ALGORITHM 64
QUICKSORT

C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure   quicksort (A,M,N);  value M,N;
               array A;  integer M,N;
```

comment Quicksort is a very fast and convenient method of sorting an array in the random-access store of a computer. The entire contents of the store may be sorted, since no extra space is required. The average number of comparisons made is 2(M−N) ln (N−M), and the average number of exchanges is one sixth this amount. Suitable refinements of this method will be desirable for its implementation on any actual computer;

```
begin       integer I,J;
            if M < N then begin partition (A,M,N,I,J);
                                quicksort (A,M,J);
                                quicksort (A, I, N)
                          end
end         quicksort
```

ALGORITHM 65
FIND

C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure   find (A,M,N,K);  value M,N,K;
               array A;  integer M,N,K;
```

comment Find will assign to A [K] the value which it would have if the array A [M:N] had been sorted. The array A will be partly sorted, and subsequent entries will be faster than the first;

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

ALGORITHM 66
INVRS

JOHN CAFFREY
Director of Research, Palo Alto Unified School District, Palo Alto, California

```
procedure Invrs (t) size : (n); value n; real array t; inte-
  ger n;
comment Inverts a positive definite symmetric matrix t, of
order n, by a simplified variant of the square root method. Re-
places the n(n+1)/2 diagonal and superdiagonal elements of t
with elements of t⁻¹, leaving subdiagonal elements unchanged.
Advantages: only n temporary storage registers are required, no
identity matrix is used, no square roots are computed, only n
divisions are performed, and, as n becomes large, the number of
multiplications approaches n³/2;
begin integer i, j, s; real array v[1:n−1]; real y, pivot;
for s := 0 step 1 until n−1 do
begin pivot := 1.0/t[1,1];
    begin pivot := 1.0/t[1,1];
                  comment If t[1,1] ≦ 0, t is not positive defi-
                  nite;
          for i := 2 step 1 until n do v[i−1] := t[1, i];
          for i := 1 step 1 until n−1 do
              begin t[i,n] := y := −v[i] × pivot;
              for j := i step 1 until n−1 do
              t[i, j] := t[i + 1, j +1] + v[j] × y
              end;
          t[n,n] := −pivot
    end;
          comment At this point, elements of t⁻¹ occupy
          the original array space but with signs reversed,
          and the following statements effect a simple re-
          flection;
    for i := 1 step 1 until n do
    for j := i step 1 until n do t[i,j] := −t[i,j]
end Invrs
```

ALGORITHM 67
CRAM

JOHN CAFFREY
Director of Research, Palo Alto Unified School District, Palo Alto, California

```
procedure CRAM (n, r, a) Result: (f); value n, r; integer
  n, r; real array a, f;
comment CRAM stores, via an unspecified input procedure
READ, the diagonal and superdiagonal elements of a square sym-
metric matrix e, of order n, as a pseudo-array of dimension
1:n(n + 1)/2. READ (u) puts one number into u. Elements e[i, j]
are addressable as a[c + j], where c = (2n − i)(i − 1)/2 and c[i + 1]
may be found as c[i] + n − i. Since c[1] = 0, it is simpler to develop
a table of the c[i] by recursion, as shown in the sequence labelled
"table". Further manipulation of the elements so stored is illus-
trated by premultiplying a rectangular matrix f, of order n, r, by
the matrix e, replacing the elements of f with the new values, re-
quiring a temporary storage array v of dimension 1:n;
begin integer i, j, k, m; real array v[1:n]; real s;
  integer array c[1:n];
table: j := − n; k := n + 1; for i := 1 step 1 until n do
        begin
        j := j + k − i; c[i] := j end;
load:  for i := 1 step 1 until n do
       begin for j := i step 1 until n do READ (v[j]); m :=
       c[i];
       for k := i step 1 until n do a[m + k] := v[k] end;
premult: for j := 1 step 1 until r do
         begin for i := 1 step 1 until n do
                begin s := 0.0;
                  for k := 1 step 1 until i do
                  begin m := c[k]; s := s + a[m + i]
                    ×f[k, j] end;
                  for k := i + 1 step 1 until n do
                      s := s + a[m + k] × f[k, j]; v[i] = s
                end;
                for k := 1 step 1 until n do f[k, j] = v[k]
         end
end CRAM
```

REMARK ON ALGORITHM 53
Nth ROOTS OF A COMPLEX NUMBER (John R. Herndon, *Comm. ACM 4*, Apr. 1961)

C. W. NESTOR, JR.
Oak Ridge National Laboratory, Oak Ridge, Tennessee

A considerable saving of machine time for $N \geqq 3$ would result from the use of the recursion formulas for the sine and cosine in place of an entry into a sine-cosine subroutine in the do loop associated with the Nth roots of a complex number. That is, one could use

$$\sin (n + 1)\theta = \sin n\theta \cos\theta + \cos n\theta \sin\theta$$
$$\cos (n + 1)\theta = \cos n\theta \cos\theta - \sin n\theta \sin\theta,$$

at the cost of some additional storage.

We have found this procedure to be very efficient in problems dealing with Fourier analysis, as suggested by G. Goerzel in chapter 24 of *Mathematical Methods for Digital Computers*.

Contributions to this department must be in the form stated in the Algorithms Department policy statement (*Communications*, February, 1960) except that ALGOL 60 notation should be used (see *Communications*, May, 1960). Contributions should be sent in duplicate to J. H. Wegstein, Computation Laboratory, National Bureau of Standards, Washington 25, D. C. Algorithms should be in the Publication form of ALGOL 60 and written in a style patterned after the most recent algorithms appearing in this department.

Although each algorithm has been tested by its contributor, no warranty, express or implied, is made by the contributor, the editor, or the Association for Computing Machinery as to the accuracy and functioning of the algorithm and related algorithm material and no responsibility is assumed by the contributor, the editor, or the Association for Computing Machinery in connection therewith.
