CAB301 Algorithms and Complexity

Nathan Perkins

April 18, 2016

Contents

1 Executive Summary **2**

2 Bubble Sort **2**
2.1 Algorithm 2
2.2 Average Case Efficiency 2

3 Better Bubble Sort **2**
3.1 Algorithm 2
3.2 Basic Operation 2
3.3 Average Case Efficiency 3

4 Methodology **3**
4.1 Computing Environment 3
4.2 Test Data Generation 4

5 Implementation **5**
5.1 Program 5
5.2 Testing 5

6 Experimental Results **5**
6.1 Operational Efficiency 5
6.2 Time Efficiency 5

7 Appendix **6**

1 Executive Summary

2 Bubble Sort

2.1 Algorithm

The general Bubble Sort algorithm works as follows. For every element in the array, successively compare each pair of adjacent elements, up until the total array size minus the current index, swapping each checked pair based on the comparison. For ascending sort, the comparison is if the previous element is larger than the next, and for descending sort the comparison is if the next element is larger than the previous. The reason that not all adjacent pairs are checked each iteration is due to the fact that for each iteration, the current highest element will shift to it's appropriate position in the array. For every iteration over the array, n, the nth element from the end of the listed is correctly indexed and therefore no longer needs to be checked with further iterations.

2.2 Average Case Efficiency

Due to the nature of the algorithm, iterating over the entire loop successively, whilst check and swapping each adjacent pair, the algorithm is inefficient when compared to other methods of sorting. Bubble Sort has a computational efficiency on the order of $\mathcal{O}(n^2)$, with a more specific computational efficiency of $\frac{n^2}{2} + \frac{n}{2}$. [1] In contrast, more efficient algorithms like Heap Sort or Quick Sort have efficiencies on the order of $\mathcal{O}(n \log n)$, which grows on a much slower scale than Bubble Sort. [2]

3 Better Bubble Sort

3.1 Algorithm

The basic Bubble Sort algorithm can be improved by introducing a swap flag. Without the swap flag, the algorithm will continue to compare elements of the array, regardless of whether the array has been sorted. A swap flag, which drives the main loop will ensure that the algorithm exits prematurely should the array elements reach a sorted state.
INSERT ALGORITHM WITH SWAP HERE

3.2 Basic Operation

The operation chosen, to analyse the efficiency of the algorithm is the number of individual swap operations. This aggregate operation defines the core design of the Bubble Sort algorithm, with it being common to both designs, with and without the swap flag.

INSERT APPENDIX LINK OF STEPS CHECK

3.3 Average Case Efficiency

The efficiency of the Better Bubble Sort algorithm is highly dependant on original algorithms efficiency. It is of the same order INSERT SOURCE HERE as the original algorithm, as the fundamental design hasn't changed. However, the addition of the swap flag allows the algorithm to exit prematurely if it has completely sorted the array before the full algorithm has completed. Due to this check flag, it is very hard to quantify the improvements that this addition makes using theoretical analysis alone, as it is highly dependant on the ordering of the array given to the algorithm.
Your report must summarise the expected time efficiency of the algorithm with respect to the size of its input(s). This should be expressed as the algorithms predicted average-case efficiency and/or order of growth. You must explain as clearly as possible how these predictions were calculated or justified. (In some cases you will find an appropriate analysis in the literature. In other cases you may need to calculate the algorithms efficiency yourself.)

4 Methodology

4.1 Computing Environment

For experimental analysis, the computing environment can play a pivotal role in the specific results and has therefore been documented.

1. Operating System: The operating system used for the construction, data generation and analysis of report is Linux, specifically Ubuntu 15.10. This was chosen as it includes easy to use, pre-installed packages that aid in the programming of C++ and python which was also used. All code was then checked to conform with Microsoft Windows for veracity of design.

2. Algorithm: For creation of the algorithm, C++ was used. C++ is an efficient, and fast language that gives good freedom over design and memory, whilst still maintaining high level functionality. The Eclipse cross platform IDE was chosen to program the algorithm in, which gave access to debugging tools and automated compiling. For veracity, this was later ported to the code::blocks IDE and tested.

3. Data Analysis

   (a) Storage: CSV file format was used extensively to store the data directly via the C++ executable. CSV, or comma separated variables is an easy to use file format that separates all variables through the use of commas, this makes it perfect to easily get file formatting and output with C++. Python has in built functionality to interpret

CSV files and was also a good choice in that regards, allowing the easy import of the data for plotting.

(b) Analysis: Python was used to interpret the data taken from CSV into graphs using the matplotlib graphing library. Python is a cross platform, high level scripting language with lots of in built scientific analysis features and is therefore a good choice to evaluate the data from any location or machine. matplotlib is a free library that easily plots data.

4. Report: To build the report and everything included, LaTeX was used. LaTex allows good control over document flow and it's primary purpose, the easy inclusion of mathematical formulae and equations fits perfectly with the overall design of the report.

4.2 Test Data Generation

To test the algorithm, it is important to note the types of arrays used as an input. Arrays sorted in specific manners can skew results, and lead to different performance than intended. As the main test of the report is on the average case, the generation of randomised arrays was of utmost importance, however, for testing, best case and worst case array generation was also performed.

1. Random arrays: Initialising the random seed based on the current system time, then generate each element as a random number from 1-100. With each element being a random number with no association to the previous or next element, a truly randomised list is created.

2. Ordered arrays: Initialising the random seed based on the current system time, setting the first element to be 1 and then proceeding to generate a random number between 0-9 and adding it to the previous array element and setting the next element equal to the result ensures that a strictly ascending array is generated.

3. Reversed arrays: Initialising the random seed based on the current system time, setting the last element to be 1 and then proceeding to generate a random number between 0-9 and adding it to the next array element and setting the previous element equal to the result ensure that a strictly descending array is generated.

5 Implementation

5.1 Program

5.2 Testing

6 Experimental Results

6.1 Operational Efficiency

Your report must explain clearly how you counted basic operations, e.g., by highlighting the relevant statements inserted into the program. In particular, it should be easy to see that the method used is accurate with respect to the original algorithm.
You must perform enough experiments to produce a clear trend in the outcomes. Your report must explain how you produced test data. Depending on the kind of algorithm involved, you may need to produce sets of random values (so that you can produce average- case results for a particular size of input), or an ordered sequence of test values (so that you can show how the algorithm grows with respect to the inputs size). In either case you may choose to create test data manually (which may be very tedious) or automatically (which may require some programming).
You must present your experimental results as a graph. NB: You must state clearly how many data points contribute to the line(s) on the graph and what each data point represents. If possible, you should use a graph drawing tool that displays each data point as a distinct symbol.
You must state whether or not the experimental results matched the predicted number of operations. If they do not match then you must offer some explanation for the discrepancy. (Normally we would expect that counting basic operations produces results that closely match the theoretical predictions, but it is possible that there is some peculiarity of your experimental set-up that skews the results, or even that the theoretical predictions are wrong.)

6.2 Time Efficiency

Your report must explain clearly how you measured execution times, e.g., by showing the relevant test program. (Alternatively, you may even choose to time your program with a stopwatch, although this is unlikely to produce accurate results.) It is often the case that small program fragments execute too quickly to time accurately. Therefore, you may need to time a large number of identical tests and divide the total time by the number of tests to get useful results.
You must perform sufficient experiments to produce a clear trend in the outcomes. Your report must make clear how you produced test data (as per the discussion above on counting basic operations).
You must present your experimental results as a graph. NB: You must state clearly how many data points contribute to the results on the graph and what

each data point represents. If possible, you should use a graph drawing tool that displays each data point as a distinct symbol.
You must state whether or not the experimental results matched the predicted order of growth. It is possible that your measured execution times may not match the prediction due to factors other than the algorithms behaviour, and you should point this out if this is the case in your experiments. For instance, an algorithm with an anticipated linear growth may produce a slightly convex scatterplot due to operating system and memory management overheads on your computer that are not allowed for in the theoretical analysis. (However, a concave or totally random scatterplot is more likely to be due to errors in your experimental methodology in this case!)

7 Appendix

References

[1] W. Min, "Analysis on bubble sort algorithm optimization," in *Information Technology and Applications (IFITA), 2010 International Forum on*, vol. 1. IEEE, 2010, pp. 208–211.

[2] R. Schaffer and R. Sedgewick, "The analysis of heapsort," *Journal of Algorithms*, vol. 15, no. 1, pp. 76–100, 1993.