

TURING

图灵原版计算机科学系列

PEARSON

# 计算机程序设计艺术

## 卷2：半数值算法

(英文版·第3版)

[美] Donald E. Knuth 著

**The Art of Computer Programming**  
**Vol 2: Seminumerical Algorithms**  
**Third Edition**



人民邮电出版社  
POSTS & TELECOM PRESS

## 图书在版编目 (CIP) 数据

计算机程序设计艺术. 第2卷, 半数值算法 : 第3版  
= The Art of Computer Programming, Vol 2:  
Seminumerical Algorithms, Third Edition : 英文 /  
(美) 高德纳 (Donald, E. K.) 著. -- 北京 : 人民邮电出版社, 2010.10  
(图灵原版计算机科学系列)  
ISBN 978-7-115-23526-8

I. ①计… II. ①高… III. ①程序设计—英文 IV.  
①TP311.1

中国版本图书馆CIP数据核字(2010)第142063号

## 内 容 提 要

《计算机程序设计艺术》系列被公认为计算机科学领域的权威之作, 深入阐述了程序设计理论, 对计算机领域的发展有着极为深远的影响。本书是该系列的第2卷, 讲解半数值算法, 分“随机数”和“算术”两章。本卷总结了主要算法范例及这些算法的基本理论, 广泛剖析了计算机程序设计与数值分析间的相互联系。

本书适合从事计算机科学、计算数学等各方面工作的人员阅读, 也适合高等院校相关专业的师生作为教学参考书, 对于想深入理解计算机算法的读者, 是一份必不可少的珍品。

图灵原版计算机科学系列

### 计算机程序设计艺术 卷2: 半数值算法 (英文版·第3版)

- ◆ 著 [美] Donald E. Knuth  
责任编辑 杨海玲  
执行编辑 谢灵芝
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号  
邮编 100061 电子函件 315@ptpress.com.cn  
网址 <http://www.ptpress.com.cn>  
北京铭成印刷有限公司印刷
- ◆ 开本: 700×1000 1/16  
印张: 48.75  
字数: 936千字 2010年10月第1版  
印数: 1-2 500册 2010年10月北京第1次印刷  
著作权合同登记号 图字: 01-2010-4018号  
ISBN 978-7-115-23526-8

定价: 119.00元

读者服务热线: (010)51095186 印装质量热线: (010)67129223

反盗版热线: (010)67171154

# 版 权 声 明

Original edition, entitled *The Art of Computer Programming, Vol 2: Seminumerical Algorithms, Third Edition*, 9780201896848 by Donald E. Knuth, published by Pearson Education, Inc., publishing as Addison-Wesley, Copyright © 1998 by Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

China edition published by PEARSON EDUCATION ASIA LTD. and POSTS & TELECOM PRESS Copyright © 2010.

This edition is manufactured in the People's Republic of China, and is authorized for sale only in the People's Republic of China excluding Hong Kong, Macao and Taiwan.

本书英文版由 Pearson Education Asia Ltd. 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

仅限于中华人民共和国境内（香港、澳门特别行政区和台湾地区除外）销售发行。

本书封面贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

# 前 言

亲爱的奥菲莉娅!

这些数很让我头疼：我不知道自己叹息多少次了。

——《哈姆雷特》(第2幕,第2场,第120行)

本书讨论的算法与数值直接相关。不过我认为把它们称为半数值算法也是恰如其分的,因为它们介于数值计算和符号计算之间。这些算法不仅仅计算了数值问题的答案,还试图很好地适应数字计算机的内部运算。读者如果对计算机的机器语言没有一定的了解,很多情况下就无法充分体会到算法之美——相应的机器程序的效率是至关重要的,与算法本身密不可分。为了寻求计算机处理数值的最佳方法,我们既要考虑数值也要研究策略。因而本书的内容无疑既属于计算数学,也属于计算机科学。

有些在“较高层次”上从事数值分析工作的人可能会认为本书讨论的是系统程序员做的事情,而那些在“较高层次”上从事系统编程工作的人又会认为这些问题是数值分析人员要去考虑的。但我希望还是会有一些人愿意认真研究本书中讲解的这些基本方法。虽然这些方法显得层次较低,但它们是用计算机解决强大的数值问题的基础,因此深入了解这些方法十分重要。本书着重考虑的是计算数学与计算机程序设计之间的接口,这两类技巧的结合使得本书充满了趣味性。

与这套书的其他各卷相比,本书所讨论的内容中数学内容所占的比例明显要大很多。多数情况下,书中数学知识的讨论几乎是从零开始(或者从第1卷的结果开始)的,但有几个小节仍然需要读者具备一定的微积分知识。

本卷包含整套书中的第3章和第4章。第3章讨论“随机数”,不仅研究了生成随机序列的各种方法,还研究了随机性的统计测试,以及一致随机数到其他类型随机量的转换——后者说明了如何在实践中使用随机数。此外,我还专门用一节内容介绍了随机性本身的特性。第4章意在介绍经过数百年的发展之后,人们在算术运算上都有哪些美妙的发现。这一章讨论了多种数值表示系统以及它们之间的相互转换,还介绍了浮点数、高精度整数、有理分式、多项式及幂级数的算术运算,包括因式分解和计算最大公因子的问题。


可以以第 3 章或第 4 章的内容为基础为大学三年级到研究生层次的学生开设一学期的课程。目前许多学校都没有关于“随机数”和“算术”的课程，但我相信读者可以从这两章的内容中发现其实际教育价值。根据我自己的经验，这些课程可以很好地向大学生传授初等概率论和数论知识。这类导论性课程中涉及的内容几乎都与实际应用有着自然的关联，而这些实际应用对学习和理解相应的理论是非常重要的。此外，每章都给出了一些涉及更深入问题的提示，可以激发许多学生进一步从事数学研究的兴趣。

除少数内容与第 1 卷介绍的 MIX 计算机有关之外，本书的大部分内容都是自成一体的。附录 B 列出了本书用到的数学符号，有些与传统数学书中的符号略有不同。

## 第 3 版前言

本书的第 2 版完成于 1980 年，它可以说是  $\text{T}_{\text{E}}\text{X}$  和  $\text{M}_{\text{E}}\text{T}_{\text{A}}\text{F}_{\text{O}}\text{N}_{\text{T}}$  这两个电子排版原型系统的第一个重要测试实例。上一版启发我开发和完善了这两个电子排版系统，现在第 3 版又要问世了，我为这两个电子排版系统的成功感到由衷的高兴。最终我可以使《计算机程序设计艺术》的各卷内容保持一致的格式，并能够适应印刷技术和显示技术上的变化。新的设置使得我可以在文字上进行数千处的改进，这是我多年来一直想做的校正。

在这一新版本中，我对文字逐字进行了认真的审阅，力图在保持原有的蓬勃朝气的同时加入一些可能更成熟的论断。新版增加了几十道新的习题，并为原有的几十道习题给出了改进过的新答案。改动可谓无处不在，但最重要的改动集中在 3.5 节（关于随机性的理论保证）、3.6 节（关于可移植的随机数生成程序）、4.5.2 节（关于二进制最大公因子算法）和 4.7 节（关于幂级数的复合与迭代）。

 《计算机程序设计艺术》这套书尚未完稿，而有关半数值算法的研究也还在快速发展。因此书中有些部分还带有“建设中”的图标，以向读者致歉——这部分内容还不是最新的。我电脑中的文件里堆满了重要的材料，打算写进第 2 卷最终的壮丽无比的第 4 版中，或许从现在算起还需要 16 年的时间。但我必须先完成第 4 卷和第 5 卷，非到万不得已，我不想拖延这两卷的出版时间。

非常感谢在过去 35 年中帮助我搜集素材和改进内容的数百人。准备这一新版本的大部分艰苦工作是由 Silvio Levy 和 Jeffrey Oldham 完成的，Silvio 十分专业地编辑了本书的电子文本，Jeffrey 则几乎把所有的插图都转换成了  $\text{M}_{\text{E}}\text{T}_{\text{A}}\text{P}_{\text{O}}\text{S}_{\text{T}}$  格式。我修正了细心的读者在第 2 版中发现的所有错误（以及读者未能察觉的一些错误），并尽量避免在新增内容中引入新的错误。但我想可能仍然有一些缺陷，我希望能尽快加以改正。因此，我很乐意向首先发现技术性错误、印刷错误或历史知识错误的

人按每处错误 2.56 美元支付报酬。下列网页上列出了所有已反馈给我的错误的最新更正：<http://www-cs-faculty.stanford.edu/~knuth/taocp.html>。

高德纳 (Donald E. Knuth)

1997 年 7 月于加利福尼亚州斯坦福市

对于一本撰写时间长达 8 年的书而言，  
需要感谢的同事、打字员、学生、老师和朋友太多了。  
此外，我不想按常规做法免除他们对于书中错误的责任，  
他们应该帮我纠正这些错误！  
有时候，他们甚至需要为一些将来被证明是错误的观点负责。  
不管怎样，我要对这些共事的伙伴们表示感谢。

—— Edward F. Campbell, Jr. (1975)

“Defendit numerus” (数值是安全的) 是傻瓜的格言，  
“Deperdit numerus” (数值有毁灭性) 是智者的格言。

—— C. C. Colton (1820)

## NOTES ON THE EXERCISES

THE EXERCISES in this set of books have been designed for self-study as well as classroom study. It is difficult, if not impossible, for anyone to learn a subject purely by reading about it, without applying the information to specific problems and thereby being encouraged to think about what has been read. Furthermore, we all learn best the things that we have discovered for ourselves. Therefore the exercises form a major part of this work; a definite attempt has been made to keep them as informative as possible and to select problems that are enjoyable as well as instructive.

In many books, easy exercises are found mixed randomly among extremely difficult ones. This is sometimes unfortunate because readers like to know in advance how long a problem ought to take—otherwise they may just skip over all the problems. A classic example of such a situation is the book *Dynamic Programming* by Richard Bellman; this is an important, pioneering work in which a group of problems is collected together at the end of some chapters under the heading “Exercises and Research Problems,” with extremely trivial questions appearing in the midst of deep, unsolved problems. It is rumored that someone once asked Dr. Bellman how to tell the exercises apart from the research problems, and he replied, “If you can solve it, it is an exercise; otherwise it’s a research problem.”

Good arguments can be made for including both research problems and very easy exercises in a book of this kind; therefore, to save the reader from the possible dilemma of determining which are which, *rating numbers* have been provided to indicate the level of difficulty. These numbers have the following general significance:

### *Rating Interpretation*

- 00 An extremely easy exercise that can be answered immediately if the material of the text has been understood; such an exercise can almost always be worked “in your head.”
- 10 A simple problem that makes you think over the material just read, but is by no means difficult. You should be able to do this in one minute at most; pencil and paper may be useful in obtaining the solution.
- 20 An average problem that tests basic understanding of the text material, but you may need about fifteen or twenty minutes to answer it completely.

solve the problem by yourself, or unless you absolutely do not have time to work this particular problem. *After* getting your own solution or giving the problem a decent try, you may find the answer instructive and helpful. The solution given will often be quite short, and it will sketch the details under the assumption that you have earnestly tried to solve it by your own means first. Sometimes the solution gives less information than was asked; often it gives more. It is quite possible that you may have a better answer than the one published here, or you may have found an error in the published solution; in such a case, the author will be pleased to know the details. Later editions of this book will give the improved solutions together with the solver's name where appropriate.

When working an exercise you may generally use the answers to previous exercises, unless specifically forbidden from doing so. The rating numbers have been assigned with this in mind; thus it is possible for exercise  $n + 1$  to have a lower rating than exercise  $n$ , even though it includes the result of exercise  $n$  as a special case.

Summary of codes:	00	Immediate
	10	Simple (one minute)
	20	Medium (quarter hour)
▶ Recommended	30	Moderately hard
<i>M</i> Mathematically oriented	40	Term project
<i>HM</i> Requiring "higher math"	50	Research problem

## EXERCISES

- ▶ 1. [00] What does the rating "*M20*" mean?
2. [10] Of what value can the exercises in a textbook be to the reader?
3. [34] Leonhard Euler conjectured in 1772 that the equation  $w^4 + x^4 + y^4 = z^4$  has no solution in positive integers, but Noam Elkies proved in 1987 that infinitely many solutions exist [see *Math. Comp.* 51 (1988), 825–835]. Find all integer solutions such that  $0 \leq w \leq x \leq y < z < 10^6$ .
4. [M50] Prove that when  $n$  is an integer,  $n > 4$ , the equation  $w^n + x^n + y^n = z^n$  has no solution in positive integers  $w, x, y, z$ .

*Exercise is the beste instrument in learnyng.*

— ROBERT RECORDE, *The Whetstone of Witte* (1557)



# CONTENTS

<b>Chapter 3 — Random Numbers</b> . . . . .	1
3.1. Introduction . . . . .	1
3.2. Generating Uniform Random Numbers . . . . .	10
3.2.1. The Linear Congruential Method . . . . .	10
3.2.1.1. Choice of modulus . . . . .	12
3.2.1.2. Choice of multiplier . . . . .	16
3.2.1.3. Potency . . . . .	23
3.2.2. Other Methods . . . . .	26
3.3. Statistical Tests . . . . .	41
3.3.1. General Test Procedures for Studying Random Data . . . . .	42
3.3.2. Empirical Tests . . . . .	61
*3.3.3. Theoretical Tests . . . . .	80
3.3.4. The Spectral Test . . . . .	93
3.4. Other Types of Random Quantities . . . . .	119
3.4.1. Numerical Distributions . . . . .	119
3.4.2. Random Sampling and Shuffling . . . . .	142
*3.5. What Is a Random Sequence? . . . . .	149
3.6. Summary . . . . .	184
<b>Chapter 4 — Arithmetic</b> . . . . .	194
4.1. Positional Number Systems . . . . .	195
4.2. Floating Point Arithmetic . . . . .	214
4.2.1. Single-Precision Calculations . . . . .	214
4.2.2. Accuracy of Floating Point Arithmetic . . . . .	229
*4.2.3. Double-Precision Calculations . . . . .	246
4.2.4. Distribution of Floating Point Numbers . . . . .	253
4.3. Multiple Precision Arithmetic . . . . .	265
4.3.1. The Classical Algorithms . . . . .	265
*4.3.2. Modular Arithmetic . . . . .	284
*4.3.3. How Fast Can We Multiply? . . . . .	294
4.4. Radix Conversion . . . . .	319
4.5. Rational Arithmetic . . . . .	330
4.5.1. Fractions . . . . .	330
4.5.2. The Greatest Common Divisor . . . . .	333
*4.5.3. Analysis of Euclid's Algorithm . . . . .	356
4.5.4. Factoring into Primes . . . . .	379

4.6. Polynomial Arithmetic . . . . .	418
4.6.1. Division of Polynomials . . . . .	420
*4.6.2. Factorization of Polynomials . . . . .	439
4.6.3. Evaluation of Powers . . . . .	461
4.6.4. Evaluation of Polynomials . . . . .	485
*4.7. Manipulation of Power Series . . . . .	525
<b>Answers to Exercises</b> . . . . .	538
<b>Appendix A — Tables of Numerical Quantities</b> . . . . .	726
1. Fundamental Constants (decimal) . . . . .	726
2. Fundamental Constants (octal) . . . . .	727
3. Harmonic Numbers, Bernoulli Numbers, Fibonacci Numbers . . . . .	728
<b>Appendix B — Index to Notations</b> . . . . .	730
<b>Index and Glossary</b> . . . . .	735

## CHAPTER THREE

### RANDOM NUMBERS

*Any one who considers arithmetical  
methods of producing random digits  
is, of course, in a state of sin.*

— JOHN VON NEUMANN (1951)

*Lest men suspect your tale untrue,  
Keep probability in view.*

— JOHN GAY (1727)

*There wanted not some beams of light  
to guide men in the exercise of their Stocastick faculty.*

— JOHN OWEN (1662)

#### 3.1. INTRODUCTION

NUMBERS that are “chosen at random” are useful in many different kinds of applications. For example:

a) *Simulation*. When a computer is being used to simulate natural phenomena, random numbers are required to make things realistic. Simulation covers many fields, from the study of nuclear physics (where particles are subject to random collisions) to operations research (where people come into, say, an airport at random intervals).

b) *Sampling*. It is often impractical to examine all possible cases, but a random sample will provide insight into what constitutes “typical” behavior.

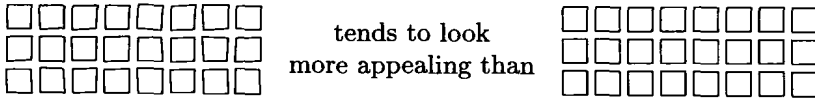
c) *Numerical analysis*. Ingenious techniques for solving complicated numerical problems have been devised using random numbers. Several books have been written on this subject.

d) *Computer programming*. Random values make a good source of data for testing the effectiveness of computer algorithms. More importantly, they are crucial to the operation of *randomized algorithms*, which are often far superior to their deterministic counterparts. This use of random numbers is the primary application of interest to us in this series of books; it accounts for the fact that random numbers are already being considered here in Chapter 3, before most of the other computer algorithms have appeared.

e) *Decision making.* There are reports that many executives make their decisions by flipping a coin or by throwing darts, etc. It is also rumored that some college professors prepare their grades on such a basis. Sometimes it is important to make a completely “unbiased” decision. Randomness is also an essential part of optimal strategies in the theory of matrix games.

f) *Cryptography.* A source of unbiased bits is crucial for many types of secure communications, when data needs to be concealed.

g) *Aesthetics.* A little bit of randomness makes computer-generated graphics and music seem more lively. For example, a pattern like



in certain contexts. [See D. E. Knuth, *Bull. Amer. Math. Soc.* 1 (1979), 369.]

h) *Recreation.* Rolling dice, shuffling decks of cards, spinning roulette wheels, etc., are fascinating pastimes for just about everybody. These traditional uses of random numbers have suggested the name “Monte Carlo method,” a general term used to describe any algorithm that employs random numbers.

People who think about this topic almost invariably get into philosophical discussions about what the word “random” means. In a sense, there is no such thing as a random number; for example, is 2 a random number? Rather, we speak of a *sequence of independent random numbers* with a specified *distribution*, and this means loosely that each number was obtained merely by chance, having nothing to do with other numbers of the sequence, and that each number has a specified probability of falling in any given range of values.

A *uniform* distribution on a finite set of numbers is one in which each possible number is equally probable. A distribution is generally understood to be uniform unless some other distribution is specifically mentioned.

Each of the ten digits 0 through 9 will occur about  $\frac{1}{10}$  of the time in a (uniform) sequence of random digits. Each pair of two successive digits should occur about  $\frac{1}{100}$  of the time, and so on. Yet if we take a truly random sequence of a million digits, it will not always have exactly 100,000 zeros, 100,000 ones, etc. In fact, chances of this are quite slim; a *sequence* of such sequences will have this character on the average.

Any specified sequence of a million digits is as probable as any other. Thus, if we are choosing a million digits at random and if the first 999,999 of them happen to come out to be zero, the chance that the final digit is zero is still exactly  $\frac{1}{10}$ , in a truly random situation. These statements seem paradoxical to many people, yet no contradiction is really involved.

There are several ways to formulate decent abstract definitions of randomness, and we will return to this interesting subject in Section 3.5; but for the moment, let us content ourselves with an intuitive understanding of the concept.

Many years ago, people who needed random numbers in their scientific work would draw balls out of a “well-stirred urn,” or they would roll dice or deal out

cards. A table of over 40,000 random digits, “taken at random from census reports,” was published in 1927 by L. H. C. Tippett. Since then, a number of devices have been built to generate random numbers mechanically. The first such machine was used in 1939 by M. G. Kendall and B. Babington-Smith to produce a table of 100,000 random digits. The Ferranti Mark I computer, first installed in 1951, had a built-in instruction that put 20 random bits into the accumulator using a resistance noise generator; this feature had been recommended by A. M. Turing. In 1955, the RAND Corporation published a widely used table of a million random digits obtained with the help of another special device. A famous random-number machine called ERNIE has been used for many years to pick the winning numbers in the British Premium Savings Bonds lottery. [F. N. David describes the early history in *Games, Gods, and Gambling* (1962). See also Kendall and Babington-Smith, *J. Royal Stat. Soc.* **A101** (1938), 147–166; **B6** (1939), 51–61; S. H. Lavington’s discussion of the Mark I in *CACM* **21** (1978), 4–12; the review of the RAND table in *Math. Comp.* **10** (1956), 39–43; and the discussion of ERNIE by W. E. Thomson, *J. Royal Stat. Soc.* **A122** (1959), 301–333.]

Shortly after computers were introduced, people began to search for efficient ways to obtain random numbers within computer programs. A table could be used, but this method is of limited utility because of the memory space and input time requirement, because the table may be too short, and because it is a bit of a nuisance to prepare and maintain the table. A machine such as ERNIE might be attached to the computer, as in the Ferranti Mark I, but this has proved to be unsatisfactory since it is impossible to reproduce calculations exactly a second time when checking out a program; moreover, such machines have tended to suffer from malfunctions that are extremely difficult to detect. Advances in technology made tables useful again during the 1990s, because a billion well-tested random bytes could easily be made accessible. George Marsaglia helped resuscitate random tables in 1995 by preparing a demonstration disk that contained 650 random megabytes, generated by combining the output of a noise-diode circuit with deterministically scrambled rap music. (He called it “white and black noise.”)

The inadequacy of mechanical methods in the early days led to an interest in the production of random numbers using a computer’s ordinary arithmetic operations. John von Neumann first suggested this approach in about 1946; his idea was to take the square of the previous random number and to extract the middle digits. For example, if we are generating 10-digit numbers and the previous value was 5772156649, we square it to get

$$33317792380594909201;$$

the next number is therefore 7923805949.

There is a fairly obvious objection to this technique: How can a sequence generated in such a way be random, since each number is completely determined by its predecessor? (See von Neumann’s comment at the beginning of this chapter.) The answer is that the sequence *isn’t* random, but it *appears* to be. In typical applications the actual relationship between one number and

its successor has no physical significance; hence the nonrandom character is not really undesirable. Intuitively, the middle square seems to be a fairly good scrambling of the previous number.

Sequences generated in a deterministic way such as this are often called *pseudorandom* or *quasirandom* sequences in the highbrow technical literature, but in most places of this book we shall simply call them random sequences, with the understanding that they only *appear* to be random. Being “apparently random” is perhaps all that can be said about any random sequence anyway. Random numbers generated deterministically on computers have worked quite well in nearly every application, provided that a suitable method has been carefully selected. Of course, deterministic sequences aren’t always the answer; they certainly shouldn’t replace ERNIE for the lotteries.

Von Neumann’s original “middle-square method” has actually proved to be a comparatively poor source of random numbers. The danger is that the sequence tends to get into a rut, a short cycle of repeating elements. For example, if zero ever appears as a number of the sequence, it will continually perpetuate itself.

Several people experimented with the middle-square method in the early 1950s. Working with numbers that have four digits instead of ten, G. E. Forsythe tried 16 different starting values and found that 12 of them led to sequences ending with the cycle 6100, 2100, 4100, 8100, 6100, . . . , while two of them degenerated to zero. More extensive tests were carried out by N. Metropolis, mostly in the binary number system. He showed that when 20-bit numbers are being used, there are 13 different cycles into which the middle-square sequence might degenerate, the longest of which has a period of length 142.

It is fairly easy to restart the middle-square method on a new value when zero has been detected, but long cycles are somewhat harder to avoid. Exercises 6 and 7 discuss some interesting ways to determine the cycles of periodic sequences, using very little memory space.

A theoretical disadvantage of the middle-square method is given in exercises 9 and 10. On the other hand, working with 38-bit numbers, Metropolis obtained a sequence of about 750,000 numbers before degeneracy occurred, and the resulting  $750,000 \times 38$  bits satisfactorily passed statistical tests for randomness. [*Symp. on Monte Carlo Methods* (Wiley, 1956), 29–36.] This experience showed that the middle-square method *can* give usable results, but it is rather dangerous to put much faith in it until after elaborate computations have been performed.

Many random number generators in use when this chapter was first written were not very good. People have traditionally tended to avoid learning about such subroutines; old methods that were comparatively unsatisfactory have been passed down blindly from one programmer to another, until the users have no understanding of the original limitations. We shall see in this chapter that the most important facts about random number generators are not difficult to learn, although prudence is necessary to avoid common pitfalls.

It is not easy to invent a foolproof source of random numbers. This fact was convincingly impressed upon the author in 1959, when he attempted to create a fantastically good generator using the following peculiar approach:

**Algorithm K** (“*Super-random*” number generator). Given a 10-digit decimal number  $X$ , this algorithm may be used to change  $X$  to the number that should come next in a supposedly random sequence. Although the algorithm might be expected to yield quite a random sequence, reasons given below show that it is not, in fact, very good at all. (The reader need not study this algorithm in great detail except to observe how complicated it is; note, in particular, steps K1 and K2.)

- K1.** [Choose number of iterations.] Set  $Y \leftarrow \lfloor X/10^9 \rfloor$ , the most significant digit of  $X$ . (We will execute steps K2 through K13 exactly  $Y + 1$  times; that is, we will apply randomizing transformations a *random* number of times.)
- K2.** [Choose random step.] Set  $Z \leftarrow \lfloor X/10^8 \rfloor \bmod 10$ , the second most significant digit of  $X$ . Go to step  $K(3 + Z)$ . (That is, we now jump to a *random* step in the program.)
- K3.** [Ensure  $\geq 5 \times 10^9$ .] If  $X < 5000000000$ , set  $X \leftarrow X + 5000000000$ .
- K4.** [Middle square.] Replace  $X$  by  $\lfloor X^2/10^5 \rfloor \bmod 10^{10}$ , that is, by the middle of the square of  $X$ .
- K5.** [Multiply.] Replace  $X$  by  $(1001001001 X) \bmod 10^{10}$ .
- K6.** [Pseudo-complement.] If  $X < 1000000000$ , then set  $X \leftarrow X + 9814055677$ ; otherwise set  $X \leftarrow 10^{10} - X$ .
- K7.** [Interchange halves.] Interchange the low-order five digits of  $X$  with the high-order five digits; that is, set  $X \leftarrow 10^5(X \bmod 10^5) + \lfloor X/10^5 \rfloor$ , the middle 10 digits of  $(10^{10} + 1)X$ .
- K8.** [Multiply.] Same as step K5.
- K9.** [Decrease digits.] Decrease each nonzero digit of the decimal representation of  $X$  by one.
- K10.** [99999 modify.] If  $X < 10^5$ , set  $X \leftarrow X^2 + 99999$ ; otherwise set  $X \leftarrow X - 99999$ .
- K11.** [Normalize.] (At this point  $X$  cannot be zero.) If  $X < 10^9$ , set  $X \leftarrow 10X$  and repeat this step.
- K12.** [Modified middle square.] Replace  $X$  by  $\lfloor X(X - 1)/10^5 \rfloor \bmod 10^{10}$ , that is, by the middle 10 digits of  $X(X - 1)$ .
- K13.** [Repeat?] If  $Y > 0$ , decrease  $Y$  by 1 and return to step K2. If  $Y = 0$ , the algorithm terminates with  $X$  as the desired “random” value. ■

(The machine-language program corresponding to this algorithm was intended to be so complicated that a person reading a listing of it without explanatory comments wouldn’t know what the program was doing.)

Considering all the contortions of Algorithm K, doesn’t it seem plausible that it should produce almost an infinite supply of unbelievably random numbers? No! In fact, when this algorithm was first put onto a computer, it almost immediately converged to the 10-digit value 6065038420, which — by extraordinary

**Table 1**  
 A COLOSSAL COINCIDENCE: THE NUMBER 6065038420  
 IS TRANSFORMED INTO ITSELF BY ALGORITHM K.

Step	$X$ (after)		Step	$X$ (after)	
K1	6065038420		K9	1107855700	
K3	6065038420		K10	1107755701	
K4	6910360760		K11	1107755701	
K5	8031120760		K12	1226919902	$Y = 3$
K6	1968879240		K5	0048821902	
K7	7924019688		K6	9862877579	
K8	9631707688		K7	7757998628	
K9	8520606577		K8	2384626628	
K10	8520506578		K9	1273515517	
K11	8520506578		K10	1273415518	
K12	0323372207	$Y = 6$	K11	1273415518	
K6	9676627793		K12	5870802097	$Y = 2$
K7	2779396766		K11	5870802097	
K8	4942162766		K12	3172562687	$Y = 1$
K9	3831051655		K4	1540029446	
K10	3830951656		K5	7015475446	
K11	3830951656		K6	2984524554	
K12	1905867781	$Y = 5$	K7	2455429845	
K12	3319967479	$Y = 4$	K8	2730274845	
K6	6680032521		K9	1620163734	
K7	3252166800		K10	1620063735	
K8	2218966800		K11	1620063735	
			K12	6065038420	$Y = 0$

coincidence—is transformed into itself by the algorithm (see Table 1). With another starting number, the sequence began to repeat after 7401 values, in a cyclic period of length 3178.

The moral of this story is that *random numbers should not be generated with a method chosen at random*. Some theory should be used.

In the following sections we shall consider random number generators that are superior to the middle-square method and to Algorithm K. The corresponding sequences are guaranteed to have certain desirable random properties, and no degeneracy will occur. We shall explore the reasons for this random-like behavior in some detail, and we shall also consider techniques for manipulating random numbers. For example, one of our investigations will be the shuffling of a simulated deck of cards within a computer program.

Section 3.6 summarizes this chapter and lists several bibliographic sources.

## EXERCISES

- 1. [20] Suppose that you wish to obtain a decimal digit at random, not using a computer. Which of the following methods would be suitable?



- a) Open a telephone directory to a random place by sticking your finger in it somewhere, and use the units digit of the first number found on the selected page.
- b) Same as (a), but use the units digit of the *page* number.
- c) Roll a die that is in the shape of a regular icosahedron, whose twenty faces have been labeled with the digits 0, 0, 1, 1, ..., 9, 9. Use the digit that appears on top, when the die comes to rest. (A felt-covered table with a hard surface is recommended for rolling dice.)
- d) Expose a geiger counter to a source of radioactivity for one minute (shielding yourself) and use the units digit of the resulting count. Assume that the geiger counter displays the number of counts in decimal notation, and that the count is initially zero.
- e) Glance at your wristwatch; and if the position of the second-hand is between  $6n$  and  $6(n+1)$  seconds, choose the digit  $n$ .
- f) Ask a friend to think of a random digit, and use the digit he names.
- g) Ask an enemy to think of a random digit, and use the digit he names.
- h) Assume that 10 horses are entered in a race and that you know nothing whatever about their qualifications. Assign to these horses the digits 0 to 9, in arbitrary fashion, and after the race use the winner's digit.

2. [M22] In a random sequence of a million decimal digits, what is the probability that there are exactly 100,000 of each possible digit?

3. [10] What number follows 1010101010 in the middle-square method?

4. [20] (a) Why can't the value of  $X$  be zero when step K11 of Algorithm K is performed? What would be wrong with the algorithm if  $X$  could be zero? (b) Use Table 1 to deduce what happens when Algorithm K is applied repeatedly with the starting value  $X = 3830951656$ .

5. [15] Explain why, in any case, Algorithm K should not be expected to provide infinitely many random numbers, in the sense that (even if the coincidence given in Table 1 had not occurred) one knows in advance that any sequence generated by Algorithm K will eventually be periodic.

► 6. [M21] Suppose that we want to generate a sequence of integers  $X_0, X_1, X_2, \dots$ , in the range  $0 \leq X_n < m$ . Let  $f(x)$  be any function such that  $0 \leq x < m$  implies  $0 \leq f(x) < m$ . Consider a sequence formed by the rule  $X_{n+1} = f(X_n)$ . (Examples are the middle-square method and Algorithm K.)

- a) Show that the sequence is ultimately periodic, in the sense that there exist numbers  $\lambda$  and  $\mu$  for which the values

$$X_0, X_1, \dots, X_\mu, \dots, X_{\mu+\lambda-1}$$

are distinct, but  $X_{n+\lambda} = X_n$  when  $n \geq \mu$ . Find the maximum and minimum possible values of  $\mu$  and  $\lambda$ .

- b) (R. W. Floyd.) Show that there exists an  $n > 0$  such that  $X_n = X_{2n}$ ; and the smallest such value of  $n$  lies in the range  $\mu \leq n \leq \mu + \lambda$ . Furthermore the value of  $X_n$  is unique in the sense that if  $X_n = X_{2n}$  and  $X_r = X_{2r}$ , then  $X_r = X_n$ .
- c) Use the idea of part (b) to design an algorithm that calculates  $\mu$  and  $\lambda$  for any given function  $f$  and any given  $X_0$ , using only  $O(\mu + \lambda)$  steps and only a bounded number of memory locations.