A FEASIBILITY STUDY OF A DATA ENCRYPTION ALGORITHM FOR USE IN AN ELECTRONIC FUNDS TRANSFER SYSTEM

BY

PETER JOHN BLATMAN

A report
submitted in partial fulfillment of the requirements
for the degree of
Master of Business Administration

JUNE 12, 1978

GRADUATE SCHOOL OF BUSINESS ADMINISTRATION
UNIVERSITY OF CALIFORNIA
BERKELEY

Supervisor Austin Hoggatt

TABLE OF CONTENTS

ABSTRACT.....ii

PART I. EFT SYSTEMS AND PUBLIC-KEY CRYPTOSYSTEMS

I. Operation.....1

II. Privacy.....3

III. Signatures.....4

PART II. THE DATA ENCRYPTION ALGORITHM

I. Operation.....6

II. Key Generation.....8

III. Security.....10

IV. Implementation.....12

V. Timing Tests.....14

VI. Conclusions.....16

REFERENCES.....17

APPENDIX I - Examples of Data Encryption and Signatures....18

APPENDIX II - Timing Test Results and Timing Plots.....22

APPENDIX III - Encryption Time Regression Runs.....38

APPENDIX IV - Program Listings.....52

ABSTRACT

This report examines the working characteristics of a data encryption algorithm to be used in the construction of an Electronic Funds Transfer (EFT) system. The algorithm itself is embedded as an integral component in a public-key cryptosystem which serves the purpose of providing privacy of communication and verification of sender identity within the framework of the EFT system.

Necessary and desirable characteristics of a data encryption algorithm are considered, with a concurrent analysis of the algorithm at hand. An actual computer implementation of the algorithm is presented and the results of encryption timing tests are discussed. A characterization of encryption times as a function of key parameters is accomplished through the application of multiple regression analysis to the results of the timing tests.

PART I. EFT SYSTEMS AND PUBLIC-KEY CRYPTOSYSTEMS

OPERATION

If an EFT system is to be usable and effective, it must preserve two important properties of current "paper" systems: (a) Transmitted messages are private and (b) messages can be "signed" in the sense that the identity of the sender can be verified by the recipient. The concept of a "Public-Key Cryptosystem" (PKC) [Diffie and Hellman, 1976] provides a solution to the problem.

In a PKC, each user places in a public file an encryption procedure E. The user does not make public the details of his corresponding decryption procedure D. For use in a PKC, these procedures must have the following properties [Rivest, 1978]:

(a) Deciphering an enciphered form of message M yields M itself. Expressed functionally,

(1) $D(E(M)) = M$

(b) E and D are both easy to apply.

(c) Publicly revealing E does not provide a practical way of computing D.

(d) E and D are inverse procedures of one another. i.e., in addition to (1) above:

(2) $E(D(M)) = M$

In actual practice, D and E are not actual procedural specifications, but rather numerical keys

to be used in conjunction with a publicly known algorithm. The same algorithm will encrypt a clear message using E as the key, or decrypt an enciphered message using D as the key.

For example, if user A wishes to transmit a message to user B, he simply encrypts the message using B's public encryption key E_B and transmits this enciphered message to B. B can then decrypt the message by using his secret decryption key D_B. As mentioned in (c), knowledge of E_B does not give any feasible method of deducing D_B which is known only to user B.

PRIVACY

Property (c) of PKC's assures that only the intended recipient of a message will be able to decrypt it. Since encryption and decryption are performed at the ends of the transmission line rather than centrally, no message ever appears "in the clear" over the lines. Therefore, an interloper who manages to tap a transmission line will intercept only meaningless ciphertext. Considering the sensitive nature of the information communicated over an EFT system, such privacy is of the utmost importance.

In addition, the nature of the PKC scheme alleviates the need for secretive key distribution methods. A user simply generates his encryption and decryption keys locally, and posts the encryption key to a public file, accessible to all other users in the system. Such a scheme would be readily amenable to having the encryption/decryption algorithm embodied in special purpose high-speed chips at each user installation.

SIGNATURES

Given that only the intended recipient of an encrypted message can decrypt it, how can the identity of the sender be verified, and more importantly, how can it be proven that the recipient did not forge the message himself? In existing "paper" systems, the recipient of a signed message has proof that the message was written by the sender. This property is stronger than mere authentication where the recipient can merely verify that the message was transmitted by the sender.

An electronic signature must be message-dependent as well as signer-dependent. Otherwise, the recipient could modify the message and claim the signature as proof of origin, or simply append the signature electronically (character concatenation) to any desired message.

Property (d) of PKC's given in the previous section allows for the implementation of message-dependent signatures. It is this property which allows a users secret decryption procedure to be applied to an unenciphered message. It is the application of the secret procedure which "signs" the message.

For example, suppose user B wishes to send user A

a "signed" message. He first computes his "signature" S for the message M using his secret decryption procedure, D_B as follows:

(3) $S = D_B(M)$

This is permissible according to property (d) of PKCs as previously mentioned. He then encrypts S using A's publicly known enciphering procedure E_A, and sends the resulting message $E_A(S)$ to user A.

A first decrypts the ciphertext by applying his secret decryption procedure D_A. This yields the ciphertext signature S. Formally:

(4) $D_A(E_A(S)) = S$

Given that A assumes that B has transmitted the message, A applies B's publicly known encryption procedure E_B to the signature S in order to extract the original message. Formally:

(5) $E_B(S) = E_B(D_B(M)) = M$

A now possesses a message-signature pair (M,S) with properties similar to those of a signed paper document. B cannot deny having sent the message, since no other user could have created $S = D_B(M)$. From equation (5), A has proof that B signed the document.

Therefore, A has received a message "signed" by B which can be proven to have been sent by B, and not been forged or tampered with by A. Clearly, these properties of the PKC satisfy the privacy and signature requirements of an EFT system.

PART II. THE DATA ENCRYPTION ALGORITHM

OPERATION

In the previous section, encryption and decryption procedures were discussed in general functional terms. In reality, some real set of procedures having the aforementioned four properties required to implement a PKC [this report, p. 1] must be developed. The method to be discussed here is due to Rivest [1978] and is currently the only such algorithm known to be practical in terms of implementation.

As mentioned previously, in practice, E and D are numbers to be used in conjunction with some publicly known and specified procedure or algorithm, rather than being algorithms themselves. In this method, E, the public encryption key is a pair of positive integers (e,n) and D, the secret decryption key is a second pair of positive integers (d,n). (N.B., n is the <u>same</u> in both keys).

A message M is encrypted as follows. M is represented as an integer in the range 0 to n-1. (Any numerical correspondence code can be used). A long message may be broken into a series of blocks, and the blocks encrypted individually. C, the encrypted form of M is generated by raising M to the e-th power, modulo n. Formally:

(6) $C = E(M) = M^e \pmod{n}$

To decrypt the ciphertext, it is raised to the d-th power, modulo n. Formally:

(7) $M = D(C) = C^d \pmod{n}$

Each user in the system will have an individual encryption key set (e,n) and an individual decryption key set (d,n).

KEY GENERATION

Encryption and decryption key sets are generated in the following manner. First, n is computed as the product of two prime numbers, p and q. These primes are very large "random" primes. Although n is made public as part of the encryption key E = (e,n), the factors p and q will remain effectively hidden due to the tremendous difficulty involved in factoring n. It is this fact which satisfies property (c) of PKC's, i.e. making E =(e,n) public does not permit ready derivation of D =(d,n)

The positive integer d is chosen as a large random integer which is relatively prime to (p-1)*(q-1). Formally, d must satisfy:

(8) gcd(d,(p-1)*(q-1)) = 1
(here gcd means greatest common divisor)

In practice, any prime number which is greater than max(p,q) will suffice for d.

The positive integer e is computed from p,q, and d as the multiplicative inverse of d, modulo (p-1)*(q-1). Formally:

(9) e*d = 1 (mod (p-1)*(q-1))

The first section of Appendix I gives a detailed example of the key generation and encryption-decryption procedures.

The theoretical mathematical under-pinnings of the encryption-decryption and key generation procedures are given in Rivest [1978]. The thrust of the remainder of this report is an empirical study of the computational complexity of the method, undertaken in order to assess its practicability and feasibility for use in an EFT system.

SECURITY

As mentioned in the last section, the security of the system lies in the fact that the decryption key D = (d,n) can only be computed if p and q are known, where n = p*q. Given that n is known publicly as part of E = (e,n), how difficult would it be to factor n in order to find p and q ?

Table 1.

Digits	Number of operations	Time
50	1.4×10^{10}	3.9 hours
75	9.0×10^{12}	104 days
100	2.3×10^{15}	74 years
200	1.2×10^{23}	3.8×10^{9} years
300	1.5×10^{29}	4.9×10^{15} years
500	1.3×10^{39}	4.2×10^{25} years

Rivest[1978]

Table 1 above gives estimates on the amount of time required to factor n on a high-speed computer using the fastest factoring algorithm known. It assumes an operating speed of one operation per micro-second, where CPU time required is solely a function of the number of digits in n.

As indicated, an 80-digit n provides moderate

security against current technology, and a 200-digit n would provide security against future technological developments as well. The barriers to factoring a sizable n appear to be insurmountable.

Factoring of large numbers is a well-known problem which dates back to the mathematicians Fermat (1601-1665), and Legendre (1752-1833) [Ore, 1967, pp. 27-37]. Although no proof exists indicating that the problem is NP-complete (not solvable in an amount of time which is bounded above by a polynomial function, and hence possibly exponential), no one has yet discovered an algorithm which can factor a 200-digit number in a reasonable or useful amount of time. It is upon this partially "certified" demonstration of difficulty that the encryption method stakes its claim to security.

IMPLEMENTATION

The Rivest procedure was implemented as a set of APL functions (Appendix IV) on the META-4 computer system, resident at the Center for Research in Management Science, at the University of California, Berkeley. The necessity of having unlimited precision in order to work accurately with large numbers was overcome by adapting a number of multi-precision arithmetic routines from Knuth [1969, pp. 162-339]. The generation of large prime numbers was accomplished through a modification of a testing procedure due to Fermat [Knuth, 1969, p. 347].

Functions were developed to convert text into numeric form (00 = blank, 01 = A, 02 = B, etc.) and vice versa. In order to avoid having to "reblock" "signed" messages prior to encryption for transmission, a procedure involving two sets of keys per user was developed. One set of keys is used for normal transmission encryption, while the other set is used to sign and verify messages.

The value of n used for signatures (denoted n_s) is always chosen to be smaller than the value of n chosen for transmission (denoted n_t), so that a "signed" message S will always be in the range $0 \leq S \leq n_t$, and

can be encrypted directly for transmission. (The preceding inequality holds because $n_s \leq n_t$ by definition, and $S \leq n_s$ by property of the encryption function which maps M into S, where S is in the range $0 \leq S \leq n_s$). A detailed example of this procedure is given in the second section of Appendix I.

TIMING TESTS

In order to properly assess the usefulness of the Rivest algorithm, a study was made of the time needed for encryption (decryption), which is mathematically equivalent to raising a large number to a high power, where the result is taken modulo some other large number. Recalling from an earlier section:

(6) $C = E(M) = M^e \pmod{n}$, for some M, e, and n.

Assuming a fixed message length of four characters (chosen because it is equivalent to a logical full-word), which translates into eight decimal digits under the previously described character to number transformation, an effort was made to measure the effects of the number of digits in the exponent (hereafter referred to as E), and the number of digits in the modulus (hereafter referred to as N) on the CPU time needed for encryption.

Appendix II gives the results of the timing runs. Values in CPU minutes were generated for all combinations of E ranging in steps of ten from ten to seventy, and N ranging in steps of ten from ten to seventy.

An initial contour plot of the timing values plotted against E and N indicated a non-linear relationship of some kind. An effort was made to isolate the effects of the individual parameters E and N on encryption time.

Plots of encryption time against E with N held constant indicated a linear relationship. Plots of encryption time against N with E held constant indicated a seemingly quadratic relationship. These plots are collected in the second section of Appendix II.

Regressions run on the individual variables confirmed these hypotheses. However, a regression run on both parameters jointly served to explain only 84% of the variation in the sample. (This regression was actually run on E and the transformed variable N^2). Given sufficient degrees of freedom, a cross-product parameter, $(E \times N^2)$, was introduced into the regression equation. Inclusion of this term helped to explain 99.98% of the variation, with t-values for all coefficients indicating significance at the 0.001 confidence level. The regression equation obtained was:

(10) $T = 0.03443(E) - 0.00021(N^2) + 0.0002(EN^2) - 0.03696$

where T is time expressed in CPU minutes.

CONCLUSIONS

The timing tests indicated that encryption time was significantly slower than had been initially hoped for. (E.g., T = 69 CPU minutes, for N = E = 70). However, given that the current implementation is in APL, a relatively slow language, and that function calls are nested six-deep during the encryption process, a study of encryption routines written in either assembly language or implemented as hardware chips is definitely warranted.

The algorithm itself has all the properties necessary to make it the cornerstone of a PKC. The elegant and useful concept of an electronic signature, as well as the practical privacy aspects of seemingly unbreakable security make the algorithm and its associated PKC highly desirable for use in an EFT system.

REFERENCES

REFERENCES

Diffie,W. and Hellman,M. "New Directions in Cryptography," IEEE Trans. Inform. Theory IT-22, 6 (Nov. 1976), 644-654.

Knuth,D.E. The Art of Computer Programming, Vol 2: Seminumerical Algorithms. Reading, Mass.: Addison-Wesley, 1969.

Ore, Oynstein. Invitation to Number Theory. New York.: Random House, 1967.

Rivest, R.L., Shamir, A. and Adleman, L. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, 21 (Feb. 1978), 120-126.

Traub, J.F. (Ed.). Algorithms and Complexity. New York.: Academic Press, 1976.

Wesolowsky, George O. Multiple Regression and Analysis of Variance. New York.: Wiley, 1976.

Wright, Harry N. First Course in the Theory of Numbers. New York.: Dover, 1971.

APPENDIX I

```
      ⍝ STEP1.  FIND TWO LARGE RANDOM NUMBERS, P AND Q

      P←PRIMEGEN 10

NO. TESTED = 3
ELAPSED TIME = 0.85 MINUTES

      Q←PRIMEGEN 10

NO. TESTED = 4
ELAPSED TIME = 1.13333 MINUTES

      DISPLAY P
2173840121
      DISPLAY Q
9265776551

      ⍝ STEP 2.  FIND A RANDOM PRIME NUMBER D > MAX(P,Q)

      D←PRIMEGEN 11

NO. TESTED = 14
ELAPSED TIME = 5.08333 MINUTES

      DISPLAY D
11080122817

      ⍝ STEP 3.  COMPUTE N=P*Q, AND PHI(N)=(P-1)*(Q-1)

      N←P MULT Q
      PHI←(P MINUS ,1) MULT (Q MINUS ,1)

      DISPLAY N
20142316818784802671
      DISPLAY PHI
20142316807345186000

      ⍝ STEP 4.  COMPUTE E, WHERE E*D=1 (MOD (P-1)*(Q-1))

      E←PHI MULTINV D

      DISPLAY E
1348514798658177153

      ⍝ STEP 5.  TEST E AND D

      (E MULT D) MOD PHI
1
```

```
⍝ STEP 6.  CONVERT MESSAGE INTO AN INTEGER (0≤M≤N-1)
⍝ EACH CHARACTER IS CONVERTED INTO A TWO-DIGIT NUMBER
      N1_2←N[1 2]
      DISPLAY N1_2
20
```

```
⍝ LETTING THE FIRST MESSAGE CHARACTER CODE BE < 20
⍝ (BLANK = 00 WILL DO), MESSAGE MAY BE TEN CHARACTERS
```

```
      MESSAGE←' IT WORKS.'
      MESSAGE←TNCONVERT MESSAGE

      DISPLAY MESSAGE
0009200023151811937
```

```
⍝ STEP 7.  ENCRYPT THE MESSAGE
⍝ (COMPUTE M TO THE E-TH POWER, MODULO N)
```

```
      ENCRYPTEDMESSAGE←ENCRYPT[MESSAGE;E;N]

      DISPLAY ENCRYPTEDMESSAGE
16416839271599238466
```

```
⍝ STEP 8.  DECRYPT THE ENCRYPTED MESSAGE
⍝ (COMPUTE EM TO THE D-TH POWER, MODULO N)
```

```
      DECRYPTEDMESSAGE←ENCRYPT[ENCRYPTEDMESSAGE;D;N]

      DISPLAY DECRYPTEDMESSAGE
92000231518111937
```

```
⍝ STEP 9.  CONVERT THE NUMERIC DECRYPTED MESSAGE TO TEXT
```

```
      DECRYPTEDMESSAGE←NTCONVERT DECRYPTEDMESSAGE

      DECRYPTEDMESSAGE
IT WORKS.
```

```
⍝ NOTE THAT LEADING BLANKS (CODE=00) WERE LOST IN THE
⍝ DECRYPTION.  THIS COULD BE AVOIDED BY ELIMINATING 00
⍝ AS A LEGITIMATE CODE
```

```
A DEMONSTRATION OF TRANSMISSION AND VERIFICATION
A OF A SIGNED MESSAGE BETWEEN PERSONS A AND B
```

```
A STEP 1.  PERSON A WISHES TO SEND THE MESSAGE
A 'BUY 8' TO PERSON B.  IT MUST FIRST BE
A CONVERTED TO AN INTEGER
```

```
      MESSAGE←'BUY 8'

      MESSAGE←TNCONVERT MESSAGE

      DISPLAY MESSAGE
0221250035
```

```
A STEP 2.  PERSON A MUST 'SIGN' THE MESSAGE BY
A APPLYING HIS PRIVATELY KNOWN SIGNATURE
A DECRYPTION KEY
```

```
      DISPLAY SIGNATURE_DECRYPT_A
153949

      DISPLAY SIGNATURE_N_A
2187533923

      MESSAGE←ENCRYPT[MESSAGE;SIGNATURE_DECRYPT_A;SIGNATURE_N_A]

      DISPLAY MESSAGE
2066433642
```

```
A STEP 3.  PERSON A MUST NOW ENCRYPT THE 'SIGNED'
A MESSAGE USING PERSON B'S PUBLICLY KNOWN ENCRYPTION
A KEY
```

```
      DISPLAY TRANSMIT_ENCRYPT_B
3068878861410677711

      DISPLAY TRANSMIT_N_B
54994590181085912 69

      MESSAGE←ENCRYPT[MESSAGE;TRANSMIT_ENCRYPT_B;TRANSMIT_N_B]

      DISPLAY MESSAGE
22201574816768803 95
```

```
      ⍝ STEP 4.  THE MESSAGE IS TRANSMITTED TO PERSON B.
      ⍝ HE MUST FIRST DECRYPT IT USING HIS PRIVATELY
      ⍝ KNOWN DECRYPTION KEY
```

```
      DISPLAY TRANSMIT_DECRYPT_B
63734134319

      DECRYPTEDMESSAGE←ENCRYPT[MESSAGE;TRANSMIT_DECRYPT_B;TRANSMIT_N_B]

      DISPLAY DECRYPTEDMESSAGE
2066433642
```

```
      ⍝ STEP 5.  THE MESSAGE CURRENTLY IS MEANINGLESS
      ⍝ (NOTE, THAT THERE IS NO TEXT EQUIVALENT OF
      ⍝ THE THIRD CODE CHARACTER 43).  IT MUST BE
      ⍝ FURTHER DECRYPTED (AND VERIFIED AS HAVING
      ⍝ BEEN SENT BY PERSON A) BY APPLYING PERSON A'S
      ⍝ PUBLICLY KNOWN SIGNATURE ENCRYPTION KEY
```

```
      DISPLAY SIGNATURE_ENCRYPT_A
653791669

      DISPLAY SIGNATURE_N_A
2187533923

      DECRYPTEDMESSAGE←ENCRYPT[DECRYPTEDMESSAGE]

      MESSAGE←ENCRYPT[DECRYPTEDMESSAGE;SIGNATURE_ENCRYPT_A;SIGNATURE_N_A]

      DISPLAY MESSAGE
221250035
```

```
      ⍝ STEP 6.  THE MESSAGE IS CONVERTED BACK TO TEXT
```

```
      MESSAGE←NTCONVERT MESSAGE

      MESSAGE
BUY 8
```

APPENDIX II

Encryption Time in CPU Minutes

E \ N	10.000	20.000	30.000	40.000	50.000	60.000	70.000
10.000	0.417	1.000	1.967	3.200	4.750	6.583	9.433
20.000	0.917	2.267	4.250	6.883	10.150	14.200	19.017
30.000	1.283	3.317	6.300	10.383	15.467	21.533	28.967
40.000	1.750	4.400	8.517	13.783	20.783	29.167	39.083
50.000	2.183	5.533	10.650	17.450	26.250	36.667	49.233
60.000	2.600	6.800	12.850	20.833	31.400	43.950	59.167
70.000	3.133	8.033	15.150	24.783	36.933	52.117	69.617

N = No. digits in modulus

E = No. digits in exponent

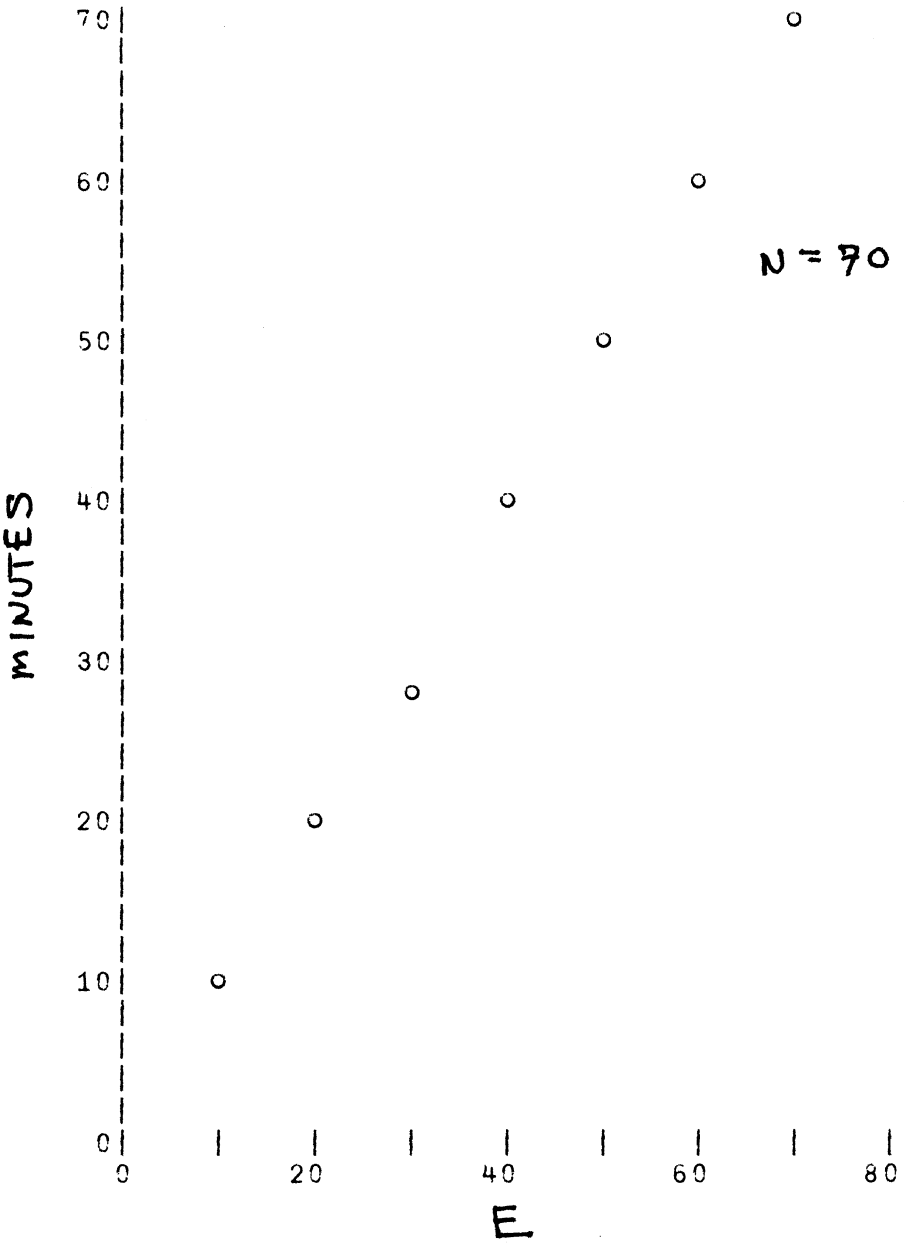
```
CONTOUR[EXPTIMES;0;3.5;20]

                    N

  10 *****.....*****.....*****.....  70

   *               BBBB    CCCCCCDDDD  *      A = 0
   *             BBB     CCCC  DDDDEEE  *      B = 3.5
   *           BBB     CCC  DDDDEEEEFF  *      C = 7
   .         BBB     CCC  DDD EEEFFFGG  .      D = 10.5
   .         B     CCC  DDD EEEFFFGGGH  .      E = 14
   .       BB    CC    DD EEEFFFGGHHHI  .      F = 17.5
   *      BB   CC  DDD EE FFGGGHHIIJ  *      G = 21
   *     BB   CC  DD  EEFFFGGHHIIJJK  *      H = 24.5
   *     B    C  DD  EEFF GGHHIIJKKL  *      I = 28
   .    BB   CC  D  EEFFGGGHIIJJKKLM  .      J = 31.5
   .    B   CC  DD EEFFGG HIIJJKLLMN  .      K = 35
   .   BB   C  DD EE F G HIIJJKLLMNO  .      L = 38.5
E  *   B   CC  D EE FFGGHHIJJKLLMNOP  *      M = 42
   *   B   C  DD E FFGGHHIJJKLLMNOPP  *      N = 45.5
   *   B   C  D EEFF GHHIIJKLLMNOPQQ  *      O = 49
   .  BB  CC DD E F GGHIIJKLLMNOPQQR  .      P = 52.5
   .  B   C  D EEFFGGHIIJKKLMNOPQQRS  .      Q = 56
   .  B  CC DD E F GHHIJJKLMNOOPQRST  .      R = 59.5
  70
      *****.....*****.....*****.....

                                         S = 63
                                         T = 66.5
                                         U = 70
```

```
CONTOUR[EXPTIMES;0;7;10]

                    N

  10 *****.....*****.....*****.....  70

   *                      BBBBBB      *      A = 0
   *                   BBBB      CCC  *      B = 7
   *                 BBB       CCCC   *      C = 14
   .               BBB       CCC   DD  .      D = 21
   .             BBB       CCC   DDD   .      E = 28
   .           BB       CCC   DDD  E  .      F = 35
   *         BB       CC   DDD  EEE  *      G = 42
   *        BB       CC   DD  EEE F  *      H = 49
   *        B       CC   DD  EE FFF  *      I = 56
   .      BB      CC  DDD EEE FF G  .      J = 63
   .     BB      CC  DD  EE  FF GG  .      K = 70
E  .     B      CC   D  EE  FF GGH  .
   *    BB     CC   DD  E  FF GGHH  *
   *    B      C   DD  EE FF GGHH   *
   *    B     CC   D  EE FF GGHHII  *
   .   BB     C   DD EE FF GGHHII   .
   .   B     CC  DD EE FF GGHHII J  .
   .  BB     C   D  E  F GGHH I JJ  .
  70 *****.....*****.....*****.....
```

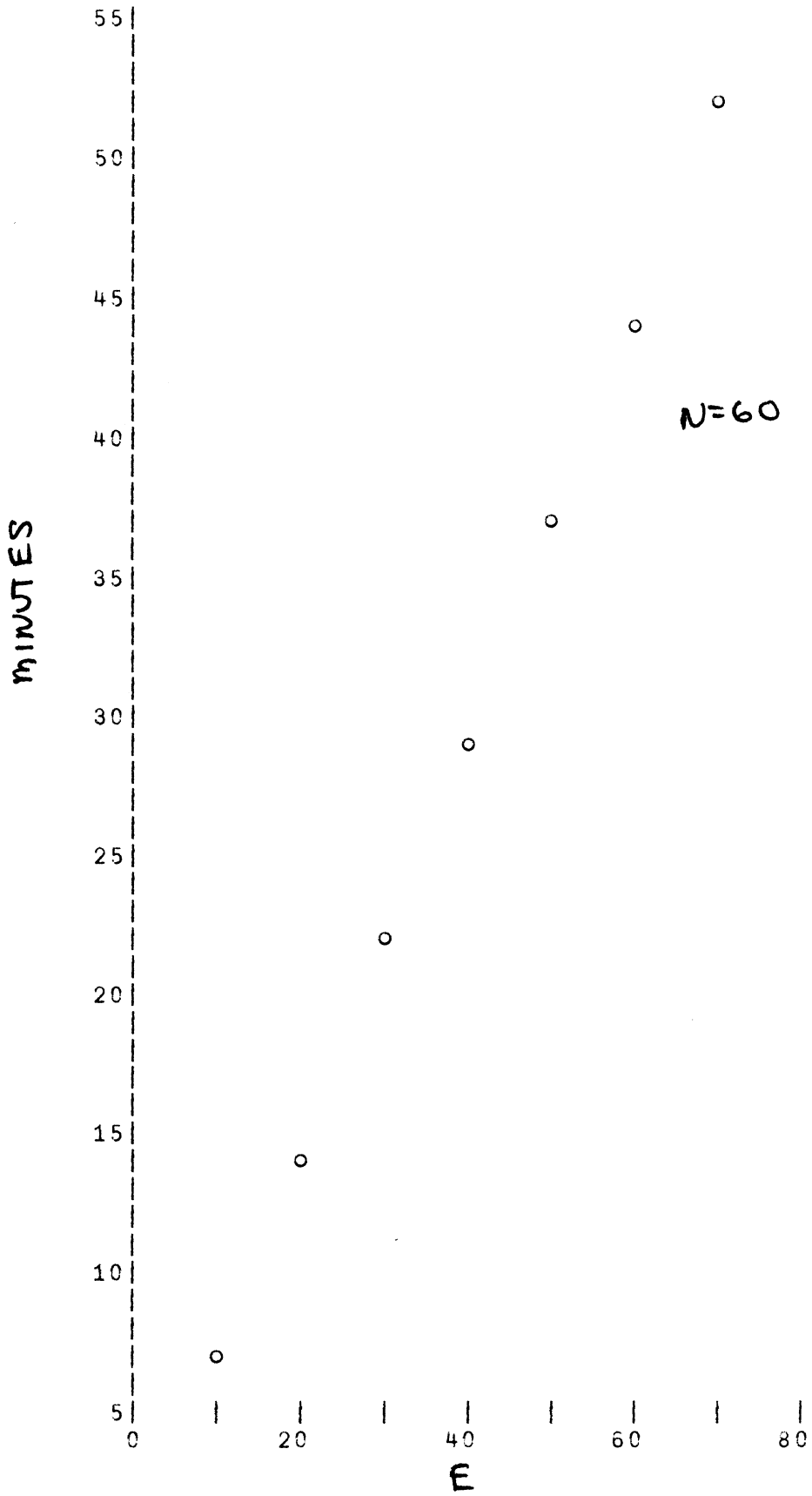
Contour Plot of Encryption Times

T = F(E,N)

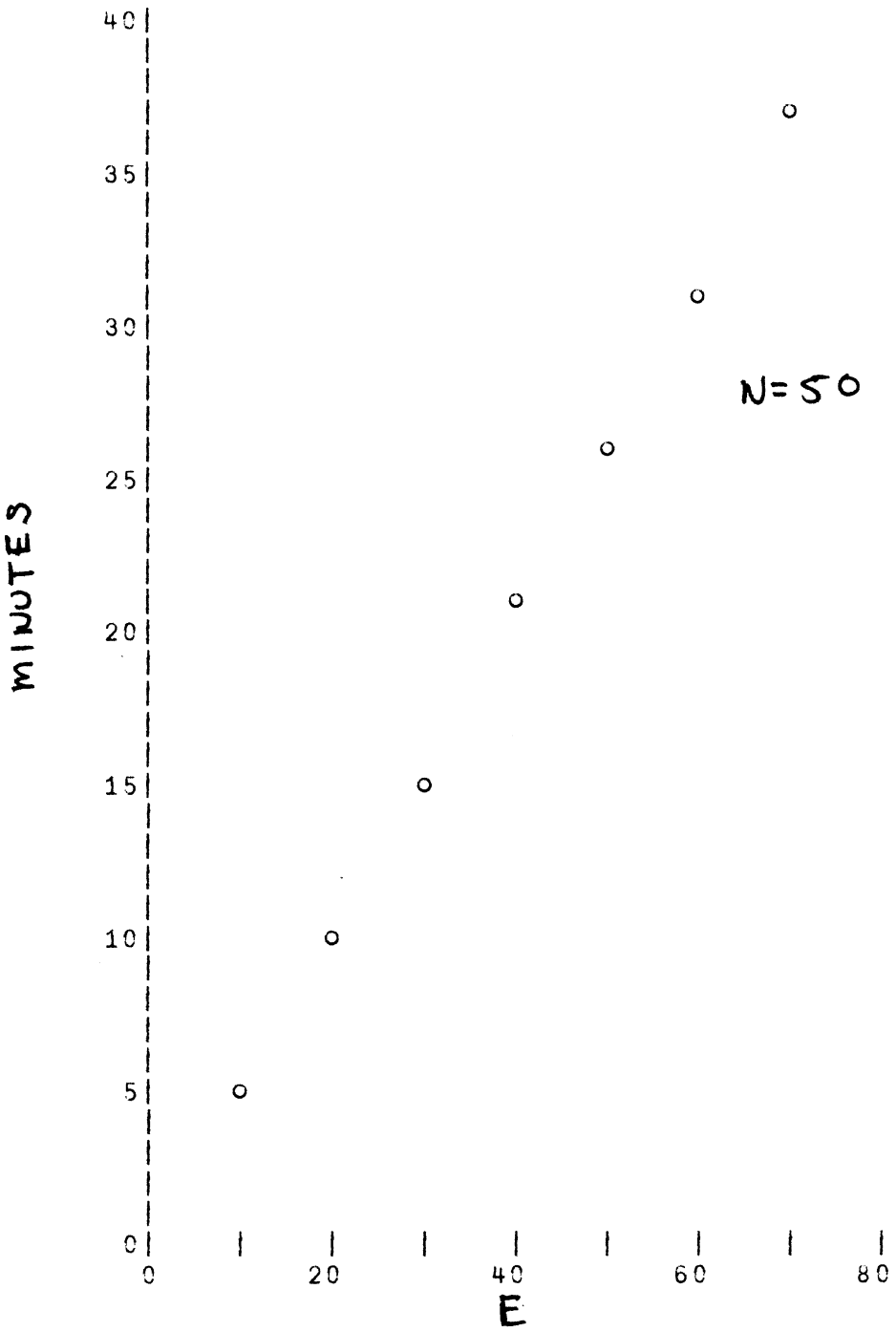
```
      50 50 PLOT EXPTIMES[;7] VS E←10 20 30 40 50 60 70
```



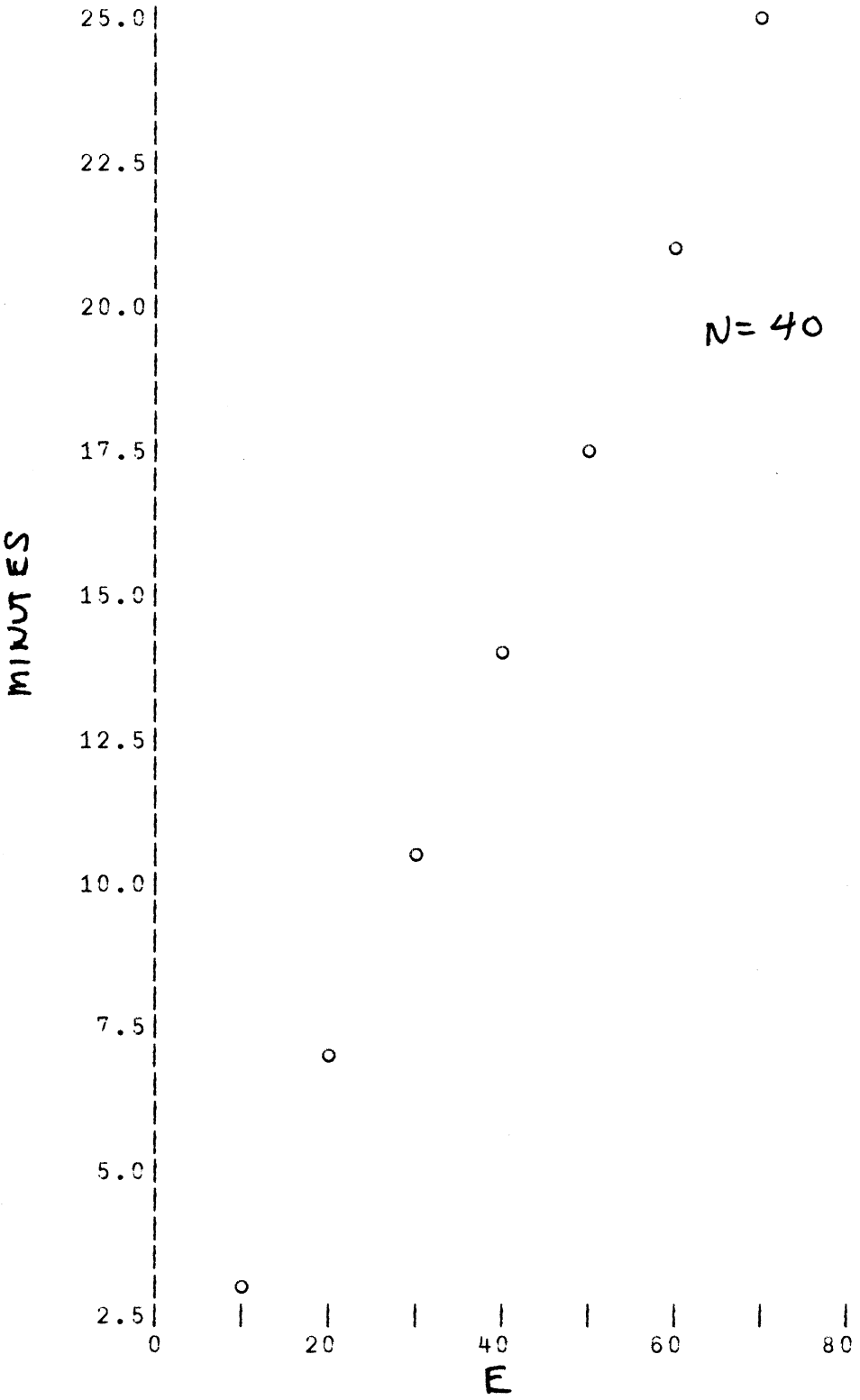
```
      50 50 PLOT EXPTIMES[;6] VS E
```



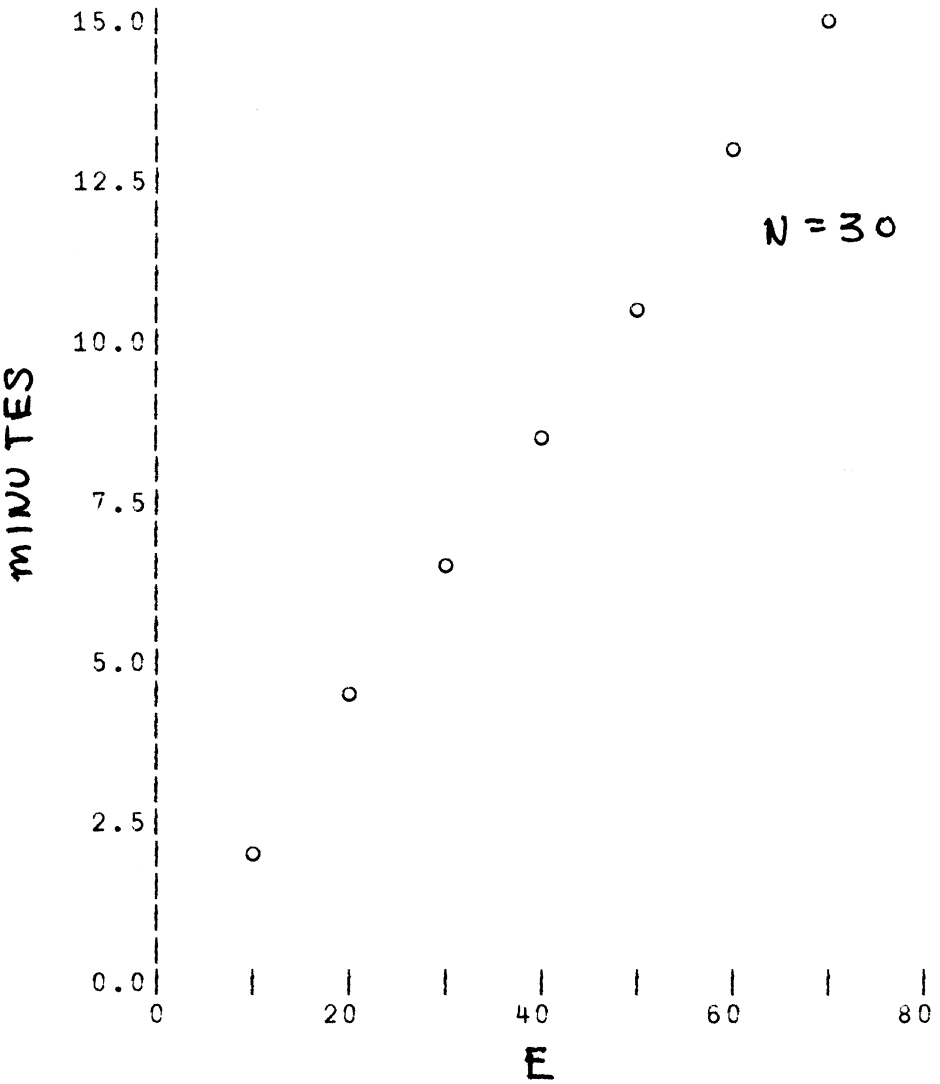
```
50 50 PLOT EXPTIMES[;5] VS E
```



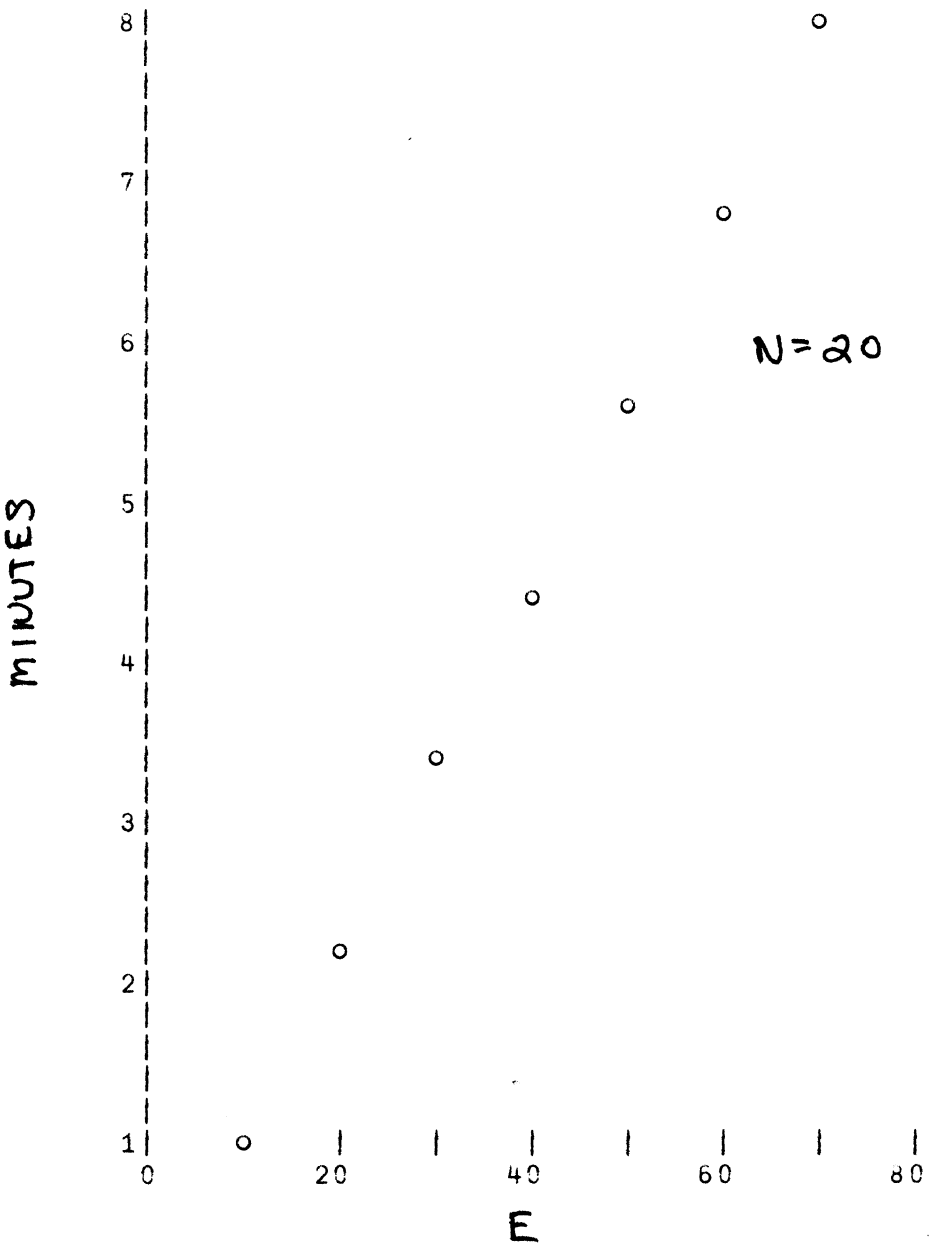
50 50 *PLOT EXPTIMES[;4] VS E*



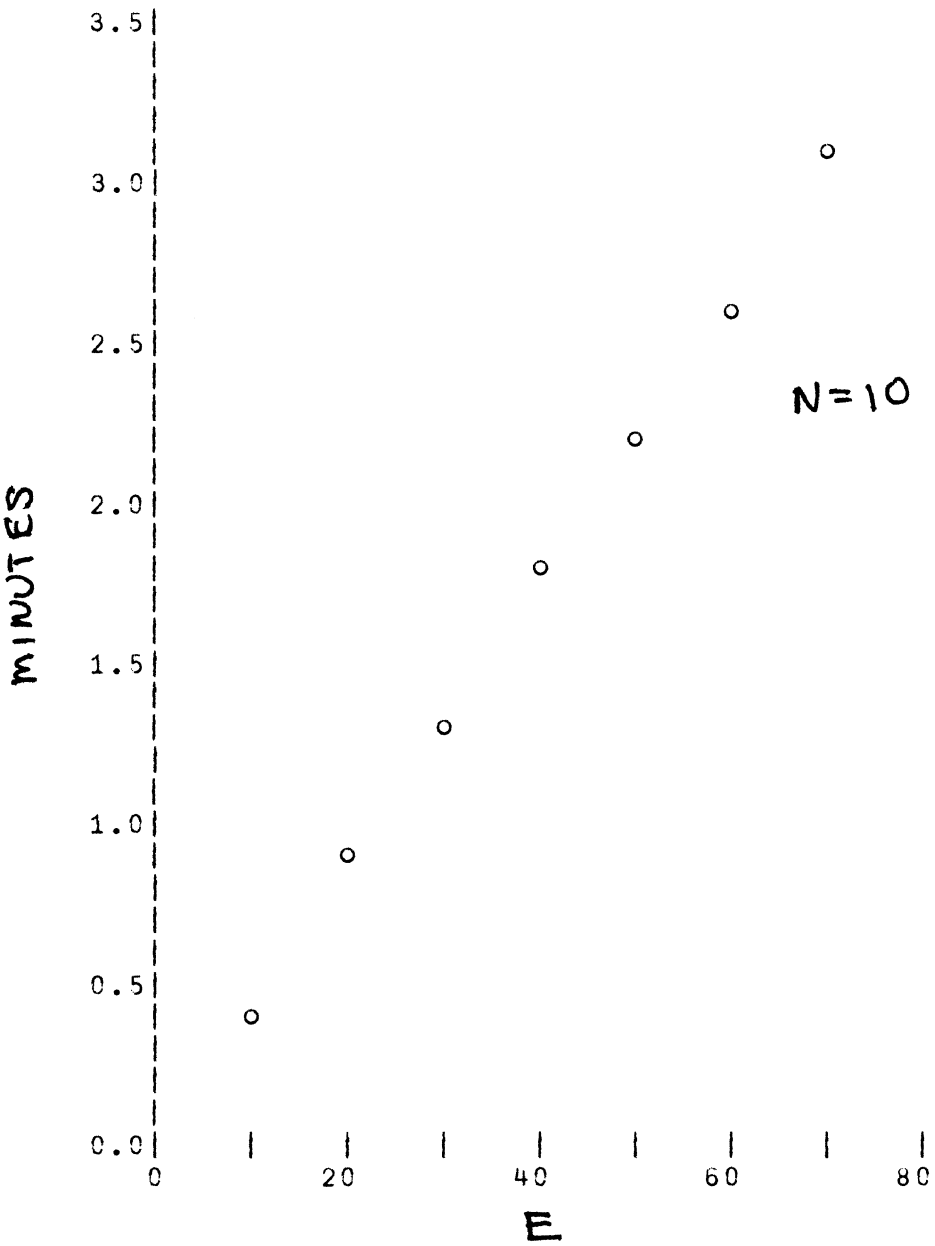
50 50 *PLOT EXPTIMES[;3] VS E*



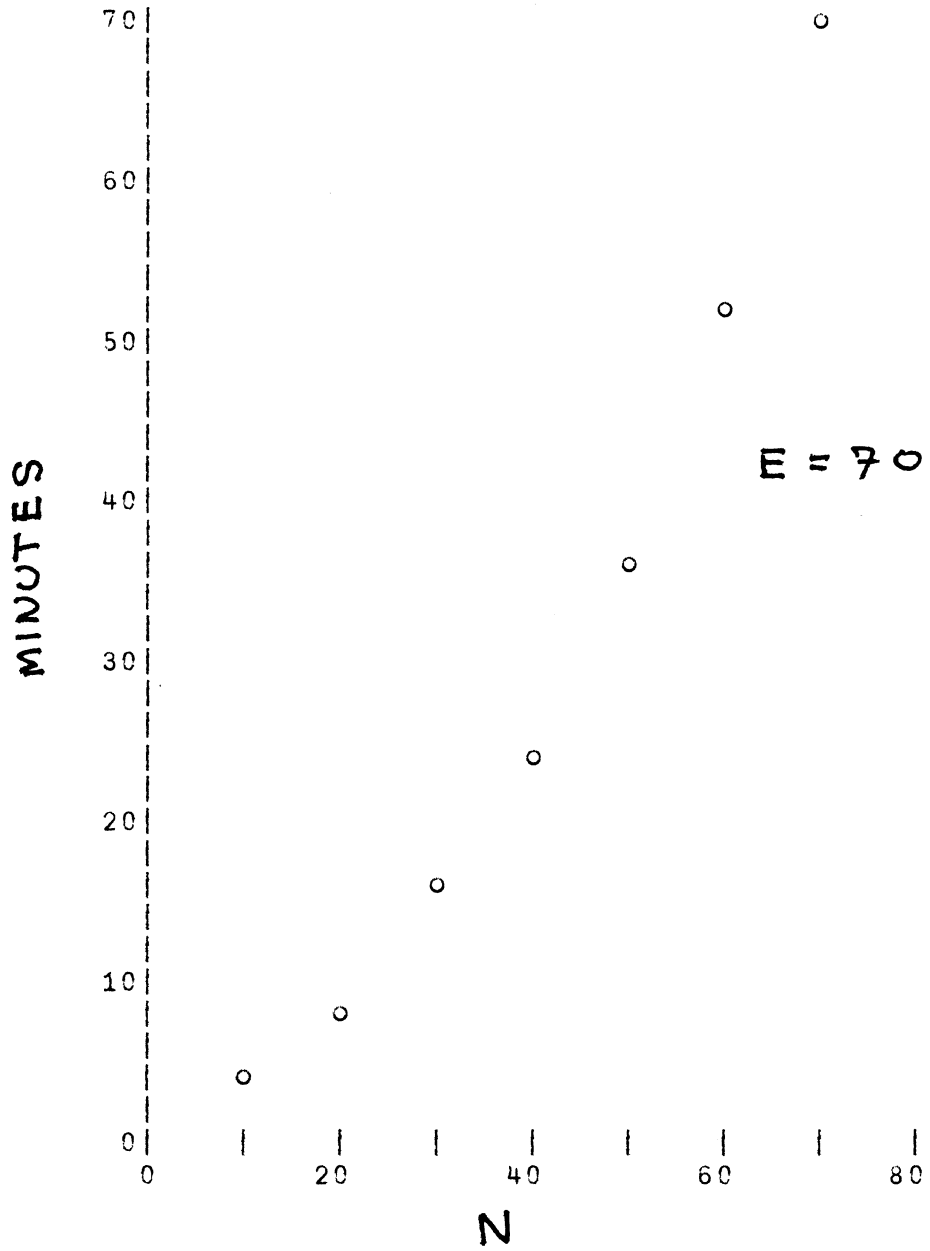
```
50 50 PLOT EXPTIMES[;2] VS E
```



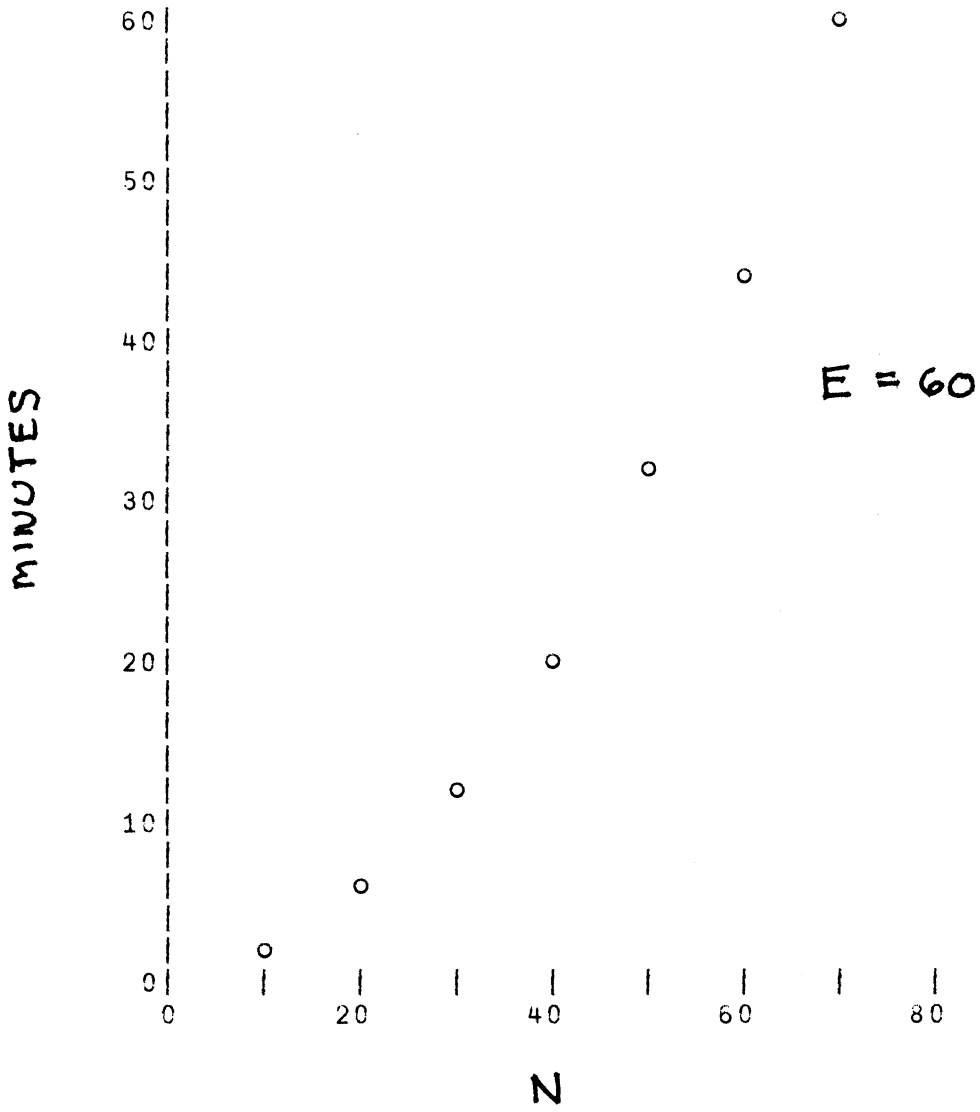
50 50 *PLOT EXPTIMES[;1] VS E*



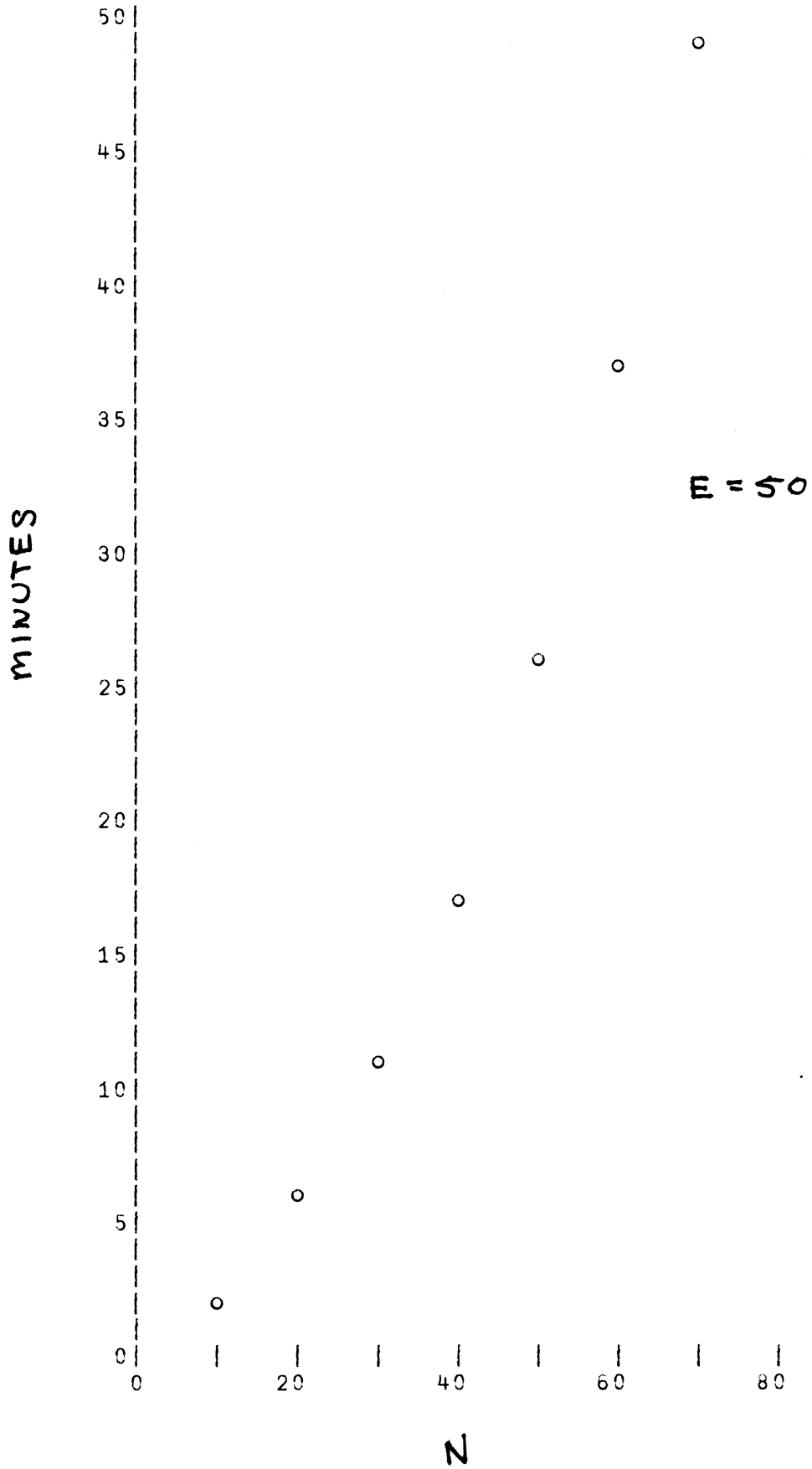
```
      50 50 PLOT EXPTIMES[7;] VS N←10 20 30 40 50 60 70
```



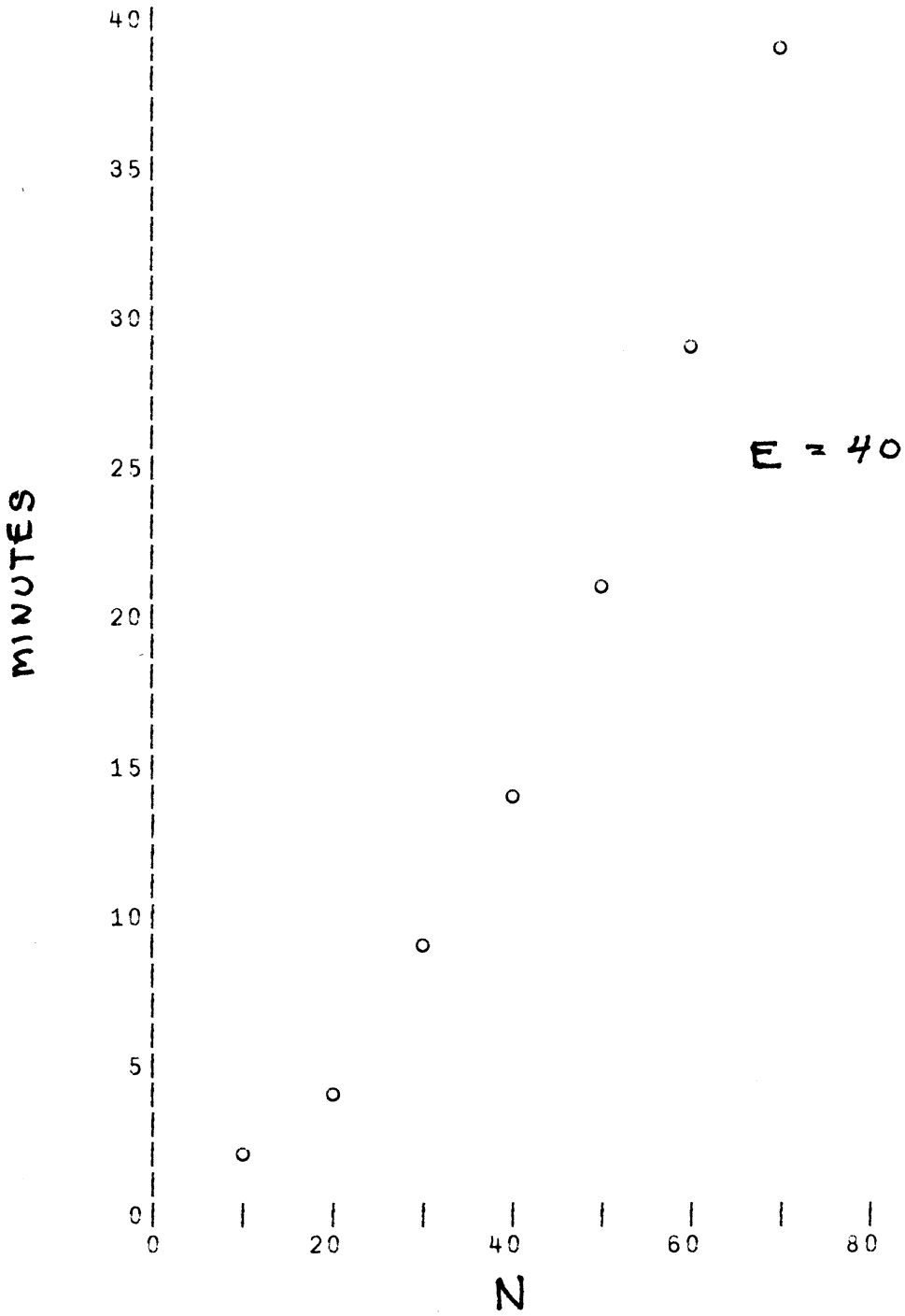
```
50 50 PLOT EXPTIMES[6;] VS N
```



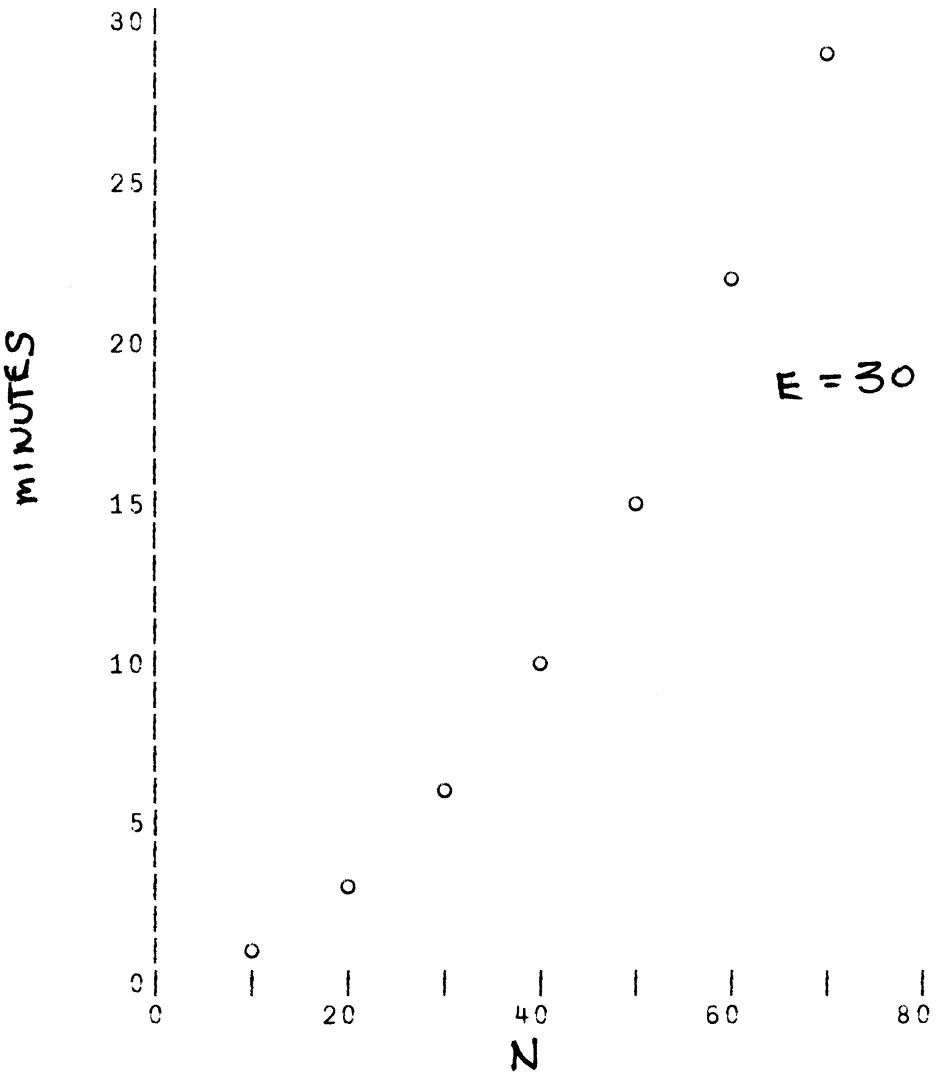
```
50 50 PLOT EXPTIMES[5;] VS N
```



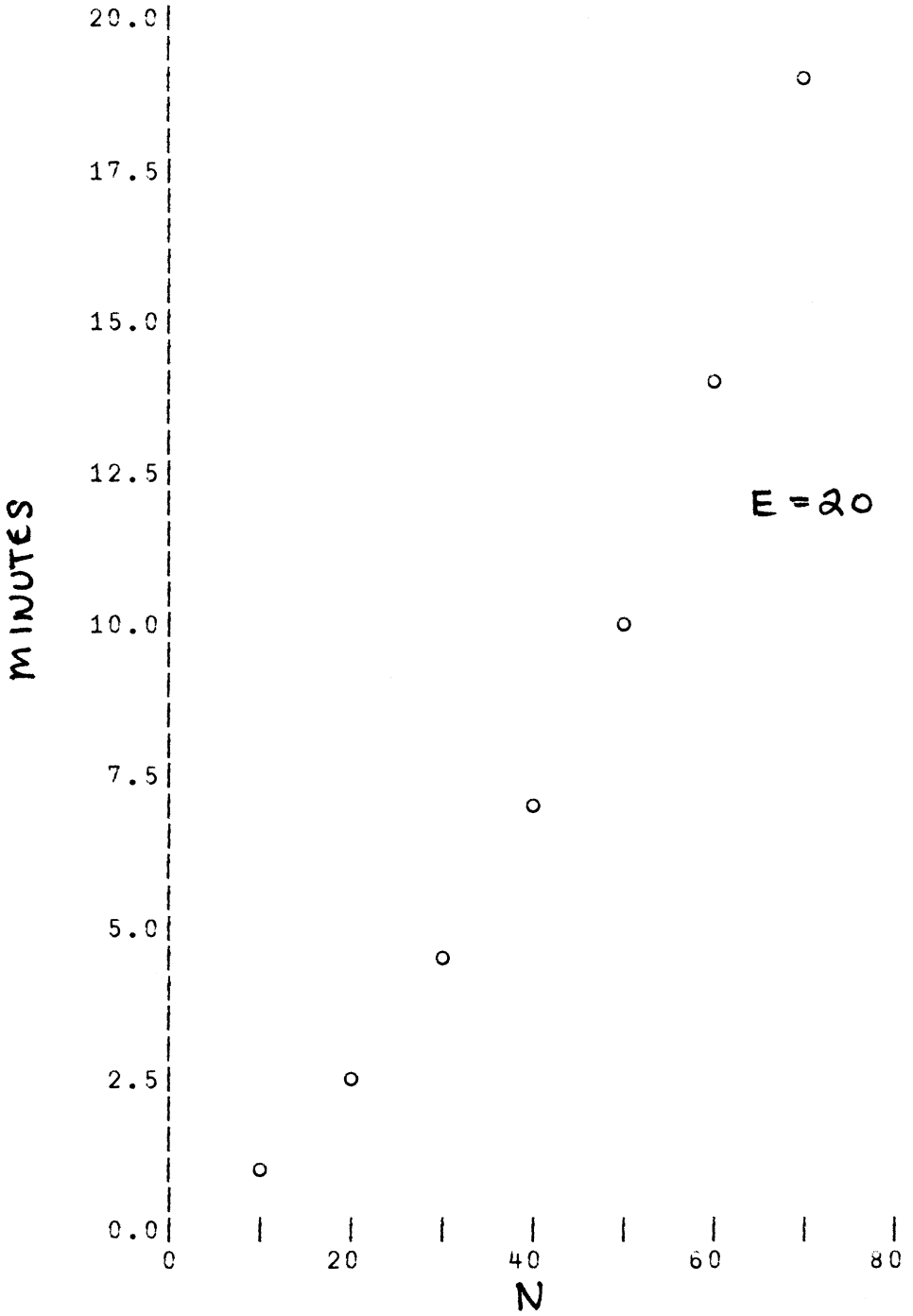
```
50 50 PLOT EXPTIMES[4;] VS N
```



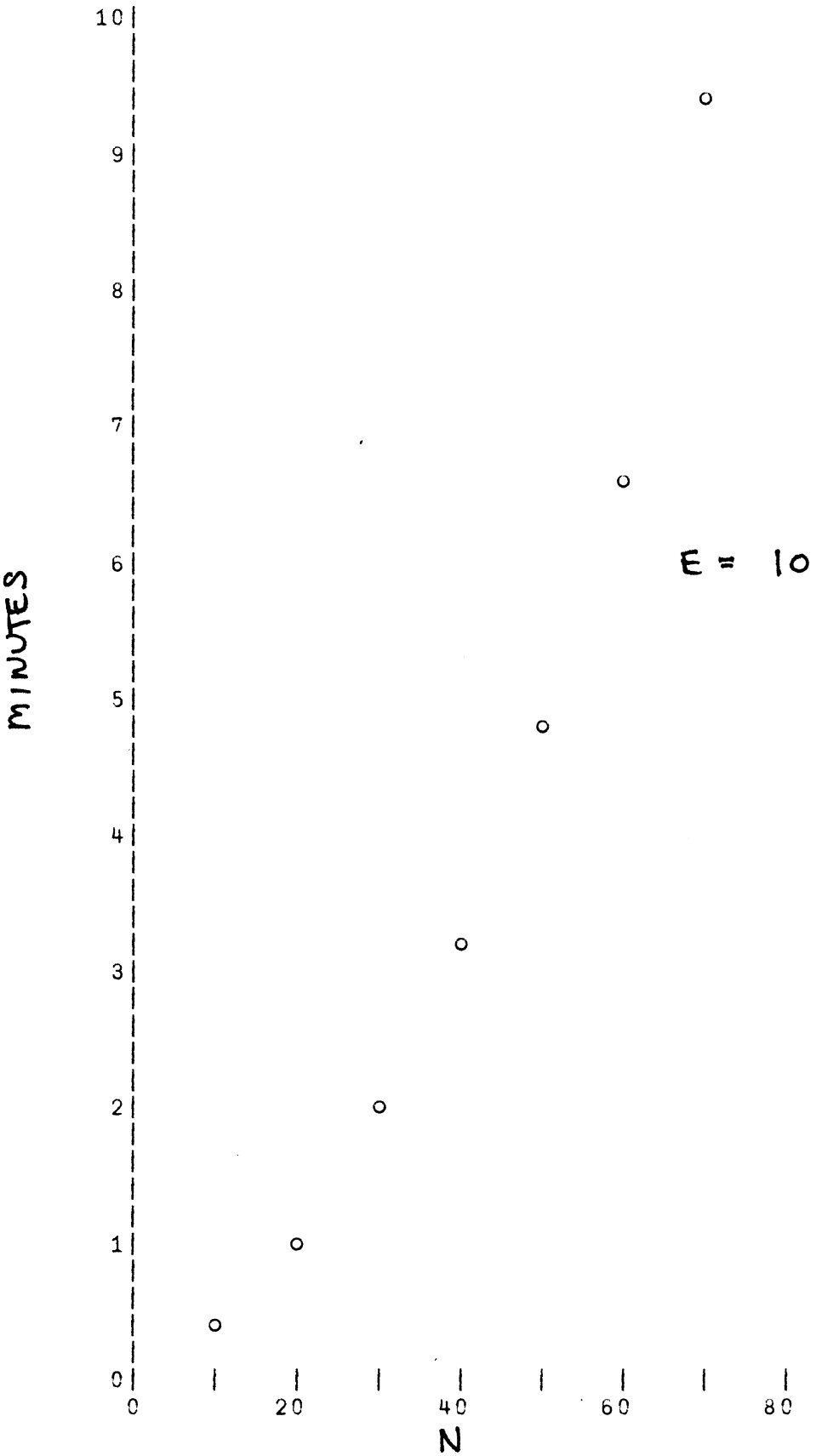
50 50 *PLOT EXPTIMES[3;] VS N*



50 50 *PLOT EXPTIMES[2;] VS N*



50 50 *PLOT EXPTIMES[1;] VS N*



APPENDIX III

EXPTIMES

10	20	30	40	50	60	70	E \ N
0.417	1.000	1.967	3.200	4.750	6.583	9.433	10
0.917	2.267	4.250	6.883	10.150	14.200	19.017	20
1.283	3.317	6.300	10.383	15.467	21.533	28.967	30
1.750	4.400	8.517	13.783	20.783	29.167	39.083	40
2.183	5.533	10.650	17.450	26.250	36.667	49.233	50
2.600	6.800	12.850	20.833	31.400	43.950	59.167	60
3.133	8.033	15.150	24.783	36.933	52.117	69.617	70

Encryption Time in
CPU Minutes

```
E←N←10 20 30 40 50 60 70
E70←EXPTIMES[7;]
N70←EXPTIMES[;7]
N2←N×N
N3←N×N×N

REGMAT←⍉5 7⍴E,N2,N3,E70,N70
```

REGMAT

E	N^2	N^3	E=70	CPU Mins. N=70
10.000	100.000	1000.000	3.133	9.433
20.000	400.000	8000.000	8.033	19.017
30.000	900.000	27000.000	15.150	28.967
40.000	1600.000	64000.000	24.783	39.083
50.000	2500.000	125000.000	36.933	49.233
60.000	3600.000	216000.000	52.117	59.167
70.000	4900.000	343000.000	69.617	69.617

Encryption Time Regressions on Individual Parameters E and N^2 (One parameter fixed in each case)

```
      MLTREG[REGMAT]
LIST DATA?
NO
PRINT BASIC STATISTICS?
YES
```

	MEAN	VARIANCE	STD.DEV.
VAR 1	40.000000	466.667000	21.602500
VAR 2	2000.000000	31266700.000	1768.240000
VAR 3	112000.0000	16168700000	127156.0000
VAR 4	29.966700	593.130000	24.354300
VAR 5	39.216700	470.444000	21.689700

```
SIMPLE CORRELATIONS?
YES
```

	VAR 1	VAR 2	VAR 3	VAR 4	VAR 5
VAR 1	1.00000	0.97736	0.93439	0.98015	0.99995
VAR 2	0.97736	1.00000	0.98795	0.99989	0.97918
VAR 3	0.93439	0.98795	1.00000	0.98602	0.93739
VAR 4	0.98015	0.99989	0.98602	1.00000	0.98183
VAR 5	0.99995	0.97918	0.93739	0.98183	1.00000

```
CONTINUE?
YES
PARTIAL CORRELATIONS?
NO


ENTER NAME FOR REGRESSION TITLE:
ENCRYPTION TIME AS FUNCTION OF DIGITS IN EXPONENT
ENTER LIST OF INDEPENDENT VARIABLES:
1
ENTER DEPENDENT VARIABLE:
5
STEPWISE REGRESSION?
NO
```

ENCRYPTION TIME AS FUNCTION OF DIGITS IN EXPONENT
T = F(E,N) , N=70

INDEPENDENT VARIABLES: 1
DEPENDENT VARIABLE: 5

	COEFF	*STD.ERROR*	*T-STATISTIC*	*VARIANCE*
VARIABLE 1	1.00399	0.00452	222.27000	99.99000

	DEG OF FREE	*SUM OF SQS*	*MEAN SQUARE*
REGRESSION	1.00000	2822.38000	2822.38000
ERROR	5.00000	0.28564	0.05713

S.E. OF ESTIMATE: 0.23902
F-VALUE: 49403.7
MULTIPLE R-SQUARED: 99.99
INTERCEPT: -0.94285

T = 1.00399(E)-0.94285

WORK ON RESIDUALS?
YES
PRINT RESIDUALS?
YES

	OBSERVED	*ESTIMATED*	*RESIDUAL*
OBSERV 1	9.43330	9.09700	0.33630
OBSERV 2	19.01670	19.13690	-0.12020
OBSERV 3	28.96670	29.17680	-0.21010
OBSERV 4	39.08330	39.21670	-0.13330
OBSERV 5	49.23330	49.25660	-0.02320
OBSERV 6	59.16670	59.29640	-0.12980
OBSERV 7	69.61670	69.33630	0.28040

RUN TEST ON RESIDUALS?
YES

SUM OF RESIDUALS: -3.8147E-5
SUM OF SQUARES OF RESIDUALS: 0.28547
DURBAN-WATSON STATISTIC: 1.451

ENCRYPTION TIME AS FUNCTION OF DIGITS IN MODULUS (QUADRATIC)

$T = F(E,N^2)$, E=70

INDEPENDENT VARIABLES: 2
DEPENDENT VARIABLE: 4

	COEFF	*STD.ERROR*	*T-STATISTIC*	*VARIANCE*
VARIABLE 2	0.01377	0.00009	151.71000	99.98000

	DEG OF FREE	*SUM OF SQS*	*MEAN SQUARE*
REGRESSION	1.00000	3558.01000	3558.01000
ERROR	5.00000	0.77295	0.15459

S.E. OF ESTIMATE: 0.39318
F-VALUE: 23015.8
MULTIPLE R-SQUARED: 99.98
INTERCEPT: 2.4233

$T = 0.01377(N^2)+2.4233$

WORK ON RESIDUALS?
YES
PRINT RESIDUALS?
YES

	OBSERVED	*ESTIMATED*	*RESIDUAL*
OBSERV 1	3.13330	3.80050	-0.66710
OBSERV 2	8.03330	7.93200	0.10140
OBSERV 3	15.15000	14.81780	0.33220
OBSERV 4	24.78330	24.45800	0.32530
OBSERV 5	36.93330	36.85250	0.08080
OBSERV 6	52.11670	52.00140	0.11530
OBSERV 7	69.61670	69.90450	-0.28790

RUN TEST ON RESIDUALS?
YES

SUM OF RESIDUALS: 1.00136E-5
SUM OF SQUARES OF RESIDUALS: 0.77424
DURBAN-WATSON STATISTIC: 1.12

```
      ∇MCONVERT[□]∇
     ∇M←MCONVERT MATRIX;E;N;ETEMP;NTEMP
[1]    M←⍳0//E←1
[2]    ELOOP: N←1//ETEMP←10×E
[3]    NLOOP: NTEMP←10×N
[4]    M←M,ETEMP,(NTEMP×NTEMP),MATRIX[E;N]
[5]    →(7≥N←N+1)⍴NLOOP
[6]    →(7≥E←E+1)⍴ELOOP
[7]    M←49 3⍴M
      ∇
```

EXPTIMES

N \ E	10	20	30	40	50	60	70
10	0.417	1.000	1.967	3.200	4.750	6.583	9.433
20	0.917	2.267	4.250	6.883	10.150	14.200	19.017
30	1.283	3.317	6.300	10.383	15.467	21.533	28.967
40	1.750	4.400	8.517	13.783	20.783	29.167	39.083
50	2.183	5.533	10.650	17.450	26.250	36.667	49.233
60	2.600	6.800	12.850	20.833	31.400	43.950	59.167
70	3.133	8.033	15.150	24.783	36.933	52.117	69.617

Encryption Time in CPU Minutes

Encryption Time Regressions on Joint Parameters E and N^2

REGMAT

E	N^2	Time CPU Mins.
10.000	100.000	0.417
10.000	400.000	1.000
10.000	900.000	1.967
10.000	1600.000	3.200
10.000	2500.000	4.750
10.000	3600.000	6.583
10.000	4900.000	9.433
20.000	100.000	0.917
20.000	400.000	2.267
20.000	900.000	4.250
20.000	1600.000	6.883
20.000	2500.000	10.150
20.000	3600.000	14.200
20.000	4900.000	19.017
30.000	100.000	1.283
30.000	400.000	3.317
30.000	900.000	6.300
30.000	1600.000	10.383
30.000	2500.000	15.467
30.000	3600.000	21.533
30.000	4900.000	28.967
40.000	100.000	1.750
40.000	400.000	4.400
40.000	900.000	8.517
40.000	1600.000	13.783
40.000	2500.000	20.783
40.000	3600.000	29.167
40.000	4900.000	39.083
50.000	100.000	2.183
50.000	400.000	5.533
50.000	900.000	10.650
50.000	1600.000	17.450
50.000	2500.000	26.250
50.000	3600.000	36.667
50.000	4900.000	49.233
60.000	100.000	2.600
60.000	400.000	6.800
60.000	900.000	12.850
60.000	1600.000	20.833
60.000	2500.000	31.400
60.000	3600.000	43.950
60.000	4900.000	59.167
70.000	100.000	3.133
70.000	400.000	8.033
70.000	900.000	15.150
70.000	1600.000	24.783
70.000	2500.000	36.933
70.000	3600.000	52.117
70.000	4900.000	69.617

```
      MLTREG[REGMAT]
LIST DATA?
NO
PRINT BASIC STATISTICS?
YES
```

	MEAN	VARIANCE	STD.DEV.
VAR 1	40.00000	408.33300	20.20730
VAR 2	2000.00000	2735830.000	1654.04000
VAR 3	16.83880	284.11100	16.85560

```
SIMPLE CORRELATIONS?
YES
```

	VAR 1	VAR 2	VAR 3
VAR 1	1.00000	0.00000	0.51856
VAR 2	0.00000	1.00000	0.76043
VAR 3	0.51856	0.76043	1.00000

```
CONTINUE?
YES
PARTIAL CORRELATIONS?
NO


ENTER NAME FOR REGRESSION TITLE:
ENCRYPTION TIME AS FUNCTION OF DIGITS IN EXPONENT AND MODULUS
ENTER LIST OF INDEPENDENT VARIABLES:
1 2
ENTER DEPENDENT VARIABLE:
3
STEPWISE REGRESSION?
NO
```

ENCRYPTION TIME AS FUNCTION OF DIGITS IN EXPONENT AND MODULUS
$T = F(E,N^2)$

INDEPENDENT VARIABLES: 1 2
DEPENDENT VARIABLE: 3

	COEFF	*STD.ERROR*	*T-STATISTIC*	*VARIANCE*
VARIABLE 1	0.43255	0.04808	9.00000	26.89000
VARIABLE 2	0.00775	0.00059	13.19000	57.83000

	DEG OF FREE	*SUM OF SQS*	*MEAN SQUARE*
REGRESSION	2.00000	11553.10000	5776.55000
ERROR	46.00000	2084.25000	45.30980

S.E. OF ESTIMATE: 6.73126
F-VALUE: 127.49
MULTIPLE R-SQUARED: 84.72
INTERCEPT: -15.9618

$T = 0.43255(E)+0.00775(N^2)-15.9618$

WORK ON RESIDUALS?
YES
PRINT RESIDUALS?
YES

	OBSERVED	ESTIMATED	RESIDUAL
OBSERV 1	0.41670	-10.86140	11.27800
OBSERV 2	1.00000	-8.53660	9.53660
OBSERV 3	1.96670	-4.66200	6.62860
OBSERV 4	3.20000	0.76250	2.43750
OBSERV 5	4.75000	7.73690	-2.98690
OBSERV 6	6.58330	16.26110	-9.67780
OBSERV 7	9.43330	26.33510	-16.90180
OBSERV 8	0.91670	-6.53590	7.45250
OBSERV 9	2.26670	-4.21110	6.47780
OBSERV10	4.25000	-0.33640	4.58640
OBSERV11	6.88330	5.08800	1.79530
OBSERV12	10.15000	12.06240	-1.91240
OBSERV13	14.20000	20.58660	-6.38660
OBSERV14	19.01670	30.66070	-11.64400
OBSERV15	1.28330	-2.21040	3.49370
OBSERV16	3.31670	0.11440	3.20220
OBSERV17	6.30000	3.98910	2.31090
OBSERV18	10.38330	9.41360	0.96980
OBSERV19	15.46670	16.38790	-0.92120
OBSERV20	21.53330	24.91210	-3.37880
OBSERV21	28.96670	34.98620	-6.01950
OBSERV22	1.75000	2.11520	-0.36520
OBSERV23	4.40000	4.43990	-0.03990
OBSERV24	8.51670	8.31460	0.20210
OBSERV25	13.78330	13.73910	0.04430
OBSERV26	20.78330	20.71340	0.06990
OBSERV27	29.16670	29.23760	-0.07090
OBSERV28	39.08330	39.31170	-0.22830
OBSERV29	2.18330	6.44070	-4.25730
OBSERV30	5.53330	8.76540	-3.23210
OBSERV31	10.65000	12.64010	-1.99010
OBSERV32	17.45000	18.06460	-0.61460
OBSERV33	26.25000	25.03890	1.21110
OBSERV34	36.66670	33.56310	3.10350
OBSERV35	49.23330	43.63720	5.59610
OBSERV36	2.60000	10.76620	-8.16620
OBSERV37	6.80000	13.09100	-6.29100
OBSERV38	12.85000	16.96560	-4.11560
OBSERV39	20.83330	22.39010	-1.55680
OBSERV40	31.40000	29.36440	2.03560
OBSERV41	43.95000	37.88860	6.06140
OBSERV42	59.16670	47.96270	11.20400
OBSERV43	3.13330	15.09170	-11.95840
OBSERV44	8.03330	17.41650	-9.38310
OBSERV45	15.15000	21.29110	-6.14110
OBSERV46	24.78330	26.71560	-1.93230
OBSERV47	36.93330	33.69000	3.24340
OBSERV48	52.11670	42.21420	9.90250
OBSERV49	69.61670	52.28820	17.32850

RUN TEST ON RESIDUALS?
YES

SUM OF RESIDUALS: -7.43866E-5
SUM OF SQUARES OF RESIDUALS: 2084.25
DURBAN-WATSON STATISTIC: 1.005

```
      ∇MCONVERT[□]∇
     ∇M←MCONVERT MATRIX;E;N;ETEMP;NTEMP
[1]    M←⍳0//E←1
[2]    ELOOP: N←1//ETEMP←10×E
[3]    NLOOP: NTEMP←10×N
[4]    M←M,ETEMP,(NTEMP×NTEMP),(ETEMP×NTEMP×NTEMP),MATRIX[E;N]
[5]    →(7≥N←N+1)⍴NLOOP
[6]    →(7≥E←E+1)⍴ELOOP
[7]    M←49 4⍴M
      ∇
```

EXPTIMES

E \ N	10	20	30	40	50	60	70
10	0.417	1.000	1.967	3.200	4.750	6.583	9.433
20	0.917	2.267	4.250	6.883	10.150	14.200	19.017
30	1.283	3.317	6.300	10.383	15.467	21.533	28.967
40	1.750	4.400	8.517	13.783	20.783	29.167	39.083
50	2.183	5.533	10.650	17.450	26.250	36.667	49.233
60	2.600	6.800	12.850	20.833	31.400	43.950	59.167
70	3.133	8.033	15.150	24.783	36.933	52.117	69.617

Encryption Times in CPU Minutes

Encryption Time Regressions on Joint Parameters E, N^2, and EN^2

REGMAT

E	N^2	EN^2	Time CPU Mins.
10.000	100.000	1000.000	0.417
10.000	400.000	4000.000	1.000
10.000	900.000	9000.000	1.967
10.000	1600.000	16000.000	3.200
10.000	2500.000	25000.000	4.750
10.000	3600.000	36000.000	6.583
10.000	4900.000	49000.000	9.433
20.000	100.000	2000.000	0.917
20.000	400.000	8000.000	2.267
20.000	900.000	18000.000	4.250
20.000	1600.000	32000.000	6.883
20.000	2500.000	50000.000	10.150
20.000	3600.000	72000.000	14.200
20.000	4900.000	98000.000	19.017
30.000	100.000	3000.000	1.283
30.000	400.000	12000.000	3.317
30.000	900.000	27000.000	6.300
30.000	1600.000	48000.000	10.383
30.000	2500.000	75000.000	15.467
30.000	3600.000	108000.000	21.533
30.000	4900.000	147000.000	28.967
40.000	100.000	4000.000	1.750
40.000	400.000	16000.000	4.400
40.000	900.000	36000.000	8.517
40.000	1600.000	64000.000	13.783
40.000	2500.000	100000.000	20.783
40.000	3600.000	144000.000	29.167
40.000	4900.000	196000.000	39.083
50.000	100.000	5000.000	2.183
50.000	400.000	20000.000	5.533
50.000	900.000	45000.000	10.650
50.000	1600.000	80000.000	17.450
50.000	2500.000	125000.000	26.250
50.000	3600.000	180000.000	36.667
50.000	4900.000	245000.000	49.233
60.000	100.000	6000.000	2.600
60.000	400.000	24000.000	6.800
60.000	900.000	54000.000	12.850
60.000	1600.000	96000.000	20.833
60.000	2500.000	150000.000	31.400
60.000	3600.000	216000.000	43.950
60.000	4900.000	294000.000	59.167
70.000	100.000	7000.000	3.133
70.000	400.000	28000.000	8.033
70.000	900.000	63000.000	15.150
70.000	1600.000	112000.000	24.783
70.000	2500.000	175000.000	36.933
70.000	3600.000	252000.000	52.117
70.000	4900.000	343000.000	69.617

```
      MLTREG[REGMAT]
LIST DATA?
NO
PRINT BASIC STATISTICS?
YES
```

	MEAN	VARIANCE	STD.DEV.
VAR 1	40.000000	408.333000	20.207300
VAR 2	2000.000000	273583.0000	1654.040000
VAR 3	80000.000000	71050000000.	84291.200000
VAR 4	16.838800	284.111000	16.855600

```
SIMPLE CORRELATIONS?
YES
```

	VAR 1	VAR 2	VAR 3	VAR 4
VAR 1	1.00000	0.00000	0.47946	0.51856
VAR 2	0.00000	1.00000	0.78492	0.76043
VAR 3	0.47946	0.78492	1.00000	0.99883
VAR 4	0.51856	0.76043	0.99883	1.00000

```
CONTINUE?
YES
PARTIAL CORRELATIONS?
NO


ENTER NAME FOR REGRESSION TITLE:
ENCRYPTION TIME AS FUNCTION OF DIGITS IN EXPONENT AND MODULUS
ENTER LIST OF INDEPENDENT VARIABLES:
1 2 3
ENTER DEPENDENT VARIABLE:
4
STEPWISE REGRESSION?
NO
```

ENCRYPTION TIME AS FUNCTION OF DIGITS IN EXPONENT AND MODULUS

$$T = F(E, N^2, EN^2)$$

INDEPENDENT VARIABLES: 1 2 3
DEPENDENT VARIABLE: 4

	COEFF	*STD.ERROR*	*T-STATISTIC*	*VARIANCE*
VARIABLE 1	0.03443	0.00282	12.23000	26.89000
VARIABLE 2	-0.00021	0.00005	-4.38000	57.83000
VARIABLE 3	0.00020	0.00000	182.75000	15.26000

	DEG OF FREE	*SUM OF SQS*	*MEAN SQUARE*
REGRESSION	3.00000	13634.60000	4544.85000
ERROR	45.00000	2.80469	0.06233

S.E. OF ESTIMATE: 0.24965
F-VALUE: 72920.1
MULTIPLE R-SQUARED: 99.98
INTERCEPT: -0.03696

WORK ON RESIDUALS?
YES
PRINT RESIDUALS?
YES

$$T = 0.03443(E) - 0.00021(N^2) + 0.0002(EN^2) - 0.03696$$

	OBSERVED	ESTIMATED	RESIDUAL
OBSERV 1	0.41670	0.48510	-0.06840
OBSERV 2	1.00000	1.01830	-0.01830
OBSERV 3	1.96670	1.90700	0.05960
OBSERV 4	3.20000	3.15130	0.04870
OBSERV 5	4.75000	4.75100	-0.00100
OBSERV 6	6.58330	6.70620	-0.12280
OBSERV 7	9.43330	9.01690	0.41650
OBSERV 8	0.91670	1.02840	-0.11180
OBSERV 9	2.26670	2.15890	0.10780
OBSERV10	4.25000	4.04290	0.20710
OBSERV11	6.88330	6.68050	0.20280
OBSERV12	10.150000	10.071800	0.07820
OBSERV13	14.200000	14.216700	-0.01670
OBSERV14	19.016700	19.115100	-0.09850
OBSERV15	1.28330	1.57180	-0.28850
OBSERV16	3.31670	3.29940	0.01730
OBSERV17	6.30000	6.17870	0.12130
OBSERV18	10.383300	10.209800	0.17350
OBSERV19	15.466700	15.392600	0.07410
OBSERV20	21.533300	21.727100	-0.19380
OBSERV21	28.966700	29.213400	-0.24670
OBSERV22	1.75000	2.11510	-0.36510
OBSERV23	4.40000	4.43990	-0.03990
OBSERV24	8.51670	8.31460	0.20210
OBSERV25	13.783300	13.739100	0.04430
OBSERV26	20.783300	20.713400	0.06990
OBSERV27	29.166700	29.237600	-0.07100
OBSERV28	39.083300	39.311700	-0.22830
OBSERV29	2.18330	2.65850	-0.47520
OBSERV30	5.53330	5.58050	-0.04710
OBSERV31	10.650000	10.450400	0.19960
OBSERV32	17.450000	17.268300	0.18170
OBSERV33	26.250000	26.034200	0.21580
OBSERV34	36.666700	36.748100	-0.08150
OBSERV35	49.233300	49.410000	-0.17660
OBSERV36	2.60000	3.20190	-0.60190
OBSERV37	6.80000	6.72100	0.07900
OBSERV38	12.850000	12.586300	0.26370
OBSERV39	20.833300	20.797600	0.03570
OBSERV40	31.400000	31.355100	0.04490
OBSERV41	43.950000	44.258600	-0.30860
OBSERV42	59.166700	59.508200	-0.34160
OBSERV43	3.13330	3.74520	-0.61190
OBSERV44	8.03330	7.86150	0.17180
OBSERV45	15.150000	14.722100	0.42790
OBSERV46	24.783300	24.326900	0.45650
OBSERV47	36.933300	36.675900	0.25750
OBSERV48	52.116700	51.769100	0.34760
OBSERV49	69.616700	69.606500	0.01020

RUN TEST ON RESIDUALS?
YES

SUM OF RESIDUALS: -0.000111282
SUM OF SQUARES OF RESIDUALS: 2.8168
DURBAN-WATSON STATISTIC: 1.185

APPENDIX IV

```
     ∇W←U PLUS V;J;K;MAX
[1]    ⍝ MULTI-PRECISION ADDITION  U AND V VECTORS
[2]    ⍝ REF. ALGORITHM A, KNUTH VOL. 2, PG. 231
[3]    MAX←(J←(⍴U)[1])⌈K←(⍴V)[1]
[4]    U←((MAX-J)⍴0),U//V←((MAX-K)⍴0),V
[5]    K←0//W←⍳0
[6]    LOOP: W←(10|SUM←U[MAX]+V[MAX]+K),W
[7]    K←⌊SUM÷10
[8]    →(0<MAX←MAX-1)⍴LOOP
[9]    W←ZTRIM K,W
     ∇

     ∇W←U MINUS V;J;K;N;KT;T;MAX
[1]    ⍝ MULTI-PRECISION SUBTRACTION.  U AND V VECTORS (U≥V)
[2]    ⍝ REF. ALGORITHM S, KNUTH VOL. 2, PG. 232
[3]    MAX←(J←(⍴U)[1])⌈K←(⍴V)[1]
[4]    U←((MAX-J)⍴0),U//V←((MAX-K)⍴0),V
[5]    W←⍳K←0
[6]    LOOP: W←((10×KT←T<0)+T←U[MAX]-V[MAX]+K),W
[7]    K←KT
[8]    →(0<MAX←MAX-1)⍴LOOP
[9]    W←ZTRIM W
     ∇

     ∇W←U SMULT D;CARRY;T;N;I
[1]    ⍝ MULTI-PRECISION DIVISION.  U IS VECTOR, D IS SINGLE DIGI
[2]    N←(⍴U)[1]//W←⍳0
[3]    I←N//CARRY←0
[4]    LOOP: T←CARRY+U[I]×D
[5]    W←(10|T),W//CARRY←⌊T÷10
[6]    →(0<I←I-1)⍴LOOP
[7]    W←CARRY,W
[8]    W←ZTRIM W
     ∇

     ∇W←U MULT V;M;N;I;K;T
[1]    ⍝ MULTI-PRECISION MULTIPLICATION.  U AND V ARE VECTORS
[2]    ⍝ REF. ALGORITHM M, KNUTH VOL. 2, PG. 233
[3]    M1: N←(⍴U)[1]//J←M←(⍴V)[1]
[4]        W←(M+N)⍴0
[5]    M3: I←N//K←0
[6]    M4: T←K+W[I+J]+U[I]×V[J]
[7]        W[I+J]←10|T//K←⌊T÷10
[8]    M5:  →(0<I←I-1)⍴M4
[9]        W[J]←K
[10]   M6:  →(0<J←J-1)⍴M3
[11]   W←ZTRIM W
     ∇

     ∇W←U SDIV D;I;N;REM
[1]    ⍝ MULTI-PRECISION DIVISION. U IS VECTOR, D IS SINGLE DIGIT
[2]    N←(⍴U)[1]//W←⍳0
[3]    I←1//REM←0
[4]    LOOP: REM←U[I]+10×REM
[5]    W←W,⌊REM÷D
[6]    REM←D|REM
[7]    →(N≥I←I+1)⍴LOOP
[8]    W←(ZTRIM W),REM
     ∇
```

```
     ∇W←U DIV V;N;J;DIVIDEND;Q;PROD
[1]    ⍝ MULTI-PRECISION QUOTIENT (U÷V)
[2]    N←(⍴U)[1]//W←⍳0
[3]    J←0//DIVIDEND←⍳0
[4]    DLOOP: →(N<J←J+1)⍴DONE
[5]    DIVIDEND←DIVIDEND,U[J]
[6]    →(~DIVIDEND LESSTHAN V)⍴QFIND
[7]    W←ZTRIM W,0
[8]    →DLOOP
[9]    QFIND: Q←10-5×DIVIDEND LESSTHAN V SMULT 5
[10]   QLOOP: →(DIVIDEND LESSTHAN PROD←V SMULT Q←Q-1)⍴QLOOP
[11]   W←W,Q
[12]   DIVIDEND←DIVIDEND MINUS PROD
[13]   →DLOOP
[14]   DONE: W←ZTRIM W
     ∇


     ∇REM←U MOD V;N;J;PROD;Q
[1]    ⍝ MULTI-PRECISION REMAINDER (U÷V)
[2]    REM←U
[3]    →(U LESSTHAN V)⍴DONE
[4]    N←(⍴U)[1]//J←0//REM←⍳0
[5]    DLOOP: →(N<J←J+1)⍴DONE
[6]    REM←REM,U[J]
[7]    →(REM LESSTHAN V)⍴DLOOP
[8]    Q←10-5×REM LESSTHAN V SMULT 5
[9]    QLOOP: →(REM LESSTHAN PROD←V SMULT Q←Q-1)⍴QLOOP
[10]   REM←REM MINUS PROD
[11]   →DLOOP
[12]   DONE: REM←ZTRIM REM
     ∇


     ∇C←M EXPON E;K;I
[1]    ⍝ COMPUTES M TO E-TH POWER
[2]    K←(⍴E←DBCONVERT E)[1]
[3]    C←,1//I←1
[4]    LOOP: C←C MULT C
[5]    →(E[I]=0)⍴NEXTBIT
[6]    C←C MULT M
[7]    NEXTBIT: →(K≥I←I+1)⍴LOOP
     ∇


     ∇C←ENCRYPT[M;E;N];K;I
[1]    ⍝ COMPUTES M TO E-TH POWER, MODULO N
[2]    K←(⍴E←DBCONVERT E)[1]
[3]    C←,1//I←1
[4]    LOOP: C←(C MULT C) MOD N
[5]    →(E[I]=0)⍴NEXTBIT
[6]    C←(C MULT M) MOD N
[7]    NEXTBIT: →(K≥I←I+1)⍴LOOP
     ∇


     ∇BIN←DBCONVERT DEC;T;N
[1]    ⍝ MULTI-PRECISION RADIX CONVERSION (REF. KNUTH VOL 2, PG. 28
[2]    N←(⍴DEC)[1]//BIN←⍳0//T←DEC
[3]    LOOP: T←T SDIV 2
[4]    BIN←(¯1↑T),BIN
[5]    →(0≠+/T←¯1↓T)⍴LOOP
     ∇
```

```
     ∇U1←U3 MULTINV V3;MOD;V1;T1;T3;SGNU1;SGNV1;SGNT1;Q;PROD
[1]    ⍝ COMPUTES MULTIPLICATIVE INVERSE MOD U3 OF V3
[2]    ⍝ REF ALGORITHM X (EXTENDED EUCLIDEAN) KNUTH VOL. 2, PG. 30
[3]    MOD←U3
[4]    X1: U1←,0//SGNU1←1//V1←,1//SGNV1←1
[5]    X2: →(∧/V3=0)⍴ECOMP
[6]    X3: Q←U3 DIV V3
[7]    T3←U3 MINUS V3 MULT Q
[8]    U3←V3
[9]    V3←T3
[10]   PROD←V1 MULT Q
[11]   →(SGNV1≠SGNU1)⍴DIFFSGN
[12]   →(U1 LESSTHAN PROD)⍴USMALL
[13]   SGNT1←SGNU1
[14]   T1←U1 MINUS PROD
[15]   →SWAP
[16]   USMALL: SGNT1←¯1×SGNU1
[17]   T1←PROD MINUS U1
[18]   →SWAP
[19]   DIFFSGN: SGNT1←SGNU1
[20]   T1←U1 PLUS PROD
[21]   SWAP: U1←V1//SGNU1←SGNV1
[22]   V1←T1//SGNV1←SGNT1
[23]   →X2
[24]   ⍝ U1 IS VALUE OF INVERSE. NORMALIZE
[25]   ECOMP: →(1=SGNU1)⍴0
[26]   U1←MOD MINUS U1
     ∇


     ∇F←U GCD V;R
[1]    ⍝ MODERN EUCLIDEAN ALGORITHM FOR GCD
[2]    ⍝ MULTI-PRECISION REF. KNUTH VOL. 2, PG. 296 (ALGORITHM A)
[3]    LOOP: →(∧/0=V)⍴DONE
[4]    R←U MOD V
[5]    U←V
[6]    V←R
[7]    →LOOP
[8]    DONE: F←U
     ∇


     ∇MULTP←A JACOBI B;EXP;R
[1]    ⍝ COMPUTES JACOBI SYMBOL J(A,B)
[2]    MULTP←1
[3]    TEST: →(A EQUALTO ,1)⍴0
[4]    →(0≠2|¯1↑A)⍴ODD
[5]    EXP←¯1↓((B MULT B) MINUS ,1) SDIV 8
[6]    A←¯1↓A SDIV 2
[7]    →MULTCOMP
[8]    ODD: EXP←¯1↓((A MINUS ,1) MULT B MINUS ,1) SDIV 4
[9]    R←A
[10]   A←B MOD A
[11]   B←R
[12]   MULTCOMP: MULTP←MULTP×(¯1 1)[1+0=2|¯1↑EXP]
[13]   →TEST
     ∇


     ∇Z←DISPLAY N
[1]    Z←'0123456789'[1+N]
     ∇
```

```
     ∇NVEC←TNCONVERT TEXT;ABET;POSN
[1]    ABET←' ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.,:;?()⍵+-×/=≠<>≤≥'
[2]    NVEC←⍳0
[3]    LOOP: POSN←¯1+ABET⍳1↑TEXT
[4]    NVEC←NVEC,(⌊POSN÷10),10|POSN
[5]    →(0≠ρTEXT←1↓TEXT)ρLOOP
     ∇

     ∇TEXT←NTCONVERT NVEC;ABET;INDX
[1]    ABET←' ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.,:;?()⍵+-×/=≠<>≤≥'
[2]    TEXT←⍳0
[3]    NVEC←((0≠2|(ρNVEC)[1])ρ0),NVEC
[4]    LOOP: INDX←1++/(10 1)×2↑NVEC
[5]    TEXT←TEXT,ABET[INDX]
[6]    →(0≠ρNVEC←2↓NVEC)ρLOOP
     ∇

     ∇Z←ZTRIM VEC
[1]    ⍝ TRIM LEADING ZEROES OFF MULTI-PRECISION VECTOR
[2]    Z←VEC
[3]    LOOP: →((0≠1↑Z)∨1=ρZ)ρ0
[4]    Z←1↓Z
[5]    →LOOP
     ∇

     ∇TIMEIN
[1]    TIME←⎕TIME
     ∇

     ∇TIMEOUT
[1]    ,'ELAPSED TIME = ',((+/(3600 60 1)×(⎕TIME-TIME)[4 5 6])÷60),' M
     ∇

     ∇TMATRIX←TIMETEST MDIGITS;ND;ED;M;E;N;TIME;R
[1]    TIMEIN
[2]    ⍝ THIS FUNCTION TESTS THE TIME NEEDED FOR EXPONENTIATION (MULTI
[3]    TMATRIX←⍳0//ED←10
[4]    M←MDIGITS⍴5
[5]    ELOOP: E←ED⍴5//ND←10
[6]    NLOOP: N←ND⍴5
[7]    TIME←⎕TIME
[8]    R←ENCRYPT[M;E;N]
[9]    TMATRIX←TMATRIX,((+/(3600 60 1)×(⎕TIME-TIME)[4 5 6])÷60)
[10]   ⎕←'*'
[11]   →(70≥ND←ND+10)ρNLOOP
[12]   ' '
[13]   →(70≥ED←ED+10)ρELOOP
[14]   TMATRIX←10 10⍴TMATRIX
[15]   ⎕CRLF,'TIME TEST DONE',⎕CRLF
[16]   TIMEOUT
     ∇
```

```
     ∇Z←PRIME P;TP;J;E;F;PM1;I
[1]    ⍝ TESTS P FOR PRIMALITY USING SOLVAY-STRASSEN ALGORITHM
[2]    Z←0//I←1//PM1←P MINUS ,1
[3]    E←-1↓PM1 SDIV 2
[4]    LOOP: TP←-1+?1+PM1
[5]    →(∧/TP=0)ρLOOP
[6]    →(~(,1) EQUALTO ,TP GCD P)ρ0
[7]    J←TP JACOBI P
[8]    F←ENCRYPT[TP;E;P]
[9]    ⍝'TP = ',TP,'  J = ',J,'  F = ',F
[10]   →(J=-1)ρMODCHECK
[11]   →(F EQUALTO ,1)ρNEXTTEST
[12]   →0
[13]   MODCHECK: →(~F EQUALTO PM1)ρ0
[14]   NEXTTEST: →(25≥I←I+1)ρLOOP
[15]   Z←1
     ∇

     ∇Z←PRIMECHK P;F;PM1
[1]    ⍝ TESTS P FOR PRIMALITY USING FERMAT'S THEOREM
[2]    ⍝ REFERENCE KNUTH VOL. 2, PG. 347
[3]    Z←0//PM1←P MINUS ,1
[4]    F←ENCRYPT[,2;PM1;P]
[5]    →(~F EQUALTO ,1)ρ0
[6]    Z←1
     ∇

     ∇P←PRIMEGEN NDIGITS;I;HEAD;TAIL
[1]    ⍝ GENERATES RANDOM PRIME OF N DIGITS (N≥2)
[2]    I←0
[3]    TIMEIN
[4]    PGEN: HEAD←?9//TAIL←(1 3 5 7 9)[?5]
[5]    P←HEAD,(-1+?(NDIGITS-2)ρ10),TAIL
[6]    I←I+1
[7]    →(~PRIMECHK P)ρPGEN
[8]    ' '
[9]    'NO. TESTED = ',I
[10]   TIMEOUT
     ∇
```