

A Sort of Statistics Lesson

KEYWORDS:

Teaching;
Sorting algorithms;
Correlation.

Chris du Feu

Queen Elizabeth's High School, Gainsborough,
England.

e-mail: chris@beckingham0.demon.co.uk

Summary

A classroom practical activity for sorting lists of numbers is described, leading to some statistical analysis

◆INTRODUCTION◆

OVER recent years in England, there seems to have been a regrettable trend towards narrowly focused teaching. Perhaps it results from the original National Curriculum breaking mathematics into a mass of minute disconnected criteria. Perhaps it results from the current obsession to measure and assess everything we do in our teaching whereas the important things in education are often intangible and unmeasurable. Whatever its cause, I believe it works against good mathematical education. The events related below came about during teaching an A-Level (age 16-18) decision mathematics module, but as the students were also taking a statistics module I hoped there might be some mutual reinforcement between material in the different modules. The syllabus for the decision mathematics module includes awareness of algorithms in general and two sorting algorithms (bubble sort and quick sort) in particular. The bubble sort algorithm is described in Appendix 1.

ing operations and the second is to help them understand how sorting times increase with list length (and hence why efficient sorting algorithms are needed). As students sort by hand they will soon see that sorting becomes very much more time-consuming as the list length increases. They should try to observe how they set about the job and which approaches work better or worse than others. This will give an understanding of the problems which sorting algorithms address and statistics can be used to enhance this understanding.

A moment's reflection will show the difficulty of predicting the time it takes to sort a list of n items, even if all details of the sorting algorithm and sorting device (be it computer or pencil-and-paper) are known. The sorting time depends not just on the list length, sorting speed and algorithm, but also on the list elements and their initial order. Clearly there can be no functional rule governing sort time as a function of list length alone; but it seems likely that a statistical relationship, which allows for other sources of variation, can be found.

◆SORTING ALGORITHMS◆

Students at this level seem generally unfamiliar with the frequency with which sorting processes are used in daily life and how time-consuming big sorts are, even using computer power. There seems to be little point in teaching sorting algorithms unless the students appreciate the needs for, and uses of, them, and the problems associated with sorting large data sets. It is important for the students to understand that, in general, sorting time is not a linear function of list length and that longer lists take very much longer to sort.

Before being taught formal sorting algorithms, students should have tried sorting numbers "by hand". The hand-sorting exercise outlined below has two purposes. The first is to give students a feel for sort-

◆CLASSROOM PRACTICAL◆

Students are given lists to sort; they record the number of elements to sort in each list and the time taken to sort them. I have found that it is convenient to use lists of 5-digit numbers from random number tables, so that students do not immediately "know the answer". I use consumable computer generated random number tables which avoid copyright problems and can be duplicated freely. I encourage students to cancel numbers on the tables once they have been "used" - this makes for easier data handling. To ensure a good spread of list lengths, I first give each student a number and they then sort that number of 5-digit random numbers from their tables. Once their list is sorted I issue another number to them. This part of the exercise stops when there have been enough lists sorted to give a good spread of list lengths and to have sufficient data for statistical analysis. I have found that it is convenient in the classroom situation to

use list lengths in the range 5 to 30.

Before starting, I had suggested that the sorting time might be a quadratic function of list length. This hypothesis is plausible (see Appendix 2) but as I had not tried this classroom practical before I was not confident that this piece of theory would turn into reality. With students sorting lists, further variation in sort time is introduced because they are using different sorting routines (which they are probably refining as they go along, and which may sometimes use informal shortcuts for particular data sets) and are also working at different rates. In spite of all these potential problems, I drew a pair of axes on the board and invited the students to plot their results of time against list length. They would soon see it was a quadratic curve, I said (more in hope than expectation!).

The practical worked quite well. Observation showed that the students were developing awareness of the problems of sorting and the need to be systematic. The plot on the board looked promising with the points scattered around something quite close to a quadratic curve.

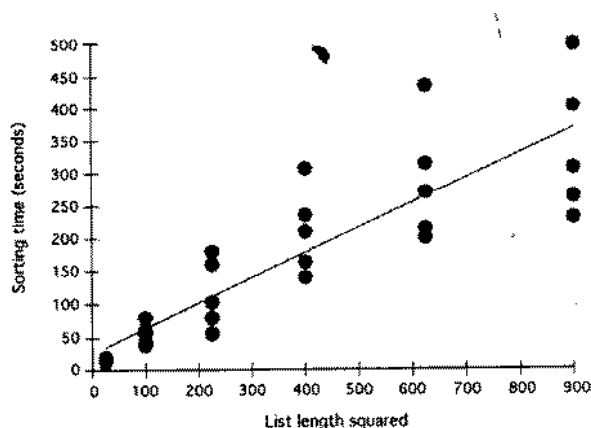


Fig 1. Class results for sorting random numbers

◆ STATISTICS APPLIED ◆
TO THE PROBLEM

Later that day I was accosted by one of the group. He had taken the raw data of list lengths and sort times, entered these into his TI-83 graphic calculator, realised how to transform the data by squaring the list lengths and found the linear correlation coefficient for the transformed data. $r = 0.87$ for $n=30$ (figure 1). Both the value of r and the transformed plot are strongly supportive of the hypothesised quadratic relationship. Naturally I was quick to re-

port this news back to the rest of the group!

Quite a successful exercise! Students had appreciated the problems of sorting but, perhaps more importantly, statistics had been used to help this understanding. Statistics could be seen as a useful tool rather than as the subject matter of an isolated mathematics module to be studied, examined, and forgotten.

Appendix 1. The Bubble Sort Algorithm

The bubble sort consists of a number of passes of the list to be sorted. In each pass, successive pairs of elements are compared and, if that pair is not in the correct order, the two elements are exchanged. A counter records the number of exchanges made on a particular pass. If the counter is not zero at the end of the pass then it is reset to zero and another pass must be made. If the counter is zero then no pairs are out of order so the sort is complete.

Observation of the sort process shows that at the end of each pass the highest as-yet-unsorted number has moved into place. This means that on the n th pass there is no need to check the last $n-1$ adjacent pairs.

An Example showing the list 56 33 26 92 23 76 being sorted into ascending order is shown below.

Appendix 2. Sorting Times

In a typical sort, the list has to be passed a number of times. On each pass we order the list a little more so on the next pass there is less work to do. In a bubble sort, for instance, we can be sure that after k passes the last k items are in order, so we need only examine the first $n-k$ items of the list of n items. Similar considerations apply to other types of sort.

Thus for a list of n items we have n passes. The k th pass need consider only $n-k+1$ items. Assuming the pass time is proportional to number of items considered, the time for the k th pass is $c(n-k+1)$ where c is some constant.

Thus the time for the complete sort of n passes is

$$c(n + n-1 + n-2 + n-3 + \dots + 1) = cn(n+1)/2$$

which is a quadratic function of the list length n .

This argument only applies to “worst-case” lists which require the maximum of n passes. The number of passes required depends on the initial list order. For example the list {1 2 3 4 5 6} which is already

sorted requires only one pass (in which the algorithm checks it is sorted) whereas the apparently almost-sorted list {2 3 4 5 6 1} requires the full complement of 6 passes (try sorting it using the algorithm!). It is possible to calculate the number of passes required for any given list, and from this build the probability distribution for the number of swaps necessary to sort a list of any given length. (Hint. For

each list length n , consider the $n!$ equiprobable permutations of {1, 2, 3 ... n } items. For each permutation find the number of passes required. The case where ties are allowed is even more complicated.) Note that even in situations where all n passes are not required, the sorting operation consists of a number of passes of decreasing length so the total sort time is still likely to be related to the square of the list length.

Pass	List					
1	<u>56</u>	<u>33</u>	26	92	23	76
	33	<u>56</u>	<u>26</u>	92	23	76
	33	26	56	<u>92</u>	<u>23</u>	76
	33	26	56	23	<u>92</u>	<u>76</u>
2	33	26	56	23	76	92*
	<u>33</u>	<u>26</u>	56	23	76	92*
	26	33	<u>56</u>	<u>23</u>	76	92*
3	26	33	23	56	76*	92*
	26	<u>33</u>	<u>23</u>	56	76*	92*
4	26	23	33	56*	76*	92*
	<u>26</u>	<u>23</u>	33	56*	76*	92*
5	23	26	33*	56*	76*	92*
	23	26	33*	56*	76*	92*

Counter	Notes
1	56>33 - exchange
2	56>26 - exchange
3	56<92 - leave, but 92>23 - exchange
4	92>76 - exchange.
	Counter \neq 0; another pass required.

* indicates elements which need not be re-examined on later passes.;

1	
2	Counter \neq 0 another pass required.
1	Counter \neq 0; another pass required.
1	Counter \neq 0; another pass required.
0	No swaps - sort terminated