RUN-TIME ANALYSIS FOR SORTING ALGORITHMS

Daniela Alexandra CRIȘAN* [1]
Gabriel Florian SIMION* [2]
Patrick Eugen MORARU* [3]

ABSTRACT:
Analysis of algorithms complexity is an issue that has always aroused great interest. This is because an algorithm, however 'smart' it may seem, it could require a huge execution time. There are analytic techniques of evaluating the complexity of an algorithm, although they usually are difficult to use. Most often, a more convenient solution is to estimate the run-time of the algorithm.

In this paper, a run-time comparative analysis is presented. It consists in evaluating the run-times of three well-known sorting algorithms: ***QuickSort, BubbleSort and InsertSort. A thousand different arrays of different sizes were randomly generated for the tests. Two analyses have been done: the first computes the mean run-time using all 1000 arrays with different sizes, the second uses for 1000 times a single 1000 items array. The three algorithms were implemented in three most common programming languages: Java, C++ and C#.***
The empirical results show that the fastest sorting algorithm is Quicksort, followed by Insertsort, then by Bubblesort. This observation conforms to the theoretical time complexity. A comparison between the three ***programming languages shows that the implementation in Java obtained the shortest run-time, followed by the C++ and C# versions. The order was the same for all three sorting algorithm.***

Keywords: sorting algorithm, complexity, QuickSort, BubbleSort, InsertSort

This paper has been financially supported by scientific research within the project entitled "PRACTICAL SCHOOL: Innovation in Higher Education and Success on the Labour Market", project identified as POSDRU/156/1.2/G/132920. The project is co-financed by the European Social Fund through the Sectorial Operational Programme for Human Resources Development 2007-2013. Investing In people!

1. Introduction

Most often in programming, when an algorithm is developed to solve a particular problem it is not regarded as a complete success. Often it emerges the question of the performance analysis for the created algorithm: complexity and portability on different systems.

[1] * Coresponding author. Associate Professor, PhD, School of Computer Science for Business Management, Romanian-American University, 1B, Expozitiei Blvd., district 1, code 012101, Bucharest, Romania. e-mail: crisan.daniela.alexandra@profesor.rau.ro
2 Bachelor's degree, School of Computer Science for Business Management, Romanian – American University, 1B, Expozitiei Blvd., district 1, code 012101, Bucharest, Romania. e-mail: florin_gabriel89@yahoo.com
3 Bachelor's degree, School of Computer Science for Business Management, Romanian – American University, 1B, Expozitiei Blvd., district 1, code 012101, Bucharest, Romania. e-mail: patrick2262@yahoo.com

Complexity is the measure by which an algorithm can be evaluated in comparison with other algorithms that solve the same problem. The aim of the complexity assess is to provide an information about the performances of an algorithm before transpose it in a programming language and without considering the particularities set of the incoming data.

Evaluating the complexity of the algorithm will assume the estimation of:

- The time complexity: number of executions of a certain statement. Usually this statement is the most significant for the particular problem to solve, referred as the base statement. In sorting problems, the base statement is the comparison between items of the array;
- The space complexity: the amount of supplementary memory needed.

Both time and space complexities are expressed as function in *n* (the input size).

The time complexity is expressed in the O notation. For instance, an algorithm that executes the base statement 2n+3 times is a linear algorithm, having the time complexity O(n). An algorithm that executes the base statement n^2+3 times is a polynomial algorithm, having the time complexity $O(n^2)$. If the algorithm executes the base statement 2^n times, it will be an exponential algorithm, having the time complexity $O(2^n)$.

Three time complexity can be computed:

- the time related to the best-case scenario; sometimes, in sorting problems, this happens when the array is already ordered;
- the time related to the worst-case scenario; sometimes, in sorting problems, this happens when the array is ordered descending;
- the mean time, the most significant of the three, is computed considering all the possible situations for the input data.

The time complexity is a theoretical measure, usually it needs complex theories to be computed, therefore, in some cases, it is estimated using the run-time complexity. This means the algorithm is implemented in any programming language and the run-time is computed on a particular machine.

2. Presentation of the sorting algorithms

2.1. The Quicksort algorithm

Quicksort is one of the most efficient sorting algorithms; it has the theoretical mean time complexity of O(n*log n). The algorithm was developed in 1959 by Charles Antony Richard Hoare (born January 11, 1934). The algorithm was published in 1962. In 1975 Robert Sedgewick wrote his Ph.D. thesis about this sorting algorithm.

When implemented properly, Quicksort can be even three times faster in comparison with other sorting algorithm because of its outstanding performance. Precisely because of these performances, it is the most widely used algorithm.

Nevertheless, like any type of sorting algorithm, sometimes it is not so efficient. The worst case scenario happens when the array is ordered descending, when it will be O (n^2). It should be noted that this behavior is rarely met.

The algorithm is based on a Divide et Impera method, it successively divides the array in two parts: the items in the left being less than a threshold, whilst the items in the right being greater than it.

2.2. The Bubble Sort algorithm

Researches show that the term "Bubble Sort" was used for the first time in 1962, by Ken Iverson [6].

The method was known before as "sorting by exchange" or "exchange sorting".

Like Quicksort, Bubblesort is widely used, but not necessarily because of its efficiency, having O(n^2) theoretical time-complexity, but because of its simplicity.

In the best-case scenario, when the array is ordered ascending, the algorithm performs n-1 comparisons and in the worst-case scenario, when the array is ordered descending, the algorithm performs n*(n-1)/2 comparisons. Therefore, Bubble Sort algorithm has a maximum polynomial time O (n^2), which is equal with its average performance time.

The Bubble Sort method sorts the array by repeated browsing, comparing each item with his successor; after the first browsing it manages to position the largest element of the array on the last position (the final position of the element), phenomenon that suggested its name.

2.3. Direct insertion sort algorithm

At this point there is no information about the inventor of this algorithm. Like Quicksort and Bubble Sort, Direct insertion sort is a comparison sort algorithm. It is a O(n^2) algorithm.

The algorithm is most inefficient when it has to sort an array that is sorted descending. In this case, like the previous sort methods, direct insertion sort algorithm has a maximum polynomial time O (n^2).

The method is to move one element at a time into the correct position by sliding it to the right. The first part of the array, until the first unsorted element, is the sorted part. This algorithm is easy to understand, as well.

3. Presentation of the programing languages

3.1. Java

Java was created by James Gosling, he is also known as the author of the emacs editor and graphical window system News. Sun Microsystems has made this project public in the ‘90s and it was officially released in 1995 at the Sun World in San Francisco.

The language was originally called Oak, being the name of a tree that grew in front of the office of James Gostling. After it was discovered that the name was used by another programming language and it was abandoned replacing it with the name Java, for the joy of the developers who loved coffee and exotic flavors. Towards the end of 2008, Sun was approached by IBM to discuss a possible union, around March 2009, the discussions have started to stagnate between IMB and Sun. By April 20, 2009, Sun and Oracle Corporation announced they are in discussions to buy the Sun Company, later in 2010, ends the purchase of the Sun and it became a division of Oracle.

Java is a language based on object-oriented classes and is easy to use even for non-professional programmers. The applications created in this programming language can be run on any platform that supports Java "write once, run anywhere".

3.2. C++

The C programing language was originally developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs, as a flexible and efficient language. It was extended as the C++ programming language with the help of Bjarne Stroustrup, a Danish computer scientist, that added to it the OOP (Object Oriented Programming) paradigm [2]. In 1985, C++ was implemented as a commercial product. In 1989 the C++ programing language had his first update and in 1998 was published the first international standard for this programing language (ISO/IEC 14882:1998). The last international standard for C++ was released in December 2014 (ISO/IEC 14882:2014) [7].

Since it was developed, C++ influenced many programing languages, two of these languages are C# and Java. In 1990 it became one of the most popular programing languages, remaining popular until today.

3.3. C#

In 1999, during the development of .NET Framework, Anders Hejlsberg, formed a team to build a new language called Cool (C-like Object Oriented Language). Untill July 2000 Microsoft changed the name of the programing language to C# (C sharp). The C# programing language was released to the public in July 2000, but its official release was

in spring 2002 (Version C# 1.0). In April 2003 was released version 1.2, in November 2005 version 2.0, in November 2007 version 3.0, in August 2010 version 4.0 and in 2012 was released the latest version (C# 5.0) [1].

C# is an object-orientated language that has a similar syntax with Java and it is capable to run fast on any computer. This programing language runs on multiple platforms such as Windows, Soralis, UNIX and LINUX.

Although C# is created by Microsoft, there are also C# compilers for other systems. This programming language offers easy and effective ways for writing various programs; it combines time-tested features with last minute innovations. One of these innovations is the interoperability of different programing languages, which is essential in achieving large software systems. Another innovation is mixed language programming.

4. Comparative run-time analysis

4.1. Presentation of the method

For this research were chosen the most used three programming languages (Java, C ++ and C #) in which were implemented the following sorting algorithms: QucikSort, BubbleSort and direct insertion sort, in order to find out which programming language provides the shortest run-time. Quicksort was chosen because it is recognized as the fastest sorting algorithm, Bubble Sort because it is one of the most known, being very easy to remember, and the direct insertion sort was selected because is almost as simple to understand as BubbleSort, but provides a lower execution time.

The execution time was measured in microseconds (10^{-6} seconds), be the most appropriate for the present analysis.

1000 vectors with random sizes between 500-1000 items were randomly generated and stored in a ".csv" file; their items had values between 1 and 5000. Each vector is represented in a row, array's elements are separated by spaces.

Nine sorting functions were implemented: three for every sorting method, in all three programming languages (java, C++, C#). The implementations in Java are listed in the Annex.

The main module follows the next structure:

- reading arrays from the 'csv' file
- for each array do:
    - start-timer
    - sorting function
    - end-timer
    - write time elapsed (expressed in milliseconds) in the result file
- end for

After obtaining all nine run-times series (for every sorting method implemented in all the three programming languages), their averages were calculated.

Two types of comparisons were considered: the first uses the mean run-time obtained with all 1000 arrays with various sizes, whilst the second uses the mean run-time obtained executing 1000 times a random array (the same in all 9 analyses).

4.2. Results

(a) The mean run-time obtained with all 1000 arrays with various sizes

The QuickSort algorithm obtained in Java an average run-time / array of 86.884558 microseconds, in C++ an average run-time / array of 125.322744 microseconds and in C# an average run-time / array of 194.921370 microseconds.

The Direct insertion sort algorithm obtained in java an average run-time / array of 218.808273 microseconds, in C ++ an average run-time / array of 999.756718 microseconds and in C# an average run-time / array of 1817.122120 microseconds.

The Bubble Sort algorithm proved to be the most inefficient, obtaining in java an average run-time / array of 1329.853091 microseconds, in C ++ an average run-time / vector of 3339.091910 microseconds, and in C# an average run-time / vector of 5591.868977 microseconds.

Of the three sorting algorithms the most efficient one was QuickSort, followed by Direct insertion sort who earned an average run-time / array up to nine times higher in C #, eight times higher in C ++ and about three times higher in java, in comparison with QuickSort; on the last place is BubbleSort who earned an average run-time / array up to three times higher in C ++ and C # and 6 times higher in Java in comparison with Direct insertion sort.

Of the three programming languages, Java was the most efficient, obtaining for the BubbleSort algorithm an average run-time / array with up to two times lower in comparison with C ++ and up to four times lower in comparison with C#, for the Direct Insertion Sort algorithm an average run-time / array with up to four times lower in comparison with C ++ and up to eight times lower in comparison with C#, for the Quicksort algorithm an average run-time / array with approximately 1.5 times lower in comparison with C ++ and with approximately 2.3 times lower in comparison with C #.

On the second place is the C ++ programming language which obtained higher times in comparison to Java, but with approximately 1.8 shorter sorting times / vector than C#.

In the last place is C# which obtained the largest average run-time / array, no matter the sort method used.

In the table below you can clearly see the differences between the average time sorting/array, of the sorting methods used in the three programing languages.

Algorithm	JAVA	C++	C#
QuickSort	86.884558	125.322744	194.921370
InsertSort	218.808273	999.756718	1817.122120
BubbleSort	1329.853091	3339.091910	5591.868977

Table 1. First analysis: Average run-times for various arrays (expressed in microseconds)

The next graphic illustrates the differences in run-time of the three methods, corresponding to the three programming languages:

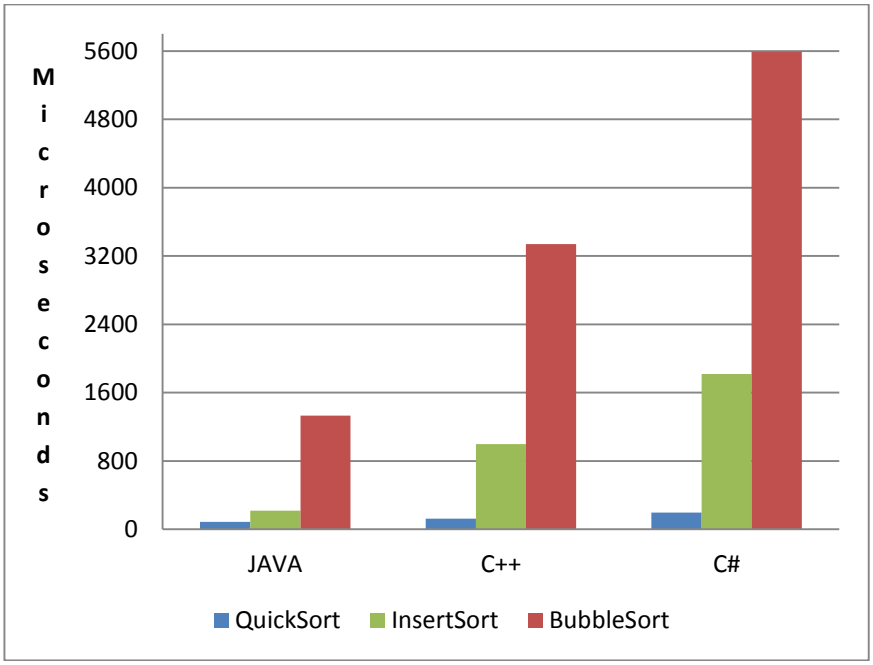

Figure 1. First analysis: Average run-times for various arrays

(b) The mean run-time obtained executing 1000 times a random array

For the second research was chosen the largest vector from the previous research (a vector of 1000 elements) and sorted repetitively, for 1000 times, using each of the three sorting algorithms, implemented in the three programing languages. The same array was used for all nine sorting functions.

The executions times of the sorting algorithm were saved in microseconds as well, in separated files. The averages execution times can be seen in the following table:

Algorithm	JAVA	C++	C#
QuickSort	125.593874	184.135936	280.951012
InsertSort	312.281325	1820.511210	3190.981604
BubbleSort	2404.126812	6028.375680	9205.011703

Table 2. Second analysis: Average run-times for the same array (expressed in microseconds)

As expected the QuickSort algorithm has the shortest execution time, followed by Direct insertion sort algorithm and the last one is the BubbleSort algorithm with the longest run-time.
Also like in the previous research, Java is the most efficient in sorting de vector, followed by C++ and C#.
From de execution time and their graphic representation we can say that the proportions of medium executions time of the sorting algorithms for sorting the same vector is the same proportion seen in the sorting algorithms that sort the 1000 vectors of different lengths.

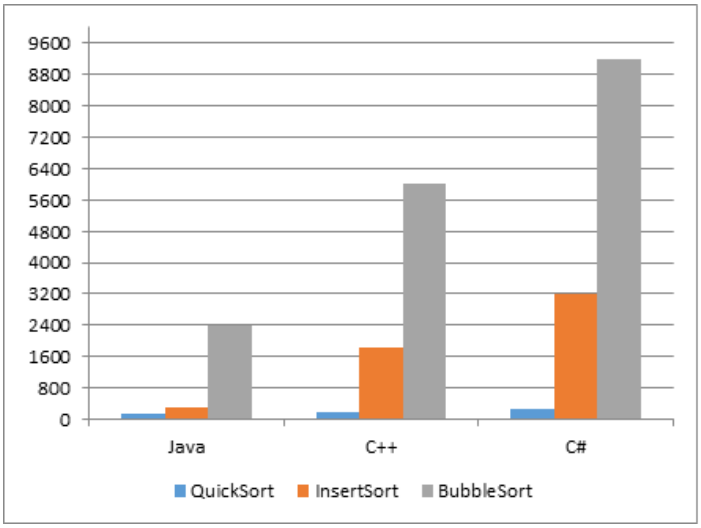

Figure 2. Second analysis: Average run-times for the same array

Conclusions

The empirical evaluations presented in this paper confirmed the theoretical complexities of the three sorting algorithms: Regardless the programming language used for implementation, the most efficient algorithm is QuickSort followed by Insertsort, then by BubbleSort.

Another important observation was that the run-time is strongly influenced by the programming language: the shortest run-time was obtained in the Java case, followed by de implementation in C++ and C#. This result maintained in the case of all three sorting method.

To straighten the conclusion, two tests were imagined: **one using all 1000 arrays with different sizes, the second using for 1000 times a single 1000 items array. The results were similar in the two cases, even the sizes are much increased in the second case since the longest arrays has been used.**

REFERENCES

1. C#, http://aboutcsharpprogramming.blogspot.ro/2012/09/history-of-c-programming.html
2. C++, http://www.cplusplus.com/info/history/
3. Crişan D.A, „Limbajul C/C++. Fundamente", vol I, Ed. Pro Universitaria, Bucureşti, 218 pg., ISBN (10) 973-8994-75-8, (13) 978-973-8994-75-4, 2006;
4. Crişan D.A., Stănică J.L., "Ingineria Programarii. Colectii de date. Tehnici de Programare ", Ed. ProUniversitaria, Bucureşti, 270 pg., ISBN 973-8446-36-8, 2010;
5. Deitel, Paul ,Java How to Program, (2012)
6. DreaminCode, http://www.dreamincode.net/forums/topic/48006-finding-execution-time-of-a-program/
7. Duke, https://www.cs.duke.edu/~ola/bubble/bubble.html
8. Gaddis, T. , Starting Out with Java: From Control Structures through Data Structures (2012)
9. Hernandocadett, http://www.hernandocadett.com/content/brief-history-c-sharp#
10. Hunt, Charlie, Java Performance, (2012)
11. Iverson, K. A. Programming Language. John Wiley, 1962
12. ISO/IEC 14882:2014, Information technology -- Programming languages -- C++, http://www.iso.org/iso/catalogue_detail.htm?csnumber=64029
13. Knuth, D., The Art of Computer Programming: Volume 1: Fundamental Algorithms (1975)
14. Knuth, D., The Art of Computer Programming: Volume 3, The: Sorting and Searching (1975)
15. Liang, D, Introduction to Java Programming: Comprehensive Version (2011)
16. Nell, D.,Programming solving with C++(1997)
17. Oracle Inc, Java Tutorial, https://docs.oracle.com/javase/tutorial/
18. Savitch, W. , Java : An Introduction to Problem Solving, (2012)
19. Sedgewick, R, Algorithms in C++: Part 1-5 (2010)
20. Stackoverflow, http://stackoverflow.com/questions/2329079/how-do-you-convert-stopwatch-ticks-to-nanoseconds-milliseconds-and-seconds
21. Stroustrup, B, The C++ Programming Language, (2009)
22. Wikibooks, http://en.wikibooks.org/wiki/Java_Programming/History

Wikipedia, http://en.wikipedia.org/

ANNEX

Source codes of the sorting functions (quick sort, direct insertion sort and bubble sort), implemented into Java.

Quicksort

```
public static void
QuickSort
(int[] A, int p, int u)
{
        int prag, temp, i, j;
        prag = A[p];
        i = p;
        j = u;
        while (i < j){
                while (i <
u && A[i] <= prag)i++;
                while (A[j]
> prag)j--;
                if (i < j){

        temp = A[i];

        A[i] = A[j];

        A[j] = temp;
                }
        }
        if (p != j){
                A[p] =
A[j];
                A[j] =
prag;
        }
        if (p < j - 1)
QuickSort(A, p, j - 1);
        if (j + 1 < u)
QuickSort(A, j + 1, u);
}
```

Insertsort

```
public static void
InsertSort
(int[] A, int n){
        int curent, i, j;
        for (i = 1; i < n;
i++){
                curent =
A[i];
                for (j = i
- 1; j >= 0 && A[j] >
curent; j--)

        A[j + 1] = A[j];
                A[j + 1]
= curent;
        }
}
```

Bubblesort

```
public static void
BubbleSort
(int A[], int n){
        int temp;
        boolean b = true;
        for (int i = 0;
b&&i < n - 1; i++){
                b =
false;
                for (int j
= 0; j < n - i - 1; j++)

        if (A[j]>A[j +
1]){

        temp = A[j];

        A[j] = A[j + 1];

        A[j + 1] = temp;

        b = true;

        }
        }
}
```