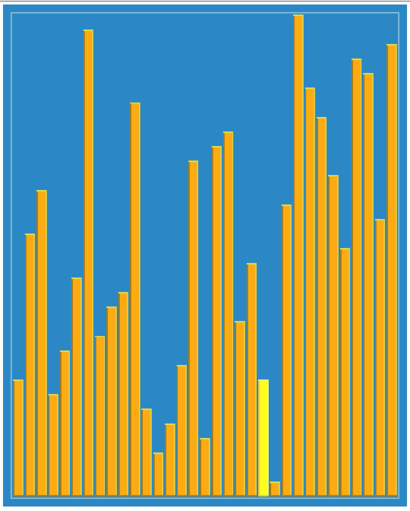
Shellsort

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.

Shellsort



Shellsort with gaps 23, 10, 4, 1 in action

Class	Sorting algorithm
Data structure	Array
Worst-case performance	O(n^2) (worst known worst case gap sequence) O($n \log^2 n$) (best known worst case gap sequence)[1]
Best-case performance	O($n \log n$) (most gap sequences) O($n \log^2 n$) (best known worst-case gap sequence)[2]
Average performance	depends on gap sequence
Worst-case space complexity	O(n) total, O(1) auxiliary

Contents

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 hth 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 list is then k-*sorted* for some smaller integer k, then the list 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.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{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

Swapping pairs of items in successive steps of Shellsort with gaps 5, 3, 1

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 not stable: 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.

```
# 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 (as this makes the final pass an ordinary insertion sort); however, the properties of thus obtained versions of Shellsort may be very different. Too few gaps slows down the passes, and too many gaps produces an overhead.

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.

OEIS	General term ($k \geq 1$)	Concrete gaps	Worst-case time complexity	Author and year of publication
	$\left\lfloor \frac{N}{2^k} \right\rfloor$	$\left\lfloor \frac{N}{2} \right\rfloor, \left\lfloor \frac{N}{4} \right\rfloor, \ldots, 1$	$\Theta(N^2)$ [e.g. when $N = 2^p$]	Shell, 1959[4]
	$2\left\lfloor \frac{N}{2^{k+1}} \right\rfloor + 1$	$2\left\lfloor \frac{N}{4} \right\rfloor + 1, \ldots, 3, 1$	$\Theta\left(N^{\frac{3}{2}}\right)$	Frank & Lazarus, 1960[8]
A168604	$2^k - 1$	$1, 3, 7, 15, 31, 63, \ldots$	$\Theta\left(N^{\frac{3}{2}}\right)$	Hibbard, 1963[9]
A083318	$2^k + 1$, prefixed with 1	$1, 3, 5, 9, 17, 33, 65, \ldots$	$\Theta\left(N^{\frac{3}{2}}\right)$	Papernov & Stasevich, 1965[10]
A003586	Successive numbers of the form $2^p 3^q$ (3-smooth numbers)	$1, 2, 3, 4, 6, 8, 9, 12, \ldots$	$\Theta\left(N \log^2 N\right)$	Pratt, 1971[1]
A003462	$\frac{3^k - 1}{2}$, not greater than $\left\lceil \frac{N}{3} \right\rceil$	$1, 4, 13, 40, 121, \ldots$	$\Theta\left(N^{\frac{3}{2}}\right)$	Pratt, 1971,[1] Knuth, 1973[3]
A036569	$\prod\limits_I a_q$, where $a_q = \min\left\{n \in \mathbb{N} : n \geq \left(\frac{5}{2}\right)^{q+1}, \forall p : 0 \leq p < q \Rightarrow \gcd(a_p, n) = 1\right\}$ $I = \left\{0 \leq q < r \mid q \neq \frac{1}{2}\left(r^2 + r\right) - k\right\}$ $r = \left\lfloor \sqrt{2k + \sqrt{2k}} \right\rfloor$	$1, 3, 7, 21, 48, 112, \ldots$	$O\left(N^{1+\sqrt{\frac{8 \ln(5/2)}{\ln(N)}}}\right)$	Incerpi & Sedgewick, 1985,[11] Knuth[3]
A036562	$4^k + 3 \cdot 2^{k-1} + 1$, prefixed with 1	$1, 8, 23, 77, 281, \ldots$	$O\left(N^{\frac{4}{3}}\right)$	Sedgewick, 1982[6]
A033622	$\begin{cases} 9\left(2^k - 2^{\frac{k}{2}}\right) + 1 & k \text{ even}, \\ 8 \cdot 2^k - 6 \cdot 2^{(k+1)/2} + 1 & k \text{ odd} \end{cases}$	$1, 5, 19, 41, 109, \ldots$	$O\left(N^{\frac{4}{3}}\right)$	Sedgewick, 1986[12]
	$h_k = \max\left\{\left\lfloor \frac{5h_{k-1}}{11} \right\rfloor, 1\right\}, h_0 = N$	$\left\lfloor \frac{5N}{11} \right\rfloor, \left\lfloor \frac{5}{11}\left\lfloor \frac{5N}{11} \right\rfloor \right\rfloor, \ldots, 1$	Unknown	Gonnet & Baeza-Yates, 1991[13]
A108870	$\left\lceil \frac{1}{5}\left(9 \cdot \left(\frac{9}{4}\right)^{k-1} - 4\right) \right\rceil$	$1, 4, 9, 20, 46, 103, \ldots$	Unknown	Tokuda, 1992[14]
A102549	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 known performance; gaps from 701 were not determined but the sequence can be further extended according to the recursive formula $h_k = \lfloor 2.25 h_{k-1} \rfloor$.

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

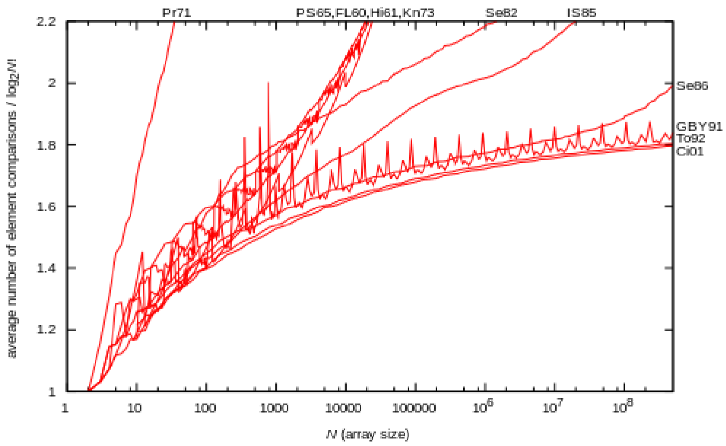
Computational complexity

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 $h_1,..., h_n$ with gcd = 1, the Frobenius number $g(h_1,..., h_n)$ is the greatest integer that cannot be represented as $a_1h_1+ ... +a_nh_n$ with nonnegative integer $a_1,..., a_n$. 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 the 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/inversions/running time. 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/inversions/running time 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 $\psi(h,g)N + \frac{1}{8}\sqrt{\frac{\pi}{c}}(c-1)N^{3/2} + O\left((c-1)gh^{1/2}N\right) + O\left(c^2g^3h^2\right)$ in the third pass. $\psi(h, g)$ in the last formula is a complicated function asymptotically equal to $\sqrt{\frac{\pi h}{128}}g + O\left(g^{-1/2}h^{1/2}\right) + O\left(gh^{-1/2}\right)$. 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 $0.41N\ln N(\ln \ln N+1/6)$ 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 $h_k = \lfloor 2.25 h_{k-1} \rfloor$.



Applying the theory of Kolmogorov complexity, Jiang, Li, and Vitányi proved the following lower bound for the order of the average number of operations/running time in a p-pass Shellsort: $\Omega(pN^{1+1/p})$ when $p \le \log_2 N$ and $\Omega(pN)$ when $p > \log_2 N$.[22] Therefore, Shellsort has prospects of running in an average time that asymptotically grows like $N\log N$ 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 lower bound was improved by Vitányi[23] for every number of passes p to $\Omega(N\sum_{k=1}^{p} h_{k-1}/h_k)$ where $h_0 = N$.

This result implies for example the Jiang-Li-Vitányi lower bound for all p-pass increment sequences and improves that lower bound for particular increment sequences. In fact all bounds (lower and upper) currently known for the average case are precisely matched by this lower bound. For example, this gives the new result that the Janson-Knuth upper bound is matched by the resulting lower bound for the used increment sequence, showing that three pass Shellsort for this increment sequence uses $\Theta(N^{23/15})$ comparisons/inversions/running time. The formula allows us to search for increment sequences that yield lower bounds which are unknown; for example an increment sequence for four passes which has a lower bound greater than $\Omega(pn^{1+1/p}) = \Omega(n^{5/4})$ for the increment sequence $h_1 = n^{11/16}, h_2 = n^{7/16}, h_3 = n^{3/16}, h_4 = 1$. The lower bound becomes $T = \Omega(n \cdot (n^{1-11/16} + n^{11/16-7/16} + n^{7/16-3/16} + n^{3/16}) = \Omega(n^{1+5/16}) = \Omega(n^{21/16})$.

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 $\Omega\left(N(\frac{\log N}{\log\log N})^2\right)$.[24]

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.[25] For similar reasons, in the past, Shellsort was used in the Linux kernel.[26]

Shellsort can also serve as a sub-algorithm of introspective sort, to sort short subarrays and to prevent a slowdown when the recursion depth exceeds a given limit. This principle is employed, for instance, in the bzip2 compressor.[27]

See also

- Comb sort

References

1. Pratt, Vaughan Ronald (1979). *Shellsort and Sorting Networks (Outstanding Dissertations in the Computer Sciences)* (http://www.dtic.mil/get-tr-doc/pdf?AD=AD0740110). Garland. ISBN 978-0-8240-4406-0.
2. "Shellsort & Comparisons" (http://www.cs.wcupa.edu/rkline/ds/shell-comparison.html).
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 978-0-201-89685-5.
4. Shell, D. L. (1959). "A High-Speed Sorting Procedure" (http://penguin.ewu.edu/cscd300/Topic/AdvSorting/p30-shell.pdf) (PDF). *Communications of the ACM*. **2** (7): 30–32. doi:10.1145/368370.368387 (https://doi.org/10.1145%2F368370.368387).
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" (https://xlinux.nist.gov/dads/HTML/shellsort.html). National Institute of Standards and Technology. Retrieved 17 July 2007.
6. Sedgewick, Robert (1998). *Algorithms in C* (https://archive.org/details/algorithmsinc00sedg/page/273). **1** (3rd ed.). Addison-Wesley. pp. 273–281 (https://archive.org/details/algorithmsinc00sedg/page/273). ISBN 978-0-201-31452-6.
7. Kernighan, Brian W.; Ritchie, Dennis M. (1996). *The C Programming Language* (2nd ed.). Prentice Hall. p. 62. ISBN 978-7-302-02412-5.
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 (https://doi.org/10.1145%2F366947.366957).
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 (https://doi.org/10.1145%2F366552.366557).
10. Papernov, A. A.; Stasevich, G. V. (1965). "A Method of Information Sorting in Computer Memories" (http://www.mathnet.ru/links/83f0a81df1ec06f76d3683c6cab7d143/ppi751.pdf) (PDF). *Problems of Information Transmission*. **1** (3): 63–75.
11. Incerpi, Janet; Sedgewick, Robert (1985). "Improved Upper Bounds on Shellsort" (https://hal.inria.fr/inria-00076291/file/RR-0267.pdf) (PDF). *Journal of Computer and System Sciences*. **31** (2): 210–224. doi:10.1016/0022-0000(85)90042-x (https://doi.org/10.1016%2F0022-0000%2885%2990042-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 (https://doi.org/10.1016%2F0196-6774%2886%2990001-5).
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 978-0-201-41607-7.
14. Tokuda, Naoyuki (1992). "An Improved Shellsort". In van Leeuven, Jan (ed.). *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture*. Amsterdam: North-Holland Publishing Co. pp. 449–457. ISBN 978-0-444-89747-3.
15. Ciura, Marcin (2001). "Best Increments for the Average Case of Shellsort" (http://sun.aei.polsl.pl/~mciura/publikacje/shellsort.pdf) (PDF). In Freiwalds, Rusins (ed.). *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory*. London: Springer-Verlag. pp. 106–117. ISBN 978-3-540-42487-1.
16. Sedgewick, Robert (1998). "Shellsort". *Algorithms in C++, Parts 1–4: Fundamentals, Data Structure, Sorting, Searching*. Reading, Massachusetts: Addison-Wesley. pp. 285–292. ISBN 978-0-201-35088-3.
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 (https://doi.org/10.1016%2FS0022-0000%2872%2980016-3).
18. Selmer, Ernst S. (1989). "On Shellsort and the Frobenius Problem". *BIT Numerical Mathematics*. **29** (1): 37–40. doi:10.1007/BF01932703 (https://doi.org/10.1007%2FBF01932703).
19. Espelid, Terje O. (1973). "Analysis of a Shellsort Algorithm". *BIT Numerical Mathematics*. **13** (4): 394–400. doi:10.1007/BF01933401 (https://doi.org/10.1007%2FBF01933401).

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 (https://doi.org/10.1016%2F0196-6774%2880%2990003-6).
21. Janson, Svante; Knuth, Donald E. (1997). "Shellsort with Three Increments". *Random Structures and Algorithms*. **10** (1–2): 125–142. arXiv:cs/9608105 (https://arxiv.org/abs/cs/9608105). doi:10.1002/(SICI)1098-2418(199701/03)10:1/2<125::AID-RSA6>3.0.CO;2-X (https://doi.org/10.1002%2F%28SICI%291098-2418%28199701%2F03%2910%3A1%2F2%3C125%3A%3AAID-RSA6%3E3.0.CO%3B2-X). CiteSeerX: 10.1.1.54.9911 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.9911).
22. Jiang, Tao; Li, Ming; Vitányi, Paul (2000). "A Lower Bound on the Average-Case Complexity of Shellsort" (http://www.cwi.nl/~paulv/papers/shell2.ps). *Journal of the ACM*. **47** (5): 905–911. arXiv:cs/9906008 (https://arxiv.org/abs/cs/9906008). doi:10.1145/355483.355488 (https://doi.org/10.1145%2F355483.355488). CiteSeerX: 10.1.1.6.6508 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.6508).
23. Vitányi, Paul (2018). "On the average-case complexity of Shellsort". *Random Structures and Algorithms*. **52** (2): 354–363. arXiv:cs/9906008 (https://arxiv.org/abs/cs/9906008). doi:10.1002/rsa.20737 (https://doi.org/10.1002%2Frsa.20737).
24. Plaxton, C. Greg; Poonen, Bjorn; Suel, Torsten (1992). *Improved Lower Bounds for Shellsort*. *Annual Symposium on Foundations of Computer Science*. **33**. pp. 226–235. CiteSeerX 10.1.1.460.2429 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.460.2429). doi:10.1109/SFCS.1992.267769 (https://doi.org/10.1109%2FSFCS.1992.267769). ISBN 978-0-8186-2900-6. CiteSeerX: 10.1.1.43.1393 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.1393).
25. Novoa, Manuel III. "libc/stdlib/stdlib.c" (http://git.uclibc.org/uClibc/tree/libc/stdlib/stdlib.c#n700). Retrieved 29 October 2014.
26. "kernel/groups.c" (https://github.com/torvalds/linux/blob/72932611b4b05bbd89fafa369d564ac8e449809b/kernel/groups.c#L105). Retrieved 5 May 2012.
27. Julian Seward. "bzip2/blocksort.c" (https://www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC/lxr/source/src/util/compress/bzip2/blocksort.c#L519). Retrieved 30 March 2011.

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 978-0-201-89685-5.
- 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

- Animated Sorting Algorithms: Shell Sort (https://web.archive.org/web/20150310043846/http://www.sorting-algorithms.com/shell-sort) at the Wayback Machine (archived 10 March 2015) – graphical demonstration
- Shellsort with gaps 5, 3, 1 as a Hungarian folk dance (https://www.youtube.com/watch?v=CmPA7zE8mx0)