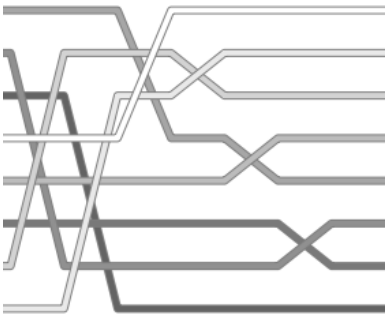
Shell sort



Shell sort in action on a list of numbers.

Class	Sorting algorithm, unstable
Data structure	Array
Worst case performance	depends on gap sequence. Best known: $O(n \log^2 n)$
Best case performance	O(*n*)
Average case performance	depends on gap sequence
Worst case space complexity	O(*n*)

Shell sort is a sorting algorithm, devised by Donald Shell in 1959, that is a generalization of insertion sort, which exploits the fact that insertion sort works efficiently on input that is already almost sorted. It improves on insertion sort by allowing the comparison and exchange of elements that are far apart. The last step of Shell sort is a plain insertion sort, but by then, the array of data is guaranteed to be almost sorted.

The algorithm is an example of an algorithm that is simple to code but difficult to analyze theoretically.

History

The Shell sort is named after its inventor, Donald Shell, who published the algorithm in 1959.[1] 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."[2]

Description

The principle of Shell sort is to rearrange the file so that looking at every *h*th element yields a sorted file. We call such a file *h-sorted*. If the file is then *k-sorted* for some other integer *k*, then the file remains *h*-sorted.[3] For instance, if a list was 5-sorted and then 3-sorted, the list is now not only 3-sorted, but both 5- and 3-sorted. If this were not true, the algorithm would undo work that it had done in previous iterations, and would not achieve such a low running time.

The algorithm draws upon a sequence of positive integers known as the *increment sequence*. Any sequence will do, as long as it ends with 1, but some sequences perform better than others.[4] The algorithm begins by performing a *gap insertion sort*, with the gap being the first number in the increment sequence. It continues to perform a gap insertion sort for each number in the sequence, until it finishes with a gap of 1. When the increment reaches 1, the gap insertion sort is simply an ordinary insertion sort, guaranteeing that the final list is sorted. Beginning with large increments allows elements in the file to move quickly towards their final positions, and makes it easier to subsequently sort for smaller increments.[3]

Although sorting algorithms exist that are more efficient, Shell sort remains a good choice for moderately large files because it has good running time and is easy to code.

Shell sort algorithm in pseudocode

The following is an implementation of Shell sort written in pseudocode. The increment sequence is a geometric sequence in which every term is roughly 2.2 times smaller than the previous one:

```
input: an array a of length n with array elements numbered 0 to n − 1
```

```
inc ← round(n/2)
while inc > 0 do:
    for i = inc .. n − 1 do:
        temp ← a[i]
        j ← i
        while j ≥ inc and a[j − inc] > temp do:
            a[j] ← a[j − inc]
            j ← j − inc
        a[j] ← temp
    inc ← round(inc / 2.2)
```

Analysis

Although Shell sort is easy to code, analyzing its performance is very difficult and depends on the choice of increment sequence. The algorithm was one of the first to break the quadratic time barrier, but this fact was not proven until some time after its discovery.[4]

The initial increment sequence suggested by Donald Shell was [1,2,4,8,16,...,2^k], but this is a very poor choice in practice because it means that elements in odd positions are not compared with elements in even positions until the very last step. The original implementation performs O(n^2) comparisons and exchanges in the worst case.[3] A simple change, replacing 2^k with 2^k-1, improves the worst-case running time to O($N^{3/2}$),[4] a bound that cannot be improved.[5]

A minor change given in V. Pratt's book[5] improved the bound to O(n $\log^2$ n). This is worse than the optimal comparison sorts, which are O(n log n), but lends itself to sorting networks and has the same asymptotic gate complexity as Batcher's bitonic sorter.

Consider a small value that is initially stored in the wrong end of the array. Using an O(n^2) sort such as bubble sort or insertion sort, it will take roughly n comparisons and exchanges to move this value all the way to the other end of the array. Shell sort first moves values using giant step sizes, so a small value will move a long way towards its final position, with just a few comparisons and exchanges.

One can visualize Shell sort in the following way: arrange the list into a table and sort the columns (using an insertion sort). Repeat this process, each time with smaller number of longer columns. At the end, the table has only one column. While transforming the list into a table makes it easier to visualize, the algorithm itself does its sorting in-place (by incrementing the index by the step size, i.e. using i += step_size instead of i++).

For example, consider a list of numbers like [13 14 94 33 82 25 59 94 65 23 45 27 73 25 39 10]. If we started with a step-size of 5, we could visualize this as breaking the list of numbers into a table with 5 columns. This would look like this:

```
13 14 94 33 82
25 59 94 65 23
45 27 73 25 39
```

```
10
```

We then sort each column, which gives us

```
10 14 73 25 23
13 27 94 33 39
25 59 94 65 82
45
```

When read back as a single list of numbers, we get [10 14 73 25 23 13 27 94 33 39 25 59 94 65 82 45]. Here, the 10 which was all the way at the end, has moved all the way to the beginning. This list is then again sorted using a 3-gap sort as shown below.

```
10 14 73
25 23 13
27 94 33
39 25 59
94 65 82
45
```

Which gives us

```
10 14 13
25 23 33
27 25 59
39 65 73
45 94 82
94
```

Now all that remains to be done is a 1-gap sort (simple insertion sort).

Gap sequence

The *gap sequence* is an integral part of the Shell sort algorithm.

Original	32	95	16	82	24	66	35	19	75	54	40	43	93	68	
After 5-sort	32	35	16	68	24	40	43	19	75	54	66	95	93	82	6 swaps
After 3-sort	32	19	16	43	24	40	54	35	75	68	66	95	93	82	5 swaps
After 1-sort	16	19	24	32	35	40	43	54	66	68	75	82	93	95	15 swaps

The Shell sort algorithm in action

The gap sequence that was originally suggested by Donald Shell was to begin with $N/2$ and to halve the number until it reaches 1. While this sequence provides significant performance enhancements over the quadratic algorithms such as insertion sort, it can be changed slightly to further decrease the average and worst-case running times. Weiss' textbook[6] demonstrates that this sequence allows a worst case $O(n^2)$ sort, if the data is initially in the array as (small_1, large_1, small_2, large_2, ...) - that is, the upper half of the numbers are placed, in sorted order, in the even index locations and the lower end of the numbers are placed similarly in the odd indexed locations.

Perhaps the most crucial property of Shell sort is that the elements remain k-sorted even as the gap diminishes.[5]

Depending on the choice of gap sequence, Shell sort has a proven worst-case running time of $O(n^2)$ (using Shell's increments that start with 1/2 the array size and divide by 2 each time), $O(n^{3/2})$ (using Hibbard's increments of $2^k - 1$), $O(n^{4/3})$ (using Sedgewick's increments of $9 \times 4^i - 9 \times 2^i + 1$, or $4^i - 3 \times 2^i + 1$), or

$O(n \log^2 n)$ (using Pratt's increments $2^i 3^j$), and possibly unproven better running times. The existence of an $O(n$ implementation of Shell sort was precluded by Poonen, Plaxton, and Suel.[7]

The best known sequence according to research by Marcin Ciura is 1, 4, 10, 23, 57, 132, 301, 701, 1750.[8] This study also concluded that "comparisons rather than moves should be considered the dominant operation in Shellsort."[9] A Shell sort using this sequence runs faster than an insertion sort, but even if it is faster than a quicksort for small arrays, it is slower for sufficiently big arrays. In his paper Ciura writes "the greatest increments play a minor role in the overall performance of a sequence," making it reasonable to extend Ciura's sequence beyond 701 as a geometric progression with a ratio roughly that of Ciura's last two elements, 1750/701. Taking the successor of each increment *g* to be floor(*g**5/2)+1 for example would then extend Ciura's sequence as 701, 1753, 4383, 10958, 27396, 68491, 171228, 428071, 1070178, …, all pairs of which can be seen by inspection to be well decorrelated, most being relatively prime and none with common divisor greater than 12.

Another sequence which performs empirically well on large arrays is the Fibonacci numbers (leaving out one of the starting 1's) to the power of twice the golden ratio, which gives the following sequence: 1, 9, 34, 182, 836, 4025, 19001, 90358, 428481, 2034035, 9651787, 45806244, 217378076, 1031612713, ….[10]

Notes

[1] Shell, D.L. (1959). "A high-speed sorting procedure". *Communications of the ACM* **2** (7): 30–32. doi:10.1145/368370.368387.

[2] "Shell sort" (http://www.nist.gov/dads/HTML/shellsort.html). National Institute of Standards and Technology. . Retrieved 2007-07-17.

[3] Sedgewick, Robert (1998). *Algorithms in C*. Addison Wesley. pp. 273–279.

[4] Weiss, Mark Allen (1997). *Data Structures and Algorithm Analysis in C*. Addison Wesley Longman. pp. 222–226.

[5] Pratt, V (1979). *Shellsort and sorting networks (Outstanding dissertations in the computer sciences)*. Garland. ISBN 0-824-04406-1. (This was originally presented as the author's Ph.D. thesis, Stanford University, 1971)

[6] Weiss, Mark Allen (2002). *Data Structures & Problem Solving using Java*. Addison Wesley. ISBN 0-201-74835-5.

[7] Poonen, Plaxton, Suel (1992). "Improved Lower Bounds for Shellsort". *Annual Symposium on Foundations of Computer Science* (33): 226–235.

[8] A102549 (http://en.wikipedia.org/wiki/Oeis:a102549) Best known gap sequence

[9] Marcin Ciura, Best Increments for the Average Case of Shellsort, 13th International Symposium on Fundamentals of Computation Theory, Riga, Latvia, 22–24 August 2001; Lecture Notes in Computer Science 2001; Vol. 2138, pp. 106-117. (http://sun.aei.polsl.pl/~mciura/publikacje/shellsort.pdf)

[10] A154393 (http://en.wikipedia.org/wiki/Oeis:a154393) The fibonacci to the power of two times the golden ratio gap sequence

External links

- Detailed analysis of Shell sort (http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/shell/shellen.htm)
- Analysis of Shellsort and Related Algorithms, Robert Sedgewick (http://www.cs.princeton.edu/~rs/shell/)
- Animated Sorting Algorithms: Shell Sort (http://www.sorting-algorithms.com/shell-sort) – graphical demonstration and discussion of Shell sort

Article Sources and Contributors

Shell sort *Source*: http://en.wikipedia.org/w/index.php?oldid=389752770 *Contributors*: A5b, Aaron Rotenberg, Abednigo, Abhinav316, AlexPlank, Alexius08, Anizzomc, Aragorn2, Bkell, Booyabazooka, BrokenSegue, BrotherE, Bruce1ee, Bubba73, CJLL Wright, Caesura, Camw, Carribeiro, Casbah, CesarB, Chopstickles, Circular17, Clemmy, CodeHive, Crashmatrix, DFRussia, Damian Yerrick, Dangling Reference, Daniel Quinlan, David Eppstein, Dcoetzee, Dindon, Dinoen, Dmercer, Dmitry Dzhus, Donhalcon, Doradus, Dysprosia, EmilJ, Enochlau, Evileye73, Fawcett5, Fresheneesz, Gaspercat, Gfis, Giftlite, Graue, GregorB, Gtong32, HJ Mitchell, Hell11421, Hephaestos, Histrion, IanOsgood, Iridescent, JamesBWatson, Jaredwf, Jdforrester, Jirka6, Jokes Free4Me, Josh Kehn, Jwlee, K2234, Kbk, Knutux, Kx1186, LOL, LiDaobing, MCiura, MacsBug, Mark T, Mav, Maximus Rex, Melchoir, Mike Schwartz, MisterSheik, Mr Elmo, Mwtoews, Neverwinterx, Nichehole, Noaa, Oberiko, Oli Filth, OoS, Oskar Sigvardsson, Paintman, Paranoid, Persian Poet Gal, Pne, Puckly, Rafomo, RedWolf, Reyk, Rhanekom, Rhebus, Sf222, Shreeniwasiyer, Sigmalmtd, Silly rabbit, Sjtu.bzhu, Smack, Swift, Thegeneralguy, Themania, Timwi, Turketwh, Udo.bellack, Uni4dfx, Usama707, Vaughan Pratt, XJamRastafire, Yugsdrawkcabeht, Zodon, Zr2d2, 209 anonymous edits

Image Sources, Licenses and Contributors

File:Shellsort-edited.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Shellsort-edited.png *License*: Public Domain *Contributors*: User:Crashmatrix

Image:Shellsort.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Shellsort.svg *License*: Public Domain *Contributors*: Booyabazooka, Nagy, 2 anonymous edits

License