MR 122/71

Increasing the Efficiency of Quicksort

M. H. van Emden
Mathematical Centre, Amsterdam, The Netherlands

A method is presented for the analysis of various generalizations of quicksort. The average asymptotic number of comparisons needed is shown to be $\alpha n \log_2 (n)$. A formula is derived expressing α in terms of the probability distribution of the "bound" of a partition. This formula assumes a particularly simple form for a generalization already considered by Hoare, namely, choice of the bound as median of a random sample. The main contribution of this paper is another generalization of quicksort, which uses a *bounding interval* instead of a single element as bound. This generalization turns out to be easy to implement in a computer program. A numerical approximation shows that $\alpha = 1.140$ for this version of quicksort compared with 1.386 for the original. This implies a decrease in number of comparisons of 18 percent; actual tests showed about 15 percent saving in computing time.

KEYWORDS AND PHRASES: sorting, quicksort, information content, entropy, distribution of median
CR CATEGORIES: 3.73, 4.49, 5.31, 5.6

1. Quicksort

By sorting a sequence we mean arranging a sequence of numbers into nondecreasing order. We shall frequently refer to the *rank* of an element, which means the index of the place it occupies in the sorted sequence; the rank of a particular element, say x, will be denoted by $r(x)$. The sequence to be sorted will be referred to as $a[1], \cdots, a[n]$ or as $a[1:n]$. Quicksort (which is due to C. A. R. Hoare—see [4, 5]) is the obvious choice for sorting a sequence that can be contained within a random access memory because it combines efficiency with the advantage that, apart from the sequence itself, the necessary additional storage is proportional to $\log_2 (n)$.

The principle of quicksort may be described as follows. Take any real number y (let us refer to this as the *bound*). Suppose that there are integers p and q such that

$$\left.\begin{array}{l} 0 \le p < q \le n+1, \\ \text{among } a[1:p] \text{ none is greater than } y, \\ \text{among } a[q:n] \text{ none is smaller than } y. \end{array}\right\} \qquad (1)$$

Suppose now that p and q are any pair of integers satisfying all three of these conditions and also such that $(q - p)$ is minimal. If $p + 1 = q$, then we are ready; otherwise we have $a[p + 1] > y$ and a $[q - 1] < y$. These elements are interchanged, and again p is increased and q is decreased as much as possible. This results in a change of at least 1 in p or q so that, continuing in this way, we at last find that $p + 1 = q$. Let the final value of p be r. A *partition* is now completed, which means that of $a[1:r]$ none is greater, and of $a[r + 1:n]$ none is smaller than y.

The problem of sorting $a[1:n]$ is now reduced to sorting $a[1:r]$ and $a[r + 1:n]$ separately. Thus the length of the sequence to be sorted is successively reduced until only sequences of length 1 or 2 are left. We shall see that the efficiency of quicksort depends on the way of selecting y. For the theoretical analysis of the number of comparisons, it will not be necessary to specify how y is selected; only the existence of a probability distribution will be postulated for the resulting r.

2. Average Number of Comparisons

Hoare [5] showed that the average number of comparisons done by quicksort is, asymptotically for large n, equal to $2n \ln (n)$. He also showed, by means of an information-theoretic argument, that the minimum number of comparisons is $n \log_2 (n)$. This minimum would be achieved if the bound is always in the middle.

This section derives, by means of a similar argument, that the average number of comparisons is asymptotically equal to $\alpha\, n \log_2 (n)$, where α is independent of n and we express α in terms of the probability distribution of the bound. Applications of this formula to the distributions considered by Hoare yield his results.

The number of comparisons required by a particular sorting algorithm may vary greatly for sequences of the same length; so, if we are to compare the performance of different sorting algorithms, we must define some sort of average. Suppose that no two elements are equal to each other; in that case not the elements themselves are important, but only their rank; hence a sequence may be regarded as a permutation of the integers $1, \cdots, n$. In the sequel we mean by "average" the average over the ensemble of all permutations of $1, \cdots, n$ where each occurs with the same probability.

An information-theoretic argument yields a formula for the average number of comparisons necessary to sort a sequence of length n. We can view the process of sorting as

This paper is related to an algorithm of the same title, which will be published in a later issue of *Communications of the ACM* [9].

one of collecting information about the particular permutation initially presented, because we would be able to reconstruct this permutation working backward from the sorted sequence if a record were kept of interchanges effected. Therefore, we must gain an amount of information equal to the uncertainty inherent in the random drawing of a permutation.

We suppose that, before the partition, any of the permutations of $1, \cdots, n$ are equally probable. According to Shannon's theory of information [7], the uncertainty in this situation equals the entropy of the discrete probability distribution $\{p_1, \cdots, p_{n!}\}$ where $p_i = 1/n!$ for $i = 1, \cdots, n!$:

$$-\sum_{i=1}^{n!} p_i \log_2 (p_i) = \log_2 (n!) \text{ bits}.$$

If, before the partition, $a[1:n]$ contains any of the possible permutations of $1, \cdots, n$ with probability $1/n!$, then afterward the equivalent statement holds for $a[1:r]$ and $a[r+1:n]$. This may be verified as follows. The algorithm described in Section 1 may also be described as the successive random drawing without replacement from an urn initially containing balls numbered $1, \cdots, n$. Suppose that $p + n - q + 1$ balls have been drawn already. Another ball is drawn, if available, and it is placed in $a[p+1]$ if smaller than y and in $a[q-1]$ otherwise. Thus $a[1:r]$ is obtained by drawing balls from the urn randomly, without replacement, under the condition that its number be not greater than r. This implies that every permutation of $\{1, \cdots, r\}$ has probability $1/r!$ of actually occurring. Moreover, which ball with number greater (not greater) than r is drawn, is independent of the balls with number not greater (greater) than r drawn previously. Therefore, the permutation being formed in $a[1:r]$ is independent of the one being formed in $a[r+1:n]$ and vice versa.

This implies that the number of possibilities after the partition is $r!(n-r)!$, each with equal probability, so that the uncertainty is $\log_2 (r!(n-r)!)$. Introducing a quantity H, we find for the information yield of a partition, which equals the decrease of uncertainty,

$$nH = \log_2 (n!) - \log_2 (r!) - \log_2 ((n-r)!).$$

In quicksort, the bound y is selected in such a way that $r(y)$ has probability $1/n$ to become equal to $1, \cdots, n$. The generalization that we will consider consists of introducing a different way of selecting the bound having a different probability distribution. We will regard r as a random variable, such that

$$\text{prob } \{\underset{\sim}{r} = r\} = f_r, \quad r = 1, \cdots, n, \quad \sum_{r=1}^{n} f_r = 1.$$

This gives for the expected information yield of a partition

$$E(n\underset{\sim}{H}) = \log_2 (n!) - \sum_{r=1}^{n} f_r (\log_2 (r!) + \log_2 ((n-r)!)) .$$

This equals, asymptotically for large n,

$$E(n\underset{\sim}{H}) \sim n \log_2 (n) - \sum_{r=1}^{n} f_r (r \log_2 (r) + (n-r) \log_2 (n-r))$$

and

$$E(\underset{\sim}{H}) \sim -\sum_{r=1}^{n} f_r \left(\frac{r}{n} \log_2 \left(\frac{r}{n}\right) + \frac{n-r}{n} \log_2 \left(\frac{n-r}{n}\right)\right).$$

For large n, the sum may be replaced by an integral:

$$E(\underset{\sim}{H}) \sim -\int_0^1 g(x)(x \log_2 (x) - (1-x) \log_2 (1-x))\, dx,$$

where $g(x)$ is the probability density function of $\underset{\sim}{x} = \underset{\sim}{r}/n$. In the sequel we shall confine our attention to symmetric distributions, that is, where $g(x) = g(1-x)$, and then we have:

$$E(\underset{\sim}{H}) \sim -2 \int_0^1 g(x) x \log_2 (x)\, dx. \qquad (2)$$

In our derivation of the efficiency of a sorting algorithm, apparently a partition of a sequence of length n is the natural unit, irrespective of the way in which this partition has been effected. However, it has become usual (see Hoare [5]) to use the comparison between an element and the bound as the unit. We need n of these to complete a partition; so $E(H)$ may be interpreted as the average information yield of a comparison.

This result (2) holds asymptotically for large n for partitions in sequences of length n. However, completing the process of sorting requires partitions in sequences of any length not smaller than, say, 2. Suppose θ is the proportion of all comparisons required for subsequences of length $\leq pn$, where $(2/n) \leq p < 1$, and n is the length of the original sequence. θ attains its maximum θ_m when all partitions end exactly in the middle:

$$\theta_m = \frac{pn \log_2 (pn)}{n \log_2 (n)} = p \left(1 + \frac{\log_2 (p)}{\log_2 (n)}\right).$$

For any $p > 0$ we can choose an n large enough to satisfy $(2/n) \leq p$; hence $0 < \theta_m < p$ and also $0 < \theta < p$. This implies that for any fixed p $(0 < p < 1)$ we can choose n so large that the proportion θ of all comparisons required for subsequences of length $\leq pn$ is smaller than p. Finally, we can make n sufficiently large to approach the asymptotic result (2) closely enough for sequences of length pn. This means, essentially, that, for $n \to \infty$, "almost all" comparisons are done in "large" sequences, and that expression (2) can be expected to apply to the total time taken by quicksort, not just to the first partitions.

Under our assumption of equally probable permutations of $1, \cdots, n$, the total amount of information to be gained during sorting is $n \log_2 (n)$; so that we find for the average number of comparisons required for a generalization of quicksort:

$$T_n \sim n \log_2 (n)/E(\underset{\sim}{H}) = \alpha n \log_2 (n), \qquad (3)$$

where

$$\alpha = \left(- \int_0^1 g(x) x \log_2 (x)\, dx \right)^{-1}.$$

This formula, which was supplied, with a faulty proof, in van Emden [2], is due to F. E. J. Kruseman Aretz [6], who obtained it by a different method. It allows us to derive two results given by Hoare [5] as special cases.

The first result applies to the original version of quicksort. Here $g(x) = 1$ if $0 \le x \le 1$, $g(x) = 0$ otherwise. Substituted in (3), this yields $\alpha = 2 \ln (2) = 1.386$, which is Hoare's result.

The second result applies to the theoretical minimum of α. Consider the random variable $\underset{\sim}{x}$ that has $g(x)$ as probability density function. We shall derive the minimum of α where $g(x)$ is allowed to vary over all symmetric functions such that $\int_0^1 g(x)\, dx = 1$. One of the forms that Jensen's inequality might assume (see, for instance, [1]) is:

$$E(f(\underset{\sim}{x})) \le f(E(\underset{\sim}{x})), \tag{4}$$

where $\underset{\sim}{x}$ is a random variable assuming nonnegative values with probability one, and f is a continuous and convex function.

If we choose $f(x) = -x \log_2 (x)$, we find

$$\begin{aligned} E(-\underset{\sim}{x} \log_2 (\underset{\sim}{x})) &= -\int_0^1 g(x) x \log_2 (x)\, dx \\ &= 1/(2\alpha) \le -E(\underset{\sim}{x}) \log_2 (E(\underset{\sim}{x}))^{\cdot} \end{aligned}$$

Because of the supposed symmetry of $g(x)$, $\mathrm{E}(\underset{\sim}{x}) = .5$; hence $\alpha \ge 1$, which is Hoare's result.

In Jensen's inequality, equality occurs if and only if $\underset{\sim}{x}$ assumes one value with probability 1. This value can only be .5 in the symmetric case; we may conclude that 1 is the lower bound for α, which is assumed if and only if $\text{prob}(\underset{\sim}{x} = .5) = 1$, which corresponds to such a choice of the bound in quicksort that every partition ends exactly in the middle.

3. Two Ways of Increasing Efficiency

3.1. *Bound as Median of a Random Sample.* Hoare [5] suggests an improvement of his algorithm to be obtained by choosing as the bound the median of random sample of size $2k + 1$, and remarks that the resulting saving is very difficult to estimate. This modification has also been studied by Frazer and McKellar [3]. The formula (3) for the asymptotic number of comparisons is applicable to this case, in which it assumes a particularly simple form.

It is a well-known fact in order statistics that, whatever the distribution of the elements themselves, their ranks are uniformly distributed and the probability density function of the median of a sample of size $2k + 1$ from a uniform distribution is $g(x) = x^k(1 - x)^k/B(k + 1, k + 1)$, where B is the Beta function. To find α, we have to evaluate the integral

$$\begin{aligned} \int_0^1 g(x) x \ln x\, dx &= (1/B(k+1, k+1)) \int_0^1 x^{k+1} \\ &\qquad \times \ln x (1-x)^k\, dx \\ &= (B(k+2, k+1)/B(k+1, k+1)) \\ &\qquad \cdot (\psi(k+2) - \psi(2k+3)) \\ &= (\psi(k+2) - \psi(2k+3))/2, \end{aligned} \tag{5}$$

where ψ is the logarithmic derivative of the gamma function:

$$\psi(x) = \frac{d}{dx} \ln \Gamma(x).$$

From (3) and (5) we find that

$$\begin{aligned} \frac{1}{\alpha_k} &= \frac{\psi(2k+3) - \psi(k+2)}{\ln (2)} \\ &= \frac{\dfrac{1}{k+2} + \dfrac{1}{k+3} + \cdots + \dfrac{1}{2k+2}}{\ln (2)} \\ &= \frac{1 - \dfrac{1}{2} + \dfrac{1}{3} - \cdots + \dfrac{1}{2k+1} - \dfrac{1}{2k+2}}{\ln (2)}. \end{aligned}$$

This shows that $\alpha_k > 1$ and $\lim_{k \to \infty} \alpha_k = 1$, as indeed required by Jensen's inequality (4). See Table I.

TABLE I

k	0	1	2	3	4	5	6	∞
α_k	1.386	1.188	1.124	1.092	1.073	1.061	1.053	1.000

It may be useful to remark that α_k is independent of n. Therefore, for any size $2k + 1$ of a random sample, for sufficiently large n, the time required to find the median of the sample is an arbitrarily small proportion of the time required to complete a partition. Although the asymptotic properties of this method do not depend on the efficiency of the method used to find the median, the following note may be of interest.

Van Wijngaarden [8] proposed the following modification of quicksort for finding the median of a sequence. Instead of sorting each of the parts designated by a partition, only that one is sorted in which the median lies. In this way the median is found as the last of a sequence of nested intervals containing it. He showed that the number of comparisons required is, asymptotically for large n, equal to βn, where β does not depend on n. Kruseman Aretz [6] showed that $\beta = 2 + 2 \ln (2)$.

3.2. *Bounding Interval.* Quicksort starts each partition by designating an element of $a[1{:}n]$ as the bound y. We found that efficiency may be improved by merely postponing the decision of what the bound is going to be. In fact, during the whole of the partition we only use an interval containing the bound; at any time any element of this interval could still be chosen without disturbing the partition obtained so far; hence the term "bounding interval" which we have chosen for this strategy.

To be specific, suppose that integers p and q satisfy the relations (1). In this situation there is no need to choose y as bound; but if, for instance,

$$xx = \max_i a[i], \qquad 1 \le i \le p,$$

and

$$zz = \min_i a[i], \qquad q \le i \le n,$$

then any element whose rank is not less than $r(xx)$ and not greater than $r(zz)$ might be chosen. To complete the partition, p must be increased and q must be decreased. It is possible to preserve the validity of (1) by, if necessary, interchanging elements, or by increasing xx, or by decreasing zz.

When at last $p + 1 = q$, the interval containing the bound has shrunk to the pair $\{xx, zz\}$, either of which could be chosen as bound. In actual fact neither is, because the necessity of a bound has vanished: the partition is complete already.

This particular way of effecting a partition implies a certain density function g. It depends on the supposition that the ranks represent a random permutation of the integers $1, \cdots, n$. It is only useful to use g to compute efficiency if it holds not only for the initial partition but also for all successive ones, that is, when the partition leaves a random permutation in each of the parts. That this is the case may be seen as follows.

Let r' be the rank of the final value of zz. A particular element in the left half is replaced if and only if its rank is $\ge r'$, even though the tests deciding its replacement are based on values of xx and zz that are, in general, not yet equal to their final values. Thus the replacements are drawn randomly, without replacement, from the uniform distribution on $1, \cdots, r' - 1$, and so are elements that remain in the left part.

4. Computing the Asymptotic Efficiency when Using a Bounding Interval

4.1. *The Succession of Intervals as a Random Walk on the Unit Half-square.* As shown in Figure 1, an interval may be represented by a point P on the unit square, where the coordinates are the rank (divided by n) of its endpoints xx and zz. Because $r(xx) < r(zz)$, P may only lie above the diagonal shown. The following description applies to the strategy embodied in procedure qsort. Picking the first x on the left that is greater than xx and the first z on the right that is smaller than zz corresponds to a uniformly distributed drawing of a point Q from the points of the unit square whose first (second) coordinate is greater (smaller) than that of xx (zz). These points form a rectangle of which P is the northwest corner; this rectangle may be referred to as the "shadow" of P.

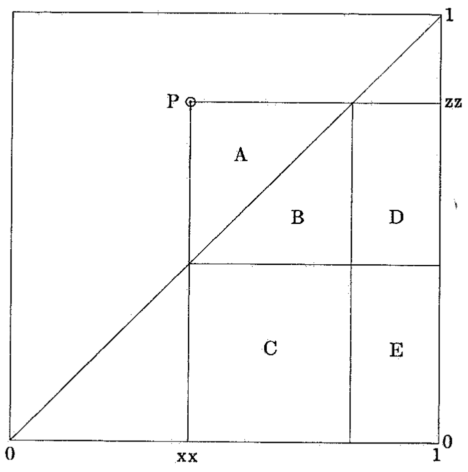

FIG. 1

The procedure qsort distinguishes whether Q is found in A, B, C, D, or E. If Q is found in A or B, both xx and zz are adjusted; if in C only zz, if in D only xx, and if in E neither xx nor zz are adjusted. Any of these adjustments is preceded by a reflection with respect to the diagonal if Q is found in B, C, D, or E. Under the assumption of a uniform distribution of Q on the shadow of P, the probability of each of these contingencies is proportional to its area. Hence we obtain the following succession rule:

If drawn from:	*Probability*	*Q uniformly distributed on:*	
A or B	$(zz\text{-}xx)^2u$	interior of A	
C	$xx(zz\text{-}xx)u$	west side of A	(6)
D	$(zz\text{-}xx)(1\text{-}zz)u$	north side of A	
E	$xx(1\text{-}zz)u$	P	

Here the factor $u = 1/(zz(1 - xx))$ ensures that these probabilities add up to 1.

This rule of succession defines a random walk on the upper half of the square with absorption on the diagonal. We may regard a partition as the following experiment:

(1) A point is drawn at random, according to a uniform distribution, from the upper half of the square.
(2) The successors according to the above rules are regarded as the stations of a random walk. (7)

Every random walk ends on the diagonal, and the probability density function resulting from this experiment is the function $g(x)$ needed to compute the asymptotic efficiency.

To approximate α we have set up the following discretized model of the experiment (7). The unit square is divided into a number of equal sized square cells, for every one of which we compute the probability mass (probability that the point is in the cell under the condition that it started where it did) at each stage of the experiment. We pick a certain cell such that all cells above it or to the left of it are empty (contain zero mass).

In this model the succession rules (6) are interpreted as rules governing the distribution of mass from a certain cell over all others of the unit half square that are not above it or to the left of it. Suppose that according to these rules a certain proportion, say p, remains in the cell. Another application of the rules leaves p^2, and so on. We require the result of infinitely many applications, and apparently this is achieved by computing the effect of a single application, multiplying all numbers so obtained by $1/(1 - p)$, and putting the mass of the selected cell equal to zero.

Let V be the set of cells not on the diagonal containing nonzero mass. As long as V is nonempty, it is possible to select an element of V that, according to the rules, may only give mass to other cells but may never receive any. Applications of the process described above to such a cell decreases the number of elements in V by 1. The computation continues until V is empty. Then all probability mass has diffused into the squares on the diagonal and their masses may be regarded as a discrete approximation of the density function g.

This computation was carried out with rules (6) corresponding to procedure qsort and it yielded $\alpha = 1.140$. Strictly, our analysis shows that the effectiveness of a partition is 18 percent greater than in quicksort, where $\alpha = 1.386$. Although qsort is very simple to program, a partition requires slightly more time than in the case of quicksort, because occasionally the bounding interval has to be adjusted. This probably explains an observed saving in computing time of about 15 percent.

Acknowledgments. The author is much indebted to Professor Dr. F. E. J. Kruseman Aretz of Philips Research Laboratories, Eindhoven, Netherlands, for valuable guidance in this matter; also to the referees, whose remarks caused the paper to be extensively revised.

Received October, 1969; revised March, 1970

REFERENCES

1. Beckenbach, E. F., and Bellman, R. *Inequalities.* Springer, New York, 1961.
2. van Emden, M. H. Iets quicker dan quicker. *Informatie 11* (1969), 30–32.
3. Frazer, W. D., and McKellar, A. C. Samplesort: a sampling approach to minimal storage time sorting. In proc. of the Third Annual Princeton Conf. on Information Sciences and Systems, 1969, 276–280.
4. Hoare, C. A. R. Algorithm 64, Quicksort. *Comm. ACM 4,* 7 (July 1961), 321.
5. Hoare, C. A. R. Quicksort. *Comput. J. 5* (1962), 10–15.
6. Kruseman Aretz, F. E. J. Private communication.
7. Shannon, C. *The Mathematical Theory of Communication.* U. of Illinois Press, Urbana, Ill., 1963.
8. van Wijngaarden, A. Private communication.
9. van Emden, M. H. Algorithm 402: Increasing the efficiency of quicksort. To appear in *Comm. ACM 11* (Nov. 1970).

MR 122 a

ALGORITHM 401 AN IMPROVED ALGORITHM TO PRODUCE COMPLEX PRIMES [A1]

PAUL BRATLEY (Recd. 25 Feb. 1970)
Département d'informatique, Université de Montréal, C.P. 6128, Montréal 101, Quebec, Canada

KEY WORDS AND PHRASES: number theory, prime numbers, complex numbers
CR CATEGORIES: 5.39

```
integer procedure cprimes(m, PR, PI);
  value m;  integer m;  integer array PR, PI;
comment The procedure generates the complex prime numbers
  located in the one-eighth plane defined by 0 ≤ y < x. Any prime
  found in that area has seven more associated primes: −x + yi,
  ±x − yi, ±y ± xi. These associated primes must be generated
  externally to cprimes. The first complex prime generated by
  cprimes is 1 + i, which exceptionally lies on x = y and has only
  three associated primes.
    The algorithm generates a list of complex primes in order of
  increasing modulus: the parameter m of the call is the highest
  modulus to be included in the list and should satisfy m > 2.
  PR and PI will contain respectively the real and imaginary
  parts of the generated list, with PR ≥ PI ≥ 0 for each prime.
  The value of the procedure is the number of primes generated.
    Algorithm 311 [1], sieve 2, is used to generate the rational
  primes less than m². Then it is known (see, for instance [2])
  that a rational prime p of the form p = 4n + 1 can be expressed
  as p = a² + b², and factorized as (a+bi)(a−bi) in the complex
  plane, where a + bi and a − bi are complex primes. For our
  present purpose we choose a > b and include only a + bi in the
  list. A rational prime p of the form p = 4n + 3 remains prime
  in the complex plane, so we include p + 0i in the list if p < m.
  Finally, the complex prime 1 + i may be thought of as one of
  the factors of the remaining rational prime 2 = (1+i)(1−i).
    Although this algorithm and Algorithm 372 [3] are not directly
  comparable, since they produce the list of complex primes in a
  different order, the accompanying remark suggests that the
  present algorithm is often to be preferred.
    REFERENCES:
  1. CHARTRES, B. A. Algorithm 311, Prime number generator 2.
       Comm. ACM 10 (Sept. 1967), 570.
  2. HARDY, G. H., AND E. M. WRIGHT. An Introduction to the
       Theory of Numbers, 4th ed. Clarendon Press, Oxford, 1965,
       Chs XII and XV.
  3. DUNHAM, K. B. Algorithm 372, An Algorithm to produce
       complex primes, CSIEVE. Comm. ACM 13 (Jan. 1970),
       52–53;
begin
  integer a, b, c, d, e, i, j, p, q;
  integer array P2[1:0.7×m↑2/ln(m)],
    P3[1:1.4×m/ln(m)];
  e := sieve 2(m↑2, P2);
  PR[1] := PI[1] := a := c := 1;
  b := 0;
  for d := 2 step 1 until e do
  begin
    p := P2[d];  q := p − 1;
    if (q÷4) × 4 ≠ q then
    begin
      if p ≤ m then
      begin b := b + 1;  P3[b] := p end
    end
    else
    begin
L1:
      if a ≤ b then
      begin
        if P3[a] ↑ 2 < p then
        begin
          c := c + 1;  PR[c] := P3[a];
          a := a + 1;  PI[c] := 0;
          go to L1
        end
      end;
      q := entier(sqrt(p/2)+1);
      for i := q step 1 until p do
      begin
        j := sqrt(p−i↑2);
        if i ↑ 2 + j ↑ 2 = p then go to L2
      end
      comment Note that the jump to L2 is always made before
        the cycle is terminated;
L2:
      c := c + 1;  PR[c] := i;  PI[c] := j
    end
  end;
L3:
  if a ≤ b then
  begin
    c := c + 1;  PR[c] := P3[a];
    a := a + 1;  PI[c] := 0;
    go to L3
  end;
  cprimes := c
end cprimes
```

ALGORITHM 402 INCREASING THE EFFICIENCY OF QUICKSORT* [M1]

M. H. VAN EMDEN (Recd. 15 Dec. 1969 and 7 July 1970)
Mathematical Centre, Amsterdam, The Netherlands

* The algorithm is related to a paper with the same title and by the same author, which was published in *Comm. ACM 13* (Sept. 1970), 563–567.

KEY WORDS AND PHRASES: sorting, quicksort
CR CATEGORIES: 5.31, 3.73, 5.6, 4.49

```
procedure qsort(a, l1, u1);
  value l1, u1;  integer l1, u1;  array a;
comment This procedure sorts the elements a[l1], a[l1+1], ··· ,
  a[u1] into nondescending order. It is based on the idea described
  in [1]. A comparison of this procedure with another procedure,
  called sortvec, obtained by combining C. A. R. Hoare's quicksort
  [2] and R. S. Scowen's quickersort [3], in such a way as to be
  optimal for the Algol 60 system in use on the Electrologica X-8
  computer at the Mathematical Centre is shown below. Here
```

"repetitions" denotes the number of times the sorting of a sequence of that "length" is repeated; "average time" is the time in seconds averaged over the repetitions; "gain" is the difference in time relative to time taken by *sortvec*.

procedure	*length*	*repetitions*	*average time*	*gain*
sortvec	30	23	.09	
qsort	30	23	.06	+.37
sortvec	300	16	1.25	
qsort	300	16	1.03	+.17
sortvec	3000	9	17.43	
qsort	3000	9	15.25	+.13
sortvec	30000	2	232.46	
qsort	30000	2	197.96	+.15

REFERENCES:
1. VAN EMDEN, M. H. Increasing the efficiency of quicksort. *Comm. ACM 13* (Sept. 1970), 563–567.
2. HOARE, C. A. R. Algorithm 64, quicksort. *Comm. ACM 4* (July 1961), 321–322.
3. SCOWEN, R. S. Algorithm 271, quickersort. *Comm. ACM 8* (Nov. 1965), 669;

```
begin
 integer p, q, ix, iz;
 real x, xx, y, zz, z;
 procedure sort;
 begin
  integer l, u;
  l := l1;  u := ul;
part:
  p := l;  q := u;  x := a[p];  z := a[q];
  if x > z then
  begin y := x;  a[p] := x := z;  a[q] := z := y end;
  if u − l > 1 then
  begin
   xx := x;  ix := p;  zz := z;  iz := q;
left:
   for p := p + 1 while p < q do
   begin
    x := a[p];
    if x ≥ xx then go to right
   end;
   p := q − 1; go to out;
right:
   for q := q − 1 while q > p do
   begin
    z := a[q];
    if z ≤ zz then go to dist
   end;
   q := p;  p := p − 1;  z := x;  x := a[p];
dist:
   if x > z then
   begin
    y := x;  a[p] := x := z;
    a[q] := z := y
   end;
   if x > xx then
   begin xx := x;  ix := p end;
   if z < zz then
   begin zz := z;  iz := q end;
   go to left;
out:
   if p ≠ ix ∧ x ≠ xx then
   begin a[p] := xx;  a[ix] := x end;
   if q ≠ iz ∧ z ≠ zz then
   begin a[q] := zz;  a[iz] := z end;
   if u − q > p − l then
   begin l1 := l;  ul := p − 1;  l := q + 1 end
   else
   begin ul := u;  l1 := q + 1;  u := p − 1 end;
   if ul > l1 then sort;
   if u > l then go to part
  end
 end of sort;
 if ul > l1 then sort
end of qsort
```

REMARK ON ALGORITHM 343 [F1]
EIGENVALUES AND EIGENVECTORS OF A REAL GENERAL MATRIX [J. Grad and M. A. Brebner. *Comm. ACM 11* (Dec. 1968), 820–826]

WILLIAM KNIGHT AND WILLIAM MERSEREAU (Recd. 7 Apr. 1970)
Computing Center, University of New Brunswick, Fredericton, New Brunswick, Canada

KEY WORDS AND PHRASES: eigenvalues, eigenvectors, latent roots, Householder's method, QR algorithm, inverse iteration
CR CATEGORIES: 5.14

This remark reports certain failures of Algorithm 343 when applied to pathological matrices. The smallest example is a 4 × 4 matrix for which 16 guard bits (5+ digits) proved insufficient; all computed eigenvalues were incorrect in the most significant digit.

The algorithm was implemented on an IBM System/360 model 50 using Fortran IV-G. The program was not modified to operate completely in double precision as was done for Knoble's certification [2]. Satisfactory agreement was obtained for the three sample matrices given with the algorithm.

Example A

$$\begin{array}{rrrr} -50 & 53 & 52 & 51 \\ -52 & 1 & 53 & 52 \\ -53 & 0 & 1 & 53 \\ -51 & 53 & 52 & 52 \end{array}$$

The exact eigenvalues are all 1. The computed eigenvalues follow. (Computed eigenvalues are reported rounded to 2 places after the decimal point, any further figures being, rather obviously, pointless.)

$$\begin{array}{r} 2.35 \\ 1.03 \pm 1.38\,i \\ -0.41 \end{array}$$

The maximum error in a computed eigenvalue exceeds 2 percent of the largest element of the matrix.

Example B

$$\begin{array}{rrrrrr} -41 & 55 & 4 & 3 & 2 & 51 \\ -2 & 10 & 55 & 4 & 3 & 2 \\ -3 & 0 & 10 & 55 & 4 & 3 \\ -4 & 0 & 0 & 10 & 55 & 4 \\ -55 & 0 & 0 & 0 & 10 & 55 \\ -51 & 55 & 4 & 3 & 2 & 61 \end{array}$$

The exact eigenvalues are all 10. The computed eigenvalues:

$$\begin{array}{r} 14.76 \pm 2.92\,i \\ 9.70 \pm 5.33\,i \\ 5.54 \pm 2.39\,i \end{array}$$

The maximum error in a computed eigenvalue exceeds 9% of the largest element in the matrix.