Sorting within TEX

Kees van der Laan
Hunzeweg 57, 9893PB
Garnwerd, The Netherlands
05941 1525
Internet: cgl@riscl.rug.nl.

Abstract

It is shown how sorting — numbers and lexicographic — can be done completely within TEX. Lexicographic sorting allows words with ligatures and diacritical marks. As applications I selected sorting of address labels, and sorting and compressing `index.tex`, Knuth's index reminders file. It is claimed that a set can be sorted within TEX once the ordering of the set is defined and encoded in a comparison macro, in compliance with the parameter macro `\cmp`.

Introduction

This paper is an abridged version of the original "Sorting in BLUe" which has appeared in NTG's MAPS93.1. In this version I strove to get across the flavour of what can be done within TEX with respect to sorting.

The original version — not limited by space, some 20 pages — contains the in-depth treatment, the detailed explanations, the various abstractions (parameterization over the comparison operation, sorting process, lexicographic ordering, and the handling of accents and ligatures), more examples, all the macro listings, as well as a listing of my test driver. Apart from sorting the storing of the data and the final typesetting — with its parameter for separation, and for numbers the use of range notation — is dealt with in that paper. I also included an extensive list of references.

In this paper I'll show, and now and then explain, how to use the Ben Lee User level sorting macros `\sortn` — for number sorting, `\sortaw` — for sorting ASCII words, and `\sortw` — for general lexicographic sorting. At the lower level I provided in the appendices the blue collar macros `\heapsort` and `\quicksort`.

Definitions and notations. A sequence is defined as a row of numbers or words, respectively, separated by spaces. The structure `\csname`$\langle k\rangle$`\endcsname` is associated with an array with index $k = 1, 2, \ldots, n$. To denote in the documentation a value pointed by the number $\langle k\rangle$, I made use of `\val{`$\langle k\rangle$`}`, with `\def\val#1{\csname#1\endcsname}`. Macro names take suffix -n, -w, when specific for number and word data respectively. For example, `\sortn` stands for sort numbers, `\prtw` stands for print words. I have typeset the in-line results of the examples in bold face.

I have used the shorthand notation `\ea`, `\nx`, and `\ag` for `\expandafter`, `\noexpand`, and `\aftergroup`, respectively. `\k` is used as counter to loop through the values $1, 2, \ldots, n$, the index domain. `\n` contains the maximum number of sequence elements, n. `\ifcontinue` is used for controlling loops. The macro `\seq` with end separator `\qes` stores the supplied data in the array.

For typesetting the data structure I used the macros `\prtn` and `\prtw`, respectively. These are not explained here either. Loosely speaking they typeset the array, `\1...\`$\langle n\rangle$ which contains the items, as you would expect.

Some background. The reader must be aware of the differences between

- the index number, $\langle k\rangle$
- the counter variable `\k`, with the value $\langle k\rangle$ as index number
- the control sequences `\`$\langle k\rangle$, $k = 1, 2, \ldots, n$, with the items to be sorted as replacement texts.

When we have `\def\3{4} \def\4{5} \def\5{6}` then `\3` yields **4**,
`\csname\3\endcsname` yields **5**, and
`\csname\csname\3\endcsname\endcsname` yields **6**.

Similarly, when we have `\k3 \def\3{name} \def\name{action}` then `\the\k` yields **3**,
`\csname\the\k\endcsname` yields **name**, and
`\csname\csname\the\k\endcsname\endcsname` yields **action**.[1] To exercise shorthand notation the last can be denoted by `\val{\val{\the\k}}`.

1. Confusing, but powerful!

Another `\csname...` will execute `\action`, which can be whatever you provided as replacement text.

Sorting of Numbers

Sorting of numbers alone is not that hard to do within TeX. To design a consistent, orthogonal, well-parameterized, and nevertheless as simple as possible set of macros is the challenge, which I claim to have attained.

Example of use. After `\input heap \input sort`

```
\seq314 1 27\qes\sortn
```

yields: **1, 27, 314.**

Design choices. The backbone of my 'sorting in an array' is the data structure

`\csname`$\langle k\rangle$`\endcsname{`$\langle k^{th} elm.\rangle$`}`, $k = 1, 2, \ldots, n$,

with k the role of array index and n the number of items to be sorted.

This encoding is parameterized by `\cmp`, the comparison macro, which differs for numbers, strings, and in general when more sorting keys have to be dealt with. The result of the comparison is stored globally in the counter `\status`.

Encoding: Input. The elements are assumed to be stored in the array `\`$\langle k\rangle$, $k = 1, 2, \ldots n$. The counter `\n` must contain the value $\langle n\rangle$.

Encoding: Result. The sorted array `\1`, `\2`, ..., `\`$\langle n\rangle$, with `\val1` ≤ `\val2` ≤ ... ≤ `\val`$\langle n\rangle$.

Encoding: The macros.

```
\def\sortn{\let\cmp\cmpn\sort\prtn}
%
\def\cmpn#1#2{%#1, #2  ex-
pand into numbers
%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}
%
\def\sort{\heapsort}.
```

Encoding: Explanation. `\cmpn` has to be defined in compliance with the parameter macro `\cmp`. `\sort` must reference to one of the blue collar sorters. `\prtn` typesets the numbers. That is all.

The above shows the structure of each of the Ben Lee User sorting macros.

Sorting: `\sortn`. A (pointer) `\def\sortn{...}` is introduced which has as replacement text the setting of the parameter `\cmp`, and the invocations of the actual sorting macro and the macro for typesetting the sorted sequence.

Comparison operation: `\cmpn`. The result of the comparison is stored globally in the counter `\status`. The values 0, 1, 2 denote =, >, <, respectively.

Exchange operation: `\xch`. The values can be exchanged via

```
\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}.
```

The macro for typesetting a sequence of numbers in range notation is provided in the full paper as well as in the special short paper about typesetting number sequences, which has also appeared in NTG's MAPS93.1.

Lexicographic Sorting

Given the blue collar workers `\heapsort` and `\quicksort`, respectively, we have to encode the comparison macro in compliance with the parameter macro `\cmp`. But lexicographic sorting is more complex than number sorting. We lack a general comparison operator for strings,[2] and we have to account for ligatures and diacritical marks.

In creating a comparison macro for words, flexibility must be built in with respect to the ordering of the alphabet, and the handling of ligatures and diacritical marks.

Example of use: Sorting ASCII words.
After `\input heap \input sort`

```
\seq a b aa ab bc bb aaa\qes\sortw
```

yields: **a aa aaa ab b bb bc.**

Example of use: ij-ligatures.
After `\input heap \input sort`

```
\seq{\ij}st{\ij}d {\ij} {\ij}s in tik
     t\ij\qes\sortw
```

yields: **in tik tij ij ijs ijstijd.**

Example of use: Sorting accented words.
After `\input heap \input sort`

```
\seq b\'e b\`e \'a\'a ge\"urm geur aa a
     ge{\ij}kt be ge\"\i nd gar\c con\qes
```

2. It is not part of the language, nor provided in plain or elsewhere with the generality I needed.

```
\sortw
```

yields: **a aa áá be bé bè garçon geïnd geur geürm geijkt**.

Because of the complexity and the many details involved I restricted myself in this paper to the simplified cases: one-(ASCII)letter-words, and ASCII strings, of undetermined length.

One-(ASCII)letter-words. The issue is to encode the comparison macro, in compliance with the parameter macro `\cmp`. Let us call this macro `\cmpolw`.[3] Its task is to compare one-letter words and store the result of each comparison globally in the counter `\status`. As arguments we have `\def`s with one letter as replacement text.

```
\def\cmpolw#1#2{%#1, #2 are defs
%Result: \status= 0, 1, 2 if
%        \val{#1} =, >, < \val{#2}.
 \ea\chardef\ea\cone\ea'#1{}%
 \ea\chardef\ea\ctwo\ea'#2{}%
 \global\status0 \lge\cone\ctwo}
%
\def\lge#1#2{%#1, #2 are letter values
%Result: \status= 0, 1, 2 if #1 =, >, < #2.
 \ifnum#1>#2\global\status1 \else
   \ifnum#1<#2\global\status2 \fi\fi}
```

Example of use. After the above

```
\seq z y A B a b d e m n o p z z u v c g
     q h j I i l k n t u r s f Y\qes
\let\cmp=\cmpolw\sort\prtw
```

yields: **A B I Y a b c d e f g h i j k l m n n o p q r s t u u v y z z z**.

Explanation `\cmpolw`. In order to circumvent the abundant use of `\expandafter`s, I needed a two-level approach: at the first level the letters are 'dereferenced', and the numerical value of each replacement text is provided as argument to the second level macro, `\lge`.[4]

ASCII words. The next level of complexity is to allow for strings, of undetermined length and composed of ASCII letters. Again the issue is to encode the comparison macro, in compliance with `\cmp`. Let us call the macro `\cmpaw`.[5] Its task is to compare ASCII words and to store the result of each comparison globally in the counter `\status`.

The problem is how to compare strings letter by letter. Empty strings are equal. This provides a natural initialization for the `\status` counter. As arguments we have `\def`s with words of undetermined length as replacement text.

```
\def\cmpaw#1#2{%#1, #2 are defs
%Result: \status= 0, 1, 2 if
%        \val{#1} =, >, < \val{#2}.
 {\let\nxt\nxtaw\cmpc#1#2}}
%
\def\cmpc#1#2{%#1, #2 are defs
%Result: \status= 0, 1, 2 if
%        \val{#1} =, >, < \val{#2}.
\global\status0 \continuetrue
{\loop\ifx#1\empty\continuefalse\fi
      \ifx#2\empty\continuefalse\fi
 \ifcontinue\nxt#1\nxtt \nxt#2\nxtu
     \lge\nxtt\nxtu
     \ifnum0<\status\continuefalse\fi
 \repeat}\ifnum0=\status
 \ifx#1\empty\ifx#2\empty\else
                  \global\status2 \fi
 \else\ifx#2\empty\global\status1 \fi
 \fi\fi}
%
\def\nxtaw#1#2{\def\pop##1##2\pop{\gdef
 #1{##2}\chardef#2'##1}\ea\pop#1\pop}
```

Example of use. After the above

```
\seq a b aa ab bc bb aaa\qes
\let\cmp\cmpaw\sort\prtw
```

yields: **a aa aaa ab b bb bc**.

Explanation comparison: `\cmpaw`. The macro is parameterized over the macro `\nxt`. The main part of `\cmpaw` has been encoded as `\cmpc`.[6] (That part is also used in the general case.)

We have to compare the words letter by letter. The letter comparison is done by the already available macro `\lge`. The `\lge` invocation occurs within a loop, which terminates when either of the strings has become empty. I added to stop when the words considered so far are unequal. At the end the status counter is corrected if the words considered are equal and one of the # is not empty: into 1, if #1 is not empty, and into 2, if #2 is not empty.

Explanation head and tail: `\nxt`. The parameter macro `\nxt` has the function to yield from the replacement text of its first argument the ASCII value

3. Mnemonics: compare one letter words.
4. Mnemonics: letter greater or equal. A nice application of the use of `\ea`, `\chardef`, and the conversion of a character into a number via the quote: **'**. Note that the values of the uppercase and lowercase letters differ (by 32) in ASCII.
5. Mnemonics: compare ASCII words.
6. Mnemonics: compare character.

of the first letter and deliver this value as replacement text of the second argument.[7] The actual macro \nxtaw pops up the first letter and delivers its ASCII value — a \chardef — as replacement text of the second argument.

Sorting Address Labels

Amy Hendrickson 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.

For the imaginative toy addresses of the three composers: Schönberg, Webern, Strawinsky, I used the following definitions.

```
\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}}
%
%the list with active list separator \as
%to be defined by the user
\def\addresslist{\as\strawinskyi
 \as\weberna\as\schonberga}
```

For the typesetting, I made use of the following simple address label format

```
\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: Selection of addresses per country. Suppose we want to select (and just \tsa them for simplicity) the inhabitants from Holland from our list. This goes as follows.

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

with result

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

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, alphabetic-
ally ordered
\k0
\loop\ifnum\k<\n\advance\k1
 \csname\csname\the\k\endcsname\endcsname
 \vskiplex\tsa
\repeat
```

with result

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

7. Splitting up into 'head and tail' is treated in the TEXbook, Appendix D.2, p.378, the macro \lop. There, use has been made of token variables instead of \defs.

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

⟨*word*⟩␣!⟨*digit*⟩␣⟨*page number*⟩.

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[8] 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: ⟨*word*⟩
secondary key: ⟨*digit*⟩, and
tertiary key: ⟨*page number*⟩.

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

Example: Sorting on primary, secondary 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[9]

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		

Design. Given the sorting macros we have to encode the special comparison macro in compliance with `\cmpw`: compare two 'values' specified by `\def`s. Let us call this macro `\cmpir`.[10] Each value is composed of: a word (action: word comparison), a digit (action: number comparison), and a page number (action: (page) number comparison).

Then we have to account for the reduction of 'duplicate' index entries, and finally the typesetting has to be done.

The comparison. I needed a two-level approach. The values are decomposed into their components by providing them as arguments to `\decom`.[11] The macro picks up the components
-the primary keys, the ⟨*word*⟩,
-the secondary keys, the ⟨*digit*⟩, and
-the tertiary keys, the ⟨*page number*⟩.
It compares the two primary keys, and if necessary the two secondary and the two tertiary keys successively. 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`.

The comparison macro.

```
\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}
```

Reducing duplicate word-digit entries. The idea is that the same index entries, except for their page

8. 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.
9. The unsorted input can be read from the verbatim listing.
10. Mnemonics: compare index reminders.
11. Mnemonics: decompose. In each comparison the defs are 'dereferenced', that is, their replacement texts are passed over. This is a standard TEXnique: a triad of `\ea`s, and the hop-over to the second argument.

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[12] 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: reduction of entries. The encoding is complicated because while looping over the index reminders either the reminder in total or just the page number has to be handled. The handling of the page numbers is done with modified versions of \prc, \prtfl, called respectively \prcrng and \strnrs.[13] 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.

Typesetting index entries. Knuth has adopted the following conventions for coding index entries.

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

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

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.

Epilogue

No robustness was sought. The encodings have been kept as simple and flexible as possible. 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!

Bibliography

Laan, C.G van der. "Sorting in BLUe.". MAPS93.1, 149-169, 1993.

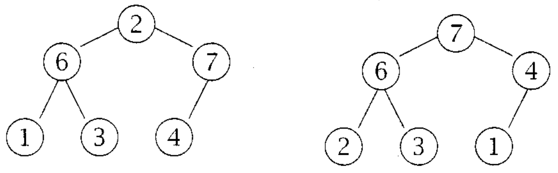
12. Mnemonics: reduce (in range notation).
13. Mnemonics: process with ranges, respectively store numbers.

Appendix: Heap Sort

The process consists of two main steps: creation of a heap, and sorting the heap. A sift operation is 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.[14]

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 $a_k > a_{n+1}, k = 1, 2, \ldots, n$.
For example, a tree and one of its heap representations of 2, 6, 7, 1, 3, 4 read



The algorithm. In a pseudo notation the algorithm, 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.

Encoding: Input. The values are stored in the control sequences \1, ..., \⟨n⟩. The counter \n must contain the value ⟨n⟩. 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, see van der Laan (1993).)

Encoding: Output. The sorted array \1, \2, ... \⟨n⟩, with \val1 ≤ \val2 ≤ ... ≤ \val⟨n⟩.

Encoding: 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 \sift\lc\n\repeat}}
%
\def\sift#1#2{%#1, #2 counter variables
```

[14] 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.

```
 \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@riscl.rug.nl
```

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

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

Explanation: `\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.

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

Examples of use: Numbers, words. After `\input heap \input sort`

```
\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[15]

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

and

> élève garçon geürm jij.

[15] `\accdef` suitably redefines the accents within this scope.

Appendix: Quick Sort

The quick sort algorithm has been discussed in many places. Here the following code due to Bentley has been transliterated.[16]

```
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 \⟨*low*⟩, ..., \⟨*up*⟩.

Encoding: Input. The values are stored in \⟨*low*⟩, ..., \⟨*up*⟩, with $1 \le low \le up \le n$. 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`, see van der Laan (1993).)

Encoding: Output. The sorted array \⟨*low*⟩, ...\⟨*up*⟩, with `\val`⟨*low*⟩ ≤ ... ≤ `\val`⟨*up*⟩.

Encoding: 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 array 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@riscl.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.

In the TEX encoding the reference value as estimate for the mean value is determined via a random selection of one of the elements. The random number is mapped into the range [*low* : *up*], via the linear transformation `\low` + (`\up` − `\low`) ∗ `\rndval`/99.[17]

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 invocation `\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`.

[16] L, U have been changed in the TEX code into low, up.

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

Examples: Numbers, words. After `\input quick \input sort`

```
\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[18]

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

and

> élève garçon geürm jij.

[18] `\accdef` suitably redefines the accents within this scope.