Information Processing Letters 28 (1988) 133–136
North-Holland

4 July 1988

BAD CASES FOR SHAKER-SORT

Mark Allen WEISS

School of Computer Science, Florida International University, University Park, Miami, FL 33199, U.S.A.

Robert SEDGEWICK

Department of Computer Science, Princeton University, Princeton, NJ 08540, U.S.A.

Communicated by David Gries
Received 10 December 1987
Revised 18 February 1988

Keywords: Shellsort, sorting, shaker-sort

1. Introduction

Shellsort is a sorting algorithm proposed by Shell in 1959 [3]. Using a sequence of integers h_t, $h_{t-1}, \ldots, h_1$, Shellsort works by performing insertion sort on subfiles consisting of elements h_i apart. We call this an h_i*-sort*. Shellsort works by performing an h_t*-sort*, an h_{t-1}*-sort*, and so on until an $h_1 = 1$*-sort*. Typically, the increment sequence is 'almost' geometric, so there are O(log N) increments (or passes). Its worst-case time is more than O(N log N), and its average case appears, at least empirically, to be more than O(N log N), too.

Recently, Incerpi and Sedgewick [1] proposed a new variant of Shellsort called "Shaker-sort". Like Shellsort, Shaker-sort uses an increment sequence (although the algorithms do not seem to share 'best' increment sequences), but it fixes the work done in each pass to be linear. At the end of the algorithm, there is a final insertion sort, which we call the *mop-up* phase. 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). Furthermore, Shaker-sort translates directly into a sorting network, so if Shaker-sort were indeed O(N log N), we would have a simple solution to the optimal sorting network problem.

Incerpi and Sedgewick 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. By using a method similar to that for lower-bounding Shellsort [4], we obtain permutations for which Shaker-sort takes quadratic time using the increments suggested in [1]. We also examine variations of these increments, but these variations also seem to yield a quadratic Shaker-sort.

2. The Shaker-sort algorithm

Shaker-sort performs *k-shakes* on a permutation. A *k-shake* consists of moving through the file from left to right, comparing x_i and x_{i+k} and exchanging if necessary, and then moving from right to left, comparing x_i and x_{i-k} and exchanging if necessary. Thus, the time to *k-shake* is $2(N-k)$.

Shaker-sort uses an increment sequence h_t, $h_{t-1}, \ldots, h_1 = 1$ and performs an h_t*-shake*, h_{t-1}*-shake*, …, 1*-shake*. Since the file is not guaranteed to be sorted, we continue using as many 1*-shakes* as required to finish the sort. Since each 1*-shake* requires linear time, at most $O(\log N)$ *mop-up 1-shakes* can be used while retaining optimality. If the increment sequence is $O(\log N)$ in size, and $O(\log N)$ *1-shakes* guarantee a sort, then the algorithm runs in $O(N \log N)$.

For Shaker-sort, one good increment sequence (determined empirically by Incerpi and Sedgewick [1]) seems to be $1, 2, 3, 5, \ldots, \lceil 1.7^i \rceil, \ldots$. (1.7 can be replaced by smaller numbers such as 1.6, but this increases the number of comparisons required. Some choices also produce increment sequences with several consecutive even numbers; these perform poorly.) For this set of increments, Incerpi and Sedgewick were unable to find any permutations requiring even one additional *1-shake*. Our own exhaustive tests confirm that, for $N \leqslant 32$, Shaker-sort always sorts using this increment sequence. In fact, for $N \leqslant 11$, the sequence $\{1, 2, 3\}$ suffices, for $N \leqslant 23$, the sequence $\{1, 2, 3, 5\}$ suffices and, for $N \leqslant 32$, $\{1, 2, 3, 5, 9\}$ suffices.

3. Lower-bounding Shaker-sort

We 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. For each, we explain how to construct a bad permutation, sketch the reasons why it will produce quadratic running time, and present the empirical results of running Shaker-sort on it. The reader interested in the mathematical details can consult [4].

First, since Shaker-sort is directly implemented as a sorting network, we have the following well-known theorem that allows us to simplify the problem (see [2, p. 224] for the proof).

3.1. Theorem (0–1 principle). *If a network with N input lines sorts all 2^N sequences of 0's and 1's into nondecreasing order, it sorts any arbitrary sequence of N numbers into nondecreasing order.*

Thus we can restrict our attention to 0–1 permutations.

The permutation we use, from [4], depends on the increments used. Assume that the increments are $h_1, h_2, \ldots$. Pick some increment h_k and store the permutation P as $p_0, p_1, \ldots, p_{N-1}$. Then our permutation is defined as follows.

3.2. Definition. $p_i \equiv 1$ iff i is representable as a linear combination of nonnegative integer coefficients of $h_k, h_{k+1}, \ldots, h_\infty$ and $\equiv 0$ otherwise.

It is a well-known fact from number theory that eventually every p_i will be 1 if $h_k, h_{k+1}, \ldots$ satisfy a few technical conditions (which they do), and thus we take $N-1$ to be the largest integer that cannot be represented. As an example, if $h_k = 2k - 1$, then, for $k = 3$, $P = 100001010$, since the representable numbers are 0, 5, 7, 9, 10, 11, 12, 13, 14….

This permutation is bad because it has many *inversions* that cannot be removed quickly. (Recall that an inversion in a permutation P is a pair of indices i, j such that $p_i < p_j$ and $i > j$.) The permutation generated by the above definition seems to have $\Omega(N^2)$ inversions, although we cannot prove this for all cases [4]. A sorted file has no inversions, and inversions are removed from the permutation as follows.

3.3. Lemma. *Swapping a 0 and a 1 a distance d apart in a 0–1 permutation removes exactly d inversions.*

For the proof, see [4].

3.4. Lemma. *A k-shake removes at most kN inversions from a 0–1 permutation of size N.*

Proof. For 0–1 permutations, observe that a *1-shake* merely swaps the leftmost 1 with the rightmost 0. For a *k-shake*, in each of the k subfiles, the leftmost 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. □

We also know that all the subfiles spaced h_k, $h_{k+1}, \ldots$ are initially sorted because of the way the permutation has been constructed. Thus, the maximum number of inversions that can be removed can be expressed as $\sum_{i=1}^{k-1} h_i N = O(N h_k)$ because the increments are geometric. It turns out that h_k is less than $O(N)$; its exact value is unimportant because this fact implies that Shaker-sort cannot remove a quadratic number of inversions before *mop-up* passes are used. Since we start out with a quadratic number of inversions, there will still be a quadratic number of inversions left when the *mop-up* passes start, so the *mop-up* phase will require quadratic time.

4. Empirical results

We use the method described in the previous section to generate bad permutations for Shaker-sort. First we attempt to find the smallest permutation for which Shaker-sort requires one *mop-up* pass, since it may be that Shaker-sort will always work for quite reasonable sizes even though we expect it eventually to deteriorate. Using the $\lceil 1.7^i \rceil$ increments, we know that Shaker-sort will always work for $N \leqslant 32$, since an exhaustive search has been run. We do not know the largest value of N for which no extra 1-*shakes* are required, but it is certainly at most 56. The following permutation, which requires a *mop-up* pass, is obtained by replacing the zeros by the numbers 1–34 and the ones by the numbers 35–57 in the start of the 0–1 permutation generated with $h_k = 9$:

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.

Next, we run Shaker-sort using the $\lceil 1.7^i \rceil$ increments on the 0–1 permutation, *P*. For $N = 734\,702$ (which corresponds to $h_k = 8273$), 35 956 *mop-up* 1-*shakes* are required to complete the sort. It is clear that Shaker-sort is not $O(N \log N)$ for these increments, since the number of *mop-up* 1-*shakes* is linear in the permutation size.

One can use several variations of Shaker-sort to try to force the large h_i-*shakes* to do some work (since this is the cause of the quadratic running time). Table 1 summarizes the tests run on some of these alternative algorithms. All these tests were run on the same permutation as above, even though one might try to construct new permutations specially designed to make the modified increment sequences perform badly. Variation *A* runs Shaker-sort twice before doing the *mop-up*. The hope is that after the first pass is done, the file will be scrambled enough so that the large shakes in the second Shaker-sort will not be neutralized. Variation *B* uses $(k-2)$-*shakes* in the second pass instead of *k*-*shakes*. Variation *C* intersperses 1-*shakes* between the original shakes and variation *D* intersperses $(h_k - 1)$-*shakes* between the original shakes (recall that the h_k-*shake* is by definition the largest to do no work using the original increments). None of these changes seem to reduce the number of *mop-up* passes below linear.

Table 1

N	Mop-up 1-*shakes*				
	Original	*A*	*B*	*C*	*D*
2938	83	0	2	74	0
8749	564	277	292	554	25
15580	878	392	406	867	14
27093	1293	483	480	1281	69
49974	2299	888	907	2286	136
96626	4286	1878	1917	4272	301
184706	8233	4140	4262	8218	780
417449	23473	16516	16547	23457	2103
734702	35956	24142	24165	35939	5029
1360732	63522	43425	43460	63504	7515
2480239	114561	80400	80437	114542	13617

5. Summary

By example, we have shown that Shaker-sort is quadratic in the worst case for the specific increments suggested by Incerpi and Sedgewick and that simple variations of these increment sequences do not do significantly better. For the same reason that Shellsort is not likely to be $O(N \log N)$ in the worst case [4], Shaker-sort is not likely to be any better than $O(N^2)$ in the worst case, for any $O(\log N)$ increment sequence. While the average case of Shaker-sort certainly does not appear to suffer this problem, Shaker-sort does not seem to be significantly faster than Shellsort for any file size, especially when good increment sequences are used for Shellsort. Shaker-sort's primary use would have been as a simple network sorter, but its quadratic worst-case precludes that possibility.

Whether Shaker-sort can be useful as a probabilistic network sorter remains an interesting open question. We have yet to randomly generate a permutation of smaller than 100 000 elements that required extra passes to sort; O(log N) passes are allowed so there is still quite a cushion for sorting files in this size range.

References

[1] J. Incerpi and R. Sedgewick, Practical variations on Shellsort, *Inform. Process. Lett.* **26** (1987) 37–43.

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

[3] D.L. Shell, A high-speed sorting procedure, *Comm. ACM* 2 (7) (1959) 30–32.

[4] M.A. Weiss and R. Sedgewick, *Tight Lower Bounds For Shellsort*, Tech. Rept. #CS-TR-137-88, Dept. of Computer Science, Princeton Univ., 1988.