1975

# An Experiment Comparing Fortran Programming Times with the Software Physics Hypothesis

R. D. Gordon

M. H. Halstead

Report Number:
75-167

# AN EXPERIMENT COMPARING FORTRAN PROGRAMMING TIMES WITH THE SOFTWARE PHYSICS HYPOTHESIS

R. D. Gordon
M. H. Halstead

TR 167
October 17, 1975

## INTRODUCTION

Recent discoveries in the area of Algorithm Structure or Software Physics [1-25] have produced a number of hypotheses. One of these relates the number of elementary mental discriminations required to implement an algorithm to measurable properties of that algorithm, and the results of one set of experiments confirming this relationship have been published [16]. That publication, while significant, made no claim to finality, suggesting instead that further experiments were warranted. This paper will present the results of a second set of experiments, having the advantages of being conducted in a single implementation language, Fortran, from problem specifications readily available in computer textbooks.

Section I of this report presents the timing hypothesis, and the elementary equations upon which it rests. Section II presents the details of the experiment and the results which were obtained, and Section III contains an analysis of the data.

## SECTION I - TIMING HYPOTHESIS

Measurable properties of any implementation of any algorithm include:

$\eta_1$ = The count of distinct operators

$\eta_2$ = The count of distinct operands (variables or constants)

$N_1$ = Total uses of operators

$N_2$ = Total uses of operands

The vocabulary, $\eta$, is given by:

$$\eta = \eta_1 + \eta_2 \qquad (1)$$

and the length, N, is:

$$N = N_1 + N_2 \qquad (2)$$

From these properties, it is possible to obtain the volume, V, in bits, as:

$$V = N \log_2 \eta \qquad (3)$$

and the implementation level, L, where $L \leq 1$, as:

$$L = \frac{\eta_1^*}{\eta_1} \; \frac{\eta_2}{N_2} \qquad (4)$$

where $\eta_1^*$ , the minimum possible number of operators, will equal 2 for most algorithms. (One for the name of a function, plus one for a grouping symbol operator). It has been shown [4] that the product L x V is invariant under translation from one language to another, and that for programs without impurities [3,6,8]:

$$N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \qquad (5)$$

From this point, the following nine steps yield the timing equation:

1. A program consists of N selections from $\eta$ elements.

2. A binary search of $\eta$ elements requires $\log_2 \eta$ comparisons.
3. A program is generated by making $N \log_2 \eta$ comparisons.
4. Therefore, the volume, V, is a count of the number of comparisons required.
5. The number of elementary mental discriminations required to complete one comparison measures the difficulty of the task.
6. The level, L, is the reciprocal of the difficulty.
7. Therefore, E, the count of elementary mental discriminations required to generate a program, is given by:

$$E = \frac{V}{L} \qquad (6)$$

8. S, the speed with which the brain makes elementary mental discriminations can be obtained from psychology [26] as:

$$5 \leq S \leq 20 \text{ discriminations per second.}$$

9. Therefore, the time to generate a <u>preconceived</u> program, by a <u>concentrating</u> programmer, <u>fluent</u> in a language, is:

$$\hat{T} = \frac{V}{SL} \qquad (7)$$

Equation 7 may be expressed in more basic terms by substituting for V from equation 3, and for L from equation 4, with $\eta_1^* = 2$, giving:

$$\hat{T} = \frac{\eta_1 N_2 N \log_2 \eta}{2\, S\, \eta_2} \qquad (8)$$

The effect of possible impurities [5] may be eliminated from equation 8 by substituting for N from equation 5. Letting S = 60 x 18 = 1080 will then give, for time in minutes:

$$\hat{T} = \frac{\eta_1 N_2 (\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2) \log_2 \eta}{2160\, \eta_2} \qquad (9)$$

Each of the variables on the right hand side of equation 9 can be readily measured (or counted) in any computer program, and the experiment described in the next section was designed to compare results from that equation with observed programming times.

## SECTION II. EXPERIMENTAL PROCEDURE

Eleven problems were arbitrarily selected from two published sources. In selecting candidates for the experiment, problems were sought which were stated in a non-procedural form. Further, the problem statement had to be complete. That is, in the course of solving a particular problem, specific laws of physics, mathematics, etc. would not have to be derived. The problems finally selected were taken from Knuth [27], and from Maurer and Williams [28], and cover a wide range of topics including character manipulation, list processing, simulation experiments and mathematical analysis. The source of each problem statement is cited in Table 1.

On each of eleven days, one of these problems was implemented by the senior author. In order to maintain a consistent level of performance all work was conducted in a quiet room, free from distractions, during the same period of the day. The time required to fully implement the problem was obtained. This total time included the number of minutes spent reading the statement of the problem, preparing flowcharts and writing preliminary versions of the code, writing the final version of the code, desk checking, and the time spent working to correct errors in the program. Time to keypunch was not included.

For a number of reasons, including availability and fluency, all of the algorithms were implemented in Fortran. In the course of solving a problem the correctness of the implementation was checked by executing a sufficiently complex test case for which a correct answer was known. In some cases the solution to a problem was written as a subroutine and testing required that a main routine be written. In such a case only the preparation of the subroutine was considered for the experiment.

In addition, several implementations made use of subroutines previously written. Such routines were also not included. The complete text of each of the eleven programs is included in Appendix A.

After each program was completed, a careful count was made to determine values of $\eta_1$, $\eta_2$, $N_1$ and $N_2$. In obtaining these values all read, write, declarative statements and comments were ignored. The results are shown in Table 1.

TABLE 1. EXPERIMENTAL DATA

| No. | Program Specifications | | | Software Parameters | | | | Implementation |
|---|---|---|---|---|---|---|---|---|
| | Ref.* | Page | Problem | $\eta_1$ | $\eta_2$ | $N_1$ | $N_2$ | Time-Minutes |
| G1 | K | 158 | 21 | 15 | 11 | 59 | 51 | 19 |
| G2 | K | 159 | 23 | 20 | 24 | 231 | 197 | 92 |
| G3 | K | 196 | 7 | 16 | 12 | 64 | 49 | 16 |
| G4 | K | 377 | 17 | 19 | 21 | 131 | 113 | 39 |
| G5 | K | 158 | 22 | 7 | 10 | 38 | 35 | 21 |
| G6 | K | 154 | 10 | 9 | 14 | 69 | 62 | 30 |
| G7 | M | 32 | 3.2.21 | 12 | 8 | 30 | 23 | 5 |
| G8 | M | 32 | 3.2.23 | 19 | 15 | 73 | 55 | 24 |
| G9 | M | 88 | 8.3.2 | 22 | 32 | 124 | 104 | 43 |
| G10 | M | 89 | 8.3.4 | 25 | 34 | 261 | 222 | 91 |
| G11 | M | 27 | 3.2.4 | 14 | 10 | 29 | 21 | 5 |

*K = Knuth [27], M = Maurer and Williams [28].

## SECTION III - ANALYSIS OF THE DATA

The programming time predicted by theory was obtained for each program by applying equation 9 to the data in Table 1. This result, T, can be compared with the observed value, T, in Table 2. In addition, a count of the number of statements in each program was obtained, and the programs were ordered according to these values.

The average of the calculated values, 34 minutes, is fortuitously close to the observed value, 35 minutes. The coefficient of correlation is 0.934, only slightly smaller than the value of 0.952 reported in an earlier experiment [16]. In further agreement with that experiment, the correlation between length and observed times, 0.887, is lower than between observed and calculated times.

In conclusion, it may again be observed that one more set of experimental data do not contradict the simple hypothesis. As a result, further carefully controlled experiments by others would appear to be warranted.

TABLE 2. EXPERIMENTAL RESULTS

| Program Number | Statement Count | Programming Time-Minutes | |
|---|---|---|---|
| | | T observed | T Equ. 9 |
| G7 | 7 | 5 | 4.6 |
| G11 | 8 | 5 | 5.4 |
| G5 | 11 | 21 | 2.5 |
| G6 | 15 | 30 | 6.8 |
| G3 | 18 | 16 | 15.6 |
| G1 | 18 | 19 | 14.6 |
| G8 | 18 | 24 | 22.9 |
| G4 | 32 | 39 | 43.6 |
| G2 | 36 | 92 | 81.5 |
| G9 | 38 | 43 | 49.2 |
| G10 | 59 | 91 | 128.5 |
| Means | | 35.0 | 34.1 |

REFERENCES:

[1] Bayer, Rudolf, "A Theoretical Study of Halstead's Software Phenomenon," CSD Tech. Rept. No. 69, Purdue, May 1972.

[2] Bayer, Rudolf, "On Program Volume and Program Modularization," CSD Tech. Rept. No. 105, Purdue, Sept. 1973.

[3] Bohrer, Robert, "Halstead's Criteria and Statistical Algorithms," Proc. 8th Computer Science/Statistics Interface Symposium, Los Angeles, Feb. 1975.

[4] Bulut, Necdet, "Invariant Properties of Algorithms," Ph.D. Thesis, Purdue, Aug. 1973.

[5] Bulut, Necdet and M. H. Halstead, "Impurities Found in Algorithm Implementations," ACM SIGPLAN Notices, 9, 3, March 1974.

[6] Bulut, Necdet, M. H. Halstead and Rudolf Bayer, "The Experimental Verification of a Structural Property of Fortran Programs," Proc. ACM Annual Conference, San Diego, 1974.

[7] Funami, Yasao and M. H. Halstead, "Software Physics Analysis of Akayama's Debug Data," CSD Tech. Rept. No. 144, Purdue, May 1975.

[8] Halstead, M. H., "Natural Laws Controlling Algorithmic Structure," ACM SIGPLAN Notices, 7, 2, Feb. 1972.

[9] Halstead, M. H., "A Theoretical Relationship Between Mental Work and Machine Language Programming," CSD Tech. Rept. No. 67, Purdue May 1972.

[10] Halstead, M. H. and Rudolf Bayer, "Algorithm Dynamics," Proc. ACM Annual Conference, Atlanta, 1973.

[11] Halstead, M. H. "An Experimental Determination of the 'Purity' of a Trivial Algorithm", ACM SIGME Performance Evaluation Review, 2, 1, March 1973.

[12] Halstead, M. H., "Language Level, A Missing Concept in Information Theory," ACM SIGME Performance Evaluation Review, 2, 1, March 1973.

[13] Halstead, M. H. and P. M. Zislis, "Experimental Verification of Two Theorems of Software Physics," CSD Tech. Rept. No. 97, Purdue, June 1973.

[14] Halstead, M. H., "Software Physics Comparison of a Sample Program in DSL ALPHA and COBOL," IBM Research Report No. RJ 1460, Oct. 1974.

[15] Halstead, M. H., "Software Physics: Basic Principles," IBM Research Report No. RJ 1582, May 1975.

[16] Halstead, M. H., "Toward a Theoretical Basis for Estimating Programming Efforts," Proc. ACM Annual Conference, Minneapolis, 1975.

[17] Kennedy, Dale and Roger Bruning, "Childrens Descriptions of Complex Objects," Report 505-68-6939, Univ. of Nebraska, Lincoln, Oct. 1974.

[18] Kulm, Gerald, "Information Content" An Alternative Measure of Reading Complexity," American Psychological Association Annual Meeting, New Orleans, Aug. 1974.

[19] Ostapko, D. L., "On Deriving a Relation Between Circuits and Input/Output by Analyzing an Equivalent Program," ACM SIGPLAN Notices, 8, 6, June 1974.

[20] Ostapko, D. L., "Analysis of Algorithms Implemented in Software and Hardware," Proc. ACM Annual Conference, San Diego, 1974.

[21] Zislis, Paul, "An Experiment in Algorithm Implementation," CSD Tech. Rept. No. 96, Purdue, June 1973.

[22] Zislis, Paul M., "Semantic Decomposition of Computer Programs: An Aid to Program Testing," ACTA Informatica 4, pp. 245-269, 1975.

[23] Zweben, S. H., "Software Physics: Resolution of an Ambiguity in the Counting Procedure," CSD Tech. Rept. No. 93, Purdue, April 1973.

[24] Zweben, S. H., "The Internal Structure of Algorithms," Ph.D. Thesis, Purdue, May 1974.

[25] Zweben, S. H., "A Recent Approach to the Study of Algorithms," Proc. ACM Annual Conference, San Diego, 1974.

Additional References:

[26] Stroud, John M. "The Fine Structure of Psychological Time," Annals of N. Y. Academy of Sciences, 1966, pp. 623-631.

[27] Knuth, Donald, "The Art of Computer Programming," Addison Wesley Publishing Co., Massachusetts, 1969, Vol. 1.

[28] Maurer, H. A. and M. R. Williams, "A Collection of Programming Problems and Techniques," Prentice-Hall, Inc., New Jersey, 1972.

# APPENDIX A

Final listings of the eleven experimental programs.

```
C
C
C                                                  TEST PROGRAM 1
      DIMENSION MAGIC(23,23)
      DATA MAGIC /529*0/
C                                      BEGIN TOP ROW, CENTER
      N=1
      IR=1
      IC=23/2 + 1
C                                      MARK A SQUARE
  100 MAGIC(IR,IC)=N
C                                      DONE WHEN N=23**2
      IF (N.EQ.529) GO TO 900
      N=N+1
C                                      MOVE UP AND LEFT
      JR=IR-1
      IF (JR.LT.1) JR=23
      JC=IC-1
      IF (JC.LT.1) JC=23
      IF (MAGIC(JR,JC).EQ.0) GO TO 200
C                                      BOX FULL - DROP DOWN
      JR=IR+1
      IF (JR.GT.23) JR=1
      JC=IC
C                                      SET INDICES
  200 IR=JR
      IC=JC
      GO TO 100
C
C     PRINT MAGIC SQUARE
C
  900 DO 920 IR=1,23,1
  920 WRITE (6,1000) (MAGIC(IR,IC), IC=1,23,1)
      STOP
C
 1000 FORMAT(23I4)
      END
```

```
C
C
C                                                              TEST PROGRAM 2
      INTEGER DIGIT(10)
      DATA DIGIT /1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
      DIMENSION MAT(24,24)
      DATA MAT /576*-1/
      INTEGER WHITE(3,5)
      DATA WHITE /1H ,1H ,1H+,
     $            1H ,1H ,1H+,
     $            1H ,1H ,1H+,
     $            1H ,1H ,1H+,
     $            1H+,1H+,1H+/
      INTEGER EDGE(3,5,5)
      DATA EDGE /1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H+,1H+,1H+,
     $           1H ,1H ,1H+,
     $           1H ,1H ,1H+,
     $           1H ,1H ,1H+,
     $           1H ,1H ,1H+,
     $           1H ,1H ,1H+,
     $           1H ,1H ,1H+,
     $           1H ,1H ,1H+,
     $           1H ,1H ,1H+,
     $           1H ,1H ,1H+,
     $           1H+,1H+,1H+,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H ,
     $           1H ,1H ,1H+/
      INTEGER OUT(120,120)
      COMMON OUT
C
      DO 110 IR=2,23,1                         READ INPUT MATRIX
  110 READ (5,1000) (MAT(IR,IC), IC=2,23,1)
C
C     DELETE +1 ADJACENT TO -1
C
      DO 220 IR=2,23,1
      JR=25-IR
      DO 210 IC=2,23,1
      JC=25-IC
      IF (MAT(IR,IC).NE.1) GO TO 200
      IF (MAT(IR-1,IC).EQ.-1 .OR. MAT(IR,IC-1).EQ.-1)
     $    MAT(IR,IC)=-1
C
  200 IF (MAT(JR,JC).NE.1) GO TO 210
      IF (MAT(JR+1,JC).EQ.-1 .OR. MAT(JR,JC+1).EQ.-1)
     $    MAT(JR,JC)=-1
  210 CONTINUE
  220 CONTINUE
```

```
C
C     DEBUG PRINTOUT
C
      DO 300 IR=1,24,1
  300 WRITE (6,9000) (MAT(IR,IC), IC=1,24,1)
C
C     MOVE TO OUTPUT ARRAY
C
      N=0
      DO 380 IR=1,23,1
      JR=(IR-1)*3
      DO 370 IC=1,23,1
      JC=(IC-1)*5
C                                          BLACK SQUARE
      IF (MAT(IR,IC).NE.1) GO TO 325
      DO 310 I=1,3,1
      DO 310 J=1,5,1
  310 OUT(JR+I,JC+J)=1H+
      GO TO 370
C                                          EDGE SQUARE
  325 IF (MAT(IR,IC).EQ.0) GO TO 350
      MODE=1
      IF (MAT(IR,IC+1).EQ.0) MODE=MODE+1
      IF (MAT(IR+1,IC).EQ.0) MODE=MODE+2
      IF (MODE.EQ.1 .AND. MAT(IR+1,IC+1).EQ.0) MODE=5
      DO 330 I=1,3,1
      DO 330 J=1,5,1
  330 OUT(JR+I,JC+J)=EDGE(I,J,MODE)
      GO TO 370
C                                          WHITE SQUARE
  350 DO 360 I=1,3,1
      DO 360 J=1,5,1
  360 OUT(JR+I,JC+J)=WHITE(I,J)
C                                          NUMBERED...
      IF ( .NOT.
     $          (MAT(IR+1,IC).EQ.0 .AND. MAT(IR-1,IC).NE.0   .OR.
     $           MAT(IR,IC-1).NE.0 .AND. MAT(IR,IC+1).EQ.0))  GO TO 370
      N=N+1
      NT=N/10
      NU=N-NT*10
      OUT(JR+1,JC+1)=DIGIT(NT+1)
      OUT(JR+1,JC+2)=DIGIT(NU+1)
  370 CONTINUE
  380 CONTINUE
C
C     OUTPUT PUZZLE
C
      WRITE (6,3000)
      DO 500 I=1,120,1
  500 WRITE (6,2000) (OUT(I,J), J=1,120,1)
      STOP
C
 1000 FORMAT(23I1)
 2000 FORMAT(1X,120A1)
 3000 FORMAT(1H1,/,1HQ)
 9000 FORMAT(24I3)
      END
*EOR
10000111111111111111111
00100011111111111111111
00001011111111111111111
```

```
01000011111111111111111
00010011111111111111111
10000111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
11111111111111111111111
```

TEST PROGRAM 3

```
C
C
C
      INTEGER C, C0, PI, PO
      INTEGER INPUT(8), OUTPUT(8), DIGIT(10)
      DATA OUTPUT /8*1H /
      DATA DIGIT /1H0, 1H1, 1H2, 1H3, 1H4, 1H5, 1H6, 1H7, 1H8, 1H9/
      INTEGER GETCH
      LOGICAL NUMERIC
      NUMERIC(I)=I.GE.1H0 .AND. I.LE.1H9
C                                   GET INPUT
      READ (5,1000) INPUT
 1000 FORMAT(8A10)
      WRITE (6,2000) INPUT
 2000 FORMAT(≠0INPUT IS ≠,8A10)
C
      PI=1                          INITIALIZE TO 1ST CHAR
      PO=1
      C0=GETCH(INPUT,PI)
      PI=PI+1
C
  100 KOUNT=1                       BEGIN COUNTING REPEATS
  200 C=GETCH(INPUT,PI)
      PI=PI+1
C
      IF (C.EQ.1H ) GO TO 200       IGNORE BLANKS
      IF (C.NE.C0) GO TO 300
      KOUNT=KOUNT+1
      GO TO 200
C
C                                   OUTPUT KOUNT-1 IF GT 1
                                    OR IF C0 IS NUMERIC
  300 IF (KOUNT.EQ.1 .AND. .NOT.NUMERIC(C0)) GO TO 400
      CALL PUTCH (DIGIT(KOUNT), OUTPUT, PO)
      PO=PO+1
C
  400 CALL PUTCH (C0, OUTPUT, PO)   OUTPUT THE CHAR
      PO=PO+1
      C0=C
C
      IF (C0.NE.1H.) GO TO 100      . ENDS INPUT AND OUTPUT
      CALL PUTCH (C0, OUTPUT, PO)
      WRITE (6,3000) OUTPUT
 3000 FORMAT(≠0OUTPUT IS ≠,8A10)
      STOP
      END
C
C     THE ROUTINES GETCH AND PUTCH WERE NOT WRITTEN AS PART OF THIS
C     EXPERIMENT.
C
      INTEGER FUNCTION GETCH(IWORD, IPOS)
C
C     THE FUNCTION GETCH RETURNS A 6 BIT CHARACTER FROM A CHARACTER
C     STRING OF POSSIBLY SEVERAL WORDS IN LENGTH.  THE VALUE RETURNED
C     IS LEFT JUSTIFIED, BLANK FILLED.  CHARACTERS ARE NUMBERED LEFT
C     TO RIGHT, 1, 2, ... .
C
C     AUTHOR:  KEVIN KOLIS  (23 JAN 75)
C
      DIMENSION IWORD(1)
C
```

```
      ISUB=(IPOS-1)/10+1
      GETCH=OR(AND(SHIFT(IWORD(ISUB),IPOS*6+54-ISUB*60),
     $           77000000000000000000B),00555555555555555555B)
      RETURN
      END

      SUBROUTINE PUTCH (CHAR, STRING, POS)
C
C     SUBROUTINE PUTCH PLACES A GIVEN CHARACTER INTO A STRING
C     AT THE SPECIFIED POSITION.  CHARACTERS ARE NUMBERED LEFT TO
C     RIGHT, 1, 2, ... .
C
C     AUTHOR:  RONALD GORDON  (24 JAN 75)
C
      INTEGER CHAR, STRING(1), POS
C
      IW=(POS-1)/10+1
      IC=(1-POS+IW*10)*6
      M=SHIFT(00 77 77 77 77 77 77 77 77 77B, IC)
      STRING(IW)=OR( AND(STRING(IW),M), AND(SHIFT(CHAR,IC),COMPL(M)) )
      RETURN
C
      END
*EOR
ABB BEE EEE E44 446 66F 0ZY W22 220 0PQ 999 999 999 R.
```

```
C
C
C                                                        TEST PROGRAM 4
      INTEGER PICK, CARD
      LOGICAL DEBUG
      INTEGER POP
      INTEGER DECK(52)
      INTEGER KOUNT(53)
      DATA KOUNT /53*0/
      INTEGER PILE(13,5)
      COMMON PILE
C
                                           SETUP
      DEBUG=.TRUE.
      PC=RANF(13.0)
C
                                           CONSTRUCT CARD DECK
      K=1
      DO 100 I=1,4,1
      DO 100 J=1,13,1
      DECK(K)=I*100 + J
  100 K=K+1
C
                                           SIMULATE 500 GAMES
      DO 900 N=1,500,1
C
                                           SET UP PILE POINTERS
      DO 150 I=1,13,1
  150 PILE(I,1)=1
C
                                           SHUFFLE THE DECK
      CALL SHUFFLE (DECK,52)
C
                                           DEAL THE CARDS INTO 13 PILES
      K=1
      DO 200 I=1,4,1
      DO 200 J=1,13,1
      CALL PUSH (DECK(K),J)
  200 K=K+1
      IF (DEBUG) WRITE (6,1000) PILE
 1000 FORMAT(≠1≠,5(/,1X,13I5),/)
C
                                           PLAY THE GAME OF CLOCK
      L=53
      PICK=1
  300 CARD=POP(PICK)
      IF (CARD.EQ.0) GO TO 400
      L=L-1
      IF (DEBUG) WRITE (6,2000) CARD, PICK
 2000 FORMAT(I4,≠ PICKED FROM≠,I3)
      PICK=MOD(CARD,100)
      GO TO 300
C
                                           GAME OVER
C                                          L = NO OF CARDS NOT PLAYED + 1
  400 KOUNT(L)=KOUNT(L)+1
      K=L-1
      IF (DEBUG) WRITE (6,3000) K
 3000 FORMAT(/,I3,≠ CARDS LEFT≠)
                                           IF (N.GT.5) DEBUG=.FALSE.
  900 CONTINUE
C
                                           PRINT STATS
      WRITE (6,5000)
      DO 920 I=0,52,1
      PC=FLOAT(KOUNT(I+1))/500.0 * 100.0
  920 WRITE (6,4000) I, KOUNT(I+1), PC
 4000 FORMAT(I7,I12,F11.2)
 5000 FORMAT(≠1CARDS LEFT   N TIMES   PERCENT≠)
```

```
      STOP
      END
      SUBROUTINE PUSH (ITEM, PICK)
      INTEGER PICK
      INTEGER PILE(13,5)
      COMMON PILE
      PILE(PICK,1)=PILE(PICK,1)+1
      PILE(PICK, PILE(PICK,1) ) = ITEM
      RETURN
      END
      INTEGER FUNCTION POP (PICK)
      INTEGER PICK
      INTEGER PILE(13,5)
      COMMON PILE
      IF (PILE(PICK,1).EQ.1) GO TO 100
      POP=PILE(PICK, PILE(PICK,1) )
      PILE(PICK,1)=PILE(PICK,1)-1
      RETURN
C                                            STACK EMPTY
  100 POP=0
      RETURN
      END
C
C     SUBROUTINE SHUFFLE WAS NOT WRITTEN AS PART OF THIS EXPERIMENT.
C
      SUBROUTINE SHUFFLE (LIST, N)
      DIMENSION LIST(N)
C
C     THIS ROUTINE WILL RANDOMLY SHUFFLE A LIST OF ITEMS.
C
C     REF: KNUTH, VOL. 2, P. 125, ALGORITHM P.
C
      J=N
  100 U=RANF(0.0)
      K=FLOAT(J)*U+1.0
      KEEP=LIST(K)
      LIST(K)=LIST(J)
      LIST(J)=KEEP
      J=J-1
      IF (J.GT.1) GO TO 100
      RETURN
      END
```

```
C
C
C                                                                TEST PROGRAM 5
      DIMENSION MAN(100)
      DIMENSION KILL(100)
      READ, N, M
C
      DO 110 I=1,N-1                               CREATE CIRCULAR LIST
  110 MAN(I)=I+1
      MAN(N)=1
C
      L2=N                                         KILL N-1 MEN
      DO 200 K=1,N-1
C
      DO 150 L=1,M                                 COUNT M MEN
      L1=L2
  150 L2=MAN(L2)
C
      KILL(L2)=K                                   KILL CURRENT MAN
  200 MAN(L1)=MAN(L2)
C
      KILL(L1)=N                                   KILL LAST MAN
      WRITE (6,1000) (KILL(I), I=1,N,1)
 1000 FORMAT(40I3)
      STOP
      END
*EOR
8, 4
```

```
C
C
C                                                    TEST PROGRAM 6
C
C     THE MAIN PROGRAM WAS NOT WRITTEN AS PART OF THIS EXPERIMENT.
      DIMENSION MATRIX (9,8)
      READ (5,1000) MATRIX
      WRITE (6,1001) MATRIX
      CALL SADDLE (MATRIX, I, J)
      PRINT, I. J
      STOP
 1000 FORMAT(9(I1,1X))
 1001 FORMAT(9I2)
      END

      SUBROUTINE SADDLF (MAT, I, J)
      DIMENSION MAT(9,8)
      DIMENSION KEEPR(9), KEEPC(8)
C                                        COMPUTE MIN - ROW
      DO 160 IR=1,9,1
      MIN=MAT(IR,1)
      DO 150 IC=2,8,1
      IF (MIN.GT.MAT(IR,IC)) MIN=MAT(IR,IC)
  150 CONTINUE
  160 KEEPR(IR)=MIN
C                                        COMPUTE MAX-COL
      DO 260 IC=1,8,1
      MAX=MAT(1,IC)
      DO 250 IR=2,9,1
      IF (MAX.LT.MAT(IR,IC)) MAX=MAT(IR,IC)
  250 CONTINUE
  260 KEEPC(IC)=MAX
C                                        LOOK FOR MATCH
      DO 370 I=1,9,1
      DO 370 J=1,8,1
      IF (KEEPR(I).EQ.KEEPC(J)) RETURN
  370 CONTINUE
      I=0
      J=0
      RETURN
      END
*EOR
1 4 2 2 4 3 0 1 1
2 4 6 1 5 5 2 2 2
2 5 6 9 6 4 0 3 2
3 5 6 7 7 3 3 4 3
3 5 6 5 8 2 0 5 5
3 5 1 1 9 1 4 6 5
4 5 9 2 8 0 0 7 5
4 6 5 3 7 1 5 8 6
```

TEST PROGRAM 7

```
C
C
      INTEGER SUMDIV
C
C     SEARCH FOR SOME FRIENDLY NUMBERS
C
      DO 100 N=1000,1500,1
      M=SUMDIV(N)
      IF (SUMDIV(M).NE.N) GO TO 100
      WRITE (6,1000) N, M
  100 CONTINUE
      STOP
 1000 FORMAT(I5,≠ AND≠,I5,≠ ARE FRIENDLY.≠)
      END

      INTEGER FUNCTION SUMDIV(N)
C
C     CALCULATE SUM OF DIVISORS OF N
C
      SUMDIV=1
      DO 100 I=2,N-1,1
      IF (N/I*I.NE.N) GO TO 100
      SUMDIV=SUMDIV+I
  100 CONTINUE
      RETURN
      END
```

TEST PROGRAM 8

```
C
C
C
      LOGICAL PRIME
      COMMON LIST(100)
C
      LIST(1)=2                              GET 100 PRIMES
      LIST(2)=3
      N=3
      DO 110 I=3,100,1
  100 N=N+1
      IF (.NOT. PRIME(N)) GO TO 100
  110 LIST(I)=N
C
      WRITE (6,1000) LIST                    DEBUG PRINTOUT
C
      N=1                                    FIND COMPOUND NUMBERS
      DO 500 L=1,8
  250 DO 300 I=0,L-1,1
      IF (.NOT. PRIME(N+I)) GO TO 300
      N=N+I+1
      GO TO 250
  300 CONTINUE
      WRITE (6,2000) L, N
  500 CONTINUE
      STOP
C
 1000 FORMAT(1X,20I6)
 2000 FORMAT(≠0SEQUENCE OF≠,I2,≠ BEGINS AT≠,I4)
      END

      LOGICAL FUNCTION PRIME (N)
      COMMON LIST(100)
C
      PRIME=.TRUE.
      LIM=SQRT( FLOAT(N) ) + 0.5
      DO 100 I=1,100,1
      IF (LIST(I).GT.LIM) RETURN
      IF (N/LIST(I)*LIST(I).NE.N) GO TO 100
      PRIME=.FALSE.
      RETURN
  100 CONTINUE
      STOP 1
      END
```

```
C
C
C                                                    TEST PROGRAM 9
      INTEGER GT
      INTEGER WAIT, TOTAL
      LOGICAL DEBUG
      DO 100 GT=10,90,10
      TOTAL=0
      DEBUG=.TRUE.
      DO 110 I=1,3,1
      CALL SIMUL (GT, WAIT, DEBUG)
      TOTAL=TOTAL+WAIT
      WRITE (6,1000) GT, WAIT
      DEBUG=.FALSE.
  110 CONTINUE
      AVG=FLOAT(TOTAL)/3.0
      WRITE (6,2000) AVG
  100 CONTINUE
      STOP
 1000 FORMAT(I3,≠ SECOND GREEN LIGHT.  WAIT =≠,I6)
 2000 FORMAT(≠ AVERAGE WAIT =≠,F7.1,/)
      END

      SUBROUTINE SIMUL (GT, WAIT, DEBUG)
      INTEGER TIME, Q1, Q2, WAIT1, WAIT2, GT, RANDOM
      LOGICAL DEBUG
      INTEGER WAIT
      INTEGER ON, OFF
C
      Q1=0
      Q2=0
      WAIT1=0
      WAIT2=0
      TIME=0
      LIGHT=1
      ON=0
      OFF=0
      IF (DEBUG) WRITE (6,3000) GT
C                                     ADD TO QUEUES
  100 Q1=Q1+RANDOM(5,15)
      Q2=Q2+RANDOM(6,24)
      IF (DEBUG) WRITE (6,2000) TIME, LIGHT, Q1, WAIT1, Q2, WAIT2
C                                     REMOVE FROM QUEUES IF GREEN
      IF (LIGHT.EQ.1) GO TO 200
      Q2=Q2-MIN0(Q2,20)
      GO TO 250
  200 Q1=Q1-MIN0(Q1,36)
C                                     ACCUMULATE WAITING TIME
  250 CONTINUE
      WAIT1=WAIT1+Q1*10
      WAIT2=WAIT2+Q2*10
      IF (DEBUG) WRITE (6,2000) TIME, LIGHT, Q1, WAIT1, Q2, WAIT2
C                                     CHANGE LIGHT
      TIME=TIME+10
      IF (LIGHT.EQ.0) GO TO 300
      ON=ON+10
      IF (ON.EQ.GT) LIGHT=0
      GO TO 400
  300 OFF=OFF+10
      IF (OFF+ON.NE.100) GO TO 400
```

```
      LIGHT=1
      ON=0
      OFF=0
C
  400 IF (TIME.LT.300) GO TO 100                5 MINUTS UP YET...
      IF (DEBUG) WRITE (6,1000) Q1, WAIT1, Q2, WAIT2
      WAIT=WAIT1+WAIT2
      RETURN
C
 1000 FORMAT(≠0CARS LEFT IN Q1 =≠,I3,≠  WAITING TIME =≠,I5,/,
     +       ≠ CARS LEFT IN Q2 =≠,I3,≠  WAITING TIME =≠,I5)
 2000 FORMAT(6I10)
 3000 FORMAT(≠1SIMULATION OF≠,I3,≠ SECOND GREEN LIGHT≠,//
     +       ≠    TIME       LIGHT        Q1         WAIT1        Q2         WAIT2≠
     +       /)
      END

      INTEGER FUNCTION RANDOM (I, J)
      X=J-I
      RANDOM=X*RANF(0.0)
      RANDOM=RANDOM+I
      RETURN
      END
```

```
C
C                                                 TEST PROGRAM 10
C
      DO 100 I=1,2,1
      WRITE (6,1000)
      CALL SIMUL (250)
  100 CONTINUE
      STOP
 1000 FORMAT(≠1BEGIN SIMULATION≠)
      END

      SUBROUTINE SIMUL (LIMIT)
C
C     PERFORM SIMULATION WITH LIMIT PEOPLE
C
      INTEGER TIME, RANDOM, CO, WAIT, TOTAL
      INTEGER Q(3,600), QL(3)
      COMMON Q
C                                   INITIALIZE
      DO 100 I=1,3
  100 QL(I)=0
      KOUNT=0
      TIME=0
      TOTAL=0
      NA=0
C                                   CREATE NEW ARIVAL
  175 IF (TIME.NE.NA) GO TO 200
      KOUNT=KOUNT+1
C                                   FIND SHORTEST LINE
      MIN=1
      IF (QL(MIN).GT.QL(2)) MIN=2
      IF (QL(MIN).GT.QL(3)) MIN=3
      CO=RANDOM(100,360)
      QL(MIN)=QL(MIN)+1
      Q(MIN,QL(MIN))=KOUNT*100000000 + CO
      NA=TIME+RANDOM(0,160)
      IF (KOUNT.EQ.LIMIT) NA=-1
      WRITE (6,1000) TIME, KOUNT, MIN, CO, QL(MIN), NA
      GO TO 175
C                                   SERVICE QUEUES
  200 TIME=TIME+1
      DO 250 I=1,3,1
      IF (QL(I).EQ.0) GO TO 250
C                                   ADD WAIT TIME
      DO 210 J=1,QL(I),1
  210 Q(I,J)=Q(I,J)+1000
C                                   PROCESS CHECKOUT TIME
      MAN=Q(I,1)/100000000
      WAIT=(Q(I,1)-MAN*100000000)/1000
      CO=MOD(Q(I,1),1000) - 1
      Q(I,1)=MAN*100000000 + WAIT*1000 + CO
      IF (CO.GT.0) GO TO 250
C                                   REMOVE FROM Q
      QL(I)=QL(I)-1
      DO 220 J=1,QL(I),1
  220 Q(I,J)=Q(I,J+1)
      WRITE (6,3000) TIME, MAN, I, WAIT, QL(I)
      CALL MAXWAIT (0, WAIT, MAN)
  250 CALL LINELEN (0, I, QL(I))
C
```

```
      IF (NA.GE.0) GO TO 175
      IF (QL(1)+QL(2)+QL(3).GT.0) GO TO 175
      CALL LINELEN (1, I, J)
      CALL MAXWAIT (1, I, J)
      RETURN
 1000 FORMAT(≠ TIME=≠,I5,≠  *ADD*  MAN=≠,I3,≠  Q=≠,I1,≠  CHECKOUT=≠,I3,
     +       ≠  Q LENGTH=≠,I3,≠  NEXT ARIVAL=≠,I5)
 3000 FORMAT(≠ TIME=≠,I5,≠  *REMOVE*  MAN=≠,I3,≠  Q=≠,I1,≠  WAIT=≠,I5,
     +       ≠  Q LENGTH=≠,I3)
      END

      SUBROUTINE MAXWAIT (MODE, L, N)
C
C     GATHERS STATS ON WAITING LINES
C
      DIMENSION MAX(10), NUM(10)
      INTEGER AVG, TOTAL
      DATA MAX, NUM, TOTAL, KNT /22*0/
C
      IF (MODE.NE.0) GO TO 200
      TOTAL=TOTAL+N
      KNT=KNT+1
      DO 100 I=1,10,1
      IF (MAX(I).GT.L) GO TO 100
      MAX(I)=L
      NUM(I)=N
      RETURN
  100 CONTINUE
      RETURN
C
  200 WRITE (6,1000)
      DO 300 I=1,10,1
      WRITE (6,2000) MAX(I), NUM(I)
      MAX(I)=0
  300 NUM(I)=0
      AVG=FLOAT(TOTAL)/FLOAT(KNT)+0.5
      WRITE (6,3000) AVG
      TOTAL=0
      KNT=0
      RETURN
 1000 FORMAT(≠0LONGEST WAITS WERE:≠)
 2000 FORMAT(≠ WAIT=≠,I5,≠  MAN=≠,I3)
 3000 FORMAT(≠ AVERAGE WAIT=≠,I5)
      END

      SUBROUTINE LINELEN (MODE, Q, L)
C
C     GATHERS LINE LENGTH STATS
C
      INTEGER Q, AVG
      INTEGER LEN(3), MAX(3), KNT(3)
      DATA LEN, MAX, KNT /9*0/
C
      IF (MODE.NE.0) GO TO 100
      LEN(Q)=LEN(Q)+L
      KNT(Q)=KNT(Q)+1
      IF (MAX(Q).LT.L) MAX(Q)=L
      RETURN
C
  100 WRITE (6,2000)
      DO 200 I=1,3,1
```

```
      AVG=FLOAT(LEN(I))/FLOAT(KNT(I)) + 0.5
      WRITE (6,1000) I, MAX(I), AVG
      LEN(I)=0
      MAX(I)=0
  200 KNT(I)=0
      RETURN
 1000 FORMAT(≠ Q=≠,I1,≠  MAX=≠,I3,≠  AVERAGE=≠,I3)
 2000 FORMAT(≠0LINE LENGTH STATISTICS!≠)
      END
```

```
C
C
C                                                      TEST PROGRAM 11
      INTEGER SUMDIG3
C
C     THIS PROGRAM HUNTS FOR SOME NUMBERS SUCH THAT THE SUM OF
C     THE CUBES OF THE DIGITS OF THE NUMBER EQUALS THE NUMBER.
C
      DO 100 I=2,500,1
      IF (I.NE.SUMDIG3(I)) GO TO 100
      WRITE (6,1000) I, I
  100 CONTINUE
      STOP
C
 1000 FORMAT(≠0THE SUM OF THE CUBES OF THE DIGITS OF≠,I4,≠ EQUALS≠,I4)
      END

      INTEGER FUNCTION SUMDIG3 (N)
C
C     COMPUTE THE SUM OF THE CUBES OF THE DIGITS OF N
C
      INTEGER WORKER
      SUMDIG3=0
      WORKER=IABS(N)
  100 IF (WORKER.EQ.0) RETURN
      SUMDIG3=SUMDIG3 + MOD(WORKER,10) ** 3
      WORKER=WORKER/10
      GO TO 100
      END
```

# APPENDIX B

## SOFTWARE PARAMETER DATA FOR PROGRAMS G1 THROUGH G11.

G1.

| i | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | EOL | 18 |
| 2. | = | 15 |
| 3. | IF( ) | 5 |
| 4. | , | 4 |
| 5. | + | 3 |
| 6. | .EQ. | 2 |
| 7. | .LT. | 2 |
| 8. | ( ) | 2 |
| 9. | - | 2 |
| 10. | / | 1 |
| 11. | Go To 100 | 1 |
| 12. | Go To 200 | 1 |
| 13. | Go To 900 | 1 |
| 14. | DO | 1 |
| 15. | .GT. | 1 |

$\eta_1 = 15, \quad N_1 = 59$

| i | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | 1 | 12 |
| 2. | JR | 8 |
| 3. | JC | 6 |
| 4. | IR | 6 |
| 5. | N | 5 |
| 6. | 23 | 5 |
| 7. | IC | 4 |
| 8. | MAGIC | 2 |
| 9. | 2 | 1 |
| 10. | 529 | 1 |
| 11. | 0 | 1 |

$\eta_2 = 11, \quad N_2 = 51$

G2.

| $i$ | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | , | 45 |
| 2. | EOL | 36 |
| 3. | = | 29 |
| 4. | ( ) | 29 |
| 5. | + | 23 |
| 6. | - | 15 |
| 7. | .EQ. | 11 |
| 8. | DO | 10 |
| 9. | IF( ) | 10 |
| 10. | .NE. | 5 |
| 11. | .OR. | 3 |
| 12. | * | 3 |
| 13. | .AND. | 3 |
| 14. | Go To 370 | 3 |
| 15. | .NOT. | 1 |
| 16. | / | 1 |
| 17. | Go To 200 | 1 |
| 18. | Go To 210 | 1 |
| 19. | Go To 325 | 1 |
| 20. | Go To 350 | 1 |

$\eta_1 = 20$, $N_1 = 231$

| $i$ | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | I | 53 |
| 2. | IR | 17 |
| 3. | IC | 17 |
| 4. | MAT | 17 |
| 5. | JR | 11 |
| 6. | JC | 11 |
| 7. | 0 | 9 |
| 8. | J | 8 |
| 9. | MODE | 8 |
| 10. | N | 5 |
| 11. | 5 | 5 |
| 12. | 1 | 5 |
| 13. | OUT | 5 |
| 14. | 2 | 4 |
| 15. | 23 | 4 |
| 16. | 3 | 4 |
| 17. | NT | 3 |
| 18. | 25 | 2 |
| 19. | 10 | 2 |
| 20. | NU | 2 |
| 21. | DIGIT | 2 |
| 22. | 1H+ | 1 |
| 23. | EDGE | 1 |
| 24. | WHITE | 1 |

$\eta_2 = 24$, $N_2 = 197$

G3.

| i | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | EOL | 18 |
| 2. | = | 11 |
| 3. | , | 8 |
| 4. | + | 5 |
| 5. | IF( ) | 4 |
| 6. | CALL PUTCH( ) | 3 |
| 7. | GETCH( ) | 2 |
| 8. | .EQ. | 2 |
| 9. | Go To 200 | 2 |
| 10. | .NE. | 2 |
| 11. | ( ) | 2 |
| 12. | Go To 300 | 1 |
| 13. | .AND. | 1 |
| 14. | .NOT. | 1 |
| 15. | Go To 400 | 1 |
| 16. | Go To 100 | 1 |

$\eta_1 = 16, \quad N_1 = 64$

| i | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | 1 | 9 |
| 2. | PO | 8 |
| 3. | PI | 7 |
| 4. | CO | 7 |
| 5. | KOUNT | 5 |
| 6. | C | 4 |
| 7. | OUTPUT | 3 |
| 8. | INPUT | 2 |
| 9. | 1Hb | 1 |
| 10. | NUMERIC | 1 |
| 11. | DIGIT | 1 |
| 12. | 1H+ | 1 |

$\eta_2 = 12, \quad N_2 = 49$

G4.

| i | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | EOL | 32 |
| 2. | , | 27 |
| 3. | = | 26 |
| 4. | ( ) | 15 |
| 5. | DO | 7 |
| 6. | + | 6 |
| 7. | - | 3 |
| 8. | * | 2 |
| 9. | IF( ) | 2 |
| 10. | .EQ. | 2 |
| 11. | RANF( ) | 1 |
| 12. | CALL SHUFFLE( ) | 1 |
| 13. | CALL PUSH( ) | 1 |
| 14. | POP( ) | 1 |
| 15. | Go To 400 | 1 |
| 16. | Go To 300 | 1 |
| 17. | FLOAT( ) | 1 |
| 18. | / | 1 |
| 19. | Go To 100 | 1 |

$n_1 = 19$, $N_1 = 131$

| i | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | 1 | 33 |
| 2. | PICK | 12 |
| 3. | PILE | 10 |
| 4. | K | 9 |
| 5. | I | 7 |
| 6. | L | 6 |
| 7. | 13.0 | 4 |
| 8. | J | 4 |
| 9. | DECK | 3 |
| 10. | 100 | 3 |
| 11. | CARD | 3 |
| 12. | D | 3 |
| 13. | KOUNT | 3 |
| 14. | PC | 2 |
| 15. | 4 | 2 |
| 16. | 500 | 2 |
| 17. | 52 | 2 |
| 18. | POP | 2 |
| 19. | N | 1 |
| 20. | 53 | 1 |
| 21. | ITEM | 1 |

$n_2 = 21$, $N_2 = 113$

G5.

| $i$ | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | EOL | 11 |
| 2. | = | 11 |
| 3. | ( ) | 7 |
| 4. | DO | 3 |
| 5. | , | 3 |
| 6. | - | 2 |
| 7. | + | 1 |

$n_1 = 7, \quad N_1 = 38$

| $i$ | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | I | 7 |
| 2. | L2 | 6 |
| 3. | N | 5 |
| 4. | MAN | 5 |
| 5. | I | 3 |
| 6. | L1 | 3 |
| 7. | K | 2 |
| 8. | KILL | 2 |
| 9. | L | 1 |
| 10. | M | 1 |

$n_2 = 10, \quad N_2 = 35$

G6.

| i | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | , | 18 |
| 2. | EOL | 15 |
| 3. | = | 14 |
| 4. | ( ) | 10 |
| 5. | DO | 6 |
| 6. | IF( ) | 3 |
| 7. | .GT. | 1 |
| 8. | .LT. | 1 |
| 9. | .EQ. | 1 |

$\eta_1 = 9$, $N_1 = 69$

| i | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | 1 | 12 |
| 2. | IR | 8 |
| 3. | IC | 8 |
| 4. | MAT | 6 |
| 5. | MIN | 4 |
| 6. | MAX | 4 |
| 7. | 9 | 3 |
| 8. | 8 | 3 |
| 9. | I | 3 |
| 10. | J | 3 |
| 11. | 2 | 2 |
| 12. | KEEPR | 2 |
| 13. | KEEPC | 2 |
| 14. | 0 | 2 |

$\eta_2 = 14$, $N_2 = 62$

G7.

| $i$ | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | EOL | 7 |
| 2. | = | 5 |
| 3. | , | 4 |
| 4. | DO | 2 |
| 5. | SUMDIV( ) | 2 |
| 6. | IF( ) | 2 |
| 7. | .NE. | 2 |
| 8. | Go To 100 | 2 |
| 9. | - | 1 |
| 10. | / | 1 |
| 11. | * | 1 |
| 12. | + | 1 |

$\eta_1 = 12$, $N_1 = 30$

| $i$ | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | N | 6 |
| 2. | 1 | 4 |
| 3. | I | 4 |
| 4. | SUMDIV | 3 |
| 5. | 1000 | 2 |
| 6. | M | 2 |
| 7. | 1500 | 1 |
| 8. | 2 | 1 |

$\eta_2 = 8$, $N_2 = 23$

G8.

| $i$ | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | EOL | 18 |
| 2. | = | 14 |
| 3. | , | 7 |
| 4. | ( ) | 6 |
| 5. | + | 5 |
| 6. | DO | 4 |
| 7. | IF( ) | 4 |
| 8. | .NOT. | 2 |
| 9. | PRIME | 2 |
| 10. | Go To 100 | 2 |
| 11. | - | 1 |
| 12. | Go To 300 | 1 |
| 13. | Go To 250 | 1 |
| 14. | SQRT( ) | 1 |
| 15. | FLOAT( ) | 1 |
| 16. | .GT. | 1 |
| 17. | / | 1 |
| 18. | * | 1 |
| 19. | .NE. | 1 |

$\eta_1 = 19, \quad N_1 = 73$

| $i$ | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | N | 12 |
| 2. | 1 | 10 |
| 3. | I | 9 |
| 4. | LIST | 6 |
| 5. | 3 | 3 |
| 6. | 2 | 2 |
| 7. | 100 | 2 |
| 8. | L | 2 |
| 9. | PRIME | 2 |
| 10. | LIM | 2 |
| 11. | 8 | 1 |
| 12. | 0 | 1 |
| 13. | .TRUE. | 1 |
| 14. | 0.5 | 1 |
| 15. | .FALSE. | 1 |

$\eta_2 = 15, \quad N_2 = 55$

G9.

| i | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | EOL | 38 |
| 2. | = | 32 |
| 3. | , | 11 |
| 4. | + | 11 |
| 5. | IF( ) | 5 |
| 6. | .EQ. | 3 |
| 7. | - | 3 |
| 8. | * | 3 |
| 9. | DO | 2 |
| 10. | RANDOM( ) | 2 |
| 11. | MINO( ) | 2 |
| 12. | Go To 400 | 2 |
| 13. | CALL SIMULA( ) | 1 |
| 14. | FLOAT( ) | 1 |
| 15. | / | 1 |
| 16. | Go To 200 | 1 |
| 17. | Go To 250 | 1 |
| 18. | Go To 300 | 1 |
| 19. | .NE. | 1 |
| 20. | .LT. | 1 |
| 21. | Go To 100 | 1 |
| 22. | RANF( ) | 1 |

$\eta_1 = 22$, $N_1 = 124$

| i | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | 0 | 14 |
| 2. | 10 | 7 |
| 3. | Q1 | 7 |
| 4. | Q2 | 7 |
| 5. | 1 | 6 |
| 6. | ON | 6 |
| 7. | LIGHT | 5 |
| 8. | OFF | 5 |
| 9. | TOTAL | 4 |
| 10. | WAIT 1 | 4 |
| 11. | WAIT 2 | 4 |
| 12. | TIME | 4 |
| 13. | GT | 3 |
| 14. | DEBUG | 3 |
| 15. | WAIT | 3 |
| 16. | RANDOM | 3 |
| 17. | I | 2 |
| 18. | 3 | 2 |
| 19. | X | 2 |
| 20. | 90 | 1 |
| 21. | .TRUE. | 1 |
| 22. | .FALSE. | 1 |
| 23. | AVG | 1 |
| 24. | 5 | 1 |
| 25. | 15 | 1 |
| 26. | 6 | 1 |
| 27. | 24 | 1 |
| 28. | 20 | 1 |
| 29. | 36 | 1 |
| 30. | 100 | 1 |
| 31. | 300 | 1 |
| 32. | J | 1 |

$\eta_1 = 32$, $N_2 = 104$

G10.



| i | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | EOL | 59 |
| 2. | = | 45 |
| 3. | ( ) | 43 |
| 4. | , | 33 |
| 5. | ÷ | 17 |
| 6. | IF( ) | 12 |
| 7. | DO | 7 |
| 8. | .GT. | 5 |
| 9. | * | 4 |
| 10. | - | 4 |
| 11. | / | 4 |
| 12. | FLOAT( ) | 4 |
| 13. | .NE. | 3 |
| 14. | Go To 175 | 3 |
| 15. | Go To 200 | 2 |
| 16. | RANDOM( ) | 2 |
| 17. | .EQ. | 2 |
| 18. | Go To 250 | 2 |
| 19. | CALL MAXWAIT( ) | 2 |
| 20. | CALL LINELEN( ) | 2 |
| 21. | Go To 100 | 2 |
| 22. | CALL SIMULA( ) | 1 |
| 23. | MOD( ) | 1 |
| 24. | .GE. | 1 |
| 25. | .LT. | 1 |

$\eta_1 = 25$, $N_1 =$ 261

| i | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | I | 33 |
| 2. | 1 | 30 |
| 3. | 0 | 21 |
| 4. | QL | 17 |
| 5. | Q | 15 |
| 6. | MIN | 9 |
| 7. | KNT | 8 |
| 8. | J | 8 |
| 9. | 3 | 6 |
| 10. | MAX | 6 |
| 11. | NA | 5 |
| 12. | KOUNT | 5 |
| 13. | TIME | 5 |
| 14. | TOTAL | 5 |
| 15. | CO | 5 |
| 16. | L | 5 |
| 17. | 2 | 4 |
| 18. | 1000 | 4 |
| 19. | MAN | 4 |
| 20. | LEN | 4 |
| 21. | WAIT | 3 |
| 22. | $10^8$ | 2 |
| 23. | $10^7$ | 2 |
| 24. | MODE | 2 |
| 25. | N | 2 |
| 26. | NUM | 2 |
| 27. | AVG | 2 |
| 28. | 0.5 | 2 |
| 29. | 250 | 1 |
| 30. | 100 | 1 |
| 31. | 360 | 1 |
| 32. | 160 | 1 |
| 33. | LIMIT | 1 |
| 34. | 10 | 1 |

$\eta_2 = 34$, $N_2 =$ 222

G11.

| $i$ | OPERATOR | $f_{1,i}$ |
|---|---|---|
| 1. | EOL | 8 |
| 2. | = | 5 |
| 3. | , | 3 |
| 4. | IF( ) | 2 |
| 5. | Go To 100 | 2 |
| 6. | DO | 1 |
| 7. | .NE. | 1 |
| 8. | ( ) | 1 |
| 9. | IABS( ) | 1 |
| 10. | .EQ. | 1 |
| 11. | + | 1 |
| 12. | MOD( ) | 1 |
| 13. | ** | 1 |
| 14. | / | 1 |

$\eta_1 = 14$, $N_1 = 29$

| $i$ | OPERAND | $f_{2,i}$ |
|---|---|---|
| 1. | WORKER | 5 |
| 2. | SUMDIG3 | 4 |
| 3. | I | 3 |
| 4. | 0 | 2 |
| 5. | 10 | 2 |
| 6. | 2 | 1 |
| 7. | 500 | 1 |
| 8. | 1 | 1 |
| 9. | N | 1 |
| 10. | 3 | 1 |

$\eta_2 = 10$, $N_2 = 21$

# An experiment comparing Fortran programming times with the software physics hypothesis

*by* R. D. GORDON and M. H. HALSTEAD
*Purdue University*
Lafayette, Indiana

ABSTRACT

Recent discoveries in the area of Algorithm Structure or Software Physics[1-25] have produced a number of hypotheses. One of these relates the number of elementary mental discriminations required to implement an algorithm to measurable properties of that algorithm, and the results of one set of experiments confirming this relationship have been published.[16] That publication, while significant, made no claim to finality, suggesting instead that further experiments were warranted. This paper will present the results of a second set of experiments, having the advantages of being conducted in a single implementation language, Fortran, from problem specifications readily available in computer textbooks.

The first section of this paper presents the timing hypothesis, and the elementary equations upon which it rests. The second section presents the details of the experiment and the results which were obtained, and the third section contains an analysis of the data.

## TIMING HYPOTHESIS

Measurable properties of any implementation of any algorithm include:

$\eta_1$ = The count of distinct operators
$\eta_2$ = The count of distinct operands (variables or constants)
$N_1$ = Total uses of operators
$N_2$ = Total uses of operands

The vocabulary, $\eta$, is given by:

$$\eta = \eta_1 + \eta_2 \qquad (1)$$

and the length, N, is:

$$N = N_1 + N_2 \qquad (2)$$

From these properties, it is possible to obtain the volume, V, in bits, as:

$$V = N \log_2 \eta \qquad (3)$$

and the implementation level, L, where $L \leq 1$, as:

$$L = \frac{\eta_1^*}{\eta_1} \frac{\eta_2}{N_2} \qquad (4)$$

where $\eta_1^*$, the minimum possible number of operators, will equal 2 for most algorithms. (One for the name of a function, plus one for a grouping symbol operator). It has been shown[4] that the product $L \times V$ is invariant under translation from one language to another, and that for programs without impurities:[3,4,5]

$$N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \qquad (5)$$

From this point, the following nine steps yield the timing equation:

1. A program consists of N selections from $\eta$ elements.
2. A binary search of $\eta$ elements requires $\log_2 \eta$ comparisons.
3. A program is generated by making $N \log_2 \eta$ comparisons.
4. Therefore, the volume, V, is a count of the number of comparisons required.
5. The number of elementary mental discriminations required to complete one comparison measures the difficulty of the task.
6. The level, L, is the reciprocal of the difficulty.
7. Therefore, E, the count of elementary mental discriminations required to generate a program, is given by:

$$E = \frac{V}{L} \qquad (6)$$

8. S, the speed with which the brain makes elementary mental discriminations can be obtained from psychology[16] as:

$5 \leq S \leq 20$ discriminations per second.

9. Therefore, the time to generate a *preconceived* program, by a *concentrating* programmer, *fluent* in a language, is:

$$\hat{T} = \frac{V}{SL} \qquad (7)$$

Equation 7 may be expressed in more basic terms by substituting for V from equation 3, and for L from equation 4, with $\eta_1^* = 2$, giving:

$$\hat{T} = \frac{\eta_1 N_2 N \log_2 \eta}{2 S \eta_2} \qquad (8)$$

The effect of possible impurities[5] may be eliminated from equation 8 by substituting for N from equation 5. Letting $S=60\times 18=1080$ will then give, for time in minutes:

$$\hat{T}=\frac{\eta_1 N_2(\eta_1\log_2\eta_1+\eta_2\log_2\eta_2)\log_2\eta}{2160\,\eta_2} \qquad (9)$$

Each of the variables on the right hand side of equation 9 can be readily measured (or counted) in any computer program, and the experiment described in the next section was designed to compare results from that equation with observed programming times.

## EXPERIMENTAL PROCEDURE

Eleven problems were arbitrarily selected from two published sources. In selecting candidates for the experiment, problems were sought which were stated in a non-procedural form. Further, the problem statement had to be complete. That is, in the course of solving a particular problem, specific laws of physics, mathematics, etc. would not have to be derived. The problems finally selected were taken from Knuth,[27] and from Maurer and Williams,[28] and cover a wide range of topics including character manipulation, list processing, simulation experiments and mathematical analysis. The source of each problem statement is cited in Table I.

On each of eleven days, one of these problems was implemented by the senior author. In order to maintain a consistent level of performance all work was conducted in a quiet room, free from distractions, during the same period of the day. The time required to fully implement the problem was obtained. This total time included the number of minutes spent reading the statement of the problem, preparing flowcharts and writing preliminary versions of the code, writing the final version of the code, desk checking, and the time spent working to correct errors in the program. Time to keypunch was not included.

TABLE I—Experimental Data

| | Program Specifications | | | Software Parameters | | | | Implementation |
|---|---|---|---|---|---|---|---|---|
| No. | Ref.* | Page | Problem | $\eta_1$ | $\eta_2$ | $N_1$ | $N_2$ | Time-Minutes |
| G1 | K | 158 | 21 | 15 | 11 | 59 | 61 | 19 |
| G2 | K | 159 | 23 | 20 | 24 | 231 | 197 | 92 |
| G3 | K | 196 | 7 | 16 | 12 | 64 | 49 | 16 |
| G4 | K | 377 | 17 | 19 | 21 | 131 | 113 | 39 |
| G5 | K | 158 | 22 | 7 | 10 | 38 | 35 | 21 |
| G6 | K | 154 | 10 | 9 | 14 | 69 | 62 | 30 |
| G7 | M | 32 | 3.2.21 | 12 | 8 | 30 | 23 | 5 |
| G8 | M | 32 | 3.2.23 | 19 | 15 | 73 | 55 | 24 |
| G9 | M | 88 | 8.3.2 | 22 | 32 | 124 | 104 | 43 |
| G10 | M | 89 | 8.3.4 | 25 | 34 | 261 | 222 | 91 |
| G11 | M | 27 | 3.2.4 | 14 | 10 | 29 | 21 | 5 |

* K = Knuth[27], M = Maurer and Williams.[28]

For a number of reasons, including availability and fluency, all of the algorithms were implemented in Fortran. In the course of solving a problem the correctness of the implementation was checked by executing a sufficiently complex test case for which a correct answer was known. In some cases the solution to a problem was written as a subroutine and testing required that a main routine be written. In such a case only the preparation of the subroutine was considered for the experiment. In addition, several implementations made use of subroutines previously written. Such routines were also not included.*

After each program was completed, a careful count was made to determine values of $\eta_1$, $\eta_2$, $N_1$ and $N_2$. In obtaining these values all read, write, declarative statements and comments were ignored. The results are shown in Table I.

## ANALYSIS OF THE DATA

The programming time predicted by theory was obtained for each program by applying equation 9 to the data in Table I. This result, $\hat{T}$, can be compared with the observed value, T, in Table II. In addition, a count of the number of statements in each program was obtained, and the programs were ordered according to these values.

The average of the calculated values, 34 minutes, is fortuitously close to the observed value, 35 minutes. The coefficient of correlation is 0.934, only slightly smaller than the value of 0.952 reported in an earlier experiment.[16] In further agreement with that experiment, the correlation between length and observed times, 0.887, is lower than between observed and calculated times.

In conclusion, it may again be observed that one more set of experimental data does not contradict the simple hypothesis. As a result, further carefully controlled experiments by others would appear to be warranted.

* Additional details available from the author.

TABLE II—Experimental Results

| Program Number | Statement Count | Programming Time-Minutes | |
|---|---|---|---|
| | | T observed | T Equ. 9 |
| G7 | 7 | 5 | 4.6 |
| G11 | 8 | 5 | 5.4 |
| G5 | 11 | 21 | 2.5 |
| G6 | 15 | 30 | 6.8 |
| G3 | 18 | 16 | 15.6 |
| G1 | 18 | 19 | 14.6 |
| G8 | 18 | 24 | 22.9 |
| G4 | 32 | 39 | 43.6 |
| G2 | 36 | 92 | 81.5 |
| G9 | 38 | 43 | 49.2 |
| G10 | 59 | 91 | 128.5 |
| Means | | 35.0 | 34.1 |

## REFERENCES

1. Bayer, Rudolf, *A Theoretical Study of Halstead's Software Phenomenon*, CSD Tech. Rept. No. 69, Purdue, May 1972.
2. Bayer, Rudolf, *On Program Volume and Program Modularization*, CSD Tech. Rept. No. 105, Purdue, September 1973.
3. Bohrer, Robert, "Halstead's Criteria and Statistical Algorithms," *Proc. 8th Computer Science/ Statistics Interface Symposium*, Los Angeles, February 1975.
4. Bulut, Necdet, *Invariant Properties of Algorithms*, Ph.D. Thesis, Purdue, August 1973.
5. Bulut, Necdet and M. H. Halstead, "Impurities Found in Algorithm Implementations," *ACM SIGPLAN Notices*, 9, 3, March 1974.
6. Bulut, Necdet, M. H. Halstead and Rudolf Bayer, "The Experimental Verification of a Structural Property of Fortran Programs," *Proc. ACM Annual Conference*, San Diego, 1974.
7. Funami, Yasao and M. H. Halstead, *Software Physics Analysis of Akayama's Debug Data*, CSD Tech. Rept. No. 144, Purdue, May 1975.
8. Halstead, M. H., "Natural Laws Controlling Algorithmic Structure," *ACM SIGPLAN Notices*, 7, 2, February 1972.
9. Halstead, M. H., *A Theoretical Relationship Between Mental Work and Machine Language Programming*, CSD Tech. Rept. No. 67, Purdue, May 1972.
10. Halstead, M. H. and Rudolf Bayer, "Algorithm Dynamics," *Proc. ACM Annual Conference*, Atlanta, 1973.
11. Halstead, M. H., "An Experimental Determination of the 'Purity' of a Trivial Algorithm," *ACM SIGME Performance Evaluation Review*, 2, 1, March 1973.
12. Halstead, M. H., "Language Level, A Missing Concept in Information Theory," *ACM SIGME Performance Evaluation Review*, 2, 1, March 1973.
13. Halstead, M. H. and P. M. Zislis, *Experimental Verification of Two Theorems of Software Physics*, CSD Tech. Rept. No. 97, Purdue, June 1973.
14. Halstead, M. H., *Software Physics Comparison of a Sample Program in DSL ALPHA and COBOL*, IBM Research Report No. RJ 1460, October 1974.
15. Halstead, M. H., *Software Physics: Basic Principles*, IBM Research Report No. RJ 1582, May 1975.
16. Halstead, M. H., "Toward a Theoretical Basis for Estimating Programming Efforts," *Proc. ACM Annual Conference*, Minneapolis, 1975.
17. Kennedy, Dale and Roger Bruning, *Childrens Descriptions of Complex Objects*, Report 505-68-6939, Univ. of Nebraska, Lincoln, October 1974.
18. Kulm, Gerald, "Information Content—An Alternative Measure of Reading Complexity," *American Psychological Association Annual Meeting*, New Orleans, August 1974.
19. Ostapko, D. L., "On Deriving a Relation Between Circuits and Input/Output by Analyzing an Equivalent Program," *ACM SIGPLAN Notices*, 8, 6, June 1974.
20. Ostapko, D. L., "Analysis of Algorithms Implemented in Software and Hardware," *Proc. ACM Annual Conference*, San Diego, 1974.
21. Zislis, Paul, *An Experiment in Algorithm Implementation*, CSD Tech. Rept. No. 96, Purdue, June 1973.
22. Zislis, Paul M., "Semantic Decomposition of Computer Programs: An Aid to Program Testing," *ACTA Informatica 4*, pp. 245-269, 1975.
23. Zweben, S. H., *Software Physics: Resolution of an Ambiguity in the Counting Procedure*, CSD Tech. Rept. No. 93, Purdue, April 1973.
24. Zweben, S. H., *The Internal Structure of Algorithms*, Ph.D. Thesis, Purdue, May 1974.
25. Zweben, S. H., "A Recent Approach to the Study of Algorithms," *Proc. ACM Annual Conference*, San Diego, 1974.

Additional References:

26. Stroud, John M., "The Fine Structure of Psychological Time," *Annals of N.Y. Academy of Sciences*, 1966, pp. 623-631.
27. Knuth, Donald, *The Art of Computer Programming*, Addison Wesley Publishing Co., Massachusetts, 1969, Vol. 1.
28. Maurer, H. A. and M. R. Williams, *A Collection of Programming Problems and Techniques*, Prentice-Hall, Inc., New Jersey, 1972.