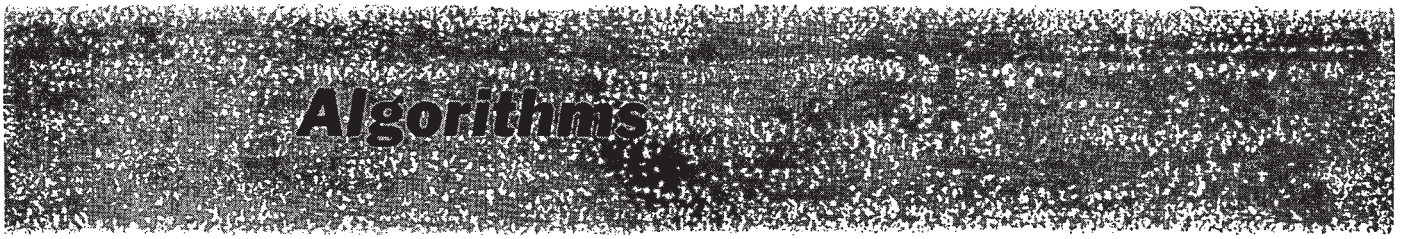

G. E. FORSYTHE, J. G. HERRIOT, Editors

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-