**Shellsort**

**Shellsort**, also known as **Shell sort** or **Shell's method**, is an in-place [comparison sort](https://en.wikipedia.org/wiki/Comparison_sort). It can be seen as either a generalization of sorting by exchange ([bubble sort](https://en.wikipedia.org/wiki/Bubble_sort)) or sorting by insertion ([insertion sort](https://en.wikipedia.org/wiki/Insertion_sort)).[[3]](https://en.wikipedia.org/wiki/Shellsort#cite_note-Knuth-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](https://en.wikipedia.org/wiki/Donald_Shell) published the first version of this sort in 1959.[[4]](https://en.wikipedia.org/wiki/Shellsort#cite_note-Shell-4)[[5]](https://en.wikipedia.org/wiki/Shellsort#cite_note-5) The running time of Shellsort is heavily dependent on the gap sequence it uses. For many practical variants, determining their [time complexity](https://en.wikipedia.org/wiki/Time_complexity) remains an [open problem](https://en.wikipedia.org/wiki/Open_problem).

**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

}

}

**Applications**

Shellsort performs more operations and has higher [cache miss ratio](https://en.wikipedia.org/wiki/CPU_cache#Cache_miss) than [quicksort](https://en.wikipedia.org/wiki/Quicksort). However, since it can be implemented using little code and does not use the [call stack](https://en.wikipedia.org/wiki/Call_stack), some implementations of the [qsort](https://en.wikipedia.org/wiki/Qsort) function in the [C standard library](https://en.wikipedia.org/wiki/C_standard_library) targeted at [embedded systems](https://en.wikipedia.org/wiki/Embedded_systems) use it instead of quicksort. Shellsort is, for example, used in the [uClibc](https://en.wikipedia.org/wiki/UClibc) library.[[24]](https://en.wikipedia.org/wiki/Shellsort#cite_note-24) For similar reasons, an implementation of Shellsort is present in the [Linux kernel](https://en.wikipedia.org/wiki/Linux_kernel).[[25]](https://en.wikipedia.org/wiki/Shellsort#cite_note-25)

Shellsort can also serve as a sub-algorithm of [introspective sort](https://en.wikipedia.org/wiki/Introsort), 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](https://en.wikipedia.org/wiki/Bzip2) compressor.[[26]](https://en.wikipedia.org/wiki/Shellsort#cite_note-26)