International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-11 E-ISSN: 2347-2693

A GUI Based Run-Time Analysis of Sorting Algorithms and their Comparative Study

Sourabh Shastri[1*], Vibhakar Mansotra[2], Arun Singh Bhadwal[3], Monika Kumari[4], Avish Khajuria[5], Dalbir Singh Jasrotia[6]

[1*]Dept. of Computer Science and IT, Kathua Campus, University of Jammu, Kathua, India
[2]Dept. of Computer Science and IT, University of Jammu, Jammu, India
[3]Dept. of Computer Science and IT, Kathua Campus, University of Jammu, Kathua, India
[4]Dept. of Computer Science and IT, Ramnagar Campus, University of Jammu, Udhampur, India
[5,6]Dept. of Computer Science and IT, Bhaderwah Campus, University of Jammu, Bhaderwah, India

[*]*Corresponding Author: sourabhshastri@gmail.com, Tel.: +91-94191-71476*

Available online at: www.ijcseonline.org

Received: 06/Oct/2017, Revised: 14/Oct/2017, Accepted: 12/Nov/2017, Published: 30/Nov/2017

***Abstract*—** The analysis of algorithms is a subject that has always arouses enormous inquisitiveness. It helps us to determine the efficient algorithm in terms of time and space consumed. There are valid methods of calculating the complexity of an algorithm. In general, a suitable solution is to calculate the run time analysis of the algorithm. The present study documents the comparative analysis of seven different sorting algorithms of data structures viz. Bubble sort, Selection sort, Insertion sort, Shell sort, Heap sort, Quick sort and Merge sort. The implementation is carried out in Visual Studio C# by creating a Graphical User Interface to calculate the running time of these seven algorithms.

***Keywords*—**Algorithm, Complexity, Runing Time, Sorting, Data Structures

I. INTRODUCTION

The study of algorithm is at the core of all automatic problem-solving activities [1]. Algorithm is a finite set of instructions to accomplish a particular task. The word algorithm comes from the name of a Persian author, Abu Ja'far Muhammad ibn Musa Al-Khwarizmi [2]. An algorithm must satisfy the criteria of Input, Output, Definiteness, Finiteness and Effectiveness. A specified set of inputs are required for an algorithm that produces set of values as output and must terminate after finite number of steps. These steps must be clear and unambiguous. The steps of the algorithm should be effective so that any naïve user can perform the steps. The efficiency of an algorithm is analyzed in terms of running time and memory also referred to as computational complexity. Computational complexity is a characterization of the time or space requirements for solving a problem by a particular algorithm [3]. Thus the analysis of an algorithm requires two main considerations i.e. time complexity and space complexity. The time complexity of an algorithm is the total time for its completion and space complexity is the total memory required for completion. Time and Space complexity is expressed in the form of a function f(n), where n is the input size for a given instance of the problem being solved [4].

Sorting is one of the vital algorithms in computer science for arranging large volumes of data in some logical order. Sorting algorithms can be categorized by number of comparisons, number of swaps, memory usage, recursion, stability and adaptability [5]. Sorting algorithms can be divided into two main categories i.e. internal sort and external sort. An internal sorting algorithm uses internal memory i.e. main memory and external sorting algorithms use external memory i.e. disk or tape for the process. Two fundamental properties of sorting algorithms are in-place and stable. The algorithm is in-place if it does not need extra memory space except for few memory units and the algorithm is stable if it preserves the relative order of any two equal elements.

In this paper, we have created a graphical user interface for the implementation of various sorting algorithms viz. bubble sort, selection sort, insertion sort, shell sort, heap sort, quick sort and merge sort. For calculating the time complexity of these sorting algorithms, Visual Studio C# language has been selected. We have run the same algorithm on ten different runs for each different input size of N = 100, 200, 300, 400 and 500 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 for sorting 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 seven algorithms used for the comparative analysis in paper viz. bubble sort, selection sort, insertion sort, shell sort, heap sort, quick sort and merge sort.

A. Bubble Sort

The bubble sort also known as exchange sort is a comparison based sorting algorithm that works by continually swapping the adjacent elements if they are not in the proper order. The largest element of the list bubble up after one pass. Similarly on each succeeding pass the next largest elements are positioned at appropriate places. An important property of bubble sort is that if there is no swapping of elements in a particular pass, there will be no further swapping of elements in the successive passes [4]. The advantages of bubble sort are simplicity and ease of implementation. The best case occurs when array is already sorted then it takes minimum time and performs n-1 comparisons. That's why time complexity for best case is O(n) and in contrast, the worst case occurs when array is reverse sorted and the algorithm performs n * (n-1) comparisons. The worst case and average case time complexity is O(n*n). Bubble sort takes auxiliary space of O(1).

B. Selection Sort

The selection sort is an in-place sorting algorithm [5] that sorts an array by continually searching the smallest element from unsorted part and inserting it at the beginning of the array. Selection sort 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. Selection sort never makes more than O(n) swaps and can be useful when memory write is a costly operation [6]. The time complexity of selection sort for worst, average and best cases is O(n*n). Selection sort takes auxiliary space of O(1).

C. Insertion Sort

In insertion sort, each iteration takes away an element from the input data and places it into the exact place in the list being sorted. The choice of the element to take away from the input data is random and this process is repetitive until all input elements have been placed at the right places. Playing cards is the best example of insertion sort. Insertion sort is more efficient than bubble sort and selection sort, however all of them have O(n*n) worst case complexity. The advantages of insertion sort are simple implementation and faster than bubble sort but it is efficient for small lists of data. The worst case and average case time complexity is O(n*n) and the best case time complexity is O(n) when the list is already sorted [7]. Insertion sort takes utmost time to sort if elements are sorted in reverse order. Insertion sort takes auxiliary space of O(1).

D. Quick Sort

Quick sort algorithm is based on an algorithmic technique known as divide and conquer. Divide and Conquer is a three step process that consists of divide, conquer and combine. Divide step partitions the array into two sub arrays. The elements of first sub array are less than the elements of second sub array. Conquer step sorts the two sub arrays by recursive calls to quick sort and Combine step means to merge the two subparts which are already sorted to make it entire array [8]. In quick sort, a pivot element is selected then reorganizes all elements of array in such a way that all elements which are smaller than 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]. The worst case occurs when the partition process always picks greatest or smallest element as pivot or in other words worst case occurs when the array is already sorted in increasing or decreasing order and worst case running time for quick sort is O(n*n) but for the average case it is very efficient and has running time of O(n log n). The best case occurs when the partition process always picks the middle element as pivot and best case running time for quick sort is O(n log n).

E. Heap Sort

Heap sort is an in-place algorithm that is part of selection sort family. Heap sort uses a tree data structure called heap to manage information during the execution of the algorithm [8]. To sort the elements using heap sort firstly create a heap by adjusting the array elements and then continually remove the root element of the heap by shifting it to the end of the array and then restore the heap structure with remaining elements [10]. Heap sort has time complexity of O(n log n) in all the cases. Unlike quick sort, the efficiency of heap sort is not affected by the initial order of elements [4].

F. Shell Sort

Shell sort is also called as diminishing increment sort and is developed by Donald L. Shell. It compares the elements at a particular distance using which the elements which are distant can be sorted also. The size of the set to be sorted gets smaller with each pass through the list, until sub list's length

becomes one [11]. 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. Running time of shell sort depends on the choice of increment sequence [5].

G. Merge Sort

Merge sort like quick sort also based on the algorithmic technique known as divide and conquer i.e. divide the list into two halves, recursively sort both half lists, and then merge the two sorted sublists [12]. Merge sort splits the list into two halves, then each half is conquered separately [13]. The advantage of merge sort is that it is used for both internal as well as external sorting and merge sort is a stable algorithm.

III. COMPARATIVE STUDY OF ALGORITHMS

We summarized the best case, average case and worst case complexity of sorting algorithms in the table 1.

Table 1. Complexity Cases [7, 9, 14, 15]

Algorithm	Best Case Complexity	Average Case Complexity	Worst Case Complexity
Bubble	O(n) [modified]	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	O(n)	$O(n^2)$	$O(n^2)$
Quick	O(n log n)	O(n log n)	$O(n^2)$
Merge	O(n log n)	O(n log n)	O(n log n)
Shell	O(n)	Depends on gap sequence	$O(n \log^2 n)$
Heap	O(n log n)	O(n log n)	O(n log n)

IV. PERFORMANCE ANALYSIS USING C#

We implemented all seven algorithms to measure the performance using C# language and calculated the running time in milliseconds 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. 100, 200, 300, 400 and 500) and tried to find running time of each sorting algorithm. For taking the size of the array and selection of the algorithm to sort random array, a graphical user interface has been created as shown in the figure 1.

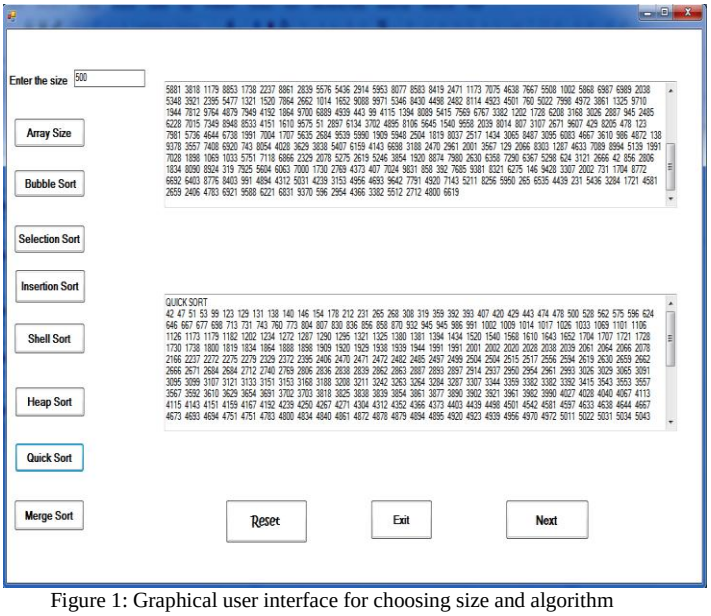

Figure 1: Graphical user interface for choosing size and algorithm

Similarly, a user can create a random array of any size taken as input and run any sorting algorithm as shown in interface of figure 1. Consequently a user can find the running time and average of the sorting algorithm in milliseconds as shown in interface of figure 2.

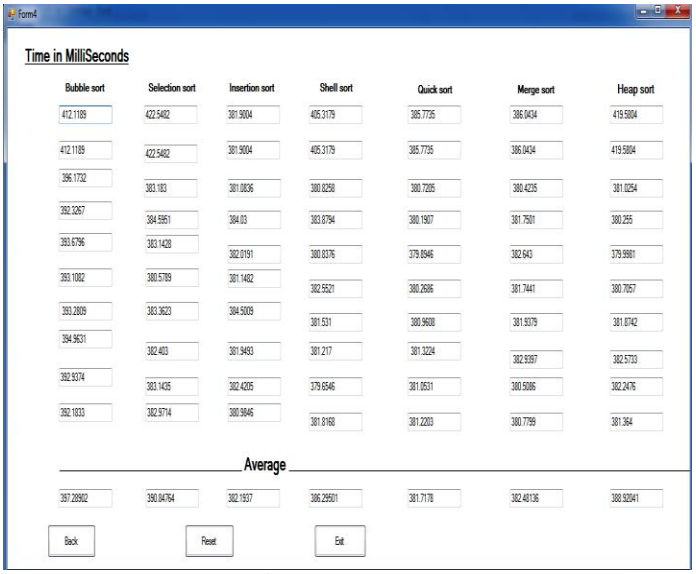

Figure 2: Graphical view of generating time and average in milliseconds

Table 2-6 displays the running time for ten runs and their average in milliseconds for all the seven algorithms. The running time is calculated for the input length of N i.e. 100, 200, 300, 400 and 500 elements.

TABLE 2
RUNNING TIME AND AVERAGE FOR 100 ELEMENTS

Run	Bubble	Selection	Insertion	Shell	Quick	Merge	Heap

1	69.74	76.98	74.73	105.76	68.05	69.92	72.75
2	69.74	76.98	74.73	105.76	68.05	69.92	72.75
3	68.75	68.01	66.64	66.64	67.38	68.87	67.12
4	69.07	67.11	67.72	66.74	67.41	68.05	66.72
5	68.70	68.02	67.44	67.26	66.95	69.83	66.30
6	67.63	68.48	66.67	67.57	67.21	67.59	66.55
7	67.99	67.08	66.69	66.77	66.76	69.53	67.40
8	68.32	67.468	66.64	67.66	67.04	67.19	66.64
9	68.28	67.40	67.43	67.78	67.05	66.90	67.54
10	68.40	67.06	67.70	66.82	68.54	66.95	66.66
Av.	68.66	69.46	68.64	74.88	67.45	68.47	68.04

TABLE 3
RUNNING TIME AND AVERAGE FOR 200 ELEMENTS

Run	Bubble	Selection	Insertion	Shell	Quick	Merge	Heap
1	226.24	197.22	205.32	212.63	195.21	231.19	201.90
2	226.24	197.22	205.32	212.63	195.21	231.19	201.90
3	201.76	193.73	193.70	193.16	192.51	192.56	194.77
4	201.32	194.20	191.11	194.53	193.23	194.33	192.38
5	199.66	194.09	195.20	193.40	193.65	194.65	194.13
6	200.13	193.70	194.80	191.70	193.63	192.54	194.66
7	200.93	194.10	193.31	191.95	193.42	192.60	192.09
8	199.51	193.94	195.21	192.32	193.78	190.32	192.91
9	201.63	193.00	193.40	194.12	193.00	193.20	193.73
10	198.90	193.07	192.56	193.75	192.31	194.85	193.65
Av.	205.63	194.43	195.99	197.02	193.60	200.74	195.21

TABLE 4
RUNNING TIME AND AVERAGE FOR 300 ELEMENTS

Run	Bubble	Selection	Insertion	Shell	Quick	Merge	Heap
1	412.11	422.54	381.90	405.31	385.77	386.04	419.58
2	412.11	422.54	381.90	405.31	385.77	386.04	419.58
3	396.17	383.18	381.08	380.82	380.72	380.42	381.02
4	392.32	384.59	384.03	383.87	380.19	381.75	380.25
5	393.67	383.14	382.01	380.83	379.89	382.64	379.99
6	393.10	380.57	381.14	382.55	380.26	381.74	380.70
7	393.28	383.36	384.50	381.53	380.96	381.93	381.87
8	394.96	382.40	381.94	381.21	381.32	382.93	382.57
9	392.93	383.14	382.42	379.65	381.05	380.50	382.24
10	392.18	382.97	380.98	381.81	381.22	380.77	381.36
Av.	397.28	390.84	382.19	386.29	381.71	382.48	388.92

TABLE 5
RUNNING TIME AND AVERAGE FOR 400 ELEMENTS

Run	Bubble	Selection	Insertion	Shell	Quick	Merge	Heap
1	637.16	596.74	621.23	611.49	598.13	612.57	623.54
2	637.16	596.74	621.23	611.49	598.13	612.57	623.54
3	610.55	596.67	596.34	593.75	598.26	599.39	598.73
4	611.70	596.98	595.35	595.66	595.41	596.14	598.59
5	613.59	596.64	598.04	593.55	596.90	600.86	598.52
6	608.25	596.84	597.01	600.14	595.25	594.95	598.97
7	609.14	596.91	595.29	594.62	597.70	598.77	599.13
8	607.42	597.32	597.19	595.66	595.34	598.11	596.77
9	608.09	599.67	597.15	598.51	598.41	598.80	598.28
10	608.82	598.20	596.32	597.38	596.01	602.41	599.99
Av.	615.19	597.27	601.52	599.22	596.95	601.46	603.61

TABLE 6
RUNNING TIME AND AVERAGE FOR 500 ELEMENTS

Run	Bubble	Selection	Insertion	Shell	Quick	Merge	Heap
1	831.96	830.38	812.49	850.04	816.51	828.01	829.25
2	831.96	830.38	812.49	850.04	816.51	828.01	829.25
3	824.67	816.61	813.37	812.29	810.61	809.06	811.28
4	827.90	809.16	812.11	813.27	810.86	808.29	809.24
5	829.48	810.90	814.03	812.94	812.20	811.19	811.41
6	827.74	815.94	812.40	809.56	808.19	810.35	811.68
7	823.53	812.50	814.12	810.23	807.66	815.63	809.72
8	824.12	811.90	815.54	811.92	810.30	811.51	810.09
9	826.09	813.72	812.30	810.46	811.11	811.54	811.06
10	826.09	813.18	809.77	811.34	810.74	811.30	807.77
Av.	827.42	816.47	812.86	819.21	811.47	814.49	814.08

V. RESULTS AND DISCUSSIONS

The seven sorting algorithms used for the investigation were run ten times for the same length of random array N i.e. 100, 200, 300, 400 and 500 elements and their average was also calculated. Figure 3 displays the average running time in milliseconds of all the seven sorting algorithms taken into consideration during the present study. Out of all results, the quick sort algorithm takes less time as compared to other sorting algorithms and therefore is the most efficient among other discussed algorithms.

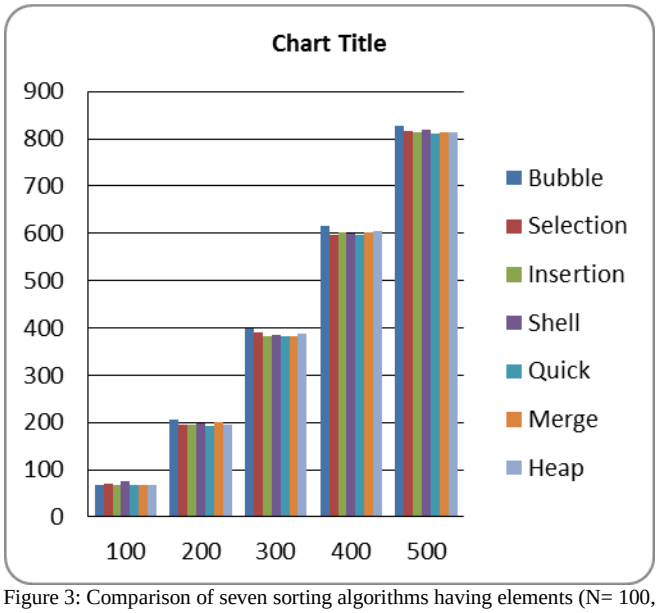

Figure 3: Comparison of seven sorting algorithms having elements (N= 100, 200, 300, 400 and 500).

VI. CONCLUSION AND FUTURE WORK

In this study, we have discussed seven sorting algorithms and their comparison. To find out the running time of all the seven sorting algorithms, we have developed a graphical user interface through C# language in which the user has to input the size of the array elements and to select the algorithm to find the running time and their average. In this piece of

investigation, the data elements of the array are randomly chosen for different data sets of N = 100, 200, 300, 400 and 500. We have calculated the average running time for each algorithm and then showed the result with the help of a graph in figure 3. From the tables 2-6 and figure 3 comparison it is clear that Quick sort is the best and efficient algorithm among all the other algorithms discussed in the study. In the future, we compare the sorting algorithms in different languages to check the running time and memory utilization. In addition to this, we shall try to compare the sorting algorithms in different operating systems.

REFERENCES

[1] S. K. Basu, "*Design Methods and Analysis of Algorithms*", PHI Publication, India, 2005.

[2] Ellis Horowitz, Sartaj Sahni and Sanguthevar Rajasekaran, "*Fundamentals of Computer Algorithms"*, Universities Press Publication, India, 2009.

[3] ISRD Group, "*Data Structures using C"*, Tata McGraw-Hill Publication, India, 2006.

[4] R. S. Salaria, "*Data Structures & Algorithms Using C*", Khanna Book Publication, India, 2012.

[5] Narasimha Karumanchi, "*Data Structures and Algorithms Made Easy in Java"*, CareerMonk Publication, India, 2015.

[6] The GeeksQuiz website. [Online].
Available: http://quiz.geeksforgeeks.org

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

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "*Introduction to Algorithms"*, Prentice-Hall Publisher, India, 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, 2015.

[10] Yashavant Kanetkar, "*Data Structures through C"*, BPB Publication, India, 2010.

[11] Bremananth R, Radhika. V and Thenmozhi. S, "Visualization of Searching and Sorting Algorithms", International Journal of Computer, Electrical, Automation, Control and Information Engineering, Vol.3, No.3, pp.633-641, 2009.

[12] Sanjoy Dasgupta, Christos Papadimitriou and Umesh Vazirani, "*Algorithms"*, Tata McGraw-Hill Publication, India, 2009.

[13] Narasimha Karumanchi, "*Data Structures and Algorithms Made Easy"*, CareerMonk Publication. India, 2011.

[14] 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, No. 5, pp. 1187-1190, 2014.

[15] Sourabh Shastri, Prof. Vibhakar Mansotra and Anand Sharma, "*Sorting Algorithms and their Run-Time Analysis with C#"*, International Journal of Computer Science and Information Technology & Security, Vol. 5, No.5, pp. 388-395, 2015.

Authors Profile

Sourabh Shastri is pursuing Ph.D. and currently working as Assistant Professor in Department of Computer Science and IT, Kathua Campus, University of Jammu, Kathua. He is a life member of Computer Society of India. He has 5 years of teaching experience at University level. His main research work focuses on Analysis and Design of Algorithms, Data Mining and Big Data Analytics.

Prof. Vibhakar Mansotra, Senior Professor 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. He is a life member of Computer Society of India.

Arun Singh Bhadwal has completed his Masters in Computer Applications from Bhaderwah Campus, University of Jammu. He has 2 years of teaching experience at University level. His main research interests are Analysis and Design of Algorithms, Compiler Design and Artificial Intelligence.

Monika Kumari has completed her Masters in Computer Applications from Bhaderwah Campus, University of Jammu. She is currently working as Teaching Assistant at Ramnagar Campus, University of Jammu. Her main research interests are Data Mining, Internet of Things and Analysis and Design of Algorithms.

Avish Khajura has completed his Masters in Computer Applications from Bhaderwah Campus, University of Jammu. His main research interests are Analysis and Design of Algorithms, Machine Learning and Data Mining.

Dalbir Singh Jasrotia has completed his Masters in Computer Applications from Bhaderwah Campus, University of Jammu. His main research interests are Data Mining, Machine Learning and Image Processing.

