Bubble Sort: An Archaeological Algorithmic Analysis

Owen Astrachan [1]
Computer Science Department
Duke University
ola@cs.duke.edu

Abstract

Text books, including books for general audiences, invariably mention *bubble sort* in discussions of elementary sorting algorithms. We trace the history of bubble sort, its popularity, and its endurance in the face of pedagogical assertions that code and algorithmic examples used in early courses should be of high quality and adhere to established best practices. This paper is more an historical analysis than a philosophical treatise for the exclusion of bubble sort from books and courses. However, sentiments for exclusion are supported by Knuth [17], "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." Although bubble sort may not be a *best practice* sort, perhaps the weight of history is more than enough to compensate and provide for its longevity.

Categories and Subject Descriptors K.3.2 [**Computers & Education**]: Computer & Information Science Education — *Computer Science Education*

General Terms Algorithms, Measurement, Theory

Keywords Analysis, Performance, Bubble sort

[1] This work was supported by NSF grants CAREER 9702550 and CRCD 0088078.

Permission to make digital or hand copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, require prior specific permission and/or a fee.
SIGCSE '03, February 19-23, Reno, Nevada, USA.
Copyright 2003 ACM 1-58113-648-X/03/0002....$5.00

1 Introduction

What do students remember from their first programming courses after one, five, and ten years? Most students will take only a few memories of what they have studied. As teachers of these students we should ensure that what they remember will serve them well. More specifically, if students take only a few memories about sorting from a first course what do we want these memories to be? Should the phrase *Bubble Sort* be the first that springs to mind at the end of a course or several years later? There are compelling reasons for excluding discussion of bubble sort[1], but many texts continue to include discussion of the algorithm after years of warnings from scientists and educators. For example, in a popular new breadth-first text [6] bubble sort is given equal footing with selection sort and quicksort in online student exercises.

Starting with Knuth's premise that "bubble sort seems to have nothing to recommend it"[17], we trace the origins and continued popularity of the algorithm from its earliest days as an unnamed sort to its current status as perhaps the most popular $O(n^2)$ sort (see below) despite wide-spread ridicule. We began this study with the intent to document (and ridicule) the continued popularity of bubble sort, but the algorithmic archaeological investigation has proven more interesting than casting aspersions.

Our initial premise that bubble sort should not be studied is reflected in [23] with a warning for potential misuse.

> For $N < 50$, roughly, the method of *straight insertion* is concise and fast enough. We include it with some trepidation: it is an N^2 algorithm, whose potential for misuse (by using it for too large an N) is great. The resultant waste of computer time is so awesome, that we were tempted not to include any

[1] A discussion of bubble sort with warnings that the performance is bad and the code isn't simple (arguably) is like telling someone "don't think about pink elephants."

N^2 routine at all. We *will* draw the line, however, at the inefficient N^2 algorithm *bubble sort*. If you know what bubble sort is, wipe it from your mind; if you don't know, make a point of never finding out!

This sentiment is similar to the reference to bubble sort found in [1], where it says of *bogo sort*, "The archetypical perversely awful algorithm (as opposed to bubble sort, which is merely the generic *bad* algorithm)."

In Section 2 we trace the origin of the algorithm, both in name and in code. In Section 3 we analyze the performance of the algorithm and the simplicity of the code. In Section 4 we summarize our study.

2 Origins of Bubble Sort

In an effort to determine why bubble sort is popular we traced its origins. Knuth [17] does not provide information on the origin of the name, though he does provide a 1956 reference [10] to an analysis of the algorithm. That paper refers to "sorting by exchange", but not to bubble sort. An extensive bibliography and sequence of articles from the 1962 ACM Conference on Sorting [11] do not use the term bubble sort, although the "sorting by exchange" algorithm is mentioned. With no obvious definitive origin of the name "bubble sort", we investigated its origins by consulting early journal articles as well as professional and pedagogical texts.

An early (1959) book on programming [21] devotes a chapter to sorting, but uses the term *exchange sorting* rather than bubble sort. The same term is used in a 1962 [4] JACM article as well as in the earlier (1961, submitted 1959) [9] JACM article referenced as the definitive source. Iverson uses the name "bubble sort" in 1962 [13]; this appears to be the first use of the term in print. As we note below, each work cited in [13] uses a phrase other than "bubble sort" to describe the algorithm we describe in Section 2.1. This reinforces the claim that Iverson is the first to use the term, though obviously not conclusively. However, we could find no work published after 1962 with a reference to bubble sort in an earlier publication than Iverson's.

Despite these earlier publications the algorithm officially enters the ACM algorithm repository as Algorithm 175 [25] in 1963 where it is named *Shuttle Sort*. Soon thereafter [14] the published algorithm is found to be "not free from errors", a gentle way of saying the published code is wrong. There a modified version of the code (that stops early when no swaps are made) is given and the author says that in this form the code was "studied in this form on the ORDVAC computer, Aberdeen Proving Ground, in 1955."

2.1 Bubble Sort, The Code

Taking the description of bubble sort in [17] as definitive the code below is bubble sort.[2] This version "bubbles" the largest elements to the end of the vector.

```
void BubbleSort(Vector a, int n)
{
    for(int j=n-1; j > 0; j--)
        for(int k=0; k < j; k++)
            if (a[k+1] < a[k])
                Swap(a,k,k+1);
}
```

Nearly every description of bubble sort describes how to terminate the sort early if the vector becomes sorted. This optimization requires checking if any swaps are made and terminating if no swaps are made after j iterations of the inner loop.

Another optimization is to alternate the direction of bubbling each time the inner loop iterates. This is *shaker sort* or *cocktail shaker sort* (see, e.g., [17, 13]).

2.2 A Sort by Any Other Name ...

Nomenclature is interesting. An early (1963) Fortran textbook [22] refers to the following code as "jump-down" sort.

```
void JumpDownSort(Vector a, int n)
{
    for(int j=n-1; j > 0; j--)
        for(int k=0; k < j; k++)
            if (a[j] < a[k])
                Swap(a,k,j);
}
```

Bubble sort as we've identified it is called a "push-down" sort. In another early (1962) work [19] the "jump-down" version is presented first in the book, with no name. The bubble sort follows also with no name. In two early works [3, 10] the jump-down sort is referred to as selection sort. Bubble sort is also covered, but referred to as *sorting by repeated comparison* and *exchanging*, respectively. In the latter paper, one of the earliest works comparing algorithms, the exchange/bubble sort is described thus: "Exchanging requires at least twice as many non-housekeeping comparisons as Inserting and for most computers will be inferior".

In moving from bubble sort to jump-down sort the only change to the code above is that one array index has been changed from $k+1$ to j. How does this differ from bubble sort? After m iterations of the outer loop the last m elements are in their final position—the same

[2] This code is close to legal in both C++ and Java and should be readable by anyone with a working knowledge of Algol-like languages.

invariant as bubble sort. However, items are not "bubbled" to the top; this code implements what is essentially a selection sort, but the current maximal element is stored/swapped into location $a[j]$ on each pass. We see why early works sometimes refer to jump-down as selection sort.

2.3 Origins of Popularity

In a survey article on sorting appearing in 1971 [20] the first sort discussed is bubble sort. It is endorsed with the following qualities.

> However, the bubble sort is easy to remember and to program, and little time is required to complete a single step.

As mentioned above, Knuth [17] belittles bubble sort. The first edition of this book appeared in 1973, but perhaps did not have the same influence as did early textbooks of the same period.[3]

In [2] (arguably the first "real" textbook on algorithms and algorithm analysis) we find the following endorsement.

> However, we assume that the number of items to be sorted is moderately large. If one is going to sort only a handful of items, a simple strategy such as the $O(n^2)$ "bubble sort" is far more expedient.

Perhaps a generation of computer scientists and teachers used this book and the acceptability of bubble sort began. In the influential 1976 work *Software Tools* [15] the first sort mentioned is bubble sort (the second is Shell sort).

Why is bubble sort popular? In [26] we get an endorsement.

> ... Nevertheless, this sorting algorithm is commonly used where the value of n is not too large and programming effort is to be kept to a minimum. ... The bubble sort has the additional virtue that it requires almost no space in addition to that used to contain the input vector.

Perhaps early concerns with allocating registers and using memory led to the adoption of bubble sort which in its primitive form (no early stopping) requires no storage other than the array and two index variables. This conclusion is supported by the early explanations of selection sort (e.g., [3, 10]) in which only the version described in Section 2.1 as "jump-down" sort is described, no separate minimum index variable is kept, and the minimal element selected in each pass of selection sort is replaced by "a series of 9's ... so that said item will never again be selected." [10].

Another early work [7] is referenced in [18] with the following warning.

> From a mathematical standpoint, Demuth's thesis was a beautiful piece of work. ... But from a practical standpoint, the thesis was of no help. In fact, one of Demuth's main results was that in a certain sense "bubble sorting" is the optimum way to sort. ... It turns out that [all other sorting methods studied] are always better in practice, in spite of the fact that Demuth has proved the optimality of bubble sorting on a certain peculiar type of machine.

Demuth's work was published 29 years after he wrote it, and he does not recall using the term "bubble sort" in his studies. [8].

2.4 Measures of Popularity

A 1988 SIGCSE paper [16] notes that bubble sort "while once the best known sort, seems relegated to the status of an example of inefficiency". Fifteen years later this sort is still with us, and students continue to use it.

Bubble sort is covered in many texts, occasionally as the only $O(n^2)$ sort, but often compared to another sort like insertion or selection. Rarely are its bad features emphasized. This may lead students to think that bubble sort is acceptable when it has provably bad performance and is arguably not the simplest sort to learn.

As unscientific albeit interesting evidence of popularity, on August 1, 2000 we searched the Internet for different sorts using the search engine Google. We repeated this search in August of 2002. For each method we used the name with and without a space, e.g., "bubblesort" and "bubble sort". Table 1 shows the results.

sort	# hits 2000	# hits 2002
Quick	26,780	80,200
Merge	13,330	33,500
Heap	9,830	22,960
Bubble	12,400	33,800
Insertion	8,450	21,870
Selection	6,720	20,600
Shell	4,540	8,620

Table 1: Web-based popularity of sorts

[3] We are not arguing that Knuth's work is less substantial, but that it may have had less of a curricular impact than books specifically designed as textbooks rather than as works of reference.

The good news is that Quicksort is by far the most-referenced sort on the web. The bad news is that bubble sort is the second or third *most popular* sort and is by far the most cited $O(n^2)$ sort.[4] Hopefully the situation is somewhat mitigated by the greater rate of increase in hits for selection sort compared to bubble sort.

3 Performance Characteristics

Here we show that by several measures bubble sort is not simpler to code than other sorts and that its performance is terrible. We worry about bubble sort because of its inexplicable popularity. We view the use of bubble sort as an instance of a larger problem: choosing examples that are not exemplars of accepted best-practices. Such examples invariably must be unlearned later which is often an impossible task.

3.1 Ease of Coding

Conventional wisdom holds that bubble sort is simple to program. For example, in [5] we find: "The bubble sort is worse than selection sort for a jumbled array—it will require many more component exchanges—but it's just as good as insertion sort for a pretty well-ordered array. More important, it's usually the easiest one to write correctly."

However, in [24] we find a weak rebuttal.

> Bubble sort's prime virtue is that it is easy to implement, but whether it is actually easier to implement than insertion or selection sort is arguable.

This rebuttal is stated more forcefully in [27].

> The bubble sort algorithm is not very useful in practice, since it runs more slowly than insertion sort and selection sort, yet is more complicated to program.

Measuring Ease of Coding Software metrics are controversial. However, perhaps the most-cited metrics based on a static analysis of code (no control-flow paths) are the Halstead Metrics [12]. These metrics are based on counts of operators and operands, though symbols such as a semi-colon are interpreted as operators. Table 2 gives the *Difficulty* and *Effort* measures for several sorts.[5]

[4] It is possible though unlikely that every web page referring to bubble sorts extols its bad qualities.

[5] These metrics were calculated automatically from the code given in this paper using a tool from `www.powersoftware.com` and verified as accurate using the Unix program `npath`.

Difficulty purports to measure how hard it is to create the program. Effort purports to measure the effort required to convert an algorithm into a program.

sort	D	E
Bubble	17.25	4165
Jump-down	14.38	3828
Select	15.95	4242
Insert	23.20	6652
Quick	7.88	3157
Partition	12.07	1522

Table 2: Halstead Complexity of Sorts

For some perspective on these numbers consider an implementation of selection divided into two parts: a function to return the index of the minimal element and the sorting function below.

```
void SelectSort(Vector a, int n)
{
    for(int j=0; j< n-1;j++) {
        Swap(a, minIndex(a,j,n), j);
    }
}
```

This version of selection sort has a difficulty of 7.88 and an effort of 966; the minimal index function has D=14 and E=2247.

Note that the implementation of quick sort is divided into two functions, one recursive (Quick in Table 2) and one with a single loop (Partition).

3.2 Performance

Although popular, bubble sort is nearly universally derided for its poor performance on random data. This derision is justified as shown in Figure 1 where bubble sort is nearly three times as slow as insertion sort.

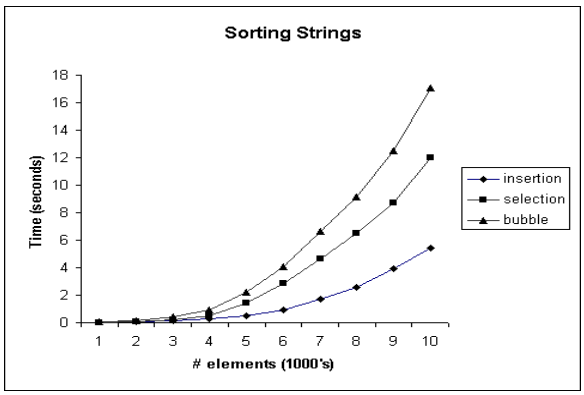

Figure 1: Sorting `Strings` in Java

3.3 Good on Nearly-Sorted Data

Some books laud bubble sort because it runs in $O(n)$ time on sorted data and works well on "nearly sorted" data. In [24] this is qualified in slightly more detail with the conclusion that insertion sort is better than bubble sort, is stable, and is the basis for the more efficient Shell sort.

This leaves little to recommend bubble sort. In any situation in which it does well insertion sort does as well and is better by other criteria. Insertion sort is used to sort small (sub) arrays in standard Java and C++ libraries.

4 Conclusion

In most practical situations the best sort to use is the one provided by the standard (e.g., Java, C, or C++) libraries. However, we study sorts because general sorts do not work in all situations and because sorting is a simple illustration of algorithmic techniques. Although examples used in first year courses must be simple enough to be understandable and complex enough to be useful in a variety of situations, they should also exemplify best practices so that these practices endure after many of the details of a course have been forgotten. In this paper we have investigated the origins of bubble sort and its enduring popularity despite warnings against its use by many experts. We confirm the warnings by analyzing its complexity both in coding and runtime.

References

[1] The jargon file. `http://www.jargon.net/jargonfile/b/bogo-sort.html`.

[2] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[3] Bell, D. The principles of sorting. *The Computer Journal 1* (1958), 71–77.

[4] Bose, R. C., and Nelson, R. J. A sorting problem. *Journal of the ACM (JACM) 9*, 2 (1962), 282–296.

[5] Cooper, D. *Oh My! Modula-2!* W.W. Norton, 1990.

[6] Dale, N., and Lewis, J. *Computer Science Illuminated*. Jones and Bartlett, 2002.

[7] Demuth, H. *Electronic Data Sorting*. PhD thesis, Stanford University, 1956.

[8] Demuth, H. personal communication. 2000.

[9] Flores, I. Analysis of internal computer sorting. *Journal of the ACM (JACM) 8*, 1.

[10] Friend, E. Sorting on electronic computer systems. *J. ACM 3* (1956), 134–168.

[11] Gotlieb, C. Sorting on computers. *Communications of the ACM 6*, 5 (May 1963), 194–201.

[12] Halstead, M. H. *Elements of Software Science, Operating, and Programming Systems Series*, vol. 7. Elsevier, 1977.

[13] Iverson, K. *A Programming Language*. John Wiley, 1962.

[14] Juelich, O. Remark on algoirthm 175 shuttle sort. *Communications of the ACM 6*, 12 (December 1963), 739.

[15] Kernighan, B. W., and Plauger, P. *Software Tools*. Addison-Wesley, 1976.

[16] Klerlein, J. B., and Fullbright, C. A transition from bubble sort to shell sort. In *The Papers of the Nineteenth Technical Symposium on Computer Science Eduction* (February 1988), ACM Press, pp. 183–184. SIGCSE Bulletin V. 20 N. 1.

[17] Knuth, D. *The Art of Computer Programming: Sorting and Searching*, 2 ed., vol. 3. Addison-Wesley, 1998.

[18] Knuth, D. E. The dangers of computer science theory. *Logic, Methodology and Philosophy of Science 4* (1973). Also in Selected Papers on Analysis of Algorithms, CLSI, 2000.

[19] Ledley, R. *Programming and Utilizing Digital Computers*. McGraw-Hill, 1962.

[20] Martin, W. A. Sorting. *ACM Computing Surveys 3*, 4 (1971), 147–174.

[21] McCracken, D., Weiss, H., and Lee, T. *Programming Business Computers*. John Wiley, 1959.

[22] Organick, E. I. *A Fortran Primer*. Addison-Wesley, 1963.

[23] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1988.

[24] Sedgewick, R. *Algorithms in C++*, 3 ed. Addison-Wesley, 1998.

[25] Shaw, C., and Trimble, T. Algorithm 175: Shuttle sort. *Communications of the ACM 6*, 6 (June 1963), 312–313.