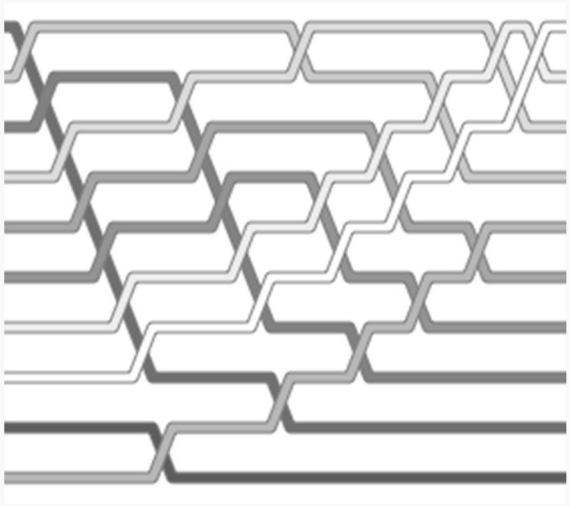
Bubble sort

From Wikipedia, the free encyclopedia
Jump to: navigation, search

Bubble sort



A visual representation of how bubble sort works.

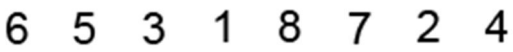
Class	Sorting algorithm
Data structure	Array
Worst case performance	$O(n^2)$
Best case performance	$O(n)$
Average case performance	$O(n^2)$
Worst case space complexity	$O(1)$ auxiliary

Bubble sort, also known as **sinking sort**, is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements "bubble" to the top of the list. Because it only uses comparisons to operate on elements, it is a comparison sort. Although the algorithm is simple, it is not efficient for sorting large lists; other algorithms are better.

Contents

[hide]

[edit] Analysis



An example on 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 is no more element left to be compared.

[edit] Performance

Bubble sort has worst-case and average complexity both $O(n^2)$, where n is the number of items being sorted. There exist many sorting algorithms with substantially better worst-case or average complexity of $O(n \log n)$. Even other $O(n^2)$ sorting algorithms, such as insertion sort, tend to have better performance than bubble sort. Therefore, bubble sort is not a practical sorting algorithm when n is large.

The only significant advantage that bubble sort has over most other implementations, even quicksort, but not insertion sort, is that the ability to detect that the list is sorted is efficiently built into the algorithm. Performance of bubble sort over an already-sorted list (best-case) is $O(n)$. 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 have this mechanism too, but it also performs better on a list that is substantially sorted (having a small number of inversions).

[edit] Rabbits and turtles

The positions of the elements in bubble sort will play a large part in determining its performance. Large elements at the beginning of the list do not pose a problem, as they are quickly swapped. Small elements towards the end, however, move to the beginning extremely slowly. This has led to these types of elements being named rabbits and turtles, respectively.

Various efforts have been made to eliminate turtles to improve upon the speed of bubble sort. Cocktail sort achieves this goal 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.

[edit] Step-by-step example

Let us take the array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort algorithm. 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 them.
(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 our 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**)

[edit] Implementation

[edit] Pseudocode implementation

The algorithm can be expressed as:

```
procedure bubbleSort( A : list of sortable items )
  repeat
```

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

[edit] 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 us to skip over a lot of the 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 we write the following:

```
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 = 0
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.

[edit] In practice

A bubble sort, a sorting algorithm that continuously steps through a list, swapping items until they appear in the correct order. Note that the largest end gets sorted first, with smaller elements taking longer to move to their correct positions.

Although bubble sort is one of the simplest sorting algorithms to understand and implement, its $O(n^2)$ complexity means it is far too inefficient for use on lists having more than a few 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.[1]

The Jargon file, which famously calls bogosort "the archetypical perversely awful algorithm", also calls bubble sort "the generic **bad** algorithm".[2] Donald Knuth, in his famous book *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.[3]

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 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 requires 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 5 times slower than insertion sort and 40% slower than selection sort.[1]

In computer graphics it 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 x axis) and with incrementing y their order changes (two elements are swapped) only at intersections of two lines.

[edit] Variations

- Odd-even sort is a parallel version of bubble sort, for message passing systems.
- Cocktail sort is another parallel version of the bubble sort
- In some cases, the sort works from right to left (the opposite direction), which is more appropriate for partially sorted lists, or lists with unsorted items added to the end.

[edit] Misnomer

Bubble sort has incorrectly been called sinking sort. Sinking sort is correctly an alias for insertion sort. This error is largely due to the National Institute of Standards and Technology listing sinking sort as an alias for bubble sort. [1] 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" this method of sorting has often been called the *sifting* or *sinking* technique.

To clarify we can also observe the behaviour of the two algorithms. In bubble sort, the larger bubbles (higher values) bubble up displacing the smaller bubbles (lower values). Insertion on the other hand, sinks each successive value down to its correct location in the sorted portion of the collection.

[edit] Notes

1. ^ *a* *b* Owen Astrachan. Bubble Sort: An Archaeological Algorithmic Analysis. SIGCSE 2003 Hannan Akhtar . (pdf)
2. ^ http://www.jargon.net/jargonfile/b/bogo-sort.html
3. ^ 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.

[edit] References

- Donald Knuth. *The Art of Computer Programming*, Volume 3: *Sorting and Searching*, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89685-0. Pages 106–110 of section 5.2.2: Sorting by Exchanging.
- 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.38.
- Sorting in the Presence of Branch Prediction and Caches
- Fundamentals of Data Structures by Ellis Horowitz, Sartaj Sahni and Susan Anderson-Freed ISBN 81-7371-605-6