Algorithms

G. E. FORSYTHE, J. G. HERRIOT, Editors

ALGORITHM 245
TREESORT 3 [M1]

ROBERT W. FLOYD (Recd. 22 June 1964 and 17 Aug. 1964)
Computer Associates, Inc., Wakefield, Mass.

```
procedure TREESORT 3 (M, n);
  value n; array M; integer n;
comment TREESORT 3 is a major revision of TREESORT
  [R. W. Floyd, Alg. 113, Comm. ACM 5 (Aug. 1962), 434] sug-
  gested by HEAPSORT [J. W. J. Williams, Alg. 232, Comm.
  ACM 7 (June 1964), 347] from which it differs in being an in-place
  sort. It is shorter and probably faster, requiring fewer compari-
  sons and only one division. It sorts the array M[1:n], requiring
  no more than 2 × (2↑p−2) × (p−1), or approximately 2 ×
  n × (log2(n)−1) comparisons and half as many exchanges in
  the worst case to sort n = 2↑p − 1 items. The algorithm is
  most easily followed if M is thought of as a tree, with M[j÷2]
  the father of M[j] for 1 < j ≦ n;
begin
  procedure exchange (x,y); real x,y;
    begin real t; t := x; x := y; y := t
    end exchange;
  procedure siftup (i,n); value i, n; integer i, n;
  comment M[i] is moved upward in the subtree of M[1:n] of
    which it is the root;
  begin real copy; integer j;
    copy := M[i];
  loop: j := 2 × i;
    if j ≦ n then
    begin if j < n then
        begin if M[j+1] > M[j] then j := j + 1 end;
      if M[j] > copy then
        begin M[i] := M[j]; i := j; go to loop end
    end;
    M[i] := copy
  end siftup;
  integer i;
  for i := n÷2 step −1 until 2 do siftup (i,n);
  for i := n step −1 until 2 do
  begin siftup (1,i);
    comment M[j÷2] ≧ M[j] for 1 < j ≦ i;
    exchange (M[1], M[i]);
    comment M[i:n] is fully sorted;
  end
end TREESORT 3
```

ALGORITHM 246
GRAYCODE [Z]

J. BOOTHROYD* (Recd. 18 Nov. 1963)
English Electric-Leo Computers, Kidsgrove, Stoke-on-Trent, England
* Now at University of Tasmania, Hobart, Tasmania, Aust.

```
procedure graycode (a) dimension: (n) parity: (s); value n,s;
  Boolean array a; integer n; Boolean s;
comment elements of the Boolean array a[1:n] may together be
  considered as representing a logical vector value in the Gray
  cyclic binary-code. [See e.g. Phister, M., Jr., Logical Design of
  Digital Computers, Wiley, New York, 1958. pp. 232, 399.] This
  procedure changes one element of the array to form the next
  code value in ascending sequence if the parity parameter s
  = true or in descending sequence if s = false. The procedure
  may also be applied to the classic "rings-o-seven" puzzle [see
  K. E. Iverson, A Programming Language, p. 63, Ex. 1.5];
begin integer i,j; j := n + 1;
  for i := n step −1 until 1 do if a[i] then begin s := ¬ s;
    j := i end;
  if s then a[1] := ¬ a[1] else if j < n then a[j+1] := ¬ a[j+1]
    else a[n] := ¬ a[n]
end graycode
```

ALGORITHM 247
RADICAL-INVERSE QUASI-RANDOM POINT SEQUENCE [G5]

J. H. HALTON AND G. B. SMITH (Recd. 24 Jan. 1964 and 21 July 1964)
Brookhaven National Laboratory, Upton, N. Y., and University of Colorado, Boulder, Colo.

```
procedure QRPSH (K, N, P, Q, R, E);
integer K, N; real array P, Q; integer array R; real E;
comment This procedure computes a sequence of N quasi-
  random points lying in the K-dimensional unit hypercube
  given by 0 < xi < 1, i = 1, 2, ··· , K. The ith component of
  the mth point is stored in Q[m,i]. The sequence is initiated by a
  "zero-th point" stored in P, and each component sequence is
  iteratively generated with parameter R[i]. E is a positive error-
  parameter. K, N, E, and the P[i] and R[i] for i = 1, 2, ··· , K,
  are to be given.
```

The sequence is discussed by J. H. Halton in *Num. Math. 2* (1960), 84–90. If any integer n is written in radix-R notation as

$$n = n_m \cdots n_2 n_1 n_0 \,.\, 0 = n_0 + n_1 R + n_2 R^2 + \cdots + n_m R^m,$$

and reflected in the radical point, we obtain the R-inverse function of n, lying between 0 and 1,

$$\phi_R(n) = 0 \,.\, n_0 n_1 n_2 \cdots n_m = n_0 R^{-1} + n_1 R^{-2} + n_2 R^{-3} + \cdots + n_m R^{-m-1}.$$

The problem solved by this algorithm is that of giving a compact procedure for the addition of R^{-1}, in any radix R, to a fraction, with downward "carry".

If $P[i] = \phi_{R[i]}(s)$, as will almost always be the case in practice, with s a known integer, then $Q[m,i] = \phi_{R[i]}(s+m)$. For quasi-randomness (uniform limiting density), the integers $R[i]$ must be mutually prime.

For exact numbers, E would be infinitesimal positive. In practice, round-off errors would then cause the "carry" to be incorrectly placed, in two circumstances. Suppose that the stored number representing $\phi_R(n)$ is actually $\phi_R(n) + \Delta$. (a) If $|\Delta| \geqq R^{-m-1}$, we see that the results of the algorithm become un-

predictable. It is necessary to stop before this event occurs. It may be delayed by working in multiple-length arithmetic. (b) If $n = R^{m+1} - 1$, so that $\phi_R(n) = 1 - R^{-m-1}$, and $\Delta < 0$, the computed successor of the stored value can be seen to be about R^{-m}, instead of $R^{-m-2} = \phi_R(n+1)$. This error can be avoided, without disturbing the rest of the computation, by adopting a value of E greater than any $|\Delta|$ which may occur, but smaller than the least $(nR)^{-1}$ (which is smaller than the least R^{-m-1}) to be encountered.

Small errors in the $P[i]$ will not affect the sequence. Any set of $P[i]$ in the computer may be considered as a set of $\phi_{R[i]}(s_i)$, for generally large and unequal integers s_i, with small round-off errors. The arguments used in J. H. Halton's paper to establish the uniformity of the sequence of points

$$[\phi_{R_1}(n), \phi_{R_2}(n), \cdots, \phi_{R_K}(n)], \quad n = 1, 2, \cdots, N$$

can be applied identically to the more general sequence

$$[\phi_{R_1}(s_1+n), \phi_{R_2}(s_2+n), \cdots, \phi_{R_K}(s_K+n)], \quad n = 1, 2, \cdots, N.$$

Thus, theoretically, any "zero-th point" P will do. However, the difficulty described in (a) above limits us to the use of $P[i]$ corresponding to relatively small integers s_i.;

```
begin integer i, m;  real r, f, g, h;
  for i := 1 step 1 until K do
  begin r := 1.0/R[i];
    for m := 1 step 1 until N do
    begin if m > 1 then f := 1.0 - Q[m-1,i] else
      f := 1.0-P[i];
      g := 1.0;  h := r;
    repeat: if f - h < E then
        begin g := h;  h := h × r;  go to repeat end;
      Q[m,i] := g + h - f
    end
  end
end QRPSH
```

CERTIFICATION OF ALGORITHM 181 [S15] COMPLEMENTARY ERROR FUNCTION—LARGE X [Henry C. Thacher, Jr., *Comm. ACM 6* (June 1963), 315]

I. Clausen and L. Hansson (Recd. 20 Aug. 1964)
DAEC, Risø, Denmark.

The procedure *erfcL* was tested in Gier-Algol with 29 significant bits and the number-range $abs(x) < 2 \uparrow 512$ (approx. $1.3_{10}154$). The statement $m := R := 0$; was corrected to $m := 0$; $R := 0$; [Because m and R are of different type; cf. Sec. 4.2.4 of the Algol Report, *Comm. ACM 6* (Jan. 1963), 1–17.—Ed.] After this the tests were successful. The procedure was checked a.o. for $x = 1.19$ (-0.01) 0.72. The differences from table values increased from $_{10}-8$ at $x = 1.1$ to $7_{10}-8$ at $x = 0.75$. Overflow occurred at $x = 0.71$.

CERTIFICATION OF ALGORITHM 224 [F3] EVALUATION OF DETERMINANT [Leo J. Rotenberg, *Comm. ACM 7* (Apr. 1964), 243]

Vic Hasselblad and Jeff Rulifson (Recd. 17 July 1964)
Computer Center, U. of Washington, Seattle, Wash.

The "Evaluation of Determinant" program was tested on an Algol 60 compiler for an IBM 709 (Share distribution #3032). When the 10th line on page 244 was changed to read:

```
begin if imax = r then go to resume else
```

correct results were obtained. It was tested up through 4×4 matrices.

CERTIFICATION OF ALGORITHM 237 [A1] GREATEST COMMON DIVISOR [J. E. L. Peck, *Comm. ACM 7* (Aug. 1964), 481]

T. A. Bray (Recd. 8 Sept. 1964)
Boeing Scientific Research Laboratories, Seattle, Washington

This procedure was translated into the Fortran IV language and tested on the Univac 1107. No corrections were required and the procedure gave correct results for all cases tested.

Revised Algorithms Policy • May, 1964

A contribution to the Algorithms department must be in the form of an algorithm, a certification, or a remark. Contributions should be sent in duplicate to the editor, typewritten double-spaced in capital and lower-case letters. Authors should carefully follow the style of this department, with especial attention to indentation and completeness of references. Material to appear in **boldface** type should be underlined in black. Blue underlining may be used to indicate *italic* type, but this is usually best left to the Editor.

An algorithm must be written in the Algol 60 Reference Language [*Comm. ACM 6* (Jan. 1963), 1–17], and normally consists of a commented procedure declaration. Each algorithm must be accompanied by a complete driver program in Algol 60 which generates test data, calls the procedure, and outputs test answers. Moreover, selected previously obtained test answers should be given in comments in either the driver program or the algorithm The driver program may be published with the algorithm if it would be of major assistance to a user.

Input and output should be achieved by procedure statements, using one of the following five procedures (whose body is not specified in Algol): [see "Report on Input-Output Procedures for ALGOL 60," *Comm, ACM 7* (Oct. 1964), 628-629].

procedure *inreal* (*channel, destination*)· **value** *channel*; **integer** *channel*; **real** *destination*; **comment** the number read from channel *channel* is assigned to the variable *destination*, . . . ;
procedure *outreal* (*channel, source*); **value** *channel, source*; **integer** *channel*; **real** *source*; **comment** the value of expression *source* is output to channel *channel*; . . . ;
procedure *ininteger* (*channel, destination*);
value *channel*; **integer** *channel, destination*; . . . ;
procedure *outinteger* (*channel, source*);
value *channel, source*; **integer** *channel, source*; . . ;
procedure *outstring* (*channel, string*); **value** *channel*; **integer** *channel*; **string** *string*; . . ;

If only one channel is used by the program, it should be designated by 1 Examples:

```
outstring (1, 'x =');  outreal (1, x);
for i := 1 step 1 until n do outreal (1, A[i]);
ininteger (1, digit [17]);
```

It is intended that each published algorithm be a well-organized, clearly commented, syntactically correct, and a substantial contribution to the Algol literature. All contributions will be refereed both by human beings and by an Algol compiler. Authors should give great attention to the correctness of their programs, since referees cannot be expected to debug them. Because Algol compilers are often incomplete, authors are encouraged to indicate in comments whether their algorithms are written in a recognized subset of Algol 60 [see "Report on SUBSET ALGOL 60 (IFIP)," *Comm. ACM 7* (Oct, 1964), 626-627].

Certifications and remarks should add new information to that already published. Readers are especially encouraged to test and certify previously uncertified algorithms. Rewritten versions of previously published algorithms will be refereed as new contributions, and should not be imbedded in certifications or remarks.

Galley proofs will be sent to the authors; obviously rapid and careful proofreading is of paramount importance.

Although each algorithm has been tested by its author, no liability is assumed by the contributor, the editor, or the Association for Computing Machinery in connection therewith.

The reproduction of algorithms appearing in this department is explicitly permitted without any charge. When reproduction is for publication purposes, reference must be made to the algorithm author and to the *Communications* issue bearing the algorithm.—G.E.F.