

A. Informasi & Ringkasan

1. Identitas

NIM	11S23043
Nama	Grace Evelin Siallagan
Kelas	12 IF 2
Judul Praktikum	Analisis Algoritma
Video Presentasi	https://youtu.be/H2omRoqNMTk?si=GQ0gRSwaWnEk-k_E

2. Capaian & Ringkasan Pemahaman

Hasil Capaian setelah mengikuti praktikum:

- A (Penyelesaian): Selesai (1), Tidak Selesai (0).
- B (Pemahaman): Tidak Paham (1), Kurang Paham (2), Cukup Paham (3), Paham (4), Sangat Paham (5).

No	Indikator	A	B
1	Analisis Algoritma	1	3
2	Pointer & Latihan	1	3
3	Tantangan	1	3

B. Penilaian Observasi

Setiap Mahasiswa yang mengikuti kegiatan praktikum perlu melakukan penilaian terhadap diri sendiri dan rekan kerja berdasarkan [[Indikator Penilaian Observasi Praktikum](#)].

1. Penilaian Terhadap Diri Sendiri

No	Indikator	Skala (1 - 5)
1	Kepatuhan Terhadap Prosedur	3
2	Kemampuan Penyelesaian Tugas	3
3	Kerja Sama Tim	4
4	Inisiatif dan Problem Solving	3
5	Kedisiplinan dan Etika Kerja	4

C. Aktivitas Praktikum

1. Analisis Algoritma

a) Membandingkan Kode Terbaik

Big O adalah cara untuk membandingkan kode satu dengan kode kedua secara matematis tentang seberapa efisien keduanya berjalan. Kode yang dibandingkan haruslah menyelesaikan permasalahan yang sama. Misal kamu mempunyai dua kode program untuk menyelesaikan permasalahan dalam melakukan perkalian matriks dan menggunakan stopwatch untuk mengukur waktu berjalan program dan mendapati kode pertama berjalan selama 15 detik dan kode kedua selama 1 menit. Berdasarkan hal ini, kamu akan mengatakan bahwa kode pertama lebih baik dari kode kedua. Ini disebut sebagai kompleksitas waktu. Kompleksitas waktu (Time Complexity) tidak diukur dalam waktu karena jika anda menjalankan kode yang sama pada dua komputer yang berbeda, kode akan lebih cepat berjalan pada komputer dengan performa yang lebih baik. Ini berarti bukan kode menjadi lebih baik tetapi komputer menjadi lebih baik. Jadi, ini diukur dalam jumlah operasi yang diperlukan program untuk menyelesaikan suatu permasalahan. Selain kompleksitas waktu kita juga perlu mengukur kompleksitas ruang. Misal kode pertama berjalan 15 detik tetapi membutuhkan banyak memori saat berjalan dan kode kedua berjalan 1 menit tetapi membutuhkan memori yang lebih sedikit sehingga hal ini perlu menjadi pertimbangan. Jika ruang memori adalah prioritas utama maka tidak masalah untuk memilih kode kedua, tetapi jika kecepatan adalah prioritas utama maka kode pertama akan menjadi pilihan terbaik.

b) Kompleksitas Ruang & Waktu

Ketika mempelajari kompleksitas ruang & waktu, kamu akan melihat tiga huruf Yunani: Ω (Omega), Θ (Theta) dan o (Omicron). Untuk menunjukkan bagaimana kita akan menggunakan huruf Yunani ini sebagai contoh terdapat array berikut:

1	2	3	4	5	6	7	8	9
Ω				Θ				o

Misalkan jika ingin mencari sebuah data dari array di atas maka kasus terbaiknya adalah satu (1) dan ini disimbolkan dengan Ω (Omega). Kasus rata-rata adalah lima (5) disimbolkan dengan Θ (Theta). Kasus terburuk adalah sembilan (9) disimbolkan dengan o (Omicron). Jadi dalam hal ini Ω (Omega) adalah kasus terbaik, Θ (Theta) adalah kasus rata-rata dan o (Omicron) adalah kasus terburuk. Untuk kedepannya kita akan fokus pada kasus terburuk dengan menggunakan notasi Big O.

c) Big O Notation

- Big O(n)

```
1 public class Main {
2     public static void printItems(int n){
3         for(int i = 0; i < n; i++){
4             System.out.println(i);
5         }
6     }
7
8     public static void main(String[] args) {
9         printItems(n:10);
10    }
11 }
```

PS D:\Semester 4 geres\Alstrudat\ifs23043-alstrudat-p2> javac Main.java
PS D:\Semester 4 geres\Alstrudat\ifs23043-alstrudat-p2> java Main
0
1
2
3
4
5
6
7
8
9

Pada kode program di atas terdapat method dengan nama `printItems()` yang menerima inputan bilangan bulat `n`. Method tersebut dipanggil di line 9 yang akan menggantikan nilai `n` menjadi 10. Karena jumlah `n = 10` dan jumlah operasi yang dilakukan dengan menggunakan perulangan `for` pada line 3-5 juga dilakukan sebanyak 10 maka kompleksitas waktunya adalah $O(n)$.

- Big O: Drop Constants

```
1 public static void printItems(int n){
2     for(int i = 0; i < n; i++){
3         System.out.println(i);
4     }
5     for(int i = 0; i < n; i++){
6         System.out.println(i);
7     }
8 }
9
10
```

Pada kode program di atas terdapat perulangan sebanyak n dan berjalan sebanyak 2 kali. Jadi dalam kasus ini terdapat n ditambah n operasi " $n + n = 2n$ ". Jika n bernilai 10 maka operasi yang dilakukan sebanyak 20 kali. Penulisan kompleksitas waktu pada notasi big O dilakukan dengan menghilangkan konstanta sehingga kompleksitas waktunya adalah $O(n)$.

- Big $O(n^2)$

```
2 public static void printItems(int n){
3     for(int i = 0; i < n; i++){
4         for(int j = 0; j < n; j++){
5             System.out.println(i + " " + j);
6         }
7     }
8 }
```

Kode program diatas akan melakukan operasi sebanyak $n * n = n^2$. Apabila n bernilai 10 maka akan terdapat sebanyak 100 operasi. Kompleksitas waktunya adalah $O(n^2)$.

- Big O: Drop Non-Dominants

```
2 public static void printItems(int n){
3     for(int i = 0; i < n; i++){
4         for(int j = 0; j < n; j++){
5             System.out.println(i + " " + j);
6         }
7     }
8     for(int k = 0; k < n; k++){
9         System.out.println(k);
0     }
1 }
```

Kode program diatas akan melakukan operasi sebanyak $n * n$ dan operasi sebanyak n . $n * n + n = n^2 + n$. Penulisan kompleksitas waktu pada notasi big O dilakukan dengan menghilangkan non-dominants (bukan kompleksitas terbesar) sehingga kompleksitas waktunya adalah $O(n^2)$.

- Big O(1)

```
2 public static int addItems(int n){
3     return n + n;
4 }
```

Kode program diatas akan melakukan operasi sebanyak satu kali. Kompleksitas waktu untuk kode program ini adalah $O(1)$.

```
2 public static int addItems(int n){
3     return n + n + n;
4 }
```

Kode program diatas akan melakukan operasi sebanyak dua kali. Kompleksitas waktu untuk kode program ini tidak ditulis menjadi $O(2)$ tetapi disederhanakan menjadi $O(1)$. Kompleksitas waktu ini disebut sebagai konstan dalam artian dengan bertambahnya nilai n jumlah operasi tetap konstan. Sebagai contoh jika n bernilai berapapun misal 100 ataupun 1000 operasi pada kode program di atas tetaplah sebanyak dua kali.

- Big O(log n)

1	3	5	7	10	20	30	40
0	1	2	3	4	5	6	7

Misalkan kita ingin mencari nilai 40 dari array di atas dengan kompleksitas waktu $\log n$ maka dapat dilakukan dengan tahapan berikut. [!] Hanya dapat digunakan untuk data yang telah terurut.

- 1) Bandingkan nilai tengah array maka yang mendekati berada pada array sebelah kanan pada index ke-4 dan seterusnya.

10	20	30	40
----	----	----	----

- 2) Bandingkan lagi nilai tengah array dari hasil tahap satu maka yang mendekati berada pada array sebelah kanan pada index ke-6 dan seterusnya.

30	40
----	----

- 3) Bandingkan lagi nilai tengah array dari hasil tahap dua maka nilai 40 ditemukan pada index ke-7.

Karena jumlah tahap yang dibutuhkan hanya 3 untuk menemukan nilai 40 dari array dengan jumlah data sebanyak 8 maka kompleksitas waktunya adalah $O(\log n)$. Karena nilai dari $\log_2(8)$ adalah 3 (mudahnya adalah 2 pangkat berapa yang hasilnya 8 jawabannya adalah 3 karena $2^3 = 8$). Oleh karena itu dengan kompleksitas yang lebih baik akan mengurangi waktu eksekusi yang dibutuhkan untuk menjalankan program.

- Big O: Different Terms for Inputs

```
public static void printItems(int n, int m){
    for(int i = 0; i < n; i++){
        System.out.println(i);
    }
    for(int j = 0; j < m; j++){
        System.out.println(j);
    }
}
```

Pada kode program diatas terdapat 2 input yaitu n dan m. Karena masing-masing perulangan menghasilkan kompleksitas waktu $O(n)$ tetapi itu bukanlah $2n$ dikarenakan input n dan m adalah dua hal yang berbeda. Kompleksitas waktu untuk kasus tersebut dituliskan dengan notasi $O(n + m)$.

```

public static void printItems(int n, int m){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            System.out.println(i + " " + j);
        }
    }
}

```

Pada kode di atas terdapat perulangan berulang tetapi kompleksitas waktu untuk program di atas tidak dinotasikan sebagai $O(n^2)$ karena menggunakan inputan n dan m pada perulangan. Perhatikan bahwa perulangan pertama menggunakan inputan n dan perulangan kedua menggunakan inputan m sehingga kompleksitas waktu untuk kode tersebut adalah $O(n*m)$.

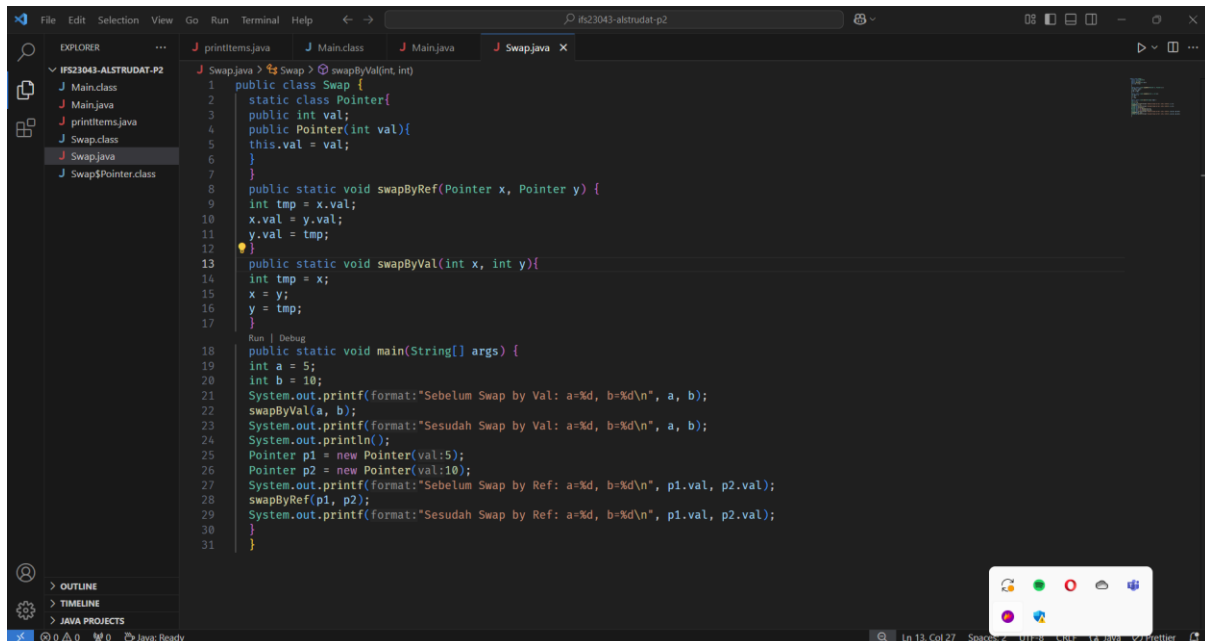
- Big O: Rangkuman

Sebelumnya kamu telah mempelajari terkait notasi Big O. Untuk merangkuman pemahaman sebelumnya kami akan melakukan pemisalan dengan sebuah program yang mempunyai nilai n sebesar 100, maka operasi yang dibutuhkan untuk masingmasing kompleksitas waktu adalah sebagai berikut:

Kompleksitas Waktu	Sebutan	Jumlah Operasi
$O(1)$	Konstan	1
$O(\log n)$	Logaritmik	≈ 7
$O(n)$	Linear	100
$O(n^2)$	Kuadrat	10000

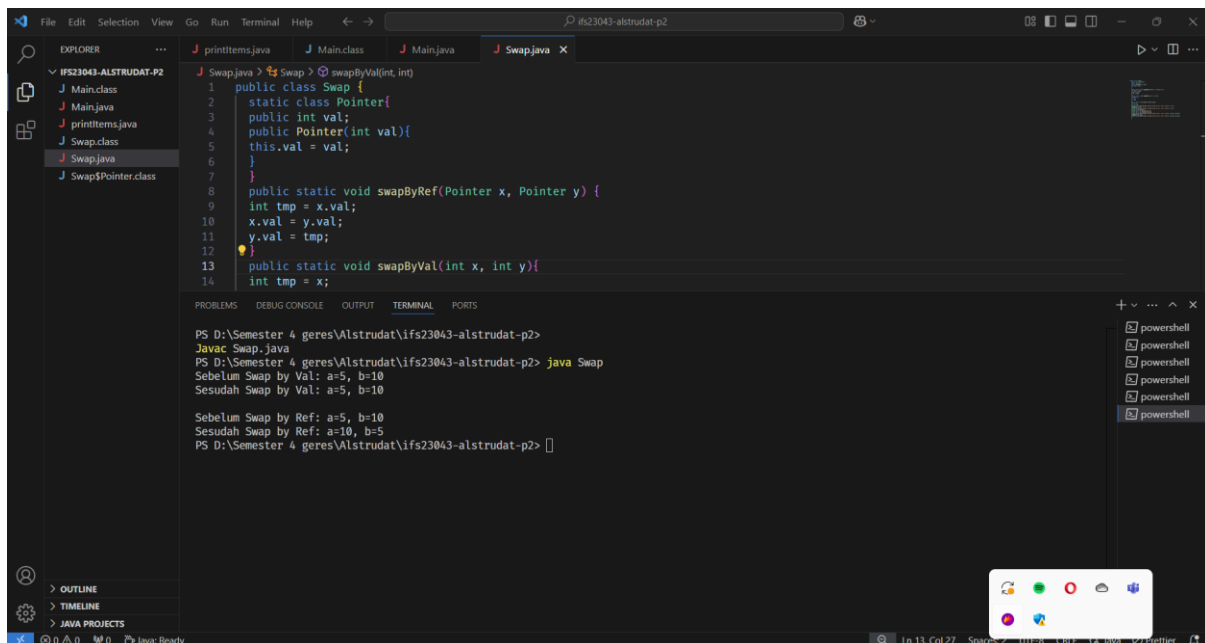
2. Pointer & Latihan

a) Swap By Value vs Reference



```
1 public class Swap {
2     static class Pointer {
3         public int val;
4         public Pointer(int val){
5             this.val = val;
6         }
7     }
8     public static void swapByRef(Pointer x, Pointer y) {
9         int tmp = x.val;
10        x.val = y.val;
11        y.val = tmp;
12    }
13    public static void swapByVal(int x, int y){
14        int tmp = x;
15        x = y;
16        y = tmp;
17    }
18    public static void main(String[] args) {
19        int a = 5;
20        int b = 10;
21        System.out.printf(format:"Sebelum Swap by Val: a=%d, b=%d\n", a, b);
22        swapByVal(a, b);
23        System.out.printf(format:"Setelah Swap by Val: a=%d, b=%d\n", a, b);
24        System.out.println();
25        Pointer p1 = new Pointer(val:5);
26        Pointer p2 = new Pointer(val:10);
27        System.out.printf(format:"Sebelum Swap by Ref: a=%d, b=%d\n", p1.val, p2.val);
28        swapByRef(p1, p2);
29        System.out.printf(format:"Setelah Swap by Ref: a=%d, b=%d\n", p1.val, p2.val);
30    }
31 }
```

Output



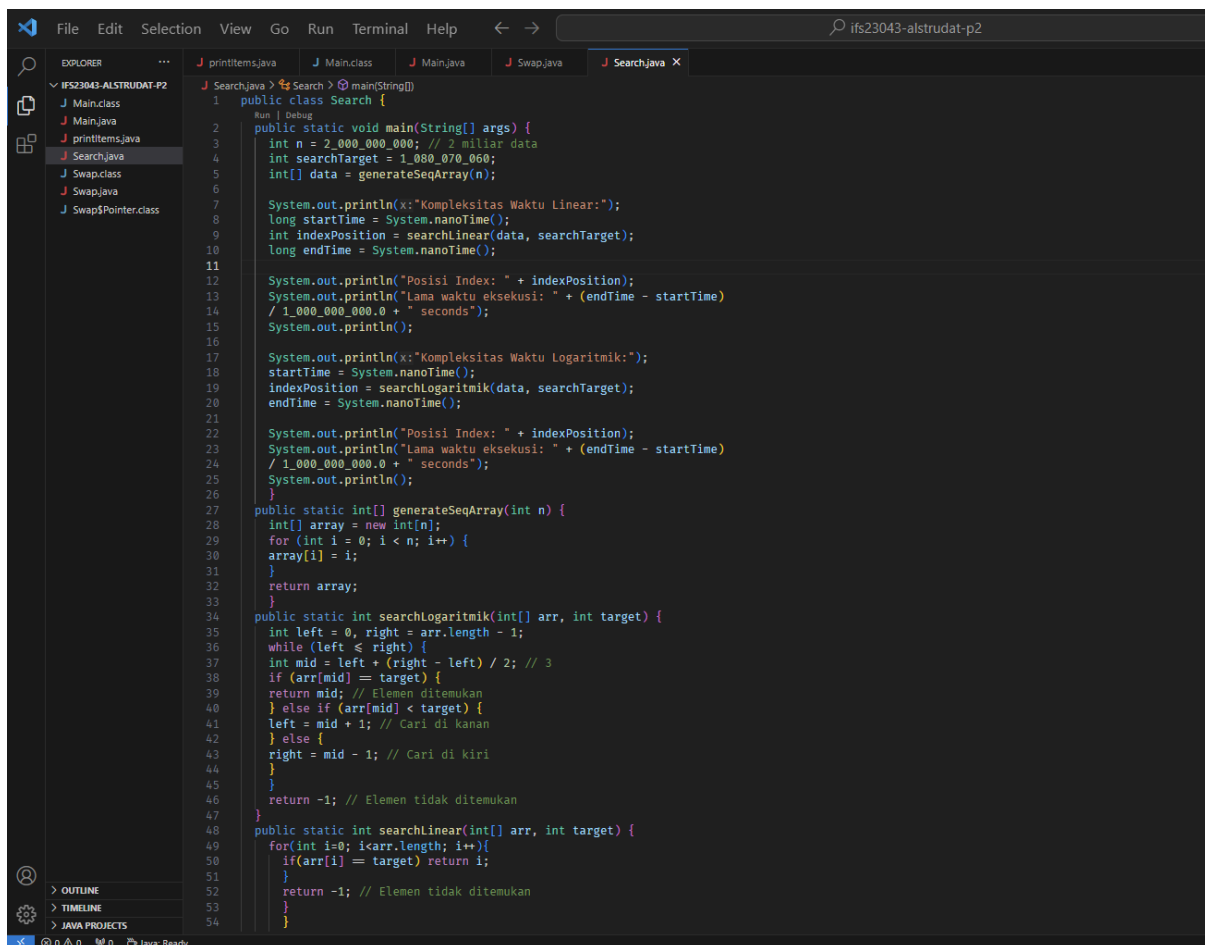
```
PS D:\Semester 4\geres\Alstrudat\ifs23043-alstrudat-p2>
JavaC Swap.java
PS D:\Semester 4\geres\Alstrudat\ifs23043-alstrudat-p2> java Swap
Sebelum Swap by Val: a=5, b=10
Setelah Swap by Val: a=5, b=10

Sebelum Swap by Ref: a=5, b=10
Setelah Swap by Ref: a=10, b=5
PS D:\Semester 4\geres\Alstrudat\ifs23043-alstrudat-p2> 
```

Pointer adalah referensi ke lokasi memori yang menyimpan suatu nilai, bukan nilai itu sendiri. Dalam program di atas, terdapat perbedaan mendasar antara metode `swapByVal` dan `swapByRef`. Metode `swapByVal` menerima dua parameter bertipe primitif `int`, sehingga saat dilakukan pertukaran, hanya terjadi pada salinan nilai yang dikirim ke dalam metode. Akibatnya, perubahan tidak berdampak pada variabel asli karena Java menggunakan `pass by value` untuk tipe primitif. Setelah metode selesai, nilai asli tetap tidak

berubah. Sebaliknya, metode `swapByRef` menggunakan objek dari kelas `Pointer`, di mana nilai yang dipertukarkan adalah referensi ke objek tersebut, bukan sekadar salinan nilai. Oleh karena itu, perubahan yang terjadi dalam metode `swapByRef` juga berdampak langsung pada objek asli di luar metode, karena objek dalam Java diperlakukan sebagai referensi. Dalam eksekusi program, terlihat bahwa setelah pemanggilan `swapByVal`, nilai `a` dan `b` tetap 5 dan 10, sedangkan setelah `swapByRef`, nilai objek `p1` dan `p2` benar-benar tertukar, membuktikan bahwa referensi terhadap objek memungkinkan perubahan yang bersifat permanen.

b) Latihan Search Data



```
1 public class Search {
2     public static void main(String[] args) {
3         int n = 2_000_000_000; // 2 miliar data
4         int searchTarget = 1_080_070_060;
5         int[] data = generateSeqArray(n);
6
7         System.out.println("Kompleksitas Waktu Linear:");
8         long startTime = System.nanoTime();
9         int indexPosition = searchLinear(data, searchTarget);
10        long endTime = System.nanoTime();
11
12        System.out.println("Posisi Index: " + indexPosition);
13        System.out.println("Lama waktu eksekusi: " + (endTime - startTime)
14            / 1_000_000_000.0 + " seconds");
15        System.out.println();
16
17        System.out.println("Kompleksitas Waktu Logaritmik:");
18        startTime = System.nanoTime();
19        indexPosition = searchLogaritmik(data, searchTarget);
20        endTime = System.nanoTime();
21
22        System.out.println("Posisi Index: " + indexPosition);
23        System.out.println("Lama waktu eksekusi: " + (endTime - startTime)
24            / 1_000_000_000.0 + " seconds");
25        System.out.println();
26    }
27
28    public static int[] generateSeqArray(int n) {
29        int[] array = new int[n];
30        for (int i = 0; i < n; i++) {
31            array[i] = i;
32        }
33        return array;
34    }
35
36    public static int searchLogaritmik(int[] arr, int target) {
37        int left = 0, right = arr.length - 1;
38        while (left <= right) {
39            int mid = left + (right - left) / 2; // 3
40            if (arr[mid] == target) {
41                return mid; // Elemen ditemukan
42            } else if (arr[mid] < target) {
43                left = mid + 1; // Cari di kanan
44            } else {
45                right = mid - 1; // Cari di kiri
46            }
47        }
48        return -1; // Elemen tidak ditemukan
49    }
50
51    public static int searchLinear(int[] arr, int target) {
52        for (int i = 0; i < arr.length; i++) {
53            if (arr[i] == target) return i;
54        }
55        return -1; // Elemen tidak ditemukan
56    }
57 }
```

```

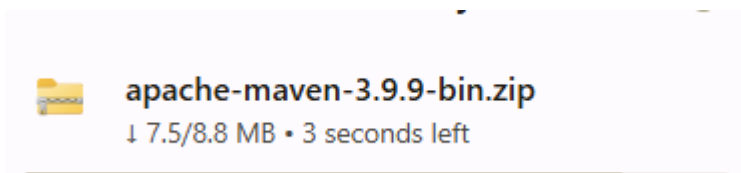
1 public class Search {
2     public static void main(String[] args) {
3         int n = 2_000_000_000; // 2 miliar data
4         int searchTarget = 1_000_070_000;
5         int[] data = generateSeqArray(n);
6
7         System.out.println("Kompleksitas Waktu Linear:");
8         long startTime = System.nanoTime();
9         int indexPosition = searchLinear(data, searchTarget);
10        long endTime = System.nanoTime();
11
12        System.out.println("Posisi Index: " + indexPosition);
13        System.out.println("Lama waktu eksekusi: " + (endTime - startTime) / 1_000_000_000.0 + " seconds");
14        System.out.println();
15
16        System.out.println("Kompleksitas Waktu Logaritmik:");
17        startTime = System.nanoTime();
18        indexPosition = searchLogaritmik(data, searchTarget);
19        endTime = System.nanoTime();
20
21        System.out.println("Posisi Index: " + indexPosition);
22        System.out.println("Lama waktu eksekusi: " + (endTime - startTime) / 1_000_000_000.0 + " seconds");
23        System.out.println();
24    }
25
26    public static int[] generateSeqArray(int n) {
27        int[] array = new int[n];
28        for (int i = 0; i < n; i++) {
29            array[i] = i;
30        }
31        return array;
32    }
33
34    public static int searchLinear(int[] data, int searchTarget) {
35        for (int i = 0; i < data.length; i++) {
36            if (data[i] == searchTarget) {
37                return i;
38            }
39        }
40        return -1;
41    }
42
43    public static int searchLogaritmik(int[] data, int searchTarget) {
44        // Placeholder for binary search logic
45        return -1;
46    }
47 }

```

c) Latihan Menuliskan Kode dengan Gaya yang Konsisten

1) Download Maven pada link berikut:

Binary zip archive [apache-maven-3.9.9-bin.zip](#) [apache-maven-3.9.9-bin.zip.sha512](#) [apache-maven-3.9.9-bin.zip.asc](#)



Name	Date modified	Type	Size
bin	8/14/2024 8:48 AM	File folder	
boot	8/14/2024 8:48 AM	File folder	
conf	8/14/2024 8:48 AM	File folder	
lib	1/31/2025 1:23 PM	File folder	
LICENSE	1/31/2025 1:23 PM	File	19 KB
NOTICE	1/31/2025 1:23 PM	File	5 KB
README.txt	1/31/2025 1:23 PM	Text Document	2 KB

11S2215 - Algorithms and Data Structures

Laporan Praktikum



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

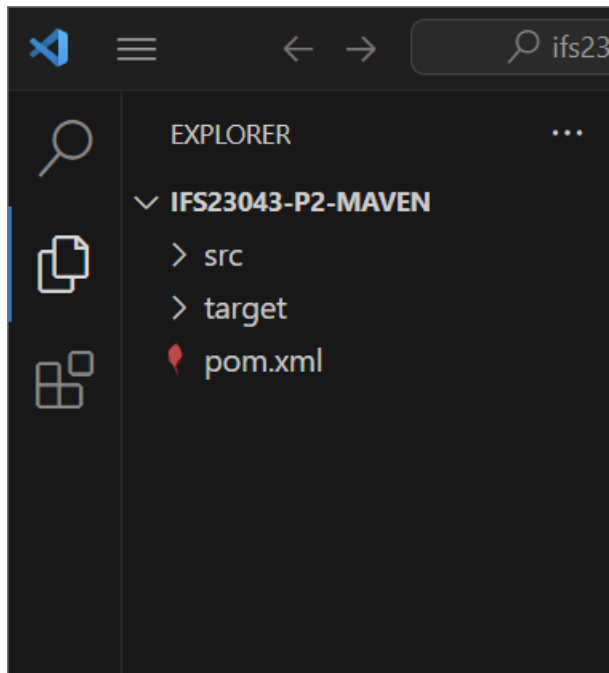
PS C:\Users\ASUS> mvn --version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: C:\tools\apache-maven-3.9.9
Java version: 23.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-23
Default locale: en_US, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
PS C:\Users\ASUS>
```

```
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-lang3/3.17.0/commons-lang3-3.17.0.jar (674 kB at 96 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache-extras/beanshell/bsh/2.0b6/bsh-2.0b6.jar (389 kB at 53 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/com/ibm/icu/icu4j/75.1/icu4j-75.1.jar (14 MB at 1.7 MB/s)
[INFO] Generating project in Batch mode
Downloaded from central: https://repo.maven.apache.org/maven2/archetype-catalog.xml
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar (4.3 kB at 100 kB/s)
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] Parameter: basedir, Value: C:\Users\ASUS
[INFO] Parameter: package, Value: del.ifs23043
[INFO] Parameter: groupId, Value: del.ifs23043
[INFO] Parameter: artifactId, Value: ifs23043-p2-maven
[INFO] Parameter: packageName, Value: del.ifs23043
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\ASUS\ifs23043-p2-maven
[INFO] BUILD SUCCESS
[INFO] Total time: 23.264 s
[INFO] Finished at: 2025-02-01T09:45:14+07:00
[INFO]
C:\Users\ASUS>
```

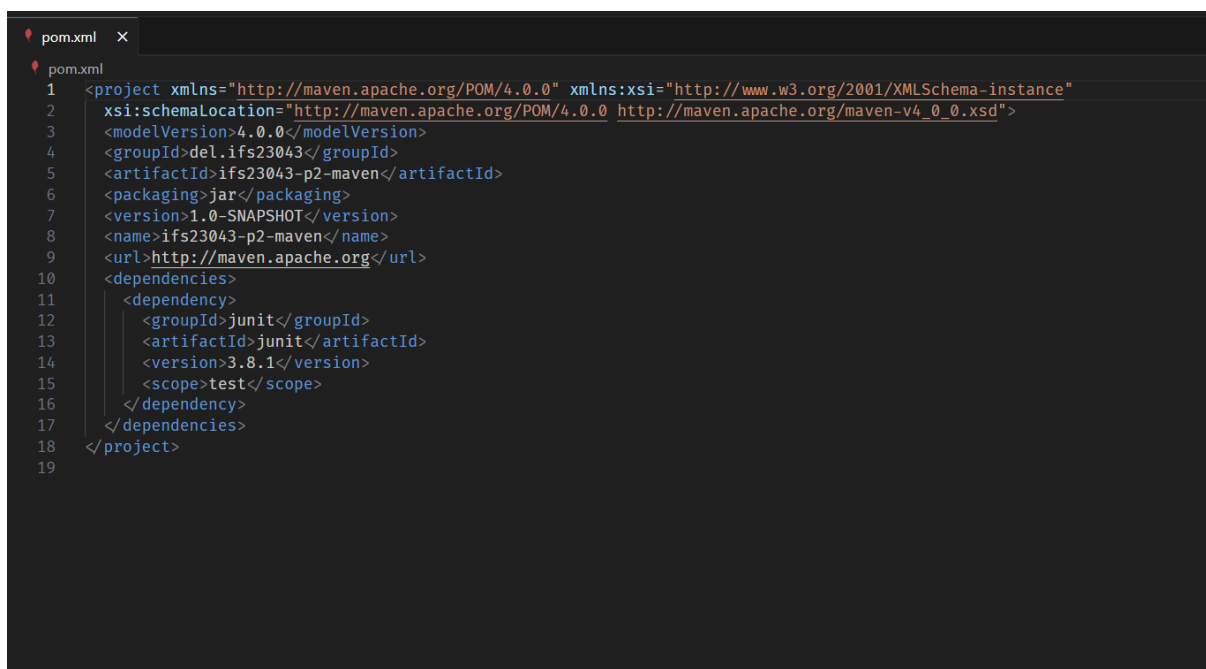
```
Microsoft Windows [Version 10.0.22631.4751]
(c) Microsoft Corporation. All rights reserved.

D:\Semester 4 geres\Alstrudat>mvn archetype:generate -DgroupId=del.ifs23043 -DartifactId=ifs23043-p2-maven -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-metadata.xml
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/maven-metadata.xml (14 kB at 12 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/maven-metadata.xml (21 kB at 17 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-archetype-plugin/maven-metadata.xml (1.0 kB at 18 kB/s)
[INFO] -----< org.apache.maven:standalone-pom >-----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----[ pom ]-----
[INFO]
[INFO] >>> archetype:3.3.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< archetype:3.3.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO] --- archetype:3.3.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
Downloaded from central: https://repo.maven.apache.org/maven2/archetype-catalog.xml
Downloaded from central: https://repo.maven.apache.org/maven2/archetype-catalog.xml (16 MB at 8.7 MB/s)
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] Parameter: basedir, Value: D:\Semester 4 geres\Alstrudat
[INFO] Parameter: package, Value: del.ifs23043
[INFO] Parameter: groupId, Value: del.ifs23043
[INFO] Parameter: artifactId, Value: ifs23043-p2-maven
[INFO] Parameter: packageName, Value: del.ifs23043
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: D:\Semester 4 geres\Alstrudat\ifs23043-p2-maven
[INFO] BUILD SUCCESS
[INFO] Total time: 9.230 s
[INFO] Finished at: 2025-02-03T16:03:08+07:00
```

7) Buka proyek Java yang dihasilkan maven menggunakan VSCode. Dalam kasus ini nama proyek yang dihasilkan adalah "ifs18005-p2-maven".

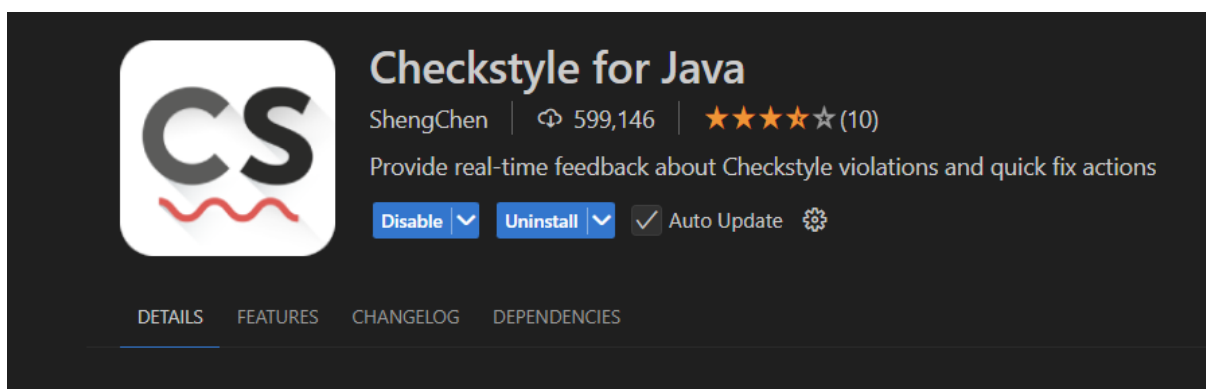


8) Buka file "pom.xml" dan modifikasi isinya menjadi seperti berikut:

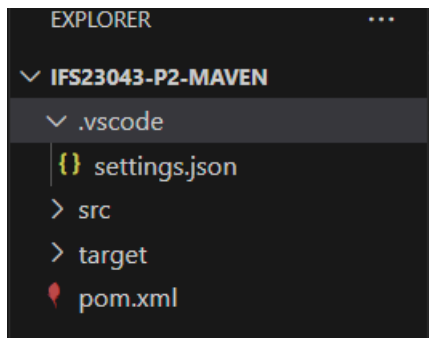


```
pom.xml x J AppTest.java {} settings.json
pom.xml > project > dependencies > dependency > version
17 </dependencies>
18
19 <build>
20 <plugins>
21 <plugin>
22 <groupId>org.apache.maven.plugins</groupId>
23 <artifactId>maven-checkstyle-plugin</artifactId>
24 <version>3.2.0</version>
25 <configuration>
26 <configLocation>google_checks.xml</configLocation>
27 </configuration>
28 <executions>
29 <execution>
30 <phase>verify</phase>
31 <goals>
32 <goal>checkstyle</goal>
33 </goals>
34 </execution>
35 </executions>
36 </plugin>
37
38 <plugin>
39 <groupId>org.apache.maven.plugins</groupId>
40 <artifactId>maven-compiler-plugin</artifactId>
41 <version>3.10.1</version>
42 <configuration>
43 <source>23</source>
44 <target>23</target>
45 </configuration>
46 </plugin>
47 </plugins>
48 </build>
49 </project>
50
```

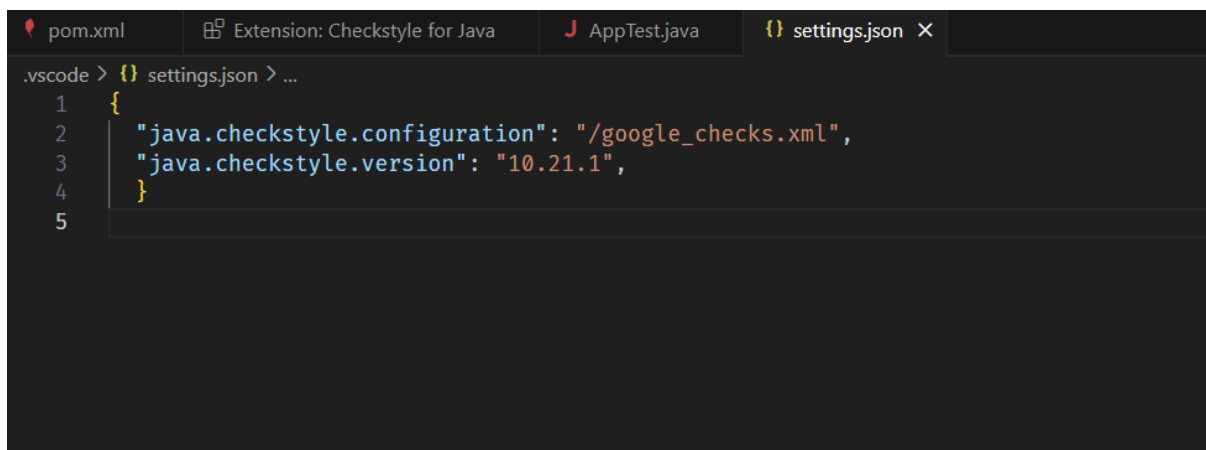
9) Download extension "Checkstyle for Java" di VSCode.



Buat folder baru pada proyek di VSCode dengan nama ".vscode" dan tambahkan file settings.json di dalamnya.

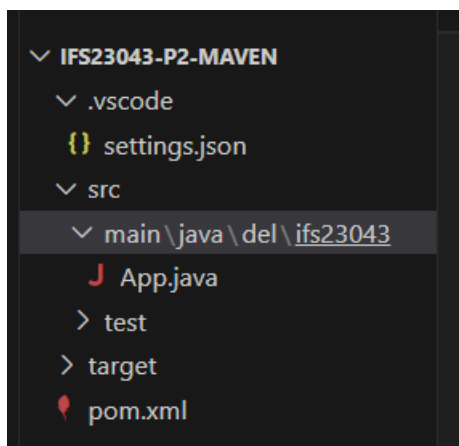


Modifikasi isi dari file "settings.json" seperti berikut:



10) Buka file App.java pada lokasi:

"src/main/java/del/{username-kamu}/App.java".



```
src > test > java > del > ifs23043 > J AppTest.java > AppTest > AppTest(String)
1 package del.ifs23043;
2
3 import junit.framework.Test;
4 import junit.framework.TestCase;
5 import junit.framework.TestSuite;
6
7 /**
8  * Unit test for simple App.
9  */
10 public class AppTest
11     extends TestCase
12 {
13     /**
14      * Create the test case
15      *
16      * @param testName name of the test case
17      */
18     public AppTest( String testName )
19     {
20         super( testName );
21     }
22
23     /**
24      * @return the suite of tests being tested
25      */
26     public static Test suite()
27     {
28         return new TestSuite( AppTest.class );
29     }
30
31     /**
32      * Rigourous Test :-)
33      */
34     public void testApp()
35     {
```

15) Ubah kode program pada file "App.java" seperti berikut untuk memperbaiki warning yang muncul dari checkstyle:

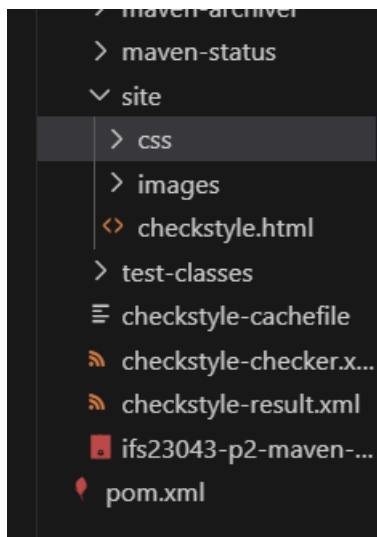
```
pom.xml  Extension: Checkstyle for Java  J App.java  X  settings.json
src > main > java > del > J App.java > App
1 package del;
2
3 /**
4  * Kelas utama untuk aplikasi "Hello World".
5  */
6 public class App {
7
8     /**
9      * Metode utama yang akan dieksekusi saat program dijalankan.
10     *
11     * @param args Argumen baris perintah.
12     */
13     public static void main(String[] args) {
14         System.out.println(x:"Hello World!");
15     }
16 }
17
```

11) Buka terminal dan jalankan perintah berikut untuk melihat laporan checkstyle terhadap kode program yang dituliskan.

```
PS D:\Semester 4 geres\Alstrudat\ifs23043-p2-maven> mvn verify
[INFO] Scanning for projects...
[INFO]
[INFO] -----< del.ifs23043:ifs23043-p2-maven >-----
[INFO] Building ifs23043-p2-maven 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ ifs23043-p2-maven ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build
[INFO] skip non existing resourceDirectory D:\Semester 4 geres\Alstrudat\ifs23043-p2-maven
[INFO]
```

Setelah selesai melakukan pengecekan buka kembali file "checkstyle.html"

12) Setelah selesai melakukan pengecekan buka file "checkstyle.html" pada web browser kamu. File ini berada pada lokasi "target/site/checkstyle.html".



Tampilan pada web browser:

Checkstyle Results

The following document contains the results of Checkstyle 9.3 with google_checks.xml ruleset.

Summary

Files	Info	Warnings	Errors
2	0	13	0

Files

File	I	W	E
del/App.java	0	3	0
del/ifs23043/App.java	0	10	0

Rules

Category	Rule	Violations	Severity
blocks	LeftCurly	2	Warning
Indentation	Indentation		

Dari laporan di atas bisa dilihat bahwa terdapat warning sebanyak 10 pada kode program "App.java" untuk detailnya dapat dilihat di bagian bawah.

13) Silahkan kunjungi dokumentasi dari checkstyle untuk mengetahui jenis-jenis masalah yang mungkin timbul dan cara memperbaikinya, pada link berikut:

checkstyle

Last Published: 2025-01-26 | Version: 10.21.2

Overview

Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard.

Checkstyle is highly configurable and can be made to support almost any coding standard. An example configuration files are supplied supporting the [Sun Code Conventions](#), [Google Java Style](#).

A good example of a report that can be produced using Checkstyle and [Maven](#) can be [seen here](#).

Features

Checkstyle can check many aspects of your source code. It can find class design problems, method design problems. It also has the ability to check code layout and formatting issues.

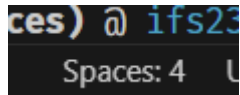
For a detailed list of available checks please refer to the [Checks](#) page.

Download

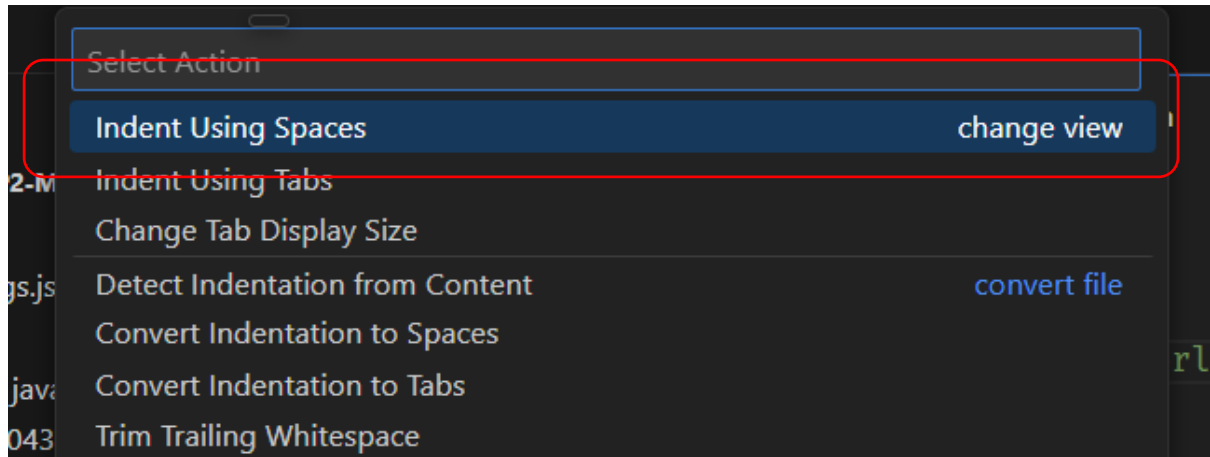
The latest release of Checkstyle can be downloaded from the [GitHub releases page](#), or [Maven central](#).

If you want to live on the bleeding edge, you can [checkout](#) the current development code from GitHub and co

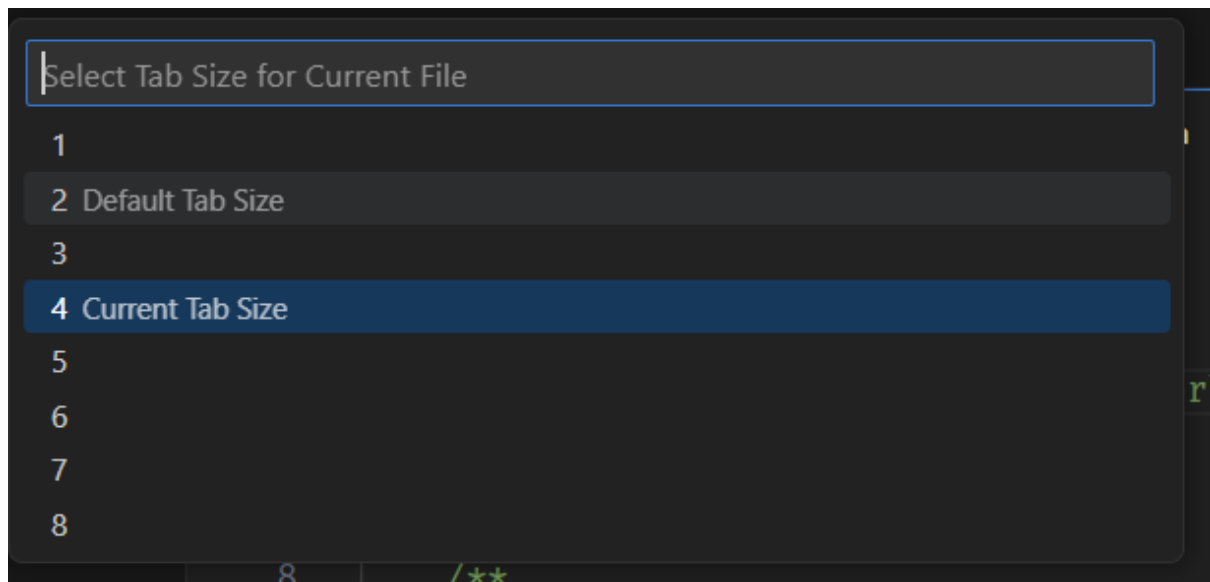
14) Untuk melakukan perbaikan kode program Java terlebih dahulu kita perlu mengubah pengaturan indentation dari 4 menjadi 2 pada VSCode karena google style menggunakan indentation 2. Pada VSCode di sudut kanan bawah akan terdapat tulisan "Spaces: 4" atau "Tab Size: 4" silahkan pilih pada tulisan berikut.



Pada dialog pilih opsi "Indent Using Spaces".



Pada dialog pilih opsi "2".



Sekarang pada text editor garis indentation telah diatur menjadi 2.

```
src > main > java > del > App.java > App
1 package del;
2
3 /**
4  * Hello World!
5  */
6 public class App {
7
8     /**
9      * Metode utama yang akan dieksekusi saat program dijalankan.
10     *
11     * @param args Argumen baris perintah.
12     */
13     public static void main(String[] args) {
14         System.out.println(x:"Hello World!");
15     }
16 }
17
```

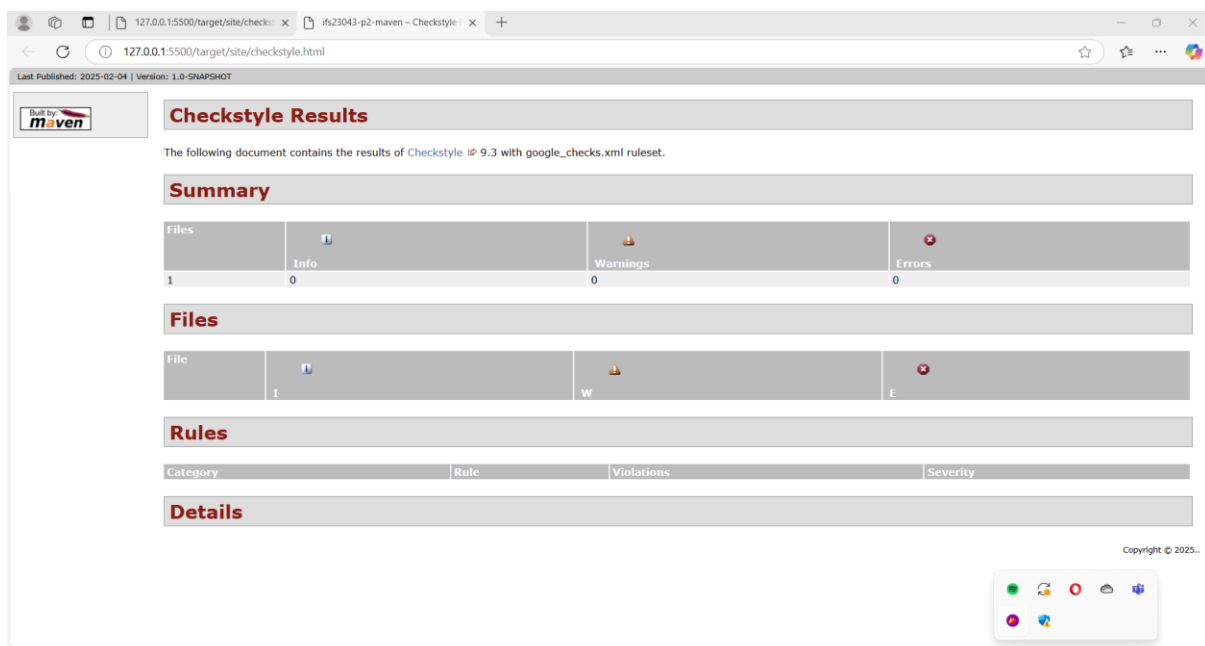
15) Ubah kode program pada file "App.java" seperti berikut untuk memperbaiki warning yang muncul dari checkstyle:

```
src > test > java > del > ifs23043 > App.java > {} del.ifs23043
1 package del.ifs23043;
2
3 /**
4  * Kelas utama untuk aplikasi "Hello World".
5  */
6 public class App {
7
8     /**
9      * Metode utama yang akan dieksekusi saat program dijalankan.
10     *
11     * @param args Argumen baris perintah.
12     */
13     public static void main(String[] args) {
14         System.out.println(x:"Hello World!");
15     }
16 }
17
```

Jalankan kembali perintah "mvn verify" untuk membuat laporan hasil pengecekan kode program menggunakan checkstyle.

```
PS D:\Semester 4 geres\Alstrudat\ifs23043-p2-maven> mvn verify
[INFO] Scanning for projects...
[INFO]
[INFO] -----< del.ifs23043:ifs23043-p2-maven >-----
[INFO] Building ifs23043-p2-maven 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ ifs23043-p2-maven ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is pl
[INFO] skip non existing resourceDirectory D:\Semester 4 geres\Alstrudat\ifs23043-p2-maven\src\
[INFO]
[INFO] --- compiler:3.10.1:compile (default-compile) @ ifs23043-p2-maven ---
```

Setelah selesai melakukan pengecekan buka kembali file "checkstyle.html"



127.0.0.1:5500/target/site/checkstyle.html

Last Published: 2025-02-04 | Version: 1.0-SNAPSHOT

Checkstyle Results

The following document contains the results of Checkstyle 9.3 with google_checks.xml ruleset.

Summary

Files	Info	Warnings	Errors
1	0	0	0

Files

File	I	W	E
------	---	---	---

Rules

Category	Rule	Violations	Severity
----------	------	------------	----------

Details

Copyright © 2025..

Jika berhasil maka tidak terdapat laporan warning lagi yang tersedia pada file "checkstyle.html". Ini berarti kode program yang dibuat telah memenuhi standar java google style.

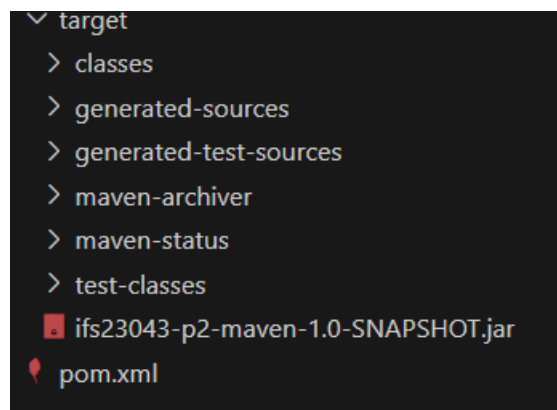
16) Untuk membuild proyek Java yang dibuat dengan maven dapat dilakukan dengan menuliskan perintah berikut pada terminal:

mvn clean package

Tampilan pada terminal

```
PS D:\Semester 4 geres\Alstrudat\ifs23043-p2-maven> mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< del.ifs23043:ifs23043-p2-maven >-----
[INFO] Building ifs23043-p2-maven 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ ifs23043-p2-maven ---
[INFO] Deleting D:\Semester 4 geres\Alstrudat\ifs23043-p2-maven\target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ ifs23043-p2-maven ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
Ready
```

Jika berhasil melakukan build proyek, maka akan terdapat file dengan nama "ifs23043-p2-maven-1.0-SNAPSHOT.jar" di dalam folder "target".



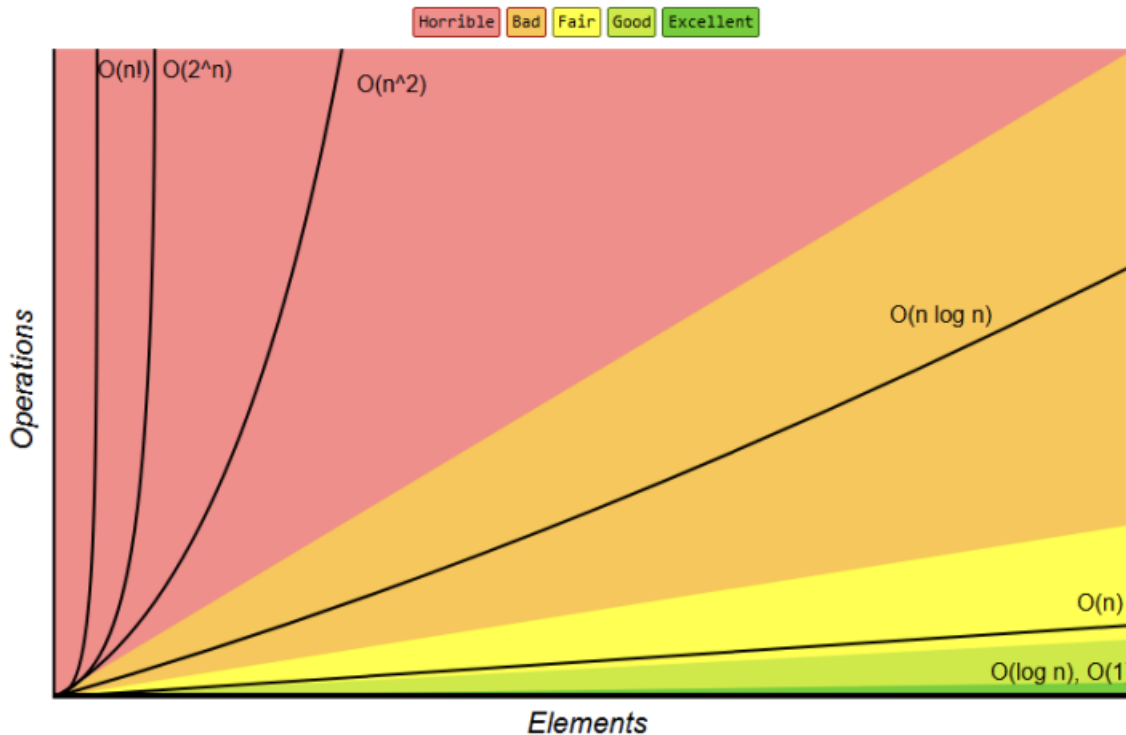
Untuk menjalankan aplikasi Java hasil build dari maven dapat dilakukan dengan menuliskan perintah berikut:

Tampilan pada terminal

```
PROBLEMS 4 DEBUG CONSOLE OUTPUT TERMINAL PORTS
PS D:\Semester 4 geres\Alstrudat\ifs23043-p2-maven> java -cp target/ifs23043-p2-maven-1.0-SNAPSHOT.jar del.ifs23043.App
Hello World!
PS D:\Semester 4 geres\Alstrudat\ifs23043-p2-maven> 
```

3. Tantangan

a) Eksplorasi Notasi Big O



Silahkan rangkum beserta berikan contohnya untuk notasi Big O lainnya yaitu $O(n!)$, $O(2^n)$ dan $O(n \log n)$.

Pembahasan:

1. $O(n!)$ - Faktorial

Kompleksitas $O(n!)$ adalah yang paling tidak efisien di antara ketiga notasi ini karena pertumbuhannya yang sangat cepat. Jika n bertambah, jumlah operasi meningkat secara faktorial, yaitu $n \times (n - 1) \times (n - 2) \dots \times 1$. Ini berarti bahkan dengan input yang relatif kecil, waktu eksekusi algoritma dapat menjadi tidak realistis untuk diterapkan.

Contoh Algoritma dengan $O(n!)$

- Brute-force dalam Traveling Salesman Problem (TSP)

Dalam TSP, kita mencari jalur terpendek yang melewati semua kota dan kembali ke titik awal. Jika menggunakan metode brute-force, kita harus mengeksplorasi semua kemungkinan jalur, yang jumlahnya mencapai $n!$.

- Permutasi dari sebuah himpunan

Jika ingin mencari semua kemungkinan urutan dari n elemen yang berbeda, kita harus menjelajahi semua $n!$ kombinasi.

- Backtracking dalam penyelesaian teka-teki kompleks

Beberapa algoritma pencarian solusi berbasis backtracking, seperti puzzle Sudoku yang diselesaikan dengan brute-force, dapat memiliki kompleksitas mendekati $O(n!)$ dalam kasus terburuk.

Karakteristik $O(n!)$

- ✓ Mengeksplorasi semua kemungkinan solusi.
- ✓ Digunakan untuk permasalahan yang bersifat kombinatorial.
- ✓ Sangat tidak efisien untuk input besar karena pertumbuhan faktorial.
- ✓ Biasanya dihindari kecuali untuk masalah dengan ukuran input kecil.

2. $O(2^n)$ - Eksponensial

Kompleksitas $O(2^n)$ menunjukkan bahwa jumlah operasi dalam algoritma berlipat ganda setiap kali ukuran input bertambah satu. Notasi ini umumnya ditemukan dalam **algoritma rekursif yang tidak dioptimalkan**, terutama dalam masalah yang membutuhkan eksplorasi semua kemungkinan subset atau keputusan.

Contoh Algoritma dengan $O(2^n)$

- Algoritma rekursif untuk menghitung Fibonacci

Jika kita menghitung angka Fibonacci menggunakan rekursi tanpa optimasi (misalnya tanpa memoization), jumlah pemanggilan fungsi akan tumbuh secara eksponensial.

- Subset problem

Dalam beberapa masalah kombinatorial, seperti Subset Sum Problem, kita harus mengeksplorasi semua kemungkinan subset dari suatu himpunan, yang jumlahnya adalah 2^n .

- Brute-force dalam pemecahan masalah NP-Hard

Misalnya dalam Knapsack Problem, algoritma brute-force harus mempertimbangkan semua kombinasi item yang mungkin.

Karakteristik $O(2^n)$

- ✓ Digunakan dalam masalah kombinatorial yang melibatkan subset atau rekursi.
- ✓ Banyak muncul dalam masalah yang tidak dapat diselesaikan secara efisien dengan metode lain.
- ✓ Tidak optimal untuk input yang besar karena pertumbuhan eksponensial.

- ✓ Dapat dioptimalkan menggunakan teknik seperti memoization atau dynamic programming.

3. $O(n \log n)$ - Linear Logaritmik

Kompleksitas $O(n \log n)$ lebih efisien dibandingkan $O(2^n)$ dan $O(n!)$, tetapi masih lebih lambat dibandingkan algoritma linear $O(n)$. Algoritma dengan kompleksitas ini sering ditemukan dalam metode **divide and conquer**, di mana input dibagi menjadi beberapa bagian kecil, diproses secara terpisah, lalu digabungkan kembali.

Contoh Algoritma dengan $O(n \log n)$

- Merge Sort dan Quick Sort (rata-rata dan terbaik)

Kedua algoritma sorting ini membagi array menjadi dua bagian secara berulang hingga mencapai ukuran minimum, lalu menggabungkannya kembali dengan cara yang efisien.

- Heap Sort:

Menggunakan struktur data heap untuk menyusun elemen dalam urutan yang diinginkan.

- Algoritma pencarian dalam struktur data pohon (Binary Search Tree):

Mencari elemen dalam Balanced Binary Search Tree (BST) memiliki kompleksitas $O(\log n)$, tetapi jika kita harus melakukan pencarian pada n elemen, kompleksitas totalnya menjadi $O(n \log n)$.

Karakteristik $O(n \log n)$

- ✓ Lebih cepat dibandingkan kompleksitas kuadratik $O(n^2)$.
- ✓ Digunakan dalam sorting dan struktur data pohon.
- ✓ Efisien untuk pemrosesan data dalam jumlah besar.
- ✓ Tidak secepat $O(n)$ dalam kasus terbaik, tetapi tetap lebih optimal dibandingkan eksponensial.

b) Checkstyle Matrix App

Silahkan memperbaiki kode program Matrix App yang telah kamu buat sebelumnya agar sesuai dengan gaya penulisan google. Laporan hasil dari pengecekan checkstyle harus di upload ke Google Drive berupa file dengan nama "checkstyle.html". Sebelum upload ubah penamaan file tersebut menjadi **"ifs23043-checkstyle-p2-matrix.html"**.

Checkstyle Results

The following document contains the results of Checkstyle 9.3 with google_checks.xml ruleset.

Summary

Files	Info	Warnings	Errors
0	0	0	0

Files

File	Info	Warnings	Errors
I		W	E

Rules

Category	Rule	Violations	Severity
----------	------	------------	----------

Details

Copyright © 2025..

Checkstyle Results

The following document contains the results of Checkstyle 9.3 with google_checks.xml ruleset.

Summary

Files	Info	Warnings	Errors
0	0	0	0

Files

File	Info	Warnings	Errors
I		W	E

Rules

Category	Rule	Violations	Severity
----------	------	------------	----------

Details

Copyright © 2025..

c) Kompleksitas Waktu Matrix App Setelah memperbaiki kode program sesuai dengan gaya penulisan google, selanjutnya lakukan analisis kompleksitas waktu yang dibutuhkan oleh program Matrix

App kamu untuk menjalankan fungsi penjumlahan, pengurangan dan perkalian matiks.

1. Kompleksitas Penjumlahan Matriks (sum)

```
1  for (int i = 0; i < x.baris; i++) {
2      for (int j = 0; j < x.kolom; j++) {
3          result.setData(i, j, x.getData(i, j) + y.getData(i, j));
4      }
5  }
```

- **Outer loop** berjalan sebanyak baris kali.
- **Inner loop** berjalan sebanyak kolom kali.
- **Operasi dalam loop** adalah operasi aritmatika sederhana dan assignment.

Operasi penjumlahan matriks dalam program ini memiliki kompleksitas waktu $O(n \times m)$, di mana n adalah jumlah baris dan m adalah jumlah kolom dalam matriks. Hal ini terjadi karena algoritma harus mengakses setiap elemen dari dua matriks input, melakukan operasi penjumlahan, dan menyimpan hasilnya dalam matriks baru. Dengan dua loop bersarang yang masing-masing berjalan sebanyak n dan m kali, jumlah total operasi yang dilakukan berbanding lurus dengan jumlah elemen dalam matriks. Oleh karena itu, operasi ini tergolong efisien dan dapat dieksekusi dengan cepat untuk matriks berukuran besar.

Sehingga, kompleksitas waktu $O(m \times n)$, di mana m adalah jumlah baris dan n adalah jumlah kolom.

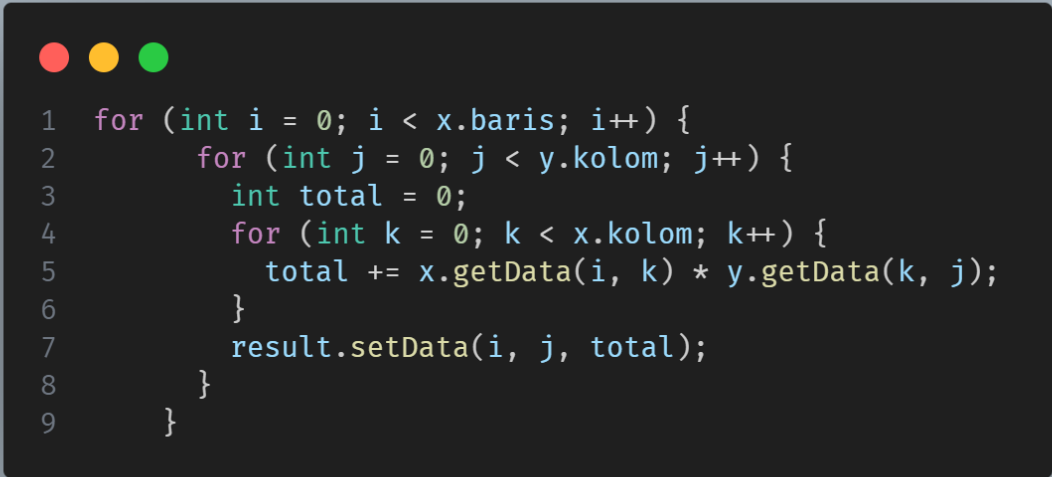
2. Kompleksitas Pengurangan Matriks (difference)

```
1  Matrix result = new Matrix(x.label, x.baris, x.kolom);
2      for (int i = 0; i < x.baris; i++) {
3          for (int j = 0; j < x.kolom; j++) {
4              result.setData(i, j, x.getData(i, j) - y.getData(i, j));
5          }
6      }
```

- Prosesnya identik dengan penjumlahan matriks.
- Kompleksitasnya juga $O(m \times n)$.

Sama seperti operasi penjumlahan, pengurangan matriks juga memiliki kompleksitas waktu $O(n \times m)$. Ini disebabkan oleh struktur perulangan yang sama, di mana setiap elemen matriks diakses satu per satu untuk dilakukan operasi pengurangan. Karena jumlah operasi yang dilakukan tetap linear terhadap jumlah elemen matriks, performanya tetap efisien dan tidak jauh berbeda dengan operasi penjumlahan. Oleh karena itu, baik operasi penjumlahan maupun pengurangan dapat dijalankan dengan cepat tanpa menyebabkan lonjakan waktu eksekusi yang signifikan.

3. Kompleksitas Perkalian Matriks (product)



```
1  for (int i = 0; i < x.baris; i++) {
2      for (int j = 0; j < y.kolom; j++) {
3          int total = 0;
4          for (int k = 0; k < x.kolom; k++) {
5              total += x.getData(i, k) * y.getData(k, j);
6          }
7          result.setData(i, j, total);
8      }
9  }
```

- Outer loop berjalan $x.baris$ kali.
- Middle loop berjalan $y.kolom$ kali.
- Inner loop berjalan $x.kolom$ kali untuk setiap elemen hasil.
- Total operasi dalam worst case adalah $m \times p \times n$, dengan m sebagai jumlah baris x , p sebagai jumlah kolom y , dan n sebagai jumlah kolom x .

Berbeda dengan penjumlahan dan pengurangan, operasi perkalian matriks memiliki kompleksitas waktu $O(n \times m \times p)$, di mana n adalah jumlah baris matriks pertama, m adalah jumlah kolom matriks pertama (atau jumlah baris matriks kedua), dan p adalah jumlah kolom matriks kedua. Kompleksitas ini lebih tinggi karena algoritma menggunakan tiga loop bersarang untuk mengalikan setiap elemen dari matriks pertama dengan elemen yang sesuai

dari matriks kedua, lalu menjumlahkan hasilnya. Jika matriks berukuran persegi $n \times n$, maka kompleksitasnya menjadi $O(n^3)$, yang berarti waktu eksekusi meningkat drastis seiring bertambahnya ukuran matriks. Untuk menangani matriks besar secara lebih efisien, dapat dipertimbangkan penggunaan algoritma optimasi seperti metode Strassen atau teknik perkalian berbasis blok.

Sehingga, kompleksitas waktu $O(m \times n \times p)$.

Link Google drive checkstyle.html

https://drive.google.com/drive/folders/12EQPJ6_CgrhQfgTmdw5vCLdhYGS02uTq?usp=sharing