

Bubble sort

Bubble sort, sometimes referred to as **sinking sort**, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. The algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list. Although the algorithm is simple, it is too slow and impractical for most problems even when compared to insertion sort.^[2] Bubble sort can be practical if the input is in mostly sorted order with some out-of-order elements nearly in position.

Contents

Analysis

Performance

Rabbits and turtles

Step-by-step example

Implementation

Pseudocode implementation

Optimizing bubble sort

Use

Variations

Debate over name


In popular culture

Notes

References

External links

Bubble sort



Static visualization of bubble sort^[1]

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

Analysis

Performance

Bubble sort has a worst-case and average complexity of $O(n^2)$, where n is the number of items being sorted. Most practical sorting algorithms have substantially better worst-case or average complexity, often $O(n \log n)$. Even other $O(n^2)$ sorting algorithms, such as insertion sort, generally run faster than bubble sort, and are no more complex. Therefore, bubble sort is not a practical sorting algorithm.

The only significant advantage that bubble sort has over most other algorithms, even quicksort, but not insertion sort, is that the ability to detect that the list is sorted efficiently is built into the algorithm. When the list is already sorted (best-case), the complexity of bubble sort is only $O(n)$. By contrast, most other algorithms, even those with better average-case complexity, perform their entire sorting process on the set and thus are more complex. However, not only does insertion sort share this advantage, but it also performs better on a list that is substantially sorted (having a small number of inversions).

Bubble sort should be avoided in the case of large collections. It will not be efficient in the case of a reverse-ordered collection.

6 5 3 1 8 7 2 4

An example of bubble sort. Starting from the beginning of the list, compare every adjacent pair, swap their position if they are not in the right order (the latter one is smaller than the former one). After each iteration, one less element (the last one) is needed to be compared until there are no more elements left to be compared.

Rabbits and turtles

The distance and direction that elements must move during the sort determine bubble sort's performance because elements move in different directions at different speeds. An element that must move toward the end of the list can move quickly because it can take part in successive swaps. For example, the largest element in the list will win every swap, so it moves to its sorted position on the first pass even if it starts near the beginning. On the other hand, an element that must move toward the beginning of the list cannot move faster than one step per pass, so elements move toward the beginning very slowly. If the smallest element is at the end of the list, it will take $n-1$ passes to move it to the beginning. This has led to these types of elements being named rabbits and turtles, respectively, after the characters in Aesop's fable of The Tortoise and the Hare.

Various efforts have been made to eliminate turtles to improve upon the speed of bubble sort. Cocktail sort is a bi-directional bubble sort that goes from beginning to end, and then reverses itself, going end to beginning. It can move turtles fairly well, but it retains $O(n^2)$ worst-case complexity. Comb sort compares elements separated by large gaps, and can move turtles extremely quickly before proceeding to smaller and smaller gaps to smooth out the list. Its average speed is comparable to faster algorithms like quicksort.

Step-by-step example

Take an array of numbers " 5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort. In each step, elements written in **bold** are being compared. Three passes will be required;

First Pass

(**5** **1** 4 2 8) → (**1** **5** 4 2 8), Here, algorithm compares the first two elements, and swaps since $5 > 1$.

(1 **5** **4** 2 8) → (1 **4** **5** 2 8), Swap since $5 > 4$

(1 4 **5** **2** 8) → (1 4 **2** **5** 8), Swap since $5 > 2$

(1 4 2 **5** **8**) → (1 4 2 **5** **8**), Now, since these elements are already in order ($8 > 5$), algorithm does not swap them.

Second Pass

(1 4 2 5 8) → (1 4 2 5 8)
(1 4 2 5 8) → (1 2 4 5 8), Swap since 4 > 2
(1 2 4 5 8) → (1 2 4 5 8)
(1 2 4 5 8) → (1 2 4 5 8)

Now, the array is already sorted, but the algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass

(1 2 4 5 8) → (1 2 4 5 8)
(1 2 4 5 8) → (1 2 4 5 8)
(1 2 4 5 8) → (1 2 4 5 8)
(1 2 4 5 8) → (1 2 4 5 8)

Implementation

Pseudocode implementation

In pseudocode the algorithm can be expressed as (0-based array):

```
procedure bubbleSort( A : list of sortable items )
  n = length( A )
  repeat
    swapped = false
    for i = 1 to n - 1 inclusive do
      /* if this pair is out of order */
      if A[i - 1] > A[i] then
        /* swap them and remember something changed */
        swap( A[i - 1], A[i] )
        swapped = true
      end if
    end for
  until not swapped
end procedure
```

Optimizing bubble sort

The bubble sort algorithm can be easily optimized by observing that the n -th pass finds the n -th largest element and puts it into its final place. So, the inner loop can avoid looking at the last $n - 1$ items when running for the n -th time:

```
procedure bubbleSort( A : list of sortable items )
  n = length( A )
  repeat
    swapped = false
    for i = 1 to n - 1 inclusive do
      if A[i - 1] > A[i] then
        swap( A[i - 1], A[i] )
        swapped = true
      end if
    end for
    n = n - 1
  until not swapped
end procedure
```

More generally, it can happen that more than one element is placed in their final position on a single pass. In particular, after every pass, all elements after the last swap are sorted, and do not need to be checked again. This allows to skip over many elements, resulting in about a worst case 50% improvement in

comparison count (though no improvement in swap counts), and adds very little complexity because the new code subsumes the "swapped" variable:

To accomplish this in pseudocode, the following can be written:

```
procedure bubbleSort( A : list of sortable items )
  n = length( A )
  repeat
    newn = 0
    for i = 1 to n - 1 inclusive do
      if A[i - 1] > A[i] then
        swap( A[i - 1], A[i] )
        newn = i
      end if
    end for
    n = newn
  until n <= 1
end procedure
```

Alternate modifications, such as the [cocktail shaker sort](#) attempt to improve on the bubble sort performance while keeping the same idea of repeatedly comparing and swapping adjacent items.

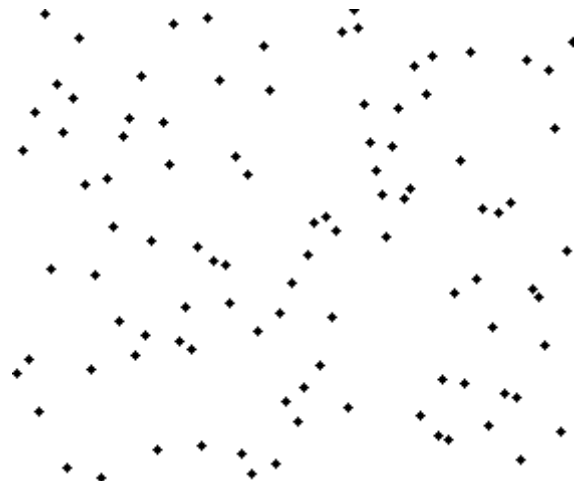
Use

Although bubble sort is one of the simplest sorting algorithms to understand and implement, its $O(n^2)$ complexity means that its efficiency decreases dramatically on lists of more than a small number of elements. Even among simple $O(n^2)$ sorting algorithms, algorithms like [insertion sort](#) are usually considerably more efficient.

Due to its simplicity, bubble sort is often used to introduce the concept of an algorithm, or a sorting algorithm, to introductory [computer science](#) students. However, some researchers such as [Owen Astrachan](#) have gone to great lengths to disparage bubble sort and its continued popularity in computer science education, recommending that it no longer even be taught.^[3]

The [Jargon File](#), which famously calls [bogosity](#) "the archetypical [sic] perversely awful algorithm", also calls bubble sort "the generic bad algorithm".^[4] [Donald Knuth](#), in *The Art of Computer Programming*, concluded that "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", some of which he then discusses.^[2]

Bubble sort is [asymptotically](#) equivalent in running time to insertion sort in the worst case, but the two algorithms differ greatly in the number of swaps necessary. Experimental results such as those of Astrachan have also shown that insertion sort performs considerably better even on random lists. For



A bubble sort, a sorting algorithm that continuously steps through a list, swapping items until they appear in the correct order. The list was plotted in a Cartesian coordinate system, with each point (x, y) indicating that the value y is stored at index x. Then the list would be sorted by bubble sort according to every pixel's value. Note that the largest end gets sorted first, with smaller elements taking longer to move to their correct positions.

these reasons many modern algorithm textbooks avoid using the bubble sort algorithm in favor of insertion sort.

Bubble sort also interacts poorly with modern CPU hardware. It produces at least twice as many writes as insertion sort, twice as many cache misses, and asymptotically more branch mispredictions. Experiments by Astrachan sorting strings in Java show bubble sort to be roughly one-fifth as fast as an insertion sort and 70% as fast as a selection sort.^[3]

In computer graphics bubble sort is popular for its capability to detect a very small error (like swap of just two elements) in almost-sorted arrays and fix it with just linear complexity ($2n$). For example, it is used in a polygon filling algorithm, where bounding lines are sorted by their x coordinate at a specific scan line (a line parallel to the x axis) and with incrementing y their order changes (two elements are swapped) only at intersections of two lines. Bubble sort is a stable sort algorithm, like insertion sort.

Variations

- Odd–even sort is a parallel version of bubble sort, for message passing systems.
- Passes can be from right to left, rather than left to right. This is more efficient for lists with unsorted items added to the end.
- Cocktail shaker sort alternates leftwards and rightwards passes.

Debate over name

Bubble sort has been occasionally referred to as a "sinking sort".^[5]

For example, in Donald Knuth's *The Art of Computer Programming*, Volume 3: *Sorting and Searching* he states in section 5.2.1 'Sorting by Insertion', that [the value] "settles to its proper level" and that this method of sorting has sometimes been called the *sifting* or *sinking* technique.

This debate is perpetuated by the ease with which one may consider this algorithm from two different but equally valid perspectives:

1. The *larger* values might be regarded as *heavier* and therefore be seen to progressively *sink* to the *bottom* of the list
2. The *smaller* values might be regarded as *lighter* and therefore be seen to progressively *bubble up* to the *top* of the list.

In popular culture

Google CEO Eric Schmidt asked president Barack Obama once during an interview about the best way to sort one million integers – and Obama, pausing for a moment, then replied: "I think the bubble sort would be the wrong way to go." ^{[6][7]}

Notes

1. Cortesi, Aldo (27 April 2007). "Visualising Sorting Algorithms" (<https://corte.si/posts/code/visualisingsorting/index.html>). Retrieved 16 March 2017.
2. Donald Knuth. *The Art of Computer Programming*, Volume 3: *Sorting and Searching*, Second Edition. Addison-Wesley, 1998. ISBN 0-201-89685-0. Pages 106–110 of section

5.2.2: Sorting by Exchanging. "[A]lthough the techniques used in the calculations [to analyze the bubble sort] are instructive, the results are disappointing since they tell us that the bubble sort isn't really very good at all. Compared to straight insertion [...], bubble sorting requires a more complicated program and takes about twice as long!" (Quote from the first edition, 1973.)

3. Astrachan, Owen (2003). "Bubble sort: an archaeological algorithmic analysis" (<http://www.cs.duke.edu/~ola/papers/bubble.pdf>) (PDF). *ACM SIGCSE Bulletin*. **35** (1): 1–5. doi:10.1145/792548.611918 (<https://doi.org/10.1145%2F792548.611918>). ISSN 0097-8418 (<https://www.worldcat.org/issn/0097-8418>).
4. "jargon, node: bogo-sort" (<http://www.jargon.net/jargonfile/b/bogo-sort.html>).
5. Black, Paul E. (24 August 2009). "bubble sort" (<https://xlinux.nist.gov/dads/HTML/bubblesort.html>). *Dictionary of Algorithms and Data Structures*. National Institute of Standards and Technology. Retrieved 1 October 2014.
6. OBAMA PASSES HIS GOOGLE INTERVIEW (<https://www.wired.com/2007/11/obama-elect-me/>) - Wired.com
7. Barack Obama, Eric Schmidt (Nov 14, 2007). *Barack Obama | Candidates at Google* (<http://web.archive.org/web/20190907131624/https://www.youtube.com/watch?v=m4yVIPqeZwo>) (Youtube). Mountain View, CA 94043 The Googleplex: Talks at Google. Event occurs at 23:20. Archived from the original (<https://www.youtube.com/watch?v=m4yVIPqeZwo>) (Video) on Sep 7, 2019. Retrieved Sep 18, 2019.

References

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Problem 2-2, pg.40.
- Sorting in the Presence of Branch Prediction and Caches (<https://www.cs.tcd.ie/publications/tech-reports/reports.05/TCD-CS-2005-57.pdf>)
- Fundamentals of Data Structures by Ellis Horowitz, Sartaj Sahni and Susan Anderson-Freed ISBN 81-7371-605-6
- Owen Astrachan. Bubble Sort: An Archaeological Algorithmic Analysis (<https://users.cs.duke.edu/~ola/bubble/bubble.html>)

External links

- Martin, David R. (2007). "Animated Sorting Algorithms: Bubble Sort" (<https://web.archive.org/web/20150303084352/http://www.sorting-algorithms.com/bubble-sort>). Archived from the original (<http://www.sorting-algorithms.com/bubble-sort>) on 2015-03-03. – graphical demonstration
- "Lafore's Bubble Sort" (<http://lecture.ecc.u-tokyo.ac.jp/~ueda/JavaApplet/BubbleSort.html>). (Java applet animation)
- OEIS sequence A008302 (Table (statistics) of the number of permutations of [n] that need k pair-swaps during the sorting) (<https://oeis.org/A008302>)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Bubble_sort&oldid=929850892"

This page was last edited on 8 December 2019, at 17:26 (UTC).

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.

