COMPUTATIONAL LABORATORIES IN NUMBER THEORY

CLINT

THIRD EDITION

by

Hugh L. Montgomery

A Supplement to

An Introduction to The Theory of Numbers

Fifth Edition

by

Ivan Niven, Herbert S. Zuckerman, Hugh L. Montgomery

John Wiley & Sons, Inc.

Publication history:

First Edition	December, 1992
Second Edition	September, 1994
Third Edition	January, 2001

Preface to the Third Edition

Throughout its long history, number theory has been characterized by discovery based upon empirically observed numerical patterns. By using a computer with appropriate software, the student can now inspect data that is both more extensive and more accurate than in former times. With this in mind, a set of 73 programs has been prepared for use in the classroom as an aid to instruction, for use by students in individual study and exploration, and also in structured laboratories. These programs are written in Borland's Turbo Pascal version 7.0, running under DOS on IBM PC-compatible machines. Source code is available by request. Some of these programs, such as FacTab and PowerTab, display data in which patterns may be detected. Other programs, such as EuAlgDem and PwrDem1a, offer demonstration of specific algorithms that are employed in computations. Finally, a third class of programs, typified by Factor and GCD, perform useful calculations on demand. The programs relevant to a particular section of NZM are listed in the table Programs by Section. Before embarking on a section, the instructor may wish to experiment with these programs, in order to become familiar with their operation.

It was intended that the algorithms employed in the accompanying programs should be limited to those discussed in the text, so that the student would be in a position to understand exactly what each program is does. As the programs developed, a few exceptions to this rule have crept in, as follows: In the program Ind, for calculating the index (i.e., discrete logarithm), a method of Shanks is used. This is explained in Laboratory 12, in the documentation of the program, and also in the demonstration provided by the program IndDem. In the program ProveP, which is based on Problem 39 at the end of §2.8, an extra device invented by H. C. Williams has been added. For details see the description of this program in the *Reference Guide to Turbo Pascal Programs*, in this manual. The scheme for calculating the Lucas functions, described in §4.4, has not been followed, because the one sidestep formula involves division by 2, which is problematic when the calculations are being done modulo m with m even. For an account of the method actually used, see the description of the program Lucas in the *Reference Guide to Turbo Pascal Programs*. The disadvantage of using only those algorithms found in the text is that in some cases faster execution could have been achieved by using some other algorithm. This particularly the case with programs that involve factoring (the number field sieve is faster), proving primality (the Atkin-Morain and the Adleman-Rumely methods are faster), or locating the roots of polynomial congruences modulo p (the Cantor-Zassenhaus method is *much* faster).

If your students have experience in programming, you may wish to make the source code of these programs available to them. By examining the source code, a student may see in detail how a particular algorithm has been implemented. On the other hand, an effort has been made to design programs whose operation is so natural that very little time will be needed to learn how to use them. Thus students unfamiliar with computers or programming should have no difficulty.

In order to avoid the necessity of programming in multiple precision arithmetic, integers in these programs are limited to at most 10^{18} in size. This is adequate for most purposes, but is a disadvantage in some contexts.

If you encounter any problem with the operation of the Turbo Pascal programs, or have suggestions for their improvement, please communicate your comments to me at `hlm@math.lsa.umich.edu`, or by snail mail.

These computational laboratories are still in an experimental stage. More labs and programs are needed. In addition, some labs may be too long, or too difficult, or may ask the wrong questions. Any thoughts you have would be appreciated. You may want to compose your own, but it is hoped that the ones here at least offer inspiration.

It is a pleasure to thank A. O. L. Atkin, J. D. Brillhart, H. Flanders, D. E. G. Malm, C. Pomerance, J. L. Selfridge, R. C. Vaughan, Ulrike M. A. Vorhauer, and S. S. Wagstaff Jr. for their helpful comments and suggestions.

Hugh L. Montgomery
29 December, 2000

Contents

Programs by Type

CALCULATIONS

Carmichael function car(m)	`car [m]`
Chinese Remainder Theorem	`crt` [$\mathtt{a}_1$ $\mathtt{m}_1$ $\mathtt{a}_2$ $\mathtt{m}_2$]
convert decimal to rational	`d2r [x]`
convert rational to decimal	`r2d [a q]`
determinant modulo m	`detmodm`
discrete logarithm base g of a modulo p	`ind [g a p]`
factor n	
by trial division	`factor [n]`
by $p-1$ method	`p-1 [n [a]]`
by rho method	`rho [n [c]]`
find next prime	`getnextp [x]`
greatest common divisor	`gcd [b c]`
index base g of a modulo p	`ind [g a p]`
Jacobi symbol $\left(\frac{P}{Q}\right)$	`jacobi [P Q]`
Lucas functions U_n, V_n modulo m	`lucas [n [a b] m]`
multiply residue classes modulo m	`mult [a b m]`
order of a modulo m	`order [a m [c]]`
phi function $\phi(n)$	`phi [n]`
$\pi(x)$	`pi [x]`
power a^k modulo m	`power [a k m]`
primitive root of prime p	`primroot [p [a]]`
prove primality of p	`provep [p]`
reduce $ax^2 + bxy + cy^2$	`reduce a b c`
represent n as sum of s k-th powers	`sumspwrs [n s k]`
roots of	
$ax \equiv b \pmod{m}$	`lincon [a b m]`
$f(x) \equiv 0 \pmod{p^j}$	`hensel`
$P(x) \equiv 0 \pmod{m}$	`polysolv`
$x^2 \equiv a \pmod{p}$	`sqrtmodp [a p]`
$A\mathbf{x} = \mathbf{b}$ in integers	`simlinde`
square root modulo p	`sqrtmodp [a p]`
strong pseudoprime test of m base a	`spsp [[a] m]`

DEMONSTRATIONS

Chinese Remainder Theorem	`crtdem`
determinants modulo m	`detdem`
discrete logarithm base g of a modulo p	`inddem [g a p]`
Euclidean algorithm	`eualgdem`

factorization	
by $p-1$ method	`p-1dem`
by rho method	`rhodem [n]`
greatest common divisors	`fastgcd`, `slowgcd`
(see also Euclidean algorithm)	
heapsort algorithm	`hsortdem`
index base g of a modulo p	`inddem [g a p]`
Jacobi symbol $\left(\frac{P}{Q}\right)$	`jacobdem [P Q]`
linear congruence $ax \equiv b \pmod{m}$	`lncndem [a b m]`
Lucas functions	`lucasdem [n [a b] m]`
multiplication of residue classes	`multdem1`, `multdem2`, `multdem3`
order of a modulo m	`orderdem [a m [c]]`
powering algorithm	`pwrdem1a [a k m]`, `pwrdem1b [a k m]`, `pwrdem2 [a k m]`
RSA encryption	`rsa`, `rsapars`
square root modulo p	`sqrtdem [a p]`
strong pseudoprime test of m base a	`spspdem [[a] m]`

TABLES

arithmetic functions $\omega(n), \Omega(n), \mu(n), d(n), \phi(n), \sigma(n)$	`arfcntab`
base conversions for integers	`basestab`
binary quadratic forms	
reduced forms	`qformtab`
forms equivalent to $f(x,y)$	`reduce`
binomial coefficients modulo m	`pascalst`
class numbers	`clanotab`
congruential arithmetic	`cngartab`
discrete logarithms	`indtab`
factorials modulo m	`fctrltab`
Farey fractions	`fareytab`, `fractab`
greatest common divisors	`gcdtab`
indices	`indtab`
intersection of arithmetic progressions	`intaptab`
Jacobi symbols	`jacobtab`
least prime factor	`factab`
linear combinations	`lncomtab`
Lucas functions	`lucastab`
Pascal's triangle modulo m	`pascalst`
order of $a \pmod{m}$	`ordertab`
powers of a modulo m	`powertab`
representations as sums of powers	`sumspwrs`, `wrngtab`
roots of	
$f(x) \equiv 0 \pmod{p^j}$	`hensel`
$P(x) \equiv 0 \pmod{m}$	`polysolv`

Programs by Section

1.2	Div, CoDivTab, SlowGCD, FastGCD, GCD, EuAlgDem, LnComTAb, GCDTab, CoMulTab
1.3	FacTab, Factor, Pi, GetNextP
1.4	PascalsT
2.1	CngArTab, FctrlTab, PowerTab, Phi, Mult, Power, DetModM, DetDem, SumsPwrs
2.2	PolySolv, LinCon, LnCnDem
2.3	IntAPTab, ResComp, CRT, CRTDem, Phi
2.4	MultDem1, MultDem2, MultDem3, PwrDem1a, PwrDem1b, PwrDem2, SPsP, SPsPDem, Rho, RhoDem, P-1, P-1Dem
2.5	RSA, RSAPars
2.6	Hensel
2.7	PolySolv
2.8	Order, OrderDem, PrimRoot, Ind, IndTab, IndDem, HSortDem, ProveP, Car
2.9	SqrtModP, SqrtDem
2.10	CngArTab
3.1	Jacobi, JacobTab
3.2	Jacobi, JacobTab
3.3	JacobDem
3.5	QFormTab, Reduce
3.6	Reduce, SumsPwrs
3.7	ClaNoTab
4.2	ArFcnTab
4.4	Lucas, LucasTab, LucasDem
5.1	SimLinDE
5.2	SimLinDE
6.1	FareyTab, FracTab
7.1	D2R

Warning

The accompanying programs are intended for educational use only. We make no warranty, express or implied, that the programs are free of error, that they meet any particular standard of merchantability, or that the values they yield are accurate. Some of these programs have been put through strenuous tests, but many others have been checked only in the most casual manner. In order to extend the range of integers that may be dealt with, most of these programs use floating-point real arithmetic in their execution. Thus the accuracy of the results cannot be guaranteed, and consequently these programs should not be used for serious mathematical research. Any such use would be entirely at the user's own risk. The author disclaims all liability for direct, incidental, or consequential damages resulting from your use of these programs.

LABORATORY 1

GCDs & The Euclidean Algorithm

Programs Used: SlowGCD, FastGCD, GCD, EuAlgDem,
LnComTab, GCDTab

1. The most direct method of calculating the greatest common divisor of two numbers b and c would be to make a list of the common divisors, and note the value of the largest common divisor. This would involve dividing each of the numbers $1, 2, \ldots, \min(|b|, |c|)$ into both b and c. This brute force method is used by the program SlowGCD. Use SlowGCD to calculate the value of $(1271, 4521)$. (Type `slowgcd [Enter]`, and then you will be prompted for the arguments.) Note how long the calculation took. Formulate a hypothesis about how much longer this program will take to evaluate $(12712, 45212)$. Test your hypothesis by running SlowGCD with these arguments. Note the running time. In trying out this program with various arguments, be careful to use small numbers. If you put in large numbers then the program will run for a long time, and you will have to type `Ctrl-Alt-Del`) to recover its use.

2. The gcd symbol (b, c) is defined for any pair of integers, not both equal to 0. This quantity enjoys four basic identities:

i) $(b, c) = (-b, c)$;
ii) $(b, c) = (b + mc, c)$ for all integers m;
iii) $(b, c) = (c, b)$;
iv) $(b, 0) = |b|$.

By using these identities systematically (recall pp. 10–12 of NZM), we may reduce the size of the arguments until iv) applies, and the value emerges. For example,

$$\begin{aligned}(31, 12) &= (31 - 2 \cdot 12, 12) = (7, 12)\\ &= (12, 7) = (12 - 1 \cdot 7, 7) = (5, 7)\\ &= (7, 5) = (7 - 1 \cdot 5, 5) = (2, 5)\\ &= (5, 2) = (5 - 2 \cdot 2, 2) = (1, 2)\\ &= (2, 1) = (2 - 2 \cdot 1, 1) = (0, 1)\\ &= (1, 0) = 1.\end{aligned}$$

Apply this reasoning to calculate $(127, 49)$, and type `eualgdem 127 49 [Return]` to verify your arithmetic.

3. In the calculation displayed above, we have written down more than we need. Since each new number is the remainder after division, it suffices to write down only these remainders, $31, 12, 7, 5, 2, 1, 0$. The gcd of any two consecutive members of this sequence is constant throughout. When we consider the last two numbers, we see that the gcd is the last positive remainder in the sequence. Sequences generated in this way are very rapidly decreasing, and hence are not very long. Thus the gcd is much more quickly determined by this method—known as the Euclidean Algorithm. The program FastGCD uses this faster method to evaluate gcds. Apply FastGCD to the same pairs of numbers that you used with SlowGCD, and record the running times. Also use FastGCD to calculate the gcd for a pair of 2 digit numbers, a pair of 4 digit numbers, a pair of 8 digit numbers, a pair of 16 digit numbers. Make a record of the numbers used, and the running times. How does the running time seem to depend on the size of the inputs?

4. The Euclidean Algorithm can be modified in various ways to make it still faster. For example, in performing divisions, one may round to the nearest integer instead of rounding down. This gives rise to negative remainders, but the remainders decrease in absolute value a little faster than formerly. For example, to calculate $(31, 12)$ in this way, we would generate the sequence of remainders $31, 12, -5, 2, 1, 0$. This sequence saves one step over the sequence of **3.** above. The program GCD uses this enhanced scheme. Type `gcd 12345 54321 [Enter]`, and note the result. This program will prompt you for the arguments if you forget to put them on the command line. Type `gcd [Enter]`, and follow the prompts.

5. The program LnComTab displays linear combinations $xb + yc$ of two given integers b and c. Start with $b = 9$, $c = 15$. Note that the resulting table is antisymmetric about the origin. Why is this? What is the smallest number (in absolute value) that you see? How is this related to $(9, 15)$? (Recall Theorem 1.4 of NZM.) Is the table periodic in any way? At what locations do you find a 0? Let $\mathcal{C}$ denote a collection of 5 consecutive columns. Show that every number that occurs somewhere in the table is found exactly once in $\mathcal{C}$. Do the same for $\mathcal{R}$, which consists of 3 consecutive rows of the table. Where do the numbers "5" and "3" come from? What would they be replaced by, if the values of b and c were changed? Take now $b = 3$, $c = 5$. How is this new table related to the one you were looking at before? Note that small values on the table follow a line pointing roughly NorthWest-SouthEast. Use the arrow keys to follow these small values. What is the slope of the line along which these small values lie?

6. Type `eualgdem [Enter]`, and then provide the arguments $b = 12345$, $c = 54321$. The remainders are now presented in a neat table. The arguments can be altered without leaving the table. Type `b`, then `54321 [Enter]`, then `c`, and finally `12345 [Enter]`. The two arguments have been reversed. What effect does this have on the sequence of remainders generated? Does this persist in general? Can you prove it?

Substitute some large arguments, say $> 10^{16}$. The table of remainders is now too large to fit on one screen. Use `PgDn` and `PgUp` to scroll down and up through the table.

7. Let $j = j(b, c)$ be the index of the last positive remainder in the sequence of remainders generated by the Euclidean Algorithm, so that $r_j = (b, c)$ and $r_{j+1} = 0$. Thus $j + 1$ divisions have been performed in the calculation. For given b and c, the value of $j(b, c)$

is easily determined by reading the index of the bottom line in the table provided by EuAlDem2. Using this program, try to answer the following questions. What is the least pair of integers b, c with $b \ge c > 0$ such that $j(b,c) = 1$? $= 2$? $= 3$? $= 4$? Can you spot a pattern? Can you prove that it persists?

8. Use EuAlgDem as in **6.** to determine the value of $j(b,c)$. If b and c are large, is $j(b,c)$ necessarily large? For 10 different pairs of "randomly chosen" large values of b and c, record the value of $j(b,c)$. How large is $j(b,c)$ on average?

9. The quotients q_i generated by the Euclidean Algorithm are displayed in the table provided by EuAlgDem. By hand calculation, find a pair b, c of integers, each $> 10^5$, such that $q_i = 1$ for all i. (Hint: Start at the end and work back toward the beginning. If $r_j = 1$ and $r_{j-1} = 1$ then

$$\begin{aligned} r_{j-2} &= r_{j-1}q_j + r_j = 1 \cdot 1 + 1 = 2, \\ r_{j-3} &= r_{j-2}q_{j-1} + r_{j-1} = 2 \cdot 1 + 1 = 3, \end{aligned}$$

and so on.) Similarly, find a pair b and c of integers $> 10^5$ for which $q_i = 10$ for all i. In both cases, confirm your results by using EuAlgDem. Quite clearly, the sequence of q_i could be anything. However, for *most* pairs b, c the q_i follow a definite statistical distribution. About 0.415 of them are $= 1$, about 0.170 of them are $= 2$, about 0.093 of them are $= 3$, and so on. More precisely, we expect that $q_i = k$ for a proportion of approximately

$$\bigl(\log(1 + 1/k) - \log(1 + 1/(k+1))\bigr)/\log 2$$

of the i. Gauss claimed to have proved this, but his proof (if he had one) is unknown. The first known proof was given in 1928 by R. O. Kuz'min. Using modern tools, one finds this result as an easy consequence of the ergodic theorem. Choose a pair of large integers b, c at random, and use EuAlgDem to generate the q_i. How close to the expected distribution are the q_i?

10. As is discussed on pp. 13–15 of NZM, each remainder r_i generated by the Euclidean Algorithm is a linear combination of the b and c that initiated the sequence. That is, $r_i = x_i b + y_i c$. These x_i and y_i are not uniquely determined. (For example, if we replace x_i by $x_i + c$ and at the same time replace y_i by $y_i - b$ then the value of $x_i b + y_i c$ is unchanged.) However, one set of natural values for the x_i and y_i is given by the recursions

$$\begin{aligned} x_i &= x_{i-2} - q_i x_{i-1}, \\ y_i &= y_{i-2} - q_i y_{i-1}. \end{aligned}$$

Indeed, it is this same recursion,

$$r_i = r_{i-2} - q_i r_{i-1}$$

that generates the r_i. These x_i and y_i are displayed by EuAlgDem. What do you note about the signs of these numbers? About their absolute values? Can you prove that these patterns hold in general? What values are taken on by $x_i y_{i-1} - x_{i-1} y_i$?

11. The program GCDTab displays the greatest common divisors of pairs of integers. After invoking this program, use the arrow keys to move away from the origin. Each *gcd* displayed is calculated (by the Euclidean Algorithm, of course) and then immediately written to the screen. Admire how quickly this is accomplished. What value occurs most frequently? Enter $b = 3300$ and $c = 2200$ to move to a new location in the screen. Note that there are two columns near the middle of the screen that consist entirely of 1's. Use the $\uparrow$ and $\downarrow$ keys to examine more entries in these columns. Why do these columns contain so many 1's? Where in these columns will one find larger entries?

12. In EuAlgDem, the quotients are initially determined by rounding down,

$$q_i = [r_{i-2}/r_{i-1}],$$

but one can switch to rounding to the nearest integer by pressing `N`. For several pairs b, c compare the to calculations. How many steps are saved when rounding to the nearest integer? Is there much in common among the two sets of r_i? Try the pairs b, c that you found in **9.** with all $q_i = 1$ and all $q_i = 10$. What do you find?

13. For the programmer. Write a program in which b and c run independently from 1 to some number N. For each pair, evaluate (b, c). Count the number K of pairs for which $(b, c) = 1$. What is the proportion K/N^2? How close is it to $6/\pi^2$? How does the running time of this program depend on N? For any fixed $g > 0$, the density of pairs b, c, for which $(b, c) = g$ is asymptotically $6/(\pi g)^2$. You could write your program so as to track the incidence of other small values of the *gcd*.

14. For the hopelessly addicted programmer. Construct a routine that evaluates $j(b, c)$. Use this in a program that chooses pairs b, c of large integers ($\approx 10^{17}$) at random, and tabulates the value of $j(b, c)$. For 10,000 such pairs, say, how are the values of j distributed? What is their mean? Max? Min? Standard deviation? For the theory behind this, consult J. Dixon, *The number of steps in the Euclidean Algorithm,* J. Number Theory **2** (1970), 414–422. How close are your numerical values to the theoretical prediction?

15. For the truly ambitious. In addition to rounding to the nearest integer, the Euclidean Algorithm may be enhanced by removing powers of 2 whenever possible. Your machine knows b as a string of binary digits, so the power of 2 dividing b can be read as a block of trailing 0's. One may divide by 2 by right-shifting the binary expansion. This is much faster than long division (or at least it should be). Suppose that $2^j \| b$ and that $2^k \| c$. Put $b' = b/2^j$, $c' = c/2^k$, and set $m = \min(j, k)$. Then $(b, c) = 2^m(b', c')$. Now use the following identities, as appropriate:

i) $(b, c) = (c, b)$	(use this to ensure that $b \ge c > 0$),
ii) $(b, c) = (b - c, c)$	(use this when $b \ge c > 0$ and b and c are both odd),
iii) $(b, c) = (b/2, c)$	(if b is even and c is odd),
iv) $(b, 0) = b$	(if $b > 0$).

The point here is that if b and c are odd then $b-c$ is even, so that iii) becomes applicable after ii) has been applied. Division by 2 is accomplished by right-shifting the binary expansion. Thus the usual division, which is slow, is avoided. With gcd evaluated in this way, write a program that calls gcd repeatedly, and keeps track of the elapsed time. Compare these times with the times obtained similarly using FastGCD. Because of slow string manipulation, in Turbo Pascal, it may emerge that your new—and complicated—version of gcd is in fact slower. It may be necessary to resort to assembly language if a gain is to be realized.

LABORATORY 2

Factorization and Prime Numbers

Programs Used: FacTab, Factor, GetNextP

The program FacTab produces a table of least prime divisors of odd numbers, up to 10^9. The values are calculated by dividing small primes into the numbers in the desired range, until the only numbers for which a least prime divisor has not been found are prime. Let p be a given prime number. The least composite integer n such that p is the least prime factor of n is $n = p^2$. (In this connection, recall Problem 24. on p. 30 of NZM.) Thus if one is to prepare a table of least prime factors of integers in an interval $[a, b]$, then it is useful to have on hand a table of all primes $p \le b^{1/2}$. In the case of FacTab, the intervals considered are of the form $[10N, 10N + 200]$ with $N \le 10^8$. Since 31607 and 31627 are consecutive primes, and since

$$31607^2 < 10^9 + 200 < 31627^2,$$

it suffices to have a table of primes through the prime 31607. Such a table of primes may be constructed as follows: Consider a sequence a_1, a_2, ..., a_{31607} of 0's and 1's. Initially we take $a_1 = 0$, and $a_i = 1$ for all $i > 1$. We operate on this sequence so that eventually $a_i = 1$ if i is prime, $a_i = 0$ otherwise. Start with $p = 2$, and while $p \le 173$ perform the following operations: Put $j = p^2$. This is the least composite integer such that a_j is still 1. Put $a_j = 0$. Replace j by $j + p$, and set $a_j = 0$. Replace j by $j + p$. Continue in this manner until $j > 31607$. By examining the numbers $a_{p+1}, a_{p+2}, \ldots$, find the least integer q such that $q > p$ and $a_q = 1$. Then q is the least prime number greater than p. Replace p by q, and start over. This method of generating primes is known as the *Sieve of Eratosthenes*. It suffices to sieve only to $p = 173$, since 173 is the largest prime $\le \sqrt{31607}$. FacTab constructs a table of small primes in this way when the program is first loaded, with one modification: Since the even numbers are immediately eliminated, FacTab saves time and memory by applying the sieve only to the odd integers.

1. Use FacTab to view the least prime factors of the odd numbers in an interval. You will note that the least prime factor of numbers of the form $10k + 5$ display a certain pattern. Describe this pattern, and prove that it persists.

2. Can you find any other patterns similar to the one noted above?

3. For $5 \le k \le 20$, how many primes lie between e^k and $e^k + 100$? How do these numbers compare with $100/k$?

4. For several values of x and h (with h small compared with x), count the primes between x and $x + h$, and compare the result with $h/\log x$.

5. How many primes lie between 20831330 and 20831530 ? By using `PgUp` and `PgDn`, determine whether this is typical of similar intervals in this vicinity. For a report of a more extensive study of the gaps between primes, see D. Shanks, *On Maximal Gaps between Successive Primes, Math. Comp.* **18** (1964), 646–651.

6. For how many integers $n \le x$ is the least prime factor of n greater than 2? Greater than 3? Than 5? Than 7? How do these numbers increase with x? Formulate a conjecture concerning the asymptotic behavior. Can you prove your conjecture? (Theorem 8.8(e) and Theorem 8.29 of NZM are relevant here.)

7. A prime number p is called a *twin prime* if $p+2$ is also prime. Repeat Problem 3 above, but this time counting only twin primes. How do the counts compare with $100/k^2$? Do you conjecture that there are infinitely many twin primes, or do you conjecture that there are only finitely many?

The program Factor determines the canonical factorization of an integer n by trial division. Suppose that prime factors $< d$ have been found, and removed, leaving an integer m yet to be factored. If $m = 1$ then we are done. If $1 < m < d^2$ then m is prime, and we are done in this case also. Otherwise, we divide d into m. If $d|m$ then d is prime, and we repeatedly divide by d until d no longer divides the remaining number. Then we replace d by $d+1$ and repeat the process. To save time, after powers of 2 have been removed, only odd d are considered. Further savings can be obtained by noting that after removing powers of 2 and of 3 it suffices to consider d of the forms $d = 6k-1$, $d = 6k+1$. FacTab takes this a step further: After powers of 2, 3, and 5 have been removed, only those d of the eight forms $30k+1$, $30k+7$, $30k+11$, $30k+13$, $30k+17$, $30k+19$, $30k+23$, $30k+29$ are considered. Thus d is replaced by $d+30$ after only 8 trial divisions. This method is in principle slightly wasteful, because it would be enough to consider prime values of d, but in practice it seems to take longer to generate a table of primes. (Try it, if you like to write programs.) Instead of generating a table of primes, one could construct a permanent file listing primes, and then call the needed primes from that file, but this seems to take still longer.

8. Use FacTab to find the largest prime $< 10^k$ for $k = 1, 2, \ldots, 9$. Apply Factor to each of these nine primes, and note the time required to perform the calculations. Is the time roughly $c\sqrt{n}$? What values of c do you observe?

9. Apply Factor to each of the numbers $10^{18} - k$ for $k = 1, 2, \ldots, 9$. In some of these cases, you will tire of waiting for a complete resolution. To interrupt the program, simply press a key, and note how the program reports its partial results. (In laboratories 8 and 19 you will be introduced to programs that deal more quickly with numbers that resist treatment by Factor.)

10. Apply factor to 20 randomly-chosen numbers $\approx 10^9$. Make a record of these numbers, and note which ones are square-free. What proportion of them are square-free? How close is this proportion to $6/\pi^2 = 0.6079\ldots$? How many integers $n \le x$ are not divisible by 4? How many are not divisible by 9? How many by neither 4 nor 9? The proportion of $n \le x$ for which $4 \nmid n$ and $9 \nmid n$ tends to a limit as x tends to infinity. What is this limit?

Can you guess how this limit would change if we also require that $25 \nmid n$? (See Theorems 8.25 and 8.29 of NZM.)

The program GetNextP yields the least prime p greater than a given number a, provided that $a \le 10^9$. For a in this range, GetNextP uses the same sieving procedure as found in FacTab. For larger values of a, $10^9 < a \le 10^{18}$, the program GetNextP locates the least integer $q > a$ that is likely to be prime. That is, the interval (a, q) contains no prime number and q is “probably” prime (in the sense that it passes several strong pseudoprime tests; this is discussed on pp. 77, 78 of NZM, and also in Laboratory 7). In Laboratory 11 technique is introduced by means of which it is possible to prove that a number q is prime, much more efficiently than would be done by trial division.

11. Use GetNextP to find the least prime $p > x$, for several $x \approx 10^8$. How are the differences $p - x$ distributed? What is their mean? If you like to program, you could conduct a larger study, and a more detailed statistical analysis.

The asyptotic situation remains a matter of conjecture, but it is expected that as x tends to infinity, the mean lies between $(1-\epsilon)\log x$ and $(1+\epsilon)\log x$. Also, for any fixed $c > 0$, it is predicted that the proportion of integers $x \le X$ such that $p - x > c\log x$ tends to e^{-c} as X tends to infinity.

12. Write a program that deletes from a given sequence of integers those that are divisible by the square of a prime. In this way, count the number of square-free integers in various short intervals, and also the number of square-free integers not exceeding 10^4.

In Theorem 8.25 of NZM it is shown that the number $Q(x)$ of square-free integers not exceeding x is $cx + O(\sqrt{x})$ where $c = 6/\pi^2$. (The O-notation is defined on p. 365 of NZM.) It is conjectured that the error term here is actually $O(x^\theta)$ for any $\theta > 1/4$. Although such a strong upper bound for the magnitude of the error term has not yet been proved, it is known that the error term does achieve the order of $x^{1/4}$ infinitely often. Does the numerical evidence generated by your program support the stronger conjecture? Still less is known concerning the variation in the number of square-free numbers in short intervals.

$$\mathbf{P = NP}$$

(if $N=1$)

LABORATORY 3

Congruences

New Programs: CngArTab, Mult, MultDem1, MultDem2, MultDem3,
PowerTab, FctrlTab, PolySolv

1. The program CngArTab displays the addition and multiplication tables of congruence arithmetic. After entering an initial modulus m, you may switch between the two tables by pressing `s`. Reduced residue classes are displayed in white, to aid in distinguishing them from non-reduced residue classes, in yellow. In the multiplication table, which rows constitute a complete residue system (each residue once and only once)? Refer to Theorem 2.6 of NZM.

2. If two reduced residue classes are multiplied, is their product necessarily a reduced residue class? Experiment, and recall Theorem 1.8 of NZM.

3. When viewing the multiplication table, the display can be restricted to the reduced residue classes by pressing `r`. Try this with $m = 15$, for example. Do the numbers in a given row of the table constitute a system of reduced residues? Refer again to Theorem 2.6 of NZM. Try this also with $m = 91$, and note the location of the gaps in the reduced residues.

4. Take $m = 35$ in CngArTab. For which $a \pmod{35}$ does there exist an x such that $ax \equiv 1 \pmod{35}$? That is, in which rows of the multiplication table do you see a 1? Is there any row containing more than one 1? (Numbers in the first column don't count.) Refer to Theorem 2.9 of NZM.

Suppose that $0 \le a < m$ and $0 \le b < m$, and that we wish to find a number c in this same interval such that $c \equiv a + b \pmod{m}$. If $a + b < m$ then this is easily accomplished by taking $c = a + b$. The only other possibility is $m \le a + b < 2m$, in which case it suffices to take $c = a + b - m$. Thus it is easy to compute the sum of two residue classes. Multiplication may be approached similarly: We first form ab, and then apply the Division Algorithm, so that $ab = qm + r$ with $0 \le r < m$. Then $ab \equiv r \pmod{m}$, and we are done. However, a computational problem arises if m is large, because ab may be nearly as large as m^2. For example, our Turbo Pascal programs perform integer arithmetic accurately only up to 10^{18}. For $m < 10^9$ we proceed as above, but for $10^9 < m \le 10^{18}$ we have a challenge: Find r, $0 \le r < m$, so that $ab \equiv r \pmod{m}$, with using only numbers in the interval $[-10^{18}, 10^{18}]$. One approach to this is sketched in Problem 21 at the end of Section 2.4 of NZM. This procedure is displayed by the program MultDem1. It works pretty well if m is not too large (say $10^9 < m \le 10^{12}$), but for *really* large m (those close to the upper limit 10^{18}), this procedure is slow. An alternative method, described in MultDem2, is faster for $10^{12} < m \le 10^{18}$. In practice we choose one or the other of these methods, depending on the size of m. This is demonstrated in MultDem3. From the command line you can multiply residue classes by using the program Mult. Try typing `mult 2 3 5 [Enter]`. Alternatively, type `mult [Enter]`, and respond to the prompts.

5. Use the program PowerTab to investigate the following questions. For which values of $a \pmod m$ is the sequence $a^0,\ a^1,\ a^2,\ \ldots \pmod m$ eventually periodic? Purely periodic? Try all values of a for several moduli (say $m = 4, 5, 6, 7$), and note the results. Formulate conjectures concerning the general situation. How does the behavior for prime m differ from composite m? Is the value of (a, m) relevant? For now you can assume that PowerTab computes the powers of $a \pmod m$ sequentially. Actually, this program can skip forward to calculate $a^n \pmod m$ quickly, without determining the intervening powers. This involves an algorithm that will be discussed in Laboratory 7.

6. The number of reduced residue classes $\pmod m$ is called $\phi(m)$. (See pp. 50, 51 of NZM.) Determine the value of $\phi(91)$ by the following method: There are precisely 13 numbers a, $0 \le a < 91$ such that $7|a$. Similarly, there are precisely 7 numbers a, $0 \le a < 91$ for which $13|a$, and precisely 1 number a, $0 \le a < 91$ for which both $7|a$ and $13|a$. Hence $\phi(91) = 91 - 13 - 7 + 1 = 72$. By using CngArTab to view the multiplication table (mod 91) with only reduced residues displayed, you can confirm that this calculation is correct. More generally, if $n = p_1p_2$ where p_1 and p_2 are distinct primes, then $\phi(n) = n - n/p_1 - n/p_2 + 1 = n(1 - 1/p_1)(1 - 1/p_2)$. This approach can be extended to numbers with more prime factors, by means of the principle of Inclusion-Exclusion (see pp. 209, 210 of NZM). An alternative method of developing a formula for $\phi(m)$, based on the Chinese Remainder Theorem, is found on p. 69 of NZM. Use PowerTab to view the powers of b, reduced modulo 91. Note that $b^{72} \equiv 1 \pmod{91}$ whenever $(b, 91) = 1$, as predicted by Euler's Congruence (Theorem 2.8 of NZM). Is there a smaller exponent with this same property?

7. The program FctrlTab generates a table of the numbers $k! \pmod m$. Use FctrlTab to view the factorials modulo 345345. What is the least k such that $k! \equiv 0 \pmod{345345}$? Is it necessarily the case that $(k, 345345) > 1$? Use Factor to determine the factorizations of k and of 345345. Use FctrlTab to view the factorials $\pmod p$ for several prime numbers p. Is there any pattern exhibited by $(p-1)! \pmod p$? By $(p-2)! \pmod p$? By $(p-3)! \pmod p$? Formulate conjectures. Can you prove that each one of these conjectures implies the others? See Wilson's Theorem (Theorem 2.11 of NZM).

8. For each integer m let $k(m)$ denote the least positive integer k such that $k! \equiv 0 \pmod m$. Clearly $k(p) = p$ if p is prime. If m is composite then $k(m)$ is smaller. How much smaller? Is it usually small? Is it usually large? Does it oscillate a lot? Use FctrlTab to determine $k(m)$ for several values of m, and interpret your findings.

9. The program PolySolv allows you to define a polynomial $f(x)$, and then find the roots of the congruence $f(x) \equiv 0 \pmod m$. The program runs rather slowly when m is large, since $f(a)$ is evaluated $\pmod m$ for every a, $0 \le a < m$. Use PolySolv to find the roots of $x^2 \equiv 1 \pmod p$ for several small primes p, and note that the results conform to Lemma 2.10 of NZM.

LABORATORY 4

Sums of Two Squares

New Programs: SumsPwrs, WrngTab

1. Apply the program PolySolv to the polynomial $f(x) = x^2 + 1$. Take the modulus to be a prime number $\equiv 3 \pmod 4$, and note that the congruence has no solution, as proved in Theorem 2.12 of NZM. Take p to be a prime $\equiv 1 \pmod 4$. How many solutions are there? How are they related to each other? Try several different primes $\equiv 1 \pmod 4$. Is the number of solutions always the same? Form a conjecture. (This conjecture can be proved by applying Corollary 2.27, or by taking $d = 4$ in Corollary 2.30 of NZM.) How many solutions are there when $p = 2$? Let $N(m)$ denote the number of solutions of the congruence $x^2 + 1 \equiv 0 \pmod m$. Use PolySolv to determine the value of $N(2^j)$, $N(3^j)$, and $N(5^j)$ for several small values of j. Does a pattern emerge?

2. The program SumsPwrs will find all representations of n as a sum of s k-th powers, by exhaustive searching. If s is large compared with k then the time required for this increases very rapidly with n. Type `sumspwrs 1105 2 2 [Enter]`, or type `sumspwrs [Enter]` and respond to the prompts. Let $R(n)$ denote the number of representations of n as a sum of two squares. That is, the number of ordered pairs (x, y) of integers such that $x^2 + y^2 = n$. (Note that x and/or y may be negative.) Thus from SumsPwrs we find that $R(1105) = 32$. A representation $n = x^2 + y^2$ is called *proper* if $(x, y) = 1$. Let $r(n)$ denote the number of proper representations of n. Using Factor, PolySolv, and SumsPwrs, determine entries for the table below:

n	Factorization	$R(n)$	$r(n)$	$N(n)$
5				
13				
17				
65				
91				
1105				

The functions $N(n)$ and $r(n)$ are closely related. Can you spot the connection? (These functions are discussed in §3.6 of NZM, as an application of the theory of binary quadratic forms.) Theorem 2.15 of NZM asserts that one can determine whether n is a sum of two squares by inspecting the canonical factorization of n. Is your data above consistent with this description?

3. Choose a prime number $p \equiv 1 \pmod 4$, and set $x = (\frac{p-1}{2})!$. Use FctrlTab to find the value of $x \pmod p$. Then use Mult to confirm that $x^2 \equiv -1 \pmod p$. This is computationally slow when p is large, because of the large number of multiplications required to evaluate the factorial. A much faster method for finding solutions of this congruence is found in Problem 2. of Laboratory 14 (and at the top of p. 111 of NZM). Once the congruence has been solved, the representation of p as a sum of two squares can be found quickly, either by using the theory of binary quadratic forms (see Example 3 in §3.6 of NZM, and also the discussion prior to Problem 3. in Laboratory 16), or by using continued fractions (as described in Problem 6. at the end of §7.3 of NZM).

4. Let $f(x) = x^2+1$. If $p \equiv 1 \pmod 4$ then f has exactly one root x for which $0 < x < p/2$. Let p run over a collection of such primes. How are the numbers $2x/p$ distributed in the interval $(0,1)$? It has long been conjectured that these quantities approach uniform distribution as p runs over all primes $\equiv 1 \pmod 4$, $p \le x$, with x tending to infinity. One could write a program to test how rapidly the distribution approaches uniformity. This conjecture was finally proved in 1994 (see W. Duke, J. B. Friedlander, and H. Iwaniec, *Equidistribution of roots of a quadratic congruence to prime moduli,* Annals of Math. (2) **141** (1995), 423–441). The proof is quite sophisticated, as it depends on the spectral theory of modular forms.

5. Let $x = (\frac{p-1}{2})!$, as in **2.** above. What is $x \pmod p$, if $p \equiv 3 \pmod 4$? Use FctrlTab to investigate, and recall Problem 18 on p. 57 of NZM. Of the two possibilities that occur here, it seems not to be known that both occur for infinitely many $p \equiv 3 \pmod 4$, although one might conjecture that each occurs asymptotically 1/2 the time. One could write a program to generate statistical data. D. H. Lehmer showed that the two possibilities are connected to whether $h(-p) \equiv 1 \pmod 4$ or $\equiv 3 \pmod 4$, where $h(-p)$ is a class number of binary quadratic forms, as defined in Problem 13 on p. 163 of NZM.

In view of the definition of $R(n)$, it is clear that the sum $\sum_{n \le x} R(n)$ is equal to the number of lattice points (x, y) in the disk of radius $\sqrt{x}$ centered at the origin. As the number N of lattice points within a convex body $\mathcal{C}$ differs from the area A of that body by an amount that is at most proportional to the perimeter P of that body. That is, $N = A + O(P)$. Applying this to the disk, we deduce that

$$\sum_{n \le x} R(n) = \pi x + O(\sqrt{x}). \tag{1}$$

Let $B(x)$ denote the number of integers $n \le x$ that can be expressed as a sum of two squares. One might think that the relation above suggests that $B(x) \sim cx$ as x tends to infinity (i.e., the sums of two squares form a set of positive asymptotic density). However, Landau proved that

$$B(x) \sim \frac{bx}{\sqrt{\log x}} \tag{2}$$

as x tends to infinity. Here b is a certain positive constant. The apparent discrepancy between these results is reconciled by recognizing that $R(n)$ is usually 0, but if $R(n) > 0$ then $R(n)$ is likely to be large. The tools required to prove (2) are similar to those used in the analytic proof of the Prime Number Theorem: Dirichlet series, Euler products, contour integration, etc. For an exposition of this, see W. J. LeVeque, *Topics in Number Theory*, vol. II, Addison-Wesley, Reading, 1956, pp. 257–263.

It is known that the limiting approximation in (2) is approached only slowly. A more accurate approximation to $B(x)$ could be constructed by introducing a second term on the right hand side of (2), of the form $b_1 x/(\log x)^{3/2}$. Here b_1 is some appropriate constant. Still greater accuracy would be achieved by introducing a term $b_2 x/(\log x)^{5/2}$, and so on. This is discussed by D. Shanks, *The second-order term in the asymptotic expansion of* $B(x)$, Math. Comp. **85** (1964), 75–86. It turns out that the constant b in (2) is

$$b = \left(2 \prod_{q \equiv 3(4)} \left(1 - \frac{1}{q^2}\right)\right)^{-1/2} = 0.764223654\ldots$$

where the product is taken over all prime numbers $q \equiv 3 \pmod 4$.

LABORATORY 5

Solutions of Congruences & Binomial Coefficients

New Program: PascalsT

The program PolySolv allows you to specify a polynomial f with integral coefficients, and a modulus m, and then it evaluates $f(a) \pmod{m}$ for each a, $0 \le a < m$. On the screen it displays the residue classes a for which $f(a) \equiv 0 \pmod{m}$, up to the first 100 of them. If there are more than 100 such a then only the first 100 are displayed, but the program still reports the total number $N_f(m)$ of roots. Since the running time of this program is proportional to m, the program will restrict you to $m < 10^6$.

1. Use PolySolv to find all roots of $7x \equiv 1 \pmod{91}$; all solutions of $7x \equiv 35 \pmod{91}$; all solutions of $2x \equiv 1 \pmod{101}$. Note conformity with Theorem 2.17 of NZM.

In the next three problems, you are asked to gather data concerning the number of roots of a polynomial $f(x) \equiv 0 \pmod{p}$, for various f and p, and then to formulate a conjecture. Do not be disturbed if your numerical evidence is too meager to be compelling. Each polynomial f has a discriminant, denoted $D(f)$, and defined on p. 487 of NZM. Prime factors of the discriminant are apt to be exceptional, and may not obey the general pattern.

2. Let $f(x) = x^3 + x + 1$, with discriminant $D(f) = -31$. Using PolySolv, for each prime number $p < 100$ determine the value of $N_f(p)$. What is the biggest value attained? What values are attained, and with what frequencies? What is the average of the values calculated? Formulate conjectures regarding the general situation.

3. Let $g(x) = x^3 + x^2 - 2x - 1$, with discriminant $D(g) = 49$. Using PolySolv, for each prime number $p < 100$ determine the value of $N_g(p)$. What is the biggest value attained? What values are attained, and with what frequencies? What is the average of the values calculated? Formulate a conjecture regarding the general situation.

4. Let $h(x) = x^2 + x + 1$, with discriminant $D(h) = -3$. Using PolySolv, for each prime number $p < 100$ determine the value of $N_h(p)$. What is the biggest value attained? What values are attained, and with what frequencies? What is the average of the values calculated? Formulate a conjecture regarding the general situation.

The situation touched on in Problems 2–4 above is quite complicated. Suppose that $f(x)$ is a polynomial of degree d with integral coefficients. Then $0 \le N_f(p) \le d$; see Corollary 2.27 in NZM. The three polynomials considered above are irreducible (over the field $\mathbf{Q}$ of rational numbers). For such polynomials, additional patterns emerge in the statistics of the $N_f(p)$. For each k, $0 \le k \le d$, the primes p for which $N_f(p) = k$ have a certain relative density d_k. That is, the limit

$$d_k = \lim_{x\to\infty} \frac{1}{\pi(x)} \sum_{\substack{p\le x \\ N_f(p)=k}} 1$$

exists. These densities are determined by the Chebotarev Density Theorem in terms of the Galois group of f. Thus the densities depend on the particular polynomial, although only finitely many configurations can arise. In the case of the polynomial of Problem 2, the Galois group is S_3, and the densities are $d_0 = 1/3$, $d_1 = 1/2$, $d_2 = 0$, $d_3 = 1/6$. In Problem 3, the Galois group is C_3, and the densities are $d_0 = 2/3$, $d_1 = d_2 = 0$, $d_3 = 1/3$. (For this polynomial, $N_f(p) = 1$ if and only if $p = 7$.) In Problem 4 the Galois group is C_2, and the densities are $d_0 = 1/2$, $d_1 = 0$, $d_2 = 1/2$, but the situation is more elementary, since by quadratic reciprocity we find that $N_f(p) = 2$ if $p \equiv 1 \pmod 3$, and $N_f(p) = 0$ if $p \equiv 2 \pmod 3$. The densities then follow by the prime number theorem for arithmetic progressions.

Concerning the densities d_k, it is obvious that $\sum_{k=1}^{d} d_k = 1$, and it is easy to show that $d_{d-1} = 0$. Not so obviously, the d_k also satisfy the relation $\sum_{k=1}^{d} kd_k = 1$. That is,

$$\lim_{x\to\infty} \frac{1}{\pi(x)} \sum_{p\le x} N_f(p) = 1 \tag{1}$$

for any irreducible polynomial with integral coefficients. This is a consequence of the prime ideal theorem (which is a natural extension of the prime number theorem to algebraic number fields). For a more detailed account of how the densities d_k are calculated, see H. Heilbronn, *Zeta-functions and L-functions*, Algebraic Number Theory (Brighton, 1965), Thompson, Washington, 1967, pp. 204–230, especially pp. 227–229.

5. For any two polynomials $f(x)$ and $g(x)$, one can define their *resultant*, $R(f, g)$. We skip the definition and fundamental theorems concerning this quantity, and mention just three useful properties: (i) If f and g have integral coefficients, then $R(f, g)$ is an integer. (ii) $R(f, g) = 0$ if and only if f and g have a common factor (i.e., a common polynomial divisor of degree > 0). (iii) There exist polynomials $u(x)$ and $v(x)$ with integral coefficients such that

$$f(x)u(x) + g(x)v(x) = R(f, g).$$

Suppose that f and g have a common root (mod p). That is, there is an a (mod p) such that both $f(a) \equiv 0 \pmod p$ and $g(a) \equiv 0 \pmod p$. On setting $x = a$ in the identity above, we see that the left hand side is divisible by p, and hence that $p | R(f, g)$. Thus if $p \nmid R(f, g)$ then f and g have no common root, and it follows that $N_{fg}(p) = N_f(p) + N_g(p)$. Let f be as in Problem 2, and g as in Problem 3, so that $f(x)g(x) = x^6 + x^5 - x^4 + x^3 - x^2 - 3x - 1$. It can be shown that $R(f, g) = 13$ in this case. By applying PolySolv, confirm that f and g have a common root when $p = 13$. Without performing any additional calculation, list the roots of $f(x)g(x)$ (mod 13). Apply PolySolv to fg, to confirm your guess.

6. Apply PolySolv with $f(x) = x^{1732} - 1$, $m = 1733$. Having determined the number of roots of f, can you deduce that m is prime? Is this a time-effective method of proving primality? Apply PolySolv with $f(x) = x^{1738} - 1$, $m = 1739$. After determining the number of roots of f, can you deduce that m is composite? (Recall Euler's Congruence, Theorem 2.8 of NZM.) Is this a time-effective method of proving compositeness?

7. Let $p = 101$, say, and consider f of the form $f(x) = x^3 + ax^2 + bx + c$. For various randomly-selected triples a, b, c use PolySolv to determine the value of $N_f(101)$. Formulate a conjecture regarding the average number of solutions of a polynomial congruence modulo a prime p, when p is fixed and the polynomial runs over all monic polynomials of some given degree. (A polynomial is *monic* if its leading coefficient is 1.) Can you prove your conjecture?

8. Let f be defined as in Problem 3 above. Suppose that p is a prime such that $N_f(p) = 2$, and that q is a prime such that $N_f(q) = 3$. Use PolySolv to determine the value of $N_f(pq)$. Try some further examples of this kind. Formulate a conjecture concerning the relationship between $N_f(m)$, $N_f(n)$, and $N_f(mn)$ when $(m, n) = 1$. (This conjecture is established as Theorem 2.20 in NZM, as an application of the Chinese Remainder Theorem.)

9. Suppose that $p \nmid x$. Explain why $x^{(p-1)/2} \equiv \pm 1 \pmod{p}$. For how many x does the $+$ sign occur? Take $f(x) = x^{(p-1)/2} - 1$ in PolySolv. Try this for several values of p. Formulate a conjecture. (This conjecture can be derived as an application of the more general Theorem 2.37 of NZM.)

10. The program PascalsT displays the entries of Pascal's Triangle (i.e., binomial coefficients), reduced (mod m). Start with $m = 2$. The pattern created by rows 0–3 is repeated twice in rows 4–7, with an inverted triangle of 0's between. Does this generalize? How would you express this in terms of equations?

11. For $0 \le n \le 15$, count the number of odd entries in the nth row of Pascal's triangle. (Take $m = 2$ in PascalsT.) The totals that arise in this way form a special class of integers. Describe.

12. When n is written in binary, the number of 1's in the expansion is called the *binary weight* of n, and is denoted $w(n)$. That is, if $n = 2^{i_1} + 2^{i_2} + \cdots + 2^{i_k}$ with $0 \le i_1 < i_2 < \cdots i_k$ then $w(n) = k$. Compute $w(n)$ for $0 \le n \le 15$. Note the relation between these values, and the totals computed in the preceding problem. Form a conjecture. (Problem 16 at the end of §2.2 of NZM is relevant here.)

13. Let p be a prime number. What is the least n such that $p \mid \binom{n}{k}$ for all k in the range $0 < k < n$? (Take $m = p$ in PascalsT, and look for 0's.) What is the second such n? The third? (Problem 14 at the end of §2.2 of NZM is relevant here.)

14. For what k, $0 \le k \le 15$, is it true that $3 \nmid \binom{15}{k}$? For what k, $0 \le k \le 15$, is it true that $5 \nmid \binom{15}{k}$? Does this suggest something?

15. Let p be a prime number. Describe all the patterns that you can find in the sequence of residues $\binom{n}{p} \pmod{p^2}$.

LABORATORY 6

Linear Congruences

& The Chinese Remainder Theorem

New Programs: LinCon, LnCnDem, IntAPTab,
CRT, CRTDem, Phi, ResComp

The program LinCon applies the extended Euclidean algorithm to find the complete solution set of the linear congruence $ax \equiv b \pmod{m}$. You can type `lincon a b m [Enter]`, or simply type `lincon [Enter]` and follow the prompts. Try it both ways, now. Note that the conclusions reached are in conformity with Theorem 2.17 of NZM.

1. The computational procedure followed by LinCon is sketched at the end of §2.2 of NZM, and is described in greater detail on the first three pages of Chapter 5. The steps involved are displayed by the program LnCnDem. Type `lncndem 17 1 101 [Enter]`, and follow the explanations given. Alternatively, type `lncndem [Enter]`, and provide the input values as prompted. Apply LnCnDem with $a = 7$, $b = 13$, $m = 91$. Also with $a = 5$, $b = 155$, $m = 345$.

2. You now have two methods for finding solutions of linear congruences. You can use either (i) LinCon or (ii) PolySolv. Try both methods on the congruence $7x \equiv 1 \pmod{1234}$. Which method takes longer to run? Estimate the running time for the two methods as a function of m. Which method is asymptotically faster? (Ignore the time it takes to supply the input information to the programs.)

3. Let m and n be given, and put $g = (m, n)$. The intersection of an arithmetic progression $a \pmod{m}$ with an arithmetic progression $b \pmod{n}$ is an arithmetic progression $(\mathrm{mod}\ [m, n])$ if $a \equiv b \pmod{g}$, and is otherwise empty. (Recall Problem 20 at the end of §2.3 of NZM.) The program IntAPTab presents these intersections in a manner reminiscent of the table on p. 68 of NZM. Rows are indexed by residues $a \pmod{m}$, and columns by $b \pmod{n}$. Type `intaptab [Enter]`, and then take $m = 5$, $n = 8$. Note that in the body of the table, each of the numbers $0, \ldots, 39$ occurs exactly once. That is, the simultaneous congruences $x \equiv a \pmod{5}$, $x \equiv b \pmod{8}$ are equivalent to the single congruence $x \equiv c \pmod{40}$, for some suitable value of c. The more general assertion that this is true whenever $(m, n) = 1$ is known as the Chinese Remainder Theorem (Theorem 2.18 of NZM). Take $m = 101$, $n = 103$ in IntAPTab, and take a stroll around the table. There are now so many entries that it is no easy to see, by inspection, that each number $0, \ldots, 10402$ occurs exactly once in the body of the table. Now take $m = 102$, $n = 104$ in IntAPTab. What proportion of the entries are blank? Why? This phenomenon becomes more pronounced when (m, n) is large. Try taking $m = 25$, $n = 35$.

The program CRT (meaning "Chinese Remainder Theorem") determines the intersection of two given arithmetic progressions. For example, the numbers x such that both $x \equiv 3 \pmod 4$ and $x \equiv 2 \pmod 5$ are precisely the numbers for which $x \equiv 7 \pmod{20}$. Type `crt 3 4 2 5 [Enter]`, and watch the results. On the other hand, there are no x for which both $x \equiv 1 \pmod{12}$ and $x \equiv 19 \pmod{28}$. To see why this is so, type `crt 1 12 19 28 [enter]`.

4. The program CRT uses LinCon to find the intersection of two arithmetic progressions, in the manner of the *Second Solution* to Example 3, on p. 67 of NZM. The program CRTDem demonstrates how this is done. Type `crtdem 3 4 2 5 [Enter]`, and watch the response. Try also `crtdem 1 12 19 28 [Enter]`.

5. By repeated use of CRT, find a number x such that $0 < x < 10^9$ and none of x, $x+1$, $\ldots$, $x+6$ is squarefree. Thus we have a gap of length at least 8 between consecutive squarefree numbers. (Hint: What if $x \equiv 0 \pmod 4$, $x \equiv -1 \pmod 9$, $x \equiv -2 \pmod{25}$, $x \equiv -3 \pmod{49}$, $x \equiv -4 \pmod{121}$, $x \equiv -5 \pmod{169}$, $x \equiv -6 \pmod{289}$.) Apply the program Factor to each of the numbers x, $x+1$, $\ldots$, $x+6$ to verify your results. Are $x-1$ and $x+7$ both squarefree? (This construction can be extended—recall Problem 18 at the end of §2.3 of NZM.)

6. Take $m = 15$, $n = 13$ in IntAPTab. Note that an entry in the body of the table is printed in White if and only both the column and row labels of that entry are printed in White. More generally, if $(m,n) = 1$, $x \equiv a \pmod m$, and $x \equiv b \pmod n$, then $(c, mn) = 1$ if and only if $(a,m) = 1$ and $(b,n) = 1$. (This is argued in the proof of Theorem 2.19, by using Theorems 1.8 and 2.4.) Hence the number of reduced residues $(\text{mod } mn)$ is the number of reduced residues $(\text{mod } m)$ times the number of reduced residues $(\text{mod } n)$. That is, $\phi(mn) = \phi(m)\phi(n)$ whenever $(m,n) = 1$. Since it is easy to see that $\phi(p^\alpha) = p^\alpha - p^{\alpha-1} = p^\alpha(1 - 1/p)$, we deduce that

$$\phi(n) = \prod_{p^\alpha \| n} \left(p^\alpha - p^{\alpha-1}\right) = n \prod_{p \mid n} \left(1 - \frac{1}{p}\right).$$

Thus we can calculate $\phi(n)$ easily, once the factorization of n has been determined. The program Phi proceeds in this way: First the argument is factored, and then the above formula is used. Try typing `phi 42`. You can confirm this answer by taking $m = 42$ in CngArTab, and viewing the multiplication table with only the reduced residues displayed.

7. By using PolySolv, find two distinct residue classes x_1 and x_2 modulo 31 so that $x^3 + x + 1 \equiv 0 \pmod{31}$. Similarly, find three distinct residue classes y_1, y_2, y_3 modulo 47 so that $x^3 + x + 1 \equiv 0 \pmod{47}$. By using CRT, find six residue classes u modulo $31 \cdot 47 = 1457$ so that $u \equiv x_i \pmod{31}$ and $u \equiv y_j \pmod{47}$, $i = 1, 2$, $j = 1, 2, 3$. Apply PolySolv with $f(x) = x^3 + x + 1$, $m = 1457$. Interpret your findings. Note the conformity of this with Theorem 2.20 in NZM.

8. Recall that the only solutions of $x^2 \equiv 1 \pmod p$ are $x \equiv \pm 1 \pmod p$. (See Lemma 2.10 of NZM.) Given that $4757 = 67 \cdot 71$, use the program CRT to find four roots of the congruence $x^2 \equiv 1 \pmod{4757}$. Verify your results by using PolySolv. When m is composite you now have two methods for locating all the roots of a polynomial congruence $f(x) \equiv 0 \pmod{m}$. You can (i) apply PolySolv directly to the modulus m, or (ii) factor m into primepowers, apply PolySolv to each of these primepowers, and then use CRT to combine these solutions to construct the solutions modulo m. Estimate the running time of these two approaches. Which one is faster for a typical large composite number? (Ignore the time it would take to input the arguments.)

LABORATORY 7

Powering Algorithms & Primality Testing

New Programs: PwrDem1a, DwrDem1b, PwrDem2, Power, SPsPDem, SPsP

The number $a^k \pmod{m}$ can be determined by $k-1$ multiplications of residue classes, but this is slow if k is large. There is a much faster way: The values of a, a^2, a^4, a^8, ..., a^{2^i}, ..., $\pmod{m}$ can be determined, by repeated squaring, in only i multiplications. The binary expansion of k provides a representation of k as a sum of powers of 2, and hence a^k is a product of an appropriate collection of the numbers a^{2^i}. For example, $13 = 2^3 + 2^2 + 2^0$, and hence $a^{13} = a^{2^3} \cdot a^{2^2} \cdot a^{2^0}$. The exact number of multiplications required by this method varies irregularly with k, but it never exceeds $2\log_2 k$. The binary expansion of k can be built from the bottom up, as demonstrated in programs PwrDem1a, PwrDem1b, or from the top down, as demonstrated in PwrDem2. The former of these two methods is discussed on pages 76, 77 of §2.4 of NZM.

1. Apply the programs PwrDem1a, PwrDem1b, DwrDem2 to several values of a, m until the process is clear to you. Apply PwrDem1b and PwrDem2 to the same k. How do the number of multiplications compare?

2. If k has binary expansion $k = 2^{i_1} + 2^{i_2} + \cdots + 2^{i_r}$ with $i_1 < i_2 < \cdots < i_r$, then our powering algorithm requires $i_r + r - 1$ multiplications to calculate a^k. In particular, it takes 6 multiplications to calculate a^{15}. Show that a^{15} can be obtained with only 5 multiplications.

The program Power evaluates $a^k \pmod{m}$. You may type `power a k m [Enter]`, or else simply type `power [Enter]`, and respond to the prompts. Try it both ways, now.

3. Use the program Power to evaluate $2^{m-1} \pmod{m}$ where $m = (10^{17}-1)/9 = 11111111111111111$. Assuming Fermat's congruence, (Theorem 2.7 of NZM), this provides a quick (but indirect) proof that 11111111111111111 is composite. Apply the program Factor to 11111111111111111, and note how long it runs. With large numbers m (of hundreds of digits), it is often the case that a quick proof that m is composite can be given, even though we know of no way to obtain the factors of m within a reasonable amount of time.

4. Is 91 prime? Evaluate $2^{90} \pmod{91}$. Is 341 prime? Evaluate $2^{340} \pmod{341}$. Now evaluate $3^{340} \pmod{341}$. What do you conclude?

5. We have no quick method to find $k! \pmod{m}$ akin to our quick method for calculating powers. There are a few special cases (such as $(p-1)! \pmod{p}$), but in general the fastest method known involves simply performing the $k-1$ multiplications. If a quick method *could* be found, then it would have important applications (to factoring, for example). Suppose that you are in possession of a quick method for calculating $\binom{2k}{k} \pmod{m}$.

Explain how this could be used to provide a quick method for calculating $k! \pmod{m}$. Suppose you have a quick method for calculating $k! \pmod{m}$. Explain how this could be used to provide a quick method for factoring m.

If $0 < a < m$ and $a^{m-1} \not\equiv 1 \pmod{m}$ then m is composite. Since it is easy to calculate powers modulo m, this provides a quick proof that m is composite—when it works. Unfortunately, the converse is false, but the counterexamples seem to be rare, so we call m a *probable prime base* a if m is odd and $a^{m-1} \equiv 1 \pmod{m}$. If m is a probable prime base a but is nevertheless composite, then we call m a *pseudoprime base* a, or, briefly, m is a PSP(a). If m is found to be a probable prime base 2, then we might try base 3, and so on, but there exist composite m that are probable primes to every base a for which $(a, m) = 1$. To see how this might happen, suppose that m is a composite squarefree number with the peculiar property that $(p-1)|(m-1)$ for every prime number p dividing m. (The least such m is 561.) Suppose that $(a, m) = 1$. If $p|m$ then $(a, p) = 1$, and hence $a^{p-1} \equiv 1 \pmod{p}$. Since $(p-1)|(m-1)$, it follows that $a^{m-1} \equiv 1 \pmod{p}$. Since this congruence holds for every p dividing m, it holds modulo the product of all the primes dividing m. But we have assumed that m is squarefree; hence $a^{m-1} \equiv 1 \pmod{m}$. An odd composite number such that $a^{m-1} \equiv 1 \pmod{m}$ whenever $(a, m) = 1$ is called an *absolute pseudoprime*, or *Carmichael number*. The least Carmichael number is 561; indeed, it can be shown that if m is a Carmichael number then m is of the form we considered: m is squarefree and $(p-1)|(m-1)$ whenever $p|m$. (This is called *Korselt's criterion*; see Problems 25–27 at the end of §2.8 of NZM.) It is not hard to show that there exist infinitely many pseudoprimes to any given base (see Problem 19 at the end of §2.4 of NZM), and it is easy to construct numerical examples of Carmichael numbers and to give arguments that suggest that Carmichael numbers form a fairly rich subset of the integers (by methods akin to the construction of Problem 20 of §2.8 of NZM). In particular, P. Erdős (*On pseudoprimes and Carmichael numbers*, Publ. Math. Debrecen **4** (1956), 201–206) formulated a heuristic argument that suggests that the number $C(x)$ of Carmichael numbers not exceeding x is larger than $x^{1-\epsilon}$ for all sufficiently large x. Although Erdős's conjecture is presumably true, it seems that the ϵ tends to 0 slowly, since numerical studies have revealed that $C(10^{10}) = 1547$, and that $C(10^{15}) = 105212$. Nevertheless, it was finally proved that there do indeed exist infinitely many Carmichael numbers. W. R. Alford, A. Granville, and C. Pomerance, (*There are infinitely many Carmichael numbers*, Ann. of Math. (2) **139** (1994), 703–722) showed that $C(x) > x^{2/7}$ for all sufficiently large x.

Since the pseudoprime test fails to establish the compositeness of some composite numbers, we consider a slightly more elaborate test, which, however, involves no more calculation than before. If m is odd, we repeatedly divide 2 into $m-1$, until we obtain a representation $m-1 = 2^r \cdot d$ with d odd. Suppose that $a \not\equiv 0 \pmod{m}$. Compute $a^d \pmod{m}$. Next, repeatedly square, forming the sequence a^{2d}, a^{4d}, ..., $a^{(m-1)/2} \pmod{m}$. Let x denote this last residue class computed. If $x^2 \not\equiv 1 \pmod{m}$ then $a^{m-1} \not\equiv 1 \pmod{m}$, and hence m is composite, by Fermat's congruence. Suppose now that $x^2 \equiv 1 \pmod{m}$. If $x \not\equiv \pm 1 \pmod{m}$ then m is composite by virtue of Lemma 2.10 of NZM. More generally, if in the sequence of powers computed we find an entry $x \not\equiv \pm 1 \pmod{m}$ followed by an entry 1, then $x^2 \equiv 1 \pmod{m}$, and hence m is composite. This test is more stringent than the previous one; if it is inconclusive then we call m a *strong probable prime base* a. If in addition m is composite then we call m a *strong pseudoprime base* a, or m is an SPSP(a). In practice, we abandon the repeated squaring if a value $\equiv \pm 1 \pmod{m}$ is encountered, since the conclusion is already clear. The exact sequence of steps performed is exhibited on p. 78 of NZM. It is known that if m is composite then there are at least $m/4$ bases a such that the compositeness of m is demonstrated by applying this strong pseudoprime test base a. Thus if m survives this test for several values of a, we can be reasonably confident that m is prime—such an m might be called an "industrial grade prime".

6. By means of lengthy calculation (see C. Pomerance, J. L. Selfridge, and S. S. Wagstaff Jr., *The pseudoprimes to* $25 \cdot 10^9$, Math. Comp. **35** (1980), 1003–1026), it has been found that there are only 13 odd integers $m < 25 \cdot 10^9$ that are SPSP(a) for $a = 2$, $a = 3$, and $a = 5$. Of these, only one, namely $m = 3215031751$ is also a SPSP(7). Apply the strong pseudoprime test to this m with bases $a = 2$, 3, 5, 7, and 11. For example, try typing `spspdem 3215031751 2 [Enter]`, or simply type `spspdem [Enter]` and follow the prompts.

7. By appropriate use of the program Power, show that 4369 and 4371 are both probable primes base 2. Are either of these numbers strong probable primes base 2? Are either of these numbers prime? (Use the program SPsP to answer this question, not Factor.) Are either of these numbers Carmichael numbers?

8. Factor 561, verify that 561 is squarefree, and that $(p-1)|560$ for every prime p dividing 561. Hence deduce that 561 is a Carmichael number.

9. If m is a PSP(a) but not a SPSP(a) then the strong pseudoprime test locates a number x such that $x \not\equiv \pm 1 \pmod{m}$, but $x^2 \equiv 1 \pmod{m}$. In such a situation not only is it established that m is composite, but also a proper divisor of m can be exhibited, namely $(x-1, m)$. Apply the program SPsPDem to $m = 561$ with $a = 2$.

10. What does the program SPsP do if you enter m on the command line, but omit a? Type `spsp 91 [Enter]`.

11. Numerical evidence suggests that most pseudoprimes are squarefree. To explain this, show that if m is a PSP(a), and if p is a prime such that $p^2|m$, then

$$a^{p-1} \equiv 1 \pmod{p^2}. \tag{1}$$

(Hint: $a^m \equiv a \pmod{m}$, and hence $a^{p-1} \equiv (a^m)^{p-1} \equiv a^{m(p-1)} \pmod{m}$. But $\phi(p^2)$ divides $m(p-1)$.) Conversely, show that if p is a prime such that (1) holds then p^2 is a PSP(a). Only a few primes have been found for which $2^{p-1} \equiv 1 \pmod{p^2}$, although it is believed that infinitely many exist. The least such prime is 1093. Use the program Factor to verify that 1093 is prime, and the program Power to verify that $2^{1092} \equiv 1 \pmod{1093^2}$. Is 1093 a SPSP(2)?

For an extensive account of primality testing see H. C. Williams, *Primality testing on a computer*, Ars Comb. **5** (1978), 127–185.

LABORATORY 8

Factoring Strategies

New Programs: RhoDem, Rho, P–1Dem, P–1

We know that trial division yields a rigorous proof of the factorization of n in at most $O(\sqrt{n})$ steps. This is slow when n is large, so we now consider methods that are faster for large n. Our object here is not to present state-of-the-art factoring, but only to drive home the point that it is possible to construct factoring strategies that are *much* faster than trial division.

1. Trial division takes $\approx \sqrt{n}$ steps if n is prime or if n is the product of two primes, $n = p_1 p_2$ with $p_1 \approx p_2 \approx \sqrt{n}$. However these are the worst cases, and trial division is much quicker for many numbers. To see why this is so, suppose that n is composite and that $p_1 < p_2 < \ldots < p_k$ are the distinct primes dividing n. Explain why only $O(\log n) + O(p_{k-1}) + O(\sqrt{p_k})$ trial divisions are required to factor n. Trial division is unlikely to yield the complete factorization of a large n in a reasonable amount of time, but nevertheless one should always try divisors through 10^5 or so, when asked to factor a number n of unknown multiplicative structure.

2. Although alternative factoring strategies can be traced as far back as Fermat and Gauss, we begin with a simple method of comparatively recent origin, namely Pollard's Rho Method, proposed in 1975 by J. M. Pollard. (Why it should be called "Rho" is explained on p. 81 of NZM.) Suppose that a prime number p has been chosen. Let $u_0 = 1$, and for $i > 0$ let the numbers u_i be determined by the relations $u_i \equiv u_{i-1}^2 + c \pmod{p}$, $0 \le u_i < p$. Here c is some constant. We usually start with $c = 1$, but other values of c are sometimes handy, as will become clear later. The sequence u_i may have a non-periodic initial segment, but once a value is repeated (as must eventually happen), the sequence becomes periodic. The program RhoDem will assist you in determining when this first repetition occurs. Type `rhodem [Enter]`, and in response to the query "Use cycle-detecting algorithm?" respond by typing `n`. In response to the prompts enter $p = 89$, and set $c = 1$. In the sequence of u_i displayed, you will see that $u_2 = u_{16} = 2$, but that $u_1 \ne u_{15}$. Thus the first repeat is at u_{16}, and the period of the repetitions is 14. In general, let $r(p)$ be the least index i such that the value u_i repeats a value found previously, and let $l(p)$ denote the least period of the repetitions. Thus $r(89) = 16$ and $l(89) = 14$. Repeat this calculation for the prime $p = 29$, and thus determine the values of $r(29)$ and $l(29)$.

By the pigeon-hole principle we see that $r(p) \le p + 1$, but the "Birthday Paradox" leads us to expect that $r(p) \approx \sqrt{p}$ for most primes, and for most choices of c. (See Lemma 2.21 in NZM, and the discussion following.)

3. In the examples above it is easy to spot the first repetition visually, but this task becomes rapidly more difficult when p is a little larger. Repeat the steps above with

$p = 3463$. Touch a key to scroll down through the table, and as you go, note the following four values:

i	$u(i)$
130	2185
131	2212
$\vdots$	$\vdots$
147	1278
148	2212

Thus $r(3463) = 147$ and $l(3463) = 17$. Since it is quite tedious (and time-consuming!) to compare each value with all the previous ones, we need a quick way to spot repetitions. This is provided by the following Cycle Detection Algorithm: Watch for an index i at which $u_i = u_{2i}$. Let $s(p)$ denote the least such i. The advantages of this approach are that only one comparison need be made, and that only the values u_i and u_{2i}. Thus we have no need to store the values of the u_i. If $u_i \neq u_{2i}$ then we use the recurrence once to compute u_{i+1}, and twice more to compute u_{2i+2}. The old values u_i, u_{2i} are discarded, and we continue with the two new values. The disadvantage of this approach is that it is slightly inefficient, in the sense that the recurrence must be used $3s(p)$ times, which is somewhat larger than $r(p)$, which would be optimal. Using RhoDem, complete the following table (the first row of which has been thoughtfully provided). Use RhoDem with no cycle-detecting first, to determine the values of $r(p)$ and $l(p)$. By inspecting the values u_i, try to determine the value of $s(p)$. Check your work by applying RhoDem a second time with cycle-detecting.

p	r	s	l
37	6	5	1
41			
43			
47			
53			

4. When we apply the Rho method to factor a number m, we compute the sequence u_i modulo m. Suppose that $p|m$. We can't construct the sequence u_i (mod p), because the prime p is unknown. However, the u_i computed are congruent (mod p) to those we would have obtained if we had worked (mod p) (recall Theorem 2.1(5) of NZM). Hence $u_i \equiv u_{2i}$ (mod p) when $i = s(p)$. Let $s_0(m)$ denote the least index i such that $(u_i - u_{2i}, m) > 1$. Then

$$s_0(m) = \min_{p|m} s(p).$$

Moreover, for this index i we have $(u_i - u_{2i}, m) < m$ unless $s(p) = s_0(m)$ for all $p|m$. Take $m = p_1 p_2$ where the p_i are selected from the above table in such a way that $s(p_1) \neq s(p_2)$. Apply RhoDem to this m, with cycle-detecting, and note the point at which a divisor is found. (To avoid all the prompts, you can enter m on the command line: Type `rhodem m [Enter]`.) Repeat this, with the p_i selected so that $s(p_1) = s(p_2)$. Note that the gcd jumps from 1 to m, but that RhoDem does not give up. What does RhoDem do, instead? Finally, choose two p_i so that $p_1 < p_2$ but $r(p_1) > r(p_2)$, and note that the prime factor found by RhoDem is not the least prime factor of m.

WARNING: The Pollard Rho Method should only be applied to numbers that are already known to be composite (as the result of a strong pseudoprime test, for example). If it were applied to a large prime number p, it would run endlessly, switching to ever larger values of c.

5. Apply RhoDem to $m = 11111111111111111 = (10^{17} - 1)/9$. What is the least s for which $(u_{2s} - u_s, m) > 1$? The program Rho will attempt to factor a given number m by means of the Pollard rho algorithm. Type `rho 11111111111111111 [Enter]`, or simply `rho [Enter]`, and answer the prompts. Is this much faster than using the program Factor? What do you expect the running time of Rho to be, on average, as a function of the size of the least prime factor of m?

6. What inequalities can be established between the three quantities $r(p)$, $s(p)$, $l(p)$? Explore.

7. In general you should avoid taking $c = 0$ or $c = -2$ in the rho method. Experiment with these values of c, using RhoDem, and try to explain why these values of c are bad. (Hint: For $c = -2$, note that if $x \cdot \overline{x} \equiv 1 \pmod{p}$ and $u \equiv x - \overline{x} \pmod{p}$ then $u^2 - 2 \equiv x^2 - \overline{x}^2 \pmod{p}$.)

8. For the programmer. Richard Brent has observed that the cycle-detecting algorithm can be made about 24% more efficient, as follows: Suppose that you have calculated u_n and u_{2n}, and that you have tried $(u_j - u_k, m)$ for pairs (j, k) with the difference $j - k$ running from 1 to n. Starting from u_{2n}, apply the iteration $n + 1$ times, to evaluate u_{3n+1}. Next compute $(u_{3n+1} - u_{2n}, m)$, $(u_{3n+2} - u_{2n}, m)$, $\ldots$, $(u_{4n} - u_{2n}, m)$. Here the differences between the subscripts range from $2n + 1$ to $4n$. If you start this with $n = 1$, then n runs through powers of 2. To speed things up further, do not calculate the *gcd* separately for each term indicated above. Instead, form a product of these numbers, keeping track of the number of factors in the product. When the number of factors reaches 8, compute the *gcd* of the product with m. The product, like everything else, is computed modulo m.

9. We now turn to a second method proposed by Pollard, the "$p-1$ Method." Suppose that m is a number to be factored, that $p|m$, and that $(p - 1)|k!$, so that $a^{k!} \equiv 1 \pmod{p}$ whenever $(a, m) = 1$, which is to say that $p|(a^{k!} - 1, m)$. Thus we use the powering algorithm to calculate a number x, $0 \leq x < m$, so that $x \equiv a^{k!} \pmod{m}$, and then we use the Euclidean algorithm to evaluate $(x-1,m)$ in the hope that this will disclose a proper factor of m. If this gcd is still 1 then we try a larger k; if it is m then we switch to a different value of a. This method is rather erratic: It is remarkably fast for some numbers, but for other numbers it is no faster than trial division. Apply the program P–1Dem to several numbers, and note how the calculation proceeds. The program P–1 is ridiculously fast when applied to $m = 99999997425160993$. Use P–1 to break m into factors, use Factor to verify that the factors are indeed prime, say $m = p \cdot p'$. Apply Factor to $p-1$ and to $p'-1$, and thus demonstrate why P–1 is so quick for this number. At the opposite extreme, the program P–1 will take an uncomfortably long time to factor the comparatively small number $m = 9904156957$. Find the prime factors p of this m, and also the factorization of $p-1$ for each such p, to explain why the method is so slow in this case. Finally, apply the program P–1 to our old favorite, $m = 11111111111111111 = (10^{17}-1)/9$.

10. For the programmer. The Pollard $p-1$ method, as explained above, is slightly inefficient because the power of 2 dividing $k!$ is much larger than is likely to be needed. Try using d_k instead of $k!$, where d_k denotes the least common multiple of the integers 1, 2, …, k. Show that $d_k = a_1 \cdot a_2 \cdot \ldots \cdot a_k$ where $a_n = p$ if n is a power of p, $a_n = 1$ otherwise. Note that $d_k = d_{k-1}$ unless k is a primepower. Thus it is necessary to compute $(a^{d_k}-1,m)$ only when k is a primepower. Does this lead to a more efficient method?

It is notable that we have no proof that the Pollard Rho Method is efficient, although we believe that on average it will yield a proper divisor of n in $O(\sqrt{p})$ steps, where p is the smallest prime factor of n. Although the $p-1$ method is erratic, the idea behind the method is used in other methods, notably the Elliptic Curve Method (ECM), devised by Lenstra in 1987. (see §5.8 of NZM.) In 1982, Carl Pomerance invented the Quadratic Sieve method (QS) of factoring, which has been further developed to become the Multiple Polynomial Quadratic Sieve (MPQS). These methods have largely usurped an older method, CFRAC, based on properties of continued fractions. A new method, the Number Field Sieve, (NFS) is currently being developed, and has already achieved some notable successes.

For more information concerning factoring, consult the following sources.

D. M. Bressoud, *Factorization and primality testing*, Springer-Verlag, New York, 1989.

D. V. Chudnovsky and G. V. Chudnovsky, *Sequences of numbers generated by addition in formal groups and new primality and factorization tests,* Adv. Appl. Math. **7** (1986), 385–434.

D. Coppersmith, *Modifications to the number field sieve*, J. Cryptology **6** (1993), 169–180.i

J. D. Dixon, *Factorization and primality tests*, Amer. Math. Monthly. **91** (1984), 333–352.

R. K. Guy, *How to factor a number,* Proc. Fifth Conf. Numerical Math., Utilitas, Winnipeg, 1975, pp. 49–89.

P. L. Montgomery, *Speeding the Pollard and elliptic methods of factorization,* Math. Comp. **48** (1987), 243–264.

C. Pomerance, *Lecture Notes on Primality Testing and Factoring*, MAA Notes 4, Math. Assoc. of America, Washington, 1984.

H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Birkhäuser, Boston, 1985, 464 pp.

H. C. Williams, *Factoring on a computer,* Math. Intell. **6** (1984), 29–36.

M. C. Wunderlich, *Computational methods for factoring large integers,* Abacus **5** (1988), 19–33.

LABORATORY 9

RSA Public Key Cryptography

New Programs: RSA, RSAPars

For centuries, one of the hazards of cryptography was that a copy of your code book might fall into enemy hands, so that all your encrypted transmissions could then be intercepted and decoded. Worse yet, you might have no way of knowing whether one of your communications stations had been taken over by the enemy: The enemy might be masquerading as one of your own troops. All this changed in 1976 when Whitfield Diffie and Martin Hellman proposed a form of encryption that should be easy to perform but would be difficult to break, even if the encryption procedure were made public. The scheme works like this: Suppose that Bob wants to receive a message from Alice without observers being able to read the message. Bob chooses a very large integer m, say $m \approx 10^{200}$, and defines a permutation π of the numbers $1, 2, \ldots, m$. The algorithm for computing π is made public, and in particular is given to Alice. The characters of Alice's message can be associated with digits in a standard way, and the digits can be broken into blocks of length not exceeding 200, so that Alice's message is equivalent to one or more integers t, each one in the interval $[1, m]$. Thus t is the *plaintext*. Alice computes $c = \pi(t)$; this is the *cryptotext*; it is also an integer in the interval $[1, m]$. Alice sends c to Bob. Since an observer may also gain access to c, for the security of the communication it is essential that there be no quick algorithm for computing the inverse permutation π^{-1}, since $t = \pi^{-1}(c)$. However, Bob possesses some secret information concerning π that allows him to compute π^{-1} quickly, and hence read Alice's message. A permutation with the peculiar property that π is easy to compute while π^{-1} is difficult (i.e., would take centuries on the fastest computers) is called a *trap door function*.

The success of the Diffie-Hellman scheme depends on being able to find trap door functions. This was achieved in 1977 by Ron Rivest, Adi Shamir, and Len Adleman. Their *RSA method* depends on the number theory that we have been investigating: Bob secretly chooses two 100-digit primes p_1, p_2, and sets $m = p_1 p_2$. Bob also chooses a large positive integer k with the property that $(k, \phi(m)) = 1$. Among the reduced residue classes (mod m), the map $\pi(x) \equiv x^k \pmod{m}$ is a permutation. Bob makes m and k public, and Alice sends him $c \equiv t^k \pmod{m}$. (Recall that we have a powering algorithm that makes this easy.) Since Bob knows how to factor m, Bob knows the value of $\phi(m)$. Hence Bob can find a positive integer k' such that $kk' \equiv 1 \pmod{\phi(m)}$. (We use the extended Euclidean algorithm to solve linear congruences, so this is also fast.) We now show that the map $x \mapsto x^{k'} \pmod{m}$ is the inverse permutation that we need. To this end, choose q so that $kk' = 1 + q\phi(m)$, and recall Euler's congruence, which asserts that if $(x, m) = 1$ then $x^{\phi(m)} \equiv 1 \pmod{m}$. Hence

$$(x^k)^{k'} = x^{kk'} = x^{1+q\phi(m)} = x(x^{\phi(m)})^q \equiv x(1)^q = x \pmod{m}.$$

Thus the decryption process for Bob is similar to Alice's encryption, but with the parameter k replaced by k'. Note that only Bob can calculate k'. Even Alice can't read her own message, once she's encoded it!

In the RSA method, the permutation being employed constitutes a trap door function only to the extent that large composite integers are difficult to factor. In the present state of knowledge one can factor a number of size 10^{150}, but there is no guarantee that there does not exist some factoring method yet to be discovered by which even much larger numbers could be factored quickly. One could imagine that such a method might be taken as a State Secret. Indeed, when Rivest, Shamir, and Adleman published their work in 1978, the Director of the National Security Agency (General Odum) gave serious consideration to going to Congress asking for legislation that would make all research in number theory "born classified" as is the case with atomic research. He was dissuaded from this, but in any case any lingering impression that number theory is the purest of the pure, totally devoid of practical application, has been forever dispelled.

Rivest, Shamir and Adleman patented their method, and formed the company RSA Data Systems to market RSA–based products. To emphasize the security of their system, they offered a prize of $100 for the first decryption of the message

$$\begin{aligned} c = {} & 9686961375462206147714092225435588290575999112457431987469512093081 62\\ & 98225145708356931476622883989628013391990551829945157815154, \end{aligned}$$

which was encrypted using the 129-digit modulus

$$\begin{aligned} m = {} & 1143816257578888676692357799761466120102182967212423625625618429357\\ & 06935245733897830597123563958705058989075147599290026879543541 \end{aligned}$$

and the public exponent

$$k = 9007.$$

The estimate at that time was that it would take 40 trillion years to factor this m. However, on 29 April, 1994, Derek Atkins, Michael Graff, Arjen Lenstra, and Paul Leyland announced that $m = p_1 p_2$ where

$$\begin{aligned} p_1 &= 3490529510847650949147849619903898133417764638493387843990820577,\\ p_2 &= 32769132993266709549961988190834461413177642967992942539798288533. \end{aligned}$$

This enabled them to determine the secret exponent,

$$\begin{aligned} k' = {} & 106698614368578024442868771328920154780709906633937862801226224496631\\ & 063125911774470873340168597462306553968544513277109053606095, \end{aligned}$$

and consequently the plaintext

$$\begin{aligned} t = {} & 2008050013010709030023151804190001180500191721050113091908001519190 9\\ & 0618010705. \end{aligned}$$

After conversion back to alphabetic characters, this reads

THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE

Lenstra (at Bellcore) and his team used the double large prime variation of the multiple polynomial quadratic sieve factoring method. The calculation took more than 5000 mips years, and was executed over a period of 8 months on over 600 different computers that were made available for the purpose by volunteers in more than 20 countries, on all continents except Antarctica. The final stage of the computation took 45 hours on a 16K MasPar MP-1 massively parallel computer. The relatively short time that it took to factor RSA-129 is partly due to increased speed and power of computer hardware, but it is mainly due to progress that has been made in developing faster factoring algorithms.

The RSA-129 modulus was factored by combining the latest factoring algorithms with enormous computing resources. With larger moduli, the RSA method is considered to be secure, and is widely used. In the spring of 1996, Rivest (a mathematician at MIT) sold his interest in the company to a venture capital firm for \$50,000,000. So being a mathematician is not only fun, but occasionally also profitable!

The program RSA automates the arithmetic operations that arise when executing the RSA algorithm. To use this program you will need a public modulus, a public exponent, and a secret exponent. Since typing such data from the keyboard is tedious and prone to error, it is best to keep the public parameters in a computer file. The program RSAPars will assist you in this. It is best to choose your private exponent k' first, since then you can take it to be something memorable, such as your parents' home phone number. Do not use your Social Security Number or something really sensitive, since you will be using a modulus $m < 10^{18}$, and hence any energetic person could use m and k to reconstruct k'. Since $(k', \phi(m)) = 1$, and since $\phi(m)$ is even when $m > 2$, your private exponent k' must be odd. Once you have chosen k', the program assists you in choosing a public modulus m, by selecting the prime factors of m. There is no need to enter a prime exactly. Simply enter an approximate size x, and the computer will find the least prime $p > x$ such that $(p-1, k') = 1$. The program will not allow you to use the same prime twice, since it is advantageous for m to be squarefree (see question **4.** below). Once you have entered two or more primes, and you are satisfied with the value attained, you can indicate that you are done, and the computer will find the complementary public exponent k. You may now save m and k to a file, so that others can use these values to send you a message. Choose a filename that identifies you, and add a tag number (Alice might take `alice1`), so that if you ever want to establish a second set of RSA parameters you will have a way of distinguishing them. The program takes '.pub' as the default extension of the file. After exiting, Alice can view the file that has been created by typing `type alice1.pub` <`Return`> at the DOS prompt.

Once Bob has the file `alice1.pub`, he can send her an encrypted message by using the RSA program. This program has no word-processing capabilities, so Bob must first compose a text file. This he can do by typing `edit bob2alic.txt` at the DOS prompt. After saving his message to disk, he invokes RSA, where he can `L`oad the `P`lain text file, and set the `V`ariables by `R`eading them from `alice1.pub`. Each letter of the text needs to be converted to a two-digit `C`ode; the codes are then concatenated to form a sequence of `R`esidues. Each residue is taken to the power k modulo m to form a new sequence of residues. This is the `E`ncryption. This new sequence of residues can be `S`aved to a file, whose name by default is `bob2alice.rsa`. In turn, Alice can `L`oad the `C`ipher text and her Variables, including her secret Decrypting exponent k'. She can then Decrypt to recover the original plaintext sequence of residues. These can be separated to form Codes, and finally Text, which can be Saved. When dealing with encrypted files it is sometimes handy to have some indication as to what is in the file. When the RSA program reads a file,

CODE	CHAR	ASCII	CODE	CHAR	ASCII	CODE	CHAR	ASCII	CODE	CHAR	ASCII
00		32	25	9	57	50	R	82	75	k	107
01	!	33	26	:	58	51	S	83	76	l	108
02	"	34	27	;	59	52	T	84	77	m	109
03	#	35	28	<	60	53	U	85	78	n	110
04	$	36	29	=	61	54	V	86	79	o	111
05	%	37	30	>	62	55	W	87	80	p	112
06	&	38	31	?	63	56	X	88	81	q	113
07	'	39	32	@	64	57	Y	89	82	r	114
08	(	40	33	A	65	58	Z	90	83	s	115
09	)	41	34	B	66	59	[	91	84	t	116
10	*	42	35	C	67	60	\	92	85	u	117
11	+	43	36	D	68	61	]	93	86	v	118
12	,	44	37	E	69	62	^	94	87	w	119
13	-	45	38	F	70	63	_	95	88	x	120
14	.	46	39	G	71	64	`	96	89	y	121
15	/	47	40	H	72	65	a	97	90	z	122
16	0	48	41	I	73	66	b	98	91	{	123
17	1	49	42	J	74	67	c	99	92	|	124
18	2	50	43	K	75	68	d	100	93	}	125
19	3	51	44	L	76	69	e	101	94	~	126
20	4	52	45	M	77	70	f	102	95	EoL	13
21	5	53	46	N	78	71	g	103	96	—	
22	6	54	47	O	79	72	h	104	97	—	
23	7	55	48	P	80	73	i	105	98	—	
24	8	56	49	Q	81	74	j	106	99	—	

Table 1. Character to Code Correspondence

it looks for lines that begin with the symbol '%'. Such lines are passed to the destination without change. Hence Bob might put at the top of his message the line

```
% This is a message from Bob to Alice.
```

The RSA program also places the encryption history in such comment lines, so that the recipient will know what parameters have been used.

Before proceeding further we consider how to convert characters into numbers. This can be done in many ways. For example, we could let A correspond to 1, B to 2, …, and Z to 26. Alternatively, computers store alphanumeric characters by their ASCII codes. (ASCII is an abbreviation for American Standard Code for Information Interchange.) The first of these methods makes no provision for punctuation, numerals, or lower case letters. The second provides all printable characters, but is inefficient because each character requires three digits (in base 10). The characters that can be typed in the standard keyboard have ASCII codes between 32 (to denote a space ' ') and 126 (for '~'). As a compromise between the two systems described above, we subtract 32 from each ASCII code to obtain a 2–digit number. These numbers run from 00 to 94. In order to preserve the line breaks in a file we need an end of line marker; we assign the code 95 for this purpose. Thus we have the codes opposite.

1. Suppose that Bob took the (ridiculously small) modulus $m = 91$, and proposed the public exponent $k = 17$. Suppose that Alice sent him the encrypted message $c = 51$. Use the programs Factor, Phi, LinCon, and Power appropriately to recover her plaintext t.

2. The proof above that $x^{kk'} \equiv x \pmod{m}$ assumed that $(x, m) = 1$. If $m = p_1p_2$ where p_1 and p_2 are distinct primes, what is the probability that $(x, m) > 1$ when x is randomly chosen?

3. Show that if m is squarefree then the restriction to $(x, m) = 1$ is unnecessary. That is, if m is squarefree and $kk' \equiv 1 \pmod{\phi(m)}$, then $x^{kk'} \equiv x \pmod{m}$ for all integers x.

4. The encrypted message

355456249 475197422 636832086 601788838

was created using the modulus $m = 670726081$ and the public exponent $k = 663599161$. The program RSA will assist in decrypting this, but first you must determine the value of $\phi(m)$, and then solve the congruence $kk' \equiv 1 \pmod{\phi(m)}$. (Use Factor and/or Phi, and then LinCon.) Next use a text editor to create a file, say `prob4.rsa`, that consists of the line displayed above. Then type `rsa [Return]`, Load the `Cipher` text `prob4.rsa`, and enter the `Variables`. Type `Esc` to return to the main menu, and then `Decrypt`. The resulting residues can be separated into 2–digit `Codes`, which may be read as `Text`. What was the message?

5. Although Bob is the only person who can decrypt a message encrypted with his parameters, he has no way of knowing that the message actually came from Alice, since anyone can use his parameters. To overcome this defect, suppose that Bob has a trap door function π_B and that Alice also has a trap door function π_A. Suppose that Alice sends $c = \pi_B(\pi_A^{-1}(t))$ to Bob. What should Bob do, to decrypt this? Can anyone else decrypt it? Can Bob now be sure that the message came from Alice?

6. In the preceding problem there was a tacit assumption that the trap door functions π_A and π_B act on the same set of numbers. Suppose now that π_A permutes the residue classes modulo m_A, and that π_B permutes the residue classes modulo m_B. If $m_A \le m_B$ then we may still proceed as above, since we may consider $\pi_A^{-1}(t)$ as lying in the interval $[0, m_A)$, which defines a unique residue class m_B. How would you modify the above procedure if $m_A > m_B$?

7. In formulating their challenge, Rivest, Shamir and Adleman did not use the system in Table 1 to convert from alphanumeric characters to a residue class t. By comparing t with the stated text can you infer the system that they used instead?

8. Let $m = 854937209155735099$, and suppose that you are given the information that m has at most two prime factors, and that $\phi(m) = 854937207303842520$. Can you find the primes?

9. Suppose that $m = 1247 = 29 \cdot 43$, so that $\phi(m) = 1176$. In order that $x^{kk'} \equiv x \pmod{m}$ for all x, it is sufficient that $kk' \equiv 1 \pmod{\phi(m)}$, but is it necessary? Suppose that $k = 5$. How many k' are there, $0 \le k' < \phi(m)$, such that $x^{kk'} \equiv x \pmod{m}$ for all x? What if you take instead $k = 11$? Why is the number of admissible k' so large? To achieve security, the acceptable k' should be very rare. How should the prime factors of m be chosen, to achieve this?

An RSA code can be broken if the value of $\phi(m)$ is known. One way to determine $\phi(m)$ is to factor m, but one could conceive that possibly $\phi(m)$ might be found more quickly, without having to factor m. However, we argue now that this is not the case: If the value of $\phi(m)$ is known then the factorization of m can be recovered with little work. Hence any quick method of evaluating $\phi(m)$ would yield a quick method of factorization.

If $\phi(m) = m - 1$ then m is prime, and we are done. (Of course such an m would not be used for RSA encryption.) Hence we may suppose that $\phi(m) < m - 1$, i.e. that m is composite.

If m is a product of two distinct primes, say $m = pq$, and if $\phi(m)$ is known, then the primes can easily be found. To see this, note that $p + q = m - \phi(m) + 1$. Since the sum $p + q$ and the product pq are both known, the values of $(p - q)^2 = (p + q)^2 - 4pq$ can be determined. By taking square roots one obtains $|p-q|$, and the primes are $(p+q \pm |p-q|)/2$. (This is, after all, how one solves for the roots of a quadratic polynomial.) This procedure may be attempted whenever m and $\phi(m)$ are both known. If it fails then we know that m is not the product of two distinct primes. To eliminate the (unlikely) possibility that $m = p^2$, we may compute $\sqrt{m}$.

Now we come to the heart of the matter: m is the product of 3 or more primes. Suppose that a number c is known with the property that $a^c \equiv 1 \pmod{m}$ whenever $(a, m) = 1$. For example, $\phi(m)$ is such a number c. It is enough to find a proper divisor of m, since the method may then be applied to the divisors, repeatedly, until all the factors are prime. The number c may be hard to factor, but at least we can determine the power of 2 in it, say $c = 2^j \cdot k$ with k odd. Choose a number a at random, $0 < a < m$. If 1 ¡ $(a, m) < m$ then we have found a proper divisor of m. If $(a, m) = 1$ then put $b = a^k$. The value of $b \pmod{m}$ is quickly found by the powering algorithm. By repeated squaring, compute b^2, b^4, ..., $b^{2^j} \pmod{m}$. Actually, there is no need to compute the last term, since this number is $\equiv 1 \pmod{m}$. In the sequence of powers of b computed, suppose that the first 1 is preceded by a number other than -1. Then we have an x such that $x \not\equiv \pm 1 \pmod{m}$, but $x^2 \equiv 1 \pmod{m}$, and hence $(x-1, m)$ is a proper divisor of m. This procedure is not guaranteed to work for every a, but should work for a large proportion of a's modulo m, provided that m is divisible by two or more odd primes. (We may assume that m is odd.) In the one remaining case, $m = p^k$ with $k > 1$, the prime p can be found quickly, since $p = m/(m, \phi(m))$.

10. Let $m = 308557669718497477$. This number is the product of two distinct primes, and $\phi(m) = 308557668607386336$. Find the primes.

10. Let $m = 144145168546451$. Given that $\phi(m) = 144136398922632$, show that m is not a prime, and is not a product of two primes. (I.e. verify that $\phi(m) < m - 1$, that m is not a perfect square, and that $(m - \phi(m) + 1)^2 - 4m$ is not a perfect square.) Use the procedure described above to find the prime factorization of m. How many different bases a do you need to consider?

For more information on public key cryptography, consult the following sources.

W. Diffie, *The first ten years of public-key cryptography,* Proc. IEEE **76** (1988), 560–577.

W. Diffie and M. E. Hellman, *New directions in cryptography,* IEEE Trans. Informat. Theory, **IT 22** (1976), 644–654.

M. Gardner, *A new kind of cipher that would take millions of years to break,* Scientific American (1977), 120–124.

K. S. McCurley, *A key distribution system equivalent to factoring,* J. Cryptology **1** (1988), 95–105.

M. Rabin, *Digitized signatures and public key functions as intractable as factorization*, Laboratory for Computer Science, Massachusetts Institute of Technology, MIT/LCS/TR-212, 1979.

R. L. Rivest, *RSA chips (past/present/future),* Eurocrypt '84, 159–165.

R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems,* Commun. ACM **21** (1978), 120–126.

H. C. Williams, *A modification on the RSA public-key encryption,* IEEE Trans. Informat. Theory **IT 26** (1980), 726–729.

H. C. Williams, *Some public-key crypto-functions as intractable as factorization,* Cryptologia **9** (1985), 223–237.

H. C. Williams, *An M^3 public-key encryption scheme,* Advances in Cryptology—CRYPTO 85, Springer-Verlag, 1986, pp. 358–368.

Number theorists are never past their prime:

2,	**3,**	**5,**	**7,**	**11,**	**13,**	**17,**	**19,**	**23,**
29,	**31,**	**37,**	**41,**	**43,**	**47,**	**53,**	**59,**	**61,**
67,	**71,**	**73,**	**79,**	**83,**	**89,**	**97,**	**101,**	**103,**
107,	**109,**	**113,**	**127,**	**131,**	**137,**	**139,**	**149,**	**151,**
157,	**163,**	**167,**	**173,**	**179,**	**181,**	**191,**	**193,**	**. . .**

LABORATORY 10

Hensel's Lemma

New Program: Hensel

Hensel's Lemma (as discussed in §2.6 of NZM), can be formulated as follows: Let $f(x)$ be a polynomial with integral coefficients, let p be a prime, and suppose that $f(a) \equiv 0 \pmod{p^j}$ for some $j \geq 1$.
Case 1. $f'(a) \not\equiv 0 \pmod{p}$. (The "non-singular" case.) There is a unique $c \pmod{p}$ such that $f(a + cp^j) \equiv 0 \pmod{p^{j+1}}$. This c is the root of the linear congruence

$$f'(a)c \equiv -f(a)/p^j \pmod{p}.$$

Case 2. $f'(a) \equiv 0 \pmod{p}$. (The "singular" case.) If $f(a) \equiv 0 \pmod{p^{j+1}}$ then $f(a+cp^j) \equiv 0 \pmod{p^{j+1}}$ for all $c \pmod{p}$. If $f(a) \not\equiv 0 \pmod{p^{j+1}}$ then $f(a+cp^j) \not\equiv 0 \pmod{p^{j+1}}$ for all $c \pmod{p}$.

1. Let $f(x) = x^2 + 1$. Note that $f(9) \equiv 0 \pmod{41}$, and that $f'(9) = 18 \not\equiv 0 \pmod{41}$. Since $f(9)/41 = 2$, it follows that to lift this root we must take c so that $18c \equiv -2 \pmod{41}$. Type `lincon 18 -2 41 [Enter]` to determine this c, and thus find a root of $f(x) \equiv 0 \pmod{41^2}$. Confirm your work by applying PolySolv to $f(x)$, first with $m = 41$, and then with $m = 41^2 = 1681$.

2. Let $f(x) = x^3 + x + 1$, as in Problem 2 of Laboratory 5. From Theorem A.5 on p. 488 of NZM we know that all roots of $f(x) \pmod{p}$ are non-singular, unless p divides the discriminant of f, denoted $D(f)$. In the present case, $D(f) = -31$. Apply PolySolv to $f(x)$, and take $m = 31$. Note the two roots. Now apply PolySolv to $f'(x) = 3x^2 + 1$. Thus discover that one of the roots of $f \pmod{31}$ is singular, and that the other one is non-singular. From Case 2 of Hensel's Lemma we know that the singular root either lifts to 31 roots $\pmod{31^2}$, or else does not lift. Apply PolySolv to $f(x)$ with $m = 31^2 = 961$, to determine which.

3. The mundane chore of applying LinCon to lift roots to higher powers of p is automated by the program Hensel. Type `hensel [Enter]`, and then take $f(x) = x^2 + 1$ by typing `1 [Enter] 2 [Enter] 1 [Enter] 0 [Enter]`. Take $p = 5$. Us the uparrow ($\uparrow$) key to view solutions $\pmod{5^j}$ that lie above the root $x \equiv 2 \pmod{5}$. Use the rightarrow ($\rightarrow$) key to view the other solution (mod 5), and those that lie above it. Note that in both cases, the sequence $c(j)$ of coefficients seems to exhibit no simple pattern.

4. The polynomial $f(x) = x^2 + 1$, when considered (mod 2), has a singular root $x \equiv 1 \pmod{2}$. Does this lift to a solution (mod 4)? By invoking the program Hensel, one may see that the answer is "No," because the uparrow key ($\uparrow$) is inactive. Now apply Hensel to $g(x) = x^2 + 3$. (That is, type `d 1 [Enter] 2 [Enter] 3 [Enter] 0 [Enter]`.) Note that $f(x)$ and $g(x)$ are the same (mod 2), but different (mod 4). The uparrow key can now be used to lift the solution $x \equiv 1 \pmod 2$ to $x \equiv 1 \pmod 4$. The rightarrow key can be used to give its companion, $x \equiv 3 \pmod 4$. Note that the two roots 1, 3 (mod 4) both lie above the single root 1 (mod 2). You may use the left- and rightarrow keys to switch between the two roots (mod 4), but in both cases the uparrow key is inactive, so neither of these roots lifts to give a root (mod 8). Finally, take $h(x) = x^2 + 7$. Note that $g(x)$ and $h(x)$ are the same (mod 4), but different (mod 8). Apply Hensel to $h(x)$, and note that the root 1 (mod 4) lifts to two roots, 1, 5 (mod 8). Also, note that the root 3 (mod 4) lifts to two roots, 3, 7 (mod 8). Use the arrow keys to explore the tree of solutions (mod 2^j), and sketch it through (mod 2^7), say. Note that two long strands are forming. How far must these strands be extended before one can be sure that they continue indefinitely? (Hint: Apply Theorem 2.24 of NZM. Theorem A.5 on p. 488 is also relevant.)

5. Use Hensel to explore the tree of solutions of $x^2 + x + 223 \pmod{3^j}$, through $j = 7$. Sketch your findings. Thus verify and extend Table 1 on p. 90 of NZM.

6. Apply Hensel to $f(x) = 5x + 3$. Note that the sequence of $c(j)$ seems to be periodic. For each prime $p < 20$, note the apparent period. We know that the base 10 expansion of a real number x is periodic if and only if x is rational, and that the least period of the base 10 expansion of a/q is the order of 10 (mod q), provided that $(10a, q) = 1$. (This is Problem 30 at the end of §2.8 of NZM.) Is there an analogue at work here? Explore.

7. Apply Hensel to $f(x) = x^4 - 10x^3 + 35x^2 - 50x + 24 \pmod{3^j}$, and report your findings. Note that you can switch between viewing singular and non-singular roots.

8. Apply Hensel to $f(x) = x^5 - 15x^4 + 85x^3 - 225x^2 + 274x - 120 \pmod{3^j}$, and report your findings.

9. Use Hensel to study $f(x) = x^k - 1 \pmod{p^j}$ for various combinations of k and p. For what combinations do you encounter singular roots? (Note: The number of roots (mod p) is $(k, p-1)$, according to Theorem 2.37 of NZM.)

LABORATORY 11

Power Residues & Primitive Roots

New Programs: OrderDem, Order, PrimRoot

The least positive integer h such that $a^h \equiv 1 \pmod{m}$ is called the *order of a modulo m*. (This is Definition 2.6 on p. 97 of NZM.) The order of a modulo m exists and is finite if $(a, m) = 1$; otherwise it is undefined.

1. Use PowerTab to determine the order of a for each reduced residue class a (mod 11). What orders occur? How many times do they occur? What is the least common multiple of these orders? Repeat this with 11 replaced by some other prime number. Formulate conjectures regarding the situation for a general prime modulus. Compare your findings with Lemma 2.35 and Theorem 2.36 of NZM.

2. Suppose that a has order h modulo m. How is h related to other numbers k such that $a^k \equiv 1 \pmod{m}$? Use PowerTab to investigate, for both prime and composite moduli, and then formulate a conjecture. Compare your conjecture with Lemma 2.31 of NZM. Euler's congruence asserts that $a^{\phi(m)} \equiv 1 \pmod{m}$ if $(a, m) = 1$. What does this imply concerning the relation between the order of a and $\phi(m)$? (See Corollary 2.32 of NZM.)

3. Suppose that a has order h modulo m. What is the order of a^k modulo m? Experiment with several configurations, and formulate a conjecture. Compare with Lemma 2.33 of NZM.

4. Suppose that a has order h modulo m, and that b has order k modulo m. How large can the order of ab be? How small? Use pairs taken from your work on Problem 1 above. If $(h, k) = 1$, what is the order of ab modulo m? Study some cases, and formulate a conjecture. Compare your findings with Lemma 2.34 of NZM.

5. Suppose that a has order h modulo m, that a has order k modulo n, and that $(m, n) = 1$. What is the order of a modulo mn? Try $a = 2$, $m = 7$, $n = 11$. Try $a = 2$, $m = 5$, $n = 17$. Try $a = 17$, $m = 7$, $n = 11$. Formulate a conjecture (after considering additional examples, if necessary).

6. Use PowerTab to determine the order of 7 (mod 101), and of 29 (mod 101), and use Mult to determine the value of $7 \cdot 29$ (mod 101). Repeat this with $17 \cdot 75$ (mod 91), and with $233 \cdot 313$ (mod 424). Suppose that $a \cdot \overline{a} \equiv 1 \pmod{m}$. Do you suspect a connection between the order of $a \pmod{m}$, and of $\overline{a} \pmod{m}$? Can you prove your conjecture? (This is found as Problem 14 at the end of §2.8 of NZM.)

The order of a modulo m can be determined by calculating a, a^2, $\ldots$ until the least h is found such that $a^h \equiv 1 \pmod{m}$. However, since this h may well be of size comparable to m, it is usually much faster to use the fact that $h | \phi(m)$. After factoring $\phi(m)$, we search for a minimal divisor h of $\phi(m)$ with the property that $a^h \equiv 1 \pmod{m}$. Note that if $a^d \equiv 1 \pmod{m}$, and if q is a prime divisor of d, then either $a^{d/q} \equiv 1 \pmod{m}$, in which case we replace d by d/q, or else $a^{d/q} \not\equiv 1 \pmod{m}$, in which case the power of q dividing d is the same as the power of q dividing the order of a. This technique is discussed on p. 100 of NZM,

7. To see how the order of 2 modulo 101 would be determined, type `orderdem 2 101 [Enter]`. To obtain the result without witnessing the calculation, type `order 2 101 [Enter]`. Since the first step is to factor m in order to calculate $\phi(m)$, some time may be saved by providing the value of $\phi(m)$, if this is known. Type `order 2 101 100 [Enter]`. The economy here can be quite noticeable: if the modulus is a 17-digit prime p, then it will be much faster to tell the machine that $\phi(p) = p - 1$, rather than let the machine try to factor p by trial division. When the values a, m, c are given to the program Order, it is not necessary that c actually be the value of $\phi(m)$. All that is required is that $a^c \equiv 1 \pmod{m}$. What happens if c does not meet this condition? Try typing `order 2 101 35 [Enter]`.

8. The program PrimRoot finds the least positive primitive root of a prime number p, by calculating the order of a for $a = 2, 3, \ldots$ until an a is found of order $p - 1$. Usually this does not take very many trials. Find the least positive primitive root of several primes in this way. For example, type `primroot 1093 [Enter]`. If you wish to find the least primitive root larger than a certain number a, type `primroot p a [Enter]`. (If you omit the a then by default a is set equal to 0.) By using the program PrimRoot repeatedly, find all the primitive roots of the prime $p = 101$. How many primitive roots do you find? (Recall Theorem 2.36 of NZM.) What is the biggest gap found between consecutive primitive roots?

9. The program PrimRoot is not equipped to find primitive roots modulo p^k when $k > 1$, but the program Order is useful in this connection. Suppose that g is a primitive root modulo p. Then g is a primitive root modulo p^2 if and only if the order of g modulo p^2 is $p(p-1)$. The only other possibility is that the order of g modulo p^2 is $p - 1$, in which case $g + tp$ is a primitive root modulo p^2 whenever $t \not\equiv 0 \pmod{p}$. (See the proof of Theorem 2.39 of NZM.) Is 2 a primitive root modulo 101^2? Show that 14 is a primitive root of 29. Is it a primitive root of 29^2? Find the least positive primitive root g of the prime 40487. Show that g is not a primitive root modulo 40487^2. (This is the least prime p whose least positive primitive root fails to be a primitive root modulo p^2.)

10. To determine the order of a residue class a modulo m, we need first a number c such that $a^c \equiv 1 \pmod{m}$. We could take $c = \phi(m)$, but usually a smaller number will do. Let $c(m)$ denote the least positive integer c such that $a^c \equiv 1 \pmod{m}$ for all reduced residue classes a. This is the *Carmichael function*. Its values are determined by the following relations. $c(1) = c(2) = 1$. $c(4) = 2$. If $k \geq 2$ then $c(2^k) = 2^{k-2}$. If p is an odd prime then $c(p^k) = p^{k-1}(p-1)$. If $(m_1, m_2) = 1$, then $c(m_1 m_2) = [c(m_1), c(m_2)]$. Use the program Car to determine the value of $c(100)$. Find a reduced residue class modulo 100 that has this maximal order.

11. For the programmer. Write a program that counts the number $N(x)$ of those primes p not exceeding x for which 2 is a primitive root of p. Would you conjecture that there are infinitely many such primes? Does it seem that this set of primes has positive asymptotic density among the set of all primes? That is, do you guess that $N(x) \sim c\pi(x)$ as $x \to \infty$ for some positive constant c? Gauss conjectured that there exist infinitely many such primes, and E. Artin suggested a particular asymptotic density. However, D. H. Lehmer, *A note on primitive roots*, Scripta Math. **26** (1963), 117–119 found that numerical evidence does not fit with Artin's conjecture. This led Artin to the realization that one aspect of the situation had been overlooked (see pp. viii, ix of Artin's Collected Works). A modified form of Artin's conjecture is now widely accepted as very likely to be true, especially since C. Hooley, *On Artin's conjecture*, J. Reine Angew. Math. **225** 1967, 209–210, showed that the modified conjecture is a consequence of the Generalized Riemann Hypothesis. The conjectured constant is

$$c = \prod_p \left(1 - \frac{1}{p(p-1)}\right)$$

where the product is taken over all primes. The number 2 can be replaced by any integer a, and the general conjecture is that there is a positive constant c_a such that $N_a(x) \sim c_a\pi(x)$ as $x \to \infty$, provided that $a \neq -1$, $a \neq 0$, and that a is not a perfect square.

LABORATORY 12

Indices — The Discrete Logarithm

New Programs: IndTab, Ind, IndDem, HSortDem

Suppose that g is a primitive root of the prime number p. If $(a,p)=1$ then there is a number ν such that $g^\nu \equiv a \pmod p$; moreover, the value of ν is uniquely determined modulo $p-1$. This ν is called the *index of a with respect to the primitive root g*. In symbols we write $\nu = \operatorname{ind}_g a$ when the value of p has already been specified. By way of analogy, any positive real number x can be written uniquely in the form e^y where $y = \ln x$. Thus $\operatorname{ind}_g a$ is a discrete analogue of $\ln x$.

For a given prime $p < 10^4$, the program IndTab displays a table of the indices of the reduced residue classes modulo p. If there are more values than can be displayed on a single screen, then you may use `PgUp` and `PgDn` or `j` to move around in the table. Initially, g is the least positive primitive root of p, but you are free to switch to a different primitive root. The program will prevent you from choosing a base that is not a primitive root. The program also provides a table of the powers of g, which is obtained by typing `e`. To return to the table of indices from the table of exponentials, type `i`. These tables may be used in the manner of tables of logarithms and exponentials, to find the solutions of multiplicative congruences. For example, to find the solutions of the congruence $x^3 \equiv 12 \pmod{97}$, we take $g = 5$, and write $x \equiv 5^\mu \pmod{97}$. From IndTab we discover that $\operatorname{ind}_5 12 = 42$. That is, $12 \equiv 5^{42} \pmod{97}$. Hence the initial congruence may be rewritten as $5^{3\mu} \equiv 5^{42} \pmod{97}$. This is equivalent to asserting that $3\mu \equiv 42 \pmod{96}$. From LinCon we discover that this is equivalent to $\mu \equiv 14 \pmod{32}$. That is, $\mu \equiv 14$, 46, or 78 modulo 96. Returning to the IndTab program, we enter $p = 97$ again, and then press `e` to switch to the table of exponentials, i.e. powers of the primitive root 5. From this table we deduce that $5^{14} \equiv 48 \pmod{97}$, that $5^{46} \equiv 31 \pmod{97}$, and that $5^{78} \equiv 18 \pmod{97}$. Hence the desired solutions are $x \equiv 48$, 31, and 18 modulo 97. As a check, one may use the program Power to verify that $48^3 \equiv 31^3 \equiv 18^3 \equiv 12 \pmod{97}$. If the actual roots of the congruence $x^3 \equiv 12 \pmod{97}$ are not needed, but only the number of roots, then one may proceed more simply, using Euler's criterion (Corollary 2.38 on p. 101 of NZM) and the program Power: Since $12^{96/(3,96)} = 12^{32} \equiv 1 \pmod{97}$, it follows that the given congruence has exactly $(3,96) = 3$ solutions.

1. Use the program IndTab to find the solutions of the congruence $x^4 \equiv 693 \pmod{1093}$.

2. Use IndTab to find all solutions of the congruence $x^5 \equiv 693 \pmod{1093}$.

3. Use IndTab in the manner above to show that the congruence $x^7 \equiv 693 \pmod{1093}$ has no solution. At what point in the argument does it become apparent that there is no solution? Use Theorem 2.37 of NZM to provide a simpler proof that this congruence has no solution.

4. Use IndTab to find all solutions of the congruence $x^{10} \equiv 475 \pmod{9973}$.

5. Use IndTab to find all x such that $2^x \equiv 133 \pmod{9973}$.

6. With $p = 9973$, use IndTab to determine the value of $\operatorname{ind}_{103} 877$.

The program IndTab is restricted to $p < 10^4$ because the entire table is computed at the outset, and held in active memory (RAM). We have efficient means to compute powers, and efficient means to located the least positive primitive root g, and thus we can easily compute values of $g^\mu \pmod{p}$ as μ runs over any given interval. What seems to be hard is to calculate values of $\operatorname{ind}_g a$ for a general a. Indeed, methods of encryption have been proposed whose security depends on the supposition that evaluating indices is computationally difficult. This computational snag is often referred to in the literature as the *problem of the discrete logarithm.*

The program IndTab first constructs a list of the values g^ν, (i.e., exponentials), and then uses it to form a table of the indices. Thus $\operatorname{ind} a$ is found for each $a \pmod{p}$, but the amount of work is proportional to p. A first step toward improving on this has been suggested by Shanks: Suppose that you wish to calculate $\operatorname{ind}_g x \pmod{p}$. Let s be a base to be described later; we want to find i and j so that $g^{is+j} \equiv x \pmod{p}$. To this end, use the extended Euclidean algorithm (i.e., the program LinCon) to find $\overline{g}$ so that $g\overline{g} \equiv 1 \pmod{p}$. Construct a table of the values $x\overline{g}^j \pmod{p}$ for $0 \le j < s$. Then, for $i = 0, 1, \ldots,$ compute $g^{is} \pmod{p}$, and look to see if the number computed is found in the table. When it is found, we have the desired values of i and j, and $\operatorname{ind}_g x = is + j$. When searching for a particular value of g^{is} in the table, it would be very slow to inspect all s values. Instead, we sort the table of values $x\overline{g}^j \pmod{p}$ by size, into increasing order. (A useful algorithm for sorting, called HeapSort, is discussed later in this laboratory.) Then one can search for a specified value in the table by binary subdivisions. To motivate the choice of s, we consider the amount of work is required. The time required to construct the table is $O(s)$, but the time required to sort it is a little greater, $O(s \log s)$. Searching by binary subdivision takes $O(\log s)$ steps, and we expect that it will be necessary to conduct $\approx p/s$ such searches. Thus the total amount of work is proportional to

$$\left(\frac{p}{s} + s\right) \log s.$$

This is minimized by taking $s \approx \sqrt{p}$, and then the time involved is $O(p^{1/2} \log p)$, a little slower than proving that p is prime by trial division. Thus we see that Shanks' algorithm is not very fast for big p, although it represents a big improvement over $O(p)$.

In practice, the parameter s above is constrained also by the amount of available memory. For example, the program Ind calculates $\operatorname{ind}_g x \pmod{p}$ by Shanks' method for $p < 10^9$; for $p < 10^8$ it takes s to be the integer nearest $\sqrt{p}$, but for larger p it takes $s = 10000$ so that the data fits into one 64K segment of memory. (Each entry of the table occupies 4 bytes, and a companion table 2 bytes each, so the tables require 60K of memory.) To witness Shanks' algorithm in action, type `inddem 2 45 101 [Enter]`. Also, try `inddem 2 3 1093 [Enter]`.

For more information concerning algorithms used to calculate indices see the following papers.

D. Coppersmith, A. M. Odlyzko, R. Schroeppel, *Discrete logarithms in* GF(p), Algorithmica **1** (1986), 1–15.

D. M. Gordon, *Discrete logarithms in* GF(p) *using the number field sieve*, SIAM J. Discrete Math. **6** (1993), 124–138.

B. A. LaMacchia, A. M. Odlyzko, *Computation of discrete logarithms in prime fields*, Des. Codes Cryptogr. **1** (1991), 47–62.

K. S. McCurley, *The discrete logarithm problem*, Cryptology and computational number theory (Boulder, 1989), Amer. Math. Soc., Providence, 1990, pp. 49–74.

A. M. Odlyzko, *Discrete logarithms in finite fields and their cryptographic significance*, Advances in Cryptography (Proc. 1984 EUROCRYPT Workshop), Springer-Verlag, New York, 1985, pp. 224–314.

7. Use PrimRoot to find a primitive root g modulo $p = 123456791$. Use the program Ind to determine $\operatorname{ind}_g 57085185 \pmod p$. For example, type `ind 17 57085185 123456791 [Enter]`. Use this information to find all roots of the congruence $x^5 \equiv 57085185 \pmod{123456791}$.

8. The program Ind searches for a specified value x among the powers of $g \pmod p$. It is essential that $(g, p) = 1$, but it is not necessary that p be prime or that g be a primitive root. Is 3 a power of 2 modulo 123456791? That is, does the congruence $2^\nu \equiv 3 \pmod{123456791}$ have a solution? Type `ind 2 3 123456791 [Enter]`. Here the modulus is prime, but the base is not a primitive root. Note that the program returns only the least non-negative solution. The period of the solutions is $p-1$ if g is a primitive root (mod p), but it is smaller in other cases. Find all ν (mod 123456790) such that $2^\nu \equiv 3 \pmod{123456791}$. (Hint: Use the program Order to determine the order of 2 (mod 123456791).) Confirm that Ind still works when p is composite, by typing `ind 2 23 91 [Enter]`. Type `2 17 123456791 [Enter]`. What happens?

9. Assume that $p = 1234567897531$ is prime. Use the programs GCD and Power to determine the number of roots of the congruence $x^{77} \equiv 13 \pmod{1234567897531}$. (Theorem 2.37 of NZM is relevant here.) Note that you do not have any tool available to find these roots, since p is so large. Such tools do exist; for example one might elaborate on the technique developed in the next laboratory. Alternatively, the polynomial $x^{77} - 13$ can be factored quickly (mod p), for example by the method of D. G. Cantor and H. Zassenhaus, *A new algorithm for factoring polynomials over finite fields*, Math. Comp. **36** (1981), 587–592.

We now consider the problem of sorting numbers by size. While not a number-theoretic problem, we find it useful (as above) to be able to sort numbers with reasonable efficiency. Suppose that $a_1, a_2, \ldots, a_n$ are n distinct numbers that we want to sort into increasing order. First, in *Bubble Sort*, one passes repeatedly through the list, transposing pairs that are found to be out of order, until a pass discloses no transpositions. This takes time $O(n^2)$, which is terrible! **Never** use Bubble Sort! So how much faster can we hope for? We derive a lower bound. When two elements a_i and a_j are compared, the set of all possible orderings is divided into two classes, those with $a_i > a_j$ and those with $a_i < a_j$. After k such comparisons have been made, the set of all possible orderings has been divided into at most 2^k classes. If $2^k < n!$ then there is a class containing two different orderings. Consequently, if the original ordering of our a_i is one of these orderings, then we have not yet distinguished it from all other possible orderings, and at least one more comparison is necessary. Thus for any sorting algorithm there is an ordering of the a_i that gives rise to more than $(\log n!)/\log 2$ comparisons. Since $n! > (n/e)^n$, the worst-case running time of any sorting algorithm is $\gg n \log n$. This lower bound is of the correct order of magnitude, since we have algorithms that run in $O(n \log n)$ time.

Among the possible methods that one might consider, we confine our attention to *HeapSort*, invented by J. W. J. Williams. This method runs in $O(n \log n)$ time, with the worst case only about 20% longer than the average. It is very easy to understand and to program, and requires little memory. In HeapSort, we think of the a_i as forming a binary tree, in which a_i has subordinates a_{2i} and a_{2i+1}, as long as these indices do not exceed n. Conversely, if $i > 1$ then a_i reports to its superior, $a_{[i/2]}$. The situation for $n = 28$ is depicted below:

$$a_1$$

$$a_2 \qquad\qquad\qquad\qquad a_3$$

$$a_4 \qquad\qquad a_5 \qquad\qquad a_6 \qquad\qquad a_7$$

$$a_8 \qquad a_9 \qquad a_{10} \qquad a_{11} \qquad a_{12} \qquad a_{13} \qquad a_{14} \qquad a_{15}$$

$$a_{16} \quad a_{17} \quad a_{18} \quad a_{19} \quad a_{20} \quad a_{21} \quad a_{22} \quad a_{23} \quad a_{24} \quad a_{25} \quad a_{26} \quad a_{27} \quad a_{28}$$

If a_i is smaller than one of its underlings then we exchange a_i with the larger of a_{2i} and a_{2i+1}. We repeatedly demote a particular entry until it majorizes its subordinates. (Executives above their level of competence are demoted.) We begin at the bottom of the table (high indices), and work up. Thus in the example above, we would compare a_{14} with a_{28}, and exchange them if a_{14} is the smaller. Then we compare a_{26} with a_{27} to determine which is the larger, and then compare that one with a_{13}. We demote a_{13} if one of the numbers under it is larger. We continue with this, until $a_i \ge a_{2i}$ and $a_i \ge a_{2i+1}$ whenever the indices lie between 1 and n. Such a configuration we call a *heap*. Once the heap has been formed, it is clear that a_1 is the largest number in the entire collection. We swap a_1 with a_n. This destroys the heap property, so we demote the new a_1 until it is restored. At the point we have a heap of $n - 1$ numbers, and a_n is ignored at the bottom. The new entry a_1 is the largest member in the heap (second largest, overall), so we exchange a_1 and a_{n-1}. At this point both a_{n-1} and a_n have reached their final resting places. We demote the new a_1 until we again have a heap, and then again the top of the heap is retired. Continuing in this manner, with successively smaller heaps, we eventually have no heap left, and $a_1 \le a_2 \le \cdots \le a_n$.

For extremely large collections (such as the Manhattan telephone directory), it is important to sort as quickly as possible. In such cases it may be worth using the more complicated QuickSort algorithm. For a detailed discussion of sorting, see D. E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, 1973. In particular, HeapSort is described on pp. 145–149, 153–158.

10. The program HSortDem demonstrates the HeapSort algorithm. Type `hsortdem [Enter]`, choose the number n of integers to be sorted, and witness the process. How does the number of comparisons compare with the lower bound derived above?

LABORATORY 13

Proving Primality

New Program: ProveP

We have seen that a composite number can be proved to be composite very quickly by means of the strong pseudoprime test. Finding the complete factorization of a composite number takes longer, but for large n we have methods that are much faster than trial division. To complete the picture we need a fast method for proving the primality of a large prime number p. In this direction, we show that proving the primality of p is no harder than factoring $p-1$. Suppose that $g^{p-1} \equiv 1 \pmod{p}$, and that $g^{(p-1)/q} \not\equiv 1 \pmod{p}$ for every prime factor q of $p-1$. Then g has order $p-1$ modulo p, and hence p must be prime. In general, if p is prime then such a g is not hard to find; thus we have a means of proving that p is prime provided that we can factor $p-1$.

1. Type `primroot 8675309 [Enter]`. The machine quickly responds with a primitive root, because 8675308 is easily factored. Thus the primality of 8675309 has been rigorously established. (David Farmer proposes that this is the largest prime number ever mentioned in a popular song.)

The simple idea used above can be strengthened in several ways. First, suppose that q is a prime factor of $p-1$, say $q^k \| (p-1)$. Let p' denote an arbitrary prime factor of p. Presumably the only such p' is $p' = p$, but this remains to be proved. Suppose that we can find a number a such that $a^{p-1} \equiv 1 \pmod{p}$ but such that $(a^{(p-1)/q} - 1, p) = 1$. Then $a^{p-1} \equiv 1 \pmod{p'}$ but $a^{(p-1)/q} \not\equiv 1 \pmod{p'}$. (Note that we can deduce this without knowing the value of p'.) Hence q^k divides the order of a modulo p', and consequently $q^k | (p'-1)$. That is, every prime factor p' of p is $\equiv 1 \pmod{q^k}$. Suppose we repeat this for several different prime factors q of $p-1$. (The value of a that works is allowed to depend on q.) Let s denote the product of the primepowers q^k for which this calculation has succeeded. Then we can assert that every prime factor p' of p is $\equiv 1 \pmod{s}$. Since the product of two or more such primes must be $> s^2$, we see that if $s > \sqrt{p}$ then p must be prime. Hence we can establish the primality of p, based only on an incomplete factorization of $p-1$, provided that we can factor s, $s|(p-1)$, and $s > \sqrt{p}$. (This analysis is related to Problems 38, 39 at the end of §2.8 of NZM.)

2. It is easy to confirm that $7^{16} + 5^{16} = 2 \cdot 16692759230113$. Use the program ProveP to demonstrate that this second factor is prime. That is, type `provep 16692759230113 [Enter]`, and note the results.

3. If, as the factorization of $p-1$ proceeds, a point is reached at which the factored portion s of $p-1$ is so large that testing p for divisors $d \equiv 1 \pmod{s}$, $d \le \sqrt{p}$ will take less time than the time already spent trying to factor $p-1$, then the program ProveP automatically switches to the latter approach. To witness an instance of this, apply ProveP to the number $5 \cdot 10^{17} + 21 = 500000000000000021$.

The method of proving primality being employed here can be made still more efficient. Suppose that as prime factors q of $p-1$ are being found, we reach a point at which $p^{1/3} < s \le p^{1/2}$. Then either p is prime or p is the product of two primes, $p = p'p''$, say. Write $p' = a's + 1$ and $p'' = a''s + 1$, so that $p = a'a''s^2 + (a' + a'')s + 1$. With a little care with inequalities, it can be shown that $0 < a'a'' < s$ and that $0 < a' + a'' < s$. Thus the representation of p that we have given here in terms of powers of s coincides with the expansion of p in base s. That is, by the division algorithm we may write $p = c_2s^2 + c_1s + 1$ with $0 \le c_i < s$, and $c_1 = a' + a''$, $c_2 = a'a''$. To determine whether such a' and a'' exist, we have only to test whether $c_1^2 - 4c_2$ is a perfect square. This embellishment is due to H. C. Williams.

4. Apply the program ProveP to demonstrate that the number 1234567897531 is prime. Also that 975312468097531 is prime. If the program GetNextP is given an argument x for which $10^9 < x < 10^{18}$, then the number p returned is the least integer $> x$ that is a strong pseudoprime to bases 2, 3, 5, 7 and 11. It is incredibly likely that p is prime, but to obtain a rigorous proof one should use the program ProveP. In this manner, find the least prime greater than 12345678987654321.

5. In most cases the method used by ProveP is reasonably quick. However, it can happen that $p - 1 = 2q$ where q is prime. In such a case, attention is focused on q. After a brief attempt to factor it by trial division fails, one should apply a strong pseudoprime test. If q passes the test, then a rigorous proof that q is prime may be obtained by applying the program ProveP to q. In attempting to factor $q-1$ one may encounter the same problem as with $p-1$. Nevertheless, by systematically employing the programs ProveP, SPsP, Factor and Rho, the needed factorizations can be rigorously established. For example, suppose that we apply the program ProveP to show that the number $p = 987292984329259$ is prime. The machine quickly finds that $p - 1$ is divisible by 2 and by 3, but then there is a pause. Touch any key to interrupt the program, and you find that it is trying to factor 164548830721543. Type `q` to quit, and then apply the program SPsP to this factor. In this way we discover that we are dealing with a composite number, so we apply the program Rho, which discloses that the factor may be written as $5378033 \cdot 30596471$. We apply the program Factor to the first of these numbers, to confirm that it is prime. Then we again apply ProveP to the original number p. Again the machine finds 2 and 3, but when it pauses, we interrupt it, type `s` to indicate that we wish to supply a prime factor, and enter 5378033. This time the program reaches a successful resolution without further intervention, and it is proved that p is indeed prime. Show that $10^{18} - 11 = 999999999999999989$ is prime. What intermediate numbers need to be factored? Give an account of the programs used, and the findings. (By applying GetNextP to this number you may confirm that this is the largest prime not exceeding 10^{18}. Similarly, show that $(10^{18} - 7)/3 = 333333333333333331$ is prime. Indicate what numbers are encountered, and how they are dealt with.

6. Show that $10^{17} + 19$ and $10^{17} + 21$ are both prime.

7. What is the first composite number in the sequence 31, 331, 3331, 33331, ... ? Apply ProveP until the first composite element is encountered. Here the k-th term is $u_k =$

$(10^k - 7)/3$. Show that for every k, the least prime factor dividing u_k is ≥ 17. Show that $17|u_k$ precisely when k lies in a certain residue class (mod 16). (Suggestion: Use the program PowerTab to display $10^k \pmod{m}$ for appropriate m.)

8. For the programmer. When attempting to prove that p is prime, we factor $p-1$. After removing the factor 2, this leaves $p_1 = (p-1)/2$ to be factored—but this may be prime. In such a case we would confirm that p_1 is prime by applying ProveP. However, it may happen that $p_1 = 2p_2 + 1$ with p_2 prime. Perhaps also $p_2 = 2p_3 + 1$ with p_3 prime. It is in such a case that our procedure for establishing primality will require the most work. How long can a chain of primes be, with $p_i = 2p_{i+1} + 1$? Construct a program to explore this. Apply the SPSP test to odd integers until a probable prime p is found. Then apply SPSP to $2p+1$, and so on, until a chain of k probable primes has been constructed. If this chain is of record-breaking length, print out p and k, so that the program ProveP can be applied to the members of the chain. The first chain you will find is $3,\ 7$; the next one is $5,\ 11,\ 23,\ 47$. Show that if p begins a chain of length at least 4, with $p > 5$, then necessarily $p \equiv 29 \pmod{30}$. For more on such chains, see D. H. Lehmer, *On certain chains of primes*, Proc. London Math. Soc. (3) **14a** (1965), 183–186.

By the method of primality proof employed here, we see that proving that p is prime is no harder than factoring $p-1$. Other methods of proving primality have been proposed, and some of these are significantly more efficient than our best factoring algorithms. Thus a prime of 1000 digits can be proved to be prime, but the record for factoring hard composite numbers stands below 200 digits. One of the methods currently in wide use is that of Adleman, Pomerance and Rumely (the APR method); it depends on Gauss sums. A method depending on elliptic curves, devised by Atkin and Morain, has achieved some striking successes lately. For more information concerning proofs of primality, consult the following papers.

L. M. Adleman, C. Pomerance, and R. S. Rumely, *On distinguishing prime numbers from composite numbers*, Ann. of Math. (2) **117** (1983), 173–206.

A. O. L. Atkin and F. Morain, *Elliptic curves and primality proving*, Math. Comp. **61** (1993), 29–68.

J. D. Dixon, *Factorization and primality tests*, Amer. Math. Monthly. **91** (1984), 333–352.

LABORATORY 14

Square Roots Modulo p

New Programs: SqrtModP, SqrtDem

In discussing pseudoprime tests and primitive roots we have generated a circle of ideas that we now harness to give a quick method for finding the roots of the quadratic congruence $x^2 \equiv a \pmod{p}$. The algorithm involved is described in detail in §2.9 of NZM. Before confronting the full algorithm, we consider two instructive examples.

1. If $p \equiv 3 \pmod 4$ then the solutions of the congruence $x^2 \equiv a \pmod p$ are given by $x \equiv \pm a^{(p+1)/4} \pmod p$. For example, suppose we wish to find solutions of the congruence $x^2 \equiv 2 \pmod{103}$. By using the program Power we find that $2^{26} \equiv 38 \pmod{103}$. Hence the desired solutions are ± 38, as we may confirm by verifying that $38^2 \equiv 2 \pmod{103}$. Use the program Power in this way to find the solutions of the congruence $x^2 \equiv 7 \pmod{103}$. What happens to this procedure if a is a quadratic nonresidue of p? For example, what happens if you try to use this method to solve the congruence $x^2 \equiv 3 \pmod{103}$? Explain why it is always the case that exactly one of a and $-a$ is a quadratic residue, if p is a prime, $p \equiv 3 \pmod 4$, and $a \not\equiv 0 \pmod p$.

2. Suppose that z is a quadratic nonresidue of p, so that by Euler's criterion $z^{(p-1)/2} \equiv -1 \pmod p$. If $p \equiv 1 \pmod 4$ then it follows that solutions of the congruence $x^2 \equiv -1 \pmod p$ are given by $x \equiv \pm z^{(p-1)/4} \pmod p$. However, to make use of this observation, we need to find the quadratic nonresidue z. Rather than give a deterministic algorithm for this, we simply try z at random, until a quadratic nonresidue is found. When z is selected, we compute $x \equiv z^{(p-1)/4} \pmod p$. Then either $x^2 \equiv -1 \pmod p$, in which case we are done, or else $x^2 \equiv 1 \pmod p$ (i.e., $x \equiv \pm 1 \pmod p$), in which case we start over with a new value of z. Since exactly half of the nonzero residue classes are quadratic nonresidues, the expected number of such trials is 2. An algorithm of this kind is referred to as a *Monte Carlo* algorithm, or as a *probabilistic* algorithm. Since the quadratic nonresidues seem to be randomly distributed between 0 and p, we do not take the trouble to use a random-number generator in selecting the values of z: It is enough to try consecutive integers (skipping the perfect squares). For example, $2^{24} \equiv -1 \pmod{97}$, and $3^{24} \equiv -1 \pmod{97}$, but $5^{24} \equiv 22 \pmod{97}$, and hence the solutions of $x^2 \equiv -1 \pmod{97}$ are given by $x \equiv \pm 22 \pmod{97}$.

3. We now proceed to the general case. To see how one would find the solutions of the congruence $x^2 \equiv 2 \pmod{97}$, type `sqrtdem 2 97 [Enter]`, and follow the prompts. What happens if you type `sqrtdem 5 97 [Enter]`? To get the same result without all the discussion, type `sqrtmodp 2 97 [Enter]`.

4. Apply the program SqrtDem to various values of a with $p = 223497217$. What is the power of 2 dividing $p - 1$?

5. Suppose that p is prime and that $p \equiv 2 \pmod 3$. Explain why $a^{(2p-1)/3}$ is the sole solution of the congruence $x^3 \equiv a \pmod p$. Use this principle and the program Power to determine the unique root of the congruence $x^3 \equiv 2 \pmod{101}$.

6. Suppose that p is prime and that $p \equiv 1 \pmod 3$. Explain how a probabilistic algorithm might be constructed to locate the roots of the congruence $x^3 \equiv 1 \pmod p$. (Hint: One might try $x \equiv z^{(p-1)/3} \pmod p$, where z is chosen randomly.) The congruence in question has exactly 3 roots, say x_0, x_1, x_2. Since 1 is one of these roots, we may suppose that $x_0 = 1$. Explain why $x_2 \equiv x_1^2 \pmod p$, and $x_1 \equiv x_2^2 \pmod p$. Thus if one of these roots can be found then so can the other. Use your method to find the solutions of the congruence $x^3 \equiv 1 \pmod{97}$. What is the probability that a given trial will be successful?

7. For the programmer. Write a program that finds the roots of the congruence $x^3 \equiv a \pmod p$. (Hint: Recall Problems 6 and 8 on p. 115 of the text.)

The algorithm we have used to take squareroots modulo p was invented by Dan Shanks in 1972; he called it *RESSOL*, because it SOLves for RESidues. This algorithm is very similar to one described much earlier by Tonelli. Other methods for taking squareroots modulo p have been given by Lehmer (related to earlier work of Cipolla), by Peralta, and by Adleman, Manders, and Miller. In addition, more general algorithms have been devised for factoring a polynomial modulo p; such an algorithm could be applied to the polynomial $x^2 - a$ in order to find the squareroots of a modulo p. One such algorithm has been proposed by Berlekamp, but the more recent method of Cantor and Zassenhaus seems to be the method of choice. Among the various methods, it is interesting to note that while Shanks' method is somewhat slower if $p-1$ is divisible by a high power of p, Peralta' method, which depends on the arithmetic of polynomials (mod p), is faster in this case. For more details one may consult the following papers.

L. Adleman, K. Manders, and G. Miller, *On taking roots in finite fields,* 18th IEEE Annual Sympos. Foundations of Computer Science, Providence, RI, 1977.

E. R. Berlekamp, *Factoring polynomials over large finite fields*, Math. Comp. **24** (1970), 713–735.

D. G. Cantor and H. Zassenhaus, *A new algorithm for factoring polynomials over finite fields,* Math. Comp. **36** (1981), 587–592.

D. H. Lehmer, *Computer technology applied to the theory of numbers,* Studies in Number Theory (W. J. LeVeque, ed.), Math. Assoc. Amer., 1969.

R. Peralta, *A simple and fast probabilistic algorithm for computing square roots modulo a prime number,* IEEE Trans. Info. Thy. **IT–32** (1986), 846–848.

M. Rabin, *Probabilistic algorithms in finite fields,* SIAM J. Comp. **9** (1980), 273–280.

D. Shanks, *Five number-theoretic algorithms,* Proceedings of the Second Manitoba Conference on Numerical Mathematics, 1972, pp. 51–70.

LABORATORY 15

Quadratic Residues

New Programs: JacobDem, JacobTab, Jacobi

As is discussed at the end of §3.3 of NZM, quadratic reciprocity provides a quick method of calculating the Jacobi symbol. The program JacobDem demonstrates the process. In addition, values of the Jacobi symbol exhibit a number of interesting and important patterns. These we can explore with the aid of the program JacobTab.

1. Use the program JacobDem to witness the calculation of the Jacobi symbol. Try typing `jacobdem [Enter]` and follow the prompts, or type `jacobdem 1234567 7654321 [Enter]`. To evaluate the Jacobi symbol without witnessing the calculation, type `jacobi 1234567 7654321 [Enter]`.

2. Use the program JacobTab to view a table of the values of the Jacobi symbol. For $p = 23$, what are the quadratic residues?

3. For $1 \le a \le p-2$, the pair $\left(\frac{a}{p}\right)$, $\left(\frac{a+1}{p}\right)$ takes on the values $(1,1)$, $(1,-1)$, $(-1,1)$, $(-1,-1)$. Using JacobTab with $p = 29$, classify the a according to which pair is generated. How many times does each configuration occur? Repeat this with $p = 37$, $p = 41$. Formulate a conjecture concerning the general situation when $p \equiv 1 \pmod 4$. Now try some primes $\equiv 3 \pmod 4$, say $p = 23$, $p = 31$, $p = 43$. Again, formulate a conjecture. Problem 18 at the end of §3.3 of NZM is relevant here.

4. Using JacobTab, evaluate the sum

$$\sum_{a=1}^{p} \left(\frac{a(a+1)}{p}\right)$$

for several primes, say $p = 11$, $p = 13$, $p = 17$, $p = 19$. Formulate a conjecture concerning the value of this sum. Note Problem 17 at the end of §3.3 of NZM.

5. Let $\delta = \pm 1$, $\epsilon = \pm 1$. The a for which $\left(\frac{a}{p}\right) = \delta$, $\left(\frac{a+1}{p}\right) = \epsilon$ are counted by the expression

$$\frac{1}{4}\sum_{a=1}^{p-2}\left(1+\delta\left(\frac{a}{p}\right)\right)\left(1+\epsilon\left(\frac{a+1}{p}\right)\right).$$

Explain why this is

$$= -\frac{1}{4}\left(1+\epsilon\left(\frac{-1}{p}\right)\right) - \frac{1}{4}(1+\delta) + \frac{1}{4}\sum_{a=1}^{p}\left(1+\delta\left(\frac{a}{p}\right)\right)\left(1+\epsilon\left(\frac{a+1}{p}\right)\right),$$

and why this in turn is

$$= \frac{1}{4}\left(p - 2 - \delta - \epsilon\left(\frac{-1}{p}\right)\right) + \frac{\delta\epsilon}{4}\sum_{a=1}^{p}\left(\frac{a(a+1)}{p}\right).$$

This identity establishes a relationship between the conjectures you made in the two preceding problems. Are your conjectures equivalent?

6. Using JacobTab, evaluate the sum

$$\sum_{a=1}^{(p-1)/2}\left(\frac{a}{p}\right)$$

for several odd prime numbers p, say $p = 11$, $p = 13$, $p = 17$, $p = 19$. Explain why this sum must vanish if $p \equiv 1 \pmod 4$. (Hint: $(\frac{a}{p}) = (\frac{-a}{p})$.) Explain why this sum never vanishes if $p \equiv 3 \pmod 4$. (Hint: What is this sum (mod 2)?) When $p \equiv 3 \pmod 4$, is there anything notable about the sign of this sum? Examine some further cases, and formulate a conjecture.

In 1839, Dirichlet proved an important class number formula, a special case of which asserts that if $p \equiv 3 \pmod 4$ and $p > 3$ then

$$\sum_{a=1}^{(p-1)/2}\left(\frac{a}{p}\right) = \left(2 - \left(\frac{2}{p}\right)\right)H(-p).$$

Here $H(-p)$ is the number of inequivalent classes of quadratic forms of discriminant $-p$, as defined in §3.5 of NZM. From this (deep) result we see that the sum on the left hand side above is always positive when $p \equiv 3 \pmod 4$. For an exposition of Dirichlet's class number formula, see §1 and §9 of H. Davenport, Multiplicative Number Theory, 2nd Edition, Springer-Verlag, New York, 1980, especially (8) on p. 9 and (15) on p. 49.

7. Using JacobTab as an aid, test the following assertion: For every prime number $p \ge 11$, the interval $[1, 10]$ contains two consecutive quadratic residues. Is the same true of the interval $[1, 9]$? Is there a similarly uniform upper bound for the first occurrence of three consecutive quadratic residues? Explore. The answer, which will come as a surprise, is given by D. H. Lehmer and E. Lehmer, *On runs of residues*, Proc. Amer. Math. Soc. **13** (1962), 102–106.

8. Let $n_2(p)$ denote the least positive quadratic nonresidue of p. Using JacobTab, determine the value of $n_2(p)$ for 25 odd primes chosen at random. What values does $n_2(p)$ take on, and how many times? Is there any reason why the number $n_2(p)$ should always be prime?

Erdős combined quadratic reciprocity and the prime number theorem for arithmetic progressions to show that $n_2(p) = 2$ for asymptotically 1/2 of the primes, that $n_2(p) = 3$

for asymptotically $1/4$ of the primes, that $n_2(p) = 5$ for asymptotically $1/8$ of the primes, that $n_2(p) = 7$ for asymptotically $1/16$ of the primes, and so on.

9. Let $p_2(p)$ denote the least prime quadratic residue of p. Using JacobTab, determine the value of $p_2(p)$ for 25 randomly chosen odd primes p. What values are taken on, and how frequently? What is $p_2(163)$?

LABORATORY 16

Binary Quadratic Forms

New Programs: ClaNoTab, QFormTab, Reduce

Whether a number n can be expressed as a sum of two squares can be elegantly characterized in terms of the canonic factorization of n into prime powers (recall Theorem 2.15 of NZM). It is therefore natural to ask whether something similar happens with other binary quadratic forms. The answer, as discussed in §3.4–3.7 of NZM, is generally less satisfactory.

1. What is the discriminant of the form $f(x,y) = 3508x^2 + 11259xy + 9034y^2$? Is this form definite or indefinite? (Recall Theorem 3.11 of NZM.) Type `reduce 3508 11259 9034 [Enter]` to find a reduced form that is equivalent to $f(x,y)$. Use the program QFormTab to view a list of all the reduced quadratic forms of this discriminant. Describe, in terms of the arithmetic progressions that they fall in, the primes represented by this form. (Suggestion: Use Corollary 3.14 and Theorem 3.17 of NZM.)

2. What is the discriminant of the form $f(x,y) = 1039x^2 + 11223xy + 30307y^2$? Is this form definite or indefinite? Using the program QFormTab,construct a list of all the reduced quadratic forms of this discriminant. Describe, in terms of arithmetic progressions that they fall in, the primes represented by this form. Type `reduce 1039 11223 30307 [Enter]` to find a reduced form $g(x,y)$ that is equivalent to the given form. From the information displayed, find values of x and y such that $g(x,y) = 1039$. Now type `reduce [Enter]`, without entering the coefficients on the command line. Then enter the coefficients in response to the prompts. This gives you an environment in which forms may be manipulated. If you type `r` then the bottom form in the table is reduced, the steps of the reduction are displayed, with the matrix that takes f to g. To view the inverse matrix, that takes g to f, say $M : g \to f$, type `m`. To express the original first coefficient 1039 properly by g, one takes $x = m_{11}$, $y = m_{21}$ where $M = [m_{ij}]$ (recall the formulæ (3.7) in the text). In this environment, enter

$$\begin{aligned} a &= 123456789876543401, \\ b &= 31971493083730684, \\ c &= 2069907153395965, \end{aligned}$$

and type `r` to reduce this form. In this way, discover a representation of the prime a as a sum of two squares.

The prime $p = 123456757$ is $\equiv 1 \pmod 4$, and hence can be written as a sum of two squares. In order to find such a representation, we first construct a quadratic form $f(x,y) = ax^2 + bxy + cy^2$ with $a = p$ and discriminant $d = -4$. That is, we must find b and c so that $b^2 - 4pc = -4$. By using SqrtModP, we find that $x^2 \equiv -4 \pmod p$

where $x = 51035038$. We need b to satisfy $b^2 \equiv -4 \pmod{4p}$. Thus we may take $b \equiv x \pmod{p}$. We also need b to be even, so that $b^2 \equiv -4 \pmod{4}$. Since x is even, it suffices to take $b = x$. Then $c = (b^2+4)/(4p) = 5274266$. (If such a calculation is beyond the capabilities of your pocket calculator, you may perform the arithmetic in the UBASIC environment. From the UBASIC prompt, type `print (51035038^2 + 4)\(4*123456757) [Enter]`. Here the \ is the UBASIC command for integer division.) Next we use the (Turbo Pascal) program Reduce to reduce this quadratic form. The only reduced form of discriminant -4 is x^2+y^2, and hence not only is $f(x,y)$ equivalent to this form, but we find the value of x and y that we should take to give a proper representation of a. From the values displayed, we find that $123456757 = 10281^2 + 4214^2$.

3. Use the programs SqrtModP and Reduce, as described above, to find a proper representation of the prime 987654337 as a sum of two squares. (This is similar to Example 3 in §3.6 of NZM.)

4. The number 20193797 is a product of two primes $\equiv 1 \pmod{4}$. Hence 20193797 can be expressed as a sum of two squares. Use the program Factor to find these prime factors, say $20193797 = p_1p_2$. Use SqrtModP to find x_i such that $x_i^2 \equiv -4 \pmod{p_i}$, for $i = 1, 2$. Then use CRT to find numbers b such that $b \equiv \pm x_1 \pmod{p_1}$, $b \equiv \pm x_2 \pmod{p_2}$, and $b \equiv 0 \pmod{2}$. Note that because of the various possible choices of the signs, there are 4 such numbers b. For each such b, put $c = (b^2+4)/(4a)$. Reduce the 4 quadratic forms to obtain representations of 20193797 as a sum of two squares. How many distinct ordered pairs (x, y) of positive integers do you obtain? Compare your findings with Theorem 3.22 of NZM.

5. Use the program QFormTab to view the reduced quadratic forms of discriminant -20. How many such forms are there? The prime number 666666667 is properly represented by the form $666666667x^2 + 200000xy + 15y^2$, whose discriminant is -20. Reduce this form, to determine a representation of 666666667 by one of the reduced forms. (Problems 5 and 10 at the end of §3.6 of NZM are relevant here.)

6. The program ClaNoTab generates a table of the class numbers of binary quadratic forms of negative discriminant. This program operates by the straightforward approach of noting the value of $b^2 - 4ac$ whenever $-a < b \le a \le c$ or $0 \le b \le a = c$, for each a, $a = 1, 2, \ldots, 57$. This gives a complete count of the reduced quadratic forms for each discriminant d in the interval $-10000 \le d < 0$. Since the computer must consider a large number of triples (roughly 10^6 of them), the program takes some time to generate the table. Scroll down through the table, looking for d for which the class number $h(d)$ is 1. How many such d do you find? Gauss found these d, and conjectured that there are no more. In 1934 it was proved that there could be at most one more such d. Finally in 1952, Heegner solved the Gauss class number problem by showing that there are no further $d < 0$ for which the class number is 1. (There are lots of $d > 0$ for which the class number is 1, and it is conjectured that there are infinitely many, though this has not yet been proved.) When $d < 0$, the numbers $h(d)$ grow irregularly with $|d|$. How does $h(d)$ compare with $\sqrt{-d}$?

It is known that if $d < 0$ then $h(d) = O(\sqrt{-d}\log -d)$, and also that if $\epsilon > 0$ then there is a $D_0(\epsilon) < 0$ such that if $d < D_0(\epsilon)$ then $h(d) > d^{1/2-\epsilon}$. Moreover, it is known that if the Generalized Riemann Hypothesis is true then $h(d)/\sqrt{-d}$ lies between $c/\log\log -d$ and $c\log\log -d$.

7. If a, b, and c are large (in absolute value), how likely is it that $d = b^2 - 4ac$ is small? Try some triples in the environment of the program Reduce. Suppose that $a = 111111222222333333$ and that $c = 333333222222111111$. How many b's are there for which $|d| < 10^{18}$?

Each form of negative discriminant is equivalent to a unique reduced form. (Recall Theorem 3.25 of NZM.) In particular, the reduced forms of given negative discriminant d are mutually inequivalent. Hence the number $H(d)$ of equivalence classes of positive definite binary quadratic forms of discriminant d, $d < 0$, is equal to the number of reduced positive definite forms of discriminant d. For $d > 0$ our reduction process is incomplete, and reduced forms may be equivalent. Thus for $d > 0$ the number of reduced forms is only an upper bound for the number $H(d)$ of equivalence classes.

8. Using the program QFormTab, construct a list of the reduced quadratic forms of discriminant 5. In the environment of Reduce, take $a = 1$, $b = 1$, $c = -1$. Type `s`, and then type `i`. Deduce that the two reduced forms are equivalent, $H(5) = 1$, and give the matrix that takes one to the other. Complete the following statement: "A prime p is represented by the form $x^2 + xy - y^2$ if and only if ... ." (This is similar to Example 2 in §3.5 of NZM.)

9. Use the program QFormTab to construct a list of reduced forms of discriminant 12. Show that $x^2 - 3y^2 = -1$ has no solution because it has no solution as a congruence modulo 3. Deduce that the two reduced forms are inequivalent, and hence that $H(12) = 2$.

10. The form $f(x, y) = 17x^2 + 8xy + y^2$ has discriminant -4, and hence is equivalent to $g(x, y) = x^2 + y^2$. In the environment of Reduce, enter $a = 1$, $b = 0$, $c = 1$. By typing a sequence of `s`'s, `t`'s, and `i`'s, try to get to f. If you are unsuccessful, and need a hint, enter $a = 17$, $b = 8$, $c = 1$, and type `r`. This gives the sequence that takes f to g. Now go backwards.

11. In the environment of Reduce, enter $a = 1$, $b = 1$, $c = 1$. Type `s` twice. Note that you are back at the original form, but that the matrix is $-I$, not I. Thus $-I$ takes the form to itself. This is called an *automorph* of the form. Type several characters, each one being one of `s t i`, and then type `r`. What matrix M now takes the form to itself? By experimenting in this way, find all the automorphs of this form. (There are 6 of them altogether, including I.) Can you prove that your list is complete? (The relevant matrices are found in the proof of Theorem 3.26 of NZM.)

12. Consider a matrix M, written as a product in which each factor is one of the matrices S, T, or T^{-1}. If there are many factors, then the elements of M are likely to be large. In the environment of Reduce, type several characters, each one being one of `s t i`. How large an element m_{ij} can you obtain in at most 20 keystrokes?

LABORATORY 17

Arithmetic Functions

New Programs: ArFcnTab, Pi

A function is called an *arithmetic function* if its domain is the set of positive integers (or perhaps the set $\mathbf{Z}$ of all integers). Among the most important and useful arithmetic functions are the following: The number $\omega(n)$ of distinct primes dividing n, $\omega(n) = \sum_{p|n} 1$. The number $\Omega(n)$ of primes dividing n, counting multiplicity, $\Omega(n) = \sum_{p^a \| n} a$. The Möbius μ-function, which is defined to be $(-1)^{\Omega(n)}$ if n is squarefree, and 0 otherwise. The divisor function $d(n)$, which is the number of positive divisors of n, $d(n) = \sum_{d|n} 1$. By the Chinese Remainder Theorem, we may show that $d(n) = \prod_{p^k \| n}(k+1)$. The Euler ϕ-function, which counts the number of reduced residues modulo n. By using the Chinese Remainder Theorem we know that $\phi(n) = n\prod_{p|n}(1-1/p)$. The σ-function is the sum of the positive divisors of n, $\sigma(n) = \sum_{d|n} d$. By using the Chinese Remainder Theorem we may show that $\sigma(n) = n\prod_{p^a\|n}(1-1/p^{a+1})/(1-1/p)$.

1. The program ArFcnTab provides a table of the six arithmetic functions defined above, for $1 \le n \le 10^9$. Type `arfcntab [Enter]`. You may use the `PgUp` and `PgDn` keys to page up or page down through the table. By typing `j` and then entering a number, you may jump to a different part of the table. When you are done using the table, type `Esc` to exit. By scrolling down through the table, make a list of those $n \le 200$ for which $d(n)$ is odd. Formulate a conjecture. Can you prove it? (Theorem 4.3 of NZM is useful here.)

2. For $1 \le n \le 10$, compute a table of values of the function $\sum_{d|n}\mu(d)$. Choose an n at random, $1 \le n \le 10^9$. Use the program Factor to factor n, and then list the divisors of n. For each d dividing n, use the factorization of d to determine the value of $\mu(d)$, and confirm that ArFcnTab provides the same values. For this n, evaluate $\sum_{d|n}\mu(d)$. Formulate a conjecture concerning the values of this sum. (See Theorem 4.7 of NZM.)

3. For $1 \le n \le 10$, construct a table of the values of $\sum_{d|n}\phi(d)$. Choose a large n at random, $1 \le n \le 10^9$. Use Factor to factor n, and construct a list of the divisors of n. Use ArFcnTab to provide the values of $\phi(d)$ for these divisors, and hence evaluate the sum $\sum_{d|n}\phi(d)$. Formulate a conjecture regarding the values of this sum. (See Theorem 4.6 of NZM.)

4. Make a list of those n, $1 \le n \le 50$, for which $\omega(n) = \Omega(n)$. What do you notice about the prime factorizations of these n? Describe these n in some other way.

5. Using ArFcnTab, look for small values of $\omega(n)$. Other than $\omega(1) = 0$, what is the smallest value you find? When does it take this small value? Does it take this value infinitely many times? Why? Now look for large values of $\omega(n)$. Make a list of those n,

$1 \le n \le 500$, for which $\omega(n)$ is larger than any previous values. That is, if $1 \le m < n$ then $\omega(m) < \omega(n)$. Give the prime factorization of each of these n. Formulate a conjecture regarding these n. Can you prove your conjecture? (See Theorem 8.30 in NZM.)

6. Proceed as in the preceding problem, but with $\omega(n)$ replaced by $\Omega(n)$. (Problem 10 at the end of §8.3 is relevant here.)

7. Construct a list of those n, $1 \le n \le 100$ for which $\phi(n)$ is larger than any preceding value. That is, if $1 \le m < n$ then $\phi(m) < \phi(n)$. Formulate a conjecture regarding these n. What information would you need concerning the distribution of prime numbers in order to prove your conjecture?

8. Construct a table of those n, $1 \le n \le 50$, for which $\phi(n)/n$ is smaller than any preceding value. That is, if $1 \le m < n$ then $\phi(m)/m > \phi(n)/n$. Formulate a conjecture concerning this set of integers n. Can you prove your conjecture? (Problem 15 at the end of §8.3 of NZM is relevant here.)

9. Construct a table of the values of $\sum_{d^2 \mid n} \mu(d)$, for $1 \le n \le 20$. Formulate a conjecture concerning the values taken by this sum. Can you prove your conjecture? (See the proof of Theorem 8.25 in NZM.)

10. Construct a table of the values of $2^{\omega(n)}$, of $d(n)$, and of $2^{\Omega(n)}$, for $1 \le n \le 20$. Formulate a conjecture concerning the relative sizes of these three functions. Can you prove your conjecture? (See the discussion in the middle of p. 395 of NZM.)

11. A number n is called *perfect* if $\sigma(n) = 2n$. That is, n is the sum of its proper divisors. What perfect numbers do you find in the interval $1 \le n \le 50$? It has long been conjectured that there are no odd perfect numbers—indeed, this is very probably the oldest unsolved problem in all of mathematics. By examining the values provided by ArFcnTab, confirm that the numbers 496, 8128, and 33550336 are also perfect. Factor these numbers, and note that their prime decompositions exhibit a common pattern. Can you show that all even perfect numbers are of this shape?

12. The values of some of our six arithmetic functions tend to be correlated. For example, $\omega(n)$ tends to be large (but is not always large) when $\Omega(n)$ is large. In the case of $\phi(n)$ and $\sigma(n)$, the correlation is negative: $\sigma(n)$ tends to be large when $\phi(n)$ is small. To investigate this principle in a quantitative form, tabulate the values of $\phi(n)\sigma(n)/n^2$ for $1 \le n \le 10$, and also for several large values of n. Do all the values observed lie in the interval $[6/\pi^2, 1]$? If so, why should they?

13. Although $d(n)$ takes on some large values for large n, these values are small compared with fractional powers of n. More precisely, for any $\delta > 0$ there is a constant C_δ such that $d(n) \le C_\delta n^\delta$ for all positive integers n. Tabulate the values of $d(n)/\sqrt{n}$ for $1 \le n \le 15$. What is the largest value observed? This is the unique maximum of this function. The unique maximum of $d(n)/n^{1/3}$ is attained at $n = 2520$. What is this maximum value?

The maximum of $d(n)/n^{1/4}$ is attained at $n = 21621600$. What is this maximum? The maximum of $d(n)/n^{1/5}$ occurs at $n = 6064949221531200$. What is this maximum? Here $n > 10^9$, so you are now beyond the range of ArFcnTab. To calculate $d(n)$ you must factor n and use the formula. The maximum of $d(n)/n^{1/6}$ occurs at $n = 2^6 \cdot 3^4 \cdot 5^3 \cdot 7^2 \cdot 11^2 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59 \cdot 61$. What is this maximum? To understand how these n are found, see the discussion leading to (8.54) on pp. 395–396 of NZM. This analysis goes back to S. Ramanujan, *Highly Composite Numbers*, Proc. London Math. Soc. **2** 1915, 347–409; Collected Papers pp. 78–128. The set of n for which $d(n)$ assumes a record-breaking value is not so easy to describe completely, although Ramanujan determined many of its properties.

14. One might expect that the Möbius function takes the values $+1$ and -1 with roughly equal frequency. To test this hypothesis, put $M(x) = \sum_{1 \le n \le x} \mu(n)$, and tabulate $M(x)$ for integral values of $x \le 100$. Here only squarefree numbers are being counted, so it is natural to consider also $L(x) = \sum_{1 \le n \le x} (-1)^{\Omega(n)}$. Form a similar table of this function. How do the values of these functions compare with $\sqrt{x}$? Here the numerical evidence may lead you to formulate false conjectures. It was conjectured by Mertens that $|M(x)| \le \sqrt{x}$ for all $x \ge 1$. Although it is now believed that $\limsup M(x)/\sqrt{x} = +\infty$, the first disprove of Mertens' conjecture was found only recently (A. M. Odlyzko and H. J. J. te Riele, *Disproof of the Mertens Conjecture*, J. Reine Angew. Math. **357** (1985), 138–160). The argument disproves Mertens' conjecture by showing that $\limsup M(x)/\sqrt{x} > 1.06$. Concerning $L(x)$, Pólya conjectured that $L(x) < 0$ for all $x \ge 2$. This was disproved by C. B. Haselgrove, *A disproof of a conjecture of Pólya*, Mathematika **5** (1958), 141–145, and later R. Sherman Lehman, *On Liouville's function*, Math. Comp. **14** (1960), 311–320 showed more explicitly that $L(906180359) = 1$. This is not necessarily the least counterexample, but it is known that Pólya's conjecture is true for all $x \le 6 \cdot 10^6$.

15. The program Pi calculates the number $\pi(x)$ of primes not exceeding x. The program operates by first sieving to construct a table of primes not exceeding 31607. Since the next prime after this, namely 31621, is larger than $\sqrt{10^9}$, it follows that primes up to 10^9 can be determined by using these small primes for sieving. To limit the use of memory, the primes are constructed in intervals of length 10^4, one interval at a time, until the limit x is reached. The program is restricted to $x \le 10^6$, because the running time (which is roughly comparable to x) is too great for larger x. For $x = 10^k$, $1 \le k \le 6$, how does $\pi(x)$ compare with $x/\log x$? A better approximation is given by

$$\operatorname{li} x = \int_2^x \frac{du}{\log u},$$

but numerical values of this integral are not so easy to compute.

Faster methods of computing $\pi(x)$ are discussed in the following papers.

J. C. Lagarias, V. S. Miller, and A. M. Odlyzko, *Computing $\pi(x)$: The Meissel-Lehmer method*, Math. Comp. **44** (1985), 537–560.

J. C. Lagarias and A. M. Odlyzko, *New algorithms for computing* $\pi(x)$, Number Theory: New York 1982, (D. V. Chudnovsky, G. V. Chudnovsky, H. Cohn and M. B. Nathanson, eds.), pp. 176–193; Lecture Notes in Mathematics 1052, Springer-Verlag (Berlin), 1984.

J. C. Lagarias and A. M. Odlyzko, *Computing* $\pi(x)$*: an analytic method*, J. Algorithms **8** (1987), 173–191.

16. G. H. Hardy and S. Ramanujan proved that for most integers n, both $\omega(n)$ and $\Omega(n)$ are approximately $\log\log n$. Their proof was complicated; the more elegant method used in proving Theorem 8.32 and Corollary 8.33 of NZM was found later by P. Turán. This has an interesting consequence: Since $2^{\omega(n)} \le d(n) \le 2^{\Omega(n)}$ for all n, it follows that for most n, $(\log n)^{c-\epsilon} < d(n) < (\log n)^{c+\epsilon}$ where $c = \log 2 = 0.693\ldots$. Using ArFcnTab, compute the averages of $\omega(n)$ and $\Omega(n)$ in the intervals $(10^k - 50, 10^k]$ for $1 \le k \le 9$, and compare these averages with $\log\log 10^k$. (Remember, as always to use natural logarithms, i.e., logs to the base e.)

17. By borrowing code from the program Pi, construct a program to count the number $\pi_2(x)$ of twin primes not exceeding x. Does this function increase at a regular rate? For interesting information regarding the distribution of prime numbers, see Don Zagier, *The First* 50 *Million Prime Numbers*, Math. Intell. **1** (1978), 7–19.

Reference Guide to Turbo Pascal Programs

ArFcnTab

Function Constructs a TABle of values of the six ARithmetic FunCtioNs $\omega(n) = \sum_{p|n} 1$, $\Omega(n) = \sum_{p^a \| n} a$, $\mu(n)$, $d(n) = \sum_{d|n} 1$, $\phi(n)$, and $\sigma(n) = \sum_{d|n} d$.

Syntax `arfcntab`

Commands

`PgUp`	Display the preceding 20 values
`PgDn`	Display the next 20 values
`J`	Jump to a new point in the table
`P`	Print 500 values, starting at the top of the displayed screen
`Esc`	Escape from the environment

Restrictions $1 \le n < 10^9$

Algorithm When the program begins execution, it first constructs a list of the primes not exceeding $10^{9/2}$, by sieving. These primes are used for trial division. The factorizations are determined simultaneously for all 20 numbers (or all 500 numbers, in the case of printing).

See also Pi

BasesTab

Function Constructs a TABle of the expansions of integers n in various BASES b.

Syntax `basestab`

Commands

`PgUp`	Display the preceding 20 values
`PgDn`	Display the next 20 values
$\leftarrow$	Shift to smaller bases
$\rightarrow$	Shift to larger bases
`J`	Jump to a new point in the table
`Esc`	Escape from the environment

Restrictions $2 \le b \le 16$, $1 \le n \le 10^{18}$

Algorithm The division algorithm is used to calculate base b digits, trailing digits first.

Car

Function Computes the CARmichael function $c(m)$, which is defined to be the least positive integer c such that $a^c \equiv 1 \pmod{m}$ whenever $(a, m) = 1$.

Syntax `car [m]`

Restrictions $1 \le m < 10^{18}$

Algorithm First the canonical factorization of m is determined by trial division. If p is an odd prime then $c(p^j) = p^{j-1}(p-1)$. Also, $c(2) = 1$, $c(4) = 2$, and $c(2^j) = 2^{j-2}$ for $j \ge 3$. Finally, $c(m)$ is the least common multiple of the numbers $c(p^\alpha)$ for $p^\alpha \| m$.

See also Phi

ClaNoTab

Function Constructs a TABle of CLAss Numbers of positive definite binary quadratic forms. The number $H(d)$ is the total number of equivalence classes of such forms of discriminant d, while $h(d)$ counts only those equivalence classes consisting of primitive forms.

Syntax `clanotab`

Commands

`PgUp`	Display the preceding 40 values
`PgDn`	Display the next 40 values
`J`	Jump to a new point in the table
`P`	Print $h(d)$ and $H(d)$ for $-2400 \le d < 0$
`Esc`	Escape from the environment

Restrictions $-10^4 \le d < 0$

Algorithm All reduced triples (a, b, c) are found, with $0 < a < \sqrt{10^4/3}$. When a reduced triple is located, the value $d = b^2 - 4ac$ is calculated, and the count of $H(d)$ is increased by 1. If $\gcd(a, b, c) = 1$ then the count of $h(d)$ is also increased by 1. The entire table is calculated before the first screen of values appears. This may take several minutes on a slow machine.

See also QFormTab, Reduce

Comments The time required to calculate class numbers in this manner in the range $-D \le d < 0$ is roughly proportional to $D^{3/2}$, and roughly D numbers must be stored. By adopting a more sophisticated algorithm, one could calculate only those values that are to appear on a given screenful, and the time required for the calculation would be much smaller, making it feasible to construct a program of this sort that would accommodate d in the range $-10^9 \le d < 0$, say. For faster algorithms, see D. Shanks, *Class number, a theory of factorization, and genera*, Proc. Sympos. Pure Math. **20**, Amer. Math. Soc., Providence, 1970, 415–440. For a method that is theoretically still faster, but that may be challenging to implement, see J. L. Hafner and K. S. McCurley, *A rigorous subexponential algorithm for computation of class groups*, J. Amer. Math. Soc. **2** (1989), 837–850.

CngArTab

Function Displays the addition and multiplication TABles for CoNGruence ARithmetic (mod m).

Syntax `cngartab`

Commands

↑	Move up
↓	Move down
←	Move left
→	Move right
`a`	Start at column a
`b`	Start at row b
`m`	Set modulus m
`s`	Switch between addition and multiplication
`r`	Display only reduced residues (in multiplication table)
`p`	Print the table (if $m \le 24$)
`Esc`	Escape from the environment

Restrictions $1 \le m < 10^9$

See also PowerTab

CoDivTab

Function Constructs a TABle of the COmmon DIVisors of two given numbers b and c.

Syntax `codivtab`

Restrictions $1 \le b < 10^9$, $1 \le c < 10^9$

Algorithm Tests every d in the range $1 \le d \le \min(b, c)$.

See also CoMulTab, DivTab

CoMulTab

Function Constructs a TABle of the COmmon MULtiples of two given numbers b and c.

Syntax `comultab`

Restrictions $|b| < 10^9$, $|c| < 10^9$

Algorithm The Euclidean Algorithm is used the calculate (b, c), and hence $[b, c]$. Then multiples of this latter number are listed.

See also CoDivTab

CRT

Function Determines the intersection of two arithmetic progressions. Let $g = (m_1, m_2)$. The set of x such that $x \equiv a_1 \pmod{m_1}$, $x \equiv a_2 \pmod{m_2}$ is empty if $a_1 \not\equiv a_2 \pmod{g}$. Otherwise the intersection is an arithmetic progression $a \pmod{m}$. In the Chinese Remainder Theorem it is required that $g = 1$, and then $m = m_1 m_2$. In general, $m = m_1 m_2 / g$.

Syntax `crt` [$\mathtt{a_1\ m_1\ a_2\ m_2}$]

Restrictions $|a_i| < 10^{18}$, $1 \le m_i < 10^{18}$

Algorithm First the linear congruence $m_1 y \equiv a_2 - a_1 \pmod{m_2}$ is solved. If $a_1 \not\equiv a_2 \pmod{g}$, then this congruence has no solution, and the intersection of the two given arithmetic progressions is empty. Otherwise, let y denote the unique solution of this congruence in the interval $0 \le y < m_2/g$. Then the intersection of the two given arithmetic progressions is the set of integers $x \equiv a \pmod{m}$ where $a = y m_1 + a_1$ and $m = m_1 m_2 / g$.

See also CRTDem, IntAPTab, LinCon, LnCnDem

CRTDem

Function Demonstrates the method employed to determine the intersection of two given arithmetic progressions.

Syntax `crtdem` [$\mathtt{a_1\ m_1\ a_2\ m_2}$]

Restrictions $|a_i| < 10^{18}$, $1 \le m_i < 10^{18}$

Algorithm See the description given for the program CRT.

See also CRT, IntAPTab, LnCnDem

D2R

Function Converts a Decimal TO Rational. That is, the program returns the rational number a/q with least q such that the initial decimal digits of a/q coincide with the decimal digits given.

Syntax `d2r` [`x`]

Restrictions $|a| < 10^{18}$, $1 \le q < 10^{18}$

Algorithm Suppose that k decimal digits of x are given after the decimal point. Put $\delta = 0.5 \cdot 10^{-k}$. We want to find a/q with q minimal such that $|x - a/q| \le \delta$. By the continued fraction algorithm the least i is found such that $|x - h_i/k_i| \le \delta$. Then the desired rational number is given by $a = ch_{i-1} + h_{i-2}$, $q = ck_{i-1} + k_{i-2}$ where c is the least positive integer such that a/q lies in the specified interval. Since this inequality holds when $c = a_i$, it suffices to search the interval $[1, a_i]$.

See also R2D

DetDem

Function Demonstrates the method used to evaluate $\det(A) \pmod{m}$.

Syntax `detdem`

Restrictions $0 < m < 10^9$, $A = [a_{ij}]$ is $n \times n$ with $1 \le n \le 9$, $|a_{ij}| < 10^9$

Algorithm See description for the program DetModM.

See also DetModM, SimLinDE

DetModM

Function Determines $\det(A) \pmod{m}$.

Syntax `detmodm`

Commands

`A`	Assign dimension of matrix
`B`	Build matrix
`C`	Choose modulus
`D`	Determine value of $\det(A) \pmod{m}$
`E`	Exit
`F`	Form altered matrix

Restrictions $0 < m < 10^9$, $A = [a_{ij}]$ is $n \times n$ with $1 \le n \le 9$, $|a_{ij}| < 10^9$

Algorithm Row operations are performed until the matrix is upper-triangular. After each row operation, the elements of the new matrix are reduced modulo m. The row operations used are of the following two types: (i) Exchange two rows (which multiplies the determinant by -1); (ii) Add an integral multiple of one row to a different row (which leaves the determinant unchanged).

See also DetDem, SimLinDE

EuAlgDem

Function DEMonstrates the EUclidean ALGorithm. If the parameters b and c are specified on the command line, then (b,c) is calculated by using the identities $(b,c)=(c,b)$, $(b,c)=(b+mc,c)$, $(b,0)=|b|$, and then the program terminates. Otherwise an environment is provided in which each remainder is expressed as a linear combination of b and c. In this case one can also toggle between rounding down and rounding to the nearest integer quotient.

Syntax `eualgdem [b c]`

Commands

`PgUp`	Display the top portion of the table
`PgDn`	Display the bottom portion of the table
`b`	Enter a new value of b
`c`	Enter a new value of c
`d`	Round down
`n`	Round to the nearest quotient
`P`	Print the table
`Esc`	Escape from the environment

Restrictions $|b|<10^{18}$, $|c|<10^{18}$

Algorithm The Euclidean Algorithm or Extended Euclidean Algorithm.

See also FastGCD, GCD, SlowGCD

FacTab

Function Constructs a TABle of the least prime FACtor of odd integers from $10N+1$ to $10N+199$.

Syntax `factab`

Commands

`PgUp`	Display the preceding 100 values
`PgDn`	Display the next 100 values
`N`	New N; view table starting at $10N+1$

`Esc`	Escape from the environment

Restrictions Integers not exceeding $10^9 + 189$ (i.e. $0 \le N \le 99999999$).

Algorithm When the program begins execution, it first constructs a list of the odd primes not exceeding $\sqrt{10^9 + 200}$, by sieving. We call these the "small primes." There are 15803 such primes, the last one being 31607. The next prime after this is 31621. When N is specified, the odd integers in the interval $[10N, 10N + 200]$ are sieved by those small primes not exceeding $\sqrt{10N + 200}$; least prime factors are noted as they are found.

See also Factor, GetNextP

Factor

Function FACTORs a given integer n.

Syntax `factor [n]`

Restrictions $|n| < 10^{18}$

Algorithm Trial division. After powers of 2, 3, and 5 are removed, the trial divisors are reduced residues modulo 30.

See also P-1, P-1Dem, Rho, RhoDem

Comments Factors are reported as they are found. The program can be interrupted by touching a key.

FareyTab

Function Constructs a TABle of FAREY fractions of order Q. Fractions are displayed in both rational and decimal form, up to 20 of them at a time.

Syntax `fareytab`

Commands

`PgUp`	View the next 19 smaller entries
`PgDn`	View the next 19 larger entries
`D`	Center the display at a decimal x
`R`	Center the display at a rational number a/q
`P`	Print the table (allowed for $Q \le 46$)
`Esc`	Escape from the environment

Restrictions $1 \le Q < 10^9$

Algorithm If a/q and a'/q' are neighboring Farey fractions of some order Q, say $a/q < a'/q'$, then $a'q - q'a = 1$. By the extended Euclidean algorithm, for given relatively prime a and q we find x and y such that $xq - ya = 1$. Then $q' = y + kq$, $a' = x + ka$ where k is the largest integer such that

$y + kq \le Q$. With a/q given, the next smaller Farey fraction a''/q'' is found similarly. The Farey fractions surrounding a given decimal number x are found by the continued fraction algorithm. Fractions are computed only as needed by the screen or the printer.

FastGCD

Function Times the execution of the Euclidean algorithm in calculating the Greatest Common Divisor of two given integers.

Syntax `fastgcd`

Restrictions $|b| < 10^{18}$, $|c| < 10^{18}$

Algorithm Euclidean algorithm, rounding down.

See also GCD, SlowGCD

FctrlTab

Function Provides a table of $n!$ (mod m). Each screen displays 100 values.

Syntax `fctrltab`

Commands

`PgUp`	View the preceding 100 entries
`PgDn`	View the next 100 entries
`J`	Jump to a new position in the table
`M`	Enter a new modulus
`P`	Print the first 60 lines of the table
`Esc`	Escape from the environment

Restrictions $0 \le n \le 10089$, $0 < m < 10^6$

Algorithm All 10089 values are calculated as soon as m is specified, unless $m < 10089$, in which case only m values are calculated.

FracTab

Function Lists FRACtions $(xa + ya')/(xq + yq')$ in a TABle with entries sorted according to the value of $\arctan y/x$, for $|x| \le Q$, $|y| \le Q$.

Syntax `fractab`

Remarks The data generated reflects some of the properties of Farey fractions.

Restrictions $1 \le a \le q < 10^3$, $1 \le a' \le q' < 10^3$, $Q < 10^3/q$, $Q < 10^3/q'$

See also FareyTab

GCD

Function Calculates the Greatest Common Divisors of two given integers.

Syntax `gcd [b c]`

Restrictions $|b| < 10^{18}$, $|c| < 10^{18}$

Algorithm Euclidean algorithm with rounding to the nearest integer.

See also EuAlgDem, FastGCD, GCDTab, LnComTab, SlowGCD

GCDTab

Function Displays (b, c) for pairs of integers.

Syntax `gcdtab`

Commands

↑	Move up
↓	Move down
←	Move left
→	Move right
`b`	Center table on column b
`c`	Center table on row c
`Esc`	Escape from the environment

Restrictions $|b| < 10^{18}$, $|c| < 10^{18}$

Algorithm Euclidean algorithm.

See also GCD, EuAlgDem, LnComTab

GetNextP

Function Finds the least Prime larger than a given integer x, if $x \leq 10^9$. If $10^9 < x < 10^{18}$, it finds an integer n, $n > x$, such that the interval (x, n) contains no prime but n is a strong probable prime to bases 2, 3, 5, 7, and 11. A rigorous proof of the primality of n can be obtained by using the program ProveP.

Syntax `getnextp [x]`

Restrictions $0 \leq x < 10^{18}$

Algorithm If $0 \leq x \leq 10^9$ then the least prime larger than x is found by sieving. If $10^9 < x < 10^{18}$ then strong probable primality tests are performed.

See also FacTab, ProveP

Hensel

Function Provides a table of solutions of $f(x) \equiv 0 \pmod{p^j}$, in the manner of HENSEL's lemma. All roots (mod p) are found, by trying every residue class. If $f(a) \equiv 0 \pmod{p}$ and $f'(a) \not\equiv 0 \pmod{p}$, then a tower of roots lying above a is displayed. If $f'(a) \equiv 0 \pmod{p}$ then roots lying above a are exhibited only one at a time. Roots (mod p^j) are displayed both in decimal notation and in base p, $a = \sum_{i \ge 1} c_i p^{i-1}$. The user must choose between viewing singular or non-singular roots. The display starts with a non-singular root, if there are any.

Syntax `hensel`

Commands

↑	Lift to larger values of j
↓	Drop to smaller values of j
←	Shift left in the table
→	Shift right in the table
`S`	Switch to singular roots
`N`	Switch to non-singular roots
`D`	Define the polynomial
`p`	Choose the prime modulus
`Esc`	Escape from the environment

Restrictions $2 \le p < 2000$, $p^j \le 10^{18}$, $f(x)$ must be the sum of at most 20 monomials

Algorithm The polynomial $f(x)$ is evaluated at every residue class, and an array is formed of the roots. For each root found, the quantity $f'(x)$ is calculated, in order to determine whether the root is singular or not.

See also PolySolv

HSortDem

Function DEMonstrates the HeapSORT algorithm of J. W. J. Williams, by applying the algorithm to n randomly chosen integers taken from the interval $[0, 99]$. This algorithm is employed in the programs Ind and IndDem.

Syntax `hsortdem`

Restrictions $1 \le n \le 31$

Ind

Function Given g, a, and p, finds the least non-negative ν such that $g^\nu \equiv a \pmod{p}$, if such a ν exists. Thus, if g is a primitive root of p, then $\nu = \operatorname{ind}_g a$.

Syntax `ind [g a p]`

Restrictions $|g| < 10^9,\ |a| < 10^9,\ 1 < p < 10^9,\ (g,p) = 1$

Algorithm First LinCon is used to find $\overline{g} \pmod p$ so that $g\overline{g} \equiv 1 \pmod p$. The number s is taken to be either the integer nearest $\sqrt{p}$ or else 10000, which ever is smaller. A table is made of the residue classes $a\overline{g}^j$ (mod p) for $0 \le j < s$. This table is sorted by the HeapSort algorithm into increasing order. For $j = 0, 1, \ldots$, a search is conducted (by binary subdivisions) to see whether the residue class g^{js} (mod p) is in the table. If a match is found, then $\nu = is + j$. If j reaches p/s without finding a match, then a is not a power of g (mod p). Thus the index is found in time $O(p^{1/2}\log p)$. This method was suggested by D. Shanks.

See also IndDem, IndTab, Power, PowerTab

IndDem

Function DEMonstrates procedure used to compute $\operatorname{ind}_g a \pmod p$.

Syntax `inddem [g a p]`

Restrictions $|g| < 10^9,\ |a| < 10^9,\ 1 < p < 10^9$

Algorithm See the description of the program Ind.

See also Ind, IndTab, Power, PowerTab

IndTab

Function Generates a TABle of INDices of reduced residue classes modulo a prime number p, with respect to a specified primitive root. Also generates a table of powers of the primitive root, modulo p. Up to 200 values are displayed a one time.

Syntax `indtab`

Commands

`PgUp`	View the preceding 200 entries
`PgDn`	View the next 200 entries
`J`	Jump to a new position in the table
`E`	Switch from indices to exponentials
`I`	Switch from exponentials to indices
`M`	Enter a new prime modulus
`B`	Choose a new primitive root to use as the base
`P`	Print table(s)
`Esc`	Escape from the environment

Restrictions $p < 10^4$

Algorithm The least positive primitive root g of p is found using the program PrimRoot. The powers of g modulo p and the indices with respect to g are generated in two arrays.

See also PowerTab, PrimRoot

IntAPTab

Function Creates a TABle with rows indexed by $a \pmod{m}$ and columns indexed by $b \pmod{n}$. The INTersection of these two Arithmetic Progressions is displayed (if it is nonempty) as a residue class $(\text{mod } [m, n])$.

Syntax `intaptab`

Commands

↑	Move up
↓	Move down
←	Move left
→	Move right
`a`	Start at row a
`b`	Sart at column b
`m`	Set modulus m
`n`	Set modulus n
`p`	Print (when table is small enough)
`Esc`	Escape from the environment

Restrictions $m < 10^4$, $n < 10^4$

Algorithm Chinese Remainder Theorem

See also CRT, CRTDem

Comments Reduced residues are written in white, the others in yellow.

JacobDem

Function DEMonstrates the use of quadratic reciprocity to calculate the JACOBi symbol $\left(\frac{P}{Q}\right)$.

Syntax `jacobdem [P Q]`

Restrictions $|P| < 10^{18}$, $0 < Q < 10^{18}$

Algorithm Modified Euclidean algorithm, using quadratic reciprocity.

See also Jacobi, JacobTab

Jacobi

Function Evaluates the JACOBI symbol $\left(\frac{P}{Q}\right)$.

Syntax `jacobi` [`P Q`]

Restrictions $|P| < 10^{18}$, $0 < Q < 10^{18}$

Algorithm Modified Euclidean algorithm, using quadra- tic reciprocity.

See also JacobDem, JacobTab

JacobTab

Function Generates a TABle of values of the JACOBi function, with 200 values displayed at one time.

Syntax `jacobtab`

Commands

`PgUp`	View the preceding 200 entries
`PgDn`	View the next 200 entries
`J`	Jump to a new position in the table
`Q`	Enter a new denominator Q
`P`	Print 500 lines, starting with the top line displayed
`Esc`	Escape from the environment

Restrictions $|P| < 10^{18}$, $0 < Q < 10^{18}$

Algorithm Values are calculated as needed, using the function Jacobi.

See also Jacobi, JacobDem

LinCon

Function Finds all solutions of the LINear CONgruence $ax \equiv b \pmod{m}$.

Syntax `lincon` [`a b m`]

Restrictions $|a| < 10^{18}$, $|b| < 10^{18}$, $0 < m < 10^{18}$

Algorithm The extended Euclidean algorithm is used to find both the number $g = (a, m)$ and a number u such that $au \equiv g \pmod{m}$. If $g \nmid b$ then there is no solution. Otherwise, the solutions are precisely those x such that $x \equiv c \pmod{m/g}$ where $c = ub/g$.

See also LnCnDem

LnCnDem

Function DEMonstrates the method used to find all solutions to the LiNear CoNgruence $ax \equiv b \pmod{m}$.

Syntax `lncndem` [`a b m`]

Restrictions $|a| < 10^{18}$, $|b| < 10^{18}$, $0 < m < 10^{18}$

Algorithm See the description given for LinCon.

See also LinCon

LnComTab

Function Creates a TABle of the LiNear COMbinations $bx + cy$ of b and c, with columns indexed by x and rows indexed by y.

Syntax `lncomtab`

Commands

↑	Move up
↓	Move down
←	Move left
→	Move right
`x`	Left column is x
`y`	Bottom row is y
`b`	Set value of b
`c`	Set value of c
`Esc`	Escape from the environment

Restrictions $|b| < 10^9$, $|c| < 10^9$, $|x| < 10^9$, $|y| < 10^9$

See also GCD, GCDTab, EuAlgDem,

Lucas

Function Calculates the LUCAS functions $U_n, V_n \pmod m$. Here the U_n are generated by the linear recurrence $U_{n+1} = aU_n + bU_{n-1}$ with the initial conditions $U_0 = 0$, $U_1 = 1$. The V_n satisfy the same linear recurrence, but with the initial conditions $V_0 = 2$, $V_1 = a$.

Syntax `lucas` [`n` [`a b`] `m`] If n, m are specified on the command line, but not a, b, then by default $a = b = 1$.

Restrictions $0 \le n < 10^{18}$, $|a| < 10^{18}$, $|b| \le 10^{18}$, $0 < m \le 10^{18}$

Algorithm To calculate $U_n \pmod m$, the pair of residue classes $U_{k-1}, U_k \pmod m$ is determined for a sequence of values of k, starting with $k = 1$. If this pair is known for a certain value of k, then it can be found with k replaced by $2k$, by means of the *duplication formulae*

$$U_{2k-1} = U_k^2 + bU_{k-1}^2,$$
$$U_{2k} = 2bU_{k-1}U_k + aU_k^2.$$

This is called “doubling.” Alternatively, the value of k can be increased by 1 by using the defining recurrence. This is called “sidestepping.” By repeatedly doubling, with sidesteps interspersed as appropriate, eventually $k = n$.

To calculate $V_n \pmod m$, the pair $V_k,\ V_{k+1}$ of residue classes (mod m) is determined for a sequence of values of k, starting with $k = 0$. The duplication formulae are now

$$V_{2k} = V_k^2 - 2(-b)^k,$$
$$V_{2k+1} = V_k V_{k+1} - a(-b)^k.$$

Instead of sidestepping separately, an arithmetic economy is obtained by doubling with sidestep included by means of the formulae

$$V_{2k+1} = V_k V_{k+1} - a(-b)^k,$$
$$V_{2k+2} = V_{k+1}^2 - 2(-b)^{k+1}.$$

By employing these transformations we eventually reach $k = n$.

The k that arise have binary expansions that form initial segments of the binary expansion of n, in the same manner as in the alternative powering algorithm discussed in the program PwrDem2.

The system of calculation here is superior to that found in the Fifth Edition of NZM, where the sidestep formula involves division by 2 and is therefore appropriate only for odd moduli.

See also LucasDem, LucasTab, PwrDem2

Comments If $a = b = 1$ then U_n, V_n are the familiar Fibonacci and Lucas sequences F_n, L_n, respectively.

LucasDem

Function DEMonstrates the method used to calculate the LUCAS functions U_n, $V_n \pmod m$.

Syntax `lucasdem [n [a b] m]`

Restrictions $0 \le n < 10^{18},\ |a| < 10^{18},\ |b| < 10^{18},\ 0 < m < 10^{18}$

Algorithm See the description given for the program Lucas.

See also Lucas, LucasDem, PwrDem2

LucasTab

Function Generates a TABle of values of the LUCAS functions $U_n, V_n \pmod m$.

Syntax `lucastab`

Commands

`PgUp`	Display the preceding 100 values
`PgDn`	Display the next 100 values
`U`	Switch from V to U
`V`	Switch from U to V
`n`	Move to a screen with n on the top line
`a`	Choose a new value for the parameter a
`b`	Choose a new value for the parameter b
`M`	Choose a new modulus m
`P`	Print the initial 60 rows of the table $(0 \le n \le 599)$
`Esc`	Escape from the environment

Restrictions $0 \le n < 10^6$, $|a| < 10^6$, $|b| < 10^6$, $0 < m < 10^6$

See also Lucas, LucasDem

Mult

Function MULTiplies residue classes. If a, b, and m are given with $m > 0$, then c is found so that $c \equiv ab \pmod{m}$ and $0 \le c < m$.

Syntax `mult [a b m]`

Restrictions $|a| < 10^{18}$, $|b| < 10^{18}$, $0 < m < 10^{18}$

Algorithm If $m \le 10^9$ then ab is reduced modulo m. If $10^9 < m \le 10^{12}$ then we write $a = a_1 10^6 + a_0$, and compute $a_1 b 10^6 + a_0 b$ modulo m, with reductions modulo m after each multiplication. Thus all numbers encountered have absolute value at most 10^{18}. If $10^{12} < m < 10^{18}$ then we write $a = a_1 10^9 + a_0$, $b = b_1 10^9 + b_0$; we compute ab/m in floating-point real arithmetic and let q be the integer nearest this quantity; we write $q = q_1 10^9 + q_0$; $m = m_1 10^9 + m_0$. Then

$$ab - qm = ((a_1 b_1 - q_1 m_1)10^9 + a_1 b_0 + a_0 b_1 - q_1 m_0 - q_0 m_1)10^9 + a_0 b_0 - q_0 m_0.$$

The right hand side can be reliably evaluated, and this quantity has absolute value less than m. If it is negative we add m to it to obtain the final result. The assumption is that the machine will perform integer arithmetic accurately for integers up to $4 \cdot 10^{18}$ in size. The object is to perform congruence arithmetic with a modulus up to 10^{18} without introducing a full multiprecision package.

See also MultDem1, MultDem2, MultDem3

MultDem1

Function DEMonstrates the method employed by the program MULT when $10^9 < m < 10^{12}$.

Syntax `multdem1`

Restrictions $|a| < 10^{18}$, $|b| < 10^{18}$, $0 < m < 10^{18}$

Algorithm See Problem *21, Section 2.4, p. 83, of the Fifth Edition of NZM.

See also Mult, MultDem2, MultDem3

MultDem2

Function DEMonstrates the method used by the program MULT when $10^{12} < m < 10^{18}$.

Syntax `multdem2`

Restrictions $|a| < 10^{18}$, $|b| < 10^{18}$, $0 < m < 10^{18}$

Algorithm See the description given for the program Mult.

See also Mult, MultDem1, MultDem3

MultDem3

Function DEMonstrates the method used by the program MULT, in which the methods of MultDem1 and MultDem2 are merged.

Syntax `multdem3`

Restrictions $|a| < 10^{18}$, $|b| < 10^{18}$, $0 < m < 10^{18}$

Algorithm See the description given for the program Mult.

See also Mult, MultDem1, MultDem2

Order

Function Calculates the ORDER of a reduced residue class $a \pmod{m}$. That is, it finds the least positive integer h such that $a^h \equiv 1 \pmod{m}$.

Syntax `order [a m [c]]`

Restrictions $|a| < 10^{18}$, $0 < m < 10^{18}$, $0 < c < 10^{18}$

Algorithm The parameter c should be any known positive number such that $a^c \equiv 1 \pmod{m}$. For example, if m is prime then one may take $c = m - 1$. If a value of c is not provided by the user, or if the value provided is incorrect, then the program assigns $c = \text{Carmichael}(m)$. (This involves factoring m by trial division.) Once c is determined, then c is factored by trial division. Prime divisors of c are removed, one at a time, to locate the smallest divisor d of c for which $a^d \equiv 1 \pmod{m}$. This number is the order of a modulo m.

See also OrderDem

OrderDem

Function DEMonstrates the method used to calculate the order of a reduced residue class $a \pmod{m}$.

Syntax `order` [`a m` [`c`]]

Restrictions $|a| < 10^{18}$, $0 < m < 10^{18}$, $0 < c < 10^{18}$

Algorithm See the description given for the program Order.

See also Order, OrderTab

OrderTab

Function Constructs a TABle of the ORDER of a modulo m.

Syntax `ordertab`

Commands

$\rightarrow$	Display the next columns
$\downarrow$	Display the next 20 rows
$\leftarrow$	Display the preceding columns
$\uparrow$	Display the preceding 20 rows
`a`	Display column a
`m`	Display row m
`P`	Print a portion of the table
`Esc`	Escape from the environment

Restrictions $-9999 \le a \le 9985$, $1 \le m \le 9999$

See also Order, OrderDem

P–1

Function Factors a number n using the Pollard $p-1$ method.

Syntax `p-1` [`n` [`a`]] If n is specified on the command line, but not a, then by default $a = 2$.

Restrictions $1 < n < 10^{18}$, $1 < a < 10^{18}$

Algorithm The powering algorithm is used to calculate $a^{k!} \pmod{n}$ for increasingly large k, in the hope that a k will be found such that $1 < (a^{k!} -$

1, $n) < n$. This method is generally fast for those n with a prime factor p such that $p-1$ is composed only of small primes.

See also P-1Dem, Rho, RhoDem, Factor

P–1Dem

Function Demonstrates the method used by the Pollard $p-1$ factoring scheme.

Syntax `p-1dem`

Restrictions $1 < n < 10^{18}$, $1 < a < 10^{18}$

Algorithm See the description given for the program P-1.

See also P-1

PascalsT

Function Constructs a table of PASCAL'S Triangle $\binom{n}{k} \pmod m$. Rows are indexed by n, columns by k. Up to 20 rows and 18 columns are displayed at one time.

Syntax `pascalst`

Commands

↑	Display the preceding 20 rows
↓	Display the next 20 rows
←	Display the preceding 20 columns
→	Display the next 20 columns
`T`	Move to the top of the triangle
`M`	Choose a new modulus
`Esc`	Escape from the environment

Restrictions $0 \le k \le n < 10^4$, $0 < m < 10^3$

Algorithm The rows are calculated inductively by the recurrence $\binom{n}{k-1} + \binom{n}{k} = \binom{n+1}{k}$. The entire nth row is calculated, where n is the top row on the current screen. Other entries in the screen are calculated from the top row.

Phi

Function Calculates the Euler PHI function of n.

Syntax `phi` [`n`]

Restrictions $1 \le n < 10^{18}$

Algorithm The canonical factorization of n is found by trial division, and then $\phi(n)$ is found by means of the formula $\phi(n) = \prod_{p^\alpha \| n} p^{\alpha-1}(p-1)$.

Pi

Function Determines the number $\pi(x)$ of primes not exceeding an integer x.

Syntax `pi [x]`

Restrictions $2 \le x10^6$

Algorithm Primes up to 31607 are constructed, by sieving. These primes are used as trial divisors, to sieve intervals of length 10^4 until x is reached.

Comments This program would run perfectly well up to 10^9, but as the the running time is roughly linear in x, the smaller limit is imposed to avoid excessive running times. For faster methods of computing $\pi(x)$, see the following papers.
J. C. Lagarias, V. S. Miller, and A. M. Odlyzko, *Computing* $\pi(x)$: *The Meissel-Lehmer method*, Math. Comp. **44** (1985), 537–560.
J. C. Lagarias and A. M. Odlyzko, *New algorithms for computing* $\pi(x)$, Number Theory: New York 1982, D. V. Chudnovsky, G. V. Chudnovsky, H. Cohn and M. B. Nathanson, eds., Lecture Notes in Mathematics 1052, Springer-Verlag, Berlin, 1984, pp. 176–193.
J. C. Lagarias and A. M. Odlyzko, *Computing* $\pi(x)$: *an analytic method*, J. Algorithms **8** (1987), 173–191.

PolySolv

Function Finds all solutions of a given polynomial congruence $P(x) \equiv 0 \pmod{m}$.

Syntax `polysolv`

Commands

`C`	Count the zeros
`D`	Define the polynomial
`M`	Choose the modulus
`Esc`	Escape from the environment

Restrictions $1 \le m < 10^4$, $P(x)$ must be the sum of at most 20 monomials, only the first 100 zeros found are displayed on the screen

Algorithm The polynomial is evaluated at every residue class modulo m.

See also SqrtModP

Comments The running time here is roughly linear in m. When m is large there is a much faster way. By the Chinese Remainder Theorem it is enough to consider primepower values of m. By Hensel's lemma, this in turn can be reduced to the consideration of prime moduli. In the case of a prime modulus p, the roots of $P(x)$ modulo p can be found by calculating $(P(x), (x-a)^{(p-1)/2}-1)$ for various values of a. Here the gcd being calculated is that of two polynomials defined mod p. In the first step of the Euclidian algorithm, the remainder when $(x-a)^{(p-1)/2}-1$ is divided by $P(x)$ should be calculated by applying the powering algorithm to determine $(x-a)^{(p-1)/2}$ $(\text{modd}\, p,\, P(x))$. This approach extends to provide an efficient method of determining the factorization of $P(x) \pmod p$. For more information, see David G. Cantor and Hans Zassenhaus, *A new algorithm for factoring polynomials over finite fields*, Math. Comp. **36** (1981), 587–592.

Power

Function Computes $a^k \pmod m$ in the sense that it returns a number c such that $0 \le c < m$ and $c \equiv a^k \pmod m$.

Syntax `power [a k m]`

Restrictions $|a| < 10^{18}$, $0 \le k < 10^{18}$, $0 < m < 10^{18}$

Algorithm Write k in binary, say $k = \sum_{j \in \mathcal{J}} 2^j$. The numbers $a^{2^j} \pmod m$ are constructed by repeated squaring; whenever a $j \in \mathcal{J}$ is encountered, the existing product is multiplied by the factor a^{2^j}.

See also PowerTab, PwrDem1a, PwrDem1b, PwrDem2

PowerTab

Function Constructs a TABle of POWERs $a^k \pmod m$.

Syntax `powertab`

Commands

$\uparrow$	Display the preceding 20 rows
$\downarrow$	Display the next 20 rows
$\leftarrow$	Display the preceding rows
$\rightarrow$	Display the next rows
`B`	Change the base
`E`	Move to a new exponent
`M`	Change the modulus
`P`	Print the first 54 lines of the table
`Esc`	Escape from the environment

Restrictions $|a| < 10^9$, $1 \le k < 10^9$, $1 \le m < 10^9$

Algorithm The first entry in each row is computed by the powering algorithm. Then the remaining entries on the screen are determined inductively.

See also Power, PowerDem

PrimRoot

Function Finds the least primitive root g of a prime number p, such that $g > a$.

Syntax `primroot` [p [a]] If p is specified on the command line but not a, then by default $a = 0$.

Restrictions $2 \le p < 10^{18}$, $|a| < 10^{18}$

Algorithm The prime factors $q_1, q_2, \ldots, q_r$ of $p-1$ are found by trial division. Then g is a primitive root of p if and only if both $g^{p-1} \equiv 1 \pmod p$ and $g^{(p-1)/q_i} \not\equiv 1 \pmod p$ for all i, $1 \le i \le r$. When a g is found that satisfies these conditions, not only is g a primitive root of p, but also the primality of p is rigorously established. The algorithm employed by the program ProveP proceeds along these lines, but with some short cuts.

See also Order, OrderDem, ProveP

Comments This program provides a user interface for a function of the same name in the unit NoThy. To see how the algorithm is implemented, inspect the file `nothy.pas`.

ProveP

Function PROVEs that a given number p is Prime.

Syntax `provep` [p]

Restrictions $2 \le p < 10^{18}$

Algorithm Trial division is applied to $p-1$. Whenever a prime factor q of $p-1$ is found, say $q^k \| (p-1)$, attempts are made to find an a such that $a^{p-1} \equiv 1 \pmod p$ but $(a^{(p-1)/q} - 1, p) = 1$. Suppose that such an a is found, and that $p' | p$. Let d denote the order of a modulo p'. Then $d | (p-1)$ but $d \nmid (p-1)/q$, and hence $q^k \| d$. But by Fermat's congruence $d | (p'-1)$, and hence it can be asserted that $q^k | (p'-1)$ for every prime factor p' of p. In other words, all prime factors p' of p are $\equiv 1 \pmod{q^k}$. If, for a given q, 200 unsuccessful attempts are made to find an admissible a, then presumably p is composite, and the program quits. Otherwise, the numbers q^k found are multiplied together to form a product s. Every prime factor p' of p is $\equiv 1 \pmod s$. If $s > \sqrt{p}$ then there can be at most one such prime, and the proof is complete. If $p^{1/3} < s \le p^{1/2}$ then there can be at most two such primes, say $p = p_1 p_2$. Write p_i in base s, $p_i = r_i s + 1$. Then $p = r_1 r_2 s^2 + (r_1 + r_2)s + 1$, and the coefficients of this polynomial in s can be found by expanding p in base s, say

$p = c_2 s^2 + c_1 s + 1$. Then r_1 and r_2 are roots of the quadratic equation $(x - r_1)(x - r_2) = x^2 - c_1 x + c_2$, and hence the discriminant $c_1^2 - 4c_2$ must be a perfect square. In the unlikely event that this quantity is a perfect square, we are led to a factorization of p; otherwise we have a proof that p is prime.

If a point is reached at which it would take less time to test p for divisibility by numbers $d \equiv 1 \pmod s$, $d \le \sqrt{p}$ than has already been spent trying to factor $p-1$, then the program automatically switches to this latter approach.

The trial division of $p-1$ can be interrupted by touching a key, and the user can then supply a prime factor q of the remaining unfactored portion. The user is responsible for verifying that q is prime.

By this method we see that proving the primality of p is no harder than factoring $p-1$, and that for many p it is easier. Further methods of proving primality have been developed that are faster than the best known factoring methods. The mathematics exploited by these methods is much more sophisticated. For more precise information, consult the following papers.

A. O. L. Atkin and F. Morain, *Elliptic curves and primality proving*, Math. Comp. **61** (1993), 29–68 .

A. K. Lenstra and H. W. Lenstra, Jr., *Algorithms in number theory*, Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, pp. 673–715.

PwrDem1a

Function DEMonstrates the powering algorithm.

Syntax `pwrdem1a` [`a k m`]

Restrictions $|a| < 10^{18}$, $0 \le k < 10^{18}$, $0 < m < 10^{18}$

Algorithm See the description given for the program Power.

See also Power, PwrDem1b, PwrDem2

PwrDem1b

Function An alternative DEMonstration of the powering algorithm.

Syntax `pwrdem1b` [`a k m`]

Restrictions $|a| < 10^{18}$, $0 \le k < 10^{18}$, $0 < m < 10^{18}$

Algorithm See the description given for the program Power.

See also Power, PwrDem1a, PwrDem2

PwrDem2

Function DEMonstrates an alternative powering algorithm.

Syntax `pwrdem2 [a k m]`

Restrictions $|a| < 10^{18}$, $0 \le k < 10^{18}$, $0 < m < 10^{18}$

Algorithm A sequence of powers of a is generated, in which the binary expansions of the exponents form initial segments of the binary expansion of k. For example, if $k = 10111$ in binary, then (with all exponents written in binary) we start with a^1, square to form a^{10}, square again to form a^{100}, multiply by a to form a^{101}, square this to form a^{1010}, multiply by a to form a^{1011}, square this to form a^{10110}, and finally multiply by a to form a^{10111}. Of course all multiplications are carried out modulo m. In the original method used by the program Power, the binary expansions of the exponents form terminal segments of the binary expansion of k. The number of multiplications is exactly the same in the two methods, but this alternative method has an advantage in situations in which multiplication by a is fast for some reason. For example, in powering a matrix A, multiplication by A is fast if A is sparse. Similarly, in computing $P(x)^k$, multiplication by $P(x)$ is fast if $P(x)$ has few monomial terms. The repeated doubling from the top down seen here is also appropriate to the calculation of solutions of linear recurrences.

See also Power, PwrDem1a, PwrDem1b, LucasDem

QFormTab

Function Generates a TABle of all reduced binary Quadratic FORMs $f(x, y) = ax^2, bxy + cy^2$ of given discriminant. These forms are reduced only in the sense defined in §3.5 of NZM. Hence if $d > 0$ then the reduced forms are not necessarily inequivalent. For each form, the content (a, b, c) is calculated.

Syntax `qformtab`

Commands

`PgUp`	Display the preceding 20 rows
`PgDn`	Display the next 20 rows
`d`	Choose a new discriminant
`P`	Print the first 600 lines of the table
`Esc`	Escape from the environment

Restrictions $|d| < 10^6$, at most 5000 forms are displayed

Algorithm Detailed search for all triples satisfying the definition. Thus the running time is essentially linear in $|d|$. This program could run for $|d|$ up to

10^9, but the stricter limit is imposed to avoid excessive running times. For faster methods, see the discussion of the program ClaNoTab.

See also ClaNoTab, Reduce

R2D

Function Converts a Rational number a/q TO Decimal form, or in base b. If a and q (and optionally b) are entered on the command line then a screenful of digits is given and the program terminates. Otherwise the first 10^9 digits may be viewed, 1000 at a time. The base b can be changed; the default is $b = 10$. When $b > 10$ the 'digit' 10 is represented by A, ..., 15 by F. (When $b = 16$ this is the standard hexadecimal convention.) The digits are initially displayed in yellow, but the periodicity of the expansion can be highlighted, in which case alternate cycles are displayed in green and red. In this latter mode the length $T(a/q)$ of the aperiodic 'tail' and the length $C(a/q)$ of the 'cycle' are also displayed. (These values also depend on b.)

Commands

`PgUp`	Move the window up one screenful
`PgDn`	Move the window down one screenful
`J`	Jump to a new position in the table of digits
`a`	enter a numerator a
`q`	enter a denominator q
`B`	enter a base b
`C`	highlight or Conceal the Cycles
`P`	Print the first 2997 digits (1 page)
`Esc`	Escape from the environment

Syntax `r2d [a q [b]]`

Restrictions $1 \le a < q \le 10^9$, $2 \le b \le 16$

Algorithm Remainders r_i are uniquely determined by the relations $0 \le r_i < q$, $r_i \equiv ab^i \pmod q$. Digits d_i are found from the identity $br_i = d_iq + r_{i+1}$. Assume that $(a, q) = 1$. If there is an integer k such that $q \mid b^k$ then let k be the least such integer; the expansion terminates after exactly k digits. Otherwise, the length $T(a/q)$ of the aperiodic tail is the least non-negative integer t such that the denominator q' of ab^t/q is relatively prime to b. The length $C(a/q)$ of the cycle is the order of b modulo q'.

See also D2R, Order

Reduce

Function REDUCEs a binary quadratic form $f(x, y) = ax^2+bxy+cy^2$. If the three coefficients are given on the command line, then a reduced form $g(x, y)$

is found, with g equivalent to f. The discriminant d of these forms is also reported. A proper representation of a by g is also noted, and then the program terminates. If the coefficients are not given on the command line, then an environment for manipulating forms is entered. When a form is being reduced in this environment, a chain of equivalences is displayed, along with the matrix M that gives the equivalence, and the operation S or T^m that was applied to derive the new form from that in the preceding row of the table. Here $S = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ and $T = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$. The user also has the option of applying the operations S, T, and T^{-1}, one at a time. The table will hold up to 500 forms.

In the case that $d > 0$, the form is reduced only to the extent that $|a| < b \le |a| < |a|$ or $0 \le b \le |a| = |c|$, and consequently two reduced forms may be equivalent.

Syntax `reduce [a b c]`

Restrictions $|a| < 10^{18}$, $|b| < 10^{18}$, $|c| < 10^{18}$

Commands

`PgUp`	Display the preceding 6 rows
`PgDn`	Display the next 6 rows
`a`	Enter a new coefficient a
`b`	Enter a new coefficient b
`c`	Enter a new coefficient c
`R`	Reduce the form at the bottom of the table
`S`	Apply the transformation S
`T`	Apply the transformation T
`I`	Apply the transformation T^{-1}
`M`	Toggle between displaying $M{:}g \to f$ and $M{:}f \to g$
`P`	Print the table
`Esc`	Escape from the environment

See also ClaNoTab, QFormTab

ResComp

Function Compares residues $x \pmod{m}$ with $x \pmod{n}$.

Syntax `rescomp`

Restrictions $|x| < 10^9$, $1 \le m < 10^9$, $1 \le n < 10^9$

Algorithm Division algorithm to find remainders.

See also CRT, CRTDem, IntAPTab

Rho

Function Factors a given composite integer n by using Pollard's RHO method. This program should only be applied to numbers that are already known to be composite; if it is applied to a prime number then it will run endlessly without reaching any conclusion. The program can be interrupted by touching any key on the keyboard.

Syntax `rho` [`n` [`c`]] If n is specified on the command line but not c, then $c = 1$ by default.

Restrictions $1 < n < 10^{18}$, $|c| < 10^{18}$

Algorithm Let $u_0 = 0$, and for $i \ge 0$ let $u_{i+1} = u_i^2 + c$. The u_i are calculated modulo n, and for each i the quantity $(u_{2i} - u_i, n)$ is determined, in the hope of finding a proper divisor of n. The numbers u_i are not stored: At any one time only u_i and u_{2i} are known. If a proper divisor is found, it is not necessarily prime, and if it is prime it is not necessarily the least prime divisor of n. Various values of c may be used, but $c = 0$ and $c = -2$ should be avoided.

See also RhoDem, P-1, P-1Dem, Factor

RhoDem

Function DEMonstrates the Pollard RHO factoring scheme.

Syntax `rhodem` [`n`]

Restrictions $1 < n < 10^{18}$, $|c| < 10^{18}$

Algorithm See description given for the program Rho.

See also Rho, P-1, P-1Dem, Fac

RSA

Function Demonstrates RSA encryption. Plaintext is taken from an ASCII file with the default extension .txt. The ASCII code of a printable keyboard character lies between 32 and 126. By subtracting 32 we obtain a number between 0 and 94. In this way each character is associated with a 2-digit code. The code 95 is used as an end-of-line marker. The codes are concatenated k at a time to represent residues modulo m where $10^{2k} \le m < 10^{2k+2}$. Ciphertext can be saved as a sequence of residues to a file with the default extension .rsa. Public RSA parameters can be entered from the keyboard or read from a file with the default extension .pub. A line in the source file that begins with the symbol '%' is treated as a comment, and is passed to the destination file without alteration. When saving, the encryption history is included as a comment. This implementation is not secure because numbers $m < 10^{18}$ are easily factored.

Syntax `rsa`

Commands

↑	Move the window up one screenful
↓	Move the window down one screenful
`V`	set the Variables
`L`	Load plain or cipher text
`E`	Encipher
`D`	Decipher
`C`	convert from text or residues to Codes
`T`	convert from codes to Text
`R`	convert from codes to Residues
`S`	Save
`P`	Print
`Esc`	Escape from the environment

Restrictions $100 \le m < 10^{18}$, $0 < k < 10^{18}$, $0 < k' < 10^{18}$

Algorithm Each residue class $a \pmod{m}$ is replaced by $b \equiv a^k \pmod{m}$. To decipher, replace b by $b^{k'} \pmod{m}$ where $kk' \equiv 1 \pmod{\phi(m)}$.

See also RSAPars

RSAPars

Function Aids in forming RSA PARameterS. The private exponent k' is chosen first, and then m is constructed by choosing primes p such that $(p-1, k') = 1$. When m has been determined, the public exponent is derived. The public parameters m and k can be saved to a file, with the default extension .pub.

Syntax `rsapars`

Restrictions $1 < k' < 10^{18}$, k' odd, $100 \le m < 10^{18}$, m squarefree.

Algorithm Primes $p < 10^9$ are found (rigorously) by sieving. Primes $10^9 < p < 10^{18}$ are found (unrigorously) by applying strong pseudoprime tests to bases 2, 3, 5, 7, and 11. Once k' and the prime factors of m have been chosen, the public exponent k is determined by solving the linear congruence $kk' \equiv 1 \pmod{\phi(m)}$.

See also RSA, LinCon

SimLinDE

Function Gives a complete parametric representation of the solutions to a system of SIMultaneous LINear Diophantine Equations $A\mathbf{x} = \mathbf{b}$. The user may request that the calculations be displayed.

Syntax `simlinde`

Restrictions A is $m \times n$ where $1 \le m \le 10$, $1 \le n \le 10$, all numbers occurring must have absolute value not exceeding 10^{18}

Algorithm Row operations and changes of variable are performed until the system is in diagonal form. The full Smith normal form is not reached. This method is prone to overflow. The program as written makes no special effort to avoid overflow, but reports when it has occurred.

SlowGCD

Function Times the calculation of the greatest common divisor of two numbers b and c, when only the definition is used. The only purpose in this is to provide a comparison with FastGCD.

Syntax `slowgcd`

Restrictions $1 \le b < 10^9$, $1 \le c < 10^9$

Algorithm For each d, $1 \le d \le \min(|b|, |c|)$, trial divisions are made to determine whether $d|b$ and $d|c$. A record is kept of the largest such d found. Since the running time is essentially linear in $\min(|b|, |c|)$, only small arguments should be used.

See also FastGCD, GCD

SPsP

Function Executes the Strong PseudoPrime test base a to the number m. This provides a rigorous proof of compositeness. If m survives such a test then it is not necessarily prime, but it is called a "probable prime" because pseudoprimes (i.e., composite probable primes) seem to form a sparse set.

Syntax `spsp [[a] m]` If m is specified on the command line, but not a, then by default $a = 2$.

Restrictions $|a| < 10^{18}$, $2 < m < 10^{18}$

Algorithm The strong pseudoprime test, as invented by John Selfridge and others. For a full description see NZM, p. 78.

See also SPsPDem, ProveP

SPsPDem

Function DEMonstrates the Strong PSeudoPrime test.

Syntax `spsp [[a] m]` If m is specified on the command line, but not a, then by default $a = 2$.

Restrictions $|a| < 10^{18}$, $2 < m < 10^{18}$

See also SPsP, ProveP

SqrtDem

Function DEMonstrates the calculation executed by the program SqrtModP.

Syntax `sqrtdem [a p]`

Restrictions $|a| < 10^{18}$, $2 \le p < 10^{18}$

Algorithm See the description given for the program SqrtModP

See also SqrtModP

SqrtModP

Function Calculates the SQuareRooT Modulo a given Prime number p. If the congruence $x^2 \equiv a \pmod p$ has a solution, then the unique solution x such that $0 \le x \le p/2$ is returned.

Syntax `sqrtmodp [a p]`

Restrictions $|a| \le 10^{18}$, $2 \le p \le 10^{18}$

Algorithm Uses the RESSOL algorithm of Dan Shanks. This is described in §2.9 of NZM. A different method, which depends on properties of the Lucas sequences, has been given by D. H. Lehmer, *Computer technology applied to the theory of numbers*, Studies in Number Theory, W. J. LeVeque, ed., Math. Assoc. Amer., Washington, 1969, pp. 117–151.

See also SqrtDem

SumsPwrs

Function Finds all representations of n as a sum of s k-th powers, and counts them in various ways.

Syntax `sumspwrs [n s k]`

Restrictions $1 \le n < 10^{11}$, $2 \le s \le 75$, $2 \le k \le 10$

Algorithm After $s-1$ summands have been chosen, a test is made as to whether the remainder is a k-th power. Summands are kept in monotonic order; the multiplicity is recovered by computing the appropriate multinomial coefficient. In some cases, such as sums of two squares, much faster methods exist for finding all representations.

See also WrngTab

WrngTab

Function Creates a TABle of the number $r(n)$ of representations of $n = \sum_{i=1}^{s} x_i^s$ as a sum of s k-th powers, as in WaRiNG's problem. If $k > 2$ then the x_i are non-negative, but for $k = 2$ the x_i are arbitrary integers.

Syntax `wrngtab`

Commands

`PgUp`	Move up
`PgDn`	Move down
`s`	Set s, the number of summands
`k`	Set k, the exponent
`N`	Start the table at $10n$
`P`	Print the table
`Esc`	Escape from the environment

Restrictions $1 \le s \le 75$, $2 \le k \le 10$, $1 \le n \le 10^{11}$

Algorithm Search for representations, with summands in monotonic order. The multiplicity of a representation is recovered by multiplying by the appropriate multinomial coefficient.

See also SumsPwrs