LOWER BOUNDS FOR SHELLSORT

Mark Allen Weiss

CS-TR-098-87

June 1987

Lower Bounds for Shellsort

Mark Allen Weiss

A DISSERTATION

PRESENTED TO THE

FACULTY OF PRINCETON UNIVERSITY

IN CANDIDACY FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE BY THE

DEPARTMENT OF COMPUTER SCIENCE

-October 1987-

ABSTRACT

Shellsort is a simple classic algorithm that runs competitively on both mid-sized and nearly sorted files. It uses an increment sequence, the choice of which can drastically affect the algorithm's running time. Since we would like an optimal sorting algorithm and a trivial lower bound for Shellsort is N * (# of increments), we require that the size of the increment sequence is $O(\log N)$, where N is the size of the file to be sorted. These increment sequences also tend to perform the best in practice, because of this lower bound.

For some time, the complexity of Shellsort was thought to be well understood, but Sedgewick was able to provide an increment sequence that lowered the upper bound, and then Incerpi and Sedgewick extended the result to give an even better upper bound. In both papers, the authors left as open the problem of whether their bounds were tight.

We prove that Sedgewick's bound is tight by analyzing the time required to sort a particularly bad permutation. Extending this proof technique seems to lead to a lower bound that matches the upper bound of Incerpi and Sedgewick for not only their increment sequence, but also for a wide class of other increment sequences.

Additionally, we show that the permutations which make Shellsort run slowly are counterexamples to a conjecture that shaker-sort, a network sorting algorithm proposed by Incerpi and Sedgewick, runs in $O(N\log N)$ time, again for a large class of increment sequences, including all those that have thus far been proposed.

ACKNOWLEDGEMENTS

Professor Robert Sedgewick joined the Princeton faculty as Chairman at the start of my third year. In spite of the enormous burden that goes along with such a position, especially a new one, he was always able to take time out of his schedule to advise me. My profuse thanks go to Professor Sedgewick.

Professors Bernard Chazelle and Richard Lipton read drafts of this thesis. Their valuable comments are greatly appreciated.

Sharon Rodgers and Winnie Waring helped me get this thesis together and through the local bureaucracy; their efforts undoubtedly saved me months.

Finally, I'd like to thank my parents, Mona and David Weiss, for their support throughout my entire academic career.

TABLE OF CONTENTS

1. Introduction

Shellsort is a sorting algorithm proposed by Donald Shell [She59] in 1959. It is a simple to code, in place sorting algorithm which runs well empirically for both nearly sorted files and mid-sized files; it is the method of choice for these applications.

Shellsort is an improvement over insertion sort, and in fact uses insertion sort as a subroutine. Insertion sort consists of examining elements from left to right, and for each element, finding its place in previously examined elements. This consists of comparing the element to elements on its left, and exchanging with larger elements until a smaller one is found. In the worst case an element may have to be exchanged with all the elements on its left, and so insertion sort has an $O(N^2)$ worst case running time. In fact, analysis reveals an $O(N^2)$ average case running time, so insertion sort is a rather bad general purpose sorting algorithm. However, for nearly sorted files, with only a few elements out of place, insertion sort performs very well.

Shellsort uses a sequence of integers $h_t, h_{t-1}, \cdots, h_1$ and works by performing insertion sort on subfiles consisting of elements h_i apart. We call this an h_i-sort. Thus, 1-sorting amounts to insertion sort, while when a file is 2-sorted, all elements in even spaces are sorted and all elements in odd spaces are sorted. (There is generally no relation implied between an arbitrary even positioned element and an arbitrary odd-positioned element.) Shellsort works by performing passes consisting of an h_t-sort , h_{t-1}-sort , and so on until an $h_1 = 1$-sort. It is both necessary and sufficient that $h_1 = 1$ for the algorithm to sort. One point of the algorithm is that instead of comparing only adjacent elements which means that each exchange can do only a little to sort a file, Shellsort can exchange elements far apart, giving potential to reduce the number of exchanges required. Another idea of the algorithm is that each pass is made easier by the work of previous passes, hence in the latter passes, the subfiles are rather sorted. Thus insertion sort is used for the subfiles. Any sort could be used, but it makes sense to use a simple algorithm that works well on nearly sorted files. Historically, insertion sort has been the subroutine used by Shellsort, and is the method we will use here.

Shellsort is important because of its potential as a network sorting algorithm. In this version of sorting, as described in [Knu73], each element is placed on a line and a network is defined consisting of comparators that compare (and exchange if necessary) two elements, passing them on to the next comparator. The rules of this game are that one has to declare where the comparators are before the sort. If each pass of Shellsort were to have a linear worst case running time, and only $O(\log N)$ passes were used, this would immediately translate into a simple $O(N\log N)$ sorting network. The existence of such a network was a major open problem that has recently been answered affirmatively [AKS83], although no practical algorithm is known.

Recent papers have lowered the worst case time bound for Shellsort considerably after over a decade of stagnation, so it is natural to ask how much lower these bounds can get. In the next section, we elaborate briefly on recent progress made in the analysis of Shellsort.

1.1. Previous Increments and Bounds

Improvements in Shellsort analysis have been made by clever choices of increment sequences. As we shall see, the choice of increment sequences can drastically alter the running time of Shellsort.

Originally, Shell [She59] proposed using increments of the form $\lfloor N/2 \rfloor, \lfloor N/4 \rfloor, \lfloor N/8 \rfloor, \cdots, 1$. Unfortunately, this sequence fails horribly, especially when N is a power of 2. For instance, if $N=32$, the increments are 16, 8, 4, 2, 1 and there are never any exchanges between elements in even and odd positions until the 1-sort. In the worst case where all the large elements are in even positions and all the small elements in odd positions, the final insertion sort would require quadratic time. The practical problem with this increment sequence is that the increments duplicate work previously done and hence don't sort enough in each pass. Similarly, if the increments are of the form c^i for a constant c, Shellsort has $\Theta(N^2)$ worst case running time, although $O(N^{3/2})$ average case running time [Knu73]. Thus it makes sense to try to have the increments relatively prime to each other. Hibbard [Hib63] suggested using 2^i-1 and it was shown by Papernov and Stasevich [PaS65] that Shellsort has an $O(N^{3/2})$ worst case time for these increments.

Pratt [Pra71] showed that Hibbard's sequence was actually $\Theta(N^{3/2})$ by constructing permutations that were hard for Shellsort to deal with. In his thesis, Pratt generalized this result to show that "almost geometric" sequences of the form α^i+c (i.e. within a constant of geometric) ran in $\Theta(N^{3/2})$. Pratt also gave increments of the form 2^i3^j which make Shellsort run in $\Theta(N\log^2 N)$. Unfortunately, while this is still the best worst-case asymptotic bound known, there are $O(\log^2 N)$ increments as opposed to $O(\log N)$ for all the other sequences, so that these increments perform poorly in practice because of its $\Omega(N\log^2 N)$ lower bound. As it also does not lead to an $O(N\log N)$ sorting network, we choose to focus on $O(\log N)$ increment sequences. This restriction along with Pratt's rather general bound led researchers in the 1970s to abandon worst case analyses. Average case analyses are even more difficult and only empirical results were obtained, which estimated Shellsort's average running time for Pratt-like sequences at about $O(N^{1.27})$ to $O(N^{1.30})$. On the other hand, some results suggest that it is possible (but not likely) that the form is actually $O(N\log^2 N)$.

Sedgewick [Sed86] in 1984 was able to obtain an $O(N^{4/3})$ worst case bound for Shellsort by using increments of the form $4 \cdot 4^j + 3 \cdot 2^j + 1$, which violate Pratt's condition, since these increments differ by more than a constant from being geometric. Sedgewick immediately extended this result to $O(N^{1+\varepsilon})$, which would seem to be as good as one can get without going to $O(N\log N)$, but Incerpi and Sedgewick [InS83] devised a sequence leading to an $O(N^{1+\varepsilon/\sqrt{\log N}})$ worst case bound. These sequences are particularly messy, but they are still geometric and also perform well in practice. Chazelle matched Incerpi and Sedgewick's $O(N^{1+\varepsilon/\sqrt{\log N}})$ with increments of the form $\alpha^i(\alpha+1)^j$. When $\alpha = 2$, we have Pratt's sequence. Chazelle was able to trade in going down in increments (from $O(\log^2 N)$ to $O(\log N)$) for an asymptotic increase in running time (from $O(N\log^2 N)$ to the above time). In none of these recent papers were the authors able to determine how tight their bounds were.

In this thesis, we will prove that the bounds given for some of the above increment sequences are tight by constructing permutations that require the time indicated in the upper bound. For the rest of the sequences, we give very strong evidence that these bounds are also tight. We also show that under very general conditions on the increment sequences, no $O(\log N)$ increment sequence can do better in the worst case than those thus far discovered.

1.2. Organization

This thesis is divided into six chapters. Chapter 2 reviews the previous results on Shellsort, and the related Frobenius problem, from additive number theory. Chapter 3 introduces the concept of inversions and the "Inversion Conjecture". This conjecture is proven for several special cases and evidence to support the claim for the general case is given. Chapter 4 deals with the lower bounds themselves. In it we generate bad permutations for Shellsort based on the increment sequence used. Chapter 5 discusses a variant of Shellsort known as shaker sort, a candidate for an $O(N\log N)$ sorting algorithm that was proposed by Incerpi [Inc85]. We essentially show that the existence of an $O(N\log N)$ shaker sort is dependent on the existence of an $O(N\log N)$ Shellsort, and is hence unlikely. Thesis results and open problems are presented in Chapter 6.

2. Upper Bounds For Shellsort

In this chapter we introduce the century-old Frobenius problem, and present the known results. We show its relation to Shellsort and review the upper bounds obtained for various increment sequences.

2.1. The Frobenius Problem

2.1.1. Definitions

Suppose that a country wishes to issue only k different types of stamps. What is the largest postage that can't be exactly placed on an (infinite sized) envelope and how many postages can't be exactly placed? This is known as the Frobenius Problem, apparently because the mathematician Frobenius mentioned it often in his lectures [Bra42]. A more formal definition follows:

Definition: $g(a_1,a_2, \cdots ,a_k) \equiv$ the largest integer which cannot be represented as a linear combination, with non-negative integer coefficients of $a_1,a_2, \cdots ,a_k$.

Definition: $n(a_1,a_2, \cdots ,a_k) \equiv$ the number of positive integers which cannot be represented as a linear combination, with non-negative integer coefficients of $a_1,a_2, \cdots ,a_k$.

As an example, consider $g(4,7)$ and $n(4,7)$. If we mark the integers 1 to 18 as either 1 (representable) or 0 (unrepresentable), we get the pattern below:

0	0	0	1	0	0	1	1	0	0	1	1	0	1	1	1	1	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

We notice that once 4 consecutive integers are representable, all are, since if N is representable then clearly $N+4$ is too. Thus we see that $g(4,7)$ is 13 and, since we can count 8 unrepresentable numbers, $n(4,7)=8$. We call the above pattern the Frobenius Pattern; we will see later how we can use it to lower bound Shellsort's running time.

We make several simple observations. First, we may assume that $a_1<a_2< \cdots <a_k$ without loss of generality. Throughout the rest of this paper, we shall make this assumption. Now if $a_1=1$, then $g()=n()=0$ as all integers are representable. Also, we may assume that each a_i is independent of the other arguments (that is it cannot be represented as a linear combination of the other arguments) since otherwise it could be removed without affecting the result.

Finally, $g(a_1,a_2, \cdots ,a_k)$ is defined if and only if $gcd(a_1,a_2, \cdots ,a_k)=1$. To see this, note first that if $a_1,a_2, \cdots ,a_k$ had a common divisor $d \neq 1$, then all integers not divisible by d would be unrepresentable. Conversely, if $a_1,a_2, \cdots ,a_k$ are relatively prime, then there exists a solution (allowing negative $c_i's$) to $\Sigma c_i a_i=1$, which implies a solution with non-negative integer coefficients to $\Sigma c_i a_i \equiv 1 (mod\ a_1)$. This implies that for each residue class of a_1 , we can find a linear combination of $a_1,a_2, \cdots ,a_k$, and thus all integers can eventually be represented.

Algorithmically, to compute g one needs to find if g is defined and if it is, merely find the point where a_1 consecutive integers are representable. Somewhat more efficient algorithms exist, for instance [Gre80], but they are of little concern here. In some cases one can use direct formulas, but as we shall see in the next section there are few of these.

2.1.2. Previous Results

In this section, we examine what is known about the Frobenius problem. The solution for two arguments has been known for over a century and is due to W.J. Curran Sharp [Sh884]:

$$g(a_1,a_2)=(a_1-1)(a_2-1)-1 \qquad (2.1a)$$
$$n(a_1,a_2)=\tfrac{1}{2}(a_1-1)(a_2-1)$$

provided, of course, that a_1 and a_2 are relatively prime.

The case for three arguments remained unsolved for quite some time, but in the interim Johnson [Joh60] provided the useful result

$$g(a_1,a_2,\cdots,a_k)=d\cdot g(\frac{a_1}{d},\frac{a_2}{d},\cdots,\frac{a_{k-1}}{d},a_k)+(d-1)\cdot a_k \qquad (2.1b)$$

where

$$d=gcd(a_1,a_2,\cdots,a_{k-1}).$$

Several results in this period deal with special cases such as arithmetic or "almost" arithmetic sequences, but are seemingly useless for Shellsort analysis.

Finally, Selmer [Sel77] provided the complicated formula:

$$g(a_1,a_2,a_3)\le\max[(s-1)a_2+(q-1)a_3,(r-1)a_2+qa_3]-a_1 \qquad (2.1c)$$

where

$$a_3\equiv sa_2 \; mod \; a_1, \qquad 1<s<a_1$$
$$a_1=qs+r$$

providing that the arguments are pairwise relatively prime and independent. Where there is not pairwise relative primeness, Johnson's formula can be used. Temkin [Tem83] provided an exact closed form solution for the case of three arguments, but it is rather complex and involves many subcases that depend on the relationship between the arguments.

Incerpi and Sedgewick [InS83] provided a nice lower bound for the Frobenius function:

$$g(a_1,a_2,\cdots,a_k)\ge\Omega(a_1^{\,1+\frac{1}{k-1}}), \qquad (2.1d)$$

provided of course that $a_1 < a_2 < \cdots < a_k$.

One other item of interest is the formula

$$\tfrac{1}{2} < \frac{n(a_1,a_2,\cdots,a_k)}{g(a_1,a_2,\cdots,a_k)} \le 1$$

due to Nijenhuis and Wilf [NiW72]. That

$$n(a_1,a_2,\cdots,a_k) \ge \tfrac{1}{2}(g(a_1,a_2,\cdots,a_k)+1)$$

is clear from observing that if x and y are positive integers with $x+y=g$, at most one of x and y can be representable. It is trivially obvious that

$$n(a_1,a_2,\cdots,a_k) \le g(a_1,a_2,\cdots,a_k);$$

what is not so clear is whether this bound is attainable. The reader may verify that

$$g(a,a+1,a+2,\cdots,2a-1) = n(a,a+1,a+2,\cdots,2a-1) = a-1$$

It should be pointed out however that this is a degenerate case and in all instances where $k \ll a$, $\frac{n}{g}$ tends to be very close to $\tfrac{1}{2}$.

2.2. Lemmas Used to Upper-Bound Shellsort

We need two important lemmas before we show how upper bounds for Shellsort are obtained. The first is the fundamental result for Shellsort:

Lemma 1: If a k-sorted file is h-sorted, it remains k-sorted.

The lemma above dates back to at least Boerner [Boe55] (before Shellsort!) and proofs can be found in [Pra71], [Knu73] or [Inc85]. The implication of Lemma 1 is that when we come to h_k-sort a file, we know that it is already h_{k+1}-sorted, h_{k+2}-sorted , ... , h_t-sorted. Thus when we do insertion sort on a subfile and come to a particular element x_i, we know that there are many elements guaranteed to be smaller than x_i, and so the insertion sort shouldn't take as long as its worst case bound would indicate. Lemma 2 tells us how many elements can be larger than x_i, and hence involved in the insertion sort:

Lemma 2: If a file is h-sorted and k-sorted, then for each i' , $x_{i'-i} \leq x_{i'}$ whenever i can be expressed as a linear combination with non-negative coefficients of h and k.

Proof: If $i = sh + tk$, then $x_{i'} \geq x_{i'-k} \geq \cdots \geq x_{i'-sh}$ since the file is h-sorted, and $x_{i'-sh} \geq x_{i'-sh-k} \geq \cdots x_{i'-sh-tk} = x_{i'-i}$ since the file is k-sorted. $\square$

Lemma 2 can easily be extended to take into account more passes, so that if a file is $h_{k+1}, h_{k+2}, \cdots, h_t$-sorted, then for any element x_i, the insertion sort needs to consider only elements that are less than $g(h_{k+1}, h_{k+2}, \cdots, h_t)$ away. This follows since all elements farther away are representable as a linear combination with non-negative coefficients of $h_{k+1}, h_{k+2}, \cdots, h_t$ and are thus guaranteed to be smaller than x_i.

Thus with Lemma 2 we have a way to bound the running time of Shellsort. For each h_k-sort, we bound the running time and then sum over all passes. The time to h_k-sort can be bounded in two ways:

(A) Since we are running h_k insertion sorts of size $\frac{N}{h_k}$, an obvious bound is $O\left(h_k \cdot \left(\frac{N}{h_k}\right)^2\right) = O\left(\frac{N^2}{h_k}\right)$. For small h_k, this is a poor bound, but for large h_k it is a good bound.

(B) By using Lemma 2, we bound how many exchanges each element can be involved in. For an element x, we have that only the other elements within $g(h_{k+1}, h_{k+2}, \cdots, h_t)$ can be involved in exchanges. Of these, only

$\frac{g(h_{k+1}, h_{k+2}, \cdots, h_t)}{h_k}$ can be involved in exchanges in the h_k-sort, since only every $h_k th$ element is examined. Applying this bound to each of the N elements leads to a bound of

$$O\left[\frac{N}{h_k}\left[h_{k+1}, h_{k+2}, \cdots, h_t\right]\right]$$

For increments h_k such that

$$g(h_{k+1}, h_{k+2}, \cdots, h_t) < O(N) \tag{2.2a}$$

bound (B) is the smaller bound of the two. Hence by summing over all h_k , using bound (B) when equation (2.2a) is satisfied and bound (A) otherwise, the running time for Shellsort is bounded.

2.3. Upper Bounds For Shellsort

In this section we apply the lemmas of the previous section to obtain upper bounds for Shellsort.

2.3.1. An $O(N^{3/2})$ Upper Bound

The first non-trivial upper bound for Shellsort is due to Papernov and Stasevich [PaS65]:

Theorem: The running time for Shellsort is $O(N^{3/2})$ for the increments 1, 3, 7, 15, ... , $2^k - 1$, ...

Proof: For $h_k > O(N^{1/2})$, we use bound (A) of the previous section; for smaller h_k, we use bound (B), taking into account only two previous passes. Now, the reader may verify that $gcd(h_{k+1}, h_{k+2}) \equiv 1$, so that

$$g(h_{k+1}, h_{k+2}) = O(h_k^{\,2}).$$

Hence, bound (B) implies a bound of $O(Nh_k)$ for small h_k. Thus the running time

T may be bounded by

$$T \leq \sum_{h_k \leq O(\sqrt{N})} N h_k + \sum_{O(\sqrt{N}) < h_k \leq O(N)} \frac{N^2}{h_k}$$

Since $2^t = O(N)$, (i.e. there are t increments less than N)

$$T \leq \ N \sum_{1 \leq j \leq \frac{t}{2}} 2^j - 1 \quad + \quad N^2 \sum_{\frac{t}{2} < j \leq t} \frac{1}{2^j - 1}$$

which implies that $T \leq O(N 2^{t/2}) = O(N^{3/2})$. □

It should be noted that the h_k-sort for $h_k = O(N^{1/2})$ takes $O(N^{3/2})$ time. Also, since the sums that we have in these proofs are geometric, the sums are the same order of magnitude as the largest term.

To get a better upper bound, one might try to take into account more previous passes. In the proof above, only two passes were used to establish the upper bound. Of course, Pratt's results imply that for this particular sequence, taking all passes into account wouldn't matter since this bound is already as low as possible. In fact, we have that for $h_k = 2^k - 1$,

$$g(h_k, h_{k+1,} h_{k+2}, \cdots, h_\infty) = (h_k)(h_k + 1) - 1 = O(h_k{}^2).$$

In the proof above, we switch sums when $g(h_{k+1}, h_{k+2}, \cdots, h_t) = O(N)$, or $h_k = N^{1/2}$ which leads to an $O(N^{3/2})$ bound. If we had $g(h_{k+1}, h_{k+2}, \cdots, h_t) = O(h_k{}^c)$, we could switch sums when $g(h_{k+1}, h_{k+2}, \cdots, h_t) = O(N^{1/c})$, which would lead to an $O(N^{2-1/c})$ upper bound. For instance, an increment sequence with $g(h_{k+1}, h_{k+2}, \cdots, h_t) = O(h_k{}^{3/2})$ would have an upper bound of $O(N^{4/3})$. This is exactly what Sedgewick did for his result.

2.3.2. An $O(N^{4/3})$ Upper Bound

Theorem: The running time of Shellsort is $O(N^{4/3})$ for the increments 1, 8, 23, 77, 281, ... , $4{\cdot}4^k+3{\cdot}2^k+1$.

Proof: Use the technique described in the previous section. All that needs to be estimated is $g(h_{k+1}, h_{k+2}, \cdots, h_t)$, and we will use $g(h_{k+1}, h_{k+2}, h_{k+3})$ to do this. Sedgewick showed that for all k, $h_{k+1}, h_{k+2}, h_{k+3}$ are pairwise relatively prime and for $k>1$, they are independent also. Using Selmer's formula with $s=4{\cdot}2^{k+1}+7$, $q=2^{k+1}-1$, and $r=8$, Sedgewick showed that

$$g(4{\cdot}4^{k+1}+3{\cdot}2^{k+1}+1,\quad 4{\cdot}4^{k+2}+3{\cdot}2^{k+2}+1,\quad 4{\cdot}4^{k+3}+3{\cdot}2^{k+3}+1)=O(8^k),$$

or $O({h_k}^{3/2})$ (specifically $\approx 16{h_k}^{3/2}$).

Thus for increments smaller than $O(N^{2/3})$, we use bound (B); otherwise we use bound (A). We have that $\dfrac{g(h_{k+1}, h_{k+2}, h_{k+3})}{h_k} = O({h_k}^{1/2})$, so

$$T \le \sum_{h_k < O(N^{2/3})} N{h_k}^{1/2} + \sum_{O(N^{2/3}) \le h_k \le O(N)} \frac{N^2}{h_k}$$

Again, if $2^t = O(N)$,

$$T \le N \sum_{1 \le j < \frac{2t}{3}} (4^{j+1}+3{\cdot}2^j+1)^{1/2} \quad + \sum_{\frac{2t}{3} \le j \le t} \frac{N^2}{4^{j+1}+3{\cdot}2^j+1}$$
$$= O(N{\cdot}2^{t/3}) = O(N^{4/3}).\square$$

Sedgewick gave a second increment sequence: $h_k = (2^k-3)(2^{k+1}-3)$ that also ran in $O(N^{4/3})$. In this case, he was able to solve the Frobenius problem by using Johnson's reduction formula for removing common divisors. The natural way to extend this would be to take $h_k = (2^k-3)(2^{k+1}-3)(2^{k+2}-3)$ which leads to an $O(N^{5/4})$ bound. Extending this further to keep on lowering the bound doesn't quite work for this particular case because h_k will always be divisible by 5, and thus the Frobenius number is undefined. Incerpi and Sedgewick used a sequence with a similar idea to lower the bound further.

2.3.3. An $O(N^{1+1/(c+1)})$ Upper Bound

Incerpi and Sedgewick were able to lower Shellsort's running time to $O(N^{1+1/(c+1)})$ using $O(\log N)$ increments. They invented a "Generalized Frobenius Function", which can be thought of as an extension of Johnson's formula. Their increment sequence is actually a merge of c sequences each of size $O(\frac{\log N}{c})$. Further details can be found in [Inc85] and the sequences themselves can be found in the appendix, however we note that (hindsight being 20-20) their bound is attainable using only $O(\frac{\log N}{c})$ increments, by using just one of their sequences and extending Sedgewick's result. As an example of how to do this (for $c=3$), suppose $h_k=(a_k)(a_{k+1})(a_{k+2})$. Now if four consecutive increments are always relatively prime and independent, we can use Johnson's formula. In this case, we need to compute

$$g(h_{k+1},h_{k+2},h_{k+3},h_{k+4})=g(a_{k+1}a_{k+2}a_{k+3},\ a_{k+2}a_{k+3}a_{k+4},\ a_{k+3}a_{k+4}a_{k+5},\ a_{k+4}a_{k+5}a_{k+6}).$$

Using Johnson's formula with $d=a_4$, we have

$$\begin{aligned} g &= a_{k+4}g(a_{k+2}a_{k+3},\ a_{k+3}a_{k+5},\ a_{k+5}a_{k+6},\ a_{k+1}a_{k+2}a_{k+3})+(a_{k+4}-1)(a_{k+1}a_{k+2}a_{k+3}) \\ &= a_{k+4}g(a_{k+2}a_{k+3},\ a_{k+3}a_{k+5},\ a_{k+5}a_{k+6})+a_{k+1}a_{k+2}a_{k+3}(a_{k+4}-1). \end{aligned}$$

Here we have removed $a_{k+1}a_{k+2}a_{k+3}$ since it is redundant. Applying Johnson's formula again, with $d=a_{k+5}$,

$$g=a_{k+4}(a_{k+5}\ (g(a_{k+3},\ a_{k+6},\ a_{k+2}a_{k+3})+(a_{k+5}-1)a_{k+2}a_{k+3}))+a_{k+1}a_{k+2}a_{k+3}(a_{k+4}-1)$$
$$g=a_{k+4}(a_{k+5}\ (g(a_{k+3},\ a_{k+6})+(a_{k+5}-1)a_{k+2}a_{k+3}))+a_{k+1}a_{k+2}a_{k+3}(a_{k+4}-1)$$

We know that $g(a_{k+3},a_{k+6})$ is about $a_{k+3}a_{k+6}$, so (tossing away insignificant terms), we have

$$\begin{aligned} g &= O(a_{k+1}a_{k+2}a_{k+3}a_{k+4}+a_{k+2}a_{k+3}a_{k+4}a_{k+5}+a_{k+3}a_{k+4}a_{k+5}a_{k+6}) \\ &= O(a_{k+3}a_{k+4}a_{k+5}a_{k+6}) \end{aligned}$$

Now, $a_{k+1},a_{k+2},a_{k+3},a_{k+4},a_{k+5},a_{k+6}$ are all $O(a_k)$, $h_{k+1}=O(a_k{}^3)$ and $g=O(a_k{}^4)$, so

$$g(h_{k+1}, h_{k+2}, h_{k+3}, h_{k+4}) = O(h_k^{4/3})$$

Following through with the proof as before yields an upper bound of $O(N^{5/4})$.

One choice for a_k is $a_k = 2^k - 3$, as in Sedgewick's sequence with $c = 2$. Another choice (the choice in Incerpi and Sedgewick) is to take $a_k =$ *the smallest prime* $> 2^k$. Thus relative primeness is always guaranteed. A third choice might be to take $a_k =$ *the smallest prime* $> 2^{k/3}$. In this case the increments would be $O(2^i)$; in the first two cases the increments would be $O(8^i)$. To prove the asymptotic bound for the first increment sequence, one needs to show relative primeness for four consecutive increments. To prove the bound for the third sequence, one would have to show that you can always find a prime. It may turn out that the sequences perform well practically, even without a proof.

This result can be extended to make h_k the product of c terms, with an asymptotic running time improved to $O(N^{1+1/(c+1)})$. This is an easy proof by induction, using exactly the same techniques as above. Unfortunately, the big-Oh notation hides a constant that is exponential in c. This is because for any c, we have $g() = O(a_{k+c} a_{k+c+1} \cdots a_{k+2c})$. We have said that $a_{k+c} = O(a_k)$, which is true up to a constant. However, in this case we are multiplying c terms together, hence the exponential constant is implied.

Selmer [Sel87] has recently obtained the same result using a theorem due to Brauer [Bra42]. He has also shown that for $3 < c < 10$, one can use $a_k = 2^k - 45$ in the base sequences, but for larger c, 45 must be replaced by progressively larger numbers.

2.3.4. $O(N^{1+\varepsilon/\sqrt{\log N}})$ Upper Bounds

The bound of 2.3.3 is the best one can do if only a constant number of previous passes is taken into account. This is because of the lower bound (2.1d) for the Frobenius function.

The way to bypass this is to construct increment sequences that let you take into account more than a predetermined number of passes. Incerpi and Sedgewick were able to construct sequences that allowed $O(\sqrt{\log N})$ previous passes to be considered. The sequences are rather difficult to explain; see [Inc85] for details; these sequences appear in the appendix.

Chazelle achieved the same upper bound by generalizing Pratt's result. Pratt showed that if the increments were of the form $2^i 3^j$, then each pass would require linear work, so Shellsort would run in $O(N\log^2 N)$ time, as there are $O(\log^2 N)$ increments. Chazelle extended this to increments of the form $\alpha^i(\alpha+1)^j$, where each pass requires $O(N\alpha^2)$ work. There are $O({\log_\alpha}^2 N)$ increments. As we need $O(\log N)$ increments, we choose α appropriately, and calculate the upper bound. Chazelle's arguments also use results for the Frobenius problem; more detail can be found (as usual) in [InS83].

We also note that there is a third, easy to construct sequence that yields the same lower bound. We know that for any c, there is a sequence that runs in time $O(N^{1+1/(c+1)})$, but that this bound is in reality exponential in c. If we rewrite the upper bound to reflect this fact, then it is of the form $O(\alpha^c N^{1+1/(c+1)})$. Now, if N is known, what value of c minimizes this bound? It is easy to solve for c, and one obtains $c = O(\sqrt{\log_\alpha N})$. Thus, for any N, we choose this value of c and can obtain (as verified by substituting c into the original bound) the same lower bound as Chazelle and Incerpi and Sedgewick.

2.4. Summary

Shellsort upper bounds have dropped significantly very quickly. Almost all the proofs follow the same line of reasoning and involve taking advantage of large common divisors when solving the Frobenius problem. This is similar to what Pratt did to get an $O(N\log^2 N)$ sort, but is counter-intuitive in some sense because the original increments used for Shellsort did poorly since they had common divisors. Some of these sequences lead to others that outperform the commonly used sequences in practice. For instance, Sedgewick's bound leads to the increments 1, 5, 19, 41, 109, ... which is the merge of $h_k = 4^k - 3 \cdot 2^k + 1$ and $h_k = 9 \cdot 4^k - 9 \cdot 2^k + 1$. The major open question remaining is whether increments exist that make Shellsort $O(N\log N)$. We shall address this question in the next two chapters.

3. Inversions and The Inversion Conjecture

In Chapter 2, we showed how upper bounds for Shellsort were obtained for various increment sequences. Pratt showed that for certain increments, the worst-case upper bound obtained was actually achievable. It is natural to ask if the new upper bounds are also tight. To do this, we need to analyze the Frobenius pattern (sec 2.1.1) further. Thus in this chapter, we digress from Shellsort analysis per se, and introduce the concept of inversions. We compute the number of inversions in the Frobenius pattern for several sequences. In Chapter 4, we will see how this relates to lower bounding Shellsort.

3.1. Inversions

First, we define what an inversion is:

Definition: Given a file of integers represented by $x_1, x_2, \cdots, x_N$, an inversion is any pair (x_i, x_j) such that $i<j$ and $x_i > x_j$.

As an example, the file 3, 5, 1, 4, 2 has 6 inversions, namely (3,1), (3,2), (5,1), (5,4), (5,2), and (4,2).

In the worst case, the file is in reverse order and there are $\binom{N}{2} = O(N^2)$ inversions. Only a sorted file has no inversions, and on average a file has $N^2/4$ inversions. Note that exchanging two adjacent elements that are out of place removes exactly one inversion so that insertion sort runs in time proportional to the number of inversions. This is why it has quadratic worst and average case running time, but good running time for nearly sorted files (with few inversions). Computationally, inversions can be counted in $O(N \log N)$ time. In the special case that we will deal with where x_i can take on only two values (0 or 1), it is easy to count inversions in linear time.

3.2. Inversions In The Frobenius Pattern

To lower bound Shellsort, it is necessary to count the number of inversions in the Frobenius pattern of the increments used in the algorithm. We call this number $I(a_1, a_2, \cdots, a_k)$. We analyze this for several special cases to try and understand what happens in the general case.

3.2.1. Two Elements (k=2)

This is the simplest case to analyze. The general technique is to write the pattern with a_1 elements to a line. For instance, for I(5, 7), we have

$$\begin{array}{ccccc}
0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 1
\end{array}$$

The next line in the pattern would be all ones. We see that once there is a 1 in a column c_k, it remains forever. Moreover, a 1 enters a column c_k only when it is a multiple of a_2 (in this case 7). In general then, for the first $\left\lfloor \frac{a_2}{a_1} \right\rfloor$ groups, there is only 1 one per line. For the next $\left\lfloor \frac{2a_2}{a_1} \right\rfloor - \left\lfloor \frac{a_2}{a_1} \right\rfloor$, there are 2 ones per line. We can keep this pattern going up until we have that for $\left\lfloor \frac{(k-1)a_2}{a_1} \right\rfloor - \left\lfloor \frac{(k-2)a_2}{a_1} \right\rfloor$ there are $k-1$ ones per line. Finally, we get a line with all ones, but this line extends beyond the Frobenius pattern. For each set of lines with p ones per group we can count $p \cdot$ { # of groups with p ones } $\cdot$ { number of zeros in all groups with $\{p+1, p+2, \cdots, a_1 - 1\}$ ones }. If we sum this over $p = \{1, 2, ..., a_1 - 1\}$, we get a lower bound on the number of inversions. In fact, the only inversions this count misses are those within a particular group, but these are not significant so the lower bound below is asymptotically tight. Thus

$$I \geq \sum_{i=1}^{a_1-1} \left\{ i\left(\frac{a_2}{a_1}\right) \sum_{j=i}^{a_1-1} (a_i - j)\left(\frac{a_2}{a_1}\right) \right\}$$

$$\geq \left[\frac{a_2}{a_1}\right]^2 \sum_{i=1}^{a_1-1} \left\{ i \sum_{j=i}^{a_1-1} (a_i - j) \right\}$$

$$\geq \left[\frac{a_2}{a_1}\right]^2 \sum_{i=1}^{a_1-1} \left\{ i \sum_{j'=1}^{a_1-i} j' \right\}$$

Using $\sum_{i=1}^{n} i^k = \Theta(\frac{n^{k+1}}{k+1})$ throughout, we have

$$\geq \left[\frac{a_2}{a_1}\right]^2 \sum_{i=1}^{a_1-1} i \left\{ \tfrac{1}{2}(a_1 - i)^2 \right\}$$

$$\geq \left[\frac{a_2}{a_1}\right]^2 \sum_{i=1}^{a_1-1} \left\{ \tfrac{1}{2}a_1{}^2 i - a_1 i^2 + \tfrac{1}{2} i^3 \right\}$$

$$\geq \left[\frac{a_2}{a_1}\right]^2 \left[\frac{a_1^4}{4} - \frac{a_1^4}{3} + \frac{a_1^4}{8}\right]$$

$$\geq \left[\frac{a_2}{a_1}\right]^2 \left[\frac{a_1^4}{24}\right]$$

$$\geq \frac{(a_1 a_2)^2}{24}$$

As g $= O(a_1 a_2)$, we have $I = \Omega(g^2)$.

3.2.2. Arithmetic Sequences

This increment sequence has little to do with any practical Shellsort sequence, but we investigate it because of the degenerate case that can arise from it.

We know that

$$g(a, a+1) = O(a^2)$$

and can show that

$$g(a, a+1, a+2) = O(a^2)$$

and in fact can seem to extend this on a computer for an arbitrary amount of terms. But we also know that

$$g(a,a+1,a+2,\cdots,2a-1)=O(a)$$

is the sort of degenerate case that makes general proofs difficult, since $I(a,a+1,a+2,\cdots,2a-1)=0$. We seek to determine

$$g(a,a+1,\ldots,a+(k-1)),\text{and } I(a,a+1,\ldots,a+(k-1))$$

so that we can perhaps gain some insight into why this sequence degenerates.

To get the values of $g()$ and $I()$, we again look at the Frobenius pattern in groups of a. In what follows, we fix k, and let a get large.

We have all 0s up to $a-1$. Numbers from a to $a+(k-1)$ are representable, and then all numbers up to $2a-1$ are unrepresentable. Numbers from $2a$ to $2(a+(k-1))$ are representable, and then numbers until $3a-1$ are unrepresentable, etc. Continuing, we have that numbers from $\frac{a}{k-1}a$ to $\frac{a}{k-1}(a+(k-1))=\frac{a^2}{k-1}+a$ are representable. This is a consecutive numbers, so we now can stop. This tells us that

$$g(a,a+1,\cdots,a+(k-1))=O(\frac{a^2}{k-1}).$$

Notice that for $k=2$, this gives the correct answer. Also, for the degenerate case, this gives the correct answer that $g=O(a)$.

To compute I, we take, for each set of ones, the product of the # of ones in the set and the number of zeros that follow them. In this calculations, as in the others of this section, the big-Oh notation is implied, and insignificant terms are liberally tossed away. Thus,

$$I=\sum_{i=1}^{\left\lfloor\frac{a}{k-1}\right\rfloor}\left\{i(k-1)\sum_{j=i}^{\left\lfloor\frac{a}{k-1}\right\rfloor}a-(k-1)j\right\}$$

since the number of zeros between the group of ones that start at pa is $\approx a-(p-1)k$. Thus,

$$I=\sum_{i=1}^{\left\lfloor \frac{a}{k-1} \right\rfloor} i(k-1)\left[\left(\left\lfloor \frac{a}{k-1} \right\rfloor - i\right)a-\frac{k-1}{2}\left\lfloor \frac{a}{k-1}\right\rfloor^2+\frac{(k-1)}{2}i^2\right]$$

Taking dominating terms only,

$$I=(k-1)\sum_{i=1}^{\left\lfloor \frac{a}{k-1} \right\rfloor}\left[a^2 i+ai^2-\frac{a^2 i}{2(k-1)}+\frac{k-1}{2}i^3\right]$$

These sums are standard to do, and the result is

$$I=\frac{a^4}{24(k-1)^2}.$$

This implies that $I=\frac{g^2}{24}$.

In the degenerate case there are no sums to take so these asymptotic arguments don't work. In all other cases, the result for the number of inversions is independent of k.

3.2.3. Hibbard's Sequence

In this section, for $a_k=2^k-1$, we will compute $I(a_k,a_{k+1},\cdots,a_\infty)$. We need really only compute $I(a_k,a_{k+1},\cdots,a_{2k-1})$, since it is known that $g(a_k,a_{k+1},\cdots,a_{2k-1}) = a_k(a_k+1)-1>a_{2k}$.

To do this calculation, we once again write h_k elements to a line and estimate the number of 1s per line. Below is the pattern for $k=4$ ($a_k=15$).

```
-00000000000000
100000000000000
110000000000000
110000000000000
111100000000000
111100000000000
111110000000000
111110000000000
111111110000000
111111110000000
111111111000000
111111111000000
111111111110000
111111111110000
111111111111000
111111111111000
```

We place a - in position 0 because the pattern is easier to see with multiples of h_k on the left, instead of the right.

While writing down the exact number of 1s in line k is somewhat difficult due to the slightly irregular distribution, it is clear (and easily proved) that the number of 1s in line k is approximately k. In fact, since we are using big-Oh notation, we may just take the number to be k. Thus the number of ones in line k is k and there are $a_k - k$ zeros. There are about a_k lines, so

$$\begin{aligned}
I &= \sum_{i=1}^{a_k} \left\{ i \sum_{j=i}^{a_k} (a_k - j) \right\} \\
&= \sum_{i=1}^{a_k} \frac{i}{2}(a_k - i)^2 \\
&= \tfrac{1}{2} \sum_{i=1}^{a_k} \left[i^3 - 2a_k i^2 + a_k^2 i \right] \\
&= \tfrac{1}{2} \left[\frac{a_k^4}{4} - \frac{2a_k^4}{3} + \frac{a_k^4}{2} \right] \\
&= \frac{a_k^4}{24.}
\end{aligned}$$

Since $g = a_k^2$, we have $I = \dfrac{g^2}{24}$, once again.

It seems as though I is always $\frac{g^2}{24}$, but what actually seems to be the case is that $I=\frac{g^2}{24}$ when $g=O(a_1{}^2)$. This is the case you get when you take almost geometric a_i, such as those sequences covered by Pratt. In fact, if you pick random numbers and compute I and g (by computer), the ratio $\frac{g^2}{I}$ is always near 24 for reasonably sized numbers. In the next case we examine, we see what happens when we have $g=O(a_i{}^{3/2})$.

3.2.4. Sedgewick's Sequences

We look at two cases here. The first computes the number of inversions in the Frobenius pattern of three consecutive increments as given by Sedgewick. The second allows a whole string of consecutive Sedgewick increments much like the previous section did with Hibbard's increments.

3.2.4.1. 3 Increments

Sedgewick's sequence 1, 8, 23, 77, 281, ... , $a_k=4\cdot 4^k+3\cdot 2^k+1$, ... has the property that $g(a_k,a_{k+1},a_{k+2})=O(a_k{}^{3/2})$. Thus in some sense, the 0s and 1s are compacted into closer patterns. Whereas in the previous sequences we expected to add about a constant number of ones per group of a_1 elements (for instance in Hibbard's sequence, ≈ 1 one was added per line), we know that at least some line in the Frobenius Pattern for Sedgewick's increments must have at least $O(a_k{}^{1/2})$ elements as there are only $O(a_k{}^{1/2})$ lines in which to add $O(a_k)$ ones. Moreover, there are no obvious patterns forming as in previous cases.

As we hope that $I(a_k,a_{k+1},a_{k+2})=O(\frac{g^2}{24})$, or at least still $O(g^2)$, we resort to a computer to evaluate I and g for small values. The results follow:

a_1	$g(a_1,a_2,a_3)$	$I(a_1,a_2,a_3)$	$\frac{g^2}{I}$	$\frac{g}{a_1^{3/2}}$	$\frac{ga_1^{3/2}}{I}$
8	153	1309	17.883	6.761	2.644
23	1055	40964	27.170	9.564	2.840
77	8251	1917790	35.498	12.211	2.906
281	65651	105053186	41.027	13.937	2.943
1073	524515	6210219274	44.300	14.923	2.968
4193	4194755	381754387994	46.092	15.449	2.983
16577	33555331	23940751076410	47.031	15.721	2.991
65921	268437251	1516648328239226	47.511	15.860	2.995
262913	2147480061	96570256514494714	47.754	15.929	2.997

From this data, it is very clear that I tends to $\frac{g^2}{48}$. The value $\frac{g}{a_1^{3/2}}$ should asymptotically equal 16; the fact that the last column (apparently) converges to 3 quickly reinforces the denominator 48, as a simple calculation will show. This leads to obvious speculation as to what the denominator is in general (assuming that $I = O(g^2)$). More on that later.

In the mean time, we can prove that $I(a_1,a_2,a_3) = O(g^2)$ for three consecutive increments of Sedgewick's sequence. Unfortunately, the proof achieves $O(g^2)$ with a constant no where near $\frac{1}{48}$, but the evidence for $\frac{1}{48}$ is so compelling that this really doesn't matter all that much. Further, an extension of this $O(g^2)$ bound is sufficient to show that Sedgewick's $O(N^{4/3})$ Shellsort bound is tight and we do this in the next chapter. For now,

Theorem: If $a_i = 4 \cdot 4^i + 3 \cdot 2^i + 1$, $I(a_i,a_{i+1},a_{i+2}) = O(g^2(a_i,a_{i+1},a_{i+2}))$.

Proof: We show that there are $O(8^i) = O(g(a_i,a_{i+1},a_{i+2}))$ ones in the first 2^i lines. This implies at least an equal number of zeros in the last 2^i lines, as there are $16 \cdot 2^i$ lines total (i.e. these ones are entirely in the first half of the pattern). Thus there will be $O(g^2)$ inversions if we consider only these zeros and ones. (This will be a gross underestimate).

Now we have the increments:

$$a_1 = 4 \cdot 4^i + 3 \cdot 2^i + 1$$
$$a_2 = 16 \cdot 4^i + 6 \cdot 2^i + 1$$
$$a_3 = 64 \cdot 4^i + 12 \cdot 2^i + 1$$

Consider a 1 on line $L \le 2^i$. It clearly must have the form

$$4L \cdot 4^i + 3M \cdot 2^i + N.$$

Thus we only need estimate the cardinality of (L, M, N) for $L \le 2^i$. Fix L, and suppose some element in line $L = pa_1 + qa_2 + ra_3$. Then

$$p + 4q + 16r = L \qquad (3.2.4.1a)$$
$$p + 2q + 4r = M \qquad (3.2.4.1b)$$
$$p + q + r = N \qquad (3.2.4.1c)$$

Now, any (p, q, r) that satisfies (3.2.4.1a) yields a unique triple (L,M,N). To see that, fix L, M and N. Then there are three independent equations in three unknowns, hence a unique solution. So how many solutions, with non-negative integers are there to (3.2.4.1a)? The equation

$$p + 4q = L'$$

clearly has about $\frac{L'}{4}$ solutions, so for each $0 \le r \le \frac{L}{16}$, there are $\frac{L-16r}{4}$ solutions. Thus (with big-Oh implied)

$$\left|(L,M,N)\right| = \sum_{r=0}^{L/16} \frac{L-16r}{4}$$
$$= \frac{L^2}{64} - \sum_{r=0}^{L/16} 4r$$
$$= \frac{L^2}{64} - \frac{L^2}{128}$$
$$= \frac{L^2}{128.}$$

Thus each line $L \le 2^i$ has about $\frac{L^2}{128}$ ones. Thus there are

$$\sum_{L=0}^{2^i} \frac{L^2}{128} = \frac{8^i}{384} = O(g)$$

ones in the first 2^i lines, proving the theorem. $\square$

It should be noted that only a small part of the pattern has been looked at to establish this bound; this accounts for the absurd constant (especially absurd when it is squared to get I).

3.2.4.2. Infinite Increments

Suppose we compute $g(a_i, a_{i+1}, \cdots, a_\infty)$ and $I(a_i, a_{i+1}, \cdots, a_\infty)$ as was done with Hibbard's sequence. Do we get the same asymptotic result?

If we proceed as before, we get

$$\begin{aligned} p+4q+16r+64s+256t+\cdots &= L \\ p+2q+4r+8s+16t+\cdots &= M \\ p+q+r+s+t+\cdots &= N \end{aligned}$$

Suppose $L = 2^i$. What is that maximum $|(M,N)|$? Clearly, $L \geq M \geq N \geq 0$ so that $|(M,N)| \leq L^2$. Thus line 2^i has at most 4^i ones by this bound. As $a_i = O(4 \cdot 4^i)$, it is clear that $g(a_i, a_{i+1}, \cdots, a_\infty) \geq \Omega(8^i)$ (i.e. more than 2^i lines of $4 \cdot 4^i$ elements). Of course we know that $g(a_i, a_{i+1}, \cdots, a_\infty) \leq g(a_i, a_{i+1}, a_{i+2}) = O(8^i)$ so that $g(a_i, a_{i+1}, \cdots, a_\infty) = \Theta(8^i) = \Theta(g(a_i^{3/2}))$. Also, we have $|(M,N)| = \Theta(L^2)$ for any line $L \leq 2^i$, so the argument in section 3.2.4.1 holds and $I = O(g^2)$.

We present below the same table as in 3.4.2.1 for the case of arbitrarily many consecutive increments from the Sedgewick sequence, that we have just discussed. Unfortunately, this data doesn't converge quite as fast, and calculations with larger numbers seem beyond the computing power available.

a_1	$g(a_1,a_2,\cdots)$	$I(a_1,a_2,\cdots)$	$\frac{g^2}{I}$	$\frac{g}{a_1^{3/2}}$	$\frac{g a_1^{3/2}}{I}$
8	153	1309	17.883	6.761	2.644
23	1055	40964	27.170	9.564	2.840
77	8251	1917790	35.498	12.211	2.906
281	61458	88885244	42.493	13.047	3.256
1073	432659	4147357900	45.135	12.309	3.666
4193	3266443	239175467622	44.610	12.030	3.708
16577	24590846	13438742458592	44.997	11.521	3.905

The theoretical value of $\frac{g}{a_1^{3/2}}$ is unknown, but a conjecture is that it approaches $\frac{32}{3}$, or 2/3 the value of the previous case. The proof above can be extended to the case $a \cdot 8^i + b \cdot 4^i + c \cdot 2^i + d$ for constants a, b, c, d. As $(2^i - 3)(2^{i+1} - 3)(2^{i+2} - 3)$ is of this form, we can show that $I = O(g^2)$ for this particular example of an $O(N^{4/3})$ Frobenius number. In fact this can be extended as in Chapter 2 to continue to obtain $I = O(g^2)$, but in the case where the polynomial in 2^i is obtained by extending the multiplications as in section 2.3.3, the constant in the denominator implied by the big-Oh notation can be exponential in c. This is due to the proof technique which uses less and less of the Frobenius pattern (always 2^i even though the pattern gets longer). A more careful proof wouldn't suffer from this problem. Of course, this would only prove a result for the case where the increments were polynomials in α^i.

3.3. The Inversion Conjecture

We've seen that when $g = O(a_1^2)$ we can prove for several cases and confirm empirically that the number of inversions in the Frobenius Pattern (I) tends to $\frac{g^2}{24}$. We've also seen that when $g = O(a_1^{3/2})$, I seems to tend to $\frac{g^2}{48}$. What happens when $g = O(a_1^{(1+1/k)})$? Do we still have $I = O(g^2)$ with a small denominator or does the denominator double every time we raise k? The answer from somewhat scant empirical data is that I tends to $\frac{g^2}{24k}$. For instance, when $g = O(a_1{}^{4/3})$, we see that I is always at least $\frac{g^2}{72}$. Unfortunately we can't prove this result, so instead we will offer some empirical evidence.

3.3.1. Empirical Results

For the case where $g = O(a_1{}^{4/3})$, we have the sequence 1, 5, 65, 1885, 22997, 221123, 1921125, ... , $a_i = (2^i - 3)(2^{i+1} - 3)(2^{i+2} - 3)$. We summarize the results below. Unfortunately, the increments grow rather quickly and thus we are unable to see the asymptotic growth of I.

a_1	$g(a_1, a_2, a_3, a_4)$	$I(a_1, a_2, a_3, a_4)$	$\frac{g^2}{I}$
5	91983	528816016	15.999
65	2745423	342801348496	21.987
1885	56535103	129518018602656	24.677

We need to try other sets of increments that don't grow quite as fast as the ones above. Another set of increments that can be tried is ... , 24679, 50431, 102601, 202429, 383323, 734161, 1397923, 2717311, ... which is formed by multiplying together three consecutive terms of the sequence ... , 17, 23, 29, 37, 47, 59, 73, 89, 113, 139, 173, ... The numbers in the latter sequence are primes spaced out in multiples of $2^{1/3}$. Thus the increments we test are $O(2^i)$. This sequence asymptotically has $g = (4 + 8 \cdot 2^{1/3} + 4^{2/3})(a_1^{4/3}) \approx 15.64 a_1{}^{4/3}$. It also doesn't quite reach its asymptotic point either, but we still have $I \geq \frac{g^2}{72}$.

a_1	$g(a_1,a_2,a_3,a_4)$	$\frac{I(a_1,a_2,a_3,a_4)}{2^{31}}$	$\frac{g^2}{I}$
11339	4365835	201.024	44.152
24679	11245075	1285.263	45.814
50431	27742699	7554.051	47.444
102601	67399039	43775.560	48.322
202429	160662223	244705.081	49.119
383323	381971839	1373060.312	49.481

A third set is

$$a_1 = 8^i$$
$$a_2 = 8^i + 4^i$$
$$a_3 = 8^i + 4^i + 2^i$$
$$a_4 = 8^i + 4^i + 2^i + 1$$

These obviously don't make good increments for Shellsort since they are all very close to 8^i, but applying Johnson's formula gives a good, small value for g. Also, since $a_4 \approx a_1$, these are a little easier to compute for larger a_1. I converges much faster for these arguments as the table below shows:

i	a_1	$g(a_1,a_2,a_3,a_4)$	$I(a_1,a_2,a_3,a_4)$	$\frac{g^2}{I}$
1	8	33	51	21.352
2	64	683	12413	37.580
3	512	11703	2647750	51.727
4	4096	192239	605971900	60.986
5	32768	3111903	146161413176	66.255
6	262144	50065343	36292446477168	69.065
7	2097152	803192703	9148486063407840	70.516

When $i=7$, we have $I(2097152, 2113536, 2113664, 2113665)$ is very nearly $\frac{g^2}{72}$.

Similar extensions are possible, using the sequences similar to the one above. (Extensions to the other sequences don't come close to converging. This isn't surprising since the other sequences weren't converging well anyway.) If we take

$$a_1 = 16^i$$
$$a_2 = 16^i + 8^i$$
$$a_3 = 16^i + 8^i + 4^i$$
$$a_4 = 16^i + 8^i + 4^i + 2^i$$
$$a_5 = 16^i + 8^i + 4^i + 2^i + 1$$

we get the following results:

i	a_1	$g(a_1,a_2,a_3,a_4,a_5)$	$I(a_1,a_2,a_3,a_4,a_5)$	$\frac{g^2}{I}$
1	16	97	343	27.431
2	256	3755	274877	51.295
3	4096	126391	227285702	70.284
4	65536	4124399	207072634300	82.148
5	1048576	133135327	199609919614008	88.798

We begin to see a convergence of I to $\frac{g^2}{96}$. The extensions to $k=6$ and 7 are summarized below. The convergence to $\frac{g^2}{120}$ and $\frac{g^2}{144}$ isn't quite as fast but it seems implied. As a_1 grows exponentially, it is difficult to extend this further. Nonetheless, the pattern seems very clear.

i	a_1	$g(a_1,a_2,a_3,a_4,a_5,a_6)$	$I(a_1,a_2,a_3,a_4,a_5,a_6)$	$\frac{g^2}{I}$
1	32	257	1935	34.133
2	1024	19115	5583037	65.445
3	32768	1273271	18231065286	88.926
4	1048576	82767599	66302953235900	103.320

i	a_1	$g(a_1,a_2,a_3,a_4,a_5,a_6,a_7)$	$I(a_1,a_2,a_3,a_4,a_5,a_6,a_7)$	$\frac{g^2}{I}$
1	64	641	9951	41.290
2	4096	92843	108074173	79.758
3	262144	12283319	1402432191174	107.584

3.3.2. The Conjecture

If we combine the results of the previous section, with the known lower bounds for the Frobenius function, we can get a lower bound for the number of inversions in the Frobenius pattern.

We know that

$$g(a_1, a_2, \cdots, a_k) \geq \Omega(a_1^{1+1/(k-1)})$$

so the inversion conjecture becomes:

INVERSION CONJECTURE:

$$I(a_1, a_2, \cdots, a_k) \geq \Omega(\frac{g(a_1, a_2, \cdots, a_k)^2}{k-1})$$

3.4. Summary

We have analyzed the number of inversions in the Frobenius pattern for several useful (as we shall see later) cases. We conjecture that the number of inversions in the pattern is at least $\Omega(\frac{g^2}{k-1})$ when there are k arguments to the Frobenius function, based on some empirical evidence. This data is hard to come by, since most patterns don't converge to what we think is the theoretical value fast enough. One pattern comes close and we use it to look at the number of inversions for several values of k. In these cases, the last pattern we look at for a given k begins to require a lot of computing power, so we can't get much more data.

Proving the conjecture for the general case is difficult. One reason is that there is at least one degenerate case for which this conjecture isn't true ($a-1$ consecutive numbers starting at a). On the other hand, we have strong evidence that for the cases we are interested in, with $k << a$, the conjecture is true because the zeros and ones are well distributed. Data suggests that the pattern is always nearly symmetric (sometimes entirely so) for all cases but the degenerate example mentioned. We also seem to have a reasonable fraction of ones in the top half. If there are k arguments behaving as the lower bound on the Frobenius function allows, then simulation suggests that 1 of every $2k$ elements in the top half is a one. This in itself implies that I is at least $O(\frac{g^2}{k^2})$. Proving even this relaxation seems hard, but it would be an important theoretical result, since for asymptotic Shellsort analysis, the two forms are equivalent.

4. Lower Bounds For Shellsort

In this chapter, we use the results of Chapter 3 to obtain lower bounds for Shellsort using various increment sequences. We analyze the running time for Shellsort on a particular permutation and obtain an $\Omega(N^{4/3})$ lower bound for Shellsort using the original increments suggested by Sedgewick. Extending these results, we can produce a lower bound that matches the upper bound for all the increment sequences discussed in Chapter 2, providing that a weak form of the Inversion Conjecture is true. We also provide a lower bound for any "geometric" increment sequence that matches the best known upper bound for these types of increment sequences.

4.1. Attempts At Bad Permutations

What does a bad permutation look like? For insertion sort, the algorithm which Shellsort improves on, the worst case is a file in reverse order. For our examples, we will take $N=10$, and the increments will be 5,3,1. If we run Shellsort on this file, the results (13 exchanges total) aren't nearly as bad as insertion sort.

Original	10	9	8	7	6	5	4	3	2	1	exchanges
After 5-sort	5	4	3	2	1	10	9	8	7	6	5
After 3-sort	2	1	3	5	4	7	6	8	10	9	4
After 1-sort	1	2	3	4	5	6	7	8	9	10	4

The reason Shellsort does well on this file is that this kind of permutation is exactly the type Shellsort is meant to deal with. In its first pass, it puts all the big elements on the right and all the small elements on the left; after that the file is nearly sorted. An obvious strategy then is to try to neutralize the first pass. Our second attempt is to put large numbers 5 apart. Thus the first pass still makes 5 exchanges but now the file is not nearly as sorted as in the first example.

Original	10	8	6	4	2	9	7	5	3	1	exchanges
After 5-sort	9	7	5	3	1	10	8	6	4	2	5
After 3-sort	2	1	4	3	6	5	8	7	10	9	9
After 1-sort	1	2	3	4	5	6	7	8	9	10	5

This strategy clearly increases the work Shellsort must do (19 exchanges). The result here is that the 3-sort does most of the sorting; the file is nearly sorted when the 1-sort comes around.

The final strategy is to make both the 5 and 3 sort do work, but not allow them to nearly sort the file and let the 1-sort have an easy time. Thus the 1-sort does a lot of work; more than twice the work than in the previous examples, thus 22 exchanges total.

Original	10	4	3	9	2	8	7	1	6	5	exchanges
After 5-sort	8	4	1	6	2	10	7	3	9	5	3
After 3-sort	5	2	1	6	3	9	7	4	10	8	7
After 1-sort	1	2	3	4	5	6	7	8	9	10	12

Actually, switching the 4 & 7 and the 6 & 9 in the last permutation gives 2 more exchanges, but it turns out that this doesn't matter asymptotically. The obvious trick in here is that elements a distance 3 or 5 from a large element are also large. An extension to this type of permutation will be what we use to lower bound Shellsort.

4.2. Correlation To The Frobenius Pattern

The permutation above has a connection to the Frobenius problem. To see this, let's divide the numbers $1-10$ into big and little numbers. It happens in this case that little is $1-4$ and big is $5-10$. If we call a big number a 1 and a little number a 0, then the permutation is

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

Now, after the 5-sort we have

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

Notice that in the world of big and little numbers, nothing has happened. After the 3-sort, we have

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

There is still no change in the 0-1 pattern. Of course after the 1-sort we have

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

and we see that the 0-1 pattern is moved quite a bit, indicating that a lot of exchanges were required.

A careful look at the (original) 0-1 pattern shows that it is the Frobenius Pattern for $g(3,5)$ with a 1 prepended at the start (in position 0) and some ones at the end (because $g(3,5)=7$, not 9). In fact, this is how the permutation is formed: The Frobenius Pattern with leading 1 is written, along with as many trailing 0s as are needed. Going from left to right, large numbers are assigned in decreasing order as 1s are encountered in the pattern. At the end, we go back from right to left assigning small numbers in increasing order as 0s are encountered. We will generalize this in the next section to form more general permutations. We will also now deal exclusively with 0-1 permutations. We will show later that this is linear-time equivalent to the general case, thus justifying this simplifying approach.

4.3. Definitions and Lemmas

In this section we will define our permutations more formally and prove two simple lemmas that enable us to lower-bound Shellsort.

Once again, suppose our increment sequence is $h_t, h_{t-1}, \cdots, h_1 = 1$, and our permutation has N elements to sort. Our bad permutation will be $P(k)$, defined as follows:

Definition: $P(k) \equiv a_0 a_1 a_2 \cdots a_{N-1}$ such that $a_i = 1$ iff a_i is representable as a linear combination in non-negative integer coefficients of $h_{t-(k-1)}, h_{t-(k-2)}, \cdots, h_t$ and 0 otherwise.

Thus, $P(k)$ is the Frobenius Pattern, with a 1 prepended at the start and trailing 1s added if necessary, of the k largest increments. With the increments 1,2,3,5 and $N=8$, we have $P(1)=10000100$, $P(2)=10010110$, $P(3)=10111111$. It is clear that $P(3)$ is not a bad permutation while $P(1)$ and $P(2)$ have some disorder in them. That $P(3)$ is ordered is not surprising since $g(2,3,5)=1$, so there are many trailing 1s. It is important then to choose k so that $g(h_{t-(k-1)}, h_{t-(k-2)}, \cdots, h_t) \approx N$.

We now prove two simple but important lemmas.

Lemma 3: For the permutations $P(k)$, no exchanges are performed by Shellsort for the increments $h_t, h_{t-1}, \cdots, h_{t-(k-1)}$.

Proof: Trivially obvious because for any $h_{t'}$ such that $t-(k-1) \le t' \le t$, if $a_x = 1$ then $a_{x+h_{t'}}$ must also equal 1, so the lemma follows. $\square$

Lemma 3 tells us that no exchanges can be performed in the early passes of

Shellsort. Thus all the exchanges are performed by the smaller passes. The next lemma tells us how many inversions these exchanges can remove.

Lemma 4 (The Swapping Lemma): Swapping a 0 and 1 a distance d apart in a 0-1 permutation removes exactly d inversions.

Proof: Consider an element between the 0 and 1 before the swap:

$$1 \ldots e \ldots 0$$

If $e=0$, one inversion is removed because the e and the first element (1 before the swap, 0 after) will no longer be inverted after the swap. Similarly, if $e=1$, then one inversion is removed because e and the last element will no longer be inverted. There are $d-1$ elements of this type. This plus the inversion that is removed by actually swapping the 0 and 1 accounts for a total of d inversions. This is all there can be since none of the elements to the left of the 1 or right of the 0 will be "uninverted". □

4.4. Proofs of Lower Bounds For Known Increment Sequences

We are now ready to lower-bound Shellsort's running time for various increment sequences. Our strategy will be to use a permutation, $P(k)$, where k is dependent on the increment sequence. Our measurement of time will be the number of exchanges needed to complete the sort.

The method of attack is illustrated by bounding the running time for Hibbard's increments. Of course Pratt showed sometime ago that this lower bound is $\Omega(N^{3/2})$, but our method is simpler and easier to extend, because our permutations are simpler.

4.4.1. Hibbard's Increments

Theorem: The running time for Shellsort is $\Theta(N^{3/2})$ for the increments 1, 3, 7, ... , $2^i - 1$, ...

Proof: We have already shown the upper bound in sec. 2.3.1, thus we need only prove the lower bound.

We have $h_i = 2^i - 1$, so we choose N to be of the form $h_i(h_i+1)$ for any i. We analyze the running time for the permutation $P(k)$ of size N, where in this case we choose $k = i$.

Now, the largest increment less than N is h_{2i-1}, so $P(k)$ consists of all integers represented as a linear combination with non-negative integer coefficients of $h_i, h_{i+1}, \cdots, h_{2i-1}$. We use the fact that $g(h_i, h_{i+1}, \cdots, h_{2i-1}) = h_i(h_i+1)-1$ to conclude that our permutation is exactly the Frobenius pattern with a one prepended. Thus we know that there are $O(N^2)$ inversions, since $N = g$ and we have proven that the Inversion Conjecture is true for this case. We also know, by Lemma 3, that after the h_i-sort there are still $O(N^2)$ inversions since the $h_{2i-1}, h_{2i-2}, \cdots, h_i$-sorts do not affect the permutation. Moreover, by the swapping lemma, all subsequent exchanges remove at most $O(h_{i-1})$ inversions, since this becomes the largest distance that any two elements can be exchanged. Thus we have a lower bound on the number of exchanges, and hence the running time:

$$T \geq \Omega\left(\frac{N^2}{h_{i-1}}\right)$$

h_{i-1} is easy to estimate. We have

$$h_{i-1} = O(h_i) = O(N^{1/2}),$$

hence,

$$T \geq \Omega(N^{3/2})$$

and thus

$$T = \Theta(N^{3/2}).\square$$

If we are concerned with permutations of the integers $1 \cdots N$, it is easy to show that this bound still holds if we assign integers as suggested earlier in this chapter.

We know that when we come to h_{i-1}-sort, we have at least as many permutations in the $1...N$ permutation as in the 0-1 permutation, since every inversion in the 0-1 permutation corresponds to an inversion in the $1...N$ permutation. Now, when exchanging elements d apart in the 0-1 permutation we remove exactly d inversions. In the $1...N$ permutation we remove between 1 and $2d-1$ inversions, but still $O(d)$. Thus the key step (the division between total inversions and maximum inversions removed in an exchange) is the same. Thus we may use 0-1 permutations instead of permutations over the integers $1...N$ in all applications of this proof technique. Thus, all future proofs involve only zeros and ones.

Pratt used the same ideas in obtaining his lower bound. His permutations are also h_i, h_{i+1}, ... , h_{2i-1}-sorted. However, his permutations are far more complicated and very difficult to generate by hand. Moreover, the constant implied in Pratt's lower bound is $\frac{1}{128}$, while our constant is $\frac{1}{12}$.

4.4.2. Sedgewick's Increments

If we apply the same techniques, we can get a tight lower bound for Shellsort using the original increments suggested by Sedgewick. We have

Theorem: The running time for Shellsort is $\Theta(N^{4/3})$ for the increments 1, 8, 23, 77, ... , $4 \cdot 4^i + 3 \cdot 2^i + 1$, *Proof:* We have already shown the upper bound in sec. 2.3.2, thus again we need only prove the lower bound.

We have $h_i = 4 \cdot 4^i + 3 \cdot 2^i + 1$, and we choose N to be of the form $g(h_i, h_{i+1}, \cdots, h_\infty)$ for any h_i. We will analyze the running time for the permutation $P(k)$ where k is the number of increments at least as large as h_i and smaller than N. As an example, if $h_i = 23$, we have (sec. 3.2.4.2) $g(23, 77, 281, 1073, \cdots) = 1055$, so we take $N = 1056$, $h_i = 23$ and $k = 3$ and use the permutation $P(3)$. Again, our permutation is exactly the Frobenius Pattern with a one prepended. Thus we start with (asymptotically) $\frac{N^2}{48} = O(N^2)$ inversions. After the h_i-sort we still have $O(N^2)$ inversions, since no exchanges have taken place. Now, all subsequent exchanges remove at most $O(h_{i-1})$ inversions, hence

$$T \geq \Omega(\frac{N^2}{h_{i-1}})$$

as before. The difference is the size of h_{i-1}. We have

$$h_{i-1} = O(h_i)$$

and

$$g(h_i, h_{i+1}, h_{i+2}, \cdots) = O(h_i^{3/2}) = N$$

so that

$$h_i = O(N^{2/3}).$$

Hence,

$$T \geq \Omega(N^{4/3})$$

and thus,

$$T = \Theta(N^{4/3}). \Box$$

Below, we show some statistics for Shellsort at the values of N used in the above proof (The permutations are over the integers 1...N). Nonexchanging comparisons are a function of the increment sequence used, not the permutation. They are a lower bound for Shellsort's running time (for instance when the input is already sorted) and are $O(N\log N)$. The random file is just that -- one random file. Previous studies show that there is not much gained by taking a lot of samples for each permutation size.

N	Non-exch. Comparisons	Exch. for random file	Exch. using P(k)
154	507	1078	3662
1056	4890	11006	28739
8252	52108	134073	427644
61459	469439	1343998	5998465
432660	3975533	12218514	77273940

We see that our permutations take considerably longer to sort than average permutations, and that exchanges dominate the running time.

We've shown that for the increments invented by Sedgewick, we can construct bad permutations that make Shellsort take $O(N^{4/3})$ time. On the other hand, we've only shown this for a few values of N. For $N<1000000$, we only have shown the 5 bad permutations above. It would be nice (but not necessary for the theoretical result) to show that for any N we can construct a permutation that takes time $O(N^{4/3})$.

Formally, suppose

$$N_1=g(h_i,h_{i+1},\cdots,h_\infty)+1$$
$$N_2=g(h_{i+1},h_{i+2},\cdots,h_\infty)+1$$

and

$$N_1\le N\le N_2$$

Thus we are between values of the Frobenius function so we don't know which one to use. If $N\approx N_1$, it makes sense to just choose k to include h_i, and use the permutation $P(k)$ which will include the Frobenius pattern with a prepended one and some $(N-N_1-1)$ ones. This shouldn't be too drastic and shouldn't affect the asymptotic running time if $N_1=O(N)$. If $N \approx N_2$, then we choose k to not include h_i, but start at h_{i+1}. Then we will have an incomplete Frobenius pattern, but if $N_2=O(N)$ it will only be missing mostly ones at the end, and the result won't be drastic.

What if N is *really* between N_1 and N_2 and not $O(N_1)$ or $O(N_2)$? Can we still use the permutation and get a tight lower bound? The answer is yes and the method is simple. Choose k to not include h_i, but start at h_{i+1} as in the case where $N=O(N_2)$. However, the trick (and it can and should be applied to the case $N=O(N_2)$) is to use the *middle* N elements of the Frobenius pattern instead of the first N. Intuitively this is because the middle of the Frobenius pattern is more "scrambled", with a greater inversion density than the Frobenius pattern in general. The Frobenius pattern starts out with mostly zeros and ends with mostly ones hence the ends don't add many inversions relative to the added length they contribute to the pattern.

Proving that there are $O(N^2)$ inversions in this case for Sedgewick's increments is trivial since the general proof looked at only the first few and last few lines and not the whole file. As lines closer to the middle of the file from above have more ones than lines on the top, and lines closer to the middle of the file from the bottom have more zeros than lines on the bottom, we are guaranteed to find enough inversions. Proving this in the general case seems to be as hard as proving the Inversion Conjecture, but we know that for this and similar cases the theorems we prove are not for just a few scant values of N.

4.4.3. Extension to Incerpi and Sedgewick's Increments

Let's look closely at how the lower and upper bounds for Shellsort were obtained for Hibbard's and Sedgewick's increments, and extend this for the sequences invented by Incerpi and Sedgewick.

Suppose we have increments chosen so that $g(h_i, h_{i+1}, \cdots, h_{i+(c-1)}) = O({h_i}^{1+1/(c-1)})$. Thus, in Hibbard's case we have $c=2$ and in Sedgewick's case we have $c=3$.

In Hibbard's case we have

$$g(h_i, h_{i+1}) \approx 2h_i^2$$

and

$$g(h_i, h_{i+1}, \cdots, h_\infty) \approx h_i^2 .$$

In Sedgewick's case we have

$$g(h_i, h_{i+1}, h_{i+2}) \approx 16 h_i^{3/2}$$

and

$$g(h_i, h_{i+1}, \cdots, h_\infty) \approx p h_i^{3/2} ,$$

where p seems to be 32/3 or 2/3 of 16. What appears to happen in general is that if

$$g(h_i, h_{i+1}, \cdots, h_{i+(c-1)}) = O({h_i}^{1+1/(c-1)})$$

then

$$g(h_i, h_{i+1}, \cdots, h_\infty) = O({h_i}^{1+1/(c-1)})$$

also. This certainly isn't a major surprise, since it takes a lot of care to lower the asymptotic growth of g and thus one wouldn't expect a smaller value than planned.

The other fact we will use is the Inversion Conjecture. If we assume it is true, then if

$$g(h_i, h_{i+1}, \cdots, h_{i-(c-1)}) = O({h_i}^{1+1/(c-1)}),$$

then

$$I(h_i, h_{i+1}, \cdots, h_\infty) = O(\frac{1}{c-1} h_i^{1+1/(c-1)}).$$

Now, lets look at the upper and lower bounds for these sequences *using these assumptions* if we tailor c consecutive increments so that the Frobenius number is asymptotically minimal. This is because we have

$$g(h_{i+1}, h_{i+2}, \cdots, h_{i+c}) = O(h_i^{1+1/(c-1)})$$

Thus,

$$\frac{g(h_i, h_{i+1}, \cdots, h_{i+(c-1)})}{h_i} = O(h_i^{1/(c-1)}),$$

and as in sec. 2.3,

$$T \le \sum_{h_i < N^{(c-1)/c}} N h_i^{1/(c-1)} + \sum_{O(N^{(c-1)/c}) \le h_i \le O(N)} \frac{N^2}{h_i}$$

Both sums are geometric with largest term $O(N^{(1+1/c)})$, hence this is the sum also.

For the lower bound, we choose N to be of the form $g(h_i, h_{i+1}, \cdots, h_\infty)+1$ for any h_i. As before,

$$T \ge \Omega(\frac{N^2}{h_{i-1}})$$

Now,

$$h_{i-1} = O(h_i)$$

and

$$g(h_i, h_{i+1}, \cdots, h_\infty) = O(h_i^{(1+1/(c-1)}) = N$$

so

$$h_i = O(N^{((c-1)/c)}) \text{ and}$$

hence $T \ge \Omega(N^{(1+1/c)})$. Thus in these case,

$$T = \Theta(N^{(1+1/c)})$$

The upper bounds of sec 2.3.3, namely $O(N^{(1+1/(c+1))})$ fall into this case. (In these bounds c refers to the number of multiplicands, which is one less than the number of consecutive increments taken into account by the Frobenius function.

Hence the slight difference in the exponent. Thus, $h_i = (2^i - 3)(2^{i+1} - 3)$ has two multiplicands while the Frobenius function for three arguments is used to bound.)

This tight bound also applies to the $O(N^{1+\varepsilon/\sqrt{\log N}})$ sequences of Incerpi and Sedgewick since they in effect use the Frobenius function for $O(\sqrt{\log N})$ increments. It should be noted that this bound assumes the Inversion Conjecture and also assumes that adding the extra arguments to the Frobenius function doesn't asymptotically alter the value of g. For this increment sequence, we show statistics for this permutation as compared to a random ordering.

N	Non-exch. Comparisons	Exch. for random file	Exch. using P(k)
134	733	577	886
1734	15656	13322	44424
3142	31008	28118	111176
4022	40688	35732	149998
67382	948383	938215	6156640
128582	1929067	1964555	18397296
182438	2844619	2925135	28408082
216098	3416839	3577776	33461851

The values of N in the above table do not grow at a smooth rate. Instead, they hover near a value and then jump sharply. This is because the of the way the increment sequence is chosen -- many of the increments are multiples of each other.

4.4.4. Chazelle's Increments

The increment sequence suggested by Chazelle doesn't quite fall into the class above, so we prove the lower bound separately. Again we assume the Inversion Conjecture. Now, the increments are of the form $\alpha^i(\alpha+1)^j$. Let $h_k = \alpha^k$ for some k, and suppose $h_{k'} = \alpha^{k+1}$. Consider $g(h_k, h_{k+1}, \cdots, h_\infty)$. It is clear that $g(h_k, h_{k+1}, \cdots, h_\infty) = g(h_k, h_{k+1}, \cdots, h_{k'-1})$ since all larger increments are merely multiples of either h_k, h_{k+1}, $\cdots$, or $h_{k'-1}$. Now there are at most k

increments between h_k and $h_{k'}$ inclusive, and since $i = \frac{\log h_k}{\log \alpha}$, we have

$$g(h_k, h_{k+1}, \cdots, h_\infty) \geq O(h_k^{1+\frac{1}{1+(\log h_k)/(\log \alpha)}})$$

Again, $h_{k-1} = O(h_k)$, and we solve for h_k. Big-Oh notation is implied,

$$h_k^{(1+\frac{\log\alpha}{\log\alpha + \log h_k})} \approx N$$

For constant α, this eventually implies

$$h_k \leq \frac{N}{\alpha}$$

Thus, since by the Inversion Conjecture

$$T \geq \Omega(\frac{N^2}{h_{k-1}})$$

we have

$$T \geq N\alpha.$$

Now, as we want $O(\log N)$ increments instead of $O(\log^2 N)$, Chazelle chose α such that

$$\frac{\log^2 N}{\log^2 \alpha} = \frac{\log N}{\alpha'}$$

or,

$$\alpha = O(N^{\sqrt{\alpha'/\log N}}).$$

This yields

$$T \geq \Omega(N^{1+\sqrt{\alpha'/\log N}})$$
$$\geq \Omega(N^{1+\varepsilon/\sqrt{\log N}})$$

for $\varepsilon = \sqrt{\alpha'}$. Again this result assumes the Inversion Conjecture. One note: The functional form of the lower bound matches the upper bound, although given α', the value of ε in the lower bound is half that of the upper bound.

4.5. Lower Bounds For General Increment Sequences

We have shown that all of the analyzed upper bounds for Shellsort are probably tight and thus clever analysis won't help lower them further. Thus, if we want to get better bounds, new increment sequences will have to be invented. We now discuss the kinds of increment sequences that might be tried. These sequences fall into two categories: uniform and non-uniform.

A uniform sequence is an infinite sequence of increments that is cut off when the size of the increments becomes larger than the permutation size. Thus, Hibbard's, Sedgewick's and Incerpi and Sedgewick's sequences are uniform, as is Pratt's $2^i 3^j$ sequence. Non-uniform sequences are allowed to choose increments based on the permutation size, N. Examples are Shell's original sequence, Chazelle's sequence and a sequence due to Gonnet [Gon84] $\lfloor N\alpha \rfloor, \left\lfloor \lfloor N\alpha \rfloor \alpha \right\rfloor, \cdots,$ with $\alpha = 5/11$. Uniform sequences have been studied more heavily because non-uniform sequences can have bad behavior for certain file sizes, N, unless designed carefully. Moreover, the current state of affairs is that uniform and non-uniform sequences are asymptotically equivalent in the worst case.

All increment sequences used or bounded have $h_1 < h_2 < \cdots < h_t$. Moreover, almost all sequences have $O(\log N)$ increments with the hope of obtaining an $O(N \log N)$ sort, and because this performs better in practice. For uniform increment sequences, this has the effect that all sequences of this type studied have been "almost geometric". That is, the increments are of the form $O(\alpha^i)$.

Our next theorem shows that this is probably a hopeless attack on finding an $O(N \log N)$ Shellsort.

Theorem: (If the Inversion Conjecture is true,) the running time for Shellsort is

$\Omega(N^{1+\varepsilon/\sqrt{\log N}})$, (with $\varepsilon=\sqrt{\log \alpha}$) for increments that are $O(\alpha^i)$.

Proof: We choose N to be of the form $g(h_i,h_{i+1},\cdots,h_\infty)+1$ for any h_i. We form the permutation $P(k)$ and solve for the minimum value of k by using two facts. First, since the increments are geometric,

$$h_i=O(\frac{N}{\alpha^k})$$

Also, by the lower bound for the Frobenius function,

$$h_i^{1+1/(k-1)}\geq N.$$

Combining these equations,

$$\frac{N^{k/(k-1)}}{\alpha^{k^2/(k-1)}}\geq N$$

And thus we can solve for k:

$$k\geq\Omega(\sqrt{\log_\alpha N})$$

by plugging the first equation into the second. Now,

$$h_{i-1}=O(h_i)=O(\frac{N}{\alpha^k})\leq O(N^{1-\sqrt{\log_N \alpha}})$$

where we have used the equality

$$\alpha^{\sqrt{\log_\alpha N}}=N^{\sqrt{\log_N \alpha}}.$$

Now, if the Inversion Conjecture is assumed, we may use the bound

$$T\geq\Omega(\frac{N^2}{kh_{i-1}})$$

which yields

$$T\geq\Omega(N^{1+\varepsilon/\sqrt{\log N}})$$

for $\varepsilon=\sqrt{\log \alpha}$, since it is easily verified that the value of k is asymptotically unimportant. $\square$

4.6. Strength of Assumptions For The General Lower Bound

We now look more closely at the assumptions made to prove the lower bound of the previous section.

4.6.1. Inversion Conjecture

The first assumption is the Inversion Conjecture, which says that the number of inversions is $O(\frac{N^2}{k})$. Clearly if the N^2 part of the conjecture is untrue, the proof falls through, but what if the correct form is really $O(\frac{N^2}{k^2})$ or $O(\frac{N^2}{k^p})$ for some p? Let's use this form in the proof and see how large p can grow without affecting the result. We have

$$k = O(\sqrt{\log_\alpha N})$$

and we need to make sure that

$$k^p < N^{1/k}$$

Thus

$$p \log k < \frac{1}{k} \log N$$

or

$$p < \frac{\log N}{k \log k}$$

Since $k = \Omega(\sqrt{\log N})$, it follows that $\log N = O(k^2)$. Thus, we need

$$p < O(\frac{k}{\log k}).$$

Hence, as long as there are more than $\Omega(\frac{N^2}{k^{(k/\log k)}}) = \Omega(\frac{N^2}{2^k})$ inversions, the proof will go through. This means any constant power of k can be in the denominator; specifically, if $I = \Omega(\frac{N^2}{k^2})$, the proof is still valid. This is important since it seems that this value of I might be somewhat easier to prove. This weaker form of the

Inversion Conjecture is also valid for the other $O(N^{1+\varepsilon/\sqrt{\log N}})$ sequences of Incerpi and Sedgewick. In the other proofs, k is a constant throughout, so as long as $I=O(N^2)$, the proofs work.

4.6.2. Increment Sequences

The other item of interest is to see how reasonable it is to assume that the increments are geometric and what ways there might be to get around the lower bound and produce a sequence that yields an $O(N\log N)$ Shellsort.

As we have noted already, virtually all $O(\log N)$ sequences studied satisfy $h_i=O(\alpha^i)$. The notable exception is Chazelle's sequence with increments 2^i3^j. Now, it seems that uniform $O(\log N)$ sequences have to be of the general form assumed although there may be ways to get around this. One way is to use a standard sequence but stick in some small (or mid-sized) increments as early passes, or jumble the increments so that the h_i-sorts are not necessarily in decreasing values of h_i. Thus instead of 255, 127, 63, 31, 15, 7, 3, 1 doing 15, (225 is useless), 127, 63, 31, 7, 3, 1 would bring the Frobenius number down to a small value quickly, and imply a large value of h_i (and hence a weaker lower bound). Thus the lower bound arguments wouldn't work as written. It is easy to do another argument for this case, however, by just choosing a permutation that makes the 15-sort take quadratic time immediately. Thus, the strategy of mixing up the increments will bypass the original proof, but then a slight modification of the ideas used in the proof will reestablish the bound.

With this in mind, it may be the case that the condition that the increments be of the form $O(\alpha^i)$ is too strong. What might suffice would be "$O(\log N)$ uniform increment sequence"(a far stronger statement, although it doesn't include non-uniform geometric sequences -- this can be added), since it seems difficult to invent a sequence of this type that bypasses the original version of the proof and slight modifications also. Moreover, it seems almost impossible that even if this type of sequence could be invented, it could be shown to make Shellsort run in $O(N\log N)$.

The non-uniform case is slightly different because Chazelle's sequence is $O(\log N)$ increments that are not $O(\alpha^i)$. It is easy to invent non-uniform $O(\log N)$ sequences that are not geometric, and it is certainly possible to invent very reasonable sequences for which the lower bound proof won't work. These still can't be geometric or "scrambled geometric", etc. Another idea is to put a big gap in the increment sequence at $O(N^{1-\varepsilon/\sqrt{\log N}})$ in the hope of forcing the Frobenius number way down, but this doesn't work. The counter-strategy is to use all the increments larger than $O(N^{1-\varepsilon/\sqrt{\log N}})$ to form the pattern, and use the middle of the pattern as the permutation. This guarantees $O(N^2)$ inversions (with an appropriate denominator that is insignificant) and $h_i < N^{1-\varepsilon/\sqrt{\log N}}$, hence the required bound.

Since non-uniform sequences have never been asymptotically better in the worst case than uniform sequences, it is easy to speculate that an $O(N\log N)$ Shellsort with non-uniform increments implies one for uniform increments. The latter seems unlikely.

4.7. Summary

For all increment sequences previously analyzed, we have shown that the upper bounds derived for Shellsort are almost certainly tight. For Sedgewick's sequence and some of the sequences that yield $O(N^{1+\varepsilon})$ Shellsorts, this is proved; in other cases the proof is conditional on a very weak form of the Inversion Conjecture that essentially requires a quadratic number of inversions, but allows the number of inversions to drop dramatically as more arguments are added to the Frobenius function.

If we assume the weak form of the Inversion Conjecture, we can then prove a very general theorem that shows that the sequences of Incerpi and Sedgewick that yield $O(N^{1+\varepsilon/\sqrt{\log N}})$ upper bounds are the best $O(\log N)$ sequence possible if we use "almost geometric" increment sequences. More importantly, this makes the prospect of an $O(N \log N)$ Shellsort very dim. This is because it would follow that the only hope would be a non-uniform increment sequence that didn't have increments of the form $O(\alpha^i)$. On the other hand, it seems that if this bound were attainable for non-uniform sequences, it would be attainable for uniform sequences as well, thus making it highly unlikely that this will work.

5. Shellsort Variations - Shaker Sort

In this chapter we discuss a variant of Shellsort invented by Incerpi and Sedgewick [Inc85] known as Shaker-sort. Shaker-sort uses an increment sequence like Shellsort, but fixes the work done in each pass to be linear. At the end of the algorithm there is an insertion sort mop-up to finish the sort. If an $O(\log N)$ increment sequence is used and the mop-up requires less than $O(N\log N)$ time, then this algorithm runs in $O(N\log N)$. Incerpi tried various increment sequences on random permutations and found that for some sequences the mop up time was always zero (empirically). This led to the conjecture that Shaker-sort was $O(N\log N)$ for certain increment sequences. Moreover, Shaker-sort is easily converted into a network sorter. Finding a practical network sorter with $O(N\log N)$ boxes is a major open problem.

5.1. Network Sorting

In network sorting, we have comparators that take as input two numbers and output them in sorted order. Outputs from comparators can be connected to more comparators, but nothing else is allowed. It is not possible to tell for instance if a comparator has actually switched the elements it compared or not. Moreover, the placement of the comparators is fixed; at the start of the algorithm. This can be viewed as general sorting on a real computer that uses the following rules: 1) The permutation is stored $x_1, x_2, \cdots$ at all times, 2) The only operation allowed is to compare and exchange if necessary x_i and x_j, and 3) The set of such (i,j) is fixed at the start of the algorithm.

Efficient networks are known for small N, but for large N network sorting seems intrinsically harder then general sorting. Floyd [Knu73] showed that network merging of two sorted lists is $\Omega(N\log N)$, while in general merging can be done in $O(N)$, hence divide and conquer strategies can yield at best $O(N\log^2 N)$. Recently, the asymptotic result for network sorting has been lowered to $O(N\log N)$ [AKS83]. Unfortunately, the constant implied by the big-Oh notation is large; the best known practical networks run in $\approx \frac{1}{4} N\log^2 N$ time. Based on empirical evidence Shaker-sort seems to have the potential to be a practical $O(N\log N)$ network sort, and the recent results of [AKS83] suggest that a simple $O(N\log N)$ network sort exists.

5.2. The Shaker-Sort Algorithm

Shaker-sort works by performing k-shakes on a permutation; hence we define a k-shake. A k-shake is analogous to a k-sort, except that a k-shake doesn't necessarily k-sort. In fact, it generally doesn't k-sort unless it starts with a nearly k-sorted file. Algorithmically, a k-shake goes left to right comparing (and exchanging if necessary) x_i and x_{i+k}. After this is done, it goes right to left comparing x_i and x_{i-k}. Thus the time to take a k-shake is roughly $2N$.

Shaker-sort works by using an increment sequence $h_t, h_{t-1}, \cdots, h_1 = 1$ and doing an h_t-shake, h_{t-1}-shake, ... , 1-shake. As the file is not guaranteed to be sorted, we continue using 1-shakes. On a general computer, we use as many 1-shakes as required to sort the particular permutation. In a network, we have to use as many 1-shakes as required in the worst case. If the increment sequence is $O(\log N)$ in size, and $O(\log N)$ 1-shakes guarantee a sort, then the algorithm runs in $O(N\log N)$.

5.3. Empirical Evidence For Shaker-Sort

Incerpi ran tests for various increment sequences and was able to eliminate her new increment sequences from being possible candidates. These sequences worked for Shellsort because increment sequences with large common divisors in general work by ensuring that an element can't mover far during an insertion sort. Since only two comparisons are made for a particular element per pass, this condition is already ensured and the fact that these increments are not relatively prime makes them bad. The best sequences for an $O(N \log N)$ sort seem to be of the form $\lfloor \alpha^i \rfloor$. For increments of this type, Incerpi ran 10 permutations per file size per increment sequence to estimate the number of mop-up 1-shakes. Some of the results are below:

α	5000	10000	20000	40000
1.41	0	1	0	1
1.5	1	1	2	2
1.6	0	0	0	0
1.7	0	0	0	0
1.8	2	2	2	2
1.9	3	2	2	3

Other tests were performed relating to how many inversions were present after each pass. Further tests compared Shaker-sort's running time on a real computer to other sorting methods; the interested reader can consult [Inc85]. The important item is that for the increments of the form $\lfloor 1.7^i \rfloor$, after millions of runs for various file sizes, no permutation was found to require any mop-up work.

Our own tests have confirmed that for $N < 32$, Shaker-sort always sorts using this increment sequence. In fact, for $N \leq 11$, the sequence { 1, 2, 3 } suffices, for $N \leq 23$, { 1, 2, 3, 5 } suffices, and for $N \leq$ *at least* 32, { 1, 2, 3, 5, 9 } suffices.

5.4. Lower-Bounding Shaker-sort

We will use techniques similar to those used to lower bound Shellsort to show that Shaker-sort is not $O(N \log N)$ for any of the increment sequences suggested.

In testing network sorters, we can use 0-1 permutations in the same way we used to bound Shellsort. In fact we have the famous theorem:

Theorem (0-1 Principle): If a network with N input lines sorts all 2^N sequences of 0's and 1's into non-decreasing order, it will sort any arbitrary sequence of N numbers into non-decreasing order.

Proof: See Knuth [Knu73].

Thus we will only consider 0-1 permutations, and we will use the exact same permutations to lower bound Shaker-sort as we did to lower bound Shellsort. We now prove an easy lemma that will enable us to count how many inversions Shaker-sort can remove from the permutation in each pass.

Lemma 5: A k-shake removes at most kN inversions from a permutation of size N.

Proof: For 0-1 permutations it is easily observed that a 1-shake merely swaps the left-most 1 with the rightmost 0. For a k-shake, in each of the k subfiles, the left-most 1 is swapped with the rightmost 0. For each of these k swaps possible, at most N inversions can be removed because that is as far apart as two elements can be in a file of size N. Hence at most kN inversions can be removed. $\square$

For the permutations that have been used to bound Shellsort, Lemma 5 shows that Shaker-sort will have problems. We know that we will start out with $O(N^2)$ inversions, and that Shaker-sort will not be able to remove this with the early shakes. It will thus have to rely on the later k-shakes which can't remove as many inversions, since k is smaller. If the Frobenius number of the increments is $O(N^c)$, then the first shake that can actually swap elements is $O(N^{1/c})$, and the total amount of inversions that all the shakes smaller than this can remove is

$O(N^{1+1/c})$. Since there are $O(N^2)$ inversions to start with, when it comes to mop-up, there will still be $O(N^2)$ inversions. This means Shaker-sort will be doomed because the mop-up phase consists entirely of 1-shakes that can remove only a linear amount of inversions per shake. Since only a few increment sequences (the ones that are simple) seem to work well empirically for Shaker-sort anyway, rather than prove the above more formally, we show in the next section what happens to Shaker-sort when presented with the permutations that we expect to make it run slow.

5.5. Empirical Evidence Against Shaker-Sort

The easiest test to run on Shaker-sort is the test Incerpi ran -- try random files. If we start running Shaker-sort on files larger than Incerpi's, we can get examples where one extra mop-up 1-shake is required. For permutations of size $N=250000$, this happens about once every 75 tries (the sample space here is small). Of course, we are allowed $O(\log N)$ mop-up passes, so this still isn't bad. However, the algorithm's running time deteriorates when we use the specific permutations that we expect Shaker-sort to die on.

Using the $\lceil 1.7^i \rceil$ increment sequence throughout this discussion, we start by trying to find the smallest permutation that requires mop-up time. We know that this permutation has at least 32 elements, since an exhaustive search has been run. This should not be taken too positively, however, because the number of comparisons used to sort 32 elements is quite high. It turns out that the following permutation of 57 elements requires a mop-up pass:

57, 30, 18, 34, 29, 17, 33, 28, 16, 56, 27, 15, 32, 26, 14, 55, 25, 13, 54, 24, 12, 31, 23, 11, 53, 52, 10, 51, 22, 9, 50, 21, 8, 49, 48, 7, 47, 20, 6, 46, 45, 5, 44, 43, 4, 42, 19, 3, 41, 40, 39, 38, 37, 2, 36, 35, 1

If the permutation is written in 0-1 form, it can be seen as the start of the Frobenius pattern of { 9, 15, 25, 42 }. We can try extending this to get more mop-up passes.

Next, we run Shaker-sort on the 0-1 permutations that we have already discussed. The results are summarized below:

N	Mop-up 1-shakes
71	1
189	9
284	3
1026	62
1430	61
2988	83
8749	564
15580	878
27093	1293
49974	2299
96626	4286
184706	8233
417449	23473
734702	35956
1360732	63522
2480239	114561

We see very clearly that Shaker-sort isn't going to run in $O(N\log N)$ for these increments. One idea to try would be to put an extra k-shake near the start, where k is small. This would make it impossible to generate a long Frobenius pattern, and hence our permutation. Since, unlike Shellsort, we are guaranteed to only do linear work per pass, we can freely duplicate k-shakes and mix them up in any manner we want without worrying about counter-strategies that force all the work on a small early pass. Actually, it is hard to come up with a sound analytical reason why this idea wouldn't work, so we resort to simulation. What we have done is run Shaker-sort twice, and then do the mop-up. Thus, to sort $N=32$, the increments are in effect 25, 15, 9, 5, 3, 2, 1, 25, 15, 9, 5, 3, 2, 1, and as many more 1-shakes as needed. The results are summarized below.

N	Mop-up 1-shakes
71	0
189	0
284	0
1026	2
1430	0
2988	0
8749	277
15580	392
27093	483
49974	888
96626	1878
184706	4140
417449	16516
734702	24142
1360732	43425
2480239	80400

The extra run through Shaker-sort doesn't help much at all. Of course, we have duplicated shakes in the second pass. This might not be the best thing to do. What would happen if the second pass was slightly different from the first? One easy variant to try is using $(k-2)$-shakes in the second pass instead of k-shakes. Thus, to sort $N=32$, the increments are 25, 15, 9, 5, 3, 2, 1, 23, 13, 7, 3, 2, 1, 1, 1 and as many 1-shakes as needed. The results (on the same permutation that we have been using) are below:

N	Mop-up 1-shakes
71	0
189	0
284	0
1026	21
1430	0
2988	2
8749	292
15580	406
27093	480
49974	907
96626	1917
184706	4262
417449	16547
734702	24165
1360732	43460
2480239	80437

This actually does a little worse than just running two passes with the same increments. Yet another plan is to throw in 1-shakes between increments. For $N=32$, the sequence would be 25, 1, 15, 1, 9, 1, 5, 1, 3, 1, 2, 1, 1, 1, plus mop-up. Again, we summarize the results below:

N	Mop-up 1-shakes
71	0
189	4
284	0
1026	55
1430	53
2988	74
8749	554
15580	867
27093	1281
49974	2286
96626	4272
184706	8218
417449	23457
734702	35939
1360732	63504
2480239	114542

We still see no change, however if we intersperse a larger shake instead of a 1-shake, we can see some improvement (although not enough to make the algorithm useful). If h_k is the smallest increment used in the Frobenius pattern to construct the permutation, we will intersperse $h_k - 1$. Thus, for $N = 32$, the sequence would be 25, 4, 15, 4, 9, 4, 5, 4, 3, 4, 2, 4, 1, 4, plus mop-up. The results are below:

N	Mop-up 1-shakes
71	0
189	0
284	0
1026	0
1430	0
2988	0
8749	25
15580	14
27093	69
49974	136
96626	301
184706	780
417449	2103
734702	5029
1360732	7515
2480239	13617

While the number of mop-up passes is greatly reduced, it is still too high to make these increments useful. Further, if we form the permutation by including $h_k - 1$ in the Frobenius pattern, thus getting a smaller value of N, the reduction will be neutralized, since the early $h_k - 1$-shakes won't do anything. Thus, this doesn't seem to be a workable solution. One final attempt is when using the original increments, do an $h_k + 1$-sort, an h_k-sort, and an $h_k - 1$-sort instead of just an h_k-sort. Thus for $N = 32$, the sequence is in effect 26, 25, 24, 16, 15, 14, 10, 9, 8, 6, 5, 4, 4, 3, 2, 3, 2, 1, 2, 1, 1, plus mop-up. Below are the results are sorting with increments of this form:

N	Mop-up 1-shakes
71	0
189	0
284	0
1026	0
1430	0
2988	0
8749	0
15580	0
27093	0
49974	0
96626	0
184706	56
417449	9558
734702	12329
1360732	23331
2480239	46238

This seems to make mop-up take about one-third the time compared to the original increments (eventuallly), but of course there are three times as many increments.

In all these cases, we have tried variants of an increment sequence, and none of these variants come close to producing sorts requiring only a little mop-up work. Moreover, these variants fail on the same permutation - we don't even have to tailor-make new permutations for the new increment sequences.

5.6. Summary

We have shown by counter-example that Shaker-sort is hopeless as a network sorter for the specific increments suggested by Incerpi. The number of extra 1-shakes required seems to make the algorithm quadratic in the worst case, and simple variations of the increments don't make shaker-sort do significantly better. The average case is entirely different matter, especially for reasonably sized files. In addition to Incerpi's tests, we have run more tests on random permutations and can't produce counterexamples randomly for files on the order of 10-100 thousand elements. It may be the case that Shaker-sort is an excellent probabilistic sorter.

6. Conclusions

In this chapter, we will summarize the results of this thesis and list some of the many open problems that still remain.

6.1. Thesis Results

We look closely at the Frobenius pattern, marking representable numbers as 0 and unrepresentables as 1, and attempt to estimate the number of inversions in the pattern. In some cases, we are able to prove that the number of inversions is quadratic in the pattern length and in all the other cases that we investigate, the number of inversions seems to asymptotically approach $\frac{g^2}{24(k'-1)}$, where g is the Frobenius number and k' is the effective number of arguments. Thus if, for example, 10 arguments to the Frobenius function yield a pattern length quadratic in the smallest number (for instance, ten consecutive numbers do this), then the 10 arguments are essentially behaving as two. We conjecture that the number of inversions is always more than $\Omega(\frac{g^2}{k-1})$, where k is the number of (real) arguments. Our results on Shellsort are still obtainable with weaker assumptions.

We use the Frobenius pattern to generate bad permutations that make Shellsort perform slowly. We assign large numbers to the ones and small numbers to the zeros. Our permutation is then assumed by the Inversion Conjecture to have quadratic inversions, but it is also h_t-, h_{t-1}-, ... , $h_{t-(k-1)}$-sorted for the value of k that makes $g(h_{t-(k-1)}, h_{t-(k-2)}, \cdots, h_t) = N-1$. Thus the early passes are rendered useless. As this runs completely counter to Shellsort's strategy, we can then easily show (with the Inversion Conjecture) the lower bounds for Shellsort. These bounds in general match the upper bounds since the latter also depend heavily on the Frobenius function. We look at how these bad permutations perform on a real computer and then show a very general lower bound for Shellsort that matches the best known upper bound (for $O(\log N)$ increment sequences). This lower bound suggests that no "almost geometric" increment sequence will make Shellsort run faster than those of Incerpi and Sedgewick. Here, "almost geometric" means that the increments are $O(\alpha^i)$, as are virtually all analyzed $O(\log N)$ increment sequences. Thus an $O(N\log N)$ Shellsort would seem to require a completely different approach.

Finally, we look at a variant of Shellsort, Shaker-sort. Shaker-sort is a network sorting algorithm that was empirically shown by Incerpi to run well for some increment sequences on random data. The key to its running time is how much mop-up work it needs to do to finish the sort. We are able to make it take huge amounts of time to sort by feeding it the exact same permutations that lower bound Shellsort. While for particular increments, Incerpi was not able to find any permutations (looking at millions as big as 130K) that required any mop-up work at all, we find permutations as small as 57 elements that require some mop-up and permutations of 2 million elements that require a hundred thousand mop-up passes (almost certainly a linear number of mop-up passes in general). We conclude that Shaker-sort is quadratic in the worst case for increment sequences that were thought to produce an $O(N\log N)$ sort.

6.2. Open Problems

While our results suggest that neither Shellsort nor Shaker-sort is likely to run in $O(N\log N)$ worst-case time, there are still several interesting questions remaining.

First of all, the Inversion Conjecture is exactly that -- a conjecture. Can we prove the lower bound on the number of inversions to be $\Omega(g^2/k)$, or equivalently for Shellsort purposes, something slightly less than $\Omega(g^2/2^k)$? One approach is to show that in the top half of the pattern, at least one of every $O(k)$ elements is a one. This would lead to an $\Omega(g^2/k^2)$ bound. Another approach might be to show that $g-n$ (the number of representables) is $\Omega(g)$. It seems, empirically, that $n \approx \frac{g}{2}$ almost always. If the Inversion Conjecture can be proved, it is probably easy to prove that the center of the pattern also has a quadratic number of inversions (relative to the size of the center).

Assuming the Inversion Conjecture is proved, and thus all the bounds are true, can it be proven that even if only some subset of increments is $O(\alpha^i)$ in some order, then Shellsort is not $O(N\log N)$. Another interesting conjecture is that uniform and non-uniform increment sequences are asymptotically equivalent. Can it be shown that any bound obtained with a non-uniform sequence is obtainable with a uniform sequence? If this was proven, it would almost certainly kill prospects for an $O(N\log N)$ sort since it seems difficult to design a uniform $O(\log N)$ increment sequence that doesn't have "almost geometric" increments. Also, the question of whether or not Shellsort can be $O(N\log N)$ on average is still open. From a practical point of view, the new sequences in Chapter 2 may outperform the commonly used sequences and thus it seems like some empirical studies ought to be done for them.

As for Shaker-sort, we have an algorithm that seems to take $O(N\log N)$ for random files, but is disastrous on bad permutations. We also start to see mop-up passes on random 250K permutations, so perhaps even larger random files don't work well either (we need more computer cycles to answer this). In this sense, it performs much like Shellsort: good for random, mid-sized files and good for nearly sorted files -- bad worst case. Shaker-sort seems to have potential as a probabilistic sorter for mid-sized permutations. Can we show that it sorts with no mop-up almost all of the time? This seems like a very difficult question since it would appear the for large files, bad things may start to happen even if the permutations are random. The average case performance of Shaker-sort doesn't seem like a pressing issue, but could likely be answered if its performance as a probabilistic sorter was analyzed.

Finally, we note that the door on an $O(N\log N)$ worst-case Shellsort is not entirely closed. Chazelle's increments which turn out to have the same worst-case performance as any $O(\alpha^i)$ sequence isn't an $O(\alpha^i)$ sequence itself. It is possible, although we think unlikely, that an $O(N\log N)$ Shellsort might still exist, using a very clever increment sequence.

7. References

[AKS83] M. Ajtai, J. Komlos, E. Szemeredi, "An $O(n\log n)$ Sorting Network" *Proceedings 15th Annual ACM Symposium of Theory of Computing,* Boston 1983, 1-9.

[Boe55] H. Boerner, *Darstellung von Gruppen,* Springer-Verlag, Berlin (1955).

[Bra42] A. Brauer, "On a Problem of Partitions", *American J Mathematics* **64** (1942), 299-312.

[Gon84] G. Gonnet, *Handbook of Algorithms and Data Structures,* Addison-Wesley, 1984.

[Gre80] H. Greenberg, "An algorithm for a linear Diophantine equation and a problem of Frobenius", *Numerische Mathematik* **34** (1980), 349-352.

[Hib63] T.N. Hibbard, "An empirical study of minimal storage sorting", *Communications of the ACM* **6** 5(1963), 206-213.

[Inc85] J. Incerpi, "A Study of the Worst-Case of Shellsort", Ph.D. Thesis, Brown University, 1985.

[InS83] J. Incerpi and R. Sedgewick, "Improved Upper Bounds on Shellsort", *Proceedings 24th Annual Symposium on Foundations of Computer Science,* Tucson 1983, 48-55.

[Joh60] S.M. Johnson, "A linear diophantine problem", *Canadian J. Math.* **12** (1960), 390-398.

[Knu73] D.E. Knuth, *The Art of Computer Programming. Volume 3: Sorting and Searching,* Addison-Wesley, Reading, Mass. (1973).

[NiW72] A. Nijenhuis and H.S. Wilf, "Representations of integers by linear forms in nonnegative integers", *J. Number Theory* **4** (1972), 98-106.

[PaS65] A.A. Papernov and G.V. Stasevich, "A method of information sorting in computer memories", *Problems of Information Transmission* **1** 3(1965), 63-75.

[Pra71] V. Pratt, *Shellsort and Sorting Networks,* Garland Publishing, New York (1979). (Originally presented as the author's Ph.D. thesis, Stanford University, 1971.)

[Sed86] R. Sedgewick, "A New Upper Bound for Shellsort", *J. of Algorithms* **2** (1986), 159-173.

[Sel77] E.S. Selmer, "On the linear diophantine problem of Frobenius", *J. reine angew. Math.* **294** (1977), 1-17.

[Sel87] E.S. Selmer, "On Shellsort and the Frobenius problem", *unpublished manuscript.*

[Sh884] W.J. Curran Sharp, Solution to Problem 7382 (Mathematics), *Educational Times,* London (1884).

[She59] D.L. Shell, "A high-speed sorting procedure", *Communications of the ACM* **2** 7(1959), 30-32.

[Tem83] B. Temkin, "On a Linear Diophantine Problem of Frobenius for Three Variables", Ph.D. Thesis, City University of New York, 1983.

8. Appendix

Listed below are some of the increment sequences that have been discussed in this thesis.

Hibbard's Increments $(2^i - 1)$

1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, 4095, 8191, 16383, 32767, 65535, 131071, 262143, 524287, 1048575, 2097151, 4194303, ...

Pratt's Increments $(2^i 3^j)$

1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 32, 36, 48, 54, 64, 72, 81, 96, 108, 128, 144, 162, 192, 216, 243, 256, 288, 324, 384, 432, 486, 512, 576, 648, 729, 768, 864, 972, 1024, 1152, 1296, 1458, 1536, 1728, 1944, 2048, 2187, 2304, 2592, 2916, 3072, 3456, 3888, 4096, 4374, 4608, 5184, 5832, 6144, 6561, 6912, 7776, 8192, 8748, 9216, 10368, 11664, 12288, 13122, 13824, 15552, 16384, 17496, 18432, 19683, 20736, 23328, 24576, 26244, 27648, 31104, 32768, 34992, 36864, 39366, 41472, 46656, 49152, 52488, 55296, 59049, 62208, 65536, 69984, 73728, 78732, 82944, 93312, 98304, 104976, 110592, 118098, 124416, 131072, 139968, 147456, 157464, 165888, 177147, 186624, 196608, 209952, 221184, 236196, 248832, 262144, 279936, 294912, 314928, 331776, 354294, 373248, 393216, 419904, 442368, 472392, 497664, 524288, 531441, 559872, 589824, 629856, 663552, 708588, 746496, 786432, 839808, 884736, 944784, 995328, 1048576, 1062882, 1119744, 1179648, 1259712, 1327104, 1417176, 1492992, 1572864, 1594323, 1679616, 1769472, 1889568, 1990656, 2097152, 2125764, 2239488, 2359296, 2519424, 2654208, 2834352, 2985984, 3145728, 3188646, 3359232, 3538944, 3779136, 3981312, 4194304, ...

Sedgewick's Increments I ($4 \cdot 4^i + 3 \cdot 2^i + 1$)

1, 8, 23, 77, 281, 1073, 4193, 16577, 65921, 262913, 1050113, 4197377, ...

Sedgewick's Increments II ($(2^i - 3)(2^{i+1} - 3)$)

1, 5, 65, 377, 1769, 7625, 31625, 128777, 519689, 2087945, ...

Sedgewick's Increments III (merge of $4^i - 3 \cdot 2^i + 1$ and $9 \cdot 4^i - 9 \cdot 2^i + 1$)

1, 5, 19, 41, 109, 209, 505, 929, 2161, 3905, 8929, 16001, 36289, 64769, 146305, 260609, 587521, 1045505, 2354689, 4188161, ...

Largest Prime Smaller Than 2^i

1, 2, 3, 7, 13, 31, 61, 127, 251, 509, 1021, 2039, 4093, 8191, 16381, 32749, 65521, 131071, 262139, 524287, 1048573, 2097143, 4194301

Smallest Prime Greater Than 2^i

1, 2, 5, 11, 17, 37, 67, 131, 257, 521, 1031, 2053, 4099, 8209, 16411, 32771, 65537, 131101, 262147, 524309, 1048583, 2097169, 4194319

Incerpi and Sedgewick Increments (Sec 2.3.3)

A base sequence a_1, a_2, ... is used with the condition that the a_is are relatively prime. An increment is formed by multiplying c different terms of the base sequence with the condition that the base sequence terms used must be consecutive, except for at most one point, where you can skip an element of the base sequence. Thus, with $c = 3$, the increments are of the form $a_i a_{i+1} a_{i+2}$, $a_i a_{i+1} a_{i+3}$, or $a_i a_{i+2} a_{i+3}$. As we noted in Section 2.3.3, if the sequence is formed by multiplying c consecutive increments, and not allowing the one skip, you can still prove the same asymptotic result. For $c=3$, and using a_i = the largest prime less than or equal to 2^i, the sequence suggested by Incerpi and Sedgewick becomes

1, 6, 14, 21, 42, 78, 182, 273, 651, 1209, 2821, 5551, 13237, 24583, 51181, 100711, 240157, 474641, 988181, 1944497.

The sequence

1, 6, 42, 273, 2821, 24583, 240157, 1944497 yields the same asymptotic result, but performs poorly in practice because the increments are $O(8^i)$. If we choose a_i = the largest prime less than $2^{1/3}$, the increments are $O(2^i)$. There seems to be a lot of empirical work do be done along these lines.

Incerpi and Sedgewick Increments (sec 2.3.4)

Again, a base sequence a_1, a_2, ... is used with the condition that the a_is are relatively prime. An increment is formed by multiplying different terms of the base sequence with the condition that the base sequence terms must be consecutive, except for at most one point, where you can skip an element of the base sequence, and the additional condition that the first term in the multiplication is always a_1. Note that there is no restriction on how many consecutive terms can be multiplied, *per se*, although it is dependent on how big N is. Thus, the sequence essentially is a_1, a_1a_2, a_1a_3, $a_1a_2a_3$, $a_1a_2a_4$, $a_1a_3a_4$, $a_1a_2a_3a_4$, $a_1a_2a_3a_5$, $a_1a_2a_4a_5$, $a_1a_3a_4a_5$, ... Multiplying 1 term, then 2 terms, etc. builds the increment sequence in increasing order, thus it is easy to construct the increments by hand. If we choose a_i to be the smallest prime greater than or equal to 2^i, the sequence generated is

1, 2, 5, 10, 22, 55, 110, 170, 374, 935, 1870, 4070, 6290, 13838, 34595, 69190, 125290, 272690, 421430, 927146, 2317865.

Shaker-sort Increments ($\left\lceil 1.7^i \right\rceil$)

1, 2, 3, 5, 9, 15, 25, 42, 70, 119, 202, 343, 583, 991, 1684, 2863, 4867, 8273, 14064, 23908, 40643, 69092, 117457, 199676, 339449, 577063, 981007, 1667712, 2835109, ...