Algorithms

H. J. WEGSTEIN, Editor

ALGORITHM 112
POSITION OF POINT RELATIVE TO POLYGON

M. Shimrat
University of Alberta, Calgary, Alberta, Canada

```
Boolean procedure POINT IN POLYGON (n, x, y, x0, y0);
value n, x0, y0;  integer n;  array x, y;  real x0, y0;
comment  if the points (x[i], y[i]) (i = 1, 2, ··· , n) are—in
  this cyclic order—the vertices of a simple closed polygon and
  (x0, y0) is a point not on any side of the polygon, then the pro-
  cedure determines, by setting "point in polygon" to true,
  whether (x0, y0) lies in the interior of the polygon;
begin integer i;  Boolean b;
  x[n + 1] := x[1];  y[n + 1] := y[1];  b := true;
  for i := 1 step 1 until n do
    if (y < y[i] ≡ y > y[i + 1]) ∧
       x0 − x[i] − (y0 − y[i]) × (x[i + 1] − x[i])/(y[i + 1] − y[i]) <0
    then b := ¬ b;
  POINT IN POLYGON := ¬ b;
end POINT IN POLYGON
```

ALGORITHM 113
TREESORT

Robert W. Floyd
Computer Associates, Inc., Woburn, Mass.

```
procedure  TREESORT (UNSORTED, n, SORTED, k); value
  n, k;
integer n, k;  array UNSORTED, SORTED;
comment  TREESORT sorts the smallest k elements of the n-
  component array UNSORTED into the k-component array
  SORTED (the two arrays may be the same). The number of
  operations is on the order of 2 × n + k × log2(n). The number
  of auxiliary storage cells required is on the order of 2 × n. It is
  assumed that procedures are available for finding the minimum
  of two quantities, for packing one real number and one integer
  into a word, and for obtaining the left and right half of a packed
  word. The value of infinity is assumed to be larger than that of
  any element of UNSORTED;
begin integer i, j;  array m[1:2 × n − 1];
for i := 1 step 1 until n do m[n + i − 1] := pack (UNSORTED
  [i], n + i − 1);
for i := n − 1 step − 1 until 1 do m[i] := minimum (m[2 × i],
  m[2 × i + 1]);
for j := 1 step 1 until k do
  begin SORTED [j] := left half (m[1]);  i := right half (m[1]);
    m[i] := infinity;
  for i := i ÷ 2 while i > 0 do m[i] := minimum (m[2 × i], m[2 ×
    i + 1])
  end
end TREESORT
```

ALGORITHM 114
GENERATION OF PARTITIONS WITH CONSTRAINTS

Frank Stockmal
System Development Corp., Santa Monica, Calif.

```
procedure  CP GENERATOR (N, K, H, p, F, Z);  integer
  N, K, H;  integer array p;  Boolean F, Z;
comment  CP GENERATOR generates a partition of N into K
  parts, no part greater than H. Each partition is represented by
  the array of parts p[1] thru p[K], where p[1] ≧ p[2] ≧ ··· ≧ p[K].
  Initial entry is made with F = true and Z = true if parts = 0
  are allowable, or F = true and Z = false if only nonzero parts
  are desired. Upon initial entry, procedure ignores the input
  array p, sets F = false, and generates the initial parti-
  tion. Subsequent calls made with F = false will cause
  procedure to operate upon the input partition to produce
  another partition if one exists, so that all possible unpermuted
  partitions with the specified constraints will be produced if CP
  GENERATOR is allowed to operate upon its previous output.
  When this scheme is followed, and initial entry is made with
  F = true,  Z = true,  K = N,  H = N, all possible un-
  permuted partitions of N will be produced. Upon generating
  the last partition, procedure resets F to true. The input param-
  eters are restricted as follows: K ≧ 1,  H ≧ 1,  p[1] ≧ p[2]
  ≧ ··· ≧ p[K]. For Z = true,  N is restricted to the range
  0 ≦ N ≦ KH, and for Z = false,  K ≦ N ≦ KH. A call should
  not be made with p[1] − p[K] < 2 and F = false;
begin integer a, b, i, j, q, r;
  if F then go to first;
  a := p[1] − p[2] − 2;  j := 2;
test:  if p[1] − p[j] ≧ 2 then go to divide;
  a := a − 1 + j × (p[j] − p[j + 1]);  j := j + 1;  go to test;
first:  if Z then go to alpha;
  a := N − K;  p[K] := 0;  go to beta;
alpha:  a := N;  p[K] := −1;
beta:  F := false;  j := K;
divide:  b := H − 1 − p[j];  q := entier (a/b);  r := a − b × q;
  for i := 1 step 1 until q do p[i] := H;
  if q = K then go to last;
  for i := q + 1 step 1 until j do p[i] := 1 + p[j];
  p[q + 1] := r + p[q + 1];
  if p[1] − p[K] ≧ 2 then go to exit;
last:  F := true;
exit:  end CP GENERATOR
```

ALGORITHM 115
PERM

H. F. Trotter
Princeton University, Princeton, N. J.

```
procedure PERM (x, n);  value n;
integer n;  array x;
comment  This algorithm was inspired by the procedure
  PERMUTE of Peck and Schrack (Algorithm 86, Comm. ACM
```

Apr. 1962) and performs the same function. Each call of *PERM* changes the order of the first n components of x, and $n!$ successive calls will generate all $n!$ permutations. A nonlocal Boolean variable *'first'* is assumed, which must be **true** when *PERM* is first called, to cause proper initialization. The first call of *PERM* makes *'first'* **false**, and it remains so (unless changed by the external program) until the exit from the ($n!$)th call of *PERM*. At that time x is restored to its original order and *'first'* is made **true**.

The excuse for adding *PERM* to the growing pile of permutation generators is that, at the expense of some extra **own** storage, it cuts the manipulation of x to the theoretical minimum of $n!$ transpositions, and appears to offer an advantage in speed. It also has the (probably useless) property that the permutations it generates are alternately odd and even;

```
begin own integer array p, d[2: n];   integer k, q;   real t;
  if first then initialize:
  begin for k := 2 step 1 until n do
    begin p[k] := 0;  d[k] := 1 end;
    first := false
  end initialize;
  k := 0;
  INDEX:  p[n] := q := p[n] + d[n];
    if q = n then
    begin d[n] := −1;  go to LOOP end;
    if q ≠ 0 then go to TRANSPOSE;
    d[n] := 1;  k := k + 1;
    LOOP:  if n > 2 then begin
      comment  Note that n was called by value;
      n := n − 1;  go to INDEX end LOOP;
  Final exit:  q := 1;  first := true;
  TRANSPOSE:  q := q + k;  t := x[q];
    x[q] := x[q + 1];  x[q + 1] := t
end PERM;
```

ALGORITHM 116
COMPLEX DIVISION

ROBERT L. SMITH
Stanford University, Stanford, Calif.

```
procedure complexdiv (a, b, c, d) results:  (e, f);
value a, b, c, d;  real a, b, c, d;
comment  complexdiv yields the complex quotient of a + ib
  divided by c + id. The method used here tends to avoid arith-
  metic overflow or underflow. Such spills could otherwise occur
  when squaring the component parts of the denominator if the
  usual method were used;
begin  real  r, den;
  if abs (c) ≧ abs (d) then
  begin r := d/c;
    den := c + r × d;
    e := (a + b × r)/den;
    f := (b − a × r)/den;
  end
  else
  begin r := c/d;
    den := d + r × c;
    e := (a × r + b)/den;
    f := (b × r − a)/den;
  end
end complexdiv
```

ALGORITHM 117
MAGIC SQUARE (EVEN ORDER)

D. M. COLLISON
Elliott Brothers (London) Limited, Borehamwood, Herts., England

```
procedure magiceven (n, x);  value n;  integer array x;  in-
  teger n;
comment  the method of Devedec for even n is described in
  "Mathematical Recreations" by M. Kraitchik, pp. 150–2. Enter
  with side of square n to produce a magic square of the integers
  1 − n ↑ 2 in x, where n ≧ 4;
begin integer a, b, n2, nn;  Boolean p, q, r;
  n2 := n ÷ 2;  nn := n × n;
  begin
procedure alpha (p, q, a, h);  value p, q, a, h;  integer p, q, a;
  Boolean h;
Comment pattern 0/0/0/ ··· ;
  begin integer r;
    for r := p step 1 until q do begin
      x[r, a] := if h then (a × n − n + r) else (nn − a × n +
      1 + n − r);  h := ¬h end;
end alpha;
procedure beta (p, q, a, h);  value p, q, a, h;  integer p, q, a;
  Boolean h;
comment pattern 1 − 1 − 1 − ··· ;
  begin integer r;
    for r := p step 1 until q do begin
      x[r, a] := if h then [nn − a × n + r) else (a × n + 1 − r);
      h := ¬ h end;
end beta;
procedure gamma (p, q, a, h);  value p, q, a, h;  integer p, q, a;
  Boolean h;
comment pattern /−/−/− ··· ;
  begin integer r;
    for r := p step 1 until q do begin
      x[r, a] := if h then (nn − a × n + n − r + 1) else (a × n
        + 1 − r);  h := ¬ h end;
end gamma;
comment program begins;
p := q := (n − (n ÷ 4) × 4 = 0); r := true;
for a := 1 step 1 until (n2 − 1) do begin
  beta (1, a − 1, a, r);  alpha (a, n2 − 1, a, true);
  x[n2, a] := if q then (nn − a × n + n2 + 1) else (nn − a ×
    n + n2);
  alpha (n2 + 1, n, a, ¬ q);
  q := ¬ q;  r := ¬ r end;
alpha (1, n2 − 1, n2, ¬ p);  alpha (n2 + 2, n, n2, false);
gamma (1, n2 − 1, n2 + 1, p);  gamma (n2 + 2, n, n2 + 1, true);
q := p;  r := true;
for a := (n2 + 2) step 1 until n do begin
  beta (1, n − a, a, q);  x[n − a + 1, a] := a × n − a + 1;
  beta (n − a + 2, n2 − 1, a, true);
  if r then for b := n2, n2 + 1 do x[b, a] := nn − a × n +
    n − b + 1
    else begin x[n2, a] := nn − a × n + n2;
      x[n2 + 1, a] := a × n − n2 + 1 end;
  beta (n2 + 2, a − 1, a, ¬ r);  alpha (a, n, a, true);
  q := ¬ q;  r := ¬ r end;
for a := n2, n2 + 1 do for b := n2, n2 + 1 do
  x[b, a] := if p then (a × n − n + b) else (nn − a × n + n −
    b + 1);
if ¬ p then begin
  for a := n2, n2 + 1 do x[n2 − 1, a] := a × n − n2 + 2;
  for b := n2, n2 + 1 do x[b, n2 + 2] := n × n2 − 2 × n + b end;
end end magiceven
```

ALGORITHM 118
MAGIC SQUARE (ODD ORDER)

D. M. Collison
Elliott Brothers (London) Limited, Borehamwood, Herts., England

```
procedure magicodd (n, x); value n; integer n; integer
  array x;
comment for given side n the procedure generates a magic
  square of the integers 1 − n ↑ 2. For the method of De la
  Loubère, see M. Kraitchik, "Mathematical Recreations," p.
  149. n must be odd and n ≧ 3;
begin integer i, j, k;
  for i := 1 step 1 until n do
    for j := 1 step 1 until n do x[i, j] := 0;
  i := (n + 1) ÷ 2;  j := n;
  for k := 1 step 1 until n × n do begin
    if x[i, j] ≠ 0 then begin i := i − 1;  j := j − 2;
      if i < 1 then i := i + n;  if j < 1 then j := j + n end;
    x[i, j] := k;
    i := i + 1;  if i > n then i := i − n;
    j := j + 1;  if j > n then j := j − n;
    end;
end magicodd
```

ALGORITHM 119
EVALUATION OF A PERT NETWORK

Burton Eisenman and Martin Shapiro
United Nuclear Corp., White Plains, N. Y.

```
procedure pert (nmax, i, j, te, st, emax, l, es, at);
comment An algorithm describing an iterative procedure for
  evaluating a PERT network that permits the use of arbitrarily
  ordered activities and event identifiers such that an upper
  triangular matrix type of solution is unnecessary.
    It has been observed by investigations of PERT networks,
  that an N × N matrix whose rows are designated as predecessor
  and whose columns are designated as successor events, has an
  entry in the (i, j)-element representing the activity time re-
  quired in going from event i to event j. By elementary transfor-
  mations, the matrix is transformed generally into an upper
  triangular matrix. The resultant upper triangular matrix is well
  ordered (i.e. any activity time appearing in a column is not
  dependent upon those activity times which appear in columns
  to the right of it).
    This precise manipulation generally demands considerable
  running time. By direct evaluation not requiring a collection of
  elementary transformations, it is possible to evaluate the net-
  work with considerable reduction of running time;
integer  nmax, emax;
real  st;
integer array  i, j, l;
real array  te, es, at;
comment Given the total number of activities, nmax, the pre-
  ceding and succeeding event identifiers, i_n and j_n, the cor-
  responding expected time, te, for each activity, and the starting
  time, st, of the network, this procedure computes the early start
  and late finish times, es_e and at_e, for each event, l_e, in the net-
  work;
begin
procedure  scan (e, t, l);
integer  e, t;
integer array  l;
comment Given the number of events, e−1, contained thus far
  in vector array, l, and an event identifier i_n or j_n, stored in t,
  this procedure scans the existing array, l, to determine whether
  the event should be added to the list or not. If it is to be added,
  it becomes l_e and e replaces the event identifier. If it is not
  added, k replaces the event identifier.;
begin
integer  k;
         if e = 1 then go to add;
         for k := e−1 step −1 until 1 do
begin    if t = l[k] then
begin    t := k;
         go to out
end
end;
add:     l[e] := t;
         t := e;
         e := e + 1;
out:
end scan;
integer  n, e, s, t, k;
real     a, x;
         e := 1;
         for n := 1 step 1 until nmax do
begin    t := j[n];
         scan (e, t, l);
         j[n] := t;
         t := i[n];
         scan (e, t, l);
         i[n] := t
end;
comment By means of the switch, s, we will either compute the
  activity times, at_e, and transfer the values to the early start
  vector, es_e, or we will compute at_e without any transfer process,
  in which case the late finish times will be obtained.;
  emax := e − 1;
         s := 1;
         a := st;
s1:      k := emax;
         for e := 1 step 1 until emax do
         at[e] := a;
s2:      for n := 1 step 1 until nmax do
begin    if l[i[n]] > 0 then
begin    switch s := b1, b2;
b1:      x := abs (at[i[n]]) + te[n];
         if x > abs (at[j[n]]) then go to l1;
         go to l2;
b2:      x := abs (at[i[n]]) − te[n];
         if x < abs (at[j[n]]) then
l1:      at[j[n]] := − x;
l2:
end
end;
         for e := 1 step 1 until emax do
begin    if l[e] < 0 then
begin    if at[e] < 0 then
begin    l[e] := abs (l[e]);
         k := k + 1;
s3:      at[e] := abs (at[e]);
         go to l3
end;
         go to l3
end;
         if at[e] ≧ 0 then
begin    l[e] := − l[e];
         k := k − 1;
         go to l3
end;
         go to s3;
```

```
l3:
end;
         if k ≠ 0 then go to s2;
         switch s := g1, g2;
g1:      s := 2;
         for n := 1 step 1 until nmax do
begin    t := i[n];
         i[n] := j[n];
         j[n] := t
end;
         a := 0;
         for e := 1 step 1 until emax do
begin    es[e] := at[e];
         l[e] := abs (l[e]);
         if at[e] > a then
         a := at[e]
end;
         go to s1;
g2:      for e := 1 step 1 until emax do
         l[e] := abs (l[e]);
end pert
```

ALGORITHM 120

MATRIX INVERSION II

Richard George*

Particle Accelerator Division Argonne National Laboratory Argonne, Illinois

* Work supported by the U. S. Atomic Energy Commission.

```
procedure INVERSION II (n, a, epsilon, ALARM, delta);
comment  This is a revision of Algorithm 58. It accomplishes in-
   version of the matrix a, with the result stored in matrix a. The
   order of the matrix is n. If in the process of calculating, any
   pivot element has an absolute value less than epsilon, there
   will be a jump to the non-local label ALARM. The variable delta
   will contain the value of the determinant of the original matrix
   on normal exit, zero or a very small number on exit to ALARM.;
value  n;
array  a;
real  epsilon, delta;
integer  n;
begin
  array b, c[1:n];  real w, y;
  integer array z[1:n];  integer i, j, k, l, p;
  delta := 1.0;
  for j := 1 step 1 until n do
    z[j] := j;
  for i := 1 step 1 until n do
    begin
      k := i;  y := a[i, i];  l := i-1;  p := i+1;
      for j := p step 1 until n do
        begin
          w := a[i, j];
      if abs(w) > abs(y) then
        begin
          k := j;
          y := w
        end;
    end;
  delta := delta × y;
  if abs(y) < epsilon then go to ALARM;
  y := 1.0 / y;
  for j := 1 step 1 until n do
    begin
      c[j] := a[j, k];
      a[j, k] := a[j, i];
      a[j, i] := - c[j] × y;
      b[j] := a[i, j] := a[i, j] × y
    end;
  a[i, i] := y;
  j := z[i];
  z[i] := z[k];
  z[k] := j;
  for k := 1 step 1 until l, p step 1 until n do
    for j := 1 step 1 until l, p step 1 until n do
      a[k, j] := a[k, j] - b[j] × c[k]
        end;
  for i := 1 step 1 until n do
    begin
REPEAT:  k := z [i];
           if k=i then go to ADVANCE;
           for j := 1 step 1 until n do
             begin
               w := a [i, j];
               a [i, j] := a [k, j];
               a [k, j] := w
             end;
           p := z [i];
           z [i] := z [k];
           z [k] := p;
           delta := - delta;
           go to REPEAT;
ADVANCE:  end;
end
```

CERTIFICATION OF ALGORITHM 18

RATIONAL INTERPOLATION BY CONTINUED FRACTIONS

[R. W. Floyd, *Comm. ACM.*, Sept. 1960]

Henry C. Thacher, Jr.*

Reactor Engineering Div., Argonne National Lab., Argonne, Ill.

* Work supported by the U. S. Atomic Energy Commission

The body of procedure *confr* was tested with the Algol translator system written for the LGP-30 computer by the Dartmouth College Computer Center. No syntactical errors were found in the procedure body, except for a missing semicolon after the array delcaration. The translated algorithm gave satisfactory results when tested on values of $(4x + 1)/(x + 4)$ at any three of the points $x = 1, 2, 3, 4$. When all four points were used, a division overflow occurred in the statement **for** $i := 1$ **step** 1 **until** $j-1$ **do** $aa := (xx - x[i])/(aa - a[i])$; which forms the reciprocal differences. An overflow of this type will occur whenever $y[j]$ is approximated to high accuracy by one of the continued fractions based only on the points $x[i]$, $i = 1, 2, \cdots, k$ with k less than j. Unless $i = j-1$, the difficulty may be overcome by setting aa equal to the largest real representable in the computer whenever division overflow would occur. When $i = j-1$, the difficulty is irretrievable, and the data points must be reordered.

CERTIFICATION OF ALGORITHM 19

BINOMIAL COEFFICIENTS [Richard R. Kenyon, *Comm. ACM* Oct., 1960]

RICHARD GEORGE*
Particle Accelerator Div., Argonne National Lab., Argonne, Ill.

* Work supported by the U. S. Atomic Energy Commission.

This procedure was tested on the LGP-30, using the compiler ALGOL-30 from Dartmouth College Computation Center. The following changes were found necessary:
(1) Within the **comment**, the line

$$C^n_{i+1} = (n - 1)C_i{}^n/(i + 1)$$

should be

$$C^n_{i+1} = (n - i)C_i{}^n/(i + 1)$$

(2) The line defining the iteration loop

for i := 0 **step** 1 **until** b **do**

should be

for i := 0 **step** 1 **until** $b-1$ **do**

(3) The sequence

end $C := a$ **end**

should be

end; $C := a$ **end**

CERTIFICATION OF ALGORITHM 35

SIEVE [T. C. Wood, *Comm. ACM*. Mar. 1961]

J. S. HILLMORE
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The statement:

go to if $n/p[i] = n \div p[i]$ **then** $b1$ **else** $b2$;

was changed to the statement:

go to if $(n \div p[i]) \times p[i] = n$ **then** $b1$ **else** $b2$;

This avoids any inaccuracy that might result from introducing real arithmetic into the evaluation of the relation.

The modified algorithm was successfully run using the Elliott ALGOL translator on the National-Elliott 803.

CERTIFICATION OF ALGORITHM 37

TELESCOPE 1 [K. A. Brons, *Comm. ACM*, Mar., 1961]

HENRY C. THACHER, JR.*
Reactor Engineering Div., Argonne National Lab., Argonne, Ill.

* Work supported by the U. S. Atomic Energy Commission.

The body of *Telescope 1* was compiled and tested on the LGP-30 using the ALGOL 60 translator system developed by the Dartmouth College Computer Center. No syntactical errors were found, and the program ran satisfactorily. The 10th degree polynomial obtained by truncating the exponential series was telescoped using lim $= .1_{10} - 2$ and $L = 1.0$. The result was $N = 3$, $eps = .2103005_{10} - 3$, and coefficients $+.9997892$, $-.9930727$, $+.4636493$, $-.1026781$. The error curve for the telescoped polynomial was computed for $x = 0(.02)1.0$. The error extrema were bounded by eps to within 0.5%. The discrepancy is within the range of input conversion and round-off error.

CERTIFICATION OF ALGORITHM 52

A SET OF TEST MATRICES [J. R. Herndon, *Comm. ACM*, Apr. 1961]

J. S. HILLMORE
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The algorithm was corrected as recommended by H. E. Gilbert in his certification [*Comm. ACM*, Aug. 1961] and then successfully run using the Elliott ALGOL translator on the National-Elliott 803. The matrices so generated were used to test the matrix inversion procedure GJR given by H. R. Schwarz in his article "An Introduction to ALGOL" [Comm. ACM, Feb. 1962].

CERTIFICATION OF ALGORITHM 57

BER OR BEI FUNCTION [John R. Herndon, *Comm. ACM*, Apr. 1961]

HENRY C. THACHER, JR.*
Reactor Engineering Div., Argonne National Lab., Argonne, Ill.

* Work supported by the U. S. Atomic Energy Commission.

The body of Algorithm 57 was tested on the LGP-30 using the ALGOL 60 translator developed by the Dartmouth College Computer Center. No syntactical errors were found. For $z = 0.1(0.1)1.0$, with a 7+ significant decimal arithmetic routine, the program gave results with errors less than 5 (and for $z = 1(1)5$ less than 12) in the seventh digit. For large values of z, serious cancellation errors may occur. For example, for $z = 20$, more than 2 decimals of significance can be lost in this way.

REMARK ON ALGORITHM 58

MATRIX INVERSION [Donald Cohen, *Comm. ACM*, May, 1961]

GEORGE STRUBLE
University of Oregon, Eugene, Oregon

For the last seven lines, beginning with $a[k, j] := a[k, i]$, substitute:

```
a[k, j] := a[k, j] − b[j] × c[k] end;
l := 0;
back:  l := l+1;
again:  k := z[l];
if k ≠ l then
  begin for i := 1 step 1 until n do
    begin w := a[l, i];
          a[l, i] := a[k, i];
          a[k, i] := w end;
    z[l] := z[k];
    z[k] := k;
    go to again end;
  else if l ≠ n go to back
end invert
```

CERTIFICATION OF ALGORITHM 58

MATRIX INVERSION [Donald Cohen, *Comm. ACM*, May, 1961]

RICHARD GEORGE*
Particle Accelerator Div., Argonne National Lab., Argonne, Ill.

* Work supported by the U. S. Atomic Energy Commission.

This procedure was programmed in FORTRAN and reduced to machine code mechanically. It was run on the Argonne-built computing machine, GEORGE. A floating-point routine was used which allows maximum accuracy to 31 bits.

There are a number of errors of various types:

(1) There are eight **begin**'s and only seven **end**'s.

(2) The line

$$a[k, j] := a[k, i] - b[j] \times c[k] \quad \textbf{end};$$

should be

$$a[k, j] := a[k, j] - b[j] \times c[k] \quad \textbf{end};$$

(3) The permutation of rows of the inverted matrix and permutation of elements of the integer array z must be carried out simultaneously. This algorithm fails to do this, and consequently the matrix at the time of exit from the procedure is left in a permuted condition.

(4) The algorithm permits the statement

$$k := z[l];$$

to be executed even though the declarations place an upper limit of n on the integer array z, and the test for $l \leqq n$ has not yet been made. Obviously, Mr. Cohen's compiling system would allow an out-of-bounds array look-up. One could easily incorporate into an ALGOL compiler a guard against such illicit array references, and therefore the published algorithm might be considered machine dependent.

(5) This algorithm requires $3n^2$ divisions, most of which are unnecessary. By inserting the statement

$$y := 1.0/y;$$

at the proper place, one may accomplish the obvious economy of reducing this to only n divisions plus $2n^2$ multiplications.

(6) If a matrix should be singular (or nearly so), some pivot element will be zero (or very small), and a test should be made, with provision for a jump to *ALARM*, a non-local label.

(7) The identifiers w and y should be declared within this procedure, to avoid trouble.

(8) This algorithm omits calculation of the determinant of the matrix. This could be computed with very little extra effort.

The revised algorithm was then tested on the LGP-30 computer, using ALGOL-30, a small subset of ALGOL. Within the restrictions of this subset, the program worked satisfactorily on test matrices.

CERTIFICATION OF ALGORITHMS 63, 64, 65
PARTITION, QUICKSORT, FIND [C. A. R. Hoare, *Comm. ACM*, July 1961]

J. S. HILLMORE
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The body of the procedure find was corrected to read:

```
begin integer I, J;
if M < N then begin partition (A, M, N, I, J);
                    if K ≦ I then find (A, M, J, K)
                    else if J ≦ K then find (A, I, N, K)
                    end
end find
```

and the trio of procedures was then successfully run using the Elliott ALGOL translator on the National-Elliott 803.

The author's estimate of $\frac{1}{3}(N-M)\ln(N-M)$ for the number of exchanges required to sort a random set was found to be correct. However, the number of comparisons was generally less than $2(N-M)\ln(N-M)$ even without the modification mentioned below.

The efficiency of the procedure quicksort was increased by changing its body to read:

```
begin integer I, J;
if M < N−1 then begin partition (A, M, N, I, J);
                      quicksort (A, M, J);
                      quicksort (A, I, N)
                      end
else if N−M = 1 then begin if A[N] < A[M] then
                                  exchange (A[M], A[N])
                        end
end quicksort
```

This alteration reduced the number of comparisons involved in sorting a set of random numbers by 4–5 percent, and the number of entries to the procedure partition by 25–30 percent.

CERTIFICATION OF ALGORITHM 71
PERMUTATION [R. R. Coveyou and J. G. Sullivan, *Comm. ACM*, Nov. 1961]

J. S. HILLMORE
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The algorithm was successfully run using the Elliott ALGOL translator on the National-Elliott 803. The integer array x was made a parameter of the procedure in order to avoid having an **own** array with variable bounds.

CERTIFICATION OF ALGORITHM 72
COMPOSITION GENERATOR [L. Hellerman and S. Ogden, *Comm. ACM*, Nov. 1961]

D. M. COLLISON
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

After

```
for j := 1 step 1 until k do d[j] := c[j]−1;
```

the statement

```
j := k;
```

should be inserted (see ALGOL 60 report, para 4.6.5). With this alteration, the algorithm was successfully run using the Elliott ALGOL translator on the National-Elliott 803.

CERTIFICATION OF ALGORITHM 75
FACTORS [J. E. L. Peck, *Comm. ACM*, Jan. 1962]

J. S. HILLMORE
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The following changes had to be made to the algorithm:

```
(1) For   if q > 1 ∧ p = 1 then
    put   if q > 1 ∧ p = q then
(2) For   begin c := c × a0; a0 := 1 end
    put   begin c := c × a[0]; a[0] := 1 end
(3) For   if q = 0 ∨ (an ÷ q) × q = an then
    put   if (if q = 0 then true else (an ÷ q) × q = an) then
```

This change is necessary to ensure that the term $(an \div q)$ is not evaluated when $q = 0$.

The algorithm, thus modified, was successfully run using the Elliott ALGOL translator on the National-Elliott 803.

REMARK ON ALGORITHM 78
RATIONAL ROOTS OF POLYNOMIALS WITH INTEGER COEFFICIENTS [C. Perry, *Comm. ACM*, Feb. 1962]

D. M. Collison
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The algorithm was successfully run using the Elliott ALGOL translator on the National-Elliott 803. It was noticed that a multiple rational root will only be printed once by the procedure.

REMARK ON ALGORITHM 84
SIMPSON'S INTEGRATION [Paul E. Hennion. *Comm. ACM*, Apr. 1962]

Richard George*
Particle Accelerator Div., Argonne National Lab., Argonne, Ill.

* Work supported by the U. S. Atomic Energy Commission.

In performing integration by the use of Simpson's rule, it is well known that the interval $[a, b]$ must be divided evenly into n equal parts, and that *it is essential for n to be an even number*.

In the published algorithm, there is neither a comment on this important restriction, nor a programmed test for the parity of n. It is therefore a potential trap for the unwary programmer.

CERTIFICATION OF ALGORITHM 85
JACOBI [T. G. Evans, *Comm. ACM*, Apr. 1962]

J. S. Hillmore
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The statement

omega := (**if** *mu* = 0.0 **then** 1 **else** *sign* (*mu*)) × (−*V2*)/*sqrt*(*V2* ↑ 2+*mu* ↑ 2);

was changed to

omega := **if** *mu* = 0.0 **then** −1.0 **else** − *sign* (*mu*) × *V2*/*sqrt* (*V2* ↑ 2+*mu* ↑ 2);

When *mu* = 0, the original statement reduces to

omega := −*V2*/*sqrt* (*V2* ↑ 2);

and a truncation error in the evaluation of the square root can make the magnitude of omega slightly greater than unity. As a result, an error stop occurs during execution of the next statement when an attempt is made to evaluate *sqrt* (1 − *omega* ↑ 2).

In its modified form the algorithm has been successfully run using the Elliott ALGOL translator on the National-Elliott 803. Matrices of order up to fifteen have been solved, yielding eigenvalues and eigenvectors with an overall accuracy of seven decimal places.

CERTIFICATION OF ALGORITHM 86
PERMUTE [J. E. L. Peck and G. F. Schrock, *Comm. ACM*, Apr. 1962]

D. M. Collison
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The algorithm was successfully run using the Elliott ALGOL translator on the National-Elliott 803. Values of n used were 0, 1, 2, 3, 4.

CERTIFICATION OF ALGORITHM 87
PERMUTATION GENERATOR [John R. Howell, *Comm. ACM*, Apr. 1962]

D. M. Collison
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The array N was removed from the value list in order that the permutations might be available outside the procedure. The algorithm was then run successfully with the Elliott ALGOL translator on the National-Elliott 803. It was rather slower than Algorithm 86.

CERTIFICATION OF ALGORITHMS 117 AND 118
MAGIC SQUARE (ODD AND, EVEN ORDERS) [D. M. Collison, *Comm. ACM*, Aug. 1962]

D. M. Collison
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

Both algorithms were checked and timed, using a special ALGOL program, with the Elliott ALGOL translator on the National-Elliott 803. The procedure for odd orders was the slower:

Procedure	*Size of Square*	*Time*
Odd order	9	10 sec.
	19	45 sec.
Even order	10	7 sec.
	20	23 sec.

Because of the different methods used and the length of the even order procedure it was decided not to combine the two. The smallest square of even order generated is given below:—

13	3	2	16
8	10	11	5
12	6	7	9
1	15	14	4