ALGORITHM 382
COMBINATIONS OF M OUT OF N OBJECTS [G6]

Phillip J. Chase (Recd. 18 Mar. 1969 and 31 Oct. 1969)
Department of Defense, Fort Meade, MD 20755

KEY WORDS AND PHRASES: permutations and combinations, permutations
CR CATEGORIES: 5.39

```
procedure TWIDDLE (x, y, z, done, p);  integer x, y, z;
  Boolean done;  integer array p;
comment  TWIDDLE can be used (1) in generating all combi-
  nations of m out of n objects, or (2) in generating all n-length
  sequences containing m 1's and (n−m) 0's.
    In the case (1), suppose the n objects are given by an array
  a[1:n], and let us successively store combinations in another
  array, say, c[1:m]. For the first combination, c[1] through c[m]
  are equated, respectively, to a[n−m+1] through a[n].
  TWIDDLE (x, y, z, done, p) is called. If done = true, then all
  combinations have been processed and we therefore stop. If
  not, a new combination is made available by setting c[z] equal
  to a[x]. TWIDDLE is called, and we continue on this loop until
  done = true.
    In the case (2), let the sequences of m 1's and (n − m) 0's be
  stored successively in an integer array, say, b[1:n]. The first
  sequence is obtained by setting b[1] through b[n−m] equal to 0,
  and b[n−m+1] through b[n] equal to 1. TWIDDLE (x, y, z,
  done, p) is called. If done = true, then all required sequences
  have been processed, and we therefore stop. If not, a new se-
  quence is made available by setting b[x] equal to 1, and b[y]
  equal to 0. TWIDDLE is again called, and we continue on this
  loop until done = true.
    m and n are used only in the initialization of the auxiliary
  integer array p[0:n+1], which is done in the main program as
  follows. (It is assumed that 0 ≤ m ≤ n and 1 ≤ n.) p[0] is set
  equal to n + 1, and p[n+1] is set equal to −2. p[1] through
  p[n−m] are set equal to 0. p[n−m+1] through p[n] are set
  equal, respectively, to 1 through m. If m = 0, then set p[1]
  equal to 1. done is set equal to false.
    The algorithm has several features which deserve mention.
  When used in generating combinations: (a) at each stage, only
  one combination number, namely c[z], is changed, (b)
  TWIDDLE is order preserving in the sense that at each stage
  c[1] through c[m] will equal, respectively, some a[i1] through
  a[im] where i1 through im are strictly increasing. When used in
  generating fixed-density 0-1 sequences: (c) at each stage, it is
  only necessary to change two numbers of the sequence, b[x]
  and b[y], and these are changed in a specific manner.
    The algorithm underlying this procedure was discovered by
  Leo W. Lathroum in 1965. Another algorithm which accom-
  plishes combinations by transpositions was discovered by
  Donald E. Knuth in 1964. The author has knowledge of the work
  of Lathroum and Knuth from private communications. He will
  include further detail in a mathematical paper, which will in-
  clude justification of this procedure, to be published elsewhere;
begin integer i, j, k;  j := 0;
L1:
  j := j + 1;  if p[j] ≤ 0 then go to L1;
  if p[j−1] = 0 then
  begin
    for i := j − 1 step −1 until 2 do p[i] := −1;  p[j] := 0;
    p[1] := x := z := 1;  y := j;  go to L4
  end;
  if j > 1 then p[j−1] := 0;
L2:
  j := j + 1;  if p[j] > 0 then go to L2;
  i := k := j − 1;
L3:
  i := i + 1;  if p[i] = 0 then
  begin p[i] := −1;  go to L3 end;
  if p[i] = −1 then
  begin
    p[i] := z := p[k];  x := i;  y := k;
    p[k] := −1;  go to L4
  end;
  if i = p[0] then begin done := true;  go to L4 end;
  z := p[j] := p[i];  p[i] := 0;  x := j;  y := i;
L4:
end of TWIDDLE
```

ALGORITHM 383
PERMUTATIONS OF A SET WITH REPETITIONS [G6]

Phillip J. Chase (Recd. 4 Aug. 1969 and 13 Feb. 1970)
Department of Defense, Fort Meade, MD 20755

KEY WORDS AND PHRASES: permutations and combinations, permutations
CR CATEGORIES: 5.39

```
procedure EXTENDED TWIDDLE (x, y, k, u, done, p);
  value k, u;  integer x, y, k, u;  Boolean done;  integer array
  p;
comment  EXTENDED TWIDDLE is a generalization both of
  TWIDDLE [2], which is used in generating combinations by
  transpositions, and of the Trotter-Johnson adjacent-transpo-
  sition permutation algorithms [5, 3].
    In the main program, to successively store all distinct permu-
  tations of C[I] numbers equal to N[I] (I=1 to J) in an array A,
  take, as the first permutation, that obtained by dividing
  A[1:C[1]+···+C[J]] into J intervals and setting the C[I]
  numbers of interval I equal to N[I] (I=1 to J). (We assume
  that J ≥ 2 and that each C[I] ≥ 1. For distinct permutations,
  we need N[I']≠N[I''] whenever I' ≠ I''. For somewhat better
  efficiency, it is desirable, but not necessary, that the sequence
  C[I] be non-increasing.)
    EXTENDED TWIDDLE (x, y, k, u, done, p) is called. If
  done = true, then all permutations have been processed and
  we therefore stop. If not, a new permutation is made available
  by transposing A[x] and A[y], EXTENDED TWIDDLE is
  called, and we continue on this loop until done = true.
    EXTENDED TWIDDLE is initialized in the main program.
  k is equated to J, u is equated to C[1] + ··· + C[J] + 1, done
  is equated to false, and p[0] and p[u] are equated to J + 1.
  p[1:u−1] is initialized by setting the members of the Ith in-
  terval, of length C[I], equal to J − I + 1(I=1 to J);
    That the procedure proceeds by transpositions (not neces-
  sarily adjacent, this being impossible in general) will introduce
  a special economy in some cases. If this feature is of no value
  in a particular application, then the algorithm of Bratley [1]
  or of Sagg [4] might be appropriate. For J = 2, TWIDDLE [2],
  which also has the transposition feature, will be more efficient
  than EXTENDED TWIDDLE. If each C[I] = 1, then Trotter's
  algorithm [5] for generating permutations by transpositions,
  is appropriate.
```

References:

1. Bratley, P. Algorithm 306, Permutations with repetitions. *Comm. ACM 10* (July 1967), 450–451.
2. Chase, P. J. Algorithm 382, Combinations of *M* out of *N* objects. *Comm. ACM 13* (June 1970), 368.
3. Johnson, S. M. Generation of permutations by adjacent transpositions. *Math. Comp. 17* (1963), 282–285.
4. Sagg, T. W. Algorithm 242, Permutations of a set with repetitions. *Comm. ACM 7* (Oct. 1964), 585.
5. Trotter, H. F. Algorithm 115, PERM. *Comm. ACM 5* (Aug. 1962), 434–435.;


```
begin integer s, i, j, b;
  j := b := s := 0;
L1:
  j := j + 1;  if abs (p[j]) = k then
  begin if p[j] < 0 then s := j;  go to L1 end;
  if p[j-1] = k then
  begin
    for i := j - s - 1 step -1 until 2 do p[s + i] := -k;
    if s > b then p[s] := k;
    p[s+1] := p[j];  p[j] := k;  x := s + 1;  y := j;  go to L4
  end;
  if s > b then p[s] := k;
L2:
  j := j + 1;  if abs (p[j]) < k then go to L2;
  if j = u then
  begin
    if k = 2 then begin done := true;  go to L4 end;
    j := b := s;  k := k - 1;  go to L1
  end;
  i := b := j - 1;
L3:
  i := i + 1;  if p[i] = k then
  begin p[i] := -k;  go to L3 end;
  if p[i] = -k then
  begin
    p[i] := p[b];  p[b] := -k;  x := b;  y := i;  go to L4
  end;
  if i = u then
  begin
    if k = 2 then begin done := true;  go to L4 end;
    u := j;  j := b := s;  k := k - 1;  go to L1
  end;
  x := j;  y := i;  p[j] := p[i];  p[i] := k;
L4:
end EXTENDED TWIDDLE
```

The following algorithm by G. W. Stewart relates to the paper by the same author in the Numerical Mathematics department of this issue on pages 365–367. This concurrent publication in Communications follows a policy announced by the Editors of the two departments in the March 1967 issue.

ALGORITHM 384
EIGENVALUES AND EIGENVECTORS OF A REAL SYMMETRIC MATRIX [F2]

G. W. Stewart (Recd. 7 Nov. 1969)

*Department of Computer Sciences, The University of Texas at Austin, *Austin, TX 78712*

* Work on this algorithm was supported in part by the National Science Foundation under grant GP-8442 and by the US Army Research Office (Durham) under grant DA-ARO(D)-31-124-G1050 at the University of Texas at Austin.

KEY WORDS AND PHRASES: real symmetric matrix, eigenvalues, eigenvectors, *QR* algorithm
CR CATEGORIES: 5.14

Description:

SYMQR finds the eigenvalues and, at the users option, the eigenvectors of a real symetric matrix. If the matrix is not initially tridiagonal, it is reduced to tridiagonal form by Householder's method [2, p. 290]. The eigenvalues of the tridiagonal matrix are calculated by a variant of the QR algorithm with origin shifts [1]. Eigenvectors are calculated by accumulating the products of the transformations used in the Householder transformations and the QR steps, a procedure which guarantees a nearly orthonormal set of approximate eigenvectors.

At each QR step the eigenvalues of the 2×2 submatrix in the lower right-hand corner are computed, and the one nearest the last diagonal element is distinguished. When these numbers settle down they are used as origin shifts.

The user may choose between absolute and relative convergence criteria. The former accepts the last diagonal element as an approximate eigenvalue when the last off-diagonal element is a small multiple (EPS) of the infinity norm of the matrix. The latter requires that the last off-diagonal be small compared to the last two diagonal elements. To avoid an excessive number of QR steps, an important consideration when eigenvectors are computed, the following guidelines should be followed. The convergence tolerance should not be smaller than the data warrants [2, p. 102]. The relative convergence criterion should be used only when there are eigenvalues, small compared to the elements of the matrix, that are nonetheless determined to high relative accuracy. Finally, when there is a wide disparity in the sizes of the elements of the matrix, the matrix should be arranged so that the smaller elements appear in the lower right hand corner.

The program will work with matrices whose elements very nearly underflow or overflow the range of a floating-point word. Some accuracy may be gained by accumulating inner products. The places where this should be done are signaled by the appearance of the variables SUM and SUM1.

References:

1. Stewart, G. W. Incorporating origin shifts into the symmetric QR algorithm for symmetric tridiagonal matrices. *Comm. ACM 13* (June 1970), 365–367.
2. Wilkinson, J. H. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, 1965.

Algorithm:

```
      SUBROUTINE SYMQR(A,D,E,KO,N,NA,EPS,ABSCNV,VEC,TRD,FAIL)
C
C     EXPLANATION OF THE PARAMETERS IN THE CALLING SEQUENCE.
C
C         A       A DOUBLE DIMENSIONED ARRAY.  IF THE MATRIX IS NOT
C                 INITIALLY TRIDIAGONAL, IT IS CONTAINED IN THE LOWER
C                 TRIANGLE OF A.  IF EIGENVECTORS ARE NOT REQUESTED
C                 THE LOWER TRIANGLF OF A IS DESTROYED WHILE THE
C                 ELEMENTS ABOVE THF DIAGONAL ARE LEFT UNDISTURBED.
C                 IF EIGENVECTORS ARE REQUESTED, THEY ARE RETURNED IN THE
C                 COLUMNS OF A.
C
C         D       A SINGLY SUBSCRIPTED ARRAY.  IF THE MATRIX IS
C                 INITIALLY TRIDIAGONAL, D CONTAINS ITS DIAGONAL
C                 ELEMENTS.  ON RETURN D CONTAINS THE EIGENVALUES OF
C                 THE MATRIX.
C
C         E       A SINGLY SUBSCRIPTED ARRAY.  IF THE MATRIX IS
C                 INITIALLY TRIDIAGONAL, E CONTAINS ITS OFF-DIAGONAL
C                 ELEMENTS.  UPON RETURN E(I) CONTAINS THE NUMBER OF
C                 ITERATIONS REQUIRED TO COMPUTE THE APPROXIMATE
C                 EIGENVALUE D(I).
C
C         KO      A REAL VARIABLE CONTAINING AN INITIAL ORIGIN SHIFT TO
C                 BE USED UNTIL THE COMPUTED SHIFTS SETTLE DOWN.
C
C         N       AN INTEGER VARIABLE CONTAINING THE ORDER OF THE
C                 MATRIX.
C
C         NA      AN INTEGER VARIABLE CONTAINING THE FIRST DIMENSION
C                 OF THE ARRAY A.
C
C         EPS     A REAL VARIABLE CONTAINING A CONVERGENCE TOLERANCE.
C
C         ABSCNV  A LOGICAL VARIABLF CONTAINING THE VALUE .TRUE. IF
C                 THE ABSOLUTE CONVERGENCE CRITERION IS TO BE USED
C                 OR THE VALUE .FALSE. IF THE RELATIVE CRITERION
C                 IS TO BE USED.
C
C         VEC     A LOGICAL VARIABLE CONTAINING THE VALUE .TRUE. IF
C                 EIGENVECTORS ARE TO BE COMPUTED AND RETURNED IN
C                 THE ARRAY A AND OTHERWISE CONTAINING THE VALUE
C                 .FALSE..
```

```
C
C         TRD     A LOGICAL VARIABLF CONTAINING THF VALUE .TRUE.
C                 IF THF MATRIX IS TRIDIAGONAL AND LOCATED IN THE ARRAYS
C                 D AND E AND OTHERWISE CONTAINING THE VALUE .FALSE..
C
C         FAIL    AN INTEGER VARIABLE CONTAINING AN ERROR SIGNAL.
C                 ON RETURN THE EIGENVALUES IN D(FAIL+1),....,D(N)
C                 AND THEIR CORRESPONDING EIGENVECTORS MAY BE PRESUMED
-                 ACCURATE.
C
      REAL
     1A(NA,1),D(1),E(1),K0,D1,D2,K,EPS,S2,CON,NINF,TEST,CB,CC,CD,
     2C,S,TEMP,P,PP,Q,QQ,NORM,R,TITTER,SUM,SUM1,MAX
      INTEGER
     1N,NM1,NM2,NA,FAIL,I,I1,J,L,L1,LL,LL1,NL,NU,NUM1,SINCOS,RETURN
      LOGICAL
     1ABSCNV,VEC,TRD,SHFT
      TITTER = 50.
      NM1 = N-1
      NM2 = N-2
      NINF = 0.
      ASSIGN 500 TO SINCOS
C
C     SIGNAL ERROR IF N IS NOT POSITIVE.
C
      IF(N.GT.0) GO TO 1
      FAIL = -1
      RETURN
C
C     SPECIAL TREATMENT FOR A MATRIX OF ORDER ONE.
C
    1 IF(N.GT.1) GO TO 5
      IF(.NOT.TRD) D(1) = A(1,1)
      IF(VEC) A(1,1) = 1.
      FAIL = 0
      RETURN
C
C     IF THE MATRIX IS TRIDIAGONAL, SKIP THE REDUCTION.
C
    5 IF(TRD) GO TO 100
      IF(N.EQ.2) GO TO 80
C
C     REDUCE THE MATRIX TO TRIDIAGONAL FORM BY HOUSEHOLDERS METHOD.
C
      DO 70 L=1,NM2
      L1 = L+1
      D(L) = A(L,L)
      MAX = 0.
      DO 10 I=L1,N
   10 MAX = AMAX1(MAX,ABS(A(I,L)))
      IF(MAX.NE.0.) GO TO 13
      E(L) = 0.
      A(L,L) = 1.
      GO TO 70
   13 SUM = 0.
      DO 17 I=L1,N
      A(I,L) = A(I,L)/MAX
   17 SUM = SUM + A(I,L)**2
      S2 = SUM
      S2 = SQRT(S2)
      IF(A(L1,L) .LT. 0.) S2 = -S2
      E(L) = -S2*MAX
      A(L1,L) = A(L1,L) + S2
      A(L,L) = S2*A(L1,L)
      SUM1 = 0.
      DO 50 I=L1,N
      SUM = 0.
      DO 20 J=L1,I
   20 SUM = SUM + A(I,J)*A(J,L)
      IF(I.EQ.N) GO TO 40
      I1 = I+1
      DO 30 J=I1,N
   30 SUM = SUM + A(J,L)*A(J,I)
   40 E(I) = SUM/A(L,L)
   50 SUM1 = SUM1 + A(I,L)*E(I)
      CON = .5*SUM1/A(L,L)
      DO 60 I=L1,N
      E(I) = E(I) - CON*A(I,L)
      DO 60 J=L1,I
   60 A(I,J) = A(I,J) - A(I,L)*E(J) - A(J,L)*E(I)
   70 CONTINUE
   80 D(NM1) = A(NM1,NM1)
      D(N) = A(N,N)
      E(NM1) = A(N,NM1)
C
C     IF EIGENVECTORS ARE REQUIRED, INITIALIZE A.
C
  100 IF(.NOT.VEC) GO TO 180
C
C     IF THE MATRIX WAS TRIDIAGONAL, SET A EQUAL TO THE IDENTITY MATRIX.
C
      IF(.NOT.TRD .AND. N.NE.2) GO TO 130
      DO 120 I=1,N
      DO 110 J=1,N
  110 A(I,J) = 0.
  120 A(I,I) = 1.
      GO TO 180
C
C     IF THE MATRIX WAS NOT TRIDIAGONAL, MULTIPLY OUT THE
C     TRANSFORMATIONS OBTAINED IN THE HOUSEHOLDER REDUCTION.
C
  130 A(N,N) = 1.
      A(NM1,NM1) = 1.
      A(NM1,N) = 0.
      A(N,NM1) = 0.
      DO 170 L=1,NM2
      LL = NM2-L+1
      LL1 = LL+1
      DO 140 I=LL1,N
      SUM = 0.
      DO 135 J=LL1,N
  135 SUM = SUM + A(J,LL)*A(J,I)
  140 A(LL,I) = SUM/A(LL,LL)
      DO 150 I=LL1,N
      DO 150 J=LL1,N
  150 A(I,J) = A(I,J) - A(I,LL)*A(LL,J)
      DO 160 I=LL1,N
      A(I,LL) = 0.
  160 A(LL,I) = 0.
  170 A(LL,LL) = 1.
C
C     IF AN ABSOLUTE CONVERGENCE CRITERION IS REQUESTED
C     (ABSCNV=.TRUE.), COMPUTE THE INFINITY NORM OF THE MATRIX.
C
  180 IF(.NOT.ABSCNV) GO TO 200
      NINF = AMAX1(ABS(D(1))+ABS(E(1)),ABS(D(N))+ABS(E(NM1)))
      IF(N.EQ.2) GO TO 200
      DO 190 I=2,NM1
  190 NINF = AMAX1(NINF,ABS(D(I))+ABS(F(I))+ABS(E(I-1)))
C
C     START THE QR ITERATION.
C
  200 NU = N
      NUM1 = N-1
      SHFT = .FALSE.
      K1 = K0
      TEST = NINF*EPS
      E(N) = 0.
C
C     CHECK FOR CONVERGENCE AND LOCATE THE SUBMATRIX IN WHICH THE
C     QR STEP IS TO BE PERFORMED.
C
  210 DO 220 NNL=1,NUM1
      NL = NUM1-NNL+1
      IF(.NOT.ABSCNV) TEST = EPS*AMIN1(ABS(D(NL)),ABS(D(NL+1)))
      IF(ABS(E(NL)) .LE. TEST) GO TO 230
  220 CONTINUE
      GO TO 240
  230 E(NL) = 0.
      NL = NL+1
      IF(NL .NE. NU) GO TO 240
      IF(NUM1 .EQ. 1) RETURN
      IF(E(200).NE.0.) PRINT 2000,(D(I),E(I),I=1,NU)
 2000 FORMAT(1H010E12.4/(1H 10E12.4))
      NU = NUM1
      NUM1 = NU-1
      GO TO 210
  240 E(NU) = E(NU)+FLOAT(NUM1-NL)
      IF(1. .EQ. 1.) GO TO 250
      IF(0. .EQ. 1.) GO TO 250
      FAIL = NU
      RETURN
C
C     CALCULATE THE SHIFT.
C
  250 CB = (D(NUM1)-D(NU))/2.
      MAX = AMAX1(ABS(CB),ABS(E(NUM1)))
      CB = CB/MAX
      CC = (E(NUM1)/MAX)**2
      CD = SQRT(CB**2 + CC)
      IF(CB .NE. 0.) CD = SIGN(CD,CB)
      K2 = D(NU) - MAX*CC/(CB+CD)
      IF(SHFT) GO TO 270
      IF(ABS(K2-K1) .LT. .5*ABS(K2)) GO TO 260
      K1 = K2
      K = K0
      GO TO 300
  260 SHFT = .TRUE.
  270 K = K2
C
C     PERFORM ONE QR STEP WITH SHIFT K ON ROWS AND COLUMNS
C     NL THROUGH NU
C
  300 IF(E(200).NE.0. .AND. K.LE.1.E-14*ABS(D(NL))) K=0.
      P = D(NL) - K
      Q = E(NL)
      ASSIGN 310 TO RETURN
      GO TO SINCOS,(500)
  310 DO 380 I=NL,NUM1
C
C     IF REQUIRED, ROTATE THE EIGENVECTORS.
C
      IF(.NOT.VEC) GO TO 330
      DO 320 J=1,N
      TEMP = C*A(J,I) + S*A(J,I+1)
      A(J,I+1) = -S*A(J,I) + C*A(J,I+1)
  320 A(J,I) = TEMP
C
C     PERFORM THE SIMILARITY TRANSFORMATION AND CALCULATE THE NEXT
C     ROTATION.
C
  330 D(I) = C*D(I) + S*E(I)
      TEMP = C*E(I) + S*D(I+1)
      D(I+1) = -S*E(I) + C*D(I+1)
      E(I) = -S*K
      D(I) = C*D(I) + S*TEMP
      IF(I .EQ. NUM1) GO TO 380
      IF(ABS(S) .GT. ABS(C)) GO TO 350
      R = S/C
      D(I+1) = -S*E(I) + C*D(I+1)
      P = D(I+1) - K
      Q = C*E(I+1)
      ASSIGN 340 TO RETURN
      GO TO SINCOS,(500)
  340 E(I) = R*NORM
      E(I+1) = Q
      GO TO 380
  350 P = C*E(I) + S*D(I+1)
      Q = S*E(I+1)
      D(I+1) = C*P/S + K
      E(I+1) = C*E(I+1)
      ASSIGN 360 TO RETURN
      GO TO SINCOS,(500)
  360 E(I) = NORM
```

```
  380 CONTINUE
      TEMP = C*E(NUM1) + S*D(NU)
      D(NU) = -S*E(NUM1) + C*D(NU)
      E(NUM1) = TEMP
      GO TO 210
C
C     INTERNAL PROCEDURE TO CALCULATE THE ROTATION CORRESPONDING TC
C     THE VECTOR(P,Q).
C
  500 PP = ABS(P)
      QQ = ABS(Q)
      IF(QQ .GT. PP) GO TO 510
      NORM = PP*SQRT(1. + (QQ/PP)**2)
      GO TO 520
  510 IF(QQ .EQ. 0.) GO TO 530
      NORM = QQ*SQRT(1. + (PP/QQ)**2)
  520 C = P/NORM
      S = Q/NORM
      GO TO RETURN,(310,340,360)
  530 C = 1.
      S = 0.
      NORM = 0.
      GO TO RETURN,(310,340,360)
      END
```

CERTIFICATION OF ALGORITHM 245 [M1]

TREESORT 3 [Robert W. Floyd, *Comm. ACM 7* (Dec. 1964), 701]: PROOF OF ALGORITHMS—A NEW KIND OF CERTIFICATION

RALPH L. LONDON* (Recd. 27 Feb. 1969 and 8 Jan. 1970)

Computer Sciences Department and Mathematics Research Center, University of Wisconsin, Madison, WI 53706

* This work was supported by NSF Grant GP-7069 and the Mathematics Research Center, US Army under Contract Number DA-31-124-ARO-D-462.

ABSTRACT: The certification of an algorithm can take the form of a proof that the algorithm is correct. As an illustrative but practical example, Algorithm 245, *TREESORT 3* for sorting an array, is proved correct.

KEY WORDS AND PHRASES: proof of algorithms, debugging, certification, metatheory, sorting, in-place sorting

CR CATEGORIES: 4.42, 4.49, 5.24, 5.31

Certification of algorithms by proof. Since suitable techniques now exist for proving the correctness of many algorithms [for example, 3–7], it is possible and appropriate to certify algorithms with a proof of correctness. This certification would be in addition to, or in many cases instead of, the usual certification. Certification by testing still is useful because it is easier and because it also provides, for example, timing data. Nevertheless the existence of a proof should be welcome additional certification of an algorithm. The proof shows that an algorithm is debuggged by showing conclusively that no bugs exist.

It does not matter whether all users of an algorithm will wish to, or be able to, verify a sometimes lengthy proof. One is not required to accept a proof before using the algorithm any more than one is expected to rerun the certification tests. In both cases one could depend, in part at least, upon the author and the referee.

As an example of a certification by proof, the algorithm *TREESORT 3* [2] is proved to perform properly its claimed task of sorting an array $M[1{:}n]$ into ascending order. This algorithm has been previously certified [1], but in that certification, for example, no arrays of odd length were tested. Since *TREESORT 3* is a fast practical algorithm for in-place sorting and one with sufficient complexity so that its correctness is not immediately apparent, its use as the example is more than an abstract exercise. It is an example of considerable practical importance.

Outline of TREESORT 3 and method of proof. The algorithm is most easily followed if the array is viewed as a binary tree. $M[k \div 2]$ is the parent of $M[k]$, $2 \le k \le n$. In other words the children of $M[j]$ are $M[2j]$ and $M[2j+1]$ provided one or both of the children exist.

The first part of the algorithm permutes the M array so that for a segment of the array, each parent is larger than both of the children (one child if the second does not exist). Each call of the auxiliary procedure *siftup* enlarges the segment by causing one more parent to dominate its children. The second part of the algorithm uses *siftup* to make the parents larger over the whole array, exchanges $M[1]$ with the last element and repeats on an array one element shorter. The above statements are motivation and not part of the formal proof.

That *TREESORT 3* is correct is proved in three parts. First the procedure *siftup* is shown to perform as it is formally defined below. Then the body of *TREESORT 3*, which uses *siftup* in two ways, is shown to sort the array into ascending order. (The proof of the procedure *exchange* is omitted.) The proofs are by a method described in [3, 4, 7]: assertions concerning the progress of the computation are made between lines of code, and the proof consists of demonstrating that each assertion is true each time control reaches that assertion, under the assumption that the previously encountered assertions are true. Finally termination of the algorithm is shown separately.

The lines of the original algorithm have been numbered and the assertions, in the form of program comments, are numbered correspondingly. The numbers are used only to refer to code and to assertions and have no other significance. One extra begin-end pair has been inserted into the body of *TREESORT 3* in order that the control points of two assertions (3.1 and 4.1) could be distinguished. In *siftup* the assertions 10.1 and 10.2 express the correct result; in the body of *TREESORT 3* the assertions 9.3 and 9.4 do likewise.

Definition of siftup and notation. We now define formally the procedure *siftup*(i,n), where n is a formal parameter and not the length of the array M. Let $A(s)$ denote the set of inequalities $M[k \div 2] \ge M[k]$ for $2s \le k \le n$. (If $s > n \div 2$, then $A(s)$ is a vacuous statement.) If $A(i+1)$ holds before the call of *siftup*(i,n) and if $1 \le i \le n \le$ *array size*, then after *siftup*(i,n):

(1) $A(i)$ holds;

(2) the segment of the array $M[i]$ through $M[n]$ is permuted; and

(3) the segment outside $M[i]$ through $M[n]$ is unaltered.

In order to prove these properties of *siftup*, some notation is required. The formal parameter i will be changed inside *siftup*. Since i is called by value, that change will be invisible outside *siftup*. Nevertheless it is necessary to use the initial value of i as well as the current value of i in the proof of *siftup*. Let i_0 denote the value of i upon entry to *siftup*.

Similarly let M_0 denote the M array upon entry to *siftup*. The notation "$M = p(M_0)$ with $M :=$ *copy*" means "if $M[i] :=$ *copy* were done, M is some permutation of M_0 as described in (2) and (3) of the definition of *siftup*." "$M = p(M_0)$" means the same without the reference to $M[i] :=$ *copy* being done.

Code and assertions for siftup.

0 **procedure** *siftup*(i, n); **value** i, n; **integer** i, n;

1 **begin real** *copy*; **integer** j;

  comment

   1.1: $1 \le i_0 = i \le n \le$ *array size*

   1.2: $A(i_0+1)$

   1.3: $M = p(M_0)$;

```
2     copy := M[i];
3     loop: j := 2 × i;
      comment
        3.1: i ≤ n
        3.2: 2i = j
        3.3: i = i0 or i ≥ 2i0
        3.4: M = p(M0) with M[i] := copy
        3.5: A(i0) or (i = i0 and A(i0+1))
        3.6: M[i÷2] > copy or i = i0
        3.7: M[i÷2] ≥ M[i] or i = i0;
4     if j ≤ n then
5     begin if j < n then
6a      begin if M[j+1] > M[j] then
6b        j := j + 1 end;
        comment
          6.1: i = j ÷ 2
          6.2: 2i ≤ j ≤ n
          6.3: i = i0 or i ≥ 2i0
          6.4: M = p(M0) with M[i] := copy
          6.5: A(i0) or (i = i0 and A(i0+1))
          6.6: M[i÷2] > copy or i = i0
          6.7: M[i÷2] ≥ M[i] or i = i0
          6.8: (2i < n and M[j] = max(M[2i], M[2i+1])) or
            (2i = n and M[j] = M[n])
          6.9: M[i] ≥ M[j] or i = i0;
7       if M[j] > copy then
8a      begin M[i] := M[j];
          comment
            8.1: i = i0 or i ≥ 2i0
            8.2: 2i ≤ j ≤ n
            8.3: M[j÷2] = M[i] = M[j] > copy
            8.4: M[i÷2] ≥ M[j] or i = i0
            8.5: M = p(M0) with M[j] := copy
            8.6: A(i0);
8b        i := j;
          comment
            8.7: i ≥ 2i0
            8.8: i = j ≤ n
            8.9: M[i÷2] > copy
            8.10: M[i÷2] ≥ M[i]
            8.11: M = p(M0) with M[i] := copy
            8.12: A(i0);
8c      go to loop end
9     end;
      comment
        9.1: M[j] ≤ copy if reached from 7 or
          2i = j > n if reached from 4;
10    M[i] := copy;
      comment
        10.1: M = p(M0)
        10.2: A(i0);
11  end siftup;
```

Verification of the assertions of siftup. Reasons for the truth of each assertion follow:

1.1–1.2: Assumptions for using *siftup*.

1.3: p is the identity permutation.

3.1–3.7: If reached from 2,

- 3.1: 1.1.
- 3.2: 3.
- 3.3, 3.5–3.7: $i = i_0$ by 1.1. 3.5 also requires 1.2.
- 3.4: 1.3 and 2.

If reached from 8, respectively, 8.8, 3, 8.7, 8.11, 8.12, 8.9 and 8.10.

6.1: At 3.2 $j = 2i$ and by 6b, j might be $2i + 1$. $i = j \div 2$ in either case.

6.2: After 4, $j \leq n$. j is altered from 3.1 to 6.2 only at 6b. Before 6b, $j < n$ by 5. Hence $j \leq n$ at 6.2. $2i \leq j$ by 6.1.

6.3–6.7: 3.3–3.7, respectively.

6.8: If 4 is **true** and 5 is **false**, $j = 2i = n$ (using 3.2) so the second clause of 6.8 holds. If 4 is **true** and 5 is **true**, then at 6a, $2i = j < n$ (using 3.2) so $M[j+1] = M[2i+1]$ is defined. Now at 6.8, $j = 2i$ or $j = 2i+1$. In either case, by 6a and 6b, the first clause of 6.8 holds.

6.9: By 6.5 $i \neq i_0$ gives $A(i_0)$. $2i_0 \leq 2i \leq j \leq n$ by 6.3 and 6.2. Hence $A(i_0)$ and 6.1 give $M[i] = M[j \div 2] \geq M[j]$.

8.1: 6.3.

8.2: 6.2.

8.3: $i = j \div 2$ by 6.1, $M[i] = M[j]$ by 8a and $M[j] > copy$ by 7.

8.4: 6.7 and 6.9.

8.5: 6.4 requires that $M[i]$ be replaced by *copy*. Since $M[i] = M[j]$ by 8a, $M[j]$ may equally well be replaced with *copy*. 8.1 and 8.2 give $i_0 \leq i \leq n$ so that the change to M at 8a is in the segment $M[i_0]$ through $M[n]$.

8.6: By 8a and if 6.8 (first clause) holds, $M[i] \geq M[2i]$ and $M[i] \geq M[2i+1]$. By 8a and if 6.8 (second clause) holds, $M[i] = M[j] = M[n] = M[2i]$ and $M[2i+1]$ does not exist for this call of *siftup*. $A(i_0+1)$ holds at 6.5 since $A(i_0)$ implies $A(i_0+1)$. If $i = i_0$, $A(i_0+1)$ and the relations above on $M[i]$ give $A(i_0)$. If $i \neq i_0$, then 8a, 8.4, $A(i_0)$ at 6.5 and the relations above on $M[i]$ give $A(i_0)$ at 8.6.

8.7: 8b, 8.1 and 8.2.

8.8: 8b and 8.2.

8.9: 8b and 8.3.

8.10: At 8.6, $2i_0 \leq j \leq n$ by 8.1 and 8.2. Hence by 8.6, $M[j \div 2] \geq M[j]$. Use 8b on $M[j \div 2] \geq M[j]$.

8.11: 8b and 8.5.

8.12: 8.6.

9.1: 9.1 is reached only if 7 is **false** or if 4 is **false**. $2i = j$ by 3.2.

10.1–10.2: If reached from 7,

- 10.1: 6.4 and 10. (6.2 and 6.3 give $i_0 \leq i \leq n$ ensuring the change to M at 10 is in the segment $M[i_0]$ through $M[n]$.)
- 10.2: By 10, 9.1, 6.2 and 6.8, $M[i] = copy \geq M[j] \geq M[2i]$ and, if $M[2i+1]$ exists, $M[j] \geq M[2i+1]$. If $i = i_0$, 10.2 follows as in 8.6. If $i \neq i_0$, 6.6 and 10 give $M[i \div 2] > copy = M[i]$. $A(i_0)$ at 6.5 now gives $A(i_0)$ at 10.2.

If reached from 4,

- 10.1: 3.4 and 10. (3.1 and 3.3 give $i_0 \leq i \leq n$.)
- 10.2: $2i > n$ means no relations in $A(i_0)$ of the form $M[i] \geq \cdots$. If $i = i_0$, 3.5 gives 10.2. If $i \neq i_0$, 3.6 and 10 give $M[i \div 2] > copy = M[i]$. $A(i_0)$ at 3.5 now gives 10.2.

Code and assertions for the body of TREESORT 3.

```
0   integer i;
    comment
      0.1: A(n÷2+1);
1   for i := n÷2 step −1 until 2 do
2   begin
      comment
        2.1: A(i+1)
        2.2: Assumptions of siftup satisfied;
3     siftup(i,n);
      comment
        3.1: A(i);
4   end;
    comment
      4.1: M[p] ≤ M[p+1] for n + 1 ≤ p ≤ n − 1
      4.2: A(2), i.e. M[k÷2] ≥ M[k] for 4 ≤ k ≤ n;
5   for i := n step −1 until 2 do
6   begin
      comment
        6.1: M[p] ≤ M[p+1] for i + 1 ≤ p ≤ n − 1
        6.2: M[k÷2] ≥ M[k] for 4 ≤ k ≤ i
        6.3: M[i+1] ≥ M[r] for 1 ≤ r ≤ i
        6.4: Assumptions of siftup satisfied;
```

```
7     siftup (1,i);
      comment
        7.1: M[p] ≤ M[p+1] for i + 1 ≤ p ≤ n − 1
        7.2: M[k÷2] ≥ M[k] for 2 ≤ k ≤ i
        7.3: M[1] ≥ M[r] for 2 ≤ r ≤ i
        7.4: M[i+1] ≥ M[1];
8     exchange (M[1], M[i]);
      comment
        8.1: M[i] ≥ M[r] for 1 ≤ r ≤ i − 1
        8.2: M[p] ≤ M[p+1] for i ≤ p ≤ n − 1
        8.3: M[k÷2] ≥ M[k] for 4 ≤ k ≤ i − 1;
9   end;
    comment
      9.1: M[p] ≤ M[p+1] for 2 ≤ p ≤ n − 1
      9.2: M[2] ≥ M[1]
      9.3: M[p] ≤ M[p+1] for 1 ≤ p ≤ n − 1, i.e. M is fully
           ordered
      9.4: M is a permutation of M0;
```

Verification of the assertions for the body of TREESORT 3. Reasons for the truth of each assertion follow:

0.1: Vacuous statement since $2(n\div 2+1) > n$.

2.1: If reached from 0.1, by 1 substitute $i = n\div 2$ in 0.1.
If reached from 3.1, by 1 substitute $i = i + 1$ in 3.1 to account for the change in i from 3.1 to 2.1.

2.2: 2.1, the bound on i implied by 1 and the array size being n.

3.1: 2.1 and the definition of *siftup*(i, n).

4.1: Vacuous statement.

4.2: If $n \geq 4$, 3 is executed; hence 3.1 with $i = 2$. If $n \leq 3$, vacuous statement.

6.1–6.3: If reached from 4.1,
6.1–6.2: By 5 substitute $i = n$ in 4.1 and 4.2.
6.3: Vacuous statement for $i = n$.
If reached from 8.1, by 5 substitute $i = i + 1$ in 8.2, 8.3 and 8.1, respectively.

6.4: 5 and 6.2, i.e. $A(2)$ for the subarray $M[1{:}i]$.

7.1: 6.1 and (3) of *siftup*.

7.2: 6.2 and (1) of *siftup*.

7.3: 7.2 noting that $M[1] = M[k\div 2]$ if $k = 2$ and using the transitivity of $\geq$.

7.4: Vacuous for $i = n$. Otherwise 6.3 for the appropriate r since by (2) of *siftup*, $M[1]$ at 7.3 is one of the $M[r]$, $1 \leq r \leq i$, at 6.3.

8.1: 7.3 with the changes caused by 8 (only $M[1]$ and $M[i]$ are altered by 8).

8.2: By 8 substitute $M[i]$ for $M[1]$ in 7.4; then 7.1 also holds for $p = i$.

8.3: 7.2 excluding only the one or two relations $M[1] \geq \cdots$, and the one relation $\cdots \geq M[i]$.

9.1–9.3: If $n \geq 2$, 8 is executed;
9.1: 8.2 with $i = 2$.
9.2: 8.1 with $i = 2$.
9.3: 9.1 and 9.2.
If $n \leq 1$, 9.1–9.3 are vacuous statements.

9.4: The only operations done to M are *siftup* and *exchange* all of which leave M as a permutation of M_0.

Proof of termination of TREESORT 3. Provided *siftup* and *exchange* terminate, it is clear that *TREESORT 3* terminates. Note that each parameter of *siftup* is called by value so that i is not changed in the body of the for loops.

The procedure *exchange* certainly terminates. In *siftup* the only possibility for an unending loop is from 3 to 8b and back to 3. Note that all changes to i (only at 8b) and to j (only at 3 and 6b) occur in this loop and that on each cycle of this loop both i and j are changed. By the test at 4, it is sufficient to show that j strictly increases in value. $i \geq 1$ means $2i > i$. At 8b, $j = i < 2i$ while at 3, $j = 2i$, i.e. $j(\text{at } 3) = 2i > i = j(\text{at } 8\text{b})$. Hence each setting to j at 3 strictly increases the value of j. The only other setting to j (at 6b), if made, similarly increases the value of j.

REFERENCES:

1. ABRAMS, P. S. Certification of Algorithm 245. *Comm. ACM 8* (July 1965), 445.
2. FLOYD, R. W. Algorithm 245, TREESORT 3. *Comm. ACM 7* (Dec. 1964), 701.
3. FLOYD, R. W. Assigning meanings to programs. Proc. of a Symposium in Applied Mathematics, Vol. 19—Mathematical Aspects of Computer Science, J. T. Schwartz (Ed.), American Math. Society, Providence, R. I., 1967, pp. 19–32.
4. KNUTH, D. E. *The Art of Computer Programming, Vol. 1—Fundamental Algorithms.* Addison-Wesley, Reading, Mass., 1968, Sec. 1.2.1.
5. MCCARTHY, J. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*, P. Braffort and D. Hirschberg (Eds.), North Holland, Amsterdam, 1963, pp. 33–70.
6. MCCARTHY, J., AND PAINTER, J. A. Correctness of a compiler for arithmetic expressions. Proc. of a Symposium in Applied Mathematics, Vol. 19—Mathematical Aspects of Computer Science, J. T. Schwartz (Ed.), American Math. Society, Providence, R. I., 1967, pp. 33–41.
7. NAUR, P. Proof of algorithms by general snapshots. *BIT 6* (1966), 310–316.

REMARK ON ALGORITHM 201 [M1]
SHELLSORT [J. Boothroyd, *Comm. ACM 6* (Aug. 1963), 445]

J. P. CHANDLER AND W. C. HARRISON* (Recd. 19 Sept. 1969)

Department of Physics, Florida State University, Tallahassee, FL 32306

* This work was supported in part by AEC Contract No. AT-(40-1)-3509. Computational costs were supported in part by National Science Foundation Grant GJ 367 to the Florida State University Computing Center.

KEY WORDS AND PHRASES: sorting, minimal storage sorting, digital computer sorting
CR CATEGORIES: 5.31

Hibbard [1] has coded this method in a way that increases the speed significantly. In SHELLSORT, each stage of each sift consists of successive pair swaps. The modification replaces each set of n pair swaps by one "save," n − 1 moves, and one insertion.

Table I gives timing information for ALGOL, FORTRAN, and COMPASS (assembly language) versions of SHELLSORT and the

TABLE I. SORTING TIMES IN SECONDS FOR 10,000 RANDOMLY ORDERED NUMBERS ON THE CDC 6400 COMPUTER

Algorithm	*Source Language*		
	ALGOL	FORTRAN	COMPASS
SHELLSORT	53.40	7.18	2.38
SHELLSORT2	36.56	5.98	1.87

modified version (called SHELLSORT2), for the CDC 6400 computer. The savings in time achieved by the modification are 32%, 17%, and 21%, respectively. The savings are greater than this when vectors of more than one word each are being sorted.

The comparative execution times of the ALGOL and FORTRAN versions, for these compilers, are quite interesting.

REFERENCES:

1. HIBBARD, T. N. An empirical study of minimal storage sorting. *Comm. ACM 6* (May 1963), 206.

REMARK ON ALGORITHM 351 [D1] MODIFIED ROMBERG QUADRATURE

[G. Fairweather, *Comm. ACM 12* (June 1969), 324]

GEORGE C. WALLICK

Mobil Research and Development Corporation, Field Research Laboratory, P. O. Box 900, Dallas, TX 75221

KEY WORDS AND PHRASES: numerical integration, Romberg quadrature, modified Romberg quadrature, trapezoid values, rectangle values

CR CATEGORIES: 5.16

Algorithm 351 was compiled and run successfully in FORTRAN IV on a CDC 6400 computer. Computation times for equivalent orders were essentially the same as for a FORTRAN version of Algorithm 60 Romberg Integration [1]; storage requirements were approximately 20 percent greater.

Algorithm 351 incorporates two modifications to the standard Romberg algorithm, each designed to reduce roundoff: (1) the Krasun and Prager [3] replacement of the table of trapezoidal values T_j^k with a table of rectangular values R_j^k; (2) the method proposed by Rutishauser [6] for the evaluation of the rectangular sums R_0^k. Since neither of these modifications has been properly evaluated we have chosen to compare integral values returned by five variants of the Romberg algorithm:

1. Conventional Romberg integration as described by Algorithm 60
2. A Krasun and Prager modification of Algorithm 60 (T_j^k table replaced by R_j^k table)
3. A Rutishauser modification of Algorithm 60 (T_j^k table extrapolation with improved evaluation of the R_0^k)
4. Modified Romberg integration as described by Algorithm 351 (R_j^k table; improved R_0^k evaluation)
5. Algorithm 351 with the Rutishauser procedure replaced by the standard evaluation of the R_0^k (R_j^k table extrapolation)

The following test integrals were investigated.

A. $\displaystyle\int_{.01}^{1.1} x^{-\alpha}\, dx, \quad \alpha = 3.0, 4.0, 5.0$

B. $\displaystyle\int_0^1 (1 + x^\alpha)^{-1}\, dx, \quad \alpha = 1.0, 4.0$

C. $\displaystyle\int_1^{10} \ln x\, dx$

D. $\displaystyle\int_0^5 e^{-x^2}\, dx$

Integral A was suggested by Thacher [7], Integral B by Rabinowitz [5], Integral C by Hillstrom [2], and Integral D by Hillstrom and by Kubik [4]. All computation was carried out in CDC 6400 single-precision floating-point arithmetic. Results were recorded to 14 decimal digits. (CDC 6400 word length corresponds to 14+ decimal digits.) The data obtained in this manner are summarized in Tables I–IV.

For a specified order of extrapolation m, Algorithm 60 variants require $2^m + 1$ function evaluations and return T_m^0. Algorithm 351 requires $2^{(m+1)} + 1$ function evaluations and returns T_m^1. Thus one cannot meaningfully compare integral values returned by the two algorithms for the same specified order. We have therefore chosen to compare integral values resulting from the same number of function evaluations and have tabulated these data in terms of the Algorithm 60 order m. The corresponding specified order for Algorithm 351 variants is $m - 1$.

In each example considered, Algorithm 351 returns integral values for the optimum extrapolation order that are more accurate than the Algorithm 60 solutions by from one to two significant figures. There is, of course, no increase in the rate of convergence and little difference in solution accuracy for approximation orders less than that corresponding to the maximum attainable accuracy. If one were interested in, e.g. six or eight significant figure accuracy, either algorithm would be satisfactory. If accuracy requirements are not severe and one is satisfied with integral values correct to a number of significant figures less than half the computer word length, either algorithm may be used. If one seeks the maximum achievable accuracy, Algorithm 351 is clearly the proper choice.

Tables I–IV include data recorded when the order was overspecified, i.e. when m was greater than that required for optimum accuracy. For both algorithms the accuracy at first increases with increasing order. This continues until an optimum accuracy obtains. With Algorithm 60 a further increase in m results in a decline, at times rather rapid, in evaluation accuracy. With Algorithm 351 there is little loss in accuracy with increasing order. The accuracy decline rate is strongly retarded and in many cases practically eliminated. This is a very significant result.

In routine use of the algorithms, the unwary may overestimate the order required for optimum convergence (Algorithm 60 terminates only when a specified order has been obtained) or may specify an accuracy criterion for termination that cannot be satisfied. With Algorithm 351 the only loss is that of computer time; with Algorithm 60 solution accuracy may be impaired.

From the data presented in Tables I–IV we may determine the extent to which each of the procedural modifications contributes to the overall superiority of Algorithm 351. It is immediately evident that the Krasun and Prager modification has little effect either on the accuracy of the algorithms or on the loss of accuracy as the optimum order is exceeded. Results obtained using this modification differ from those returned by Algorithm 60 by at most 2 in the 14th figure. When the Rutishauser procedure is subtracted from Algorithm 351, the algorithm becomes, for all practical purposes, equivalent in accuracy to Algorithm 60. This conclusion has been further supported by results obtained in the evaluation of eight additional test integrals selected from the literature.

If, on the other hand, the Rutishauser procedure is added to Algorithm 60, the results obtained are essentially the same as those recorded for Algorithm 351. Clearly the Rutishauser modification is the dominant factor determining the superiority of Algorithm 351.

The success of the Rutishauser modification tempts one to expand the procedure to include an additional summation level. Experiments with such expansions indicate that they may be of value where slow Romberg convergence requires the use of orders $m > 13$.

The following changes are suggested as possible improvements in the algorithm. The integration interval $(B-A)$ is now computed $K + 2$ times where K is the order of approximation on exit

TABLES. Comparisons of Romberg Method Variations

(KP = Krasun-Prager Modification; RUT = Rutishauser Modification; NSF = Number of Significant Figures)

I. In the Evaluation of $I(\alpha) = \int_0^1 (1 + x^\alpha)^{-1}\, dx$

$I(1) = 0.69314\ 71805\ 59945$; $I(4) = 0.86697\ 29873\ 3991$

α	Romberg Order m	Variations Returning T_m^0: Algorithm 60, Digits 1–14	NSF	Algorithm 60 + KP, Digits 11–14	NSF	Algorithm 60 + RUT, Digits 11–14	NSF	Variations Returning T_m^1: Algorithm 351 (KP + RUT), Digits 6–14	NSF	Algorithm 351 (KP only), Digits 11–14	NSF
1.0	3	69314 74776 4482	6	4482	6	4482	6	79014 8123	5	8123	5
	4	69314 71819 1673	8	1673	8	1673	8	71830 7192	8	7192	8
	5	69314 71805 6227	11	6228	11	6227	11	71805 6360	11	6360	11
	6	69314 71805 5991	13	5992	13	5992	13	71805 5993	13	5992	13
	7	69314 71805 5987	12	5988	12	5991	13	71805 5992	13	5988	12
	8	69314 71805 5984	12	5984	12	5990	13	71805 5992	13	5984	12
	9	69314 71805 5971	12	5972	12	5989	12	71805 5990	13	5972	12
	10	69314 71805 5951	12	5951	12	5988	12	71805 5989	12	5951	12
	11	69314 71805 5906	11	5906	11	5991	13	71805 5990	13	5906	11
	12	69314 71805 5822	11	5822	11	5987	12	71805 5989	12	5822	11
4.0	4	86697 29736 8070	7	8070	7	8070	7	30046 3711	7	3711	7
	5	86697 29872 2539	9	2539	9	2539	9	29872 1216	9	1216	9
	6	86697 29873 4006	12	4006	12	4007	12	29873 4005	12	4003	12
	7	86697 29873 3983	12	3984	12	3987	13	29873 3988	13	3984	12
	8	86697 29873 3977	12	3978	12	3986	13	29873 3987	13	3979	12
	9	86697 29873 3963	12	3964	12	3985	12	29873 3986	13	3964	12
	10	86697 29873 3939	11	3940	11	3985	12	29873 3985	12	3940	11
	11	86697 29873 3890	11	3890	11	3984	12	29873 3986	13	3890	11
	12	86697 29873 3787	11	3788	11	3983	12	29873 3985	12	3788	11

II. In the Evaluation of $I(\alpha) = \int_{.01}^{1.1} x^{-\alpha}\, dx$

$I(3) = 0.49995\ 86776\ 85950 \times 10^4$; $I(4) = 0.33333\ 30828\ 95066 \times 10^6$; $I(5) = 0.24999\ 99982\ 9247 \times 10^8$

α	Romberg Order m	Variations Returning T_m^0: Algorithm 60, Digits 1–14	NSF	Algorithm 60 + KP, Digits 11–14	NSF	Algorithm 60 + RUT, Digits 11–14	NSF	Variations Returning T_m^1: Algorithm 351 (KP + RUT), Digits 6–14	NSF	Algorithm 351 (KP only), Digits 11–14	NSF
3.0	8	50289 45604 1249	2	1249	2	1255	2	49952 9475	2	9469	2
	9	50007 88217 4010	3	4010	3	4037	3	88324 8156	3	8128	3
	10	49996 05996 3754	5	3755	5	3813	5	05997 5088	5	5029	5
	11	49995 86888 2917	7	2917	7	3041	7	86888 3087	7	2962	7
	12	49995 86777 0553	10	0553	10	0814	10	86777 0815	10	0553	10
	13	49995 86776 8069	10	8070	10	8588	12	86776 8590	12	8070	10
	14	49995 86776 7547	10	7549	10	8585	12	86776 8587	12	7549	10
	15	49995 86776 6495	10	6496	10	8581	12	86776 8583	12	6496	10
4.0	8	33918 76383 3713	1	3713	1	3717	1	83321 8573	1	8568	1
	9	33362 40891 0012	3	0011	3	0028	3	41103 2353	3	2337	3
	10	33333 86458 8643	4	8642	4	8682	4	86461 5904	4	5865	4
	11	33333 31207 4466	7	4466	7	4547	7	31207 4679	7	4598	7
	12	33333 30829 8056	9	8055	9	8220	9	30829 8220	9	8056	9
	13	33333 30828 9178	11	9178	11	9508	13	30828 9509	13	9178	11
	14	33333 30828 8842	10	8843	10	9500	12	30828 9501	12	8843	10
	15	33333 30828 8163	10	8163	10	9497	12	30828 9499	12	8163	10
5.0	8	25979 73076 7608	1	7608	1	7611	1	82577 2026	1	2023	1
	9	25058 17539 3846	2	3846	2	3857	2	17890 9312	2	9300	2
	10	25001 31264 6257	4	6257	4	6282	4	31270 0511	4	0486	4
	11	25000 01021 0524	6	0524	6	0576	6	01021 0887	6	0835	6
	12	24999 99985 6515	9	6515	9	6621	9	99985 6622	9	6516	9
	13	24999 99982 9053	11	9053	11	9267	12	99982 9268	12	9054	11
	14	24999 99982 8817	11	8818	11	9242	13	99982 9243	13	8818	11
	15	24999 99982 8379	10	8380	10	9241	12	99982 9242	13	8380	10

III. In the Evaluation of $I = \int_1^{10} \ln x\, dx = 14.025\ \ 85092\ 99404\ 6$

α	Romberg Order m	Variations Returning T_m^0: Algorithm 60, Digits 1–14	NSF	Algorithm 60 + KP, Digits 11–14	NSF	Algorithm 60 + RUT, Digits 11–14	NSF	Variations Returning T_m^1: Algorithm 351 (KP + RUT), Digits 6–14	NSF	Algorithm 351 (KP only), Digits 11–14	NSF
	4	14025 60234 7275	5	7275	5	7275	5	60498 3885	5	3885	5
	5	14025 84455 4627	6	4627	6	4627	6	84433 5675	6	5675	6
	6	14025 85085 2042	8	2043	8	2043	8	85085 0505	8	0505	8
	7	14025 85092 9556	11	9556	11	9556	11	85092 9552	11	9551	11
	8	14025 85092 9938	13	9938	13	9939	13	85092 9939	13	9938	13
	9	14025 85092 9937	13	9937	13	9940	14	85092 9940	14	9937	13
	10	14025 85092 9934	12	9934	12	9939	13	85092 9940	14	9934	12
	11	14025 85092 9928	12	9929	12	9939	13	85092 9940	14	9929	12
	12	14025 85092 9916	12	9916	12	9940	14	85092 9939	13	9916	12

IV. In the Evaluation of $I = \int_0^5 e^{-x^2}\, dx = 0.88622\ \ 69254\ 51396$

α	Romberg Order m	Variations Returning T_m^0: Algorithm 60, Digits 1–14	NSF	Algorithm 60 + KP, Digits 11–14	NSF	Algorithm 60 + RUT, Digits 11–14	NSF	Variations Returning T_m^1: Algorithm 351 (KP + RUT), Digits 6–14	NSF	Algorithm 351 (KP only), Digits 11–14	NSF
	5	88622 59970 9402	5	9043	5	9042	5	59296 9073	5	9073	5
	6	88622 69310 8538	7	8539	7	8541	7	69308 5739	7	5736	7
	7	88622 69254 4529	10	4529	10	4535	10	69254 4570	10	4564	10
	8	88622 69254 5117	12	5117	12	5134	12	69254 5135	13	5117	12
	9	88622 69254 5093	12	5094	12	5131	12	69254 5134	12	5095	12
	10	88622 69254 5053	11	5054	11	5135	13	69254 5134	12	5054	11
	11	88622 69254 4974	11	4975	11	5130	12	69254 5133	12	4976	11
	12	88622 69254 4801	11	4802	11	5129	12	69254 5131	12	4803	11
	13	88622 69254 4463	10	4463	10	5128	12	69254 5129	12	4464	10
	14	88622 69254 3801	10	3802	10	5125	12	69254 5127	12	3803	10

from the routine. We suggest an initial definition of a variable, e.g. $SH = (B-A)$ and the replacement of $(B-A)$ by SH in these statements where $(B-A)$ appears. Initialization should also include a test to insure that the maximum extrapolation order $MAXE$ permitted is less than or equal to 15 with a possible replacement $MAXE = 15$ if this condition is violated. Alternatively, one could replace the statement DO 11 $K = 1$, $MAXE$ with DO 11 $K = 1$, 15 and test for $K < MAXE$ prior to executing statement no. 11. The GO TO 3 statement following statement no. 1 should read GO TO 4. If $N \leq 32$, N is also ≤ 512.

Upon exit, the input parameter $MAXE$ is assigned either the value $MAXE = K$, where K is the approximation order, or $MAXE = 0$ if the accuracy criterion has not been satisfied. We believe that it is poor programming practice to have a subroutine alter the value of an input parameter. We suggest the addition of an output parameter, e.g. $MFIN = K$ which returns the order on exit. Where we now set $MAXE = 0$, we could set $MFIN = 16$. One can test as easily for $MFIN \leq 15$ as for $MAXE = 0$. This would eliminate the necessity for resetting $MAXE$ each time the subroutine is entered. It is also useful to return the final value of the accuracy ERR. In the event that $MAXE = 0$, one could test ERR to determine whether or not the returned integral value falls within acceptable limits.

In practical applications we prefer to express the procedure as a function subprogram and to add the name of the generating function F to the argument list. We also consider a test for relative error rather than absolute error to be more useful in routine use of the algorithm.

The author wishes to thank the Mobil Research and Development Corporation for permission to publish this information.

References:

1. Bauer, F. L. Algorithm 60, Romberg integration. *Comm. ACM 4* (June 1961), 255.
2. Hillstrom, K. Certification of Algorithm 257, Havie integrator. *Comm. ACM 9* (Nov. 1966), 795.
3. Krasun, A. M., and Prager, W. Remark on Romberg quadrature. *Comm. ACM 8* (Apr. 1965), 236–237.
4. Kubik, R. N. Algorithm 257, Havie integrator. *Comm. ACM 8* (June 1965), 381.
5. Rabinowitz, P. Automatic integration of a function with a parameter. *Comm. ACM 9* (Nov. 1966), 804–806.
6. Rutishauser, H. Description of Algol 60. In *Handbook for Automatic Computation, Vol. 1*, Springer-Verlag, New York, 1967, Part a, 105–106.
7. Thacher, H. C., Jr. Certification of Algorithm 60, Romberg integration. *Comm. ACM 5* (Mar. 1962), 168.

REMARK ON ALGORITHM 361 [G6]
PERMANENT FUNCTION OF A SQUARE MATRIX I AND II [Bruce Shriver, P. J. Eberlein, and R. D. Dixon, *Comm. ACM 12* (Nov. 1969), 634]

BRUCE SHRIVER, P. J. EBERLEIN, AND R. D. DIXON (Recd. 22 Jan. 1970)
State University of New York at Buffalo, Amherst, NY 14226

KEY WORDS AND PHRASES: matrix, permanent, determinant
CR CATEGORIES: 5.30

The authors would like to cite the following misprints in the above two algorithms:
(A) In procedure *per*1(*A*, *n*)
  (1) in line 43, the variable name *pira* should be *pera*
  (2) in line 44, the variable name *per* should be *per*1.
(B) In procedure *per*2(*A*, *n*)
  (1) in line 47, the variable name *per* should be *per*2.

REMARK ON ALGORITHM 382 [G6]
COMBINATIONS OF *M* OUT OF *N* OBJECTS [Phillip J. Chase, *Comm. ACM 13* (June 1970), 368]

PHILLIP J. CHASE (Recd. 18 Mar. 1969 and 31 Oct. 1969)
Department of Defense, Fort Meade, MD 20755

KEY WORDS AND PHRASES: permutations and combinations, permutations
CR CATEGORIES: 5.39

The following driver program illustrates the use of Algorithm 382.

```
begin integer m, n, i, x, y, z, q, r;  Boolean done;
  integer array a, b, c[1:30],  p[0:31];
  procedure TWIDDLE (x, y, z, done, p);
  comment Body of TWIDDLE is to be inserted here;
  comment TWIDDLE is here used to generate: (1) all combi-
    nations c[1:m] of a[1:n]. Here we take a[i] equal to i, each i.
    (2) all sequences b[1:n] consisting of m 1's and (n−m) 0's.
    The user must supply m and n such that 0 ≤ m ≤ n and 1 ≤ n.
    (Our declarations here require n ≤ 30.);
  ininteger (2, m);  ininteger (2, n);
  for i := n step −1 until 1 do a[i] := i;
  comment We initialize the parameters p and done of
    TWIDDLE as follows;
  r := n − m;
  for i := r step −1 until 1 do p[i] := 0;
  for i := m step −1 until 1 do p[r+i] := i;
  p[0] := n + 1;  p[n+1] := −2;  done := false;
  if m = 0 then p[1] := 1;
  comment We initialize c[1:m];
  for i := m step −1 until 1 do c[i] := a[r+i];
  comment Next we initialize b[1:n];
  for i := m step −1 until 1 do b[r+i] := 1;
  for i := r step −1 until 1 do b[i] := 0;
  comment Now we generate and output our successive com-
    binations and sequences;
  q := 0;
L:
  q := q + 1;
  outinteger (1, q);
  for i := m − 1 step −1 until 0 do outinteger (1, c[m−i]);
  for i := n − 1 step −1 until 0 do outinteger (1, b[n−i]);
  TWIDDLE (x, y, z, done, p);
  if ¬ done then
  begin
    c[z] := a[x];  b[x] := 1;  b[y] := 0;  go to L
  end
end of driver program
```

REMARK ON ALGORITHM 383 [G6]
PERMUTATIONS OF A SET WITH REPETITIONS [Phillip J. Chase, *Comm. ACM 13* (June 1970), 368]

PHILLIP J. CHASE (Recd. 4 Aug. 1969 and 13 Feb. 1970)
Department of Defense, Fort Meade, MD 20755

KEY WORDS AND PHRASES: permutations and combinations, permutations
CR CATEGORIES: 5.39

The following driver program illustrates the use of Algorithm 383.

```
begin integer x, y, k, u, J, Q, I, L;  Boolean done;
  integer array p[0:31], A, C, N[1:30];
  procedure EXTENDED TWIDDLE (x, y, k, u, done, p);
  comment Body of EXTENDED TWIDDLE is to be inserted
    here;
  comment Program uses EXTENDED TWIDDLE in generat-
    ing all permutations of C[I] numbers equal to N[I] (I=1 to J).
    They are successively stored in A and output. The user must
    supply: 1. J (indexing above requires J≤30); 2. C[I] (I=1 to
    J), each ≥ 1 (indexing above requires C[1]+···+C[J]≤30);
    3. N[I] (I=1 to J), distinct numbers (declarations above
    requires integer type);
  ininteger (2, J);
  for I := 1 step 1 until J do
  begin ininteger (2, C[I]);  ininteger (2, N[I]) end;
  comment The array A is initialized;
  L := 1;
  for I := 1 step 1 until J do
  for Q := C[I] step −1 until 1 do
  begin A[L] := N[I];  L := L + 1 end;
  comment EXTENDED TWIDDLE is initialized;
  L := 1;
  for I := 1 step 1 until J do
  for Q := C[I] step −1 until 1 do
  begin p[L] := J − I + 1;  L := L + 1 end;
  p[0] := p[L] := J + 1;
  done := false;
  k := J;  u := L;
  comment Permutations are successively generated and
    output;
  Q := 0;  L := u − 1;
L1:
  Q := Q + 1;
  outinteger (1, Q);
  for I := u − 2 step −1 intil 0 do outinteger (1, A[L−I]);
  EXTENDED TWIDDLE (x, y, k, u, done, p);
  I := A[x];  A[x] := A[y];  A[y] := I;
  if ¬ done then go to L1
end of driver program
```