

R Textbook Companion for  
Elementary Number Theory  
by David M. Burton<sup>1</sup>

Created by  
Sneha Ranjeet Sonaye  
B.E.  
Computer Science and Engineering  
Atharva College of Engineering  
Cross-Checked by  
R TBC Team

September 1, 2022

<sup>1</sup>Funded by a grant from the National Mission on Education through ICT  
- <http://spoken-tutorial.org/NMEICT-Intro>. This Textbook Companion and R  
codes written in it can be downloaded from the "Textbook Companion Project"  
section at the website - <https://r.fossee.in>.

# Book Description

**Title:** Elementary Number Theory

**Author:** David M. Burton

**Publisher:** Mcgraw-hill,1221 Avenue Of The Americas, New York

**Edition:** 7

**Year:** 2011

**ISBN:** 978-0-07-338314-9

R numbering policy used in this document and the relation to the above book.

**Exa** Example (Solved example)

**Eqn** Equation (Particular equation of the above book)

For example, Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means an R code whose theory is explained in Section 2.3 of the book.

# Contents

List of R Codes	4
1 PRELIMINARIES	5
2 DIVISIBILITY THEORY IN THE INTEGERS	7
3 PRIMES AND THEIR DISTRIBUTION	10
4 THE THEORY OF CONGRUENCES	13
5 FERMATS THEOREM	21
6 NUMBER THEORETIC FUNCTIONS	25
7 EULERS GENERALIZATION OF FERMATS THEOREM	30
8 PRIMITIVE ROOTS AND INDICES	36
9 THE QUADRATIC RECIPROCITY LAW	47
10 INTRODUCTION TO CRYPTOGRAPHY	56
13 REPRESENTATION OF INTEGERS AS SUMS OF SQUARES	69
15 CONTINUED FRACTIONS	74
16 SOME MODERN DEVELOPMENTS	83

# List of R Codes

Exa 1.1	Second Principle of Finite Induction . . . . .	5
Exa 2.2	The greatest common divisor . . . . .	7
Exa 2.3	the Euclidean Algorithm . . . . .	8
Exa 2.4	Applying the Euclidean Algorithm to the linear Diophantine equation . . . . .	8
Exa 3.1	for determining the canonical form of an integer . . . . .	10
Exa 4.1	seful characterization of congruence modulo $n$ in terms of remainders upon division by $n$ . . . . .	13
Exa 4.3	use congruences in carrying out certain types of computations . . . . .	14
Exa 4.4	to illustrate With suitable precautions cancellation can be allowed . . . . .	15
Exa 4.5	to illustrate binary exponential algorithm . . . . .	15
Exa 4.6	a well known test for divisibility by 11 . . . . .	16
Exa 4.7	solving linear congruences . . . . .	17
Exa 4.8	solve the linear congruence . . . . .	18
Exa 4.9	solving linear congruences using Chinese Remainder Theorem . . . . .	19
Exa 5.1	concrete example of Wilsons theorem . . . . .	21
Exa 5.2	To illustrate the application of Fermats method . . . . .	21
Exa 5.3	factor the positive integer using the Euclidean Algorithm . . . . .	22
Exa 5.4	factorization method by Maurice Kraitchik . . . . .	23
Exa 6.1	to find the sum of positive divisors of $n$ . . . . .	25
Exa 6.2	to find the number of zeros with which the decimal representation of 50 factorial terminates . . . . .	26
Exa 6.3	to clarify a Corollary . . . . .	27
Exa 6.4	to calculate the day of the week on which March 1 1990 fell . . . . .	27

Exa 6.5	to calculate on what day of the week will January 14 2020 occur . . . . .	28
Exa 7.1	to calculate phi of a number . . . . .	30
Exa 7.2	to reduce large powers modulo n using Eulers theorem	31
Exa 7.3	a numerical example of Gauss theorem . . . . .	33
Exa 7.4	an example of theorem 7 7 . . . . .	34
Exa 8.1	to find the integers that also have order 12 modulo 13	36
Exa 8.3	primitive roots for prime . . . . .	37
Exa 8.4	solve congruences using theory of indices . . . . .	40
Exa 8.5	solve congruences . . . . .	43
Exa 9.1	to find quadratic residues and non residues . . . . .	47
Exa 9.2	check residues of a number . . . . .	48
Exa 9.3	Using the Legendre symbol to display results . . . . .	48
Exa 9.4	to check if a congruence is solvable . . . . .	49
Exa 9.5	to prove a Legendre corollary . . . . .	51
Exa 9.6	to find the solution of a quadratic congruence with a composite . . . . .	53
Exa 10.1	Example of Vigeneres method of crypography using autokey . . . . .	56
Exa 10.2	To illustrate Hills cipher . . . . .	57
Exa 10.3	example of cryptographic systems involving modular exponentiation . . . . .	59
Exa 10.4	an illustration of the RSA public key algorithm . . . . .	60
Exa 10.5	to solve the superincreasing knapsack problem . . . . .	62
Exa 10.6	A public key cryptosystem based on the knapsack problem . . . . .	63
Exa 10.7	to encrypt a message using knapsack . . . . .	64
Exa 10.8	illustrate the selection of the public key . . . . .	65
Exa 10.9	to encrypt a message using ElGamal . . . . .	66
Exa 10.10	Using ElGamal cryposystem to authenticate a received message . . . . .	67
Exa 13.1	to represent a positive integer as sum of two squares . . . . .	69
Exa 13.2	to prove Lemma 2 . . . . .	71
Exa 13.3	to write an integer as sum of four squares . . . . .	72
Exa 15.3	solve a linear Diophantine equation . . . . .	74
Exa 15.5	to find continued fraction expansion of a number . . . . .	76
Exa 15.6	to find continued fraction expansion of a number . . . . .	76

Exa 15.7	an example of illustrating the corollary to sought a fraction . . . . .	77
Exa 15.8	to solve an application of above theorem . . . . .	78
Exa 15.9	to find a solution of an equation for the smallest positive integer . . . . .	80
Exa 16.1	factorization of a number using Pollards method . . . .	83
Exa 16.2	to obtain a nontrivial divisor of a number . . . . .	84
Exa 16.3	to factor a number using the continued fraction factorization method . . . . .	85
Exa 16.4	to factor a number using the continued fraction factorization method . . . . .	87
Exa 16.5	an example of the quadratic sieve algorithm . . . . .	89
Exa 16.6	Using a theorem to check if a number is prime . . . . .	93
Exa 16.7	to find four square roots of a modulo n . . . . .	94
Exa 16.8	to solve an example of blums game . . . . .	95

# Chapter 1

## PRELIMINARIES

R code Exa 1.1 Second Principle of Finite Induction

```
1 #page 6
2 a1 <- 1
3 a2 <- 3
4 arr <- array(c(a1, a2))
5 n <- 1
6 while (n <= 9) {
7   if (n >= 3 && n <= 9) {
8     arr[n] <- arr[n - 1] + arr[n - 2]
9   }
10  n <- n + 1
11 }
12 n <- n - 1
13 c <- 0
14 while (n > 0) {
15   if (isTRUE(arr[n] < ((7 / 4) ^ n))) {
16     c <- c + 1
17   }
18   n <- n - 1
19 }
20 if (isTRUE(c == 9))
21   print("Hence proved")
```





## Chapter 2

# DIVISIBILITY THEORY IN THE INTEGERS

R code Exa 2.2 The greatest common divisor

```
1 #page 21
2 print_divisors <- function(x) {
3   if (x < 0) {
4     x <- x * (- 1)
5   }
6   for (i in 1 : x) {
7     if ((x %% i) == 0) {
8       print(i)
9     }
10  }
11 }
12 gcd <- function(x, y) {
13   while (y) {
14     temp <- y
15     y <- x %% y
16     x <- temp
17   }
18   if (x < 0) {
19     return(- x)
```

```

20   }else {
21     return(x)
22   }
23 }
24 print_divisors(-12)
25 print_divisors(30)
26 print(gcd(-12, 30))
27 print(gcd(-5, 5))
28 print(gcd(8, 17))
29 print(gcd(-8, -36))

```

---

**R code Exa 2.3** the Euclidean Algorithm

```

1 #page 27
2 gcd <- function(x, y) {
3   while (y) {
4     temp <- y
5     y <- x %% y
6     x <- temp
7   }
8   if (x < 0)
9     return(- x)
10  else
11    return(x)
12 }
13 print(gcd(12378, 3054))

```

---

**R code Exa 2.4** Applying the Euclidean Algorithm to the linear Diophantine equation

```

1 #page 35
2 gcd <- function(x, y) {
3   while(y) {

```

```
4     temp = y
5     y = x %% y
6     x = temp
7 }
8 if(x<0)
9     return(-x)
10 else
11     return(x)
12 }
13 print(gcd(172,20))
```

---

## Chapter 3

# PRIMES AND THEIR DISTRIBUTION

R code Exa 3.1 for determining the canonical form of an integer

```
1 #page 45
2 a <- 2093
3 prime_factors <- vector()
4
5
6 canonical_form <- function(a) {
7   y <- ceiling(sqrt(a))
8   arr <- prime_numbers(y)
9   p <- new_y(a, arr)
10  return(p)
11 }
12
13 prime_numbers <- function(n) {
14   if (n >= 2) {
15     x <- seq(2, n)
16     prime_nums <- c()
17     for (i in seq(2, n)) {
18       if (any(x == i)) {
19         prime_nums <- c(prime_nums, i)
```

```

20         x <- c(x[(x %% i) != 0], i)
21     }
22 }
23     return(prime_nums)
24 }
25 }
26
27 new_y <- function(n, ar) {
28     for (i in ar) {
29         if (n %% i == 0) {
30             break ()
31         }
32     }
33     return(i)
34 }
35
36 check_prime <- function(h) {
37     flag <- 0
38     if (h > 1) {
39         flag <- 1
40         for (i in 2 :(h - 1)) {
41             if ((h %% i) == 0) {
42                 flag <- 0
43                 break
44             }
45         }
46     }
47     if (h == 2) {
48         flag <- 1
49     }
50     if (flag == 1) {
51         return(TRUE)
52     }else {
53         return(FALSE)
54     }
55 }
56
57 while (isFALSE(check_prime(a))) {

```

```
58 p <- canonical_form(a)
59 prime_factors <- c(prime_factors, p)
60 a <- a / p
61 }
62 prime_factors <- c(prime_factors, a)
63 print(prime_factors)
```

---

# Chapter 4

## THE THEORY OF CONGRUENCES

**R code Exa 4.1** useful characterization of congruence modulo  $n$  in terms of remainders upon division by  $n$

```
1 #page 65
2 n <- 7
3 find_modulo <- function(a, b) {
4   if (a > b) {
5     big <- a
6   }else {
7     big <- b
8   }
9   repeat {
10    r1 <- a %% n
11    r2 <- b %% n
12    n <- n + 2
13    if (r1 == r2) {
14      n <- n - 2
15      break ()
16    }
17    if (n == big) {
18      break ()
```



```

19     }
20 }
21   if (r1 == r2) {
22     return(n)
23   }else {
24     return(0)
25   }
26 }
27 verify_modulo <- function(p, q, r) {
28   r1 <- p %% r
29   r2 <- q %% r
30   if (r1 == r2) {
31     return(TRUE)
32   }else {
33     return(FALSE)
34   }
35 }
36 print(find_modulo(-56, -11))
37 print(verify_modulo(-31, 11, 7))

```

---

**R code Exa 4.3** use congruences in carrying out certain types of computations

```

1 #page 66
2 find_rm <- function(f, d) {
3   factorial <- 1
4   sum <- 0
5   for (n in 1 : f) {
6     for (i in 1 : n) {
7       factorial <- factorial * i
8     }
9     if (factorial %% d == 0)
10    break ()
11    sum <- sum + factorial
12    factorial <- 1

```

```

13   }
14   print(sum %% d)
15 }
16 (find_rm(100, 12))

```

---

**R code Exa 4.4** to illustrate With suitable precautions cancellation can be allowed

```

1 #page 67
2 gcd <- function(x, y) {
3   while (y) {
4     temp <- y
5     y <- x %% y
6     x <- temp
7   }
8   if (x < 0)
9     return(-x)
10  else
11    return(x)
12 }
13 check <- function(p, q, r) {
14   cmn <- (gcd(p, q))
15   p <- p / cmn
16   q <- q / cmn
17   if (gcd(cmn, r) == cmn)
18     r <- r / cmn
19   print(c(p, q, r))
20 }
21 check(33, 15, 9)
22 check(-35, 45, 8)

```

---

**R code Exa 4.5** to illustrate binary exponential algorithm

```

1 #page 71
2 library(gmp)
3 library(binaryLogic)
4 library(base)
5 calculate_power_mod <- function(x, y, p) {
6   val <- as.integer(vector())
7   prod <- 1
8   b <- as.binary(y)
9   for (j in 1 : 6) {
10    val <- append(val, powm(5, 2 ^ j, 131))
11  }
12  count <- 7
13  for (v in b) {
14    count <- count - 1
15    if (v) {
16      prod <- prod * val[count]
17    }
18  }
19  print(prod %% p)
20 }
21 calculate_power_mod(5, 110, 131)

```

---

**R code Exa 4.6** a well known test for divisibility by 11

```

1 #page 72
2 check_num <- function(num, y) {
3   digits <- as.integer(vector())
4   while (num > 0) {
5     digits <- append(digits, num %% 10)
6     num <- as.integer(num / 10)
7   }
8   digits <- rev(digits)
9   if (y == 9) {
10    return(sum(num))
11  }

```

```

12   else if (y == 11) {
13     return(sum(num))
14   }
15 }
16 sum <- function(d) {
17   s <- 0
18   for (v in d) {
19     s <- s + v
20   }
21   if (s %% 9 == 0) {
22     return(TRUE)
23   }
24   return(FALSE)
25 }
26 al_sum <- function(d) {
27   s <- 0
28   for (v in d) {
29     if (v %% 2 == 0) {
30       s <- s - v
31     }else {
32       s <- s + v
33     }
34   }
35   if (s %% 11 == 0) {
36     return(TRUE)
37   }
38   return(FALSE)
39 }
40 print(check_num(1571724, 9))
41 print(check_num(1571724, 11))

```

---

R code Exa 4.7 solving linear congruences

1 #page 77  
2

```

3 find_x <- function(a, p, q) {
4   x <- as.integer(vector())
5   s <- gcd(a, q)
6   if (p %% s == 0) {
7     i <- q / s
8     while (s > 0) {
9       t <- (4 + i * s) %% q
10      x <- append(x, t)
11      s <- s - 1
12    }
13    x <- sort(x)
14    return(x)
15  }
16 }
17 gcd <- function(x, y) {
18   while (y) {
19     temp <- y
20     y <- x %% y
21     x <- temp
22   }
23   if (x < 0) {
24     return(-x)
25   } else {
26     return(x)
27   }
28 }
29
30 print(find_x(18, 30, 42))

```

---

**R code Exa 4.8** solve the linear congruence

```

1 #page 77
2 find_x <- function(a, p, q) {
3   x <- as.integer(vector())
4   s <- gcd(a, q)

```

```

5   if (p %% s == 0) {
6     a <- a / 3
7     p <- p / 3
8     q <- q / 3
9     a <- a * 7
10    p <- p * 7
11    a <- 1
12    p <- 9
13    for (s in 0 : 2) {
14      t <- p + q * s
15      x <- append(x, t)
16    }
17    x <- sort(x)
18    return(x)
19  }
20 }
21 gcd <- function(x, y) {
22   while (y) {
23     temp <- y
24     y <- x %% y
25     x <- temp
26   }
27   if (x < 0) {
28     return(-x)
29   } else {
30     return(x)
31   }
32 }
33
34 print(find_x(9, 21, 30))

```

---

**R code Exa 4.9** solving linear congruences using Chinese Remainder Theorem

1 #page 80

```
2 find_x <- function(p1, p2, p3, q1, q2, q3) {
3   n <- q1 * q2 * q3
4   n1 <- n / q1
5   n2 <- n / q2
6   n3 <- n / q3
7   x1 <- find_x0(n1, 1, q1)
8   x2 <- find_x0(n2, 1, q2)
9   x3 <- find_x0(n3, 1, q3)
10  x <- p1 * n1 * x1 + p2 * n2 * x2 + p3 * n3 * x3
11  return(x %% n)
12 }
13 find_x0 <- function(n, a, q) {
14   for (x in 1 : 9) {
15     if (((n * x) %% q) == a) {
16       return(x)
17     }
18   }
19 }
20 print(find_x(2, 3, 2, 3, 5, 7))
```

---

## Chapter 5

# FERMATS THEOREM

R code Exa 5.1 concrete example of Wilsons theorem

```
1 #page 94
2 prove_wilsons_theorem <- function(p) {
3   l <- factorial(p - 1)
4   if ((l + 1) %% p == 0) {
5     return(TRUE)
6   }else {
7     return(FALSE)
8   }
9 }
10 print(prove_wilsons_theorem(13))
```

---

R code Exa 5.2 To illustrate the application of Fermats method

```
1 #page 98
2 fermat_factorization <- function(n) {
3   n <- as.integer(n)
4   lb <- ceiling(sqrt(n))
5   ub <- ((n + 1) / 2) - 1
```



```

6   lb <- as.integer(lb)
7   ub <- as.integer(ub)
8   for (k in lb : 352) {
9     f <- sqrt(k ^ 2 - n)
10    if (perfect(f)) {
11      factors <- c(k + f, k - f)
12      return(factors)
13    }
14  }
15 }
16 perfect <- function(a) {
17   b <- floor(a)
18   if ((a / b) == 1) {
19     return(TRUE)
20   }else {
21     return(FALSE)
22   }
23 }
24 print(fermat_factorization(119143))

```

---

**R code Exa 5.3** factor the positive integer using the Euclidean Algorithm

```

1 #page 100
2 factorize <- function(n) {
3   uy <- floor(sqrt(n))
4   ux <- floor(n / 2)
5   for (xn in ux : uy) {
6     for (yn in uy : 1) {
7       c <- xn ^ 2 - yn ^ 2
8       m <- c %% n
9       if (m == 0) {
10        ans <- c(gcd(xn - yn, n), gcd(xn + yn, n))
11        return(ans)
12      }
13    }

```

```

14   }
15 }
16 gcd <- function(x, y) {
17   while (y) {
18     temp <- y
19     y <- x %% y
20     x <- temp
21   }
22   if (x < 0) {
23     return(-x)
24   }else {
25     return(x)
26   }
27 }
28 print(factorize(2189))

```

---

**R code Exa 5.4** factorization method by Maurice Kraitchik

```

1 #page 100
2 factorize <- function(n) {
3   uy <- floor(sqrt(n))
4   ux <- floor(n / 2)
5   for (xn in ux : uy) {
6     for (yn in uy : 1) {
7       c <- xn ^ 2 - yn ^ 2
8       m <- c %% n
9       if (m == 0) {
10        ans <- c(gcd(xn - yn, n), gcd(xn + yn, n))
11        return(ans)
12      }
13    }
14  }
15 }
16 gcd <- function(x, y) {
17   while (y) {

```

```
18     temp <- y
19     y <- x %% y
20     x <- temp
21   }
22   if (x < 0) {
23     return(- x)
24   } else {
25     return(x)
26   }
27 }
28 print(factorize(12499))
```

---

## Chapter 6

# NUMBER THEORETIC FUNCTIONS

R code Exa 6.1 to find the sum of positive divisors of n

```
1 #page106
2 library(collections)
3 solve <- function(n) {
4   p <- vector()
5   k <- vector()
6   i <- 0
7   while (n %% 2 == 0) {
8     i <- i + 1
9     n <- n / 2
10  }
11  if (i != 0) {
12    p <- append(p, 2)
13    k <- append(k, i)
14  }
15  for (num in 3 : sqrt(n)) {
16    if (num %% 2 == 1) {
17      i <- 0
18      while (n %% num == 0) {
19        i <- i + 1
```

```

20         n <- n / num
21     }
22     if (i != 0) {
23         p <- append(p, num)
24         k <- append(k, i)
25     }
26 }
27 }
28 tau <- no_of_divisors(k)
29 print(tau)
30 sigma <- sum_of_divisors(p, k)
31 print(sigma)
32 }
33 sum_of_divisors <- function(p, k) {
34     sum <- 1
35     c <- length(p)
36     for (x in 1 : c) {
37         sum <- sum * (((p[x] ^ (k[x] + 1)) - 1) / (p[x]
38             - 1))
39     }
40     return(sum)
41 }
42 no_of_divisors <- function(k) {
43     no <- 1
44     for (x in k) {
45         no <- no * (x + 1)
46     }
47     return(no)
48 }
49 solve(180)

```

---

**R code Exa 6.2** to find the number of zeros with which the decimal representation of 50 factorial terminates

1 #page 118

```

2 n <- 50
3 pow_of_2 <- 0
4 pow_of_5 <- 0
5 for (v in 1 : 5) {
6   pow_of_2 <- pow_of_2 + floor(n / (2 ^ v))
7 }
8 print(pow_of_2)
9 for (v in 1 : 2) {
10  pow_of_5 <- pow_of_5 + floor(n / (5 ^ v))
11 }
12 print(pow_of_5)

```

---

**R code Exa 6.3** to clarify a Corollary

```

1 #page 120
2 n <- 6
3 tau <- 0
4 sigma <- 0
5 for (num in 1 : n) {
6   tau <- tau + floor(n / num)
7 }
8 print(tau)
9 for (num in 1 : n) {
10  sigma <- sigma + (num * floor(n / num))
11 }
12 print(sigma)

```

---

**R code Exa 6.4** to calculate the day of the week on which March 1 1990 fell

```

1 #page 125
2 c <- 19
3 y <- 90

```

```

4 d <- (3 - 2 * c + y + floor(c / 4) + floor(y / 4))
  %% 7
5 if (d == 0) {
6   print("Sunday")
7 } else if (d == 1) {
8   print("Monday")
9 } else if (d == 2) {
10  print("Tuesday")
11 } else if (d == 3) {
12  print("Wednesday")
13 } else if (d == 4) {
14  print("Thursday")
15 } else if (d == 5) {
16  print("Friday")
17 } else {
18  print("Saturday")
19 }

```

---

**R code Exa 6.5** to calculate on what day of the week will January 14 2020 occur

```

1 #page 126
2 m <- 11
3 d <- 14
4 c <- 20
5 y <- 19
6 w <- (d + floor((2.6) * m - 0.2) - 2 * c + y + floor
  (c / 4) + floor(y / 4)) %% 7
7 if (w == 0) {
8   print("Sunday")
9 } else if (w == 1) {
10  print("Monday")
11 } else if (w == 2) {
12  print("Tuesday")
13 } else if (w == 3) {

```

```
14     print("Wednesday")
15 } else if (w == 4) {
16     print("Thursday")
17 } else if (w == 5) {
18     print("Friday")
19 } else {
20     print("Saturday")
21 }
```

---



## Chapter 7

# EULERS GENERALIZATION OF FERMATS THEOREM

R code Exa 7.1 to calculate phi of a number

```
1 #page 134
2 n <- 360
3 number <- n
4 p <- vector()
5 k <- vector()
6 i <- 0
7 while (n %% 2 == 0) {
8   i <- i + 1
9   n <- n / 2
10 }
11 if (i != 0) {
12   p <- append(p, 2)
13   k <- append(k, i)
14 }
15 for (num in 3 : sqrt(n)) {
16   if (num %% 2 == 1) {
17     i <- 0
18     while (n %% num == 0) {
19       i <- i + 1
```

```

20     n <- n / num
21   }
22   if (i != 0) {
23     p <- append(p, num)
24     k <- append(k, i)
25   }
26 }
27 }
28 pos_prime <- function(p, n) {
29   sum <- number
30   c <- length(p)
31   for (x in 1 : c) {
32     sum <- sum * (1 - (1 / p[x]))
33   }
34   return(sum)
35 }
36 phi <- pos_prime(p, n)
37 print(phi)

```

---

**R code Exa 7.2** to reduce large powers modulo n using Eulers theorem

```

1 #page 138
2 calculate <- function(a, r, n) {
3   c <- 0
4   while (r %% 2 == 0 & r != 0) {
5     c <- c + 1
6     r <- r / 2
7   }
8   ans <- a
9   for (var in 1 : c) {
10    ans <- (ans ^ 2) %% n
11  }
12  return(ans)
13 }
14

```

```

15 gcd <- function(x, y) {
16   while (y) {
17     temp <- y
18     y <- x %% y
19     x <- temp
20   }
21   if (x < 0) {
22     return(- x)
23   }else {
24     return(x)
25   }
26 }
27 a <- 3
28 r <- 256
29 n <- 100
30 print(gcd(a, n))
31 number <- n
32 p <- vector()
33 k <- vector()
34 i <- 0
35 while (n %% 2 == 0) {
36   i <- i + 1
37   n <- n / 2
38 }
39 if (i != 0) {
40   p <- append(p, 2)
41   k <- append(k, i)
42 }
43 for (num in 3 : sqrt(n)) {
44   if (num %% 2 == 1) {
45     i <- 0
46     while (n %% num == 0) {
47       i <- i + 1
48       n <- n / num
49     }
50     if (i != 0) {
51       p <- append(p, num)
52       k <- append(k, i)

```

```

53     }
54   }
55 }
56 pos_prime <- function(p, n) {
57   sum <- number
58   c <- length(p)
59   for (x in 1 : c) {
60     sum <- sum * (1 - (1 / p[x]))
61   }
62   return(sum)
63 }
64 phi <- pos_prime(p, n)
65 print(phi)
66 q <- floor(r / phi)
67 rd <- r %% phi
68 r <- rd
69 ans <- calculate(a, r, number)
70 print(ans)

```

---

**R code Exa 7.3** a numerical example of Gauss theorem

```

1 #page 142
2 phi <- function(n) {
3   c <- 0
4   for (v in 1 : n) {
5     if (gcd(v, n) == 1) {
6       c <- c + 1
7     }
8   }
9   return(c)
10 }
11
12 gcd <- function(x, y) {
13   while (y) {
14     temp <- y

```

```

15     y <- x %% y
16     x <- temp
17   }
18   if (x < 0) {
19     return(- x)
20   }else {
21     return(x)
22   }
23 }
24 n <- 10
25 d <- vector()
26 for (m in 1 : n) {
27   d <- append(d, gcd(m, n))
28 }
29 d <- unique(d)
30 sum_phi <- 0
31 for (v in d) {
32   sum_phi <- sum_phi + phi(v)
33 }
34 print(sum_phi == n)

```

---

**R code Exa 7.4** an example of theorem 7.7

```

1 #page 143
2 n <- 30
3 number <- n
4 p <- vector()
5 k <- vector()
6 i <- 0
7 while (n %% 2 == 0) {
8   i <- i + 1
9   n <- n / 2
10 }
11 if (i != 0) {
12   p <- append(p, 2)

```

```

13   k <- append(k, i)
14 }
15 s <- sqrt(number)
16 for (num in 3 : s) {
17   if (num %% 2 == 1) {
18     i <- 0
19     while (n %% num == 0) {
20       i <- i + 1
21       n <- n / num
22     }
23     if (i != 0) {
24       p <- append(p, num)
25       k <- append(k, i)
26     }
27   }
28 }
29 pos_prime <- function(p, n) {
30   sum <- n
31   c <- length(p)
32   for (x in 1 : c) {
33     sum <- sum * (1 - (1 / p[x]))
34   }
35   return(sum)
36 }
37 phi <- pos_prime(p, number)
38 rel_prime <- vector()
39 for (v in 1 : number) {
40   if (gcd(v, number) == 1) {
41     rel_prime <- append(rel_prime, v)
42   }
43 }
44 sum <- 0
45 for (v in rel_prime) {
46   sum <- sum + v
47 }
48 desired_sum <- (1 / 2) * number * phi
49 print(isTRUE(all.equal(sum, desired_sum)))

```

---

## Chapter 8

# PRIMITIVE ROOTS AND INDICES

R code Exa 8.1 to find the integers that also have order 12 modulo 13

```
1 #page 149
2 gcd <- function(x, y) {
3   while (y) {
4     temp <- y
5     y <- x %% y
6     x <- temp
7   }
8   if (x < 0) {
9     return(-x)
10  }else {
11    return(x)
12  }
13 }
14 n <- 13
15 ans <- vector()
16 for (num in 1 : n) {
17   for (v in 1 : n) {
18     if (((num ^ v) %% n) == 1) {
19       ans <- append(ans, v)
```

```

20     break ()
21   }
22 }
23 }
24 print(ans)
25 for (x in 2 : 3) {
26   if (ans[2 ^ x] == (ans[2] / gcd(x, ans[2]))) {
27     print(TRUE)
28   }
29 }
30 for (x in 1 : 12) {
31   if (gcd(x, 12) == 1) {
32     print(x)
33   }
34 }

```

---

### R code Exa 8.3 primitive roots for prime

```

1 #page 157
2 primitive_root <- function(g, n) {
3   number <- n
4   i <- 0
5   ptt <- vector()
6   while ((n %% 2) == 0) {
7     i <- i + 1
8     n <- n / 2
9   }
10  if (i != 0) {
11    ptt <- append(ptt, number / 2)
12  }
13  for (var in 3 : sqrt(number)) {
14    if (var %% 2 == 1) {
15      i <- 0
16      while (n %% var == 0) {
17        i <- i + 1

```



```

18         n <- n / var
19     }
20     if (i != 0) {
21         ptt <- append(ptt, number / var)
22     }
23 }
24 }
25 ptt <- sort(ptt)
26 for (num in 2 : number) {
27     i <- 0
28     for (x in ptt) {
29         if ((num ^ x) %% g == 1) {
30             break ()
31         }else {
32             i <- i + 1
33         }
34     }
35     if (i == length(ptt)) {
36         return(num)
37     }
38 }
39 }
40 phi <- function(n) {
41     number <- n
42     p <- vector()
43     k <- vector()
44     i <- 0
45     while ((n %% 2) == 0) {
46         i <- i + 1
47         n <- n / 2
48     }
49     if (i != 0) {
50         p <- append(p, 2)
51         k <- append(k, i)
52     }
53     for (num in 3 : sqrt(number)) {
54         if (num %% 2 == 1) {
55             i <- 0

```

```

56     while (n %% num == 0) {
57         i <- i + 1
58         n <- n / num
59     }
60     if (i != 0) {
61         p <- append(p, num)
62         k <- append(k, i)
63     }
64 }
65 }
66 pos_prime <- function(p, n) {
67     sum <- number
68     c <- length(p)
69     for (x in 1 : c) {
70         sum <- sum * (1 - (1 / p[x]))
71     }
72     return(sum)
73 }
74 if (length(p) == 0) {
75     phi <- number - 1
76 }else {
77     phi <- pos_prime(p, n)
78 }
79 return(phi)
80 }
81 ord <- 6
82 mod <- 31
83 npr <- phi(ord)
84 p <- (phi(mod))
85 pr <- primitive_root(mod, p)
86 kn <- vector()
87 for (k in 1 : p) {
88     if ((p / gcd(k, p)) == ord) {
89         kn <- append(kn, k)
90     }
91 }
92 for (p in kn) {
93     print((pr ^ p) %% mod)

```

94 }

---

**R code Exa 8.4** solve congruences using theory of indices

```
1 #page 165
2 mod <- function(a, z, l) {
3   ans <- vector()
4   for (k in 1 : l) {
5     if (k %% z == a) {
6       ans <- append(ans, k)
7     }
8   }
9   return(ans)
10 }
11 gcd <- function(x, y) {
12   while (y) {
13     temp <- y
14     y <- x %% y
15     x <- temp
16   }
17   if (x < 0) {
18     return(- x)
19   } else {
20     return(x)
21   }
22 }
23 primitive_root <- function(g, n) {
24   i <- 0
25   number <- n
26   ptt <- vector()
27   while ((n %% 2) == 0) {
28     i <- i + 1
29     n <- n / 2
30   }
31   if (i != 0) {
```

```

32     ptt <- append(ptt, number / 2)
33   }
34   for (var in 3 : sqrt(number)) {
35     if (var %% 2 == 1) {
36       i <- 0
37       while (n %% var == 0) {
38         i <- i + 1
39         n <- n / var
40       }
41       if (i != 0) {
42         ptt <- append(ptt, number / var)
43       }
44     }
45   }
46   ptt <- sort(ptt)
47   for (num in 2 : number) {
48     i <- 0
49     for (x in ptt) {
50       if ((num ^ x) %% g == 1) {
51         break ()
52       }else {
53         i <- i + 1
54       }
55     }
56     if (i == length(ptt)) {
57       return(num)
58     }
59   }
60 }
61 phi <- function(n) {
62   number <- n
63   p <- vector()
64   k <- vector()
65   i <- 0
66   while ((n %% 2) == 0) {
67     i <- i + 1
68     n <- n / 2
69   }

```

```

70   if (i != 0) {
71     p <- append(p, 2)
72     k <- append(k, i)
73   }
74   for (num in 3 : sqrt(number)) {
75     if (num %% 2 == 1) {
76       i <- 0
77       while (n %% num == 0) {
78         i <- i + 1
79         n <- n / num
80       }
81       if (i != 0) {
82         p <- append(p, num)
83         k <- append(k, i)
84       }
85     }
86   }
87   pos_prime <- function(p, n) {
88     sum <- number
89     c <- length(p)
90     for (x in 1 : c) {
91       sum <- sum * (1 - (1 / p[x]))
92     }
93     return(sum)
94   }
95   if (length(p) == 0) {
96     phi <- number - 1
97   } else {
98     phi <- pos_prime(p, n)
99   }
100  return(phi)
101 }
102 r <- 4
103 ind_a <- 9
104 n <- 13
105 ind <- vector()
106 a <- vector()
107 ans_x <- vector()

```

```

108 phi <- phi(n)
109 pr <- primitive_root(13, phi)
110 for (an in 1 : phi) {
111   if (gcd(an, n) == 1) {
112     for (k in 1 : n) {
113       if (((pr ^ k) %% n) == an) {
114         ind <- append(ind, k)
115         a <- append(a, an)
116         break ()
117       }
118     }
119   }
120 }
121 indxx9 <- ind[7] - ind[4]
122 indx <- mod(1, 4, phi)
123 for (x in a) {
124   if (is.element(ind[x], indx)) {
125     ans_x <- append(ans_x, x)
126   }
127 }
128 print(ans_x)

```

---

### R code Exa 8.5 solve congruences

```

1 #page 166
2 mod <- function(a, z, l) {
3   ans <- vector()
4   for (k in 1 : l) {
5     if (k %% z == a) {
6       ans <- append(ans, k)
7     }
8   }
9   return(ans)
10 }
11 gcd <- function(x, y) {

```

```

12  while (y) {
13    temp <- y
14    y <- x %% y
15    x <- temp
16  }
17  if (x < 0) {
18    return(- x)
19  } else {
20    return(x)
21  }
22 }
23 phi <- function(n) {
24   number <- n
25   p <- vector()
26   k <- vector()
27   i <- 0
28   while ((n %% 2) == 0) {
29     i <- i + 1
30     n <- n / 2
31   }
32   if (i != 0) {
33     p <- append(p, 2)
34     k <- append(k, i)
35   }
36   for (num in 3 : sqrt(number)) {
37     if (num %% 2 == 1) {
38       i <- 0
39       while (n %% num == 0) {
40         i <- i + 1
41         n <- n / num
42       }
43       if (i != 0) {
44         p <- append(p, num)
45         k <- append(k, i)
46       }
47     }
48   }
49   pos_prime <- function(p, n) {

```

```

50     sum <- number
51     c <- length(p)
52     for (x in 1 : c) {
53         sum <- sum * (1 - (1 / p[x]))
54     }
55     return(sum)
56 }
57 if (length(p) == 0) {
58     phi <- number - 1
59 } else {
60     phi <- pos_prime(p, n)
61 }
62 return(phi)
63 }
64 solution <- function(n, a, k) {
65     if (gcd(a, n) != 1) {
66         print("gcd is not 1")
67     }
68     phi <- phi(n)
69     d <- gcd(k, phi)
70     if ((a ^ (phi / d) %% n) == 1) {
71         print(paste(d, "Solutions exist"))
72     } else {
73         print("No solution exists")
74     }
75 }
76 n <- 13
77 a <- 4
78 k <- 3
79 p <- phi(n)
80 solution(n, a, k)
81 a <- 5
82 solution(n, a, k)
83 ax <- vector()
84 ind <- vector()
85 ans_x <- vector()
86 for (an in 1 : p) {
87     if (gcd(an, n) == 1) {

```



```
88     for (c in 1 : n) {
89         if (((pr ^ c) %% n) == an) {
90             ind <- append(ind, c)
91             ax <- append(ax, an)
92             break ()
93         }
94     }
95 }
96 }
97
98 a <- 9
99 n <- 12
100 a <- (a / k)
101 n <- n / k
102 indx <- mod(a, n, p)
103 for (x in ax) {
104     if (is.element(ind[x], indx)) {
105         ans_x <- append(ans_x, x)
106     }
107 }
108 print(ans_x)
```

---

## Chapter 9

# THE QUADRATIC RECIPROcity LAW

R code Exa 9.1 to find quadratic residues and non residues

```
1 #page 171
2 n <- 13
3 residues <- vector()
4 non_residues <- vector()
5 for (v in 1 : (n - 1)) {
6   residues <- append(residues, (v ^ 2) %% n)
7 }
8 residues <- sort(unique(residues))
9 print(residues)
10 for (v in 1 : (n - 1)) {
11   if (!is.element(v, residues)) {
12     non_residues <- append(non_residues, v)
13   }
14 }
15 print(non_residues)
16 n_consecutive_pairs <- (1 / 4) * (n - 4 - (- 1) ^ ((
17   n - 1) / 2))
17 print(n_consecutive_pairs)
```

---

**R code Exa 9.2** check residues of a number

```
1 #page 172
2 check_residue <- function(a, p) {
3   f <- (a ^ ((p - 1) / 2)) %% p
4   if (f == 1 | f == (p - 1)) {
5     print(paste(a, "is residue of", p))
6   }
7 }
8 p <- 13
9 a <- 2
10 check_residue(a, p)
11 a <- 3
12 check_residue(a, p)
```

---

**R code Exa 9.3** Using the Legendre symbol to display results

```
1 #page 176
2 n <- 13
3 ls <- vector()
4 residues <- vector()
5 non_residues <- vector()
6 for (v in 1 : (n - 1)) {
7   residues <- append(residues, (v ^ 2) %% n)
8 }
9 residues <- sort(unique(residues))
10 for (v in 1 : (n - 1)) {
11   if (!is.element(v, residues)) {
12     non_residues <- append(non_residues, v)
13   }
14 }
15 for (var in 1 : (n - 1)) {
```

```

16   if (is.element(var, residues)) {
17     ls <- append(ls, 1)
18   } else {
19     ls <- append(ls, - 1)
20   }
21 }
22 l <- length(ls)
23 for (var in 1 : l) {
24   ans <- sprintf("(%d/%d) = %d", var, n, ls[var])
25   print(ans)
26 }

```

---

**R code Exa 9.4** to check if a congruence is solvable

```

1 #page 177
2 find <- function(l, s) {
3   if (l < 0) {
4     l <- l * (- 1)
5   }
6   m <- l %% s
7   l <- m
8   squares <- vector()
9   pk <- vector()
10  k <- vector()
11  i <- 0
12  n <- l
13  while (n %% 2 == 0) {
14    i <- i + 1
15    n <- n / 2
16  }
17  if (i != 0) {
18    pk <- append(pk, 2)
19    k <- append(k, i)
20  }
21  for (num in 3 : sqrt(n)) {

```

```

22     if (num %% 2 == 1) {
23         i <- 0
24         while (n %% num == 0) {
25             i <- i + 1
26             n <- n / num
27         }
28         if (i != 0) {
29             pk <- append(pk, num)
30             k <- append(k, i)
31         }
32     }
33 }
34 for (x in seq_len(length(k))) {
35     if (k[x] == 2) {
36         squares <- append(squares, pk[x])
37     }
38 }
39 for (sq in squares) {
40     l <- 1 / (sq ^ 2)
41 }
42 mod <- ((1 ^ ((s - 1) / 2)) %% s)
43 if (mod == (s - 1)) {
44     return(- 1)
45 } else {
46     return(mod)
47 }
48 }
49
50 a <- -46
51 p <- 17
52 l <- -46
53 s <- 17
54 ls <- find(l, s)
55 if (ls == (- 1)) {
56     print("No solution")
57 } else {
58     print("solution exists")
59 }

```

---

**R code Exa 9.5** to prove a Legendre corollary

```
1 #page 188
2 solve <- function(p, q) {
3   if (p == 2) {
4     if (q %% 8 == 1 | q %% 8 == 7) {
5       return(1)
6     } else if (q %% 8 == 3 | q %% 8 == 5) {
7       return(- 1)
8     }
9   } else {
10    t <- p
11    p <- q
12    q <- t
13    p <- p %% q
14    solve(p, q)
15  }
16 }
17 p <- 29
18 q <- 53
19 i <- 0
20 final <- 1
21 answer <- vector()
22 squares <- vector()
23 factors <- vector()
24 pk <- vector()
25 px <- vector()
26 k <- vector()
27 m1 <- p %% 4
28 m2 <- q %% 4
29 if (m1 == m2) {
30   if (m1 == 1) {
31     t <- p
32     p <- q
```

```

33     q <- t
34   } else {
35     t <- p
36     p <- q * - 1
37     q <- t
38   }
39 }
40 p <- p %% q
41 n <- p
42 while (n %% 2 == 0) {
43   i <- i + 1
44   n <- n / 2
45 }
46 if (i != 0) {
47   pk <- append(pk, 2)
48   k <- append(k, i)
49 }
50 for (num in 3 : sqrt(n)) {
51   if (num %% 2 == 1) {
52     i <- 0
53     while (n %% num == 0) {
54       i <- i + 1
55       n <- n / num
56     }
57     if (i != 0) {
58       pk <- append(pk, num)
59       k <- append(k, i)
60     }
61   }
62 }
63 for (x in seq_len(length(k))) {
64   if ((k[x] >= 2) & (k[x] %% 2 == 0)) {
65     squares <- append(squares, pk[x])
66     px <- append(px, k[x])
67   } else if (k[x] == 1) {
68     factors <- append(factors, pk[x])
69     p <- p / pk[x]
70   } else {

```

```

71     squares <- append(squares, pk[x])
72     px <- append(px, (k[x] - 1))
73   }
74 }
75 for (sq in squares) {
76   for (pw in px) {
77     p <- p / (sq ^ pw)
78   }
79 factors <- append(factors, p)
80 for (f in factors) {
81   ans <- solve(f, q)
82   answer <- append(answer, ans)
83 }
84 for (a in answer) {
85   final <- final * a
86 }
87 }
88 print(final)

```

---

**R code Exa 9.6** to find the solution of a quadratic congruence with a composite

```

1 #page 189
2 p <- 196
3 q <- 1357
4 i <- 0
5 pk <- vector()
6 k <- vector()
7 squares <- vector()
8 for (q1 in 2 : sqrt(q)) {
9   if (q %% q1 == 0) {
10     q2 <- q / q1
11     break ()
12   }
13 }

```



```

14 p1 <- p %% q1
15 n <- p1
16 while (n %% 2 == 0) {
17   i <- i + 1
18   n <- n / 2
19 }
20 if (i != 0) {
21   pk <- append(pk, 2)
22   k <- append(k, i)
23 }
24 for (num in 3 : sqrt(p1)) {
25   if (num %% 2 == 1) {
26     i <- 0
27     while (n %% num == 0) {
28       i <- i + 1
29       n <- n / num
30     }
31     if (i != 0) {
32       pk <- append(pk, num)
33       k <- append(k, i)
34     }
35   }
36 }
37 for (x in seq_len(k)) {
38   if (k[x] == 2) {
39     squares <- append(squares, pk[x])
40   }
41 }
42 for (sq in squares) {
43   p1 <- p1 / (sq ^ 2)
44 }
45 if (p1 == 3) {
46   if (q1 %% 12 == 1 | q1 %% 12 == (12 - 1)) {
47     ls1 <- 1
48   } else if (q1 %% 12 == 5 | q1 %% 12 == (12 - 5)) {
49     ls1 <- -1
50   }
51 }

```

```

52 p2 <- p %% q2
53 if (q2 > p2) {
54   m1 <- p2 %% 4
55   m2 <- q2 %% 4
56   if (m1 == m2) {
57     if (m1 == 1) {
58       t <- p2
59       p2 <- q2
60       q2 <- t
61     } else if (m1 == 3) {
62       t <- p2
63       p2 <- q2
64       q2 <- t
65       s <- -1
66     }
67   }
68 }
69 if (p2 > q2) {
70   p2 <- p2 %% q2
71 }
72 if (p2 == 2) {
73   if (q2 %% 8 == 1 | q2 %% 8 == 7) {
74     ls2 <- s * 1
75   } else if (q2 %% 8 == 3 | q2 %% 8 == 5) {
76     ls2 <- s * -1
77   }
78 }
79 if (ls1 == 1 & ls2 == 1) {
80   print("solvable")
81 }

```

---

# Chapter 10

## INTRODUCTION TO CRYPTOGRAPHY

**R code Exa 10.1** Example of Vigeneres method of cryptography using autokey

```
1 #page 200
2 library(gtools)
3 library(readr)
4 pv <- vector()
5 kv <- vector()
6 cv <- vector()
7 c <- ""
8 plain_text <- "ONE IF BY DAWN"
9 p <- gsub(" ", "", plain_text, fixed = TRUE)
10 seed <- "K"
11 k <- paste(seed, substr(p, 1, nchar(p) - 1), sep = "
    ")
12 p_split <- strsplit(p, "")
13 k_split <- strsplit(k, "")
14 for (ch in p_split) {
15   pv <- append(pv, asc(ch) - 65)
16 }
17 for (ch in k_split) {
```

```

18 kv <- append(kv, asc(ch) - 65)
19 }
20 for (num in seq_len(length(pv))) {
21   for (n in seq_len(length(kv))) {
22     if (n == num) {
23       cv <- append(cv, (kv[n] + pv[num]) %% 26)
24     }
25   }
26 }
27 for (n in cv) {
28   num <- n + 65
29   c <- paste(c, chr(num), sep = " ")
30 }
31 c <- sub("\\s+$", "", gsub("(.{3})(.{2})(.{2})", "
    \\1 \\2 \\3 ", c))
32 print(c)

```

---

**R code Exa 10.2** To illustrate Hills cipher

```

1 #page 201
2 library(gtools)
3 hill_cipher <- function(block) {
4   p1 <- substr(block, 1, 1)
5   p2 <- substr(block, 2, 2)
6   p1 <- asc(p1) - 65
7   p2 <- asc(p2) - 65
8   c1 <- (a * p1 + b * p2) %% 26
9   c2 <- (c * p1 + d * p2) %% 26
10  c <- paste0(chr(c1 + 65), chr(c2 + 65))
11  return(c)
12 }
13 decrypt <- function(block) {
14   c1 <- substr(block, 1, 1)
15   c2 <- substr(block, 2, 2)
16   c1 <- asc(c1) - 65

```

```

17   c2 <- asc(c2) - 65
18   p1 <- (da * c1 + db * c2) %% 26
19   p2 <- (dc * c1 + dd * c2) %% 26
20   p <- paste0(chr(p1 + 65), chr(p2 + 65))
21   return(p)
22 }
23 a <- 2
24 b <- 3
25 c <- 5
26 d <- 8
27 bl_v <- vector()
28 message <- "BUY NOW"
29 m <- gsub(" ", "", message, fixed = TRUE)
30 blocks <- sub("\\s+$", "", gsub("(.{2})", "\\1 ", m)
   )
31 block1 <- substr(blocks, 1, 2)
32 c1 <- hill_cipher(block1)
33 block2 <- substr(blocks, 4, 5)
34 c2 <- hill_cipher(block2)
35 block3 <- substr(blocks, 7, 8)
36 c3 <- hill_cipher(block3)
37 cipher <- paste0(c1, c2, c3)
38 cipher <- sub("\\s+$", "", gsub("(.{3})", "\\1 ",
   cipher))
39 print(cipher)
40 da <- d
41 db <- -1 * b
42 dc <- -1 * c
43 dd <- a
44 bl_v <- vector()
45 cph <- gsub(" ", "", cipher, fixed = TRUE)
46 blocks <- sub("\\s+$", "", gsub("(.{2})", "\\1 ",
   cph))
47 block1 <- substr(blocks, 1, 2)
48 p1 <- decrypt(block1)
49 block2 <- substr(blocks, 4, 5)
50 p2 <- decrypt(block2)
51 block3 <- substr(blocks, 7, 8)

```

```

52 p3 <- decrypt(block3)
53 secret_msg <- paste0(p1, p2, p3)
54 secret_msg <- sub("\\s+$", "", gsub("(.{3})", "\\1 ",
    , secret_msg))
55 print(secret_msg)

```

---

**R code Exa 10.3** example of cryptographic systems involving modular exponentiation

```

1 #page 204
2 library(gtools)
3 library(stringr)
4 encipher <- function(b, p) {
5   two <- (b ^ 2) %% p
6   four <- (two ^ 2) %% p
7   eight <- (four ^ 2) %% p
8   sixteen <- (eight ^ 2) %% p
9   nineteen <- (b * two * sixteen) %% p
10  return(nineteen)
11 }
12 message <- "SEND MONEY"
13 p <- 2609
14 k <- 19
15 char <- " "
16 plain_text <- gsub(" ", "[", message)
17 plain_text <- strsplit(plain_text, "")
18 plain_text_number <- vector()
19 encrypted_message <- vector()
20 for (ch in plain_text) {
21   plain_text_number <- append(plain_text_number, asc
    (ch) - 65)
22 }
23 block1 <- plain_text_number[1] * 100 + plain_text_
    number[2]
24 block2 <- plain_text_number[3] * 100 + plain_text_

```

```

    number[4]
25 block3 <- plain_text_number[5] * 100 + plain_text_
    number[6]
26 block4 <- plain_text_number[7] * 100 + plain_text_
    number[8]
27 block5 <- plain_text_number[9] * 100 + plain_text_
    number[10]
28 encrypted_message <- append(encrypted_message,
    encipher(block1, p))
29 encrypted_message <- append(encrypted_message,
    encipher(block2, p))
30 encrypted_message <- append(encrypted_message,
    encipher(block3, p))
31 encrypted_message <- append(encrypted_message,
    encipher(block4, p))
32 encrypted_message <- append(encrypted_message,
    encipher(block5, p))
33 for (i in seq_len(length(encrypted_message))) {
34   encrypted_message[i] <- str_pad(encrypted_message[
    i], 4, pad = "0")
35 }
36 print(encrypted_message)
37 n <- round((1 - 4 * p) / k)
38 recovery_n <- (p - 1) + n
39 print(recovery_n)

```

---

**R code Exa 10.4** an illustration of the RSA public key algorithm

```

1 #page 206
2 library(stringr)
3 library(gtools)
4 phi <- function(n) {
5   for (num in 2 : sqrt(n))
6     if (n %% num == 0) {
7       p <- num

```

```

8       q <- n / num
9     }
10    return((p - 1) * (q - 1))
11  }
12  encipher <- function(b, n) {
13    two <- (b ^ 2) %% n
14    four <- (two ^ 2) %% n
15    eight <- (four ^ 2) %% n
16    sixteen <- (eight ^ 2) %% n
17    thirty_two <- (sixteen ^ 2) %% n
18    forty_seven <- (b * two * four * eight * thirty_
      two) %% n
19    return(forty_seven)
20  }
21  message <- "NO WAY TODAY"
22  n <- 2701
23  k <- 47
24  plain_text_number <- vector()
25  encrypted_message <- vector()
26  plaintext <- gsub(" ", "[", message, fixed = TRUE)
27  p_split <- strsplit(plaintext, "")
28  for (ch in p_split) {
29    plain_text_number <- append(plain_text_number, asc
      (ch) - 65)
30  }
31  block1 <- plain_text_number[1] * 100 + plain_text_
      number[2]
32  block2 <- plain_text_number[3] * 100 + plain_text_
      number[4]
33  block3 <- plain_text_number[5] * 100 + plain_text_
      number[6]
34  block4 <- plain_text_number[7] * 100 + plain_text_
      number[8]
35  block5 <- plain_text_number[9] * 100 + plain_text_
      number[10]
36  block6 <- plain_text_number[11] * 100 + plain_text_
      number[12]
37  encrypted_message <- append(encrypted_message,

```



```

    encipher(block1, n))
38 encrypted_message <- append(encrypted_message,
    encipher(block2, n))
39 encrypted_message <- append(encrypted_message,
    encipher(block3, n))
40 encrypted_message <- append(encrypted_message,
    encipher(block4, n))
41 encrypted_message <- append(encrypted_message,
    encipher(block5, n))
42 encrypted_message <- append(encrypted_message,
    encipher(block6, n))
43 for (i in seq_len(length(encrypted_message))) {
44   encrypted_message[i] <- str_pad(encrypted_message
    [i], 4, pad = "0")
45 }
46 print(encrypted_message)
47 phi <- phi(n)
48 for (j in 2 : phi - 1) {
49   if ((k * j) %% phi == 1) {
50     return(j)
51   }
52 }
53 print(j)

```

---

**R code Exa 10.5** to solve the superincreasing knapsack problem

```

1 #page 210
2 lhs <- 28
3 co_x1 <- 3
4 co_x2 <- 5
5 co_x3 <- 11
6 co_x4 <- 20
7 co_x5 <- 41
8 ans <- vector()
9 if (co_x5 > lhs) {

```

```

10   x5 <- 0
11 }
12 if (co_x4 < lhs) {
13   if ((co_x1 + co_x2 + co_x3) < lhs) {
14     x4 <- 1
15     ans <- append(ans, co_x4)
16   }
17 }
18 lhs <- lhs - (co_x5 * x5) - (co_x4 * x4)
19 if (co_x3 > lhs) {
20   x3 <- 0
21 }
22 if (co_x2 < lhs) {
23   if ((co_x1 + co_x2) == lhs) {
24     ans <- append(ans, co_x1)
25     ans <- append(ans, co_x2)
26   }
27 }
28 ans <- sort(ans)
29 print(ans)

```

---

**R code Exa 10.6** A public key cryptosystem based on the knapsack problem

```

1 #page 212
2 library(binaryLogic)
3 secret_key <- c(3, 5, 11, 20, 41)
4 m <- 85
5 a <- 44
6 mm <- vector()
7 cipher_text <- vector()
8 encryption_key <- (secret_key * a) %% m
9 message <- "HELP US"
10 plain_text <- gsub(" ", "", message)
11 for (ch in plain_text) {

```

```

12 mm <- append(mm, asc(ch) - 65)
13 }
14 mm <- as.binary(mm, size = 2, n = 5)
15 for (num in mm) {
16   sum <- 0
17   for (bit in 1 : 5) {
18     sum <- sum + ((as.integer(num[bit])) * encryption
19       _key[bit])
20   }
21   cipher_text <- append(cipher_text, sum)
22 }
23 print(cipher_text)

```

---

**R code Exa 10.7** to encrypt a message using knapsack

```

1 #page 213
2 library(binaryLogic)
3 library(gtools)
4 secret_key <- c(3, 5, 11, 20, 41, 83, 179, 344, 690,
5   1042)
6 m <- 2618
7 a <- 929
8 count <- 0
9 digit <- 0
10 big_m <- vector()
11 block <- vector()
12 cipher_text <- vector()
13 encryption_key <- (secret_key * a) %% m
14 message <- "NOT NOW"
15 plain_text <- gsub(" ", "", message)
16 for (ch in plain_text) {
17   big_m <- append(big_m, asc(ch) - 65)
18 }
19 big_m <- as.binary(big_m, signed = FALSE,
20   littleEndian = FALSE, size = 2, n = 5, logic =

```

```

    FALSE)
19 for (cond in big_m) {
20   digit <- digit + 1
21   for (n in 1:5) {
22     if (digit %% 2) {
23       if (cond[n]) {
24         count <- count + encrytion_key[n]
25       }
26     } else {
27       if (cond[n]) {
28         count <- count + encrytion_key[n + 5]
29       }
30       if (n == 5) {
31         print(count)
32         count <- 0
33       }
34     }
35   }
36 }

```

---

**R code Exa 10.8** illustrate the selection of the public key

```

1 #page 215
2 p <- 113
3 r <- 3
4 k <- 37
5 two <- (r ^ 2) %% p
6 four <- (two ^ 2) %% p
7 eight <- (four ^ 2) %% p
8 sixteen <- (eight ^ 2) %% p
9 thirty_two <- (sixteen ^ 2) %% p
10 a <- (r * four * thirty_two) %% p
11 public_key <- c(p, r, a)
12 print(public_key)

```

---

R code Exa 10.9 to encrypt a message using ElGamal

```
1 #page 216
2 library(base)
3 message <- "SELL NOW"
4 k <- 15
5 public_key <- c(43, 3, 22)
6 p <- public_key[1]
7 r <- public_key[2]
8 a <- public_key[3]
9 j <- 23
10 m <- vector()
11 m_ <- ""
12 plain_text <- gsub(" ", "", message)
13 for (ch in plain_text) {
14   m <- append(m, asc(ch) - 65)
15 }
16 two <- (r ^ 2) %% p
17 four <- (two ^ 2) %% p
18 eight <- (four ^ 2) %% p
19 sixteen <- (eight ^ 2) %% p
20 r_digit <- (r * two * four * sixteen) %% p
21 two <- (a ^ 2) %% p
22 four <- (two ^ 2) %% p
23 eight <- (four ^ 2) %% p
24 sixteen <- (eight ^ 2) %% p
25 digit <- (a * two * four * sixteen) %% p
26 for (b in m) {
27   str <- (digit * b) %% p
28   if (floor(str / 10) == 0) {
29     m_ <- paste(m_, "0", toString(str), sep = "")
30   } else {
31     m_ <- paste(m_, toString(str), sep = "")
32   }
```

```

33 }
34 s <- substr(m_, 1, 2)
35 s1 <- paste0("(", r_digit, ", ", s, ")")
36 s <- substr(m_, 3, 4)
37 s2 <- paste0("(", r_digit, ", ", s, ")")
38 s <- substr(m_, 5, 6)
39 s3 <- paste0("(", r_digit, ", ", s, ")")
40 s <- substr(m_, 7, 8)
41 s4 <- paste0("(", r_digit, ", ", s, ")")
42 s <- substr(m_, 9, 10)
43 s5 <- paste0("(", r_digit, ", ", s, ")")
44 s <- substr(m_, 11, 12)
45 s6 <- paste0("(", r_digit, ", ", s, ")")
46 s <- substr(m_, 13, 14)
47 s7 <- paste0("(", r_digit, ", ", s, ")")
48 cipher_text <- paste0(s1, s2, s3, s4, s5, s6, s7)
49 print(cipher_text)

```

---

**R code Exa 10.10** Using ElGamal cryposystem to authenticate a received message

```

1 #page 217
2 p <- 43
3 r <- 3
4 a <- 22
5 k <- 15
6 b <- 13
7 j <- 25
8 message <- "SELL NOW"
9 c <- (r ^ j) %% p
10 digit <- (b - c * k) %% (p - 1)
11 for (d in 1 : 20) {
12   if (((j * d) %% (p - 1)) == digit) {
13     break ()
14   }

```

```
15 }
16 ans <- c(c, d)
17 print(ans)
18 v1 <- ((a ^ c) %% p * (c ^ d) %% p) %% p
19 v2 <- (r ^ B) %% p
20 if (v1 == v2) {
21   print("TRUE")
22 }
```

---

## Chapter 13

# REPRESENTATION OF INTEGERS AS SUMS OF SQUARES

R code Exa 13.1 to represent a positive integer as sum of two squares

```
1 #page 268
2 perfect_sq <- function(a) {
3   sq <- sqrt(a)
4   flr <- floor(sq)
5   if ((sq - flr) == 0) {
6     return(TRUE)
7   }else {
8     return(FALSE)
9   }
10 }
11 n <- 54145
12 p <- vector()
13 k <- vector()
14 ans <- list()
15 equation <- list()
16 i <- 0
17 while (n %% 2 == 0) {
```



```

18   i <- i + 1
19   n <- n / 2
20 }
21 if (i != 0) {
22   p <- append(p, 2)
23   k <- append(k, i)
24 }
25 for (num in 3 : (n - 1)) {
26   if (num %% 2 == 1) {
27     i <- 0
28     while (n %% num == 0) {
29       i <- i + 1
30       n <- n / num
31     }
32     if (i != 0) {
33       p <- append(p, num)
34       k <- append(k, i)
35     }
36   }
37 }
38 for (num in length(p)) {
39   if (k[num] == 1) {
40     if ((k[num] %% 4) == 1) {
41       if (perfect_sq(p[num] - 1)) {
42         square <- p[num] - 1
43         equation <- append(equation, 1)
44         equation <- append(equation, sqrt(square))
45         ans <- append(list(ans), list(equation))
46       }else if (perfect_sq(p[num] - 4)) {
47         square <- p[num] - 4
48         equation <- append(equation, 1)
49         equation <- append(equation, sqrt(square))
50         ans <- append(list(ans), list(equation))
51       }
52     }
53   }else {
54     ans <- append(ans, p[num])
55   }

```

```
56   }
57   print(ans)
```

---

**R code Exa 13.2** to prove Lemma 2

```
1 #page 274
2 p <- 17
3 s1 <- vector()
4 s2 <- vector()
5 s1int <- vector()
6 s2int <- vector()
7 for (n in 0 : ((p - 1) / 2)) {
8   s1 <- append(s1, ((1 + (n ^ 2))))
9   s2 <- append(s2, (- (n ^ 2)))
10 }
11 s1int <- s1 %% 17
12 s2int <- s2 %% 17
13 for (x in s1int) {
14   for (y in s2int) {
15     if (x == 0 | x == 1) {
16       next ()
17     }
18     if (y == 0 | y == 1) {
19       next ()
20     }
21     if ((1 + (x ^ 2) %% p) == y) {
22       x0 <- x
23       y0 <- y
24       return()
25     }
26   }
27 }
28 b <- which(s2int == y0)
29 y0 <- s2[b]
30 y <- sqrt(abs(y0))
```

```

31 x <- x0
32 print(x)
33 print(y)
34 k <- (1 + (x ^ 2) + (y ^ 2)) / p
35 print(k)

```

---

**R code Exa 13.3** to write an integer as sum of four squares

```

1 #page 277
2 perf_sq <- function(i) {
3   sqr <- sqrt(i)
4   sqr_round <- round(sqrt(i))
5   if ((sqr - sqr_round) == 0) {
6     return(TRUE)
7   } else {
8     return(FALSE)
9   }
10 }
11 n <- 459
12 sq <- vector()
13 for (i in 4 : n) {
14   if (perf_sq(i)) {
15     sq <- append(sq, sqrt(i))
16   }
17 }
18 for (a in sq) {
19   for (b in sq) {
20     for (c in sq) {
21       for (d in sq) {
22         if (b >= a | c >= b | d >= c) {
23           next ()
24         }
25         if ((a * a + b * b + c * c + d * d) == n) {
26           x <- a
27           y <- b

```

```
28         z <- c
29         w <- d
30     }
31 }
32 }
33 }
34 }
35 print(x)
36 print(y)
37 print(z)
38 print(w)
```

---

# Chapter 15

## CONTINUED FRACTIONS

R code Exa 15.3 solve a linear Diophantine equation

```
1 #page 316
2 library(MASS)
3 getfracs <- function(frac) {
4   tmp <- strsplit(frac, "/")[[1]]
5   list(num = as.numeric(tmp[1]), deno = as.numeric(
6     tmp[2]))
7 }
8 convergents <- function(cf, p, q) {
9   l <- length(cf)
10  p <- append(p, cf[1])
11  q <- append(q, 1)
12  for (n in 2 : l) {
13    s <- 0
14    t <- n
15    repeat {
16      if (t == n | t == (n + 1)) {
17        s <- as.fractions(s + (1 / cf[n]))
18      } else {
19        s <- (1 / s) + (1 / cf[n])
20      }
21    }
22  }
```

```

21     if (n == 1) {
22         break
23     }
24 }
25 s <- (1 / s) + cf[1]
26 s <- (as.fractions(s))
27 s <- attr(s, "fracs")
28 fracs <- getfracs(s)
29 p <- append(p, fracs$num)
30 q <- append(q, fracs$den)
31 }
32 print(p)
33 q[2] <- 1
34 print(q)
35 x <- c * q[3]
36 y <- (- c) * p[3]
37 print(x)
38 print(y)
39 }
40 eucli <- function(a, b) {
41     cf <- vector()
42     repeat {
43         cf <- append(cf, floor(a / b))
44         r <- a %% b
45         if (r == 0) {
46             break
47         }
48         a <- b
49         b <- r
50     }
51     return(cf)
52 }
53 gcd <- function(x, y) {
54     while (y) {
55         temp <- y
56         y <- x %% y
57         x <- temp
58     }

```

```

59   if (x < 0) {
60     return(- x)
61   }else {
62     return(x)
63   }
64 }
65 p <- vector()
66 q <- vector()
67 a <- 172
68 b <- 20
69 c <- 1000
70 g <- gcd(a, b)
71 a <- a / g
72 b <- b / g
73 c <- c / g
74 cf <- eucli(a, b)
75 convergents(cf, p, q)

```

---

**R code Exa 15.5** to find continued fraction expansion of a number

```

1 #page 326
2 n <- sqrt(23)
3 x <- vector()
4 a <- vector()
5 x[1] <- n
6 a[1] <- floor(x[1])
7 for (i in 2 : 10) {
8 x[i] <- 1 / (x[i - 1] - a[i - 1])
9 a[i] <- floor(x[i])
10 }
11 print(a)

```

---

**R code Exa 15.6** to find continued fraction expansion of a number

```

1 #page 327
2 n <- pi
3 x <- vector()
4 a <- vector()
5 x[1] <- n
6 a[1] <- floor(x[1])
7 for (i in 2 : 10) {
8   x[i] <- 1 / (x[i - 1] - a[i - 1])
9   a[i] <- floor(x[i])
10 }
11 print(a)

```

---

**R code Exa 15.7** an example of illustrating the corollary to sought a fraction

```

1 #page 337
2 library(MASS)
3 library(fractional)
4 gcd <- function(x, y) {
5   while (y) {
6     temp <- y
7     y <- x %% y
8     x <- temp
9   }
10  if (x < 0) {
11    return(- x)
12  }else {
13    return(x)
14  }
15 }
16 farey_seq <- function(i) {
17   f <- vector()
18   f[1] <- 0 / 1
19   f[2] <- 1
20   for (m in 2 : i) {

```



```

21     f <- append(f, 1 / m)
22     for (g in 2 : m) {
23         if (gcd(g, m) == 1) {
24             f <- append(f, g / m)
25         }
26     }
27 }
28 f <- sort(as.fractions(f))
29 return(f)
30 }
31 n <- 5
32 x <- sqrt(7)
33 val <- x - 2
34 fn <- farey_seq(5)
35 for (k in seq_len(length(fn))) {
36     if ((val > fn[k]) & (val < fn[k + 1])) {
37         nu1 <- numerators(fn[k])
38         d1 <- denominators(fn[k])
39         nu2 <- nu1 + numerators(fn[k + 1])
40         d2 <- d1 + denominators(fn[k + 1])
41         if (nu2 / d2 > val) {
42             u <- nu1
43             v <- d1
44         } else {
45             u <- nu2 - nu1
46             v <- d2 - d1
47         }
48     }
49 }
50 if (val - (u / v) < 1 / (v * (n + 1))) {
51     ans <- as.fractions((u / v) + 2)
52 }
53 print(ans)

```

---

**R code Exa 15.8** to solve an application of above theorem

```

1 #page 347
2 convergents <- function(cf) {
3   l <- length(cf)
4   ss <- vector()
5   for (n in 2 : l) {
6     s <- 0
7     t <- n
8     repeat {
9       if (t == n) {
10        s <- (s + (1 / cf[n]))
11      } else {
12        s <- 1 / (s + cf[n])
13      }
14      n <- n - 1
15      if (n == 1) {
16        break
17      }
18    }
19    s <- s + cf[1]
20    s <- fractional(s)
21    ss <- append(ss, s)
22  }
23  return(ss)
24 }
25 cont_frac <- function(i) {
26 n <- sqrt(i)
27 x <- vector()
28 a <- vector()
29 x[1] <- n
30 a[1] <- floor(x[1])
31 for (k in 2 : 12) {
32   x[k] <- 1 / (x[k - 1] - a[k - 1])
33   a[k] <- floor(x[k])
34 }
35 return(a)
36 }
37 d <- 7
38 p <- vector()

```

```

39 q <- vector()
40 l <- vector()
41 cf <- cont_frac(d)
42 n <- 4
43 p <- append(p, cf[1])
44 q <- append(q, 1)
45 s <- convergents(cf)
46 for (j in 2 : length(s)) {
47   p <- append(p, numerators(s[j]))
48   q <- append(q, denominators(s[j]))
49 }
50 q[2] <- 1
51 if (n %% 2 == 0) {
52   for (k in 1 : 3) {
53     l <- append(l, (k * n) - 1)
54   }
55 }else {
56   for (k in 1: 3) {
57     l <- append(l, (2 * k * n) - 1)
58   }
59 }
60 for (num in l) {
61   print(p[num])
62   print(q[num])
63 }

```

---

**R code Exa 15.9** to find a solution of an equation for the smallest positive integer

```

1 #page 347
2 library(fractional)
3 convergents <- function(cf) {
4   l <- length(cf)
5   ss <- vector()
6   for (n in 2 : l) {

```

```

7     s <- 0
8     t <- n
9     repeat {
10      if (t == n) {
11        s <- (s + (1 / cf[n]))
12      } else {
13        s <- 1 / (s + cf[n])
14      }
15      n <- n - 1
16      if (n == 1) {
17        break
18      }
19    }
20    s <- s + cf[1]
21    s <- fractional(s)
22    ss <- append(ss, s)
23  }
24  return(ss)
25 }
26 cont_frac <- function(i) {
27   n <- sqrt(i)
28   x <- vector()
29   a <- vector()
30   x[1] <- n
31   a[1] <- floor(x[1])
32   for (k in 2 : 10) {
33     x[k] <- 1 / (x[k - 1] - a[k - 1])
34     a[k] <- floor(x[k])
35   }
36   return(a)
37 }
38 d <- 13
39 p <- vector()
40 q <- vector()
41 cf <- cont_frac(d)
42 n <- 5
43 p <- append(p, cf[1])
44 q <- append(q, 1)

```

```
45 s <- convergents(cf)
46 for (j in 2 : length(s)) {
47   p <- append(p, numerators(s[j]))
48   q <- append(q, denominators(s[j]))
49 }
50 q[2] <- 1
51 k <- 1
52 if (n %% 2 == 0) {
53   l <- (k * n) - 1
54 }else {
55   l <- (2 * k * n) - 1
56 }
57   print(p[l])
58   print(q[l])
```

---

# Chapter 16

## SOME MODERN DEVELOPMENTS

R code Exa 16.1 factorization of a number using Pollards method

```
1 #page 359
2 gcd <- function(x, y) {
3   while (y) {
4     temp <- y
5     y <- x %% y
6     x <- temp
7   }
8   if (x < 0) {
9     return(- x)
10  }else {
11    return(x)
12  }
13 }
14 f <- function(x) {
15   return((x * x) - 1)
16 }
17 n <- 30623
18 x <- vector()
19 x[1] <- 3
```

```

20 for (k in 2 : 9) {
21   x[k] <- f(x[k - 1]) %% n
22 }
23 for (k in seq_len(9 / 2)) {
24   a <- x[2 * k] - x[k]
25   g <- gcd(a, n)
26   if (g != 1) {
27     break
28   }
29 }
30 p <- n / g
31 print(p)
32 print(g)
33 x <- x %% g
34 print(x)

```

---

**R code Exa 16.2** to obtain a nontrivial divisor of a number

```

1 #page 361
2 library(gmp)
3 gcd <- function(x, y) {
4   while (y) {
5     temp <- y
6     y <- x %% y
7     x <- temp
8   }
9   if (x < 0) {
10    return(- x)
11  }else {
12    return(x)
13  }
14 }
15 n <- 2987
16 a <- 2
17 q <- 7

```

```

18 s <- a
19 s <- as.bigz(s)
20 for (pow in 2 : q) {
21   s <- (s ^ pow) %% n
22 }
23 s <- asNumeric(s)
24 ans <- gcd(s - 1, n)
25 print(ans)

```

---

**R code Exa 16.3** to factor a number using the continued fraction factorization method

```

1 #page 362
2 library(fractional)
3 gcd <- function(x, y) {
4   while (y) {
5     temp <- y
6     y <- x %% y
7     x <- temp
8   }
9   if (x < 0) {
10    return(- x)
11  }else {
12    return(x)
13  }
14 }
15 convergents <- function(cf) {
16   l <- length(cf)
17   ss <- vector()
18   ss <- append(ss, cf[1])
19   for (n in 2 : l) {
20     s <- 0
21     t <- n
22     repeat {
23       if (t == n) {

```



```

24         s <- (s + (1 / cf[n]))
25     } else {
26         s <- 1 / (s + cf[n])
27     }
28     n <- n - 1
29     if (n == 1) {
30         break
31     }
32 }
33 s <- s + cf[1]
34 s <- fractional(s)
35 ss <- append(ss, numerators(s))
36 }
37 return(ss)
38 }
39 cont_frac <- function(i) {
40     n <- sqrt(i)
41     x <- vector()
42     a <- vector()
43     x[1] <- n
44     a[1] <- floor(x[1])
45     for (k in 2 : 9) {
46         x[k] <- 1 / (x[k - 1] - a[k - 1])
47         a[k] <- floor(x[k])
48     }
49     return(a)
50 }
51 n <- 3427
52 s <- vector()
53 t <- vector()
54 a <- cont_frac(n)
55 p <- convergents(a)
56 s <- append(s, 0)
57 t <- append(t, 1)
58 for (num in seq_len(8)) {
59     s[num + 1] <- (a[num] * t[num]) - s[num]
60     t[num + 1] <- (n - (s[num + 1] ^ 2)) / t[num]
61 }

```

```

62 for (num in t) {
63   if (num == 1) {
64     next ()
65   }
66   sq <- sqrt(num)
67   d <- round(sqrt(num))
68   if (d == sq) {
69     index <- num
70     return()
71   }
72 }
73 ans <- gcd(p[index - 1] + sqrt(t[index]), n)
74 ans2 <- gcd(p[index - 1] - sqrt(t[index]), n)
75 print(ans)
76 print(ans2)

```

---

**R code Exa 16.4** to factor a number using the continued fraction factorization method

```

1 #page 363
2 library(fractional)
3 gcd <- function(x, y) {
4   while (y) {
5     temp <- y
6     y <- x %% y
7     x <- temp
8   }
9   if (x < 0) {
10    return(- x)
11  }else {
12    return(x)
13  }
14 }
15 convergents <- function(cf) {
16   l <- length(cf)

```

```

17  ss <- vector()
18  ss <- append(ss, cf[1])
19  for (n in 2 : 1) {
20    s <- 0
21    t <- n
22    repeat {
23      if (t == n) {
24        s <- (s + (1 / cf[n]))
25      } else {
26        s <- 1 / (s + cf[n])
27      }
28      n <- n - 1
29      if (n == 1) {
30        break
31      }
32    }
33    s <- s + cf[1]
34    s <- fractional(s)
35    ss <- append(ss, numerators(s))
36  }
37  return(ss)
38 }
39 cont_frac <- function(i) {
40  n <- sqrt(i)
41  x <- vector()
42  a <- vector()
43  x[1] <- n
44  a[1] <- floor(x[1])
45  for (k in 2 : 9) {
46    x[k] <- 1 / (x[k - 1] - a[k - 1])
47    a[k] <- floor(x[k])
48  }
49  return(a)
50 }
51 n <- 2059
52 s <- vector()
53 t <- vector()
54 a <- cont_frac(n)

```

```

55 p <- convergents(a)
56 s <- append(s, 0)
57 t <- append(t, 1)
58 for (num in seq_len(8)) {
59   s[num + 1] <- (a[num] * t[num]) - s[num]
60   t[num + 1] <- (n - (s[num + 1] ^ 2)) / t[num]
61 }
62 for (num in t) {
63   for (num2 in t)
64     if (num == 1 | num2 == 1 | num == num2) {
65       next ()
66     }
67   sq <- sqrt(num * num2)
68   d <- round(sqrt(num * num2))
69   if (d == sq) {
70     return()
71   }
72 }
73 index <- match(num, t)
74 index2 <- match(num2, t)
75 x <- sqrt(t[index] * t[index2])
76 y <- (p[index - 1] * p[index2 - 1]) %% n
77 ans <- gcd(x + y, n)
78 ans2 <- n / ans
79 print(ans)
80 print(ans2)

```

---

**R code Exa 16.5** an example of the quadratic sieve algorithm

```

1 #page 365
2 library(primes)
3 factorize <- function(n, f) {
4   k <- vector()
5   for (g in f) {
6     i <- 0

```

```

7     if (g == - 1) {
8         if (n < 0) {
9             n <- -1 * n
10            k <- append(k, 1)
11        } else {
12            k <- append(k, 0)
13        }
14        next ()
15    }
16    while (n %% g == 0) {
17        i <- i + 1
18        n <- n / g
19    }
20    if (i != 0) {
21        k <- append(k, i)
22    } else {
23        k <- append(k, 0)
24    }
25 }
26 if (n == 1) {
27     return(k)
28 } else {
29     return(66)
30 }
31 }
32 fofx <- function(x) {
33     return((x^2) - n)
34 }
35 check_residue <- function(a, p) {
36     if (a == -1) {
37         return(-1)
38     }
39     if (a > 1) {
40         a <- a %% p
41     }
42     if (a == 1) {
43         return(1)
44     }

```

```

45   if (a %% 2 == 0) {
46     if (p %% 8 == 1 | p %% 8 == 7) {
47       a <- a / 2
48     } else {
49       a <- (- 1 * a) / 2
50     }
51     return(check_residue(a, p))
52   }
53   if (a %% 2 != 0 && p %% 2 != 0) {
54     if ((a %% 4 == 3) && (p %% 4 == 3)) {
55       return(check_residue(- 1, a))
56     } else {
57       return(check_residue(p, a))
58     }
59   }
60   return(0)
61 }
62 n <- 9487
63 kdata <- vector()
64 x <- floor(sqrt(n))
65 fb <- vector()
66 ex <- vector()
67 fb[1] <- -1
68 fb[2] <- 2
69 ap <- generate_primes(max = 30)
70 for (num in ap) {
71   if (num == 2) {
72     next ()
73   }
74   if (check_residue(n, num) == 1) {
75     fb <- append(fb, num)
76   }
77 }
78 f <- seq(x - 16, x + 16)
79 for (w in f) {
80   k <- factorize(fofx(w), fb)
81   if ( length(k) == 1) {
82     ex <- append(ex, w)

```

```

83     next ()
84   }
85   kdata <- c(kdata, k)
86 }
87 f <- f[!f %in% ex]
88 r <- length(fb)
89 c <- length(kdata) / r
90 p <- matrix(kdata, nrow = r, ncol = c, dimnames =
      list(fb, f))
91 for (i in seq_len(length(f))) {
92   for (j in i : length(f)) {
93     for (k in j : length(f)) {
94       if (i == j | j == k | k == i) {
95         next ()
96       }
97       for (h in seq_len(length(fb))) {
98         m <- (p[h, i] + p[h, j] + p[h, k]) %% 2
99         if (m != 0) {
100           break
101         } else if (h == length(fb)) {
102           a <- i
103           b <- j
104           c <- k
105           return()
106         }
107       }
108     }
109   }
110 }
111 lh <- (f[a] * f[b] * f[c]) %% n
112 sum <- 1
113 ma <- (p[, a] + p[, b] + p[, c])
114 for (h in seq_len(length(fb))) {
115   if (ma[h] == 0) {
116     next ()
117   } else if (ma[h] == 2) {
118     sum <- sum * fb[h]
119   } else {

```

```

120     sum <- sum * ((fb[h]) ^ (ma[h] - 2))
121   }
122 }
123 if (sum < 0) {
124   sum <- -1 * sum
125 }
126 sum <- sum %% n
127 ans <- gcd(sum + lh, n)
128 print(ans)
129 ans2 <- n / ans
130 print(ans2)

```

---

**R code Exa 16.6** Using a theorem to check if a number is prime

```

1 #page 367
2 mod <- function(a, b) {
3   ans <- 1
4   for (num in 1: b) {
5     ans <- (ans * a) %% n
6   }
7   if (ans == n - 1) {
8     ans <- -1
9   }
10  return(ans)
11 }
12 factorize <- function(n) {
13   number <- n
14   p <- vector()
15   i <- 0
16   while ((n %% 2) == 0) {
17     i <- i + 1
18     n <- n / 2
19   }
20   if (i != 0) {
21     p <- append(p, 2)

```



```

22   }
23   for (num in 3 : sqrt(number)) {
24     if (num %% 2 == 1) {
25       i <- 0
26       while (n %% num == 0) {
27         i <- i + 1
28         n <- n / num
29       }
30       if (i != 0) {
31         p <- append(p, num)
32       }
33     }
34   }
35   p <- append(p, n)
36   return(p)
37 }
38 n <- 997
39 a <- 7
40 m <- vector()
41 modulus <- mod(a, n-1)
42 print(modulus)
43 p <- factorize(n-1)
44 for (num in p) {
45   m <- append(m, mod(a, (n - 1) / num))
46 }
47 print(m)

```

---

**R code Exa 16.7** to find four square roots of a modulo n

```

1 #page 372
2 library(primes)
3 solve_on <- function(a, b) {
4   for (i in seq_len(10)) {
5     c <- (1 - q * i) / p
6     cr <- round(c)

```

```

7     if (c == cr) {
8         d <- i
9         break
10    }
11  }
12  x <- p * c * b + q * d * a
13  return(x %% n)
14 }
15 a <- 324
16 n <- 391
17 ans <- vector()
18 for (h in generate_primes(max = sqrt(n))) {
19     if (n %% h == 0) {
20         p <- h
21         q <- n / h
22         break
23     }
24 }
25 x1 <- a %% p
26 x2 <- a %% q
27 x2 <- sqrt(q + x2)
28 ans <- append(ans, solve_on(x1, -x2))
29 ans <- append(ans, solve_on(-x1, x2))
30 ans <- append(ans, solve_on(x1, x2))
31 ans <- append(ans, solve_on(- x1, - x2))
32 ans <- sort(ans)
33 print(ans)

```

---

**R code Exa 16.8** to solve an example of blums game

```

1 #page 374
2 mod <- function(a, b, n) {
3     ans <- 1
4     for (num in 1: b) {
5         ans <- (ans * a) %% n

```

```

6   }
7   if (ans == n - 1) {
8     ans <- -1
9   }
10  return(ans)
11 }
12 solve_on <- function(a, b) {
13   for (i in seq_len(20)) {
14     c <- (1 - q * i) / p
15     cr <- round(c)
16     if (c == cr) {
17       d <- i
18       break
19     }
20   }
21   x <- p * c * b + q * d * a
22   return(x %% n)
23 }
24 gcd <- function(x, y) {
25   while (y) {
26     temp <- y
27     y <- x %% y
28     x <- temp
29   }
30   if (x < 0) {
31     return(- x)
32   } else {
33     return(x)
34   }
35 }
36 p <- 43
37 q <- 71
38 n <- p * q
39 s <- 192
40 ans <- vector()
41 a <- (s ^ 2) %% n
42 x1 <- a %% p
43 x2 <- a %% q

```

```
44 if (p %% 4 == 3 && q %% 4 == 3) {
45   x1 <- mod(x1, ((p + 1) / 4), p)
46   x2 <- mod(x2, ((q + 1) / 4), q)
47 }
48 x1 <- p - x1
49 x2 <- q - x2
50 ans <- append(ans, solve_on(x1, -x2))
51 ans <- append(ans, solve_on(-x1, x2))
52 ans <- append(ans, solve_on(x1, x2))
53 ans <- append(ans, solve_on(- x1, - x2))
54 ans <- sort(ans)
55 guess <- sample(ans, 1)
56 g1 <- gcd(s + guess, n)
57 g2 <- gcd(s - guess, n)
58 if (g1 == 1 && g2 == n) {
59   print("Alice wins!")
60 } else {
61   print("Bob wins!")
62 }
```

---