In Random Structures and Algorithms, 7:4, 1995, pp. 287-300

Optimal Sampling Strategies for Quicksort

C. C. McGeoch
Department of Mathematics and Computer Science, Amherst College, Amherst, MA 01002

J. D. Tygar*
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213

ABSTRACT

A well-known improvement on the basic Quicksort algorithm is to sample from the subarray at each recursive stage and to use the sample median as the partition element. *General sampling strategies*, which allow sample size to vary as a function of subarray size, are analyzed here in terms of the total cost of comparisons required for sorting plus those required for median selection. Both this generalization and this cost measure are new to the analysis of Quicksort. A square-root strategy, which takes a sample of size $\Theta(\sqrt{n})$ for a subarray of size n, is shown to be optimal over a large class of strategies. The square-root strategy has $O(n^{1.5})$ worst-case cost. The exact optimal strategy for a standard implementation of Quicksort is found computationally for n below 3000. © 1995 John Wiley & Sons, Inc.

1. INTRODUCTION

Quicksort is among the most thoroughly analyzed of comparison-based sorting methods. The basic operation of the algorithm partitions an array $X = x_1, x_2, \ldots, x_n$ around a partition element $p = x_i$ so that elements with value less than

* Supported in part by ARPA Contracts F33615-90-C-1465 and F22615-93-1-1330, NSF Presidential Young Investigators Award CCR-88-508087, matching funds from Motorola and TWR, a contract from the US Postal Service, and by an equipment grant from IBM.

p are to its left in the array and larger elements are to its right. Quicksort then recurs on the two subarrays on either side of p. A strategy ensuring average-case behavior is to choose p at random at each stage. A well-known generalization (see [8], [13], or [14]) takes a random sample of fixed size t at each recursive stage and uses the sample median as the partition element. We will call such a strategy a *fixed-sample (FS) strategy*.

We introduce a class of Quicksort algorithms for which the sample size can vary as a function of subarray size at each recursive stage. A selection strategy S corresponds to a *sampling function* $\sigma(n)$, which takes subarray size n and returns a simple size s. At each recursive stage, the *partitioning cost* is the number of comparisons required to partition a subarray of size n (always a fixed function of n), and the *selection cost* is the expected number of comparisons required to find the median of a sample of size s. Intuition suggests that large sample sizes will decrease partitioning cost but increase selection cost: we analyze the *total cost*, of partitioning plus selection, summed over all recursive stages. This cost model and the generalization to varying sample sizes are new to the analysis of Quicksort. Although this model only counts comparisons, the analytical methods developed here can be extended to cover other operations such as data swaps.

Total costs for fixed-sample strategies can be derived from standard analyses in the literature. We present the following results.

- We consider *two-tier (TT) strategies* which take varying sample sizes at the first stage only; at lower levels of recursion the sample is of fixed size t.

  For this class we derive an exact formula for the optimal sampling function when $t = 1$. We then generalize the formula to obtain the leading term of the optimal sampling function for arbitrary t. These sampling functions are of the form $\Theta(\sqrt{n})$. For any problem size N, the optimal two-tier strategy has lower total cost than any fixed-sample strategy.
- We also consider *square-root (SQ) strategies* which use a sampling function of the form $\Theta(\sqrt{n})$ at all recursive stages. We describe a square-root strategy that has lower total cost than any TT strategy.
- We show that any square-root strategy requires $O(n^{1.5})$ total comparisons in the worst case (including selection cost). There appears to be no other Quicksort-based algorithm in the literature that requires fewer than $O(n^2)$ worst-case comparisons, although it is known in the folklore that a median selection algorithm with an $O(n)$ worst-case time bound will give an $O(n \log n)$ worst-case bound for Quicksort. Our bound holds even if the selection algorithm itself has quadratic worst-case cost.
- For a given implementation we can compute exactly the optimal strategy using dynamic programming; we give optimal sample sizes for small n along with comparisons to costs of standard fixed-sample strategies. The number of comparisons required for a strategy using optimal sample sizes is about 88% of the number required for a tuned median-of-3 strategy. The optimal sample sizes obtained here grow roughly as $\sqrt{n}$.

Although two-tier strategies are introduced here primarily to simplify the analysis of the square-root strategy, they may provide an easy-to-implement

improvement over existing fixed-sample Quicksort programs. It is not clear whether the square-root strategy would give speedup in practice. These and other implementation issues are discussed further in Section 4.

The remainder of this section gives a brief review of selection strategies for Quicksort and of median-selection algorithms.

The original Quicksort algorithm, proposed by Hoare [6, 7] selected the leftmost subarray element at each recursive stage. Hoare noted that a randomly selected partition element would ensure expected-case behavior and suggested that larger samples might improve performance.

Sedgewick's [13, 14] implementations of Quicksort in the MIX language have provided the standard analytical model for exact analyses of Quicksort. Under this model, all keys are distinct, and the array elements are randomly ordered. The model also assumes that when subarray size n is less than sample size t, the sample is drawn with replacement. Sample size is always an odd positive integer.

Singleton [16] and Sedgewick [13, 14] analyzed fixed-sample strategies and recommended *median-of-3* Quicksort, where $t = 3$ at all recursive stages. Sedgewick showed that the percentage improvement in partitioning cost is small for $t > 3$, and he argued that selection cost for larger samples would quickly overtake any improvement in partitioning cost. In fact, the optimal sample size t is determined by some slowly increasing function of problem size N, but $t = 3$ has long been considered the best choice for typical problem sizes. Sedgewick's analysis was not made precise because selection cost was not an explicit part of the model; rather it was considered part of the overhead of each recursive call.

Sedgewick also analyzed a MIX implementation of *Samplesort*, which was proposed by Frazer and McKeller [4]. This strategy takes a single large sample from the array and sorts the sample once. The sample median is used as the partition element at the top level of recursion, the two sample quartiles at the next level, and so forth. Total partitioning cost is minimized when the sample size is $\Theta(n/\log n)$. Sedgewick showed that recursively sorting the sample by Samplesort gives optimal expected partitioning cost. However, because of high overhead, this approach is unlikely to be efficient in practice.

Many strategies have been proposed to improve performance of Quicksort when something is known about the initial input permutation, for example, when the elements are "mostly sorted." For comparative studies of such variations, see [1], [9], and [10].

Our generalization of Quicksort requires a median-selection algorithm as a subroutine. Hoare [6] presented a selection algorithm called FIND with behavior very similar to Quicksort. The sample is partitioned around a randomly chosen partition element and the algorithm recurs on the subsample that must contain the median. Let s be the sample size, and let the half-sample size be denoted by $h = (s - 1)/2$. The median element therefore has rank $h + 1$. To simplify some derivations in the following sections, we express the selection cost as a function of $h + 1$. Knuth ([8], Exercise 5.2.2-32) showed that the expected number of comparisons required for FIND to select the median of s elements is

$$FIND(h + 1) = 4(h + 1)(H_{2h+1} - H_{h+1}) + 4(h + 1) - 4H_{h+1} + 4/3 .$$

The last term disappears when $h = 0$; H_i denotes the ith harmonic number. The function $FIND(h + 1)$ is very well approximated by $6.772(h + 1) - 4 \ln(h + 1) - 2.3086$.

Floyd and Rivest [2, 3] presented a selection algorithm, called SELECT, which requires $3s/2 + O(\sqrt{s})$ [or equivalently $3h + O(\sqrt{h})$] comparisons to find the median in the expected case. Both FIND and SELECT can require a quadratic number of comparisons in the worst case. Schönhage, Paterson, and Pippenger [15] describe an algorithm that finds the median of s elements using $3s + o(s) = 6h + 3 + o(h)$ comparisons in the worst case.

2. ANALYSES OF SELECTION STRATEGIES

This section presents derivations of optimal selection strategies for each class. Following standard analyses, we assume that the keys to be sorted are distinct and that all initial input permutations are equally likely; for analysis purposes, this is equivalent to taking a random sample of the elements at each level. We first give the general formula for total expected cost for a strategy with sampling function $\sigma(n)$. We assume throughout that the sample size is found by rounding the sampling function value to the nearest odd integer. It will be convenient to distinguish between N, the input problem size, and n, the size of a subarray somewhere during the recursion. Thus we have $n = N$ only at the first stage of recursion.

Under standard models, partitioning cost at a single stage is equal to $n + 1$ (see, for example, [8], [13], or [14]). Note that it is possible, using sentinels, to implement a partitioning routine that requires only $n - 1$ comparisons per stage. We use $n + 1$ for direct compatibility with previous results, but it is straightforward to substitute $n - 1$ in the following derivations. Function $f(h + 1)$ denotes the expected cost of selecting the median of s elements, where $s = 2h + 1$.

The rank r of the partition element after partitioning determines the size of the two subarrays on either side of the partition and therefore the cost of recurring. Note that r refers to the relative rank of p among the n elements of the subarray rather than its absolute rank in the complete array. Since the partition element is the median of the random sample, the probability that p has rank r is the probability that the random sample of size s is chosen such that h of the elements have rank less than r, and h have greater rank.

The total expected cost $S(n)$ for sampling function $\sigma(n)$ is given by the following recursive formula:

$$S(n) = n + 1 + f(h+1) + \sum_{r=1}^{n} \frac{\binom{n-r}{h}\binom{r-1}{h}}{\binom{n}{2h+1}} [S(n-r) + S(r-1)]$$

$$= n + 1 + f(h+1) + \frac{2}{\binom{n}{2h+1}} \sum_{r=1}^{n} \binom{n-r}{h}\binom{r-1}{h} S(r-1).$$

All that is required is to find a closed form for this formula and to determine the function $\sigma(n)$ that minimizes $S(n)$. Of course, deriving a closed form for arbitrary $\sigma(n)$ is quite difficult. Instead, we will find optimal functions for simpler classes of strategies.

In fixed-sample (FS) strategies, $\sigma(n)$ equals a constant t for all $n \in [1 \ldots N]$. Although Sedgewick did not explicitly analyze selection costs, total costs for FS

strategies can be easily derived from his results [13, 14]. The following results are presented in the remainder of this section.

- Two-tier (TT) strategies have sampling functions which depend on both n and some (fixed)t:

$$\sigma_t(n) = \begin{cases} \gamma(t, n) & \text{if } n = N, \\ t & \text{if } n < N. \end{cases}$$

  In Section 2.1 we derive an exact formula for optimal sampling function $\gamma^*(1, n)$. We then derive the leading term of the optimal sampling function $\gamma^*(t, n)$ for general t. It turns out that $\gamma^*(t, n) = \Theta(\sqrt{n})$.

  Note that $\gamma^*(t, n)$ is only optimal for a given t. The *globally optimal* TT strategy would apply $\gamma^*(t^*, n)$ with the optimal choice of t^*, where the optimal t^* depends upon the problem size N. In Section 2.2 we prove that, for any N, the globally optimal TT strategy has lower total cost than any FS strategy.
- Square-root (SQ) strategies have sampling functions of the form $\sigma(n) = \Theta(\sqrt{n})$. In Section 2.2 we prove that the SQ strategy based on

$$\sigma(n) = \gamma^*(t^*, n) \qquad \text{for } 1 \le n \le N$$

  has lower total cost than the globally optimal TT strategy.
- Finally we show that any SQ strategy has sub-quadratic total cost in the worst case.

2.1. Optimal Two-Tier Strategies

A two-tier strategy takes a sample of size $s = \gamma(t, n)$ at the top level of recursion only. At all lower stages of recursion the partition element is chosen from some fixed sample size t. We first derive the optimal strategy $\gamma^*(1, n)$.

Sedgewick [13, 14] showed that the expected total partition cost of Quicksort when $t = 1$ is $2(n + 1)(H_{n+1} - 1)$. A one-element sample clearly has selection cost of zero. Therefore, the expected cost of a TT strategy with $t = 1$ is

$$TT_1(n) = n + 1 + f(h + 1) + \frac{2}{\binom{n}{2h+1}} \sum_{r=1}^{n} \binom{n-r}{h}\binom{r-1}{h} 2r(H_r - 1).$$

Considering the summation term alone, we have

$$\sum_{r=1}^{n} \binom{n-r}{h}\binom{r-1}{h} 2rH_r - \sum_{r=1}^{n} \binom{n-r}{h}\binom{r-1}{h} 2r,$$

which is equivalent to

$$2(h+1) \sum_{r=1}^{n} \binom{n-r}{h}\binom{r}{h+1} H_r - 2(h+1) \sum_{r=1}^{n} \binom{n-r}{h}\binom{r}{h+1}.$$

To find a closed form, we can use the following identities (see Graham, Knuth, and Patashnik [5], identities 5.26 and 7.63):

$$\sum_{r=1}^{n} \binom{n-r}{h}\binom{r}{h+1} = \binom{n+1}{2h+2},$$
$$\sum_{r=1}^{n} \binom{n-r}{h}\binom{r}{h+1} H_r = \binom{n+1}{2h+2}(H_{n+1} - H_{2h+2} + H_{h+1}). \quad (1)$$

The summation term becomes

$$2(h+1)\binom{n+1}{2h+2}(H_{n+1} - H_{2h+2} + H_{h+1} - 1),$$

and the complete formula for expected total cost is the surprisingly simple formula

$$TT_1(n) = n + 1 + f(h+1) + 2(n+1)(H_{n+1} - H_{2h+2} + H_{h+1} - 1).$$

Our goal is to find the value of h (and therefore of s) that minimizes $TT_1(n)$. Assume that a linear-expected-time selection algorithm is used, and replace $f(h+1)$ with the approximation $a(h+1) + b\ln(h+1) + c$ for appropriate constants a, b, and c. Replacing each harmonic number by the first three terms of the approximation $H_n \approx \ln n + \delta + 1/2n - O(1/n^2)$ (where $\delta \approx 0.577216$ is Euler's constant), we obtain

$$TT_1(n) \approx n + 1 + a(h+1) + b\ln(h+1) + c +$$
$$2(n+1)\left(\ln(n+1) - \ln 2 + \delta + \frac{1}{2(n+1)} + \frac{1}{4(h+1)} - 1\right).$$

The derivative with respect to h is

$$a + \frac{b}{(h+1)} - \frac{n+1}{2(h+1)^2},$$

and the approximation to $TT_1(n)$ is minimized when

$$h^* = \frac{-b + \sqrt{b^2 + 2a(n+1)}}{2a} - 1$$

or

$$s^* = \gamma^*(1, n) = \frac{-b + \sqrt{b^2 + 2a(n+1)}}{a} - 1.$$

The accuracy of this formula depends upon the quality of the approximations to $f(h+1)$ and to H_n. Note that for fixed n the function $TT_1(n)$ has two terms involving h: The first is $f(h+1)$ which is $\Theta(h)$, and the second is $2(n+1)(-H_{2h+2} + H_{h+1})$, which is approximately equal to $(n+1)/(2h+2) + c$ for a constant c. The sum of these two parts, one growing linearly in h and one decreasing as n/h, determine the optimal value h^*. Since this curve increases more sharply around its minimal value as n grows, and since the error terms in the approximations go to zero as n grows, we can argue that the formula for h^* must be correct when n is "large enough."

The correctness of the sampling function at small n depends upon the accuracy of the approximation to $f(h+1)$, which in turn depends upon the choice of

selection algorithm and its analysis. Assuming that algorithm FIND is used, we can plug in the constants $a = 6.772$, $b = -4$, and $c = -2.30886$, and check h^* against a straightforward computation of $TT_1(n)$. In this case the derived formula works perfectly for small n as well as for large n. Note that only the constant a contributes to the coefficient of the root-n term of s^*. With $a = 6.772$ for FIND, the coefficient is 0.5437, and with $a = 3$ for SELECT the coefficient is 0.8164.

The above analysis can be generalized to show that the optimal strategy for any t is a square-root function of n. Let $s = 2h + 1$ as before and let $t = 2k + 1$. Sedgewick [13] showed that partitioning cost for fixed t is

$$P_t(n) = \frac{(n+1)H_{n+1}}{H_{2k+2} - H_{k+1}} + \alpha(n),$$

where $\alpha(n)$ is a complicated $O(n)$ function. Total expected selection cost is proportional to selection cost $f(k+1)$ at each stage, times the number of stages $\beta(n, t)$. An exact formula for $\beta(n, t)$, which must decrease in t and must be $O(n)$ for fixed t, is not known for general t. Sedgewick shows that $\beta(n, 1) = n$ and $\beta(n, 3) = 6(n+1)/7 - 1$. Total cost for an arbitrary TT strategy is given by

$$TT_t(n) = n + 1 + f(h+1) + 2\sum_{r=1}^{n}\left[\frac{\binom{r-1}{h}\binom{n-r}{h}}{\binom{n}{2h+1}}\left(\frac{rH_r}{H_{2k+2} - H_{k+1}} + \alpha(r) + \beta(r, t)f(k+1)\right)\right].$$

We must deal with the unknown functions $\alpha(r)$ and $\beta(r, t)f(k+1)$. Both are $O(r)$; we shall see below that the approximations $\alpha(r) \approx c_1 r + c_2$ and $\beta(r, k)f(k+1) = d_1 r + d_2$ are sufficient to produce the leading term for the optimal sample size. We have

$$TT_t(n) \approx n + 1 + f(h+1) + 2\sum_{r=1}^{n}\left[\frac{\binom{r-1}{h}\binom{n-r}{h}}{\binom{n}{2h+1}}\left(\frac{rH_r}{H_{2k+2} - H_{k+1}} + c_1 r + c_2 + d_1 r + d_2\right)\right].$$

The summation can be broken into three parts, and closed forms found for each using the identities as before. First,

$$\begin{aligned}\frac{2}{\binom{n}{2h+1}(H_{2k+2} - H_{k+1})}\sum_{r=1}^{n}\binom{r-1}{h}\binom{n-r}{h} rH_r &= \frac{2(h+1)\binom{n+1}{2h+2}(H_{n+1} - H_{2h+2} + H_{h+1})}{\binom{n}{2h+1}(H_{2k+2} - H_{k+1})} \\ &= \frac{(n+1)(H_{n+1} - H_{2h+2} + H_{h+1})}{H_{2k+2} - H_{k+1}}.\end{aligned}$$

Second,

$$\frac{2(c_1 + d_1)}{\binom{n}{2h+1}}\sum_{r=1}^{n}\binom{r-1}{h}\binom{n-r}{h} r = (c_1 + d_1)(n+1).$$

Third,

$$2(c_2 + d_2) \sum_{r=1}^{n} \frac{\binom{r-1}{h}\binom{n-r}{h}}{\binom{n}{2h+1}} = 2(c_2 + d_2) .$$

Gathering terms, we have

$$TT_t(n) \approx (n+1)(1 + c_1 + d_1) + 2(c_2 + d_2) + f(h+1) + (n+1)\left[\frac{H_{n+1} - H_{2h+2} + H_{h+1}}{H_{2k+2} - H_{k+1}}\right].$$

Making substitutions as before for the harmonics and for $f(h+1)$, and differentiating with respect to h, we find that the approximation to $TT_t(n)$ is minimized when

$$h^* = \frac{-b + \sqrt{b^2 + a(H_{2k+2} - H_{k+1})(n+1)}}{2a(H_{2k+2} - H_{k+1})} - 1$$

or

$$s^* = \frac{-b + \sqrt{b^2 + a(H_{2k+2} - H_{k+1})(n+1)}}{a(H_{2k+2} - H_{k+1})} - 1 .$$

As before, the constants a, b, and c depend upon the choice of selection algorithm. The sum $H_{2k+2} - H_{k+1}$ is approximately equal to $\ln 2 - 1/(4k+4)$ or $0.693 - 1/(2t+2)$.

We have derived the leading term for the optimal sampling function for $TT_t(n)$, for given t. Note that once the closed forms are obtained, the approximations to $\alpha(n)$ and to $\beta(n,t)f(k+1)$ contribute only to a term that is constant with respect to h; therefore, they do not affect the leading term of the optimal sampling function. Tighter approximations to these functions contain extra terms that are $o(h)$, and those terms can only contribute a quantity that goes to zero as h increases. Similarly, the error terms in the approximations to $f(m)$ and the harmonics vanish asymptotically as h grows and cannot affect the leading term. As before, we can argue that this formula is correct for large n, and its accuracy can be checked for small n once a selection algorithm is chosen.

Let $\gamma^*(t,n)$ denote the true optimal sampling function for $TT_t(n)$. For any problem size N there must be some optimal t^* that minimizes total comparison cost over all TT strategies that use $\gamma^*(t,n)$. Thus the optimal strategy is a "global" strategy for problem size N rather than one that only depends upon subarray size. Our computational results suggest that the optimal t^* is determined by some slowly increasing function of N: For now we denote the optimal sampling function by

$$\sigma_N^*(n) = \begin{cases} \gamma^*(t^*, n) & \text{if } n = N , \\ t^* & \text{if } n < N . \end{cases}$$

The total cost of sorting an array of size n by this strategy is denoted by $TT_N^*(n)$.

2.2. The Square-Root Strategy

The optimal TT strategy finds an optimal sample size at the first stage of recursion, assuming fixed samples all lower stages. As an improvement to TT strategies, we consider a *square-root strategy*, which applies the sampling function

$$\sigma(n) = \gamma^*(t^*, n) \qquad \text{for } 1 \le n \le N .$$

The $\gamma^*(t^*, n)$ and t^* used here are as defined above for the optimal TT strategy.

In this section we will show that this strategy is optimal over a large class of strategies, and provide a ranking of the following five strategies according to total expected cost.

- Let $\widehat{FS}(N)$ denote the cost of the optimal fixed-sample strategy for a given problem size N. This strategy applies the sampling function $\sigma(n) = \hat{t}$, where $\hat{t}$ is optimal for N. Inspection of Sedgewick's formulas (modified to include selection cost) shows that $t = 1$ is optimal for values of N up to 2^{14} and $t = 3$ is optimal for N up to at least 2^{16}. In general, $\hat{t}$ is governed by a slowly growing function of N (see [11]).
- Let $FS^*(N)$ denote the expected cost of the FS strategy that applies $\sigma(n) = t^*$, where t^* is optimal for TT strategies. Very limited computational experience suggest that t^* grows more slowly than $\hat{t}$.
- Let $\widehat{TT}^*_N(N)$ denote the cost of the globally optimal TT strategy, as derived in the previous section. This strategy applies the sampling function

$$\sigma(n) = \begin{cases} \gamma^*(t^*, n) & \text{if } n = N , \\ t^* & \text{if } n < N . \end{cases}$$

- Let $\widehat{TT}_N(N)$ denote the cost of the TT strategy that applies

$$\sigma(n) = \begin{cases} \gamma^*(\hat{t}, n) & \text{if } n = N , \\ \hat{t} & \text{if } n < N , \end{cases}$$

  where $\hat{t}$ is as defined above.
- Let $SQ(n)$ denote the cost of the square root strategy, which applies

$$\sigma(n) = \gamma^*(t^*, n) \qquad \text{for } 1 \le n \le N .$$

Note that the problem size N is fixed beforehand and determines the globally optimal t^* and $\hat{t}$ for their respective strategy classes. The theorem below claims only that the relationships hold for the *total* expected cost of sorting, not necessarily for all $n < N$.

Theorem. *For a problem of size N, total comparison costs are related by*

$$SQ(N) \le TT^*_N(N) \le \widehat{TT}_N(N) \le \widehat{FS}(N) \le FS^*(N) .$$

Proof. We will demonstrate the inequalities working from right to left and numbering the arguments for reference.

(1) The fourth inequality is immediate since $\hat{t}$ is optimal for fixed-sample strategies.

(2) The third inequality is also immediate. When $n < N$, the two strategies are identical and their costs are identical; when $n = N$, the two-tier strategy chooses a sample size $\gamma^*(\hat{t}, n)$, which is optimal.

(3) The second inequality is immediate from the definition of t^* as optimal for all two-tier strategies.

(4) The first inequality, that $SQ(N) \leq TT^*_N(N)$, remains. We shall establish this inequality by induction on problem size N. Let R be an arbitrary value smaller than N, and consider the TT^*_R strategy that applies the sampling function

$$\sigma_R(n) = \begin{cases} \gamma^*(t^*, n) & \text{if } n = R\,, \\ t^* & \text{if } n < R\,. \end{cases}$$

The cost of sorting an array of size R by this strategy is denoted $TT^*_R(R)$. This is different from the TT^*_N strategy above because the "top level" is at $n = R$ rather than at $n = N$. (The t^* is the same in both, however).

The induction hypothesis is that $SQ(R) \leq TT^*_R(R)$ for all $R < N$. We shall show that this implies $SQ(N) \leq TT^*_N(N)$.

Now, $FS^*(R)$ is the cost of applying the fixed-sample strategy with t^* to a problem of size R. By the optimality of $\gamma^*(t^*, n)$ we have

$$SQ(R) \leq TT^*_R(R) \leq FS^*(R)\,.$$

Furthermore, $TT^*_N(R)$ is equal to $FS^*(R)$ whenever $R < N$, since the sampling functions are identically t^*. Therefore, we have

$$SQ(R) \leq TT^*_N(R) \qquad \text{whenever } R < N\,.$$

Consider now the costs $SQ(N)$ and $TT^*_N(N)$. When $n = N$, the two strategies are identical since they both take samples of size $\gamma^*(t^*, N)$. Let $C(N)$ denote partition plus selection cost when $n = N$, and let $P(N, r)$ denote the probability that the partition element has rank r. The recursive formulas for the two strategies are

$$SQ(N) = C(N) + 2\sum_{r=1}^{N} P(N, r)SQ(r)$$

and

$$TT^*_N(N) = C(N) + 2\sum_{r=1}^{N} P(N, r)TT^*_N(r)\,.$$

Since r can replace R in the above inequality, we have $SQ(r) \leq TT^*_N(r)$ for all $r < N$, which immediately implies that $SQ(N) \leq TT^*_N(N)$.

As a base case for the induction, note that both the square-root strategy and the optimal two-tier strategy will use sample size 1 when N is small and will therefore have equivalent costs. □

The above theorem shows that for any N the optimal TT strategy dominates all fixed-sample strategies, and that there is an SQ strategy that is at least as good as the optimal TT strategy. Furthermore, argument (4) implies that the SQ strategy dominates at all recursive stages.

Corollary. *$SQ(n) \leq ST^*_N(n)$ for all $n \in [1 \ldots N]$.*

Finally, we demonstrate that the worst-case total cost of any square-root strategy is $O(n^{1.5})$. To simplify notation, assume that the sample drawn is of size at least $2b\sqrt{n}+1$ and at most $c\sqrt{n}$ for appropriate constants b and c. Worst-case partitioning cost occurs when minimal or maximal elements are chosen as partition elements at each recursive stage. But if a sample of size at least $2b\sqrt{n}+1$ is drawn at each level, it is impossible for elements with rank below $b\sqrt{n}$ or above $n-b\sqrt{n}$ to become sample medians. The deepest level of recursion that can be reached by the algorithm is $O(\sqrt{n})$, and the algorithm does $O(n)$ work at each level (including partitioning and selection). Therefore, the total cost is $O(n^{1.5})$. Note that this bound holds even with quadratic worst-case cost for selection, since the sample is of size $O(\sqrt{n})$.

3. AN EXACT OPTIMAL STRATEGY

Given an exact formula for selection cost it is possible to compute directly the optimal sample size. A dynamic-programming algorithm can find $s=\sigma(n)$ (with $s=2h+1$ as before) to minimize costs under the standard analytical model of Quicksort:

$$C_0(n)=n+1+f(h+1)+\sum_{r=1}^{n}\frac{\binom{r-1}{h}\binom{n-r}{h}}{\binom{n}{2h+1}}\left[C_0(r-1)+C_0(n-r)\right].$$

The running time of the dynamic program is just quadratic in the largest n examined: Since the optimal $\sigma(n)$ is sublinear in n, it is only necessary to compare the current optimal value of s to $s+2$ at each value of n.

Since all median-selection algorithms manage to partition a sample around the median, a significant reduction in comparisons can be realized by avoiding reexamining the sample during the partitioning stage. If the sample is a contiguous section from the middle of the subarray, sentinels can be used to avoid checking for the sample ends during partitioning. The partitioning cost at each stage would be reduced from $n-1$ to $n-s$. A small modification to the dynamic program finds the optimal sample sizes and total comparison costs $C_T(n)$ for this tuned algorithm, using the above formula with $n-1$ replaced by $n-s$.

Hoare's selection algorithm FIND, although not the most efficient known, does have an exact formula for selection cost. Recall from Section 1 that

$$FIND(h+1)=4(h+1)(H_{2h+1}-H_{h+1})+4(h+1)-4H_{h+1}+4/3,$$

where the last term disappears when $h=0$.

Using this cost for $f(h+1)$ in the above computation of $C_0(n)$ and $C_T(n)$, the dynamic program produces the optimal sample sizes shown in Table I. The first row in each part gives the minimum subarray size n for which the corresponding s is optimal: For example, for the standard cost model, the best sample size is 3 for n between 35 and 92. Programs to generate these tables for larger n are available by request from the first author. Linear least-squares regression on a log-log scale suggests that the best power-law fit for the pairs from $C_0(n)$ is proportional to $n^{.588}$, and the best fit for the pairs from $C_T(n)$ is proportional to at least $n^{.475}$.

Table II presents total expected cost for the optimal strategy under the tuned model, the optimal strategy under the standard cost model, and for the fixed-

TABLE I. Optimal sample sizes using Hoare's selection algorithm, under a standard and a tuned cost model

Optimal Sample Sizes for Standard Model													
n	1	35	93	197	337	515	730	984	1274	1603	1968	2372	2813
s_n	1	3	5	7	9	11	13	15	17	19	21	23	25
Optimal Sample Sizes for Tuned Model													
n	1	23	58	129	224	346	495	670	891	1099	1354	1635	1942
s_n	1	3	5	7	9	11	13	15	17	19	21	23	25

sample strategies with $t = 1$ and $t = 3$. The numbers in parentheses give the ratio to the lower bound $N \log_2(N) - N/\ln 2$ on the expected number of comparisons required for sorting (see [8], Section 5.3.1). Note that these exact costs were obtained computationally, not by simulation.

In the table the fixed-sample costs $C_1(n)$ and $C_3(n)$ correspond to highly tuned implementations found in [13], [14], and [8] (Section 5.2.2 and Exercises). The median-of-3 implementation, for example, incorporates a sentinel to reduce partitioning comparisons at each stage; also the selection algorithm optimizes the number of comparison required for samples of size 3. The tuned optimal strategy uses a fairly inefficient general-purpose selection algorithm (Hoare's FIND), but nevertheless gives a considerable improvement in total comparisons. At these problems sizes, the number of comparisons required by the tuned optimal strategy is about 88% of the number required by the median-of-3 strategy. It would be interesting to learn how much further improvement is gained when the SELECT algorithm is incorporated in the model, but the dynamic program cannot be used here because an exact cost analysis for SELECT is not available.

4. CONCLUSIONS

An exact analysis shows that the square-root strategy and the optimal two-tier strategy require fewer total comparisons than widely known fixed-sample strategies for Quicksort. Computational results show that an optimal strategy using

TABLE II. Total comparison costs for four strategies[a]

N	2000	4000	6000	8000
$C_T(N)$	21599	47338	74613	102881
	(1.13)	(1.12)	(1.12)	(1.12)
$C_0(N)$	24664	53502	83879	115254
	(1.29)	(1.27)	(1.25)	(1.25)
$C_3(N)$	24420	53581	84535	116654
	(1.28)	(1.27)	(1.27)	(1.27)
$C_1(N)$	27395	60321	95339	131716
	(1.44)	(1.43)	(1.43)	(1.43)

[a] Numbers in parentheses represent ratios to lower bounds on average comparisons

Hoare's selection algorithm requires about 88% as many total comparisons as the standard median-of-3 strategy, and use of better selection algorithm would give even more improvement.

It remains to be seen whether these strategies can be made efficient in practice. One problem with the square-root strategy is the added cost of computing the sample size at each level: A table-lookup scheme which approximates computation of the square root (or just the optimal sample size) might be faster. Limited experimental results suggest that comparison cost is quite robust with respect to small variations in the sampling function; perhaps a hybrid strategy that chooses one of a small set of sample sizes, each corresponding to a highly tuned subroutine, would be best. A two-tier strategy, or a generalization that only computes square roots at the top few levels of recursion, may provide an easily-implemented improvement to existing Quicksort programs.

Finally, we note that a strategy that minimizes total comparisons does not necessarily minimize total running time: The number of swaps of subarray items also contributes to the leading term of running time, and the cost of procedure calls can dominate at small n. Limited experimental comparisons of other cost measures for fixed-sample strategies and the optimal strategy are described in [11] and [12].

ACKNOWLEDGMENTS

We thank Jon Bentley, who first suggested the idea of allowing sample sizes to vary as a function of array size. Mike Langston suggested the hybrid scheme which may be most efficient in practice. An anonymous referee made several useful suggestions.

REFERENCES

[1] C. R. Cook and D. J. Kim, Best sorting algorithm for nearly sorted lists, *Commun. ACM*, **23**(11), 620–624 (1980).

[2] R. W. Floyd and R. L. Rivest, The algorithm select—for finding the ith smallest of n elements, *Commun. ACM*, **18**(3), 173 (1975).

[3] R. W. Floyd and R. L. Rivest, Expected time bounds for selection, *Commun. ACM*, **18**(3), 165–172 (1975).

[4] W. D. Frazer and A. C. McKellar, Samplesort: a sampling approach to minimal storage tree sorting, *J. Assoc. Comput. Machinery*, **17**(3), 496–507 (1970).

[5] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1989.

[6] C. A. R. Hoare, Partition (algorithm 63), Quicksort (algorithm 64), and Find (algorithm 65), *Commun. ACM*, **4**(7), 321–322 (1961).

[7] C. A. R. Hoare, Quicksort, *Comput. J.* **5**(4), 10–15 (1962).

[8] D. E. Knuth, *The Art of Computer Programming: Volume 3, Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[9] R. Loeser, Some performance tests of "Quicksort" and "descendents,". *Commun. ACM*, **17**(3), 143–152 (1974).

[10] R. Loeser, Survey on algorithms 347, 426, and Quicksort, *ACM Trans. Math. Software*, **2**(3), 290–299 (1976).

[11] C. C. McGeoch, Experimental analysis of algorithms, Ph.D. dissertation, CMU-CS-87-124, Carnegie Mellon University, 1986.

[12] C. C. McGeoch, An experimenal study of median-selection in quicksort, *Proc. 24th Allerton Conference on Communication, Control, and Computing*, Univ. of Illinois, October 1986, pp. 19–28.

[13] R. Sedgewick, Quicksort, Ph.D. thesis, Stanford University, 1975.

[14] R. Sedgwick, Analysis of Quicksort Programs, *Acta Inf.*, **7**(4), 327–355 (1977).

[15] A. Shönhage, M. Paterson, and N. Pippenger, Finding the median, *J. Comput. Syst. Sci.*, **13**, 184–199 (1976).

[16] R. C. Singleton, An efficient algorithm for sorting with minimal storage (algorithm 347), *Commun. ACM*, **12**(3), 185–186 (1969).

Received October 19, 1993
Accepted October 17, 1994