JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 12 (2019) 246-251
Wysłane: 2019-06-18
Przyjęte: 2019-06-26

Analiza rozwoju środowiska uruchomieniowego systemu Android

Kostiantyn Honcharenko*, Jakub Smołka
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono rozwój środowiska uruchomieniowego systemu Android. Zaprezentowano własną aplikację dla systemu Android, w której zaimplementowano testy wydajności wykorzystane do przeprowadzenia badań w różnych wersji środowiska. Test jest metodą, która implementuje różne operacji w systemie Android oraz mierzy czas ich wykonania.

Słowa kluczowe: Android; środowisko uruchomieniowe; Dalvik; ART

*Autor do korespondencji.

Adres e-mail: kostiantyn.honcharenko@pollub.edu.pl

Analysis of the development Android's runtime

Kostiantyn Honcharenko*, Jakub Smołka
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents the development of the Android runtime. Own application for Android is presented, which implements performance benchmarks used to test different versions of the Android runtime. The methods measure the benchmark execution times in different versions of the Android runtime environment.

Keywords: Android; run time; Dalvik; ART

*Corresponding author.

E-mail address: kostiantyn.honcharenko@pollub.edu.pl

1. Wstęp

Historia wersji systemu operacyjnego Android rozpoczęła się wraz z wydaniem Androida beta w listopadzie 2007. Pierwsza komercyjna wersja, czyli Android 1.0, została wydana we wrześniu 2008 roku [1]. System ten doczekał się wielu aktualizacji, które krok po kroku modyfikują jego pierwotną wersję. Aktualizacje te są wydawane najczęściej w celu poprawiania błędów, zmian wizualnych, dodawania nowych funkcji i zwiększania ogólnej wydajności systemu.

Zaczynając od wersji 1.0 do wersji 4.3 Android wykorzystywał maszynę wirtualną Dalvik, jako domyślnie środowisko, w którym działały wszystkie aplikacje Android. W wersji 4.4 po raz pierwszy pojawiła się alternatywa – ART (Android Runtime). Użytkownik mógł własnoręcznie, w ustawieniach telefonu, zmienić środowisko wykonawcze z Dalvik na ART i odwrotnie. Twórcy Androida skupili się na wymianie kompilatora JIT (just-in-time) na AOT (ahead-of-time) i zapewniali, że zmiana kompilatora przyspieszy pracę urządzenia. Już od wersji 5.0, całkowicie zrezygnowano z maszyny Dalvik i na wszystkich urządzeniach środowisko ART było stosowane domyślnie.

2. System Android

System Android oparty jest na jądrze Linuksa. Jądro zarządza zasobami smartfona, w tym dostępem do sprzętu, pamięcią stałą, uruchamianiem, zatrzymywaniem i migracją procesów między rdzeniami procesora jak i wielozadaniowością. Struktura systemu jest widoczna na rysunku 1. Jak w przypadku każdego innego systemu, jądro — to serce systemu Android [3].

W najniższej warstwie systemu Android znajduje się jądro systemu operacyjnego. Zapewnia ono funkcjonowanie systemu i jest odpowiedzialne za bezpieczeństwo, zarządzanie pamięcią, zużyciem energii, procesami, a także stosem sieciowym i sterownikami. Jądro jest najważniejszą częścią systemu operacyjnego Linux, w przeciwieństwie do innych jego części, zostało ono przeniesione do systemu Android prawie bez zmian [4].

Warstwa abstrakcji sprzętowej (Hardware Abstarction Layer) udostępnia standardowe interfejsy, które mapują możliwości urządzeń sprzętowych na wyższy poziom interfejsu Java API. HAL składa się z wielu modułów biblioteki, z których każdy implementuje interfejs dla określonego typu składnika sprzętowego.

Powyżej w kolejnej warstwie systemu operacyjnego, znajduje się zestaw bibliotek (Native C/C++ Libraries), przeznaczony do rozwiązywania typowych zadań wymagających wysokiej wydajności. Ta warstwa jest odpowiedzialna za:

- Zapewnienie realizowanych algorytmów dla kolejnych warstw;
- Wsparcie formatów plików;
- Realizację kodowania i dekodowania informacji (na przykład, multimedialne kodeki);

- Renderowanie grafiki i wiele więcej.

Biblioteki są zaimplementowane w języku C/C++ i skompilowane dla konkretnej architektury urządzenia, wraz z którym są dostarczane przez producenta [5].

Środowisko uruchomieniowe Android (ART) pozwala uruchomić wiele wirtualnych maszyn na urządzeniach o małej pamięci, wykonując pliki DEX, format bajtowy zaprojektowany specjalnie dla Androida, zoptymalizowany pod kątem wykorzystania pamięci.

Rys. 1. Struktura systemu Android [3]

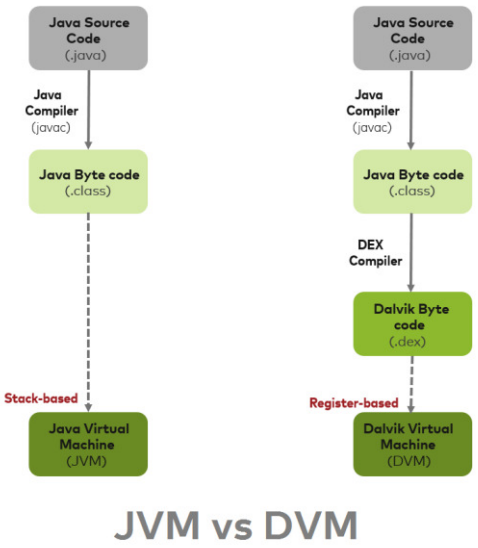
Cały zestaw funkcji systemu Android jest dostępny dla użytkowników za pośrednictwem interfejsów API napisanych w języku Java. Te interfejsy API tworzą bloki potrzebne do tworzenia aplikacji na Androida, upraszczając ponowne wykorzystanie podstawowych, modułowych komponentów systemu i usług.

Na szczycie w architekturze systemu Android znajduje się warstwa aplikacji (Applications). W skład tej warstwy wchodzą: przeglądarka www, klient poczty elektronicznej, program do wysyłania SMSów, mapy, kalendarz, menedżer kontaktów i wiele innych. Oprócz tego podstawowego zestawu do poziomu aplikacji należą wszystkie aplikacje dla platformy Android, w tym te zainstalowane przez użytkownika [3].

2.1. Dalvik

Dalvik Virtual Machine jest częścią platformy mobilnej Android. Jest to maszyna wirtualna, której autorem jest Dan Bronstein. Kod źródłowy jest rozpowszechniany jako wolne oprogramowanie na warunkach zgodnej z BSD licencji Apache 2.0. W dużej mierze fakt ten odegrał swoją rolę w decyzji Google aby zrezygnować z JME (Java Micro Edition) [6], na którą należało uzyskać licencję od firmy Sun.

Dalvik wykorzystuje własny kod bajtowy. Aplikacje dla Androida są tłumaczone przez kompilator do postaci specjalnego niezależnego od maszyny niskopoziomowego kodu. Podczas uruchamiania kodu na platformie Dalvik, ona interpretuje i wykonuje program [7]. Ponadto, Dalvik jest w stanie przetłumaczyć kod bajtowy Java do kodu we własnym formacie, a także wykonywać go w swoim środowisku wirtualnym. Przykłady przetwarzania kodu maszyn wirtualnych Java oraz Dalvik przedstawiono na rysunku 2. Kod aplikacji tworzy się w języku Java, a po kompilacji pliki .class mogą być konwertowane do formatu .dex (nadające się do interpretacji w maszynie Dalvik) za pomocą compilatora DEX, wchodzącego w skład SDK systemu Android.



Rys. 2. Przetwarzania kodu JVM a Dalvik VM [8]

Dalvik został zaprojektowany specjalnie dla platformy Android. Uwzględniono fakt, że platforma uruchamia wszystkie procesy w izolacji, każdy w swojej przestrzeni adresowej. Maszyna wirtualna jest zoptymalizowana pod kątem niskiego zużycia pamięci i pracy na mobilnym sprzęcie. Począwszy od wersji systemu Android 2.2, Dalvik, używa kompilacji JIT (Just-in-Time) [7].

2.2. ART

ART to środowisko uruchomieniowe aplikacji Android, które zostało opracowane przez Google, jako zamiennik maszyny Dalvik. Główną różnicą jest zmiana kompilatora JIT na kompilator AOT [9].

Podczas instalacji oprogramowania aplikacji, kod bajtowy dex kompiluje się do kodu maszynowego (kompilacja AOT), podczas gdy Dalvik kompiluje kod bajtowy do dex (Dalvik executable) a po uruchomieniu programu do kodu maszynowego w czasie rzeczywistym (kompilacja JIT). Z tego względu ART przyczynia się do oszczędzania energii. Co prawda, odbywa się to kosztem zwiększenia ilości używanego miejsca i spowolnienia instalacji aplikacji. Google twierdzi, że spowolnienie nie jest krytyczne. Innowacje w rodzaju ART stały się możliwe dzięki zwiększeniu ilości pamięci w nowoczesnych smartfonach [1].

Kompilator AOT również przegrywa z kompilatorem JIT pod względem możliwości optymalizacji kodu maszynowego. Po prostu nie ma wystarczająco dużo informacji na temat zachowania aplikacji i specyfiki jej pracy. Można było ją uzyskać, tylko uruchamiając aplikację. Dodatkowo kompilator AOT znacznie zwalniał instalację aplikacji i pierwsze uruchomienie systemu operacyjnego [10].

W czerwcu 2014 roku firma Google poinformowała o obsłudze 64-bitowej architektury w wersji Android L. Do zalet architektury 64-bitowej wymienionych w informacji firmy Google zaliczają się: zwiększona liczba rejestrów, nowe zestawy instrukcji i rozszerzone adresowane miejsca w pamięci, co pozwala do adresowania większej puli adresów [11].

Jeśli aplikacja jest napisana w języku Java, kod automatycznie skorzysta z zalet nowej 64-bitowej architektury. Firma Google zaktualizowała NDK do wersji 10b i dodała obraz emulatora, który można wykorzystać do przygotowania aplikacji do pracy na urządzeniach z 64-bitowymi procesorami Intel. Jest to bardzo wygodne do tworzenia aplikacji dla użądzeń z 64-bitową architekturą [11].

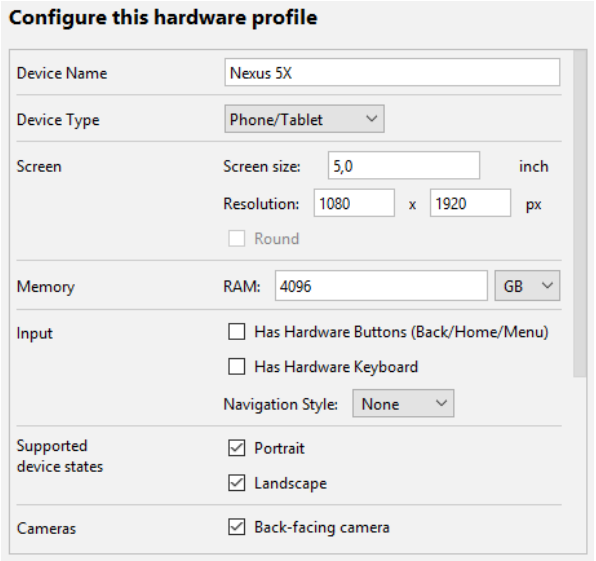
W systemie Android N dodano dynamiczny kompilator JIT z profilowaniem kodu dla środowiska ART, który pozwala stale podnosić wydajność aplikacji Android w trakcie ich pracy. Kompilator JIT uzupełnia bieżący kompilator AOT i pomaga zwiększyć wydajność, zmniejszyć zużycie pamięci, a także przyspieszyć aktualizacje systemu i aplikacji [12].

Jedną z najbardziej wymiernych korzyści kompilatora JIT w ART jest szybkość instalacji aplikacji i aktualizacji systemu. Nawet duże aplikacje, które wymagają kilku minut w celu ich optymalizacji i instalacji na Androidzie 6.0, w aktualnych wersjach systemu Android mogą być zainstalowane w ciągu kilku sekund [12]. Aktualizacje systemu również są wykonywane szybciej, ponieważ etap optymalizacji teraz jest nieobecny.

W Android Pie została wprowadzona optymalizacja profili ART w Play Cloud - nowa funkcja optymalizacji, która znacznie poprawia czas uruchamiania aplikacji po nowej instalacji lub aktualizacji. Średnio aplikacje uruchamiają się o 15% szybciej [13] (zimny start) na różnych urządzeniach.

3. Aplikacja badawcza

Dla uruchomienia aplikacji badawczej wydajności środowiska Androida od wersji 4.1 do 9.0, zdecydowano się wykorzystać emulator Android Studio. Dla emulatora wybrano profil urządzenia Nexus 5X, który przedstawiony na rysunku 3. Emulator ma ekran 5 cali z rozdzielczością 1080x1920 pikseli i RAM o ilości 4096 MB.



Rys. 3. Główny widok aplikacji

Testy wykonano z pomocą aplikacji stworzonej na Androida z wykorzystaniem Android SDK. Aplikacja jest kompatybilna ze wszystkimi wersjami Androida, licząc od Androida 4.1. Do testowania środowiska uruchomieniowego od wersji Android 4.1 do wersji Android 9.0 zostały przygotowane następujące scenariusze testowe:

- Szyfrowanie tekstu algorytmem AES;
- Sortowanie tablicy typu long;
- Operacji z tablicą.

Strukturę części kodu, który bada wydajność środowiska, pokazano na listingu 1. Po kliknięciu przyciska rozpoczęcia testu, aplikacja włącza stoper i zaczyna wykonywać scenariusz testowy. Po poprawnym zakończeniu testu stoper wyłącza się, rezultat wyświetla się na ekranie emulatora i się resetuje.

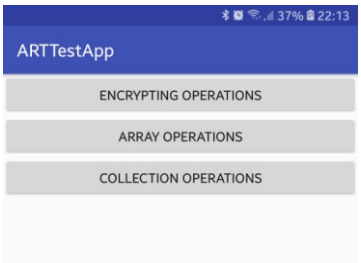
Przykład 1. Metoda badająca wydajności

```
start.setOnClickListener(view -> {
    try {
        stopwatch.start();
        // scenariusz testowy
        stopwatch.stop();
        resultText.setText(stopwatch.toString());
        stopwatch.reset();
    } catch (Exception e) {
        e.printStackTrace();
    }
});
```

Aplikacja składa się z 3 tematycznych klas dziedziczących po Activity, z których każda uwzględnia przypadek testowy. Rysunek 4 przedstawia główny widok programu, w którym można wybrać rodzaj testów.

Do zapisu dużej ilości danych, na przykład dla zachowania tablicy o wielkości 10 000 elementów, wykorzystano package assets, który jest używany dla

przechowywania plików z informacją, które będą przetwarzane i używane w przypadku potrzeby. Ponieważ maszyna wirtualna Java ogranicza długość nazw pól i metod, opisów pól i metod, a także innych stałych wartości ciągów do 65535 znaków [14].



Rys. 4. Główny widok aplikacji

4. Wyniki testów

W ramach badań wykonano kilka rodzajów testów reprezentujących elementarne operacje wykonywane w aplikacjach. Każdy test był powtarzany 25 razy. Opisano je poniżej.

4.1 Szyfrowanie tekstu

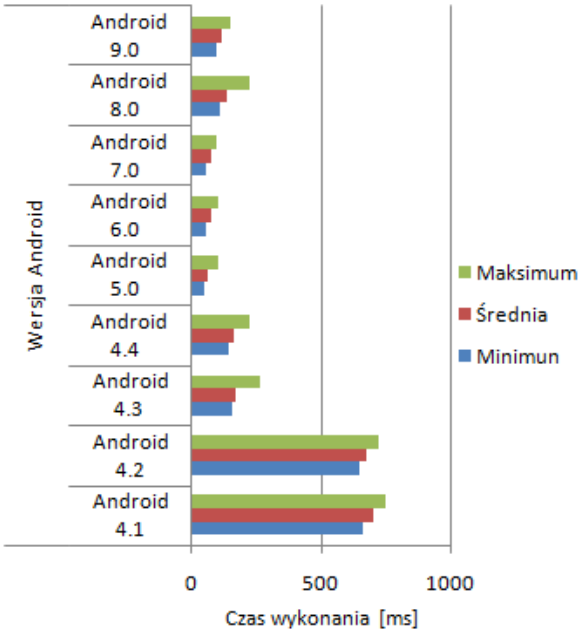
W trakcie testu najpierw tworzona jest zmienna String o długości 100 000 znaków, po czym, z pomocą symetrycznego szyfru AES bajty są szyfrowane kluczem o rozmiarze 192 bit. Następnie uzyskana tablica jest deszyfrowana i otrzymywane są dane początkowe.

Tabela 1. Wyniki testu szyfrowania tekstu

		Minimun	Średnia	Maksimum
Czas wykonania [ms]	Android 4.1	663	707,2	752
	Android 4.2	651	676,52	728
	Android 4.3	159	173,44	271
	Android 4.4	150	170,92	229
	Android 5.0	56	66,44	109
	Android 6.0	63	78,72	107
	Android 7.0	61	80,32	99
	Android 8.0	112	142,4	228
	Android 9.0	104	120,52	155

Przejście z Dalvik na nowe środowisko uruchomieniowe dało znaczny wzrost wydajności (Rysunek 4, tabela 1). Najszybciej z zadaniem poradził sobie emulator z systemem Android w wersji 5.0. Jednak zaczynając od wersji 5.0 do wersji 9.0, czas wykonywania testu wzrósł.

Najwolniejszym systemem Android w wersji ze środowiskiem ART okazała się wersja 8.0, która ma maksymalny czas wykonywania testu zbliżony do wskaźników wersji 4.3 i 4.4. Rezultaty testu zamieszczono w tabeli 1.



Rys. 4. Wykres wartości minimalnej, medialny oraz maksymalnej dla testu szyfrowanie tekstu

4.2 Operacje na kolekcjach

Test polega na tworzeniu nowej kolekcji ArrayList z tablicy typu long z liczbą elementów wynoszącą 10 000 w zakresie od 0 do 9223372036854775807. Tablica została wstępnie załadowana z zasobów assets. Po tym kolekcja jest sortowana i wykonywane jest ponowne zapisanie elementów. Dalej do utworzonej, nowej kolekcji skopiowano zawartość poprzedniej za pomocą klasy Collections. W realizacji tego testu użyto klasy Collections, która zawiera zestaw przydatnych metod pomagających wykonywać różne czynności z kolekcjami. Kod źródłowy opisanogo testu przedstawiono na listingu 2.

Przykład 2. Operacji z kolekcją

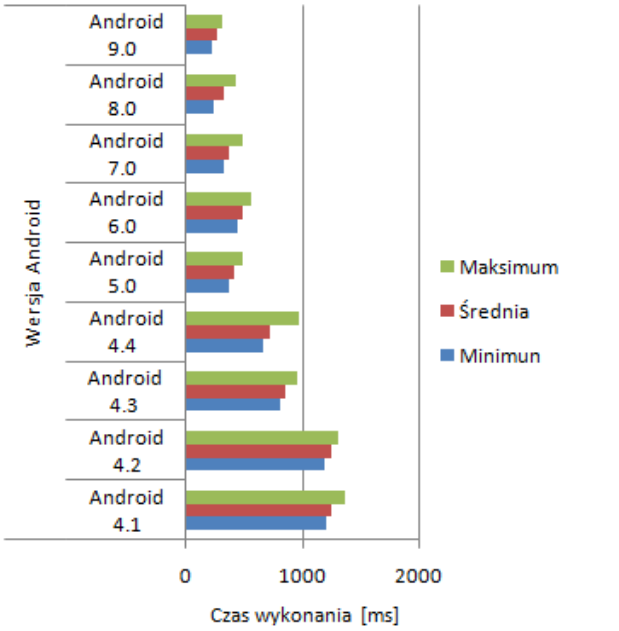
```
private void collectionOperation (long[] longArrTmp) {
    ArrayList<Long> longColl = new ArrayList<>();
    for (Long l : longArrTmp) {
        longColl.add(l);
    }
    Collections.sort(longColl);
    Collections.reverse(longColl);
    ArrayList<Long> newLongColl = new ArrayList<>();
    Collections.copy(longColl, newLongColl);
}
```

Rezultaty wykonania zaprezentowano w tabeli 2 i na rysunku 6. Otrzymano następujące wyniki - od wersji 4.1 do wersji 9.0 następowała stała poprawa wydajności.

Porównując początkową i najnowszą wersję wydajność środowiska wzrosła więcej niż 4 razy. Skok wydajności widoczny jest po przejściu na wersję 5.0. Średni czas wykonania zmniejszył się z 723 ms do 407 ms. Od wersji 6.0 do wersji 9.0 czas realizacji zadania stopniowo się zmniejszał.

Tabela 2. Wyniki testu operacje na kolekcjach

		Minimun	Średnia	Maksimum
Czas wykonania [ms]	Android 4.1	1206	1251	1366
	Android 4.2	1194	1246,64	1312
	Android 4.3	802	853,96	951
	Android 4.4	660	723,16	965
	Android 5.0	372	407,08	488
	Android 6.0	440	491,88	555
	Android 7.0	320	369,76	491
	Android 8.0	240	330,04	423
	Android 9.0	215	260,88	308



Rys. 6. Wykres wartości minimalnej, medialny oraz maksymalnej dla testu operacje na kolekcjach

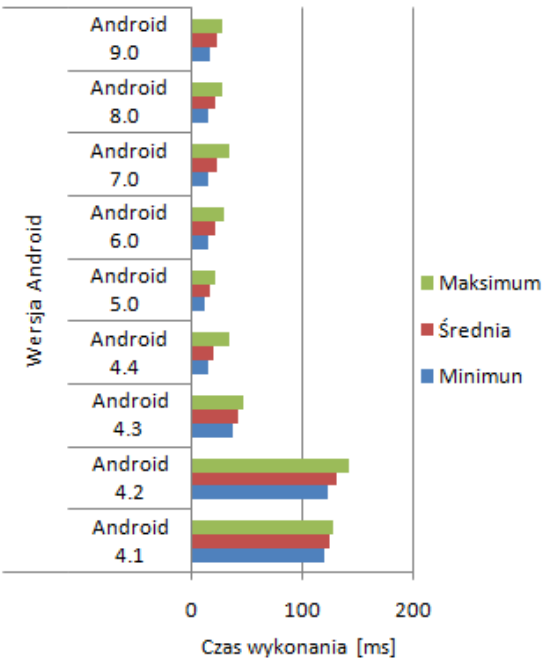
4.3 Sortowanie tablicy liczb całkowitych

Metoda badawcza sortuje tablicę elementów typu long, która ma w sobie 10 000 elementów. Typ danych long przechowuje liczbę, którą można umieścić w 64 bitach. Zakres liczby jest od -9223372036854775808 do 9223372036854775807. Wszystkie elementy dla tablicy zostały wygenerowane losowo w zakresie od 0 do 9223372036854775807.

Najbardziej zauważalna różnica jest między wynikami wykonania testu w wersji 4.2 i 4.3. Jednak w przeciwieństwie do poprzednich wyników już nie ma stopniowego wzrostu wydajności w każdej kolejnej wersji. Z wyników należy, że najszybszy emulator jest działający na wersji na Androida 5.0. Przyczyną tego wyniku może być dodanie w systemie Android wsparcia dla 64-bitowej architektury w wersji 5.0. Szczegółowe wyniki testu przedstawiono w tabeli 3 i na rysunku 7.

Tabela 3. Wyniki testu operacji na tablicy elementów long

		Minimun	Średnia	Maksimum
Czas wykonania [ms]	Android 4.1	120	123,8	128
	Android 4.2	123	130,04	142
	Android 4.3	38	42,04	48
	Android 4.4	16	20,6	35
	Android 5.0	13	17,2	22
	Android 6.0	16	22,12	30
	Android 7.0	16	24,12	34
	Android 8.0	16	21,32	28
	Android 9.0	18	23,56	28

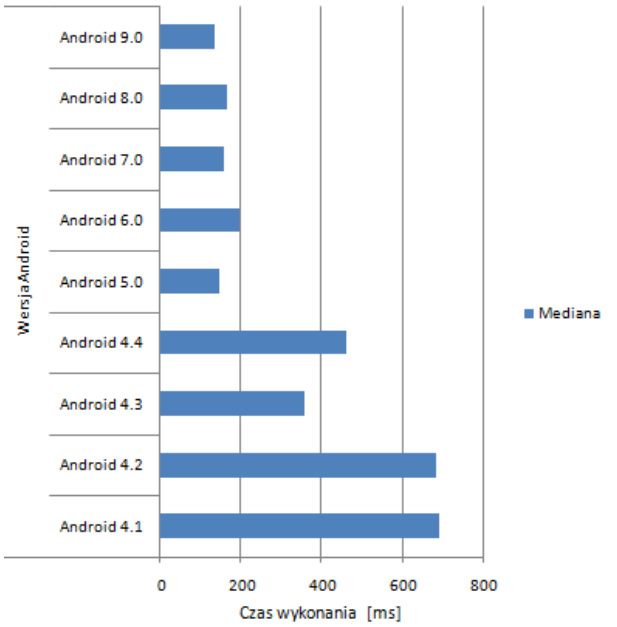


Rys. 7. Wykres wartości minimalnej, medialny oraz maksymalnej dla testu sortowanie tablicy liczb całkowitych

5. Wnioski

Na podstawie uzyskanych wyników można zauważyć, że w trakcie testowania różnych wersji Android była zauważalna ogólna tendencja, która widoczna na rysunku 8. Mediana pokazuje średni czasu wykonania trzech przypadków testowych w każdej wersji Androida. Z tego wynika, że najszybszymi wersjami były 5.0 i 9.0.

Warto zauważyć, że w piątej wersji system Android zaczął używać środowiska ART jako środowiska domyślnego. Najwolniejsze wersje są 4.1, 4.2 i 4.4. Główną przyczyną może być to, że oni działają w starym środowisku Dalvik.



Rys. 8. Wykres wartości medialny dla wszystkich operacji

Największa różnica w wydajności istnieje pomiędzy wersjami 4.4 i 5.0. Jest to związane z następującymi czynnikami: przejście z wersji testowej ART na produkcyjną, pełnej rezygnacji z maszyny Dalvik, wsparcia dla 64-bitowej architektury. Kolejne wersje nie przyniosły znaczącej zmiany.

Badania pokazało, że wydajność środowiska uruchomieniowego Android nie rosła w każdej nowej wersji w przygotowanych scenariuszach testowych. To może być związane z konfiguracją emulatora i go stałymi charakterystykami. Ponieważ w trakcie rozwinięcia systemu Android wydajność urządzeń mobilnych rośnie w każdym roku. Na przykład telefon Nexus 4 z wersją Androida 4.1 pokazuje wynik o ilości 27 300 punktów w teście wydajności Antutu, podczas gdy Pixel 3XL z wersją 9.0 dostaje 275 000 punktów [15]. Dlatego system Android nie musi mieć więcej wydajności w każdej nowej wersji, jednak jak widać z wyników ona znacznie wzrosła od wersji systemu 4.1 do 9.0 .

Literatura

[1] A. Frumusanu: A Closer Look at Android RunTime (ART) in Android L: AnandTech, 2014.

[2] C. Stewart , B. Phillips , K. Marsicano: Programowanie aplikacji dla Androida The Big Nerd Ranch Guide: Helion, 2017.

[3] https://tech-geek.ru/how-android-works, How does the operating system Android, Linux core and Android Runtime [04.2019].

[4] D. A. Heger: Mobile Devices – An Introduction to the Android Operating Environment, Design, Architecture, and Performance Implications: DHTechnologies (DHT), 2012.

[5] R. Meier: Professional Android 4 Application Development: John Wiley & Sons, 2012.

[6] https://studopedia.su/12_142429_Java-mashina-Dalvik.html, Dalvik virtual machine structure, core features, standart libraries [02.2019].

[7] B. Cheng; B.Buzbee:. A JIT Compiler for Androids Dalvik VM: Google, 2010.

[8] https://android.jlelse.eu/closer-look-at-android-runtime-dvm-vs-art-1dc5240c3924, Closer look at Android Runtime, comparing DVM vs ART, execution Java code in Android [03.2019].

[9] https://wikipedia.org/wiki/Android_Runtime, Android Runtime structure, advantages and disadvantages, compilators [05.2019].

[10] https://xakep.ru/2018/01/10/android-5-core-techs, The five pillars of Android, virtual machine, Google services, Linux core and runtime [03.2019].

[11] https://software.intel.com/ru-ru/android/articles/64-bit-android-and -android-run-time, 64-bit versions of Android and Android runtime, development and support 64-bit processors for Android [03.2019].

[12] https://developer.android.com/about/versions/nougat/android-7.0?hl=ru, Android N for developers, new features, changing JIT compilator for perfomance [03.2019].

[13] https://android-developers.googleblog.com/2019/04/improving-app-performance-with-art.html, Improving app performance with ART optimizing profiles in the cloud, instruction for implementing [04.2019].

[14] https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.1, Chapter describes the Java Virtual Machine class file format, limits for name pool and values [03.2019].

[15] https://www.kimovil.ru, Performance tests for smartphones from old versions to current.