QSORT -- Version 4.11

Text File Sorting Utility

Table of Contents

List of Tables

List of Figures

Introduction

QSORT was first designed to be a replacement for, and to overcome the limitations of DOS SORT. 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 sorting a file, it terminates with ERRORLEVEL set to zero.

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

About Shareware

QSORT is the copyrighted property of Ben Baker, and 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 lower, this fee is usually a fraction of the normal commercial price the product might carry, but it is NOT zero!

Version 4 of QSORT will continue to be distributed and supported as shareware. It may be freely copied and distributed, provided that 1) it is distributed under the name "QSORT," and 2) the documentation file always accompanies it.

If you find this program useful and it meets your needs, noncommercial users are asked to pay a license fee of $20 for each machine on which it is used.

The license fee for commercial use of QSORT, version 4, is $35 each for the first nine machines on which it is used. Liberal quantity discounts, and site licensing are available.

Vendors wishing to distribute QSORT, version 4, as a part of commercial products may contact the author at the address below for

terms.

Send checks or correspondence to:

Baker & Associates
One Mark Twain Plaza, Ste 325G
Edwardsville, IL 62024
Phone: (618) 656-8850

From time to time, as necessary, maintenance updates will be released to correct deficiencies, but no new features or capabilities will be added to this shareware product. All future development will go into the commercial version of QSORT. At this writing, version 5 is available as a commercial product. It is significantly faster than this version and has a number of new features, not the least of which is the ability to sort dBASE database files.

The complete commercial package (version 5 or later), including printed documentation and technical support may be purchased from System Enhancement Associates at the following address:

System Enhancement Associates, Inc.
925 Clifton Ave.
Clifton, NJ 07013
Phone: (201) 473-5153

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, underscored, and are surrounded by angle brackets (<>). They are replaced by actual data such as a file name.

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, they are optional. You need not select either.

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.

EXAMPLES are shown in **bold** upper case characters.

The QSORT Command and Options

QSORT is invoked with the following command:

```
QSORT [<in file> [<out file>]] [/<key spec>]. . .
     [/F<len> | /D[<fields>][<delim>] | /T[<tag>]]
     [/N<term>[<term2>]] [/R] [/S[V]] [/?]
     [@<command file>] ["2><error file>"]
```

Note that all parameters on the command line are optional. The <in file> and <out file> parameters are ASCII 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.)

The simplest of all invocations of the QSORT program is just:

```
QSORT
```

DOS assigns input to the key board, and output to the screen. As you type lines of input, QSORT salts them away in its sort buffer. When all lines have been entered, type a CTRL-Z (hold the "Ctrl" key and press "Z") and press "Enter." This signals DOS that you are finished and QSORT sorts its buffer and the output is displayed on your screen.

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:

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

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

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" or "tagged" records, and locates this key within a particular field of a delimited-field record or line of a tagged record. Under certain circumstances limits are placed on the [<field>.] element (see the /D parameter below). If [<field>.] is omitted, the first field or line is assumed. For consistency, all records are assumed to have "fields." In all cases except delimited-field or tagged 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 the case of tagged records, column 1 is the first character of the identified line. 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 or tagged records, is assumed to be part of the key.

The [<type>] element is an optional suffix letter which tells QSORT how to compare these keys. At present, there are only two key types recognized by QSORT, but others are planned for future versions.

If no [<type>] is given, or if [<type>] is "A" these keys will be ordered according to the ASCII character sequence. This is QSORT's default behavior.

If [<type>] is "L" QSORT uses a "lexicographic" sequence for this key. Lexicographic sequence is ordered first by spelling, then, when keys have identical spelling, by capitalization. (See the section on Lexicographic Sorting later in this manual.)

To avoid confusion, [<type>] must not be the only <key spec> element given. In other words, if you wish to use the entire record as the sort key, but sort records in lexicographic order, use /+L, not /L. The "+" in the first form serves only to identify this as a <key spec> parameter.

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

Records and Record Types

The QSORT program sorts logical units called records. In a majority of cases, a record is simply a variable length line of text in a standard ASCII text file, but QSORT does provide support for fixed length records, and several variations on the variable length record theme.

Fixed length records are just what the name implies -- each record consists of a defined number of bytes. Record parsing consists of merely counting.

A variable length record is one or more variable length lines of ASCII text. A line is a possibly empty string of characters ending with a newline sequence.

The QSORT program supports three kinds of variable length record:

The default: Standard lines of text, as mentioned above, with keys located at fixed positions within each line (by default, the entire line).

Delimited records: These are lines which are divided into variable length fields separated by a special character called a delimiter. A sort key may be located within any particular field.

Tagged records: These are logical records consisting of many lines of text. QSORT finds the end of each record by searching for either an empty line or a line

containing a particular tag character. Sort keys are

usually located on the first line of the record, but may be defined to be on any particular line.

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, but if these conventions are mixed, it will affect sorting order.

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 /N<term>[<term2>] Parameter

When sorting variable length records, the QSORT program finds the end of each line by searching for the newline sequence. Most files are well behaved in this respect. Their lines end with a carriage-return character (CR), followed by a line-feed character (LF). The /N<term>[term2>] is provided to handle the exceptions. It allows you to redefine the newline sequence to be any one- or two-character sequence, and thus match the most pathological convention a file might observe!

For example, a file might contain lines ending with a naked LF character. This is a standard convention for files imported from a UNIX environment. Without redefining the newline sequence, QSORT could not sort these files. (Another situation where this is useful will be described under the /D parameter, below.)

<term> and the optional <term2> define the one- or two-character sequence QSORT is to use to find the end of each line of text.

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. Table 1 lists the special characters recognized by QSORT.

To perform a simple sort on a UNIX file, as mentioned above, you would use the command:

```
QSORT UNIXFILE /N\L
```

Note that the \N character listed in table 1 has no meaning in this context. It means the newline sequence and cannot be used to define itself.

\B	Space character
\F	Form feed character
\L	Line feed character
\N	Newline sequence
\R	Carriage return
\T	Tab character
\/	The slant bar character
\\	Back-slash character itself

Table 1 - Special character notation

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.

The special characters defined in table 1 may be used to represent <tag>. Thus an invisible tab character might be used to end a multi-line logical record. Notice that the slant bar (/), when used as a delimiter character in the /T or /D parameters, must be prefixed by the back-slant to prevent it from being interpreted as the beginning of a new parameter.

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

The /D[<fields>][<delim>] 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.

If a <delim> character is present, QSORT uses it as a field delimiter character. Otherwise a comma (,) is assumed to be the delimiter. The same character codes listed in table 1 may be used to represent these characters. Note that "\N" means the newline sequence, either CRLF or as redefined by the /N parameter.

The <fields> element defines the number of variable length fields

contained in each record. In most cases, this element is

```
+-------------------------------------------------------------+
|    437248 bytes of buffer space available                   |
|                                                             |
|     61344 records sorted                                    |
|       126 bytes in longest record                           |
|                                                             |
|    885731 sort phase comparisons                            |
|     99510 merge phase comparisons                           |
|                                                             |
|    985241 total comparisons                                 |
|      16.0 comparisons per input record                      |
|                                                             |
|         11 temporary merge files created                    |
|          1 merge passes                                     |
|        2.0 average passes over data                         |
|                                                             |
|       4:29 elapsed time                                     |
+-------------------------------------------------------------+
```

Figure 1 - Sort statistics report

unnecessary. The QSORT program accommodates a variable number of delimited fields. If a sort key is defined for a field which does not exist in the current record, it is treated as NULL and sorts low when compared with a record which contains the field and its key.

There is a special case in which the <fields> element is required, and there must be exactly <fields> fields in each and every record. When <delim> is \N, the QSORT program has no way to find the end of each logical record, except to count fields. (Why would you want to do this? See the mailing label example on page 18.)

> NOTE: The /F, /T and /D parameters define different record formats, and are therefore incompatible. Only one of these parameters may be used for any given execution of the QSORT program.

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, by default, the console. Figure 1 is an

actual statistics report produced after QSORT had sorted a four and one half megabyte file.

The first number is the size of the sort buffer the QSORT program was able to obtain from DOS. It depends on the amount of memory available in your machine at the time QSORT is executed. The more there is, the better QSORT will perform.

The next two numbers are self-explanatory. Then come 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. This number is typically 10 to 20. 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." Extremely large files may produce more temporary files than QSORT can read at one time, and thus require more than one merge pass. In that event, this number will be higher.

The optional subparameter, [V] (for verbose), causes the QSORT program to make running progress reports to the screen. 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.

> NOTE: The sort reported in Figure 1 was performed on a 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 4.5 megabyte file took under four and a half minutes.

The /? Parameter

The /? parameter requests help or parameter evaluation. When

QSORT is executed with the /? parameter alone, it lists a short

```
+-------------------------------------------------------------+
|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                        |
+-------------------------------------------------------------+
```

Figure 2 - Parameter evaluation report

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 /? /5:12L /-3:2 /22 /T /R <INFILE.TXT >OUTFILE.TXT
```

produces a screen report similar to figure 2.

This display lists everything QSORT knows about the proposed sort. It shows the file name(s), if known, or in this case, the fact that QSORT is being used as a "filter" and file names are unknown. It lists file characteristics, here showing that the input file has records "tagged" with an empty line. And it lists characteristics of all defined key fields. The third key in this report has an unspecified length. The value "65535" merely means that this key extends to the end of the first line of each record.

The @<command file> parameter

Normally you give QSORT its parameters on the command line, but it is also possible to place a complicated set of parameters in a "command file," and tell QSORT to get its parameters from that file. For example, the command:

```
DIR | QSORT /30:2 /24:5 /39 /34:5 /1
```

would produce a directory listing sorted by file date and time, oldest file first. That's fine if you can remember which columns DOS uses in its directory listings. But I can't, can you?

A file named DATESORT.DIR is supplied on your distribution diskette. Figure 3 is a listing of its contents.

```
+-----------------------------------------------------------------+
|  /30:2     ;; year - 2 digits                                   |
|  /24:5     ;; month/day - 2 digits/2 digits                     |
|  /39:1     ;; 'a' or 'p' for AM or PM flag                      |
|  /34:5     ;; hour:min - 2 digits:2 digits                      |
|  /1        ;; file name, if date and time are equal             |
+-----------------------------------------------------------------+
```

Figure 3 - Sample command file listing

Using this file, the command:

```
DIR | QSORT @DATESORT.DIR
```

will produce the same sorted directory listing, and the file name is a lot easier to remember than the five sort keys! It is also possible to mix command line and command file parameters. The command:

```
DIR | QSORT @DATESORT.DIR /R
```

produces a sorted list, newest rather than oldest file first.

In DATESORT.DIR, each key is placed on a line by itself. Obviously, this works, but it is not necessary. Parameters in a command file need only be separated by "white space," spaces, tabs or new lines.

A command file is invoked by placing an "at sign" (@) before its name, and it may contain any valid QSORT parameters except the "redirection" parameter, below. It may even contain another command file parameter, but QSORT cannot detect loops in command files invoked this way, so be careful.

There is a limit to the depth command files may be nested. Since a command file is not closed until it has been completely read, you cannot have more command files than you have available file handles. And it is possible QSORT would run out of stack space before it runs out of handles, and crash pitifully! QSORT command file nesting has been tested to a depth of two, and it's hard to envision ever needing more than that (famous last words!).

Notice, also that there is a bunch of commentary in DATESORT.DIR which would hopelessly confuse QSORT if it were entered on the command line. If QSORT finds two semi-colons (;;) in a row in a command file, it stops looking at that line, and goes on to the next line. This is only necessary if you wish to place commentary into a command file. The reason that two semi-colons are needed to introduce a comment is that a command file might contain a perfectly legitimate parameter, /D; for example, in which a semi-colon is part of the parameter.

There is only one situation in which two semi-colons might be part of a parameter, and it seems so remote that it's hardly worth mentioning. If you wish to sort a file so pathological that it uses two semi-colons as a newline sequence, you can put /N;; on the command line, but not in a command file. You could use /N;\; instead, and accomplish the same thing.

What if you wish to sort a file named @SILLY.FIL? The best thing to do is rename it and get rid of the at sign. But if QSORT encounters a parameter which begins with two at signs (@@), it discards the first at sign and takes the rest of the parameter as a literal file name. Thus:

```
QSORT @@SILLY.FIL @@SILLY.OUT
```

would sort your silly file, placing the sorted output in a file named @SILLY.OUT.

The "2><error file>" parameter

The QSORT program writes its messages and help and statistics screens to the DOS standard error device. DOS allows the user to redirect standard input and standard output, as discussed earlier, but makes no provision for redirecting standard error. As far as DOS is concerned, error messages belong on the screen and nowhere else!

The "2><error file>" parameter provides the capability of redirecting QSORT's messages and screens to the file named in <error file>. UNIX users should recognize the syntax immediately. It is borrowed from the Bourne-Shell. Some replacements for DOS' COMMAND.COM, particularly those which emulate the UNIX environment, also support this syntax. The parameter should be enclosed in quote marks as shown to prevent later versions of DOS from trying to perform the redirection wrong!

Two variations on this parameter are also recognized by the QSORT program. "2>><error file>" tells QSORT to open <error file> in append mode and add the messages for this run to the end of the contents already in <error file>. "2>&-" turns off all message output from QSORT.

To summarize:

```
QSORT JUNK /S "2>MESSAGES.TXT"
```

sorts the file JUNK in place, writing the statistics screen to the file MESSAGES.TXT. If that file already exists, it is replaced by the output of this QSORT run.

```
QSORT JUNK /S "2>>MESSAGES.TXT"
```

sorts the file JUNK in place, appending the statistics screen to the end of the file MESSAGES.TXT.

```
QSORT JUNK /S "2>&-"
```

sorts the file JUNK in place. The /S parameter is meaningless, since the "2>&-" parameter turns off all message output from QSORT!

Command line 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.

INPUT	ASCII	CASELESS	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

Table 2 - Key type sorting examples

Lexicographic Sorting

The lexicographic sorting capability was born 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.

Table 2 illustrates the problem. 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, or caseless 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 spelling together, but in the third column, upper and lower case spelling are in arbitrary order, while the fourth column places upper case spelling ahead of lower case spelling.

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 lexicographic sorting will always put them in this order. Notice, also that the two occurrences of delaPort are not together in column 3, but are brought together in column 4.

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

Lexicographic keys are defined, as indicated above, by placing the letter 'L' in the <type> element at the end of <key spec> definitions.

Lexicographic sorting can be very useful when needed, but be aware that unnecessarily specifying lexicographic 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:

```
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:

```
DIR | QSORT /30:2 /24:5 /39 /34:5 /1 | 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, time, and finally file name. The output of QSORT is then piped to MORE for display.

Alternatively, the command:

```
DIR | QSORT @DATESORT.DIR | MORE
```

using command file **DATESORT.DIR** supplied on your QSORT distribution diskette, does the same thing.

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

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

or

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

or

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

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:

with redirection

```
QSORT /T  <GLOSS.TXT >GLOSS.SRT
```

or without redirection

```
QSORT /T   GLOSS.TXT  GLOSS.SRT
```

A lawyer keeps a running log of his billable activities in TIME.LOG. The first line of each entry is the account number, and the second line is date and time in the form "mm/dd/yy hh:mm." 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:

```
QSORT  /1. /2.7:2 /2.1:5 /2.10:5 /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.

```
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.

```
QSORT /D /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 <u>not</u> 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.

```
QSORT  SEX.LST /70:10 /1:30 /N\L /SV
```

The /N 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 (CARRT) order.

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

We must use a "delimited field" sort rather than a "tagged line" sort because 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. Since each label always contains six lines, we can treat it as six fields delimited by the newline sequence, but we must inform QSORT of the total number of fields.

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 bad 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 "F3" key once more, then back space over the "/?" parameter, then hit "Enter" and QSORT will do the rest.

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. (See the section on "Performance and Sort Keys" for more information.)

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\N /6.1:3" would provoke QSORT into issuing this message. It's hard to find field 6 in a 5-field record. Another possibility is that you may have forgotten to use a /D or /T parameter, altogether. Since multiple fields are only valid for these record types, one must be present whenever you use a field number greater than 1. This message will never appear for tagged records, or for delimited field records where the field delimiter and the record delimiter are different. In those cases, QSORT permits a variable number of fields, and any field number is legitimate in a key specification.

Multiple STDERR redirections

At most, the standard error output may be redirected once. A second attempt to do so will fail with this message.

Invalid STDERR redirection

Two conditions cause this message; use of 2> or 2>> as the last parameter, with no file specified, or use of 2>&<x> where <x> is any text other than an unadorned hyphen.

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 even-

tually 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 open "<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. DOS will also complain if you attempt to open a file with the same full name as an existing subdirectory.

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

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

Table 3 - ERRORLEVEL return codes

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. Table 3 lists the ERRORLEVEL codes QSORT uses, and their meanings.

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 as much memory as it can get as a sort buffer 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

When QSORT must create temporary merge files, it first creates a temporary subdirectory to hold them. The subdirectory is named SRTnnnnn, where nnnnn is a random number generated from the DOS clock. When QSORT completes, even if it aborts for any reason, it deletes any temporary files and the temporary subdirectory it has created.

With nothing else to guide it, QSORT places its temporary subdirectory in the default directory. Any of three "environment variables" can override this. (See your DOS manual for information on environment 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 600K 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!

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 somewhat larger than the size of the input file because DOS allocates file space in clusters of 2 or 4 kilobytes. 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 reduced 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. Assuming a 400K sort buffer, a 2 megabyte 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 suggest 25. 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
```

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 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 by way of a strategy, when you need a particular file sorted in a particular way, but you should at least be aware.

If no sort key parameters and no record type parameters are given, the entire simple ASCII 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."

Lexicographic 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 lower case. Then the converted character from each record is compared. This is obviously slower than directly comparing two characters. In the event lexicographic keys compare equal, they are compared a second time using a direct compare technique! Files with lexicographic 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 making only the necessary decisions on a record-by-record basis.

Performance and Memory Size

If you have as little as 50 kilobytes of usable memory, the QSORT program will perform correctly, but memory space this limited will only allow a sort buffer of about eight kilobytes. Sorting files any larger than this will cause QSORT to create many small temporary merge files, and performance will suffer.

QSORT is capable of using as much memory as DOS has to give it. So long as the file being sorted is smaller than QSORT's sort buffer, no merging will be required and optimum performance is achieved. The more memory you can give QSORT, the better it will perform!

There are, of course, limits.

Performance and File Size

If you have 640 kilobytes of installed memory, and if you are not running a multi-tasking system such as Windows, Desqview or

DoubleDOS, and if you do not have a ton of terminate-and-stay-

resident programs installed in your system, the QSORT program should have between 400 and 500 kilobytes of memory to use as a sort buffer. Files smaller than this will not require merging and will sort quite fast, the sorting time being nearly proportional to file size times the logarithm of file size.

A file larger than the available sort buffer size will require more than one sort pass, and hence, at least one merge pass. Merging is approximately a linear process, so sort time will increase proportional to file size.

For a file larger than the number of available handles times the size of the sort buffer (typically a very large file), 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. Doubling the file size 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.

LIMITED WARRANTY

IMPORTANT NOTICE: Any use of this software for any period of time for any purpose whatsoever constitutes your unqualified acceptance of this LICENSE and subjects you to all of the terms and conditions set forth below:

Baker & Associates ("B&A") warrants to any Licensee that acquires the program from B&A or an authorized B&A representative ONLY that:

1) All diskettes B&A provides constitute an accurate duplication of the software and B&A will replace any diskette found to be defective within 30 days from date of acquisition. B&A will not honor this warranty if the diskette has been subjected to physical abuse, or used in defective or non-compatible equipment.

2) Software distributed by B&A will perform substantially as described in the documentation B&A regularly supplies with that software, if operated as prescribed in such documentation including the hardware and software environment specified.

3) If a significant defect in any program is found, Licensee's only remedy shall be to receive refund of the actual fee Licensee paid for such defective program. In no event will such a refund exceed the fee B&A charges for such program.

4) B&A makes no warranty or representation that the software will be error free nor that its use by Licensee will be uninterrupted.

Except as provided above, B&A disclaims all other warranties, either express or implied, including but not limited to any implied warranty of merchantability or fitness for any particular purpose.

Licensee agrees to take full responsibility for the selection of and any use whatsoever made of the software.

IN NO EVENT WILL B&A BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING WITHOUT LIMITATION DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION OR THE LIKE) ARISING OUT OF THE USE OF, INTERRUPTION IN THE USE OF, OR INABILITY TO USE THIS SOFTWARE, EVEN IF B&A HAS BEEN ADVISED OF ANY POSSIBILITY OR LIKELIHOOD OF SUCH DAMAGES.