Erik Bugge
Aarhus Universitets Regnecenter
May 1968.

Sorting of Multidimensional Arrays by Reference.
ALGOL procedure.

ABSTRACT:

The procedure, ≪ sorter ≫ , sorts the values of a twodimensional array corresponding to a fixed second subscript value, building into a onedimensional array a reference which is the corresponding values of the first subscript. Versions handling multidimensional arrays and methods for multiprecision numbers are mentioned.

CONTENTS:

1. Function.
2. Time and Space requirements.
3. The Procedure.
4. Other Versions.
5. Note on Multiprecision.

1. Function:

If ⟨type⟩ <u>array</u> files[1:n,1:m] is the array to be sorted and <u>integer</u> <u>array</u> ref[1:n] contains the numbers 1 through n in some order then the call:

sorter(files,item1,ref,1,n);

will rearrange the numbers contained in ref so that files[ref[1], item1] ≤ files[ref[2],item1] ≤ files[ref[3],item1] ≤ etc.

Note that the values contained in files are <u>not</u> rearranged.

Note also that the sorting is conservative i.e. no changes of order are made in case of equality. This means that if a sorting-criterion (the set of numbers corresponding to a fixed value of the second subscript) contains one or more ranges of equal numbers then the order gained from the previous sorting(s) will remain inside such ranges, a feature which permits the sorting of multiprecision numbers and the use of several sortingcriteria in a hierarchic order.

The user may restrict the sorting to some part of the array if the reference to this part has been placed in an unbroken range of the reference array. This option is governed by the two last parameters of the call, thus

sorter(files,item2,ref,from,to);

will rearrange the numbers contained in ref[from] through ref[to] only, leaving the rest of ref unchanged. (For a method using this feature for hierarchic sorting see note on multiprecision below).

The working principle of the procedure is: A secondary reference array subscripted [from:to] is declared, adjacent numbers from ref are ordered two by two and put into this sec. ref., then adjacent groups of two are merged into ordered groups of four in ref which subsequently merges into groups of eight in sec.ref. etc. the proces continuing until the whole range of numbers is contained in one single ordered group. If this group happens to end in sec. ref. it is finally copied onto the corresponding part of ref.

2. Time and Space requirements:

The time taken to sort a reference containing N numbers is 2 sec. and 22 sec. for N equal to 100 and 800 respectively, or roughly:

$$t = 10 \times N \log N \quad \text{milliseconds}$$

(using the Gier ALGOL IV compiler without subscript check).

For an array[1:n,1:m] to be sorted is two onedimensional arrays of length n required, and so the total space neccessary is $n \times (m+2)$. It is seen that the <u>extra</u> space required is less dominant the larger m is.

3. The Procedure:

The procedure heading is

<u>procedure</u> sorter(lager,nr,a,fra,til);
<u>value</u> nr,fra,til;
<u>integer</u> nr,fra,til;
<u>integer</u> <u>array</u> a; <u>integer</u> <u>array</u> lager;

The meaning of the formal parameters is:

lager: the array to be sorted
nr: the value of the second subscript corresponding to the sortingcriterion
a: the reference array
fra: the lower bound, and
til: the upper bound of the part of the reference array in question.

The subscript bounds of a need not to be the same as the bounds of the first subscript of lager but each of the relevant values of the first subscript must occur once and only once as values in a. The two last parameters must be included between the bounds of a and satisfy the condition: til $\geq$ fra.

A full account of the procedure is found in the appendix, it consists of:

a) the declaration of lokal variables and of the sec. ref. array b

b) an initialization of the transport direction from a to b

c) the major for statement which multiplies the group length, step, by two and reverses the transport direction for each loop

d) an eventual copiing of b onto a.

The outer for statement is made up of two nearly identical parts, one for each transport direction, consisting of a for statement which advances the treatment through the whole range, inside which two smaller loops perform the merging. Theese smaller loops are forced not to include segment transitions by the use of <u>end</u> comments ending by <u>for</u> (a feature of Gier ALGOL IV) to save time rather than program space.

4. Other Versions:

The array, lager, is specified as <u>integer</u>, this must be altered to <u>real</u> if the procedure is to handle <u>real</u> arrays. (If the procedure is compiled by the Gier ALGOL IV compiler it cannot handle both <u>integer</u> and <u>real</u> arrays. It can if lager is specified as <u>real</u> and the Gier ALGOL III compiler is used, but in this case the operation times are longer than stated above).

The dimensionality of lager and the different role of the subscripts are determined by the two occurrences of the if clause:

<u>if</u> lager[n1,nr] $\leq$ lager[n2,nr] <u>then</u> . . .

only. Alterations of the number of subscripts must be accompagnied by corresponding alterations of the formal parameters if the procedure is to be kept void of global variables.

The procedure uses about 3 tracks of program space, this can be cut down by about 50 %, with a subsequent doubling of the operating time, if the direction governing clause:

<u>if</u> i a <u>then</u> . . . <u>else</u> . . .

is placed inside the following for statement anywhere the two halves of the procedure are different. Time saving has been the author's major concern while writing the present version.

5. Note on Multiprecision:

If <u>integer</u> <u>array</u> numbers[1:n,1:2] is taken as an onedimensional array of doubleprecision numbers they can be sorted either thus:

```
sorter(numbers,2,ref,1,n);
sorter(numbers,1,ref,1,n);
```

due to the conservative nature of the sorting, or thus:

```
sorter(numbers,1,ref,1,n);
to:= 1;
for from:= to while from< n do
```

```
begin
x:= numbers[ref[from],1];          to:= to + 1;
if x = numbers[ref[to],1] then
begin
for to:= to + 1 while (if to ≤ n then x = numbers[ref[to],1]
                                   else false) do;
sorter(numbers,2,ref,from,to - 1)
end
end;
```

The outer for statement scans the whole reference for ranges of equality of the more significant part which, if they contain more than just one member, are then sorted according to the less significant part.
The inner, empty, for statement explores the lengths of such ranges.

The former method is the simpler but the latter is likely to be the faster in most cases. Both methods are easily extended to hierarchic sorting of higher degrees.

APPENDIX:

```
procedure sorter(lager,nr,a,fra,til);
value nr,fra,til;
integer nr,fra,til;
integer array a;    integer array lager;
begin
integer l,l1,l2,sl1,sl2,n1,n2,step,diff;
integer array b[fra:til];        boolean i a;
i a:= true;
for step:= 1,2×step while step < til - fra +1 do
begin
l:= fra -1;
if i a then
begin
for l1:= l+1 while l < til do
begin
sl1:= l + step;    l2:= sl1 + 1;    sl2:= sl1 + step;
if sl1 > til then sl1:= sl2:= til else
if sl2 > til then sl2:= til;    diff:= sl2 - sl1;
for l:= l+1 while l1 ≤ sl1 ∧ l2 ≤ sl2 do
begin
n1:= a[l1];    n2:= a[l2];
if lager[n1,nr] ≤ lager[n2,nr] then begin b[l]:= n1; l1:= l1+1 end
else begin b[l]:= n2; l2:= l2+1 end
end for l:= for ;
if l1 ≤ sl1 then begin
for l:= l step 1 until sl2 do begin b[l]:= a[l-diff] end for end
else for l:= l step 1 until sl2 do begin b[l]:= a[l] end for;
l:= sl2
end for l1:=
end if i a else
begin comment det samme med a og b byttet om;
for l1:= l+1 while l < til do
begin
sl1:= l + step;    l2:= sl1 + 1;    sl2:= sl1 + step;
if sl1 > til then sl1:= sl2:= til else
if sl2 > til then sl2:= til;    diff:= sl2 - sl1;
for l:= l+1 while l1 ≤ sl1 ∧ l2 ≤ sl2 do
begin
n1:= b[l1];    n2:= b[l2];
if lager[n1,nr] ≤ lager[n2,nr] then begin a[l]:= n1; l1:= l1+1 end
else begin a[l]:= n2; l2:= l2+1 end
end for l:= for ;
if l1 ≤ sl1 then begin
for l:= l step 1 until sl2 do begin a[l]:= b[l-diff] end for end
else for l:= l step 1 until sl2 do begin a[l]:= b[l] end for;
l:= sl2
end for l1:=
end if ¬ i a;
i a:= ¬ i a
end for step:=;
if ¬ i a then begin
for l:= fra step 1 until til do a[l]:= b[l] end
end procedure sorter;
```