Sorting algorithms - comparison and analysis

CS22120 2013-2014 Assignment 2

Evdzhan Mustafa
Student at University of Aberystwyth, Wales
enm3@aber.ac.uk

ABSTRACT

This paper has the aim to compare 7 sorting algorithms, in order to determine each algorithm's behaviour given specific amount of input.

1. INTRODUCTION

Each algorithm will be tested with up to 10 files[1]. The files contain numbers, and are unordered state. For the given file, each algorithm's completion time will be recorded, and then compared with the other algorithms, or variation of the same algorithm.
When appropriate, an algorithm might be tested with an input file, that will be in ascending or descending sorted state. That is to determine the algorithms behaviour to variations in the file.

2. THE *EXPERIMENT*

A program written in JAVA was used to record the running times of the tested algorithms. All the algorithms were tested in the same environment, with no other programs running in the background. An algorithm was deployed several times, and the average running time was calculated. For an algorithm, the average running time was recorded and shown in a chart. The unit used for time is seconds. Files 1 through 10 contain 1 000, 2 000, 5 000, 10 000, 20 000, 50 000, 100 000, 200 000, 500 000, 1 000 000, 2 000 000, 5 000 000 and 10 000 000 numbers respectively.

3. ALGORITHMS TESTED

3.1 $O(n^2)$ algorithms

3.1.1 Bubble Sort

A very bad sorting algorithm. Easy to implement, but terribly slow. Although it can be improved, the time complexity remains $O(n^2)$. If an improved version is used on an already sorted array the time complexity drops down to $O(n)$. In such case, the algorithm essentially goes through every value once, without doing any swaps. The improved one will quit the sorting after the first iteration if the array is sorted. Due to the extremely big growth rate of Bubble sort, only the files containing up to 50 000 items are shown on the graph. It is impractical to run the tests for larger datasets, because of the time it takes to finish the sorting. An interesting fact is that if the array is reverse (i.e. sorted in descending order),

[1]Some algorithms are so slow that is not practical to use all the files

Figure 1: Bubble sort.

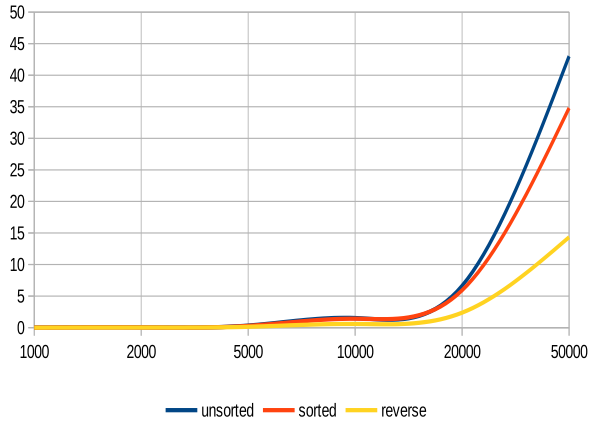
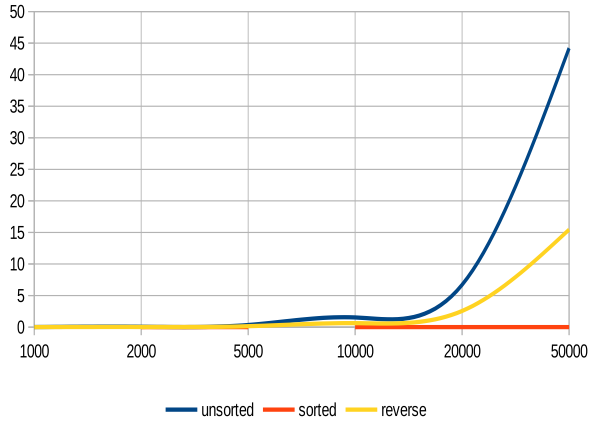


Figure 2: Improved Bubble Sort.

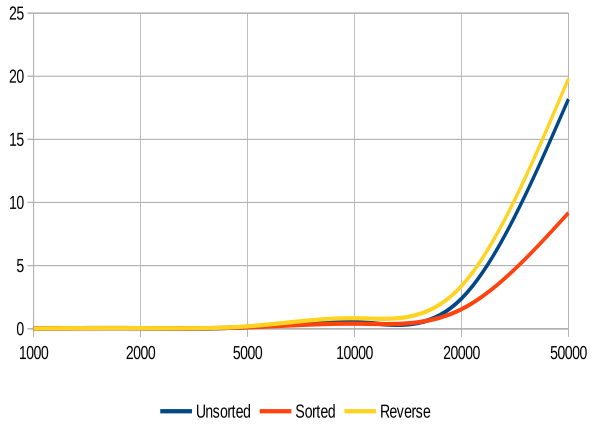


the sorting is faster. This is probably because of Branch Predictor [2].

3.1.2 Selection Sort

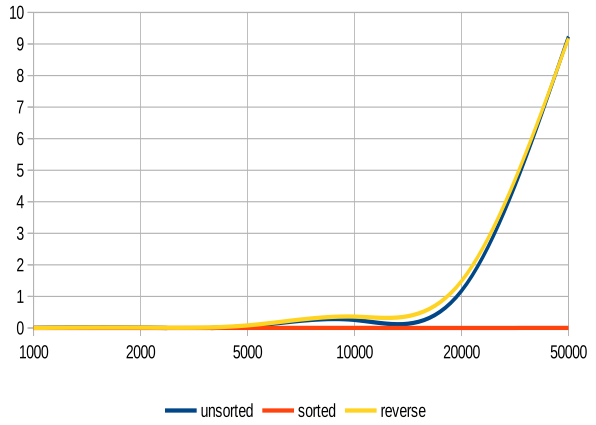
It's average running time is similar to the one of the bubble sort, but is faster. It's con is that it is very easy to implement. The strategy used is to repeatedly go through the array and find the lowest value (select it). As seen on the graph, trying to sort already sorted array, takes less time. This is because no swaps occur. Only comparisons are performed. Having the array in reverse order hardly makes any difference compared to having it unsorted.

Figure 3: Selection sort.



3.1.3 Insertion Sort

Figure 4: Insertion sort.



The insertion sort is the quickest of the three $O(n^2)$ algorithms. It's con is that it performs very quickly on small sets of data. As in the case with improved bubble sort, this algorithm can recognize if the initial array is already sorted. This leads to all loop iteration being skipped. Furthermore even if the array wasn't sorted completely, but rather nearly sorted, it would still cause many loop iterations to be skipped. This is why insertion sort finds application in hybrid sorting algorithms.

One of the drawbacks of this algorithm is that it repeatedly has to shift certain amount of elements to insert the current element to its correct place. This is the part where it spends most of its running time. The graph clearly shows that the time taken for a sorted array is negligible. That is not a single shift operation has happened, but only comparisons that actually are all unsuccessful. And yet again the Branch Predictor greatly improves the overall performance.

[2]http://en.wikipedia.org/wiki/Branch_predictor

3.1.4 $O(n^2)$ algorithms analysis

3.2 n log n Alorithms

One of the quickest sorting algorithms. Sorts every element by find it's correct position.

APPENDIX

A. HEADINGS IN APPENDICES

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations

Display Equations

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LaTeX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The acm_proc_article-sp document class file itself is chockfull of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LaTeX, you may find reading it useful but please remember not to change it.