WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
فارسی
Српски / srpski
Edit links

Cycle sort

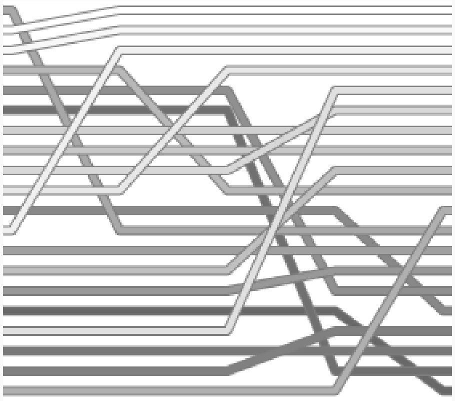
From Wikipedia, the free encyclopedia

Cycle sort is an in-place, unstable sorting algorithm, a comparison sort that is theoretically optimal in terms of the total number of writes to the original array, unlike any other in-place sorting algorithm. It is based on the idea that the permutation to be sorted can be factored into cycles, which can individually be rotated to give a sorted result.

Unlike nearly every other sort, items are *never* written elsewhere in the array simply to push them out of the way of the action. Each value is either written zero times, if it's already in its correct position, or written one time to its correct position. This matches the minimal number of overwrites required for a completed in-place sort.

Minimizing the number of writes is useful when making writes to some huge data set is very expensive, such as with EEPROMs like Flash memory where each write reduces the lifespan of the memory.

Cycle sort



Example of cycle sort sorting a list of random numbers.

Class	Sorting algorithm
Data structure	Array
Worst case performance	Θ(n^2)
Best case performance	Θ(n^2)
Average case performance	Θ(n^2)
Worst case space complexity	Θ(*n*) total, Θ(*1*) auxiliary

Algorithm [edit]

The following algorithm finds cycles and rotates them, giving a sorted result. Arrays are zero-indexed.

```
# Sort an array in place and return the number of writes.
procedure cycleSort(array):
  writes = 0

  # Loop through the array to find cycles to rotate.
  for cycleStart from 0 to length(array) - 2, inclusive:
    item = array[cycleStart]

    # Find where to put the item.
    pos = cycleStart
    for i from cycleStart + 1 to length(array) - 1, inclusive:
      if array[i] < item:
        pos += 1

    # If the item is already there, this is not a cycle.
    if pos == cycleStart:
      continue

    # Otherwise, put the item there or right after any duplicates.
    while item == array[pos]:
      pos += 1
    array[pos], item = item, array[pos]
    writes += 1

    # Rotate the rest of the cycle.
    while pos != cycleStart:
      # Find where to put the item.
      pos = cycleStart
      for i from cycleStart + 1 to length(array) - 1, inclusive
        if array[i] < item:
          pos += 1

      # Put the item there or right after any duplicates.
      while item == array[pos]:
        pos += 1
```

```
            array[pos], item = item, array[pos]
            writes += 1

    return writes
```

Situation-specific optimizations [edit]

When the array contains only duplicates of a relatively small number of items, a constant-time perfect hash function can greatly speed up finding where to put an item[1], turning the sort from $\Theta(n^2)$ time to $\Theta(n + k)$ time, where k is the total number of hashes. The array ends up sorted in the order of the hashes, so choosing a hash function that gives you the right ordering is important.

Before the sort, create a histogram, sorted by hash, counting the number of occurrences of each hash in the array. Then create a table with the cumulative sum of each entry in the histogram. The cumulative sum table will then contain the position in the array of each element. The proper place of elements can then be found by a constant-time hashing and cumulative sum table lookup rather than a linear search.

External links [edit]

^ "Cycle-Sort: A Linear Sorting Method", The Computer Journal (1990) 33 (4): 365-367.

- Original source of unrestricted variant

v · t · e	Sorting algorithms [hide]
Theory	Computational complexity theory · Big O notation · Total order · Lists · Inplacement · Stability · Comparison sort · Adaptive sort · Sorting network · Integer sorting
Exchange sorts	Bubble sort · Cocktail sort · Odd–even sort · Comb sort · Gnome sort · Quicksort · Stooge sort · Bogosort
Selection sorts	Selection sort · Heapsort · Smoothsort · Cartesian tree sort · Tournament sort · **Cycle sort**
Insertion sorts	Insertion sort · Shellsort · Splaysort · Tree sort · Library sort · Patience sorting
Merge sorts	Merge sort · Cascade merge sort · Oscillating merge sort · Polyphase merge sort · Strand sort
Distribution sorts	American flag sort · Bead sort · Bucket sort · Burstsort · Counting sort · Pigeonhole sort · Proxmap sort · Radix sort · Flashsort
Concurrent sorts	Bitonic sorter · Batcher odd–even mergesort · Pairwise sorting network
Hybrid sorts	Block sort · Timsort · Introsort · Spreadsort · JSort
Other	Topological sorting · Pancake sorting · Spaghetti sort

Categories: Sorting algorithms | Comparison sorts | Online sorts