CARNEGIE MELLON UNIVERSITY

AN ASYMPTOTIC ANALYSIS OF THE NUMBER OF COMPARISONS IN MULTIPARTITION QUICKSORT

A DISSERTATION
SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY
in
STATISTICS

by

KOK HOOI TAN

Department of Statistics
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

August 1993

CARNEGIE MELLON UNIVERSITY

COLLEGE OF HUMANITIES & SOCIAL SCIENCES

DISSERTATION

Submitted in Partial Fulfillment of the Requirements

For the Degree of **DOCTOR OF PHILOSOPHY**

Title **An asymptotic analysis of the number of comparisons in Multipartition Quicksort**

Presented by KOK H. TAN

Accepted by the DEPARTMENT OF STATISTICS

Readers _____ 7/23/93
(Chairman of Dissertation) Date

_____ 23 July 1993
Date

_____ 7/23/93
Date

_____
Date

Approved by the Committee on Graduate Degrees

_____ 7/25/93

Abstract

Quicksort can be generalized by splitting the keys to be sorted into s subsets ($s > 2$) during each partitioning stage. Quicksort is then the particular case of s=2. The asymptotic distribution of the number of comparisons in this class of *Multipartition Quicksort* is studied, with emphasis on the $s = 4$ version. The exact distribution of $T(n)$, the number of comparisons to sort n keys, is obtained recursively for small n. Exact upper and lower bounds for $T(n)$ are derived, and the asymptotic form of the mean is obtained by a regression approach. A class of recursively constructed distributions on the permutations of keys is derived under which the distribution of $T(n)$ may be found recursively.

A theorem of Rösler (1991) that established an asymptotic distribution for $\frac{T(n)-ET(n)}{n}$ in Quicksort is generalized to hold under a class of non-uniform distribution of permutations, and to the multipartition setting. The limit distribution is shown to have a positive density function almost everywhere, and to satisfy a fixed-point equation.

Numerical methods are used to iteratively approximate the limit distribution. The primary tool is that of successive substitution. The characteristic function, the density, and a random sample from the asymptotic distribution are obtained this way. These methods are computing-intensive, and the efficiency of implementation is considered. In particular, distributed and asynchronous iterative methods are discussed.

Two empirical studies compare the performance of Multipartition Quicksort with varying s in terms of the quantity $T(n)$, as well as the actual CPU time for sorting.

An analysis of Mergesort is also presented. The asymptotic distribution of the number of comparisons required is shown to be Gaussian.

Advisor: William F. Eddy

Acknowledgements

I would to thank my advisor Bill Eddy for introducing me to the topics in this thesis, and for his guidance and patience with me in the course of this research.

Thanks are also due to Mark Schervish for his insightful advice and discussion in addressing some of the research problems. A paper on Quicksort by Eddy and Schervish (1992) is the genesis of this thesis.

I would also like to extend my gratitude to the other members of my committee – Marc Berger, John Lehoczky, and Mike Meyer – for their comments and critiques of an earlier draft of this thesis, from which the current version has benefited.

I thank Audris Mockus and Shingo Que for discussing some theoretical and algorithmic problems in the thesis. In addition I gladly acknowledge Petros for sharing the interest in, and pursuit of, some related topics in Quicksort.

Last but not least, I must acknowledge the support provided by the helpful system administrators and secretarial staff of the department.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

This thesis presents a study of the asymptotic number of comparisons in Multipartition Quicksort, a class of sorting algorithms generalized from Quicksort. The number of comparisons in this family of Quicksort algorithms is a random variable, whose distribution is induced by that of the input data keys to sort. A formal description of Multipartition Quicksort will come later in Chapter 3. To motivate the reader's interest, I shall begin with some relevant background information on sorting.

Sorting is among the most frequently analyzed topics in the study of computer algorithms. As a means to achieve data reduction, sorting also has an important role in statistical analysis. Many routine descriptive statistics require data to be sorted, as do various non-parametric tests based on ranks. While an in-depth analysis of sorting algorithms may not be appealing to statisticians, a basic understanding of sorting algorithms and their efficiency is helpful towards more efficient implementation of many statistical tasks.

In the study of algorithms (e.g., Sedgewick, 1988), one speaks of sorting a *file* of *records* containing *keys*. Here, it suffices to view a *record* as a collection of information, and a *key*

as that piece of information within a record used for ordering the records. For my analysis, we can simply think of each record as consisting of a numeric value, which serves as the key.

Sorting algorithms can be classified into internal and external sorting. In internal sorting, all the records in a file can be stored in the computer's memory, thus permitting random access. Sorting files from disk or tape constitutes external sorting. The generally sequential nature of data access makes external sorting undesirable, unless the records cannot all fit inside the computer's memory. There are many different internal sorting algorithms available, such as Bubblesort, Heapsort, Insertion Sort, Mergesort, Shellsort, and of course, Quicksort. Each such algorithm may be competitive under some circumstances, and usually no one algorithm uniformly dominates the rest. For instance, Insertion Sort is a rather simple method often used for sorting bridge hands. It works well for sorting a small but not a large number of items. It is also efficient when the records are already in partial order, while Quicksort suffers under such a situation. A simple description of sorting n keys with Insertion Sort is as follows. Starting with $m = 2$, compare the m-th key with the previous $m - 1$ sorted keys, and insert it in the correct position thus far. Increase m by one, and repeat until $m = n$.

Even though there is no single "best" sorting algorithm, Quicksort is "probably more widely used than any other," and the various analyses and refinements of it have made it "the method of choice in a broad variety of practical sorting applications." (Sedgewick, 1988, p115.) For instance, the programming language C features a `qsort()` function, and the `sort` command in Unix is based on Quicksort as well. Invented by Hoare (1962), Quicksort

has been given extensive mathematical and empirical analysis, notably by Sedgewick (1975, 1977). For a comparative study of other sorting algorithms, the reader is referred to Dell (1958), Knuth (1973) or Sedgewick (1988).

Another way to classify sorting algorithms is into comparison versus non-comparison based methods. (See Dell, 1958, for a more general classification.) Comparison-based sorting operates by making pairwise comparisons among keys to determine the correct ordering of keys. The list of algorithms mentioned earlier (Bubblesort, etc) are in this class. Examples of non-comparison based methods are Radix Sort, which uses the base p representation of keys (for some integer p) as its basis, or Distribution Counting, which sorts according to the distribution function of the keys.

I shall not be concerned with non-comparison based sorting methods, since they are not as commonly studied or implemented. The purpose of making this classification, however, is to highlight the fact that for comparison based sorting, the time complexity is often reflected by the number of comparisons required. (There are of course other factors that contribute to the execution time; but their importance varies with the algorithms, and in the case of Quicksort they are less interesting.) Because of this, the number of comparisons is often used as an abstract measure of the time complexity of a sorting algorithm.

This thesis presents an analysis of the asymptotic number of comparisons in Multipartition Quicksort. Here, "asymptotic" refers to the case when the number of records to sort, n, approaches infinity. The goal of doing an asymptotic analysis is twofold. First, from a pragmatic standpoint the complexity of an algorithm becomes interesting (or at least warrants attention) when n is large. Second, an exact analysis in the case of finite but large

n can often be difficult and sometimes intractable, whereas asymptotic analysis can usually produce a much simpler result. One can use asymptotic results to approximate those in the finite case, to get an overall understanding of the complexity problem.

This thesis does not seek to produce an analysis of Multipartition Quicksort *as a sorting algorithm*, á la Sedgewick's (1980) treatise on Quicksort. To do this one must consider all factors affecting the algorithm's performance, and perhaps the optimal implementation of it with regard to some computer architecture. Rather, a stochastic characteristic of the algorithm (namely the number of comparisons) is highlighted for a probabilistic analysis, and its asymptotic distribution estimated. In particular, this random variable (upon a linear transformation) has an asymptotic distribution that satisfies a fixed point equation. This fixed point equation defies an analytic solution, but admits several approximate solutions by numerical methods. Chapter 5 is devoted to these numerical methods of solutions, the essence of which is the method of successive substitution.

The work in this thesis serves to bring together, and generalize, some results in the literature on the theoretical and numerical aspects of the problem on the number of comparisons in Quicksort. This will be elaborated shortly.

1.2 Previous work in the literature

Analyses of sorting algorithms abound in the computer science literature. Knuth (1973) is the classical reference for many sorting algorithms, and Sedgewick (1988) is a more modern and readable reference. Various other authors have contributed to the refinement and analysis of Quicksort, after the original idea by Hoare (1962). (See more discussion in

Chapter 3.) However, Sedgewick's (1975) doctoral thesis can be considered as the most comprehensive treatment of the subject, both from a combinatorial and algorithmic perspective.

The basic idea of Quicksort is rather simple. To sort n keys in an array, one first selects a key in the array, and rearranges the remaining keys into $s = 2$ partitions: one with keys smaller than this key, the other with keys larger than it. Now the initial problem is *divided* into two smaller problems, and one may *conquer* each recursively. This is an example of the divide-and-conquer method of problem solving.

For Quicksort, if the permutation of the data input is unknown, then the number of comparisons becomes a random variable, denoted by $T(n)$. Under the appropriate assumption on the distribution of the data input, the distribution for Quicksort may be found recursively. The typical assumption is that every permutation of the data input is equally likely. *Unless otherwise noted, this will be so assumed when reference is made to previous work in the literature.* Also, in the usual analysis of algorithms, the *best* and *average case* performance of Quicksort, in terms of $T(n)$, are very competitive, while its *worst case* performance is disappointing. See Chapter 2 for details. However, this characterization in itself is not very meaningful without knowing the probability of the worst case scenario. The variance of $T(n)$ has been known since Knuth (1973). Hennequin (1989, 1991) later derived expressions for higher moments of $T(n)$. Tail behavior of $T(n)$, in terms of its deviation from the mean, was reported by Karp (1990), Rösler (1991), and McDiarmid and Hayward (1992). The latter showed, for instance, that $\Pr[\frac{|T(n)-ET(n)|}{ET(n)} > \epsilon] = n^{-2\epsilon[\log\log n + \mathcal{O}(\log\log\log n)]}$. A probability distribution over the number of comparisons will give a better depiction of

Quicksort's efficiency. Because such explicit description for large n remains intractable, we turn to asymptotic results for approximation.

The interest in identifying the asymptotic distribution of $T(n)$ has arisen relatively recently. The existence of such a distribution was established by Régnier (1990) in a non-constructive way, using a martingale argument. A stronger result was obtained by Rösler (1991), who showed that this distribution is the fixed point of a contraction operator. Despite this, no closed form expression for the asymptotic distribution is available. The fixed point property however enabled Eddy and Schervish (1992) to apply various iterative methods to numerically approximate this distribution. Some properties of the limit distribution function and its moment generating function are characterized by Tan and Hadjicostas (1993).

Multipartition Quicksort is an obvious generalization of Quicksort, by splitting the keys to be sorted into s subsets ($s > 2$) during each partitioning stage. The idea was noticed by Sedgewick (1980) but he did not pursue it (see Chapter 3). Dobosiewicz (1978) presented a method of sorting by *distributive partitioning*, which has the flavor of Multipartition Quicksort, but is more of a generalized Distribution Counting method. In particular it does not sort *in place*.

In his PhD thesis, Hennequin (1991) studied the combinatorial properties of various parameters associated with a more general class of Quicksort and search trees. The term "Multipartition Quicksort" apparently is due to him. Hennequin's formulation of the algorithm is actually more general; see Chapter 3. He derived the cumulants of $T(n)$ in Multipartition Quicksort in terms of complicated recurrent relations, made possible by the

use of a symbolic mathematics package to differentiate a multivariate generating function. The asymptotic property of the cumulants enabled Hennequin to conclude that a limiting distribution for $T(n)$ exists.

To my knowledge Multipartition Quicksort has not since been studied in the literature, especially from a practical implementation standpoint. Hennequin himself studied only the mathematical and not algorithmic aspect of it. In this thesis, a contribution is made by having computer programs to implement Multipartition Quicksort, the empirical performance of which is evaluated.

1.3 Outline of thesis

The main goal of this thesis is to present an asymptotic analysis of the number of comparisons in Multipartition Quicksort, based on the tools and theories of Régnier (1990), Rösler (1991), and Eddy and Schervish (1992).

Chapter 2 presents a review of previous work. The implementation of Quicksort is explained, followed by a stochastic model for the number of comparisons to sort n keys, denoted by $T(n)$. Some assumptions in the model are explained and discussed. These assumptions allow the distribution of $T(n)$ to be derived recursively. A class of distributions on the permutation of keys that admits such recursive identification of $T(n)$ is presented. A binary recursion tree structure is used to help in understanding the recurrence property of $T(n)$, in particular of the best and worst case scenarios.

The goal of Chapter 3 is to extend the analysis of Quicksort in the previous chapter to "Multipartition Quicksort". The chapter begins with a discussion of some refinements

and generalizations of the original Quicksort idea. This leads to a particular class of Quicksort variants called *Multipartition Quicksort*, the focus of this thesis. For brevity it will at times be referred to as M-Quicksort. In particular, our discussion focuses on the simplest, four-partition, version of M-Quicksort, characterized by the parameter $s = 4$. The implementation of this algorithm is presented, and a stochastic model for $T(n)$ given. The conceptual difference between this and the Quicksort algorithm is made, using tools such as the binary recursion trees and the distribution of subarray leagths. A class of distributions on permutations that admits recursive identification of $T(n)$ in this setting is presented. The exact upper and lower bounds of $T(n)$ are derived. A regression approach is used to estimate the asymptotic form of $ET(n)$, the mean number of comparisons.

Chapter 4 is concerned with generalizing a convergence theorem for $T(n)$ in Quicksort (Rösler, 1991) under more general probability distributions over the permutations of keys. With some modification, the result extends to the M-Quicksort (s=4) setting, and thus to arbitrary s as well. The asymptotic distribution function is proven to be absolutely continuous and strictly increasing.

In Chapter 5, various numerical methods based on Eddy and Schervish (1992) are used to approximate the asymptotic distribution in M-Quicksort (s=4). The essential tool is *successive substitution*, made possible because the limit distribution is a fixed point of a contraction operator. The computational complexity in the current problem requires some modification of the methods of Eddy and Schervish. In particular, parallel and asynchronous techniques of computation are used. Computational issues are discussed in some detail.

Chapter 6 presents the results of two experiments to understand other related properties

in M-Quicksort, which may provide directions of future research in the practical implementation of M-Quicksort. The first experiment studies the effect of varying s (the parameter in M-Quicksort) on the actual sorting time on a workstation. The second experiment studied the effect of s on the number of sorting comparisons.

Chapter 7 gives a brief analysis of the number of comparisons in another related sorting algorithm, *Mergesort*. The goal is to show how the methodology used in the analysis of Quicksort is applicable here. In this case the asymptotic distribution turns out to be the familiar Normal distribution.

Finally, the contribution of this thesis, in terms of new results and generalization of previous work in the literature, is summarized in Chapter 8. Some issues encountered, but not addressed, in the course of this research are indicated, and may serve as ideas for future research.

In the Appendices, a general program for M-Quicksort is included with an explanation of the algorithm. Also included are the various computer programs used in the numerical computation in Chapter 5. Some description of the C-Linda parallel programming tool is provided to help in understanding the programs that used C-Linda for parallel computing.

1.4 Some hardware details

Chapters 5 and 6 of this thesis involve substantial amount of computational work, in which the time complexity may be of interest. To give the reader a sense of what the same work will require on other computing platforms – and because the computing in this thesis has been done on various processors – the following tables list a number of relevant features

of the computing hardware used for comparative purpose.

The hardware available for these computations is a collection of DEC Unix workstations maintained by the Statistics Department at Carnegie Mellon University. The workstations share a common Network File System, and communicate through an Ethernet network. This enables the distributed computation described in Chapter 5. However, this also makes a workstation subject to multiple demands from different users at a single time. This explains why there was not a single dedicated workstation for all the uniprocessor computation.

At the time of research for this thesis, there are four different kinds of such workstations, the essential difference being highlighted in the following table:

model	processor	speed (MHz)	RAM (Mb)	quantity
3100	MIPS R2000	20	12	15
5000/120	MIPS R3000	20	16	14
5000/200	MIPS R3000	33	24	3
5000/240	MIPS R3000	40	40	1

To give a more concrete idea of the performance of these workstations, the LINPACK benchmark was run on these machine to obtain the approximate MFLOPS (millions of floating point operations per second). The statistics below are actually produced by the software MATLAB (The MathWorks, Inc, Natick, MA), which also gives comparison statistics for other processors.

machines	MFLOPS
80386/80387 (20MHz)	.232
DEC3100	1.542
DEC5000/120	1.690
DEC5000/200	2.705
DEC5000/240	3.822
Cray X-MP	33

Chapter 2

Quicksort

2.1 An algorithm for Quicksort

Quicksort is based on the *divide and conquer* paradigm of problem-solving on a computer. Essentially, the paradigm says that a large problem should be subdivided into smaller parts that are similar to the original problem, and the subdivision continues until the parts are small enough that their solution is simple or even trivial.

Consider the task of sorting n *records* stored in contiguous locations $A[1], \ldots, A[n]$ in an array A of the computer's memory. Recall from Chapter 1 that for my purpose a record consists of only a single number, and so may also be referred to as a *key*. In the simplest version of Quicksort, one out of the n keys is selected as the *partitioning key*. The role of this partitioning key is to partition the remaining keys into two subarrays. One subarray has keys equal to or smaller than the partitioning key, the other has keys larger. The correct position, or rank, of the partitioning key among the n keys is thus determined. The problem is now reduced to separately sorting the two subarrays, which together have one fewer key than previously. The partitioning procedure just used is then applied recursively to each subarray, provided it has at least two keys. When this recursive algorithm terminates, the

```
qsort(int low, int high, int A[])
{  int key;

   if (high <= low)  return;
   key = partition(low, high, A);
   qsort(low,  key-1, A);
   qsort(key+1, high, A);
}

partition (int low, int high, int A[]) {
/*  'low' and 'high' mark the first and last location of
    a subarray to be partitioned. */
   int key;

   key = low;
   low++;
   for (;;)
     { while ((A[key] >= A[low])&&(low < high)) low++ ;
       while ((A[key] < A[high]))                high--;
       if (low < high) {
         swap(&A[high], &A[low]);
         low++ ; high--;
       }
       else break;
     }
   if (A[key] < A[low]) low--;
   swap(&A[key], &A[low]);
   return low;
}

swap(int *x, int *y)
{
  int temp;
  temp = *x;  *x = *y ;  *y = temp;
}
```

Figure 2.1: A recursive Quicksort algorithm in C