JOURNAL OF ALGORITHMS **20,** 205–217 (1996)
ARTICLE NO. 0011

On the Best Case of Heapsort*

B. Bollobás†

Department of Pure Mathematics, University of Cambridge, Cambridge CB2 1TN, United Kingdom

T. I. Fenner

Department of Computer Science, Birkbeck College, University of London, London WC1E 7HX, United Kingdom

and

A. M. Frieze‡

Department of Mathematics, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213

Received August 14, 1991; revised July 24, 1994

Although discovered some 30 years ago, the Heapsort algorithm is still not completely understood. Here we investigate the *best case* of Heapsort. Contrary to claims made by some authors that its time complexity is $O(n)$, i.e., linear in the number of items, we prove that it is actually $O(n \log n)$ and is, in fact, approximately half that of the worst case. Our proof contains a construction for an asymptotically best-case heap. In addition, the proof and construction provide the *worst-case* time complexity and an asymptotically worst-case example for *Bottom-up* versions of Heapsort.

1. INTRODUCTION

In spite of its age, there are still some aspects of Heapsort (discovered by Williams [11]) which have not been completely sorted out. Its worst-case performance is reasonably well understood, but the average-case performance remains a mystery (see Knuth [7, pp. 156–157] for some empirical

*A preliminary version of this paper was first presented at the Cambridge Combinatorial Conference in honour of the 75th birthday of Professor Paul Erdős, March 1988.

†Supported by NSF Grant RG0088/89.
‡Supported by NSF Grant CCR-8900112.

0196-6774/96 $18.00

data on this subject). In this paper, we examine another aspect of Heapsort which has no obvious answer, i.e., the best case. We prove an asymptotically tight bound on the minimum number of operations taken by this algorithm. Some authors, e.g., Lorin [8], have claimed erroneously that it is linear, and Wirth [12] makes the comment that Heapsort seems to ''like'' sequences which are in inverse sorted order. As we will see, the best case of the algorithm is rather more complicated than this. Note that we are restricting our attention to the case in which all of the elements are distinct—otherwise, it is easy to see that the best case is when all the elements are identical, and then Heapsort runs in linear time.

(We mention in passing that heap building has attracted some attention lately, e.g., Bollobás and Simon [1], Frieze [3], Gonnet and Munroe [4], Hayward and McDiarmid [5], and McDiarmid and Reed [9].)

We now establish our notation. A (max) heap is an array $H[1..n]$ of integers satisfying $H[i] < H[\lfloor i/2 \rfloor]$ for $1 < i \leq n$. We will for simplicity assume that $n = 2^k - 1$ for some positive integer k, although our results can be generalized to arbitrary n. As usual, we imagine H as representing a (complete) binary tree T_n in which position i is the parent of positions $2i$ and $2i + 1$. In order to be precise, we will give a description of Heapsort.

ALGORITHM HEAPSORT.

```
begin
      BUILDHEAP;
      for i := n step -1 until 2 do
      begin
A:        interchange H[1] and H[i];
B:        HEAPIFY (i − 1)
      end
end
```

PROCEDURE HEAPIFY(w).

```
begin
      i := 1;
      while i ≤ w/2 do
      begin
            let H[j] = max{H[ℓ]  : ℓ ≤ w and ℓ ∈ {2i, 2i + 1}};
            if H[i] < H[j] then
              begin
C:              interchange H[i] and H[j];
                i := j
              end
            else i := w
      end
end
```

Since BUILDHEAP can be implemented to run in $O(n)$ time [7, p. 145], we will not need to dwell on this aspect of the algorithm. We will measure the execution time on a particular instance by the number of executions of Statement C. As stated, this seems to be about half of the number of comparisons needed, because it is necessary to make two comparisons prior to each interchange at Statement C. There are, however, versions of HEAPIFY which attempt to make the number of comparisons roughly equal to the number of executions of Statement C, assuming that the inserted element goes down to near the level of the leaves; see Knuth [7, p. 158, ex. 18], also Carlsson [2], and McDiarmid and Reed [9]. It appears that the inserted element usually does this in the "average case." Wegener [10] discusses one such version which he calls Bottom-up-Heapsort. Our example in Section 3 provides an asymptotically *worst-case* example for this and similar versions of the algorithm.

The basic idea is to assume that the inserted element will become a leaf and identify where it would be inserted, moving the larger child up at each level as before. This takes one interchange and only one (instead of two) comparisons per level. The actual final position of the element will be somewhere on the path from this leaf to the root. Finding this position and inserting the element there is accomplished by linear search up this path from the leaf. This takes one comparison and one interchange per level from the leaf to the final position. The final position is the same for both versions; if this is at level d and k is as defined in the following theorem, then the numbers of comparisons and interchanges to insert the element are approximately $2d$ and d for the standard algorithm, whereas both are $2k - d$ for the modified version. Thus minimizing d is best-case and worst-case, respectively.

Let $\alpha(H)$ denote the number of executions of Statement C, starting with heap H, and let $\mu(n)$ denote the minimum of $\alpha(H)$ over all heaps of size n. Our main result is:

THEOREM 1. *If* $n = 2^k - 1$ *for some integer* k *then*

$$\mu(n) = \frac{1}{2} n \lg n + O(n \lg \lg n)$$

(lg *denotes logarithm to the base* 2).

It is well known (see Knuth [7, p. 149]) that the maximum of $\alpha(H)$ over all heaps of size n is $n \lg n + O(n)$.

2. A LOWER BOUND ON $\mu(n)$

The lower bound of our theorem, although easy to prove, does not seem to be well known. This was discovered by the authors, and essentially the same result was obtained independently by Wegener [10]. We give a proof here for completeness.

When we execute Statement A, the largest element of the heap is put into its final position. We will refer to this as the value in $H[1]$ being deleted from the heap and the heap decreasing in size by one.

Round i removes the $(k - i + 1)$th level from the heap. Thus, for example, the $(n + 1)/2$ largest elements are deleted in Round 1. We will show that

Round i requires at least $2^{k-i-1}(k - i - 3)$ *executions of Statement* C. (1)

Hence, for $k \geq 4$,

$$\begin{aligned} \mu(n) &\geq \sum_{j=1}^{k-4} j2^{j+2} \\ &= (k-5)2^{k-1} + 8 \\ &\geq \frac{1}{2} n \lg n - \frac{5}{2} n. \end{aligned} \tag{2}$$

Clearly, we need only prove (1) for $i = 1$. Assume now, w.l.o.g., that $H[1]$, $H[2], \ldots, H[n]$ is a permutation of $[n] = \{1, 2, \ldots, n\}$. We say that i is *small* if $i < (n + 1)/2$ and *large* otherwise.

Let now

$$\begin{aligned} L &= \left\{ t \;:\; t < \frac{n+1}{2}, H[t] \geq \frac{n+1}{2} \right\} \\ &= \{\text{positions of large elements in levels } 1, 2, \ldots, k-1\}; \end{aligned}$$

i.e., L is the set of positions of large elements which are not leaves.

We will also say that a node in the tree is large when it contains a large element. Now the elements which are initially placed in $H[t]$ for $t \in L$ are large, so they are deleted in Round 1. To accomplish this they must be brought to the top of the heap by interchanges at Statement C. Hence the number of exchanges in Round 1 is at least

$$\sum_{t \in L} \text{depth}(t),$$

where depth(t) = the number of arcs in the path from t to the root of H.

Observe next that the positions corresponding to L correspond to a subtree rooted at 1, since $i \in L$ implies that the parent of i is in L. Thus from Knuth [6, pp. 399–400], it is easy to show that

$$\sum_{t \in L} \text{depth}(t) \geq |L| \lg |L| - 2|L|.$$

Thus (1) and (2) follow once we have shown that

$$|L| \geq 2^{k-2}. \tag{3}$$

To do this let $\pi(1), \pi(2), \ldots, \pi(n)$ be the sequence of nodes visited in an in-order traversal of T_n (see Knuth [6, pp. 316–321]).

Note that if $\pi(j)$, $j \geq 2$, is a leaf of T_n then $\pi(j-1)$ is not. In particular, if $\pi(j)$ is a large leaf then $\pi(j-1) \in L$. Thus the number of large leaves cannot exceed $|L| + 1$, even if $\pi(1)$ is a large leaf. Since the total number of large elements is 2^{k-1}, inequality (3) now follows.

3. AN UPPER BOUND ON $\mu(n)$

We will now describe an example where the number of exchanges (at Statement C) in Round 1 $\approx \frac{1}{4} n \lg n$, and then (inductively) the number of exchanges overall $\approx (\frac{1}{4} + \frac{1}{8} + \cdots) n \lg n = \frac{1}{2} n \lg n$. Since the number of exchanges involving elements in positions in L is already $\approx \frac{1}{4} n \lg n$, we must find an example where most of the large elements of the lowest level do not "fall very far" after they are placed at the top of the heap in Statement A. Note that BUILDHEAP will generally do nothing if the initial permutation is in heaporder; so any particular heap can be constructed by BUILDHEAP.

Consider Fig. 1, which gives some idea of the initial heap. Here $p = \lceil 10 \lg \lg n \rceil$. We assume the element positions are numbered from left to right and we imagine the bottom $p+1$ levels of T_n divided into 2^{k-p-1} subtrees $\tau_1, \tau_2, \ldots, \tau_M$, $M = 2^{k-p-1}$ (where τ_1 is the *rightmost* subtree), each subtree having $2^{p+1} - 1$ elements of which 2^p are leaves. So the leaves of τ_1 are first placed at the top of the heap in Statement A, then those of τ_2, etc.

The labels L or S inside each triangle indicate that the corresponding subtree is filled with (mostly) large or (all) small elements. The subtrees τ_1, $\tau_2, \tau_3, \ldots, \tau_{2m}$ are alternately (mostly) filled with large or filled with small elements, where

$$m = \left\lfloor \frac{2^{k-1} - 2^{k-p-1} + 1}{2^{p+1} - 1} \right\rfloor.$$

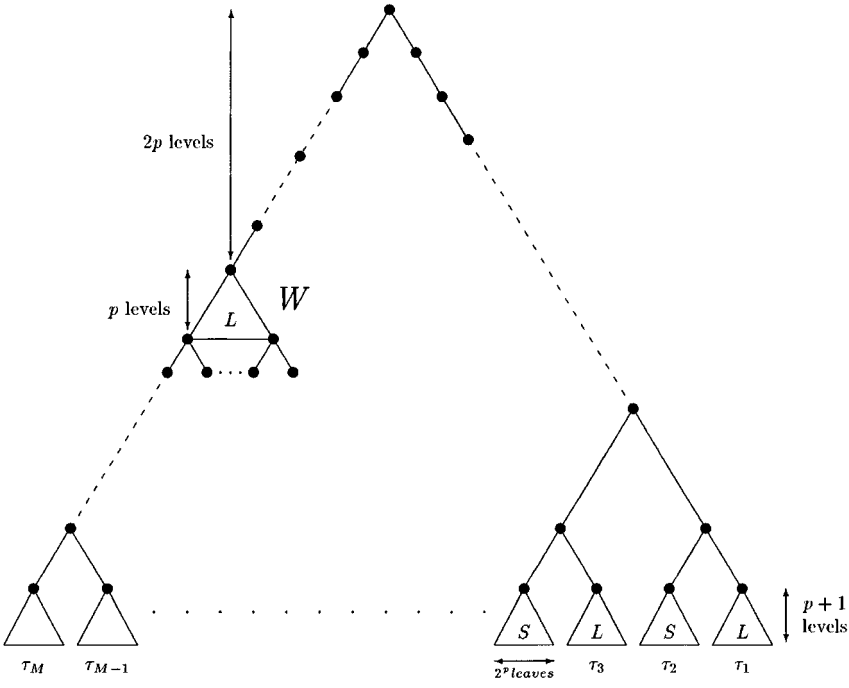

FIG. 1. Initial heap.

The remaining subtrees $\tau_{2m+1}, \ldots, \tau_M$ can be filled arbitrarily in a heap-consistent fashion. We further assume that all of the $2^{k-p-1} - 1$ elements in the first $k - p - 1$ levels of the tree are large.

We now come to the purpose of the subtree labeled W (the *waiting room*). The leaves of the subtrees $\tau_1, \tau_3, \tau_5, \ldots, \tau_{2m-1}$ are (almost all) large. The leaves of the trees $\tau_2, \tau_4, \ldots, \tau_{2m}$ are, of course, all small. We will show how to design a heap such that (in essence) in Round 1

$$\textit{large elements contained in the leaves of } \tau_{2i-1}, \ 1 \le i \le m, \textit{ drop into } W \quad (4)$$

and

$$\textit{the leaves of } \tau_{2i} \textit{ drop into the first } p \textit{ levels of } \tau_{2i-1}, \quad 1 \le i \le m. \quad (5)$$

This implies that no leaf of τ_j, $1 \le j \le 2m$, will interfere with or displace any leaf in a subtree to the left of itself.

Given this, we see that no matter what happens to the remaining leaves, we will have achieved the objective of making most of the large elements at the lowest level fall a short distance only.

We do not need to give a complete description of the initial heap. It will suffice for us to give enough information about its structure to verify (4) and (5).

We use a partition of $[n]$ into sets M_0^-, $\{M_i^+, M_i^0, M_i^-, i = 1, 2, \ldots, m\}$, M_{m+1}^+, which contain the large elements, and $S = [(n-1)/2]$, which contains the small elements. All we need to specify is that

$$x \in M_i^*,\ y \in M_{i+1}^{\#}, \quad *, \# \in \{-, 0, +\} \text{ implies } x > y, \tag{6}$$

$$x \in M_i^+,\ y \in M_i^0,\ z \in M_i^- \text{ implies } x > y > z. \tag{7}$$

We now fill in more details in Fig. 1 in order to give Fig. 2. First of all, observe the contents of τ_{2i-1}, $i = 1, 2, \ldots, m$; except for the bottom level,

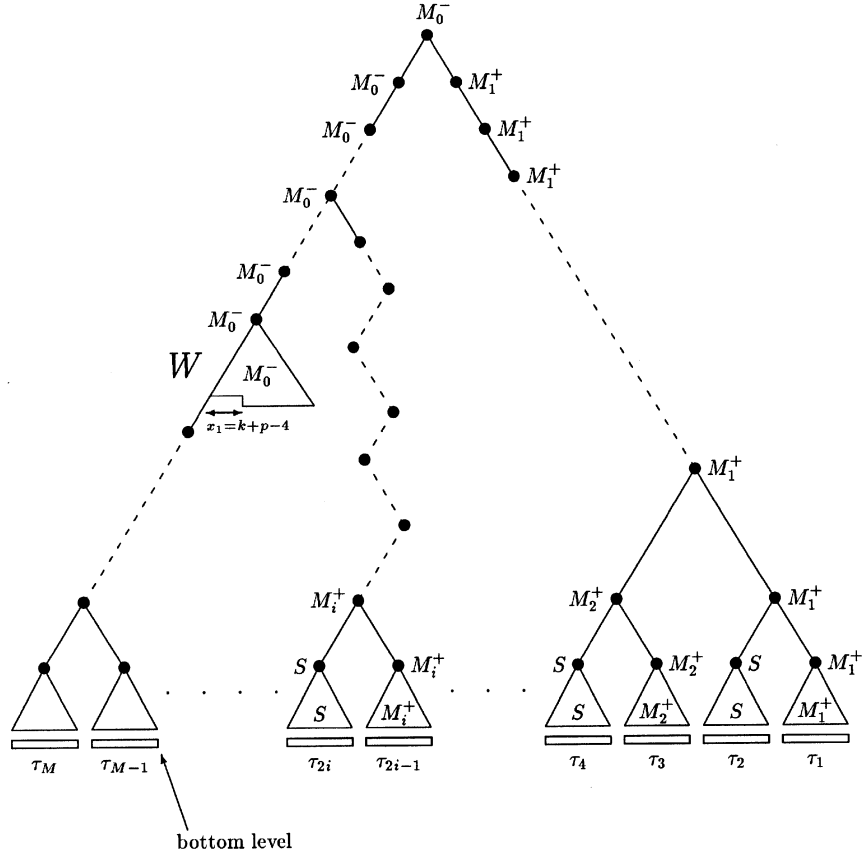

FIG. 2. Initial heap (more detailed).

these are all in M_i^+; the contents of the bottom levels are revealed in Fig. 3b below. The other trees $\tau_{2m+1}, \ldots, \tau_M$ are filled with elements from S and M_{m+1}^+. The contents of W are in M_0^-, which implies $H[2^i] \in M_0^-$ for $i = 0, 1, \ldots, 2p - 1$, as well. Notice that $x_1 = k + p - 4$ of the bottom positions are not considered to be part of W. For any position t in the remainder of the tree we specify that $H[t] \in M_j^+$ where j is as large as possible consistent with heap order. Thus $H[3] \in M_1^+$, $H[5] \in M_{M/4+1}^+$, $H[6] \in M_{M/8+1}^+$, $H[7] \in M_1^+$, etc. Notice that this determines the subsets of our partition. We do not need to be more specific about the actual contents, of say τ_1, other than requiring that consistency with heap order is maintained.

Having described the initial position, we now describe the ith position, $i = 1, 2, \ldots, m$ ($i = 1$ is the initial position). Figure 3 deals with i odd and Fig. 4 deals with i even.

The first thing to be checked is that Fig. 3a with $i = 1$ is consistent with Fig. 2. (Note that $q_1 = 1$.)

Assume inductively that Fig. 3 correctly represents the state of the heap after $2(i - 1)2^p$ executions of Statements A, B of Heapsort, where i is odd.

Consider the insertion of the next 2^p elements at the top of the heap (see Fig. 3b); these are the leaves of τ_{2i-1}.

It should be clear that

(i) The first $k - p - q_i - g_i - 2$ elements in M_i^0 will fall all of the way into τ_{2i-1} and then all the elements on the path XY will be in M_i^+. If any element in M_i^0 falls to the bottom level then the element it is swapped with must also be in M_i^0 so this does not affect the partition in Fig. 3b.

(ii) The next $2^p - x_i + 2p - q_i - 1$ elements in M_i^- will fall into W and fill it along with the path QR.

(iii) The next $q_i + g_i$ elements in M_i^0 will fall down into τ_{2i-1} and, afterwards, the path from τ_{2i-1} to the root will contain elements in M_i^+ only.

At this point all of the elements in M_{i-1}^0 and M_{i-1}^- have been deleted from the heap. (We see that the purpose of the x_i "missing" elements in W is to make room for steps (i) and (iii) above.)

(iv) Then the 2^p small elements in the bottom level of τ_{2i} will fall down into τ_{2i-1} and make all of its elements small, along with the element in position P.

The state of the heap is now as in Fig. 4, with i increased to $i + 1$. We now consider what happens when i is even. It is, in fact, very similar to the previous case. The only difference is that we insert h_i small elements into the bottom row of τ_{2i-1}. Their purpose is to make sure that the path UV is

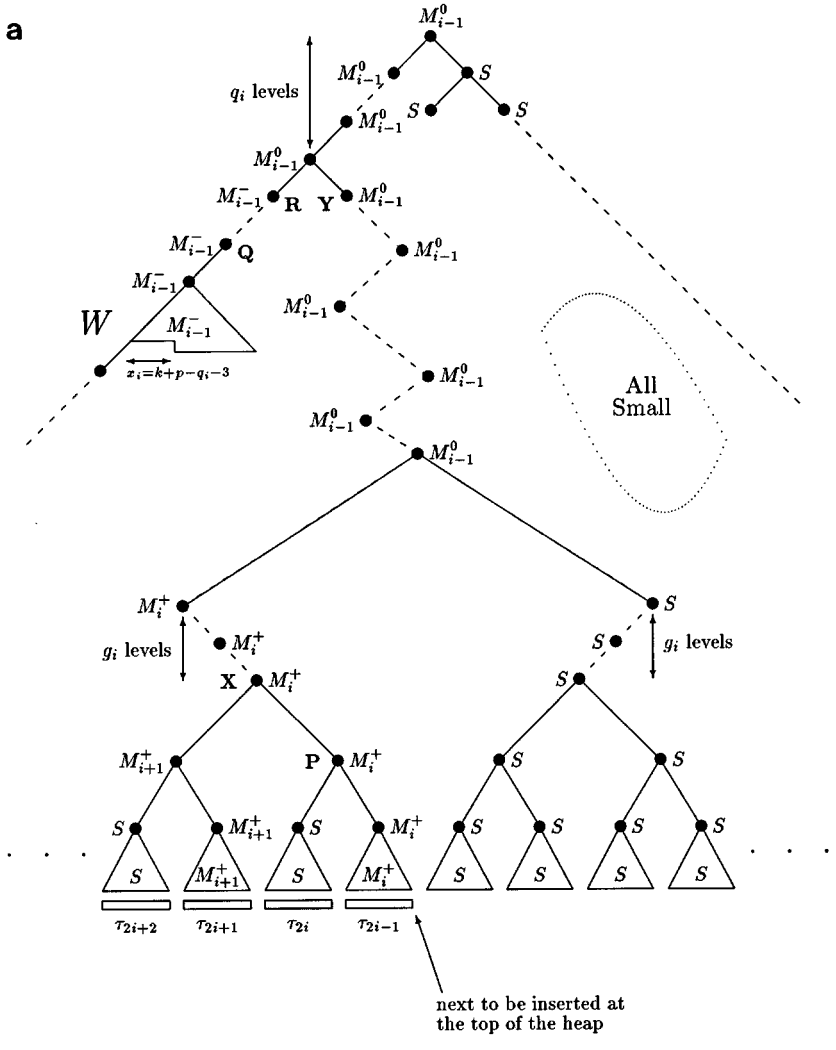

FIG. 3. (a) Odd i; (b) contents of the bottom level of τ_{2i-1}, i odd.

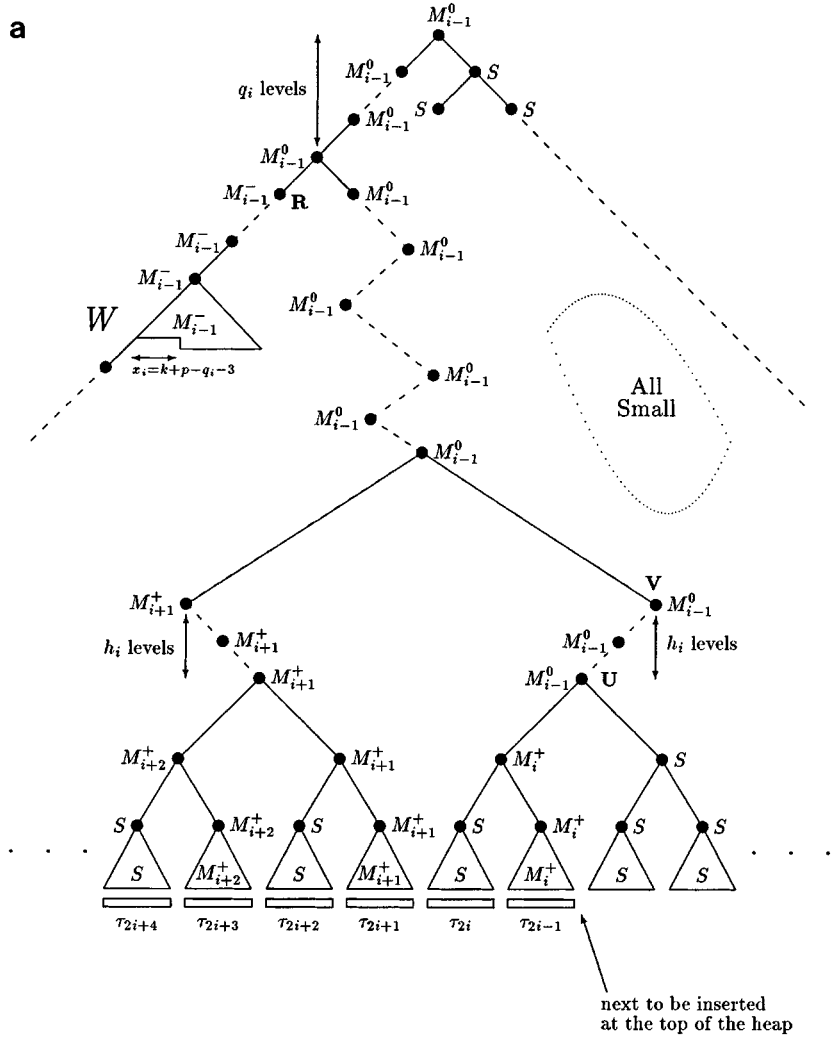

FIG. 4. (a) Even i; (b) contents of the bottom level of τ_{2i-1}, i even.

filled with small elements after the bottom levels of τ_{2i-1}, τ_{2i} have been inserted. By arranging the M_i^+ elements of τ_{2i-1} with the larger ones to the right, we can assume that any small element that falls to the bottom level will always be swapped with another small element.

The next point to consider is what happens at those points at which q_i increases by 1, i.e., when $q_{i+1} = q_i + 1$. In this case, $h_i = k - p - q_i - 2$ and $g_{i+1} = k - p - q_{i+1} - 2$, so the rightmost blocks of M_i^0 in Figs. 4b and 3b, for i and $i + 1$, respectively, will be empty. Increasing q_i also causes x_i to decrease by 1, i.e., $x_{i+1} = x_i - 1$. This can be achieved by letting one of the x_i "missing" elements in Fig. 4a be of size M_i^- instead of small.

Observe that if $i \le m$ then $q_i < 2p$ so that the node R is always above W.

The final point to consider in the insertion of level k is what happens to the elements in $\tau_{m+1}, \ldots, \tau_M$. Basically, we handle these in a worst-case scenario as they only contribute to the error term.

Now let $\nu =$ the number of executions of Statement C caused by level k. Then

$$\nu \le m2^p 3p + (2^{k-1-p} - m)2^p k + mk^2.$$

However, $m = 2^{k-p-2} - 2^{k-2p-3} + O(2^{k-3p})$ and so

$$\begin{aligned}\nu &\le np + \left(2^{k-1} - 2^{k-2} + O(k2^{k-p})\right)k \\ &= \frac{1}{4}n \lg n + np + O\left(2^{-p} n (\lg n)^2\right).\end{aligned}$$

We have said nothing yet about the disposition of the small elements. At first sight, it would seem that, since we have ignored the relationship between them, we can assume that after the first round the heap looks like a slightly smaller version of what has been described and that we can proceed inductively (indeed, we proceeded under this delusion until it was time to write the paper). As it turns out, Heapsort is just a little bit more complicated. Even though we would like to ignore it, we have, in fact, learned something about the small elements. We know for example that the small elements that were leaves of subtree τ_i are smaller "on average" than the nonleaves of τ_i, but they have been placed "to the right" in the next subtree. This is not how we would like the heap to be.

To fix this we will have to assume that after Round 1 the heap looks as in Fig. 1 except that the labels L and S in the trees τ_i, $i = 1, 2, ...,$ M are interchanged and that p is replaced by $p - 1$ and k is replaced by $k - 1$. Of course L and S now refer to those elements which leave the heap in Round 2 and those which do not. The reader should convince him/herself

that such a structure is consistent with what we have assumed about Round 1. In the ith step of the analysis of Round 2 we will concentrate on τ_{2i} and τ_{2i+1}. This does not include τ_1. Now we have not made any assumptions about the small elements that were in τ_M at the start of Round 1. It will be convenient now to assume that they were, in fact, the largest of the small elements, and it is possible to arrange that at the start of Round 2 the 2^{p-1} largest elements lie on the leftmost path of the heap and in the leftmost part of τ_M. Then during the first 2^{p-1} insertions of Round 2 the elements in the leaves of τ_1 will drop down the left-hand side of the heap. We can assume that after these elements have been inserted that the heap looks as in Fig. 2, except that inside the triangles representing τ_i, $i = 1, 2, \ldots, M$ we have S in τ_1, M_1^+ in τ_2, S in τ_3, M_2^+ in τ_4, etc. The partition into M_i^+, M_i^0, M_i^-, etc., has the same intent as that of Round 1, but of course the elements are smaller than in the previous partition. The elements in the (smaller) waiting room come from the left-hand side of the original heap. This is consistent with what we have assumed about the left-hand side of the original heap. Things will now work out much as before. The important thing is that the elements in τ_{2i} are larger than those in τ_{2i+1}. Round 2 progresses in an almost identical manner to Round 1, the contents of the bottom rows of the τ_i's being as in Figs. 3b and 4b but with p, k replaced by $p-1, k-1$.

The next question is what about Round 3? Fortunately this is identical to Round 1. This is because there is nothing to stop us making this assumption about the small elements of Round 2. Let us, for example, compare the contents of what remains of τ_1, τ_2. The contents of τ_2 come from what were the leaves of τ_3 in Round 2. There is nothing to stop us assuming that they are all smaller than those left in τ_1. There is also no reason why we cannot assume that what is now in τ_1 is greater than τ_j, $j \geq 2$. The elements of τ_2 that caused the complication for Round 2 have now all gone.

The above analysis can be made to hold for the first, say $p/2$, levels. After which there are only $o(n/\lg n)$ elements left and these can be treated in a worst-case manner. Consequently,

$$\begin{aligned}\mu(n) &\leq \sum_{i=2}^{\infty} \frac{1}{2^i} n \lg n + 2np + O\left(2^{-p} n (\lg n)^2\right) + O\left(2^{-p/2} n \lg n\right) \\ &= \frac{1}{2} n \lg n + O(n \lg\lg n),\end{aligned}$$

as claimed.

ACKNOWLEDGMENT

The authors thank the referee for helpful comments and careful reading of the paper.

Note added in proof. The average case performance has recently been obtained by Schaffer and Sedgewick (Journal of Algorithms 15, (1993) 61–75). They also obtained a construction for an asymptotically best-case heap which is of a similar type to that contained in this paper.

REFERENCES

1. B. Bollobás and I. Simon, Repeated random insertion into a priority queue, *J. Algorithms* **6** (1985), 466–477.
2. S. Carlsson, Average case results on heapsort, *BIT* **27** (1987), 2–17.
3. A. M. Frieze, On the random construction of heaps, *Inform. Process. Lett.* **27** (1988), 103–109.
4. G. H. Gonnet and J. I. Munroe, Heaps on heaps, *SIAM J. Comput.* **15** (1986), 964–971.
5. R. Hayward and C. J. H. McDiarmid, Average case analysis of heap building by repeated insertion, *J. Algorithms* **12** (1991), 126–153.
6. D. E. Knuth, ''The Art of Computer Programming, Volume 1, Fundamental Algorithms,'' Addison–Wesley, Reading, MA, 1973.
7. D. E. Knuth, ''The Art of Computer Programming, Volume 3, Sorting and Searching,'' Addison–Wesley, Reading, MA, 1973.
8. H. Lorin, ''Sorting and Sort Systems,'' Addison–Wesley, Reading, MA, 1975.
9. C. J. H. McDiarmid and B. Reed, Building heaps fast, *J. Algorithms* **10** (1989), 352–365.
10. I. Wegener, Bottom-up-Heapsort, a new variant of Heapsort beating on average Quicksort (if n is not too small.), *Theoret. Comput. Sci.* **118** (1993), 81–98.
11. J. W. J. Williams, ''Algorithm 232,'' *Comm. ACM* **7** (1964), 347–348.
12. N. Wirth, ''Algorithms + Data Structures = Programs,'' Prentice–Hall, Englewood Cliffs, NJ, 1976.