Best Increments for the Average Case of Shellsort

Marcin Ciura

Department of Computer Science, Silesian Institute of Technology,
Akademicka 16, PL–44–100 Gliwice, Poland

Abstract. This paper presents the results of using sequential analysis to find increment sequences that minimize the average running time of Shellsort, for array sizes up to several thousand elements. The obtained sequences outperform by about 3% the best ones known so far, and there is a plausible evidence that they are the optimal ones.

1 Shellsort

A well implemented Shellsort is among the fastest general algorithms for sorting arrays of several dozen elements, and even for huge arrays it is not prohibitively slow. Moreover, it is an *adaptive* method that runs faster on "nearly sorted" arrays that often occur in practice. Published by D. L. Shell in 1959 [11], it is one of the earliest sorts discovered, it can be easily understood and implemented, yet its analysis is difficult and still incomplete.

Shellsort for N elements $X[0, \ldots, N-1]$ is based on a predetermined sequence of integer *increments* $0 < h_0, \ldots, h_{t-1} < N$, where $h_0 = 1$. In general, the increment sequences can change with N, but customarily initial elements of some infinite sequence are used for simplicity.

The algorithm performs t *passes* over the array: one pass for each increment from h_{t-1} down to h_0. The pass number $t-k$ sorts by straight insertion all the subarrays that consist of elements h_k apart: $X[0, h_k, \ldots]$,$X[1, h_k+1, \ldots]$, $\ldots$, $X[h_k-1, 2h_k-1, \ldots]$. This way each pass involves sorting subarrays that are either small or nearly in order, and straight insertion sort performs well in these circumstances.

The number of operations made by the algorithm depends on the increment sequence, and indeed many sequences have been proposed and used. References [7,10] contain broad surveys of previous research on Shellsort. Theoretical analysis of its running time is, however, confined to worst case. The average running time was susceptible to analysis only in cases that do not cover the sequences used in practice [2,5,14]. Also, sequences of increments that minimize the average running time of Shellsort were not known so far.

2 Sequential Analysis

Sequential analysis is a method of verifying statistical hypotheses developed by Abraham Wald in the 1940s [13]. Whereas classical statistical criteria fix the

R. Freivalds (Ed.): FCT 2001, LNCS 2138, pp. 106–117, 2001.

size of the random sample before it is drawn, in sequential analysis its size is determined dynamically by analyzing a sequentially obtained series of data.

Sequential analysis has been employed in fields, where sampling is costly, for example drug investigation and destructive qualification testing of goods. We use it to determine in reasonable time the best increments for the average case of Shellsort. For example, suppose that we are interested in minimizing C, the number of comparisons made by a five-increment Shellsort when sorting 128 elements. A good fairy tells us that there are only a few dozen sequences, for whose the average number of comparisons EC is less than 1005, and the distribution of C for all of them, being inherently discrete, can be approximated by the normal distribution with a standard deviation $SC \approx 34$. After all, $EC < 1005$ surely causes SC to be $\leq \sigma_{\max} = 40$.

Using this information, we can construct a sequential test that acts like a low-band filter and allows to shorten the time of computations by a factor of thousands. We are willing to accept a good sequence when its $EC < \theta_0 = 1005$ and reject a bad one when, say, $EC > \theta_1 = 1015$. We consent to accidental rejecting a good sequence with probability $\alpha = 0.01$ and accidental accepting a bad one with probability $\beta = 0.01$. With each sequence probed, we are running Shellsort on randomly generated permutations and summing c_i, the number of comparisons made in the ith trial. We prolong the test as long as

$$a_k = \frac{\sigma_{\max}^2}{\theta_1 - \theta_0} \ln \frac{\beta}{1-\alpha} + k\frac{\theta_0 + \theta_1}{2} \leq \sum_{i=1}^{k} c_i \leq \frac{\sigma_{\max}^2}{\theta_1 - \theta_0} \ln \frac{1-\beta}{\alpha} + k\frac{\theta_0 + \theta_1}{2} = r_k.$$

If the sum is less than a_k, the sequence is accepted; if it is greater than r_k, the sequence is rejected; if it is between a_k and r_k, k gets incremented, a_k and r_k are adjusted, and another trial is made.

The sequences that passed the test have biased estimate of the average number of comparisons, so it has to be evaluated again on some number of independent permutations, but the vast majority of sequences (those with large EC) is rejected in the test after just a few trials. Fig. 1 shows the mean and standard deviation of the number of comparisons made by Shellsort using 790 sequences that passed the test described above, evaluated subsequently in 10000 trials.

To make the sequential test fast, it is essential to choose θ_0 near the actual minimal EC. To estimate EC and SC of the best sequences in default of a fairy, we can use a random sample of sequences that begin with increments known to be good (more on good increments below).

If our assumptions are true, we should have missed no more than 1% of sequences with $EC < 1005$. In fact, the distribution of the number of comparisons is not normal, but skewed (see Fig. 2 for a typical example). In an attempt to compensate this asymmetry, the chosen value of $\sigma_{\max}$ is greater than the actual standard deviations. Moreover, by the Central Limit Theorem, the sum of c_i's obtained in independent trials tends to the normal distribution.

This reasoning can seem fallacious, as it involves knowing in advance, what we are looking for [1], but in fact it is not. If there exists a sequence with $EC < \theta_0$ and $SC > \sigma$, the probability that it was accidentally rejected is less than 1/2

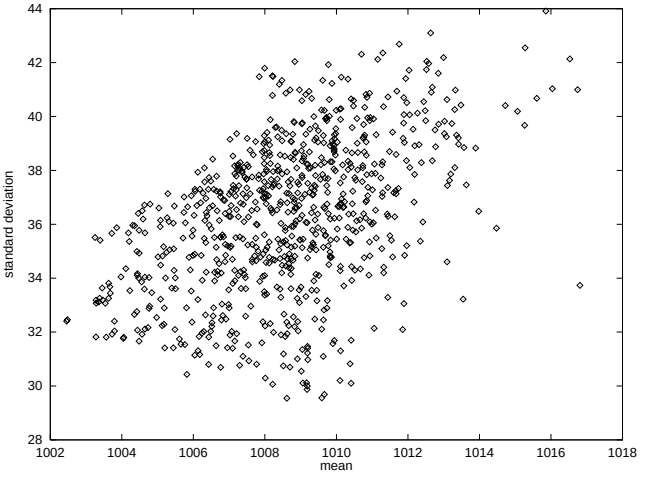

Fig. 1. Mean and standard deviation of the number of comparisons in Shellsort. Five-increment sequences for sorting 128 elements that passed the sequential test are shown

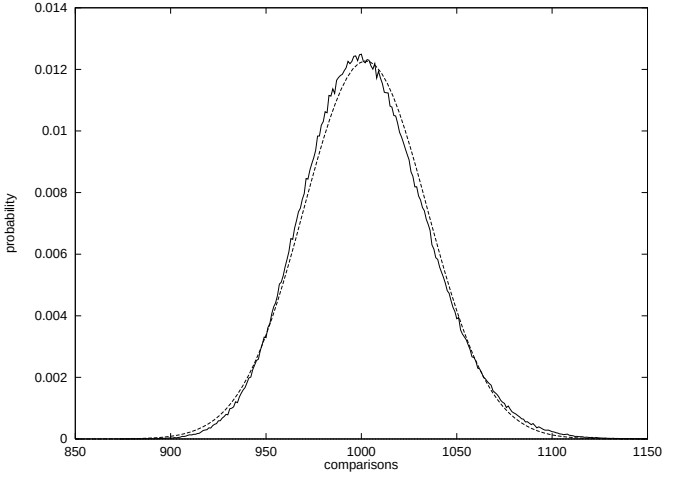

Fig. 2. Distribution of the number of comparisons in Shellsort using the sequence $(1, 4, 9, 24, 85)$ for sorting 128 elements (*solid line*), and the normal distribution with the same mean and standard devistion (*dashed line*)

in the symmetrical distribution model, and still less than one for any skewed distribution. Therefore, when the search is independently run several times, the sequence should pass the test at least once. It never happened in author's search: the standard deviation of the best sequences was always similar.

3 The Dominant Operation in Shellsort

Almost all studies of Shellsort treat its running time as proportional to the number of element moves. It is probably because the number of moves can be expressed in terms of the number of permutation inversions, and there are known techniques for inversion counting. These techniques give satisfactory answers for algorithms like insertion sort, where the ratio of the number of comparisons to the number of moves approaches 1 quickly.

In Shellsort, the picture is different. Figures 3 and 4 show the average number of computer cycles per move and per comparison for $10 \le N \le 10^8$. They concern Knuth's implementation of Shellsort for his mythical computer MIX and several increment sequences: Hibbard's $(1, 3, 7, 15, \ldots)$ [3], Knuth's $(1, 4, 13, 40, \ldots \mid 2h_k < N)$ [7], Tokuda's $(h_k = \lceil (9(\frac{9}{4})^k - 4)/5 \rceil \mid \frac{9}{4}h_k < N)$ [12], Sedgewick's $(1, 5, 19, 41, \ldots)$ [9], Incerpi-Sedgewick $(1, 3, 7, 21, \ldots)$ [4], and $(1, 4, 10, 23, 57, 132, 301, 701)$ (up to $N = 4000$) [see below]. Knuth's discussion assumes that the running time can be approximated as $9\times$number of moves, while Figures 3 and 4 show that for each sequence the number of key comparisons is a better measure of the running time than the number of moves. The asymptotic ratio of 9 cycles per move is not too precise for $N \le 10^8$, and, if some hypothetical sequence makes $\Theta(N \log N)$ moves, it is never attained. Analogous plots for other computer architectures would lead to the same conclusion.

Treating moves as the dominant operation leads to mistaken conclusions about the optimal sequences. Table 1 leads us to believe that the move-optimal sequence is Pratt's one $(1, 2, 3, 4, 6, 8, 9, 12, \ldots) = \{2^p 3^q\}$ [8], with $\Theta(\log^2 N)$ passes. However, the best practical sequences known so far are approximately geometrical, so they make $\Theta(\log N)$ passes. Also, a recent result [6] is that if there is a sequence that yields Shellsort's average running time $\Theta(N \log N)$, it has to make precisely $\Theta(\log N)$ passes.

Compare-optimal sequences seem to make $\Theta(\log N)$ passes. It is illustrated in Tables 1 and 2 that show the best sequences of various length for sorting 128 elements, obtained in the above described way with respect to the average number of moves and comparisons. In Table 1, there are no empirical sequences with more than 12 increments, since finding them would take too much time, but hopefully the point is clear. In Table 2, the difference between the last two lines is within possible error; sequences with more than six increments do not improve the results.

In fact, concentrating on the number of moves, we can obtain sequences that are good for practical purposes, but we have to guess an appropriate number of passes for a given N, lest they be too 'stretched' or too 'squeezed.' When

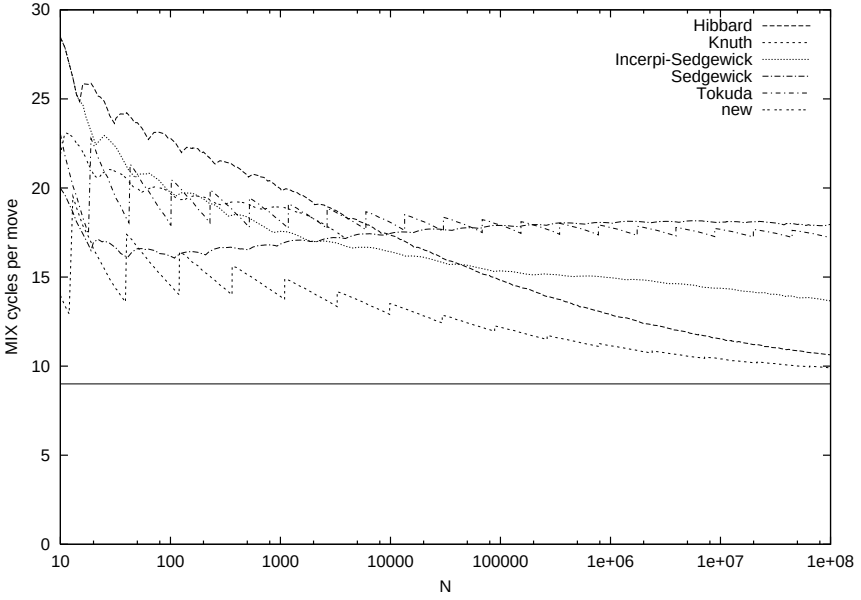

Fig. 3. Average MIX computer time per move in Shellsort using various sequences of increments

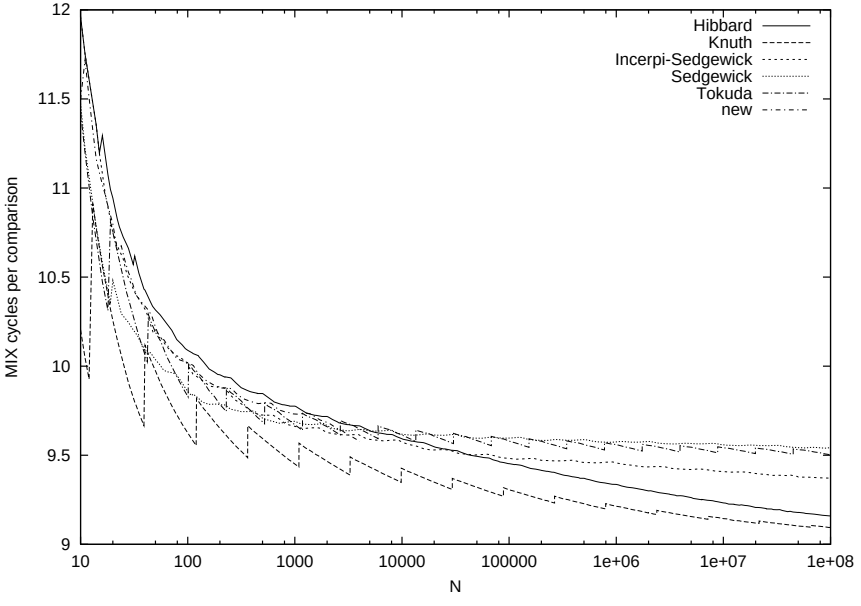

Fig. 4. Average MIX computer time per comparison in Shellsort using various sequences of increments

Table 1. Move-optimal sequences for sorting 128 elements

Increments	Moves	Comparisons	MIX time
1	4064.0	4186.8	37847
1 8	1280.2	1506.2	13958
1 4 17	762.80	1090.69	10422.4
1 4 9 24	588.25	1018.74	9956.9
1 3 7 15 35	506.56	1032.39	10256.0
1 3 7 12 20 51	458.18	1071.99	10761.8
1 3 4 10 15 28 47	427.43	1151.14	11625.6
1 2 5 7 13 22 33 56	405.20	1220.71	12393.3
1 3 4 6 11 18 26 35 56	389.36	1308.48	13323.4
1 2 3 5 8 12 18 27 41 75	375.70	1390.80	14301.2
1 2 3 5 8 12 18 27 38 59 84	365.83	1440.45	14717.1
1 2 3 4 6 11 14 18 27 37 62 86	357.63	1545.17	15793.3
.....			
1 2 3 4 6 8 9 12 16 18 ... 96 108	338.08	2209.59	22700.4

Table 2. Compare-optimal sequences for sorting 128 elements

Increments	Comparisons	Moves	MIX time
1	4186.8	4064.0	37847
1 9	1504.6	1280.7	13945
1 4 17	1090.69	762.80	10422.4
1 4 9 38	1009.75	598.90	9895.0
1 4 9 24 85	1002.22	538.06	9919.9
1 4 9 24 85 126	1002.25	535.71	9933.2

working with comparisons, at least choosing the number of passes too high does no harm.

4 Further Enhancements to the Method

The investigations were begun for small N and a broad range of sequences with $2 \le h_1 \le 10$ and $1 < h_k/h_{k-1} < 4$ for $k > 1$. It turns out that the best sequences had $h_1 \in \{4, 5\}$ and $h_k/h_{k-1} \in (2, 3)$, for $k > 0$; except perhaps the last increments, where a larger value of h_k/h_{k-1} is sometimes better. Also, having $h_{k+1} \bmod h_k = 0$ is always a hindrance.

The smallest increments are stable among the best sequences with a maximal t for various N. Indeed, for N greater than a few dozen, the best sequences are $(1, 4, 9, \ldots)$, $(1, 4, 10, \ldots)$, $(1, 4, 13, \ldots)$, $(1, 5, 11, \ldots)$. A few other beginnings are also not bad, yet consistently worse than these four. The increment h_3 crystallizes when $N \approx 100$, and the top sequences are $(1, 4, 9, 24, \ldots)$, $(1, 4, 9, 29, \ldots)$, $(1, 4, 10, 21, \ldots)$, $(1, 4, 10, 23, \ldots)$, and $(1, 4, 10, 27, \ldots)$.

As N grows, the feasible values of the remaining increments show up, too: given $(h_0, \ldots, h_{k-1})$, there is always a small set of good values for h_k. Figures 5 and 6 show the average number of comparisons made by sequences beginning with $(1, 4, 10, 23)$ and $(1, 4, 10, 21)$ when sorting 300 and 1000 elements as a function of h_4.

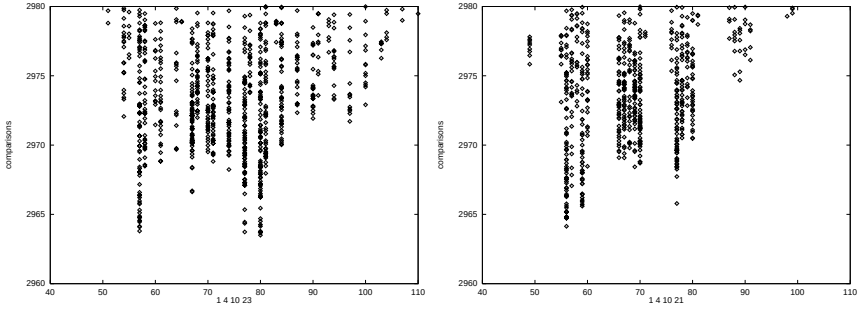

Fig. 5. Performance of Shellsort that uses the sequences $(1, 4, 10, 23, h_4, h_5)$ and $(1, 4, 10, 21, h_4, h_5)$ for $N = 300$ depending on h_4

Thus, we can speed the search up, considering only the sequences beginning with these increments, and imposing more strict conditions on the remaining increments. The author would like to stress that he took a conservative approach and checked a much wider fan of sequences. It was seeing some patterns consistently losing for several N's that encouraged him not to consider them for a greater number of elements.

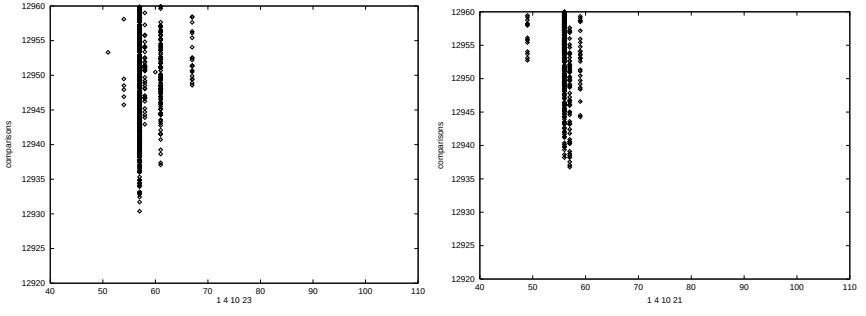

Fig. 6. Performance of Shellsort that uses the sequences $(1, 4, 10, 23, h_4, h_5, h_6)$ and $(1, 4, 10, 21, h_4, h_5, h_6)$ for $N = 1000$ depending on h_4

5 The Results

The size of this paper limits the author to present only a digest of his numerical results. Table 3 shows the best sequences of length 6–8 for sorting 1000 elements, and, for some increments, the best sequences beginning with them. The omitted sequences differ only with the biggest increments from those shown. However, there is a chance that some sequences that should be on these lists were accidentally rejected in the sequential test.

Table 3. The best 6-, 7-, and 8-pass sequences for sorting 1000 elements

Increments	Comparisons
1 4 13 32 85 290	13059.0 ± 195.9
1 4 13 32 85 284	13060.4 ± 196.3
1 5 11 30 81 278	13061.5 ± 198.2
1 5 11 30 81 277	13063.1 ± 201.2
1 4 10 23 57 156 409	12930.4 ± 157.5
1 4 10 23 57 155 398	12931.7 ± 157.5
1 4 10 23 57 156 401	12932.4 ± 157.6
(21 seq. omitted)	
1 4 10 21 57 143 390	12936.8 ± 157.9
(14 seq. omitted)	
1 4 10 21 56 125 400	12938.5 ± 157.0
(22 seq. omitted)	
1 4 9 24 58 153 396	12940.3 ± 158.9
1 4 10 23 57 156 409 995	12928.9 ± 158.1
1 4 10 23 57 156 409 996	12929.0 ± 157.2
1 4 10 23 57 155 393 984	12929.1 ± 156.9
(98 seq. omitted)	
1 4 10 21 56 135 376 961	12931.9 ± 155.8
(18 seq. omitted)	
1 4 10 21 57 143 382 977	12932.1 ± 156.2
(735 seq. omitted)	
1 4 10 23 61 154 411 999	12936.6 ± 159.9
(366 seq. omitted)	
1 4 10 23 58 135 388 979	12937.9 ± 155.5
(278 seq. omitted)	
1 4 9 24 58 153 403 991	12938.6 ± 158.1

As the greatest increments play a minor role in the overall performance of a sequence, the author abandoned the idea of finding truly optimal sequences up to the greatest increment for greater N, and concentrated instead on finding the feasible values of the middle increments.

To this end, 6-increment beginnings perform best for $N = 1000, 2000, 4000$ were selected. For each of them, all the sequences with $h_6 \in (2h_5, 3h_5)$ and

$h_7 \in (2h_6, 3h_6)$ were generated. For each of the 8-increment beginnings obtained, 100 random endings $h_8 \in (2h_7, 3h_7)$, $h_9 \in (2h_8, \min(3h_8, 8000))$ were drawn. The sequential test was then run for each 10-increment sequence for $N = 8000$. The percentage of (h_8, h_9) endings that passed the test was recorded for each $(h_0, \ldots, h_7)$. The top 8-increment beginnings were re-examined in 10000 independent tests.

Table 4. The best 8-increment beginnings of 10-pass sequences for sorting 8000 elements

Increments	Ratio passed
1 4 10 23 57 132 301 758	0.6798
1 4 10 23 57 132 301 701	0.6756
1 4 10 21 56 125 288 717	0.6607
1 4 10 23 57 132 301 721	0.6573
1 4 10 23 57 132 301 710	0.6553
1 4 9 24 58 129 311 739	0.6470
1 4 10 23 57 132 313 726	0.6401
1 4 10 21 56 125 288 661	0.6335
1 4 10 23 57 122 288 697	0.6335

From limited experience with yet larger array sizes, the author conjectures that the sequence beginning with $1, 4, 10, 23, 57, 132, 301, 701$ shall turn up optimal for greater N.

6 Why Are Some Sequences Better than Others

It seems that some sequences are better than others on the average not only due to a good global ratio h_{k+1}/h_k, but also because they cause less redundant comparisons, that is comparisons between elements that have already been directly or indirectly compared, which depends on local interactions between passes. Tables 5 and 6 show the average number of comparisons C_i and redundant comparisons R_i in each pass for two increment sequences.

Some heuristic rules on good sequences can be deduced from the observation that the subarrays sorted in each pass are quite well ordered and the elements move just a few h on the average in each h-sorting, as exemplified in Fig. 7.

Let's consider h_{k+1} expressed as a linear combination with integer coefficients of h_{k-1} and h_k. Patterns like these on Figures 5 and 6 can be forecasted to some extent using the following rule: if there is a solution of the Diophantine equation $h_{k+1} = ah_k + bh_{k-1}$ with a small value of $|b|$, say ≤ 5, then this h_{k+1} is certainly not a good choice. The value of b in analogous expressions with small multiples of h_{k+1} on the left side is nearly as much important.

The distribution of the distance travelled in a given pass is similar for all the elements far enough from the ends of the table. The order of two elements that are h_{k+1} apart after h_{k+1}-sorting is known. In subsequent pass both elements move

Table 5. Average number of comparisons and redundant comparisons for Shellsort using the sequence $(1, 4, 10, 23, 57, 132, 301, 701, 1750)$, for $N = 40, 400, 4000$

h_k	C_i	R_i
23	17	0
10	41.3 ± 2.6	0
4	62.4 ± 5.5	3.3 ± 2.3
1	90.8 ± 10.5	25.0 ± 7.6
Σ	211.5 ± 12.1	28.3 ± 8.7

h_k	C_i	R_i
301	99	0
132	334 ± 6	0
57	543 ± 14	0
23	720 ± 27	2.6 ± 2.3
10	792 ± 38	41 ± 14
4	809 ± 33	158 ± 24
1	976 ± 41	354 ± 33
Σ	4274 ± 73	557 ± 51

h_k	C_i	R_i
1750	2570 ± 10	0
701	5200 ± 40	0
301	6740 ± 70	0
132	7720 ± 110	7 ± 4
57	8410 ± 180	90 ± 20
23	9020 ± 190	370 ± 50
10	8570 ± 130	840 ± 60
4	8310 ± 100	1880 ± 90
1	9830 ± 140	3700 ± 120
Σ	66370 ± 430	6890 ± 220

Table 6. Average number of comparisons and redundant comparisons for Shellsort using Knuth's sequence $(1, 4, 13, 40, 121, 364, 1093)$, for $N = 40, 400, 4000$

h_k	C_i	R_i
13	36.4 ± 1.9	0
4	74.0 ± 7.1	0.02 ± 0.2
1	98.4 ± 13.0	22.7 ± 9.2
Σ	208.8 ± 15.5	22.7 ± 9.2

h_k	C_i	R_i
121	401 ± 7	0
40	713 ± 21	0
13	984 ± 51	53 ± 16
4	1223 ± 116	164 ± 54
1	1092 ± 47	376 ± 44
Σ	4413 ± 153	593 ± 81

h_k	C_i	R_i
1093	4480 ± 30	0
364	7430 ± 50	0
121	9850 ± 170	560 ± 40
40	11930 ± 310	1470 ± 120
13	14860 ± 930	2160 ± 280
4	15050 ± 770	3530 ± 460
1	11020 ± 40	4100 ± 100
Σ	74620 ± 1800	11510 ± 780

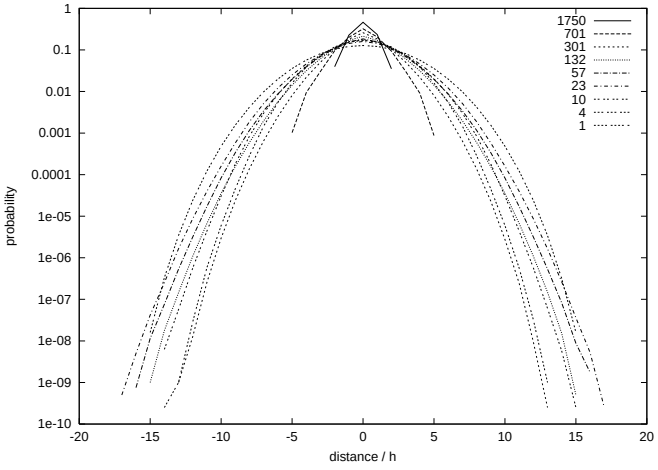

Fig. 7. Distance travelled by elements in subsequent passes of Shellsort using the sequence $(1, 4, 10, 23, 57, 132, 301, 701, 1750)$ for sorting 4000 elements

a few h_k to the left or to the right, and then their distance is $d = h_{k+1} + ah_k$. If d turns out to be a multiple of h_{k-1}, then they are redundantly compared again in subsequent h_{k-1}-sorting.

Unfortunately, the interdependence between the quality of a sequence and the solutions of equations $h_{k+1} = a_0 h_k + \ldots + a_l h_{k-l}$ becomes more and more obscure as l grows. However, there is some evidence that in good sequences the norm of the shortest vector-solution $(a_0, \ldots, a_l)$ for fixed l asymptotically grows as we move on to greater increments. It seems to exclude the possibility that the optimal uniform sequence can be defined by a linear recurrence with constant coefficients or by interleaving such sequences. See Fig. 8 for a plot of average number of comparisons made by Shellsort using various increment sequences. Both Knuth's and Hibbard's sequences are relatively bad, because they are defined by simple linear recurrences. The Sedgewick's sequence that consists of two interleaved sequences defined by linear recurrences, also becomes to deteriorate for $N > 10^6$.

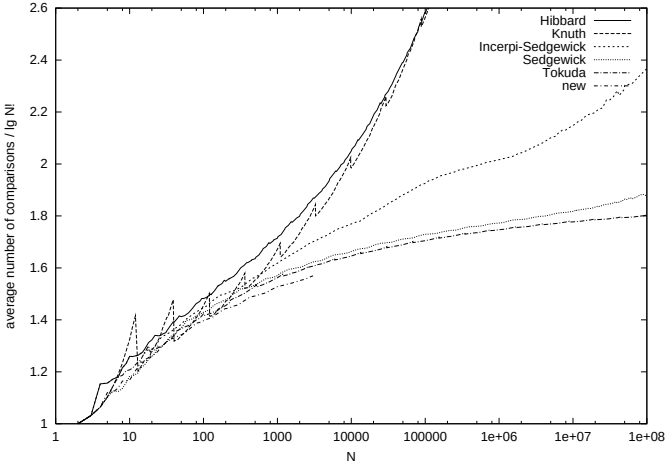

Fig. 8. Average number of comparisons divided by $\lg N!$ for Shellsort using selected increment sequences

7 Summary

Using sequential analysis, the search for optimal increment sequences for Shellsort was accelerated enough to find them for arrays up to several thousand elements. The obtained results show that comparisons rather than moves should be considered the dominant operation in Shellsort. It was also possible to state some heuristic rules about good sequences of increments.

However, the sequences obtained so far are too short to draw a reliable conclusion whether an appropriate sequence of increments can make Shellsort a $\Theta(N \log N)$ sorting method on the average. Some hypotheses may be possible when the sequences are prolonged to sort arrays of about 10^5 elements.

References

1. Ἀριστοτέλης: Ἀναλυτικά προτέρα, $64^{\mathrm{b}}28$–$65^{\mathrm{a}}37$; Σοφιστικοὶ ἔλεγχοι, $181^{\mathrm{a}}15$. In: *Aristotelis Opera. Vol. 1: Aristoteles græce*, Academia Regia Borussica, Berolini, 1831.
2. Ghoshdastidar, D., Roy, M. K.: A study on the evaluation of Shell's sorting technique. *Computer Journal* **18** (1975), 234–235.
3. Hibbard, T. N.: An empirical study of minimal storage sorting. *Communications of the ACM* **6** (1963), 206–213.
4. Incerpi, J., Sedgewick, R.: Improved upper bounds on Shellsort. *Journal of Computer and System Sciences* **31** (1985), 210–224.
5. Janson, S., Knuth, D. E.: Shellsort with three increments. *Random Structures and Algorithms* **10** (1997), 125–142.
6. Jiang, T., Li, M., Vitányi, P.: The average-case complexity of Shellsort. *Lecture Notes in Computer Science* **1644** (1999), 453–462.
7. Knuth, D. E.: *The Art of Computer Programming. Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, MA, 1998.
8. Pratt, V. R.: *Shellsort and Sorting Networks.* Garland, New York, 1979, PhD thesis, Stanford University, Department of Computer Science, 1971.
9. Sedgewick, R: A new upper bound for Shellsort. *Journal of Algorithms* **7** (1986), 159–173.
10. Sedgewick, R.: Analysis of Shellsort and related algorithms. *Lecture Notes in Computer Science* **1136** (1996), 1–11.
11. Shell, D. L.: A high-speed sorting procedure. *Communications of the ACM* **2** (1959), 30–32.
12. Tokuda, N: An improved Shellsort. *IFIP Transactions* **A-12** (1992), 449–457.
13. Wald, A.: *Sequential Analysis.* J. Wiley & Sons, New York, 1947.
14. Yao, A. C.: An analysis of $(h, k, 1)$-Shellsort. *Journal of Algorithms* **1** (1980), 14–50.