# The Worst Case Permutation for Median-of-Three Quicksort

**Hannu Erkiö**
Department of Computer Science, University of Helsinki, Tukholmankatu 2, SF-00250 Helsinki 25, Finland

**An algorithm is given which forms the worst case permutation for one of the most efficient versions of quicksort (median-of-three quicksort). This makes the experimental evaluation of this important algorithm possible. The paper includes a simple experimental comparison of the median-of-three and original versions of quicksort.**

## 1. INTRODUCTION

Practically the only serious weakness of Hoare's quicksort[1] is its possibility to require $O(N^2)$ time units, which happens when an extremely unfavourable input permutation is sorted (the worst case). The effect of this drawback can be diminished in amount and made less likely by choosing the partitioning key as the median of a small sample of keys instead of one strictly determined key as in the original version of the algorithm, where the first key is used. The size of the sample and the ways of implementing quicksort are analysed by Sedgewick who recommends a sample of only three elements[2] and presents an efficient algorithm for the median-of-three quicksort.[3] (The idea of using a sample was already presented in Ref. 1).

As quicksort is generally considered to be the best internal sorting algorithm, and it is often used as a yardstick to which the efficiency of other algorithms is compared, it is essential that its performance is thoroughly understood. This includes the knowledge of the worst case behaviour of the algorithm, and especially when the algorithm is experimentally evaluated, one must be able to generate the worst possible input permutation for it.

We denote the part of the array $A[1:N]$ to be partitioned in one specific pass of quicksort by $A[L:R]$, and the partitioning element by $V$ (as in Ref. 3, for example). In Ref. 2 it is shown that the worst case of original quicksort is caused by successive degenerated partitions with partitioning elements always smaller or larger than all the other elements. In the original quicksort, $A[L]$ is used as the partitioning element, and thus the ordered input permutation $\{1, 2, \ldots, N\}$ trivially leads to the worst possible behaviour of the algorithm. The $O(N^2)$ complexity of this worst case is easily derived by counting the number of key comparisons needed in successive partitioning passes:

$$C1 = (N+1) + N + (N-1) + \cdots + 3 = (N^2)/2 + O(N)$$

In Section 2 we give an algorithm which forms the corresponding (non-trivial) worst case permutation for the median-of-three quicksort presented in Ref. 3 (Program 2, with a correction given in Ref. 4). In Section 3 we give some experimental results showing the practical worst case behaviour of the ordinary and median-of-three versions of quicksort.

## 2. THE ALGORITHM

By using the median of the first, middle, and last elements of $A[L:R]$, efficient partitions into parts of fairly equal sizes can be achieved in most practical situations. In the worst case, however, nothing prevents this sample of three elements from containing the two smallest (or, equivalently, the two largest) elements. The partitions are thus almost as degenerated as in the original quicksort, and the number of key comparisons is given by

$$C2 = (N+1) + (N-1) + (N-3) + \cdots$$
$$= (N^2)/4 + O(N)$$

In Ref. 2 an example of a corresponding worst case input permutation of numbers 1, 2, ..., 15 is given; the partitioning elements in consecutive passes are 2, 4, 6, ....

We base our generation of the worst case permutation on Sedgewick's Program 2 of Ref. 3 with the correction mentioned above. The three-element sample of this procedure consists of the second, middle, and last elements of the array to be partitioned; replacing the first element by the second has hardly any effect on the practical performance of the algorithm but it helps to keep the subarrays resulting from the partitioning random. Sedgewick's Program 2 terminates with a 'global' insertion sort which, after all partitions done, sorts the short (length less than or equal to $M$) subarrays which have not been partitioned any further. When the worst case permutation is sorted, there actually remains only one such subarray in true disorder.

The generation algorithm follows a strategy which is a reversal of the sorting: first, a reverse-order subarray $A[N-M+1:N]$ containing elements $N$, $N-1$, ..., $N-M+1$ is formed for the (worst case of) insertion sort, and then this subarray is repeatedly lengthened by the two next smaller elements until a permutation of $N$ elements is achieved. As the array is always extended to the left, symbols $N$ and $R$ can be equalized. In connection with the lengthening, elements $A[L]$, $A[L+1]$, $A[(L+N) \text{ div } 2]$, and $A[N]$ are arranged so that the degenerated partitioning occurs in the corresponding pass of sorting. Besides this maximization of the number of comparisons, the construction also induces the maximum number of exchanges in selecting the partitioning elements.

The generation algorithm is given below. The correctness of the algorithm can be shown by a simple inductive

CCC–0010–4620/84/0027–0276 $01.00

proof which we here sketch only informally. In the beginning of the cycle in the main loop of the algorithm, $A[L+2:N]$ contains the worst case permutation of $N-L-1$ elements which is a resulting subarray of one pass in the median-of-three quicksort. The four assignments in the loop extend the subarray by two elements which again corresponds to the result of the previous pass in the median-of-three quicksort. By comparing these four assignments with the operations which precede the selection of the partitioning element in the median-of-three quicksort it is easy to see that at the end of the loop $A[L:N]$ contains the worst case permutation of $N-L+1$ elements.

```
procedure worstpermutation (A, N, M);
value N, M; integer N, M;
integer array A;
begin
  integer i, L, L1;
  comment to avoid some non-essential special cases we assume that
          N > 3 (and M < N);
  if M = 1 then M := 2;
  if 2* ((N - M) div 2) ≠ N - M then M := M - 1;
  L1 := N - M - 1;
  if M = 1 then
  begin
    A[N - 2] := N - 1; A[N - 1] := N; A[N] := N - 2;
    L1 := L1 - 2
  end
  else if M = 2 then
  begin
    A[N - 3] := N - 2;        A[N - 2] := N - 1;        A[N - 1] := N;
  A[N] := N - 3;
    L1 := L1 - 2
  end
  else
    for i := 1 step 1 until M do A[N - M + i] := N + 1 - i;
  for L := L1 step - 2 until 1 do
  begin
    A[L] := A[(L + N) div 2];
    A[(L + N) div 2] := A[N];
    A[N] := L;
    A[L + 1] := L + 1;
  end;
end;
```

## 3. EXPERIMENTAL EVALUATION OF WORST CASES

In order to have a rough idea about the relative 'badness' of the median-of-three and the original versions of quicksort, we made some experiments with both procedures. Experiments were run on a Burroughs B7800, and the procedures were coded in Algol. The version for original quicksort (described in detail in Ref. 5) is non-recursive, and includes also the global insertion sort at the end. Thus the main difference between the two algorithms is just the choice of the partitioning element. The same value $M = 9$ was used for both algorithms. The results are presented in Table 1. For comparison, Table 1 also includes the average times used to sort some random materials by both algorithms.

The results show the differences between the two versions of quicksort: for random material the difference is only slight (generally less than 10%, consistent with Ref. 3), but in the worst case the median-of-three version uses only half the time of the original quicksort. The latter relation agrees very well with that between the numbers of comparisons, $C1$ and $C2$. Further, the known $O(N^2)$ worst case behaviour of the median-of-three quicksort is clearly visible.

**Table 1. Execution times of two versions of quicksort (in milliseconds)**

| | Worst case permutation | | Random material | |
|---|---|---|---|---|
| *N* | Median-of-3 quicksort | Original quicksort | Median-of-3 quicksort | Original quicksort |
| 100 | 6 | 11 | 5 | 5 |
| 250 | 31 | 58 | 12 | 12 |
| 500 | 117 | 228 | 26 | 29 |
| 1000 | 440 | 886 | 49 | 60 |
| 2500 | 2922 | 5764 | 158 | 159 |
| 5000 | 12,288 | 24,595 | 342 | 364 |

## REFERENCES

1. C. A. R. Hoare, Partition: Algorithm 63, Quicksort: Algorithm 64, and Find: Algorithm 65. *Comm. ACM* **4** (7), 321–322 (1961).
2. R. Sedgewick, Quicksort. *Ph.D. Thesis*, Stanford Comptr. Sci. Rep. STAN-CS-75-492, Stanford U., Stanford, Calif. (1975).
3. R. Sedgewick, Implementing quicksort programs. *Comm. ACM* **21** (10), 847–857 (1978).
4. R. Sedgewick, Corrigendum to Ref. 3. *Comm. ACM* **22** (6), 368 (1979).
5. H. Erkiö and E. Peltola, Algorithms for experimental analysis of some internal sort algorithms. *Report A-1978-1*, Department of Computer Science, University of Helsinki, Finland (1978).

Received February 1983