RCSL No: 31-D562

Edition: April 1979

Author: Jørgen Winther
Inge Borch

Title:

RC8000 Backing Storage Area Sorting
mdsort - mdsortproc

A/S REGNECENTRALEN af 1979

Keywords:

RC8000, algol, backing storage package, sorting, manual.

Abstract:

Describes a program, mdsort, and an external algol procedure, mdsort-proc, which sort records stored in an RC8000 backing storage area. The records may be either of fixed or variable length and the key composed of any mix of simple fields.

(20 printed pages)

RCSL 42-i1905

TABLE OF CONTENTS PAGE

1. INTRODUCTION 1.

The mdsort-system (merge disc sorting) is intended for fast sorting of records in an RC8000 backing storage area. The system consists of a procedure, mdsortproc, and a program, mdsort, with the same function, so it may be used either from an application program or directly from a job file. An abstract of the system is given below:

1.1 File Conventions. 1.1

Input to the system is a file blocked either in algol format or in SQ-format (ref.2). Output will be stored in a file in the same format as that of the input file, and may be the same as the input file, another user file, or a file created by the system. Scope rules are explained in section 2. and 3.

The user may specify a blocklength of the output file, which differs from that of the inputfile. The maximum blocklength is 40 segments.

The length of the file is given either as the number of file records to be sorted or as an end of file record. The first possibility is recommended as it gives the most efficient sorting.

1.2 Record Conventions. 1.2

Records may be of either fixed or variable length. The format is that of the algol high level zone procedures. (Ref.1). The maximum length is the blocklength.

A record key may be composed of up to 50 key fields in the program and 169 in the procedure. Their types may be 12-bits integer, 24-bits integer, 48-bits integer, 48-bits real, or 12-bits positive integer. (Text fields may be handled as the second, the third, or the fifth mentioned type). Each key field can be used in either ascending or descending mode. The mutual order of the key fields is free.

The address of each field must be specified within the maximum record length, and the highest priority key field be present in every record. If some key fields are not present, they will be substituted by the value 0.

Synonymous records will appear in the same order in the output file as in the input file.

1.3 Sorting Strategy. 1.3

The basic sorting method is the merge technique: sorted strings of records, as long as possible, are generated by internal sorting during the first reading of the input file. After that, these strings are merged repeatedly until only one sorted string is left. The internal sorting method is the heap as referred in ref.3.

The system will try to minimize the sorting time by variation of mergepowers, and blocklengthes for workareas, by use of single- or double buffering, and by utilization of two discs, if possible.

1.4 System Requirements. 1.4

The merge technique requires 2 backing storage areas able to hold the file. One of these may be the input file, if the system may clear it.

The minimum primary storage requirement is about 15000 halfwords for blocklengthes of 2 segments, but it is emphasized that this amount will give a very inefficient sorting. 30000 to any size would be more appropriate, depending on the data volume.

2. MDsort, User's Description

Purpose.

mdsort, merge_disc_sort, is a sorting program 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 number of records of the file is either supplied by the sq system (contents = 21) or given in the catalog entry of the input file:

```
    inputfile=set <segments> <disc name> <number of records>
```

or by a special end of file record.

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 discs if available.

The mutual ordering of records with equal values in all keyfields, i.e. synonyms, is not changed by the sorting if the maximum record length is less than 2046.

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 2 message buffers, but the computer can be utilized harder with a greater number, up to about 20, especially with great core size.
The safe minimum core size is given in halfwords by the following expression:

```
    12000 + 512*(inputblocklength + outputblocklength)
    + 4*maxlength + 48*noofkeys.
```

The blocklengths are given in segments, (512 halfwords), and the maximum recordlength in halfwords.
The minimum core size for blocklengths of 2 segments is thus about 15000 halfwords, but it is emphasized that this core size will give a very inefficient sort, 30000 to any size 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             ; input and output blocklengths in
                            ; segments.
        var.34              ; variable reclength, max 34 halfw.
        long.8              ; first sorting criterion, ascending
        real.20.d           ; second, descending
        abshalf.11.16.d     ; criterion 3, 6 unsigned halfwords
        word.10
```

Program call definition.

The program call consists of the following parts:

```
mdsort  <sortfiles>  <sortspecifikation>
```

Both <sortfiles> and <sortspecifikation> are groups of fp-parameters which will be defined in the sequel.

Sortfiles.

Fp-parameters defining input and output.

```
<sortfiles>::=  in.<input file>  (.clear)0/1
               out.<output file> (.<output disc> (.<scope>)0/1 )0/1

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

<scope>::= temp ! login ! user ! project
```

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 output file is created by the program, and placed on the output disc, if this parameter is given.
If the output disc is specified the scope of the output file will be either temp or the scope specified.
If an output disc is not specified the name of the output file is looked up with two possible results:

1. A file with this name does not exist on scope temp to project. In this case the output scope will be temp and the disc for the output file is selected according to the most efficient sorting strategy.
2. A file with this name exists on scope temp to project. The name for the output disc and the scope for the output file is fetched from this file.

In any case all files of the name of the output file on scope temp, login, or of a scope not greater than the scope selected for the output file, will be removed with a warning before the sorting starts (or in mdsortproc if it is the input file).

Sortspecification.

Fp-parameters defining the details of the sort.

```
<sortspecification>::=
          block.<input blocklength>(.<output blocklength>)0/1
          <fix or var>.<maxlength>
          ( eof.<eof 1>.<eof 2> )0/1
          ( spill.<yes or no> )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 halfwords). The maximum blocklength is 40 segments.
If only one integer is specified, this value is used as both input and output blocklength.

<fix or var>::= fix ! var ! vnc

Defines whether the records of the input file was created by outrec or by outvar.
vnc means invar without checksum.

<maxlength>::= <integer>

Defines the fixed or the maximum length of a record measured in halfwords.
It must be even, and not less than 2.
It is important for the efficiency of the sort that maxlength is given as accurate as possible in case of variable record length.

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

This parameter is mainly of historical significance and it is not recommendable for new systems.
Normally the number of records contained in the input file should be either supplied by the sq system or given in the catalog entry:
     inputfile=set <segments> <disc> <number of records>

But the end of the input file may for a non sq file be specified 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., halfword 1 to 4 of fixed length, and halfword 5 to 8 of variable length records.
The number of sorted records is inserted in the catalog entry of the output file, and an end of file record is written at the end of the file if eof is used and not sq file.

spill.<yes or no>

With spill.no the standard check of integer and real overflow is switched off. But note that an overflow in the comparison of keys may yield an invalid sorting.

<keyfield>::= <type>.<firstaddr> (.<lastaddr>)0/1 (.descending)0/1

This is the specification of one keyfield.
The order of specification gives the priority of the keyfields.

<type>::= half ! word ! long ! real ! abshalf

The types correspond to the types 1 to 5 in the internal sorting system of rc8000 algol.

<firstaddr> and <lastaddr>::= <integer>

Specifies the position of the keyfield as for a field variable.
The keyfield must be entirely within a maximum length record and only a half or abshalf keyfield may have an odd position.
The keyfield consists of at least one simple field of the type specified. This field will have <firstaddr> as the field address.
If <lastaddr> is used, it must have the value:

<lastaddr> = <firstaddr> + (n - 1) * type_length

where n is the number of simple keyfields and type_length the length of one simple field, 1, 2, or 4 halfwords.
The composite keyfield counts as n in the calculation of number of keys (noofkeys).
The maximum value of noofkeys is 50.

.descending

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

Note on the syntax of the fp-parameters.

The individual parametergroups may occur in any order.
A parametergroup is defined as a sequence of parameters, separated by points.
The last occurrence is used except for the keyfields, where all occurrences and their mutual ordering is significant.
Only the first 3 characters of the keywords are checked, so for example output can be used instead of out and blok instead of block.
For in and descending only the 2 first and the first character is checked, and for out and block the forms ud and segm are allowed.

Variable length records.

The sum check facility of invar is used during the reading of the input file, unless the parameter vnc is used instead of var.
The record length must not excede maxlength, and it must be even.
The record length must not be less than 4 and not less than the position of the first keyfield.
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.

Sq system files.

If the contents key of the catalog entry of the input file is equal to 21, it is supposed that the file conforms to the conventions of the sq file system.
In this case, the block parameter is not necessary, and the eof parameter is irrelevant.
The blocklength and number of records of the input file is supplied by the sq system (opensq).
The output file will be created as an sq file, either with the same blocklength or with the blocklength specified.

Printed output and the execution of the program.

1. The program call is listed on current output.
   Two different errorindications may be printed among the fp-parameters:

<*>	the preceding fp-parameter is illegal.
<*<	the preceding parametergroup is incomplete.

2. Alarms are printed if the accepted fp-parameters are incomplete.

3. The input file is looked up, and the name, the tail of its catalog entry, and its scope is printed.

4. If any errors have been detected up to this point, the run is stopped by a runtime alarm setting the ok-bit false.

5. All files of the name given for the output file which have to be removed before the sorting are looked up and shown in the same way as the input file, just before the removal.
   The first of these files may actually be the input file, it is not removed at this point but in mdsortproc as if the clear parameter had been used.

6. The text: sort start: is printed just before the call of the sorting procedure mdsortproc.

7. The sorting may be stopped by a runtime alarm from mdsortproc.

8. After return from mdsortproc the text: sort ok: is printed.

9. The output file is scoped and looked up.

10. The number of segments and records produced in the output file and the real time and the cpu time is shown.

11. If any errors occurred at 5. or 9. return is performed by a runtime alarm, otherwise the ok-bit will be true.

Alarms.

The possible alarmsources are multiple, but the user errors should be caught by either mdsort or mdsortproc.
These alarms are preceded by the text: ***mdsort alarm: and ***mdsortproc alarm: respectively.

***mdsort alarm:

```
    error in fp-parameters: <n> wrong parametergroups
    in.<inputfile> not given
    out.<outputfile> not given
    block.<blocklength> not given
    fix, var or vnc.<maxlength> not given
    no keyspecifikation
    more than 50 keyfields
    negative number of records from tail(6)
    inputfile does not exist
    no resources for scope of outputfile
    monitor procedure: <n> result: <r>
```

These alarms are supposed to explain themselves.
The last one concerns peculiar results from calls of monitor procedures and should not occur.

***mdsortproc alarm:

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.
remove	monitor result	input file cannot be cleared.
size	-lacking halfw.	the size of the job must be increased so much before mdsortproc will do the sorting.
disc	-lacking entr.	too few catalog entries in main catalog.
	segments	not enough segments for one work file of this size.
out disc	-1	the output disc is not mounted.
	segments	not room for file of this size on the output disc.
integer...trap	passnumber	normally spill in key comparison.

In addition alarms from opensq or stderror may occur if file or record 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, specifying the number of input records accepted before the error was detected.

3. MDsortproc, User,s Description

Purpose.

Mdsortproc, merge_disc_sorting_procedure, is a procedure, intended for fast sorting of one backing storage area.
The procedure can be called from a program coded in algol or fortran for RC 4000, RC 6000, and RC 8000.

Function.

The procedure sorts a backing storage file holding records of either fixed or variable length, using backing storage throughout.
The basic sorting method, is the merge technique:
Sorted strings, as long as possible, are generated by internal sorting during the first reading of the input file.
After that, these strings are merged repeatedly until only one sorted string is left.

The procedure will try to minimize the sorting time by variation of mergepowers, blocklengths, use of single- or double-buffering, and by utilization of two disc-stores, if available.
The procedure needs in total a backing storage area of about twice the size of the data to be sorted.
It can be specified that the input file shall be cleared, so its area can be used for the merge.
The free core, when the procedure is called, must be more than about 10000 halfwords, depending on the blocklengths and recordlengths specified.
The value of 10000 is valid for blocklengths up to 2 segments.
The sort can use any amount of free core to speed up the sorting and room for a work file on two different discs will reduce the time for input output.

Call.

mdsortproc	(param, keydescr, names, eof, noofrecs, result, explanation)
param(1:7)	(call value, integer array) This array holds various parameters of type integer and type boolean, describing the files and the records.
param(1)	segsperinblock. Blocklength of the input file, given as a number of segments. 1 <= segsperinblock <= 40. Supplied by sq system if contents = 21.
param(2)	clearinput. 1: The input file is cleared, and its area can be used for the merge. 0: The input file must not be cleared.
param(3)	segsperoutblock. Blocklength of the final output file, given as a number of segments. 1 <= segsperoutblock <= 40. If param(3) = 0 , segsperoutblock:= segsperinblock.
param(4)	fixedlength. 1: Fixed recordlength. Inrec6/outrec6 are used. 0: variable recordlength. Invar/outvar are used. 2: variable recordlength but no checksum.
param(5)	maxlength. The maximum length of variable length records, and the length of fixed length records, measured in halfwords. Maxlength >= 2 and maxlengh <= segsperinblock * 512 and maxlength <= segsperoutblock * 512. Maxlength must be even. It is important for the efficiency of the sort that maxlength reflects the real maximum length of variable length records.
param(6)	noofkeys. The number of keyfields in the sorting key. 1 <= noofkeys <= maxlength and <= 169.
param(7)	concerns the reaction on resource troubles. 0: Resource troubles will not stop the execution, the procedure returns with result > 1. <>0: Resource troubles causes runtime alarm.

keydescr(1: noofkeys, 1:2)	(call value, integer array) The description of the sorting key. Keyfield n is specified as: +/- type, position, in keydescr(n, 1:2). The type ranges from 1 to 5, indicating: signed halfword, integer, long, real, or abshalfword. The sign of the type specifies the sequencing: + for ascending, and - for descending order. The position of the keyfield is specified as the number of the last halfword in the field, as for algol field variables. The position may not exceed 2047. The entire keyfield must be within a maximum length record. The length of variable length records must not be less than the position of keyfield 1, the highest priority keyfield. records having equal values in all keyfields are sorted according to their occurrence in the input file, i.e. their mutual order is not changed. In connection with very long records (maxlength= param(5) >= 2046) the facility is switched off.
names(1:6)	(call and return value, real array) Contains 3 file and disc names.
names(1:2)	inputfile. The name of a backing storage area. The procedure assumes that the size of the area reflects the amount of data to be sorted.
names(3:4)	outputfile. If names(3) = real<::> then the name of the outputfile is returned in names(3:4), otherwise the name given is used for the final output file. An existing file of this name on scope temp is cleared without warning, just before the end. The sort is not able to use the resources of such a file.
names(5:6)	outdisc. If names(5) = real<::> then the output disc is selected according to the most efficient strategy, otherwise the disc specified is used.

eof	(call value, real) If the parameter noofrecs is negative, then the end of the input file is indicated by a record holding the bitpattern given by eof in the first 4 halfwords of the userpart. Halfword 1 to 4 in case of fixed length and halfword 5 to 8 in case of variable length records. The final output file is terminated by an end of file record of maximum length in this case.
noofrecs	(call and return value, integer) If noofrecs is non negative, then the number of records in the input file is given by the value of noofrecs and an eof record is not created. The number of sorted records is returned in any case in this parameter. with an sq file noofrecs <= 0 means that noofrecs is supplied by the sq system, noofrecs > 0 means that only this number of records are to be sorted.
result	(return value, integer) The value of result specifies the result of the call of the procedure. In general, resource problems will yield a result different from 1, whereas errors concerning the parameters or hard errors will stop the execution by a runtime alarm. If param(7) <> 0 only result = 1 will occur, the other results are transformed to alarms.
explanation	(return value, integer) The value of this parameter should give a further explanation of result. See the next section.

Sq system files.

If the contents key of the catalog entry of the input file is equal to 21, it is supposed that the file conforms to the conventions of the sq file system.
The output file will be created as an sq file as well.

Results.

result	explanation	comment
1	segments output	the sort was ok
2	-lacking core halfwords	not sufficient core
3	see alarm disc	not sufficient backing storage
4	see alarm out disc	backing store specified by names (5:6) does not exist or has too few resources.

Results > 1 are only given if param(7) = 0.

The parameter noofrecs will contain the number of sorted records if result = 1 , otherwise it is unchanged.
The output file will, provided result = 1, be cut to the minimum size, and tail(6) of the catalog entry will contain the same value as noofrecs, if not sq file.

Requirements.

The available amount of core storage before the call of the procedure must satisfy the condition:

```
  free_half_words
              > 7000
              + 512*(segsperinblock + segsperoutblock)
              + 4*maxlength + 24*noofkeys.
```

The procedure requires as working areas two disc files of the size of the input file.
This means in the case when the input file is removed that the procedure must be able to create one work file of that size.

If the input file is not removed the procedure must be able to create two work files of the size of the input file.
If an output disc is specified this disc must be able to hold the final output file.
already at the beginning of the procedure it is checked that the output disc is capable of holding a file of the size of the input file. (this is of course the case if the inputfile is placed on the disc specified and has to be removed).
the blocking of records may be changed by the sorting, so the output file may have a greater size than the input file.
work files are kept at minimum size by concatenation of the records without regards to block limits.

The procedure needs 2 catalog entries, 2 area processes, and 1 message buffer. So, the job process should at least be the owner of 4 area processes, and 2 message buffers.
But it is recommended to have a greater number of message buffers, (10 to 20), especially in the case of a sort of small records, (about 2 to 20 halfwords), with great core size.

Variable length records.

The sum check facility of invar is used during the reading of the input file if param(4) = 0 (invar with checksum control).
The record length must not exceed maxlength.
The minimum record length is given by the greatest of the two values: 4 (if noofrecs < 0 then 8) and keydescr(1, 2).
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.

Alarms.

Parameter errors and hard file errors will stop the run with a run time alarm.

alarmtext	integer	comment
param	param number	error at param(param number)
keyfield	keyfield	illegal position or type of keyfield <integer>.
create	monitor result	abnormal result in call of the monitor procedure create entry.
lookup	monitor result	abnormal result from lookup entry. This alarm will normally indicate that names(1:2) does not specify a catalog entry.
change	monitor result	abnormal result in call of change entry. Should not occur.
rename	monitor result	it is impossible to rename the final output file to names(3:4).
remove	monitor result	abnormal result in call of remove entry. The alarm will normally concern the original input file.
infile	tail(1)	names(1:2) does not point to a catalog entry describing an area.
r.length	record length	variable length record of a length greater than maxlength, less than keydescr(1, 2), or less than 8 if eof used.
passes	20	the sort could not be done in 20 merging passes. This alarm should never occur.
reccount	record count	this is a hard error or a programming error. The counts of records in the first pass and the last, are not equal, the last count is shown.
size	-lacking halfwords	not enough core.
disc	-lacking entries	too few entries in main catalog.
	segments	not enough segments, the value of segments is the size of one workfile.
out disc	-1	the wanted output disc is not mounted.
	segments	not room for one workfile on the wanted output disc.
trap	passnumber	normally spill in key comparison.
nrecs sq	noofrecs	param noofrecs > records in sq file.

The alarms: size, disc and out disc (resources) will only be given in case param(7) <> 0, otherwise the corresponding results, 2 to 4, with explanation will be given.
In addition, index alarms may occur, if the parameter arrays are incorrectly declared, and alarms from opensq or stderror may occur, if file or record formats are illegal or in case of hard errors.
Alarms, with the exception of index alarms, are preceded by the text, ***mdsortproc alarm:.

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

A. REFERENCES.

1. RCSL NO: 42-I 0781, January 1979.
   Algol 7, User's Manual, Part 1.

2. RCSL NO: 31-D 561, April 1979.
   RC8000 SQ-SYSTEM.

3. RCSL NO: 55-D 66, January 1970.
   Sorting in Algol 5.

RETURN LETTER

Title: RC8000 Backing Storage Area Sorting mdsort - mdsortproc   RCSL No.: 31-D562

A/S Regnecentralen af 1979/RC Computer A/S maintains a continual effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback, your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

Do you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Name: ____________ Title: ____________

Company: ____________

Address: ____________

Date: ________

Thank you

. . . . Fold here . . . .

. . . . Do not tear - Fold here and staple . . . .

Affix
postage
here

REGNECENTRALEN
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark