```
 \ea\let\ea\auxone\csname\the#1\endcsname
 \ea\let\ea\auxtwo\csname\the#2\endcsname
 \ea\global\ea\let\csname\the#2\endcsname
 \auxone
 \ea\global\ea\let\csname\the#1\endcsname
 \auxtwo}.
```

To verify your understanding, what is the result[39] of

```
\m3\n4\def\3{first}\def\4{second}
\xch\m\n
\the\m, \the\n; \3, \4.
```

4.3 Some testing

Apart from the examples as given above, lexicographic sorting has been tested on sequences of random words. For this purpose use has been made of Reid's [30] work for generating random paragraphs in TEX.

Timings. On my 8086 MS-DOS PC the (word) sorting, without the time needed to create the array but with the time needed to write to the dvi-file, had the performance

No	≈ Time
15	30 seconds
50	3 minutes.

The University's VAX8650[40] needed ≈ 5 minutes to sort 500 random words. The measurements were done with `\heapsort` as sorting macro, with each word of random length.

5 Applications

5.1 Sorting address labels

Amy Hendrickson [15] used sorting of address labels to illustrate various macro writing TEXniques. However, she used external sorting routines. Here I will do the sorting within TEX, and enrich her approach further by separating the mark-up phase from the data base query and the report generating phases. Because this paper concentrates on sorting aspects, let us assume that each address is supplied as a definition, with the definitions of the name and address components as replacement text. Furthermore, it is handy to create a list of all the addresses: the names of the address definitions separated by `\as`, the address separator.[41] For the imaginative toy adresses of the three composers: Schönberg, Webern, Strawinsky, the structures look like as follows.

```
\def\schonberga{\def\initial{A}
 \def\sname{Arnold}\def\cname{Sch\"onberg}
 \def\street{Kaisersallee}\def\no{10}
 \def\county{}\def\pc{9716HM}
 \def\phone{050-773984}\def\email{as@tuw.au}
 \def\city{Vienna}\def\country{AU}}
%
\def\strawinskyi{\def\initial{I}
 \def\sname{Igor}\def\cname{Strawinsky}
 \def\street{Longwood Ave}\def\no{57}
 \def\county{MA}\def\pc{02146}
 \def\phone{617-31427}
 \def\email{igor@ai.mit.edu}
 \def\city{Boston}\def\country{USA}}
%
\def\weberna{\def\initial{A}
 \def\sname{Anton}\def\cname{Webern}
 \def\street{Amstel}\def\no{143}
 \def\county{Noord-Holland}\def\pc{9893PB}
 \def\phone{020-225143}\def\email{aw@uva.nl}
 \def\city{Amsterdam}\def\country{NL}}
%
%and the list
\def\addresslist{\as\strawinskyi
 \as\weberna\as\schonberga}
```

For the typesetting I made use of the following simple address label format[42]

```
\def\tsa{%The current address info is set
 \par\initials \cname \par
 \no\ \street\ \city\par
 \pc\ \county\ \country\par}
%
\def\initials{\ea\fifo\initial\ofif}
\def\fifo#1{\ifx\ofif#1\ofif\fi#1. \fifo}
\def\ofif#1\fifo{\fi}
```

Example (Database query: selection of addresses per country)
Suppose we want to select (and just `\tsa` them for simplicity[43]) the inhabitants from Holland from our list. This goes as follows.

```
\def\search{NL}
\def\as#1{#1\ifx\country\search\tsa\fi}
\addresslist
```

The above yields the result
A. Webern
143 Amstel Amsterdam
9893PB Noord-Holland NL

[39] Answer: 3, 4; second, first.

[40] Just to get the flavor of it because VMS is a time sharing system.

[41] By this set-up we can do a lot more than just sorting address labels. What about mailmerge? What about 'TEX as a database report generator?' Jurriens [18] coined the term, although most of the work there was done via UNIX scripts.

[42] The encoding of printing special address labels has been worked out by for example Damrau & Wester [7]. It is left as an exercise to the reader to modify `\tsa` such that address labels are typeset in an m-by-n grid, each label of size h-by-w with parameters m, n (counters), and h, w (dimensions).

In this example `\initials` is not used. It has been added to allow for multiple initials of which all letters must end with a period.

[43] We could also create a new address list for that country and apply another query, or just sort.

Example (Sorting address labels)
Amy's example can be done completely within TEX, as follows.

```
%Prepare sorting
\def\as#1{\advance\k1 \ea\xdef\csname
 \the\k\endcsname{\ea\gobble\string#1}}
%
\def\gobble#1{}
%
\k0{}\addresslist%Create array to be sorted
\n\k\def\prtw{}%Suppress default \prtw
\sortw%Sort the list
%Typeset addresses, alphabetically ordered
\k0
\loop\ifnum\k<\n\advance\k1
 \csname\csname\the\k\endcsname\endcsname
 \vskip1ex\tsa
\repeat
```

The above yields the results

A. Schönberg
10 Kaisersallee Vienna
9716HM AU

I. Strawinsky
57 Longwood Ave Boston
02146 MA USA

A. Webern
143 Amstel Amsterdam
9893PB Noord-Holland NL

Remarks. The automatic mark-up of address data supplied in a TEX independent way, is not the subject of this paper. The given set-up allows to add, in any order, the address information to the database, under the restriction that definitions with the same names must be used for the address components.[44] The list must be modified too.

As can be seen from the above, and also in Amy's free format, it is not easy to keep the file ordered while extending the database. Therefore sorting is needed, such that the database can be extended in an arbitrary way. Database TEXniques have it that modifications to the data are independent from the report generating, thanks to the sorting tools.

5.2 Sorting Knuth's index reminders

An index reminder, as introduced by Knuth, consists of index material to be further processed for typesetting an index. In the TEXbook, p. 424, Knuth gives the syntax of an index reminder

$$\langle word\rangle\textvisiblespace!\langle digit\rangle\textvisiblespace\langle page\ number\rangle.$$

The reminders, one per line, are written to a file because only the OTR knows the page numbers. Knuth considered this file, `index.tex`,

> '. . . a good first approximation to an index.'

He also mentions the work of Winograd and Paxton [36][45] for automatic preparation of an index. Here we will provide a second approximation to an index: the index reminders are sorted and compressed. The sorting is done on the three keys

primary key: $\langle word\rangle$
secundary key: $\langle digit\rangle$, and
tertiary key: $\langle page\ number\rangle$.

The compressing comes down to reducing the index reminders with the same $\langle word\rangle$ $\langle digit\rangle$ part to one, with instead of one page number all the relevant page numbers in non-decreasing order.

We assume that the index reminders are already stored in the array.[46] Similarly, I didn't bother about writing the sorted and reduced array to a file. It is up to the index preparator what to do with the array and how to typeset it. Furthermore, it is not complete, because of subentries, subsubentries, or 'see. . . ,' and 'see also. . . ,' which are not considered here.[47]

Example (Sorting on primary, secundary and tertiary keys)

```
\def\1{z !3 1}\def\2{a !1 2}\def\3{a !1 3}
\def\4{a !1 1}\def\5{ab !1 1}\def\6{b !0 1}
\def\7{aa !1 1}\def\8{a !2 2}\def\9{aa !1 2}
\n9\k0\kk0
\let\cmp\cmpir\sort\let\sepw\\\null
\hfil\vtop{\hsize2cm\noindent
 after sorting\\[.5ex]\prtw}
\hfil\vtop{\hsize2.5cm\noindent
 after reduction\\[.5ex]\redrng\prtw}
\hfil\vtop{\hsize2cm\noindent
 typeset in\\index:\\[.5ex]\prtind.}\hfil
```

The above yields[48]

after sorting:	after reduction:	typeset in index:
a !1 1	a !1 1–3	`a 1–3`
a !1 2	a !2 2	`\a 2`
a !1 3	aa !1 1, 2	`aa 1, 2`
a !2 2	ab !1 1	`ab 1`
aa !1 1	b !0 1	`b 1`
aa !1 2	z !3 1	⟨z⟩ 1.
ab !1 1		
b !0 1		
z !3 1		

[44] Of course one can change the chosen names.

[45] Later Lamport provided makeindex and Salomon a plain version of it, to name but two persons who contributed to the development. The Winograd Paxton Lisp program is also available in Pascal.

[46] The process for storing the contents of `index.tex` in the array `\1`, `\2`,. . . , `\`$\langle n\rangle$, has been described in Storing from a file, section 2, and will not be repeated here.

[47] An approach to handle sub(sub)entries is to allow for composite primary keys, for example separated by `\se`, respectively `\sse`. In `\decom, and \typind` we have to account for the various possibilities. I will come back to the issue of typesetting Indexes within TEX, another time.

[48] The unsorted input can be read from the verbatim listing.

Design

Given the sorting macros we just have to encode the special comparison macro in compliance with `\cmpw`: compare two 'values' specified by `\def`-s. Let us call this macro `\cmpir`.[49] Each value is composed of

- a word (action: word comparison),
- a digit (action: number comparison), and
- a page number (action: (page) number comparison).

The macros read as follows.

```
\def\cmpir#1#2{%#1, #2 defs
%Result: \status= 0, 1, 2 if
%        \val{#1} =, >, < \val{#2}
 \ea\ea\ea\decom\ea#1\ea;#2.}
%
\def\decom#1 !#2 #3;#4 !#5 #6.{%
 \def\one{#1}\def\four{#4}\cmpaw\one\four
 \ifnum0=\status%Compare second key
   \ifnum#2<#5\global\status2 \else
     \ifnum#2>#5\global\status1 \else
       %Compare third key
       \ifnum#3<#6\global\status2
       \else\ifnum#3>#6\global\status1 \fi
       \fi
     \fi
   \fi
 \fi}
```

Explanation. I needed a two-level approach. The values are decomposed into their components by providing them as arguments to `\decom`.[50] The macro picks up the components

- the primary keys, the $\langle word \rangle$,
- the secundary keys, the $\langle digit \rangle$, and
- the tertiary keys, the $\langle page\ number \rangle$.

It compares the two primary keys, and if necessary successively the two secondary and the two tertiary keys. The word comparison is done via the already available macro `\cmpaw`.

To let this work with `\sort`, we have to `\let`-equal the `\cmp` parameter to `\cmpir`.

Reducing duplicate word-digit entries

The idea is that the same index entries, except for their page numbers, are compressed into one, thereby reducing the number of elements in the array. Instead of one page number all the relevant page numbers are supplied in non-descending order in the remaining reminder, in range notation. The macro is called `\redrng`[51] and is given below

```
\def\redrng{%Reduction of \1,...,\n, with
%page numbers in range representation
 {\k1\kk0
 \ea\let\ea\record\csname\the\k\endcsname
 \ea\splitwn\record.\let\refer\word
 \let\nrs\empty\prcrng\num
 \loop\ifnum\k<\n\advance\k1
  \ea\let\ea\record\csname\the\k\endcsname
  \ea\splitwn\record.%
  \ifx\refer\word%extend \nrs with number
    \prcrng\num
  \else%write record to \kk
   \advance\kk1 \strnrs \ea\xdef
   \csname\the\kk\endcsname{\refer{} \nrs}
   \let\nrs\empty\init\num\prcrng\num
    \let\refer\word
  \fi
 \repeat\ifnum1<\n\advance\kk1 \strnrs\ea
  \xdef\csname\the\kk\endcsname{\word{}
   \nrs}\global\n\kk\fi}}
%auxiliaries
\def\splitwn#1 !#2 #3.{\def\word{#1 !#2}%
 \def\num{#3}}
%
\def\prcrng#1{\init{#1}\def\prcrng##1{%
 \ifnum##1=\lst\else\ifnum##1=\slst
  \lst\slst\advance\slst1 \else
  \strnrs\init{##1}\fi\fi}}
%
\def\strnrs{\dif\lst\advance\dif-\frst
 \edef\nrs{\ifx\nrs\empty\else\nrs\sepn\fi
  \the\frst\ifnum0<\dif
   \ifnum1=\dif\sepn\the\lst
   \else\nobreak--\nobreak\the\lst
   \fi
  \fi}}
```

Explanation. The encoding is complicated because while looping over the index reminders either the reminder in total or just the page number has to handled. The handling of the page numbers is done with modified versions of `\prc`, `\prtfl`, called respectively `\prcrng` and `\strnrs`.[52] I encoded to keep track of the numbers in the macro `\nrs`, in the case of duplicate word-$digit$-entries. Another approach is while typesetting the array element to process the page numbers via `\prc` [25].

Typesetting index entries

Knuth has adopted the following conventions for coding index entries.

<table>
<tr><th>Mark up</th><th>Typeset in copy*</th><th>In <code>index.tex</code></th></tr>
<tr><td><code>^{...}</code></td><td>...</td><td><code>... !0</code> ⟨page no⟩</td></tr>
<tr><td><code>^^{...}</code></td><td>'silent'</td><td><code>... !0</code> ⟨page no⟩</td></tr>
<tr><td><code>^|...|</code></td><td><code>|...|</code></td><td><code>... !1</code> ⟨page no⟩</td></tr>
<tr><td><code>^|\...|</code></td><td><code>|\...|</code></td><td><code>... !2</code> ⟨page no⟩</td></tr>
<tr><td><code>^|<...>|</code></td><td>⟨...⟩</td><td><code>... !3</code> ⟨page no⟩</td></tr>
</table>

* `|...|` denotes manmac's, TUGboat's,... verbatim.

[49] Mnemonics: compare index reminders

[50] Mnemonics: decompose. In each comparison the def-s are 'dereferenced,' that is their replacement texts are passed over. This is a standard TEXnique: a triad of `\ea`-s, and the hop-over-s to the second argument.

[51] Mnemonics: reduce (in range notation). The macro `\red`, which does not yield the page numbers in range notation is supplied in the file `sort.tex` too.

[52] Mnemonics: processc with ranges, respectively store numbers.

The typesetting as such can be done via the following macro.

```
\def\typind#1{%#1 a def
 \ea\splittot#1.%
 \ifcase\digit\word\or
         {\tt\word}\or
  {\tt\char92\word}\or
  $\langle\hbox{\word}\rangle$\fi{}
 \pagenrs}
%
\def\splittot#1 !#2 #3.{\def\word{#1}%
 \chardef\digit#2{}\def\pagenrs{#3}}
%
\def\prtind{{\def\\{\hfil\break}\k\kzero
 \def\sep{\let\sep\sepw}%
 \loop\ifnum\k<\n\advance\k1 \sep
  \ea\typind\csname\the\k\endcsname
 \repeat}}
```

The typesetting of the index à la TEXbook Appendix I has been dealt with in the Grandmaster chapter of the TEXbook, p. 261–263.

5.3 More than one index

Erik Frambach posed the following question on the tex-nl@hearn discussion list

> How to prepare automatically two index files: one for commands and one for the rest?

A solution to this problem is to create the information in two files, one for the control sequences and the other for the rest. This works independently of the used tool. Another solution is splitting the `index.tex` file, depending upon the $\langle digit\rangle$ code.[53] Knuth associated control sequences with code 2, when writing the index entry to `index.tex`, TEXbook p. 423.

Example (Separate sorting of control sequences)

```
\def\1{wd !2 7}\def\2{wrd !1 1}
\def\3{wd !2 2}\def\4{a !1 1}
\def\5{wd !2 5}\def\6{wd !2 3}
\def\7{z !3 7}\def\8{wrd !1 5}
\def\9{wd !2 1} \n9
\let\sepw\\\null
\hfil\vtop{\hsize=2cm\noindent
 data:\\[.5ex]\prtw}
\hfil\vtop{\hsize=2.2cm\sortcs\noindent
 after splitting:\\[.5ex] {\n\pk\prtw}
  \\[.5ex]\kzero\pk\prtw}
\hfil\vtop{\hsize2.5cm\let\cmp\cmpir
 {\low1\up\pk\quicksort}
 {\low\pkone\up\n\quicksort}\noindent
 after sorting\\both parts,\\
 compressing,\\and typesetting:\\[.5ex]
 \redrng\n3 \prtind\\[.5ex]\typind\4.}
```

yields[54]

data:	after splitting:	after sorting both parts, compressing, and typesetting:
wd !2 7	wrd !1 5	a 1
wrd !1 1	wrd !1 1	`wrd` 1, 5
wd !2 2	z !3 7	$\langle$z$\rangle$ 7
a !1 1	a !1 1	`\wd` 1–3, 5, 7.
wd !2 5	wd !2 5	
wd !2 3	wd !2 3	
z !3 7	wd !2 2	
wrd !1 5	wd !2 7	
wd !2 1	wd !2 1	

Encoding

```
\def\getdig#1 !#2 #3.{\def\dig{#2}}
%
\def\sortcs{\global\k0\global\pk\n
\global\pkone\pk\global\advance\pkone1
%Invariant: 1:k non-cs-s,
%             and pk+1:n cs-s
\loop\global\advance\k1
\ifnum\k<\pkone
 \ea\ea\ea\getdig\csname\the\k\endcsname.%
 \if2\dig{\continuetrue%        <--
cs<=>2!
  \loop
   \ifnum\k=\pk\global\pkone\pk
    \global\advance\pk-1 \continuefalse
   \else\ea\ea\ea\getdig\csname\the\pk
    \endcsname.%
    \if2\dig\global\pkone\pk
     \global\advance\pk-1
     \ifnum\k=\pk\continuefalse\fi
    \else\xch\k\pk\global\pkone\pk
                 \global\advance\pk-1
     \continuefalse
    \fi
   \fi
  \ifcontinue
  \repeat}%
 \fi
\repeat}%Result\1:\pk non-cs, \pkone:\n cs
```

Explanation. Suppose that the file `index.tex` is stored in the array. Loop through the array and compare the $\langle digit\rangle$ with 2. In case of a control sequence swap this index entry with an appropriate entry at the end.

The invariant of the loop is: $\backslash 1 : \backslash\langle k\rangle$[55] contains no control sequences, and $\backslash\langle pk+1\rangle : \backslash\langle n\rangle$ contains control sequences.

As result the array is partitioned with the control sequences at the end of the array, that is the replacement texts of $\backslash\langle pk+1\rangle : \backslash\langle n\rangle$.

[53] Conversely, merging two separate index files is easy, and can be done via `\storefrom`$\{\langle 1^{st}\text{-}file\rangle\}$ `\kzero\n` `\storefrom`$\{\langle 2^{nd}\text{-}file\rangle\}$.

[54] The sorting of the control sequences can be done via a slightly more efficient `\cmpir`, because of the same $\langle digit\rangle$.

[55] Not pk, but k!

6 Epilogue

A glossary or an index is usually processed outside of TEX, that is via other tools. 'How to encode in TEX,' was explored via the classic example of sorting. No robustness was strived after. The encodings have been kept as simple and flexible as possible.[56] As a consequence no attention has been paid to safeguarding goodies like the prevention of name confusions with those already in use by an author.

Silent redefinitions do occur when not alert. Beware!

Looking back. Much of the work has been done in the spirit of

> Abstraction is our only mental tool
> to master complexity E.W. Dijkstra
>
> A professional starts
> where an amateur ends G.E. Forsythe

7 TEXniques used

The printing of a sequence parameterized by the separator.
FIFO and the active list separator to store a sequence in an array.
Parameter separators to select parts of an argument.
Peeling off characters one by one from a string.
Expanding the parameters before the invokation of the macro (Use of triads of `\ea`-s).
Using the ASCII values of characters for comparison.
Transforming numbers into characters.
Generating random elements for testing.
`\ag` to store random generated letters as words.
Setting up an address database.
Selecting from a database via queries implemented via the active list separator.
Maintaining a heap structure.
Initialization of loops and recursion: on first traversal some actions are different from the rest.
Ending recursion via gobbling up the tokens including the invokation for the next level.
Parameterizing sorting with respect to the comparison operation.
`\chardef`-s to parameterize the ordering table of the alphabet.
Sorting (accented) words.
Sorting on keys, with composite values.
Compressing index reminders.
Nested `\csname`.
Not only exchanging expansion order but also processing order, via `\ag`.

Hard things. One can rhetorically question whether the macros have been coded in a near optimal way?[57]
I'm convinced that the basic approach

> to parameterize as much as possible

is a good thing. I also believe that the modular approach to encode small pieces, with clear functional tasks, is the way to build something of a reasonable size, and to keep it readable and maintainable. Literate programming avant la lettre?

Often I needed the `\xdef` functionality, but partially expanded. As a typical example the following.
Instead of

```
\xdef\<name>{\csname\the\k\endcsname}
```

I had to use (also with `\global`)

```
\ea\let\ea\<name>\csname\the\k\endcsname
```

when I incorporated the handling of accents. Also for the non-accent case the latter is better, because it leaves the contents of $\backslash\langle name\rangle$ untouched. It is not clear to me whether the use of token variables instead, would have been better.

Exchanging the order of expansion is abundantly used in the TEXbook. In generating random words I needed to delay the storing. In that particular case—reasonable size of the wordlength—I could fruitfully made use of `\ag`. From this I learned that the use of `\ag` comes in when 'stomach' processes have to be exchanged on the fly.

A TEXfall is that `\global`, so easily used with counters and definitions, does not extend to `\newif`-s. In first instance I tried to keep track of the status of the comparison of two strings by Booleans, at an inner level. Because, I could not use them globally otherwise than adapting the `\newif` macro, I have used a counter—`\status`—instead.

Another TEXfall is that the `\body` of a loop is silently redefined when nesting loops without scope braces. This occurs for example when in a loop a macro is invoked which contains an (unbraced) inner loop in its replacement text. This is different from the TEXfall where the first (inner) `\repeat` is mistaken for the outer one. Difficulties with nesting of loops, especially to keep quantities local, have been alluded to earlier by Pittman [29].

In debugging I traced every comparison and exchange via `\immediate\write16...`-s.

In articles like this it is difficult to circumvent unwanted spaces when in horizontal mode. My solution is to do the sorting in vertical mode and when done typeset in horizontal mode. I have taken notice of Eijkhout's suggestions [11].

---

[56] But, alas, full of details.

[57] Indeed, because of the many ways one can encode in TEX, it is very hard, if not impossible, to decide which code is best. Perhaps we have to get used to it that programming is like life. Polymorph! Knuth experienced similar things as can be distilled from 'Always remember, however, that there's usually a simpler and better way to do something than the first way that pops into your head.' The TEXbook, p. 373. Apart from the set-up, much has been given an afterthought or two.

On the other hand in the encoding of `\seq` I had to insert an empty group after `\wofif` in order to retain the separator ␣. This insertion of the empty group was also necessary in `\redrng` when rewriting the array: not `\ ` but `{}` ! I also had to compensate for Southall's 'buses and weirdness'-TEX effect, about which he lectured so vividly at the 1990 SGML-TEX meeting at Groningen.

Conclusion

I believe that my macros can be of use for preparing indexes completely within TEX. In the discussion about the NTS (New Typsetting System), in [28] and [35], it is argued to think in pre- and post-processing outside of TEX. Sorting index items is neither a pre- nor a post-process. Generally it is done in between. A file with index reminders is written while TEXing the compuscript. Then external sorting and the like is done outside of TEX, and finally the typesetting of the index is done again by TEX. Now all can be done within TEX, despite the good and abundant external sorters available.

At the danger of being accused of misusing TEX as

'. . . another American screw driver'[58]

for situations not envisioned in the design, I found that encoding a non-trivial example in TEX illustrates the power of TEX's language. But, . . . I also sadly endured TEX's negative side

Encoding in TEX is error-prone! if not for being so unusual.

This despite of its author being the initiator of literate programming.[59]

A discipline of TEX encoding? Absolutively!

Acknowledgements

Ronald Kappert is kindly acknowledged for his suggestions and remarks while proofing.

References

[1] Alexander, J.C (1986): Tib, a reference setting package. *TUGboat* 8, no. (2), 102.

[2] Amstel, J.J van, J Bomhoff, G.J Schoenmakers (1978): Inleiding tot het programmeren 1. Academic Service.

[3] Arseneau, D (1992): overcite.sty, drftcite.sty, cite.sty. (from the file server)

[4] Bentley, J (1986): Programming Pearls. Addison-Wesley.

[5] Bechtolsheim, S von (1989): `\csname` and `\string`. *TUGboat* 10, no. (2), 203–206. (Apart from the basics, SvB discusses the convenient (read non-double) loading of macro files, and the cross-referencing. Common to both applications is the use of `\csname` and `\string`. No nesting of `\csname`, and no mentioning of associating `\csname...` with arrays.)

[6] Chen, P, M Harrison (1987): Automatic index preparation. CSB-TR 87/347. UCB. (A nice survey of issues relevant to preparing indexes automatically. Existing indexing tools in use in various systems are discussed. The paper emerged from the experience gained in writing the makeindex C program.)

[7] Damrau, J, M Wester (1991): Form letters with 3-across labels capability. TUG '91. *TUGboat* 12, no. (4), 510–516.

[8] Durst, L (1989): Bibliographic citations, or variations on the old shell game. *TUGboat* 10, no. (3), 390–394.

[9] Durst, L (1991): Some tools for making indexes. *TUGboat* 12, no. (2), 248–252.

[10] Eijkhout, V (1992): TEX by Topic. Addison-Wesley.

[11] Eijkhout, V (1993): The bag of tricks. *TUGboat* 13, no. (4), 494–495.

[12] Green, I (1992): citesort.sty. (from the file server)

[13] Greene, A.M (1989): TEXreation—Playing Games with TEX's mind. *TUGboat* 10, no. (4), 691–705.

[14] Hendrickson, A (1989): MacroTEX.

[15] Hendrickson, A (1990): Getting TEXnical: Insights into TEX macro writing techniques. *TUGboat* 11, no. (3), 359–370.

[16] Jeffreys, A (1990): Lists in TEX's mouth. *TUGboat* 11, no. (2), 237–244.

[17] Jensen, K, N Wirth (1975): PASCAL user manual and report. Springer-Verlag.

[18] Jurriens, T.A (1992): TEX as database. MAPS92.2, 100–101.

[19] Kappert, R (1992): scite.sty. (A compilation of the earlier versions of Arseneau and Green. From the file server.)

[20] Knuth, D.E (1973): The Art of Computer Programming 3. Sorting and searching. Addison-Wesley.

[21] Knuth, D.E (1984): *The TEXbook*, Addison-Wesley.

[22] Laan, C.G van der (1992a): Syntactic Sugar. MAPS92.2, 130–136. (Submitted to TUG '93.)

[23] Laan, C.G van der (1992b): FIFO & LIFO sing the BLUes. MAPS92.2, 139–144. (To appear TUGboat 14.1. An earlier version has appeared in the EuroTEX '92 proceedings.)

[24] Laan, C.G van der (1992c): Tower of Hanoi, revisited. *TUGboat* 13, no. (1), 91–94. Also: MAPS92.1, 125–127.

[25] Laan, C.G van der (1993): Typesetting number sequences. MAPS93.1. (4 pages. Submitted TUGboat.)

[26] Lamport, L (1986): LATEX, user's guide & reference manual. Addison-Wesley.

[58] To paraphrase Perlis.

[59] With the purpose to program like writing literature. Not only to be processed by computers, but also to be read by humans, with pleasure!

[27] Lamport, L (1987): Makeindex, an index processor for LaTeX. (A clear user's guide for using makeindex—a C program—together with LaTeX.)
[28] Palais, R (1992): Moving a fixed-point. *TUGboat* 13, no. (4), 425–432.
[29] Pittman, J.E (1988): Loopy.TeX. *TUGboat* 9, no. (3), 289–291.
[30] Reid, T.J (1987): Floating figures at the right — and— Some random text for testing. *TUGboat* 8, no. (3), 315–320.
[31] Salomon, D (1989): Macros for indexing and table-of-contents preparation. *TUGboat* 10, no. (3), 394–400.
[32] Salomon, D (1992a): NTG's Advanced TeX course: Insights & Hindsights. MAPS 92 Special. 252p.; revised $\approx 500p$.
[33] Salomon, D (1992b): Index preparation for TeX related documents. MAPS92.2, 111–114. (The adaptation of makeindex for plain is discussed.)
[34] Spivak, M.D (1989): L$\mathcal{AMS}$-TeX. TeXplorators.
[35] Taylor, P (1992): The future of TeX. Proceedings EuroTeX '92. 235–254. (Reprinted in *TUGboat* 13, no. (4), 433–442.)
[36] Winograd, T, B Paxton (1980): An indexing facility for TeX. *TUGboat* 1, no. (x), A1–A12. (The work uses TeX version 1.x and a Lisp program. It does not contain such a clear user guide as for makeindex [27]. It provides numbers in range notation and allows for subentries, and cross-references like see ... and see also .... The program can merge files. The program has been converted into Pascal some years later. The latter version is available on file servers.)
[37] Wirth, N (1976): Algorithms + Data Structures = Programs. Prentice-Hall.
[38] Youngen, R.E (1992): TeX-based production at AMS. MAPS92.2, 63–68.

Appendix A: Heap sort

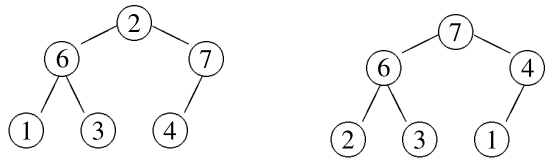
The process consists of two main steps, [2], [20]

- creation of a heap
- sorting the heap

with a sift operation to be used in both.

In comparison with my earlier release of the code in MAPS92.2, I adapted the notation with respect to sorting in *non-decreasing* order.[60]

What is a heap? A sequence $a_1, a_2, \ldots, a_n$, is a heap if $a_k \geq a_{2k} \wedge a_k \geq a_{2k+1}, k = 1, 2, \ldots, n \div 2$, and because a_{n+1} is undefined, the notation is simplified by defining $a_k > a_{n+1}, k = 1, 2, \ldots, n$.
A tree and one of its heap representations of $2, 6, 7, 1, 3, 4$ read



The algorithm. In PASCAL-like notation the algoritm, for sorting the array a[1:n], reads

%heap creation
$l := n$ **div** $2 + 1$;
while $l \neq 1$ **do**
$l := l - 1; sift(a, l, n)$ **od**
%sorting
$r := n$;
while $r \neq 1$ **do**
$(a[1], a[r]) := (a[r], a[1])\%exchange$
$r := r - 1; sift(a, 1, r)$ **od**
%sift #1 through #2
$j := \#1$
while $2j \geq \#2 \wedge (a[j] < a[2j] \vee a[j] < a[2j+1])$ **do**
$mi := 2j +$ **if** $a[2j] > a[2j+1]$ **then** 0 **else** 1 **fi**
$exchange(a[j], a[mi])\ j := mi$ **od**

Encoding

Purpose. Sorting values given in an array.

Input. The values are stored in the control sequences `\1`, ..., `\`$\langle n\rangle$. The counter `\n` must contain the value $\langle n\rangle$. The parameter for comparison, `\cmp`, must be `\let`-equal to `\cmpn`, for numerical comparison, to `\cmpw`, for word comparison, to `\cmpaw`, for word comparison obeying the ASCII ordering, or to a comparison macro of your own. (The latter macro variants, and in general the common definitions for `\heapsort`, and `\quicksort`, are supplied in the file `sort.tex`.)

Output. The sorted array `\1`, `\2`, ... `\`$\langle n\rangle$, with `\val1` $\leq$ `\val2` $\leq \ldots \leq$ `\val`$\langle n\rangle$.

Source

```
%heapsort.tex                        Jan, 93
\newcount\n\newcount\lc\newcount\r
\newcount\ic\newcount\uone
\newcount\jc\newcount\jj\newcount\jjone
\newif\ifgoon
%Non-descending sorting
\def\heapsort{%data in \1 to \n
\r\n\heap\ic1
{\loop\ifnum1<\r\xch\ic\r
   \advance\r-1 \sift\ic\r
\repeat}}
%
\def\heap{%Transform \1..\n into heap
 \lc\n\divide\lc2{}\advance\lc1
 {\loop\ifnum1<\lc\advance\lc-1
```

[60] It is true that the reverse of the comparison operation would do, but it seemed more consistent to me to adapt the notation of the heap concept with the smallest elements at the bottom.

```
  \sift\lc\n\repeat}}
%
\def\sift#1#2{%#1, #2 counter variables
 \jj#1\uone#2\advance\uone1 \goontrue
 {\loop\jc\jj \advance\jj\jj
  \ifnum\jj<\uone
   \jjone\jj \advance\jjone1
   \ifnum\jj<#2 \cmpval\jj\jjone
         \ifnum2=\status\jj\jjone\fi\fi
   \cmpval\jc\jj\ifnum2>\status%
                          \goonfalse\fi
 \else\goonfalse\fi
\ifgoon\xch\jc\jj\repeat}}
%
\def\cmpval#1#2{%#1, #2 counter variables
%Result: \status= 0, 1, 2 if
%values pointed by
%              #1 =, >, < #2
 \ea\let\ea\aone\csname\the#1\endcsname
 \ea\let\ea\atwo\csname\the#2\endcsname
 \cmp\aone\atwo}
\endinput                     %cgl@rug.nl
```

Explanation

\heapsort. The values given in `\1, ...` $\backslash\langle n\rangle$, are sorted in non-descending order.

\heap. The values given in \1,...,$\backslash\langle n\rangle$, are rearranged into a heap.

\sift. The first element denoted by the first (counter) argument has disturbed the heap. Sift rearranges the part of the array denoted by its two arguments, such that the heap property holds again.

\cmpval. The values denoted by the counter values, supplied as arguments, are compared.

Examples (Numbers, words)

```
\def\1{314}\def\2{1}\def\3{27}\n3
\let\cmp\cmpn\heapsort
\begin{quote}\prtn,\end{quote}
%
\def\1{ab}\def\2{c}\def\3{aa}\n3
\let\cmp\cmpaw\heapsort
\begin{quote}\prtw,\end{quote}
and
\def\1{j\ij}\def\2{ge\"urm}\def\3{gar\c con}
\def\4{\'el\`eve}\n4
\let\cmp\cmpw {\accdef\heapsort}
\begin{quote}\prtw\end{quote}
```

yields

> 1, 27, 314,
> aa ab c,

and

> élève garçon geürm jij.

Appendix B: Quick sort

The quick sort algorithm has been discussed in many places, for example [20]. Here the following code due to Bentley [4], p. 112, has been transliterated.

```
procedure QSort(L,U)
if L<U then Swap(X[l], X[RandInt(L,U)])
   T:=X[L] M:=L
   for I:=L+1 to U do
       if X[I]<T M:=M+1
          Swap(X[M], X[I]) fi
   od Swap(X[L], X[M])
   QSort(L, M-1) QSort(M+1, U)
fi
```

Encoding

Purpose. Sorting of the values given in the array $\backslash\langle low\rangle, \ldots, \backslash\langle up\rangle$.

Input. The values are stored in $\backslash\langle low\rangle, \ldots, \backslash\langle up\rangle$, with $1 \leq low \leq up \leq n$. The parameter for comparison, `\cmp`, must be `\let`-equal to `\cmpn`, for number comparison, to `\cmpw`, for word comparison, to `\cmpaw`, for word comparison obeying the ASCII ordering, or to a comparison macro of your own. (The latter macros, and in general the common definitions for `\heapsort`, and `\quicksort`, are supplied in the file `sort.tex`.)

Output. The sorted array $\backslash\langle low\rangle, \ldots \backslash\langle up\rangle$, with `\val`$\langle low\rangle \leq \ldots \leq$ `\val`$\langle up\rangle$.

Source

```
%quick.tex                        Jan 93
\newcount\low\newcount\up\newcount\m
\def\quicksort{%Values given in
%\low,...,\up are sorted, non-descending.
%Parameters: \cmp, comparison.
 \ifnum\low<\up\else\brk\fi
%\refval, a reference value selected at random.
 \m\up\advance\m-\low%Size-1 of ar-
ray part
 \ifnum10<\m\rnd\multiply\m\rndval
   \divide\m99 \advance\m\low \xch\low\m
 \fi
 \ea\let\ea\refval\csname\the\low\endcsname
 \m\low\k\low\let\refvalcop\refval
 {\loop\ifnum\k<\up\advance\k1
   \ea\let\ea\oneqs\csname\the\k\endcsname
   \cmp\refval\oneqs\ifnum1=\status
      \global\advance\m1 \xch\m\k\fi
   \let\refval\refvalcop
  \repeat}\xch\low\m
 {\up\m\advance\up-1 \quicksort}%
 {\low\m\advance\low1 \quicksort}\krb}
%
\def\brk#1\krb{\fi}\def\krb{\relax}
\endinput                     %cgl@rug.nl
```

Explanation. At each level the array is partitioned into two parts. After partitioning the left part contains values less than the reference value and the right part contains values greater than or equal to the reference value. Each part is again partitioned via a recursive call of the macro. The array is sorted when all parts are partitioned.

[61] If the array is big enough. I chose rather arbitrarily 10 as threshold.

In the TEX encoding the reference value as estimate for the mean value is determined via a random selection of one of the elements.[61] Reid's [30] `\rnd` has been used. The random number is mapped into the range $[\,low\,:\,up\,]$, via the linear transformation `\low` + (`\up` − `\low`) ∗ `\rndval`/99.[62]

The termination of the recursion is encoded in a TEX peculiar way. First, I encoded the infinite loop. Then I inserted the condition for termination with the `\fi` on the same line, and not enclosing the main part of the macro. On termination the invokation `\brk` gobbles up all the tokens at that level up to its separator `\krb`, and inserts its replacement text: a new `\fi`, to compensate for the gobbled `\fi`.

Examples (Numbers, words)

```
\def\1{314}\def\2{1}\def\3{27}\n3
\low1\up\n\let\cmp\cmpn
\quicksort
\begin{quote}\prtn,\end{quote}
%
\def\1{ab}\def\2{c}\def\3{aa}
\def\4{\ij}\def\5{ik}\def\6{z}\def\7{a}\n7
\low1\up\n\let\cmp\cmpw
\quicksort
\begin{quote}\prtw,\end{quote}
and
\def\1{j\ij}\def\2{ge\"urm}\def\3{gar\c con}
\def\4{\'el\`eve}\n4
\low1\up\n\let\cmp\cmpw
{\accdef\quicksort}
\begin{quote}\prtw.\end{quote}
```

yields

> 1, 27, 314,
> a aa ab c ik ij z,

and

> élève garçon geürm jij.

Appendix C: The file sort.tex

This file contains the common definitions of `\heapsort` and `\quicksort`, the macros for storing, the macros for sorting, the macros for typesetting, some variants for the parameter macros, and the ordering table.

```
%sort.tex                          Jan 93
%Shorthands
\let\ag=\aftergroup
\let\ea=\expandafter\let\nx=\noexpand
%Counters
\newcount\n\newcount\k\newcount\kk\n=0
\newcount\kzero%Start value in prt k-loops
\newcount\pk\newcount\pkone%Used in sortcs
\newcount\frst%First value of range
\newcount\lst %Last value of range
\newcount\slst%Successor \lst
\newcount\dif %Difference \lst-\frst
\newcount\nw  %Number of words
\newcount\nc  %Number of characters/comp
\newcount\numex %Number of exchanges
\newcount\rndval%Random number
\newcount\rndnum%Seed random generator
\newcount\rndtmp%Temporary value
\newcount\status%Status comparison
%Newif-s
\newif\ifcontinue%controls loops
\newif\iffound%locating accent cs
\newif\ifproof\prooftrue
%
%Storing: from copy
\def\seq#1\qes{\k\kzero\fifow#1 \wofif{} }
%Auxiliaries: FIFO
\def\fifow#1 {\ifx\wofif#1\n\k\wofif\fi
 \processw{#1}\fifow}
\def\wofif#1\fifow{\fi}
\def\processw#1{\advance\k1 \ea
 \gdef\csname\the\k\endcsname{#1}}
%
%Storing: from file
\newread\rec
\def\storefrom#1{%#1 is file name
 \openin\rec#1 \k\kzero \continuetrue
 \loop\ifeof\rec\continuefalse\fi
 \ifcontinue\advance\k1 \read\rec to\xyz
  \ea\let\csname\the\k\endcsname\xyz
 \repeat\advance\k-1\n\k\closein\rec}
%
%Storing: random numbers
\def\storerandomn#1{%#1 number of numbers
 \n#1\k0
 \loop\ifnum\k<\n\advance\k1 \rnd\ea
  \xdef\csname\the\k\endcsname{\the\rndval}
 \repeat}
%
%With, due to Reid, 1987
\def\rnd{\global\multiply\rndnum371
 \global\advance\rndnum1
 \ifnum\rndnum>99999
  \rndtmp\rndnum \divide\rndtmp100000
  \multiply\rndtmp100000
  \global\advance\rndnum-\rndtmp
 \fi\global\rndval\rndnum
 \global\divide\rndval1000 }
%
%Storing: random words
\def\storerandomw#1{%#1 number of words
 \n#1\nw\n\def\defarr{\ea\gdef
  \csname\the\nw\endcsname}
 {\loop\ifnum0<\nw{\ag\defarr\ag{%
   \randomword}}\advance\nw-1
 \repeat}}%end s-r-w.
%
\def\randomword{\rnd \nc\rndval
 \divide\nc15 \advance\nc2
 \loop\ifnum0<\nc\randomchar
   \advance\nc-1
 \repeat}%end r-word
%
%Random character is modified
\def\randomchar{\rnd
 \multiply\rndval29 \divide\rndval100
 \ifnum26=\rndval\rndval0 \fi
 \ifnum26<\rndval\rndval4 \fi
%Mod cgl: I \ag-ed the letter
\ea\ag\ifcase\rndval
 a\or b\or c\or d\or e\or f\or g\or h\or
 i\or j\or k\or l\or m\or n\or o\or p\or
 q\or r\or s\or t\or u\or v\or w\or x\or
 y\or z\fi}%end r-char
%
%Typeset
%Parameters: Separators
\def\sepn{, }%Number separator
```

[62] Note that the number is guaranteed within the range.

```
\def\sepw{ } %Word separator
\let\sep\sepw
%
\def\prc#1{\init{#1}\def\prc##1{%
 \ifnum\lst=##1{}\else\ifnum\slst=##1{}%
  \lst\slst\advance\slst1{}\else
  \prtfl\sepn\init{##1}\fi\fi}}
%
\def\init#1{\frst#1\lst\frst \slst\frst
   \advance\slst1 }
%
%Print range: \frst-\lst (or \lst).
\def\prtfl{\the\frst\ifnum\frst<\lst
 \advance\frst1 \ifnum\frst=\lst\sepn
 \else\nobreak--\nobreak\fi\the\lst\fi}
%
%Printing sequences
\def\prts{{\k\kzero%print \1,...\n
 \def\sep{\let\sep\sepw}%
 \loop\ifnum\k<\n\advance\k1
  \sep\csname\the\k\endcsname
 \repeat}}%end \prts
%
\let\prtw\prts
%
\def\prtn{{\k\kzero%Print number sequence
 \loop\ifnum\k<\n\advance\k1
  \ea\prc\csname\the\k\endcsname
 \repeat\prtfl}}%end \prtn
%
\def\typind#1{%#1 a def
 \ea\splittot#1.%
 \ifcase\digit\word\or
        {\tt\word}\or
  {\tt\char92\word}\or
  $\langle\hbox{\word}\rangle$\fi{}
  \pagenrs}
%
\def\splittot#1 !#2 #3.{\def\word{#1}%
  \chardef\digit#2{}\def\pagenrs{#3}}
%
\def\prtind{{\def\\{\hfil\break}\k\kzero
 \def\sep{\let\sep\sepw}%
 \loop\ifnum\k<\n\advance\k1
  \sep\ea\typind\csname\the\k\endcsname
 \repeat}}
%
%Sorting in O(nlog n)
\def\sortn{\let\cmp\cmpn\sort\prtn}
%
\def\sortaw{\let\cmp\cmpaw\sort\prtw}
%
\def\sortw{\let\cmp\cmpw{\accdef\sort}\prtw}
%
\def\sort{\heapsort}
%
%Paramaters: ij and accent string
\def\accstr{\`\'\"\^\c}
%
\def\accdef{\def\i{i}\def\j{j}%
 \def\'##1{##1a}\def\`##1{##1g}%
 \def\"##1{##1t}\def\^##1{##1h}%
 \def\c##1{##1c}}
%
\def\ij{ij}
%
%Sorting parameters: exchange macro
\def\xch#1#2{%#1, #2 counter variables
 \ea\let\ea\auxone\csname\the#1\endcsname
 \ea\let\ea\auxtwo\csname\the#2\endcsname
 \ea\global\ea\let\csname\the#2\endcsname
 \auxone
 \ea\global\ea\let\csname\the#1\endcsname
 \auxtwo}
%
%Sorting parameters: number comparison
\def\cmpn#1#2{%#1, #2 are def-s
%Result: \status= 0, 1, 2, if
%        \val{#1} =, >, < \val{#2}
 \ifnum#1=#2\global\status0 \else
   \ifnum#1>#2\global\status1 \else
                \global\status2 \fi\fi}
%
%Parameters: comparison of words
\def\cmpw#1#2{%#1, #2 are def-s
%Result: \status= 0, 1, 2, if
%        \val{#1} =, >, < \val{#2}
 \let\nxt\nxtw\cmpc#1#2}
%
\def\cmpaw#1#2{%#1, #2 are defs with as
%replacement text the words.
%Result: \status= 0, 1, 2, if
%        \val{#1} =, >, < \val{#2}
 \let\nxt\nxtaw\cmpc#1#2}
%
\def\cmpc#1#2{%#1, #2 are def-s
%Result: \status= 0, 1, 2, if
%        \val{#1} =, >, < \val{#2}
\ifproof\global\advance\nc1
        \let\aa#1\let\bb#2\fi
 \global\status0 \continuetrue
 {\loop\ifx\empty#1\continuefalse\fi
       \ifx\empty#2\continuefalse\fi
  \ifcontinue\nxt#1\nxtt\nxt#2\nxtu
            \lge\nxtt\nxtu
  \repeat}\ifnum0=\status
 \ifx\empty#1\ifx\empty#2\else
                 \global\status2 \fi
 \else\ifx\empty#2\global\status1 \fi
 \fi\fi
 \ifproof\immediate\write16{\aa
  \ifnum0=\status=\else
   \ifnum1=\status>\else
                  <\fi\fi\bb.}
 \fi%end ifproof
}
%
\def\lge#1#2{%#1 and #2 letter values
%Result: \status= 0, 1, 2, if
%              #1 =, >, <  #2.
%and \continuefalse if #1=/#2.
 \ifnum#1=#2{}\else\continuefalse
  \ifnum#1<#2\global\status2 \else
            \global\status1 \fi
 \fi}
%
\def\nxtw#1#2{\def\pop##1##2\pop{%
 \gdef#1{##2}\def\head{##1}}%head and tail
 \ea\pop#1\pop%split in head and tail
 \ea\loc\head\accstr%\head is an accent cs?
 \iffound\let\acs\head
  \ea\pop#1\pop%next head and tail
  \ea\let\ea#2\csname ot\acs\head\endcsname
 \else\ea\let\ea#2\csname ot\head\endcsname
 \fi}
%
\def\loc#1#2{\def\locate##1#1##2\end
 {\ifx\empty##2\empty\foundfalse
 \else\foundtrue\fi}\ea\locate#2.#1\end}
%
%Parameters:  for ASCII words
\def\nxtaw#1#2{%Result: value of first
%letter of string supplied in #1 is delivered
%in #2. (To be used as a number (\chardef)).
%#1, #2 are control sequences.
 \def\pop##1##2\pop{\gdef#1{##2}%
  \chardef#2`##1{}}\ea\pop#1\pop}
%
```

```
\def\cmpir#1#2{%#1, #2 defs
%Result: \status= 0, 1, 2 if
%        \val{#1} =, >, < \val{#2}
 \ea\ea\ea\decom\ea#1\ea;#2.}
%
\def\decom#1 !#2 #3;#4 !#5 #6.{%
 \def\one{#1}\def\four{#4}\cmpaw\one\four
 \ifnum0=\status%Compare secondary keys
   \ifnum#2<#5{}\global\status2 \else
     \ifnum#2>#5{}\global\status1 \else
                %Compare tertiary keys
       \ifnum#3<#6{}\global\status2 \else
         \ifnum#3>#6{}\global\status1 \fi
       \fi
     \fi
   \fi
 \fi}
%
\def\red{%Reduction of \1,...,\n
 \k0\kk0\let\refer\empty
 \loop\ifnum\k<\n\advance\k1
  \ea\let\ea\record\csname\the\k\endcsname
  \ea\splitwn\record.%
  \ifx\refer\word%extend with number
    \ea\xdef\csname\the\kk\endcsname{%
       \csname\the\kk\endcsname, \num}%
  \else%write record to \kk
   \advance\kk1\let\refer\word\ea\global
   \ea\let\csname\the\kk\endcsname\record
  \fi
 \repeat\n\kk}
%
\def\redrng{%Reduction of \1,...,\n, with
%range representation of page numbers
 {\k1\kk0
 \ea\let\ea\record\csname\the\k\endcsname
 \ea\splitwn\record.\let\refer\word
 \let\nrs\empty\prcrng\num
 \loop\ifnum\k<\n\advance\k1
  \ea\let\ea\record\csname\the\k\endcsname
  \ea\splitwn\record.%
  \ifx\refer\word%extend \nrs with number
    \prcrng\num
  \else%write record to \kk
    \advance\kk1 \strnrs
    \ea\xdef\csname\the\kk\endcsname{\refer{}
      \nrs}\let\nrs\empty\init\num\prcrng\num
    \let\refer\word
  \fi
 \repeat\ifnum1<\n
  \advance\kk1 \strnrs
  \ea\xdef\csname\the\kk\endcsname{\word{}
    \nrs}
 \global\n\kk\fi}}
%
\def\prcrng#1{\init{#1}\def\prcrng##1{%
 \ifnum##1=\lst\else\ifnum##1=\slst
  \lst\slst\advance\slst1 \else
  \strnrs\init{##1}\fi\fi}}
%
\def\strnrs{\dif\lst\advance\dif-\frst
 \edef\nrs{\ifx\nrs\empty\else\nrs\sepn\fi
  \the\frst\ifnum0<\dif
   \ifnum1=\dif\sepn\the\lst
   \else\nobreak--\nobreak\the\lst
   \fi
  \fi}}
%
\def\splitwn#1 !#2 #3.{\def\word{#1 !#2}%
 \def\num{#3}}
%
\def\getdig#1 !#2 #3.{\def\dig{#2}}
%
\def\sortcs{\global\k0\global\pk\n
\global\pkone\pk\global\advance\pkone1
%Invariant: 1:k non-cs; pk+1:n control seq-s
\loop\global\advance\k1
\ifnum\k<\pkone
 \ea\ea\ea\getdig\csname\the\k\endcsname.%
 \if2\dig{\continuetrue
  \loop
   \ifnum\k=\pk\continuefalse
   \else\ea\ea\ea\getdig\csname\the\pk
                          \endcsname.%
    \if2\dig\else\xch\k\pk\continuefalse\fi
   \fi\global\pkone\pk\global\advance\pk-1
  \ifcontinue
  \repeat}%
 \fi
\repeat}%Result\1:\pk non-cs, \pkone:\n cs
%
%Parameters: Ordering table
\chardef\ota32 \chardef\otA32
 \chardef\otaa33 \chardef\otag33
 \chardef\otat34 \chardef\otah35
\chardef\otb39 \chardef\otB39
\chardef\otc46 \chardef\otC46
 \chardef\otcc47 \chardef\otcc47
\chardef\otd53 \chardef\otD53
\chardef\ote60 \chardef\otE60
 \chardef\otea61 \chardef\oteg62
 \chardef\otet63 \chardef\oteh64
\chardef\otf67 \chardef\otF67
\chardef\otg74 \chardef\otG74
\chardef\oth81 \chardef\otH81
\chardef\oti88 \chardef\otI88
 \chardef\otit91 \chardef\otih92
\chardef\otj95 \chardef\otJ95
 \chardef\otjt98
\chardef\otk102 \chardef\otK102
\chardef\otl109 \chardef\otL109
\chardef\otm116 \chardef\otM116
\chardef\otn123 \chardef\otN123
\chardef\oto130 \chardef\otO130
 \chardef\otoa131 \chardef\otog132
 \chardef\otot133 \chardef\otoh134
\chardef\otp137 \chardef\otP137
\chardef\otq143 \chardef\otQ143
\chardef\otr150 \chardef\otR150
\chardef\ots157 \chardef\otS157
\chardef\ott164 \chardef\otT164
\chardef\otu171 \chardef\otU171
 \chardef\otut174 \chardef\otuh175
\chardef\otv178 \chardef\otV178
\chardef\otw185 \chardef\otW185
\chardef\otx192 \chardef\otX192
\chardef\otij199 \chardef\otIJ199
\chardef\oty200 \chardef\otY200
\chardef\otz206 \chardef\otZ206
\endinput                   %cgl@rug.nl
```

Appendix D: The file sort.tst

Writing macros is one thing and testing another. I find testing software as difficult as writing a variant from scratch. For convenience I have provided my (plain) testdriver below.

The test path—Which sorting worker? Tracing on/off? How many random data?—is determined in a dialogue with TEX. Rudimentary, but useful.

```
%sort.tst                           Jan 93
%Separately needed is index.tex, as data.
\input sort.tex
\input heap.tex
```

```
\input quick.tex
\immediate\write16{Heap sort as sorter? (y/n):}
\read16 to\yesorno
\if y\yesorno Heap sort.
   \def\sort{\heapsort}
\else Quick sort.
   \def\sort{\low1\up\n\quicksort}
\fi (\number\day/\number\month/\number\year)
\immediate\write16{Proofing/Tracing? (y/n):}
\read16 to\yesorno
\if y\yesorno\prooftrue
 \nopagenumbers\tracingmacros2
\else\prooffalse
\fi

\smallskip Numbers.\par
\seq314 1 27\qes
Input: \prtn.\par
Result:\sortn.
\immediate\write16{Result: \1, \2, \3,
           ...\csname\the\n\endcsname.}

\smallskip Words (ASCII).\par
\seq a b aa ab bc bb aaa\qes
Input: \prtw.\par
Result: \sortaw.
\immediate\write16{Result: \1, \2, \3,
\4, \5, \6,...\csname\the\n\endcsname.}

\smallskip Words.\par
\seq a b aa ab bc bb aaa\qes
Input: \prtw.\par
Result: \sortw.
\immediate\write16{Result: \1, \2, \3,
\4, \5, \6 ...\csname\the\n\endcsname.}

\smallskip Accented words.\par
\def\1{z}\def\2{c}\def\3{\'a\'a}\def\4{\"ab}
\def\5{ge\"urm}\def\6{ge\"{\i}nd}
\def\7{gar\c{c}on}\def\8{a}\def\9{ge{\ij}kt}
\n=9
Input: \prtw.\par
Result: \sortw.
\immediate\write16{Result: \1, \2, \3,
           ...\csname\the\n\endcsname.}

%Test and timing: random generated elements
\smallskip Sort numbers.\par
\immediate\write16{Give seed for r-generator:}
\read16 to\seed
\immediate\write16{Give maximum of num-
bers to be
   generated:}
\read16 to\total \n\total
Seed=\seed. \rndnum\seed
\storerandomn\n \par
Input: \prtn.\par
Result: \sortn.
\immediate\write16{Result: \1, \2, \3,
           ...\csname\the\n\endcsname.}

\smallskip Sort words. \par
\immediate\write16{Give seed for r-generator:}
\read16 to\seed
\immediate\write16{Give maximum of words to be
   generated:}
\read16 to\total \n\total
Seed=\seed. \rndnum\seed
\storerandomw\n \par
Input: \prtw.\par
Result: \sortw.
\immediate\write16{Result: \1, \2, \3,
           ...\csname\the\n\endcsname.}

\smallskip Sort index reminders.\par
\storefrom{index.tex}
{\def\\{\hfil\break}\let\sepw\\
\let\cmp\cmpir\k0\kk0 \null
\hfil\vtop{\hsize2.25cm\noindent
  Data:\sepw\prtw}
\hfil\vtop{\hsize2.5cm\sort\noindent
  After sorting:\sepw\prtw}
\hfil\vtop{\hsize3.5cm\redrng\noindent
  After reduction:\sepw\prtw}
\hfil\vtop{\hsize3cm\noindent
  Typeset:\sepw\prtind.}
\immediate\write16{Index rem: \1, \2, \3,
           ...\csname\the\n\endcsname.}

\smallskip Frambach's example.\par
\def\1{wd !2 7}\def\2{wrd !1 1}
\def\3{wd !2 2}\def\4{a !1 1}
\def\5{wd !2 5}\def\6{wd !2 3}
\def\7{z !3 7}\def\8{wrd !1 5}
\def\9{wd !2 1} \n9
\let\sepw\\\null
\hfil\vtop{\hsize2cm\noindent
 Data:\sepw\prtw}
\hfil\vtop{\hsize2.5cm\sortcs\noindent
 After splitting:\sepw{\n\pk\prtw}
  \sepw\kzero\pk\prtw}
\hfil\vtop{\hsize3cm\let\cmp\cmpir
 {\low1\up\pk\quicksort}
 {\low\pkone\up\n\quicksort}\noindent
 After sorting\sepw both parts,\sepw
 and compressing:\sepw\redrng\n4 \prtw}
\hfil\vtop{\hsize3cm\noindent\n4
  Typeset:\sepw\prtind.}
}
\immediate\write16{EF's exam: \1, \2, \3,
             ...\csname\the\n\endcsname.}
\bye                          cgl@rug.nl
```

Appendix E: Contents