Heap Sort in TEX

Kees van der Laan

Hunzeweg 57,
9893 PB Garnwerd, The Netherlands
cgl@rug.nl

September 1992

Abstract

Sorting in plain TEX is implemented via heap sort. The heap sort algorithm is explained and the encoding given.

Keywords: (heap) sort, array addressing, plain TEX, macro writing, education.

1 Introduction

A TEX encoding of the heap sort algorithm is given. A simpler user interface and examples of use, also with respect to lexicographic sorting, will be treated in Sorting in BLUe, to come.

2 Example of use

```
\ea\def\csname1\endcsname{27}
\ea\def\csname2\endcsname{1}
\ea\def\csname3\endcsname{314} \n=3
\heapsort
\csname1\endcsname,
\csname2\endcsname,
\csname3\endcsname.
```

yields 314, 27, 1.

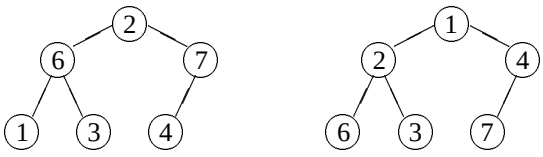
3 Heap sort

The process consist of two main steps

- creation of a heap
- sorting the heap

with a sift operation to be used in both.

What is a heap? A sequence $a_1, a_2, \ldots, a_n$, is a heap if $a_k \leq a_{2k} \wedge a_k \leq a_{2k+1}, k = 1, 2, \ldots, n \div 2$, and because a_{n+1} is undefined, it is defined that $a_k < a_{n+1}$ is true. A tree and one of its heap representations of $2, 6, 7, 1, 3, 4$, is displayed below



3.1 The algorithm

In PASCAL-like notation the algoritm reads
%heap creation in '1:n'
$l := n$ **div** $2 + 1;$
while $l \neq 1$ **do** $l := l - 1; sift(a, l, n)$ **od**
%sorting in '1:n'
$r := n;$ **while** $r \neq 1$ **do**
$exchange(a[1], a[r]) r := r - 1; sift(a, 1, r)$ **od**
%sift #1 through #2
$j := \#1$
while $2j \leq \#2 \wedge (a[j] > a[2j] \vee a[j] > a[2j + 1])$ **do**
$mi := 2j +$ **if** $a[2j] < a[2j + 1]$ **then** 0 **else** 1 **fi**
$exchange(a[j], a[mi]) j := mi$ **od**

3.2 TEX encoding

```
\let\ea=\expandafter          %28/9/92
\newcount\n\newcount\l\newcount\r
\newcount\i\newcount\uone
\newcount\j\newcount\jj\newcount\jjone
\newif\ifcmp\newif\ifgoon \newif\iflt
\def\heapsort{%data in \1 to \n
\r=\n \heap \i=1 %
\loop\ifnum\r>1 \exchange\i\r%
   \advance\r by-1 {\sift\i\r}%
\repeat}
%
\def\heap{%Transform \1..\n into heap
\l=\n\divide\l by2 \advance\l by1 %
\loop\ifnum\l>1 %
   \advance\l by-1 {\sift\l\n}%
\repeat}
%
\def\cmpval#1#2{%#1, #2 counter variables
\xdef\aone{\csname\the#1\endcsname}%
\xdef\atwo{\csname\the#2\endcsname}%
\cmptrue\ifnum\aone>\atwo{}\cmpfalse\fi}
%
\def\exchange#1#2{%#1, #2 counter variables
\edef\aux{\csname\the#1\endcsname}%
\ea\xdef\csname\the#1\endcsname{\csname
\the#2\endcsname}%
```

```
\ea\xdef\csname\the#2\endcsname{\aux}}
%
\def\sift#1#2{%#1, #2 counter variables
\jj=#1 \uone=#2 \advance\uone1 \goontrue%
\loop \j=\jj \advance\jj by\jj%
 \ifnum\jj<\uone%
   \jjone=\jj \advance\jjone by1%
   \ifnum\jj<#2{}\cmpval\jj\jjone%
         \ifcmp\else\jj=\jjone\fi\fi%
   \cmpval\j\jj\ifcmp\goonfalse\fi%
 \else\goonfalse\fi%
\ifgoon\exchange\j\jj\repeat}
```

heapsort
The values of `\1`,...,`\n`, are sorted in descending order: `\1` $\geq$ `\2` $\geq \cdots \geq$ `\n`.

heap
The values `\1`,...,`\n`, are rearranged into a heap.

sift
The first element denoted by the first (counter) argument has disturbed the heap, because of the exchange of the first and the last (one higher than the second argument) element. **sift** sifts this first element through the heap until the heap property holds again.