Information Processing Letters 24 (1987) 247–250
North-Holland

2 March 1987

A VARIANT OF HEAPSORT WITH ALMOST OPTIMAL NUMBER OF COMPARISONS

Svante CARLSSON

Department of Computer Science, Lund University, Box 118, S-22100 Lund, Sweden

Communicated by T. Lengauer
Received 7 October 1985
Revised 6 May 1986

An algorithm, which asymptotically halves the number of comparisons made by the common HEAPSORT, is presented and analysed in the worst case. The number of comparisons is shown to be $(n+1)(\log(n+1)+\log\log(n+1)+1.82)+O(\log n)$ in the worst case to sort n elements, without using any extra space. QUICKSORT, which usually is referred to as the fastest in-place sorting method, uses $1.38n \log n - O(n)$ in the average case (see Gonnet (1984)).

Keywords: Heapsort, sorting

1. Introduction

The HEAPSORT algorithm was presented in 1964 by Williams [5] as an in-place sorting algorithm with a worst-case time of O(n log n). Later that year, Floyd [2] made some improvements on the algorithm, so that the number of comparisons to sort n elements was $2n(\log n - 1)$, if $n = 2^k - 1$.

The idea of the HEAPSORT algorithm is to regard the elements in an array as nodes in a complete binary tree, where any node k has the children 2k and 2k + 1. This tree should first be arranged into a heap: a tree with the largest element at the root, and its children as roots of subheaps. This can be done in O(n) time. After that, the root changes place with the last element in the array, and the heap is rearranged with one element less. The rearrangement is repeated n times, each time at O(log n) cost.

2. The improved algorithm

The critical part of the HEAPSORT algorithm is the rearrangement procedure, where a leaf and the root in the heap change places. The former leaf is swapped with the largest of its new sons, and so on, until it is larger than both sons or it is a leaf. To do this, two comparisons are made at each level of the heap below the root, and the number of levels is at most $\lfloor \log k \rfloor$, if k is the number of elements left to be sorted.

In this improved version, no comparisons are made between a node and its children. Instead, a path of maximum sons is found, by letting the larger of the sons of the root be in that list, and connecting it with the path of maximum sons for that node. A binary insertion in such a path can be performed, since any path from the root to a leaf in a heap is an ordered list. The length of that path is at most $\lfloor \log k \rfloor$.

The only part of the HEAPSORT algorithm that has to be changed is the rearrangement procedure. First, a path of maximum sons is found. Choose the element to be inserted as the root of the heap to be rearranged during the creation of the heap, and the last leaf during the sorting. Then, a binary search for the place of the element, which has to be inserted in the path, is performed. This can easily be done after observing that any node j has the ancestors $j \text{ div } 2^k$, where k is the length of the path between the node and its ancestor ($j \text{ div } 2^k$

```
procedure REARRANGE(var A:vector; X:elementtype; I,N:integer);

var J,K, NrOfLevels, Bot, Top, Mid:integer;

begin
  J := 2*I;

{ * * * LINEAR SEARCH * * * }

  while J < N do
  if A[J] < A[J + 1] then J := 2 * (J + 1)
                     else J := 2 * J;

  if J > N then J := J div 2;   {J is now a leaf}

{ * * * BINARY SEARCH * * * }

  NrOfLevels := log(J div I);
  {Number of levels between I and J}
  TOP := NrOfLevels;   BOT := 0;

  while Top > Bot do
  begin
    Mid := (Top + Bot) div 2;
    if X ⩽ A[J div 2 * * Mid] then Top := Mid
                              else Bot := Mid + 1
  end;

{ * * *  INSERTION * * * }

  for K := NrOfLevels - 1 downto Top do A[J div 2 * * (K + 1)] := A[J div 2 * * K];
  A[J div 2 * * Top] := X;
end;

procedure HEAPSORT(var A:vector; N:integer);
{ * * * This procedure is similar to the usual HEAPSORT * * * }

var I     : integer;
    Temp: elementtype;

begin

{ * * * HEAP CREATION * * * }
```

Fig. 1. The HEAPSORT algorithm implemented in PASCAL, where 2 * *k means 2^k, and log(n) is a function that returns $\lfloor \log_2 n \rfloor$.

```
  for I := N div 2 downto 1 do REARRANGE(A, A(I), I, N);

{ * * * SORTING * * * }

  for I := N - 1 downto 1 do
  begin Temp := A[I + 1]; A[I + 1] := A[1];
        REARRANGE(A, Temp, 1, I)
  end;

end;
```

Fig. 1 (*continued*).

can be computed by shifting j k steps to the right). Finally, all elements that are larger than the last leaf are moved one step up, and the element is inserted. Note that the path of maximum sons depends only on the current configuration of the heap and not on the value of the element, which is to be inserted. In Fig. 1, the procedure HEAPSORT is given. The auxiliary procedure REARRANGE is at first used for the creation of the heap and then for the necessary rearrangement after removing the current maximum element from the root of the heap.

3. Analysis of the algorithm

The maximum number of comparisons during one call of the REARRANGE procedure is $h + \lfloor \log h \rfloor + 1$, where h is the height of the heap to be rearranged. If the number of elements to be sorted is $2^k - 1$, the number of comparisons between elements can be shown to be as follows.

During the creation of the heap

$\frac{1}{4}(n+1)$ heaps of height 1, $\frac{1}{8}(n+1)$ heaps of height 2, $\frac{1}{16}(n+1)$ heaps of height 3, and so on, are rearranged. This gives a total number of

$$\sum_{k=1}^{\log(n+1)-1} (k + \lfloor \log k \rfloor + 1)\frac{n+1}{2^{k+1}}$$

$$= (n+1)\left(\sum_{k=1}^{\log(n+1)-1} \frac{k}{2^{k+1}} + \sum_{k=1}^{\log(n+1)-1} \frac{1 + \lfloor \log k \rfloor}{2^{k+1}} \right)$$

$$\leqslant (n + 1 - O(\log n)) + 0.8164\ldots(n+1)$$

$$\leqslant 1.82n - O(\log n).$$

During the sorting

$\frac{1}{2}(n+1)$ heaps of height $\log(n+1) - 1$, $\frac{1}{4}(n+1)$ heaps of height $\log(n+1) - 2$, and so on, are rearranged to give a total of

$$\sum_{k=1}^{\log(n+1)-1} (\log(n+1) - k + \lfloor \log(\log(n+1) - k) \rfloor + 1)\frac{n+1}{2^k}$$

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

$$\leqslant (n+1)(\log(n+1) + \log\log(n+1)).$$

Compared with the average result for QUICKSORT, which is $> 1.38n(\log n - 2) + \log n$ (cf. [3]), HEAPSORT now uses fewer comparisons in the worst case than QUICKSORT does in the average case. The result for HEAPSORT is almost as good as

that for MERGESORT, which has optimal number of comparisons, n log n − n. MERGESORT requires extra storage space of O(n), which HEAPSORT does not.

4. Comments

Similar results have been presented in [4], where Gonnet and Munro by an elegant trick show that it is necessary and sufficient with log n + g(n) − ε(n) comparisons to rearrange a heap, where ε(n) is a function in the range [0, 1], and that 1.625... n + O(log n * g(n)) comparisons are sufficient to create the heap. That result is better than the algorithm described here, but no implementation of the HEAPSORT algorithm is given or analysed.

The number of comparisons can further be reduced by using a binary heap with scattered leaves (see [1]). The creation of a heap will then take 1.75n − O(log n) comparisons, and the sorting part $\frac{1}{2}$n less than when using a normal heap, if $\frac{1}{3}(n+1)$ is a power of 2.

References

[1] S. Carlsson, Improving worst-case behavior of heaps, Rept. LUNDFD6/(NFCS-3004)/(1-21)/(1983), Dept. of Computer Science, Lund Univ., 1983.
[2] R.W. Floyd, Algorithm 245—Treesort3, Comm. ACM 7 (12) (1964) 701.
[3] G.H. Gonnet, Handbook of Algorithms and Data Structures (Addison-Wesley, Reading, MA, 1984).
[4] G.H. Gonnet and J.I. Munro, Heaps on heaps, Proc. 9th ICALP, Aarhus, Denmark, July 12–16 (1982) 282–291.
[5] J.W.J. Williams, Algorithm 232, Comm. ACM 7 (6) (1964) 347–348.