A minor improvement of Heapsort.

Heapsort is an efficient algorithm for sorting in situ the elements of a linear array $M(i)$, $0 \leq i < N$. When sorting the elements in ascending order, the algorithm maintains $H(p)$, defined by

$$H(p) : (\forall i, j : p \leq i < j < q \land 2 \cdot i \leq j \leq 2 \cdot (i+1) : M(i) \geq M(j))$$

which enjoys the useful property

$$H(0) \Rightarrow (\forall j : 0 \leq j < q : M(0) \geq M(j)) \quad (0)$$

The algorithm has the following form:

$$p, q := N \, \text{div} \, 2, N ; \{ H(p) \}$$

$$\text{do } p > 0 \rightarrow p := p - 1; \{ H(p+1) \} \, \text{sift} \, \{ H(p) \} \, \text{ od };$$

$$\text{do } q > 1 \rightarrow \{ H(0) \} \, q := q - 1; \, M: \text{swap}(0,q);$$

$$\{ H(p+1) \} \, \text{sift} \, \{ H(p) \} \, \text{ od }$$

Since $p=0$ is a further invariant of the second repetition, property $(0)$ ensures that the sorted sequence is built up "from right to left."

The routine sift establishes - by $w := p$ - and maintains $SH$, defined by

$$SH : (\forall i, j : p \leq i < j < q \land 2 \cdot i \leq j \leq 2 \cdot (i+1) : M(i) \geq M(j) \lor i=w),$$
which enjoys the useful property

$$SH \land 2 \cdot w + 1 \geq q \Rightarrow H(p)$$

Routine sift can repeatedly perform under invariance of $SH$ either $w := 2 \cdot w + 1$ or $w := 2 \cdot w + 2$; sift compares each time $M(w)$ with the maximum of $M(2 \cdot w + 1)$ and $M(2 \cdot w + 2)$. If $M(w)$ is large enough, $H(p)$ holds and sift terminates; otherwise $w$ can be "doubled" at the price of 2 comparisons and 1 swap in array $M$. For further details we refer the reader to [0].

We can do better (in terms of numbers of comparisons and swaps needed) by replacing $H(p)$ by $H_3(p)$ — and, similarly, $SH$ by $SH_3$ —

$$H_3(p): (A \land j: p \leq i < j \leq q \land 3 \cdot i < j \leq 3 \cdot (i + 1): M(i) \geq M(j)).$$

Firstly, we can then start with a smaller $p$, viz. $(N + 1) \div 3$; secondly, sift can then "triple" $w$ at the cost of 3 comparisons and 1 swap in array $M$. Thus 6 comparisons and 2 swaps multiply $w$ by 9, whereas originally 6 comparisons and 3 swaps were needed for a factor of 8. (With the analogous $H_5(p)$, the gain in comparisons needed is lost again: $2^3 < 3^2$, but $2^4 < 4^2$. Since $2^5 > 5^2$, $H_5(p)$ is expected to lead to more comparisons in sift.)
A worst-case sift is one that terminates with \(2 \cdot w + 1 \geq q\) or \(3 \cdot w + 1 \geq q\) respectively. A sort in which all sifts are worst-case sifts would clearly be a worst-case sort. Since such sorts can occur—see below—and our modification improves worst-case sifts, the worst-case performance of Heapsort has, indeed, been improved.

The crucial observation is that, when upon completion of a call of sift the final value of \(w\) is not destroyed, the effect of that call can be undone: sift itself has a unique inverse sift\(^{-1}\) (ending with \(w=p\)). Starting with an increasing array \(M\), we can play Heapsort backwards, supplying each time sift with a "proper" initial value for \(w\) such that \(2 \cdot w + 1 \geq q\) or \(3 \cdot w + 1 \geq q\) respectively—for a detailed discussion of the notion "proper", see below—. Our backwards game ends with an \(M\) that would lead to a sort with worst-case sifts only.

Now a detailing of the notion "proper". Our backwards game starts increasing \(q\) repeatedly by

\[
\{ H(p) \} \text{ sift}^{-1} \{ H(p+1) \}; \\
M: \text{swap}(q,0); q := q + 1 \{ H(0) \}
\]  

(2)

Independently of our choice of \(w\), \(H(0)\) holds after the swap, because the new \(M(0)\) satisfies \((\forall j: 0 \leq j < q: M(0) > M(j))\). But does \(H(0)\) hold after
\( q := q + 1 \) ? It does if \( M(q-1) \) is then small enough. We can achieve this, for instance, by initializing for \( \text{shift}^{-1} \) \( w = q - 1 \); program section (2) then maintains \((\forall c : p \leq c < q : M(c) \geq M(q-1))\). Our backwards game continues increasing \( p \) repeatedly by

\[ \{H(p)\} \text{ shift}^{-1} \{H(p+1)\} ; p := p + 1 \]

Since \( p = 0 \) is now not an invariant, we must take precautions to ensure that \( \text{shift}^{-1} \) can end with \( w = p \); here "proper" means that the initial value of \( w \) is such that \( \text{do} w \neq p \rightarrow w := (w-1) \text{div} 2 \text{ od} \)

(or \( \text{do} w \neq p \rightarrow w := (w-1) \text{div} 3 \text{ od} \) terminates.

Compared to the above worst-case analysis, the analysis of the average case seems too difficult and insufficiently rewarding.

Acknowledgements. I am indebted to Ross A. Honsberger who sent me a series of combinatorial problems, one of which used \( 2^3 < 3^2 \). (The problem was how to partition a given positive integer into positive integer parts such that the product of the parts is maximal. The solution is to take as many parts \( = 3 \) as is possible without introducing a remaining part \( = 1 \). The preponderance of 3's is not amazing; 3 is the nearest integer approximation of \( e \).)
I am also indebted to R.W. Bulterman, who spotted an error in my original form of H3(p), which failed to satisfy the analogue of (0); in the literature, Heapsort traditionally sorts M(c:15c'sN) and unthinkingly I had adopted that unfortunate convention, which was responsible for my error. Finally I am indebted to Eric C.R. Hehner and the members of the Tuesday Afternoon Club, who helped me with the worst-case analysis, in which we clearly benefitted from our earlier work on program inversion (see [1]).


Plataanstraat 5 14 May 1981
5671 AL Nijen
The Netherlands prof. dr. Edsger W. Dijkstra
Burroughs Research Fellow.