QSORT -- Version 3.20

Text File Sorting Utility

Table of Contents

Introduction

QSORT was first designed to be a replacement for, and to overcome the limitations of DOS SORT, but has been enhanced a number of times and moved to new compilers twice. The current version will sort files whose size is limited only by available disk space. File name(s) may be given explicitly or QSORT will sort from standard input to standard output, and so, may be used in pipes or with redirection. Multiple keys may be specified. Binary files with fixed-length records may be sorted, provided only that keys are ASCII character strings.

QSORT tries to be very protective of your data. If QSORT has an error of any kind, it will terminate with the input file still intact, and will return to DOS with a non-zero ERRORLEVEL. When QSORT successfully completes a sorting a file, it terminates with ERRORLEVEL set to zero.

The command line syntax is a super-set of SORT's, so QSORT may be used without other changes in batch files using SORT, but in most cases you will probably want to make use of QSORT's greater capabilities.

Notation

In defining the command line and its various parameters, the following notation is used:

[optional] items are enclosed in square brackets.

<variable> items appear in lower case and are enclosed in angle brackets. They are replaced by actual data such as a file name. The angle brackets are NOT typed.

THIS | THAT Choices are separated by a vertical bar. Select one or the other but not both.

[THIS | THAT] When the choices are enclosed in square

brackets, you may also select neither.

REPEAT. . . The ellipsis (. . .) means the item to its left may be repeated as many times as necessary.

UPPER CASE items and all special characters not defined above represent themselves. They are entered exactly as they appear.

The QSORT Command and Options

QSORT is invoked with the following command:

```
QSORT [<in_file> [<out_file>]] [/<key_spec>. . .]
     [/F<len> | /T[<tag> | /D[[<fields>][<delim>[<term>]]]
     [/R] [/S[V]] [/?]
```

Note that all parameters on the command line are optional. The <in_file> and <out_file> parameters are "ASCII-Z" file specifiers. They may contain disk and path information in the standard DOS format, but must not contain "wild-card" characters. If <in_file> is missing, QSORT sorts from standard input to standard output. These are files defined and opened by DOS before QSORT is loaded. (See your DOS manual concerning the use of redirection and pipes.)

If <in_file> is given but <out_file> is missing, QSORT creates a temporary file in the directory containing <in_file> and sorts to the temporary file. On successful completion of the sort, <in_file> is deleted and the temporary is renamed to <in_file>. The effect is an apparent "sort-in-place."

If both file names are given, <in_file> is unchanged and the sorted output is written to <out_file>. Note that the following two commands are exactly equivalent:

```
QSORT  FILE.TXT  FILE.SRT

QSORT <FILE.TXT >FILE.SRT
```

In the first, QSORT opens the files. In the second, redirection is specified and DOS opens the files. The result is the same. It is an error QSORT can't detect if you mix these. For instance:

```
QSORT  FILE.TXT >FILE.SRT
```

will result in a sort-in-place. QSORT will open FILE.TXT but won't know DOS has opened FILE.SRT for it, and will ignore it.

The /<key_spec> Parameter

Up to 30 /<key_spec> parameters may be used to specify sort keys and are ordered major to minor from left to right. The /<key_spec> argument has the form:

```
/[L][+|-][<field>.][<col>][:<length>]
```

Note that all elements of this argument are "optional," but at least one element must be present following the slant-bar (/).

The 'L', if present, specifies "lexical" sequence for this key. Lexical sequence is ordered first by spelling, then, when keys have identical spelling, by capitalization.

The minus (-) sign reverses the sorting order for this key, while the plus (+) sign (or no sign) specifies normal sort order.

There are three numbers associated with every sort key; the field number, the starting column within the field, and the length of the key in characters. Any, or all of them may be given in a /<key_spec> parameter. QSORT uses punctuation to identify each number. A number followed by a period (.) is a field number. A number preceded by a colon (:) is a length number. A column number has no punctuation associated with it. It follows the field number, if any, and precedes the length number, if any.

The [<field.>] element is used only for "delimited-field" records, and locates this key within a particular field. The value of <field.> must be less than or equal to the number of fields defined with the /D parameter (see below). If [<field>.] is omitted when sorting delimited-field records, the first field is assumed. For consistency, all records are assumed to have "fields." In all cases except delimited-field records, there is precisely one field, and it spans the entire record.

If present, [<col>] defines the beginning column of the key. If omitted, column 1 is assumed. In the case of delimited-field records, column 1 is the first character of the identified field. In all other cases, column 1 is the first character of the record.

If present, [:<length>] defines the key length in columns (or characters). If [:<length>] is omitted, the rest of the record, or field in delimited-field records, is assumed to be part of the key.

If no key parameters are given, the entire record, or the entire first field is the key.

When sorting variable-length records, any key which begins beyond the end of its field in a particular record is treated as a null (zero length) key for that record, and will sort low relative to all records with non-null values for that key. When sorting fixed-length records, all defined keys must fall within the defined record length. <key_spec> parameters must appear in order of importance, primary key first.

The /F<len> Parameter

The /F<len> parameter denotes the record length for a file of fixed-length records. All records in the input file MUST be exactly <len> bytes long. The records need not (but may) be terminated with a CR/LF sequence. They may contain any data, even binary data, but the keys must be ASCII strings. Strings may be

terminated with a null (binary zero) character, or may be padded with trailing spaces to the full length of the key.

Note that QSORT does not attempt to support Pascal style strings. These are strings which begin with a character whose binary value is a character count. This is followed by <count> characters of ASCII data, which in turn is followed by random data out to the maximum length of the string. These strings may be used as keys, but the programmer must insure that either the last real character is a null character, or the key is padded to its full length with spaces. QSORT must be told that the key begins in the second character position (the first character of real data).

The /T[<tag>] Parameter

The /T[<tag>] parameter, if present, indicates that the "records" to be sorted may be more than a single line long.

If <tag> is also present, it defines a character to be used to tag the "end-of-record." If <tag> is not present, the first empty line terminates the record. For this purpose, "empty" means "no characters." A line containing but a single space is NOT empty! A line may be "tagged" by placing the <tag> character anywhere on the last line of a logical record. The entire line,

including the tag character will appear as the last line of the record.

Some characters cannot be used to represent themselves in a DOS command line. For that reason, QSORT uses codes to represent them. These codes are actually a pair of characters. The first is always a back-slash (\). The second character identifies the special character it represents. The following is a table of characters recognized by QSORT:

\B - Space character	20 Hex	32 Decimal
\F - Form feed character	0C	12
\L - Line feed character	0A	10
\N - Newline sequence		
\R - Carriage return	0D	13
\T - Tab character	09	9
\\ - Back-slash character itself		

Thus an invisible tab character might be used to end a multi-line logical record. The other characters in this code list don't make much sense in this context, but will be useful in the /D parameter (see below).

Note that the /F<len> and /T[<tag>] parameters are incompatible, and may not both be specified.

The /D[<fields>][<delim>[<term>]] Parameter

The /D[<fields>][<delim>[<term>]] parameter, if present, states that this file contains delimited-field records. In other words,

a record is made up of distinct, variable length fields separated from one another by a particular character, or "delimiter." Records are separated, or "delimited" by the "newline sequence."

The <fields> element defines the number of variable length fields contained in each record. All fields must be present in each and every record. A "null" field will be represented by two successive delimiter characters. There must always be exactly <fields> minus 1 delimiter characters in a record.

If a <delim> character is present, QSORT uses it as a field delimiter character. Otherwise a comma (,) is assumed to be the delimiter.

If a <term> character is also present, QSORT uses it as a record delimiter character. In fact, it literally redefines the "newline sequence" to QSORT. More on this in a moment.

The same character codes listed under the /T parameter may be used to represent these characters. Note that "\N" means the "newline sequence." If <term> is not present, this is the CR-LF character pair. If <term> is present, it represents the <term> character. Thus:

```
/D3\N\T
```

says that the records are separated by tab characters, and that the three fields within each record are also separated by tab characters. On the other hand:

```
/D3\N
```

says that each group of three lines constitutes one logical record, and each line is a field within that record.

The /D parameter is always incompatible with the /F parameter, and usually incompatible with the /T parameter, but there is an exception when <fields> is missing, or is equal to 1.

If <fields> equals 1 (or is missing) it says that there is only one field spanning the entire record. But that is what QSORT assumes if the whole /D parameter is missing! So why bother?

In most ASCII files a "line" ends with a carriage return character (CR) followed by a line feed character (LF). QSORT searches for this character pair when it is looking for a "newline sequence."

But not all files use CR-LF as a line terminator. For instance, files imported from UNIX or XENIX usually terminate lines with a

naked line feed character! And some editors produce files whose lines end in a naked carriage return character! So:

```
/D,\L
```

says "for this file, the newline sequence is a single line feed character." In this case, the comma is a place holder. There really is no "delimiter character," but one must be present in the parameter in order to define the <term> character.

The /R Parameter

The /R parameter is included for compatibility with DOS SORT and is redundant. It reverses the sense of sort direction for all sort keys.

The /S[V] Parameter

The /S parameter tells QSORT to make a statistics report to the screen at the end of a run. The report is written to the "standard error" device, the console, and may not be redirected. The following is an actual statistics report "cut" from the screen after QSORT had sorted a 1.3+ megabyte file:

```
-----------------------------------------------------------------
QSORT - Text Sort - Version 3.20
Copyright (c) 1985, 1986, 1987  - Ben Baker - All rights reserved

     12115 records sorted
       150 bytes in longest record

    127131 sort phase comparisons
     73232 merge phase comparisons

    200363 total comparisons
      16.5 comparisons per input record

        27 temporary merge files created
         2 merge passes
       2.4 average passes over data

      2:51 elapsed time
-----------------------------------------------------------------
```

The first two numbers are self-explanatory. The next two are the number of times two records were compared during the sort phase and the merge phase respectively, followed by the total comparisons.

The next number is total comparisons divided by the number of records in the input file. If there is no merge phase, this number is typically 10 to 12. If the file is large enough to require merging, it is 12 to 20, on average. If it is much larger than 20, it usually means that there is something unusual about

your input file. It may already be sorted, or there may be large blocks of records which compare equal. This can happen if you sort on, say column 50 and the input file contains a large number of records shorter than 50 bytes. In this case, a minor sort key at column 1 may significantly speed sorting.

The next two items are self-explanatory. "Average passes over data" reflects the number of times each record was read and written. For short files not requiring a merge pass, this number will be 1.0. When merging is needed, the last merge pass is the one which writes the output file and it must read and write every record exactly once. Thus when only one merge pass is made, there will be exactly 2.0 "average passes over data." In the above case the first merge pass processed about 40% of the records, hence the value of 2.4.

The above sort was performed on an Zenith 248, an eight megahertz AT clone with two hard drives (C and D). The input file was on C; the temporary merge files were placed on D; and the output file was written to C. The sort of a 1.3 megabyte file took under three minutes. The same sort on an XT should take about seven minutes.

The optional subparameter, [V] (for verbose), causes the QSORT program to make running progress reports to the screen, as well. Each pass during both the sort phase and the merge phase (if any) issues a 1-line report telling the merge file(s) and the number of records being processed during that particular pass. This is not terribly useful for short files, but for the big ones, it can give the user a warm comfortable feeling that something is actually being done.

The /? Parameter

The /? parameter requests help or parameter evaluation. When QSORT is executed with the /? parameter alone, it lists a short description of the QSORT parameters. If /? is entered as one of several parameters, QSORT will produce a short report on the screen describing the sort it would perform based on those parameters without actually doing a sort.

For example:

```
QSORT /? /L5:12 /-3:2 /22 /T /R <INFILE.TXT >OUTFILE.TXT
```

requests a sort on a file of tagged records. It defines three sort keys, one of them inverted (-). The /R parameter reverses the sense of all three keys. Since redirection is specified, QSORT does not see or know about the names of the files it will sort. The /? parameter requests a report, rather than a sort, and QSORT obligingly produces on the screen, the following:

```
------------------------------------------------------------------
QSORT - Text Sort - Version 3.20
Copyright (c) 1985, 1986, 1987  - Ben Baker - All rights reserved

With the present arguments, QSORT would sort from STDIN to STDOUT
Records are multiple lines ending with an empty line

Key fields in descending order of importance are:
  Field  Pos   Len Type

      1    5    12 Lexical Descending
      1    3     2 ASCII
      1   22 65535 ASCII   Descending
------------------------------------------------------------------
```

The third key has an unspecified length. The value "65535" merely means that this key extends to the end of each record.

The /M<len> supported in earlier versions of QSORT is no longer required, but will be accepted (and ignored) by QSORT. There is no "hard-coded" maximum record length in QSORT, but there is a practical limit. At some time during every sort, the two longest records in the input file must be compared. Therefore, the two longest records must be able to fit together in the sort buffer. The sum of their lengths cannot exceed about 50K -- not an altogether unreasonable limitation. QSORT can be shoe-horned into tighter memory and will run if it can find 4K for a sort buffer and 4K for an output buffer, but the two longest records must still fit in the sort buffer together.

Arguments may appear in any order on the command line except that <in_file> must appear before <out_file>, and /<key_spec> arguments must appear in descending order of importance.

Lexical Sorting

The lexical sorting capability was borne out of my own need to sort word lists with mixed capitalization. ASCII sequence produced some bizarre results when words beginning with 'Z' sorted before those beginning with 'a.' Case-insensitive sorting wasn't much better because upper and lower case got mixed randomly.

The following table will illustrate what I mean:

INPUT	ASCII	CASE INSENSITIVE	LEXICAL
DeLaPort	Baker	Baker	Baker
Smith	Brown	brown	Brown
brown	DeAngelo	bRown	bRown
deLaPorte	DeLaPort	Brown	brown
Deangelo	Deangelo	Deangelo	DeAngelo
deAngelo	Deangelo	deangelo	Deangelo
Brown	DelaPort	Deangelo	Deangelo
smith	DelaPorte	deAngelo	deAngelo
delaPorte	Harry	DeAngelo	deangelo
DelaPort	Smith	delaPort	DeLaPort
DeAngelo	bRown	DelaPort	DelaPort
DelaPorte	brown	delaPort	delaPort
deangelo	deAngelo	DeLaPort	delaPort
Harry	deLaPorte	DelaPorte	DelaPorte
delaPort	deLaPorte	deLaPorte	deLaPorte
Baker	deangelo	delaPorte	deLaPorte
deLaPorte	delaPort	deLaPorte	delaPorte
Deangelo	delaPort	Harry	Harry
bRown	delaPorte	smith	Smith
delaPort	smith	Smith	smith

The first column is a list of names in arbitrary order. The second is an ASCII sort of that list. Third we have one possible case-insensitive sort of the list. The fourth column is what I really wanted. It is sorted the way these words would be sorted in a dictionary (or lexicon). The third and fourth columns both collect words of identical spellings together, but in the third column, upper and lower case spellings are in arbitrary order, while the fourth column places upper case spellings ahead of lower case spellings.

For example, the two occurrences of Smith are widely separated in column 2 because one is capitalized and the other is not. Column 3 brings the two together, but in the wrong order. They might have been in the right order, but the order is strictly arbitrary. In column 4, Smith comes before smith, and lexical sorting will always put them in this order.

Lexical sorting is achieved by making case-insensitive comparisons of entire keys. If the keys compare equal, an ASCII comparison is made to arbitrate ties. In other words, when "lexical" keys in two records have different spellings, the case-insensitive comparison determines the order of the records. When "lexical" keys are spelled the same, the case-sensitive comparison determines the order of the records.

Lexical keys are defined, as indicated above, by placing the letter 'L' immediately following the slant-bar (/) in <key_spec> definitions.

Lexical sorting can be very useful when needed, but be aware that unnecessarily specifying lexical ordering may degrade performance of QSORT.

Examples

Produce a sorted directory listing and display it on the console a screen's worth at a time:

```
A>DIR | QSORT | MORE
```

This demonstrates the use of QSORT as a "filter" in a "pipe."

Produce a directory listing sorted by creation date and time, and display it on the console a screen's worth at a time:

```
A>DIR | QSORT /30:2 /24:5 /39 /34:5 | MORE
```

The output of the DIR command is piped to QSORT. The keys defined are, from left to right (major to minor), year (2 digits), month and day, AM/PM flag and time. The output of QSORT is then piped to MORE for display.

Next, replace the unsorted FILE.TXT with the same data sorted in reverse order. Use columns 10 to 16 as the sort key:

```
>QSORT FILE.TXT /-10:7
```

or

```
A>QSORT FILE.TXT /10:7 /R
```

or (SORT compatible)

```
A>QSORT FILE.TXT /R /+10
```

Next, perform a simple sort on a file with up to 240-byte records

```
A>QSORT LARGE.REC /M240
```

or

```
A>QSORT LARGE.REC
```

Note that the "/M240" parameter is no longer needed, but will not hurt.

GLOSS.TXT is an unsorted glossary of terms. The term being defined by each entry appears first, followed by several lines of definition. The entries are separated by empty lines. Produce GLOSS.SRT, a sorted version of the glossary:

```
A>QSORT /T  <GLOSS.TXT >GLOSS.SRT      (with redirection)

     or

A>QSORT /T   GLOSS.TXT  GLOSS.SRT      (without redirection)
```

A lawyer keeps a running log of his billable activities in TIME.LOG. The first line of each entry is "mm/dd/yy hh:mm account#." He always places a tilde (~) in the last line of each entry. He wishes to sort the log by account number, and by ascending date and time within each account:

```
A>QSORT  /16:7 /7:2 /1:5 /10:5 /T~  TIME.LOG
```

The directory of users for a bulletin board system is kept in a binary file of fixed-length records 180 bytes long. The user name is a 26-character field beginning in the first position and the city/state field is a 16-character field beginning in the fortieth position. Sort the file by city/state and name.

```
C>QSORT /F180 /40:16 /1:26 USER.BBS
```

DB.TXT is a delimited field output file from dBASE III, Each record contains 7 fields, delimited by commas. Sort the file to the screen using field 3 as a sort key.

```
C>QSORT /D7 /3. <DB.TXT
```

Here, "standard input" has been redirected to the file. Since no

redirection is given for "standard output," DOS assigns it to the console by default. This is NOT a sort-in-place!"

You have received a member list from the Society of End-users of XENIX (SEX.LST). Sort the list by special interest (10 columns beginning at 70) and name (30 columns beginning at 1). Note that the file contains no carriage return characters. Since SEX.LST

is a very large file, we wish to obtain running status reports and a final statistics report.

```
C>QSORT SEX.LST /70:10 /1:30 /D,\L /SV
```

The /D parameter is used to redefine the newline sequence as a naked line feed character.

The file LABEL.TXT contains mailing label images. Each label is 6 lines (1 inch) high. Line six is always empty and line three is frequently empty. An extended Zip code always begins in column 20 of line 5, and extends to the end of the line. In order to take advantage of bulk mailing rates, the labels must be sorted into carrier route (CRT) order.

```
QSORT LABEL.TXT /5.20 /D6\N
```

We must use a "delimited field" sort rather than a "tagged line" sort for two reasons: 1) Line six is empty, not tagged with a special character. When line three is also empty a label would be broken into two pieces and separated by the sorting process. 2) Our sort key is not at any known offset from the beginning of the label. Its position is fixed only relative to line five.

Error Messages and Return Codes

The QSORT program can encounter a number of different errors during execution. Each will generate a brief error message on your console. This section will attempt to list the messages you may see, and give you a little more detailed information about what might have caused the problem.

Command Line Errors

The most common causes of error messages are errors in the command line parameters. Particularly when using a complicated set of keys, I recommend the use of "/?" as the last parameter. If QSORT discovers an error, it will be reported. the QSORT program will also show you exactly what it would have done, had the "/?" parameter not been there, but will NOT perform a sort.

You may then hit the "F3" key to recall the command, edit any bat parameters using the left and right cursor keys and the "INS" and "DEL" keys. When the command parses without error, and the report looks like the kind of sort you wish to make, hit the "F#" key once more, then back space over the "/?" parameter, then hit "Enter" and QSORT will do the rest.

A frequent cause of command line errors is the use of multiple parameters without separating them with spaces. "/5:3/S" is wrong. "/5:3 /S" is right.

One or more of the following errors might be encountered in the command line:

Three file names specified

At most, only two file names may be given, an input file and an output file. The most likely cause of this message is forgetting to use the "/" character at the beginning of a key spec or other parameter.

Invalid command line parameter "<parameter>"

This message is issued if QSORT receives a parameter it does not understand. It is usually a typographic error. You meant "/D" and hit "/E" by mistake. The message displays the actual parameter it did not understand.

/D, /F and /T parameters are incompatible

Each of the above parameters tells QSORT to use a different scanning routine to parse records. Since only one such routine can be used, it is an error to use more than one of these parameters. In those unusual situations where more than one might apply, use the most efficient one. (The order of the parameters in this message is from most efficient to least efficient. See the section on "Performance and Input Record Type" for more informa-

tion.)

Multiple /<parameter> parameters encountered

This message again applies to the /D, /F and /T parameters. In this case, the same parameter appears twice in the command line.

/F<length> parameter with invalid <length>

No substitution is made for "<length>" in this message. This is the actual message displayed. It means that either there was no length specified, or the specified length was zero.

Keyfield "<key_spec>" begins beyond end of record

Keyfield "<key_spec>" extends beyond end of record

These two messages refer to fixed-length records. A key specification has told QSORT that data exists beyond the bounds of the record. For instance, suppose that /F20 has been specified. Then /23 would invoke the first message because the record is only 20 characters long. Similarly /18:5 begins before the end of the record but extends beyond it, and would invoke the second

message. Note that /18 is OK. QSORT will assume a length of three in this case.

Invalid delimited field specification - "<key_spec>"

This one is similar to the previous messages. The "field number" portion of a key specification was greater than the defined number of fields. For example "/D5 /6.1:3" would provoke QSORT into issuing this message. It's hard to find field 6 in a 5-field record.

ABORT -- Error(s) in command line parameter(s)

If any of the above messages are issued, QSORT will continue to scan the command line and evaluate the parameters, but will eventually issue this message too. If there are command line errors, QSORT will NOT guess about your data. It will stop!

Memory Errors

ABORT -- Buffer allocation error

An error of unknown origin occurred when QSORT was trying to allocate memory for its buffers. The most likely cause here is a

"memory poor" condition caused by a too small partition under a multitasker such as DoubleDOS, or perhaps too many "terminate-and-stay-resident" programs. As an absolute minimum, QSORT must be able to obtain eight kilobytes of contiguous memory for its sort buffer.

ABORT -- Insufficient memory

This one can occur at any time during the sort. QSORT must have a sort buffer large enough to hold the two largest records in the file. Typically, the sort buffer is about fifty kilobytes, which means that if records are shorter than about twenty five kilobytes, QSORT can usually handle them. This is normally a problem only when using the /T parameter.

I/O Errors

ABORT -- Unable to open "<file_spec> for input"

QSORT was attempting to open <file_spec> for input. If <file_spec> is your input file, you probably misspelled the name. If <file_spec> has the form "number.SRT" QSORT could not find a merge file it thought it had created. If this happens you may have discovered a bug. Please send me full particulars ASAP!

ABORT -- Unable to create "<file_spec>" for output

QSORT was attempting to open <file_spec> for output, and the open operation failed. The most likely cause is that you ran out of disk space, and DOS was unable to expand a subdirectory. A root

directory cannot be expanded, and you may have run out of directory space.

ABORT -- Error reading input or merge file

The section of the program which issues this message does not know the file name, so cannot help you much there. This message may mean that your disk has a sector going bad. (Well, it can't all be good news!)

ABORT -- Error writing to merge or output file

This one could also mean a bad sector, but a far more likely cause is that you just ran out of disk space.

Internal Errors

ABORT -- Internal QSORT error

In theory, this is an error which "can't happen." If you EVER get this message, please notify me with as many details as you can supply. Actually I have NEVER seen this message issued by a released version of QSORT.

ERRORLEVEL Return Codes

When QSORT successfully completes a sort, it terminates with DOS ERRORLEVEL set to zero. (See your DOS manual for more information on ERRORLEVEL.) If it terminates for ANY other reason, it sets ERRORLEVEL to a non-zero value, which can be tested in a batch file. The following are the ERRORLEVEL codes QSORT uses, and their meanings:

Code	Meaning
0	Successful completion
1	Command line error and/or "/?" parameter specified
2	Open-for-read error
3	Open-for-write error
4	I/O error reading file
5	I/O error writing file
6	Memory error
255	Internal error

Implementation Notes

General Information

QSORT is intended as an enhanced replacement for DOS SORT. It is nearly fully upward compatible, but provides much more flexibility. Multiple sort keys may be specified, a pseudo in-place sort may be performed and files and/or records of any size may be sorted provided only that there is sufficient disk space for work files and the output file. QSORT uses the "quick sort" algorithm, which cannot guarantee the order of records whose keys are

all equal. This is the one "incompatibility" with DOS SORT, which retains the original order of records when its only key compares equal. This is important to SORT because it must be invoked multiple times to effect a multiple key sort. With QSORT, you only sort once and there are usually enough keys available to insure you get the order you want the first time.

QSORT uses a sort buffer of about 50K bytes and will fill the buffer as full as possible, and then sort its contents. If the end of the input file has been reached and no temporary work files have been generated, the sorted contents of the buffer are written to the output file, completing the sort operation.

If the input file is too large to fit into the sort buffer, as much of the input file as possible is read into the buffer, sorted, then written to a temporary work file. This process is repeated as many times as necessary to process the entire input file, each time creating a new work file for the sorted output.

Upon completion of the "sort phase," QSORT begins a "merge phase." Each work file is a sorted sub-set of the input file. Thus, work files may be read sequentially and combined to produce a sorted output. QSORT will open as many work files as DOS permits (more on this later). If all the remaining work files can be opened, the sorted result is written to the output file. Otherwise, a new work file is created and another merge pass will be required. On each merge pass, the number of work files is reduced and eventually all remaining work files will be opened and the sorted output file will be written completing the sort operation.

Performance and DOS Configuration

QSORT is smart enough to never have just one work file remaining, which would require an unnecessary copy operation. In fact, QSORT is smarter than just that in its handling of the merge phase. If more than one merge pass is required, all the data merged during the first pass will have to be merged again, so QSORT attempts to minimize the first pass. For example, if QSORT discovers it may only open 15 files at a time, and there are 16 temporary files, it will only merge two files on the first pass, creating a 17th file as it does. Then in the second pass, it

will merge all 15 remaining files to the output file. The less data it processes twice, the faster it performs the sort!

With nothing else to guide it, QSORT places its temporary files in the default directory. Either of two "environment variables" can override this. (See your DOS manual for information on envi-

ronment variables and the SET command.) The DOS command:

```
SET QSTMP=<path>        or

SET TMP=<path>          or

SET TEMP=<path>
```

will define a path for QSORT to use for its temporaries. QSORT first looks for the environment variable QSTMP. If it does not exist, QSORT next looks for TMP or TEMP in that order. TMP and TEMP are de facto standards used by many programs, and are usually defined in your AUTOEXEC.BAT batch file. You might have TMP specifying a 64K RAM disk to speed up your compiler. In this case, an attempt to sort a 100K file is doomed to failure. Rather than redefine TMP, you may define QSTMP to force QSORT to use some directory on your hard disk. In fact:

```
SET QSTMP=\
```

tells QSORT to always use the root directory of the default drive!

> CAUTION! The root directory has a fixed size, and is NOT expandable. For a hard disk, it typically has room for only 512 file names, less one for each subdirectory and one for the volume label (if any). Large files may fail to sort if the QSORT program must place too many merge files in a root directory. Subdirectories, on the other hand, are limited only by available disk space.

QSORT, to work properly, needs enough space on the output disk to hold the output file. Even if the input file is to be deleted and resides in the same directory, that is not done until after the output file has been successfully written. If one merge pass is required, the disk space QSORT uses for temporary merge files will be about 10% larger than the size of the input file. If more than one merge pass will be required, allow about twice the size of the input file as temporary merge file space.

One of the advantages of controlling where QSORT places its temporary files is to insure adequate space for them. A second is speed. If the temporary files can be placed on a separate disk from the input and output files, disk seeking is minimized and performance improved.

Each time QSORT must create a new temporary merge file, the data put into it will be processed again. Obviously, the more files QSORT can open during the merge phase, the fewer times it will have to handle each record and the faster it can sort large files. If DOS is properly pre-conditioned, QSORT can have up to 15 temporary merge files open at once, and very large files can be sorted with just one sort pass and one merge pass. Unfortunately, that capability is not automatic.

DOS has a fixed number of file "handles" that it associates with open files. The default number is eight, but DOS opens five of them for standard input, standard output, standard error, standard printer and standard auxiliary device. That leaves three for merging. A 250K input file would produce five temporary merge files and that would take three merge passes; merge two into one, leaving four; merge two into one leaving three; and finally merge three into the output file. In the process, QSORT must read and write about 80% of the file twice during the merge phase.

Worse yet, since you need at least three handles for merging, if you have resident programs that have open files, you can't merge at all!

DOS can be told to set aside more space for file handles. Each handle is only 39 bytes and it's memory very well spent. One process can have a maximum of 20 handles open at one time, but since resident processes may be using handles, I recommend 25 to 35. To do this, the root directory of the DISK OR DISKS YOU BOOT FROM must contain a file named CONFIG.SYS. If your boot disk(s) already contains a CONFIG.SYS, edit it, or if not, create it to contain the following line:

```
FILES=25        (or more)
```

While we're at it, let's add one more thing to CONFIG.SYS which will improve the performance of QSORT and many other programs as well. DOS provides, by default, two disk buffers. These are the buffers it uses to do its disk reads and writes. During the merge phase QSORT may have many files open at once, reading from them in more or less random order. DOS may have to read the same physical sector several times to get all its data. But DOS can remember what's in each buffer and where it came from, and will not re-read a sector it already has in a buffer. DOS needs 528 bytes for each buffer. I recommend 20 buffers to make QSORT perform well under the most adverse conditions. This will require an additional 9504 bytes or slightly more than 9K, again memory well spent, so we add to CONFIG.SYS the following line:

```
BUFFERS=20
```

See your DOS manual for more information on CONFIG.SYS.

Performance and Input Record Type

QSORT must read and parse logical records before sorting them, then reassemble them before final output. The type of records contained in the file being sorted determines how much work this requires, and therefore has an impact on performance.

The present version of QSORT can handle four record types: simple ASCII, tagged ASCII, delimited field ASCII and fixed length, determined by the presence or absence of a /T, /D or /F parameter on the command line.

Fixed length records are very structured and require no parsing. Other things equal, files of fixed length records will sort the fastest.

When parsing simple ASCII records, QSORT must find and mark the newline sequence, then restore it for final output. In general, this is relatively fast, but is affected by line length. In particular, lines containing "over-strikes" (naked CR characters followed by more data) can significantly slow down the parsing.

Tagged ASCII records are parsed in a fashion similar to simple ASCII records, if a tag character is defined. First the tag character is found, then the next newline sequence is found and marked. The time required is of course dependant on the total length of the logical record, but is fairly fast. If no tag character is defined, two successive newline sequences must be found. This depends not only on total length, but the number of lines contained in a logical record.

To parse a delimited field record with n fields, n minus one delimiters must be found and marked, then the newline sequence must be found and marked. It is similar to tagged records with no defined tag character, but because records of this type are usually shorter than tagged records, parsing delimited field records may be a little faster. It is certainly slower than parsing simple ASCII records.

Performance and Sort Keys

The sort keys defined on the command line have a lot to do with QSORT's performance. There isn't much you can do in the way of a strategy you can use when you need a particular file sorted in a particular way, but you should at least be aware.

Several decisions must be made in comparing two records. Which field contains the current key? Is the field long enough to contain the key in one, both or neither record? Are the keys lexical or ASCII? If the answers to any of these questions will remain constant over the course of a sort run, they should be answered once, not several thousand times!

QSORT has ten record comparison routines varying in degree of complexity. At the beginning of each sort run it selects the simplest one possible, based on the parameters given, to be used throughout the run.

If no sort key parameters are given, the entire record is used as a key. The compare routine has no decisions it must make -- it simply compares the two strings handed it. This is the "simple sort," and is the fastest possible case.

A sort key that does not begin at the beginning of a variable length record, may not be contained in a particular record at all, while a fixed length record is known to contain all keys. Other things equal, files of fixed length records will sort somewhat faster because the compare routine does not have to test for "key containment."

Lexical keys are first compared with a "case insensitive" technique. Each character is tested to see if it is alphabetic. If it is, it is converted to upper case. Then a converted character from each record is tested. This is obviously slower than directly comparing two characters. In the event lexical keys compare equal, they are compared again using a direct compare technique! Files with lexical keys sort slower than similar files without them.

In the case of files with delimited field records, the compare routine must find the correct field for each key, determine if the keys are contained within the fields, and finally compare them. The added step of searching for fields slows record comparison.

In general, the more complex the data, the more complex the sorting task and the longer it will take. QSORT attempts to optimize its performance by making as many decisions as it can about your data up front, then selecting a compare routine that makes only the necessary decisions on a record-by-record basis.

Performance and File Size

I received a letter from someone which included a graph showing QSORT's performance in sorting time vs. file size. He said he had expected an exponential, or at least a logarithmic curve. Instead time increased linearly with file size. I admit it puzzled me at the time, but Codeview, Microsoft's debugger, made it ease for me to measure the performance of the various parts of the QSORT program. It turns out that actual sorting of data accounts for a very small percentage of QSORT's running time. It

spends most of it's time doing I/O. For files up to about 50 kilobytes, it will read and write each record once. From 50K to about 750K there will be one merge pass and each record will be read and written twice. Since the amount of I/O increases linearly over this range of file sizes, so will sorting time.

Above about 750K a second merge pass will be needed, but in this size range, only seven to ten percent of the data will be processed in the first merge pass, so the sort time vs size curve will steepen slightly, but will not experience a large step (as it did in versions 1 and 2). Doubling the file size to 1.5 megabytes should increase the sort time about three times.

Sorting time will be approximately proportional to file size times the "average passes over data" number from the statistics report. Since this number remains a constant "2.0" over a wide range of file sizes, sorting time will be a linear function of file size in that range.

I hope you find this program useful. Your comments and suggestions are welcome. My address is in the next section.

About Shareware

QSORT is made available under the "shareware" concept. Shareware products are distributed freely and publicly. You are invited to "test drive" them without cost. But shareware is NOT FREE! If you use a product, you are expected to pay a fee for its use. Because overhead costs are minimal, this fee is usually a fraction of the normal commercial price the product might carry, but it is NOT zero!

If you find this program useful, you are asked to send its author a license fee of $20 for each machine on which you use it. This will encourage the development of other useful, affordable tools.

QSORT may be freely copied and distributed. provided that 1) it is distributed under the name "QSORT," and 2) this documentation file always accompanies it. Vendors wishing to distribute QSORT commercially, or with commercial products may contact the author at the address below for terms.

Send checks to:

Ben Baker
Baker's Acre
RR #1, Box 637
E. Alton, Il 62024

Bug reports or suggestions may be sent to the above address or via electronic mail to FidoNet node 100/10 or AlterNet node 44/76, or by logging into Baker's Acre BBS at 618-251-2169.

→