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**, **shuttle sort** or **happy hour sort**, is a variation of bubble sort that is both a stable sorting algorithm and a comparison sort. The algorithm differs from bubble sort in that sorts in both directions each pass through the list. This sorting algorithm is only marginally more difficult than bubble sort to implement, and solves the problem with so-called turtles in bubble sort.

Cocktail sort

Class	Sorting algorithm
Data structure	Array
Worst case performance	$O(n^2)$
Optimal	No

Contents

- 1 Pseudocode
- 2 Differences from bubble sort
- 3 Complexity
- 4 External links

Pseudocode

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 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
```

Differences from bubble sort

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

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 ($k \geq 1$) from the position it is going to end up in, the complexity of cocktail sort becomes $O(k * n)$.

External links

- Java source code and an animated demo of cocktail sort (called bi-directional bubble sort) and several other algorithms (http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html)
- .NET Implementation of cocktail sort and several other algorithms (http://www.sharpdeveloper.net/content/archive/2007/08/14/dot-net-data-structures-and-algorithms.aspx)

Retrieved from "http://en.wikipedia.org/wiki/Cocktail_sort"
Categories: Sorting algorithms | Comparison sorts | Stable sorts | Articles with example pseudocode

- This page was last modified on 15 June 2009 at 04:56.

- Text is available under the Creative Commons Attribution/Share-Alike License; additional terms may apply. See Terms of Use for details.
  Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a U.S. registered 501(c)(3) tax-deductible nonprofit charity.