DOI: 10.5604/01.3001.0010.5211

COMPARISON OF PROGRAMMING LANGUAGES ON THE IOS PLATFORM IN TERMS OF PERFORMANCE

Kamil Gut, Maria Skublewska-Paszkowska, Edyta Łukasik, Jakub Smołka
Lublin University of Technology, Institute of Computer Science

Abstract. *In 2014, Apple unveiled a completely new programming language for the iOS and OS X platforms. Swift was presented as a modern programming language, such as: safe, easy to learn and easy to use. This article presents the performance comparison between the Swift and Objective-C languages. For the purpose of the research, two applications were developed, one in each language, implementing sorting algorithms and data structures such as arrays, dictionaries and sets.*

Keywords: Swift, Objective-C, performance, time of sorting algorithms

PORÓWNANIE JĘZYKÓW PROGRAMOWANIA NA PLATFORMIE IOS POD WZGLĘDEM WYDAJNOŚCI

Streszczenie. *W 2014 roku firma Apple zaprezentowała nowy język programowania na platformę iOS oraz OS X. Swift został przedstawiony jako nowoczesny język programowania: bezpieczny, łatwy do nauki i prosty w użyciu. Artykuł przedstawia porównanie wydajności języków Swift i Objective-C biorąc pod uwagę czasy wykonania algorytmów. W celu przeprowadzenia badań powstały w obu językach aplikacje implementujące algorytmy sortowania oraz operacje na strukturach danych takich jak: tablice, słowniki oraz zbiory.*

Słowa kluczowe: Swift, Objective-C, wydajność, czasy algorytmów sortowania

Introduction

The growing demand for mobile devices has contributed to the creation of modern mobile operating systems. Because of the huge demand for the expansion of their functionality, these systems have been equipped with advanced development environments and libraries in order to increase the efficiency of programmers. One of these operating systems is iOS, created by Apple. The factor making a platform attractive for software developers is the language in which this software is developed. Creators of software for the iOS platform use the Objective-C language, built in 1983. It is based on the Small Talk language and is an extension of the C language, giving the possibility of object-oriented programming. This language was originally used in many different areas, and eventually became known as the main programming language used by Apple. Over time, Objective-C became difficult to understand for new developers who had not previously dealt with languages like C or Small Talk. Languages such as Java, C #, Python or JavaScript have become widely used. They have set new standards for modern programming languages. Developers began to complain about Objective-C, which is often regarded as difficult to learn, and very inconvenient to use. These difficulties meant that more and more developers creating applications for iOS and OS X began to shift to software development for Android, which allows them to use the Java language. Apple could not afford to completely change the programming language for its platforms, as that would mean the need to completely rewrite frameworks such as Cocoa [1] or Cocoa Touch [3]. One way to solve this problem was introducing the possibility of using another language while maintaining the option of using code written in Objective-C.

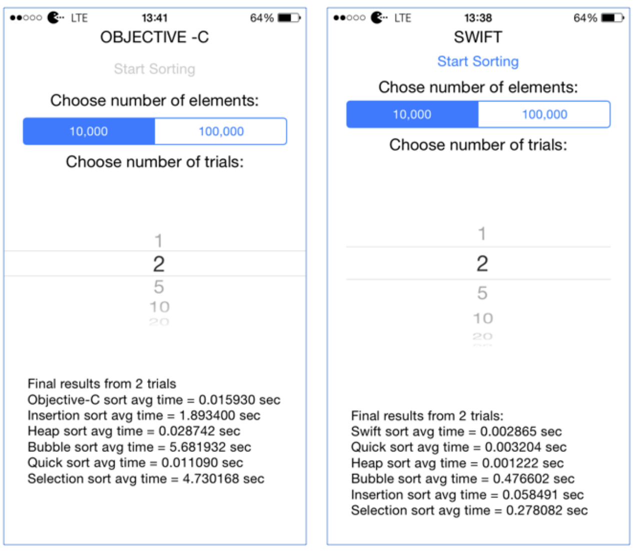
In June 2014, during the annual WWDC conference, Apple presented a new programming language for developers who wanted to create applications for the platforms iOS and OS X. The new programming language was named Swift [4]. This language is quite different than Objective-C [2], nevertheless it ensures compatibility with code written in Objective-C. As a result, Apple may phase out an earlier programming language, replacing it with a new one. Immediately after Swift reached a stable version 1.0, the company began to accept in the App Store applications written in the new language. Swift had been kept secret until the announcement at the WWDC conference, which was a big surprise to the developer community. Apple had to demonstrate to developers that Swift was worth the extra effort and time required for learning it. During the presentation, the company claimed that the language was much more efficient in terms of speed than the current Objective-C, and had all the features common to modern new programming languages, being safe, easy to learn and simple to use.

The aim of the article is to compare the Objective-C and Swift languages in terms of performance time.

1. Applications

1.1. Applications that implement sorting algorithms

In order to conduct performance tests, two applications were developed: one using Objective-C, the other – the Swift language. The applications have a graphical user interface shown in figure 1, and were designed to operate on an iPhone 5s.



```
Final results from 2 trials
Objective-C sort avg time = 0.015930 sec
Insertion sort avg time = 1.893400 sec
Heap sort avg time = 0.028742 sec
Bubble sort avg time = 5.681932 sec
Quick sort avg time = 0.011090 sec
Selection sort avg time = 4.730168 sec
```

```
Final results from 2 trials:
Swift sort avg time = 0.002865 sec
Quick sort avg time = 0.003204 sec
Heap sort avg time = 0.001222 sec
Bubble sort avg time = 0.476602 sec
Insertion sort avg time = 0.058491 sec
Selection sort avg time = 0.278082 sec
```

Fig. 1. Interface of applications implementing sorting algorithms

Both applications implement selected sorting algorithms [6]. The algorithms in Objective-C and Swift were implemented in as similar a manner as was possible, while maintaining due diligence to provide a meaningful comparison. Data types such as NSMutableArray and NSNumber were used deliberately as counterparts of Array and Int in Swift. Methods available in language libraries have also been selected accordingly. Also implemented was the possibility of a choice of test parameters, such as the number of trials and the number of items to be sorted. The applications measure the performance time of various sorting algorithms [5]. If more than one trial is selected, the application calculates the arithmetic average for each of the sorting algorithms.

1.2. Applications that use the XCTest library

The second part of the test was to see how much time it takes to perform operations on data structures such as arrays, dictionaries and sets. The applications implement additions of elements, access to the value of the item and item deletion. To measure the speed of operations performed the XCTest [7] library was used – the default library for creating unit tests in the Xcode environment, supporting two compared languages. The applications do not have a graphical interface. To run them requires a computer running the OS X system, the Xcode development environment and a cable to connect the device to a computer. The results can be read directly from the debugging console or by going to the "Report Navigator" panel in Xcode. Due to the use of its XCTest library, each tested method is activated by default ten times, and the final result consists of the average time of the ten trials.

2. Research methodology

One way to measure performance is to determine how much time it takes to perform an operation. The faster the operation is performed, the higher the performance of the programming language. For analysis, the sorting algorithms such as quick sorting (Quick Sort), heap sorting (Heap Sort), sorting by insertion (Insertion Sort), by selection (Selection Sort), bubble sort (Bubble Sort) and sorting in the standard library of each tested language (Foundation) were used [8]. Another part of the study was to measure the time of the operations performed on data structures such as arrays, dictionaries and sets.

2.1. Sorting algorithms

Sorting data is one of the fundamental problems of development. Sorting algorithms seem to be a good way to compare the performance between programming languages because their computational complexity is known. Implemented in both languages studied with due diligence, that is the selection of the corresponding data types and methods, they are only limited by boundaries and paradigms of the programming languages in which they were implemented.

Applications that use sorting algorithms have been compiled on the corresponding levels of compiler optimisation for each language, installed and running on the same device. Before starting the sorting in both one and the other application, the same parameters were set (the number of items to be sorted and the number of trials). A trial consisted in generating a random array of integers of the interval 0 to 4 294 967 298 (unit32.max), then transferring a copy of the generated array to each of the six sorting algorithms, followed by sorting. The performance time of each sorting algorithm in the test was saved. When selecting more than one trial, the application counts the arithmetic mean of all the trials for each sorting algorithm. To achieve the most reliable and system-independent results, 10 trials were made. During the tests the iOS device worked in the "Aeroplane" mode.

2.2. Operations on data structures

Another way to measure the performance of the Swift language as compared to Objective-C was to see how much time it takes to perform operations such as adding, deleting, and access to an item in the commonly used data structures such as arrays, dictionaries and sets. Data types were suitably selected, thus for Swift: Array Dictionary [2] and Set, and for Objective-C: NSMutableArray, NSMutableDictionary and NSMutableSet [4]. The data structures for both applications were filled with a million elements of the String and NSString type. Due to the use of the XCTest library, each unitary method was run ten times. The result consists of the arithmetic mean of ten trials, just as it did in the case of tests using sorting algorithms.

2.3. Tests

All tests were carried out on the iPhone 5s with the following parameters:

- processor: Apple A7 with 64-bit architecture, dual-core, clocked at 1.3 GHz;
- RAM: 1 GB of RAM;
- internal memory: 32 GB;
- system: iOS 8.3 (12F70).

In the case of the application investigating the sorting time, ten trials were made with ten thousands of items to sort.

Using applications investigating the time of operations on data structures, the number of elements which filled the data structures amounted to a million.

Applications were compiled at the default level of compiler optimisation for both languages, suggested by the Xcode environment for the "Release" version. In the case of Swift it was the Fast [O] level, and for Objective-C – the Fastest, Smallest [-Os].

Additionally, for applications that implement sorting algorithms a test was conducted without optimisation. For this purpose, applications were compiled at code optimisation levels used during software development. For Swift the level was None [-O0], and for Objective-C – None [-Onone].

3. Results

In order to visualise the exact results of the tests, graphs with the obtained results were created. The tables show the calculated values representing as the quotient the performance time in Objective-C by the performance time in Swift language.

3.1. Sorting – standard level of optimisation algorithms

The results for sorting at the standard level of optimisation are shown in figures 2 and 3 and in table 1.

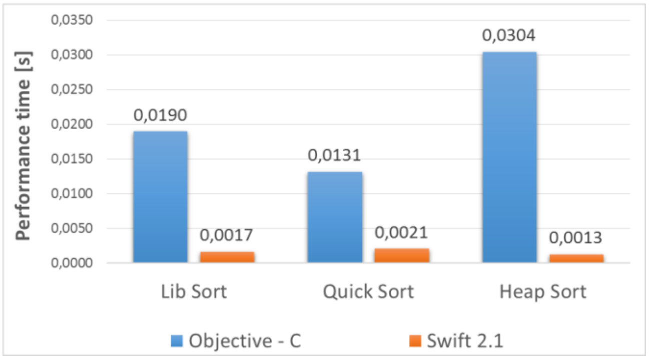


Fig. 2. Comparing the performance times of the sorting algorithms at the standard optimisation level (part 1)

Table 1. The ratio of the performance time of algorithms in Objective-C to Swift for the sorting algorithms

Sorting algorithm	Objective-C / Swift
Lib	11.37
Quick	6.40
Heap	23.70
Insert	32.50
Select	17.44
Bubble	12.90

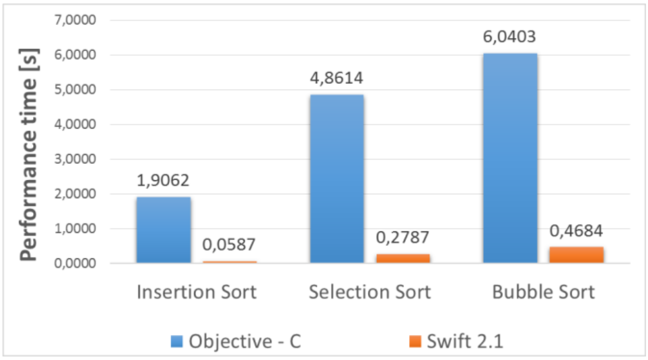


Fig. 3. Comparing the performance times of the sorting algorithms at the standard optimisation level (part 2)

3.2. Operations on data structures – the standard level of optimisation

The results of measuring the speed of operations on data structures are shown in figures 4, 5 and 6. A summary for each of the data structures can be found in tables 2.

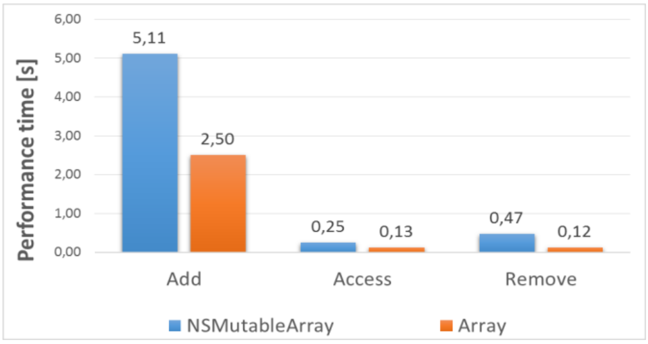


Fig. 4. Comparison of operation performance time – arrays

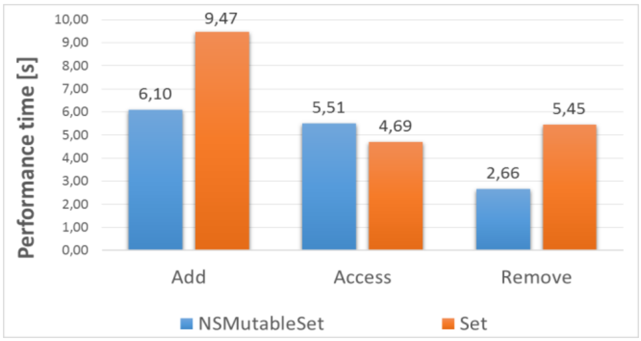


Fig. 5. Comparison of operation performance time – sets

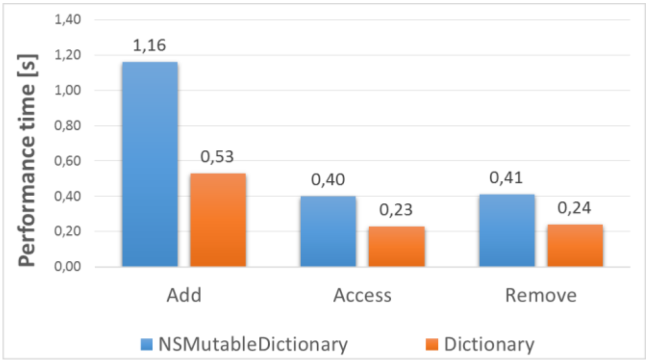


Fig. 6. Comparison of operation performance time – dictionaries

Table 2. The ratio of the time in Objective-C to Swift for operation on data structures

Data structure	Add	Access	Remove
Arrays	2.04	1.92	3.92
Sets	0.64	1.17	0.49
Dictionaries	2.19	1.74	1.71

3.3. Sorting – without optimisation

The results of the test checking the sorting speed in Swift and Objective-C in the "Debug" mode are shown in figures 7 and 8 and in table 3.

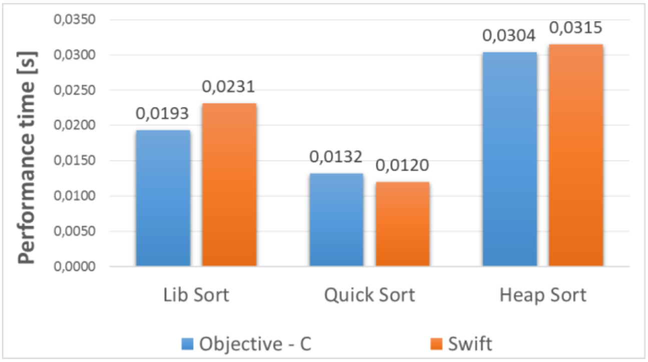


Fig. 7. Comparing the performance times of the sorting algorithms without code optimisation (part 1)

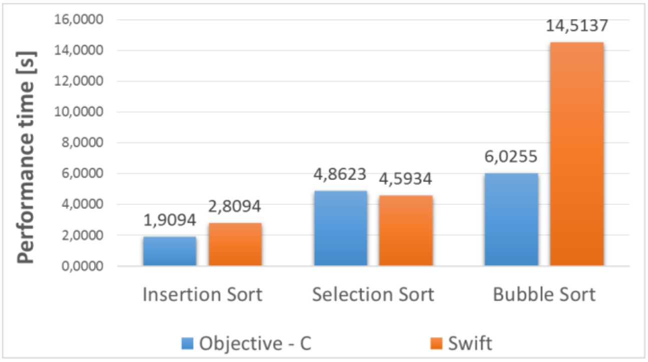


Fig. 8. Comparing the performance times of the sorting algorithms without code optimisation (part 2)

Table 3. The ratio of the performance time of algorithms in Objective-C to Swift for the sorting algorithms without code optimisation

Sorting algorithm	Objective-C / Swift
Lib	0.84
Quick	1.10
Heap	0.97
Insertion	0.68
Selection	1.06
Bubble	0.42

3.4. Operations on data structures – without optimisation

The results of the test checking the sorting speed in Swift and Objective-C in the "Debug" mode are shown in figures 9, 10 and 8 and in table 4.

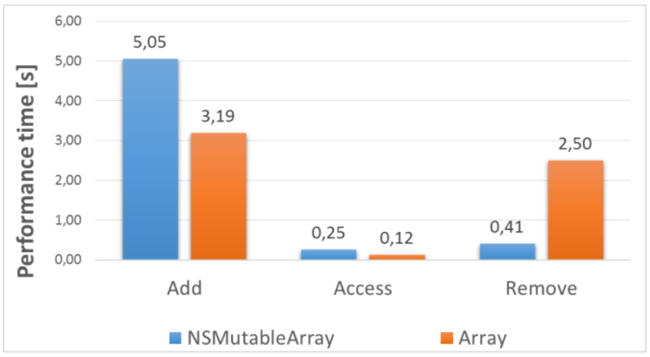


Fig. 9. Comparison of operation performance time without optimisation – arrays

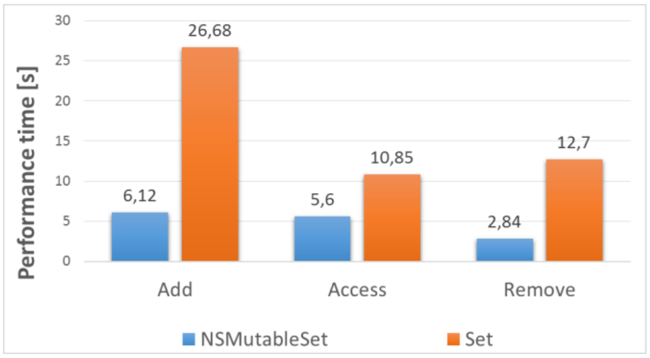


Fig. 10. Comparison of operation performance time without optimisation – sets

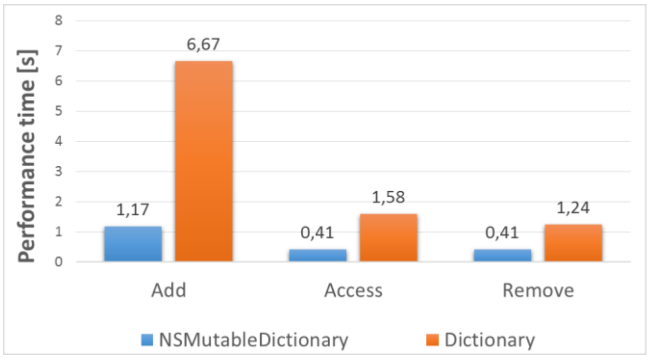


Fig. 11. Comparison of operation performance time without optimisation – dictionaries

Table 4. The ratio of the time in Objective-C to Swift for operation on data structures without optimisation

Data structure	Add	Access	Remove
Arrays	1.58	2.08	0.16
Sets	0.23	0.52	0.22
Dictionaries	0.18	0.26	0.33

4. Conclusions

Both in the tests using sorting algorithms and in operations on data structures one can see the advantage of the Swift language in speed operations. A special role is played here by the Swift compiler, which is seen in figures 1 and 2, and 6 and 7. The difference in performance time of the sorting algorithms at a standard level of optimisation and that without optimisation is huge. For operations without optimisation on the relevant data structures: arrays, collections and dictionaries, Objective-C was much faster. Swift with optimization reached lower execution times on data structures.

Swift uses static typing, so that the compiler can use the knowledge about the types to carry out a wide range of optimisation.

Bibliography

[1] Hillegass A., Preble A., Chandler N.: Cocoa Programming for OS X: The Big Nerd Rach Guide(5th Edition), Big Nerd Ranch, 2015.
[2] Hillegass A., Ward M.: Objective-C Programming: The Big Nerd Ranch Guide (2nd Edition), Big Nerd Ranch, 2013.
[3] Kelley J.: Learn Cocoa Touch for iOS, Apress, 2012.
[4] Mathias M., Gallagher J.: Swift Programming: The Big Nerd Ranch Guide, Big Nerd Ranch, 2015,
[5] Pollice G.: Algorithms in a Nutshell, O'Reilly, 2008.
[6] Wróblewski P.: Algorytmy, struktury danych i techniki programowania, Helion, 2015.
[7] About Testing with Xcode – Apple Developer, https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/01-introduction.html, [15.07.2016]
[8] Framework Foundation, https://developer.apple.com/reference/foundation, [12.09.2016]

M.Sc. Eng. Kamil Gut
e-mail: kamilgut01@gmail.com

A graduate of the Faculty of Computer Science, Electronics and Telecommunications of the AGH University of Science and Technology where he received his Eng. He also graduated the Faculty of Electrical Engineering and Computer Science of the Lublin University of Technology, where he received his M.Sc. specialization in Web Application. He interests in market, motor sports, new technologies and retro gaming. He works as .net developer.

Ph.D. Eng. Maria Skublewska-Paszkowska
e-mail: maria.paszkowska@pollub.pl

Researcher-lecturer in the Institute of Computer Science at the Faculty of Electrical Engineering and Computer Science of the Lublin University of Technology, where she received her M.Sc. She obtained her Ph.D. at the Silesian University of Technology. Her research activity is connected with: motion capture methods, 3D motion data analysis, 3D algorithms and mobile programming.

Ph.D. Edyta Łukasik
e-mail: e.lukask@pollub.pl

Graduated in mathematics at the Marie Skłodowska-Curie University in Lublin. Received her Ph.D. from the Faculty of Mathematics, Physics and Computer Science there. Since 1998 member of the academic staff of the Lublin University of Technology. Her research interests are mainly programming languages and algorithmization, data structures, numerical and optimization methods, mobile applications as well as acquisition methods of 3D motion.

Ph.D. Eng. Jakub Smołka
e-mail: jakub.smolka@pollub.pl

Research worker at the Institute of Computer Science, Faculty of Electrical Engineering and Computer Science at the Lublin University of Technology. Earned his master's degree there, and his doctoral degree at the Silesian University of Technology. His research activity is in the area of processing digital images, in particular their segmentation and compression, 3D motion analysis and mobile programming.

otrzymano/received: 14.09.2016 *przyjęto do druku/accepted: 14.08.2017*