Sorting Algorithms and their Run-Time Analysis with C#

Sourabh Shastri
Dept. of Computer Science & IT, Bhaderwah Campus, University of Jammu, J&K.

Prof. Vibhakar Mansotra
Dept. of Computer Science & IT, University of Jammu, J&K.

Anand Sharma
Dept. of Computer Science & IT, University of Jammu, J&K.

Abstract--- Analysis of algorithms is an issue that has always stimulate enormous curiosity. There are logical techniques of estimating the complexity of an algorithm. Generally, a more convenient solution is to estimate the run time analysis of the algorithm. The present piece of investigation documents the comparative analysis of six different sorting algorithms of data structures viz. Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort and Shell Sort. The running time of these algorithms is calculated with C# language. These sorting algorithms are also compared on the basis of various parameters like complexity, method, memory etc.

Keywords: Sorting; Algorithm; Complexity; Running Time; Data Structure.

I. INTRODUCTION

The development of an algorithm is fundamental to computer programming and is a vital component of computer science studies. Once an algorithm is prepared, the subsequent action for the result is to program the algorithm through a computer system by using mathematical and data processing methods. An algorithm is an ordered sequence of instructions for solving a problem. To solve any problem, the first step is to develop the algorithm for that problem. The characteristics of an algorithm are Input, Output, Definiteness, Finiteness and Effectiveness. Input means the algorithm receives some input. Output means the algorithm produces some output. Definiteness means that each step should be clear and unambiguous, Finiteness means that the algorithm terminates after a finite number of instructions. Effectiveness means that each instruction must be sufficiently basic [1].

Algorithmic complexity is associated with the algorithms to know how fast or slow particular algorithm performs. The objective of the computational complexity is to classify algorithms according to their performances. Complexity can be measured in two ways: Space complexity and Time complexity. Space complexity means the number of memory cells which an algorithm needs. Time complexity describes the amount of time in terms of number of comparisons, number of times some inner loop is executed etc. that is related to the amount of real time the algorithm will take. There is a trade-off between time and space complexity. By increasing the amount of space for storing the data, one may be able to reduce the time needed for processing the data or vice versa [2].

The term analysis of algorithms is used to describe approaches to the study of the performance of algorithms. The worst case runtime complexity of the algorithm

describes the maximum number of steps taken on any instance of size n. The best case runtime complexity of the algorithm describes the minimum number of steps taken on any instance of size n. The average case runtime complexity of the algorithm describes an average number of steps taken on any instance of size n.

Sorting is one of the important techniques which are mostly used in computer studies to organize a collection of items into a particular order. Sorting technique can be used to arrange data either in ascending order or descending order. The complexity of sorting algorithm measures the running time of a function in which n numbers of items are to be stored [3]. Different concepts such as exchanging, selection, insertion, partitioning etc. are used in different algorithms of sorting. There are mainly two types of sorting i.e. internal sorting and external sorting. In internal sorting all the data elements to sort are stored in memory. In external sorting, data elements are stored somewhere outside of memory say on disk and only loaded into memory in small chunks. External sorting is typically functional in cases when data can't fit into memory completely. Two essential properties of sorting algorithms are stable and in place. If the sorting algorithm preserves the relative order of any two equal elements, then the sorting algorithm is stable. If an algorithm does not require an extra memory space except for few memory units, then the algorithm is said to be in place.

In this paper, we implemented various sorting algorithms which include Bubble sort, Selection sort, Insertion sort, Quick sort, Merge sort and Shell sort and calculated the running time of these sorting algorithms. For calculating the time complexity of these sorting algorithms, C# language is used. We have run the same algorithm on ten different runs for each different value of N = 10000, 20000, 30000, 40000 and 50000 and finally calculated the average running time for each algorithm separately. User has to input only the value of N i.e. how many elements are required to sort and a random number generator generates n number of elements randomly.

II WORKING PROCEDURE OF ALGORITHMS

In this section, we discuss the working procedure of all the six algorithms used for the comparative analysis in paper viz. Bubble sort, Selection sort, Insertion sort, Quick sort, Merge sort and Shell sort.

A. Bubble Sort

Bubble sort also known as exchange sort is a basic sorting algorithm that starts with evaluating the first two elements and if the first element is greater than the second element then swaps those two elements [4]. Similarly it continues with the same procedure for the next elements in the data set. The largest element bubble up after one pass. Similarly on each succeeding pass the next largest elements are positioned at appropriate places. In bubble sort, there is an important property that if there is no exchanging of elements in a particular pass, there will be no further exchanging of elements in the succeeding passes [1]. The advantages of bubble sort are simplicity and ease of implementation. The best case is when the elements are in ascending order, then algorithm performs

n-1 comparisons and in contrast, the worst case occurs when the elements are in descending order and the algorithm performs n * (n-1) comparisons.

B. Selection Sort

The selection sort is an in-place sorting algorithm [5] that sorts an array by repeatedly finding the smallest element from unsorted part and placing it at the start of the array. The algorithm preserves two sub arrays in a given array. The first sub array which is already sorted and the remaining second sub array which is unsorted. In each iteration of selection sort, the smallest element from the unsorted sub array is chosen and placed to the sorted sub array. The advantages of selection sort algorithm are simplicity and easy to implement but it is inefficient for large lists.

C. Insertion Sort

Insertion sort is an efficient sort in which each iteration takes away an element from the input data and places it into the exact position in the list being sorted. The insertion sort works efficiently on input that is already almost sorted. The selection of the element to take away from the input data is random and this procedure is continual until all input elements have been placed at the right places. Insertion sort works the way we sort playing cards in our hands. Insertion sort is more efficient than bubble sort and selection sort basically even though all of them have $O(n^2)$ worst case complexity. The advantages of insertion sort are simple implementation and faster than bubble sort [6] but it is efficient for small lists of data.

D. Quick Sort

Quick sort is also called as partition exchange sort that uses the algorithmic technique of divide and conquer and relies on a partition operation [7]. The principle behind divide and conquer is to partition a problem into smaller sub problems. Again each sub problem into smaller sub problems and so on until a sub problem is not decomposable. Solving a problem means to solve all the sub problems [8]. In quick sort, firstly a pivot element is chosen then rearranges all elements in such a way that all elements which are smaller that the pivot element goes to the left side and all those elements which are greater than the pivot element go to the right side. The quick sort algorithm again applies recursively to the left and the right parts. The advantages of quick sort are efficient and fast but it proves to be a bit space costly when it comes to large data sets [9].

E. Merge Sort

Merge sort is again an example of divide and conquer. In merge sort algorithm, the original data set of n elements is divided into two parts. Each part is separately sorted and finally the resulting sequences are merged to construct a single sorted sequence of n elements [10]. The advantages of merge sort are that it is used for both internal sorting and external sorting but there is a disadvantage i.e. space complexity is very high [9]. The time complexity for heap sort in average and worst case lies same i.e. O(n logn) [11**]**.

F. Shell Sort

Shell sort is also called as diminishing increment sort and is based on insertion sort. In insertion sort, adjacent elements are compared only. Shell sort compares the elements at a particular distance using which the elements which are distant can be sorted also. After that in each step there is a decrease in increment sequence and at last adjacent elements of

distance one are compared like insertion sort [12]. So the last step of shell sort is equivalent to insertion sort. The advantage of shell sort is that it is an efficient sort for medium size lists. The best case in shell sort is when the data set is already sorted in the right order.

III. Comparative Study of Algorithms on various Parameters

We summarized the different logical features of various sorting algorithms in the table 1.

Table 1: Comparison of various sorting algorithms on various parameters [9, 3].

Algorithm	Best Case Complexity	Average Case Complexity	Worst Case Complexity	Auxiliary Memory	Stable	Method
Bubble Sort	O(n) [Improved version]	$O(n^2)$	$O(n^2)$	O(1)	Yes	Exchanging
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	No	Selection
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes	Insertion
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n)	depends	Partitioning
Merge Sort	O(n log n)	O(n log n)	O(n log n)	depends	Yes	Merging
Shell Sort	O(n)	Depends on gap sequence	$O(n \log^2 n)$	O(1)	NO	Insertion

IV. Implementation to measure the Performance of Algorithms using C#

We implemented all six algorithms to measure the performance using C# language and calculated the running time in microseconds by using System.Diagnostics.Stopwatch class. The data set for the analysis contains random numbers. We ran ten times for each value of N (i.e. 10000, 20000, 30000, 40000, 50000) and tried to find running time of each sorting algorithm.

In the table 2, the running time of each algorithm for ten runs is displayed. We have also calculated the average running time in microseconds based upon the running times of ten runs.

Table 2: Running time of all the algorithms for ten runs and average of ten runs.

First Run (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	2180788	806576	683886	6988	105135	9665
20,000	8774246	3258285	2734305	15907	376794	21480
30,000	19142388	7469748	6150113	21478	1074778	29766

40,000	33550603	13087561	10909505	29334	1957153	38910
50,000	52893434	20493255	17589393	37022	2910074	48794
Second Run (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	2146964	920496	679971	7176	120963	9445
20,000	8336702	3569092	2779304	15822	417925	19972
30,000	16691729	6524389	5068377	22016	1268099	30563
40,000	34119364	13512334	11419851	36224	2068140	38845
50,000	52783601	20754754	16665809	64939	2888487	49526
Third Run (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	2274579	880662	689355	7838	94247	9715
20,000	8350222	3342267	2742150	14066	344984	188967
30,000	18871268	7386523	6358369	21492	970059	30758
40,000	30547863	13301658	10861454	30997	1700717	39854
50,000	52470609	20189101	17216136	39880	2998222	49833
Fourth Run (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	1894154	939659	675284	6977	92570	10721
20,000	8200388	3298426	2776640	14874	345130	19262
30,000	19100054	7732178	6350600	21737	1224353	28620
40,000	33468916	13415166	10846975	29677	2004398	60974
50,000	52759724	20701571	17146909	48612	2920271	54023
Fifth Run (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	2292221	921790	679437	6995	104426	9529
20,000	8469661	3339844	2932538	14441	412646	21288
30,000	18668169	7353120	6207884	22256	1269351	29872
40,000	33732432	13311896	10796622	31474	1909693	41896
50,000	52357035	19871201	17399448	44733	2954691	50354
Sixth Run (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	2107441	855756	670449	7121	110780	9450
20,000	8182525	3377570	2653186	15921	475453	19366
30,000	18970842	7360761	6118669	32203	1262018	28895
40,000	33941732	11509792	9984363	29023	1975809	46676
50,000	52462690	21201986	16983845	38487	2384926	51834
Seventh Run (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	2258472	889517	745069	6986	125267	9715
20,000	8515755	3461090	2826384	15292	413479	20497
30,000	17973405	7374202	5437962	21563	1184875	29762

40,000	34003863	12829535	10780076	29213	1990903	45133
50,000	52404393	20217902	16297113	51256	2781478	54154
Eighth Run (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	2093460	900319	623937	6968	112093	9723
20,000	8293529	3384024	2814025	14210	416289	20341
30,000	18811351	7510568	6009061	24370	1099028	30217
40,000	33595775	13170252	10963068	29860	2055561	38861
50,000	52998363	19808076	16760589	39892	3050193	48764
Ninth Run (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	2197068	934212	704286	7598	95504	9691
20,000	8508708	3421322	2572465	14321	421611	20278
30,000	18227398	7519736	6128963	22105	1292253	31589
40,000	34347089	12792359	11071325	29167	1971046	38615
50,000	52868961	19989157	16658171	38203	2753008	63037
Tenth Run (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	2093502	949690	721627	7046	111326	9574
20,000	8623293	3336424	2669969	14297	420010	19246
30,000	18929052	7521023	6254270	23006	1145913	31605
40,000	33530545	11937859	10743386	33051	2048099	39725
50,000	53425476	21109542	17152122	39767	2919646	75837
Average (Time in microseconds)						
N	Bubble Sort	Selection Sort	Insertion Sort	Quick Sort	Merge Sort	Shell Sort
10,000	2153864.9	899867.7	687330.1	7169.3	107231.1	9722.8
20,000	8425502.9	3378834.4	2750096.6	14915.5	366966.1	37069.7
30,000	18538565.6	13965224.8	11518426.8	23222.6	1179072.7	30164.7
40,000	33483818.2	25086841.2	9865662.5	30802	1968151.9	42948.9
50,000	52742428.6	20433654.5	1698953.5	44279.1	2856099.6	54615.6

V. Results and Discussions

From the table 2, it is clear that Shell sort, Merge 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, Insertion sort and Merge sort according to the average case. It is very much clear from the figure 2 that among all the sorting algorithms taken into consideration during the present study, Quick sort is the most efficient.

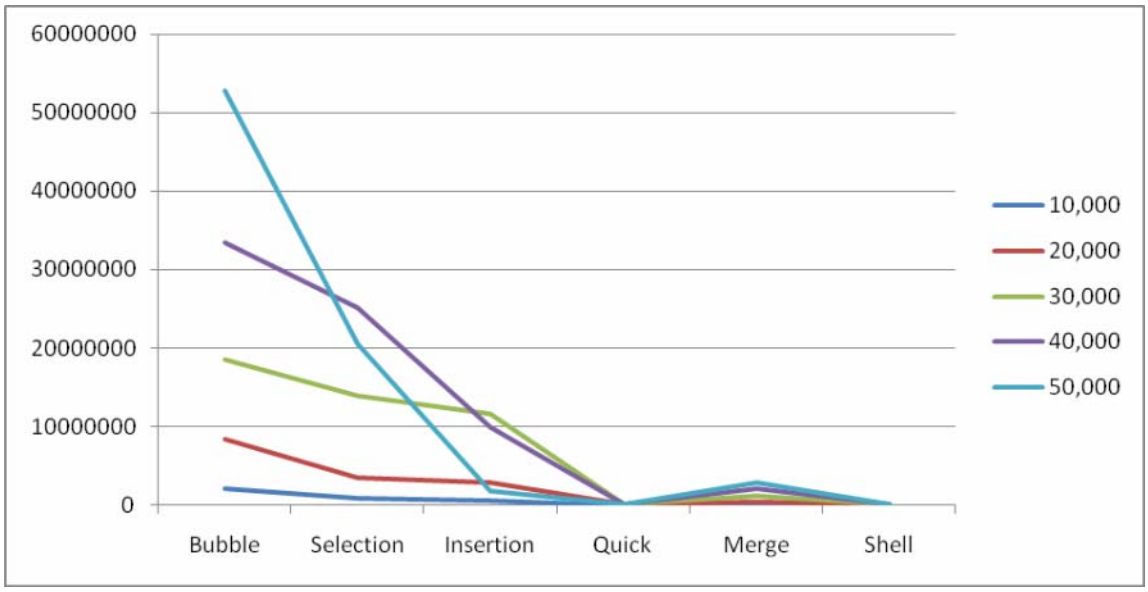

Figure 2: Comparison of six sorting algorithms having elements N= 10000, 20000, 30000, 40000 and 50000.

VI. Conclusion

In this study, we have studied about various sorting algorithms and their comparison. There are advantages and disadvantages in all algorithms. To find the running time of all sorting algorithms, we used C# language in which data elements are randomly chosen for different data sets of N = 10000, 20000, 30000, 40000 and 50000. We calculated the average waiting time for each algorithm and then showed the result with the help of a graph in figure 2. From the figure 2 comparison it is clear that Quick sort is the best and efficient for large data sets among all the algorithms discussed in the study.

VII. Future Work

In the future, we compare the sorting algorithms in different languages to check the running time and memory utilization. Apart from this, we shall try to compare the sorting algorithms in different operating systems and design a graphical user interface for comparing various sorting algorithms.

References:

[1] R. S. Salaria, *Data Structures & Algorithms Using C*, 2nd ed., India: Khanna Book Publishing Co. Ltd.

[2] Seymour Lipschutz, G A Vijayalakshmi Pai, Data *Structures*, Special Indian ed., India: Tata MacGraw-Hill, 2006.

[3] Sourabh Shastri , “Studies on the Comparative Analysis and Performance Prediction of Sorting Algorithms in Data Structures,” *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, issue 5, pp. 1187-1190, May 2014.

[4] Muhammad Usman, Mudassar Afzal and Zaman Bajwa, “Performance Analysis of Sorting Algorithms with C#,” *International Journal of Research in Applied Science & Engineering*

Technology, vol. 3, issue 1, pp. 201-204, Jan. 2015.

[5] Narasimha Karumanchi, *Data Structures and Algorithms Made Easy*, 2nd ed., India: CareerMonk Publications.

[6] Pankaj Sareen, "Comparison of Sorting Algorithms (On the Basis of Average Case)," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, issue 3, pp. 522-532, March 2013.

[7] Sonal Beniwal and Deepti Grover, "Comparison of Various Sorting Algorithms: A review," International Journal of *Emerging Research in Management and Technology*, vol. 2, issue 5, pp. 83-86, May 2013.

[8] D. Samanta, *Classic Data Structures*, First ed., India: Prentice-Hall, 2001.

[9] Miraj Gul, Noorul Amin and M. Suliman, "An Analytical Comparison of Different Sorting Algorithms in Data Structure," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 5, issue 5, pp.1289-1298, May 2015.

[10] S.K. Basu, *Design Methods and Analysis of Algorithms*, First ed., India: PHI, 2005.

[11] Nidhi Chhajed, Imran Uddin and Simarjeet Singh Bhatia, " A Comparison Based Analysis of Four Different Types of Sorting Algorithms in Data Structures with Their Performances," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, issue 2, pp. 373-381, Feb. 2013.

[12] S.K. Srivastava, Deepali Srivastava, *Data Structures Through C in Depth*, First ed., India: BPB Publications, 2003.

AUTHORS PROFILE

Sourabh Shastri, Assistant Professor in Department of Computer Science and IT, Bhaderwah Campus, University of Jammu. He is also pursuing Ph.D. from University of Jammu, J&K.
Email: sourabhshastri@gmail.com

Prof. Vibhakar Mansotra, Professor and Convener of Board Studies in Department of Computer Science and IT, University of Jammu. He is also Director IT, University of Jammu. He has been conferred with the Best Teacher award in Information Technology by the Amar Ujala B-School Excellence Awards. His research interests are Data Mining, Information Retrieval and Computer Graphics.
Email: vibhakar20@yahoo.co.in

Anand Sharma, pursuing Ph.D. from Department of Computer Science and IT, University of Jammu, J&K.
Email: andz24@gmail.com