Performance Analysis of Sorting Algorithms Through Time Complexity

Blerta Prevalla[1], Ivana Stojanovska[2] and Agni Dika[3]

Abstract – **Performance analysis through time complexity means analysing the time needed for execution of a program. It is very useful because it provides information of how to distribute efforts and resources in order to ensure greater efficiency and to keep developers focused on the essential goals of the program.**
In this paper will be analysed the performance of five sorting algorithms via analytical methods and experimental ones. Sorting algorithms are chosen to analyse because many scientist consider sorting as one of the most crucial problems in the study of algorithms and programs.
Sorting algorithms will be implemented in C++ and tested in 3 machines with different configurations and with different input values to see the changes of running time depending on many circumstances.

Keywords – **Time Complexity, Sorting Algorithms, Performance Analysis.**

I. Introduction

By analysing the performance of a program we mean analysing the amount of computer time needed to run a program. To determine the performance of a program we use two approaches. One is analytical and the other experimental. During the performance analysis we use analytical methods and for performance measurement we conduct experiments.

Performance measurement indicates what a program is accomplishing and whether results are being achieved. It helps programers by providing them information on how resources and efforts should be allocated to ensure effectiveness. Performance measurement must often be coupled with evaluation data to increase our understanding of why results occur and what value a program adds.

The problem of sorting is one of the most widely studied practical problems in computer science, meaning using computer to put files in certain order. Many computer programs use sorting as an mediator step and that's why there are many sorting programs. is faced with the problem of determining which of the many available algorithms is best suited for his purpose.
This task is becoming less difficult than it once was for two reasons. First, sorting is an area in which the mathematical analysis of algorithms has been particularly successful: we can predict the performance of many sorting methods and compare them intelligently. Second, we have a great deal of experience using sorting algo- rithms, and we can learn from that experience to separate good algorithms from bad ones.

[1]Blerta Prevalla is with the Faculty of Information and Communication Technologies, Vojvodina bb, 1000 Skopje, Macedonia, E-mail: blerta.prevalla@fon.edu.mk

[2] Ivana Stojanovska is with the Faculty of Information and Communication Technologies, Vojvodina bb, 1000 Skopje, Macedonia, E-mail: ivana.stojanovska@fon.edu.mk

[2]Agni Dika is with the Faculty of Contemporary Sciences and Technologies, Ilindenska bb, 1200 Tetovo, Macedonia, E-mail: a.dika@seeu.edu.mk

II. Performance analysis

To analyze the performance of an algorithm we must first identify the resourses of primary interest so that the detailed analysis may be properly focused. We describe the process in terms of studying the runing time since it is the resourse most relevant here. A complete analysis of the running time of an algorithm invloves the following steps:

- Implement the algorithm completely.
- Determine the time required for each basic operation.
- Identify unknown quantities that can be used to describe the frequency of execution of the basic operations.
- Develop a realistic model for the input to the program.
- Analyze the unknown quantities, assuming the modelled input.

A. Steps in analysing an algorithm

The first step in analysis is to carefully implement the algorithm on a particular computer. This implementation not only provides a concrete object to study, but also can give useful empirical data to aid in or to check the analysis. Presumably the implementation is designed to make efficient use of resources, but it is a mistake to overemphasize efficiency too early in the process.

The next step is to model the input to the program, to form a basis for the mathematical analysis of the instruction frequencies. The values fo the unknown frequencies are dependent on the input to the algorithm: the input size is normally the primary parameter used to express our results, but the order or value of input data items also ordinarily affect the running time, as well. For these sorting algorithms it is normally convenient to assume that the inputs are randomly ordered and distinct. Another possibility for sorting algorithm is to assume that the inputs are random numbers taken from a relatively large range.

Several different models can be used for the same algorithm: one model might be chosen to make the analysis as simple as possible; another model might better reflect the actual situation in which the program is to be used.

The average case results can be compared with empirical data to verify the implementation, the model, and the analysis. The end goal is to gain enough confidence in these that they can be used to predict how the algorithm will perform under whatever circumstances present themselves in particular applications. For example, we may wish to evaluate the possible impact of a new machine architecture on the performance of an important algorithm. [1]

Often it is possible to do so through analysis, perhaps before the new architecture comes into existence. Another important example is when an algorithm itself has a paramenter that can be adjusted: analysis can show what value is best.

III. Time complexity of sorting algorithms

A. Time Complexity

The time complexity of a program is the amount of computer time it needs to run to completion. [2]
We are mainly interested in that how long does the sorting programs run. It possibly takes a very long time on large inputs until the program has completed its work and gives a sign of life again. Sometimes it makes sense to be able to estimate the running time *before* starting a program. Nobody wants to wait for a sorted phone book for years! Obviously, the running time depends on the number n of the strings to be sorted.

We are interested in the time complexity of a program because some computer systems require the user to provide an upper limit on the amount of time the program will run. Once this upper limit is reached the program is aborted. An easy way out is to simply specify a time limit of a few thousand years. However, this solution could result in serious fiscal problems if the program runs into an infinite loop caused by some discrepancy in the data and you acgtually get billed for the computer time used. We would like to provide a time limit that is just slightly above the expected run time. Also, the program we are developing might need to provide a satisfactory real-time response. For example, all interactive programs must provide such a response.

B. Conducted experiments for sorting algorithms

During the experimental study we:

- Write a program to implement the current algorithm.
- Run the program for different input values.
- Get exact measurements from the actual execution time.
- Compare results.

The analysis of the average-case performance depends on the input being randomly ordered. This assumption is not likely to be strictly valid in many practical situations. In general, this reflects one of the mos serious challenges in the analysis of algorithms: the need to properly formulate models of inputs that might appear in practice. Fortunately there is often a way to circumvent this difficulty: “randomize” the inputs before using the algorithm. This simply amounts to randomly permuting the input file before sort. If this is done, then probabilistic statements about performance such as those made above are completely valid and will accurately predict performance in practise, no matter what the input.

Limitations of experiments:

- It is necessary to implement the algorithm, which may be difficult
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used

In this paper will be shown the experiments done with most famous programs for sorting: Merge Sort, Insertion Sort, Selection Sort, Quick Sort and Bubble Sort.
We will see the following case:

➢ When we have as input random 10000, 15000, 25000, 30000, 45000, 50000, 65000, 75000, 90000, 100000 numbers. With these entries we will see the running time of algorithms when we execute them in machines with different performances like:

- Computers with normal performance (Toshiba Satellite with processor: Intel Core 2 Duo, 1:50 GHz, 500 MHz and 1GB of RAM memory)
- Faster Computer (T555 Dell Studio, the processor: Intel Core 2 Duo P7450 1033 MHz, 2.13 GHz, 1033 MHz and 4GB RAM memory)
- Slower one (Dell Latitude D800, Intel (R) Pentium (R) M processor 1.70 GHz, 209 MHz and 512 RAM memory).

TABLE I
RUNNING TIME OF MERGE SORT ON THREE DIFFERENT MACHINES

N	**Dell studio 1555**	**Toshiba Satellite A200**	**Dell Latitude D800**
10 000	0.3750	0.6050	1.7520
15 000	0.5620	0.7810	2.6340
25 000	0.9220	1.2930	4.3860
30 000	1.0940	1.6250	5.2480
45 000	1.6560	2.6310	7.9210
50 000	1.8590	2.7630	8.7430
65 000	2.3600	3.6440	11.3460
75 000	2.7820	3.8930	13.1190
90 000	3.2810	4.6780	15.7330
100 000	3.6720	5.1820	17.5350

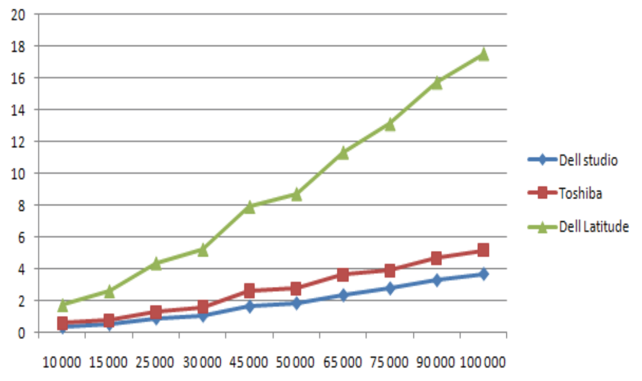

Fig. 1. The comparison of running time for Merge Sort

TABLE II
RUNNING TIME OF INSERTION SORT ON THREE DIFFERENT MACHINES

N	Dell studio 1555	Toshiba Satellite A 200	Dell Latitude D800
10 000	0.1410	0.2440	0.5110
15 000	0.3280	0.4610	1.1810
25 000	0.9060	1.2810	3.8560
30 000	1.3130	1.8420	5.0770
45 000	2.9370	4.6250	10.8760
50 000	3.6560	5.1280	13.4690
65 000	6.0940	8.6220	24.1850
75 000	8.2810	12.5440	30.7340
90 000	11.7500	17.5940	43.8930
100 000	14.6250	20.4590	55.1300

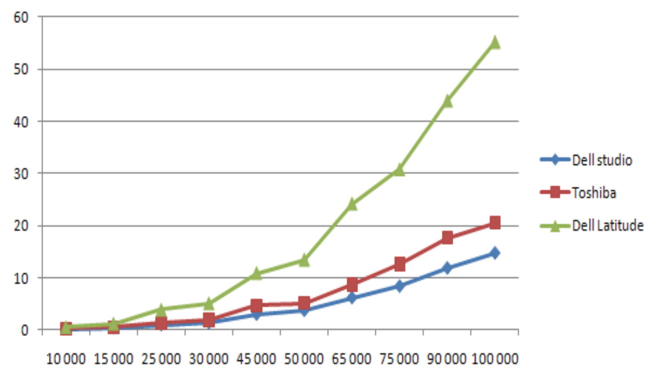

Fig. 2. The comparison of running time for Insertion Sort

TABLE III
RUNNING TIME OF QUICK SORT ON THREE DIFFERENT MACHINES

N	Dell studio 1555	Toshiba Satellite A 200	Dell Latitude D800
10 000	0	0.014	0.030
15 000	0	0.019	0.120
25 000	0.016	0.029	0.120
30 000	0.015	0.039	0.160
45 000	0.032	0.054	0.251
50 000	0.031	0.062	0.320
65 000	0.047	0.080	0.381
75 000	0.047	0.091	0.421
90 000	0.046	0.113	0.541
100 000	0.047	0.128	0.671

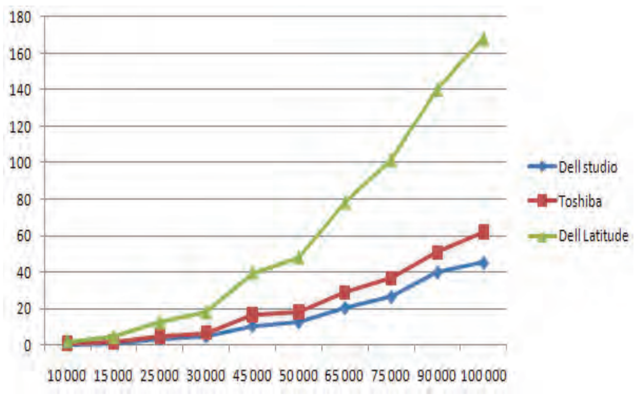

Fig. 3. The comparison of running time for Quick Sort

TABLE IV
RUNNING TIME OF SELECTION SORT ON THREE DIFFERENT MACHINES

N	Dell studio 1555	Toshiba Satellite A 200	Dell Latitude D800
10 000	0.5320	0.8470	2.0720
15 000	1.1720	1.6680	4.6160
25 000	3.2190	4.7890	12.6180
30 000	4.6410	6.6740	18.0960
45 000	10.5150	16.6840	39.5670
50 000	12.5940	18.2250	48.1900
65 000	20.3590	28.8850	78.3920
75 000	26.4380	36.6340	101.5860
90 000	39.8900	50.7140	140.2720
100 000	45.2340	62.0270	168.1320

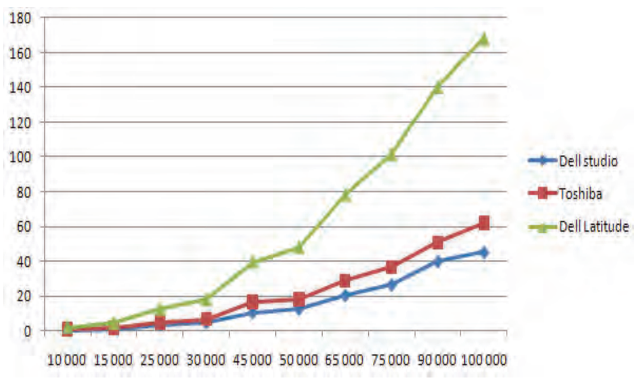

Fig. 4. The comparison of running time for Selection Sort

TABLE V
RUNNING TIME OF BUBBLE SORT ON THREE DIFFERENT MACHINES

N	Dell studio 1555	Toshiba Satellite A 200	Dell Latitude D800
10 000	0.5310	0.8410	2.1630
15 000	1.2040	1.6780	4.8170
25 000	3.3430	4.8870	13.4490
30 000	4.8280	7.7300	19.2380
45 000	10.9850	16.0330	43.3820
50 000	13.3440	19.8430	53.5470
65 000	22.7030	32.6130	90.2900
75 000	30.1560	50.1970	126.4020
90 000	43.4690	67.4900	173.6290
100 000	53.8750	85.1890	216.6920

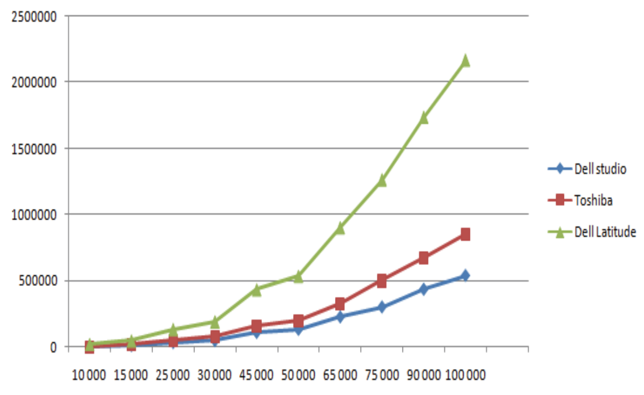

Fig. 5. The comparison of running time for Bubble Sort

From the results presented in the tables and the graphs above, we can see that the performance and the running time of a program depends directly from the machine we are running it.

If we see the Merge Sort program for sorting 100000 random numbers we see that on the faster computer it took 3.672 sekonds to sort them, and in the slower computer it took 17.535 almost 14 seconds more. The faster computer for sorting 100000 elements with Insertion sort program needs 14.625 seconds and the slower computer needs 55.13 seconds to sort those numbers.

We can see that the difference here is bigger than in the Merge Sort, its almost 41 seconds. In contrary, the difference in running time of sorting 100000 numbers with Quick Sort is much smaller, it takes only 0.047 seconds for sorting the elements with the computer with better performances and 0.671 seconds for the slower computer, 0.624 seconds more.
Bubble Sort and Selection Sort in principal are programs that need more time to do sorting, and they are not so propriate to use because they need much more time. Selection Sort running on the faster computer needs 45.234 seconds to sort 100000 numbers and almost 123 seconds more for sorting them with the slower computer.

Even though Bubble Sort its famous for sorting because it is easier to program it, still it takes more time to sort than any other sorting algorithms. It takes 53.875 to sort 100000 elements on the computer with high performances and 216.692 seconds for sorting with the slower computer.
We can conclude that the slower a program is, the bigger is the difference when we execute it on different machines.

IV. CONCLUSION

A full performance analysis like that above requires a fair amount of effort that should be reserved only for our most important algorithms. Fortunately, there are many fundametal methods that do share the basic ingredients that make analysis worthwhile:

- Realistic input models can be specified.
- Mathematical descriptions of performance can be derived.
- Concise, accurate solutions can be developed.
- Results can be used to compare variants and compare with other algorithms, and help adjust values of algorithms parameters.

We conclude performance analysis is very important because we can predict the time needed by a program to solve a problem and also we'll know how to distribute efforts and resources in order to ensure greater efficiency.
These are the areas involving the most significant intellectual challenge, and deserve the attention that they get.

REFERENCES

[1] Sedgewick, R., Flajolet, P. *An Introduction to the Analysis of Algorithms* English Reprint Edition (2006) Pearson Education Asia Limited and China Machine Press
[2] Sahni, S. (2002)*Structures, Algorithms, and Applications in C++*, University of FloridaData McGraw-Hill
[3] Cormen, T.H., & Leiserson, C.E., & Rivest, R., & Stein, C.,(2001) *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill
[4] Flamig, B. *Practical algorithms in C++*, John Ëiley & Sons Inc., Canada, 1995
[5] Knuth,D. *The art of Computer Programming, Volume 3: Searching and Sorting* (2nd ed.), Addison-Wesley, 1998. Longman Publishing Co., Inc., RedWood City, CA, 1998
[6] Mehta, D., Sahni, S. *Handbook of Data Structures and Applications*, (2005) Chapman & Hall / CRC Press Company
[7] Preiss, R.B, Data Structures and Algorithms With Object-Oriented Design Patterns in C++, (1997) Waterloo, Canada