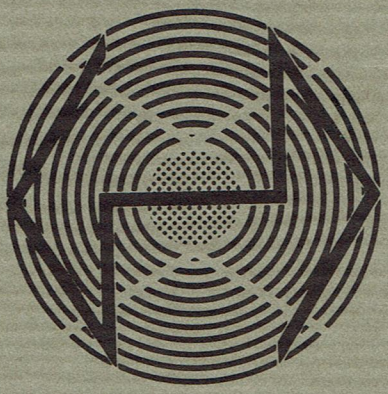

A/S REGNECENTRALEN

SCANDINAVIAN INFORMATION PROCESSING SYSTEMS

RC SYSTEM LIBRARY: FALKONERALLE 1 · DK-2000 COPENHAGEN F

RCSL No: 28-D3

Edition: March 1972

Author: Jørgen Winther

Title: Mdsort

Keywords: RC 4000, Software, Disc, Sorting, Program

Abstract: Mdsort sorts a backing storage area holding records of either fixed or variable length. Mdsort is based upon the standard procedure mdsortproc. 8 pages.

Purpose.

Mdsort, merge-disc-sort, is a sorting program intended for fast sorting of one disc file holding records of either fixed or variable length.

File format.

Records of fixed length are handled by means of inrec6/outrec6, records of variable length are handled by invar/outvar.
The length of a file is either given in the catalog entry of the input file:
inputfile=set <segments> <storage device> <number of records>
or by a special end of file record. See the call definition below.

Method.

The program is based upon the external algol procedure mdsortproc. This procedure performs the sort in 2 phases:

1: The input file is read, and sorted strings of the maximum length are output consecutively in one area.

2: The strings generated in phase 1 are merged together in the needed number of passes.

The procedure will optimize the sorting by variation of the number of passes, the blocklengths, and the number of shares, and by utilization of 2 disc stores if available.

Requirements.

The merge technique requires 2 backing storage areas able to hold the data.
One of these areas can be the input file if the program is allowed to clear it.
The job process must own at least 7 message buffers, but it is recommended to have at least 8, and especially in connection with sorting of short records, (4 to 20 bytes), a higher number of message buffers can be utilized (10 to 18).
If the number of free message buffers indicates that the job process has got less than 10 message buffers, a warning is given.
The minimum core size is given in bytes by the following expression:

11000 + 512×(inputblocklength + outputblocklength)
+ 4×maxlength + 12×noofkeys.

The blocklengths are given in segments, (512 bytes), and the maximum recordlength in bytes.

The minimum core size for blocklengths of 2 segments is thus about 13 to 14000 bytes, but it is emphasized that this core size will give a very inefficient sort, 30 to 50000 would be more appropriate, depending on the data volume.

Example of program call.

For a more exhaustive definition, see the next section.

```
mdsort  in.file1  out.file2 , input and output files
        block.2.2           , input and output blocklengths in
                            , segments.
        var.34              , variable reclength, max 34 bytes.
        long.8              , first sorting criterion, ascending
        real.20.d           , second, descending
        word.10             , etc.
        byte.11.d
```

Program call definition.

Two different calls are possible.

1. mdsort <sortfiles> <sortspecification>

2. mdsort <sortspec.file> <sortfiles>

The call syntax defined in the following demands that points are used where specified here, and only there, also in the sortspec.-file.

Sortfiles.

Fp-parameters defining input and output.

<sortfiles>::= in.<input file> <.clear>0/1 <.count>0/1
out.<output file> <.<output disc> >0/1

<input file>, <output file>, and <output disc>::= <name>

The signature 0/1 means that the preceding quantity can be omitted.
The clear parameter defines, whether the input file should be cleared or not. The parameter may be necessary in connection with great data volumes.
The count parameter is only usefull in connection with the eof-parameter in the sortspecification, because its effect is to counteract the eof-parameter.
The output file is created by the program, and placed on the output disc, if this parameter is given.
An existing file of the same name as output file is cleared.

Sortspecification.

Fp-parameters defining the details of the sort.

<sortspecification>::= block.<input blockl.>.<output blockl.>
<fix or var>.<maxlength>
< eof.<eof 1>.<eof 2> >0/1
<keyfield> 1/n

<input blocklength>, and <output blocklength>::= <integer>

Two integers specifying the blocklengths of the input file and the final output file as a number of segments, (512 bytes). The maximum blocklength is 20 segments.

<fix or var>::= fix | var

Defines whether the records of the input file was created by outrec or by outvar.

<maxlength>::= <integer>

Defines the fixed or the maximum length of a record measured in bytes.
It must be even, and not less than 4 in the case of fixed record-length, and not less than 8 in the case of variable recordlength.
It is important for the efficiency of the sort that maxlength is given as accurate as possible in the case of variable record-length.

<eof 1>, and <eof 2>::= <integer>

If the eof parameter does not occur, then the length of the input file is given by the content of the catalog entry of the file.
The word succeeding the name of the disc storage device in the tail of the entry must contain the number of records to be sorted:

inputfile=set <segments> <storage dev.> <number of records>

But if the eof parameter is given, then the file end of the input file is defined by a special record having <eof 1>, and <eof 2> as the values of the first 2 words of the user part of the record, i.e., byte 1 to 4 of fixed length, and byte 5 to 8 of variable length records.
If the count parameter occurred in <sortfiles>, then the number of records in the input file is fetched from the catalog entry in any case.
The number of sorted records is always inserted in the catalog entry of the output file, and an end of file record is written at the end of the file.
If the eof parameter is not given the value zero is used for <eof 1>, and <eof 2> in this record.

<keyfield>::= <type>.<position> <.d>0/1

This is the specification of one keyfield.
Up to 30 keyfields can be defined in the order of decreasing priority.

<type>::= byte | word | long | real

The types correspond to the types 1 to 4 in the internal sorting system of rc4000 algol.

<position>::= <integer>

Specifies the field address of the keyfield.
The keyfield must be entirely within the user part of a maximum length record, and only a byte keyfield may have an odd position.

<.d>

If the sorting order should be descending, this parameter must occur.

Sortspec. file.

In case of program call 2, the sortspecification is given as the content of a textfile, defined by <sortspec. file>.

<sortspec.file>::= <name>

The same fp-syntax is maintained however, with the exceptions that the cancel character, and the construction <s> are not allowed.
Empty lines at the beginning of the sortspec. file are skipped, but if the sortspecification should continue over several lines the comma must be used as usual as the continuation symbol.

The philosophy behind program call 2 is that the more dynamic parts of the program call, the file names, should be separated from the more static parts, the sortspecification.
It is in fact allowed to break the call into two parts in any way after the <sortspec. file> parameter.
The call: mdsort <sortspec. file>, is thus completely legal, if <sortspec. file> contains the required parameters.

Variable length records.

The sum check facility of invar is used during the reading of the input file.
The record length must not exceed maxlength, and it must not be odd.
The minimum record length is given by the greatest of the two values: 8 and the position of the first keyfield.
Thus some of the keyfields of a short record may in fact be situated outside the record.
Such a record is sorted as if all the bits of keyfields outside the record were equal to zero.

Printed output.

1. The call is listed on current output in a standard way.

2. The text: sort start:, is printed when all parameters have been read and syntax checked.

3. The procedure mdsortproc will print a warning if the job owns less than 10 message buffers.

4. The procedure mdsortproc will print the expected remaining sorting time in minutes two times, the first one, when the parameters have been finally checked, and the second time, just before phase 2 is entered.
The expected sorting time is printed both as a parent message, and on current output.

5. The text: sort ok:, and a report of the number of records and segments output, and the time consumed is printed on current output if the sort was successfull.

6. Alarms.
The sort is stopped by a runtime alarm if some parameter is illegal, in case of the lack of some resource, or in case of some error concerning the file format.
The alarm is preceded by the text: xxxmdsort alarm:, if mdsort detected the error, if mdsortproc did it, the text: xxxmdsortproc alarm:, precedes the alarm.
The ok-bit will be false after an alarm.

alarms from mdsort.

alarm text.	alarm integer.	comment
syntax	param number	syntax error in sortspec.file.
wr.param	param number	wrong parameter structure.
noofkeys	number of keys	too many keyfields.
no input	monitor result	error in lookup input file.
noofrecs	tail(6)	number of recs in tail < 0.
c.store	needed bytes	not sufficient core storage.
b.store	needed segments	not sufficient backing storage.
out disc	0	output disc does not exist.

Note that if the clear parameter is used, then the input file may have been cleared in the case of the b.store alarm occurring after the first print of the expected sorting time, and that it has been cleared for certain if some alarm occurs after the second print of the expected sorting time, but alarms so late in the sort should be very uncommon.

alarms from mdsortproc.

alarm text.	alarm integer.	comment
param	1	wrong input blocklength.
	3	wrong output blocklength.
	5	wrong maxlength.
	6	noofkeys > maxlength.
keyfield	keyfield no.	illegal position of keyfield.
infile	tail(1)	input file is not an area.
r.length	record length	illegal variable length.
mess.buf	buffers of job	too few message buffers.
create	monitor result	should not occur.
lookup	-	-
change	-	-
rename	-	final output cannot be renamed.
remove	-	input file cannot be cleared.
passes	20	should not occur.
reccount	record count	-

In addition alarms from stderror may occur if record or file formats are strongly illegal.

The alarm r.length, and stderror alarms occurring during the reading of the input file are also preceded by a line, providing the number of input records accepted before the error was detected.

Examples of time consumption.

1. Fixed record length, 100 bytes.

   core size = 42000 bytes.
   noofrecs = 15000 records.
   file size = 3000 segments.
   one RC 433 disc store available.
   real time = 8.7 minutes = 35 milliseconds per record.
   cpu time = 2.9 minutes.

2. Fixed record length, 4 bytes.

   core size = 42000 bytes.
   noofrecs = 20000 records.
   file size = 157 segments.
   one RC 433 disc store available.
   real time = 1.9 minutes = 5.7 milliseconds per record.
   cpu time = 1.6 minutes.

3. Variable record length, max 160 bytes.

   random record length, 8 to 160 bytes.
   core size = 42000 bytes.
   noofrecs = 7000 records.
   file size = 1190 segments.
   one RC 433 disc store available.
   real time = 4.6 minutes = 39 milliseconds per record.
   cpu time = 1.9 minutes.

4. Variable record length, max 80 bytes.

   mean record length = 50 bytes.
   core size = 65000 bytes.
   noofrecs = 75785 records.
   file size = 7350 segments.
   two RC 433 disc stores available.
   real time = 20.7 minutes = 16.4 milliseconds per record.
   cpu time = 15.2 minutes.

   The job had only got 7 message buffers, a higher number of bufs might have decreased the real time to about the cpu time above.