ALGORITHM 76
SORTING PROCEDURES

IVAN FLORES
Private Consultant, Norwalk, Connecticut

comment The following ALGOL 60 algorithms are procedures for the sorting of records stored within the memory of the computer. These procedures are described in detail, flow-charted, compared, and contrasted in "Analysis of Internal Computer Sorting" by Ivan Flores [*J. ACM 8* (Jan. 1961)]. Although sorting is usually a business computer application, it can be described completely in ALGOL if we stretch our imagination a little. Sorting is ordering with respect to a key contained within the record. If the key *is* the active record, the sorting is trivial. A means is required to extract the key from the record. This is essentially string manipulation, for which no provision, as yet, has been made in ALGOL. We circumambulate this difficulty by defining an **integer procedure** K(I) which "creates" a key from the record, I. ALGOL *does* provide for machine language code substitutions, which is one way to think of K(I). This could be more accurately represented by using the string notation proposed by Julien Green ["Remarks on ALGOL and Symbol Manipulation," *Comm. ACM 2* (Sept. 1959), 25-27]. The function **sub** ($,i,g) represents the procedure, K(I). $ corresponds to the record I, i corresponds to the starting position of the key and g corresponds to the length of the key. Both i and g are **values** which must be specified when the sort procedure is called for as a statement instead of a declaration.

Another factor, which might vex some, is that the key might be alphabetic instead of numeric. Then, of course, K(I) would not be integer. It would, however, be string when such is defined eventually. Note, also, that keys are frequently compared. This is done using the ordering relations ">" for "greater than," etc. These are not really defined in the ALGOL statement [NAUR, PETER, ET AL. "Report on the Algorithmic Language ALGOL 60". *Comm. ACM 3* (May 1960), 294-314]. They can simply be defined so that $Z > Y > \cdots > A > 9 > \cdots > 1 > 0$. Also the assignment X[i] := z should be interpreted as "Assign the key 'z' which is larger than any other key." For any sort procedure (I,N,S), "I" is the set of unsorted records, "N" is their number, and "S" the sorted set of records.

Caution, these algorithms were developed purely for the love of it: No one was available with the combined knowledge of sorting and ALGOL to check this work. Hence each algorithm should be carefully checked before use. I will be glad to answer any questions which may arise;

```
Sort insert (I,N,S); value N; array I[1:N], S[1:N];
  integer procedure K(I); integer N;
  begin integer i, j, k;
    S[1] := I[1];
    for i := 2 step 1 until N do begin
      for j := i - 1, j - 1 while K(I[i]) > K(S[j]) do
      for k := i step - 1 until j + 1 do
        S[k] := S[K - 1];
      S[j + 1] := I[i] end end

Sort count (I,N,S); value N; array I[1:N], S[1:N];
  integer procedure K(I); integer N;
  begin integer array C[1:N]; integer i,j;
    for i := 1 step 1 until N do C[i] := 0;
    for i := 2 step 1 until N do
      for j := 1 step 1 until i - 1 do
        if K(I[i]) > K(I[j]) then C[i] := C[i] + 1
        else C[j] := C[j] + 1;
    for i := 1 step 1 until N do
      S[C[i]] := I[i] end

Sort select (I,N,S); value N; array I[1:N], S[1:N];
  integer procedure K(I); integer N;
  begin integer i,j,A,h;
    for i := 1 step 1 until N do begin
    h := K(I[1]);
    for j := 2 step 1 until N do
    if h > K(I[j]) then begin h := K(I[j]); A := j end;
    S[i] := I[A];
    I[A] := z end end

Sort select exchange (I,N); value N; array I[1:N];
  integer procedure K(I); integer N;
  begin integer h,i,j,H; real T;
    for i := 1 step 1 until N do begin
      H := K(I[i]); h := i;
      for j := i + 1 step 1 until N do
        if K(I[j]) < H then begin
        H := K(I[j]); h := j end
      T := I[i]; I[i] := I[h]; I[A] := T end
  end

Sort binary insert (I,N,S); value N; array I[1:N], S[1:N];
  integer procedure K(I); integer N;
  begin integer i,k,j,l;
    if K(I[1]) < K(I[2]) then begin
      S[1] := I[1]; S[2] := I[2] end
      else begin S[1] := I[2]; S[2] := I[1] end;
  start:     for   i := 3 step 1 until N do begin
                   j := (i + 1) ÷ 2;
  find spot:       for k := (i + 1) ÷ 2, (k + 1) ÷ 2 while k > 1 do
                     if K(I[i]) < K(S[j]) then j := j - k
                     else j := j + k;
                   if K(I[i]) ≥ K(S[j]) then j := j - 1;
  move items:      for l := i step - 1 until j do
                     S[l + 1] := S[l];
  enter this
  one:             S[j] := I[i] end end

Sort address calculation (I,N,S,F); value N;
  array S[1:M], I[1:N]; integer procedure F(K), K(I);
  integer N,M;
             begin integer i,j,G,H,F,M;
               M := entier(2.5 × N)
               for i := 1 step 1 until M do S[i] = 0;
 Address:      for i := 1 step 1 until N do begin
               F := F(K(I[i]));
               if S[F] = 0 then begin S[F] := I[i];
                 go to NEXT end
               else if K(S[F]) > K(I[i]) then go to SMALLER;
 LARGER:       for H := F, H + 1 while K(S[H]) < K(I[i]) do
               for G := H, G + 1 while K(S[G]) ≠ 0 do
               for j := G step -1 until H + 1 do
                 S[j] := S[j - 1];
               S[H] := I[i]; go to NEXT;
 SMALLER:      for H := F, H - 1 while K(S[H]) > K(I[i]) do
```

```
              for G := H, G − 1 while K(S[G]) ≠ 0 do
              for j := G step 1 until H − 1 do
                S[j] = S[j + 1];
              S[H] := I[i];
NEXT:       end end
```

```
Sort quadratic select (I,N,S); value N; array I[1:N], S[1:N];
  integer procedure K(I); integer N;
                begin integer i,j,k,C,D,J,M;
                  integer array C[1:M], D[1:M];
                  array I[1:M, 1:M];
  Divide inputs:  M := entier (sqrt (N)) + 1; j := k := 1;
                  for i := 1 step 1 until N do begin
                    I[j,k] := I[i]; k := k + 1;
                    if k > M then begin k := 1;
                      j := j + 1 end end
  Fill up inputs: I[j,k] := z; k := k + 1;
                  if k > M then begin k := 1; j := j + 1 end
                  if j ≤ M then go to Fill up inputs;
  Set controls:   for j := 1 step 1 until M do begin
                  C[j] := K(I[j, 1]); D[j] := 1;
                  for k = 2 step 1 until M do
                    if C[j] > K(I[j,k]) then begin
                    C[j] := K(I[j,k]); D[j] := k end end;
                  i := 1;
  Find least:     C := C[1]; D := D[1]; J := 1;
                  for j := 2 step 1 until M do
                    if C > C[j] then begin C := C[j];
                      D := D[j]; J := j end;
  Fill file:      S[i] := I[J,D]; i := i + 1; I[J,D] := z;
                  if i = N + 1 go to STOP;
  Reset controls: for j := J do begin
                  C[j] := K(I[j, 1]); D[j] := 1;
                  for k := 2 step 1 until M do
                    if C[j] > K(I[j,k]) then begin C[j] :=
                      K(I[j,k]; D[j] := k end end;
                  go to Find least;
  STOP:           end
```

```
Presort quadratic selection (I,N,S); value N;
  array I[1:N], S[1:N]; integer procedure K(I); integer N;
                begin integer i,j,k,C,J,M;
                  integer array C[1:M], D[1:M];
                    array I[1:M,1:M];
  Divide inputs:  M := entier (sqrt(N)) + 1; j := k := 1;
                  for i := 1 step 1 until N do begin
                    I[j,k] := I[i]; k := k + 1;
                    if k > M then begin k := 1;
                      j := j + 1 end end
  Fill up inputs: I[j,k] := z; k := k + 1;
                  if k > M then begin k := 1; j = j + 1 end
                  if j ≤ M then go to Fill up inputs;
  First sort:     for j := 1 step 1 until M do
                  sort select exchange (I[j,k],M);
  Set controls:   for j := 1 step 1 until M do begin
                    C[j] := K(I[j,1]); D[j] := 1 end
                  i := 1;
  Find least:     C := C[1]; J := 1;
                  for j := 1 step 1 until M do
                    if C > C[j] then begin C := C[j];
                      J := j end;
  Fill file:      S[i] := I[J,D[J]]; i := i + 1;
                  if i = N + 1 go to STOP
  Reset control:  for j := J do begin
                    D[j] := D[j] + 1;
                    if D[j] > M then C[j] := z else C[j] :=
                      K(I[j, D[j]]) end
                  go to Find least;
  STOP:           end
```

```
Sort binary merge (I,N,S); value N; array I[1:N];
  integer procedure K(I); integer N;
  begin real array S[1:N];
  integer array A[0:1, 0:J[a]], B[0:1, 0:K[b]], Aloc[0:1, 0:J[a]],
    Bloc[0:1, 0:K[b]], J[0:1], K[0:1], j[0:1], k[0:1];
   integer a,b,i,j,k;
  distribute:      a := b := j[0] := j[1] := 1;
                   for i := 1 step 1 until N do begin
                     if K(I[i]) < K(I[i−1] then
                       if a = 1 then a := 0 else a := 1;
                     A[a, j[a]] := K(I[i]); Aloc[a, j[a]] := i;
                       j[a] := j[a] + 1 end;
                   J[0] := j[0]; J[1] := j[1];
  next sort:       begin a := b := j[0] := j[1] := k[0] :=
                     k[1] := 1;
  two inputs:      if A[1, j[1]] ≤ A[0, j[0]] then a := 1 else
                     a := 0;
                   B[b, k[b]] := A[a, j[a]];
                     Bloc[b, k[b]] := Aloc[a, j[a]];
                   j[a] := j[a] + 1; k[b] := k[b] + 1;
                   if A[a, j[a]] ≥ A[a, j[a] − 1] then go to two
                     inputs else
                   if a = 1 then a := 0 else a := 1;
  single step:     B[b, k[b]] := A[a, j[a]];
                     Bloc[b, k[b]] := Aloc[a, j[a]];
                   j[a] := j[a] +1; k[b] := k[b] + 1;
                   if A[a, j[a]] ≥ A[a, j[a] − 1] then go to
                     single step;
  switch file:     if b = 1 then b := 0 else b := 1;
  check rollout:   for a := 0, 1 do
                     if j[a] = J[a] then go to rollout;
                   go to two inputs;
  rollout:         B[b, k[b]] := A[a, j[a]];
                     Bloc[b, k[b]] := Aloc [a, j[a]];
                   k[b] := k[b] + 1; j[a] := j[a] + 1;
                   if j[a] = J[a] then go to interchange files;
                   if A[a, j[a]] < A[a, j[a] − 1] then
                     if b = 1 then b := 0 else b := 1;
                   go to rollout;
  interchange files: K[0] := k[0]; K[1] := k[1];
                   if K[0] = 1 then go to output end
                   for b := 1, 0 do begin
                     for k[b] := 1 step 1 until K[b] do begin
                       A[b, k[b]] := B[b, k[b]];
                         Aloc[b, k[b]] := Bloc[b, k[b]];
                       J[b] := K[b] end end
                   go to next sort;
  output:          for i := 1 step 1 until N do
                     S[i] := I[Bloc[0, i]];
                   end
```

REMARK ON ALGORITHM 76
SORTING PROCEDURES (Ivan Flores, *Comm. ACM* *5*, Jan. 1962)

B. Randell
Atomic Power Div., The English Electric Co., Whetstone, England

The following types of errors have been found in the Sorting

Procedures:

1. Procedure declarations not starting with **procedure**.
2. Bound pair list given with array specification.
3. = used instead of :=, in assignment statements, and in a **for** clause.
4. A large number of semicolons missing (usually after **end**).
5. Expressions in bound pair lists in array declarations depending on local variables.
6. Right parentheses missing in some procedure statements.
7. Conditional statement following a **then.**
8. No declarations for A, or z, which is presumably a misprint.
9. In several procedures attempt is made to use the same identifier for two different quantities, and sometimes to declare an identifier twice in the same block head.
10. In the Presort quadratic selection procedure an array, declared as having two dimensions, is used by a subscripted variable with only one subscript.
11. At one point a subscripted variable is given as an actual parameter corresponding to a formal parameter specified as an array.
12. In several of the procedures, identifiers used as formal parameters are redeclared, and still assumed to be available as parameters.
13. In every procedure K is given in the specification part, with a parameter, whilst not given in the formal parameter list.

No attempt has been made to translate, or even to understand the logic of these procedures. Indeed it is felt that such a grossly inaccurate attempt at ALGOL should never have appeared as an algorithm in the *Communications*.