MSORT Reference Manual

William J. Poser
billposer@alum.mit.edu

Contents

1. Introduction

Msort is a program for sorting text files in sophisticated ways, intended especially for linguistic databases. It allows you to specify arbitrary sort orders, to sort blocks of text delimited in a number of ways rather than just lines, and to specify particular fields of a record as sort keys. *Msort* is capable of sorting on several keys, so that when two records tie on one key, the tie may be broken on another. Any or all keys may be optional. It provides a number of specialized types of comparison in addition to the usual numeric and lexicographic comparisons.

msort is a complex program which often requires a long and intimidating-looking command line. With luck, this manual will resolve any difficulties. However, the reader may be interested to know that a graphical interface to *msort* called *msg* is available.

2. What *msort* does

The use of *msort* is best understood in terms of the operations it performs. It performs four tasks:

a. parsing the input into records;
b. extracting the sort keys from each record;
c. sorting the records;
d. writing out the sorted records;

2.1. Parsing into Records

A record is the basic unit on which *msort* operates. When it reads its input, it breaks it up into a sequence of records. A record may consist of a single line of text, as in typical sorting programs. To specify that a record is a single line, give the command line option *–l*.

However, for many applications it is desirable for a record to consist of several lines. For example, a lexical database might contain records consisting of a prounciation field, a gloss, and a category, such as this:

P:ditnikwun
G:dandelion
C:N

msort therefore recognizes a second type of record, one consisting of a block of text.

Sometimes records are separated by a special separator character. (Records consisting of a single line are really just the special case of this where the separator is an end-of-line character.) *msort* recognizes such records if the separator character

is specified by the *–r* option. Backslash escapes are recognized and may be used to supply values otherwise difficult to enter on the command line.

msort also recognizes blocks terminated by one or more blank lines. *msort* treats such blocks as records by default. This may be explicitly specified by giving the command line option *–b*.

One other type of record is recognized: a record consisting of a specified number of bytes. These are selected by the *-O* option. Such fixed-length records are primarily found in old-fashioned databases but are still encountered from time-to-time.

2.2. Extracting Keys

The information used for sorting is known as the sort key. For example, if we sort a mailing list by postal code, ignoring the name and address, the postal code is the sort key. By default *msort* uses the entire record as the sort key. (This default may be explicitly specified by giving the command line option *–w*.) If only part of the record is used as the sort key, this may be done in several ways: (a) a position range, based on fields, may be used as the key; (b) a particular field may be used as the key; (c) a character range may be used as the key.

2.2.1. Parsing Records into Fields

A field is a portion of a record beginning at the beginning of a record or at the end of the previous field, and ending when a terminator character is encountered. For example, if the record consists of blocks of text and the field terminator is the end-of-line character, then each line of the record will be a field. If a record consists of a single line of text and the field terminators are space and tab (collectively known as "whitespace"), then each "word" of the line will constitute a field.

msort sets default values for the field terminators which depend on whether the record is a single line or a block of text. If the record is a single line, the field terminators are space and tab. If the record is a block of text, the field terminator is end-of-line. These defaults may be over-ridden by using the command line option *–d*. Backslash escapes are recognized and may be used to supply values otherwise difficult to enter on the command line.

2.2.2. Specifying Keys

The first way of specifying a key is by means of a position range. A position is a location within the record consisting of a field number and an optional character offset into the field. A position range is a pair of positions. The key consists of everything between the two positions, including the endpoints.

Keys are selected by position range by using the command line option *–n*. In the simplest case, the argument is a field number. Field *1* is the first field, field *2* the second field, etc. Field numbers may be negative, in which case they are interpreted relative to the end of the record. That is, field *-1* is the last field, field *-2* the next-to-last, etc. A field number of *0* is meaningless and is treated as an error. A range consisting of a single field number contains all of that field. Thus, the key specification *-n 1* means all of the first field.

A position may also contain a character offset, consisting of a positive integer separated from the field number by a period. The key specification *-n 1.1* is the same as *-n 1*, while *-n 1.3* says to use all of the first field except for the first two characters.

If two positions are used to specify a range, they are separated by a comma. The key specification *-n 1.2,3.1* indicates that the key starts at the third character of the first field and ends at the second character of the third field. The specification *-n 1.1,-1.2* identifies the first field through the last field, with only the first character of the last field in the key.

The beginning and end of a range may be the same. In this case, the key consists of a single character. For example, the specification *-n 1.2,1.2* indicates a key consisting of the second character of the first field.

The second way of specifying the field to use as a sort key is by its tag, the characters with which the field begins. For example, if we specify the sort key as the field with tag *P:*, the field beginning with the characters *P:* will be used as the sort key. Only the characters following the tag form part of the key; the tag is used only to identify the key field. Therefore, in the example above, if we give the command line option *–t P:* to specify the key as the field with tag *P:*, the field *P:ditnikwun* will be the key field, and the actual key will be *ditnikwun*.

Although tags that are simply strings are sufficient most of the time, *msort*'s tag specifications are actually regular expressions. This means that they may contain wildcards, character sets, and disjunctions. The syntax used is that of the PCRE library, which is approximately that of PERL. If your tags are plain strings you need not concern yourself with the more general regular expression facility except in the case in which characters in your strings have special meanings. The characters with special meaning are: ^ $. * [ ] ( ) + ? – and \. The character "–" has special meaning only when enclosed by square brackets. In general, you may prevent these characters from taking on their special meaning by quoting them with a preceding \. To specify a literal backslash, use two backslashes: \\.

Regular expressions as tags have several uses. First, they can be used to absorb characters that are properly speaking part of the beginning of the key but that we wish to ignore in sorting. For example, the regular expression *P:–?* has the effect of sorting on the field with tag *P:* and of ignoring a hyphen at the beginning of the key.

Second, character sets can be used when, for example, the database is inconsistent in capitalization. For example, the tag specification *[pP]:* has the effect of sorting on the field with tag *P:* or the tag *p:*, whichever comes first within the record if both are present.

Third, some databases using tagged fields separate the field name from the data in the field by a separator consisting of an arbitrary amount of whitespace (spaces or tabs). This whitespace can be absorbed into the tag by using appropriate wildcards. For example, to specify a key of *P:* followed by any amount of whitespace, use the tag specification *P:[ ]** where the brackets contain a space and a tab.

Here are some examples of more complex regular expressions. To select a tag of either *p:* or *P:* followed by any amount of whitespace, use the tag specification *[pP]:[ ]** where as above the second pair of brackets contain a space and a tab.

To select a tag of *P:* or *p:* followed by any amount of whitespace and to ignore leading hyphens, use the tag specification *[pP]:[ ]*–?* where as above the second pair of brackets contain a space and a tab.

Occasionally, you may wish to sort on one of two quite different tags. For example each entry in a lexical database might have either a gloss, tagged *g:*, or a definition, tagged *df:*. To sort on whichever one is present, specify the tag as *df:|g:*.

Note that what you consider the tag need not be what was intended to be the field label when the database was created. For example, suppose that you use the label *S:* to indicate the source of the material in an entry, and that some entries come from a written source abbreviated *M*, which is followed by the section or page number, e.g. *M82*. You might extract the entries obtained from this source and then sort them into their original order (perhaps in order to check them against the written source) by specifying the tag as *S:M* and performing a numerical sort. Treating the *M* as part of the field label has the effect of stripping it from the key and so preventing it from interfering with the numerical sort.

The third way of specifying the key is by character ranges. That is, using the *-e* option, you may specify that the key consists of the *m*th through *n*th characters in the record. Positive values start at one. Negative values indicate characters counting from the end of the record. For example, the range *3,-2* consists of the third character through the next-to-last character.

It is possible for a record not to contain a key field, in the case of numerical specification because there are not enough fields, in the case of specification by character range because there are not enough characters, and in the case of specification by tag because there is no field with the specified tag. In this case, if the key is not optional, *msort* reports the number of the offending record, writes a copy of the offending record into the log, and aborts.

If, as is possible when keys are specified by tag, a record contains more than one key field, *msort* uses the first and ignores the remainder, without reporting an error.

Since *msort* has to start at the beginning of each record and search for key fields when key fields are specified by tags, sorting is faster if the key fields are near the beginning of the record. If you know that you are likely to want to sort on some fields and unlikely to want to sort on others, you can improve the speed of the sort by putting the fields on which you are likely to sort at the beginning of the record. With current computers this is not very important.

2.2.3. Reversing Keys

The command line option *–R* causes the characters of the associated key to be reversed if it is lexicographic. (If a numeric, time, or date comparison is specified, this option has no effect.) That is, the last character of the key is treated as the first, the next to last character as the second, etc. This is useful for the generation of reverse dictionaries. Key reversal is performed after the interpretation of multigraphs.

2.2.4. Multiple Keys

When sorting on multiple keys, you specify the keys to sort on in the order in which they are to be used. For each key field, you may also specify the nature of the comparison to be used, that is, whether textual or numeric. If it is textual, you may specify the sort order to use. The specification of the field to use as the key and the nature of the comparison may be separated by other options, but each comparison specification is paired with the nearest preceding key field specification. Options that are not key-specific may appear anywhere. For example, in order to sort first on the *P:* field using the sort order in the file *c2.ord* and then on the *C:* field using the sort order in the file *cats.ord*, you would give a command beginning like this:

msort –t "P:" –s c2.ord –t "C:" –s cats.ord

If you give a non-key-specific option, such as *–l*, which tells *msort* to treat lines as records, it may appear in any postion.

For example, the following is an acceptable command line:

msort –t "P:" –l –s c2.ord –t "C:" –s cats.ord

msort can still determine that c2.ord contains the sort order to be used on the key obtained from the field with tag "P:" because *–t "P:"* is the nearest preceding key field selector.

To sort on the section number following the letter *M* in the field labelled *S:* and then on the pronunciation in the *P:* field using the sort order in the file *c2.ord*, you would give a command beginning like this:

msort –t "S:M" –n –t "P:" –s c2.ord

To sort using ASCII order on the second field and then in decreasing numerical order on the first field, you would give a command beginning like this:

msort –n 2 –n 1 –i

Here the *–i* option reverses numerical comparisons and inverts the sort order for lexicographic comparisons. Note the omission of a comparison specification after the first field specification. This causes *msort* to use the default of textual comparison using the ASCII order, assuming that the machine collating sequence is ASCII.

2.2.5. Optional Keys

Any or all keys may be specified as optional except when sorting on the entire record. A key is declared optional by giving the command line option *–o*. The argument to this option determines how records with missing values for this key are handled.

There are three possibilities. In some cases, it is desirable to have all records that lack a key sort before or after all records that possess it. If, for example, we are editing a database and wish to add the information in question to records that lack it, we may wish to arrange for them to come first. On the other hand, if records that lack the information in question are of no immediate interest, we may wish to move them to the end. The third possibility is that we wish the missing key to have no effect at all, which will result if we arrange for missing keys to compare as equal

to whatever they are compared to. Thus, for each optional key we specify whether missing keys are to compare as less than (that is, sort ahead of), greater than (that is, sort after), or equal to records that possess the key. The possible arguments to this option are therefore:

less than	l
	L
	<
greater than	g
	G
	>
equal to	e
	E
	=

Only the first character of the argument is checked, so if you wish you may spell these out or use familiar abbreviations, e.g. *less than* or *lt*. If the argument given is not recognized, the default is equality.

The optional key facility is useful for purposes other than dealing with missing values. It can also be used to put an entire class of key values into a certain category. For example, my database for the Stuart/Trembleur Lake dialect of Carrier contains entries taken from Father Adrien-Gabriel Morice's book *The Carrier Language*. In the source field, these are identified by the letter "M" followed by the section number. The source field may contain multiple entries, separated by slashes, e.g.:

%S:D/ROHA/YVPI/M454

This field indicates that the entry is to be found in the *Central Carrier Bingual Dictionary* ("D") and in Father Morice's book in section 454, and that it has also been recorded from two Carrier speakers whose names are abbreviated "ROHA" and "YVPI".

Someone working on another Carrier dialect wanted to use this lexical database, which is by far the largest for any Carrier dialect, to suggest words to look for in her own dialect. She was particularly interested in old words such as those to be found in Father Morice's book. I therefore wanted to sort the lexicon so that words found in Father Morice's work would be listed first. I used the following command to do this:

```
msort -t "%S:(M[0-9]+|.*/M[0-9]+).*" -o g -t "%P:"
```

The first tag specification is a regular expression that selects those source fields in which either the letter M plus one or more digits immediately follow either the source tag or a slash. The effect is to pick out all of the source fields which contain a reference to Father Morice's book. By making this field optional and specifying that if it is not present the record follows one in which it is present, we arrange for all of the records containing an entry found in Father Morice's book to precede the others. Since the tag exhausts the field, the source field has no other effect on the sort. The second key is the Carrier headword, which results in the records being ordered alphabetically within the two major categories.

2.3. Sorting

Once *msort* has extracted the keys from the records, it performs the sort. Each time it compares two records, it compares the keys in order. If the first key is sufficient to order the two records, no other keys are used. If, however, two records tie on a key and there is a further key available, *msort* moves on to the next key.

2.3.1. Sort Algorithms

Msort offers a choice of several algorithms for performing the actual sort: Insertionsort, Mergesort, Quicksort, and Shellsort, selected by the *–a* flag. (The "Quicksort" used is actually a hybrid, with Insertionsort used for small subpartitions. This is generally faster than a pure Quicksort.) For most users for most purposes the differences among the four algorithms will be of no concern. The only difference in the result of sorting is that between Insertionsort and Mergesort, which are *stable* algorithms, and Quicksort and Shellsort, which are *unstable*. A stable sorting algorithm is one that preserves the original order of two records whenever they compare equal. An unstable sorting algorithm may reorder records that compare equal. For example, suppose that the input consists of:

smith
Smith
Saiki
Smyth

and that we sort this input without regard to case. Two outputs will satisfy the ordering criteria:

Saiki
smith
Smith
Smyth

and

Saiki
Smith
smith
Smyth

An unstable sort algorithm could produce either output. A stable algorithm is guaranteed to produce the first output since it preserves the original order of the two items, *smith* and *Smith*, that compare as equal.

Most of the time it makes no difference whether the sort algorithm is stable or unstable, but in some circumstances stability is desirable. If you require stability, choose Mergesort or Insertionsort. Otherwise, the algorithms differ in their speed for particular types of input data. Mergesort and Quicksort are generally faster than Shellsort and Insertionsort for larger numbers of records. However, Insertionsort and Shellsort tend to be faster if the input data is already in order or very close to it. Similarly, Insertionsort and Shellsort tend to be faster if the input data is

in reverse order or close to it. Mergesort has better worst case performance than Quicksort, but in many cases Mergesort will be faster than Quicksort.

2.3.2. Comparison Types

Nine types of comparison are available, selected using the *–c* option. In addition to the usual numeric and lexicographic comparisons, a hybrid comparison is provided. Specialized comparisons are provided for angles, dates, times, and ISO8601 date-time combinations. Comparison by string length (measured in Unicode characters) is available. Finally, comparisons may be random, which allows for unsorting.

The sense of the comparison for a particular key may be inverted by giving the *–i* option. The *–I* option, with which key-specific *–i* options may be combined, inverts the overall order of the output. Giving the *–I* option is equivalent to first performing the sort and then reversing the output item-by-item.

2.3.2.1. Numeric Comparison

Msort can sort on the numerical value of a field rather than on its textual value. The numerical value will be used if the argument of the *–c* option begins with *n* or *N*. If the argument to the *–c* option is *x* or *X*, the key field will be interpreted as a hexadecimal integer.

Msort stores numeric keys internally as double precision floating point numbers. The conversion is done using the standard C library function strtod(3), which understands both standard and scientific notation, decimal and hexadecimal. The special values INFINITY and NAN are also supported. For details, see the documentation for strtod(3). The interpretation of strings as numbers is governed by the locale setting. Numbers containing delimiters such as commas can be sorted numerically by using *msort*'s exclusion facilities to remove the delimiters.

Whitespace (as defined for the current locale) is stripped from numeric keys before they are converted from string to numeric form. This is done in order to prevent msort from thinking that the field does not consist entirely of numbers and reporting an error when neither records nor fields consist of of lines.

2.3.2.2. Numeric String Comparison

Numeric string comparison is an alternative way of handling numbers. Instead of converting numbers to floating point and comparing numerical values, the numbers are stored as strings and compared using a special form of string comparison that produces the same result as numeric comparison. The advantage of numeric string comparison is that there is no loss of precision due to conversion to floating point representation. If your data contains very large or very small values, or values that differ by minute amounts, it is advisable to use numeric string comparison.

Numeric string comparison is slightly slower than numeric comparison and uses somewhat more memory. It is also more limited in the form of the numbers for which it can be used. Acceptable numbers match the following POSIX regular expression: $(+|-)?[0123456789]*.?[0123456789]*$. That is, they must be decimal numbers in standard notation with an optional leading sign. Scientific notation is not understood, nor are bases other than 10.

2.3.2.3. Lexicographic Comparison

Lexicographic comparison is done if the argument of the *–c* option begins with *l* or *L*.

Lexicographic comparisons are based on a sort order that is normally read from a file specified by the *–s* command line option. If the argument is not a filename it is taken to be a locale name. In that case, the collation rules for the specified locale are used. As a special case, instead of a filename or locale name the argument to the *–s* flag may be the word *locale*. In this case, the collation rules for the current locale are used. If the *-s* flag is not used characters are ranked in accordance with the numerical value of their Unicode codepoints.

If a sort order is specified but values are omitted, characters for which no rank is specified will sort in Unicode order compared to each other but after all characters, including multigraphs, for which a rank is specified.

The order of the lines in the sort order file determines the sort order. Characters appearing on an earlier line sort before those appearing on a subsequent line. For example, if the sort order file consists of the lines:

a
b
c

a will sort before *b* which will sort before *c*.

Characters appearing on the same line, separated by separator characters, have the same sort rank. For example, if we wish no distinction to be made, for purposes of sorting, between upper-and lower-case letters, we would put them on the same line:

a A
b B
c C

etc.

Sequences of characters are often used to represent single sounds, and in such cases we often want to treat them as if they were single letters for purposes of sorting. For example, in languages in which the digraph *lh* is used to represent the voiceless lateral fricative, *lh* is often alphabetized as a single letter ordered after *l*, so that, for example, *lho* is ordered after *lo*, not before it as it would be if the *l* and *h* were treated as separate letters and the usual order of the Roman alphabet followed.

msort allows such multigraphs to be defined in the sort order file. Any characters not separated by tabs are treated as a single letter. For example, to add *lh* to the Roman alphabet as above the relevant portion of the sort order file would look like this:

k
l
lh

m

Like ordinary letters, multigraphs may be assigned the same sort rank. To continue the above example, if we wanted upper- and lower-case letters to have the same sort rank, the relevant portion of the sort order file would look like this:

```
K     k
L     l
Lh    lh
M     m
```

msort imposes no limit on the length of multigraphs. The limit on the number of multigraphs is set so high as to be unlikely to be of any practical importance.

Notice that you may wish to define as multigraphs for sorting purposes sequences of letters that are not intended to represent single sounds. For example, suppose that you wish to sort a lexical database by part of speech. You might define a sort order like this:

```
N       Noun
V       Verb
A       Adjective       Adj     ADJ
ADV     Adverb          Adv
PP      Postposition
INT     Interjection
```

Notice how variants are placed on the same line so that they will not be distinguished for purposes of sorting, and how words are treated as multigraphs.

Similarly, the multigraph facility can be used to sort records according to a classification scheme having nothing to do with spelling or sound. For example, dictionary entries tagged for semantic field can be sorted by semantic field. For example, here is a fragment of the sort order for the topical index to a dictionary:

```
animals-land
animals-water
animals-domestic
animals-distant
animals-misc
```

Since every character in the sort order file counts, it is important to put into it exactly what you mean to put into it. Do not include blank lines or comments.

Unless your character set is very unusual, it is recommended that you include the space (blank) character explicitly in your sort order and that you rank it first. Doing this has two advantages. First, it makes sure that items intended to contain spaces will be sorted as they are usually desired to be. Second, it avoids problems with unintended trailing spaces. It is easy to add unintended, invisible space characters at the end of a line. If the space character is not ranked before the other characters, this can result in incorrect orderings whose cause is mysterious. For example, the word *an* should precede the word *and*. If, however, a space has inadvertently been

added to the end of *an*, and you have not specified that space should precede all other characters, the default ranking will cause *an* to be ordered after *and*, and since the space is invisible, it will be difficult to detect the cause of the problem.

In addition to ordinary single-byte characters *msort* recognizes backslash escapes in the sort order file. These escapes may be used to specify characters not easily entered by the user's word-processor, or which are given a special interpretation in the sort order file. For example, the tab and space characters that serve to separate entries in the sort order file may be entered into the sort order by specifying them as \011 and \040 since the ASCII codes for tab and space are octal 011 (decimal 9) and octal 040 (decimal 32) respectively.

By default, a wide range of "whitespace" characters are treated as separators, including space (0x0020), tab (0x0009), and the ideographic space (0x3000). It is sometimes desirable to use other characters as separators. For example, if a unique ordering consisting of names is defined, it may be desirable to treat names with components separated by spaces as multigraphs. In order to allow for this, you would need to prevent spaces from being treated as separators. You can redefine the separator characters by means of the *-W* flag. Its argument should be the name of a file whose contents are the character or characters that you wish to use as separators. This option must precede the *-s* flag for its key.

2.3.2.4. Hybrid Comparison

Hybrid comparison is like lexicographic comparison except for the fact that strings of digits are treated as numbers. For example, the strings:

A13
A3
A235

will be sorted

A13
A235
A3

in ordinary lexicographic comparison, but

A3
A13
A235

in hybrid comparison.

The collation rules for the non-integer portions of the input are determined as for lexicographic keys.

2.3.2.5. Comparison of Dates

msort will treat the key as a date if the argument of the *–c* option begins with *d* or *D*. Dates consist of a numerical day specification (an integer ranging from 1 through

31), a numerical month specification (an integer ranging from 1 through 12), and a numerical year specification (an integer), with the three components separated by a designated separator. Positive years are interpreted as years C.E. Dates B.C.E. may be specified by making the year negative.

By default, the separators are slashes and the components are given in the order year–month–day. The *f* option may be used to set the date format used. The argument to this option should consist of the letters "y", for year, "m", for month, and "d" for day in the desired order, separated by the desired separators. For example, if the date *1999/1/3* is to be interpreted as January 3, 1999, the argument would be *y/m/d*. If it is to be interpreted as March 1, 1999, the argument would be *y/d/m*. If *12-3-1956* should be interpreted as December 3, 1956, the argument should be *m-d-y*. Although in practice the two separators will usually be the same, *msort* allows them to be distinct. An argument of *m;d,y* would interpret a date of *12;3,1994* as December 3, 1994.

The default is the International Date Format, which is usual in Japan. It is equivalent to an argument of *y/m/d*. The usual American format requires the argument *m/d/y*. The format usual in most other western countries requires the argument *d/m/y*.

2.3.2.6. Comparison of Times

msort will treat the key as a time if the argument of the *–c* option begins with *t* or *T*.

Times must be of the form it HH:MM(.SS), or *HH:MM:SS(.ss)*. Negative values, hours outside the range 0-23, minutes outside the range 0-59, and seconds greater than or equal to 60 are detected and reported as errors. The *a.m./p.m* notation is not recognized: 24 hour "railroad time" is assumed.

2.3.2.7. Comparison of ISO8601 Date-Time Combinations

msort will treat the key as a combined date and time in ISO8601 format if the argument of the *–c* option begins with *i* or *I*.

Such combined dates and times consist of a date in ISO8601 format, the upper case letter *T*, and a time in international format. An ISO8601 date consists of a year number followed by month number followed by day number, separated by hyphens. A time in international format is a 24-hour time with hours preceding minutes separated by a colon. Seconds optionally follow, separated from minutes by a period. For example, *2005-04-25T14:11.33* represents 33 seconds past 2:11 p.m. on April 25th, 2005.

2.3.2.8. Comparison of Month Names

If the argument to the *–c* option begins with *m* or *M* *msort* will treat the key as the name of a month. If the *–s* option is also used with this key and its argument is the name of a file, month names will be read from the file. The file should have the same format as a sort order specification file. All entries on the same line will

be given the same sort rank. The sort rank will follow the order of the lines. This approach permits the use of calendars with more than twelve months. It also allows multiple abbreviations or names for the same month.

If the *–s* option is given and its argument is the name of a locale, the names of the months and their abbreviations will be obtained from the system's locale information if the specified locale is available. The locale system only supports calendars with twelve months and only allows for one name and one abbreviation for each month. If the argument is *current* or *locale*, the current locale will be used.

If the *–s* option is not given for this key, the month names and abbreviations will be obtained from the current locale.

If the system does not provide locale support and the month names are not provided via the *–s* option with filename as argument, *msort* defaults to the English month names and their standard three-letter abbreviations.

2.3.2.9. Comparison of Angles

If the argument of the *–c* option begins with *a* or *A*, *msort* will interpret the field as an angle. The expected formats are: (-/+)DDD:MM:SS.sss and (-/+)DDD MM SS.sss , where DDD is a degree value, MM is a minute value in the range [0,59], and SS.sss is a fractional seconds value in the range [0,60). In the second format, the components may be separated by any amount of whitespace. Angles whose absolute value is greater than or equal to 360 are replaced by their residue mod 360. Negative angles are converted to the corresponding positive angle by subtracting them from 360.

2.3.2.10. Comparison of Sizes

If the argument of the *–c* option begins with *s* or *S*, *msort* will sort on the size of the key. The size is the length of the key in characters, after preprocessing, including multigraph compression and deletion of excluded characters. Keys compared for size are treated exactly like lexicographic keys except that when comparisons are finally done on them, the comparison is a numeric comparison of the length of the key rather than a lexicographic comparison of the key as a string.

If you want multigraphs to be treated as single characters for the purpose of size comparison (as you might if sorting by word length in phonemes, for example), provide a sort order definition for the key just as you would when sorting lexicographically, using the *-s* flag. The sort order will be ignored, but the multigraph definitions will be used.

2.3.2.11. Random Comparison

This option is used for randomizing data, not actually for sorting it. Its effect is that, each time two records are compared on the specified key, the result of the comparison is determined by a random number generator rather than by the contents of the key fields. The records will be reported as comparing less-than, equal, or greater-than with equal probability.

The random number generator is reseeded each time *msort* is run using the current time as the seed. This means that different runs on the same data with the same parameters will produce different orderings.

2.3.3. Case Sensitivity

Lexicographic, hybrid, and string length comparisons are by default case-sensitive, meaning that characters differing only in case are treated as distinct. Comparisons may be made insensitive to case in two ways. First, upper- and lower-case characters may be assigned the same rank in the sort order specification. This is useful if for some reason it is desired to ignore case distinctions on a character-by-character basis. If, however, it is desired to ignore case distinctions in general, the case-folding option *-C* will be found less burdensome.

The case-folding option maps characters to a canonical representation that eliminates case distinctions. For ASCII characters, this mapping consists of downcasing, that is, replacing upper-case characters with their lower-case equivalents. Unicode case-folding is more complex. A full explanation may be found in version 4.0 or later of the Unicode standard or at: http://www.unicode.org/reports/tr21/tr21-5.html.

Msort implements full Unicode case-folding, meaning that case-folding is performed for all characters that make case distinctions, even if the folded equivalent of a character consists of multiple characters. This is why case-folding may affect string length comparison. For example, the upper-case equivalent of the German character *eszet* "sharp s" U+00DF is the sequence *SS*. If case-folding is used, each token of *eszet* contributes two characters, whereas if case-folding is not used, each contributes only one character.

By default the case-folding used is appropriate for languages other than certain Turkic languages. The *-z* option causes two additional mappings to be performed. Turkic case folding is appropriate for certain Turkic languages written in Roman letters, in particular Turkish and Azerbaijani. It maps U+0049 (upper case I) to U+0131 (lower case dotless I) and U+0130 (upper case I with dot) to U+0069 (lower case I).

2.4. Writing Out the Sorted Records

Once *msort* has sorted the records using the keys it has extracted, it writes out the records in the sorted order. If the command line option *–I* is given, it will write them out in reverse order. The effect of this option is the same as that of the key-specific *–i* option except, of course, that it is global. Giving both the *–i* option and the *–I* option for a sort on a single key therefore has no effect; the two options cancel each other out.

On the other hand, using these two options is not the same as inverting the sort order, which can be done by using a sort order specification file. For example, if you put lines like:

z
y
x

w
...

into a sort order file, the effect is to invert the sort ranking of individual letters. For example, *z* will now sort before *y* instead of after. Reranking individual letters in this way does not affect the rule that nothing precedes something. Consider, for example, the following data, which are in standard alphabetical order:

bad
badger
bag

If we use the *–i* option:

msort –l -w –i

we obtain the following, in which the order is strictly reversed:

bag
badger
bad

Using the *–I* option would have the same effect since we have only one key, and using both will give us the original order. If, however, we put the inverted sort order beginning with *z* and ending with *a* in a file called *rev.ord* and make this the sort order for our single key thus:

msort -l -w -s rev.ord

we obtain the following output:

bag
bad
badger

The reason is that inverting the sort order causes *g* to sort before *d*, but *bad* still precedes *badger* because it is a prefix.

3. Exclusions

Sometimes it is desirable to exclude certain characters in the key field from consideration in sorting. For example, a leading hyphen, used to indicate that a morpheme must be preceded by a prefix, will generally be ignored so that bound and unbound forms will sort together. Similarly, if numbers contain commas or other delimiters, which most functions for converting textual representations of numbers to binary do not accomodate, they can be sorted correctly by first stripping the delimiters.

In order to accomodate this need, *msort* provides an exclusion facility. For each key, the user may specify a file, using the *–x* option, containing a list of characters to ignore when sorting. In order to facilitate the use of exclusions when doing simple,

one-shot sorts, exclusions may also be specified directly on the command line, using the *–X* option.

Typical exclusion facilities do not meet the full range of needs, since we typically do not wish to ignore a character completely. To see why a simple exclusion facility, one that allows certain characters to be ignored everywhere they occur, will not suffice, consider the situations in which we may wish to ignore characters. First, we often want to specify exclusion only in certain circumstances. For example, we may want to consider word-internal hyphens but to ignore them at the beginning of a stem, where they may be used to indicate that the stem must be preceded by a prefix.

Second, we may not really wish to ignore a character entirely but rather to treat it as white space. For example, if we want to treat hyphenated words as if the hyphen were not there, so that word-internal and wordinternal are of the same sort order, exclusion of hyphens will suffice, but we may wish to treat such hyphens as spaces, so that word-internal has the same sort order as word internal. In this case simple exclusion will not work, but replacement of word-internal hyphens will, as will specification of hyphen and space as having the same rank in the sort order.

Consequently, the most general aproach to character exclusion is to use another program to create a new key field, attach this stripped key field to each entry in your database, and specify this field as the sort key.

For example, if you wish to ignore leading hyphens, you would convert an entry like this:

```
P:-gan
G:arm
```

into one like this:

```
P:-gan
G:arm
K:gan
```

and specify the *K*: field as the sort key.

However, since the most typical situation is one in which we wish to distinguish only between initial, medial, and final position, and since using another program like AWK can be time-consuming, *msort* provides a simple context-sensitive exclusion facility. You may specify that a character is to be ignored in any combination of field-initial, field-medial, or field-final position.

The exclusion file consists of one line per character excluded, with two fields, separated by whitespace, on each line. The first field contains a specification of the character to exclude, either as a single character or as a backslash escape. The second field consists of one to three of the letters “i”, “f”, and “m” (upper- or lower-case, in any order), indicating that the character is to be ignored in initial, final, or medial position. For example, the exclusion file:

```
-  if
'  i
```

will cause hyphens to be ignored in initial and final position, and apostrophes to be ignored only in initial position.

Where the material that you wish to ignore is always at the beginning of a tagged field, there is a simpler alternative, namely treating it as part of the tag and making use of the fact that, using regular expressions, you may specify tags that are disjunctive or contain optional material. For example, suppose that you wish to sort on the field with tag "P:" and that you wish to ignore leading hyphens. You can do this by defining the tag as "P:–|P:", that is, as the regular expression consisting of "P:–" or "P:". Whenever a leading hyphen is encountered it will be treated as part of the tag and thereby stripped from the key proper. When there is no leading hyphen, the second disjunct "P:" will match the tag specification. An alternative in this case is to specify the tag as "P:–?", where the question mark makes the hyphen optional. If a field begins with "P:–", the hyphen will be treated as part of the tag and so effectively stripped from the content of the field. A field beginning with "P:" not followed by a hyphen will also be recognized as the tag.

It is often not desirable to exclude a character completely. For example, if we ignore a leading hyphen so as to sort on the first regular character, forms with and without leading hyphens may be interspersed (depending on their order in the input). If the input contains two tokens of *ba* and two of *-ba*, they may end up ordered:

ba
-ba
ba
-ba

If, instead, we want the forms with leading hyphens to be grouped together, like this:

ba
ba
-ba
-ba

we must take into account the hyphen. To obtain this effect, use the same key field twice, in the primary key excluding leading hyphens, in the secondary key, including them, and defining the sort order so that they follow the alphabetic characters. On the first key, the forms with and without leading hyphens will tie. The tie will be broken on the second key, and the forms with hyphens will be made to follow those without.

When exclusions are specified directly on the command line using the *–X* option, the characters listed are excluded in all positions. If you wish to make the exclusion context-sensitive, you must use the *–x* option instead.

If a key is present but becomes empty as a result of exclusions, it is treated just as if the field had been missing from the outset.

4. Character Simplifications

Because it is often desired to treat "fancy" characters as equivalent to their plain counterparts, several transformations of large groups of characters are offered. These are specified by the command-line option *–T*, which takes as argument a sequence of any one or more of the letters *d*, *e*, and *s*.

The argument *d* specifies that diacritics are to be stripped. Separately encoded combining diacritics such as U+0301 COMBINING ACUTE ACCENT are removed. Characters with diacritics represented bysingle codepoints are replaced with the corresponding ASCII character without the diacritics, if there is one. For example, U+00E9 LATIN SMALL LETTER E WITH ACUTE is replaced by U+0065 LATIN SMALL LETTER E.

The argument *e* specifies that enclosed characters, that is, characters within circles or parentheses, are to be replaced with the corresponding plain ASCII character if there is one. for example, U+24A0 PARENTHESIZED LATIN SMALL LETTER E is replaced by U+0065 LATIN SMALL LETTER E.

The argument *s* specifies that characters in special styles are to be replaced with the corresponding plain ASCII character if there is one. The special styles replaced include: the small capitals (e.g. U+1D04), script forms (e.g. U+212C), black letter forms (e.g. U+212D), Hebrew presentation forms (e.g. U+FB1D), Arabic presentation forms (e.g. U+FE81), fullwidth forms (e.g. U+FF01), halfwidth forms (e.g. U+FF7B), and the mathematical alphanumeric symbols (e.g. U+1D400). For example, U+1D07 LATIN LETTER SMALL CAPITAL E and U+FF45 ULLWIDTH LATIN SMALL LETTER E are both replaced by U+0065 LATIN SMALL LETTER E.

5. Substitutions

The substitution mechanism allows you to define more-or-less arbitrary transformations of keys. Each set of substitutions is specific to a particular key. Each individual substitution consists of two parts: a regular expression and a fixed string. Each portion of the original key that matches the regular expression is replaced by the fixed string. In the simplest case a regular expression is itself a fixed string, so if you are not familiar with regular expressions you can still use this mechanism to replace one fixed string with another. The regular expression notation is the same as that used for matching tags. A link to a description of the notation will be found on the Help menu.

A substitution file contains one substitution rule per line. Each line consists of two fields separated by a tab. The first field is the regular expression; the second is the string to be substituted for it. If the first character of a line is a crosshatch, the line is treated as a comment and ignored.

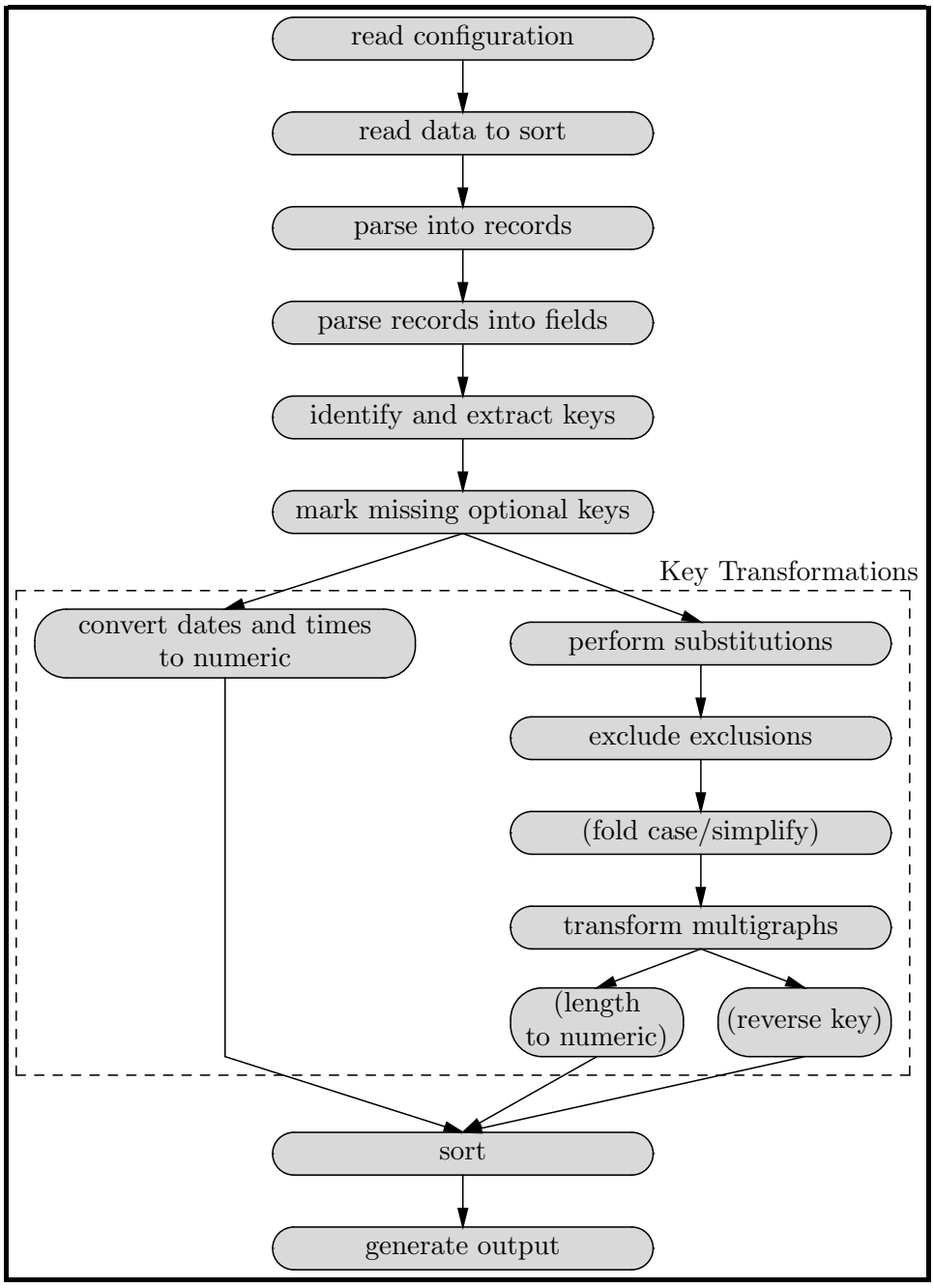
Substitution has several uses. It provides a way to handle cases in which certain components of names are alphabetized as if they were written differently. For example, in English the prefix Mc is supposed to sort as if it were spelled Mac (as

it sometimes is). "McArthur" should precede "MacCawley", as if it were spelled "MacArthur". Similarly, it can be used to disregard components of names that are supposed to be ignored in alphabetization, such as "de" and "van".

This device can also be used to handle the rare cases in which a character is supposed to sort as if it were a sequence of characters. An example is the German letter 0x00DF, "eszet", which is treated as if it were "ss" for purposes of sorting.

6. Overview of the Sorting Process

The overall process of sorting may usefully be schematized as follows:



The order of key transformations is as shown in the diagram. Substitutions are performed first, then exclusions, then case-folding, then multigraph mapping and finally key reversal.

7. Backslash Escapes

The backslash escapes recognized by *msort* are as follows. A backslash followed by one of the digits 0-3 followed by two of the digits in the range 0-7 is treated as an octal number in the range 0-377 (0-255 decimal). The escapes \n for newline, \t for tab, and \ *space* for space are also recognized. A sequence of two backslashes is interpreted as a single backlash. Any other sequence beginning with a backslash is treated literally. Examples:

\ t	a tab
\\	a backslash
\\\\	two backslashes
\\\t	a backslash followed by a tab
\a	a backslash followed by an “a”
\n	a newline
\011	octal 011 (decimal 9), the ASCII code for a tab
\411	the four characters backslash, “4”, “1”, “1”.
\\\011	a backslash followed by a tab

Take care to understand that this description of backslash escapes is in terms of what *msort* is passed by the operating system. If the backslash character has special meaning to your operating system or command interpreter, it may be necessary to quote backslashes in order to present the desired input to *msort*. On a UNIX system, you can put the argument within double quotes to prevent the shell from interpreting the backslash.

8. The Command Line

There are two types of command line options; some apply to a particular key; others are global. You will probably find it less confusing to group together options pertaining to a particular key, but *msort* does not require you to do this; global options may be interspersed among key-specific options.

msort interprets its command line from left to right. Whenever it encounters one of the two options that specify keys, namely *–n* and *–t*, it terminates processing of the previous key, if any, and begins constructing the specification for a new key. All key-specific options are interpreted as pertaining to the current key.

For example, suppose that the command line options are:

–n 3 –l –c n –n 1 –d ; –c l

This tells *msort* to sort on two keys. The primary key will the third field in the record. The secondary key will be the first field in the record. The primary key is numeric. The second is lexicographic. Records consist of single lines, with fields delimited by semi-colons.

msort knows that specification that sort is to be lexicographic applies to the secondary key because it follows the field specifier for that key and no other field specifier intervenes. In effect, *msort* parses the command line like this:

[–n 3 –l –c n] [–n 1 –d ; –c l]

Any key specific-options within a group apply to that key. As a result, global options may be placed anywhere, but key-specific options must be occur within the domain of their field-specifier.

9. Examples

Here are a number of examples of the use of *msort* with a variety of databases.

9.1. Example 1

Records consist of single lines with fields delimited by whitespace. A typical database fragment looks like this:

```
datsan crow N
khuni word N
```

To sort on the second field we would give the following command:

```
msort –l –n 2 db.txt
```

No specification of the field delimiter is necessary since whitespace is the default when records consist of single lines.

9.2. Example 2

Records consist of single lines with fields delimited by whitespace. A typical database fragment looks like this:

```
datsan 21
khuni 18
```

To sort in numerical order on the second field, we would give the following command:

```
msort –l –n 2 –c n db.txt
```

9.3. Example 3

Records consist of single lines with fields delimited by colons. A typical database fragment looks like this:

```
datsan:crow:N
khuni:word:N
```

To sort on the second field we would give the following command:

```
msort –l –n 2 –d : db.txt
```

9.4. Example 4

Records consist of single lines with fields delimited by whitespace. Each field is marked by a tag. A typical database fragment looks like this:

```
P:datsan G:crow C:N
P:khuni G:word C:N
```

To sort on the field with tag P: we would give the following command:

```
msort –l –t P: db.txt
```

9.5. Example 5

Records consist of blocks of text with each field a separate line. A typical database fragment looks like this:

```
P:datsan
G:crow
C:N

P:khuni
G:word
C:N
```

To sort on the field with tag *P:* we would give the following command:

```
msort –b –t P: db.txt
```

9.6. Example 6 — *Shoebox* Format

The Summer Institute of Linguistics uses a format in its *Shoebox* program, its successor *Toolbox*, and other software in which records consist of blocks of text. Each field begins with a tag consisting of a backslash (\) immediately followed by a field name of up to four characters, separated by a space from the content of the field. A field may consist of more than one line as newline characters do not serve as delimiters of fields. A typical database fragment looks like this:

```
\le datsan
\df crow
\ps N

\le khuni
\df word
\ps N
```

From *msort*'s perspective the fields of SIL-format databases are **terminated** by the backslash that begins the tag. The field name and the space that separates it

from the content of the field must be treated as the tag. To sort on the field with field name *le*, we would therefore use the following command on a UNIX system:

msort –b –t "le " –d "\" db.txt

The quotes around the tag are necessary to include the space. Recall that the space is, from *msort*'s point of view, part of the tag.

If the field label is separated from the content of the field by arbitrary amounts of whitespace, the regular expression facility allows this to be handled as follows:

msort –b –t "le[\040\011]+" –d "\" db.txt

Here the brackets after "le" contain the octal character codes for space and tab. The + causes the regular expression to match one or more tokens of space or tab.

10. Logging

msort writes a record of its activity into a log file called *msort.log*. The information written is information useful when something goes wrong. In particular, ill-formed records are copied to the log file to make it easier for you to find them and repair them. The result of running *msort* with command line options that produce information about *msort* is also copied to the log file. The file is overwritten each time *msort* is run.

11. Checking Whether the Input is Already Sorted

It is sometimes only necessary to find out whether the input is already sorted. This can of course be done by comparing the output of a sort with the input, but it is considerably more efficient to use the option provided by *msort* for this purpose. It avoids creating a copy of the input file as well as the i/o necessary to create it, reduces the memory consumed during the run, and speeds up the run since it is not necessary to allocate memory to store the records and the internal pointer comparison that *msort* performs is much faster than any comparison of the data itself.

The *–Q* flag is used to request a check of whether the input is already in order. If it is, *msort* returns an exit status of 0. If it is not, *msort* returns an exit status of 11. In addition, if the *–q* option is not used, a message is printed stating whether or not the input is already in order.

12. Speed

The time required for a sort depends on a variety of factors: the speed of the computer's processor, the size of the file, the number of records, storage media, the number and length of the keys, the complexity of the regular expressions defining tags, the number of multigraphs defined, and whether the sort is lexical or numerical.

In most cases almost all (over 99%) of *msort*'s run time is devoted to processing records on input, that is, to parsing them and extracting keys. The actual sort takes a small fraction of the time.

The only situation in which the actual sort takes a large fraction of the time is when there are many keys that compare the same. In this case, providing an additional key on which to break ties is likely to speed up the sort.

Contrary to naive intuition, using more sort keys does not necessarily add to the time required for the sort. In fact, in some cases it can cut it considerably. When there are many records that have the same sort rank, the sort takes a long time because a lot of comparisons are made. In such a situation, adding another key that will serve as a tie-braker reduces the number of comparisons made.

For example, sorting the list of 45,143 words found in /usr/share/dict/words on most Unix systems by word length took 84 seconds on my laptop and required 119,265,582 comparisons. This is because there are only 22 different word lengths, so thousands of words have the same length and compare as equal. When I added a second key, namely the same field compared lexicographically, the time dropped to less than one second because only 949,521 comparisons were made. Adding the second key reduced the number of comparisons by two orders of magnitude.

13. Character Set and Encoding

All input and output is in UTF-8 Unicode. Field and record separators, tag regular expressions, exclusions, and sort order specifications also expect UTF-8 Unicode. Case-folding is implemented in its full Unicode form.

Since the first 128 codepoints in Unicode are identical to ASCII, ASCII input will be treated just as in previous versions of *msort*. However, this version differs from versions up to version 7 in that input in single-byte encodings using codepoints above 0x7F (127), such as the various ISO-8859 encodings (Latin-1 etc.) are no longer acceptable as they conflict with UTF-8 Unicode. The great majority of such encodings can readily be converted to Unicode.

If input is known to be restricted to the Basic Multilingual Plane (Plane 0), that is, if no character code exceeds 0xFFFF, *msort* may be informed of this by means of the *-B* command line option. This permits a significant reduction in memory usage. Almost all of the symbols of writing systems in normal use are located within the BMP. Ranges beyond the BMP are allocated to ancient and exotic writing systems and various kinds of special symbols.

14. Limits

msort imposes the following limits. These may also be obtained by giving the command line option *–L*.

maximum length of numeric key	256
number of multigraphs per key	

Default	130,668
BMP only	6,400
Private Use areas reserved	2,048

15. Defaults

These are the default values of parameters that may be set via command line options. These values may also be obtained by giving the command line option *–D*.

Field terminator characters	
record = line	whitespace
record = block	newline
Key specification	whole record
Initial maximum number of records	8192
Record type	double-newline terminated block
Sense of sort	forward

The phrase "initial maximum number of records" requires some explanation. *msort* imposes no intrinsic limit on the number of records it can handle; the only real limit is available memory. However, it allocates space for a certain number of records and then allocates more as necessary. The initial value used is the "initial maximum". It has the default value shown above and may be set from the command line using the *–M* option. *msort* increases the maximum number of records by 50% each time it finds that it has run out. This means that the time taken for storage allocation and the sometimes necessary relocation of data in memory will be larger if you use a small initial maximum. On the other hand, using a large initial maximum and thereby a large increment may cause *msort* to allocate more storage than it really needs, conceivably using up memory needed for other purposes. If you know the approximate number of records in your database, the best strategy is to set the initial maximum a little bit larger than your estimate. If you have little idea how many records there are in a new database and want to maximize the chance of fitting into available memory, choose a small initial maximum and hence a small increment. *msort* will then fit the storage fairly closely to that actually required, at the expense of more time for repeated expansion of the record list. Most users should never need to pay attention to this option. It is provided to allow fine-tuning in special cases.

16. Exit Status

msort returns the following status codes:

0	success
1	error opening file
2	input/output error
3	provided information and exited without sorting
4	limit exceeded

5 invalid command line option
6 invalid argument to command line option
7 ran out of memory
8 ill-formed record encountered
9 other error

17. Summary of Command Line Options

General:

–a,––algorithm ⟨sort algorithm⟩	I(nsertionsort), M(ergesort) Q(uicksort), S(hellsort)
–b,––block	A record is terminated by 2+ end-of-line characters.
–l,––line	A record consists of a single line.
–r,––record–separator ⟨ separator⟩	A record is terminated by separator.
–O,––fixed–size–record–size ⟨ bytes⟩	A record consists of the specified number of bytes.
–d,––field–separators ⟨character⟩$^+$	Fields are delimited by the named character(s).
–n,––position ⟨field number⟩	Sort on the specified field (counting from one).
–t,––tag ⟨tag regexp⟩	Sort on the field with the specified tag.
–w,––whole–record	Sort on the entire text of the record.
–M,––initial–maximum–records ⟨records⟩	Set initial maximum number of records.
–m,––line–end–carriage–return	In the input data end-of-line is marked by CR (0x000D).
–I,––invert–globally	Invert sense of comparisons globally.
–Q,––check–only	Check whether input is already sorted
–q,––quiet	Be quiet — do not chat while working.
–D,––defaults	List defaults.
–F,––general–options	List general command line options.
–K,––key–specific–options	List key-specific command line options.
–L,––limits	List limits.
	[These two options must precede the first -s option.]
–B,––BMP	The input contains no characters beyond Plane 0
–p,––reserve–private–use–area	Do not use the Private Use areas internally.

Key Specific:

–C,––fold–case	Fold case
–c,––comparison–type ⟨key type⟩	l(exicographic), n(umeric), N(umeric string), h(ybrid), s(tring length) i(so8601 date/time), t(ime), d(ate), m(onth), a(ngle), r(andom)
–f,––date–format ⟨date format⟩	Permutation of ymd with separators (e.g. y/m/d)
–i,––invert–locally	Invert sense of comparisons on key.
–o,––optional ⟨comparison⟩	Optional key: compare as $(<, =, >)$ to present key if absent
–R,––reverse–key	Reverse characters in key.
–s,––sort–order ⟨file name⟩	Read the sort order from the named file.
–T,––transformations ⟨(d)(e)(s)⟩	Apply the specified character simplifications. d strip diacritics e convert enclosed characters to plain s convert special styles to plain
–x,––exclusion–file ⟨file name⟩	Read the exclusions from the named file.
–X,––exclude–characters ⟨characters⟩	Exclude the specified characters.
–z,––turkic–case–folding	

18. Copyright and License