PRACNIQUES

The Techniques Department is interested in publishing short descriptions of Techniques which improve the logistics of information processing. To quote from the policy statement, Communications of the ACM 1 (Jan. 1958), 5: "It is preferable that techniques contributed be factual and in successful usage, rather than speculative or theoretical. One of the major criteria for acceptance and the question one should answer before submitting any material is—Can the reader use this tomorrow?" Clear, concise statements of fairly well-known but rarely documented methods will contribute significantly to raising the general level of professional competence.—C.L.McC.

ON THE INVERSE OF A TEST MATRIX

In reference to the test matrix for determinants and inverses suggested by John Caffrey [*Comm. ACM 6*, 6 (June 1963)], it is possible to state the elements of the inverse explicitly. The test matrix was given as

$$A = a_{ij} = k\binom{i+j-2}{i-1}$$

where k is an arbitrary nonzero constant.

The inverse matrix is

$$V = v_{ij} = \frac{(-1)^{i+j}}{k} \sum_{g=\max(i,j)}^{n} \binom{g-1}{i-1}\binom{g-1}{j-1}$$

where n is the order of the matrix. The inverse elements can be computed recursively as follows:

$$v_{in} = v_{ni} = \frac{(-1)^{n+i}}{k}\binom{n-1}{i-1};$$

$$v_{ij} = v_{i+1,j} + v_{i,j+1} + \frac{(-1)^{i+j}}{k}\binom{n}{i}\binom{n}{j},$$

$$\text{for} \quad 1 \leqq i, \; j \leqq n-1.$$

Let $V^{(n)}$ denote the inverse of order n. Then for fixed k, $V^{(n+1)}$ can be generated from $V^{(n)}$:

$$v_{11}^{(n+1)} = \frac{n+1}{k};$$

$$v_{1,n+1}^{(n+1)} = v_{n+1,1}^{(n+1)} = \frac{(-1)^n}{k};$$

$$v_{n+1,n+1}^{(n+1)} = \frac{1}{k};$$

$$v_{i1}^{(n+1)} = v_{1i}^{(n+1)} = v_{i1}^{(n)} - v_{i-1,1}^{(n)} \qquad \text{for} \quad 2 \leqq i \leqq n;$$

$$v_{i,n+1}^{(n+1)} = v_{n+1,i}^{(n+1)} = v_{i-1,n}^{(n)} - v_{i,n}^{(n)}, \qquad \text{for} \quad 2 \leqq i \leqq n;$$

$$v_{ij}^{(n+1)} = v_{ij}^{(n)} + v_{i-1,j-1}^{(n)} - v_{i,j-1}^{(n)} - v_{i-1,j}^{(n)},$$

$$\text{for} \quad 2 \leqq i, \; j \leqq n.$$

FRANK J. STOCKMAL
System Development Corporation
Santa Monica, California

Algorithms

J. H. WEGSTEIN, Editor

ALGORITHM 207
STRINGSORT

J. BOOTHROYD
English Electric-Leo Computers, Ltd.
Staffordshire, England

```
procedure stringsort (a, n); comment elements a[1] ··· a[n]
  of a[1:2n] are sorted into ascending sequence using a[n+1] ···
  a[2n] as auxiliary storage. Von Neumann extended string logic
  is employed to merge input strings from both ends of a sending
  area into output strings which are sent alternately to either
  end of a receiving area. The procedure takes advantage of
  naturally occurring ascending or descending order in the origi-
  nal data;
value n; integer n; array a;
begin integer d, i, j, m, u, v, z; integer array c[-1:1];
  switch p := jz1, str i; switch q := merge, jz2;
oddpass: i := 1; j := n; c[-1] := n + 1; c[1] := 2 × n;
allpass: d := 1; go to firststring;
merge: if a[i] ≧ a[z]
         then begin go to p[v];
           jz1: if a[j] ≧ a[z]
                  then ij: begin if a[i] ≧ a[j]
                                   then str j: begin a[m] := a[j];
                                     j := j - 1 end
                                   else str i: begin a[m] := a[i];
                                     i := i + 1 end
                           end
                  else begin v := 2; go to str i end
         end
         else begin u := 2;
              jz2: if a[j] ≧ a[z]
                     then go to str j
                     else begin d := -d; c[d] := m;
                            firststring: m := c[-d];
                              v := u := 1;
                            go to ij
                          end
              end;
z := m; m := m + d; if j ≧ i then go to q[u];
if m > n + 1 then begin comment evenpass; i := n + 1;
                    j := 2 × n; c[-1] := 1; c[1] := n; go to
                    allpass end
                  else if m < n + 1 then go to oddpass
end stringsort;
```

ALGORITHM 208
DISCRETE CONVOLUTION

WILLIAM T. FOREMAN, JR.
Collins Radio Co.
Newport Beach, Calif.

```
procedure Discrete Convolution (m, n, prs) result: (Conv);
integer m, n; real procedure prs; real array conv;
comment This procedure finds the probability distribution of
  the sum of m independent variables, each with a known distribu-
```

tion over the nonnegative integers. A real procedure *prs* with results *pr*[*k*] is assumed to find each probability distribution in succession. The maximum sum for which probabilities are computed must be fixed by the user. The number of iterations is roughly $m^2n/2$. The procedure *prs* will in general depend on additional parameters and should include the read-in of the parameters for that distribution. It may include the selection of one function from a set;

```
begin integer i, j, k, ix1, ix2;
real array prob [1:2, 0:m], pr[0:m];
i := 1;  ix1 := 1;  ix2 := 2;  prs (m) result:  (pr);
for j := 0 step 1 until m do prob[ix1, j] := pr[j];
for i := 2 step 1 until n do
  begin
    if ix1 = 1 then begin ix2 := 1;  ix1 := 2 end
      else begin ix2 := 2;  ix1 := 1 end
    prs (m) result:  (pr);
    for j := 0 step 1 until m do
    begin
      prob[ix1, j] := 0;
      for k := 0 step 1 until j do
        prob[ix1, j] := prob[ix1, j] + pr[k] × prob[ix2, j−k]
    end j
  end i;
for j := 0 step 1 until m do conv[j] := prob[ix1, j]
end Discrete Convolution
```

comment The convolution of discrete probability series is isomorphic to the multiplication of polynomials. A useful variation is to omit the parameters *i*, *n* and have *prs* recognize the end of input. A FORTRAN program using this procedure has been run on the IBM 7090 to find the sum of queue lengths in a teletype switching center, where messages arrived according to the Poisson distribution and message lengths were distributed negative-exponentially. The following was used as the probability procedure;

```
procedure prs (m) result:  (pr);
value m;  procedure read;
real array pr;  integer m;
begin real trafficrate, linespeed, rho;  integer j;
  read (trafficrate, linespeed);
  rho := trafficrate/linespeed;
  pr[0] : 1 − rho;
  for j := 1 step until m do pr[j] := rho × pr[j−1]
end prs
```


ALGORITHM 209
GAUSS

D. IBBETSON,
Elliott Brothers (London) Ltd.,
Elstree Way, Borehamwood, Herts., England

real procedure *Gauss*(*x*); **value** *x*; **real** *x*;
comment Gauss calculates $(1/\sqrt{2\pi})\int_{-\infty}^{x} exp(-\frac{1}{2}u^2)\,du$ by means of polynomial approximations due to A. M. Murray of Aberdeen University;

```
begin real y, z, w;
  if x = 0 then z := 0
  else
  begin y := abs(x)/2;
    if y ≧ 3 then z := 1
    else if y < 1 then
    begin w := y × y;
      z := (((((((((0.000124818987 × w
        −0.001075204047) × w +0.005198775019) × w
        −0.019198292004) × w +0.059054035642) × w
        −0.151968751364) × w +0.319152932694) × w
        −0.531923007300) × w +0.797884560593) × y × 2
    end
    else
    begin y := y − 2;
      z := (((((((((((((−0.000045255659 × y
        +0.000152529290) × y −0.000019538132) × y
        −0.000676904986) × y +0.001390604284) × y
        −0.000794620820) × y −0.002034254874) × y
        +0.006549791214) × y −0.010557625006) × y
        +0.011630447319) × y −0.009279453341) × y
        +0.005353579108) × y −0.002141268741) × y
        +0.000535310849) × y +0.999936657524
    end
  end;
  Gauss := if x > 0 then (z+1)/2 else (1−z)/2
end Gauss;
```

ALGORITHM 210
LAGRANGIAN INTERPOLATION

GEORGE R. SCHUBERT*
University of Dayton, Dayton, Ohio

* Undergraduate research project, Computer Science Program, Univ. of Dayton.

procedure *LAGRANGE* (*N*, *u*, *X*, *Y*, *ANS*); **real array** *X*, *Y*; **integer** *N*; **real** *u*, *ANS*;
comment This procedure evaluates an *N*th degree Lagrange polynomial, given *N* + 1 data coordinates, and *u* the value where interpolation is desired. *X* is the abscissa array and *Y* the ordinate array. *ANS* is the resultant value of the function at *u*. The notation is that used in R. W. Hamming, *Numerical Methods for Scientists and Engineers*, pp. 94–95 (McGraw-Hill Book Company, Inc., 1962);

```
begin integer i, j;  real L;
  ANS := 0.0;
  for j := step 1 until N+1 do
begin L := 1.0;
  for i := step 1 until N+1 do
begin if i ≠ j then L := L × (u−X[i])/(X[j]−X[i])
end;
  ANS := ANS + L × Y[j]
end end
```

ALGORITHM 211
HERMITE INTERPOLATION

George R. Schubert*
University of Dayton, Dayton, Ohio

* Undergraduate research project, Computer Science Program, Univ. of Dayton.

```
procedure HERMITE (n, u, X, Y, Y1, ANS);  real array X, Y,
    Y1;
  integer n;  real u, ANS;
comment  This procedure evaluates a(2n+1)th degree Hermite
  polynomial, given the value of the function and its first deriva-
  tive at each of n + 1 points. X is the abscissa array, Y the
  ordinate array, and Y1 the derivative array. ANS is the interpo-
  lated value of the function at u. REFERENCE:  R. W. Hamming,
  Numerical Methods for Scientists and Engineers, pp. 96-97 (Mc-
  Graw-Hill Book Company, Inc., 1962);
begin integer i, j;  real h, a;
  ANS := 0.0;
  for j := 1 step 1 until n + 1 do
begin h := 1.0;  a := 0.0;
  for i := 1 step 1 until n + 1 do
begin if i = j then go to out;
  h := h × (u−X[i])↑2/(X[j]−X[i])↑2;
  a := a + 1.0/(X[j]−X[i]);
out: end;
  ANS := ANS + h × ((X[j]−u)×(2×a×Y[j]−Y1[j])+Y[j])
end end
```

ALGORITHM 212
FREQUENCY DISTRIBUTION

Malcolm D. Gray
The Boeing Co., Seattle, Wash.

```
procedure FREQUENCY (N, A, B, IUL, K, X, KA);
integer N, IUL;  integer array KA;  real A, B, K;
real array X;
comment  Given a set X of variables in some interval I = [a, b]
  such that a ≦ min x, max x ≦ b, FREQUENCY determines the
  frequency distribution of X over k equal, half open subintervals
  of I. The interval I is transformed to the interval J = [0, k]
  with unit subintervals by x' = (x_i−a)/[(b−a)/k],  i = 1, 2, ···,
  n, and considering x' = L × M,  L and M integers. The value
  L then immediately determines the subinterval and M is used
  for boundary points. If IUL = 0, the subintervals are open
  on the upper end, except the kth. On entry, the array KA is
  assumed identically zero; on return, KA[i] contains the fre-
  quency of X in the ith subinterval;
begin integer i, L;  real BAK, XP;
  BAK := (B−A)/K;
  for i := 1 step 1 until N do begin
    XP := (X[i]−A)/BAK;
    L := entier (XP);
    if XP = L then go to p2 else L := L + 1;  go to p5;
p2: if IUL = 0 then go to p3 else if L = 0 then L := L + 1;
      go to p5;
p3: if XP ≠ K then L := L + 1;
p5: KA[L] := KA[L] + 1;
  end;
end FREQUENCY
```

ALGORITHM 213
FRESNEL INTEGRALS

Malcolm D. Gray
The Boeing Co., Seattle, Wash.

real procedure *FRESNEL* (w, S, C); **value** w; **real** S, C;
comment *FRESNEL* computes the Fresnel sine and cosine integrals $S(w) = \int_0^w \sin[(\pi/2)t^2]\,dt$ and $C(w) = \int_0^w \cos[(\pi/2)t^2]\,dt$ using the series expansions

$$S(w) = w \sum_{i=1}^{\infty} \frac{(-1)^{i+1}x^{2i-1}}{(4i-1)(2i-1)!} \quad \text{and}$$

$$C(w) = w \sum_{i=1}^{\infty} \frac{(-1)^{i+1}x^{2i-2}}{(4i-3)(2i-2)!}$$

for $|w| < \sqrt{22/\pi}$ and $x \equiv \pi w^2/2$, and using the asymptotic series

$$S(w) = \alpha - \frac{1}{\pi w}[P(x)\sin(x) + Q(x)\cos(x)],$$

$$C(w) = \alpha - \frac{1}{\pi w}[P(x)\cos(x) - Q(x)\sin(x)]$$

where $|w| \geqq \sqrt{22/\pi}$, $x \equiv \pi w^2/2$,

$$Q(x) = 1 - \sum_{i=2}^{\infty} \frac{(-1)^i(4i-5)!!}{(2x)^{2i-2}}, \quad P(x) = \sum_{i=1}^{\infty} \frac{(-i)^{i+1}(4i-3)!!}{(2x)^{2i-1}},$$

and $n!! \equiv n(n-2)(n-4)\cdots 1$. If $w \geqq 0$, then $\alpha = \frac{1}{2}$, or if $w < 0$, then $\alpha = -\frac{1}{2}$.

This algorithm is a translation of a FAP coded subroutine currently in use on the IBM 7094 at the Boeing Company. The FAP program yields the following errors when tested at 0.05 increments of x:

x	ΔS	ΔC
0.00, 1.00	$<1 \times 10^{-7}$	$<1 \times 10^{-7}$
1.05, 8.65	$<1 \times 10^{-6}$	$<1 \times 10^{-6}$
8.70, 10.30	3×10^{-6}	2×10^{-6}
10.35, 11.00	5×10^{-6}	4×10^{-6}
11.05, 12.15	$<1 \times 10^{-6}$	3×10^{-6}
12.20, 15.00	$<1 \times 10^{-6}$	$<1 \times 10^{-6}$

where ΔS and ΔC are the approximate average absolute deviations (over the range) from the reference. The user must supply $S(w) = C(w) = \pm\frac{1}{2}$ if $w \to \pm\infty$. References: ALGORITHMS 88–90, J. L. Cundiff, *Comm. ACM*, May 1962; Born, M. and Wolf, E., *Principles of Optics*, Pergamon Press (1958), pp. 369–431;

```
begin real x, x2, eps, term;  integer n;  eps := 0.000001;
  x := w × w/0.6366198;
  x2 := −x × x;  if x ≧ 11.0 then go to asympt;
  begin real frs, frsi;
        frs := x/3;  n := 5;  term := x × x2/6;
        frsi := frs + term/7;
loops:  if abs(frs−frsi) ≦ eps then go to send;  frs := frsi;
        term := term × x2/n/(n−1);  frsi := frs + term/(n+n+1);
        n := n + 2;  go to loops;
send:   S := frsi × w;  end;
  begin real frc, frci;
        frc := 1;  n := 4;  term := x2/2;  frci := 1 + term/5;
loopc:  if abs(frc−frci) ≦ eps then go to cend;  frc := frci;
        term := term × x2/n/(n−1);  frci := frc + term/(n+n+1);
        n := n + 2;  go to loopc;
cend:   C := frci × w;  end;  go to aend;
asympt:  begin real S1, S2, half, temp;  integer i;
        x2 := 4 × x2;  term := 3/x2;  S1 := 1 + term;  n := 8;
        for i := 1 step 1 until 5 do begin n := n + 4;
          term := term × (n−7) × (n−5)/x2;  S1 := S1 + term;
          if abs(term) ≦ eps then go to next;  end i;
next:   for i := 1 step 1 until 5 do begin n := n + 4;
          term := term × (n−5) × (n−3)/x2;  S2 := S2 + term;
          if abs(term) ≦ eps then go to final;  end i;
final:  if w < 0 then half := −0.5 else half := 0.5;  term := cos(x);
  temp := sin(x);  x2 := 3.1415927 × w;
  C := half + (temp×S1−term×S2)/x2;
  S := half − (term×S1+temp×S2)/x2;
  end;
aend:  end FRESNEL
```

CERTIFICATION OF ALGORITHM 27
ASSIGNMENT [Roland Silver, *Comm. ACM*, Nov. 1960]

Albert Newhouse
University of Houston, Houston, Texas

The ASSIGNMENT algorithm was translated into MAD and successfully run on the IBM 709/7094 after the following corrections were made:

All references to array a and d refer to the same array, i.e. change all $a[i, j]$ to $d[i, j]$. Furthermore:

(a) 3rd line after *LABEL*: **comment**: Label and scan; should read
  begin if $d[i, j] \neq 0 \vee$ lambda $[j] \neq 0$ **then go**

(b) first line after $J3$: **end** j; should read
  $min := d[r[1], cb[1]]$;

(c) line $I2$: should read
  $I2$: **for** $l := 1$ **step** 1 **until** cbl **do**

Since there is no provision made for this algorithm to end the following additions were made:

(1) in the integer declaration add the variable: *flag*

(2) first line after *START*: **comment**: $\cdots$
  add the line
  $flag := n$;

(3) first line before $I1$: **end** i;
  change to read
  $rl := rl + 1$; $r[rl] := i$; $mu[i] := -1$; $flag := flag - 1$

(4) add a line after $I1$: **end** i;
  if $flag = n$ **then go to** *FINI*;

(5) change the last line of the algorithm to read:
  FINI: **end** Assignment

In order to obtain the minimum value of the $\sum_{i=1}^{n} a_{ix_i}$ (in the following called total) the following additions may be made:

Add a real variable *total* and

(A) new line after *INITIALIZE*;
  $total := 0$;

(B) new line after the first **end** i;
  $total := total + min$;

(C) new line after the first **end** j;
  $total := total + min$;

(D) after the line **end** k; after $J3$: **end** j;
  add the line
  $total := total + (rl + cbl - n) \times min$;

REMARK ON ALGORITHMS 88, 89 AND 90
EVALUATION OF THE FRESNEL INTEGRALS [J. L. Cundiff, *Comm. ACM*, May 1962]

Malcolm D. Gray
The Boeing Co., Seattle, Wash.

While coding these algorithms in Fortran for the IBM 7094, modifications were required (both in the formulation and in the language) before execution with any degree of speed and accuracy could be obtained. In the process it was found that the reference, *Pearcy*, contains an error in the formula for $C(u)$. This error is contained in Algorithm 88 in the formula

$$C(u) = \frac{1}{2} - \frac{\sin(x)}{\sqrt{2\pi x}} [\quad] - \cdots .$$

The first minus sign above should be a plus sign.

After the necessary modifications were made, the three algorithms were found to be too large and uneconomical for our usage. A single algorithm, incorporating these three procedures, was written and is in current usage in a computer program which requires several thousand evaluations of each Fresnel integral.

REMARK ON ALGORITHM 123
ERF(x) [Martin Crawford and Robert Techo, *Comm. ACM*, Sept. 1962]

D. Ibbetson
Elliott Brothers (London) Ltd.
Elstree Way, Borehamwood, Herts., England

(1) The specification **value** x; was added to allow x to be an expression and to prevent side effects.

(2) The algorithm was then modified to give the Gaussian integral $(1/\sqrt{2\pi})\int_{-\infty}^{x} \exp(-\frac{1}{2}u^2)\, du$ by

(a) changing its name to *Gauss* (x),

(b) inserting $x := x*0.70710678$; immediately before $Z := 0$; , and

(c) changing the final statement to
  $Gauss := (Z+1)/2$ **end** *Gauss*

(3) The algorithm with the above changes was tested on a National Elliott 803 computer using the Elliott-Algol translator with ${}_{10}{-8}$ substituted for ${}_{10}{-10}$. It was found to produce wrong answers when $x = \pm 1$ (corresponding to $Erf(\pm 1/\sqrt{2})$) giving 0.5 ± 0.3467899 instead of 0.5 ± 0.3413447.

REMARK ON ALGORITHM 157
FOURIER SERIES APPROXIMATION [Charles J. Mifsud, *Comm. ACM*, Mar. 1963]

George R. Schubert*
University of Dayton, Dayton, Ohio

* Undergraduate research project, Computer Science Program, Univ. of Dayton.

Algorithm 157 has been modified to fit $2N$ data points and has run successfully on the Burroughs 220 using Balgol. With the modifications, $2N$ constants a_p ($p=0, 1, \cdots, N$) and b_p ($p=1, 2, \cdots, N-1$) are determined such that the equation $f_n = a_0/2 + \sum_{p=1}^{N-1} (a_p \cos \pi np/N + b_p \sin \pi np/N) + a_N/2 \cos \pi n$ is satisfied.

In the modified procedure, the second and third lines after the integer declaration should read:

$C[1] := \cos(pi/N)$;
$S[1] := \sin(pi/N)$;

The second **for** statement should read:

for $i := 2 \times N-1$ **step** -1 **until** 1 **do**

The lines containing the a and b coefficients should read:

$a[p] := (f[0]+u[1]\times C[2]-u[2])/N$;
$b[p] := (u[1]\times S[2])/N$;

Reference: R. W. Hamming, *Numerical Methods for Scientists and Engineers*, pp. 68–73 (McGraw-Hill, 1962).

CERTIFICATION OF ALGORITHM 160
COMBINATORIAL OF M THINGS TAKEN N AT A TIME [M. L. Wolfson and H. V. Wright, *Comm. ACM*, April 1963]

Robert F. Blakely
Indiana Geological Survey, Bloomington, Ind.

Algorithm 160 was translated into Algo, a compiler for the Control Data Corp. G-15 computer (formerly the Bendix G-15).

With the restriction that $m \geqq n \geqq 0$, correct results were obtained for all integer values of m and n, where $0 \leqq m \leqq 10$. Several other values were tested and all results were correct.

CERTIFICATION OF ALGORITHM 161
COMBINATORIAL OF M THINGS TAKEN ONE AT A TIME, TWO AT A TIME, UP TO N AT A TIME [H. V. Wright and M. L. Wolfson, *Comm. ACM*, Apr. 1963]

DAVID H. COLLINS
Indiana Geological Survey, Bloomington, Ind.

Algorithm 161 was translated into ALGO, a compiler for the Control Data Corp. G-15 computer (formerly the Bendix G-15).

With the restriction that $m \geqq n \geqq 1$, correct results were obtained for all integer values of m and n, where $1 \leqq m = n \leqq 15$. Several other values were tested (including cases where $m \neq n$) and all results were correct.

CERTIFICATION OF ALGORITHM 173
ASSIGN [Otomar Hájek, *Comm. ACM*, June 1963]

R. S. SCOWEN
English Electric Co. Ltd., Whetstone, Leicester, England

Algorithm 173 (ASSIGN) has been tested successfully using the DEUCE ALGOL 60 compiler. The only changes necessary were the addition of specifications for the formal parameters a, b (DEUCE ALGOL 60 compiler requires specifications for all formal parameters).

The author's example, *assign* ($a[i[1], i[2]]$, (**if** $i[3]=1$ **then** 0.0 **else** $a[i[1], i[2]]$) + $b[i[1], i[3]] \times c[i[3], i[2]]$, 3, $i[j]$, 1, **if** $j = 1$ **then** n **else if** $j = 2$ **then** m **else** p, j);
did form the matrix product $B \times C$ and store it in A.

The algorithm was also used to read a matrix into the computer using the procedure call

```
assign (b[i[1], i[2]], read, 2, i[j], 1,
if j = 1 then n else p, j);
```

(*read* is a real procedure which takes the value given by the next number on the input tape).

These examples took about three times as long to run as the simpler equivalent statements

```
for i := 1 step 1 until n do
for j := 1 step 1 until m do
begin
  a[i, j] := 0.0;
  for k := 1 step 1 until p do
  a[i, j] := a[i, j] + b[i, k] × c[k, j]
end;
```

and

```
for j := 1 step 1 until p do
for i := 1 step 1 until n do
b[i, j] := read;
```

CERTIFICATION OF ALGORITHM 173
ASSIGN [O. Hájek, *Comm. ACM*, July 1963]

Z. FILSAK and L. VRCHOVECKÁ
Research Institute of Mathematical Machines, Prague, and Computing Center Kancelářské stroje, Prague

The algorithm was modified for input to the Elliott-ALGOL system as follows. In Elliott-ALGOL, name-called parameters in recursive procedures are prescribed. Luckily, the only parameter which varies during the recursive call in the body of *Assign* is called by value (it is the parameter *dim* which determines depth of recursion). The body of *Assign* was replaced by (i) a procedure declaration *Ass*(*dim*), whose body is that of the original *Assign*, but with the recursive call of *Assign* replaced by that of *Ass*, and (ii) a single statement, the activation of *Ass*(*dim*).

The resulting procedure was tested (on the National-Elliott 803 in the Computing Center), on a rather large set of examples, including those described in the text following Algorithm 173. It was found that in the last example, matrix multiplication, indices i_1 and i_3 should be interchanged throughout.

No changes of the algorithm itself were necessary. It seems that the modification described above, motivated by limitations of Elliott-ALGOL, also improve efficiency, at least for large dimensions of the arrays concerned.

CERTIFICATION OF ALGORITHM 175
SHUTTLE SORT [C. J. Shaw and T. N. Trimble, *Comm. ACM*, June 1963]

GEORGE R. SCHUBERT*
University of Dayton, Dayton, Ohio

* Undergraduate research project, Computer Science Program, Univ. of Dayton.

Algorithm 175 was translated into BALGOL and ran successfully on the Burroughs 220. The following actual sorting times were observed:

Number of Items	*Average Time* (sec)
25	1.6
50	6.2
100	25.8
250	181
500	684

The algorithm can be extended so that the sort is made on one array, while retaining a one-to-one correspondence to a second array. This is done by inserting immediately before **end** of the j loop the following:

Temporary := $Y[j]$; $Y[j] := Y[j+1]$; $Y[j+1]$:= *Temporary*; where $Y[k]$ is the element to be associated with $N[k]$. Other variations are obviously possible.

CERTIFICATION OF ALGORITHM 210
HERMITE INTERPOLATION [George R. Schubert, *Comm. ACM*, Oct. 1963]

THOMAS A. DWYER
Argonne National Laboratory, Argonne, Ill.

The body of *HERMITE* was transcribed for the Dartmouth SCALP processor for the LGP-30 computer and ran successfully without corrections. It was tested using the error function and its derivatives. Roundoff error in the LGP-30 began to appear for values of n greater than 3. For n equal to 2 (third degree polynomial) the interpolated value agreed with the function within machine limitations (six significant figures) for steps in the argument data of 0.005.