**Abstract.**

A new algorithm for rearranging a heap is presented and analysed in the average case. The average case upper bound for deleting the maximum element of a random heap is improved, and is shown to be less than $\lceil \log n \rceil + 0.299 + M(n)$ comparisons,*) where $M(n)$ is between 0 and 1. It is also shown that a heap can be constructed using $1.650n + O(\log n)$ comparisons with this algorithm, the best result for any algorithm which does not use any extra space. The expected time to sort $n$ elements is argued to be less than $n \log n + 0.670n + O(\log n)$, while simulation result points at an average case of $n \log n + 0.4n$ which will make it the fastest in-place sorting algorithm. The same technique is used to show that the average number of comparisons when deleting the maximum element of a heap using Williams' algorithm for rearrangement is $2\lceil \log n \rceil - 1.299 + L(n)$ where $L(n)$ also is between 0 and 1, and the average cost for Floyd-Williams Heapsort is at least $2n \log n - 3.72n$, counting only comparisons. An analysis of the number of interchanges when deleting the maximum element of a random heap, which is the same for both algorithms, is also presented.

CR categories: E.2, F.2.2.

**Keywords:** Sorting, priority queues, heaps, average-case analysis, heapsort.

1. **Introduction.**

The Heapsort algorithm was presented in 1964 by Williams [10] as an in-place sorting algorithm with a worst-case time of $O(n \log n)$. Later that year Floyd [5] made some improvements on the algorithm, so that the number of comparisons to sort $n$ elements became $2n \log n - 2n$, if $n = 2^h + 1 - 1$. Carlsson [1] introduced a variant that uses only $n(\log n + \log \log n - 0.82)$ comparisons.

The average number of comparisons of Heapsort has not been fully analysed, but Doberkat has presented several results on the subject [2, 3, 4]. Simulation gives $2n \log n - 3.0250n$ comparisons on the average [6].

The idea of the Heapsort algorithm is to regard the elements in an array as nodes in a complete binary tree of height $h = \lceil \log n \rceil$, where any node located at index $k$ has its children located at $2k$ and $2k + 1$. This tree is first arranged as a heap; a tree with the largest element at the root, and its children as roots of subheaps. This can be done in linear time. After that, the root is swapped

---

*) All logarithms in the paper are of base 2.

Received April 1986. Revised November 1986.
with the last element, and the heap rearranged with one element less. The rearrangement is repeated \( n \) times, each at \( O(\log n) \) cost.

A heap can also be used to implement a priority queue, where inserting an element or deleting the largest element in a heap can be done in \( O(\log n) \) time. An element to be inserted is stored at the first free position of the array and is swapped with its father if it is larger. This is repeated until the new element has found its place. The number of comparisons for the insertion is in the worst case equal to the height \( h \), and in the average case 2.607, which was shown by Porter and Simon [9]. The deletion of the largest element is very similar to the rearrangement in the Heapsort algorithm. The only difference is that the former root is not stored last in the array, but is deleted. Doberkat [3] showed that the average number of comparisons and interchanges was asymptotically equal to the respective numbers in the worst case if the size of the heap is a power of two, which is \( 2h \) and \( h \), respectively.

The heap construction algorithm of Floyd [5] takes \( 2n \) comparisons in the worst case, and Doberkat [2] gave the average of 1.88 \( n \) comparisons and 0.744 \( n \) interchanges. Gonnet and Munro [7] gave an algorithm for constructing heaps in 1.625 \( n \) comparisons in the worst case, but it requires extra space. They also gave an algorithm with a worst case of 1.82 \( n \) comparisons, and almost the same in the average case, which does not use any extra space.

2. An improved rearrangement procedure.

The central part of Heapsort and of deleting the largest element in a priority queue, is the rearrangement. There are two strategies for doing this. The most used is the one described by Williams, where the element moved from the last place in the array is compared with the sons of the root. The largest element is stored at the root, and the subheap where the largest element came from is rearranged until the element from the leaf has found its place. Two comparisons are made at each level of the heap which the element has passed; this is equal to the height of the heap in the worst case.

Munro and Gonnet [7] and Carlsson [1] use another technique where a path of maximum sons is found, and an element is inserted in that path by binary insertion. A path of maximum sons of a heap is the largest son of the root concatenated with the path of maximum sons of the subheap with that element as the root. This path is found in \( h \) comparisons. To get a good worst-case performance the element stored last in the array can be inserted in this path by binary insertion using only \( O(\log\log n) \) comparisons. On the other hand, if the element that has to be inserted in the path of maximum sons is inserted by using linear search from the leaf-end of the path the average cost is constant which will be shown in this paper, but the worst-case cost is now \( O(\log n) \). The