INVESTIGATIONS OF SHELLSORT

By

HONG-LEE YU

Bachelor of Science

National Chiao Tung University

Taiwan, R. O. C.

1979

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fullfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1984

Thesis
1984
Y94i
cop. 2

INVESTIGATIONS OF

SHELLSORT

Thesis Approved:

J P Chandler
Thesis Adviser

[illegible]

[illegible]

Norman N. Durham
Dean of the Graduate College

PREFACE

This thesis investigates many aspects of Shellsort. A large number of experiments were conducted and best sequences, which seem to minimize the number of comparisons for different sizes of lists and number of passes, are given. A proof that shows the average behavior of the original Shellsort, when N is a power of 2, is also presented.

I wish to thank my committee members, Dr. D. W. Grace and Dr. S. Thoreson, for their contributions and advice, and Dr. Michael J. Folk for substituting during my oral examination. A special thank goes to my major advisor, Dr. J. P. Chandler, for his help on this thesis and for his assistance throughout my studies at Oklahoma State University.

A final thank is due to my dear wife, Tresie. This thesis would not have been possible without her encouragement and patience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Sorting is worth studying not only because a large percent of the running time on computers is spent on sorting, but also because there are numerous methods to choose from, none of which can dominate all of others [9]. Of those methods Shellsort, proposed by D. L. Shell in 1959 [12], has the virtues of requiring no additional memory space and taking at most O(N**(3/2)) units of time for most suggested variations.

Shell's method consists of several passes, each of which sorts h sublists of the list L(i), L(i+h), L(i+2*h), ..., L(i+j*h) corresponding to i =1, 2, ..., h, where j is the largest integer such that j*h <= N, by insertion sort; h decreases from pass to pass according to a prescribed procedure. In the final pass, h is equal to 1, which ensures that the entire list L is sorted.

Insertion sort moves items only one position at a time and its worst case and average case running times are proportional to N**2, since the average displacement of each element from its final position is N/3 for a random list of N elements. However, in Shell's method, items can take long leaps instead of short steps during early passes, then shorter leaps later. Hence the average running time O(N**2) can be reduced. Each pass of Shell's method is operating not on a random sequence but on one which has been given a degree of order by previous passes. This leads to the question of choice of an optimal sequence h. Though the method was proposed as early as 1959, nobody has been able to determine the best possible sequence h for large values of N.

The objective of this thesis is to investigate Shellsort. Due to the complex mathematical problems in the analysis of Shellsort, which are not yet completely resolved, most of the work will be done empirically. In order to determine what the best sequence is, we need to establish a criterion to measure its efficiency. Any measurement using real time on a given computer has some inherent problems. The running time depends not only on the underlying sequence but also on the characteristics of the computer, and thus should not be used [1]. The number of comparisons or the number of moves are usually employed in analyzing sorting algorithms since both of these measures are often roughly proportional to the running time. For empirical study, the number of comparisons may give better measurement since it consists of the number of in-order comparisons and the number of out-of-order comparisons (number of moves); it dominates the running time. However, the number of moves (inversions) is easier to use when analyzing the asymptotic average behavior mathematically, because there exist many combinatorial theorems related to number of inversions (discussed in chapter 2) that can be applied to analyze the algorithm.

Though there exist many interesting problems arising from Shellsort, this thesis will concentrate on the following problems which have not been solved yet.

1. What are the optimum sequences of increments for Shellsort? The term "optimum sequence" causes certain ambiguity, since we are unable to examine all of the possible sequences; even the number of possible sequences for a moderate N, say N=100, is big enough that we cannot afford examining all of these and there are no proved theorems that allow us to eliminate a large number of sequences from examination without taking a risk of discarding the real optimum one. Some assumptions, most based on empirical results, must be applied without proof to keep the number of examinees of reasonable size. Here "optimum sequence" refers to some empirical values of h(t), h(t-1),..., h(1) which takes minimal average number of comparisons to sort, among all of the sequences we examine.

2. What is the order of the asymptotic average behavior of the Shellsort? One well-known work about average behavior of Shellsort was done by Knuth's students [9]; they claimed that asymptotic average behavior of Hibbard's sequence is about O(N**1.26), though they also found that N*(ln N)**2 also gave a good fit to an observed data. Bauer [2] conducted a similar experiment and concluded that if the

average case of Shellsort has a time complexity that is asymptotically a power of N, then the power must be less than 1.20. Whether the exponential form or N*(ln N)**2 form gives the true asymptotic behavior is still open to research. We do not intend to solve it. Instead, we shall examine the simplest case, the original Shellsort when N is a power of 2, in detail; this helps us to dig out some inherent characteristcs behind the average behavior of Shellsort.

The remainder of the thesis will be concerned with the following. In chapter 2, a short literature review is given. We also present methods of finding the optimal sequence. In chapter 3, we show all of the empirical results and attempt to explain why the sort behaved in this way. Suggestions about determining the best sequence are also given. In chapter 4, we discuss the average behavior of Shellsort. The original Shellsort, when N is a power of 2, will be examined in detail. Chapter 5 presents a more detailed summary and unification of the results of chapter 2 to 4 and suggests problems for further research.

CHAPTER II

LITERATURE REVIEW AND METHOD

In this chapter, we introduce some theorems and, in the first three sections, present a literature review. In the last section we discuss the survey method and the hypothesis.

Terminology and Theorem

Before going further, we need to introduce some terminology and relevant theorems which provide the fundamental background for analyzing Shellsort. Most material in this section can be found in [8, 9].

Let a(1)a(2)...a(n) be a permutation of the set {1, 2,..., n}. If $i < j$ and $a(i) > a(j)$, the pair (a(i), a(j)) is called an inversion of the permutation. For example, the permutation 3 1 4 2 has three inversions: (3,1), (3,2) and (4,2). Each inversion is actually a pair of elements that is out of order and the only permutation with no inversion is the sorted permutation 1 2 ... n.

The inversion vector of a permutation is the sequence of integers

d(1)d(2)....d(n)

obtained by letting d(j) be the number of a(i) such that (a(i),a(j)) is an inversion. In other words, d(j) is the number of elements greater than a(j) and to its left in its sequence, so 0<=d(j)<j. For example, the inversion vector of the permutation 3 1 4 2 is

j	1	2	3	4
d(j)	0	1	0	2.

So the number of inversions can be obtained by summing the inversion vector, which yields 3 in this case.

A p-chain of list L is a sequence of elements of L occurring at intervals of p. For example, if N = 8, then L has three 3-chains, namely, {L(1), L(4), L(7)}, {L(2), L(5), L(8)} and {L(3), L(6)}. In general, L has min(N,p) p-chains, each of length E(N/p), or E(N/p)+1, where E(N/p) is an integral part of N/p.

When each of L's p-chains is in ascending order, L is said to be p-ordered. To p-sort L is to sort all of L's p-chains.

The most fascinating theorem about Shellsort is perhaps the following:

<u>Theorem</u> <u>A</u> : If a k-ordered list is h-sorted, it remains k-ordered.

An example of this remarkable property appears in TABLE I. After being 2-sorted, the list's three 3-chains (7,13,29,44), (5,18,24,63), (8,19,31,82) are still in ascending order; it remains 3-ordered.

TABLE I

AN EXAMPLE OF SHELLSORT

	7	19	24	13	31	8	82	18	44	63	5	29
3-sort	7	5	8	13	18	24	63	19	29	82	31	44
2-sort	7	5	8	13	18	19	29	24	31	44	63	82
1-sort	5	7	8	13	18	19	24	29	31	44	63	82

Theorem B : Suppose that h and k are relatively prime; the largest integer which cannot be represented in the form a*h + b*k, a,b >= 0, is h*k-h-k.

Proof. If n = h*k-h-k, which can be represented in the form a*h+ b*k , then a mod k = k-1, and b mod h = h-1; hence a*h+b*k >= (k-1)*h + (h-1)*k > h*k-h-k. This is a contradiction. Conversely if n >= (h-1)*(k-1), choose a, b so that mod(a*h, k) = mod (n, k) 0<=a<k and b = (n-a*h)/k; hence n is representable.

From Theorem A and Theorem B, we can conclude that if a list L is h-ordered and k-ordered, and gcd(h,k)=1, we have L(i) < L(j) whenever j-i >= (h-1)*(k-1). For the example in Table I where h=3 and k=2, we have

$L(i) < L(j)$ if $j-i > 1$.
This led to the idea of Pratt's sequence, which we shall discuss in the next section.

Families of Increments

Shell originally suggested using the increments $\lfloor N/2 \rfloor$, $\lfloor N/4 \rfloor$, ..., 1, but this has a serious defect when the binary representation of N contains a long string of zeros; there is little interaction among chains, which results in many sorted but distinct chains. For example, if N is a power of 2, the worst permutation before the last pass is processed is N/2, 1, N/2+1, 2, ..., N, which needs about N**2/8 inversions to sort. Frank and Lazarus [6] first recognized this defect and suggested that the even elements in Shell's sequence be incremented by one. Table II, derived from Frank and Lazarus's paper, shows their empirical results. The original Shellsort performs as well as F & L's Shellsort when N is odd (eg. N=12287). But in case the binary representation of N contains a lot of zeros, the use of F & L Shellsort takes a significant less time as compared to the use of the original Shellsort.

Table II

IBM 704 RUNNING TIMES FOR DIFFERENT VALUES OF N
USING SHELL'S SEQUENCE OF INCREMENTS

Number of Elements N	N in Octal	Average Sorting Times in Seconds	
		F & L	Shell
127	177	.15	.16
128	200	.16	.20
1000	1750	2.0	2.2
7936	17400	25	26
12287	27777	44	46
12288	30000	43	119
18432	44000	74	144
24575	57777	103	104
24576	60000	103	338
32544	77440	143	150

Since then, many other sequences [cf. Hibbard 1963] have been suggested. Most of them have following properties:

1. Sequences form fuzzy geometric progressions.
2. Each element of a sequence is relatively prime to at least one of its nearby predecessors.

Papernov and Stasevich [10] proved that an upper bound for Hibbard's sequence is O(N**(3/2)) and later Pratt [11] extended it to sequences which have the above properties. Pratt's own sequence certainly does not fall into the above category and its average asympototic behavior O(N*(Lg N)**2) is known to be the best so far. The idea behind Pratt's sequence is that each h(s)-chain in the sth pass is 2-ordered and 3-ordered. Following the discussion above, for any h(s)-chain, we have

L(i) < L(j) if j-i >= 2*h(s).

For any element L(j) in the h(s)-chain, we need only compare it with L(j-h(s)) and swap two elements if they are out of order. At most N comparisons are needed in each pass and hence the running time is of order N*(Lg N)**2, since there are about (Lg N)**2 passes. The major drawback in using Pratt's sequence is that N has to be extremely large before Pratt's sequence is more efficient than the other popular sequences. This defect was first pointed out by Pratt [11] and was confirmed by Bauer's experiment [2]. Of those suggested sequences, Knuth's sequence and Hibbard's sequence are very similar and have been widely used. Bauer [2] compared these two sequences and found that the use of Knuth's sequence results in approximately two or three percent fewer comparisons than the use of Hibbard's sequence. Table III summarizes the main families and their characteristics.

Table III

Families of Increments

suggested by	h	Upper Bound	Average case

Shell	h(t) = floor(N/2)
	h(k) = floor(h(k-1))
		O(N**2)	unknown

Hibbard	h = 1,3,7,...,2**J-1<N	O(N**1.5)	unknown
Knuth	h = 1,4,...,(3**t-1)/2 where h(t+2) > N	O(N**1.5)	unknown
Pratt	h = 1,2,3,4,6,... h is of the form 2**p*3**q p, q > 0	O(N*(Lg N)**2)	same

How to Code Shellsort

In Shellsort, each pass of each sift consists of successive pair swaps. Boothroyd [3] coded this method in a way that a swap follows each out-of-order comparison while Hibbard [7] replaced each set of n pair swaps by one "save," n-1 moves, and one insertion (see Program A, Figure 1.) Chandler and Harrison [4] reported that Hibbard's algorithm runs 17% faster than Boothroyd's algorithm, on a CDC 6400 computer and coded in FORTRAN.

```
procedure C(x,n); array x(1:n)
comment Shell's method using Hibbard's increments
begin integer d,i,j
  d := 2**entier(lg N)   - 1;
C1: if d<=0 then go to exit; i:=1;
C2: j:=i; y:=x[i+d];
C3: if y<x[j] then go to C4;
C5: x[j+d]:=y; i:= i+1;
    if i+d<=n then go to C2;
    d:=(d-1)/2; go to C1;
C4: x[j+d] := x[j]; j:=j-d;
    if j>0 then go to C3; go to C5;
exit: end;
```

"entier" is an ALGOL function equivalent to "floor"

Figure 1. Program A by T. N. Hibbard

Bauer [2] noted that program A can be tuned by eliminating "unconditional" saves (y:=x[i+d] in C2 in Program A). The saving in time achieved by this modification is about 9% for Hibbard's sequence. His experiment also indicated that the initial increment, h(t), should not exceed about 0.24*N when Hibbard's increments are used and should not exceed about 0.50*N when Knuth's increments are used. Program B, in appendix A, is the FORTRAN implementation based on Bauer's algorithms. Bauer's thesis [2] has a complete treatment about how to code Shellsort.

Methods of Investigation

In this section, we are going to present our methodology used to find the optimal sequence. We might view the problem of finding the optimal sequence as a constrained optimization problem - requiring the minimization of running time (number of comparisons) for a specific N and subject to

$1 = h(1) < h(2) < \ldots < h(t-1) < h(t) < N$,

t, h(i) i=1,t are integers.

Unlike traditional optimization problems, the number of variables is not fixed -- there are t variables (h(1) is not a variable). The first decision we need to make is to choose some values of N on which the experiment will be based. Since there are some existing sorting algorithms, quicksort, for example, which have better performance than Shellsort when N becomes large (say, N=500), it is reasonable to choose N from such range in which Shellsort is better than or at least equal to any existing sorting methods. The interval (20,500) seems to be a good choice since some simple straight sorting algorithms, insertion sort, for example, are better than Shellsort when N becomes small because of low overhead for bookkeeping. We choose 5 points: 20, 50, 100, 250, 500.

For a given N, we can fix the number of passes t (t=1) first and find some empirical values h(1), h(2),..., h(t), which seems to minimize the number of comparisons, in the sense that if one of the h's is varied while the others are fixed, the average number of comparisons increases. Later, we increment t by one and repeat the above search; in this manner, best sequences for different values of t can be obtained.

However, since t can be theoretically as large as N-1, another question immediately arises : when do we stop the above search? Unfortunately, we cannot find any theory to answer it. From Bauer's thesis, we know that it is unwise to have too many passes. Knuth, based on his MIX computer, also estimated that saving one pass is about as desirable as saving 10/9*N moves. Therefore the following convergence criterion seems to be a reasonable assumption.

<u>HYPOTHESIS</u> <u>A</u> : Sequence h (with number of passes equal to t) is said to be the best possible if all of the possible sequences with the number of passes less than or equal to t+1 require more comparisons than h does.

We have already discussed how to choose N and to determine the maximum limit of the number of passes t. For a given N and t, do we have to examine all of the possible sequences (there are about 5000 possible sequences of three-pass Shellsort when N is equal to 100); can we eliminate some sequences, which are not likely to be the best sequence, without loss of generality? Theorem A and Theorem B show us that it is desirable to sort with relatively prime incements. Empirical results, shown in Table IV (see also Fiqure 2), agree with Theorem A.

Table IV

BEHAVIOR OF THE NO. OF COMPARISONS AS A FUNCTION OF h(3) IN THREE-PASS SHELLSORT
h(2)=3, N=50

h(3)	Average Number of Comparisons*
4	340
5	325
6	344
7	302
8	292
9	326
10	292
11	290
12	314
13	287
14	291
15	318

* 20 random lists were sorted

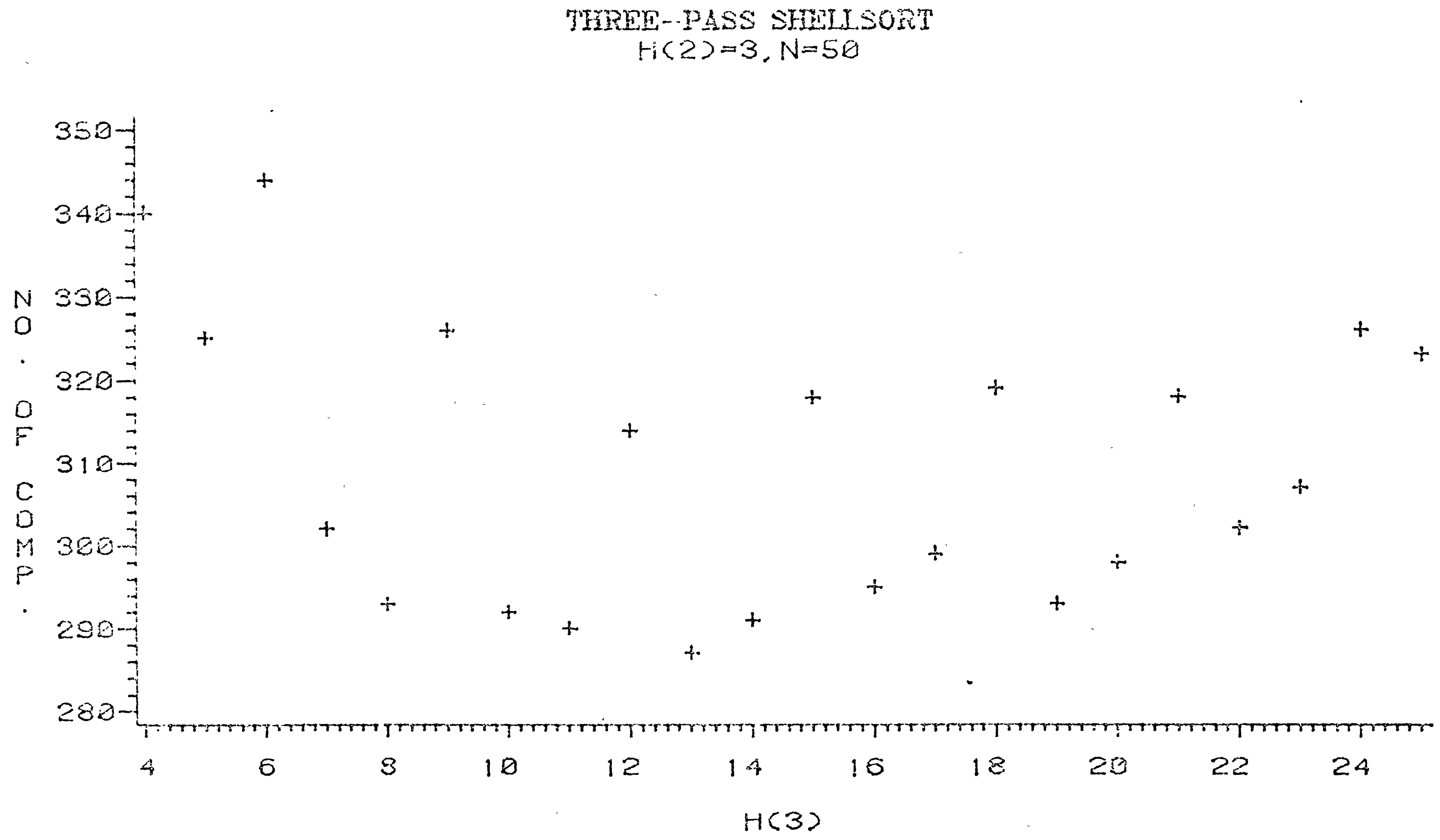


Figure 2. Illustration of Importance of Relative Primeness

It is not too difficult to explain that the number of comparisons increases sharply when h(2) is not relatively prime to h(3). Let us use two specific sequences (1, 3, 11) and (1, 3, 12) (called sequence A and sequence B respectively) as an example. Since the above two sequences differ only in h(3) by one, we can conjecture that the running time of the first two passes for sequence A and sequence B are about equal, assuming that the original list is random; the final 1-sorting determines which sequence is the best. In sequence B, the second pass, 3-sorting, is just a straight insertion sort on 3 (h(3)/3)-ordered chains which results in a random 3-ordered permutation; after the second pass, each element of the inversion vector d(j) can be (j-1)/3*2 in the worst case. But in sequence A, d(j), independent of the value of j, can be at most equal to 10, since

```
  L(i) < L(j) if j-i >= 20              (Theorem A)
  L(j) > L(j-3) > L(j-6) >....> L(j-18) (L is 3-ordered)
```

and

```
  L(j) > L(j-11) > L(j-14) > L(j-17)    (L is 11-ordered);
```

so

```
  L(j-1), L(j-4), L(j-7), L(j-10), L(j-13), L(j-16), L(j-19)
  L(j-2), L(j-5) and L(j-8)
```

are the only possible elements to the left of L(j) which are greater than L(j). So sequence A is significantly better than sequence B in the worst case. What is about the average behavior? The average value of d(j) is obtained by summing probabilities P(L(i) > L(j)) for all i < j.

Although we are unable to figure out what the exact probability distributions in the two cases are, it is reasonable to conjecture that sequence A will outperform sequence B in the average case as well. As shown in Table V, sequence B performs poorly in the final 1-sorting as compared to sequence A and sequence C. Sequence B requires the same number of comparisons in the final pass as sequence D does, although sequence D has one pass less than sequence B.

Table V

COMPARISONS PER PASS WITH N=50

	Increments	Number of Comparisons			
			pass		total
		1	2	3	
A	1,3,11	71	119	100	289
B	1,3,12	64	110	140	314
C	1,3,13	60	120	107	287
D	1,3	236	141	-	377

In order to assure that h's are relatively prime, let

$h(2)$ = any integer > 0,

$h(3)$ = any prime number $> h(2)$,

$h(k)$ = any prime number $> h(k-1)$.

Apparently we may suffer from the possibility of discarding some sequences of which increments are relatively prime and may give the best empirical results, for instance, (1, 3, 10) in Table III. Note that the Figure 2 is almost horizontal when $h(3) >= 7$, $h(3) <= 14$, except for a few values of $h(3)$, which are a multiple of 3. This indicates that such criteria cam be applied without much loss of generality.

Now we come to the problem of the choice of the test data. Throughout this thesis, all of the sorts are run against random lists; here we define a random list L as, for any two elements $L(i)$ and $L(j)$,

$$P(L(i) > L(j)) = P(L(i) < L(j))$$

and

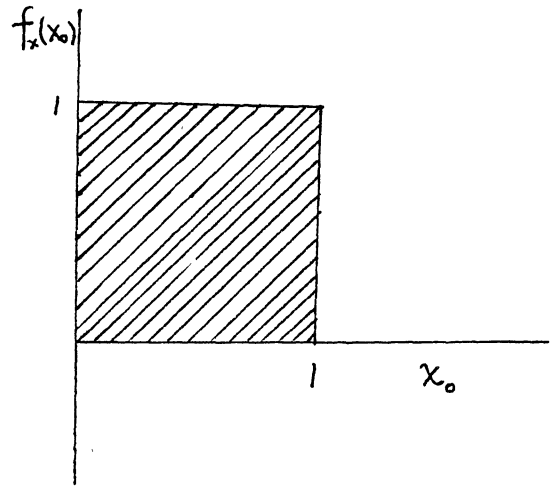
$$P(L(i) = L(j)) = 0.$$

A random number generator, which can repeatedly generate uniformly distributed random numbers over the interval (0,1), satisfies our needs. Here the uniform distribution over the interval (0,1) is defined as

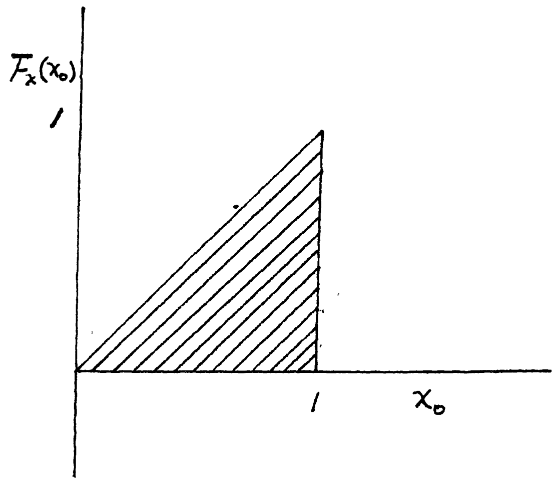
$$f(x(0))=1 \qquad \text{for } 0 < x(0) < 1,$$

$$F(x(0))=x(0) \qquad \text{for } 0 < x(0) < 1,$$

where the probability density function (P. D. F.) $f(x(0))$ = Probability $(x=x(0))$ and cumulative distribution function (C. D. F.) $F(x(0))$ = probability $(x<=x(0))$. Figure 3 shows the probability density fnction and the cumulative distribution function. Our random number generator is based on a shuffled congruential method. To gather more accurate statistical data, the sorts are run against 20 different data lists of 5 sizes 20, 50, 100, 250, 500. All of the empirical results, shown in a later chapter, are the average of 20 different runs.



(A) P. D. F.



(B) C. D. F.

Figure 3. P. D. F. and C. D. F. for Uniform Distribution Over Interval (0,1)

CHAPTER III

TOWARD THE OPTIMAL SEQUENCE

In this chapter, we study the problem of finding the optimal sequence h to minimize the running time. We examine the simplest case, two-pass Shellsort, first in section 1, followed by generalizing to multiple-pass Shellsort in section 2. In section 3, we present a new variation of Shellsort, intended to speed up the inner loop, and empirical results are also given.

Two-Pass Shellsort

In this section, we examine the characteristics of two-pass Shellsort, which consists of a h(2)-sorting, followed by h(1)-sorting, where

$$h(1) = 1,$$
$$h(2) > h(1) \text{ and } h(2) < N.$$

Let us consider first the 2-ordered list. It is easy to see that the number of permutations a(1)a(2)....a(N) of {1 ,2,..., N} such that a(i) <= a(i+2) for 1 <= i <= N-2 is

$$\binom{N}{\lfloor N/2 \rfloor}, \qquad (3.1)$$

since this equals the number of ways partitioning N elements into two groups; $\lfloor N/2 \rfloor$ elements to put in even-numbered positions a(2)a(4)..., with the remaining $\lceil N/2 \rceil$ elements to put in odd-numbered positions. Each 2-ordered permutation is equally likely after a random list has been 2-sorted, since the number of permutations for a 2-ordered list (Eq. 3.1) is a divisor of N!.

Let A(N) be the total number of inversions among all 2-ordered of {1, 2,..., N}. Knuth [9] used a "lattice diagram" to compute A(N), which has the surprisingly simple form

$$\lfloor N/2 \rfloor * 2{*}{*}(N-2) \qquad (3.2)$$

Hence the average number of inversions in a random 2-ordered permutation can be obtained by dividing Eq. 3.2 by Eq. 3.1. By Stirling's approximation this is asymptotically

$$0.15*N{*}{*}(3/2) \qquad (3.4)$$

Thus if a list is 2-ordered, the average running time to sort this list is proportional to N**(3/2). Now consider the general two-pass Shellsort, when the increments are h and 1. In the first pass, we need to sort h random chains by insertion sort, of which r chains are of length q+1 and h-r of length q, where

q is the integral part of N/h

r is the remainder part of N/h.

Since insertion sort takes on average n*(n-1)/4 inversions for a list of length n, the average number of inversions needed in the first pass is :

$$\frac{r}{2}\binom{q+1}{2} + \frac{h-r}{2}\binom{q}{2} \qquad (3.5)$$

Each inversion in the second pass comes from a pair of distinct chains, and a given pair of distinct chains in a random h-ordered permutation constitutes a random 2-ordered permutation. For example, let L be a 3-ordered list of length 6. Inversion can only come from following three random 2-ordered lists, (L(1), L(2), L(4), L(5)), (L(1), L(3), L(4), L(6)) and (L(2), L(3), L(5), L(6)). The average number of inversions is therefore the sum of the average numbers of inversions between each pair of distinct chains, namely

$$\binom{r}{2} \frac{A(2*q+2)}{\binom{2*q+2}{q+1}} + r(h-r) \frac{A(2*q+1)}{\binom{2*q+1}{q}} + \binom{h-r}{2} \frac{A(2*q)}{\binom{2*q}{q}} \qquad (3.6)$$

Equation 3.5 is approximately equal to 2*N**2/h, while equation 3.6 approximately equals $\sqrt{\pi * N**3*h}$. The best choice of h, which can be found by minimizing the summation of the above two approximations, is approximately 1.72*N**(1/3). With this choice of h we can make a substantial improvement over straight insertion, from O(N**2) to O(N**(5/3)).

To get a more clear view, let us look at some plots based on our empirical data. Figure 4 and Figure 5 are plots of number of comparisons (or number of moves) vs. h(2) when N=100. It explores the following interesting facts:

1. If we smooth the curve in Figure 4, it becomes convex with a minimum point h(2)=7. If we view straight insertion sort (one-pass Shellsort) as a two-pass Shellsort with a large value of h(2) (say, h(2)>N), then Figure 4 clearly shows the improvement from insertion sort to two-

pass Shellsort.

2. In Figure 4, we use two different measurements -- number of comparisons and number of moves; both give very similar curves. This indicates, in this case, that either one of these measurements can serve as the criterion to measure its efficiency. However, a defect using number of moves as the criterion , found in our later experiment, is that the use of number of moves can not completely reflect the overhead associated with an increase of one more pass, even though it asymptotically dominates the running time. As t becomes larger, the number of moves will decrease while the number of comparisons increases (that is, the number of in-order-comparisons increases). Accidentally, we find another reason to use the number of comparisons as the criterion.

3. We decompose the number of moves into two parts: the number of moves in the first pass and the number of moves in the second pass as shown in Figure 5. The number of moves in the first pass is proportional to 1/h(2) (Eq. 3.5); its curve is monotonically decreasing as Eq. 3.5 suggests. The other curve, due to the complexity of Eq. 3.6, is not very smooth, but tends to go up as h(2) become larger.

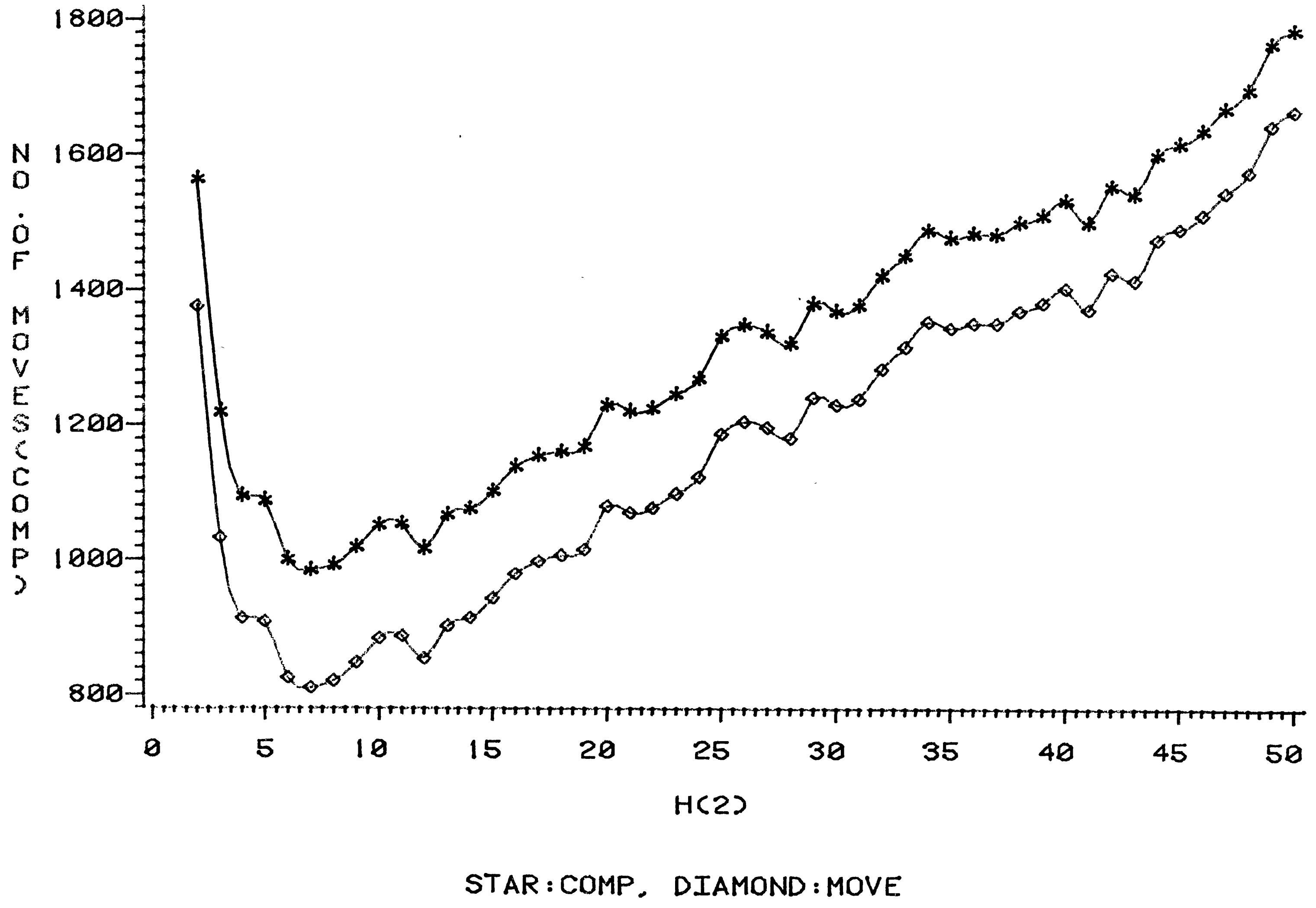


Figure 4. Two-Pass Shellsort (N=100)

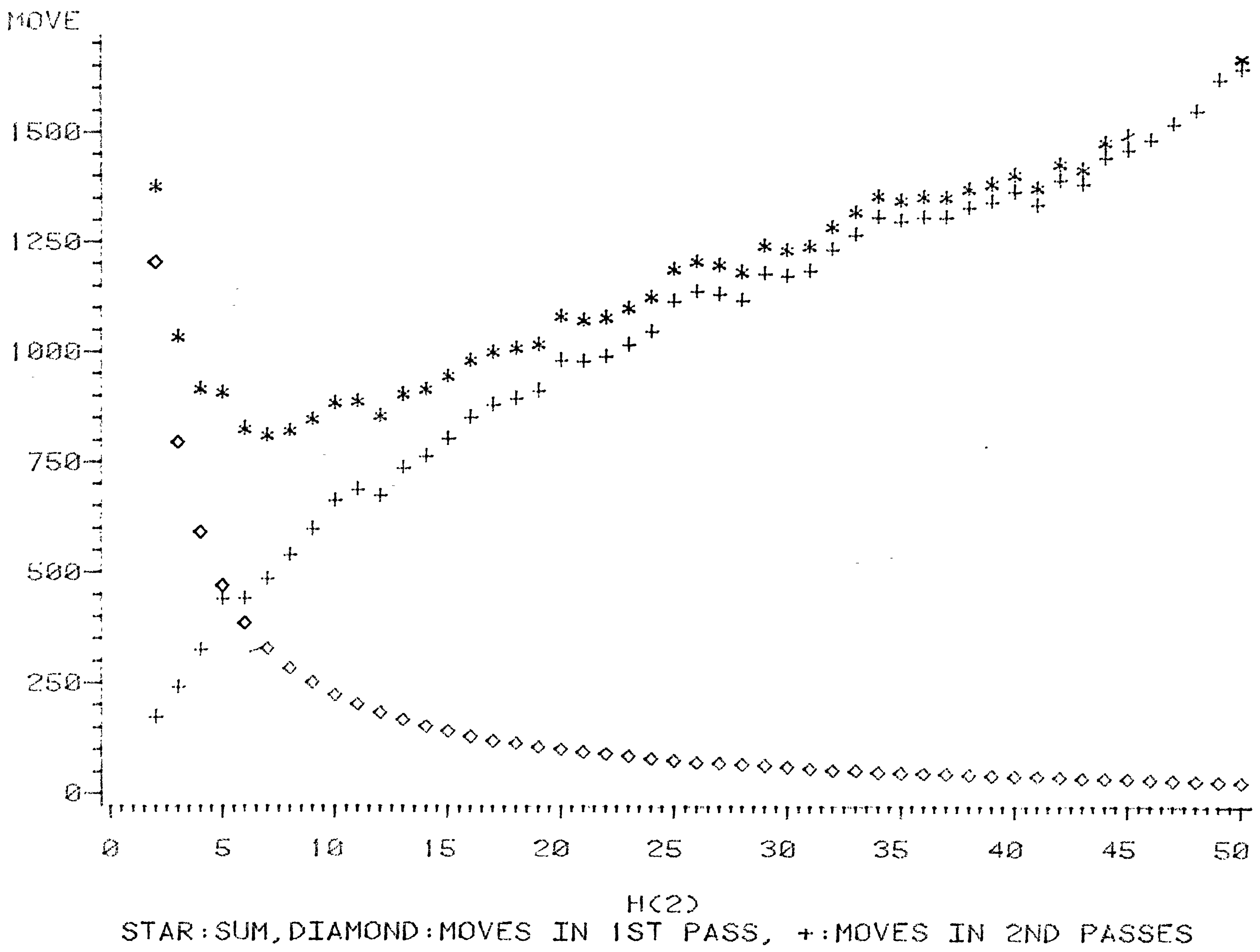


Figure 5. Decomposition of Two-Pass Shellsort (N=100)

The Multiple-Pass Shellsort

We have seen that two-pass Shellsort can break the O(N**2) bound with the right choices of increments. Clearly we can do even better when more increments are used. But how much improvement can we make with more increments? Analysis of general Shellsort, unlike that of two-pass Shellsort, faces a difficult problem that has baffled everyone so far: the permutations in a given pass may not be equally probable, so that all of the combinatorics techniques in analysis of two-pass Shellsort seem useless. As an example, a 3-ordered list of length 4 has 12 permutations which are equally likely. After 2-sorting, some permutations are obtained more often than others as shown in Table IV. Unless we can figure out the probability distribution for a list with given degrees of order, analyzing general Shellsort using mathematics will be very difficult.

So our primary analysis of general Shellsort has been based on statistical results rather than on mathematical derivation. A large number of experiments (more than 50,000 lists were sorted) were conducted based on the procedure described in Chapter II. Appendix B tabulates the best optimal sequences for different values of t and N. It should be noted that there are a lot of sequences which are not listed in Appendix B, but which take almost the same amount of time as the best sequences (the difference is less than one percent) and the difference probably is due to sampling error. Thus we should not regard the sequences in Appendix B as the only best sequences.

TABLE IV

PROBABILITY DISTRIBUTION FOR
A 2- and 3-ORDERED LIST OF LENGTH 4

PERMUTATION	PROBABILITY
1234	3/12
1324	4/12
1243	2/12
2143	1/12
2134	2/12

Perhaps the most fascinating observation is that all of the best sequences form geometric progressions (slightly perturbed), just like most existing Shellsorts. Nobody has commented on why geometric progressions were originally chosen for almost all Shellsorts except Pratt [11]. He suggested thinking of Shellsorting as progressively bringing each element closer to its final position, in jumps of decreasing size, then it is "natural" to arrange that increments decrease geometrically; this is what happens in a binary search. Possibly such consideration has motivated the choice of a fuzzy geometric progression for almost all Shellsorts. Our empirical results show that it is necessary, at least for the range we searched, to make the sequence a geometric progression.

So we focus our attention on sequences that form fuzzy geometric progressions which are parameterized by a common ratio. Our next observation is that Knuth's sequence (common ratio = 3) is almost optimum for the interval we examined. The sequence (1, 4, 13, 41) was observed to be the best possible sequence for N=100 (Knuth's sequence: (1, 4, 13, 40)). Note that Knuth's sequence, like most suggested sequences, requires only every consecutive pair of elements in the sequence are coprime, while in our experiment,

$$\gcd(h(1),h(2),....h(t)) = 1$$

One might wonder whether there is improvement if we perturb each of $h(i)$, $i > 2$, of Knuth's sequence, to a nearest prime number. Figure 6 shows that perturbing $h(4)$ from 40 to 41 (or 37) in this case, can result in a 1.2% reduction in number of comparisons. We extend our experiment for different values of N as shown in Table VII, and results show that it is beneficial to make the elements of Knuth's sequence relatively prime. When the running time is a critical factor, it is worthwhile storing a perturbed Knuth's sequence in an auxiliary array ($O(\text{Lg } N)$ storage).

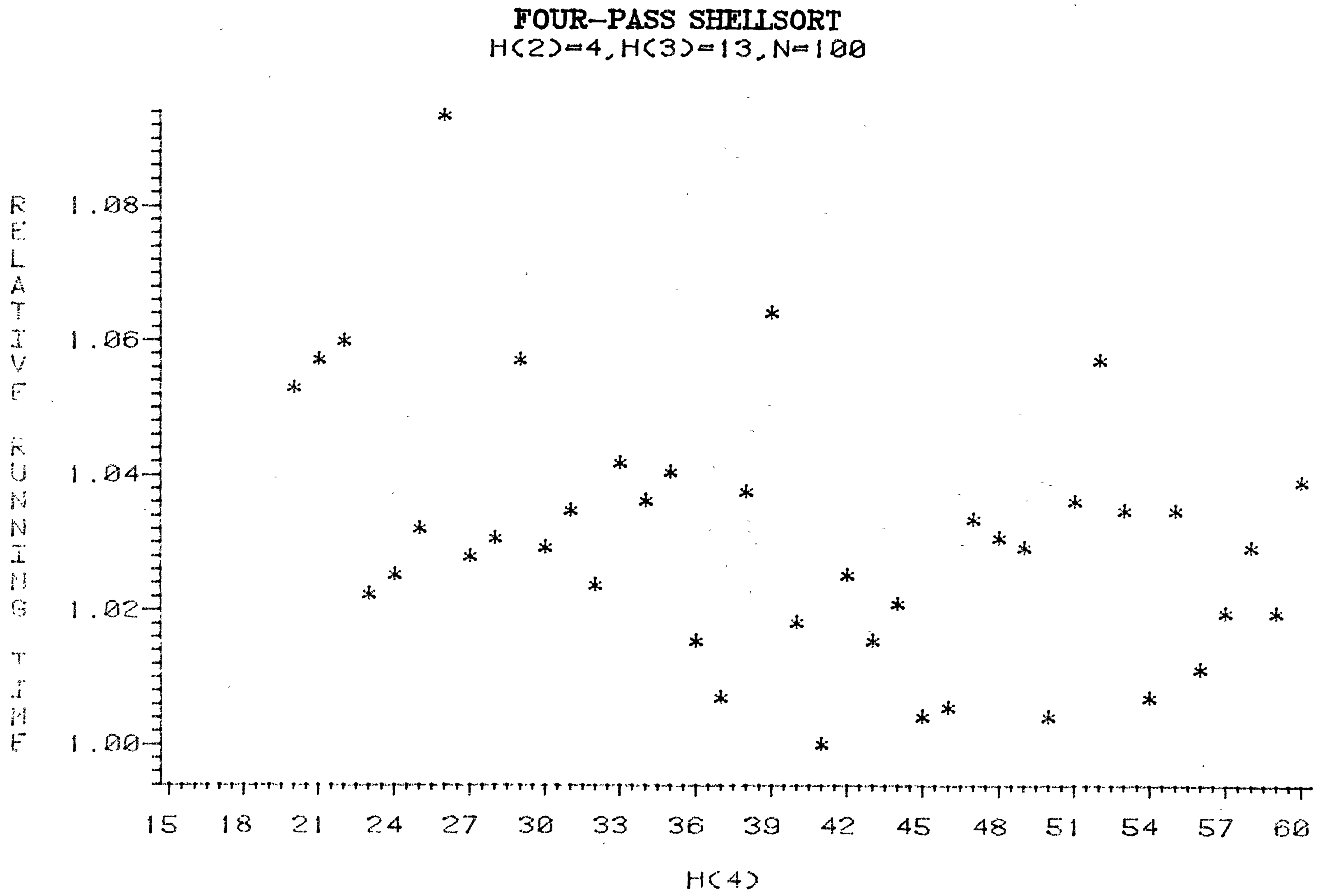


Figure 6. Behavior of Relative Running Time As a Function of H(4)

TABLE VII

COMPARISON BETWEEN KNUTH'S AND PERTURBED KNUTH'S SEQUENCE

N	t	KNUTH		MODIFIED		IMPROVEMENT
		S	COMP	S	COMP	%
100+	3	1,4,13	771	1,4,13	771	0.0
100*	4	1,4,13,40	730	1,4,13,41	717	1.2
250+	4	1,4,13,40	2486	1,4,13,41	2434	2.1
250*	5	1,4,13,40, 121	2442	1,4,13,41, 127	2378	2.6
500+*	5	1,4,13,40, 121	5882	1,4,13,41, 127	5882	1.9
1000+		1,4,13,40, 121	14219	1,4,13,41, 127	13977	1.2
1000*	6	1,4,13,40, 121,364	13788	1,4,13,41, 127,367	13250	4.0

+: Knuth's criterion
*: Bauer's criterion

Table VII also suggests that Bauer's criterion to determine the number of passes t (that is, choose t such that h(t)<=0.5*N and h(t+1) > 0.5*N) be preferable to the Knuth's criterion (that is, choose t such that h(t), h(t+1) < N, and h(t+2) >= N), as far as the number of comparisons is concerned. When Knuth's criterion is used, the increment in the first pass h(t) can be only about N/9 in the worst case, which is apparently too small, since the average displacement of each element from its final position is N/3.

In conclusion, there are actually a lot of sequences which are comparable to the best sequences we listed in Appendix B. All such sequences form fuzzy geometric progressions too. Fuzzy geometric progression with common ratio equal to 3 seems a good choice. Thus Knuth's sequence is recommended, with Bauer's modification which determines the number of pass t.

A New Variation Of Shellsort

Our previous discussion was based on standard Shellsort: in the s-th pass, we h(s)-sort the list by insertion sort and afterword the list is h(s)-ordered. Each pass faces more than O(N) running time due to the characteristics of insertion sort. In this section, we want to present and experiment with a new variation of Shellsort, in which the running time of the first t-1 passes can be guaranteed to be O(N) each, at the expense of the final pass of which the running time is unknown. The algorithm is given as follows:

```
For s=t downto 2 by -1 do
  for j= h(s)+1 to N do
  if ( L(j-h(s)) > L(j) ) then
       swap(L(j-h(s)), L(j))
insertionsort(L, N)
```

The result of interchanges is not propogated; no comparison between L(j-h(s)) and L(j-2*h(s)) is made. Thus, the list will not always be h(s)-sorted. The final insertion sort ensures that the whole list is sorted. In this case, the first t-1 passes take (t-1)*N comparisons and these preliminary passes help to decrease the running time of the final pass. If the running time of the final pass is proportional to O(N* (Lg N)**2) or even better, its average running time will beat any variation of Shellsort (assume t is roughly equal to Lg N.)

The empirical results, shown in table VIII, discourage our optimistic guessing. The number of inversions required in the final pass is not decreased enough. It tends to be slower as compared to traditional Shellsort when N becomes larger. This indicates that an early comparison contributes more than a comparison in the final psss. The true asymptotic behavior is unknown.

TABLE VIII

COMPARISON BETWEEN SHELLSORT AND NEW VARIATION
USING HIBBARD'S SEQUENCE

N	NUMBER OF COMPARISONS	
	SHELLSORT	NEW
20	81	79
50	302	297
100	787	834
250	2567	3567
500	6157	10588
1000	14747	35269

CHAPTER IV

AVERAGE BEHAVIOR OF SHELLSORT

In this chapter, we shall study the most difficult problem about Shellsort: asymptotic average behavior. At the time of this writing, most work on average behavior has been based on data fitting. Here we present a simple analytical model that can be used to analyze some passes of Shellsort. We introduce Chandler's experiment first, and then show some of his comparison tables can be derived by the use of combinatorics. Finally, we prove that the asymptotic behavior of the original Shellsort, when N is a power of 2, is O(N**1.5).

Usually the derivation of such "asymptotic" formulas involves higher mathematics and therefore may be hard or impossible to isolate, so one might think to solve this problem with the help of computer -- sort all permutations of N keys and compute the exact average running time (number of inversions). It is obvious that such a method can only be applied to small values of N since the number of permutations is N! that it takes too much computer time to experiment with moderate values of N (10! = 3628800). Experimenting on some small values of N, however, may give us insight into the average behavior of large values of N. Based on such idea, Chandler [5] conducted an experiment on original Shellsort for N up to 8. As shown in Appendix C, he tabulated number of out-of-order comparisons (horizontally) vs. number of in-order comparisons (vertically) in each pass of Shell's original algorithm. Given such tables, the average number of inversions can be easily computed by summing all p(i)*i, where p(i) is the probability of having i inversions (the number of permutations to have i inversions divided by total possible permutations).

It is astonishing at first sight that a large common factor can be taken out of each of these tables, which afterword are in a fairly simple form. Tables of whose N/h equals 2 even form diagonal matrices (with diagonal line from northeast down to southwest) and coefficients on that diagonal line are binomial coefficients, namely

$$\binom{h}{i} \quad \text{where h is the increment in that pass} \quad i = 0, 1, \ldots, h. \qquad (4.1)$$

For example,

N=4
h=2
CM=6

	0	1	2
0	0	0	1
1	0	2	0
2	1	0	0

N=6
h=3
CM=90

	0	1	2	3
0	0	0	0	1
1	0	0	3	0
2	0	3	0	0
3	1	0	0	0

CM=6 denotes a common factor of 6 that has been removed from that table, etc.

But there is nothing amazing after we viewed it as a combinatorial problem; the common factor is equal to the number of ways to partition N into N/2 subgroups with 2 elements each. For N = 4, common factor is

$$\binom{4}{2}\binom{2}{2} = 6;$$

while N=6, it is equal to

$$\binom{6}{2}\binom{4}{2}\binom{2}{2} = 90.$$

In each subgroup, two elements are either in order (no inversion) or out of order (one inversion). There are therefore

$\binom{N/2}{i}$ permutations to have i inversions,

since there are N/2 subgroups. This explains the above two tables.

Such analysis can be extended to some tables in which h is not a divisor of N. For example, if N=5, h=2, common factor is equal to the number of permutations for a 2-ordered list of length 5, that is,

$$\binom{5}{2} = 10$$

The list is split into two sublists now -- one of length 3 (sublist 1) and the other of length 2 (sublist 2). Let x(0), x(1) and x(2) be the random variables representing the number of inversions for the list, sublist 1 and sublist 2 respectively. Clearly,

P(x(0)=0) = P(x(1)=0)*P(x(2)=0) = (1/6)*(1/2) = 1/12,

P(x(0)=2) = P(x(1)=2)*P(x(2)=0) + P(x(1)=1)*P(x(2)=1)

= (2/6)*(1/2) + (2/6)*(1/2) = 4/12.

Using similar concepts, we can compute all of the first-pass tables by hand. Also, the equations for the two-pass Shellsort (Eq. 3.6 together with Eq. 3.2) provide us with the capability to compute all of the two-pass tables (eg., N=6, h=1).

Note that the above analysis is limited to lists in random order. Later passes of Shellsort, in which each of the possible permutations may not be equally likely, certainly do not have such property. Thus it is applicable only for the first pass.

However, one exception occurs when N is a power of two. In this case, lengths of every sublists during a specific pass are the same and each permutation is equally likely, since the number of permutations in any pass is always a divisor of N!. Recalling from chapter 3, where we have presented the average number of inversions for a 2-ordered list, we are able to show following theorem.

<u>Theorem C</u>: The asymptotic average behavior of original Shell's algorithm, when N is a power of 2, is of order N**1.5.

<u>Proof</u>. We have 2**t elements to be sorted. By using Shell's increments, it takes t passes to sort this list, with increments equal to 2**(t-1), 2**(t-2),..., 1. In each pass s=t, t-1,...,1, there are 2**(t-1) sublists; each sublist is of same length 2**(t-s+1) and 2-ordered. From Eq. 3.4, for a list of length k, we have

no. of inversions $\simeq$ 0.15*k**1.5

The total average number of inversions can be obtained by summing the average number of inversions in each pass.

$$\text{Total} \simeq \sum_{i=1}^{t} *0.15*(2**i)**1.5 * 2**(t-i).$$

$$= \sum_{i=1}^{t} *0.15*2**(t+0.5*i).$$

$$= 0.15*N* \sum_{i=1}^{t} *2**(0.5*i)$$

$$\simeq 0.51*N*(\sqrt{N} - 1)$$

We have just showed that the running time of a special case of original Shellsort, when N is a power of 2, is proportional to O(N**(3/2)). But can we extend it to the general Shellsorts? Does the exponential form give the true asymptotic behavior of Shellsort? This question has baffled everyone so far. We are unable to answer it. The main contribution of this chapter, if any, is to illustrate the importance of mathematics, especially combinatorics and probability, in analysis of algorithms. The analytical model we built may not be very useful for analysis of general Shellsort.

CHAPTER V

SUMMARY, CONCLUSION, AND SUGGESTIONS FOR FUTURE RESEARCH

Summary and Conclusions

For each chapter, we summarize its results and give our conclusions.

In chapter II, we reviewed the previous work and presented our methodology used to find the optimal sequences. We also illustrated why the primeness of increments is necessary by showing empirical data and by giving theoretical analysis.

In Chapter III, we discussed the characteristics of two-pass Shellsort, in which the optimum sequence can be derived by mathematics. We also pointed out why the analysis of algorithms in multiple-pass Shellsort is difficult -- the permutations in a given pass may not be equally probable. In addition, we presented our empirical results based on the procedure described in chapter III. In order to provide an easier view, some plots are given. Empirical results suggested that Shell's increments be in the form of geometric progression and the corresponding common ratio be 3. All of the existing sequences perform about equally well; Knuth's sequence is slightly better than the other sequences. However, Knuth's criterion to determine the number of passes t is not so great and Bauer's criterion is recommended.

In chapter IV, we introduced Chandler's comparison tables first and showed that some tables can be derived by hand rather by expensive experiment. We also noted that when N is a power of 2, the permutaions in any pass are equally likely, if Shell's increments are implemented. This helps us to prove that the average behavior of original Shellsort, when N is a power of 2, is of order O(N**(3/2)). The exact form of the asymptotic average behavior of other varities of Shellsorts is still unknown.

We hope that this thesis can provide a tutorial view of Shellsort, especially for people who are not mathematically inclined. We illustrated many characteristics of Shellsort by examples and plots, and also tried to explain it, using simple combinatorics and probability. Those with little mathematical background should find no difficulty with it.

Suggestions for Future Work

Study of Shellsort seems to be of academic interests only, since there exist sorting techniques which are better than Shellsort in most cases. But due to its simplicity and the good performance in the worst case, Shellsort can be applied for moderately large N (say, N<=500). Also, it provides an excellent case study in the analysis of an algorithm. There are many suggestions for future research on the Shellsort:

1. If a sequence S is found to be the best possible for N, will S be the optimum for any N1, where N1 is not equal to N; that is, does there exist an optimal sequence for any N (with diffent t).

2. We have briefly introduced Pratt's sequence in chapter II. It can be generalized to sequences of the form x**p*y**q. What is the average time for Shellsort using sequences of the form 2**p*3**q, etc.?

A SELECTED BIBLIOGRAPHY

[1] Aho, A. V., J. E. Hopcroft, J. D. Ullman. Data Structures and Algorithms, Reading MA: Addison-Wesley Publishing Co., 1983

[2] Bauer, L. B. "An Empirical Study of Shellsort." Unpublished M. S. Thesis, Oklahoma State University, 1980.

[3] Boothroyd, J. "Shellsort: Algorithm 201." Communications of the ACM, 6 (1963), 445.

[4] Chandler, J. P., W. C. Harrison. "Remark on Algorithm 201." Communications of the ACM, 6 (1970), 373-374.

[5] Chandler, J. P. Unpublished Notes, Oklahoma State University, 1981.

[6] Frank, R. M.,R. B. Lazarus. "A High-Speed Sorting Procedure." Communications of the ACM, 3 (1960), 20-22.

[7] Hibbard, Thomas N. "An Empirical Study of the Minimal Storage Sorting." Communications of the ACM, 3 (1960), 206-213.

[8] Knuth, D. E. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 2nd ed. Reading, MA: Addison-Wesley Publishing Co., 1973.

[9] Knuth, D. E. The Art of Computer Programming, Vol. 3: Searching and Sorting, 2nd ed. Reading MA: Addison-Wesley Publishing Co., 1973.

[10] Papernov, A. A., G. V. Stasevich. "A Method of Information Sorting in Computer Memories." Problems of Information Transmission, 3 (1965), 63-75.

[11] Pratt, V. R. Shell Sorting and Sorting Networks, Garland, New York, 1979.

[12] Shell, D. L. "A High-Speed Sorting Procedure." Communications of the ACM, 2 (1959), 30-32

APPENDIX A

BAUER'S SHELLSORT

```
      SUBROUTINE SHELL(JR,N)
      DIMENSION ARRAY(1000)
C
C
C
C        SHELLSORT
C        INPUT PARAMETERS
C         ARRAY  ARRAY TO BE SORTED
C         N      NUMBER OF ELEMENTS IN ARRAY
C
C        RETURNS ARRAY IN ASCENDING ORDER
C
C        HIBBARD'S INCREMENTS ARE IMPLEMENTED BUT MAY
C        BE MODIFIED TO USE KNUTH7S INCREMENTS
C
C
      IF ( N .LT. 4) RETURN
      JH=1
C        REPLACE .24 BY .50 IF KNUTH'S
C        INCREMENTS ARE USED
      JHMAX = .24*FLOAT(N)
 10   IF (JH .GE. JHMAX) GO TO 20
C
C        REPLACE 2*JH BY (3*JH)-1 IF KNUTH'S
C        INCREMENTS ARE USED
      JH = (JH-1)/2
      GO TO 10
C        REPLACE (JH-1)/2 BY (JH-1)/3 IF KNUTH'S
C        INCREMENTS ARE USED
 20   JH=(JH-1)/2
 30   NMJH=N-JH
      DO 60 K=1,NMJH
         JHPK=JH+K
         IF (ARRAY(JHPK) .GE. ARRAY(K)) GO TO 60
         TEMP=ARRAY(JHPK)
         ARRAY(JHPK)=ARRAY(K)
         J=K-JH
         IF (J .LE. 0) GO TO 50
 40      IF (TEMP .GE. ARRAY(J)) GO TO 50
         JHPJ=JH+J
         ARRAY(JHPJ)=ARRAY(J)
         J=J-JH
         IF (J.GT.0) GO TO 40
 50      JHPJ=JH+J
         ARRAY(JHPJ)=TEMP
 60    CONTINUE
C        REPLACE JH/2 BY JH/3 IF KNUTH'S
C        INCREMENTS ARE USED
```

```
      JH =JH/2
      IF(JH .GE. 1) GO TO 30
      RETURN
      END
```

APPENDIX B

BEST SEQUENCES

N	t	Sequence	No. of Comparisons
20	1	1	114
20	2	1,6	77
20	3	1,4,11	75
50	1	1	660
50	2	1,7	330
50	3	1,5,13	287
50	4	1,4,13,41	285
100	1	1	2526
100	2	1,8	988
100	3	1,4,19	754
100	4	1,4,13,41	717
100	5	1,3,11,31,59	729
250	1	1	15812
250	2	1,14	4449
250	3	1,6,19	2839
250	4	1,4,11,47	2442
250	5	1,4,13,41,131	2370
250	6	1,3,7,19,37,157	2371
500	1	1	63238
500	2	1,12	13645
500	3	1,6,31	7521
500	4	1,5,13,53	6144
500	5	1,4,13,37,101	5685
500	6	1,4,13,23,61,197	5537

APPENDIX C

CHANDLER'S COMPARISON TABLES

Number of out-of-order Comparisons (horizontally) vs. number of in-order comparisons (vertically) in each pass of Shell's original algorithm, for all permutations of N distinct keys. 12* denotes a common factor of 12 that has been removed from that table, etc.

N=2
1*

	0	1
0	0	1
1	1	0

N=3
1*

	0	1	2	3
0	0	0	0	1
1	0	1	2	0
2	1	1	0	0

N=4
h=2
6*

	0	1	2
0	0	0	1
1	0	2	0
2	1	0	0

h=1
4*

	0	1	2	3
2	0	1	1	1
3	1	2	0	0

N=5
h=2
10*

	0	1	2	3	4
0	0	0	0	0	1
1	0	0	1	3	0
2	0	2	3	0	0
3	1	1	0	0	0

h=1
12*

	0	1	2	3
3	0	1	2	1
4	1	3	1	1

N=6
h=3
90*

	0	1	2	3
0	0	0	0	1
1	0	0	3	0
2	0	3	0	0
3	1	0	0	0

N=6
h=1
6*

	0	1	2	3	4	5	6	7	8	9
3	0	0	0	1	2	3	3	3	2	1
4	0	1	5	10	10	9	6	4	0	0
5	1	4	9	9	4	3	0	0	0	0

N=7
h=3
210*

	0	1	2	3	4	5
0	0	0	0	0	0	1
1	0	0	0	1	4	0
2	0	0	3	6	0	0
3	0	3	4	0	0	0
4	1	1	0	0	0	0

N=7
h=1
24*

	0	1	2	3	4	5	6	7	8	9
4	0	0	0	1	3	6	6	4	3	1
5	0	1	6	16	23	21	16	9	3	1
6	1	5	14	20	16	15	9	7	2	1

N=8
h=4
2520*

	0	1	2	3	4
0	0	0	0	0	1
1	0	0	0	4	0
2	0	0	6	0	0
3	0	4	0	0	1
4	1	0	0	0	0

N=8
h=2
1120*

	0	1	2	3	4	5	6
4	0	0	1	2	3	2	1
5	0	2	6	6	4	0	0
6	1	4	4	0	0	0	0

N=8
h=1
576*

	0	1	2	3	4	5	6	7	8	9	10
6	0	1	5	7	7	4	4	3	2	1	1
7	1	6	10	8	6	2	2	0	0	0	0

VITA

Hong-Lee Yu

Candidate for the Degree of

Master of Science

Thesis: INVESTIGATIONS OF SHELLSORT

Major Field: Computing and Infomation Science

Biographical:

Personal Data: Born in Taiwan, R.O.C., January 11, 1958, The son of Yao-Nei and Jung-li Yu

Education: Graduated from Kaoshiung High School, Taiwan, R.O.C., in May, 1975; received Bachelor of Science degree in Management Science from Chiao Tung University, Taiwan, R.O.C., in May, 1979; completed requirements for the Master of Science degree at Oklahoma State University in December, 1984.

Professional Experience: Planner at PEBEI; Kaoshiung, Taiwan, R.O.C., from Jan., 1982 to August, 1982. Graduate Teaching Assistant, Department of Computing and Information Science, Oklahoma State University, Stillwater, Oklahoma, August, 1983 to July, 1984; member of the Association for Computing Machinery.