

IF2211 Strategi Algoritma
Laporan Tugas Kecil 2



Disusun oleh:
Muhammad Kinan Arkansyaddad (13523152)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025

Daftar Isi

Daftar Isi	2
BAB I	
DESKRIPSI MASALAH DAN ALGORITMA	4
1.1 Quadtree dalam Kompresi Gambar	4
1.2 Algoritma Divide and Conquer	6
1.3 Algoritma Divide and Conquer dalam Kompresi Gambar	9
1.4 Metode Pengukuran Error	12
1.4.1 Variance	12
1.4.2 Mean Absolute Difference (MAD)	13
1.4.3 Max Pixel Difference (MPD)	14
1.4.4 Entropy	15
1.4.5 Structural Similarity Index (SSIM)	15
BAB II	
IMPLEMENTASI PROGRAM DALAM BAHASA JAVA	18
2.1 App.java	18
2.2 QuadtreeNode.java	18
2.3 QuadtreeImageCompressor.java	22
2.4 InputHandler.java	24
2.5 OutputHandler.java	26
BAB III	
SOURCE CODE PROGRAM	28
3.1 Repository Program	28
3.2 Pustaka Eksternal	28
3.2 Source Code Algoritma Divide and Conquer	28
BAB IV	
EKSPERIMENT	40
4.1 Test Case 1	40

4.2 Test Case 2	41
4.3 Test Case 3	43
4.4 Test Case 4	45
4.5 Test Case 5	47
4.6 Test Case 6	49
4.7 Test Case 7	51
4.8 Test Case 8	52
4.9 Test Case 9	53
4.10 Test Case 10	55
4.11 Test Case 11	56
4.12 Test Case 12	57
BAB V	
ANALISIS DAN PEMBAHASAN	60
5.1 Analisis Kompleksitas Waktu dan Ruang Program	60
5.2 Analisis Hasil Percobaan	61
BAB VI	
LAMPIRAN	65
Daftar Pustaka	66

BAB I

DESKRIPSI MASALAH DAN ALGORITMA

1.1 Quadtree dalam Kompresi Gambar



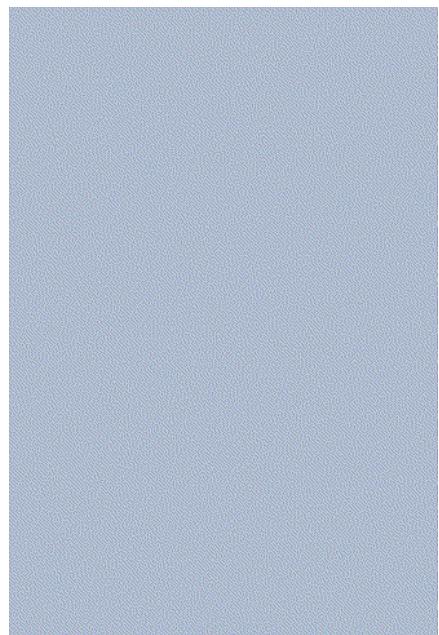
Gambar 1. Quadtree dalam Kompresi Gambar

(Sumber:

<https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>)

Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan analisis sistem warna RGB, yaitu dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel di dalamnya. Jika bagian tersebut tidak seragam, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan.

Dalam implementasi teknis, sebuah Quadtree direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar



Gambar 2. Proses Pembentukan Quadtree dalam Kompresi Gambar (.gif)

(Sumber:

https://miro.medium.com/v2/resize:fit:640/format:webp/1*LHD7PsbmbgNBFrYkxyG5dA.gif

Tugas ini meminta program sederhana dalam bahasa C/C#/C++/Java (CLI) yang mengimplementasikan algoritma

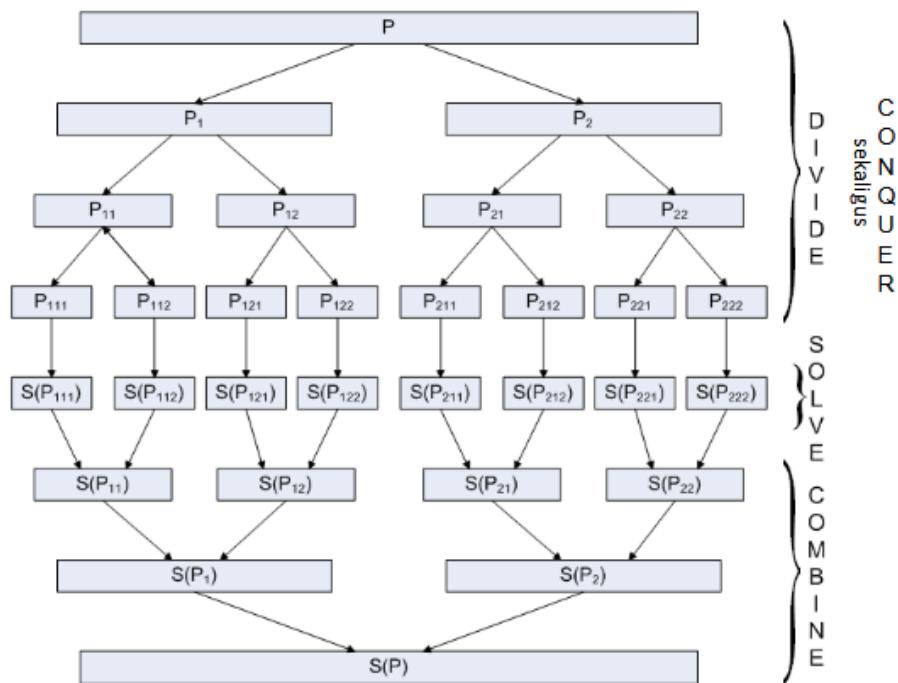
divide and conquer untuk melakukan kompresi gambar berbasis quadtree yang mengimplementasikan seluruh parameter yang telah disebutkan sebagai user input. Alur program adalah sebagai berikut:

1. [INPUT] alamat absolut gambar yang akan dikompresi.
2. [INPUT] metode perhitungan error (gunakan penomoran sebagai input).
3. [INPUT] ambang batas (pastikan range nilai sesuai dengan metode yang dipilih).
4. [INPUT] ukuran blok minimum.
5. [INPUT] Target persentase kompresi (floating number, 1.0 = 100%), beri nilai 0 jika ingin menonaktifkan mode ini, jika mode ini aktif maka nilai threshold bisa menyesuaikan secara otomatis untuk memenuhi target persentase kompresi (bonus).
6. [INPUT] alamat absolut gambar hasil kompresi.
7. [INPUT] alamat absolut gif (bonus).
8. [OUTPUT] waktu eksekusi.
9. [OUTPUT] ukuran gambar sebelum.
10. [OUTPUT] ukuran gambar setelah.
11. [OUTPUT] persentase kompresi.
12. [OUTPUT] kedalaman pohon.
13. [OUTPUT] banyak simpul pada pohon.
14. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.
15. [OUTPUT] GIF proses kompresi pada alamat yang sudah ditentukan (bonus).

1.2 Algoritma Divide and Conquer

Divide and Conquer adalah algoritma yang menyelesaikan persoalan dengan cara:

- Divide: Membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula, tetapi berukuran lebih kecil, idealnya setiap upa-persoalan berukuran hampir sama.
- Conquer: Menyelesaikan (*solve*) masing-masing upa-persoalan secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar.
- Combine: Menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.



Keterangan:
 P = persoalan
 S = solusi

Gambar 3. Ilustrasi Proses Algoritma Divide and Conquer
(Sumber:

[https://informatika.stei.itb.ac.id/~rinaldi.munir/stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)

Objek persoalan yang dibagi di algoritma *divide and conquer* adalah masukan (*input*) atau *instances* persoalan yang berukuran n , seperti tabel (larik), matriks, eksponen, polinom, dan lainnya tergantung persoalan. Tiap-tiap upa-persoalan memiliki karakteristik yang sama (*the same type*) dengan karakteristik persoalan semula namun berukuran lebih kecil sehingga metode *divide and conquer* lebih natural diungkapkan dengan skema rekursif. Skema umum algoritma *divide and conquer* dapat ditunjukkan dengan *pseudocode* berikut:

```

procedure DIVIDEandCONQUER(input p: problem, n: integer)
{ Menyelesaikan persoalan P dengan algoritma divide and conquer. Masukan: masukan persoalan P berukuran n.
Luaran: solusi dari persoalan semula }
Deklarasi
r: integer

Algoritma
if n <= n0 then {ukuran persoalan p sudah cukup kecil}
    SOLVE persoalan P yang berukuran n ini
else
    DIVIDE menjadi r upa-persoalan,  $P_1, P_2, \dots, P_r$ , dengan
    ukuran  $n_1, n_2, \dots, n_r$ 
    for masing-masing  $P_1, P_2, \dots, P_r$ , do
        DIVIDEandCONQUER( $P_i, n_i$ )
    endfor
    COMBINE solusi dari  $P_1, P_2, \dots, P_r$  menjadi solusi
    persoalan semua
endif
```

Kompleksitas algoritma *divide and conquer* dapat dinyatakan sebagai

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \dots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

dengan $T(n)$ adalah kompleksitas waktu penyelesaian persoalan P yang berukuran n , $g(n)$ adalah kompleksitas waktu untuk SOLVE jika n sudah berukuran kecil, $T(n_1) + T(n_2) + \dots + T(n_r)$ sebagai kompleksitas waktu untuk memproses setiap upa-persoalan, dan $f(n)$ adalah kompleksitas waktu untuk COMBINE solusi dari masing-masing upa-persoalan. Tahap DIVIDE dapat dilakukan dalam $O(1)$ sehingga tidak dimasukkan ke dalam formula.

1.3 Algoritma Divide and Conquer dalam Kompresi Gambar

Penulis menerapkan algoritma *divide and conquer* menggunakan bahasa pemrograman Java untuk menyelesaikan proses kompresi gambar menggunakan struktur data Quadtree. Algoritma ini bekerja dengan cara membagi gambar menjadi blok-blok kecil dan menyimpan representasi warna rata-rata pada blok-blok yang dianggap cukup seragam. Berikut adalah langkah-langkah dalam penyelesaiannya:

1. Pengguna memasukkan alamat absolut gambar asli dan program membaca gambar sebagai objek `BufferedImage` dan mencatat ukuran asli gambar (lebar, tinggi, dan ukuran file asli).
2. Program membaca input parameter dari user, dari metode perhitungan error (Variance, MAD, MPD, Entropy, atau SSIM), ambang batas (*threshold*) error, ukuran blok minimum dan target kompresi berdasarkan ukuran file. Jika target kompresi lebih dari 0, program akan mengabaikan input *threshold* dan ukuran blok minimum dan akan menyesuaikan dua parameter tersebut sehingga menghasilkan ukuran akhir gambar sesuai target kompresi dengan toleransi 1%.
3. Program membentuk objek `QuadtreeImageCompressor` (akan dijelaskan lebih lanjut di bab selanjutnya) berdasarkan gambar dan parameter input. Objek ini

menyimpan informasi seperti dimensi gambar, threshold kompresi, dan jenis metode error yang digunakan.

4. Program menjalankan fungsi kompresi berbasis *divide and conquer*. Proses dimulai dengan membagi gambar menjadi satu blok besar. Jika blok tersebut tidak seragam (error lebih besar dari threshold), blok akan dibagi menjadi 4 kuadran. Proses ini dilakukan secara rekursif untuk tiap simpul/blok hingga setiap blok cukup seragam atau ukuran blok mencapai batas minimum.
5. Setiap blok yang sudah tidak dibagi lagi akan disimpan sebagai leaf Quadtree. Node tersebut menyimpan warna rata-rata dan koordinat posisi blok.
6. Jika target kompresi berdasarkan ukuran file digunakan ($\text{target kompresi} > 0\%$), program akan menggunakan pendekatan binary search untuk mencari nilai threshold terbaik yang menghasilkan ukuran file mendekati target dan menghentikan iterasi jika ukuran hasil berada dalam toleransi $\pm 1\%$ atau iterasi sudah mencapai iterasi maksimal, yaitu 100 iterasi. [BONUS]
7. Setelah struktur Quadtree selesai dibangun, program akan membuat kembali gambar terkompresi dengan merepresentasikan setiap node daun sebagai blok warna rata-rata.
8. Program menyimpan hasil gambar terkompresi dalam format yang sama seperti di awal. Jika pengguna ingin output dalam format GIF, maka program akan membentuk animasi yang memperlihatkan proses pembentukan Quadtree dari kedalaman 0 hingga maksimum.
9. Program menampilkan ringkasan hasil kompresi kepada pengguna, mulai dari ukuran file sebelum dan sesudah kompresi, persentase kompresi, waktu eksekusi, kedalaman pohon, dan jumlah simpul pohon.

Untuk memperjelas bagaimana *divide and conquer* ini diimplementasikan dalam proses kompresi gambar, berikut adalah pemetaan tahapan-tahapan utama ke dalam empat bagian utama dari algoritma Divide and Conquer: DIVIDE, CONQUER, SOLVE, dan COMBINE.

Tahapan SOLVE dilakukan ketika suatu blok gambar telah memenuhi syarat sebagai blok yang seragam. Hal ini ditentukan berdasarkan dua kondisi: ukuran blok sudah mencapai ukuran minimum yang ditentukan atau error blok berada di bawah ambang batas error yang ditentukan. Jika salah satu dari kondisi ini terpenuhi, blok tidak akan dibagi lagi. Sebagai gantinya, program menyimpan blok tersebut sebagai simpul daun (leaf node) dalam pohon Quadtree, lengkap dengan informasi posisi, ukuran, dan warna rata-ratanya. Inilah dasar dari proses kompresi: hanya menyimpan informasi minimum dari blok yang tidak mengandung detail yang signifikan.

Jika blok tidak memenuhi kriteria untuk diselesaikan, maka tahapan DIVIDE dilakukan. Pada tahap ini, blok tersebut akan dibagi menjadi empat sub-blok (kuadran) dengan ukuran yang lebih kecil namun proporsional, yaitu: kiri atas (top-left), kanan atas (top-right), kiri bawah (bottom-left), dan kanan bawah (bottom-right). Pembagian ini bertujuan untuk mempersempit cakupan analisis ke area-area yang lebih kecil agar variasi warna dapat diamati lebih akurat.

Setelah seluruh blok atau sub-blok diproses dan dikompresi, hasilnya digabungkan kembali dalam tahap COMBINE. Pada tahap ini, semua blok terkompresi disusun ulang berdasarkan posisi aslinya dalam gambar untuk membentuk gambar hasil akhir yang sudah terkompresi. Dengan demikian, algoritma ini

memanfaatkan prinsip Divide and Conquer secara utuh, mulai dari pemecahan masalah besar, penyelesaian bagian-bagian kecil, hingga penyusunan kembali solusi akhir menjadi solusi dari permasalahan awal, yaitu kompresi gambar.

1.4 Metode Pengukuran Error

Proses kompresi gambar memerlukan suatu kriteria untuk menentukan apakah suatu blok gambar cukup seragam sehingga dapat langsung dikompresi, atau perlu dibagi menjadi sub-blok yang lebih kecil. Kriteria ini ditentukan melalui metode pengukuran error, yaitu teknik evaluasi yang menghitung sejauh mana nilai-nilai piksel dalam suatu blok menyimpang dari representasi yang disederhanakan, seperti warna rata-rata. Nilai error yang diperoleh kemudian dibandingkan dengan nilai ambang batas (threshold) tertentu sebagai dasar pengambilan keputusan dalam proses pembagian atau penyelesaian blok. Setiap metode pengukuran error memiliki range threshold error dan cara kerja masing-masing.

1.4.1 Variance

Metode variance menilai kualitas gambar hanya dari varians dari masing-masing kanal warna gambar tersebut. Varians mengukur seberapa besar penyebaran nilai pixel terhadap rata-rata dalam suatu blok gambar. Semakin tinggi nilai varians, semakin besar perbedaan warna dalam blok tersebut. Sebaliknya, nilai varians yang rendah mengindikasikan bahwa warna dalam blok relatif seragam dan tidak memiliki banyak variasi.

Dalam implementasinya, varians dihitung secara terpisah untuk masing-masing kanal RGB menggunakan formula ini:

$$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$$

Dimana σ_c^2 adalah variansi tiap kanal warna c (R, G, B) dalam satu blok, $P_{i,c}$ adalah nilai piksel pada posisi i untuk kanal warna c, μ_c adalah nilai rata-rata tiap piksel dalam satu blok, dan N adalah banyaknya piksel dalam satu blok. Nilai error akhir merupakan rata-rata dari ketiga varians tersebut. Nilai ini kemudian dibandingkan dengan nilai threshold yang telah ditentukan oleh pengguna.

Range nilai threshold dari metode variance adalah 0 sampai 16256.25. Nilai varians maksimum didapatkan ketika setengah dari piksel memiliki nilai intensitas minimum(0) dan setengahnya lagi memiliki nilai intensitas maksimum(255) sehingga rata-ratanya adalah 127.5. Hasil kuadrat dari nilai ekstrem dikurangi dengan rata-rata akan menghasilkan 16256.25.

1.4.2 Mean Absolute Difference (MAD)

Metode Mean Absolute Difference (MAD) menghitung seberapa jauh, secara rata-rata, nilai-nilai piksel menyimpang dari rata-rata blok, tetapi tanpa memperbesar pengaruh nilai-nilai ekstrem seperti pada metode Variance. Nilai MAD rendah berarti blok seragam, sebaliknya nilai MAD tinggi berarti blok bervariasi dan perlu dibagi lagi. Karena tidak ada pemangkatan, MAD lebih ringan secara komputasi dan hasilnya lebih stabil terhadap fluktuasi kecil pada nilai piksel.

Sama seperti Variance, MAD dihitung secara terpisah untuk masing-masing kanal RGB. Formula MAD adalah sebagai berikut:

$$MAD_c = \frac{1}{N} \sum_{i=1}^N |P_{i,c} - \mu_c|$$

dimana $P_{i,c}$ adalah nilai piksel pada posisi i untuk kanal warna c, μ_c adalah nilai rata-rata tiap piksel dalam satu blok, dan N adalah banyaknya piksel dalam satu blok. Nilai error akhir merupakan rata-rata dari ketiga varians tersebut. Nilai ini kemudian dibandingkan dengan nilai threshold yang telah ditentukan oleh pengguna.

Dalam gambar 8-bit (0–255), selisih maksimum antara nilai piksel dengan rata-ratanya terjadi jika setengah piksel bernilai 0 dan setengahnya lagi 255, maka rata-rata 127.5. Setiap piksel memiliki selisih 127.5 dari rata-rata karena seluruh piksel memiliki selisih maksimum, maka nilai MAD maksimum juga 127.5 dan minimumnya adalah 0.

1.4.3 Max Pixel Difference (MPD)

Metode Max Pixel Difference (MPD) adalah metode pengukuran error yang sangat sederhana dan langsung dengan mengetahui seberapa besar perbedaan nilai piksel paling ekstrem dalam satu blok. Metode ini cukup mencari nilai piksel tertinggi dan terendah dalam satu blok, lalu menghitung selisihnya. Nilai MPD dihitung pada semua kanal warna RGB dan digabungkan dengan bobot yang sama.

$$MPD = \max(P_i) - \min(P_i)$$

Metode MPD memiliki rentang threshold dari 0 sampai 255 karena nilai maksimum dari selisih nilai piksel pada kanal 8 bit adalah 255.

1.4.4 Entropy

Metode Entropy digunakan untuk mengukur tingkat ketidakteraturan atau ketidakpastian dalam distribusi nilai warna dalam suatu blok gambar. Semakin tidak seragam sebaran nilai piksel dalam blok, semakin tinggi nilai entropy-nya. Dalam konteks kompresi gambar, entropy digunakan untuk menilai seberapa kompleks atau acak distribusi warna dalam suatu blok, sehingga dapat digunakan untuk menentukan apakah blok tersebut dapat disederhanakan atau perlu diproses lebih lanjut.

Entropy diukur berdasarkan histogram nilai piksel dalam satu blok. Histogram ini menunjukkan frekuensi kemunculan masing-masing nilai warna (R, G, dan B), kemudian dikonversi menjadi distribusi probabilitas. Dari distribusi ini, nilai entropy dihitung menggunakan rumus Shannon entropy:

$$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$$

dimana $P_c(i)$ adalah probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B). Nilai akhir diambil dari rata-rata semua entropi kanal RGB.

Entropy maksimum, dalam kanal tunggal, terjadi saat semua 256 nilai muncul sama banyak sehingga entropy maksimum adalah $\log_2(256) = 8$ dan nilai terendahnya adalah 0.

1.4.5 Structural Similarity Index (SSIM)

Metode Structural Similarity Index (SSIM) merupakan pendekatan pengukuran error yang berbasis persepsi visual manusia. Berbeda dengan metode statistik seperti Variance dan MAD yang hanya menghitung perbedaan nilai piksel secara matematis,

SSIM mempertimbangkan bagaimana manusia melihat struktur dan detail dalam gambar. Oleh karena itu, SSIM dianggap lebih representatif dalam menilai kualitas visual gambar hasil kompresi.

SSIM mengevaluasi kemiripan struktur antara dua blok gambar, yaitu blok asli dan blok yang direpresentasikan secara sederhana (misalnya, dengan warna rata-rata). Penilaian dilakukan berdasarkan tiga komponen utama:

1. Luminance (Pencahayaan): Membandingkan nilai rata-rata kedua blok.
2. Contrast (Kontras): Membandingkan variansi di antara dua blok.
3. Structure (Struktur): Membandingkan kovarians antarpiksel di dua blok.

SSIM dapat dirumuskan sebagai berikut:

$$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$$

dimana μ_x , μ_y adalah rata-rata nilai piksel dari blok x dan y, σ_x^2 , σ_y^2 adalah variansi dari blok x dan y, σ_{xy} adalah kovarians antara x dan y, C1 adalah konstanta stabilisasi perbandingan intensitas, dan C2 adalah konstanta stabilisasi perbandingan kontras. Nilai konstanta C1 dan C2 bergantung pada rentang dinamis warna pada gambar dan dapat didefinisikan sebagai berikut

$$\begin{aligned} C_1 &= (K_1 L)^2 \\ C_2 &= (K_2 L)^2 \end{aligned}$$

Dimana L adalah rentang dinamis warna pada gambar (255 untuk kanal warna 8 bit), K1 dan K2 konstanta bernilai $\ll 1$

dengan K1 umumnya 0.01 dan K2 umumnya 0.03. Nilai 0.01 dan 0.03 dipilih karena terbukti secara eksperimen (pada paper [1]) cukup kecil untuk tidak mengganggu hasil, tapi cukup besar untuk menjaga stabilitas numerik.

Dalam sistem kompresi gambar berbasis Quadtree, setiap blok gambar hasil kompresi direpresentasikan sebagai blok warna tunggal, yaitu warna rata-rata dari seluruh piksel dalam blok asli. Karena blok hasil penyederhanaan ini bersifat monoton (semua pikselnya memiliki nilai warna yang sama), maka perhitungan SSIM dapat disederhanakan secara signifikan.

Pada umumnya, perhitungan SSIM membandingkan dua blok gambar penuh, yaitu blok asli dan blok hasil transformasi, masing-masing memiliki nilai rata-rata, variansi, dan kovarians. Namun dalam kasus ini, blok hasil kompresi (blok Y) memiliki variansi 0 karena semua piksel sama dan kovarians 0 karena tidak ada fluktuasi dalam blok kompresi. Blok hasil kompresi juga memiliki nilai rata-rata yang sama dengan blok asli sehingga secara keseluruhan rumus SSIM dapat disederhanakan menjadi

$$SSIM_c(X, Y) = \frac{C_2}{\sigma_x^2 + C_2}$$

Nilai SSIM berada pada rentang 0 hingga 1, dengan 1 menunjukkan dua blok identik secara struktural dan 0 sebaliknya. Karena nilai error SSIM berada dalam rentang 0 hingga 1, maka threshold yang digunakan untuk menentukan pemisahan blok juga berada dalam rentang ini.

Dalam sistem kompresi, nilai error SSIM sering dikonversi menjadi $1 - SSIM$. Hal ini bertujuan untuk menyamakan arah interpretasi dengan metode error lainnya, yaitu semakin tinggi nilai error, semakin buruk blok tersebut, dan semakin besar kemungkinan blok harus dibagi lagi.

BAB II

IMPLEMENTASI PROGRAM DALAM BAHASA JAVA

2.1 App.java

App.java	
Class App adalah class utama yang menjalankan program	
Atribut	
-	
Konstruktor	
-	-
Metode	
main	Metode utama yang menjalankan aplikasi

2.2 QuadtreeNode.java

QuadtreeNode.java
Kelas QuadtreeNode merepresentasikan sebuah node dalam struktur data Quadtree yang digunakan untuk kompresi gambar berbasis blok piksel. Node ini menyimpan informasi tentang posisi, ukuran, warna rata-rata, dan tingkat kesalahan (error) dari area gambar yang diwakilinya. Kelas ini juga dapat membagi dirinya menjadi empat anak (split) jika diperlukan berdasarkan error yang dihitung menggunakan berbagai metode.
Atribut

private int x, y	Koordinat kiri-atas area yang direpresentasikan node.
private int width, height	Ukuran area gambar
private int depth	Kedalaman node dalam pohon
private double error	Nilai error blok gambar
private QuadtreeNode[] children	Array berisi empat node anak jika node ini dibagi
private boolean isLeaf	Penanda apakah node ini adalah daun (tidak dibagi lagi).
private int avgColor	Warna rata-rata blok gambar
private int method	Metode perhitungan error yang digunakan (1-5)
Konstruktor	
public QuadtreeNode(BufferedImage image, int x, int y, int width, int height, int depth, int method)	Konstruktor yang membuat simpul dengan atribut yang telah ditentukan sesuai input dan melakukan perhitungan untuk mendapatkan avgColor dan nilai error
Metode	
public boolean isLeaf()	Memeriksa apakah node merupakan simpul daun atau bukan

public void setLeafTrue()	Meng-set isLeaf menjadi true
public int getX()	Mengembalikan nilai x
public int getY()	Mengembalikan nilai y
public QuadtreeNode[] getChildren()	Mengembalikan children dari node ini
public double getError()	Mengembalikan nilai error blok ini
public int getDepth()	Mengembalikan kedalaman simpul ini
public int getWidth()	Mengembalikan lebar blok ini
public int getHeight()	Mengembalikan tinggi blok ini
public int getArea()	Mengembalikan luas blok ini
public int getAvgColor()	Mengembalikan Warna rata-rata blok gambar
public boolean split(BufferedImage image)	Membagi node menjadi empat bagian jika memungkinkan.
private int computeAvgColor(BufferedImage image, int x, int y, int width, int height)	Menghitung rata-rata warna pixel gambar ini
private double computeError(BufferedImage	Menghitung error dari node ini sesuai dengan metode yang

img, int x, int y, int w, int h, int method)	dipilih
private double computeVarianceError(BufferedImage image, int x, int y, int height, int width)	Metode perhitungan error Variance
private double computeMADError(BufferedImage image, int x, int y, int height, int width)	Metode perhitungan error MAD
private double computeMPDError(BufferedImage image, int x, int y, int height, int width)	Metode perhitungan error MPD
private double computeEntropyError(BufferedImage image, int x, int y, int height, int width)	Metode perhitungan error Entropy
private double computeEntropy(int[] color, int totalPixels)	Menghitung entropi dari distribusi warna
private double computeSSIMError(BufferedImage image, int x, int y, int height, int width)	Metode perhitungan error SSIM

2.3 QuadtreeImageCompressor.java

QuadtreeImageCompressor.java	
<p>Kelas QuadtreeImageCompressor adalah implementasi dari algoritma kompresi gambar berbasis Quadtree, yang mampu mengecilkan ukuran gambar dengan membagi gambar menjadi blok-blok kecil secara rekursif berdasarkan keseragaman warna (menggunakan ambang error atau target ukuran file). Kelas ini mendukung dua mode utama: kompresi berdasarkan ambang error dan kompresi berdasarkan target ukuran file. Ia menggunakan struktur QuadtreeNode untuk membentuk pohon yang merepresentasikan bagian-bagian gambar.</p>	
Atribut	
private static int numOfNodes	Jumlah total node dalam quadtree (statis, dihitung saat proses pembentukan pohon)
private QuadtreeNode root	Node akar quadtree yang mewakili seluruh gambar
private int width, height	Dimensi gambar input
private int minimumBlockSize	Ukuran blok minimum yang tidak akan dibagi lagi
private int maxDepth	Kedalaman terdalam pohon
private float compressionTarget	Target kompresi gambar

<code>private double errorThreshold</code>	Ambang batas error
<code>private String formatName</code>	Format gambar input
<code>private long originalSize</code>	Size original gambar
Konstruktor	
<code>public QuadtreeImageCompressor(</code> <code> BufferedImage image, int</code> <code> minimumBlockSize, float</code> <code> compressionTarget, double</code> <code> errorThreshold, int method,</code> <code> String formatName, long</code> <code> originalSize)</code>	Inisialisasi objek, lalu memilih mode kompresi (berdasarkan error threshold atau ukuran target). Jika <code>compressionTarget == 0</code> , maka langsung gunakan <code>buildTree()</code> , jika tidak, gunakan <code>compressToTargetSize()</code>
Metode	
<code>public void</code> <code>setErrorThreshold(double</code> <code>threshold)</code>	Mengatur nilai ambang error
<code>public int getMaxDepth()</code>	Mengembalikan kedalaman maksimum pohon
<code>public int getNumOfNodes()</code>	Mengembalikan jumlah total node
<code>public String</code> <code>getFormatName()</code>	Mengembalikan format gambar output
<code>private void</code> <code>buildTree(BufferedImage</code> <code>image, QuadtreeNode node)</code>	Membangun pohon quadtree berdasarkan ambang error; membagi blok jika <code>error > threshold</code>

<code>public List<QuadtreeNode> getLeafNodes(int depth)</code>	Mengembalikan node daun pada kedalaman tertentu
<code>private void collectLeafNodes(QuadtreeNo de node, int depth, List<QuadtreeNode> leafNodes)</code>	Rekursif mengumpulkan node-node daun
<code>public BufferedImage createImageFromDepth(int depth)</code>	Membangun ulang gambar berdasarkan node-node daun pada kedalaman tertentu
<code>public void compressToTargetSize(Buffere dImage image, double targetSizeKB, int method)</code>	Mencari ambang error terbaik agar hasil kompresi mendekati ukuran file target dengan metode biner pencarian
<code>private double estimateImageSize(BufferedIm age image)</code>	Mengestimasi ukuran file (dalam KB) dari gambar terkompresi menggunakan ByteArrayOutputStream

2.4 InputHandler.java

InputHandler.java
Kelas InputHandler berfungsi untuk menangani semua input dari pengguna yang dibutuhkan dalam proses kompresi gambar, seperti path file, metode error, threshold, ukuran blok minimum, target kompresi, dan output file. Kelas ini melakukan

validasi terhadap input agar sesuai dengan batasan dan format yang diharapkan. Jika terjadi kesalahan input, kelas ini akan meminta pengguna untuk memasukkan ulang data.

Atribut

<code>private String inputImagePath</code>	Path absolut ke gambar input
<code>private int errorMethod</code>	Metode perhitungan error (1–5)
<code>private double errorThreshold</code>	Nilai ambang batas error sesuai metode
<code>private int minimumBlockSize</code>	Ukuran minimum blok yang bisa diproses oleh quadtree
<code>private float compressionTarget</code>	Target persentase kompresi (0–1)
<code>private String outputImagePath</code>	Path absolut untuk gambar hasil kompresi
<code>private String outputGifPath</code>	Path absolut untuk output animasi GIF (opsional)

Konstruktor

-	-
---	---

Metode

<code>public void readInputs()</code>	Membaca semua input dari pengguna menggunakan Scanner, sekaligus memvalidasi input
---------------------------------------	--

	berdasarkan jenis dan batasannya. Jika ada kesalahan, input akan diminta ulang.
public String getFileType(String path)	Mengembalikan tipe file (ekstensi) dari sebuah path.
getInputImagePath(), getErrorMethod(), getErrorThreshold(), getMinimumBlockSize(), getCompressionTarget(), getOutputImagePath(), getOutputGifPath()	Masing-masing mengembalikan nilai atribut terkait untuk digunakan oleh kelas lain

2.5 OutputHandler.java

OutputHandler.java	
Kelas OutputHandler bertanggung jawab untuk menyimpan hasil kompresi gambar dari objek QuadtreeImageCompressor ke dalam file gambar statis atau animasi GIF. Kelas ini mengambil hasil representasi gambar dari quadtree berdasarkan kedalaman tertentu dan menulisnya ke file output. Ini memungkinkan visualisasi bertahap dari proses kompresi.	
Atribut	-
Konstruktor	

-	-
Metode	
<pre>public void renderAtDepth(QuadtreeImageCompressor compressor, int depth, String outputPath)</pre>	<p>Menyimpan hasil kompresi gambar pada kedalaman tertentu ke path output dalam format yang sesuai.</p> <p>Menggunakan ImageIO.write() untuk menulis file gambar. Jika proses gagal, akan mencetak pesan error.</p>
<pre>public void renderGIF(QuadtreeImageCompressor compressor, String outputGifPath, int delayMs)</pre>	<p>Menghasilkan animasi GIF dari hasil kompresi dengan menulis gambar dari kedalaman 0 hingga maksimum (getMaxDepth()). Menggunakan AnimatedGIFWriter dari library eksternal untuk membuat GIF, dan memberi delay per frame berdasarkan parameter delayMs</p>

BAB III

SOURCE CODE PROGRAM

3.1 Repository Program

Repository Program dapat diakses melalui tautan Github berikut:

https://github.com/kin-ark/Tucil2_13523152

3.2 Pustaka Eksternal

AnimatedGIFWriter - dragon66:

<https://github.com/dragon66/animated-gif-writer>

3.2 Source Code Algoritma Divide and Conquer

QuadtreeNode.java:

```
package com.kinan.imgcompressor.quadtree;

import java.awt.image.BufferedImage;

public class QuadtreeNode {
    private final int x, y, width, height, depth;
    private final double error;
    private QuadtreeNode[] children;
    private boolean isLeaf;
    private final int avgColor;
    private final int method;

    public QuadtreeNode(BufferedImage image, int x, int y, int width, int height, int depth, int method)
    {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.depth = depth;
        this.isLeaf = false;
        this.children = null;
        this.method = method;

        this.avgColor = computeAvgColor(image, x, y, width,
height);
        this.error = computeError(image, x, y, width,
height, method);
    }
}
```

```

    }

    public boolean split(BufferedImage image)
    {
        if (width <= 1 || height <= 1)
        {
            isLeaf = true;
            return false;
        }

        int halfW = width / 2;
        int halfH = height / 2;
        int remW = width - halfW;
        int remH = height - halfH;

        children = new QuadtreeNode[4];
        children[0] = new QuadtreeNode(image, x, y, halfW,
halfH, depth + 1, method);
        children[1] = new QuadtreeNode(image, x + halfW, y,
remW, halfH, depth + 1, method);
        children[2] = new QuadtreeNode(image, x, y + halfH,
halfW, remH, depth + 1, method);
        children[3] = new QuadtreeNode(image, x + halfW, y
+ halfH, remH, depth + 1, method);
        return true;
    }

    private int computeAvgColor(BufferedImage image,
int x, int y, int width, int height)
{
    long r = 0, g = 0, b = 0;
    int area = width * height;

    for (int i = x; i < x + width; i++)
    {
        for (int j = y; j < y + height; j++)
        {
            int pixel = image.getRGB(i, j);
            r += (pixel >> 16) & 0xFF;
            g += (pixel >> 8) & 0xFF;
            b += pixel & 0xFF;
        }
    }

    int avgR = (int) (r/area);
    int avgG = (int) (g/area);
}

```

```

        int avgB = (int) (b/area);
        return (avgR << 16) | (avgG << 8) | avgB;
    }

    private double computeError(BufferedImage img, int
x, int y, int w, int h, int method) {
    switch (method) {
        case 1:
            return computeVarianceError(img, x, y, h,
w);
        case 2:
            return computeMADError(img, x, y, h, w);
        case 3:
            return computeMPDError(img, x, y, h, w);
        case 4:
            return computeEntropyError(img, x, y, h,
w);
        case 5:
            return computeSSIMError(img, x, y, h, w);
        default:
            throw new
IllegalArgumentException("Invalid error method: " +
method);
    }
}

private double computeVarianceError(BufferedImage
image, int x, int y, int height, int width)
{
    int avgR = (avgColor >> 16) & 0xFF;
    int avgG = (avgColor >> 8) & 0xFF;
    int avgB = (avgColor) & 0xFF;

    int area = height * width;

    double errorR = 0;
    double errorG = 0;
    double errorB = 0;

    int maxX = Math.min(x + width, image.getWidth());
    int maxY = Math.min(y + height, image.getHeight());

    for (int i = x; i < maxX; i++)
    {
        for (int j = y; j < maxY; j++)
        {

```

```

        int pixel = image.getRGB(i, j);
        int r = (pixel >> 16) & 0xFF;
        int g = (pixel >> 8) & 0xFF;
        int b = (pixel) & 0xFF;

        errorR += Math.pow((r - avgR), 2);
        errorG += Math.pow((g - avgG), 2);
        errorB += Math.pow((b - avgB), 2);
    }
}

errorR /= area;
errorG /= area;
errorB /= area;

return ((errorR + errorG + errorB) / 3);
}

private double computeMADError(BufferedImage image,
int x, int y, int height, int width)
{
    int avgR = (avgColor >> 16) & 0xFF;
    int avgG = (avgColor >> 8) & 0xFF;
    int avgB = (avgColor) & 0xFF;

    int area = height * width;

    double errorR = 0;
    double errorG = 0;
    double errorB = 0;

    int maxX = Math.min(x + width, image.getWidth());
    int maxY = Math.min(y + height, image.getHeight());

    for (int i = x; i < maxX; i++)
    {
        for (int j = y; j < maxY; j++)
        {
            int pixel = image.getRGB(i, j);
            int r = (pixel >> 16) & 0xFF;
            int g = (pixel >> 8) & 0xFF;
            int b = (pixel) & 0xFF;

            errorR += Math.abs((r - avgR));
            errorG += Math.abs((g - avgG));
            errorB += Math.abs((b - avgB));
        }
    }
}

```

```

        }

    }

    errorR /= area;
    errorG /= area;
    errorB /= area;

    return ((errorR + errorG + errorB) / 3);
}

private double computeMPDError(BufferedImage image,
int x, int y, int height, int width) {
    int minR = 255, minG = 255, minB = 255;
    int maxR = 0, maxG = 0, maxB = 0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            int rgb = image.getRGB(j, i);
            int r = (rgb >> 16) & 0xFF;
            int g = (rgb >> 8) & 0xFF;
            int b = rgb & 0xFF;

            if (r < minR) minR = r;
            if (g < minG) minG = g;
            if (b < minB) minB = b;

            if (r > maxR) maxR = r;
            if (g > maxG) maxG = g;
            if (b > maxB) maxB = b;
        }
    }

    int diffR = maxR - minR;
    int diffG = maxG - minG;
    int diffB = maxB - minB;

    return ((diffR + diffG + diffB) / 3);
}

private double computeEntropyError(BufferedImage
image, int x, int y, int height, int width)
{
    int[] red = new int[256];
    int[] green = new int[256];
    int[] blue = new int[256];
    int totalPixels = width * height;
}

```

```

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                int rgb = image.getRGB(j, i);
                int ri = (rgb >> 16) & 0xFF;
                int gi = (rgb >> 8) & 0xFF;
                int bi = rgb & 0xFF;

                red[ri]++;
                green[gi]++;
                blue[bi]++;
            }
        }

        double redEntropy = computeEntropy(red,
totalPixels);
        double greenEntropy = computeEntropy(green,
totalPixels);
        double blueEntropy = computeEntropy(blue,
totalPixels);

        return ((redEntropy + greenEntropy + blueEntropy) /
3);
    }

    private double computeEntropy(int[] color, int
totalPixels)
{
    double entropy = 0;

    for (int i = 0; i < 256; i++)
    {
        if (color[i] > 0)
        {
            double p = (double) color[i] /
totalPixels;
            entropy -= p * (Math.log(p) /
Math.log(2));
        }
    }

    return entropy;
}

private double computeSSIMError(BufferedImage
image, int x, int y, int height, int width)

```

```

{
    int avgR = (avgColor >> 16) & 0xFF;
    int avgG = (avgColor >> 8) & 0xFF;
    int avgB = (avgColor) & 0xFF;

    double C2 = Math.pow(0.03 * 255, 2);

    int n = width * height;
    double varR = 0, varG = 0, varB = 0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            int rgb = image.getRGB(j, i);
            int r = (rgb >> 16) & 0xFF;
            int g = (rgb >> 8) & 0xFF;
            int b = rgb & 0xFF;

            varR += Math.pow((r - avgR), 2);
            varG += Math.pow((g - avgG), 2);
            varB += Math.pow((b - avgB), 2);
        }
    }

    varR /= n;
    varG /= n;
    varB /= n;

    // SSIM (varY = 0, covar = 0)
    double ssimR = (C2) / (varR + C2);

    double ssimG = (C2) / (varG + C2);

    double ssimB = (C2) / (varB + C2);

    double ssim = (ssimR + ssimG + ssimB) / 3.0;

    return 1 - ssim;
}
}

```

QuadtreeImageCompressor.java:

```

package com.kinan.imgcompressor.quadtree;

import java.awt.Color;

```

```

import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;

public class QuadtreeImageCompressor {
    private static int numOfNodes = 1;
    private QuadTreeNode root;
    private int width, height, minimumBlockSize,
maxDepth;
    private float compressionTarget;
    private double errorThreshold;
    private String formatName;
    private long originalSize;

    public QuadtreeImageCompressor(BufferedImage image,
int minimumBlockSize, float compressionTarget, double
errorThreshold, int method, String formatName, long
originalSize)
    {
        this.width = image.getWidth();
        this.height = image.getHeight();
        this.minimumBlockSize = minimumBlockSize;
        this.maxDepth = 0;
        this.compressionTarget = compressionTarget;
        this.originalSize = originalSize;
        this.root = new QuadTreeNode(image, 0, 0, width,
height, 0, method);
        this.formatName = formatName;
        if (compressionTarget == 0)
        {
            this.errorThreshold = errorThreshold;
            buildTree(image, root);
        }
        else
        {
            this.minimumBlockSize = 1;
            compressToTargetSize(image, (double)
originalSize * (1 - compressionTarget), method);
        }
    }

    public void setErrorThreshold(double threshold)
}

```

```

{
    this.errorThreshold = threshold;
}

private void buildTree(BufferedImage image,
QuadtreeNode node)
{
    if (node.getError() <= errorThreshold || node.getArea()/4 < minimumBlockSize)
    {
        if (node.getDepth() > this.maxDepth)
        {
            this.maxDepth = node.getDepth();
        }
        node.setLeafTrue();
        return;
    }

    if (node.split(image))
    {
        numOfNodes += 4;
        for (QuadtreeNode child : node.getChildren())
        {
            buildTree(image, child);
        }
    }
}

public List<QuadtreeNode> getLeafNodes(int depth)
{
    if (depth > maxDepth)
    {
        throw new IllegalArgumentException("Depth > max tree depth.");
    }

    List<QuadtreeNode> leafNodes = new ArrayList<>();
    collectLeafNodes(root, depth, leafNodes);
    return leafNodes;
}

private void collectLeafNodes(QuadtreeNode node,
int depth, List<QuadtreeNode> leafNodes)
{
    if (node.isLeaf() || node.getDepth() == depth)
    {
}

```

```

        leafNodes.add(node);
    }
    else if (node.getChildren() != null)
    {
        for (QuadtreeNode child: node.getChildren())
        {
            collectLeafNodes(child, depth,
leafNodes);
        }
    }
}

public BufferedImage createImageFromDepth(int
depth) {
    BufferedImage image = new BufferedImage(width,
height, BufferedImage.TYPE_INT_RGB);
    Graphics2D g = image.createGraphics();

    g.setColor(Color.WHITE);
    g.fillRect(0, 0, width, height);

    List<QuadtreeNode> leafNodes = getLeafNodes(depth);
    for (QuadtreeNode node : leafNodes) {
        g.setColor(new Color(node.getAvgColor()));
        g.fillRect(node.getX(), node.getY(),
node.getWidth(), node.getHeight());
    }

    g.dispose();
    return image;
}

public void compressToTargetSize(BufferedImage
image, double targetSizeKB, int method) {
    double low, high;
    switch (method) {
        case 1:
            low = 0;
            high = 16256.5;
            break;
        case 2:
            low = 0;
            high = 127.5;
            break;
        case 3:
            low = 0;

```

```

        high = 255;
        break;
    case 4:
        low = 0;
        high = 8;
        break;
    case 5:
        low = 0;
        high = 1;
        break;
    default:
        throw new AssertionError();
}

double bestThreshold = high;
BufferedImage bestImage = null;

int iteration = 0;
double tolerance = 0.01;

while (iteration < 100 && (high - low) > 0.0001) {
    iteration++;
    double mid = (low + high) / 2.0;

    setErrorThreshold(mid);
    this.root = new QuadtreeNode(image, 0, 0,
width, height, 0, method);
    this.maxDepth = 0;
    buildTree(image, root);
    BufferedImage result =
createImageFromDepth(this.maxDepth);
    double sizeKB = estimateImageSize(result);

    double diff = Math.abs(sizeKB - targetSizeKB);
    if (diff / targetSizeKB <= tolerance) {
        bestThreshold = mid;
        bestImage = result;
        // System.out.printf("Selesai di iterasi
%d\n", iteration);
        break;
    }

    if (sizeKB > targetSizeKB) {
        low = mid;
    } else {
        bestThreshold = mid;
    }
}

```

```
        bestImage = result;
        high = mid;
    }

    // System.out.printf("Iteration %d ->
Threshold: %.4f | Size: %.2f KB | Target: %.2f KB\n",
iteration, mid, sizeKB, targetSizeKB);
}

// System.out.printf("Selected threshold: %.4f
after %d iterations\n", bestThreshold, iteration);
}

private double estimateImageSize(BufferedImage
image)
{
    try (ByteArrayOutputStream baos = new
ByteArrayOutputStream()) {
        ImageIO.write(image, formatName, baos);
        return baos.size() / 1024.0;
    } catch (IOException e) {
        e.printStackTrace();
        return Double.MAX_VALUE;
    }
}
}
```

BAB IV

EKSPERIMENT

4.1 Test Case 1

Variance + Small File Size + No GIF + 0.5 Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\l.PNG
Pilih metode perhitungan error (1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 1
Masukkan ambang batas error [0, 16256.25]: 1
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi (1.0 = 100%, 0 untuk menonaktifkan): 0.5
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\l_1.PNG
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```



Keluaran

Terminal:

```
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\l_1.PNG
Lama Eksekusi: 2144ms
Size Sebelum Dikompres: 171KB
Size Setelah Dikompres: 85KB
Persentase Kompresi: 50.52%
Kedalaman Pohon: 9
Jumlah Simpul: 317201
```

Gambar:



4.2 Test Case 2

MAD + Small File Size + No GIF + 0.5 Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\1.PNG
Pilih metode perhitungan error (1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 2
Masukkan ambang batas error (misal: 127.5): 1
Masukkan ukuran blok kromatik: 1
Masukkan target percepatan kompresi (1.0 = 100%, 0 untuk menonaktifkan): 0.5
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_2.png
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```



Keluaran

Terminal:

```
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_2.png
Lama Eksekusi: 1318ms
Size Sebelum Dikompres: 171KB
Size Setelah Dikompres: 85KB
Persentase Kompresi: 50.48%
Kedalaman Pohon: 9
Jumlah Simpul: 34/993
```

Gambar:



4.3 Test Case 3

MAD + Small File Size + No GIF + 0.5 Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikomprimasi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\1.PNG
Pilih metode perhitungan error (1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 3
Masukkan ambang batas error [0, 255]: 1
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi (1.0 = 100%, 0 untuk menonaktifkan): 0.5
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_3.PNG
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```



Keluaran

Terminal:

```
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_3.PNG
Lama Eksekusi: 2710ms
Size Sebelum Dikompres: 171KB
Size Setelah Dikompres: 73KB
Persentase Kompresi: 57.19%
Kedalaman Pohon: 9
Jumlah Simpul: 828353
```



Gambar:

4.4 Test Case 4

Entropy + Small File Size + No GIF + 0.5 Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\1.PNG
Pilih metode perhitungan error [1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM]: 4
Masukkan angka batas error [0 .. 8]: 1
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi (1.0 = 100%, 0 untuk menonaktifkan): 0.5
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_4.png
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```

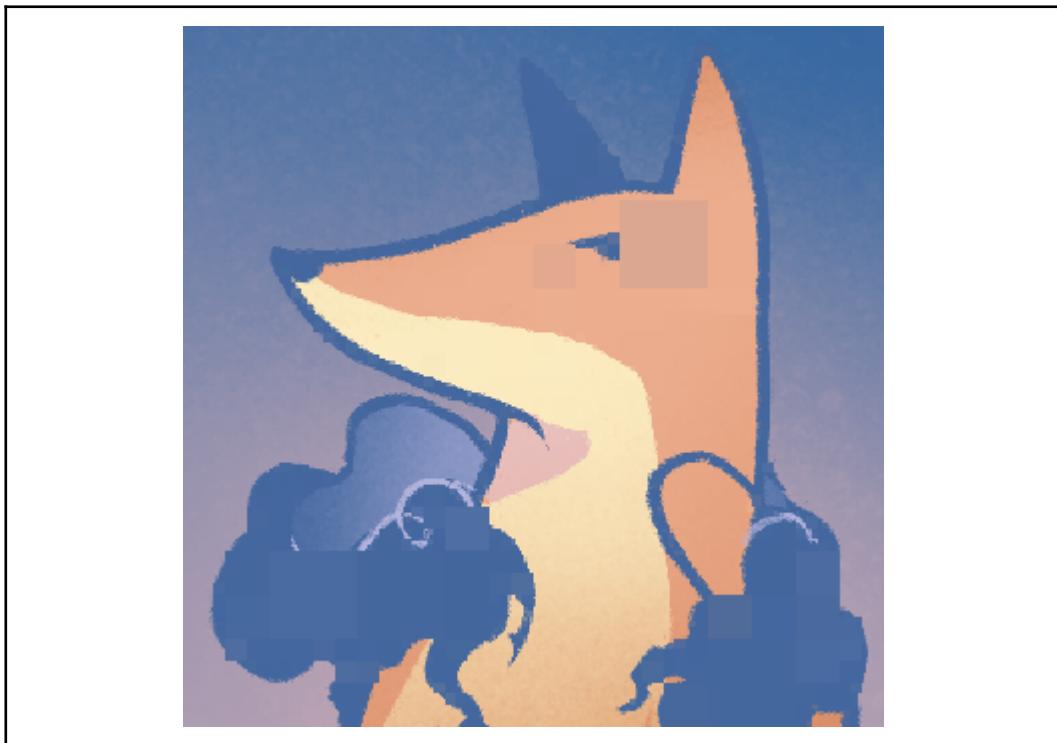


Keluaran

Terminal:

```
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_4.png
Lama Eksekusi: 3573ms
Size Sebelum Dikompres: 171KB
Size Setelah Dikompres: 87KB
Persentase Kompresi: 49.02%
Kedalaman Pohon: 9
Jumlah Simpul: 856481
```

Gambar:



4.5 Test Case 5

SSIM + Small File Size + No GIF + 0.5 Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikomprimasi: C:\Users\Kinan\Documents\Kinan_Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\1.PNG
Pilih metode perhitungan error (1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 5
Masukkan ambang batas error [0, 1]: 0.5
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi (1.0 = 100%, 0 untuk menonaktifkan): 0.5
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_5.PNG
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```



Keluaran

Terminal:

```
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_5.PNG
Lama Eksekusi: 1339ms
Size Sebelum Dikompres: 171KB
Size Setelah Dikompres: 85KB
Persentase Kompresi: 50.48%
Kedalaman Pohon: 9
Jumlah Simpul: 386525
```

Gambar:



4.6 Test Case 6

Variance + Big File Size + No GIF + No Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikomprimasi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\2.jpg
Pilih metode perhitungan error (1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 1
Masukkan ambang batas error [0, 16256.25]: 400
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi (1.0 = 100%, 0 untuk menonaktifkan): 0
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_6.jpg
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```



Keluaran

Terminal:

```
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_6.jpg
Lama Esekusi: 8461ms
Size Sebelum Dikompres: 6496KB
Size Setelah Dikompres: 636KB
Persentase Kompresi: 90.20%
Kedalaman Pohon: 12
Jumlah Simpul: 172249
```

Gambar:



4.7 Test Case 7

MAD + Big File Size + No GIF + No Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\2.jpg
Pilih metode penitungan error [1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM]: 2
Masukkan ambang batas error [0, 127.5]: 16
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi [1, 0 = 100%, 0 untuk nonaktifkan]: 0
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_7.jpg
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```

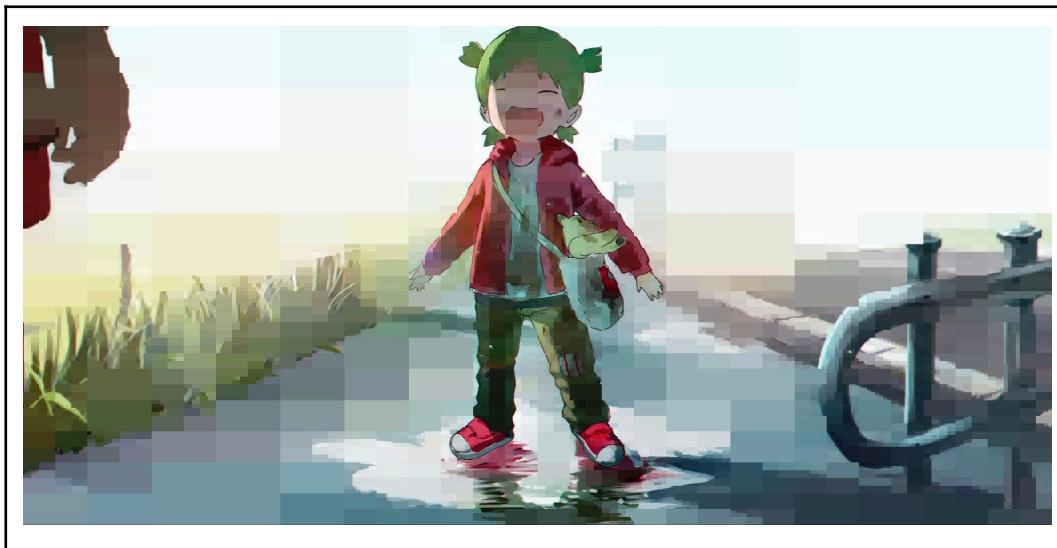


Keluaran

Terminal:

```
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_7.jpg
Lama Esekusi: 8399ms
Size Sebelum Dikompres: 6496KB
Size Setelah Dikompres: 618KB
Persentase Kompresi: 96.48%
Kedalaman Pohon: 12
Jumlah Simpul: 149895
```

Gambar:



4.8 Test Case 8

MPD + Big File Size + No GIF + No Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\2.jpg
Pilih metode perhitungan error (1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 3
Masukkan ambang batas error [0, 255]: 16
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi (1.0 = 100%, 0 untuk menonaktifkan): 0
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_8.jpg
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```



Keluaran

Terminal:

```
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_8.jpg
Lama Eksekusi: 10995ms
Size Sebelum Dikompres: 6496KB
Size Setelah Dikompres: 911KB
persentase Kompresi: 85.97%
Kedalaman Pohon: 12
Jumlah Simpul: 1291977
```

Gambar:



4.9 Test Case 9

Entropy + Big File Size + No GIF + No Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\2.jpg
pilih metode perhitungan error [1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM]: 4
Masukkan ambang batas error [0, 8]: 0.2
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi (1.0 = 100%, 0 untuk menonaktifkan): 0
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_9.jpg
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```



Keluaran

Terminal:

```
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_9.jpg
Lama Eksekusi: 26897ms
Size Sebelum Dikompres: 6496KB
Size Setelah Dikompres: 898KB
Persentase Kompresi: 86.17%
Kedalaman Pohon: 12
Jumlah Simpul: 8922473
```

Gambar:



4.10 Test Case 10

SSIM + Big File Size + No GIF + No Target Compression

Masukan

```
C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\2.jpg
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\2.jpg
Pilih metode perhitungan error (1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 5
Masukkan ambang batas error [0, 1]: 0.2
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi (1.0 = 100%, 0 untuk menonaktifkan): 0
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_10.jpg
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```



Keluaran

Terminal:

```
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_10.jpg
Lama Eksekusi: 10319ms
Size Sebelum Dikompres: 6496KB
Size Setelah Dikompres: 909KB
Persentase Kompresi: 86.00%
Kedalaman Pohon: 12
Jumlah Simpul: 1982541
```

Gambar:



4.11 Test Case 11

MAD + Big File Size + No GIF + 0.5 Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikomprimasi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\2.jpg
Pilih metode perhitungan error (1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 2
Masukkan ambang batas error (0, 127.5): 1
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi (1.0 = 100%, 0 untuk menonaktifkan): 0.5
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_11.jpg
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati):
```



Keluaran

Terminal:

```
Selected threshold: 0.0001 after 21 iterations
Kompresi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\2_11.jpg
Lama Eksekusi: 376617ms
Size Sebelum Dikompres: 6496KB
Size Setelah Dikompres: 897KB
Persentase Kompresi: 86.18%
Kedalaman Pohon: 12
Jumlah Simpul: 134759165
```

Gambar:



4.12 Test Case 12

Variance + Small File Size + GIF + 0.5 Target Compression

Masukan

```
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\test_photo\1.PNG
Pilih metode perhitungan error (1: Variance, 2: Mean Absolute Difference, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 1
Masukkan ukuran blok maksimum: 16256,256]: 1
Masukkan ukuran blok minimum: 1
Masukkan target persentase kompresi (1,0 = 100%, 0 untuk menonaktifkan): 0.5
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_12.png
Masukkan alamat absolut gif (opsional, tekan enter untuk melewati): C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_12.gif
```



Keluaran

Terminal:

```
Selected threshold: 1.7364 after 16 iterations
Kompreksi Selesai!
Alamat absolut gambar yang sudah dikompres: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_12.png
Alamat absolut GIF: C:\Users\Kinan\Documents\Kinan Docs\Kuliah\Jurusan\Semester 4\Stima\Tucil2_13523152\test\1_12.gif
Lama Esekusi: 1880ms
Size Sebelum Dikompres: 171KB
Size Setelah Dikompres: 85KB
Persentase Kompreksi: 50.52%
Kedalaman Pohon: 9
Jumlah Simpul: 317201
```

Gambar:



BAB V

ANALISIS DAN PEMBAHASAN

5.1 Analisis Kompleksitas Waktu dan Ruang Program

Kompleksitas waktu algoritma kompresi gambar menggunakan pendekatan Quadtree dapat dianalisis dengan memodelkan algoritma sebagai proses rekursif yang mengikuti pola Divide and Conquer. Dalam proses ini, suatu blok gambar berukuran $n \times n$ diperiksa terhadap nilai error tertentu. Jika nilai error melebihi threshold, blok tersebut akan dibagi menjadi 4 sub-blok yang masing-masing berukuran $\frac{n}{2} \times \frac{n}{2}$. Proses pembagian ini berlanjut secara rekursif hingga ukuran blok mencapai *minimum block size* atau nilai error telah di bawah threshold.

Model rekurensi dari algoritma ini dapat dituliskan sebagai

$$T(n) = \begin{cases} \mathcal{O}(1), & n \leq t \\ 4T\left(\frac{n}{2}\right) + \mathcal{O}(n^2), & n > t \end{cases}$$

Dalam model tersebut, n adalah panjang sisi blok gambar, t adalah ukuran minimum blok, dan $\mathcal{O}(n^2)$ adalah biaya untuk menghitung nilai error dari blok yang sedang diproses. Karena setiap blok yang memenuhi syarat akan dibagi menjadi empat bagian berukuran setengah, maka rekurensi ini dapat diselesaikan dengan Teorema Master. Berdasarkan parameter $a=4$, $b=2$, dan $f(n)=\Theta(n^2)$, maka kasus yang terjadi adalah Kasus 2 dari Teorema Master, yaitu saat fungsi pembagi setara dengan $n^{\log_b a}$. Oleh karena itu, hasil dari rekurensi ini adalah:

$$T(n) = \Theta(n^2 \log n)$$

Kompleksitas ini merupakan representasi dari jumlah total waktu yang dibutuhkan untuk memproses seluruh blok dalam citra, termasuk perhitungan error dan pembuatan pohon Quadtree. Meskipun pada kasus terbaik tidak semua blok harus dibagi,

bentuk ini menggambarkan kompleksitas pada situasi penuh di mana proses pembagian dilakukan secara maksimal. Adapun dari sisi kompleksitas ruang, seluruh piksel gambar tetap perlu disimpan dalam memori, baik untuk gambar asli maupun gambar hasil kompresi. Selain itu, struktur pohon Quadtree juga memerlukan alokasi memori untuk setiap simpulnya. Dalam kasus terburuk, ketika semua blok dibagi hingga unit terkecil, jumlah simpul dalam pohon dapat mendekati jumlah piksel dalam gambar. Oleh karena itu, kompleksitas ruang algoritma ini adalah:

$$S(n) = \Theta(n^2)$$

Algoritma Quadtree untuk kompresi gambar memiliki kompleksitas yang efisien dan berskala sebanding dengan jumlah piksel dalam gambar. Namun, efisiensi aktualnya sangat bergantung pada distribusi warna dalam gambar dan parameter threshold yang digunakan.

5.2 Analisis Hasil Percobaan

Pada test case 1 sampai dengan test case 5 dilakukan test case menggunakan gambar dengan size relatif kecil. Setiap metode pengukuran error ditest dengan target kompresi 0.5 dan tanpa menghasilkan GIF.

Tabel Hasil Test Case 1 – 5

Metode	Waktu (ms)	Ukuran Akhir	Rasio Kompresi	Depth	Jumlah Simpul
Variance	2144	85KB	50.52%	9	317.201
MAD	1318	85KB	50.48%	9	344.993
MPD	2710	73KB	57.19%	9	828.353

Entropy	3573	87KB	49.02%	9	856.481
SSIM	1339	85KB	50.48%	9	380.525

Secara umum, seluruh metode mampu mencapai target ukuran kompresi 50% dengan tingkat keberhasilan yang tinggi, ditunjukkan oleh ukuran file akhir yang sangat mendekati 85 KB. Waktu eksekusi berbeda antarmetode, dengan MAD dan SSIM tampil sebagai metode tercepat, dan Entropy sebagai yang paling lambat karena kompleksitas perhitungannya.

Metode MPD menghasilkan ukuran file paling kecil, namun dengan jumlah node paling banyak (828.353) – menandakan banyak blok yang tidak dibagi lebih lanjut karena perbedaan maksimum piksel masih dalam batas ambang. Sebaliknya, metode Entropy justru menghasilkan visual paling buruk, meskipun ukuran file masih dalam target. Ini kemungkinan besar disebabkan oleh entropi tidak cukup representatif dalam mendeteksi struktur atau variasi warna dalam blok yang bersifat monoton, menyebabkan banyak bagian dibagi tanpa benar-benar meningkatkan kualitas hasil.

Variance, MAD, dan SSIM menunjukkan hasil yang konsisten baik dari segi ukuran, kualitas visual, dan efisiensi waktu. Untuk gambar dengan ukuran kecil dan karakteristik warna yang cukup lembut, metode Variance, MAD, dan SSIM adalah pilihan yang ideal karena menawarkan keseimbangan antara kualitas visual, ukuran file, dan waktu proses.

Gambar hasil TC6 hingga TC10 memiliki satu kesamaan utama, yaitu ukuran file asli yang besar, yakni 6496 KB. Namun, meskipun ukuran awalnya seragam, hasil kompresi dan karakteristik

prosesnya menunjukkan variasi yang signifikan. Gambar 2_7 dan 2_6 merupakan dua gambar dengan hasil paling efisien. Gambar 2_7 mencapai tingkat kompresi tertinggi sebesar 90,48% dengan hanya 140.805 simpul dan waktu eksekusi 8309 ms. Gambar 2_6 menyusul dengan kompresi 90,20%, simpul sebanyak 172.249, dan waktu 8461 ms. Hal ini mengindikasikan bahwa kedua gambar tersebut memiliki struktur visual yang relatif homogen, sehingga metode quadtree dapat mengelompokkan area besar tanpa perlu banyak pembagian. Namun, kecepatan eksekusi dan kehematan size tersebut harus dibayar dengan kualitas visual yang kurang baik.

Sebaliknya, gambar 2_9 dan 2_10 menunjukkan kompleksitas visual yang jauh lebih tinggi. Gambar 2_9 membutuhkan waktu paling lama untuk diproses, yaitu 26.097 ms, dan menghasilkan 8.922.473 simpul, menandakan bahwa gambar ini memiliki banyak variasi piksel yang memaksa pohon quadtree membagi blok secara berulang. Sementara itu, gambar 2_10 juga cukup kompleks dengan 1.982.541 simpul dan waktu eksekusi 10.319 ms. Gambar 2_8 berada di tengah-tengah spektrum dengan jumlah simpul 1.291.977 dan kompresi 85,97%.

Secara umum, hasil ini menunjukkan bahwa ukuran file awal tidak menjadi indikator utama efisiensi kompresi menggunakan metode quadtree. Justru, tingkat kompleksitas visual gambar—terutama variasi nilai piksel dalam blok-blok kecil—memegang peran penting dalam menentukan jumlah pembagian (simpul), waktu pemrosesan, dan hasil kompresi akhir. Gambar yang memiliki area homogen besar memberikan hasil yang jauh lebih efisien baik dari sisi performa maupun ukuran akhir file.

Hasil pengujian TC1 dengan penambahan fitur pembuatan GIF menunjukkan fenomena yang menarik, yaitu waktu eksekusi total justru lebih cepat dibandingkan saat fitur GIF tidak diaktifkan. Dalam kasus ini, waktu yang dibutuhkan hanya 1880 ms dengan hasil kompresi dari 171 KB menjadi 85 KB (50,52%) dan kedalaman pohon maksimal 9 dengan total simpul sebanyak 317.201. Penambahan fitur GIF sebenarnya tidak memengaruhi proses kompresi inti karena pembuatan GIF dilakukan setelah proses pembentukan pohon Quadtree selesai. GIF hanya mengambil gambar pada setiap kedalaman pohon menggunakan data simpul yang telah tersedia dan tidak melakukan komputasi ulang terhadap pohon atau nilai error. Karena kedalaman pohon hanya sembilan, proses pembuatan frame GIF berlangsung sangat cepat dan ringan. Selain itu, adanya perbedaan waktu ini juga bisa disebabkan oleh variasi sistem selama eksekusi, seperti kondisi cache, optimasi memori oleh JVM, atau aktivitas I/O pada sistem operasi. Oleh karena itu, meskipun secara logis penambahan fitur seharusnya menambah beban, dalam praktiknya proses pembuatan GIF tidak signifikan terhadap waktu total, bahkan bisa tampak lebih cepat karena faktor non-deterministik sistem. Kesimpulannya, fitur GIF dapat dianggap ringan dan tidak berdampak besar pada performa kompresi, terutama ketika jumlah kedalaman pohon relatif rendah.

BAB VI

LAMPIRAN

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
3	Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	✓	
4	Mengimplementasi seluruh metode perhitungan error wajib	✓	
5	[Bonus] Implementasi persentase kompresi sebagai parameter tambahan	✓	
6	[Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	✓	
7	[Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	✓	
8	Program dan laporan dibuat (kelompok) sendiri	✓	

Repository Program: https://github.com/kin-ark/Tucil2_13523152

Daftar Pustaka

Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). *Image Quality Assessment: From Error Visibility to Structural Similarity*. *IEEE Transactions on Image Processing*, 13(4), 600–612.
<https://doi.org/10.1109/TIP.2003.819861>

Munir, R. (2025). *Algoritma Divide and Conquer (Bagian 1)*. Institut Teknologi Bandung. Diakses dari
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)

Fan, S.-K. S., & Chuang, Y.-C. (2013). *An Entropy-based Method for Color Image Registration*. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP-2013)* (pp. 417–421). SCITEPRESS – Science and Technology Publications.

<https://doi.org/10.5220/0004210504170421>

dragon66. (n.d.). *animated-gif-writer*. GitHub. Diakses dari
<https://github.com/dragon66/animated-gif-writer>