Volume 4, Issue 5, May 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Studies on the Comparative Analysis and Performance Prediction of Sorting Algorithms in Data Structures

Sourabh Shastri
Assistant Professor
Department of Computer Science and IT
Bhaderwah Campus, University of Jammu

Abstract: - The present piece of investigation documents the comparative analysis of five different sorting algorithms of data structures viz. Bubble Sort, Selection Sort, Insertion Sort, Quick Sort and Shell Sort by comparing their running times calculated by using Turbo C compiler. This comparison was accompanied by the review of various important parameters like complexity, method, memory etc. to reach our conclusion. Besides, an attempt to unveil the working of the aforementioned algorithms was also made during the study.

Keywords-Sorting, Algorithms, Running Times, Complexity, Memory, Turbo C Compiler.

I. INTRODUCTION

In computer science, algorithms and data structures play a vital role for the implementation and design of any software. Sorting algorithms are needed to arrange the elements in increasing or decreasing order in case of numerical data and in alphabetical order in case of non-numerical data [1]. The complexity of a sorting algorithm measures the running time of a function in which n numbers of items are to be sorted. The choice of sorting algorithm depends on various parameters like the amount of memory or machine time needed for running a program, how fast and accurately it sorts a list. Most of the sorting algorithms are data sensitive i.e. it depends on the order of the data and most of them have time complexity from o(n log n) to $o(n^2)$[2].Sorting is an area in which the mathematical analysis of algorithms has been particularly successful and where the prediction of the performance can be done intelligently. To find the running time of each sorting algorithm, Turbo C++ compiler for comparing the time (in seconds) was utilised. After running the same program on five different runs (for each different value N=5000, 10000, 15000, 20000, 25000, 30000), the average running time for each algorithm was deduced.

II. WORKING PROCEDURE OF ALGORITHMS

A. *Bubble Sort*

In bubble sort every element is compared with its adjacent element and if the elements are out of order they are swapped and the largest element bubble up after one pass. Similarly on each successive pass the next largest elements are placed at proper positions. Once there is no interchanging of elements in a particular pass then there will be no further interchanging of elements in the subsequent passes. This algorithm's average and worst case performance is $o(n^2)$[3].

Properties: [4]

1) Stable
2) O(1) extra space
3) $O(n^2)$ comparisons and swaps
4) Adaptive: O(n) when nearly sorted

B. *Selection Sort*

In selection sort we repeatedly find the next largest (or smallest) element in the array and move it to its final position in the sorted array. To sort the array in ascending order we have to begin by selecting the largest element and moving it to the highest index position. We can do this by swapping the element at the highest index and the largest element. We then reduce the effective size of the array by one element and repeat the process on the smaller (sub) array. The process stops when the effective size of the array becomes 1 (an array of 1 element is already sorted) [5].

Properties: [6]

1) Not stable
2) O(1) extra space
3) $\Theta(n^2)$ comparisons
4) $\Theta(n)$ swaps
5) Not adaptive

C. *Insertion Sort*

In insertion sort, it will go through the sequence checking each element starting from the second onwards, and for each element, it will loop back the array to search for the best place for the number to be. In more common terms it backtracks will it find the exact spot, where the current number should belong and put it there. But along with the

backtracking, it actually swaps each intermediate element, so in the end of one main cycle the whole sequence has a change[7].

Properties: [8]

1) Stable
2) O(1) extra space
3) $O(n^2)$ comparisons and swaps
4) Adaptive: O(n) time when nearly sorted
5) Very low overhead

D. Quick Sort

In quicksort, we divide the array of items to be sorted into two partitions and then call the quicksort procedure recursively to sort the two partitions, i.e. we divide the problem into two smaller ones and conquer by solving the smaller ones. For the strategy to be effective, the partition phase must ensure that the pivot is greater than all the items in one part (the lower part) and less than all those in the other (upper) part. To do this, we choose a pivot element and arrange that all the items in the lower part are less than the pivot and all those in the upper part greater than it. In the most general case, we don't know anything about the items to be sorted, so that any choice of the pivot element will do - the first element is a convenient one. If several items are the same as the pivot, these items can be grouped with the pivot in a third (middle) partition or left in the lower part: change "less than" in the description above to "less than or equal" [9].

Properties: [10]

1) Not stable
2) O(log(n)) extra space
3) $O(n^2)$ time, but typically O(n.log(n)) time
4) Not adaptive

*E. **Shell Sort***

The shell sort compares elements that are at a certain distance away (d positions away) from each other and the distance between comparisons decreases as the algorithm runs until the last phase, in which adjacent elements are compared(bubble sort only compares adjacent elements.) It uses the equation d = (n + 1) / 2[11].

Properties: [12]

1) Not stable
2) O(1) extra space
3) $O(n^{3/2})$ time
4) Adaptive: O(n.log(n)) time when nearly sorted

TABLE I
COMPARISON OF THE FIVE SORTING ALGORITHMS ON VARIOUS PARAMETERS [13, 14]

Sort	**Best**	**Average**	**Worst**	**Stability**	**Method**	**Extra Space**	**Remarks**
Bubble	n	n^2	n^2	Yes	Exchanging	O(1)	Tiny code size
Selection	n^2	n^2	n^2	No	Selection	O(1)	Stable with O(n) extra space, for example using lists
Insertion	n	n^2	n^2	Yes	Insertion	O(1)	In the best case (already sorted), every insert requires constant time
Quick	n log n	n log n	n^2	Typical in-place sort is not stable; stable version exists	Partitioning	O(log n)	Randomly picking a pivot value (or shuffling the array prior to sorting) can help avoid worst case scenarios such as a perfectly sorted array
Shell	N	$n \log^2 n$ or $n^{3/2}$	$n \log^2 n$	No	Insertion	O(1)	Small code size, reasonably fast

III. EXPERIMENT AND RESULT TO MEASURE THE PERFORMANCE OF ALGORITHMS

In this experiment, determination of the efficiency of various sorting algorithms was executed by using Turbo C++ 3.0 compiler in which data set contains random numbers. Time selection is the time in the seconds. Each sorting function

was run to find the running time of that sorting algorithm and for that I passed different number of elements (N=5000, 10000, 15000, 20000, 25000, 30000). Table II shows the running time of each algorithm for first, second, third, fourth and fifth run. I have also calculated the average running time (in seconds) based upon the running time. The chart (Figure1) shows the comparison of all five sorting algorithms for the elements (N=5000, 10000, 15000, 20000, 25000, 30000).

TABLE II
RUNNING TIME OF EACH ALGORITHM FOR FIRST, SECOND, THIRD, FOURTH, FIFTH AND AVERAGE RUN

First Run(Time in seconds)					
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Shell Sort
5,000	1.373626	1.043956	1.428571	0.109890	0.164835
10,000	4.670330	2.527473	4.120879	0.109890	0.329670
15,000	10.329670	5.054945	9.065934	0.164835	0.329670
20,000	17.692308	8.351648	16.153846	0.219780	0.604396
25,000	27.087912	12.637363	24.835165	0.274725	0.494505
30,000	38.406593	18.021978	35.439560	0.329670	0.494505
Second Run(Time in seconds)					
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Shell Sort
5,000	1.648352	1.098901	1.648352	0.054945	0.219780
10,000	4.615385	2.637363	4.175824	0.109890	0.274725
15,000	10.000000	5.109890	9.560440	0.164835	0.329670
20,000	17.857143	8.626374	16.043956	0.219780	0.549451
25,000	28.186813	12.582418	25.054945	0.219780	0.549451
30,000	38.351648	18.681319	35.439560	0.274725	0.549451
Third Run(Time in seconds)					
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Shell Sort
5,000	1.593407	1.153846	1.593407	0.054945	0.164835
10,000	4.835165	2.637363	4.835165	0.109890	0.329670
15,000	10.989011	5.164835	8.956044	0.219780	0.439560
20,000	19.065934	9.010989	15.769231	0.274725	0.549451
25,000	26.813187	12.802198	24.780220	0.274725	0.549451
30,000	38.296703	17.967033	35.384615	0.329670	0.659341
Fourth Run(Time in seconds)					
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Shell Sort
5,000	1.263736	0.934066	1.263736	0.054945	0.164835
10,000	4.725275	2.802198	4.285714	0.164835	0.274725
15,000	10.439560	5.329670	8.901099	0.219780	0.384615
20,000	18.351648	8.571429	16.098901	0.274725	0.494505
25,000	26.923077	12.637363	24.230769	0.274725	0.604396
30,000	38.626374	18.076923	35.000000	0.384615	0.604396
Fifth Run(Time in seconds)					
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Shell Sort
5,000	1.648352	1.263736	1.373626	0.109890	0.109890
10,000	4.560440	2.527473	4.065934	0.164835	0.329670
15,000	10.219780	6.208791	9.120879	0.164835	0.439560
20,000	17.472527	8.626374	15.989011	0.274725	0.439560
25,000	28.186813	12.527473	24.450549	0.329670	0.494505
30,000	38.791209	18.516484	34.780220	0.274725	0.549451
Average(Time in seconds)					
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Shell Sort
5,000	1.5054946	1.098901	1.4615384	0.076923	0.164835
10,000	4.681319	2.626374	4.2967032	0.131868	0.307692
15,000	10.3956042	5.3736262	9.1208792	0.186813	0.384615
20,000	18.087912	8.6373628	16.010989	0.252747	0.5274726
25,000	25.4395604	12.637363	24.6703296	0.274725	0.5384616
30,000	38.4945054	18.2527474	35.208791	0.318681	0.5714288

From the table II it is concluded that Shell Sort and Quick Sort takes less time as compared to Bubble Sort, Selection Sort and Insertion Sort but the Quick Sort is taking the least time in all the cases. Shell Sort is the most efficient sort among

Bubble Sort, Selection Sort and Insertion Sort. Thus, to infer, one can say that among all the sorting algorithms taken into consideration during the present study, Quick Sort was the most efficient (Figure1).

IV. CONCLUSION

The main aim of the paper was to illustrate methods by which a typical computer can be made to sort a file as quickly and conveniently as possible. Quick Sort is popular because it is not difficult to implement, works well for a variety of different kinds of input data and is substantially faster than any other sorting method in typical applications, thereby acting as a good choice. Quick Sort is considered to be quicker because the coefficient is smaller than any other known algorithm. Moreover, its recursive structure is relatively cache-friendly, thus, the present study augments the diverse significances of the faster sorting algorithm i.e. Quick Sort.

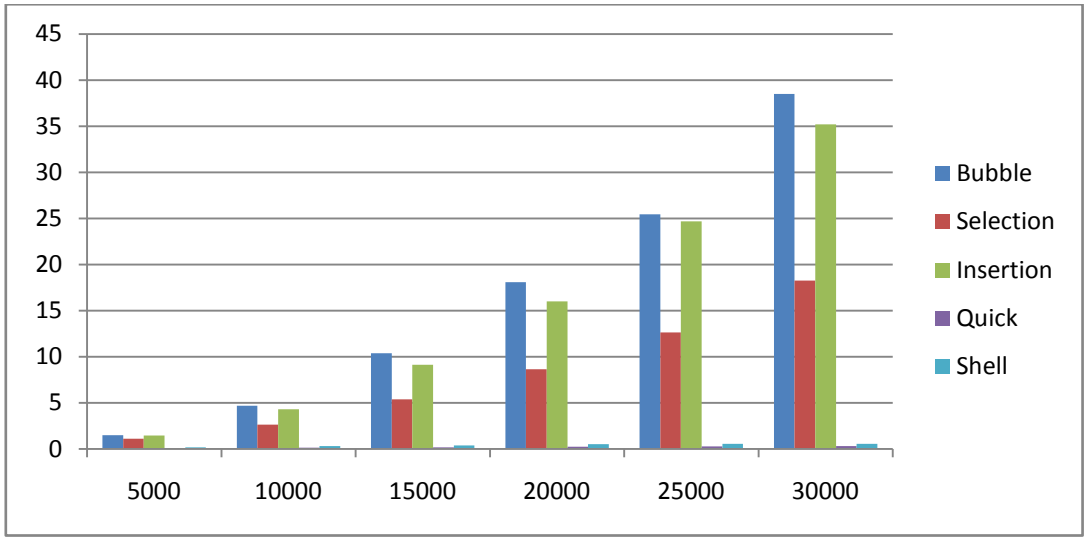

Figure 1: Comparison of five sorting algorithms having elements (N=5000, 10000, 15000, 20000, 25000, 30000)

Acknowledgment

The author is thankful to the Head, Department of Computer Science and IT, University of Jammu for providing the necessary facilities to carry out the study.

REFERENCES

[1] S. Lipschutz , and P. G. A Vijayalakshmi, *Data Structures* by (Tata McGraw Hill companies), Indian adapted edition-2006
[2] *Data Structures Using C*, ISRD Group (Tata McGraw Hill Publishing Company) Limited.
[3] R.S.Salaria, *Data Structures and Algorithms Using C*, Khana Book Publishing Co.
[4] http://www.sorting-algorithms.com/bubble-sort
[5] http://www-ee.eng.hawaii.edu/~tep/EE160/Book/chap10/subsection2.1.2.1.html
[6] http://www.sorting-algorithms.com/selection-sort
[7] http://www.go4expert.com/articles/insertion-sort-algorithm-absolute-t27893/
[8] http://www.sorting-algorithms.com/insertion-sort
[9] https://www.cs.auckland.ac.nz/~jmor159/PLDS210/qsort.html
[10] http://www.sorting-algorithms.com/quick-sort
[11] http://www.codingunit.com/shell-sort-algorithm
[12] http://www.sorting-algorithms.com/shell-sort
[13] http://en.wikipedia.org/wiki/Sorting_algorithm
[14] http://www.cprogramming.com/tutorial/computersciencetheory/sortcomp.html

