# PERBANDINGAN PERFORMA BAHASA PEMROGRAMAN JAVA DAN KOTLIN PADA ANDROID APPS

**[1]Nikodemus Galih Candra Wicaksono, [2]Hironimus Leong**
[1,2]Program Studi Teknik Informatika Fakultas Ilmu Komputer,
Universitas Katolik Soegijapranata
[2]marlon.leong@unika.ac.id

## ABSTRAK

*Java merupakan salah satu bahasa pemrograman terpopuler. Bahasa ini memiliki kelebihan yang sering disebutkan yaitu write once, run anywhere (WORA). Java merupakan bahasa yang digunakan dalam pembuatan aplikasi android. Selain itu terdapat juga bahasa kotlin yang baru saja pada tahun 2017 diperkenalkan sebagai bahasa resmi pembuatan aplikasi android. Namun dalam performa dari masing-masing bahasa manakah yang lebih baik. Karena itu saya melakukan penelitian ini untuk mencari performa dari masing-masing bahasa yang digunakan untuk pembuatan aplikasi android.*

**Keywords:** Java, Kotlin, Android Apps

## PENDAHULUAN

Saat ini hampir semua orang memiliki telepon genggam atau *mobile device*. Apalagi di masa pandemi Covid-19, membuat penggunaan *mobile device* semakin bertambah dikarenakan semua dilakukan secara *online* dan juga karena lebih murah dibandingkan laptop maupun komputer. *Mobile device* memiliki berapa sistem operasi yang berbeda-beda. Sistem operasi yang paling banyak digunakan di dunia adalah Android dan iOS [1]. Menurut data dari statista.com [2], pembelian mobile pada Juni 2021 dengan sistem operasi Android sebesar 72,84%, iOS sebesar 26,34% dan 0,82% untuk sistem operasi lainnya.

Android sendiri merupakan sistem operasi (OS) perangkat yang bisa digunakan di beberapa perangkat seperti *mobile,* televisi, jam dan bahkan mobil. OS yang bisa digunakan di berbagai perangkat dan juga digunakan oleh brand besar membuat penggunanya lebih banyak dibandingkan OS lainnya. Salah satu kelebihan lainnya adalah lebih mudahnya dalam pembuatan aplikasi untuk OS android. Dalam pembuatan aplikasi android saat ini dapat menggunakan bahasa pemrograman Java ataupun Kotlin.

Java merupakan salah satu bahasa pemrograman popular yang bisa digunakan berbagai hal. Contohnya seperti aplikasi android *mobile*, web, desktop, server dan masih banyak lainnya [3]. Bahasa yang dibuat sejak tahun 1995 ini merupakan salah satu bahasa yang digunakan dalam pembuatan aplikasi android. Begitu pula dengan bahasa Kotlin. Kedua bahasa tersebut didesain berjalan di *Java Virtual Machine*. Kotlin yang diperkenalkan sebagai bahasa resmi untuk pembuatan aplikasi android pada 2017 oleh Google saat ini sudah banyak digunakan [4].

Karena mulai bertambahnya penggunaan bahasa Kotlin dalam pembuatan aplikasi android, performa dalam bahasa pemrograman menjadi hal penting dalam pembuatan aplikasi atau program [5]. Dengan membandingkan kedua bahasa tersebut diharapkan dapat diketahui bahasa manakah yang lebih baik dalam performanya pada aplikasi android.

## TINJAUAN PUSTAKA

Dari penelitian yang dilakukan Cheon dan Torre [6], mereka meneliti dampak bahasa Java terhadap memori aplikasi android. Dengan menggunakan fitur yang ada pada bahasa Java seperti *lambda expressions, Stream API, for-each statement* dan *iterators*. Ditemukan dengan mengeksekusi beberapa fitur tersebut, aplikasi membutuhkan memori yang signifikan tergantung jumlahnya memori. Selain itu juga dapat mempengaruhi daya beterai karena terlalu panas [7]. Selain itu struktur data dan varibel yang digunakan mempengaruhi efisiensi memori aplikasi android.

Dari penelitian performa aplikasi android [8], dibandingkan dengan penggunaan 2 bahasa pemrograman yang berbeda yaitu C Native dan Java. Terlihat bahwa performa bahasa C Native lebih cepat dalam pemrosesan data dengan tipe *integer, floating-point* dan akses memorinya. Tetapi dalam pemrosesan data dengan tipe data *string* bahasa Java lebih efisien dibandingkan bahasa C Native.

Dari kedua jurnal tersebut, saya mencoba membandingkan performa aplikasi android saat menggunakan bahasa Java dan bahasa Kotlin seperti jurnal ke [8]. Berdasarkan penelitian Cheon dan Torre [6], saya akan membandingkan dengan jenis dan jumlah variabel yang sama dalam kedua bahasa tersebut. Karena variabel juga mempengaruhi performa dari memori aplikasi android.

## METODE PENELITIAN

Dalam pembuatan program pada penelitian ini, hanya menggunakan code sederhana yaitu iterasi while. Dengan iterasi sederhana tersebut dapat dilihat performa dari program yang sudah dibuat dan dijalankan pada android. Iterasi akan dijalankan dengan waktu yang ditentukan yaitu 1 detik dan 10 detik. Masing-masing waktu akan dilakukan sebanyak 5 kali supaya dapat dilihat perbedaannya. Untuk program iterasi while karena mampu menggambarkan performa dari sebuah kode [5]. Proses pengujian dilakukan dengan melakukan pemanggilan function seperti dibawah ini.

```
private void Iteration(){
        long startTime = currentTimeMillis();
        waktu = et_time.getText().toString();
        long time = Long.parseLong(waktu) * 1000;
        while ((currentTimeMillis() - startTime) < time) {
            iteration += 1;
        }
```

```
}
```

Function iterasi while untuk bahasa Java tersebut digunakan untuk menambah integer(tipe data long) untuk mengetahui jumlah integer yang dapat diproses dalam waktu yang ditentukan. Sedangkan untuk bahasa Kotlin, code yang digunakan hampir sama yaitu seperti pada function berikut ini.

```
fun Iteration(){
        var startTime = currentTimeMillis()
        waktu = et_time.text.toString()
        var time = (waktu.toLong()) * 1000
        while ((currentTimeMillis() - startTime) < time) {
            iteration +=1
        }
     }
```

Dari kedua bahasa tersebut akan menambah 1 jumlah integer pada variabel global yang diberi nama iteration. Kemudian akan ditampilkan jumlah iterasi yang dilakukan dan waktu yang ditentukan. Untuk melakukan selama waktu yang ditentukan digunakan function currentTimeMillis(). Jika waktu sudah sampai yang ditentukan iterasi while akan berhenti.

Selain melakukan penelitian untuk tipe data Integer, disini juga dilakukan dengan mencoba untuk tipe data string. Variabel yang digunakan untuk menyimpan sama seperti sebelumnya yaitu variabel global. Untuk kedua bahasa diinisialisasikan terlebih dahulu string yang yang akan digenerate. Kemudian dibuat fungsi untuk megenerate string selama waktu yang ditentukan seperti dibawah ini.

*Java:*

```
public static final String KARAKTER ="ABC DEF GHI JKL MNO PQR STU
VWX YZa bcd efg hij klm nop qrs tuv wxy z12 345 678 90";
private void randomString(){
        long startTime = currentTimeMillis();
        waktu = et_time.getText().toString();
        long time = Long.parseLong(waktu) * 1000;
        while ((currentTimeMillis() - startTime) < time) {
            result += KARAKTER.charAt(new
Random().nextInt(KARAKTER.length()));
        }
    }
```

*Kotlin:*

```
val KARAKTER = "ABCD EFGH IJKL MNOP QRST UVWX YZab cdef ghij klmn
opqr stuv wxyz 1234 5678 90"
```

```
private fun randomString(){
        val startTime = currentTimeMillis()
        waktu = et_time.text.toString()
        val time = waktu.toLong() * 1000
        while (currentTimeMillis() - startTime < time) {
            result += KARAKTER.random()
        }
    }
```

Perbedaan keduanya saat didalam generate string, adalah pada fungsi random(). Di Java fungsi random() tersebut untuk mendapatkan nilai integer acak [9], kemudian akan dicari karakter ke dari nilai acak tersebut menggunakan fungsi charAt() [9]. Sedangkan pada Kotlin, random() langsung mendapatkan karakter random yang sudah disediakan yaitu dari variabel KARAKTER [10].

Selain mencari perbandingan dari tipe data yaitu integer dan string, disini juga dihitung seberapa lama waktu yang dibutuhkan untuk melakukan semua proses yang dilakukan. Waktu yang dihitung adalah waktu nano. Untuk menghitungnya dengan cara seperti dibawah ini.

*Java:*

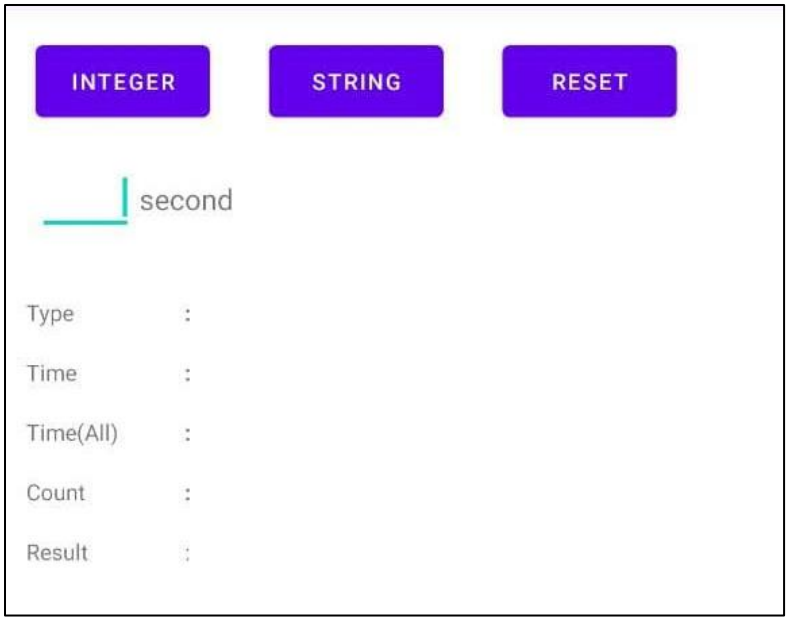
```
long startTime = nanoTime();
memanggil function (Iteration/randomString)
long endTime = nanoTime();
long time = endTime-startTime;
```

*Kotlin:*

```
var time = measureNanoTime {
memanggil function (Iteration/randomString)
}
```

Untuk bahasa Java diperlukan inisialisasi untuk mendapatkan waktu memulai dengan function nanoTime() [9]. Kemudian setelah program untuk melakukan iterasi while atau mengenerate string, memanggil lagi fungsi nanoTime() dikurang waktu memulai untuk mendapatkan waktu melakukan program. Sedangkan pada bahasa Kotlin cukup memanggil function measureNanoTime dan melakukan program didalamnya.

Selain itu dengan menggunakan tools dari Android Studio IDE, dapat diukur penggunaan dari cpu dan juga memory aplikasi saat berjalan. Dengan tools yang disediakan tersebut kita akan membandingkannya dari kedua bahasa dan masing-masing tipe data. Hasil tampilan dari kedua aplikasi akan sama, yaitu seperti pada gambar dibawa ini.



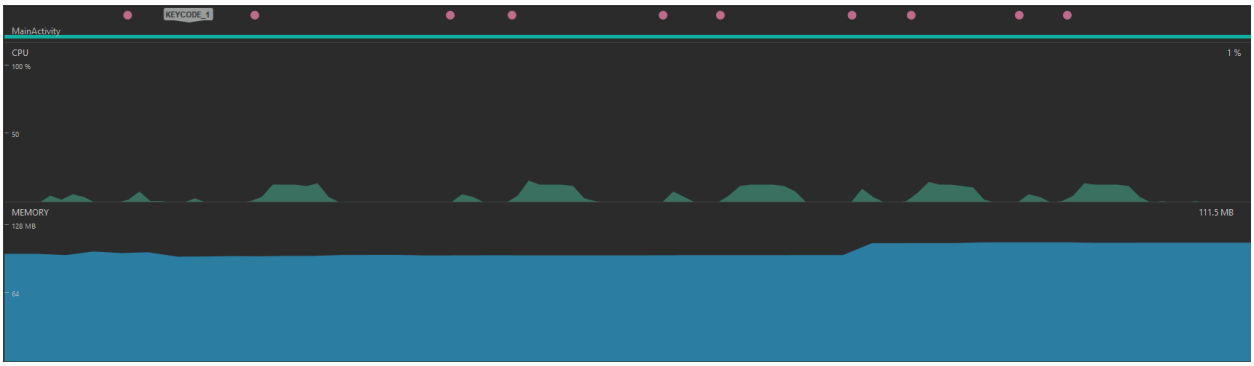

**Gambar 1.** Tampilan aplikasi android

## HASIL DAN PEMBAHASAN

Dengan melakukan eksperimen langsung aplikasi pada android di smartphone Poco X3 NFC / M2007J20CG. CPU Octa-core dengan chipset Qualcomm SM7150-AC Snapdragon 732G (8nm) dan 8GB RAM serta versi android 11. Dengan eksperimen terhadap tipe data integer dan String dengan 2 jenis waktu yaitu 1 detik dan 10 detik didapatkan hasil seperti dibawah ini.

### *Java*

*Integer Java 1 detik*




**Gambar 2.** CPU dan Memory Integer Java 1 detik

Berdasarkan kinerja aplikasi menggunakan bahasa dapat dilihat penggunaan CPU (atas/hijau) dan memory(bawah/biru). CPU bekerja hanya jika terdapat proses yang berjalan. Proses yang dimaksud adalah menekan tombol integer, memanggil function Iteration() sampai menampilkan hasilnya di layar aplikasi. Sedangkan untuk penggunaan memori, meskipun sedang

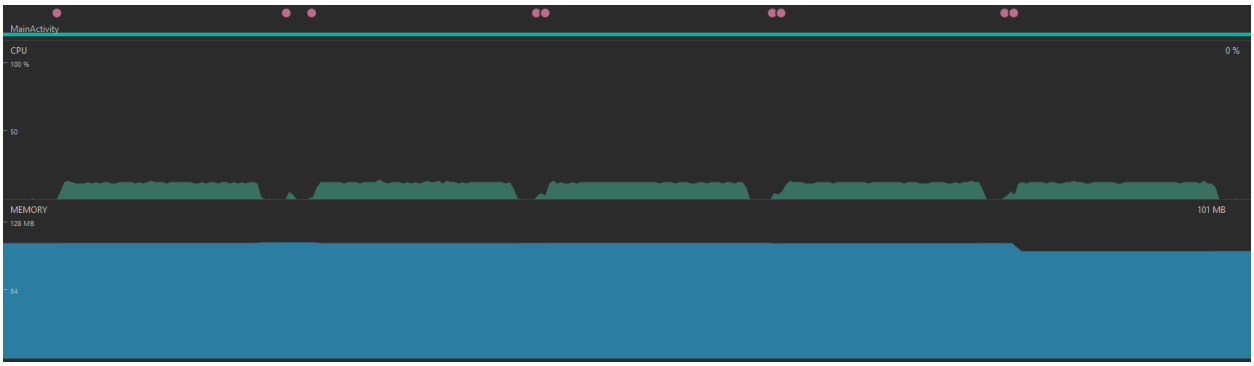
tidak menjalankan proses tetapi memori tetap bekerja. Untuk lebih rinci, hasil dari penggunaan seperti tabel dibawah ini.

**Tabel 1.** Testing Integer Java 1 detik

| Test ke | Waktu (nanodetik) | Total | CPU (%) | | Memory (MB) | |
|---|---|---|---|---|---|---|
| | | | ***Min*** | ***Max*** | ***Min*** | ***Max*** |
| 1 | 1.002.943.385 | 7.299.992 | 0 | 14 | 98,8 | 99,9 |
| 2 | 1.002.159.010 | 7.237.711 | 1 | 16 | 99,6 | 99,7 |
| 3 | 1.001.492.916 | 7.365.395 | 0 | 13 | 99,8 | 99,9 |
| 4 | 1.008.446.406 | 7.331.796 | 1 | 15 | 111,1 | 111,8 |
| 5 | 1.002.070.573 | 7.314.987 | 1 | 13 | 111,4 | 111,9 |
| Rata-rata | 1.003.760.429 | 7.308.723 | 0,50 | 14,50 | 102,33 | 102,83 |

Untuk waktu pemrosesan program integer selama 1 detik, yaitu mulai dari menekan tombol sampai menampilkan hasil memiliki rata-rata 1,003 detik dengan total iterasi while sebanyak 7,308 juta kali. Sedangkan dari kinerja cpu dari rentang 0% sampai 16%. Dan untuk penggunaan memori antara 98,8 MB sampai 111,9 MB.

### *Integer Java 10 detik*




**Gambar 3.** CPU dan Memory Integer Java 10 detik

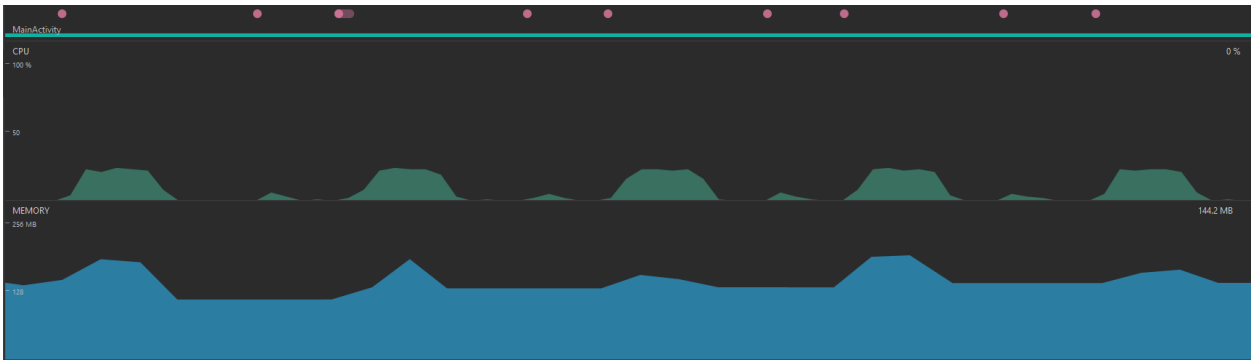
Pada Integer Java selama 10 detik sama seperti dengan 1 detik. Hanya saja rentang waktunya lebih lama. Untuk detailnya seperti pada tabel dibawah ini.

**Tabel 2.** Testing Integer Java 10 detik

| Test ke | Waktu (nanodetik) | Total | CPU (%) | | Memory (MB) | |
|---|---|---|---|---|---|---|
| | | | ***Min*** | ***Max*** | ***Min*** | ***Max*** |
| 1 | 10.001.816.975 | 73.496.941 | 2 | 13 | 108,7 | 109,2 |
| 2 | 10.001.910.309 | 73.504.949 | 2 | 14 | 108,7 | 109,4 |
| 3 | 10.001.728.434 | 73.491.112 | 3 | 13 | 108,8 | 108,9 |
| 4 | 10.001.947.288 | 73.524.173 | 4 | 13 | 108,5 | 108,6 |
| 5 | 10.002.106.871 | 73.572.208 | 4 | 14 | 101 | 106,8 |
| Rata-rata | 10.001.901.975 | 73.517.877 | 3,00 | 13,40 | 107,14 | 108,58 |

Pada testing ini, dibutuhkan tambahan sekitar 0,0019 detik untuk menyelesaikan semua proses dari waktu yang ditentukan yaitu 10 detik. Karena selama 10 detik, maka total iterasinya 10 kali lipat dari 1 detik yaitu 73,5 juta iterasi. Untuk penggunaan CPUnya antara 2%-14% sedangkan memorinya hampir stabil di sekitar 108MB kecuali saat tes ke 5 nya.

*String Java 1 detik*




**Gambar 4.** CPU dan Memory String Java 1 detik

Pada testing string, terlihat bahwa grafiknya berbeda saat testing integer. Penggunaan CPU lebih besar dibandingkan integer. Dan juga penggunaan memori terdapat kenaikan signifikan saat terdapat proses. Tidak seperti integer yang tetap stabil saat proses maupun tidak. Untuk detailnya seperti pada tabel dibawah.

**Tabel 3.** Testing String Java 1 detik

| Test ke | Waktu (nanodetik) | Total (karakter) | CPU (%) | | Memory (MB) | |
|---|---|---|---|---|---|---|
| | | | *Min* | *Max* | *Min* | *Max* |
| 1 | 1.062.162.083 | 19.935 | 4 | 24 | 115,2 | 188,7 |
| 2 | 1.048.943.229 | 20.144 | 3 | 24 | 121,4 | 186,6 |
| 3 | 1.049.388.854 | 20.073 | 2 | 23 | 135,6 | 159,2 |
| 4 | 1.047.785.833 | 20.079 | 4 | 24 | 152,8 | 196 |
| 5 | 1.065.022.760 | 20.180 | 5 | 23 | 144,4 | 169 |
| Rata-rata | 1.054.660.552 | 20.082 | 3,60 | 23,60 | 133,88 | 179,90 |

Waktu yang dibutuhkan sampai proses selesai, membutuhkan waktu tambahan 0,054 detik. Waktu ini lebih lama dibandingkan memproses integer. Total dari string yang diproses selama 1 detik sekitar 20 ribu karakter. Untuk batas atasnya berbeda cukup jauh dari integer, yaitu pada kisaran 23,60%. Begitu juga dengan penggunaan memori yang membutuhkan lebih banyak memori untuk memproses tipe data string.

*String Java 10 detik*


**Gambar 5.** CPU dan Memory String Java 10 detik

Pada proses string selama 10 detik ini, terlihat berbeda dalam penggunaan memori. Dalam penggunaan saat menjalankan proses, penggunaan memori naik turun. Kemudian cenderung stabil saat tidak menjalankan proses.

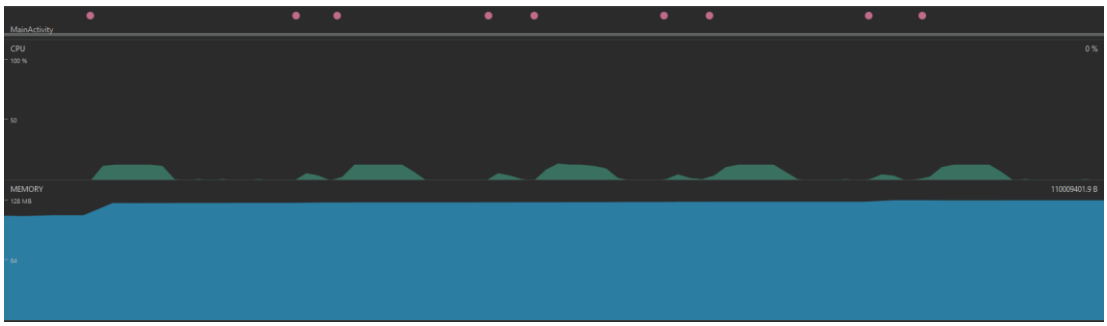
**Tabel 4.** Testing String Java 10 detik

| Test ke | Waktu (nanodetik) | Total (karakter) | CPU (%) | | Memory (MB) | |
|---|---|---|---|---|---|---|
| | | | *Min* | *Max* | *Min* | *Max* |
| 1 | 10.336.234.683 | 69.965 | 6 | 26 | 118,7 | 265,3 |
| 2 | 10.324.212.392 | 70.551 | 3 | 26 | 121,9 | 265,4 |
| 3 | 10.326.921.090 | 70.615 | 2 | 25 | 152,3 | 224,6 |
| 4 | 10.333.519.267 | 70.272 | 9 | 27 | 146,3 | 224,9 |
| 5 | 10.325.802.079 | 70.324 | 8 | 26 | 128,3 | 292,9 |
| Rata-rata | 10.329.337.902 | 70.345 | 5,60 | 26,00 | 133,50 | 254,62 |

Untuk tambahan waktunya disini dibutuhkan 0,329 detik. Namun untuk jumlah karakter yang diproses sekitar 3,5 kali lipatnya saja dari testing 1 detik. Itu berarti semakin lama waktu, semakin berkurang jumlah string yang diproses. Untuk CPU kurang lebih hampir sama dengan yang 1 detik, tetapi untuk memorinya dapat mencapai 292,9 MB.

### *Kotlin*

*Integer Kotlin 1 detik*




**Gambar 6.** CPU dan Memory Integer Kotlin 1 detik

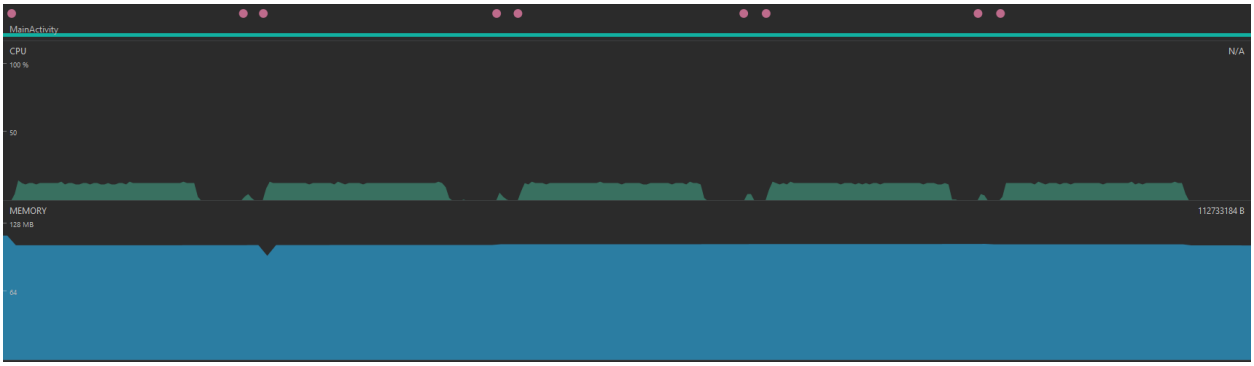
Untuk CPU dan memori, bahasa kotlin kurang lebih sama seperti java. CPU hanya bekerja saat ada proses dan memorinya stabil.

**Tabel 5.** Testing Integer Kotlin 1 detik

| Test ke | Waktu (nanodetik) | Total | CPU (%) | | Memory (MB) | |
|---|---|---|---|---|---|---|
| | | | ***Min*** | ***Max*** | ***Min*** | ***Max*** |
| 1 | 1.007.789.895 | 7.160.388 | 1 | 13 | 94,9 | 102,6 |
| 2 | 1.005.816.406 | 7.321.040 | 3 | 13 | 103 | 103,1 |
| 3 | 1.005.073.020 | 7.310.819 | 2 | 14 | 103,3 | 103,4 |
| 4 | 1.005.487.609 | 7.304.624 | 1 | 13 | 103,6 | 103,7 |
| 5 | 1.005.543.905 | 7.386.515 | 1 | 13 | 104,9 | 104,9 |
| Rata-rata | 1.005.942.167 | 7.296.677 | 1,60 | 13,20 | 101,94 | 103,54 |

Untuk waktu semua proses, Kotlin lebih lama dibandingkan menggunakan java. Dimana java perlu tambahan 0,0037 detik sedangkan kotlin 0,0059 detik. Untuk jumlah iterasinya dapat di katakan sama untuk kedua bahasa. Begitu juga untuk CPU dan memori

*Integer Kotlin 10 detik*




**Gambar 7.** CPU dan Memory Integer Kotlin 10 detik

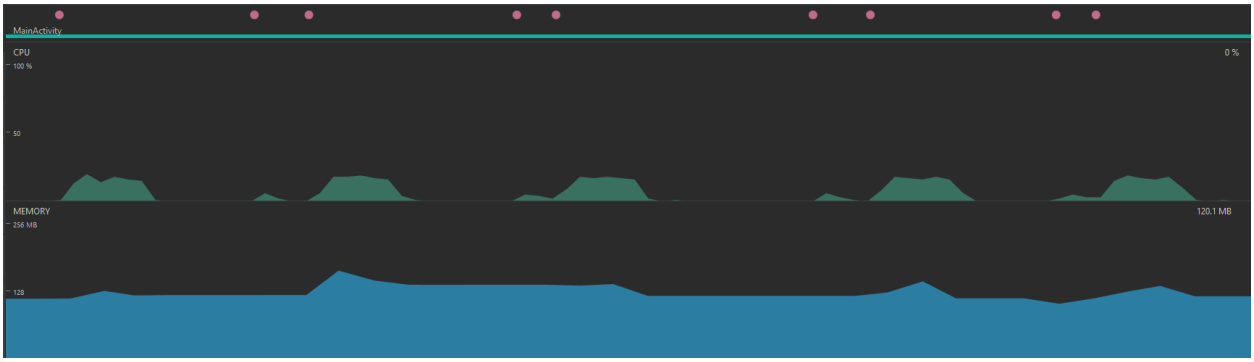
Untuk integer kotlin 10 detik, juga sama seperti java. Namun disini memori terlihat lebih stabil. Hanya terdapat sedikit perubahan memori kemudian kembali stabil kembali.

**Tabel 6.** Testing Integer Kotlin 10 detik

| Test ke | Waktu (nanodetik) | Total | CPU (%) | | Memory (MB) | |
|---|---|---|---|---|---|---|
| | | | ***Min*** | ***Max*** | ***Min*** | ***Max*** |
| 1 | 10.007.478.590 | 72.932.489 | 3 | 14 | 107,7 | 112 |
| 2 | 10.004.526.715 | 73.805.665 | 2 | 14 | 98,1 | 107,9 |
| 3 | 10.004.489.892 | 73.849.689 | 2 | 14 | 108,6 | 108,7 |
| 4 | 10.005.471.506 | 73.841.621 | 1 | 14 | 108,8 | 108,8 |
| 5 | 10.003.665.309 | 73.946.750 | 3 | 14 | 108,4 | 108,5 |
| Rata-rata | 10.005.126.402 | 73.675.243 | 2,20 | 14,00 | 106,32 | 109,18 |

Untuk Integer 10 detik menggunakan bahasa kotlin juga kurang lebih sama seperti java. Waktu yang dibutuhkan lebih lama kotlin, sedangkan total iterasinya dapat dikatakan sama karena 73 juta juga. Untuk CPU dan memori, tidak terlalu berbeda signifikan dari kedua bahasa.

*String Kotlin 1 detik*




**Gambar 8.** CPU dan Memory String Kotlin 1 detik

Berbeda dengan java, memori pada bahasa kotlin peningkatan memori saat pemrosesan tidak terlalu lama di puncak penggunaan memori. Bahasa kotlin langsung turun setelah puncak tertinggi penggunaan memori sedanakan pada java cenderung stabil di pucak penggunaan memori beberapa saat baru akan turun.

**Tabel 7.** Testing String Kotlin 1 detik

| Test ke | Waktu (nanodetik) | Total (karakter) | CPU (%) | | Memory (MB) | |
|---|---|---|---|---|---|---|
| | | | ***Min*** | ***Max*** | ***Min*** | ***Max*** |
| 1 | 1.050.518.584 | 11.614 | 1 | 20 | 115,7 | 130 |
| 2 | 1.040.883.125 | 14.723 | 1 | 19 | 124,3 | 168,1 |
| 3 | 1.039.840.885 | 14.525 | 2 | 18 | 141,3 | 120,7 |
| 4 | 1.060.877.395 | 14.569 | 0 | 18 | 116 | 147,6 |
| 5 | 1.055.501.822 | 14.922 | 1 | 19 | 118,4 | 139,3 |
| Rata-rata | 1.049.524.362 | 14.071 | 1,00 | 18,80 | 123,14 | 141,14 |

Waktu pemrosesan kotlin saat memproses string lebih cepat dibandingkan java. Namun jumlah karakter yang dapat diproses terbilang cukup jauh. Dibandingkan java yang mencapai 20 ribu, kotlin yang di kisaran 14 ribu karakter saja setiap 1 detik. Namun dalam hal penggunaan CPU dan memori kotlin lebih baik dibandingkan java karena lebih sedikit. Perbandingan CPU dan memori tersebut berbeda cukup banyak.

*String Kotlin 10 detik*


**Gambar 9.** CPU dan Memory String Kotlin 10 detik

Untuk grafik dari proses string pada kotlin selama 10 detik, sama seperti pada java. Yaitu CPU akan bekerja saat ada proses dan penggunaan memori akan naik turun selama proses berjalan.

**Tabel 8.** Testing String Kotlin 10 detik

| Test ke | Waktu (nanodetik) | Total (karakter) | CPU (%) | | Memory (MB) | |
|---|---|---|---|---|---|---|
| | | | ***Min*** | ***Max*** | ***Min*** | ***Max*** |
| 1 | 10.229.452.027 | 52.394 | 3 | 21 | 120,6 | 190,6 |
| 2 | 10.236.110.309 | 53.250 | 2 | 21 | 124,4 | 214,3 |
| 3 | 10.255.252.184 | 53.117 | 1 | 20 | 126,4 | 190,2 |
| 4 | 10.239.196.767 | 52.773 | 5 | 19 | 123,2 | 195,8 |
| 5 | 10.271.396.715 | 53.199 | 5 | 21 | 104,2 | 179 |
| Rata-rata | 10.246.281.600 | 52.947 | 3,20 | 20,40 | 119,76 | 193,98 |

Untuk waktu proses keseluruhan, kotlin masih lebih cepat dibandingkan java meskipun dengan waktu yang berbeda. Jumlah karakter juga lebih sedikit dibandingkan java. Tetapi jumlah pada string kotlin 10 detik adalah jumlah 3,78 kali dari 1 detik nya. Sedangkan 10 detik string java hanya 3,5 kali. Itu berarti semakin lama waktunya, kotlin yang lebih baik mesikpun jumlahnya masih jauh dibawahnya bahasa java. Dalam penggunaan CPU dan memori juga meskipun waktunya lebih lama tetap lebih baik kotlin

## KESIMPULAN

Dari penelitian yang telah dilakukan dengan melakukan testing performa bahasa kotlin dan java. Yaitu dengan mengimplementasikan code sederhana dalam pembuatan aplikasi dengan tipe data yang berbeda. Dapat disimpulkan bahwa performa java dan kotlin tergantung dari performa yang diukur.

Saat pemrosesan integer, java lebih unggul dalam waktu keseluruhan. Selain itu, java dan kotlin kurang lebih memiliki performa yang sama. Namun dalam pemrosesan string, terlihat perbedaan dari kedua bahasa. Java lebih unggul dalam jumlah string yang dapat diproses, bisa

dikatakan jauh di atas kotlin. Tetapi dalam waktu keseluruhan, CPU dan memori lebih unggul kotlin saat memproses string.

Saran untuk penelitian selanjutnya dapat dilakukan dengan membandingkan selain dari jumlah iterasi yang dapat dilakukan dari kedua bahasa. Dikarenakan sedikitnya karya tulis tentang performa dari android apps, diharapkan dengan adanya penelitian ini dapat memunculkan karya tulis lainnya yang lebih berbobot.

## DAFTAR PUSTAKA


[1] R. Sikder, M. S. Khan, M. S. Hossain, and W. Z. Khan, "A survey on android security: development and deployment hindrance and best practices," *TELKOMNIKA*, vol. 18, no. 1, p. 485, Feb. 2020, doi: 10.12928/telkomnika.v18i1.13288.

[2] "Mobile Operating System Market Share Worldwide," *StatCounter Global Stats*. https://gs.statcounter.com/os-market-share/mobile/worldwide/2021 (accessed Apr. 18, 2023).

[3] J. Chan, *Learn Java in One Day and Learn It Well*. CreateSpace Independent Publishing Platform, 2016.

[4] J. Horton, *Android Programming with Kotlin for Beginners: Build Android apps starting from zero programming experience with the new Kotlin programming language*. Packt Publishing Ltd, 2019.

[5] H. Farooq, P. Rahnamayiezekavat, and S. Moon, "While Loop Algorithm to Enhance the Efficiency of Work Sampling Method in Performance Measurement," *ISARC Proceedings*, pp. 6–13, Jul. 2017, Accessed: Apr. 18, 2019. [Online]. Available: https://www.iaarc.org/publications/2017_proceedings_of_the_34rd_isarc/while_loop_algorithm_to_enhance_the_efficiency_of_work_sampling_method_in_performance_measurement.html

[6] Y. Cheon and A. E. D. L. Torre, "Impacts of Java Language Features on the Memory Performances of Android Apps," 2017. Accessed: Apr. 18, 2020. [Online]. Available: https://www.semanticscholar.org/paper/Impacts-of-Java-Language-Features-on-the-Memory-of-Cheon-Torre/fc71d6e7b0f13b22d0c1fd4691d0cd9d8a25cff4

[7] J. Parkkila and J. Porras, "Improving battery life and performance of mobile devices with cyber foraging," in *2011 IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communications*, Sep. 2011, pp. 91–95. doi: 10.1109/PIMRC.2011.6140102.

[8] J. K. Lee and J. Y. Lee, "Android programming techniques for improving performance," in *2011 3rd International Conference on Awareness Science and Technology (iCAST)*, Sep. 2011, pp. 386–389. doi: 10.1109/ICAwST.2011.6163105.

[9] "Java Cookbook, 4th Edition [Book]." https://www.oreilly.com/library/view/java-cookbook-4th/9781492072577/ (accessed Apr. 18, 2020).

[10] "Kotlin Cookbook [Book]." https://www.oreilly.com/library/view/kotlin-cookbook/9781492046660/ (accessed Apr. 18, 2020).