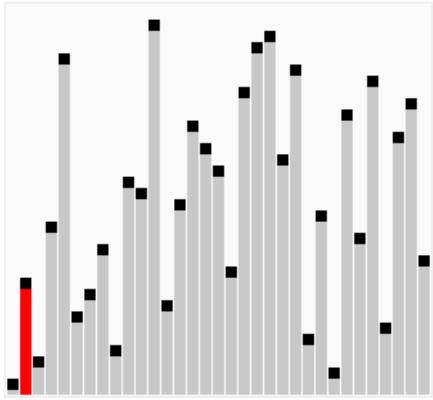
Cocktail sort

From Wikipedia, the free encyclopedia

Cocktail sort, also known as **bidirectional bubble sort**, **cocktail shaker sort**, **shaker sort** (which can also refer to a variant of selection sort), **ripple sort**, **shuffle sort**,[1] or **shuttle sort**, is a variation of bubble sort that is both a stable sorting algorithm and a comparison sort. The algorithm differs from a bubble sort in that it sorts in both directions on each pass through the list. This sorting algorithm is only marginally more difficult to implement than a bubble sort, and solves the problem of turtles in bubble sorts. It provides only marginal performance improvements, and does not improve asymptotic performance; like the bubble sort, it is not of practical interest (insertion sort is preferred for simple sorts), though it finds some use in education.

Cocktail sort



Class	Sorting algorithm
Data structure	Array
Worst case performance	$O(n^2)$
Best case performance	$O(n)$
Average case performance	$O(n^2)$
Worst case space complexity	$O(1)$

Pseudocode [edit]

The simplest form of cocktail sort goes through the whole list each time:

```
procedure cocktailSort( A : list of sortable items ) defined as:
  do
    swapped := false
    for each i in 0 to length( A ) - 2 do:
      if A[ i ] > A[ i + 1 ] then // test whether the two elements are in the wrong
order
        swap( A[ i ], A[ i + 1 ] ) // let the two elements change places
        swapped := true
      end if
    end for
    if swapped = false then
      // we can exit the outer loop here if no swaps occurred.
      break do-while loop
    end if
    swapped := false
    for each i in length( A ) - 2 to 0 do:
      if A[ i ] > A[ i + 1 ] then
        swap( A[ i ], A[ i + 1 ] )
        swapped := true
      end if
    end for
  while swapped // if no elements have been swapped, then the list is sorted
end procedure
```

The first rightward pass will shift the largest element to its correct place at the end, and the following leftward pass will shift the smallest element to its correct place at the beginning. The second complete pass will shift the second largest and second smallest elements to their correct places, and so on. After *i* passes, the first *i* and the last *i* elements in the list are in their correct positions, and do not need to be checked. By shortening the

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
Català
Čeština
Deutsch
Español
فارسی
Français
한국어
Հայերեն
Italiano
Қазақша
Magyar
日本語
Polski
Português
Русский
Slovenčina
Српски / srpski
Türkçe
Українська
中文
Edit links

part of the list that is sorted each time, the number of operations can be halved (see bubble sort).

```
procedure cocktailSort( A : list of sortable items ) defined as:
  // `begin` and `end` marks the first and last index to check
  begin := -1
  end := length( A ) - 2
  do
    swapped := false
    // increases `begin` because the elements before `begin` are in correct order
    begin := begin + 1
    for each i in begin to end do:
      if A[ i ] > A[ i + 1 ] then
        swap( A[ i ], A[ i + 1 ] )
        swapped := true
      end if
    end for
    if swapped = false then
      break do-while loop
    end if
    swapped := false
    // decreases `end` because the elements after `end` are in correct order
    end := end - 1
    for each i in end to begin do:
      if A[ i ] > A[ i + 1 ] then
        swap( A[ i ], A[ i + 1 ] )
        swapped := true
      end if
    end for
  while swapped
end procedure
```

This is an example of the algorithm in MATLAB/OCTAVE.

```
function A = cocktailSort(A)
% `beginIdx` and `endIdx` marks the first and last index to check
beginIdx = 0;
endIdx = length(A)-1;
swapped = false;
while ~swapped
    swapped = false;
    % increases `beginIdx` because the elements before `beginIdx` are in correct
order
    beginIdx = beginIdx + 1;
    for ii= beginIdx:endIdx
        if A(ii) > A(ii + 1)
            [A(ii+1), A(ii)] = deal(A(ii), A(ii+1));
            swapped = true;
        end
    end
    if ~swapped
        break;
    end
    swapped = false;
    % decreases `endIdx` because the elements after `endIdx` are in correct order
    endIdx = endIdx - 1;
    for ii= endIdx:-1:beginIdx
        if A(ii) > A(ii + 1)
            [A(ii+1), A(ii)] = deal(A(ii), A(ii+1));
            swapped = true;
        end
    end
    swapped = false;
end
end
```

Differences from bubble sort [edit]

Cocktail sort is a slight variation of bubble sort. It differs in that instead of repeatedly passing through the list from bottom to top, it passes alternately from bottom to top and then from top to bottom. It can achieve slightly

better performance than a standard bubble sort. The reason for this is that bubble sort only passes through the list in one direction and therefore can only move items backward one step each iteration.

An example of a list that proves this point is the list (2,3,4,5,1), which would only need to go through one pass of cocktail sort to become sorted, but if using an ascending bubble sort would take four passes. However one cocktail sort pass should be counted as two bubble sort passes. Typically cocktail sort is less than two times faster than bubble sort.

Another optimization can be that the algorithm remembers where the last actual swap has been done. In the next iteration, there will be no swaps beyond this limit and the algorithm has shorter passes. As the Cocktail sort goes bidirectionally, the range of possible swaps, which is the range to be tested, will reduce per pass, thus reducing the overall running time.

Complexity [edit]

The complexity of cocktail sort in big O notation is $O(n^2)$ for both the worst case and the average case, but it becomes closer to $O(n)$ if the list is mostly ordered before applying the sorting algorithm, for example, if every element is at a position that differs at most k (k ≥ 1) from the position it is going to end up in, the complexity of cocktail sort becomes $O(k * n)$. Such cases may be approached by algorithms like comb sort.

Cocktail sort is also briefly discussed in the book *The Art of Computer Programming*, along with similar refinements of bubble sort. In conclusion, Knuth states about bubble sort and its improvements (Knuth 1998, p. 110):

> But none of these refinements leads to an algorithm better than straight insertion [that is, insertion sort]; and we already know that straight insertion isn't suitable for large *N*. [...] In short, the bubble sort seems to have nothing to recommend it, except a catchy name and the fact that it leads to some interesting theoretical problems.
>
> —D. E. Knuth[2]

Notes [edit]

1. ^ Martin Duhl: Die schrittweise Entwicklung und Beschreibung einer Shuffle-Sort-Array Schaltung in HYPERKARL aus der Algorithmischen Darstellung des BUBBLE-SORT-ALGORITHMUS, Projektarbeit, 1986, Technical University of Kaiserslautern
2. ^ *The Art of Computer Programming*, Vol. 3: Sorting and Searching, Second Edition, Addison-Wesley, 1998.

References [edit]

- Paul E. Black and Bob Bockholt, "bidirectional bubble sort", in Dictionary of Algorithms and Data Structures (online), Paul E. Black, ed., U.S. National Institute of Standards and Technology. 24 August 2009. (accessed: 5 Feb 2010)
- R. Hartenstein: THE GRAND CHALLENGE TO REINVENT COMPUTING - A new World Model of Computing; Proc. CSBC_2010, July 20–23, 2010, Belo Horizonte, Brasil, [1]

External links [edit]

- Interactive demo of cocktail sort
- Java source code and an animated demo of cocktail sort (called bi-directional bubble sort) and several other algorithms
- .NET Implementation of cocktail sort and several other algorithms

The Wikibook *Algorithm implementation* has a page on the topic of: ***Cocktail sort***

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 | Stable sorts

This page was last modified on 12 July 2015, at 21:12.

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.

Privacy policy About Wikipedia Disclaimers Contact Wikipedia Developers Mobile view