INFORMATION AND COMPUTATION 97, 86–96 (1992)

# The Worst Case Complexity of McDiarmid and Reed's Variant of BOTTOM-UP HEAPSORT Is Less than $n \log n + 1.1n$

INGO WEGENER*

*Fachbereich Informatik, LS II, Universität Dortmund, Postfach 500 500, 4600 Dortmund 50, Germany*

BOTTOM-UP HEAPSORT is a variant of HEAPSORT which beats on average even the clever variants of QUICKSORT, if $n$ is not very small. Up to now, the worst case complexity of BOTTOM-UP HEAPSORT has been able to be estimated only by $1.5n \log n$. McDiarmid and Reed (1989) have presented a variant of BOTTOM-UP HEAPSORT which needs extra storage for $n$ bits. The worst case number of comparisons of this (almost internal) sorting algorithm is estimated by $n \log n + 1.1n$. It is discussed how many comparisons can be saved on average.


## 1. INTRODUCTION

Sorting is one of the most fundamental problems in computer science. In this paper only general and sequential sorting algorithms are studied.

All results should be compared with the simple lower bound

$$\log(n!) = n \log n - n \log e + \Theta(\log n) \approx n \log n - 1.442695n$$

for the worst and average case number of comparisons of general sorting algorithms. With respect to this lower bound sorting by merging and sorting by insertion and binary search are quite efficient. But MERGESORT works efficiently only on an array of length $2n$ and INSERTIONSORT uses $\Theta(n^2)$ operations for the data transport.

HEAPSORT (Floyd (1964), Williams (1964)) needs $2n \log n$ comparisons. Because of the factor 2 HEAPSORT is in almost all cases less efficient than QUICKSORT. All versions of QUICKSORT are inefficient in the worst case but efficient in the average case. Let $H(n) = 1 + 1/2 + \cdots + 1/n$ be the $n$th harmonic number, $Q(n)$ the average number of comparisons of QUICKSORT, and $CQ(n)$ the average number of

*Supported in part by DFG Grants We 1066-2/1 and Me 872-1/3.



0890-5401/92 $3.00

comparisons of the best-of-three variant of QUICKSORT called CLEVER QUICKSORT. Then (see, e.g., Wegener (1989))

$$Q(n) = 2(n+1)H(n) - 4n \approx 1.386294n \log n - 2.845569n$$

and for $n \geqslant 6$

$$CQ(n) = \frac{12}{7}(n+1)H(n-1) - \frac{477}{147}n + \frac{223}{147} + \frac{252}{147n}$$

$$\approx 1.188252n \log n - 2.255385n.$$

Because of these results HEAPSORT has been considered for a long time only for theoretical reasons. Carlsson (1987a) presented a new variant of HEAPSORT whose average and worst case complexity is $n \log n + \Theta(n \log \log n)$. This algorithm does not beat CLEVER QUICKSORT on average for $n \leqslant 10^{16}$. Another variant of HEAPSORT is called BOTTOM-UP HEAPSORT (Carlsson (1987b), Wegener (1989)). Its average case complexity cannot be computed because the heaps constructed in the selection phase are not random. Using realistic models one can argue that the average case number of comparisons equals $n \log n + f(n)n$ where $f(n) \in [0.355, 0.39]$ for $n \geqslant 3000$ depends on $n$ (see Wegener (1989)). BOTTOM-UP HEAPSORT is a general and internal sorting algorithm which beats on average QUICKSORT for $n \geqslant 400$ and CLEVER QUICKSORT for $n \geqslant 16{,}000$. The worst case complexity can be bounded by $1.5n \log n - 0.4n$ (Wegener, 1989), but it may be conjectured that the worst case complexity is $n \log n + o(n \log n)$.

In this situation a variant of BOTTOM-UP HEAPSORT (McDiarmid and Reed (1989)) which we call MDR-HEAPSORT is interesting. MDR-HEAPSORT is not an internal sorting algorithm in the strong sense, since extra storage for $n$ bits is necessary.

In Section 2 we present MDR-HEAPSORT and discuss some details of its implementation. McDiarmid and Reed (1989) have analysed the average case complexity of the heap creation phase of their algorithm and have left the average and worst case analysis of the whole algorithm as an open problem.

In Section 3 we prove that the worst case number of comparisons of MDR-HEAPSORT is remarkably small: it can be bounded by $n \log n + 1.1n$.

No rigorous analysis of the average case complexity of any HEAPSORT variant is known, since the heaps constructed during the selection phase are not random. In Section 4 we use realistic models to estimate the difference between the average and worst case complexity of MDR-HEAP-

SORT. We finish the paper with a comparison of MDR-HEAPSORT and QUICKSORT.

## 2. MDR-HEAPSORT

We assume that the reader is familiar with HEAPSORT. The same basic idea is used here.

mdr-heapsort $(a, n)$ [$a$ is an array of length $n$]
1. [heap creation phase]
for $i = \lfloor n/2 \rfloor, \ldots, 1$: mdr-reheap$(n, i)$.
2. [selection phase]
for $m = n, \ldots, 2$: interchange $a(1)$ and $a(m)$, mdr-reheap $(m-1, 1)$.

Procedure mdr-reheap does the same as the well-known procedure reheap but in another way.

Procedure reheap starts at the root. It stops if the object at the vertex just considered is not larger than the objects at its two sons. Otherwise it interchanges the object at the vertex just considered with the smaller object of the two son objects and considers the corresponding son. (In this paper heaps are defined by $a(j) \leqslant a(2j)$ and $a(j) \leqslant a(2j+1)$.) Procedure reheap needs 2 comparisons at each vertex in order to compute the minimum of the three objects at the vertesx and its 2 sons. In many cases procedure reheap needs almost $2d$ comparisons, if $d$ is the depth of the heap. We like to get by with approximately $d$ comparisons. Therefore we use only 1 comparison at each vertex. With 1 comparison we can decide which son of the vertex just considered contains the smaller object and we go to this son. By this procedure it is not possible to stop at the correct position, since we do not consider the object at the root. Hence, we walk down a path in the heap until we reach a leaf. This path will be called *special path*, and the last object on this path will be called *special leaf*. This part of the algorithm is done by the procedure leaf-search, which returns the special leaf. The procedure reheap of HEAPSORT places the root object at some position $p$ of the special path and all objects on the special path from the root to position $p$ are shifted into their father vertices. BOTTOM-UP HEAPSORT and MDR-HEAPSORT look for the same position $p$. But these algorithms start their search from the special leaf and search bottom-up by the procedure bottom-up search. Finally, the procedure interchange performs the data transport. Hence, all HEAPSORT variants construct the same heaps.

Let $d$ be the length of the special path and let $j$ be the length of the path from the root of the special path to the vertex at position $p$. The procedure reheap of HEAPSORT needs $2 \min\{d, j+1\}$ comparisons while the procedure bottom-up reheap needs $d + \min\{d, d-j+1\}$ comparisons (see below). For large $j$ we save almost half of the comparisons.

With MDR-HEAPSORT we save even more comparisons by using old information. MDR-HEAPSORT works with an extra array called info. The possible values of info($j$) are

— unknown, abbreviated by $u$,
— left, abbreviated by $l$,
— right, abbreviated by $r$,

and can be coded by 2 bits. The interpretation is the following. If info($j$) = $l$ (or $r$), then the left son contains a smaller object than the right one (or vice versa). If info($j$) = $u$, nothing is known about the smaller son. We need info($j$) only for $j = 1, ..., \lfloor (n-1)/2 \rfloor$, since the other nodes do not have two sons. Hence, $2\lfloor (n-1)/2 \rfloor < n$ extra bits are sufficient. The parameters are initialized as $u$.

Now we are prepared to describe the procedure mdr-reheap.

mdr-reheap ($m$, $i$) 1. leaf-search ($m$, $i$).
2. bottom-up search ($i$, $j$).
3. interchange ($i$, $j$).

leaf-search($m$, $i$)
1. $j := i$.
2. while $2j < m$ do begin
if info($j$) = $l$ then $j := 2j$,
else if info($j$) = $r$ then $j := 2j + 1$,
else if $a(2j) < a(2j+1)$ then, info($j$) := $l$, $j := 2j$
else info($j$) := $r$, $j := 2j + 1$, end.
3. if $2j = m$ then $j := m$.
return $j$

bottom-up search($i$, $j$) [$j$ is the output of leaf-search]
1. while ($i < j$ and $a(i) < a(j)$) do $j := \lfloor j/2 \rfloor$.
return $j$

The procedure bottom-up search walks the special path bottom-up until the new position for the root object is found. The data transport is done by a cyclic shift on the special path from the root $i$ to the position $j$ computed by bottom-up search. Comments on an efficient implementation of interchange can be found in Wegener (1989). By bin($j$) we denote the length of the binary representation of $j$.

interchange($i$, $j$) [$j$ is the output of bottom-up-search]
1. $l := \text{bin}(j) - \text{bin}(i)$, $x := a(i)$.
2. for $k = l-1, ..., 0$: $a(\lfloor j/2^{k+1} \rfloor) := a(\lfloor j/2^k \rfloor)$, info($\lfloor j/2^{k+1} \rfloor$) := $u$.
3. $a(j) := x$.

The correctness of the new algorithm is obvious, since (by our considerations above) reheap (for HEAPSORT), bottom-up reheap (for BOTTOM-UP HEAPSORT) and mdr-reheap (for MDR-HEAPSORT) all construct the same heap. MDR-HEAPSORT is a general sorting algorithm which is easy to implement. The number of operations which are not comparisons between array objects is $O(n \log n)$. Because of the $2\lfloor (n-1)/2 \rfloor$ extra bits MDR-HEAPSORT is not an internal sorting algorithm in the strong sense.

We remark that MDR-HEAPSORT often can be implemented as an internal sorting algorithm. If, e.g., we have addresses of length 32 and $n \leqslant 2^{30}$, two bits of the address vector may be used for info($j$). Such versions of MDR-HEAPSORT can also be implemented easily.

## 3. The Worst Case Analysis

A single call of mdr-reheap may cause $2d$ comparisons, if $d$ is the actual depth of the heap. But this cannot happen quite often. Hence, we do not sum up the worst case numbers for the single calls of mdr-reheap. In place of that we estimate the amortized number of comparisons.

We investigate the special path with the nodes $b(0), \ldots, b(d)$. We use the notion of *pebbles* in order to illustrate our considerations. A node $j$ is pebbled, if info($j$) $\neq u$ and $j$ has two sons in the actual heap. Pebbles are *created* during Step 2 of the procedure leaf-search. Pebbles are *deleted* during Step 2 of procedure interchange. Pebbles *vanish*, if a node looses its right son, since we consider a smaller array during the selection phase. During a single call of mdr-reheap the nodes on the special path which have two sons are pebbled and, afterwards, the pebbles from some initial segment of the path are deleted. We conclude that for each actual special path the situation, before we run through this special path, is the following. For some $k$, $b(0), \ldots, b(k)$ are unpebbled and $b(k+1), \ldots, b(d)$ are pebbled, if these nodes have two sons; i.e., $b(d)$ and perhaps $b(d-1)$ are also unpebbled. Let $j$ be chosen such that procedure interchange performs a cyclic shift on $b(0), \ldots, b(j)$. Then pebbles on $b(0), \ldots, b(j-1)$ are deleted during procedure interchange.

Now we analyze the call of mdr-reheap with this actual special path. The number of comparisons during the procedure leaf-search is $k+1$, and the number of comparisons during the procedure bottom-up-search is $\min\{d, d-j+1\}$, since the root object is not compared with itself. We estimate the number of comparisons by

$$k+1+d-j+1 = d+1-(d-1-k)+(d-j).$$

We interpret this number in the following way.

— $d$ is the length of the special path; i.e., $d$ is either the depth of the heap or one less than the depth.

— 1 stands for the comparison between $b(0)$ and $b(d)$.

— $d-1-k$ is the number of "old" pebbles on this path before the call of procedure leaf-search (this number is $d-2-k$, if $b(d-1)$ has only one son).

— $d-j$ is the number of "new" pebbles on this path after the call of the procedure interchange (this number is $d-1-j$, if $b(d-1)$ has only one son).

We sum up these terms in order to estimate the worst case number of comparisons.

The algorithm starts without any pebble and stops without any pebble, since the last heap consists of two nodes and, therefore, it does not contain any node with two sons. Let OP be the sum of the numbers of "old" pebbles and NP be the sum of the numbers of "new" pebbles. Then $\mathrm{NP}-\mathrm{OP}$ is the number of vanishing pebbles. This number is bounded by $\lfloor (n-1)/2 \rfloor$, since pebbles lie only on the nodes $1, \ldots, \lfloor (n-1)/2 \rfloor$ and since for each node only one pebble may vanish.

Now we sum the terms "1"; i.e., we count the number of calls of procedure bottom-up search. This number equals $n+\lfloor n/2 \rfloor - 2$.

At last we sum the depths of the considered heaps.

LEMMA 1. *The sum of the depths of the heaps considered during the heap creation phase is in the interval* $[n-\lfloor \log n \rfloor - 1, n-1]$.

*Proof.* We have to consider $\lfloor n/2 \rfloor$ heaps with the roots $1, \ldots, \lfloor n/2 \rfloor$. Only the nodes $1, \ldots, \lfloor n/4 \rfloor$ are roots of heaps of depth 2 or larger. In general, only the nodes $1, \ldots, \lfloor n2^{-h} \rfloor$ are roots of heaps of depth $h$ or larger. Hence, the sum of the depths equals the sum of all $\lfloor n2^{-h} \rfloor$, $1 \leqslant h \leqslant \lfloor \log n \rfloor$. This sum can easily be estimated in that way as stated in the claim of the lemma. ∎

LEMMA 2. *The sum of the depths of the heaps considered during the selection phase equals* $n\lfloor \log n \rfloor - 2^{\lfloor \log n \rfloor + 1} + \lfloor \log n \rfloor + 2$.

*Proof.* We have to consider heaps with $n, \ldots, 2$ nodes. The depth of a heap on $i$ nodes equals $\lfloor \log i \rfloor$. The sum of all $\lfloor \log i \rfloor$, $2 \leqslant i \leqslant n$, equals

$$\sum_{1 \leqslant i \leqslant \lfloor \log n \rfloor - 1} i\,2^i + \lfloor \log n \rfloor (n - 2^{\lfloor \log n \rfloor} + 1)$$

$$= (\lfloor \log n \rfloor - 2)\,2^{\lfloor \log n \rfloor} + 2 + n\lfloor \log n \rfloor - 2^{\lfloor \log n \rfloor} + \lfloor \log n \rfloor$$

$$= n\lfloor \log n \rfloor - 2^{\lfloor \log n \rfloor + 1} + \lfloor \log n \rfloor + 2. \quad ∎$$

We like to express $n\lfloor \log n \rfloor - 2^{\lfloor \log n \rfloor + 1}$ as $n \log n - A(n)n$. Hence,

$$A(n) := \log n - \lfloor \log n \rfloor + 2^{\lfloor \log n \rfloor + 1}/n.$$

We consider the interval $I = [2^k, 2^{k+1})$ such that $\lfloor \log x \rfloor = k$ for $x \in I$. It is easy to see that $A(x)$ takes its maximum for $x = 2^k$, $A(2^k) = 2$, and $A(x)$ takes its minimum for $x = (2 \ln 2)2^k$, where

$$A((2 \ln 2)2^k) = \log(2 \ln 2) + (\ln 2)^{-1} \geqslant 1.913928.$$

Altogether we have finished our analysis of the worst case complexity of MDR-HEAPSORT.

THEOREM 1. *Let* $A(n) := \log n - \lfloor \log n \rfloor + 2^{\lfloor \log n \rfloor + 1}/n$. *Then*

$$A(n) \in [\log(2 \ln 2) + (\ln 2)^{-1}, 2] \subseteq [1.913928, 2].$$

*The worst case number of comparisons of* MDR-*HEAPSORT is bounded by*

$$n \log n + (3 - A(n))n + \lfloor \log n \rfloor - 1 \leqslant (n+1) \log n + 1.086072n.$$

*For* $n = 2^k$, *the number of comparisons can be bounded by* $(n+1) \log n + n$.

*This upper bound is only* 2.528767$n$ *larger than the lower bound for general sorting algorithms.*

## 4. THE AVERAGE CASE ANALYSIS

Theorem 1, is, of course, also an upper bound on the average case complexity of MDR-HEAPSORT. The proof of Theorem 1 is the first *proof* of an $n \log n + O(n)$ upper bound on the complexity of any HEAPSORT variant.

Our worst case analysis in Section 3 implies that MDR-HEAPSORT can save comparisons only because of one of the following reasons:

— The number of comparisons during the procedure bottom-up search equals $\min\{d, d-j+1\}$ and is estimated by $d-j+1$. We save one comparison, if $j = 0$.

— The length of the special path is estimated by the depth of the heap. Such special paths are called long paths. Special paths also can be short paths whose length is 1 smaller that the depth of the heap.

— The number of vanishing pebbles has been estimated by $\lfloor (n-1)/2 \rfloor$. This number can be equal to 0.

We are able to analyse the first effect.

LEMMA 3. *Let*

$$\alpha := \sum_{2 \leqslant i < \infty} \frac{1}{2^i(2^i-1)}.$$

*Then* $\alpha \in [0.1066948, 0.1066950]$. MDR-*HEAPSORT saves on average* $\alpha n + \Theta(\log n)$ *comparisons, since the procedure bottom-up search avoids a comparison of the root object with itself.*

*Proof.* No comparison is saved during the selection phase. The root object has been a leaf object. Because of the heap property it is not smaller than the object at the son of the root on the special path.

During the heap creation phase one comparison is saved, if the root object is the smallest object of the subtree. For heaps with $k$ objects the probability of this event equals $1/k$, since the root object is at this time a random object. If $n = 2^k - 1$, we consider during the heap creation phase exactly $2^h$ heaps with $2^{k-h} - 1$ objects, $0 \leqslant h \leqslant k-2$. The expected number of saved comparisons equals

$$\sum_{0 \leqslant h \leqslant k-2} 2^h \frac{1}{2^{k-h}-1} = 2^k \sum_{0 \leqslant h \leqslant k-2} \frac{1}{2^{k-h}(2^{k-h}-1)}$$

$$= (n+1) \sum_{2 \leqslant i \leqslant k} \frac{1}{2^i(2^i-1)} \leqslant \alpha(n+1).$$

For general $n$, the probability can be computed in a similar way. The difference from $\alpha n$ is $\Theta(\log n)$. ∎

During the heap creation phase most of the heaps are complete, i.e., they have $2^j - 1$ nodes for some $j$. Hence, only $\lfloor \log n \rfloor$ special paths can be short. During the selection phase almost all of the $n-2$ special paths may be short. Hence, we have proved the following theorem.

THEOREM 2. *The average case number of comparisons of* MDR-*HEAPSORT is at most*

$$n \log n + (3 - A(n) - \alpha)n + \Theta(\log n)$$

*and at least*

$$n \log n + (1.5 - A(n) - \alpha)n + \Theta(\log n).$$

We remark that $3 - A(n) - \alpha \leqslant 0.979378$ and $3 - A(n) - \alpha \leqslant 0.893306$ for $n = 2^k$. The difference between upper and lower bound for the average case complexity of MDR-HEAPSORT is only $1.5n + \log n$.

We cannot compute the average case complexity exactly, since MDR-HEAPSORT does not construct random heaps during the selection phase. In the rest of the paper we attempt to get an idea about the difference between the worst case and the average case complexity.

Carlsson (1987*b*) has shown that the expected number of short paths is approximately $0.3n$, if one investigates random heaps. Experiments show that random heaps do not quite model the reality well. Wegener (1989) has presented another model which led to the following conjecture. This conjecture is well established by experiments.

*Conjecture* 1. Let $B(n)n$ be the expected number of short paths. Then

$$B(n) \in [0.5, 0.55] \qquad \text{for} \quad n \geqslant 400.$$

The parameter $B(n)$ is approximately 0.5, if $n \approx 2^k$, and approximately 0.55, if $n \approx \sqrt{2} \times 2^k$.

Finally, we have to discuss the expected number of vanishing pebbles. By definition, no pebble vanishes during the heap creation phase. Hence, it is interesting to know the expected number of pebbles at the end of the heap creation phase.

LEMMA 4. *Let $C(n)n$ be the expected number of pebbles in the heap at the end of the heap creation phase. Then $C(n) \approx 0.127983$; i.e., for large $n$, $C(n) \in [0.127982, 0.127984]$.*

*Proof.* We count the number of comparisons during the heap creation phase in two different ways.

McDiarmid and Reed (1989) have established a recurrence relation for the expected number of comparisons. By an approximation of this recurrence relation they have shown that this number equals $Z(n)n$ for some function $Z$, where $Z(n) \to 1.521288...$ as $n \to \infty$.

In Section 3 we have used another method of counting the comparisons. By Lemma 1, the sum of the lengths of the special paths equals $n + \Theta(\log n)$. The sum of the terms "1" equals $\lfloor n/2 \rfloor$. The difference between the sum of the numbers of "new" pebbles and the sum of the numbers of "old" pebbles equals $C(n)n$, the number of pebbles still in the heap. By Lemma 3, we have to subtract $\alpha n + \Theta(\log n)$ comparisons, since we avoid comparisons of the root object with itself.

Since the result of McDiarmid and Reed is correct only up to 6 digits, also the following equation is also correct only up to 6 digits for the constants of the linear terms and up to logarithmic terms. Since $\alpha = 0.106695...$,

$$1.521288n = 1.5n - 0.106695n + C(n)n \qquad \text{and} \qquad C(n) = 0.127983. \quad \blacksquare$$

We suppose that only a small number of pebbles are created during the selection phase.

*Conjecture* 2. Let $D_i(n)n$ be the expected number of calls of bottom-up search during the selection phase, which cause exactly $i$ comparisons. Then, for $n \geqslant 2000$,

$$D_1(n) \in [0.846, 0.850] \qquad \text{and} \qquad \lim_{n \to \infty} (D_1(n) + D_2(n)) = 1.$$

The second part of this conjecture has been proved for random heaps by Doberkat (1982). Wegener (1989) presented a realistic model leading to this conjecture, which is also well established by experiments.

In our worst case analysis the number of vanishing pebbles equals $0.5n$. Lemma 4 and Conjecture 2 imply that, in the average case, only $0.28n$ pebbles have the chance to vanish. How large is the probability that a pebble vanishes?

We consider the situation $n = 2^k - 1$ and a pebble on a random node whose sons are leaves. There are $2^{k-2}$ such nodes. The pebble at position $i$ ($1 \leqslant i \leqslant 2^{k-2}$, the nodes are numbered right to left) vanishes iff the first $2i-1$ special paths do not use this node. We assume that the probability that a special path does not use this node equals $q := 1 - 2^{-(k-2)}$ and that the special paths are independent. Then the probability that the pebble vanishes equals

$$2^{-(k-2)} \sum_{1 \leqslant i \leqslant 2^{k-2}} q^{2i-1} = (1-q)q \sum_{0 \leqslant i \leqslant 2^{k-2}-1} (q^2)^i$$
$$= (1-q)\, q(q^{2^{k-1}} - 1)/(q^2 - 1) = q(1 - q^{2^{k-1}})/(1+q).$$

For large $k$, $q \approx 1$ and $q^{2^{k-2}} \approx e^{-1}$. Hence, our probability can be estimated by $(1 - e^{-2})/2 \approx 0.43$.

Obviously, this is only a rough model. There are pebbles placed during the heap creation phase which are placed on nodes whose sons are inner nodes. By a similar argument to that above, it follows that these pebbles do vanish with an arbitrarily small probability (for large $n$). During the selection phase more pebbles are created if the special path is a long one. For these pebbles the probability of vanishing is larger than in our model. Since we are discussing only $0.28n$ pebbles and since we are not able to prove exact results on the average case number of comparisons, we stop the discussion here.

Let $n \log n + E(n)$ be the average case complexity of MDR-HEAPSORT, then, by Theorem 2, for large $n$,

$$-0.520622 \leqslant E(n) \leqslant 0.979378.$$

We may conjecture by our considerations that $E(n) \in [-0.05, 0.10]$ and that $E(n)$ depends on $n$ in the following way: $E(n)$ is small if $n \approx 2^k$, and large if $n \approx 1.4 \times 2^k$.

Hence, we can suppose that MDR-HEAPSORT uses in the average case approximately $0.35n$ fewer comparisons that BOTTOM-UP HEAPSORT. For this advantage we have to pay with storage for $n$ extra bits and $\Theta(n \log n)$ bit tests. The most important advantage of MDR-HEAPSORT is the provably very small number of comparisons in the worst case. For $n \geqslant 1000$, the *worst* case number of comparisons of MDR-HEAPSORT is smaller than the *average* case number of comparisons of QUICKSORT. The corresponding critical value for CLEVER QUICKSORT is approximately 200.000. Finally, we conjecture that, in the average case, MDR-HEAPSORT beats QUICKSORT if $n \geqslant 200$, and CLEVER QUICKSORT if $n \geqslant 4500$.

RECEIVED April 23, 1990; FINAL MANUSCRIPT RECEIVED September 11, 1990

## REFERENCES

CARLSSON, S. (1987a). A variant of HEAPSORT with almost optimal number of comparisons. *Inform. Process. Lett.* **24**, 247–250.

CARLSSON, S. (1987b). Average-case results on HEAPSORT, *BIT* **27**, 2–17.

DOBERKAT, E. E. (1982). Deleting the root of a heap. *Acta Informat.* **17**, 245–265.

DOBERKAT, E. E. (1984). An average case analysis of Floyd's algorithm to construct heaps. *Inform. Control* **61**, 114–131.

FLOYD, R. W. (1964). Algorithm 245, treesort 3. *Comm. ACM* **7**, 701.

MCDIARMID, C. J. H., AND REED, B. A. (1989). Building heaps fast. *J. Algorithms* **10**, 352–365.

WEGENER, I. (1989). BOTTOM-UP HEAPSORT, a new variant of HEAPSORT beating on average QUICKSORT (if $n$ is not very small). Submitted for publication.

WILLIAMS, J. W. J. (1964). Algorithm 232. *Comm. ACM* **7**, 347–348.