Heapsort.

Heapsort is an efficient algorithm for sorting in situ. For brevity's sake we shall sort integers. We take for Heapsort the following functional specification:

|[N: int $\{N \geq 1\}$
; $m(i: 0 \leq i < N)$: array of int
$\{BM$: bag of int such that $P0$: $BM = (\underline{B}\, i: 0 \leq i < N: m(i))\}$
; Heapsort
$\{m$ such that R: $P0 \wedge (\underline{A}\, i,j: 0 \leq i < j \wedge 1 \leq j < N: m(i) \leq m(j))\}$
]|

The above states no more than that Heapsort is a sorting algorithm.

* * *

We shall approach Heapsort by numbered versions, starting with Heapsort0.

Heapsort0:
|[q: int
; $q := N$
$\{$invariant $P1$:
$1 \leq q \leq N \wedge P0 \wedge (\underline{A}\, i,j: 0 \leq i < j \wedge q \leq j < N: m(i) \leq m(j))\}$
; *[$q \neq 1$
$\rightarrow$ S0 $\{m, q$ such that $P1$; $\underline{\text{dec}}\ q\}$
]* $\{P1 \wedge q = 1$, hence$\}$
]|

The above states that the sorted sequence is built up "from right to left", i.e. in the order of decreasing subscript.

Our first version of S0 is

S0:
|[p: int
; S1
{ $m(i: 0 \le i < q)$, p such that
$R1: 0 \le p < q \wedge P1 \wedge (\underline{A}\, i: 0 \le i < q : m(p) \ge m(i))$ }
; $q := q-1$; $m{:}swap(p, q)$
]|

An unsophisticated S1 would leave m unchanged and would only locate a maximum value $m(p)$ among the leftmost q elements of m; the N^2-algorithm that would result is known as Bubble Sort. In view of our later transition to Heapsort1 we allow S1 to rearrange $m(i: 0 \le i < q)$ as well in order to establish a relation H about $m(i: p \le i < q)$, to be used as follows in our final version of S0.

S0:
|[p: int
; S1 { $m(i: 0 \le i < q)$, p such that $H \wedge R1$ }
; $q := q-1$; $m{:}swap(p, q)$ { $H(p := p+1)$ }
; $p := p+1$ { H }
]|

In order to justify the last two assertions in the above we require H, besides being an assertion about $m(i: p \leq i < q)$, to satisfy

$$H \Rightarrow H(p, q := p+1, q-1) \qquad (0)$$

In order that H —which is about $m(i: p \leq i < q)$— assist in establishing the last factor of $R1$ —which is about $m(i: 0 \leq i < q)$— we require H to satisfy

$$(H \wedge p = 0) \Rightarrow (\underline{A}\, i: p \leq i < q: m(p) \geq m(i)) \qquad (1)$$

and $S1$ to terminate with $p = 0$. Hence we suggest for $S1$

```
S1:
  p:= h(q)
  {invariant P2: 0≤p   ∧ P1 ∧ H}
; *[ p≠0 → p:= p-1
           {H(p:= p+1)}
           ; S2
           {m(i: p≤i<q) such that H}
  ]*
```

where h is such that

$$(p = h(q)) \Rightarrow (H \wedge 0 \leq p) \qquad . \qquad (2)$$

Substituting S1 in our final version of S0 and substituting the result in Heapsort0, we get a program in which relation H — together with p — can be taken outside the repetition of Heapsort0. The transformation is very likely to improve the efficiency since we can conclude from S1 that S0 restores H with $p=1$, a value very likely to be much smaller than $h(q)$. The result of the transformation is Heapsort1.

```
Heapsort1:
|[ p,q : int
; q:= N  {P1}
; p:= h(q) {invariant P2}
; *[ q≠1
     → {invariant P2}
       *[ p≠0
           → p:= p-1
             {H(p:=p+1)}
             ;S2
             {m(i: p≤i<q) such that H}
       ]*
     ; q:=q-1 ; m: swap(p,q) ; p:=p+1
  ]*
]|
```

Our remaining task is the choice of an appropriate H and the design of the corresponding S2 and h.

A possibility for H would be

$$(\underline{A}\, i,j:\ p \le i < j < q:\ m(i) \ge m(j))$$

but — besides begging the question — it is stronger than necessary since the right-hand side of (1) would be implied by H all by itself. Hence the above suggestion is weakened by requiring $m(i) \ge m(j)$ for a subset of (i,j)-pairs with $i < j$:

$$H:\quad (\underline{A}\, i,j: p \le i < j < q \wedge c(i,j):\ m(i) \ge m(j))$$

Requirement (0) is satisfied; (2) is satisfied by $h(q)=q$. Viewing the natural numbers ($<q$) as the vertices of a directed graph and the truth of $c(i,j)$ as the presence of a directed edge from vertex i to vertex j, (1) is equivalent to the requirement that all vertices are reachable from vertex 0, in formula

$$(\underline{A}\, j: j > 0: (\underline{E}\, i: 0 \le i < j: c(i,j))) \quad . \qquad (3)$$

For the purpose of describing S2, we reformulate H in terms of the transitive closure cc of c :

$$cc(i,j) = c(i,j) \vee (\underline{E}\, k: i < k < j: c(i,k) \wedge cc(k,j))$$

as

$$H:\ (\underline{A}\, i,j: p \le i < j < q \wedge cc(i,j):\ m(i) \ge m(j)) \quad .$$

S2 can then establish H using SH, given by

$$\text{SH:}\quad (\underline{A}\, i,j : p \leq i < j < q \wedge cc(i,j) : m(i) \geq m(j) \vee i = w)\ ,$$

which has the useful property

$$(\text{SH} \wedge (\underline{A}\, j : w < j < q \wedge c(w,j) : m(w) \geq m(j))) \Rightarrow H \quad .$$

A still somewhat abstract form of S2 is

```
S2:
|[ w : int
  {H(p:= p+1)}
; w:= p {invariant SH}
; *[(E j: w<j<q ∧ c(w,j))
     → |[ v : int
        ; S3 {v such that:
                  w<v<q ∧ c(w,v) and m(v) maximal}
       ;-[ m(w) ≥ m(v) → {H} w:=q {SH}
        [] m(w) < m(v) → m: swap(v,w); w:= v {SH, see Note}
        ]-
       ]|
  ]*
]|
```

Note. See page 7.

Our next task is to propose a suitable c . Requirement (3) states that for $j > 0$ the equation in i

$$0 \leq i < j \wedge c(i,j)$$

has at least 1 solution; since nothing is gained by giving it more solutions c will be chosen such that it has exactly 1 solution. In other words, the directed graph we referred to takes the form of a rooted tree. The structure of S2 shows that for given w the solutions j of $c(w,j)$ have to be generated; for reasons of convenience these solutions will be consecutive integers. We propose for some integer d

$$c(i,j) = (i = (j-1) \underline{\text{div}}\ d) \quad .$$

<u>Remark.</u> With $d=2$ we obtain the traditional Heapsort. Since $d=3$ gives a better worst-case behaviour, we present the code for general d. (End of Remark.)

In our following version of S2, h and k are used to delimit the solutions j of $w < j < q \wedge c(w,j)$: they are those of $h \leq j < k$. In the code replacing S3, h is used for scanning. Further optimizations — e.g. reducing the number of subscriptions — are left to the reader.

S2:

```
|[ w, h, k : int
; w := p ; h := d·w + 1 ; k := min(h+d, q)
; *[ h < k
     → |[ v : int
        ; v, h := h, h+1
        ; *[ h < k
              → -[ m(v) ≥ m(h) → h := h+1
                 [] m(v) ≤ m(h) → v, h := h, h+1
                 ]-
          ]*
        ; -[ m(w) ≥ m(v) → skip { h=k }
          [] m(w) < m(v) → m:swap(v,w) ; w := v
                            ; h := d·w + 1 ; k := min(h+d, q)
          ]-
       ]|
  ]*
]|
```

<u>Note</u> (To be inserted on page 5.) Relation SH states that m(w) is the only element that may have descendants exceeding itself. If so, being the only one, m(w) has a son exceeding itself. Because m(v) is a maximum son of m(w), after the swap m(v) is the only element that may have descendants exceeding itself. Note that for this conclusion it was not essential that sons have a unique father. (End of Note.)

References

Williams, J.W.J., Algorithm 232 HEAPSORT,
C.A.C.M. 7,6 (Jun 1964) 347-348

Floyd, Robert W., Algorithm 242 TREESORT 3,
C.A.C.M 7,12 (Dec 1964) 701

drs. A.J.M. van Gasteren
BP Venture Research Fellow
W.H.J. Feijen
Department of Mathematics
and Computing Science
University of Technology
5600 MB EINDHOVEN
The Netherlands

22 June 1981

prof.dr. Edsger W. Dijkstra
Burroughs Research Fellow
Plataanstraat 5
5671 AL NUENEN
The Netherlands