

Shellsort

From Wikipedia, the free encyclopedia

Shellsort, also known as **Shell sort** or **Shell's method**, is an in-place comparison sort. It can be seen as either a generalization of sorting by exchange (bubble sort) or sorting by insertion (insertion sort).^[3] The method starts by sorting pairs of elements far apart from each other, then progressively reducing the gap between elements to be compared. Starting with far apart elements, it can move some out-of-place elements into position faster than a simple nearest neighbor exchange. Donald Shell published the first version of this sort in 1959.^{[4][5]} The running time of Shellsort is heavily dependent on the gap sequence it uses. For many practical variants, determining their time complexity remains an open problem.

Contents

- 1 Description
- 2 Pseudocode
- 3 Gap sequences
- 4 Computational complexity
- 5 Applications
- 6 See also
- 7 References
- 8 Bibliography
- 9 External links

Description

Shellsort is a generalization of insertion sort that allows the exchange of items that are far apart. The idea is to arrange the list of elements so that, starting anywhere, considering every *h*th element gives a sorted list. Such a list is said to be *h*-sorted. Equivalently, it can be thought of as *h* interleaved lists, each individually sorted.^[6] Beginning with large values of *h*, this rearrangement allows elements to move long distances in the original list, reducing large amounts of disorder quickly, and leaving less work for smaller *h*-sort steps to do.^[7] If the file is then *k*-sorted for some smaller integer *k*, then the file remains *h*-sorted. Following this idea for a decreasing sequence of *h* values ending in 1 is guaranteed to leave a sorted list in the end.^[6]

An example run of Shellsort with gaps 5, 3 and 1 is shown below.

	<i>a</i> ₁	<i>a</i> ₂	<i>a</i> ₃	<i>a</i> ₄	<i>a</i> ₅	<i>a</i> ₆	<i>a</i> ₇	<i>a</i> ₈	<i>a</i> ₉	<i>a</i> ₁₀	<i>a</i> ₁₁	<i>a</i> ₁₂
Input data	62	83	18	53	07	17	95	86	47	69	25	28
After 5-sorting	17	28	18	47	07	25	83	86	53	69	62	95
After 3-sorting	17	07	18	47	28	25	69	62	53	83	86	95
After 1-sorting	07	17	18	25	28	47	53	62	69	83	86	95

The first pass, 5-sorting, performs insertion sort on five separate subarrays (*a*₁, *a*₆, *a*₁₁), (*a*₂, *a*₇, *a*₁₂), (*a*₃, *a*₈), (*a*₄, *a*₉), (*a*₅, *a*₁₀). For instance, it changes the subarray (*a*₁, *a*₆, *a*₁₁) from (62, 17, 25) to (17, 25, 62). The next pass, 3-sorting, performs insertion sort on the three subarrays (*a*₁, *a*₄, *a*₇, *a*₁₀), (*a*₂, *a*₅, *a*₈, *a*₁₁), (*a*₃, *a*₆, *a*₉, *a*₁₂). The last pass, 1-sorting, is an ordinary insertion sort of the entire array (*a*₁, ..., *a*₁₂).

As the example illustrates, the subarrays that Shellsort operates on are initially short; later they are longer but almost ordered. In both cases insertion sort works efficiently.

Shellsort is unstable: it may change the relative order of elements with equal values. It is an adaptive sorting algorithm in that it executes faster when the input is partially sorted.

Pseudocode

Using Marcin Ciura's gap sequence, with an inner insertion sort.

Shellsort



The step-by-step process of replacing pairs of items during the shell sorting algorithm.

```

# Sort an array a[0...n-1].
gaps = [701, 301, 132, 57, 23, 10, 4, 1]

# Start with the largest gap and work down to a gap of 1
[[foreach]] (gap in gaps)
{
    # Do a gapped insertion sort for this gap size.
    # The first gap elements a[0..gap-1] are already in gapped order
    # keep adding one more element until the entire array is gap sorted
    for (i = gap; i < n; i += 1)
    {
        # add a[i] to the elements that have been gap sorted
        # save a[i] in temp and make a hole at position i
        temp = a[i]
        # shift earlier gap-sorted elements up until the correct location for a[i] is found
        for (j = i; j >= gap and a[j - gap] > temp; j -= gap)
        {
            a[j] = a[j - gap]
        }
        # put temp (the original a[i]) in its correct location
        a[j] = temp
    }
}

```

Gap sequences

The question of deciding which gap sequence to use is difficult. Every gap sequence that contains 1 yields a correct sort; however, the properties of thus obtained versions of Shellsort may be very different.

The table below compares most proposed gap sequences published so far. Some of them have decreasing elements that depend on the size of the sorted array (N). Others are increasing infinite sequences, whose elements less than N should be used in reverse order.

General term ($k \geq 1$)	Concrete gaps	Worst-case time complexity	Author and year of publication
$\left\lfloor \frac{N}{2^k} \right\rfloor$	$\left\lfloor \frac{N}{2} \right\rfloor, \left\lfloor \frac{N}{4} \right\rfloor, \dots, 1$	$\Theta(N^2)$ [when $N = 2^p$]	Shell, 1959 ^[4]
$2 \left\lfloor \frac{N}{2^{k+1}} \right\rfloor + 1$	$2 \left\lfloor \frac{N}{4} \right\rfloor + 1, \dots, 3, 1$	$\Theta(N^{3/2})$	Frank & Lazarus, 1960 ^[8]
$2^k - 1$	$1, 3, 7, 15, 31, 63, \dots$	$\Theta(N^{3/2})$	Hibbard, 1963 ^[9]
$2^k + 1$, prefixed with 1	$1, 3, 5, 9, 17, 33, 65, \dots$	$\Theta(N^{3/2})$	Papernov & Stasevich, 1965 ^[10]
Successive numbers of the form $2^p 3^q$	$1, 2, 3, 4, 6, 8, 9, 12, \dots$	$\Theta(N \log^2 N)$	Pratt, 1971 ^[11]
$\frac{3^k - 1}{2}$, not greater than $\left\lfloor \frac{N}{3} \right\rfloor$	$1, 4, 13, 40, 121, \dots$	$\Theta(N^{3/2})$	Pratt, 1971 ^[11]
$\prod_I a_q$, where $a_q = \min \left\{ n \in \mathbb{N} : n \geq \left(\frac{5}{2} \right)^{q+1}, \forall p: 0 \leq p < q \Rightarrow \gcd(a_p, n) = 1 \right\}$ $I = \left\{ 0 \leq q < r \mid q \neq \frac{1}{2}(r^2 + r) - k \right\}$ $r = \left\lfloor \sqrt{2k + \sqrt{2k}} \right\rfloor$	$1, 3, 7, 21, 48, 112, \dots$	$O\left(N^{1 + \sqrt{\frac{8 \ln(6/2)}{\ln(N)}}}\right)$	Incerpi & Sedgewick, 1985, ^[11] Knuth ^[3]
$4^k + 3 \cdot 2^{k-1} + 1$, prefixed with 1	$1, 8, 23, 77, 281, \dots$	$O(N^{4/3})$	Sedgewick, 1986 ^[6]
$9(4^{k-1} - 2^{k/2}) + 1, 4^{k+1} - 6 \cdot 2^{(k+1)/2} + 1$	$1, 5, 19, 41, 109, \dots$	$O(N^{4/3})$	Sedgewick, 1986 ^[12]
$h_k = \max \left\{ \left\lfloor \frac{5h_{k-1}}{11} \right\rfloor, 1 \right\}, h_0 = N$	$\left\lfloor \frac{5N}{11} \right\rfloor, \left\lfloor \frac{5}{11} \left\lfloor \frac{5N}{11} \right\rfloor \right\rfloor, \dots, 1$	Unknown	Gonnet & Baeza-Yates, 1991 ^[13]
$\left\lceil \frac{9^k - 4^k}{5 \cdot 4^{k-1}} \right\rceil$	$1, 4, 9, 20, 46, 103, \dots$	Unknown	Tokuda, 1992 ^[14]
Unknown (experimentally derived)	1, 4, 10, 23, 57, 132, 301, 701	Unknown	Ciura, 2001 ^[15]

When the binary representation of N contains many consecutive zeroes, Shellsort using Shell's original gap sequence makes $\Theta(N^2)$ comparisons in the worst case. For instance, this case occurs for N equal to a power of two when elements greater and smaller than the median occupy odd and even positions respectively, since they are compared only in the last pass.

Although it has higher complexity than the $O(N \log N)$ that is optimal for comparison sorts, Pratt's version lends itself to sorting networks and has the same asymptotic gate complexity as Batcher's bitonic sorter.

Gonnet and Baeza-Yates observed that Shellsort makes the fewest comparisons on average when the ratios of successive gaps are roughly equal to 2.2.^[13] This is why their sequence with ratio 2.2 and Tokuda's sequence with ratio 2.25 prove efficient. However, it is not known why this is so. Sedgewick recommends to use gaps that have low greatest common divisors or are pairwise coprime.^[16]

With respect to the average number of comparisons, Ciura's sequence^[15] has the best performance; gaps from 701 were not determined but the sequence can be further extended according to the recursive formula $h_k = \lfloor 2.25h_{k-1} \rfloor$.

Tokuda's sequence, defined by the simple formula $h_k = \lceil h'_k \rceil$, where $h'_k = 2.25h'_{k-1} + 1$, $h'_1 = 1$, can be recommended for practical applications.

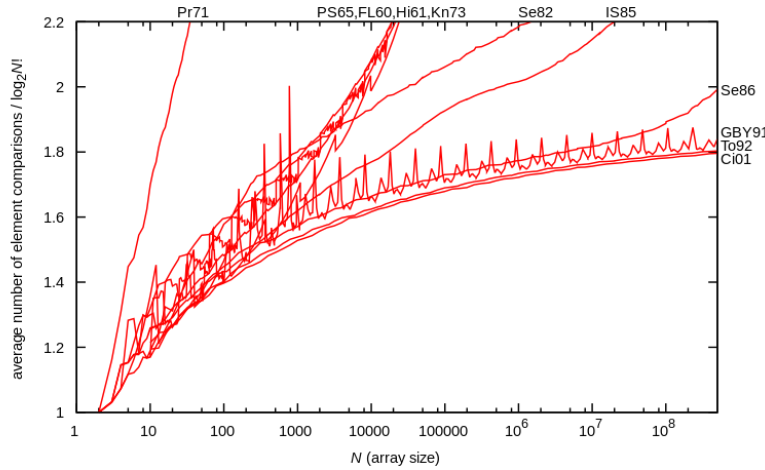
Computational complexity

The following property holds: after h_2 -sorting of any h_1 -sorted array, the array remains h_1 -sorted.^[17] Every h_1 -sorted and h_2 -sorted array is also $(a_1h_1+a_2h_2)$ -sorted, for any nonnegative integers a_1 and a_2 . The worst-case complexity of Shellsort is therefore connected with the Frobenius problem: for given integers h_1, \dots, h_n with $\gcd = 1$, the Frobenius number $g(h_1, \dots, h_n)$ is the greatest integer that cannot be represented as $a_1h_1 + \dots + a_nh_n$ with nonnegative integer a_1, \dots, a_n . Using known formulae for Frobenius numbers, we can determine the worst-case complexity of Shellsort for several classes of gap sequences.^[18] Proven results are shown in the above table.

With respect to the average number of operations, none of proven results concerns a practical gap sequence. For gaps that are powers of two, Espelid computed this average as $0.5349N\sqrt{N} - 0.4387N - 0.097\sqrt{N} + O(1)$.^[19] Knuth determined the average complexity of sorting an N -element array with two gaps $(h, 1)$ to be $\frac{2N^2}{h} + \sqrt{\pi N^3 h}$.^[3] It follows that a two-pass Shellsort with $h = \Theta(N^{1/3})$ makes on average $O(N^{5/3})$ comparisons. Yao found the average complexity of a three-pass Shellsort.^[20] His result was refined by Janson and Knuth:^[21] the average number of comparisons made during a Shellsort with three gaps $(ch, cg, 1)$, where h and g are coprime, is $\frac{N^2}{4ch} + O(N)$ in the first pass, $\frac{1}{8g}\sqrt{\frac{\pi}{ch}}(h-1)N^{3/2} + O(hN)$ in the second pass and $\psi(h, g)N + \frac{1}{8}\sqrt{\frac{\pi}{c}}(c-1)N^{3/2} + O((c-1)gh^{1/2}N) + O(c^2g^3h^2)$ in the third pass. $\psi(h, g)$ in the last formula is a complicated function asymptotically equal to $\sqrt{\frac{\pi h}{128}}g + O(g^{-1/2}h^{1/2}) + O(gh^{-1/2})$. In particular, when $h = \Theta(N^{7/15})$ and $g = \Theta(N^{1/5})$, the average time of sorting is $O(N^{23/15})$.

Based on experiments, it is conjectured that Shellsort with Hibbard's gap sequence runs in $O(N^{5/4})$ average time,^[3] and that Gonnet and Baeza-Yates's sequence requires on average $0.41MnN(\ln \ln N + 1/6)$ element moves.^[13] Approximations of the average number of operations formerly put forward for other sequences fail when sorted arrays contain millions of elements.

The graph below shows the average number of element comparisons in various variants of Shellsort, divided by the theoretical lower bound, i.e. $\log_2 N!$, where the sequence 1, 4, 10, 23, 57, 132, 301, 701 has been extended according to the formula $h_k = \lfloor 2.25h_{k-1} \rfloor$.



Applying the theory of Kolmogorov complexity, Jiang, Li, and Vitányi proved the following lower bounds for the order of the average number of operations in an m -pass Shellsort: $\Omega(mN^{1+1/m})$ when $m \leq \log_2 N$ and $\Omega(mN)$ when $m > \log_2 N$.^[22] Therefore, Shellsort has prospects of running in an average time that asymptotically grows like $M \log N$ only when using gap sequences whose number of gaps grows in proportion to the logarithm of the array size. It is, however, unknown whether Shellsort can reach this asymptotic order of average-case complexity, which is optimal for comparison sorts.

The worst-case complexity of any version of Shellsort is of higher order: Plaxton, Poonen, and Suel showed that it grows at least as rapidly as $\Omega(N(\log N / \log \log N)^2)$.^[23]

Applications

Shellsort performs more operations and has higher cache miss ratio than quicksort. However, since it can be implemented using little code and does not use the call stack, some implementations of the qsort function in the C standard library targeted at embedded systems use it instead of quicksort. Shellsort is, for example, used in the uClibc library.^[24] For similar reasons, an implementation of Shellsort is present in the Linux kernel.^[25]

Shellsort can also serve as a sub-algorithm of introspective sort, to sort short subarrays and to prevent a pathological slowdown when the recursion depth exceeds a given limit. This principle is employed, for instance, in the bzip2 compressor.^[26]

See also

- Comb sort

References

1. Pratt, Vaughan Ronald (1979). *Shellsort and Sorting Networks (Outstanding Dissertations in the Computer Sciences)*. Garland. ISBN 0-8240-4406-1.
2. "Shellsort & Comparisons".
3. Knuth, Donald E. (1997). "Shell's method". *The Art of Computer Programming. Volume 3: Sorting and Searching* (2nd ed.). Reading, Massachusetts: Addison-Wesley. pp. 83–95. ISBN 0-201-89685-0.
4. Shell, D. L. (1959). "A High-Speed Sorting Procedure" (PDF). *Communications of the ACM*. **2** (7): 30–32. doi:10.1145/368370.368387.
5. Some older textbooks and references call this the "Shell-Metzner" sort after Marlene Metzner Norton, but according to Metzner, "I had nothing to do with the sort, and my name should never have been attached to it." See "Shell sort". National Institute of Standards and Technology. Retrieved 2007-07-17.
6. Sedgewick, Robert (1998). *Algorithms in C*. **1** (3rd ed.). Addison-Wesley. pp. 273–281. ISBN 0-201-31452-5.
7. Kernighan, Brian W.; Ritchie, Dennis M. (1996). *The C Programming Language* (2nd ed.). Prentice Hall. p. 62. ISBN 7-302-02412-X.
8. Frank, R. M.; Lazarus, R. B. (1960). "A High-Speed Sorting Procedure". *Communications of the ACM*. **3** (1): 20–22. doi:10.1145/366947.366957.
9. Hibbard, Thomas N. (1963). "An Empirical Study of Minimal Storage Sorting". *Communications of the ACM*. **6** (5): 206–213. doi:10.1145/366552.366557.
10. Papernov, A. A.; Stasevich, G. V. (1965). "A Method of Information Sorting in Computer Memories" (PDF). *Problems of Information Transmission*. **1** (3): 63–75.
11. Incerpi, Janet; Sedgewick, Robert (1985). "Improved Upper Bounds on Shellsort". *Journal of Computer and System Sciences*. **31** (2): 210–224. doi:10.1016/0022-0000(85)90042-x.
12. Sedgewick, Robert (1986). "A New Upper Bound for Shellsort". *Journal of Algorithms*. **7** (2): 159–173. doi:10.1016/0196-6774(86)90001-5.
13. Gonnet, Gaston H.; Baeza-Yates, Ricardo (1991). "Shellsort". *Handbook of Algorithms and Data Structures: In Pascal and C* (2nd ed.). Reading, Massachusetts: Addison-Wesley. pp. 161–163. ISBN 0-201-41607-7.
14. Tokuda, Naoyuki (1992). "An Improved Shellsort". In van Leeuwen, Jan. *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture*. Amsterdam: North-Holland Publishing Co. pp. 449–457. ISBN 0-444-89747-X.
15. Ciura, Marcin (2001). "Best Increments for the Average Case of Shellsort" (PDF). In Freiwalds, Rusins. *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory*. London: Springer-Verlag. pp. 106–117. ISBN 3-540-42487-3. doi:10.1016/S0022-0000(72)80016-3.
16. Sedgewick, Robert (1998). "Shellsort". *Algorithms in C++, Parts 1–4: Fundamentals, Data Structure, Sorting, Searching*. Reading, Massachusetts: Addison-Wesley. pp. 285–292. ISBN 0-201-35088-2.
17. Gale, David; Karp, Richard M. (1972). "A Phenomenon in the Theory of Sorting". *Journal of Computer and System Sciences*. **6** (2): 103–115. doi:10.1016/S0022-0000(72)80016-3.
18. Selmer, Ernst S. (1989). "On Shellsort and the Frobenius Problem". *BIT Numerical Mathematics*. **29** (1): 37–40. doi:10.1007/BF01932703.
19. Espelid, Terje O. (1973). "Analysis of a Shellsort Algorithm". *BIT Numerical Mathematics*. **13** (4): 394–400. doi:10.1007/BF01933401.
20. Yao, Andrew Chi-Chih (1980). "An Analysis of $(h, k, 1)$ -Shellsort". *Journal of Algorithms*. **1** (1): 14–50. doi:10.1016/0196-6774(80)90003-6.
21. Janson, Svante; Knuth, Donald E. (1997). "Shellsort with Three Increments". *Random Structures and Algorithms*. **10** (1–2): 125–142. arXiv:cs/9608105. doi:10.1002/(SICI)1098-2418(199701/03)10:1/2<125::AID-RSA6>3.0.CO;2-X. CiteSeerX: 10.1.1.54.9911.
22. Jiang, Tao; Li, Ming; Vitányi, Paul (2000). "A Lower Bound on the Average-Case Complexity of Shellsort". *Journal of the ACM*. **47** (5): 905–911. doi:10.1145/355483.355488. CiteSeerX: 10.1.1.6.6508.
23. Plaxton, C. Greg; Poonen, Bjarne; Suel, Torsten (1992). "Improved Lower Bounds for Shellsort". *Annual Symposium on Foundations of Computer Science*. **33**: 226–235. doi:10.1109/SFCS.1992.267769. ISBN 0-8186-2900-2. CiteSeerX: 10.1.1.43.1393.
24. Novoa, Manuel III. "libc/stdlib/stdlib.c". Retrieved 2014-10-29.
25. "kernel/groups.c". Retrieved 2012-05-05.
26. Julian Seward. "bzip2/blocksort.c". Retrieved 2011-03-30.

Bibliography

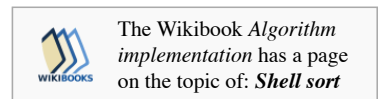
- Knuth, Donald E. (1997). "Shell's method". *The Art of Computer Programming. Volume 3: Sorting and Searching* (2nd ed.). Reading, Massachusetts: Addison-Wesley. pp. 83–95. ISBN 0-201-89685-0.
- Analysis of Shellsort and Related Algorithms (<http://www.cs.princeton.edu/~rs/shell/>), Robert Sedgewick, Fourth European Symposium on Algorithms, Barcelona, September 1996.

External links

- Shellsort with gaps 5, 3, 1 as a Hungarian folk dance (<http://www.youtube.com/watch?v=CmPA7zE8mx0>)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Shellsort&oldid=748273578"

Categories: Sorting algorithms | Comparison sorts



- This page was last modified on 7 November 2016, at 10:08.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.