C|) = |E| - |V| + |C|$$

其中，$|E_c^{(tree)}|$和$|V_c|$分别是第c个连通分量中树向边和顶点的数量。。

4. 向第 1 题的图应用 BFS 遍历的结果如下所示：



a b c d e f g



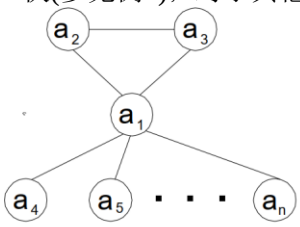
(i) (ii) (iii)

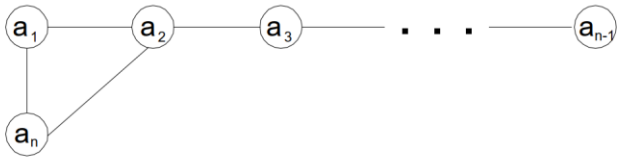
(i)图；(ii)遍历队列；(iii)树(树向边和交叉边分别用实线和虚线显示)。

5. 可用反证法来证明。假设某个无向图的 BFS 树有一条连接两个顶点u和v的交叉边，使得$level[u] \geq level[v] + 2$($level$代表层级)。但是，$level[u] = d[u]$，且$level[v] = d[v]$，其中$d[u]$和$d[v]$分别是从根到顶点$u$和$v$的最少边路径的长度。所以，我们有$d[u]\ \geq\ d[v]\ +\ 2$。最后一个不等式与$d[u]$是根到顶点$u$的最少边路径的长度这一事实相悖，因为从根到顶点$v$的最少边路径长度$d[v]$，后跟边$(v,u)$，其边数小于了$d[u]$。

6. a. 当且仅当 BFS 森林有一个交叉边的时候，图才有环。

b. DFS 和 BFS 这两种遍历方法都可以用来检查图的无环性(acyclicity)。对于某些图来说，DFS 遍历在其 DFS 森林中发现一条回边比 BFS 遍历发现一条交叉边更快(参见例 i)；对于其他图来说，情况则正好相反(参见例 ii)。





(i) (ii)

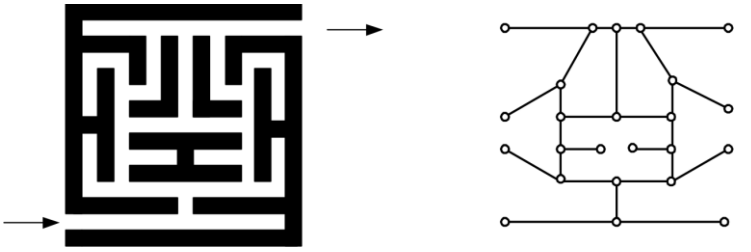
7. 从任意顶点开始 DFS(或 BFS)遍历，并将访问过的顶点标记为 1。当遍历栈(队列)变为空时，与起始顶点相同的连通分量中的所有顶点(并且只有这些顶点)会被标记为 1。如果还有未访问的顶点，则从其中一个重新开始遍历，并将所有访问到的顶点标记为 2。以此类推，直到没有未访问的顶点。

8. a. 设F是一个图的 DFS 森林。不难看出，当且仅当不存在连接“都在奇数层或都在偶数层的两个顶点”的回边时，F是二分图。DFS 遍历需要验证的正是这一属性。注意，DFS 遍历可在第一次到达顶点时将其标记为偶数或奇数。

   b. 与(a)部分类似，当且仅当一个图的 BFS 森林没有连接同一层级上的顶点的交叉边时，该图是二分图。使用 BFS 遍历来检查这种交叉边是否存在。

9. n/a

10. a. 迷宫及其图如下所示：



   b. DFS 在穿越迷宫时比 BFS 方便得多。当 DFS 移动到下一个顶点时，它通过一条边与当前顶点相连(即物理迷宫中的“就近”或 close nearby)，而 BFS 通常不是这种情况。事实上，DFS 可认为是对一个古老的右手穿越迷宫规则的概括：穿越迷宫时，总是右手摸着墙走。

11. 以下序列用 6 步解决了这个谜题，这是最少的步数。

步骤#	8品脱壶	5品脱壶	3品脱壶
	8	0	0
1	3	5	0
2	3	2	3
3	6	2	0
4	6	0	2
5	1	5	2
6	1	4	3

三壶问题的解

虽然也能通过试错来求解，但有一个系统性的方法可以得到这个解。可以用由非负整数构成的三元组(triple)来表示水壶的状态，其中每个数字分别表示 3 品脱、5 品脱和 8 品脱壶中的水量。所以，先从三元组 008 开始。将以 BFS 方式考虑从水壶的当前状态到新的可能状态的所有合法转换。用初始状态三元组 008 初始化队列，并重复以下步骤，直至首次遇到目标状态，即含有一个 4 的三元组。用队头状态的三元组标记可从它到达的所有新状态，将这些新状态添加到队列，然后从队列中删除队头状态。首次到达目标状态后，沿着标签回溯，即获得解决该谜题的最短转换序列。

将上述算法应用于该谜题，可得到以下队列状态序列，前面提到的标记在新的三元组被首次入队时显示为下标：

$008 \mid 305_{008}, 053_{008} \mid 053,\ 035_{305}, 350_{305} \mid 035,\ 350,\ 323_{053} \mid 350,\ 323,\ 332_{035} \mid 323,\ 332 \mid$
$332,\ 026_{323} \mid 026,\ 152_{332} \mid 152,\ 206_{026} \mid 206,\ 107_{152} \mid 107,\ 251_{206} \mid 251,\ 017_{107} \mid 017,\ 341_{251}$

从 341 的标签回溯，即可得到最少用 6 步来解决该谜题的以下转换序列：

008 → 053 → 323 → 026 → 206 → 251 → 341

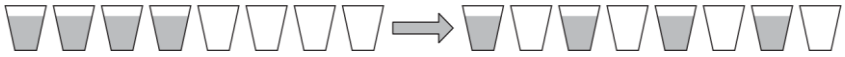
第 4 章

习题 4.1

1．　士兵渡河　n个士兵组成的小分队必须渡过一条又深又宽又没有桥的河。他们注意到在岸旁有两个 12 岁大的小男孩在玩划艇。然而船非常小，只能容纳两个男孩或者一名士兵。怎样才能让所有士兵渡河，最后剩下两个男孩一起操纵这条船？这条船要在岸与岸之间横渡多少次？

2．　交替放置的玻璃环

a. $2n$个玻璃杯挨个排成一行，前n个装满水，其余n个杯子为空。交换杯子的位置，使之按照满－空－满－空的模式排列，而且杯子移动的次数要最少([Gar78]，p. 7)。



b. 如果$2n$个杯子——n个有水，n个是空的——最初以随机顺序排列，请解决同样的问题。

3．　标记单元格　为下列任务设计一个算法。n 为任意偶数，在一张无限大的绘图格子纸上标记 n 个单元格，使得每个被标记的单元格有奇数个相邻的标记单元格。相邻是指两个单元格在水平方向或垂直方向上相邻，但非对角方向上相邻。被标记的单元格必须形成连续域，也就是说区域中任意一对标记单元格之间有一条经过一系列相邻的、被标记的单元格的路径。([Kor05])

4．　设计一个减一算法，生成一个n元素集合的幂集(一个集合S的幂集是S的所有子集的集合，包括空集和S本身)。

5.　对于以邻接矩阵定义的图，用如下算法检测其连通性。

```
算法  Connected(A[0..n – 1, 0..n – 1])
      //输入：无向图 G 的邻接矩阵 A[0..n – 1, 0..n – 1])
      //输出：如果 G 是连通的，输出 1(true) ，否则输出 0(false)
      if n = 1 return 1 //单一顶点的图显然是连通的
      else
          if not Connected(A[0..n – 2, 0..n – 2]) return 0
          else for j ← 0 to n – 2 do
              if A[n – 1, j] return 1
              return 0
```

该算法是否对每个 $n > 0$ 个顶点的无向图都能正确运行？如果回答是，请说明最坏情况下的算法效率类型；如果回答否，请说明为什么。

6. **球队排序** 给定一个完全循环赛的比赛结果，其中 n 个球队两两比赛一次。每场比赛以一方胜出或者平局结束。设计一个算法，把 n 个球队排序，序列中每个球队都不曾输给紧随其后的那个球队。说明该算法的时间效率类型。

7． 应用插入排序将序列 E, X, A, M, P, L, E 按照字母顺序排序。

8． **a．**对于插入排序来说，为了避免在内部循环的每次迭代时判断边界条件 $j \geq 0$，我们应该在待排序数组的第一个元素前放一个什么样的哨兵？

b．带哨兵的版本和原版本的效率类型相同吗？

9． 能不能实现一个对链表排序的插入排序算法？它是不是和数组版本都一样有着 $O(n^2)$ 效率呢？

10．将课本中的插入排序实现和以下版本做比较。

算法 *InsertSort2*($A[0..n-1]$)
  for $i \leftarrow 1$ **to** $n-1$ **do**
    $j \leftarrow i-1$
    while $j \geq 0$ **and** $A[j] > A[j+1]$ **do**
      swap($A[j], A[j+1]$)
      $j \leftarrow j-1$

该算法的时间效率如何？和 4.1 节给出的版本比又如何？

11．设 $A[0..n-1]$是n个可排序元素的数组(简单起见，假设所有元素互不相同)。对于$(A[i], A[j])$这样的对，如果 $i < j$ 且 $A[i] > A[j]$，我们将其称为一个**倒置**。

a．规模为n的数组在什么情况下具有最大数量的倒置？倒置数为多少？如果问的是最小数量的倒置呢？

b．▶为什么插入排序的平均键比较次数符合以下公式？

$$C_{avg}(n) \approx \frac{n^2}{4}$$

12．希尔排序(由 D. L. 希尔发明)是一种重要的排序算法，它对一个给定序列的若干步长子序列分别应用插入排序。对序列的每一遍操作，都根据一些事先定义好的递减的步长队列 $h_1 > \cdots > h_i > \cdots > 1$ 来构造所要求的子序列，这个步长队列必须以 1 作为结尾。(该算法对任意步长队列都有效，但有些步长队列的效率要比其他的高。例如，对于希尔排序来说，步长队列 1, 4, 13, 40, 121 的效率是最高的。当然，使用的时候要反过来。)

a．对以下序列应用希尔排序：

$$S, H, E, L, L, S, O, R, T, I, S, U, S, E, F, U, L$$

b．希尔排序是一个稳定的排序算法吗？

c．任意选择一种语言实现希尔排序、直接插入排序、选择排序以及冒泡排序，然后对于序列大小为 10^n，$n = 2, 3, 4, 5, 6$ 的随机序列、升序序列和降序序列分别比较它们的性能。

习题 4.1 提示

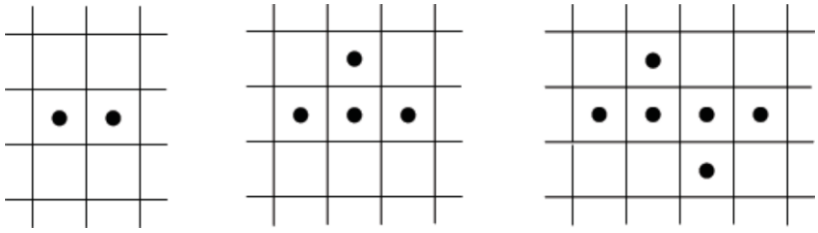
1． 在$n=1$时对问题求解。
2． 可认为将水从满杯子倒到空杯子是一次移动(one move)。
3． 用自下而上法比较容易。
4． 利用这样一个事实，一个 n 元素集合 $S=\{a_1, \cdots, a_n\}$的所有子集能够分为两组：包含 a_n 的和不包含 a_n 的。
5． 答案是“否”。
6． 用类似插入排序的思路。
7． 跟踪该算法，就像课本中对另一个输入的做法(参见图 4.4)。
8． **a．**哨兵应该防止最小的元素向前超出数组的第一个位置。
b．将课本中所做的分析对哨兵版本再做一遍。
9． 以前讲过，对于单链表来说，我们只能顺序访问它的元素。
10．比较这两种算法最内层循环的运行时间。
11． **a．**先对一个三元素数组回答该问题，就能得出一个一般性的答案。
b．为简单起见，可假设所有元素都各不相同的，而且将 $A[i]$插入它前 i+1 个位置的概率都是相同的。先分析该算法的哨兵版。
12． **a．**注意，对子列表并行排序会更方便，也就是说，$A[0]$和 $A[h_i]$比较，$A[1]$和 $A[1+h_i]$比较，以此类推。
b．以前讲过，交换两个相隔很远的元素的排序算法通常是不稳定的。

习题 4.1 答案

1. 首先，两个男孩把船开到另一边，之后其中一个男孩把船开回来。然后，一个士兵把船开到另一边，呆在那里，另一个男孩把船开回来。这四次渡河将规模为n(需要运送的士兵数量)的问题实例减少到规模$n-1$。因此，如果这个四次渡河的过程总共重复n次，问题将在总共$4n$次旅行后得到解决。
2. a. 假设杯子从1到 $2n$ 从左到右编号，将2号杯的水倒入$2n-1$号杯。这使第一对和最后一对杯子以所要求的模式交替出现，从而将问题的规模减少为$2(n-2)$个中间杯子的相同问题。如果n是偶数，这个操作需要重复的次数就等于$n/2$；如果n是奇数，就等于$(n-1)/2$。公式$\lfloor n/2 \rfloor$为这两种情况提供了一个闭合形式的解。注意，这个解也可以通过求解递推式$M(n)=M(n-2)+1$，$n>2$，$M(2)=1$，$M(1)=0$来获得。其中，$M(n)$是上述减 2 算法的移动次数。针对$\lfloor n/2 \rfloor$对不重叠的充满水的玻璃杯中的每一对，由于该问题的任何算法都必须移动其中至少一个充满水的杯子，所以$\lfloor n/2 \rfloor$是解决这个问题所需的最少移动次数。
   关于另一种算法，请参考(b)部分中该问题的一个更常规的版本。
   注：该问题是在马丁·加德纳(Martin Gardner)的“aha!Insight”中提出的，Scientific American/W.H.Freeman, p. 7。
   b. 在杯子的最终状态中，所有在奇数位置的杯子都必须填满，所有在偶数位置的

杯子都必须为空。如果在谜题的初始状态下，偶数位置有$k(0 \le k \le n)$个满的杯子，那么奇数位置也有k个空杯。为了以最少的移动次数解开谜题，有必要且足够将水从偶数位置的k个杯子倒入奇数位置的k个空杯子。为了找出有多少对这样的杯子，可以简单地扫描这一排杯子，找到下一个在偶数位置的装满水的杯子和下一个在奇数位置的空杯子。

3. 如果$n = 2$，一个明显的解如下图所示(左图)。在这个解中，标记和最右边的单元格相邻的两个单元格——一个在水平方向，另一个在垂直方向(例如上方)——就可以得到$n = 4$的解(中图)。再次重复同样的操作，但在垂直方向标记最右边的单元格的下方邻居，而不是上方邻居，从而得到$n = 6$的解(右图)。采取这种方式，可以解任何给定偶数值n的谜题。



“标记单元格”谜题在$n = 2$，$n = 4$和$n = 6$时的解

注：该谜题来自 B. A. Kordemsky 的《Mathematical Charmers》一书， Oniks，2005 (俄语)。

4. 下面概括了创建$\{a_1, \ldots, a_n\}$所有子集的列表$L(n)$的递归算法。(更详细的讨论请参见第 4.3 节)。

if $n = 0$ **return** 列表$L(0)$，其中包含空集作为其唯一元素
else 以递归方式创建列表$L(n - 1)$，其中包含$\{a_1, \ldots, a_{n-1}\}$的所有子集
将a_n附加到$L(n - 1)$的每个元素以得到列表T
return 通过连接$L(n - 1)$和T得到的$L(n)$

5. 下面这行代码不正确：

if not *Connected*(A[0..n – 2, 0..n – 2]) **return** 0

作为一个反例，考虑在一个图中，前$n - 1$个点之间没有边，但每个点有一条边连接到第n个顶点。

6. 用任何一支球队初始化目标列表。对于其他每支球队，扫描列表，将其插入它没有输掉的第一支球队之前；如果它输给了列表中的所有球队，则插入列表末尾。该算法的效率是$O(n^2)$，因为在最坏情况下，每支球队在检查完$1 + 2 + \ldots + (n - 1) = (n - 1)n/2$支已经在名单上的球队后，才被插入列表末尾。

7. 用插入排序算法对列表 E，X，A，M，P，L，E 按字母顺序进行排序：

$$
\begin{array}{lllllll}
E & X & A & M & P & L & E \\
E & \mid \mathbf{X} & & & & & \\
E & X & \mid \mathbf{A} & & & & \\
A & E & X & \mid \mathbf{M} & & & \\
A & E & M & X & \mid \mathbf{P} & & \\
A & E & M & P & X & \mid \mathbf{L} & \\
A & E & L & M & P & X & \mid \mathbf{E} \\
A & E & E & L & M & P & X
\end{array}
$$

8. a. $-\infty$，或者更一般地说，小于或等于数组中每个元素的任意值。

   b. 相同，效率类型保持不变。严格递减数组（最坏情况下的输入）的键比较次数将是：

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=-1}^{i-1} 1 = \sum_{i=1}^{n-1}(i+1) = \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 = \frac{(n-1)n}{2} + (n-1) \in \Theta(n^2)$$

9. 可以，但必须在插入$A[i]$时从左向右扫描排好序的部分，以获得和数组版本一样的$O(n^2)$效率。

10. 两个版本的效率类型是一样的。InsertionSort 的内层循环包括一次键赋值和一次索引递减；InsertionSort2 的内层循环则包括一次键交换（即三次键赋值）和一次索引递减。如果不考虑花在索引递减上的时间，运行时间的比率估计为$\frac{3c_a}{c_a} = 3$；如果考虑到花在索引递减上的时间，比率的估计值就变成了$(3c_a + c_d)/(c_a + c_d)$，其中c_a和c_d分别是一次键赋值和一次索引递减的时间。

11. a. $A[i]$（$0 \le i \le n-1$）的最大倒置数是$n-1-i$；如果$A[i]$大于它右边的所有元素，就会发生这种情况。因此，整个数组的最大倒置数量发生在严格递减的数组中。这个最大的数字由以下求和公式给出：

$$\sum_{i=0}^{n-1}(n-1-i) = (n-1) + (n-2) + \cdots + 1 + 0 = \frac{(n-1)n}{2}$$

   $A[i]$（$0 \le i \le n-1$）的最小倒置数是0；如果$A[i]$小于或等于它右边的所有元素，就会发生这种情况。因此，对于非递减数组，整个数组的最小倒置数是 0。

   b. 假设所有元素都各不相同，而且在其之前$i+1$个可能的位置插入$A[i]$的可能性是相同的，我们得到算法的哨兵版本的以下第i次迭代的预期键比较次数：

$$\frac{1}{i+1}\sum_{j=1}^{i+1} j = \frac{1}{i+1}\frac{(i+1)(i+2)}{2} = \frac{i+2}{2}$$

   所以，键的平均比较次数$C_{avg}(n)$是：

$$C_{avg}(n) = \sum_{i=1}^{n-1} \frac{i+2}{2} = \frac{1}{2}\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 = \frac{1}{2}\frac{(n-1)n}{2} + n - 1 \approx \frac{n^2}{4}$$

   对于无哨兵版本，在$A[0]$前后插入$A[i]$的键比较次数是一样的。因此，在无哨兵版本的第i次迭代中，预期的键比较次数为：

$$\frac{1}{i+1}\sum_{j=1}^{i} j + \frac{i}{i+1} = \frac{1}{i+1}\frac{i(i+1)}{2} + \frac{i}{i+1} = \frac{i}{2} + \frac{i}{i+1}$$

所以，键的平均比较次数$C_{avg}(n)$是：

$$C_{avg}(n) = \sum_{i=1}^{n-1}(\frac{i}{2} + \frac{i}{i+1}) = \frac{1}{2}\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1}\frac{i}{i+1}$$

我们得到第一个和的闭合公式：

$$\frac{1}{2}\sum_{i=1}^{n-1} i = \frac{1}{2}\frac{(n-1)n}{2} = \frac{n^2-n}{4}$$

第二个和可像下面这样估算：

$$\sum_{i=1}^{n-1}\frac{i}{i+1} = \sum_{i=1}^{n-1}(1-\frac{1}{i+1}) = \sum_{i=1}^{n-1} 1 - \sum_{i=1}^{n-1}\frac{1}{i+1} = n-1-\sum_{j=2}^{n} j = n - H_n$$

其中，$H_n = \sum_{j=1}^{n} 1/j \approx \ln n$(基于附录 A 引用的一个著名的公式)。因此，对于无哨兵版本的插入排序，我们也得到：

$$C_{avg}(n) \approx \frac{n^2-n}{4} + n - H_n \approx \frac{n^2}{4}$$

12. a. 使用步长 13、4 和 1 将希尔排序应用于以下列表：

$$S_1, H, E_1, L_1, L_2, S_2, O, R, T, I, S_3, U_1, S_4, E_2, F, U_2, L_3$$

将得到以下结果(如果比较导致交换，则只显示交换的结果)：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
S_1	H	E_1	L_1	L_2	S_2	O	R	T	I	S_3	U_1	S_4	E_2	F	U_2	L_3
E_2													S_1			
	F													H		
		E_1													U_2	
			L_1													L_3

E_2				L_2												
	F				S_2											
		E_1				O										
			L_1				R									
				L_2				T								
					I				S_2							
	F				I											
						O				S_3						
							R				U_1					
								S_4				T				
				L_2				S_4								
									S_2				S_1			
										H				S_3		
						H				O						
		E_1				H										
											U_1				U_2	
												L_3				T
								L_3				S_4				
				L_2				L_3								

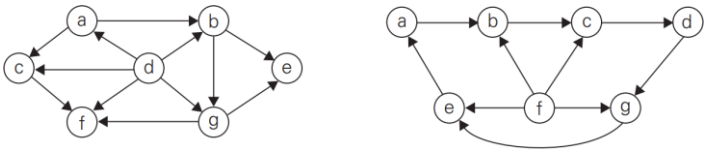
E_2	F	E_1	L_1	L_2	I	H	R	L_3	S_2	O	U_1	S_4	S_1	S_3	U_2	T

由于过于简单，这个解省略了步长为 1 的最后一遍（通过插入排序对最后的数组进行排序）。注意，由于在最后的数组中，因为希尔排序之前的工作，只剩下少数几个没有排好序的元素，所以在此时进行插入排序，需要执行的比较次数比应用于初始数组时少得多。

b. 希尔排序是不稳定的。作为一个步长 4 和 1 的希尔排序的反例，考虑一下数组 5，1，2，3，1。步长为 4 的第一遍将用最后一个 1 交换 5，从而改变了数组中两个 1 的相对顺序。步长为 1 的第二遍，也就是插入排序，将不会进行任何交换，因为数组已经排好序了。

习题 4.2

1． 对于以下有向图，应用基于 DFS 的算法来解拓扑排序问题。



2． **a．**请证明，当且仅当有向图是无环时，它的拓扑排序问题才是有解的。

b．对于一个具有n个顶点的有向图，拓扑排序问题最多会有多少个不同的解？

3． **a．**基于 DFS 的拓扑排序算法的时间效率是怎样的？

b．如何修改基于 DFS 的算法，以避免反转 DFS 生成的顶点顺序？

4． 能否利用顶点入 DFS 栈的顺序(代替它们的出栈顺序)来解拓扑排序问题？

5． 对第 1 题中的有向图应用源删除算法。

6． **a．**请证明一个无环有向图必定至少具有一个源。

b．在用邻接矩阵表示的有向图中，我们如何求得一个源(或者确定这样一个顶点不存在)？这种操作的时间效率如何？

c．在用邻接链表表示的有向图中，我们如何求得一个没有输入边的顶点(或者确定这样一个顶点不存在)？这种操作的时间效率如何？

7． ▷是否能对一个用邻接矩阵表示的有向图实现源删除算法，使得它的运行时间属于 $O(|V|+|E|)$？

8． 任选一种语言实现这两种拓扑排序算法并做一个实验来比较它们的运行时间。

9． 如果对于任意两个不同的顶点u和v，存在一个从u到v的有向路径以及一条从v到u的有向路径，这样的有向图被称为是**强连通**(strongly connected)的。一般来说，一个有向图的顶点可以分割成一些顶点的互不相交的最大子集，每个子集的顶点之间可以通过有向图中的有向路径相互访问，这些子集被称为**强连通分量**(strongly connected component)。有两种基于 DFS 的算法来确定强连通分量。以下是两个中较简单(但效率较低)的一种。

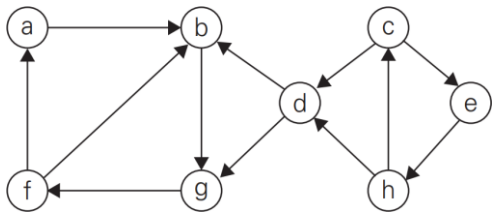
第一步：对给定的有向图执行一次 DFS 遍历，然后按照顶点变成死端（dead ends）的顺序对它们进行编号。

第二步：反转有向图中所有边的方向。

第三步：对于新的有向图，从仍未访问过的顶点中编号最大的顶点开始(而且必要时可重新开始)做一遍 DFS 遍历。

在最后一次遍历中得到的每一棵 DFS 树的顶点构成的子集就是一个强连通分量。

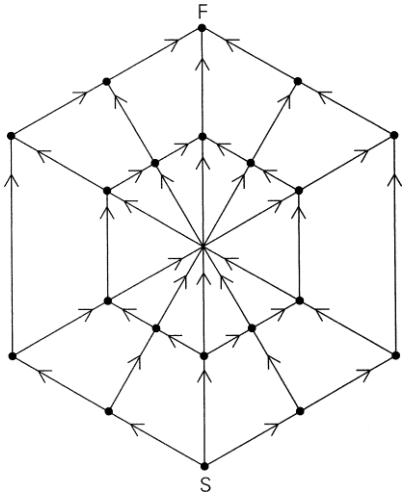
a . 对下图应用该算法，确定它的强连通分量。



b . 该算法属于哪种时间效率类型？对于一个输入图的邻接矩阵表示法和邻接链表表示法分别回答这个问题。

c . 一个无环有向图会有多少个强连通分量？

10 . 蛛网问题 一只蜘蛛位于网的底端(点 S)，而一只苍蝇位于网的顶端(F)。沿箭头方向在线上移动，蜘蛛有多少不同的路径到达苍蝇处？([Kor05])



习题 4.2 提示

1 . 按照课本中对另一个图的做法跟踪算法(参见图 4.7)。

2 . a . 需要证明两个断言：(i)如果有向图中具有有向回路，那么拓扑排序问题无解；(ii)如果有向图中不包含有向回路，那么拓扑排序问题有解。

b . 考虑一种极端类型的有向图。

3 . a . 它和 DFS 的时间效率具有什么样的关系？

b . 我们是否知道该算法生成的顶点列表的长度？从 DFS 遍历的顶点栈中出栈的第一个顶点，它的最终位置在哪里？

4 . 试着对一两个较小的例子做做看。

5 . 按照课本中的做法，针对给定的实例跟踪该算法(参见图 4.8)。

6 . a . 用反证法证明。

b . 如果回答这个问题有困难，可以考虑一个有向图的例子，图中有一个顶点没有输入边，然后写出它的邻接矩阵。

c．要根据汇点(sink)和邻接链表的定义来得到答案。

7．对于余下子图中的每个顶点，存储它的输入边数量。维护一个源顶点的队列。

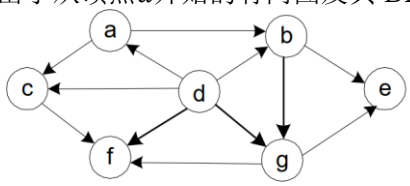
9． **a．**按照指示，遵循算法的每个步骤，对于给定的实例跟踪该算法。

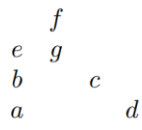
b．确定该算法三个主要步骤中每一步的效率，然后再确定总效率。当然，这个回答并不依赖于这个图是用邻接矩阵表示的还是用邻接链表表示的。

10．充分利用拓扑排序以及图的对称性。

习题 4.2 答案

1. a. 下面给出了从顶点a开始的有向图及其 DFS 遍历栈：





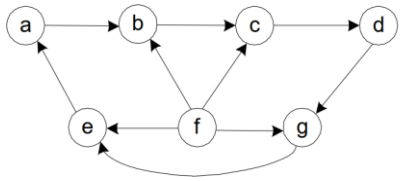
顶点按以下顺序出栈：

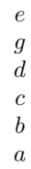
$$e\ f\ g\ b\ c\ a\ d$$

通过反转上述列表得到的拓扑排序是：

$$d\ a\ c\ b\ g\ f\ e$$

b. 以下有向图不是一个无环有向图。它从从a开始的 DFS 遍历遇到了一条从e到a的回边：





2. a. 可用反证法证明如果有向图有一个有向的回路，那么拓扑排序问题无解。假设$v_{i_1},\ldots,v_{i_n}$是具有有向回路的一个有向图的拓扑排序问题的解。设v_{i_k}是这个回路在列表$v_{i_1},\ldots,v_{i_n}$上最左边的顶点。由于该回路的边从右向左进入v_{i_k}，所以产生了一个悖论，从而证明了该断言。

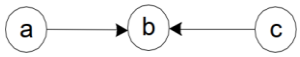
如果有向图不存在有向的回路，那么拓扑排序问题的解可通过本节讨论的两种算法中的任何一种来获得。(基于 DFS 的算法的正确性已进行了解释；去源算法的正确性源于问题 6a 的断言）。

b. 对于一个有n个顶点且无边的有向图，其顶点的任何排列都能解决拓扑排序问题。因此，这个问题的答案是$n!$。

3. a. 由于反转顶点从 DFS 遍历栈中的出栈顺序属于$\Theta(|V|)$，所以算法的运行时间与 DFS 的运行时间相同（只是如果遇到回边，它可以在处理完整个有向图之前停止）。因此，基于 DFS 的算法如果使用邻接矩阵，那么运行时间为$O(|V|^2)$；如果使用邻接链表，则为$O(|V|+|E|)$。

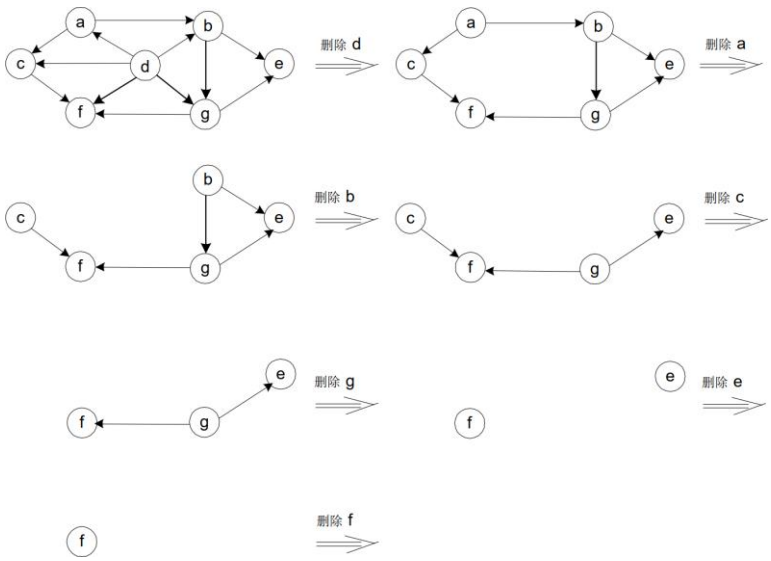
b. 使用从 DFS 遍历栈中出栈的顶点从右向左填充长度为$|V|$的数组。

4. 答案是否。下面是一个简单的反例：



从a开始的 DFS 遍历以a, b, c的顺序使顶点入栈，这种排序或其反转都不能正确解决拓扑排序问题。

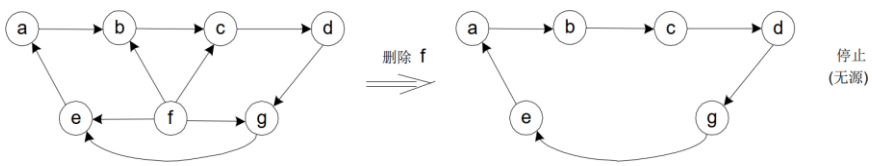
5. a.



获得的拓扑排序是：

$$d \quad a \quad b \quad c \quad g \quad e \quad f$$

b.



不可能进行拓扑排序。

6. a. 假设相反，存在一个无环有向图(dag)，它的每个顶点都有一条输入边（incoming edge）。反转其所有边，就会产生每个顶点都有一条出发边（outgoing edge）的无环有向图。那么，从一个任意的顶点开始，沿着这些出发边构成的一个链条，我们将在不晚于$|V|$步后得到一个有向回路。这一悖论证明了断言。

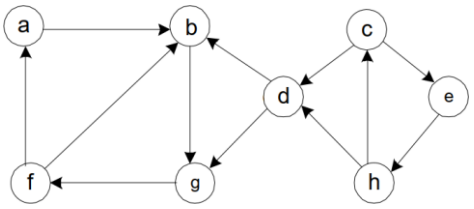
   b. 当且仅当邻接矩阵中的对应列只包含 0 时，无环有向图的一个顶点是一个源。寻找这样的列是一个$O(|V|^2)$操作。

   c. 当且仅当一个无环有向图的顶点没有出现在无环有向图的邻接链表中时，该顶点才是一个源。寻找这样的顶点是一个$O(|V| + |E|)$操作。

7. 这个著名问题的答案是“可以”(参见[KnuI], pp. 264- 265)。

8. n/a

9. a. 给定的有向图是：

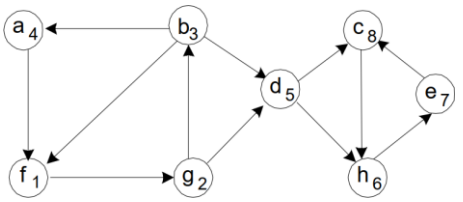


以 a 为起始顶点第一次 DFS 遍历的栈如下所示：

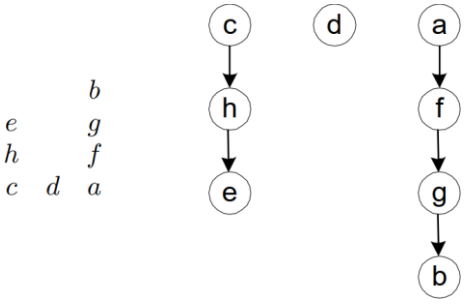
$$\begin{array}{lll} f_1 & & \\ g_2 & & h_6 \\ b_3 & d_5 & e_7 \\ a_4 & c_8 & \end{array}$$

(数字代表顶点的出栈顺序。)

具有反转边的有向图是：



第二个有向图的 DFS 遍历栈和 DFS 树（只显示树向边）如下所示：



给定有向图的强连通分量是：

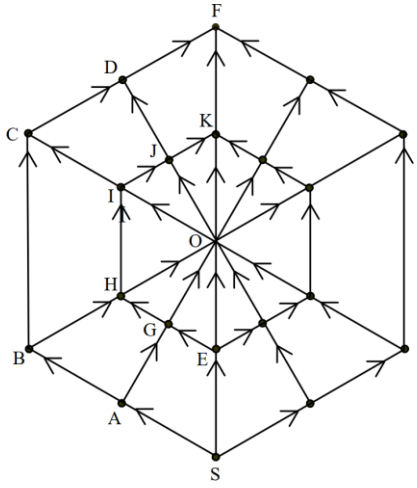
$$\{c, h, e\}, \quad \{d\}, \quad \{a, f, g, b\}$$

b. 如果一个图用它的邻接矩阵表示，那么第一次 DFS 遍历的效率为$\Theta(|V|^2)$。 边反转步骤（在给定图的邻接矩阵中，如果$A[i,j] = 1$，则在新的有向图的邻接矩阵中将$B[j,i]$设为 1，否则设为 0）的效率也是$\Theta(|V|^2)$。 新图的最后一次 DFS 遍历的时间效率也是$\Theta(|V|^2)$。因此，整个算法的效率是$\Theta(|V|^2)+ \Theta(|V|^2) + \Theta(|V|^2) = \Theta(|V|^2)$。

通过类似的推理（对中间步骤进行必要的调整），由邻接链表表示的图的答案是$\Theta(|V| + |E|)$。

10. 从 S 到蛛网图的一个顶点v的有向路径总数是从 S 到所有顶点u的有向路径的总和，这些顶点有一条从u到v的有向边。由于从 S 到 F 的所有路径都是对称的，对于这条路径上的每个顶点（包括 F），为了简化路径之和的计算，可将从左边进入顶点的路径数量加倍，而不必考虑从右边进入顶点的路径。为了能计算路径之和，

我们需要对有向图左边部分进行拓扑排序；下表展示了这一线性排序的结果。然后，可通过处理这个线性顺序的顶点来计算总和，将从左边进入目标顶点的每条边的计数加倍。图下方的列表展示了这些数字。



S(1)–A(1)–B(1)–E(1)–G(1)–H(3)–O(11)–I(14)–C(15)–J(25)–D(40)–K(61)–F(141)

所以，从 S 到 F 共有 141 条路径。

习题 4.3

1． 在你的计算机上实现一个要求生成 25 个元素组成的集合的全部排列的算法是否现实？如果是生成该集合的所有子集呢？

2． 使用下面的方法生成$\{1, 2, 3, 4\}$的全部排列：

a．自下而上的最小变化算法。

b．Johnson-Trotter 算法。

c．字典序算法。

3． 将 *LexicographicPermute* 算法应用到多重集$\{1, 2, 2, 3\}$。它是否能正确生成字典序的所有排列？

4． ▶请考虑下面这个生成排列算法的实现，这个算法是由 B. 希普(B. Heap)发明的([Hea63])。

算法 *HeapPermute*(*n*)

```
//实现生成排列的 Heap 算法
//输入：一个正整数 n 和一个全局数组 A[1..n]
//输出：A 中元素的全排列
if n = 1
    write A
else
    for i ← 1 to n do
        HeapPermute(n − 1)
```

if n 是奇数

swap $A[1]$ and $A[n]$

else *swap* $A[i]$ and $A[n]$

a．对于 $n = 2, 3, 4$ 的情况，人工跟踪该算法。

b．证明 Heap 算法的正确性。

c．*HeapPermute* 的时间效率如何？

5． 用本节介绍的两种算法生成四元素集合 $A = \{a_1, a_2, a_3, a_4\}$ 的所有子集。

6． 有什么简单的小窍门可以使得基于位串的算法可以按照挤压序生成子集？

7． 为生成所有 2^n 个长度为 n 的位串的递归算法写伪代码。

8． 写一个生成 2^n 个长度为 n 的位串的非递归算法，它用数组来实现位串，并且不使用二进制加法。

9． **a．**生成 4 位的二进制反射格雷码。

b．跟踪下面生成 4 位二进制反射格雷码的非递归算法。以全 0 的 n 位串开始。而对于 $i = 1, 2, \cdots, 2^{n-1}$，则通过反转前一位串中的第 b 位来生成第 i 个位串，在这里 b 是 i 的二进制形式中最低位 1 的位置。

10．▶设计一个减治算法来生成 n 个元素的 k 个分量的所有组合，也就是说，一个给定的 n 个元素集合的所有 k 个元素的子集。你设计的算法是最小变化算法吗？

11．格雷码和汉诺塔

a. ▷为什么汉诺塔的经典递归算法产生的移动盘子动作可以用来生成二进制反射格雷码？

b. ▶如何利用二进制反射格雷码来解汉诺塔问题？

12．展会彩灯 早年的展会可能会看到这样一种彩灯：一个被连接到若干开关上的电灯泡，只当所有开关都闭合的时候才会发光。每一个开关由一个按钮控制；按下按钮就会切换开关状态，但是开关的状态是无法知道的。目标就是点亮灯泡。设计一个点亮灯泡的算法，使其在有 n 个开关时，在最坏的情况下，需要按动按钮的次数最少。

习题 4.3 提示

1． 利用组合对象数量的标准公式。为简单起见，可假设生成一个组合对象的时间和一次赋值是相同的。

2． 本节对一个更小的实例跟踪过这些算法。

3． 参见本节对该算法的描述。

4． **a．**对于 $n = 2$ 跟踪该算法。在对 $n = 3$ 跟踪这个算法时，利用以上跟踪结果，然后再将新的跟踪结果用于 $n = 4$。

b．证明该算法能够生成 $n!$ 个排列，而且各个排列互不相同。使用数字归纳法。

c．建立该算法的交换次数的递推关系。求它的解以及这个解的增长量级。对于较大的 n，可能会用到公式 $e \approx \sum_{i=0}^{n} \frac{1}{i!}$。

5． 本节对更小的实例跟踪过这两种算法。

6． 窍门讲出来就变得乏味了。

7． 这道题并不难，因为从一个长度为 $n-1$ 的位串生成一个长度为 n 的位串的方法是显而易见的。

8． 仍然可以模仿二进制加法，但不要明确地使用它。

9． 对于 $n=4$，跟踪算法。

10． 对于这个问题有若干种减治算法。它们比我们想象当中的要复杂。按照预定义的次序生成组合不仅有助于设计，也有助于正确性证明。下面这个简单的特性是非常有用的。在不失一般性的前提下，我们假设给定的集合是 $\{1, 2, \cdots, n\}$，那么最小元素是 i 的 k 元素子集一共有 $\binom{n-i}{k-1}$ 个，$i=1, 2, \cdots, n-k+1$。

11． 翻转二进制 n 元组中的二进制位来表示盘子的移动。

12． 可将开关视为位串中的二进制位，这有帮助，但非必要。

习题 4.3 答案

1. 由于 $25! \approx 1.5 \cdot 10^{25}$，即使在超级计算机上生成这个数量的排列也需要不切实际的漫长时间。另一方面，$2^{25} \approx 3.3 \cdot 10^{7}$，在每秒进行一亿次运算的计算机上生成约耗时 0.3 秒。

2. a. 由自下而上的最小变化算法所生成的{1，2，3，4}的排列如下所示：

start	1			
insert 2 into 1 right to left	12	21		
insert 3 into 12 right to left	123	132	312	
insert 3 into 21 left to right	321	231	213	
insert 4 into 123 right to left	1234	1243	1423	4123
insert 4 into 132 left to right	4132	1432	1342	1324
insert 4 into 312 right to left	3124	3142	3412	4312
insert 4 into 321 left to right	4321	3421	3241	3214
insert 4 into 231 right to left	2314	2341	2431	4231
insert 4 into 213 left to right	4213	2413	2143	2134

b. 以下是由 Johnson-Trotter 算法生成的{1，2，3，4}的排列。(横着读；最大的移动元素以粗体显示）：

$$
\begin{array}{cccc}
\overleftarrow{1}\,\overleftarrow{2}\,\overleftarrow{3}\,\overleftarrow{\mathbf{4}} & \overleftarrow{1}\,\overleftarrow{2}\,\overleftarrow{\mathbf{4}}\,\overleftarrow{3} & \overleftarrow{1}\,\overleftarrow{\mathbf{4}}\,\overleftarrow{2}\,\overleftarrow{3} & \overleftarrow{4}\,\overleftarrow{1}\,\overleftarrow{2}\,\overleftarrow{\mathbf{3}} \\
\overrightarrow{\mathbf{4}}\,\overleftarrow{1}\,\overleftarrow{3}\,\overleftarrow{2} & \overleftarrow{1}\,\overrightarrow{\mathbf{4}}\,\overleftarrow{3}\,\overleftarrow{2} & \overleftarrow{1}\,\overleftarrow{3}\,\overrightarrow{\mathbf{4}}\,\overleftarrow{2} & \overleftarrow{1}\,\overrightarrow{\mathbf{3}}\,\overleftarrow{2}\,\overleftarrow{4} \\
\overleftarrow{3}\,\overleftarrow{1}\,\overleftarrow{2}\,\overleftarrow{\mathbf{4}} & \overleftarrow{3}\,\overleftarrow{1}\,\overrightarrow{\mathbf{4}}\,\overleftarrow{2} & \overleftarrow{3}\,\overrightarrow{\mathbf{4}}\,\overleftarrow{1}\,\overleftarrow{2} & \overrightarrow{4}\,\overleftarrow{3}\,\overleftarrow{1}\,\overrightarrow{\mathbf{2}} \\
\overrightarrow{\mathbf{4}}\,\overrightarrow{3}\,\overleftarrow{2}\,\overleftarrow{1} & \overleftarrow{3}\,\overrightarrow{\mathbf{4}}\,\overleftarrow{2}\,\overleftarrow{1} & \overrightarrow{3}\,\overleftarrow{2}\,\overrightarrow{\mathbf{4}}\,\overleftarrow{1} & \overrightarrow{\mathbf{3}}\,\overleftarrow{2}\,\overleftarrow{1}\,\overleftarrow{4} \\
\overleftarrow{2}\,\overleftarrow{3}\,\overleftarrow{1}\,\overleftarrow{\mathbf{4}} & \overleftarrow{2}\,\overleftarrow{3}\,\overrightarrow{\mathbf{4}}\,\overleftarrow{1} & \overleftarrow{2}\,\overrightarrow{\mathbf{4}}\,\overleftarrow{3}\,\overleftarrow{1} & \overleftarrow{4}\,\overleftarrow{2}\,\overrightarrow{\mathbf{3}}\,\overleftarrow{1} \\
\overrightarrow{\mathbf{4}}\,\overleftarrow{2}\,\overleftarrow{1}\,\overleftarrow{3} & \overleftarrow{2}\,\overrightarrow{\mathbf{4}}\,\overleftarrow{1}\,\overleftarrow{3} & \overleftarrow{2}\,\overleftarrow{1}\,\overrightarrow{\mathbf{4}}\,\overleftarrow{3} & \overleftarrow{2}\,\overleftarrow{1}\,\overleftarrow{3}\,\overleftarrow{4}
\end{array}
$$

c. 以下是字典序的{1，2，3，4}的排列（同样横着读）：

1234	1243	1324	1342	1423	1432
2134	2143	2314	2341	2413	2431
3124	3142	3214	3241	3412	3421
4123	4132	4213	4231	4312	4321

3. 1223, 1232, 1322, 2123, 2132, 2213, 2231, 2312, 2321, 3122, 3212, 3221。

4. a. 在$n = 2$的情况下：

12　21

在$n = 3$的情况下（顺着行阅读）：

123　213
312　132
231　321

在$n = 4$的情况下（顺着行阅读）：

1234	2134	3124	1324	2314	3214
4231	2431	3421	4321	2341	3241
4132	1432	3412	4312	1342	3142
4123	1423	2413	4213	1243	2143

b. 设$C(n)$是算法写一个新的排列的次数（当$n = 1$时，递归调用完成）。我们对$C(n)$有如下的递推关系：

$$C(n) = \sum_{i=1}^{n} C(n-1) \quad 或 \quad C(n) = nC(n-1)，其中 n > 1, \quad C(1) = 1$$

它的解是$C(n) = n!$（参见 2.4 节）。该算法生成的所有排列都各不相同，这一事实可通过数学归纳法来证明。

c. 我们有以下关于交换次数$S(n)$的递推关系：

$$S(n) = \sum_{i=1}^{n} (S(n-1)+1) \quad 或 \quad S(n) = nS(n-1)+n，其中 n > 1, S(1) = 0$$

虽然也可以通过反向替换法来解决，但两边都除以$n!$更容易。

$$\frac{S(n)}{n!} = \frac{S(n-1)}{(n-1)!} + \frac{1}{(n-1)!} \quad 其中\ n > 1, \quad S(1) = 0$$

用$T(n) = \frac{S(n)}{n!}$进行替换，获得以下递推式：

$$T(n) = T(n-1) + \frac{1}{(n-1)!} \quad 其中 n > 1, \quad T(1) = 0$$

用反向替换法解最后一个递推式，得到：

$$T(n) = T(1) + \sum_{i=1}^{n-1} \frac{1}{i!} = \sum_{i=1}^{n-1} \frac{1}{i!}$$

由于变量$S(n) = n!\,T(n)$，我们得到：

$$S(n) = n! \sum_{i=1}^{n-1} \frac{1}{i!} \approx n!(e - 1 - \frac{1}{n!}) \in \Theta(n!)$$

5. 自下而上生成四元素集合 $A = \{a_1, a_2, a_3, a_4\}$的所有子集：

n	子集							
0	$\varnothing$							
1	$\varnothing$	$\{a_1\}$						
2	$\varnothing$	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$				
3	$\varnothing$	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$	$\{a_3\}$	$\{a_1, a_3\}$	$\{a_2, a_3\}$	$\{a_1, a_2, a_3\}$
4	$\varnothing$	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$	$\{a_3\}$	$\{a_1, a_3\}$	$\{a_2, a_3\}$	$\{a_1, a_2, a_3\}$
	$\{a_4\}$	$\{a_1, a_4\}$	$\{a_2, a_4\}$	$\{a_1, a_2, a_4\}$	$\{a_3, a_4\}$	$\{a_1, a_3, a_4\}$	$\{a_2, a_3, a_4\}$	$\{a_1, a_2, a_3, a_4\}$

用位向量生成四元素集合 $A = \{a_1, a_2, a_3, a_4\}$的所有子集：

位串	0000	0001	0010	0011	0100	0101	0110	0111
子集	$\varnothing$	$\{a_4\}$	$\{a_3\}$	$\{a_3, a_4\}$	$\{a_2\}$	$\{a_2, a_4\}$	$\{a_2, a_3\}$	$\{a_2, a_3, a_4\}$
位串	1000	1001	1010	1011	1100	1101	1110	1111
子集	$\{a_1\}$	$\{a_1, a_4\}$	$\{a_1, a_3\}$	$\{a_1, a_3, a_4\}$	$\{a_1, a_2\}$	$\{a_1, a_2, a_4\}$	$\{a_1, a_2, a_3\}$	$\{a_1, a_2, a_3, a_4\}$

6. 建立$A = \{a_1, \ldots, a_n\}$的子集与长度为n的位串$b_1 \ldots b_n$之间的对应关系，将位i与元素a_{n-i+1}的存在或不存在联系起来，$i = 1, \ldots, n$。

7.

算法 *BitstringsRec*(n)

//以递归方式生成给定长度的所有位串

//输入：一个正整数n

//输出：作为全局数组$B[0 \ldots n-1]$的内容的所有长度为n的位串

if $n = 0$
    print(B)
else
    $B[n-1] \leftarrow 0$; *BitstringsRec*$(n-1)$
    $B[n-1] \leftarrow 1$; *BitstringsRec*$(n-1)$

8.

算法 *BitstringsNonrec*(n)

//以非递归方式生成给定长度的所有位串

//输入：一个正整数n

//输出：作为全局数组$B[0 \ldots n-1]$的内容的所有长度为n的位串

for $i \leftarrow 0$ **to** $n-1$ **do**
    $B[i] = 0$
repeat
    print(B)
    $k \leftarrow n-1$
    while $k \geq 0$ **and** $B[k] = 1$
        $k \leftarrow k-1$
    if $k \geq 0$
        $B[k] \leftarrow 1$
        for $i \leftarrow k+1$ **to** $n-1$ **do**
            $B[i] \leftarrow 0$
until $k = -1$

9. a. 本节末尾已给出$n = 3$时的格雷码：

000 001 011 010 110 111 101 100

遵循$BRGC(n)$算法，得到$n = 4$的二进制反射格雷码：

$L1$ 000 001 011 010 110 111 101 100

$L2$ 100 101 111 110 010 011 001 000

L 0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110 1010 1011 1001 1000

b. 跟踪生成问题陈述的4位二进制反射格雷码的非递归算法，我们得到如下结果：

i	0	1	2	3	4	5	6	7
i 的二进制	0	1	10	11	100	101	110	111
格雷码	0000	0001	0011	0010	0110	0111	0101	0100
i	8	9	10	11	12	13	14	15
i 的二进制	1000	1001	1010	1011	1100	1101	1110	1111
格雷码	1100	1101	1111	1110	1010	1011	1001	1000

10. 下面是伊恩·帕贝里（Ian Parberry）的《算法问题》（Problems on Algorithms）中的一个递归算法[Par95, p.120]。

调用$Choose(1, k)$，其中：

算法 *Choose(i, k)*

//以分量的降序生成存储在全局数组$A[1..k]$中的所有k个子集$\{i, i+1, \ldots, n\}$

//输入：一个正整数n

//输出：作为全局数组$B[0\ldots n-1]$的内容的所有长度为n的位串

if $k = 0$

    print(A)

else

    for $j \leftarrow i$ **to** $n-k+1$ **do**

        $A[k] \leftarrow j$

        $Choose(j+1, k-1)$

11. a. 按盘子大小的递增顺序从1到n对盘子进行编号。盘子的移动用n位的一个元组表示，其中的二进制位从右到左计数，使最右边的位代表最小的盘子的移动，最左边的位代表最大的盘子的移动。将元组初始化为全0。对于该谜题的解中的每一次移动，如果这一次移动涉及到第i个盘子，就翻转第i位。

b. 利用 a 部分描述的二进制反射格雷码的位串与汉诺塔谜题中的盘子移动之间的对应关系，以及当需要选择在哪里放置盘子时以下附加规则：当面临放盘子的选择时，总是将奇数盘放在偶数盘的上面；如果没有偶数盘，就将奇数盘放在一个空桩子上。类似地，将偶数盘放在奇数盘上（如果有奇数盘的话），否则放在一个空桩子上。

12. 这个问题可通过按编号为1~n的按钮的以下递归算法来解决。如果$n = 1$，且灯泡没有亮，就按下按钮1。如果$n > 1$且灯泡没有亮，就递归地按下前$n-1$个按钮。如果这不能使灯泡亮，就按下按钮n，再次递归地按下前$n-1$个按钮。

在最坏的情况下，按下按钮的次数的递推式是：

$$M(n) = 2M(n-1) + 1, \text{ 其中} n > 1, \quad M(1) = 1$$

这与 2.4 节讨论的汉诺塔谜题的递归式相同，它的解是$M(n) = 2^n - 1$。

另外，由于开关可能处于两种状态中的一种，所以可把它看作是一个n位串中的一个二进制位，其中 0 和 1 分别代表，比如说，开关的初始状态和相反的状态。这样的位串（开关配置）的总数等于2^n；其中一个代表初始状态，其余$2^n - 1$个位

串包含将点亮灯泡的状态。在最坏情况下，所有$2^n - 1$个开关组合都必须被检查。为了用最少的按动按钮次数来完成这个任务，每次按动都必须生成一个新的开关组合。具体地说，我们可像下面这样利用二进制反射格雷码。将开关从右到左从1到n编号，并使用格雷码的位串序列来指导该按哪个按钮：如果下一个位串的右数第i位有别于紧邻的前一个位串（直接前趋），就按第i个按钮。例如，对于$n = 4$，格雷码是

0000 0001 0011 0010 0110 0111 0101 0100
1100 1101 1111 1110 1010 1011 1001 1000

它指导我们按以下次序按动按钮：

121312141213121

习题 4.4

1. **切割木棍** 一根 n 英寸长的木棍需要切割成 n 段 1 英寸长的小段。如果一次能将木棍切成几段，描述以最小切割次数完成该任务的算法。同时给出最小切割次数的公式。

2． 设计一个减半算法来计算$\lfloor log_2 n \rfloor$，并确定它的时间效率。

3． **a．**在下面的数组中查找一个键时，折半查找最多需要进行多少次键比较？

3	14	27	31	39	42	55	70	74	81	85	93	98

b．列出对数组进行折半查找时需要最多键比较次数的所有键。

c．在对该数组折半查找成功的前提下，求键比较的平均次数(假设查找每一个键的概率都是相同的)。

d．在对该数组折半查找失败的前提下，求键比较的平均次数(假设查找键位于该数组构成的 14 个区间内的概率都是相同的)。

4． 请估计一下，对于一个包含一百万个元素的有序数组进行成功查找，折半查找比顺序查找平均快多少倍？

5． 无论是用数组还是用链表实现一个列表，使用顺序查找的效率都是基本相同的。这对于折半查找来说也成立吗？

6． **a．**设计一个只使用两路比较的折半查找的版本，例如只用≤和=。可以任选一种语言来实现，并认真地调试：众所周知，这类程序很容易有错误。

b．对于 a 所设计的两路比较算法，分析其时间效率。

7． **猜图片** 一个非常流行的解题游戏是这样的：向选手出示 42 张图片，每行 6 张，共 7 行。选手可以提出一些是非题来确定目标图片。进一步要求选手用尽可能少的问题来确定目标图片。给出解决该问题的最有效的算法，并指出需要提问的最大次数。

8．请考虑**三重查找**(ternary search)——就是下面这个查找有序数组 $A[0..n-1]$的算法：如果$n = 1$，就把数组中的唯一元素和查找键 K 进行比较。否则，通过比较 K 和$A[\lfloor n/3 \rfloor]$来进行递归查找。如果 K 较大，把它和$A[\lfloor 2n/3 \rfloor]$进行比较，以确定在数

组的三段中的哪一段中继续查找过程。

a . 该算法是以哪种算法思想为基础的？

b . 为最差情况下的键比较次数建立一个递推式(我们可以假设 $n = 3^k$)。

c . 在 $n = 3^k$ 的情况下解该递推式。

d . 将该算法的效率和折半查找的进行比较。

9. 一个数组$A[0..n-2]$包含 $n-1$ 个从 1 到 n 依次递增的整数(因此在这个范围内缺失了一个整数)。尽你所能设计一个求缺失整数的最高效算法，并说明它的时间效率。

10 . a . 为假币问题的三分算法写一段伪代码。请确保该算法会正确处理所有的n值，而不仅仅是那些 3 的倍数。

b . 为假币问题的三分算法的称重次数建立一个递推关系，并在 $n = 3^k$ 的情况下对它求解。

c . 当n的值非常大时，该算法要比把硬币分成两堆的算法快多少倍？这个答案应该与n无关。

11 . a . 应用俄式乘法来计算 26×47。

b . 从时间效率的角度看，我们用俄式乘法算法计算$n \times m$和$m \times n$有区别吗？

12 . a . 为俄式乘法算法编写伪代码。

b . 说明俄式乘法的时间效率类型。

13 . 求$J(40)$ ——在$n = 40$的情况下，约瑟夫斯问题的解。

14 . 请证明，对于所有为 2 的乘方的n来说，它的约瑟夫斯问题的解是 1。

15 . ▶对于约瑟夫斯问题：

a . 当$n = 1, 2, \ldots, 15$时，计算$J(n)$。

b . 通过观察，从前 15 个n值的解中发现一个模式，然后证明它在一般情况下的正确性。

c . 有一种做法是将n的二进制表示向左循环移一位来得到$J(n)$，证明它的正确性。

习题 4.4 提示

1 . 需要关心当前最长段的长度。

2 . 如果一个规模为 n 的实例的计算时间是$\lfloor log_2 n \rfloor$，规模为$n/2$的实例的计算时间需要多少？两者是何关系？

3 . 对于 a，利用公式可以立即得到答案。回答 b、c、d 的最有效手段是二叉查找树，该树映射了该算法查找任意键值时的操作。

4 . 请比较在成功查找时，顺序查找与折半查找的平均键比较次数的比率。

5 . 如何访问一个链表的中间元素？

6 . a . 当$m \leftarrow \lfloor (l + r)/2 \rfloor$，比较$K \leq A[m]$，直到$l = r$。然后检查查找是否成功。

b . 本题的分析与课本中折半查找的分析几乎完全相同。

7 . 对图片编号，然后在提问中使用编号。

8 . 显然，这个算法和折半查找很相似。在最坏的情况下，每次迭代时要做多少次键

比较，数组中尚需处理的元素占多少比例？

9． 从比较中间元素$A[m]$和$m+1$开始。

10． 虽然当$n \bmod 3 = 0$或$n \bmod 3 = 1$时，需要怎么处理是显而易见的的，但在$n \bmod 3 = 2$时的做法就没有那么明显了。

11．a． 对于给定的数字跟踪该算法，就像课本中对另一个输入的做法(参见图 4.14(b))。

b． 该算法要做多少次迭代？

12． 既可以用递归也可以用非递归来实现这个算法。

13． 得到答案的最快的方法就是使用本节结尾提到的公式，它利用了n的二进制表示。

14． 使用n的二进制表示。

15．a． 对于课本中给定的递推方程使用前向替换法(参见附录 B)。

b．从 a 部分给出的前 15 个n的值中观察出一个模式，用解析法表示。然后用数学归纳法证明其正确性。

c．从n的二进制表示开始，将 b 部分得到的$J(n)$的公式转换成二进制表示。

习题 4.4 答案

1. 由于能一次将给定的棍子切成几段，所以只需要关注将目前最长一段的长度减少到1的切割算法。这意味着在每次迭代中，最佳算法必须将最长的一段（同时其他长度大于1的所有段）削减一半（或尽可能接近一半）。换言之，它将每一段长度为$l > 1$的木棍分别切割成长度为$\lceil l/2 \rceil$和$\lfloor l/2 \rfloor$的两段。迭代在最长的一段（进而其他所有段）长度为1之后停止。该优化算法对n个单位的棍子的切割（迭代）次数等于$\lceil \log_2 n \rceil$，即满足$2^k \geq n$的最小的k。更正式地说，切割次数$C(n)$可通过解以下递推式得到：

$$C(n) = C(\lceil n/2 \rceil) + 1，\text{其中} n > 1，\ C(1) = 0$$

2.

算法 *LogFloor*(*n*)

//输入：一个正整数

//输出：返回 $\lfloor \log_2 n \rfloor$

if $n = 1$ **return** 0

else return $LogFloor(\lfloor \frac{n}{2} \rfloor) + 1$

该算法与 2.4 节讨论的计算二进制位数的算法几乎相同。加法次数的递归关系是：

$$A(n) = A(\lfloor n/2 \rfloor) + 1, \text{其中} n > 1, \ A(1) = 0$$

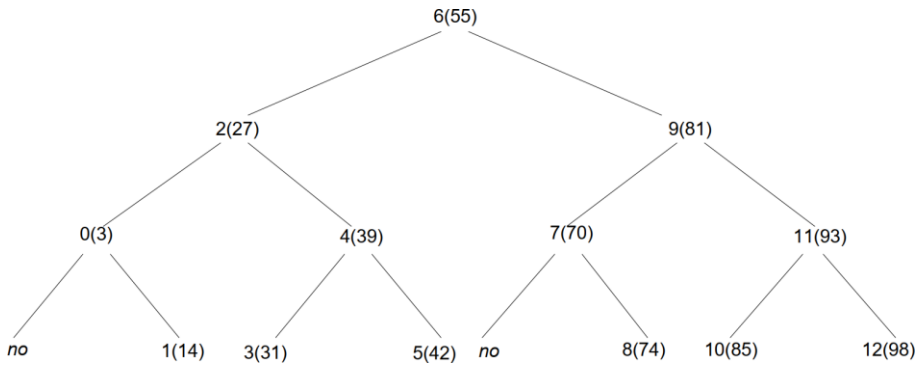
它的解是：

$$A(n) = \lfloor \log_2 n \rfloor \in \Theta(\log n)$$

3. a. 根据公式(4.5)：

$$C_{worst}(13) = \lceil \log_2(13+1) \rceil = 4$$

b. 在下面的比较树中，第一个数字是元素的索引，第二个数字是元素的值。



对树的最后一层的每个元素的查找——即位置 1（14），3（31），5（42），8（74），10（85），和 12（98）的这些元素——需要最大次数的键比较。

c. $C_{avg}^{yes} = \frac{1}{13} \cdot 1 \cdot 1 + \frac{1}{13} \cdot 2 \cdot 2 + \frac{1}{13} \cdot 3 \cdot 4 + \frac{1}{13} \cdot 4 \cdot 6 = \frac{41}{13} \approx 3.2$

d. $C_{avg}^{no} = \frac{1}{14} \cdot 3 \cdot 2 + \frac{1}{14} \cdot 4 \cdot 12 = \frac{54}{14} \approx 3.9$

4. 对于成功的查找，题中描述的比率可以这样估计：

$$\frac{C_{avg}^{seq.}(n)}{C_{avg}^{bin.}(n)} \approx \frac{n/2}{\log_2 n} = (\text{其中} n = 10^6)\frac{10^6/2}{\log_2 10^6} = \frac{1}{2 \cdot 6}\frac{10^6}{\log_2 10} \approx 25\ 000$$

5. 数组中任何元素都可以在恒定的时间内被访问。相反，到达链表的中间元素是一个$\Theta(n)$的操作。所以，尽管原则上也能用，但对已排好序的链表进行折半查找时，算法的效率非常低下。

6. a. 下面是算法的伪代码。

算法 $TwoWayBinarySearch(A[0..n-1], K)$

//用两路比较实现折半查找

//输入：一个有序数组$A[0..n-1]$和一个查找键K

//输出：等于K的一个数组元素的索引；如果没有这样的一个元素，则返回-1

$l \leftarrow 0$; $r \leftarrow n-1$

while $l < r$ **do**

  $m \leftarrow \lfloor (l+r)/2 \rfloor$

  if $K \leq A[m]$

    $r \leftarrow m$

  else $l \leftarrow m+1$

if $K = A[l]$ **return** l

else return -1

b. *TwoWayBinarySearch* 算法在最坏情况下进行$\lceil \log_2 n \rceil + 1$次双向比较，这是由解$n > 1$时的递归式$C_w(n) = C_w(\lceil n/2 \rceil) + 1$得到的，$C_w(1) = 1$。还要注意的是，这个算法最佳情况下的效率不是$\Theta(1)$，而是$\Theta(log\ n)$。

7. 使用图片编号应用折半查找的双向比较版本。换言之，假设图片从 1 到 42 到编号。从一个问题开始，比如“图片编号是否> 21？”。可能需要的最大问题数是 6。（因为可假定查找肯定成功，所以比$TwoWayBinarySearch$需要的比较次数少，这里得出$\lceil \log_2 42 \rceil = 6$）。

8. a. 该算法基于减常量因子（等于 3）策略。

b. $C(n) = 2 + C(n/3)$ 其中$n = 3^k$ $(k > 0)$, $C(1) = 1$

c. $C(3^k) = 2 + C(3^{k-1})$ [sub. $C(3^{k-1}) = 2 + C(3^{k-2})$]
$= 2 + [2 + C(3^{k-2})] = 2 \cdot 2 + C(3^{k-2}) =$ [sub. $C(3^{k-2}) = 2 + C(3^{k-3})$]
$= 2 \cdot 2 + [2 + C(3^{k-3})] = 2 \cdot 3 + C(3^{k-3}) = \ldots = 2i + C(3^{k-i}) = \ldots =$
$2k + C(3^{k-k}) = 2\log_3 n + 1$
（sub.代表子公式）

d. 必须将这个公式与折半查找中最坏情况下的键比较次数进行比较，后者大约是$\log_2 n + 1$。由于：

$$2\log_3 n + 1 = 2\frac{\log_2 n}{\log_2 3} + 1 = \frac{2}{\log_2 3}\log_2 n + 1$$

而且$2/\log_2 3 > 1$，折半查找有一个较小的乘法常数，因此在最坏情况下效率更高（约为$2/\log_2 3$的因数），尽管这两种算法都属于同一对数类别。

9. 这个问题可通过减半算法来解决，它基于以下观察。将中间的元素$A[m]$与$m+1$进行比较：如果$A[m] = m + 1$，则缺失的数字大于$m + 1$，因此应该在数组的后半部分寻找；否则（即如果$A[m] > m + 1$），应该在数组的前半部分寻找。下面是该算法的非递归版本的伪代码。

算法 $MissingNumber(A[0..n-2])$
//输入：范围从1到n，按递增顺序排列的一个数组，其中只有$n-1$个元素
//输出：不在数组中的一个整数（值在1到n之间）
$l \leftarrow 0$; $r \leftarrow n - 2$
while $l < r$ **do**
    $m \leftarrow \lfloor (l + r)/2 \rfloor$
    if $A[m] = m + 1$
        $l \leftarrow m + 1$
    else $r \leftarrow m - 1$
if $A[l] = l + 1$ **return** $l + 2$
else return $l + 1$

注：用$n(n+1)/2$与数组元素之和的差来计算缺失数字的算法显然是线性的，所以效率不如上面给出的对数算法。但它的优点是不要求数组元素已排好序。

10. a. 如果n是3的倍数（即$n \bmod 3 = 0$），我们可将硬币分成三堆，每堆$n/3$个硬币，并称量其中两堆。如果$n = 3k + 1$（即$n \bmod 3 = 1$），可将硬币分成大小为k，k和$k+1$或$k+1$、$k+1$和$k-1$的三堆（我们使用第二个选项）。最后，如果$n = 3k + 2$（即$n \bmod 3 = 2$），我们将硬币分成大小为$k+1$，$k+1$和k的堆。以下伪码假设在给定硬币中正好有一个假币，而且这个假币比其他硬币轻。

if $n = 1$ 发现假币
else 将硬币分为三堆：分别有$\lceil n/3 \rceil$，$\lceil n/3 \rceil$和$n - 2\lceil n/3 \rceil$ 枚硬币，
对前两堆称重
if 重量相同
放弃它们，继续处理第三堆硬币
else 从前两堆较轻的那一堆继续

b. 在最坏的情况下，所需的称重次数$W(n)$的递归关系如下：

$$W(n) = W(\lceil n/3 \rceil) + 1，其中 n > 1,\ \ W(1) = 0$$

如果$n = 3k$，递归关系变成$W(3^k) = W(3^{k-1}) + 1$。用反向替换法求解，得到$W(3^k) = k = \log_3 n$。

c. 在最坏情况下，对于大的n值，两者的称重次数之比可以近似地表示为：

$$\frac{\log_2 n}{\log_3 n} = \frac{\log_2 n}{\log_3 2 \log_2 n} = \log_2 3 \approx 1.6$$

11. a. 用俄式乘法来计算 26×47：

n	m	
26	47	
13	94	94
6	188	
3	376	376
1	752	752
		1 222

b. 俄式乘法在计算$n \times m$时进行了$\lfloor \log_2 n \rfloor$次迭代，计算$m \times n$时进行了$\lfloor \log_2 m \rfloor$次迭代。

12. a. 下面展示了俄式乘法的非递归和递归实现的伪代码：

算法 $Russe(n, m)$
//非递归地实现俄式乘法
//输入：两个正整数n和m
//输出：n和m的乘积
$p \leftarrow 0$
while $n \neq 1$ **do**
    if $n \bmod 2 = 1 \quad p \leftarrow p + m$
    $n \leftarrow \lfloor n/2 \rfloor$
    $m \leftarrow 2 * m$
return $p + m$

算法 $RusseRec(n, m)$
//递归地实现俄式乘法
//输入：两个正整数n和m
//输出：n和m的乘积
if $n \bmod 2 = 0$ **return** $RusseRec(n/2, 2m)$
else if $n = 1$ **return** m
else return $RusseRec((n-1)/2, 2m) + m$

b. 俄式乘法的时间效率类型是$\Theta(log\ n)$，其中n是乘积的第一个因数。作为b（n的二进制位数）的函数，它是$\Theta(b)$。

13. 利用$J(n)$可通过n向左循环移动 1 位来获得这一事实，我们得到以下$n = 40$时的结果：

$$J(40) = J(101000_2) = 10001_2 = 17$$

14. 还是利用$J(n)$可通过n向左循环移动 1 位来获得这一事实。如果$n = 2^k$，其中k是一个非负的整数，那么$J(2^k) = J\left(1\underbrace{0 \ldots 0}_{k\text{个零}}\right) = 1$。

15. a. 利用初始条件$J(1)=1$和分别针对偶数和奇数n值的递归式$J(2k)=2J(k)-1$和$J(2k+1)=2J(k)+1$，我们得到以下$J(n)$的值，其中$n=1，2，\ldots，15$：

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$J(n)$	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15

b. 检查(a)部分得到的值不难发现，对于在2的连续次方之间的n值，即对于$2^k \le n < 2^{k+1}$（$k=0，1，2，3$）或$n=2^k+i$，其中$i=0,1,\ldots,2^k-1$，相应的$J(n)$值是从1到$2^{k+1}-1$的奇数，这一观察结果可用以下公式表示：

$$J(2^k+i)=2i+1，\ 其中 i=0,1,...,2^k-1$$

我们将通过对k的归纳来证明这个公式可解决任何非负整数k的约瑟夫斯问题的递归式。对于基值$k=0$，我们有$J(2^0+0)=2\times 0+1=1$，因为它应该是初始条件。假设对于一个给定的非负整数k和每个$i=0,1,\ldots,2^k-1$，$J(2^k+i)=2i+1$，我们需要证明：

$$J(2^{k+1}+i)=2i+1，\ 其中\ i=0,1,...,2^{k+1}-1$$

如果i是偶数，它可以表示成$2j$，其中$j=0,1,\ldots,2^k-1$。然后我们得到：

$$J(2^{k+1}+i)=J(2(2^k+j))=2J(2^k+j)-1$$

并且，利用归纳法的假设，我们可以像下面这样继续：

$$2J(2^k+j)-1=2[2j+1]-1=2i+1$$

如果i是奇数，它可以表示为$2j+1$，其中$0\le i<2^k$。 然后我们得到：

$$J(2^{k+1}+i)=J(2^{k+1}+2j+1)=J(2(2^k+j)+1)=2J(2^k+j)+1$$

并且，利用归纳法的假设，我们可以像下面这样继续：

$$2J(2^k+j)+1=2[2j+1]+1=2i+1$$

c. 设$n=(b_k b_{k-1}\ldots b_0)_2$，其中第一个二进制位$b_k$是 1。在(b)部分使用的$n$的表示中，$n=2^k+i$，其中$i=(b_{k-1}\ldots b_0)_2$。此外，正如(b)部分所证明的那样：

$$J(n)=2i+1=(b_{k-1}...b_0 0)_2+1=(b_{k-1}...b_0 1)_2=(b_{k-1}...b_0 b_k)_2$$

它是$n=(b_k b_{k-1}\ldots b_0)_2$的向左一位循环移位。

注：题 15 的解来自葛立恒（Graham, R.L.）、高德纳（Knuth, D.E.）和欧伦·帕塔许尼克（Patashnik, O.）所著的《Concrete Mathematics: a Foundation for Computer Science》（具体数学：计算机科学中的一块基石），第 2 版，Addison-Wesley，1994。

习题 4.5

1． **a.** 如果用第二个参数n的规模来度量m和n的最大公约数问题的实例规模，在用欧几里得算法计算 gcd (*m*, *n*)时，实例的规模会消减多少？

b. 请证明，在欧几里得算法做了两次连续的迭代以后，实例的规模总是会至少消去一个大于 2 的因子。

2． 应用快速选择算法求数列$9, 12, 5, 17, 20, 30, 8$的中位数。

3． 请写出非递归快速选择算法的伪代码。

4． 推导插值查找算法的公式。

5． 给出插值查找最差输入的一个例子，并说明该算法在最差情况下是线性的。

6． **a．**为了使$log_2 log_2 n + 1$大于 6，n的最小值是多少？

b．下列断言中哪些是正确的？

i．$loglog\, n \in o(log\, n)$　**ii.** $loglog\, n \in \Theta(log\, n)$　**iii.** $loglog\, n \in \Omega(log\, n)$

7． **a．**描述一个在二叉查找树中寻找最大键的算法。你会把你的算法归类为减可变规模算法吗？

b．你的算法在最坏情况下的效率类型是哪一种？

8． **a．**描述一个在二叉查找树中删除一个键的算法。你会把你的算法归类为减可变规模算法吗？

b．你的算法在最坏情况下的效率类型是哪一种？

9． 对于所有顶点的度（degree，相邻节点的数量）都是偶数的连通图，给出构造其欧拉回路的减可变规模算法。

10．Misère 单堆拈游戏　来考虑单堆拈游戏的 Misère 版本，它规定谁拿走最后一个棋子就输了。游戏的其他条件都不变，即该堆棋子有n个，每次每个玩家最多拿走m个，最少拿走 1 个棋子。请指出该游戏的胜局和败局(对于接下来要走的玩家来说)是怎样的？

11．a．变质的巧克力　两个玩家轮流掰一块$m \times n$格的巧克力，其中有一个1×1格的小块是坏的。每次掰只能顺着方格的边界，沿直线一掰到底。每轮到谁掰，他就将掰下来的不含变质巧克力的那一块吃掉，谁最后掰来的巧克力包含坏的那一小块就输了。在这个游戏中，先走好还是后走好？

b．写一个互动程序和计算机玩这个游戏。程序在胜局应该走出致胜一步，在败局中则只要随机下出合规的一步就好。

12．▷翻煎饼　有n张大小各不相同的煎饼，一张叠在另一张上面。允许将一个翻板插到其中一张煎饼下面，然后可以把板上方的这叠煎饼翻个身。目标是根据煎饼的大小重新安排它们的位置，最大的饼要在最下面。可在网站“Interactive Mathematics Miscellany and Puzzles”(交互式的数学杂题和智力游戏)上找到该游戏的一个动态演示([Bog])。设计一个算法来解这个谜题。

13．▷假设需要在一个$n \times n$矩阵中搜索一个给定数字，该矩阵每行每列都按升序排列。你能为这个问题设计一个$O(n)$算法吗？([Laa10])

习题 4.5 提示

1． **a．**从支撑欧几里得算法的公式中可以立即得到答案。

b．设$r = m\ mod\ n$。研究r值和n值关系的两种情况。

2． 遵循本节对另一个输入的做法，对给定的输入跟踪该算法。

3． 本节例题的一个具体实例应用了该算法的非递归版本。

4． 写出穿过点$(l, A[l])$和$(r, A[r])$的直线公式，然后对于该直线上 y 坐标为v 的点，求

出它的 x 坐标。

5． 构造一个数组，使得插值查找在每次迭代时只能消去剩余子数组的一个元素。

6． **a．**解不等式$log_2 log_2 n + 1 > 6$。

b．计算$\lim_{n\to\infty} \frac{log log n}{log n}$。请注意，加上一个乘数常量以后，我们可以把这个对数当作自然对数，也就是以e为底的对数。

7． **a．**这个算法是由二叉查找树的定义确定的。

b．该算法的最坏情况是什么？对于这样一种输入，它要做多少次键比较？

8． **a．**分别考虑三种情况：(1)这个键所在的节点是叶子，(2)这个键所在的节点有一个子女，(3)这个键所在的节点有两个子女。

b．假设我们知道要被删除的键的位置。

9． 从图的一个任意顶点开始，将未遍历的边遍历一遍，直到所有边被遍历或者没有未遍历边存在即可。

10．本节在分析该游戏的标准版本时用了什么方法？

11．在纸上先玩几轮以熟悉这个游戏。再考虑一下坏巧克力的某些特殊位置可以帮助我们求解该问题。

12．为了得到比较好的效果，试着依靠自己的能力设计一个算法。这个算法不必是最优的，但效率应该比较高。

13．一开始，比较查找元素和第一行最后一个元素。

习题 4.5 答案

1. a. 由于该算法使用了公式$gcd(m,n) = gcd(n, m \bmod n)$，新对的大小将是$m \bmod n$，因此它可以是0~$n-1$之间的任何整数。所以，$n$可递减1~$n$之间的任何数字。

   b. 欧几里德算法的两次连续迭代是根据以下公式进行的：

   $$\gcd(m,n) = \gcd(n,r) = \gcd(r, n \bmod r)，其中 r = m \bmod n$$

   我们需证明$n \bmod r \le n/2$。考虑两种情况：$r \le n/2$和$n/2 < r < n$。如果$r \le n/2$，那么：

   $$n \bmod r \le n/2。$$

   如果$n/2 < r < n$，那么：

   $$n \bmod r = n - r < n/2。$$

2. 由于$n = 7$，$k = \lceil 7/2 \rceil = 4$，而且$k - 1 = 3$。对数列$9, 12, 5, 17, 20, 30, 8$进行快速选择，我们得到以下划分：

0	1	2	3	4	5	6
9	12	5	17	20	30	8
s		i				
9	5	12	17	20	30	8
	s					i
9	5	8	17	20	30	12
		s				
8	5	**9**	17	20	30	12

0	1	2	3	4	5	6
s	i					
9	12	5	17	20	30	8
s		i				
9	12	5	17	20	30	8
	s		i			
9	5	12	17	20	30	8
	s					i
9	5	12	17	20	30	8
		s				
9	5	8	17	20	30	12
8	5	**9**	17	20	30	12

由于$s = 2 < k-1$，我们继续数列的右边部分：

0	1	2	3	4	5	6
8	5	9	**17**	20	30	12
			s			i
			17	12	30	20
				s		i
			12	**17**	30	20

0	1	2	3	4	5	6
			s	i		
			17	20	30	12
			s			i
			17	20	30	12
				s		
			17	12	30	20
			12	**17**	30	20

由于$s = 4 > k-1$，我们从数列的左边部分继续，其中只有一个元素 12，是以下数列的中位数：

0	1	2	3	4	5	6
			s			
8	5	9	**12**	20	30	20

3. a.

算法$Quickselect(A[0..n-1], k)$

//通过基于划分的算法解快速选择问题

//输入：包含可排序元素的数组$A[0..n-1]$以及整数k $(1 \le k \le n)$

//输出：数组$A[0..n-1]$中的第k个最小元素的值

$l \leftarrow 0$; $r \leftarrow n-1$

$A[n] \leftarrow \infty$ //append sentinel

while $l \le r$ **do**

  $p \leftarrow A[l]$ //the pivot

  $i \leftarrow l$; $j \leftarrow r+1$

  repeat

    repeat $i \leftarrow i+1$ **until** $A[i] \ge p$

    repeat $j \leftarrow j-1$ **until** $A[j] \le p$ **do**

    swap$(A[i], A[j])$

  until $i \ge j$

  swap$(A[i], A[j])$ //undo last swap

  swap$(A[l], A[j])$ //partition

  if $j > k-1$ $r \leftarrow j-1$

  else if $j < k-1$ $l \leftarrow j+1$

  else return $A[k-1]$

b. 调用$QuickselectRec(A[0..n-1], k)$，其中：

算法 $QuickselectRec(A[l..r],\ k)$

//通过递归的、基于划分的算法解快速选择问题
//输入：包含可排序元素的子数组$A[l..r]$以及整数k $(1 \le k \le r-l+1)$
//输出：数组$A[l..r]$中第k个最小的元素的值
$s \leftarrow Partition(A[l..r])$ //参见 4.5节; 如果$l=r$,肯定返回l
if $s > l+k-1$ $QuickselectRec(A[l..s-1], k)$
else if $s < l+k-1$ $QuickselectRec(A[s+1..r], k-1-s)$
else return $A[s]$

4. 使用已知两点求直线方程的一般式，我们得到：

$$y - A[l] = \frac{A[r]-A[l]}{r-l}(x-l)$$

将给定的值v代入y，对第二项进行必要的四舍五入以保证索引l为整数后，求解x的方程得到：

$$x = l + \lfloor \frac{(v-A[l])(r-l)}{A[r]-A[l]} \rfloor$$

5. 如果$v = A[l]$或$v = A[r]$，公式（4.4）将分别生成$x = l$和$x = r$，在将v与$A[x]$比较后，对v的查找将成功停止。如果$A[l] < v < A[r]$，那么：

$$0 < \frac{(v-A[l])(r-l)}{A[r]-A[l]} < r-l$$

所以：

$$0 \le \lfloor \frac{(v-A[l])(r-l)}{A[r]-A[l]} \rfloor \le r-l-1$$

即：

$$l \le l + \lfloor \frac{(v-A[l])(r-l)}{A[r]-A[l]} \rfloor \le r-1$$

所以，如果插值查找在当前迭代中没有停止，它至少会将剩余的待检查的数组的大小减 1。因此，它最坏情况下的效率是$O(n)$。我们想要证明它实际是$\Theta(n)$。以数组$A[0..n-1]$为例，其中$A[0] = 0$。在$i = 1,2,\ldots,n-1$的情况下，$A[i] = n-1$。如果通过插值查找在这个数组中查找$v = n-1.5$，其第k次迭代（$k = 1,2\ldots,n$）将有$l = 0$和$r = n-k$。我们将通过对k的数学归纳来证明这一论断。事实上，对于$k = 1$，我们有$l = 0$和$r = n-1$。对于一般情况，假设该断言在某次迭代$k(1 \le k < n)$中是正确的，这样$l = 0$和$r = n-k$。在这次迭代中，我们将通过应用算法的公式得到如下结果：

$$x = 0 + \lfloor \frac{((n-1.5)-0)(n-k)}{(n-1)-0} \rfloor$$

由于：

$$\frac{(n-1.5)(n-k)}{(n-1)} = \frac{(n-1)(n-k)-0.5(n-k)}{(n-1)} = (n-k) - 0.5\frac{(n-k)}{(n-1)} < (n-k)$$

即：

$$\frac{(n-1.5)(n-k)}{(n-1)} = (n-k) - 0.5\frac{(n-k)}{(n-1)} > (n-k) - \frac{(n-k)}{(n-1)} \ge (n-k) - 1,$$

$$x = \lfloor \frac{(n-1.5)(n-k)}{(n-1)-0} \rfloor = (n-k) - 1 = n - (k+1)$$

所以$A[x] = A[n-(k+1)] = n-1$（除非$k = n-1$），这意味着在下一次（$k+1$）迭代中，$l = 0$，而且$r = n-(k+1)$（如果$k = n-1$，该断言对下一次和最后一次迭代也成立：$A[x] = A[0] = 0$，意味着$l = 0$ *和*$r = 0$）。

6. a. 可像下面这样解不等式$log_2 log_2 n + 1 > 6$。

$$\begin{aligned} \log_2 \log_2 n + 1 &> 6 \\ \log_2 \log_2 n &> 5 \\ \log_2 n &> 2^5 \\ n &> 2^{32} \ (> 4 \cdot 10^9) \end{aligned}$$

b. 使用公式$log_a n = log_a e \ln n$，可以像下面这样计算出极限：

$$\lim_{n\to\infty} \frac{\log_a \log_a n}{\log_a n} = \lim_{n\to\infty} \frac{\log_a e \ln(\log_a e \ln n)}{\log_a e \ln n} = \lim_{n\to\infty} \frac{\ln \log_a e + \ln \ln n}{\ln n}$$

$$= \lim_{n\to\infty} \frac{\ln \log_a e}{\ln n} + \lim_{n\to\infty} \frac{\ln \ln n}{\ln n} = 0 + \lim_{n\to\infty} \frac{\ln \ln n}{\ln n}$$

第二个极限可用洛必达法则(L'Hôpital's rule)算出：

$$\lim_{n\to\infty} \frac{\ln \ln n}{\ln n} = \lim_{n\to\infty} \frac{[\ln \ln n]'}{[\ln n]'} = \lim_{n\to\infty} \frac{(1/\ln n)(1/n)}{1/n} = \lim_{n\to\infty} (1/\ln n) = 0$$

因此，$loglogn \in o\ (logn)$。

7. a. 以递归方式到右子树，直至到达右子树为空的一个节点；返回该节点的键。可将这个算法看成是一个减可变规模算法：每向右一步，都会得到同一个问题的较小实例（无论用高度还是节点数来衡量树的规模）。

b. 该算法最坏情况下的效率是线性的；我们应期望其平均情况下的效率是对数的（参见 4.5 节的讨论）。

8. a. 这是一个重要和著名的算法。情况 1：如果要删除的键在叶（这种节点无子）中，将它的父节点到该键的节点的指针变成空(如果没有父节点，即它是一个单节点树的根，则使该树为空)。 情况 2：如果要删除的键在有一个单子的节点中，让它的父节点到键的节点的指针指向该子节点(如果要删除的节点是带有单子的根，则使其子节点成为新的根)。 情况 3：如果要删除的键 *K* 在有两个子的节点中，删除它可像下面这样分三步进行。首先，在 *K* 的节点的右子树中找到最小的键 *K'*。(*K'*是给定二叉树的中序遍历中的直接后继；也可通过从 *K* 的节点向右走一步，然后一直向左走，直至到达一个没有左子树的节点来找到它）。第二，交换 *K* 和 *K'*。第三，根据情况 1 或情况 2，在其新的节点上删除 *K*，具体取决于该节点是叶还是有一个子。

这不是一个减可变规模算法，因为它不是通过将问题减少到从较小的二叉树中删除一个键来工作的。

b. 作为最坏情况下输入的一个例子，考虑从连续插入键2，1，n，$n-1$，...，3得到的二叉树中删除根的任务。由于找到右子树中最小的键需要走$n-2$个指针的一个链条，删除算法的最坏情况下的效率是$\Theta(n)$。由于由n个随机键构建的二叉树的平均高度是一个对数函数（参见第 5.6 节），我们应期望删除算法的平均效率也是对数的。

9. 从图的一个任意顶点开始，遍历其未遍历的边的序列，直至到达的一个顶点没有

未遍历的边。遍历的路径是一个回路 C。如果 C 包括图的所有边，问题就解决了。否则，就从图中删除回路 C。剩下的子图 G'将是连通的，而且只有偶数度的顶点。找到一个既属于 C 又属于 G'的顶点 v（这种顶点必然存在）。从顶点 v 开始，以递归方式找到子图 G'的欧拉回路，并将 C 拼接到其中，从而得到给定图的欧拉回路。

10. 如果$n = 1$，根据 Misère 游戏的定义，玩家 1（先走棋的玩家）输了，因为他/她别无选择，只能拿最后一个棋子。如果$2 \le n \le m + 1$，玩家 1 通过拿走$n - 1$个棋子让玩家 2 剩下一个棋子而获胜。如果$n = m + 2 = 1 + (m + 1)$，玩家 1 输了，因为随便怎么走（只要合规）都会使玩家 2 处于胜局。如果$m + 3 \le n \le 2m + 2$（即$2 + (m + 1) \le n \le 2(m + 1)$），玩家 1 可通过取走$(n - 1) mod (m + 1)$个棋子让玩家 2 有$m + 2$个棋子而获胜，这对接下来要走棋的玩家来说是一个败局。因此，当且仅当$n \, mod (m + 1) = 1$时，一局棋对玩家 1 来说是败局。否则，玩家 1 通过取走$(n - 1) mod (m + 1)$个棋子而获胜；任何偏离这一必胜策略的行为都会使对手处于必胜位置。这个解的正确性的正式证明是通过强归纳法进行的。
11. 这个问题等同于拈(尼姆)游戏，只是棋子堆变成了变质的那一格巧克力和整块巧克力的边缘之间的行、列。所以，本节概述的拈理论可供确定该游戏的胜局和相应行动。根据这一理论，当且仅当拈游戏的二进制数位之和（拈和）至少包含一个 1 时，拈游戏的实例就是胜局（对于走下一步的人）。已知自己处于胜局，胜手可按以下方法找到。用二进制位串表示各堆的棋子数，求这些位串之和，扫描结果，直到遇到第一个 1。设j是这个 1 的位置。选择位置j为 1 的那个位串，这就为了赢棋而需取走一些棋子的那一堆。为了知道这一堆应留下多少棋子，从位置j开始扫描它的位串，翻转它的二进制位，使新的二进制数位之和只包含 0。

    注：Yan Stuart 还在《Math Hysteria: Fun and Games with Mathematics》（牛津大学出版社，2004 年）讨论了这个问题的一个特例，即“Yucky Chocolate”，它的特殊之处在于变质的那一格巧克力在整块巧克力的角落。在这种情况下，如果$m = n$，即巧克力块是正方形的，先走的玩家就输了；如果$m \ne n$，就赢了。可用强归纳法来证明，其中不涉及堆大小的二进制表示。如果$m = n = 1$，根据游戏的定义，先走的一方会输。假设对于$k \le n$的每个$k \times k$的正方形巧克力块来说，这个断言都是成立的，那么考虑$n + 1 \times n + 1$的巧克力块。任何行动（即掰巧克力），都会生成一边大小为$k \le n$，另一边大小仍为$n + 1$的长方形巧克力块。第二个玩家行动时，肯定能留下一个带有变质巧克力格（在角落）的$k \times k$的正方形巧克力块。根据归纳假设，这是一个输局。而如果$m \ne n$，第一个棋手总是可以通过留下边长为$min\{m, n\}$的正方形使第二个玩家处于败局。
12. 下面是该问题的减治算法。重复以下步骤，直至问题得到解决。找到最大的那张顺序不对的饼。(如果没有找到，问题就解决了。)如果它不在饼叠的顶部，将翻板插入它的下方，然后翻转，使其跑到顶部。将翻板插入从下往上位置不对的第一个饼的下方，然后翻转，使位置正确的煎饼数量至少增加一个。

    在最坏的情况下，这个算法需要的翻转次数是$W(n) = 2n - 3$，其中$n \ge 2$是煎饼的数量。下面用数学归纳法证明。在$n = 2$的情况下，这个断言是正确的：两个饼

一叠，如果大的在上方，该算法只需进行一次翻转。如果大的在下方，该算法无需翻转。现在，假设$n \geq 2$的某个值的最坏情况下的翻转次数由公式$W(n) = 2n - 3$给出。考虑一叠任意$n + 1$个饼子。在两次或更少次翻转的情况下，该算法将最大的饼放在这叠饼子的底部，在那里它不会参与任何进一步的翻转。因此，对于一叠任意$n + 1$个饼子，所需的总翻转次数上界是：

$$2 + W(n) = 2 + (2n - 3) = 2(n + 1) - 3$$

事实上，这个上界是在一叠$n + 1$个饼上获得的，其构造如下：翻转最坏情况下的一叠n个饼，在最上面的饼和上数第二个饼之间插入一个比其他所有饼都大的饼。(对于新的这一叠饼，算法将进行两次翻转，将问题减少到翻转最坏情况下的一叠n个饼）。这就完成了以下事实的证明：

$$W(n + 1) = 2(n + 1) - 3$$

这进而完成了我们的数学归纳证明。

注：问题陈述中提到的网站，除了一个可视化的小程序外，还包含对该问题的有趣讨论。(其中一个事实是，比尔·盖茨发表的唯一一篇学术论文就是关于这个问题的。)

13. 将搜索的数字与第一行最后一个元素进行比较。 如果匹配，算法停止。如果搜索的数字比矩阵元素小，则前者不可能在矩阵最后一列，其元素可从后续搜索中排除。如果搜索的数字大于第一行最后一个元素，则前者不可能在矩阵第一行，其元素可从后续搜索中排除。 对变小的矩阵重复此步骤，直到找到匹配项或剩余矩阵缩小为空矩阵。由于在每次迭代中，该算法都从后续搜索中消除了矩阵的一行或一列，因此其时间效率类型为$O(n)$。