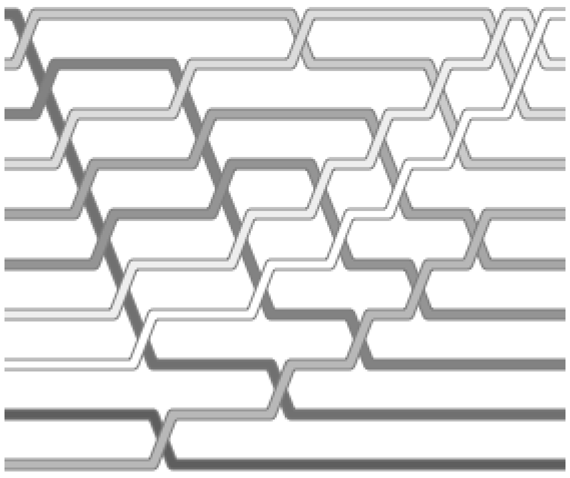
Bubble sort

From Wikipedia, the free encyclopedia

Bubble sort is a simple sorting algorithm. It 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.

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	$O(n)$ total, $O(1)$ auxiliary
Optimal	No

Contents

Analysis

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.

However, one significant advantage that BubbleSort has over most other implementations, even QuickSort, is that the ability to detect that the list is sorted is efficiently built into the algorithm. Performance of BubbleSort 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, InsertionSort also has this mechanism, and also performs better on a list that is substantially sorted (having a small number of inversions).

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 large gaps apart 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

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.

First Pass:
(**5** **1** 4 2 8) $\longrightarrow$ (**1** **5** 4 2 8), Here, algorithm compares the first two elements, and swaps them.
(1 **5** **4** 2 8) $\longrightarrow$ (1 **4** **5** 2 8), Swap since 5 > 4
(1 4 **5** **2** 8) $\longrightarrow$ (1 4 **2** **5** 8), Swap since 5 > 2
(1 4 2 **5** **8**) $\longrightarrow$ (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) $\longrightarrow$ (**1** **4** 2 5 8)
(1 **4** **2** 5 8) $\longrightarrow$ (1 **2** **4** 5 8)
(1 2 **4** **5** 8) $\longrightarrow$ (1 2 **4** **5** 8)
(1 2 4 **5** **8**) $\longrightarrow$ (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) $\longrightarrow$ (**1** **2** 4 5 8)
(1 **2** **4** 5 8) $\longrightarrow$ (1 **2** **4** 5 8)
(1 2 **4** **5** 8) $\longrightarrow$ (1 2 **4** **5** 8)
(1 2 4 **5** **8**) $\longrightarrow$ (1 2 4 **5** **8**)
Finally, the array is sorted, and the algorithm can terminate.

Pseudocode implementation

The algorithm can be expressed as:

```
procedure bubbleSort( A : list of sortable items ) defined as:
  do
    swapped := false
    for each i in 0 to length(A) - 2 inclusive do:
      if A[i] > A[i+1] then
        swap( A[i], A[i+1] )
        swapped := true
      end if
    end for
  while swapped
end procedure
```

Alternative implementations

The performance of bubble sort can be improved marginally in the following manner. First observe that, after each comparison (and contingent swap), the largest element encountered in the current pass will reside in the last position traversed. Therefore, after the first pass, the largest element in the array will be in the very last position; that is, given a list of size *n*, the *n^{th}* element will be in its final position after the first pass. Thus, inductively, it suffices to sort the remaining *$n - 1$* elements: after the second pass, the *$n - 1^{th}$* element will be in its final place, and so on. So each pass can be one step shorter than the previous pass, instead of every pass continuing to traverse all the elements at the end, which are already in their final positions and will not move in any case.

To accomplish this in pseudocode we write the following:

```
procedure bubbleSort( A : list of sortable items ) defined as:
  n := length( A )
  do
    swapped := false
    for each i in 0 to n - 2  inclusive do:
      if A[ i ] > A[ i + 1 ] then
        swap( A[ i ], A[ i + 1 ] )
        swapped := true
      end if
    end for
    n := n - 1
  while swapped
end procedure
```

Here, instead of doing as many as $n \cdot (n-1)$ comparisons, we do at most $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$ comparisons (given by an arithmetic series). So the execution time is still in $O(n^2)$ but in the worst case (the input being a reverse-sorted array) bubble sort with this improvement is twice as fast as without.

In practice

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.

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]

Variations

- Odd-even sort is a parallel version of bubble sort, for message passing systems.
- 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.

Notes

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

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

External links

- Bubble Sort in 29 languages
- Animated Sorting Algorithms: Bubble Sort – graphical demonstration and discussion of bubble sort
- Bubble Sort Demo
- Bubble Sort Demonstration
- Lafore's Bubble Sort
- Sorting Applets in C++
- Analyze Bubble Sort in an online Javascript IDE
- A colored graphical Java applet which allows experimentation with initial state and shows statistics
- BubbleSort tutorial, code, and a simple example
- An animated video that explains bubble sort and quick sort and compares their performance.
- Static visualization of the action of bubble sort.