An alternative to Heapsort for sorting in situ.

I am getting a bit tired of sorting algorithms because, since I happen to look at them, I get the impression that their number is unbounded. It is therefore with great hesitation that I record having discovered a next class of them.

An often quoted disadvantage of Heapsort — whether this disadvantage is serious or not is none of my concerns — is that it absolutely fails to exploit the circumstance that the sequence is initially "almost sorted." While sharing the $N \cdot \log N$ characteristic with Heapsort, the alternative does not share with Heapsort this disadvantage — it is, however, of greater clerical complexity —: in the extreme case that the sequence is initially sorted, no rearrangement whatsoever takes place.

For the sake of brevity I shall consider sorting the integer array $m(c: 0 \leq c < N)$ in ascending order. For integer $p$ the following two relations are of interest.

$P_0 : (\forall i, j : 0 \leq i < p \land 0 \leq j < N : m(i) \leq m(j))$.

It expresses that the first $p$ elements of $m$ have their final value; initialization is possible since $(p = 0) \Rightarrow P_0$ and we are done when $P_0 \land p = N - 1$. 
Relation \( P_t \) is, in addition, about a rooted tree \( t \) of which the elements \( m(c: p \leq c < N) \) are the vertices.

\( P_t: \) in tree \( t \), no element exceeds any of its successors in \( t \), and pre-order traversal of \( t \) visits the elements of \( m(c: p \leq c < N) \) in the order of increasing index.

(In the pre-order traversal the root of any subtree precedes all other vertices of that subtree, and the successors of a vertex are ordered.)

From \( P_t \) we conclude, firstly that the root is the minimum of \( m(c: p \leq c < N) \) and, secondly, that \( m(p) \) is the root. Hence \( m(p) = \min m(c: p \leq c < N) \). If, in addition, \( m(p) \) has (at most) 1 successor, \( p := p + 1 \) maintains \( P_0 \land P_t \).

The structure of the sorting algorithm is

"choose \( t \) and rearrange \( m(c: 0 \leq c < N) \) such as to establish \( P_0 \land P_t \) for \( p = 0 \);"
do \( p \neq N - 1 \rightarrow 
  "rearrange \( m(c: p \leq c < N) \) and modify \( t \) under invariance of \( P_0 \land P_t \) such that \( m(p) \) has a single successor";
  \( p := p + 1 \)

od
Initially we choose $t$ reasonably well-balanced. (It need not be a binary tree; I am now not interested in its optimal shape.) Like in Heapsort, the first phase can be done by:

```
p := N-1;
do p \neq 0 \rightarrow p := p-1; \text{sift}(p) \text{ od } \{P0 \land P1 \land p=0\}
```

Let the root have more than one successor. Let $m(v)$ be its last successor and $m(w)$ its last successor but one. We can reduce the number of successors of the root by (changing $t$ and) making $m(v)$ the last successor of $m(w)$; this rearrangement does not violate the pre-order traversal of $t$ and a subsequent call of $\text{sift}(w)$ restores $P1$.

It is the administration of $t$ that makes this algorithm clerically awkward. The possible values (= shapes) of $t$ are fully determined by its initial value $t0$. In $t$ each node has at most one "singular successor", i.e. a successor it did not have in $t0$. A singular successor is always a last successor, and the potential singular successor of any node is uniquely determined by $t0$.

In $t0$, the potential singular successor of a vertex $v$ is given recursively as follows:

- if $v$ has no predecessor (i.e. is the root)
  - its potential singular successor is void,
if \( v \) is the last successor of its predecessor, its potential singular successor is the potential singular successor of its predecessor.

if \( v \) is not the last successor of its predecessor, its potential successor is the next successor of its predecessor.

If we choose, with \( K \) the minimal value such that \( 2^K > N \), for \( 2^K - 1 \) elements, determining the regular successors of a vertex is no problem provided its level is known. Since the total number of singular successors is bounded by \( K \), there is a fair chance that, at the expense of a modest amount of additional storage, the scheme can be implemented quite efficiently.

Plataanstraat 5 19 June 1981
5671 AL NUENEN prof. dr. Edsger W. Dijkstra
The Netherlands Burroughs Research Fellow

P.S. For the above class of sorting algorithms I have invented the name "smoothsort", for what is a sorting algorithm without a name?

EWD.