

IF2211 – Strategi Algoritma
Kompresi Gambar dengan Metode Quadtree

Laporan Tugas Kecil 2

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma pada Semester 2
Tahun Akademik 2024/2025



Disusun oleh:
Fachriza Ahmad Setiyono (13523162)

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

2024

Daftar Isi

Daftar Isi.....	2
Daftar Gambar.....	4
BAB I	
DESKRIPSI MASALAH.....	5
BAB II	
TEORI SINGKAT.....	7
2.1. Divide and Conquer.....	7
2.1.1. Divide.....	7
2.1.2. Conquer.....	7
2.1.3. Combine / Merge.....	8
2.2. Quadtree.....	8
2.3. Image Quality Assessment (IQA).....	9
2.3.1. Varians.....	10
2.3.2. Mean Absolute Deviation (MAD).....	11
2.3.3. Max Pixel Difference (MPD).....	11
2.3.4. Entropi.....	12
2.3.5. Structural Similarity Index Measure (SSIM).....	13
BAB III	
ALGORITMA PROGRAM.....	15
3.1. Divide.....	15
3.2. Conquer.....	15
3.3. Combine / Merge.....	15
BAB IV	
IMPLEMENTASI PROGRAM.....	16
4.1. image.....	17
4.1.1. ImageData.....	17
4.1.2. errormeasuremethod.....	18
4.1.2.1. ErrorMeasurementMethod.....	19
4.1.2.2. VarianceError.....	19
4.1.2.3. MADError.....	21
4.1.2.4. MPDError.....	21
4.1.2.5. EntropyError.....	23
4.1.2.6. SSIMError.....	24
4.2. quadtree.....	25
4.2.1. ImageQuadTree.....	25
4.2.2. ImageQuadTreeBuilder.....	26
4.2.3. ImageQuadTreeExporter.....	29
4.3. utils.....	31
4.3.1. CrashLogger.....	31
4.3.2. ImageUtil.....	32

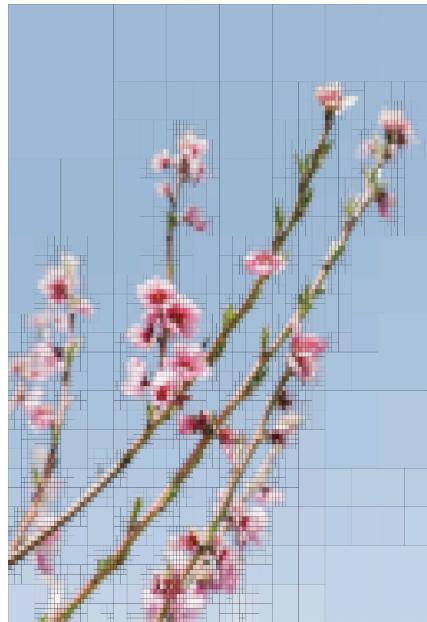
4.3.3. SafeScanner.....	34
4.3.4. TimeProfiler.....	35
4.4. ImageCompressor.....	35
BAB V	
IMPLEMENTASI BONUS.....	41
5.1. Target Persentase Kompresi.....	41
5.2. Structural Similarity Index Measure (SSIM).....	42
5.3. GIF.....	43
BAB VI	
EKSPERIMENT DAN ANALISIS.....	45
6.1. Pengujian Program.....	45
6.2. Analisis Algoritma Program.....	50
6.2.1. Analisis Algoritma Pembentukan Quadtree.....	50
6.2.2. Analisis Algoritma Pencarian Biner Nilai Ambang Batas.....	52
6.2.3. Analisis Algoritma Pembuatan GIF.....	52
6.2.4. Kompleksitas Waktu Total Program Beserta Bonus.....	53
Lampiran.....	54
Daftar Pustaka.....	54

Daftar Gambar

Gambar 1.1. Quadtree dalam kompresi gambar.....	5
Gambar 1.2. Proses pembentukan quadtree dalam kompresi gambar (.gif).....	6
Gambar 2.1.1. Proses penyelesaian masalah dengan algoritma Divide and Conquer.....	7
Gambar 2.2.1. Struktur data quadtree dalam ruang 2-dimensi.....	8
Gambar 4.1.1.1. Kelas ImageData: atribut.....	17
Gambar 4.1.1.2. Kelas ImageData: fungsi dan metode.....	18
Gambar 4.1.2.1.1 Kelas ErrorMeasurementMethod.....	19
Gambar 4.1.2.2.1. Kelas VarianceError.....	20
Gambar 4.1.2.3.1. Kelas MADError.....	21
Gambar 4.1.2.4.1. Kelas MPDError.....	22
Gambar 4.1.2.5.1. Kelas EntropyError.....	23
Gambar 4.1.2.5.2. Kelas EntropyError: fungsi getEntropy.....	24
Gambar 4.1.2.6.1. Kelas SSIMError.....	25
Gambar 4.2.1.1. Kelas ImageQuadTree.....	26
Gambar 4.2.2.1. Kelas ImageQuadTreeBuilder: atribut dan getter setter.....	27
Gambar 4.2.2.2. Kelas ImageQuadTreeBuilder: fungsi build.....	28
Gambar 4.2.2.3. Kelas ImageQuadTreeBuilder: kelas BuildQuadTreeAsync.....	29
Gambar 4.2.3.1. Kelas ImageQuadTreeExporter: fungsi terkait pengisian buffer.....	30
Gambar 4.2.3.2. Kelas ImageQuadTreeExporter: fungsi exportImage & exportGIF.....	31
Gambar 4.3.1.1. Kelas CrashLogger.....	32
Gambar 4.3.2.1. Kelas ImageUtil.....	33
Gambar 4.3.3.1. Kelas ImageUtil.....	34
Gambar 4.3.4.1. Kelas SafeScanner.....	35
Gambar 4.4.1. Kelas ImageCompressor: atribut dan konstruktur.....	36
Gambar 4.4.2. Kelas ImageCompressor: getter setter.....	37
Gambar 4.4.3. Kelas ImageCompressor: fungsi terkait kompresi.....	38
Gambar 4.4.4. Kelas ImageCompressor: fungsi main bagian 1.....	39
Gambar 4.4.5. Kelas ImageCompressor: fungsi main bagian 2.....	40
Gambar 5.1.1. Algoritma pencarian biner.....	41
Gambar 5.3.1. Setiap frame GIF, dipetakan ke gambar.....	44
Gambar 5.3.2. Instruksi penggunaan pustaka AnimatedGIFWriter.....	44

BAB I

DESKRIPSI MASALAH



Gambar 1.1. 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 1.2. Proses pembentukan quadtree dalam kompresi gambar (.gif)
(Sumber: https://miro.medium.com/v2/resize:fit:640/format:webp/1*LHD7PsbmbgNBFrYkxyG5dA.gif)

Untuk tugas ini, diminta 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** berikut sebagai user input:

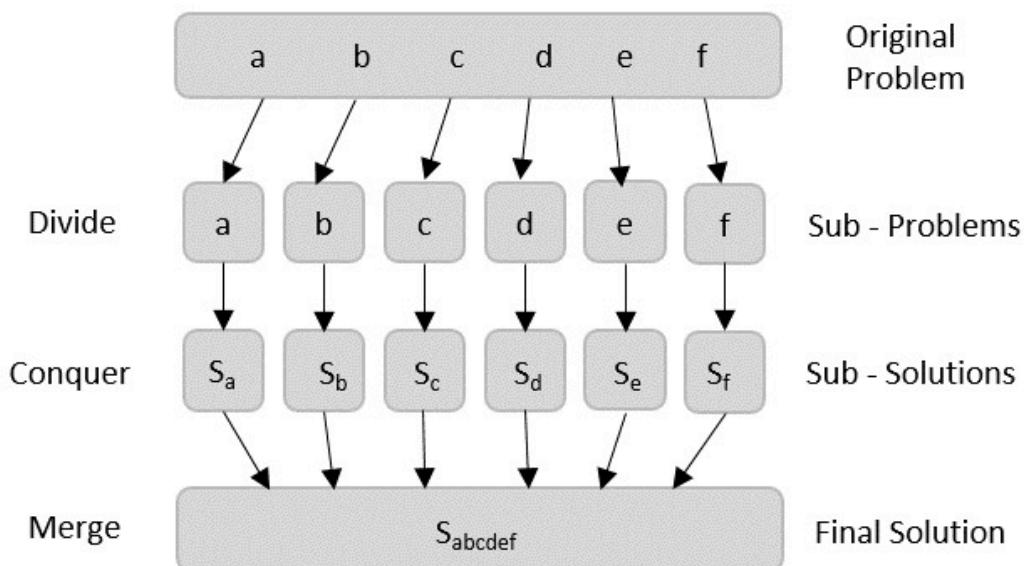
1. [INPUT] **Alamat absolut** gambar yang akan dikompresi.
2. [INPUT] Metode perhitungan galat (dengan penomoran sebagai *input*).
3. [INPUT] Ambang batas sesuai *range* dari metode yang dipilih.
4. [INPUT] Ukuran blok (simpul quadtree) minimum.
5. [INPUT] Target persentase kompresi dengan range 0.0 - 1.0. Nilai 0 berarti mode dinonaktifkan. Jika mode aktif, maka nilai ambang batas akan disesuaikan secara otomatis dan ukuran blok minimum tidak berlaku.
6. [INPUT] **Alamat absolut** gambar hasil kompresi.
7. [INPUT] **Alamat absolut** GIF.
8. [OUTPUT] Waktu eksekusi.
9. [OUTPUT] Ukuran gambar sebelum kompresi.
10. [OUTPUT] Ukuran gambar sesudah kompresi.
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 (*looping*) pada alamat yang sudah ditentukan.

BAB II

TEORI SINGKAT

2.1. Divide and Conquer

Dalam konteks informatika, *divide and conquer* adalah sebuah algoritma dengan karakteristik membagi suatu masalah menjadi beberapa upa-masalah yang berkaitan hingga upa-masalah tersebut cukup kecil untuk diselesaikan secara langsung menjadi suatu upa-solusi, yang nantinya akan dikumpulkan kembali menjadi solusi dari masalah utama. Proses-proses ini dinamakan “Divide”, “Conquer”, dan “Combine/Merge”.



Gambar 2.1.1. Proses penyelesaian masalah dengan algoritma Divide and Conquer
(Sumber:https://www.tutorialspoint.com/data_structures_algorithms/divide_and_conquer.htm)

2.1.1. Divide

Di proses *divide*, persoalan dibagi menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil. Idealnya, setiap upa-persoalan di tingkat yang sama memiliki ukuran relatif sama. Tujuan dari membagi persoalan utama menjadi bagian-bagian yang kecil adalah agar persoalan yang kompleks bisa diselesaikan dengan lebih sederhana.

2.1.2. Conquer

Setelah permasalahan dipecah, maka tiap upa-masalah bisa dicoba untuk diselesaikan agar didapat solusinya. Jika belum cukup kecil, maka permasalahan terus dibagi lagi hingga cukup kecil.

2.1.3. Combine / Merge

Di akhir proses *divide and conquer*, semua upa-solusi yang dihasilkan dari proses sebelumnya digabung untuk membentuk solusi dari permasalahan semula.

Karena tiap upa-masalah bisa dibagi lagi menjadi upa-masalah lainnya yang lebih kecil, maka algoritma *divide and conquer* ini biasanya lebih natural bila direpresentasikan dengan skema rekursif.

Algoritma *divide and conquer* pada umumnya memiliki kompleksitas waktu $T(n)$ yang direpresentasikan sebagai fungsi *piecewise* berikut:

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

Dimana:

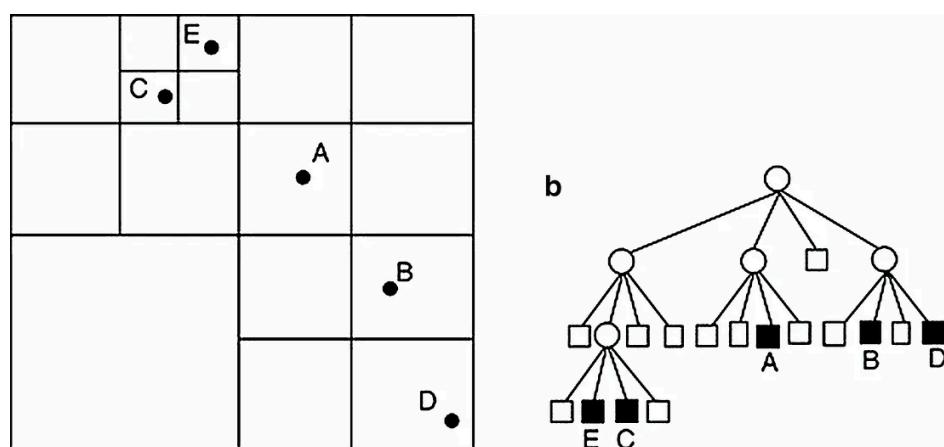
$T(n_i)$: Kompleksitas waktu penyelesaian persoalan ke- i dengan ukuran n .

$g(n)$: Kompleksitas waktu untuk menyelesaikan permasalahan jika n sudah cukup kecil.

$f(n)$: Kompleksitas waktu untuk menggabungkan solusi dari masing-masing upa-persoalan.

2.2. Quadtree

Quadtree adalah sebuah struktur data berbentuk *tree/pohon* dengan jumlah simpul anak sebanyak empat atau nol (jika berupa simpul daun). Kegunaan *quadtree* yang paling umum adalah untuk membagi/mempartisi suatu ruang 2-dimensi menjadi empat sub-ruang 2-dimensi yang setara (juga bisa disebut kuadran) secara rekursif. Walau begitu, bentuk *quadtree* bisa saja tidak sempurna persegi, misal jika ruang 2-dimensi di simpul akar berupa persegi panjang.



Gambar 2.2.1. Struktur data quadtree dalam ruang 2-dimensi
(Sumber: https://www.tutorialspoint.com/data_structures_algorithms/divide_and_conquer.htm)

Dalam konteks kompresi gambar, *quadtree* digunakan untuk menyimpan informasi warna yang telah dinormalisasi untuk merepresentasikan warna rata-rata dalam ruang yang dibatasi oleh simpul terkait. Namun, jika galat pada simpul terlalu besar dari suatu ambang batas, maka simpul akan dibagi lagi menjadi 4 kuadran. Proses ini dilakukan terus-menerus secara rekursif hingga ukuran simpul sudah terlalu kecil atau galat berada di bawah ambang batas.

Pada dasarnya, proses kompresi gambar menggunakan *quadtree* bekerja dengan merepresentasikan warna yang cukup seragam di suatu blok hanya dengan satu nilai warna rata-rata. Hal ini akan mengurangi memori yang digunakan untuk menyimpan data gambar bila disimpan dalam representasi non-*visual*. Dalam kata lain, kompresi gambar *quadtree* memiliki tujuan utama untuk mengurangi representasi warna dalam gambar, **bukan** untuk mengurangi ukuran gambar.

Format gambar pada umumnya seperti PNG atau JPG/JPEG tidak menyimpan seluruh data warna dalam gambar. Format-format tersebut menggunakan algoritma kompresi masing-masing untuk mengurangi ukuran gambar. Alasan mengapa kompresi berbasis *quadtree* dapat mengurangi ukuran gambar bahkan dalam representasi visual adalah karena dengan menyeragamkan warna-warna dalam blok, algoritma kompresi yang digunakan oleh PNG dan JPG/JPEG dapat mengkompresi lebih banyak data. Secara teori, jika algoritma kompresi *quadtree* digunakan dengan tujuan untuk mengurangi ukuran sebuah file *raster* mentah seperti .bmp (bitmap), maka tidak akan ada perubahan pada ukuran gambar karena banyak data yang disimpan tetap sama.

2.3. Image Quality Assessment (IQA)

Tidak ada ukuran formal yang bisa secara akurat menentukan ukuran kualitas gambar. Bahkan, tidak ada definisi formal untuk kualitas gambar. Namun, kualitas gambar dapat dikatakan sebagai tingkat akurasi representasi sinyal yang membentuk gambar. Kualitas gambar dapat terpengaruh oleh beberapa faktor yang mendistorsi sinyal yang akan ditangkap dan/atau ditampilkan suatu media. Dalam konteks IQA, distorsi akibat masalah lensa/optik tidak dipermasalahkan karena masalah tersebut masuk ke permasalahan pemrosesan sinyal. Untuk mempermudah penilaian kualitas gambar, didefinisikan atribut-atribut kualitas gambar berupa ketajaman, *noise*, kontras, artifak visual, distorsi, dan lain-lain.

Metode pengukuran kualitas gambar dibagi menjadi dua yaitu **metode subjektif** dan **metode objektif**. Metode subjektif menggunakan sekumpulan manusia sampel sebagai penilai kualitas gambar. Nilai-nilai tersebut dikumpulkan dan dihitung menjadi nilai kualitas gambar akhir. Namun, karena mengandalkan manusia, metode ini kurang akurat dan konsisten untuk menilai suatu kualitas gambar.

Beragam metode pengukuran gambar objektif menggunakan metode pengukuran atribut gambar yang berbeda-beda, tetapi pada dasarnya memiliki alur proses yang sama, yaitu dengan mengekstrak fitur-fitur gambar dan menghitung skor kualitas gambar. Metode ini dibagi menjadi tiga, yaitu:

- **Full-reference (FR)** – Gambar sampel dikomparasi dengan gambar referensi yang “sempurna”.
- **Reduced-reference (RR)** – Fitur-fitur dari gambar sampel dan referensi diekstrak dan dikomparasi.

- **No-reference (NR)** – Kualitas gambar sampel dinilai tanpa ada gambar lain sebagai referensi.

Dalam tugas ini, metode IQA yang digunakan adalah metode objektif *non-reference* (dan *full-reference* untuk bonus).

2.3.1. Varians

Metode IQA dengan pengukuran varians termasuk ke dalam metode IQA-NR. Kualitas gambar dinilai hanya dari gambar tersebut dengan menghitung varians dari masing-masing kanal warna yang mencakup keseluruhan ruang sampel, kemudian disatukan dengan pembobotan merata.

Nilai varians yang dihasilkan berkorelasi terhadap atribut ketajaman dan detail fitur gambar. Gambar yang “tajam” dan memiliki kontras tinggi umumnya memiliki varians yang tinggi. Sebaliknya gambar yang cenderung memiliki fitur monoton akan memiliki nilai varians yang rendah.

Untuk mengukur nilai varians per kanal warna dari sampel gambar, digunakan rumus berikut:

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

Dimana:

σ_c^2 : Varians kanal warna c .

$P_{i,c}$: Nilai kanal warna c pada piksel ke- i .

μ_c : Nilai rata-rata kanal warna c .

N : Banyak piksel dalam ruang sampel.

Nilai varians dihitung untuk semua kanal warna, lalu digabung dengan bobot yang sama.

Rentang nilai dari metode perhitungan ini dapat dicari dengan menghitung nilai minimum dan maksimum. Nilai maksimum didapat ketika semua sampel berada di nilai terjauh dari nilai rata-rata sampel. Dalam kasus ini, rentang nilai sampel serupa dengan rentang nilai kanal warna 8-bit, yaitu [0, 255]. Jika setengah sampel memiliki nilai minimum dan setengah lainnya memiliki nilai maksimum, maka didapat selisih rerata dan sampel selalu bernilai 127.5, sehingga:

$$\begin{aligned}\sigma_{c max}^2 &= \frac{1}{N} \sum_{i=1}^N 127.5^2 \\ &= 127.5^2 \\ &= 16256.5\end{aligned}$$

Sedangkan nilai minimum didapat ketika semua sampel memiliki nilai yang sama. Karena selisih pasti bernilai 0, maka didapat $\sigma_{c \min}^2$ bernilai 0.

2.3.2. Mean Absolute Deviation (MAD)

Metode IQA dengan pengukuran MAD termasuk ke dalam metode IQA-NR. Metode ini hampir serupa dengan metode pengukuran varians, perbedaan terdapat di tidak adanya pengkuadratan dan juga penggunaan fungsi mutlak nilai selisih sampel dengan rerata. Nilai yang dihasilkan juga menyatakan hal yang serupa dengan metode varians. Hanya saja, metode ini kurang sensitif untuk mendeteksi sampel *outlier*.

Untuk mengukur nilai MAD per kanal warna dari sampel gambar, digunakan rumus berikut:

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

Dimana:

MAD_c : MAD kanal warna c .

$P_{i,c}$: Nilai kanal warna c pada piksel ke- i .

μ_c : Nilai rata-rata kanal warna c .

N : Banyak piksel dalam ruang sampel.

Nilai MAD dihitung untuk semua kanal warna, lalu digabung dengan bobot yang sama.

Serupa dengan metode varians, nilai maksimum didapat jika setengah sampel berada di nilai minimum dan setengah lainnya di nilai maksimum. Didapat nilai selisih selalu bernilai 127.5, sehingga:

$$\begin{aligned} MAD_{c \max} &= \frac{1}{N} \sum_{i=1}^N 127.5 \\ &= 127.5 \end{aligned}$$

Sedangkan nilai minimum didapat ketika semua sampel memiliki nilai yang sama. Karena selisih pasti bernilai 0, maka didapat $MAD_{c \min}$ bernilai 0.

2.3.3. Max Pixel Difference (MPD)

Metode IQA dengan pengukuran MPD termasuk ke dalam metode IQA-NR. Metode ini bisa dibilang merupakan metode ternaif dibanding dengan metode-metode lainnya yang digunakan di sini. Nilai MPD disebut juga sebagai rentang nilai sampel (*dynamic range*). Nilai yang dihasilkan dari metode ini bisa menggambarkan tingkat kontras

atau detail di gambar, atau tidak bisa sama sekali jika gambar memiliki *noise* karena metode perhitungan ini sensitif terhadap *outlier*.

Untuk mengukur nilai MPD per kanal warna dari sampel gambar, digunakan rumus berikut:

$$MPD_c = \max(P_{i,c}) - \min(P_{i,c})$$

Dimana:

MPD_c : Rentang nilai sampel kanal warna c .

$P_{i,c}$: Nilai kanal warna c pada piksel ke- i .

Nilai MPD dihitung untuk semua kanal warna, lalu digabung dengan bobot yang sama.

Nilai maksimum didapat ketika setidaknya ada satu sampel dengan nilai tertinggi, dan satu sampel dengan nilai terendah. Sehingga diperoleh nilai $MPD_{c\ max}$ adalah 255.

Sedangkan nilai terendah terjadi ketika semua sampel memiliki nilai yang sama sehingga didapat $MPD_{c\ min}$ adalah 0.

2.3.4. Entropi

Metode IQA dengan pengukuran entropi warna juga termasuk ke metode IQA-NR. Nilai entropi pada gambar mengukur nilai ketidakaturan pada gambar. Dalam kata lain, metode ini mengukur seberapa tidak beragam dan tidak beraturannya sampel-sampel warna di gambar. Walaupun metode ini bisa mengukur fitur dan detail gambar, namun bisa saja tidak akurat jika gambar memiliki banyak *noise*.

Untuk mengukur nilai entropi per kanal warna dari sampel gambar, digunakan rumus berikut:

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

Dimana:

H_c : Nilai entropi kanal warna c .

$P_c(i)$: Probabilitas sampel warna i dalam kanal warna c .

Nilai entropi dihitung untuk semua kanal warna, lalu digabung dengan bobot yang sama.

Nilai maksimum dari entropi tergantung pada rentang nilai sampel. Untuk kasus ini, sampel ada di rentang [0, 255]. Ketidakaturan tertinggi terjadi jika semua sampel berbeda di rentang tersebut, sehingga memiliki nilai probabilitas yang sama kecil. Contoh, pada gambar gradien dengan luas total 256 piksel tanpa ada nilai piksel yang sama, maka:

$$\begin{aligned}
H_{c \max} &= - \sum_{i=1}^{256} \frac{1}{256} \log_2 \left(\frac{1}{256} \right) \\
&= - (256) \frac{1}{256} \log_2 \left(\frac{1}{256} \right) \\
&= - \log_2 \left(\frac{1}{256} \right) \\
&= 8
\end{aligned}$$

Sedangkan nilai entropi minimum didapat ketika semua sampel memiliki nilai yang sama:

$$\begin{aligned}
H_{c \min} &= - \frac{256}{256} \log_2 \left(\frac{256}{256} \right) \\
&= - \log_2 (1) \\
&= 0
\end{aligned}$$

2.3.5. Structural Similarity Index Measure (SSIM)

Metode IQA yang terakhir merupakan metode IQA-FR. Berbeda dengan metode IQA-NR lainnya, metode ini menggunakan gambar lain sebagai referensi gambar yang “sempurna”. Dalam kata lain, semakin dekat suatu gambar sampel dengan gambar referensi, maka nilai kualitas gambar sampel yang dihasilkan akan lebih tinggi karena mendekati gambar yang “sempurna”.

Fungsi SSIM terdiri dari tiga komponen utama, yaitu fungsi komparasi *luminance* $l(x, y)$, fungsi komparasi kontras $c(x, y)$, dan fungsi komparasi struktur gambar $s(x, y)$. Ketiga fungsi tersebut digabung dalam suatu fungsi kombinasi $f(l(x, y), c(x, y), s(x, y))$ menjadi satu fungsi SSIM per kanal yang berbentuk:

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

Dimana:

$SSIM_c(x, y)$: Nilai SSIM kanal warna c antar blok sinyal x dan y .

$\mu_{x,c}$: Rerata sinyal x di kanal warna c .

$\mu_{y,c}$: Rerata sinyal y di kanal warna c .

$\sigma_{x,c}^2$: Varians sinyal x di kanal warna c .

$\sigma_{y,c}^2$: Varians sinyal y di kanal warna c .

$\sigma_{xy,c}$: Kovarians sinyal x dan y di kanal warna c .

C_1 : Konstanta untuk mencegah ketidakstabilan perbandingan intensitas.

C_2 : Konstanta untuk mencegah ketidakstabilan perbandingan kontras.

Nilai konstanta C_1 dan C_2 bergantung pada rentang dinamis warna pada gambar. Didefinisikan rumus sebagai berikut:

$$C_1 = (K_1 L)^2$$

$$C_2 = (K_2 L)^2$$

Dimana:

L : Rentang dinamis warna gambar. 255 untuk kanal warna 8-bit.

K_1 : Konstanta bernilai $\ll 1$. Umumnya 0.01

K_2 : Konstanta bernilai $\ll 1$. Umumnya 0.03

Setelah didapat nilai SSIM dari semua kanal warna, nilai SSIM digabungkan dengan nilai pembobotan sesuai spesifikasi ruang warna **BT.601-7 / Rec.601-7**:

$$SSIM(x, y) = (0.299)SSIM_R(x, y) + (0.587)SSIM_G(x, y) + (0.114)SSIM_B(x, y)$$

Metode SSIM ini memiliki rentang nilai $[-1, 1]$.

BAB III

ALGORITMA PROGRAM

Program dibagi menjadi beberapa bagian utama, masing-masing dengan algoritma dan tujuannya sendiri. Input program berupa parameter yang telah dijelaskan di bab sebelumnya. Setelah gambar dimuat ke memori program, dimulai proses pembentukan *quadtree* secara rekursif dengan metode *divide and conquer* dengan tujuan untuk menyelesaikan permasalahan kompresi gambar. Komponen-komponen *divide and conquer* dapat dijabarkan sebagai berikut:

3.1. Divide

Pada proses *divide*, program akan membagi *input* gambar menjadi 4 upa-gambar yang (sebisa mungkin) sama besar. Hal ini terus diulang secara rekursif. Proses pembagian ini menunjukkan bahwa permasalahan kompresi pada iterasi tersebut masih belum bisa diselesaikan, sehingga program akan terus membagi gambar dengan tujuan permasalahan kompresi bisa diselesaikan di upa-masalah yang lebih kecil (simpul yang lebih dalam dan lebih kecil).

3.2. Conquer

Dalam permasalahan kompresi gambar, blok gambar akan terus dibagi menjadi 4 kuadran yang lebih kecil. Di setiap pembagian, dilakukan perhitungan nilai galat blok sesuai metode IQA yang dipilih pengguna. Jika nilai galat blok yang dihitung melebihi nilai ambang batas yang telah ditentukan pengguna dan ukuran blok masih besar, maka proses pembagian dilakukan lagi. Pembagian blok menjadi upa-blok ini terus dilakukan secara rekursif hingga ukuran blok kurang dari ukuran blok minimum atau nilai galat blok di bawah nilai ambang batas. Jika pembagian sudah selesai, maka dapat dikatakan bahwa upa-solusi dari upa-masalah tersebut adalah warna hasil normalisasi blok tersebut.

3.3. Combine / Merge

Di akhir proses, program akan melakukan penggabungan seluruh simpul daun dari *quadtree* menjadi gambar yang utuh kembali. Tiap simpul daun memiliki nilai warna yang ternormalisasi (rerata) yang dapat merepresentasikan gambar awal sebelum kompresi. Tiap simpul daun tersebut merepresentasikan upa-solusi dari upa-masalah kompresi gambar. Hasil gambar yang terkompresi tersebut adalah solusi dari permasalahan kompresi gambar di awal.

Selain itu, proses penggabungan juga dilakukan saat pembuatan file GIF. Hanya saja, proses penggabungan simpul di sini tidak harus mencapai simpul daun. Di proses ini, semua simpul di semua kedalaman akan dicetak ke *frame*, yang nantinya akan disatukan menjadi animasi *looping* file GIF.

BAB IV

IMPLEMENTASI PROGRAM

Program diimplementasikan dalam bahasa pemrograman **Java** versi 21. Paradigma pemrograman yang digunakan adalah paradigma berorientasi objek. Seluruh program kompresi gambar berada dalam *package* `com.fachriza.imagequadtree`. Program menggunakan pustaka eksternal untuk membantu pembuatan file GIF. Pustaka yang digunakan adalah “*AnimatedGIFWriter*” milik dragon66.

Secara keseluruhan, berikut adalah struktur kode sumber program:

```
Tucil2_13523162/
├── bin/
├── doc/
│   └── Tucil2_K3_13523162.pdf
├── input/
│   └── ...input images
├── lib/
│   └── AnimatedGIFWriter.jar
├── manifest/
│   └── MANIFEST.MF
└── src/com/fachriza/imagequadtree/
    ├── image/
    │   ├── errormeasuremethod/
    │   │   ├── ErrorMeasurementMethod.java
    │   │   ├── EntropyError.java
    │   │   ├── MADError.java
    │   │   ├── MPDError.java
    │   │   ├── SSIMError.java
    │   │   └── VarianceError.java
    │   └── ImageData.java
    ├── quadtree/
    │   ├── ImageQuadTree.java
    │   ├── ImageQuadTreeBuilder.java
    │   └── ImageQuadTreeExporter.java
    └── utils/
        ├── CrashLogger.java
        ├── ImageUtil.java
        ├── SafeScanner.java
        └── TimeProfiler.java
    └── ImageCompressor.java
└── test/
    └── ...test results
└── build.bat
└── build.sh
└── README.md
```

Di sub-bab berikutnya akan diberikan kode sumber program beserta penjelasan yang dirasa diperlukan. Setiap sub-bab menggambarkan jalur *package* program.

4.1. image

Package ini berkaitan dengan pengolahan data gambar.

4.1.1. ImageData

ImageData adalah kelas yang memegang data-data terkait gambar yang dikompres. Kelas ini juga bertugas untuk merubah gambar *input* menjadi representasi *array of bytes*.



```
package com.fachriza.imagequadtree.image;

import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;

public class ImageData {

    private byte[] redBuffer;
    private byte[] greenBuffer;
    private byte[] blueBuffer;

    public final int width;
    public final int height;
    public final String format;

    ...
}
```

Gambar 4.1.1.1. Kelas ImageData: atribut
(Sumber: Arsip penulis)

Selain data warna di *array of bytes*, ImageData juga menyimpan atribut final dan publik terkait dimensi gambar dan formatnya. Format disimpan agar program bisa mengekspor gambar kembali dengan format yang sama di akhir proses kompresi.



```
public class ImageData {  
    ...  
  
    public ImageData(BufferedImage image, String format) {  
        this.width = image.getWidth();  
        this.height = image.getHeight();  
        this.format = format;  
  
        // Get color array  
        int size = width * height;  
  
        redBuffer = new byte[size];  
        greenBuffer = new byte[size];  
        blueBuffer = new byte[size];  
        byte[] pixelData = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();  
  
        boolean hasAlphaChannel = image.getAlphaRaster() != null;  
        int numberofValues = hasAlphaChannel ? 4 : 3;  
        int valueIndex = hasAlphaChannel ? 1 : 0;  
  
        for (int i = 0; valueIndex + 2 < pixelData.length; valueIndex += numberofValues, i++) {  
            blueBuffer[i] = pixelData[valueIndex];  
            greenBuffer[i] = pixelData[valueIndex + 1];  
            redBuffer[i] = pixelData[valueIndex + 2];  
        }  
    }  
  
    public int getBufferIndex(int x, int y) {  
        return y * width + x;  
    }  
  
    public byte getRed(int x, int y) {  
        return redBuffer[getBufferIndex(x, y)];  
    }  
  
    public byte getGreen(int x, int y) {  
        return greenBuffer[getBufferIndex(x, y)];  
    }  
  
    public byte getBlue(int x, int y) {  
        return blueBuffer[getBufferIndex(x, y)];  
    }  
}
```

Gambar 4.1.1.2. Kelas *ImageData*: fungsi dan metode
(Sumber: Arsip penulis)

Di fungsi konstruktor, *ImageData* akan membaca objek *bufferedImage* dan merubahnya menjadi tiga *array of bytes* berisi informasi warna.

4.1.2. errormeasuremethod

package ini berisi kelas-kelas terkait IQA.

4.1.2.1. ErrorMeasurementMethod

Kelas abstrak untuk semua metode IQA.

```
● ● ●

package com.fachriza.imagequadtree.image.errormeasuremethod;

import com.fachriza.imagequadtree.image.ImageData;

public abstract class ErrorMeasurementMethod {

    protected ImageData imageData;

    public ErrorMeasurementMethod(ImageData imageData) {
        this.imageData = imageData;
    }

    public abstract float getErrorValue(
        float[] mean,
        int x,
        int y,
        int width,
        int height);

    public abstract float getMaxErrorValue();

}
```

*Gambar 4.1.2.1.1 Kelas ErrorMeasurementMethod
(Sumber: Arsip penulis)*

4.1.2.2. VarianceError

Kelas metode IQA varians.

```
● ● ●

package com.fachriza.imagequadtree.image.errormeasuremethod;

import com.fachriza.imagequadtree.image.ImageData;

public class VarianceError extends ErrorMeasurementMethod {

    public VarianceError(ImageData imageData) {
        super(imageData);
    }

    @Override
    public float getErrorValue(
        float[] mean,
        int x,
        int y,
        int width,
        int height) {

        float[] variance = getVariance(mean, x, y, width, height);
        float avgVariance = (variance[0] + variance[1] + variance[2]) / 3;
        return avgVariance;
    }

    @Override
    public float getMaxErrorValue() {
        return 16256.25f;
    }

    protected float[] getVariance(
        float[] mean,
        int x,
        int y,
        int width,
        int height) {

        float[] variance = { 0, 0, 0 };
        int count = width * height;
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                float varR = (imageData.getRed(j, i) & 0xff) - mean[0];
                float varG = (imageData.getGreen(j, i) & 0xff) - mean[1];
                float varB = (imageData.getBlue(j, i) & 0xff) - mean[2];
                variance[0] += (varR * varR);
                variance[1] += (varG * varG);
                variance[2] += (varB * varB);
            }
        }

        for (int i = 0; i < 3; i++) {
            variance[i] /= count;
        }

        return variance;
    }
}
```

Gambar 4.1.2.2.1. Kelas VarianceError
(Sumber: Arsip penulis)

4.1.2.3. MADError

Kelas metode IQA *mean absolute deviation*.



```
● ● ●

package com.fachriza.imagequadtree.image.errormeasuremethod;

import com.fachriza.imagequadtree.image.ImageData;

public class MADError extends ErrorMeasurementMethod {

    public MADError(ImageData imageData) {
        super(imageData);
    }

    @Override
    public float getErrorValue(
        float[] mean,
        int x,
        int y,
        int width,
        int height) {

        float[] absoluteDifference = getAbsoluteDifference(mean, x, y, width, height);
        float avgDifference = (absoluteDifference[0] + absoluteDifference[1] + absoluteDifference[2]) / 3;

        return avgDifference;
    }

    @Override
    public float getMaxErrorValue() {
        return 127.5f;
    }

    protected float[] getAbsoluteDifference(
        float[] mean,
        int x,
        int y,
        int width,
        int height) {

        float[] absoluteDifference = { 0, 0, 0 };
        int count = width * height;
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                absoluteDifference[0] += Math.abs((imageData.getRed(j, i) & 0xff) - mean[0]);
                absoluteDifference[1] += Math.abs((imageData.getGreen(j, i) & 0xff) - mean[1]);
                absoluteDifference[2] += Math.abs((imageData.getBlue(j, i) & 0xff) - mean[2]);
            }
        }

        for (int i = 0; i < 3; i++) {
            absoluteDifference[i] /= count;
        }

        return absoluteDifference;
    }
}
```

Gambar 4.1.2.3.1. Kelas MADError
(Sumber: Arsip penulis)

4.1.2.4. MPDError

Kelas metode IQA *max pixel difference*.

```
● ● ●

package com.fachriza.imagequadtree.image.errormeasuremethod;

import com.fachriza.imagequadtree.image.ImageData;

public class MPDError extends ErrorMeasurementMethod {

    public MPDError(ImageData imageData) {
        super(imageData);
    }

    @Override
    public float getErrorValue(
        float[] mean,
        int x,
        int y,
        int width,
        int height) {

        float[] minMaxDifference = getMinMaxDifference(x, y, width, height);
        float avgDifference = (minMaxDifference[0] + minMaxDifference[1] + minMaxDifference[2]) / 3;
        return avgDifference;
    }

    @Override
    public float getMaxErrorValue() {
        return 255.0f;
    }

    protected float[] getMinMaxDifference(
        int x,
        int y,
        int width,
        int height) {

        float[] minVal = { Float.MAX_VALUE, Float.MAX_VALUE, Float.MAX_VALUE };
        float[] maxVal = { -Float.MAX_VALUE, -Float.MAX_VALUE, -Float.MAX_VALUE };

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                float red = (imageData.getRed(j, i) & 0xff);
                minVal[0] = red < minVal[0] ? red : minVal[0];
                maxVal[0] = red > maxVal[0] ? red : maxVal[0];

                float green = (imageData.getGreen(j, i) & 0xff);
                minVal[1] = green < minVal[1] ? green : minVal[1];
                maxVal[1] = green > maxVal[1] ? green : maxVal[1];

                float blue = (imageData.getBlue(j, i) & 0xff);
                minVal[2] = blue < minVal[2] ? blue : minVal[2];
                maxVal[2] = blue > maxVal[2] ? blue : maxVal[2];
            }
        }

        for (int i = 0; i < 3; i++) {
            maxVal[i] -= minVal[i];
        }
    }

    return maxVal;
}
}
```

Gambar 4.1.2.4.1. Kelas MPDError
(Sumber: Arsip penulis)

4.1.2.5. EntropyError

Kelas metode IQA entropi.

```
● ● ●

package com.fachriza.imagequadtree.image.errormeasuremethod;

import com.fachriza.imagequadtree.image.ImageData;

public class EntropyError extends ErrorMeasurementMethod {

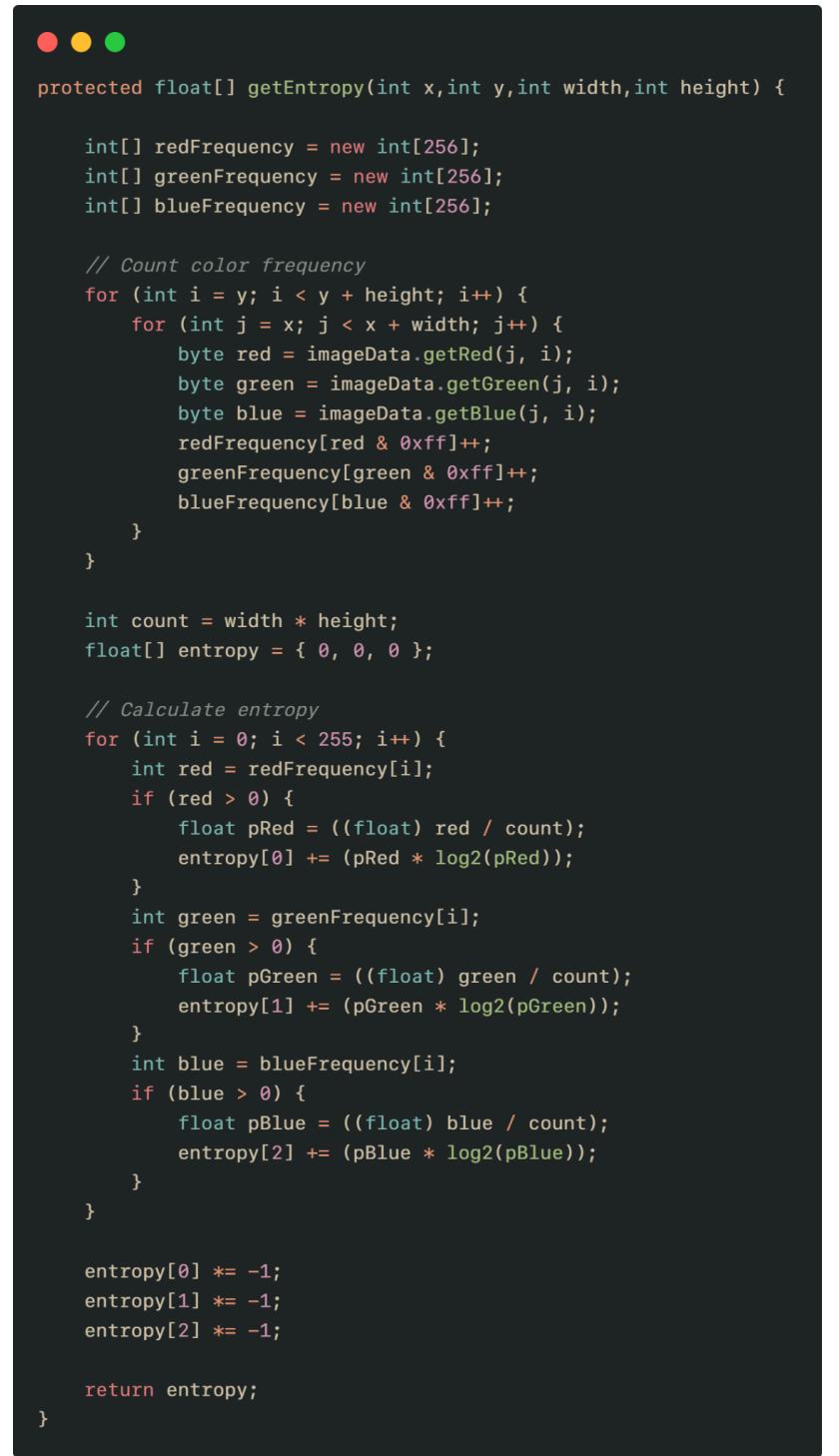
    public EntropyError(ImageData imageData) {
        super(imageData);
    }

    @Override
    public float getErrorValue(
        float[] mean,
        int x,
        int y,
        int width,
        int height) {

        float[] entropy = getEntropy(x, y, width, height);
        float avgEntropy = (entropy[0] + entropy[1] + entropy[2]) / 3;
        return avgEntropy;
    }

    @Override
    public float getMaxErrorValue() {
        // Maximum entropy for 8-bit color channel
        return 8.0f;
    }
    ...
}
```

*Gambar 4.1.2.5.1. Kelas EntropyError
(Sumber: Arsip penulis)*



```
protected float[] getEntropy(int x,int y,int width,int height) {  
  
    int[] redFrequency = new int[256];  
    int[] greenFrequency = new int[256];  
    int[] blueFrequency = new int[256];  
  
    // Count color frequency  
    for (int i = y; i < y + height; i++) {  
        for (int j = x; j < x + width; j++) {  
            byte red = imageData.getRed(j, i);  
            byte green = imageData.getGreen(j, i);  
            byte blue = imageData.getBlue(j, i);  
            redFrequency[red & 0xff]++;  
            greenFrequency[green & 0xff]++;  
            blueFrequency[blue & 0xff]++;  
        }  
    }  
  
    int count = width * height;  
    float[] entropy = { 0, 0, 0 };  
  
    // Calculate entropy  
    for (int i = 0; i < 255; i++) {  
        int red = redFrequency[i];  
        if (red > 0) {  
            float pRed = ((float) red / count);  
            entropy[0] += (pRed * log2(pRed));  
        }  
        int green = greenFrequency[i];  
        if (green > 0) {  
            float pGreen = ((float) green / count);  
            entropy[1] += (pGreen * log2(pGreen));  
        }  
        int blue = blueFrequency[i];  
        if (blue > 0) {  
            float pBlue = ((float) blue / count);  
            entropy[2] += (pBlue * log2(pBlue));  
        }  
    }  
  
    entropy[0] *= -1;  
    entropy[1] *= -1;  
    entropy[2] *= -1;  
  
    return entropy;  
}
```

Gambar 4.1.2.5.2. Kelas EntropyError: fungsi getEntropy
(Sumber: Arsip penulis)

4.1.2.6. SSIMError

Kelas metode IQA SSIM. Kelas ini mengekstensi kelas VarianceError karena metode SSIM memerlukan perhitungan varians blok.

```
● ● ●
package com.fachriza.imagequadtree.image.errormeasuremethod;

import com.fachriza.imagequadtree.image.ImageData;
import com.fachriza.imagequadtree.utils.ImageUtil;

public class SSIMError extends VarianceError {

    public SSIMError(ImageData imageData) {
        super(imageData);
    }

    @Override
    public float getErrorValue(
        float[] mean,
        int x,
        int y,
        int width,
        int height) {
        /*
        * C2 = ((K2)(L))^2
        *
        * where:
        * K2 << 1, 0.03 was recommended
        * L = color dynamic range, 255 for 8-bit channel
        * thus:
        * C2 = ((0.03)(255))^2
        * C2 = 58.5225
        */
        final float C2 = 58.5225f;

        float[] variance = getVariance(mean, x, y, width, height);

        float[] SSIM = {
            C2 / (variance[0] + C2),
            C2 / (variance[1] + C2),
            C2 / (variance[2] + C2)};

        float perceivedSSIM = (SSIM[0] * ImageUtil.SRGB_LUMINANCE_CONSTANTS[0]
            + SSIM[1] * ImageUtil.SRGB_LUMINANCE_CONSTANTS[1]
            + SSIM[2] * ImageUtil.SRGB_LUMINANCE_CONSTANTS[2]);

        // Because SSIM denotes "similarity" and not "error value", we invert the value
        return (1.0f - perceivedSSIM);
    }

    @Override
    public float getMaxErrorValue() {
        return 1.0f;
    }
}
```

Gambar 4.1.2.6.1. Kelas SSIMError
(Sumber: Arsip penulis)

4.2. quadtree

Package ini berkaitan dengan struktur data *quadtree*.

4.2.1. ImageQuadTree

ImageQuadTree adalah kelas yang merepresentasikan struktur data *quadtree*. Struktur kelas cukup sederhana. Atribut terdiri dari warna ternormalisasi dari blok yang dibatasi ruang antar (x, y) dengan $(x + width, y + height)$. Selain fungsi *getter* dan *setter*, ada juga fungsi untuk mengecek apakah simpul merupakan simpul daun dan fungsi untuk menghitung kedalaman pohon.



```
● ● ●
package com.fachriza.imagequadtree.quadtree;

public class ImageQuadTree {

    public final byte[] averageColor;
    public final int x;
    public final int y;
    public final int width;
    public final int height;
    private ImageQuadTree[] children;

    public ImageQuadTree(float[] averageColor, int x, int y, int width, int height) {
        this.averageColor = new byte[] {
            (byte) averageColor[0],
            (byte) averageColor[1],
            (byte) averageColor[2]
        };
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.children = null;
    }

    public ImageQuadTree[] getChildrenArray() {
        return children;
    }

    public void setChildrenArray(ImageQuadTree[] children) {
        this.children = children;
    }

    public boolean isLeafNode() {
        return children == null;
    }

    public int getDepth() {
        if (children != null) {
            int maxDepth = 0;
            for (ImageQuadTree child : children) {
                int thisDepth = child.getDepth();
                maxDepth = thisDepth > maxDepth ? thisDepth : maxDepth;
            }
            return 1 + maxDepth;
        }
        return 1;
    }
}
```

Gambar 4.2.1.1. Kelas ImageQuadTree
(Sumber: Arsip penulis)

4.2.2. ImageQuadTreeBuilder

Kelas ini adalah kelas pembangun *quadtree* gambar. Algoritma *divide and conquer* diimplementasikan di fungsi `build`. Untuk mempercepat komputasi, digunakan pustaka “concurrent” milik Java agar program bisa membangun *quadtree* dengan konkuren (hampir serupa dengan paralel). Berdasarkan eksperimen penulis, ditemukan percepatan komputasi dengan menggunakan metode ini.

```
● ○ ●

package com.fachriza.imagequadtree.quadtree;

import java.util.concurrent.RecursiveTask;
import java.util.concurrent.atomic.AtomicInteger;

import com.fachriza.imagequadtree.image.ImageData;
import com.fachriza.imagequadtree.image.errormeasuremethod.ErrorMeasurementMethod;
import com.fachriza.imagequadtree.utils.ImageUtil;

public class ImageQuadTreeBuilder {

    private ErrorMeasurementMethod emm;
    private ImageData imageData;
    private float threshold;
    private int minimumBlockSize;
    // Use atomic to prevent race condition
    private final AtomicInteger nodeCount;

    public ImageQuadTreeBuilder(
        ErrorMeasurementMethod emm,
        ImageData imageData,
        float threshold,
        int minimumBlockSize) {

        this.emm = emm;
        this.imageData = imageData;
        this.threshold = threshold;
        this.minimumBlockSize = minimumBlockSize;
        this.nodeCount = new AtomicInteger(0);
    }

    public ImageData getImageData() {
        return imageData;
    }

    public int getNodeCount() {
        return nodeCount.get();
    }

    public void resetNodeCount() {
        this.nodeCount.set(0);
    }

    public void setThreshold(float threshold) {
        this.threshold = threshold;
    }
    ...
}
```

Gambar 4.2.2.1. Kelas *ImageQuadTreeBuilder*: atribut dan getter setter
(Sumber: Arsip penulis)

```
public ImageQuadTree build(int x, int y, int width, int height) {
    nodeCount.incrementAndGet();

    /*
     * Always calculate the mean, even if its not a leaf node.
     * We need the mean for error calculation and GIF anyway.
     */
    float[] mean = ImageUtil.getAverageColor(imageData, x, y, width, height);
    ImageQuadTree node = new ImageQuadTree(mean, x, y, width, height);

    int size = width * height;
    if (size < minimumBlockSize)
        return node;

    float error = emm.getErrorValue(mean, x, y, width, height);

    // Calculate children regions.
    int halfLowerWidth = width / 2;
    int halfLowerHeight = height / 2;
    int halfUpperWidth = width - halfLowerWidth;
    int halfUpperHeight = height - halfLowerHeight;
    int halfUpperSize = halfUpperHeight * halfUpperWidth;

    /*
     * Check if future children size are big enough to split.
     * No need to check halfLowerSize.
     */
    if (error > threshold && halfUpperSize >= minimumBlockSize) {
        ImageQuadTree n1, n2, n3, n4;
        if (halfUpperSize > 100000) {

            // do 2 build tasks in other thread if blocks are big enough
            BuildQuadTreeAsync task3 = new BuildQuadTreeAsync(
                x,
                y + halfLowerHeight,
                halfLowerWidth,
                halfUpperHeight);

            BuildQuadTreeAsync task4 = new BuildQuadTreeAsync(
                x + halfLowerWidth,
                y + halfLowerHeight,
                halfUpperWidth,
                halfUpperHeight);

            // build asynchronously
            task3.fork();
            task4.fork();

            // do 2 build task in current working thread
            n1 = build(x, y, halfLowerWidth, halfLowerHeight);

            n2 = build(x + halfLowerWidth, y, halfUpperWidth, halfLowerHeight);

            n3 = task3.join();
            n4 = task4.join();
        } else {
            // else do all build task in current working thread
            n1 = build(x, y, halfLowerWidth, halfLowerHeight);

            n2 = build(x + halfLowerWidth, y, halfUpperWidth, halfLowerHeight);

            n3 = build(x, y + halfLowerHeight,
                       halfLowerWidth, halfUpperHeight);

            n4 = build(x + halfLowerWidth, y + halfLowerHeight,
                       halfUpperWidth, halfUpperHeight);
        }
        ImageQuadTree[] children = { n1, n2, n3, n4 };
        node.setChildrenArray(children);
    }
    return node;
}
```

Gambar 4.2.2.2. Kelas *ImageQuadTreeBuilder*: fungsi *build*
(Sumber: Arsip penulis)

Agar bisa menjalankan fungsi secara asinkron, diperlukan kelas *wrapper* yang mengekstensi kelas `RecursiveTask<T>` milik Java, dengan T adalah kelas `ImageQuadTree`.

```
● ● ●

private class BuildQuadTreeAsync extends RecursiveTask<ImageQuadTree> {

    private int x;
    private int y;
    private int width;
    private int height;

    public BuildQuadTreeAsync(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

    @Override
    protected ImageQuadTree compute() {
        return build(x, y, width, height);
    }
}
```

Gambar 4.2.2.3. Kelas *ImageQuadTreeBuilder*: kelas `BuildQuadTreeAsync`
(Sumber: Arsip penulis)

4.2.3. ImageQuadTreeExporter

Kelas ini bertugas untuk merekonstruksi struktur data *quadtree* menjadi gambar, sesuai format gambar *input* yang disimpan di `ImageData`. Selain mengekspor menjadi gambar, kelas ini juga memiliki fungsi untuk mengekspor GIF.

```
● ● ●
package com.fachriza.imagequadtree.quadtree;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.OutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.imageio.ImageIO;

import com.fachriza.imagequadtree.image.ImageData;
import com.fachriza.imagequadtree.utils.ImageUtil;

import com.github.dragon66.AnimatedGIFWriter;

public class ImageQuadTreeExporter {

    private static void fillImageBuffer(
        int[] buffer,
        ImageQuadTree node,
        ImageData imageData,
        int iteration,
        int targetDepth) {

        boolean isTargetIteration = iteration == targetDepth;

        if (node.isLeafNode() || isTargetIteration) {
            int imageWidth = imageData.width;
            int imageHeight = imageData.height;
            int heightBound = node.y + node.height;
            int widthBound = node.x + node.width;

            int packed = ImageUtil.pack24BitColors(node.averageColor);

            for (int i = node.y; i <= heightBound && i < imageHeight; i++) {
                for (int j = node.x; j <= widthBound && j < imageWidth; j++) {
                    buffer[i * imageWidth + j] = packed;
                }
            }
            return;
        }

        int nextIteration = ++iteration;
        for (ImageQuadTree child : node.getChildrenArray()) {
            fillImageBuffer(buffer, child, imageData,
                nextIteration, targetDepth);
        }
    }

    public static int[] getCompressedImageBuffer(
        ImageQuadTree root,
        ImageData imageData,
        int targetDepth) {

        int[] compressedImageBuffer = new int[imageData.width * imageData.height];
        fillImageBuffer(compressedImageBuffer, root, imageData, 1, targetDepth);

        return compressedImageBuffer;
    }

    ...
}
```

Gambar 4.2.3.1. Kelas ImageQuadTreeExporter: fungsi terkait pengisian buffer
(Sumber: Arsip penulis)



```
...
public static void exportImage(
    ImageQuadTree iqt,
    ImageQuadTreeBuilder builder,
    File outputFile)
throws IOException {

    ImageData imageData = builder.getImageData();

    String format = imageData.format;

    int width = imageData.width;
    int height = imageData.height;

    int[] buffer = getCompressedImageBuffer(iqt, imageData, 0);
    BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);

    image.setRGB(0, 0, width, height, buffer, 0, width);

    ImageIO.write(image, format, outputFile);
}

public static void exportGIF(
    ImageQuadTree iqt,
    ImageQuadTreeBuilder builder,
    File outputFile)
throws Exception {

    ImageData imageData = builder.getImageData();

    int width = imageData.width;
    int height = imageData.height;

    AnimatedGIFWriter writer = new AnimatedGIFWriter(false);
    OutputStream os = new FileOutputStream(outputFile);
    writer.prepareForWrite(os, -1, -1);

    int depth = iqt.getDepth();
    BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    for (int i = 1; i <= depth; i++) {
        int[] buffer = getCompressedImageBuffer(iqt, imageData, i);
        image.setRGB(0, 0, width, height, buffer, 0, width);
        writer.writeFrame(os, image, 1000);
    }

    writer.finishWrite(os);
}
}
```

Gambar 4.2.3.2. Kelas *ImageQuadTreeExporter*: fungsi *exportImage* & *exportGIF*
(Sumber: Arsip penulis)

4.3. utils

Package ini berkaitan dengan kelas-kelas utilitas yang berisi fungsi pembantu jalan program..

4.3.1. CrashLogger

Kelas untuk menulis pesan *error* dari *stack* ke file .log, sehingga tidak mengotori CLI.



```
package com.fachriza.imagequadtree.utils;

import java.io.*;
import java.net.URISyntaxException;
import java.nio.file.*;

public class CrashLogger {
    public static void logException(Exception e) {
        try {
            // Get the root folder
            Path jarDir = Paths.get(CrashLogger.class
                .getProtectionDomain()
                .getCodeSource()
                .getLocation()
                .toURI())
                .getParent()
                .getParent();

            Path logDir = jarDir.resolve("log");
            Files.createDirectories(logDir);

            Path logFile = logDir.resolve("crash.log");

            try (PrintWriter pw = new PrintWriter(new FileWriter(logFile.toFile(), true))) {
                pw.println("---- CRASH LOG ----");
                pw.println("Timestamp: " + java.time.LocalDateTime.now());
                e.printStackTrace(pw);
                pw.println();
            }

            } catch (IOException | URISyntaxException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Gambar 4.3.1.1. Kelas CrashLogger
(Sumber: Arsip penulis)

4.3.2. ImageUtil

Kelas terkait pemrosesan gambar. Berisi konstanta *luminance* RGB standar ruang warna BT.601-7 / Rec.601-7, fungsi *packing* warna, serta fungsi untuk mendapatkan warna rata-rata dari blok gambar.

```
● ○ ●
package com.fachriza.imagequadtree.utils;

import java.io.File;
import java.io.IOException;

import com.fachriza.imagequadtree.image.ImageData;

public class ImageUtil {

    // Constants in REC. 601-7 / BT.601-7 colorspace
    public static final float[] SRGB_LUMINANCE_CONSTANTS = { 0.299f, 0.587f, 0.114f };

    public static int pack24BitColors(byte[] colors) {
        int packed = (0b11111111 << 24)
                    | ((colors[0] & 0xff) << 16)
                    | ((colors[1] & 0xff) << 8)
                    | ( colors[2] & 0xff );
        return packed;
    }

    public static float[] getAverageColor(
        ImageData imageData,
        int x,
        int y,
        int width,
        int height) {

        float[] sum = { 0, 0, 0 };
        int count = width * height;
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                sum[0] += (imageData.getRed(j, i) & 0xff);
                sum[1] += (imageData.getGreen(j, i) & 0xff);
                sum[2] += (imageData.getBlue(j, i) & 0xff);
            }
        }

        for (int i = 0; i < 3; i++) {
            sum[i] /= count;
        }

        return sum;
    }

    public static String getFormatName(File file) throws IOException {
        String fileName = file.getName().toLowerCase();
        if (fileName.endsWith(".jpg") || fileName.endsWith(".jpeg"))
            return "jpeg";
        if (fileName.endsWith(".png"))
            return "png";
        if (fileName.endsWith(".gif"))
            return "gif";
        return null;
    }
}
```

Gambar 4.3.2.1. Kelas ImageUtil
(Sumber: Arsip penulis)

4.3.3. SafeScanner

Kelas untuk mendapatkan *input* pengguna secara aman dengan memaksa pengguna untuk memasukkan *input* sesuai tipe objek yang diminta serta batasannya.



```
● ● ●
package com.fachriza.imagequadtree.utils;

import java.util.InputMismatchException;
import java.util.Scanner;

public class SafeScanner implements AutoCloseable {

    private Scanner scanner;

    public SafeScanner(Scanner scanner) {
        this.scanner = scanner;
        scanner.useDelimiter("\s+");
    }

    @Override
    public void close() {
        if (scanner != null) {
            scanner.close();
        }
    }

    public <T> T getInput(String prompt, Class<T> type) {
        while (true) {
            try {
                System.out.print(prompt + ": ");

                if (type == Integer.class) {
                    return type.cast(scanner.nextInt());
                } else if (type == Double.class) {
                    return type.cast(scanner.nextDouble());
                } else if (type == Boolean.class) {
                    return type.cast(scanner.nextBoolean());
                } else if (type == String.class) {
                    return type.cast(scanner.next());
                } else {
                    throw new IllegalArgumentException("Tipe data tidak dikenal!");
                }

            } catch (InputMismatchException e) {
                System.out.println("Input invalid. Masukkan " + type.getSimpleName() + "!");
                scanner.nextLine();
            }
        }
    }

    public <T extends Number> T getBoundedInput(String prompt, Class<T> type, T min, T max) {
        while (true) {
            try {
                System.out.print(prompt + " (" + min + " - " + max + "): ");

                Number value;
                if (type == Integer.class) {
                    value = scanner.nextInt();
                } else if (type == Double.class) {
                    value = scanner.nextDouble();
                } else if (type == Long.class) {
                    value = scanner.nextLong();
                } else if (type == Float.class) {
                    value = scanner.nextFloat();
                } else {
                    throw new IllegalArgumentException("Tipe numerik tidak disupport!");
                }

                if (value.doubleValue() < min.doubleValue() || value.doubleValue() > max.doubleValue()) {
                    System.out.println("Input harus berada di antara " + min + " dan " + max + "!");
                } else {
                    return type.cast(value);
                }

            } catch (InputMismatchException e) {
                System.out.println("Input invalid. Masukkan " + type.getSimpleName() + "!");
                scanner.nextLine();
            }
        }
    }
}
```

Gambar 4.3.3.1. Kelas ImageUtil
(Sumber: Arsip penulis)

4.3.4. TimeProfiler

Kelas untuk mencatat waktu eksekusi dari bagian-bagian (*sections*) proses program. Memiliki fungsi untuk mencetak detail dan total waktu ke CLI..



```
● ● ●
package com.fachriza.imagequadtree.utils;

import java.util.ArrayDeque;
import java.util.Deque;

public class TimeProfiler {

    private class Section {

        private String title;
        private float time;
        private long tempTime;

        public Section(String title) {
            this.title = title;
            tempTime = System.nanoTime();
        }

        public void stop() {
            time = (System.nanoTime() - tempTime) * 1e-6f;
        }
    }

    private Deque<Section> sections;

    public TimeProfiler() {
        sections = new ArrayDeque<Section>();
    }

    public void startSection(String sectionTitle) {
        sections.offerLast(new Section(sectionTitle));
    }

    public void stopSection() {
        sections.peekLast().stop();
    }

    public void print() {
        // Find lengthiest title
        int maxLength = 0;
        for (Section section : sections) {
            int length = section.title.length();
            maxLength = length > maxLength ? length : maxLength;
        }

        // Add some padding
        maxLength += 2;

        // Print column title
        System.out.format("%" + (maxLength / 2 + 3) + "s", "Proses");
        System.out.format("%" + (8 + maxLength / 2) + "s%n", "Waktu");

        float totalTime = 0.0f;
        for (Section section : sections) {
            System.out.format("%-" + maxLength + "s : ", section.title);
            System.out.format("%+11.2f ms%n", section.time);
            totalTime += section.time;
        }

        // Print total execution time
        System.out.format("%" + maxLength + "s : ", "Total Waktu");
        System.out.format("%+11.2f ms%n", totalTime);
    }
}
```

Gambar 4.3.4.1. Kelas SafeScanner
(Sumber: Arsip penulis)

4.4. ImageCompressor

Program utama kompresi gambar. Memiliki fungsi statis `main` sebagai *entry point* program. Di fungsi `rebuild`, algoritma pencarian biner untuk mencari nilai ambang batas diimplementasikan.

```
● ● ●

package com.fachriza.imagequadtree;

import java.io.File;
import java.io.IOException;
import java.util.Scanner;

import javax.imageio.ImageIO;

import com.fachriza.imagequadtree.image.ImageData;
import com.fachriza.imagequadtree.image.errormeasuremethod.*;
import com.fachriza.imagequadtree.quadtree.*;
import com.fachriza.imagequadtree.utils.*;

public class ImageCompressor {

    private float threshold;
    private int minimumBlockSize;
    private float compressionLevel;

    private ImageQuadTree root;
    private ImageQuadTreeBuilder builder;
    private ErrorMeasurementMethod emm;

    private long fileSize;
    private File outputFile;

    private ImageData imageData;

    public ImageCompressor(File inputFile) throws IOException {
        fileSize = inputFile.length();
        imageData = new ImageData(ImageIO.read(inputFile), ImageUtil.getFormatName(inputFile));
    }
    ...
}
```

*Gambar 4.4.1. Kelas ImageCompressor: atribut dan konstruktor
(Sumber: Arsip penulis)*

```
● ● ●
public class ImageCompressor {
    ...
    public ImageCompressor setMethod(int method) {
        switch (method) {
            case 1:
                emm = new VarianceError(imageData);
                break;
            case 2:
                emm = new MADError(imageData);
                break;
            case 3:
                emm = new MPDError(imageData);
                break;
            case 4:
                emm = new EntropyError(imageData);
                break;
            case 5:
                emm = new SSIMError(imageData);
                break;
        }
        return this;
    }

    public ImageCompressor setThreshold(float threshold) {
        this.threshold = threshold;
        return this;
    }

    public ImageCompressor setMinimumBlockSize(int minimumBlockSize) {
        this.minimumBlockSize = minimumBlockSize;
        return this;
    }

    public ImageCompressor setCompressionLevel(float compressionLevel) {
        this.compressionLevel = compressionLevel;
        if (compressionLevel > 0.0f) {
            threshold = compressionLevel * emm.getMaxErrorValue();
            minimumBlockSize = 1;
        }
        return this;
    }

    public ImageCompressor setOutputFile(File outputFile) {
        this.outputFile = outputFile;
        return this;
    }

    public float getCompressRatio() {
        return (1.0f - ((float) outputFile.length() / fileSize));
    }

    public float getMaxErrorValue() {
        if (emm != null)
            return emm.getMaxErrorValue();
        return 0.0f;
    }

    public int getNodeCount() {
        return builder.getNodeCount();
    }

    public ImageQuadTree getCompressedTree() {
        return root;
    }
    ...
}
```

Gambar 4.4.2. Kelas ImageCompressor: getter setter
(Sumber: Arsip penulis)

```
public class ImageCompressor {
    ...
    public ImageQuadTree compress() throws IOException {
        builder = new ImageQuadTreeBuilder(emm, imageData, threshold, minimumBlockSize);

        root = builder.build(0, 0, imageData.width, imageData.height);

        return root;
    }

    private boolean rebuild() throws IOException {
        exportImage();

        int width = imageData.width;
        int height = imageData.height;

        float upperBound = getMaxErrorValue();
        float lowerBound = 0.0f;
        float sizeDelta = getCompressRatio() - compressionLevel;
        float thresholdDelta = upperBound - lowerBound;

        final float RATIO_LENIENCE = 0.01f; // 1%

        byte iteration = 0;
        while (iteration < 100 && Math.abs(sizeDelta) > RATIO_LENIENCE && thresholdDelta > 0.1f) {
            iteration++;
            if (sizeDelta < 0.0) {
                lowerBound = threshold;
                threshold = (threshold + upperBound) / 2.0f;
            } else {
                upperBound = threshold;
                threshold = (lowerBound + threshold) / 2.0f;
            }

            builder.setThreshold(threshold);
            builder.resetNodeCount();

            root = builder.build(0, 0, width, height);

            exportImage();

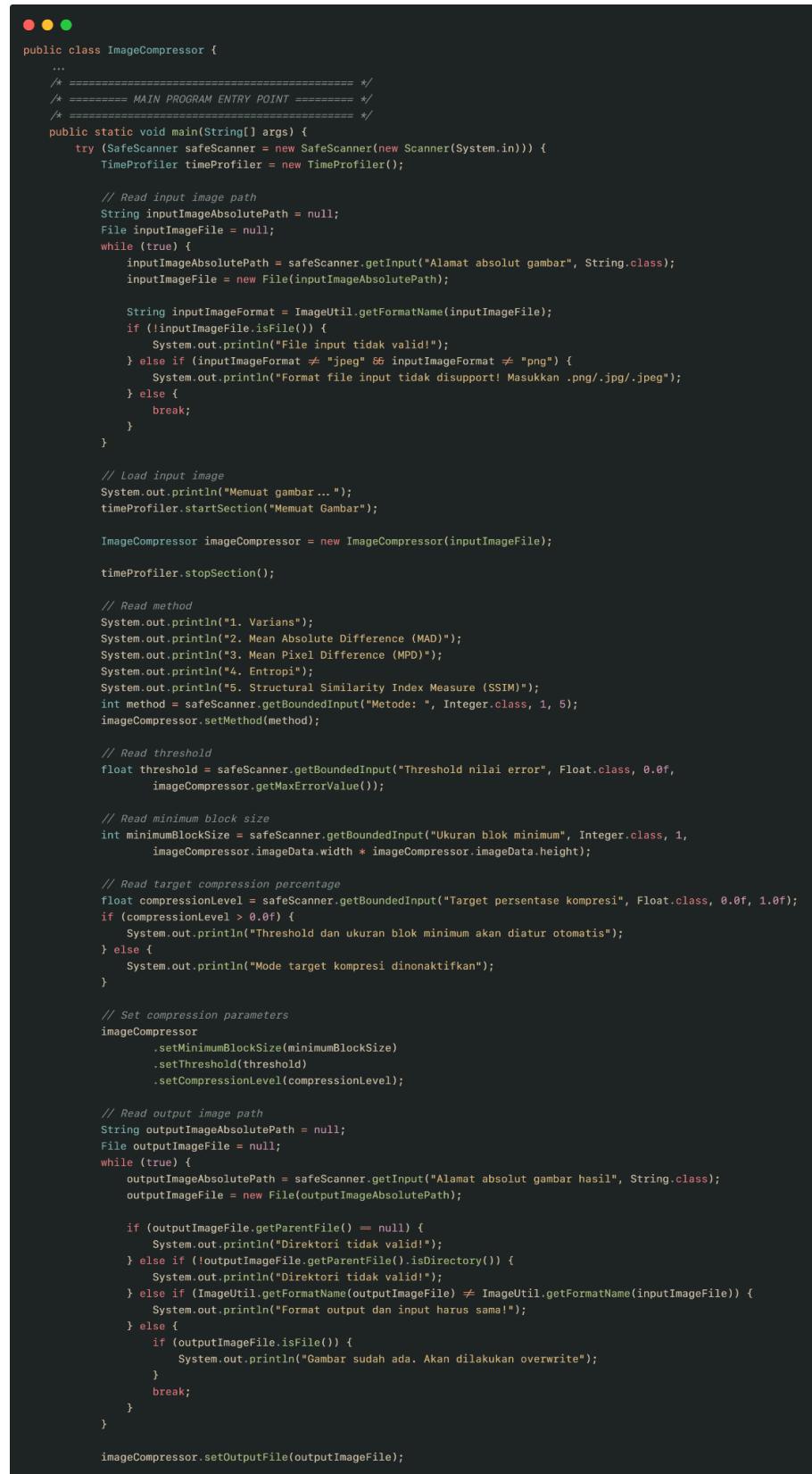
            sizeDelta = getCompressRatio() - compressionLevel;
            thresholdDelta = upperBound - lowerBound;
        }
        return (Math.abs(sizeDelta) <= RATIO_LENIENCE);
    }

    public void exportImage() throws IOException {
        ImageQuadTreeExporter.exportImage(root, builder, outputFile);
    }

    public void exportGIF(File outputGIF) throws Exception {
        ImageQuadTreeExporter.exportGIF(root, builder, outputGIF);
    }

    public boolean isTargetPercentageEnabled() {
        return compressionLevel > 0.0f;
    }
    ...
}
```

*Gambar 4.4.3. Kelas ImageCompressor: fungsi terkait kompresi
(Sumber: Arsip penulis)*



```
public class ImageCompressor {
    ...
    /* ===== MAIN PROGRAM ENTRY POINT ===== */
    /* ===== */
    public static void main(String[] args) {
        try (SafeScanner safeScanner = new SafeScanner(new Scanner(System.in))) {
            TimeProfiler timeProfiler = new TimeProfiler();

            // Read input image path
            String inputImageAbsolutePath = null;
            File inputFile = null;
            while (true) {
                inputImageAbsolutePath = safeScanner.getInput("Alamat absolut gambar", String.class);
                inputFile = new File(inputImageAbsolutePath);

                String inputImageFormat = ImageUtil.getFormatName(inputFile);
                if (!inputImageFormat.isFilename()) {
                    System.out.println("File input tidak valid!");
                } else if (inputImageFormat != "jpeg" && inputImageFormat != "png") {
                    System.out.println("Format file input tidak disupport! Masukkan .png/.jpg/.jpeg");
                } else {
                    break;
                }
            }

            // Load input image
            System.out.println("Memuat gambar ...");
            timeProfiler.startSection("Memuat Gambar");

            ImageCompressor imageCompressor = new ImageCompressor(inputFile);
            timeProfiler.stopSection();

            // Read method
            System.out.println("1. Varians");
            System.out.println("2. Mean Absolute Difference (MAD)");
            System.out.println("3. Mean Pixel Difference (MPD)");
            System.out.println("4. Entropi");
            System.out.println("5. Structural Similarity Index Measure (SSIM)");
            int method = safeScanner.getBoundedInput("Metode: ", Integer.class, 1, 5);
            imageCompressor.setMethod(method);

            // Read threshold
            float threshold = safeScanner.getBoundedInput("Threshold nilai error", Float.class, 0.0f,
                imageCompressor.getMaxErrorValue());

            // Read minimum block size
            int minimumBlockSize = safeScanner.getBoundedInput("Ukuran blok minimum", Integer.class, 1,
                imageCompressor.imageData.width * imageCompressor.imageData.height);

            // Read target compression percentage
            float compressionLevel = safeScanner.getBoundedInput("Target persentase kompresi", Float.class, 0.0f, 1.0f);
            if (compressionLevel > 0.0f) {
                System.out.println("Threshold dan ukuran blok minimum akan diatur otomatis");
            } else {
                System.out.println("Mode target kompresi dinonaktifkan");
            }

            // Set compression parameters
            imageCompressor
                .setMinimumBlockSize(minimumBlockSize)
                .setThreshold(threshold)
                .setCompressionLevel(compressionLevel);

            // Read output image path
            String outputImageAbsolutePath = null;
            File outputFile = null;
            while (true) {
                outputImageAbsolutePath = safeScanner.getInput("Alamat absolut gambar hasil", String.class);
                outputFile = new File(outputImageAbsolutePath);

                if (outputImageFile.getParentFile() == null) {
                    System.out.println("Direktori tidak valid!");
                } else if (!outputImageFile.getParentFile().isDirectory()) {
                    System.out.println("Direktori tidak valid!");
                } else if (ImageUtil.getFormatName(outputImageFile) != ImageUtil.getFormatName(inputFile)) {
                    System.out.println("Format output dan input harus sama!");
                } else {
                    if (outputImageFile.isFile()) {
                        System.out.println("Gambar sudah ada. Akan dilakukan overwrite");
                    }
                    break;
                }
            }

            imageCompressor.setOutputFile(outputImageFile);
        }
    }
}
```

*Gambar 4.4.4. Kelas ImageCompressor: fungsi main bagian 1
(Sumber: Arsip penulis)*

```
// Read output GIF path
String outputGifAbsolutePath = null;
File outputGiffile = null;
while (true) {
    outputGifAbsolutePath = safeScanner.getInput("Alamat absolut GIF hasil (n untuk skip)", String.class);
    if (outputGifAbsolutePath.equalsIgnoreCase("n")) {
        System.out.println("Tidak akan membuat GIF");
        outputGiffile = null;
        break;
    }
    outputGifFile = new File(outputGifAbsolutePath);

    if (outputGifFile.getParentFile() == null) {
        System.out.println("Direktori tidak valid!");
    } else if (!outputGiffile.getParentFile().isDirectory()) {
        System.out.println("Direktori tidak valid!");
    } else if (ImageUtil.getFormatName(outputGifFile) != "gif") {
        System.out.println("Format output harus berupa .gif!");
    } else {
        if (outputGifFile.isFile()) {
            System.out.println("GIF sudah ada. Akan dilakukan overwrite");
        }
        break;
    }
}

// Construct quadtree
System.out.println("Membangun quadtree ...");
timeProfiler.startSection("Build Quadtree");

imageCompressor.compress();
timeProfiler.stopSection();

boolean targetReached = true;
if (imageCompressor.isTargetPercentageEnabled()) {
    // Quadtree rebuild + export image
    System.out.println("Mencoba memenuhi target kompresi ...");
    timeProfiler.startSection("Rebuild + Eksport");

    targetReached = imageCompressor.rebuild();
} else {
    // Export image
    System.out.println("Mengespor gambar ...");
    timeProfiler.startSection("Eksport Gambar");

    imageCompressor.exportImage();
}

timeProfiler.stopSection();
if (!targetReached) {
    System.out.println("Tidak mampu mencapai target kompresi");
}

if (outputGifFile != null) {
    // Export GIF
    System.out.println("Mengekspor GIF ...");
    timeProfiler.startSection("Eksport GIF");

    imageCompressor.exportGIF(outputGifFile);
    timeProfiler.stopSection();
}

// Display compression statistics
System.out.println("Kompresi berhasil");
System.out.println("");
timeProfiler.print();
System.out.println("");
System.out.format("Sebelum      : %.2f KB\n", (float) inputImageFile.length() / 1024.0f);
System.out.format("Sesudah     : %.2f KB\n", (float) outputImageFile.length() / 1024.0f);
System.out.format("Ratio Kompresi : %.2f%%", imageCompressor.getCompressRatio() * 100.0f);
if (imageCompressor.isTargetPercentageEnabled() && !targetReached) {
    System.out.println(
        " (Target tidak bisa dicapai karena iterasi terlalu banyak atau tidak memungkinkan)");
} else {
    System.out.println("");
}
System.out.format("Kedalaman Pohon : %d\n", imageCompressor.getCompressedTree().getDepth());
System.out.format("Jumlah Simpul  : %d\n", imageCompressor.getNodeCount());
System.out.println("");
System.out.format("Alamat Gambar  : %s\n", inputImagePath);
if (outputGifFile != null)
    System.out.format("Alamat GIF     : %s\n", outputGifAbsolutePath);

} catch (Exception e) {
    System.out.println("Terjadi error. Cek log/crash.log untuk melihat log");
    CrashLogger.logException(e);
}
}
```

Gambar 4.4.5. Kelas ImageCompressor: fungsi main bagian 2
(Sumber: Arsip penulis)

BAB V

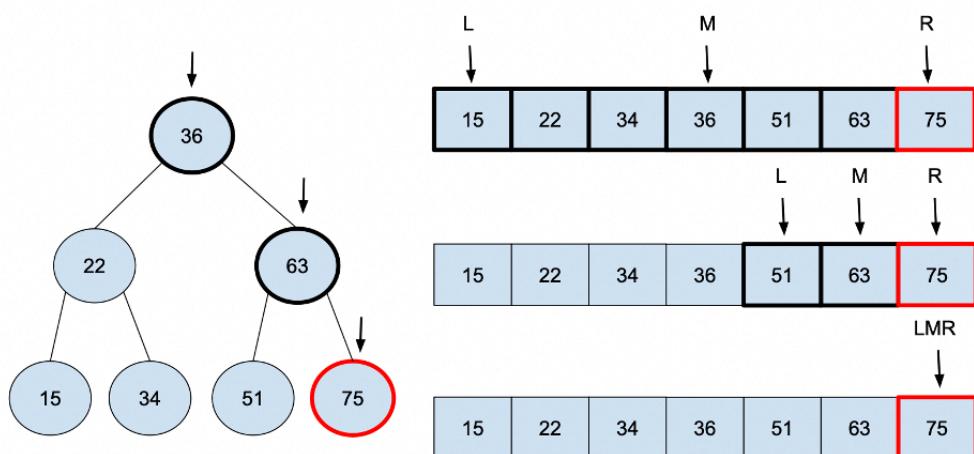
IMPLEMENTASI BONUS

Pada bab ini, akan dibahas lebih dalam terkait pengimplementasian fitur-fitur bonus pada program kompresi gambar.

5.1. Target Persentase Kompresi

Bonus ini memungkinkan pengguna menentukan target persentase kompresi berupa nilai *floating point* dengan range 0 sampai 1 yang menandakan target kompresi 0% sampai 100%. Jika pengguna memberikan nilai 0, mode ini akan dinonaktifkan.

Ketika mode ini diaktifkan, algoritma akan menyesuaikan nilai ambang batas secara otomatis untuk mencapai target persentase kompresi yang ditentukan. Penyesuaian nilai ambang batas ini dilakukan secara dinamis, memberikan fleksibilitas dalam proses kompresi untuk memenuhi target efisiensi sambil mempertahankan kualitas gambar. Parameter ukuran blok minimum juga dinonaktifkan jika mode ini digunakan.



Gambar 5.1.1. Algoritma pencarian biner.
(Sumber: <https://medium.com/geekculture/binary-search-bc12054cb3a>)

Untuk mengatur nilai ambang batas, program menggunakan algoritma pencarian biner (*binary search*). Singkatnya, algoritma ini dapat mencari nilai (ambang batas) yang dicari dengan cara menghilangkan setengah dari seluruh sampel di tiap iterasinya. Ruang sampel berupa rentang nilai galat penilaian gambar, sesuai dengan metode IQA yang dipilih pengguna. Untuk rentang nilai galat tiap metode IQA sudah dibahas di [Bab 2.3](#).

Jika batas bawah, batas atas, dan nilai ambang batas direpresentasikan dengan L , R , M , maka alur algoritma dapat dijabarkan sebagai runtutan instruksi berikut:

- 1) Bangun *quadtree* dengan nilai awal $M = 0.5 \times (\maxThreshold)$.
- 2) Jika rasio kompresi sama dengan target kompresi, berhenti.
- 3) Jika selisih rasio kompresi bernilai negatif (*under-compression*), maka:
 - a) Ubah L menjadi M .
 - b) Ubah M menjadi: $\frac{(M + R)}{2}$.

Jika tidak (*over-compression*), maka:

- c) Ubah R menjadi M .
 - d) Ubah M menjadi: $\frac{(M+L)}{2}$.
- 4) Bangun ulang *quadtree* dengan nilai M yang baru.
- 5) Jika jumlah iterasi masih dibawah batas maksimum iterasi, ulangi dari langkah 2).

Rasio kompresi setiap saat dapat ditentukan dengan rumus:

$$R = 1 - \frac{\text{Ukuran Gambar Terkompresi}}{\text{Ukuran Gambar Asli}}$$

Perlu diingat bahwa algoritma pencarian biner umumnya digunakan pada persoalan dengan domain yang diskrit, misal mencari angka pada larik terurut. Namun, dalam kasus mencari nilai ambang batas, pencarian nilai dilakukan pada **domain kontigu**.

Jika algoritma pencarian biner diterapkan pada domain kontigu $[a, b]$, maka perlu dicari nilai x di selang tersebut sehingga $f(x) = c$. Pada kasus ini, fungsi $f(x)$ merupakan fungsi yang menerima nilai ambang batas x dan c adalah rasio kompresi gambar. Fungsi $f(x)$ tidak diketahui karena proses kompresi juga dilakukan oleh format PNG atau JPEG. Agar algoritma pencarian biner dapat diterapkan, diasumsikan bahwa fungsi tersebut bersifat monoton naik.

Di sisi teknis, algoritma pencarian biner tetap bekerja dengan cara yang sama yaitu dengan menentukan batas bawah, atas, dan nilai tengah. Kemudian di tiap iterasi, nilai-nilai tersebut diubah sesuai dengan selisih nilai tengah dengan nilai yang dicari. Namun, karena representasi bilangan kontigu di komputer terbatas pada presisi *floating point*, maka pengecekan solusi perlu dilakukan dengan konstanta toleransi ϵ . Dalam kata lain, algoritma akan mencari nilai x yang memenuhi persamaan:

$$|f(x) - c| \leq \epsilon$$

Untuk itu, dilakukan **modifikasi** pada algoritma pencarian biner, yaitu dengan memodifikasi langkah 2) dengan persamaan di atas.

Dalam implementasinya, nilai ϵ menunjukkan akurasi program dalam mencapai target kompresi yang diharapkan oleh pengguna. Program kompresi gambar ini memiliki nilai $\epsilon = 0.01$ atau galat rasio kompresi sebesar 1% (selama target kompresi mungkin untuk dicapai).

5.2. Structural Similarity Index Measure (SSIM)

Bonus ini memungkinkan penggunaan metode IQA-FR SSIM sebagai pengukuran galat gambar setiap blok. Untuk menambahkan metode IQA baru ke dalam program, cukup dengan membuat kelas pengukur galat baru yang mengeksten kelas abstrak **ErrorMeasurementMethod**.

Dalam tugas ini, disebutkan bahwa SSIM dilakukan untuk mengukur nilai galat antara blok gambar sebelum dan sesudah kompresi. Pengkompresian blok gambar menyebabkan warna blok ternormalisasi menjadi warna rata-rata blok tersebut. Oleh karena itu, jika x dan y adalah blok gambar sebelum dan sesudah kompresi, maka diperoleh nilai $\sigma_{y,c}^2$ dan

$\sigma_{xy,c}$ sama dengan 0. Selain itu, karena x dan y pada dasarnya merupakan blok gambar yang sama, maka berlaku juga $\mu_{x,c} = \mu_{y,c}$. Sehingga, didapat rumus simplifikasi:

$$\begin{aligned} SSIM_c(x, y) &= \frac{(2\mu_c^2 + C_1)(2(0) + C_2)}{(2\mu_c^2 + C_1)(\sigma_{x,c}^2 + (0) + C_2)} \\ &= \frac{C_2}{\sigma_{x,c}^2 + C_2} \end{aligned}$$

dengan mensubstitusi nilai $C_2 = (K_2 L)^2$ dengan $K_2 = 0.03$ dan $L = 255$, maka diperoleh rumus akhir SSIM simplifikasi per kanal warna:

$$\begin{aligned} SSIM_c(x, y) &= \frac{((0.03)(255))^2}{\sigma_{x,c}^2 + ((0.03)(255))^2} \\ &= \frac{58.5225}{\sigma_{x,c}^2 + 58.5225} \end{aligned}$$

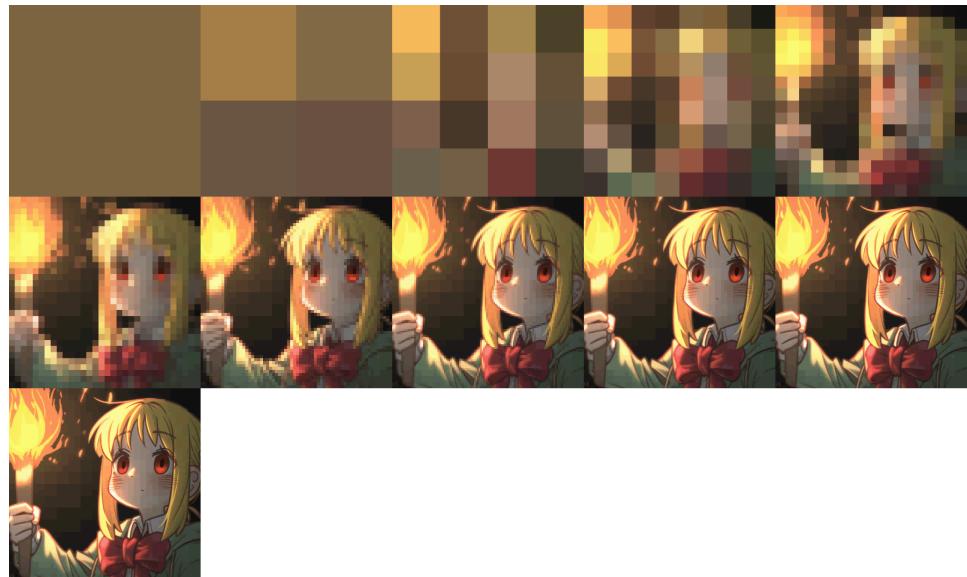
Terlihat dari rumus di atas bahwa nilai dari SSIM yang dimodifikasi hanya bergantung dari varians blok gambar sebelum terkompresi. Sesuai dengan pembahasan di [Bab 2.3.1](#), rentang galat metode varians adalah $[0, 16256.25]$. Sehingga, didapat rentang nilai galat SSIM termodifikasi adalah $[0, 1)$.

Perlu diingat bahwa **SSIM menyatakan kemiripan** sinyal x dengan sinyal y . Program mengharapkan nilai galat, sehingga nilai kemiripan SSIM tersimplifikasi perlu dibalik terlebih dahulu dengan rumus:

$$SSIM_{error} = 1 - SSIM_{RGB}$$

5.3. GIF

Bonus ini memungkinkan pengguna untuk mengekspor file GIF yang berisi animasi (*looping*) berisi proses pembangunan *quadtree*. Di setiap *frame*, file akan menunjukkan gambar yang terbentuk oleh simpul-simpul di kedalaman saat itu. Kedalaman simpul dimulai dari 1 hingga simpul terdalam.



Gambar 5.3.1. Setiap frame GIF, dipetakan ke gambar
(Sumber: Arsip penulis)

Hingga saat program ini dibuat, belum ada pustaka bawaan dari Java untuk memudahkan penulisan file GIF, sehingga pustaka eksternal “AnimatedGIFWriter” milik dragon66 digunakan.

Menyiapkan penulisan file GIF sangat mudah dengan pustaka ini. Pemilik pustaka memberikan sampel program sederhana terkait penggunaan pustakanya sebagai berikut:

```
// True for dither. Will use more memory and CPU
AnimatedGIFWriter writer = new AnimatedGIFWriter(true);
OutputStream os = new FileOutputStream("animated.gif");
// Grab the BufferedImage whatever way you can
BufferedImage frame;
// Use -1 for both logical screen width and height to use the first frame dimension
writer.prepareForWrite(os, -1, -1)
writer.writeFrame(os, frame);
// Keep adding frame here
writer.finishWrite(os);
// And you are done!!!
```

Gambar 5.3.2. Instruksi penggunaan pustaka AnimatedGIFWriter.
(Sumber: <https://github.com/dragon66/animated-gif-writer>)

Dalam implementasinya di program kompresi gambar, objek penulis GIF dibuat sebelum melakukan *looping* sebanyak kedalaman pohon maksimum. Di setiap iterasinya, warna pada simpul di target kedalaman (atau simpul daun jika kedalaman simpul kurang) akan dicetak ke *array of integer*. Nilai *integer* disini berisi nilai warna ARGB 8-bit yang dimampatkan menjadi satu angka 32-bit. Di akhir tiap iterasi, fungsi *writeFrame* dipanggil dengan parameter *buffer* terkait.

BAB VI

EKSPERIMENT DAN ANALISIS

6.1. Pengujian Program

Semua *input* uji berada di dalam folder input. Keluaran atau hasil kompresi gambar dan GIF berada di dalam folder test. Berikut adalah data hasil uji kompresi gambar dengan format input (file input – resolusi, metode, nilai ambang batas, ukuran blok minimum, target kompresi, GIF):

No	Input	Output																									
1.	(input1.png–2544x1849, varians, 500, 10, 0, NO GIF) → (output1.png)	  <pre> > java -jar bin/ImageCompressor.jar Alamat absolut gambar: C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\input\input1.png Memuat gambar ... 1. Varians 2. Mean Absolute Difference (MAD) 3. Mean Pixel Difference (MPD) 4. Entropi 5. Structural Similarity Index Measure (SSIM) Metode: (1 - 5): 1 Threshold nilai error (0.0 - 16256.25): 500 Ukuran blok minimum (1 - 4703856): 10 Target persentase kompresi (0.0 - 1.0): 0 Mode target kompresi dinonaktifkan Alamat absolut gambar hasil: C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output1.png </pre> <table border="1"> <thead> <tr> <th>Proses</th> <th>Waktu</th> </tr> </thead> <tbody> <tr> <td>Memuat Gambar</td> <td>: 279.73 ms</td> </tr> <tr> <td>Build Quadtree</td> <td>: 188.21 ms</td> </tr> <tr> <td>Eksport Gambar</td> <td>: 368.97 ms</td> </tr> <tr> <td>Total Waktu</td> <td>: 828.91 ms</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Sebelum</th> <th>:</th> <th>2034.34 KB</th> </tr> </thead> <tbody> <tr> <td>Sesudah</td> <td>:</td> <td>218.80 KB</td> </tr> <tr> <td>Rasio Kompresi</td> <td>:</td> <td>89.24%</td> </tr> <tr> <td>Kedalaman Pohon</td> <td>:</td> <td>10</td> </tr> <tr> <td>Jumlah Simpul</td> <td>:</td> <td>24585</td> </tr> </tbody> </table> <p>Alamat Gambar : C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output1.png</p>	Proses	Waktu	Memuat Gambar	: 279.73 ms	Build Quadtree	: 188.21 ms	Eksport Gambar	: 368.97 ms	Total Waktu	: 828.91 ms	Sebelum	:	2034.34 KB	Sesudah	:	218.80 KB	Rasio Kompresi	:	89.24%	Kedalaman Pohon	:	10	Jumlah Simpul	:	24585
Proses	Waktu																										
Memuat Gambar	: 279.73 ms																										
Build Quadtree	: 188.21 ms																										
Eksport Gambar	: 368.97 ms																										
Total Waktu	: 828.91 ms																										
Sebelum	:	2034.34 KB																									
Sesudah	:	218.80 KB																									
Rasio Kompresi	:	89.24%																									
Kedalaman Pohon	:	10																									
Jumlah Simpul	:	24585																									
2.	(input1.png–2544x1849, varians, 0, 1, 0.5, no GIF) → (output2.png)	 																									

No	Input	Output												
	<pre>> java -jar bin/ImageCompressor.jar Alamat absolut gambar: C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\input\input1.png Memuat gambar... 1. Varians 2. Mean Absolute Difference (MAD) 3. Mean Pixel Difference (MPD) 4. Entropi 5. Structural Similarity Index Measure (SSIM) Metode: (1 - 5): 1 Threshold nilai error (0.0 - 16256.25): 0 Ukuran blok minimum (1 - 4708356): 1 Target persentase kompresi (0.0 - 1.0): 0.5 Threshold dan ukuran blok minimum akan diatur otomatis Alamat absolut gambar hasil: C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output2.png</pre>	<p>Kompresi berhasil</p> <table> <thead> <tr> <th>Proses</th> <th>Waktu</th> </tr> </thead> <tbody> <tr> <td>Memuat Gambar</td> <td>: 279.18 ms</td> </tr> <tr> <td>Build Quadtree</td> <td>: 69.63 ms</td> </tr> <tr> <td>Rebuild + Ekspor</td> <td>: 3928.08 ms</td> </tr> <tr> <td>Total Waktu</td> <td>: 4276.90 ms</td> </tr> </tbody> </table> <p>Sebelum : 2034.34 KB Sesudah : 1006.45 KB Rasio Kompresi : 50.53% Kedalaman Pohon : 13 Jumlah Simpul : 436421</p> <p>Alamat Gambar : C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output2.png</p>	Proses	Waktu	Memuat Gambar	: 279.18 ms	Build Quadtree	: 69.63 ms	Rebuild + Ekspor	: 3928.08 ms	Total Waktu	: 4276.90 ms		
Proses	Waktu													
Memuat Gambar	: 279.18 ms													
Build Quadtree	: 69.63 ms													
Rebuild + Ekspor	: 3928.08 ms													
Total Waktu	: 4276.90 ms													
3.	(input2.png–565x565, varians, 0, 1, 0.5, GIF) → (output3.png output3.gif)													
	<pre>> java -jar bin/ImageCompressor.jar Alamat absolut gambar: C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\input\input2.png Memuat gambar... 1. Varians 2. Mean Absolute Difference (MAD) 3. Mean Pixel Difference (MPD) 4. Entropi 5. Structural Similarity Index Measure (SSIM) Metode: (1 - 5): 1 Threshold nilai error (0.0 - 16256.25): 0 Ukuran blok minimum (1 - 319229): 1 Target persentase kompresi (0.0 - 1.0): 0.5 Threshold dan ukuran blok minimum akan diatur otomatis Alamat absolut gambar hasil: C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output3.png Alamat absolut GIF hasil (n untuk skip): C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output3.gif</pre>	<p>Kompresi berhasil</p> <table> <thead> <tr> <th>Proses</th> <th>Waktu</th> </tr> </thead> <tbody> <tr> <td>Memuat Gambar</td> <td>: 111.32 ms</td> </tr> <tr> <td>Build Quadtree</td> <td>: 23.32 ms</td> </tr> <tr> <td>Rebuild + Ekspor</td> <td>: 789.04 ms</td> </tr> <tr> <td>Ekspor GIF</td> <td>: 312.06 ms</td> </tr> <tr> <td>Total Waktu</td> <td>: 1235.74 ms</td> </tr> </tbody> </table> <p>Sebelum : 750.07 KB Sesudah : 378.80 KB Rasio Kompresi : 49.50% Kedalaman Pohon : 11 Jumlah Simpul : 175197</p> <p>Alamat Gambar : C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output3.png Alamat GIF : C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output3.gif</p>	Proses	Waktu	Memuat Gambar	: 111.32 ms	Build Quadtree	: 23.32 ms	Rebuild + Ekspor	: 789.04 ms	Ekspor GIF	: 312.06 ms	Total Waktu	: 1235.74 ms
Proses	Waktu													
Memuat Gambar	: 111.32 ms													
Build Quadtree	: 23.32 ms													
Rebuild + Ekspor	: 789.04 ms													
Ekspor GIF	: 312.06 ms													
Total Waktu	: 1235.74 ms													
4.	(input2.png–565x565, MAD, 0, 1, 0.5, GIF) → (output4.png output4.gif)													

No	Input	Output
	 <pre> Alamat absolut gambar: C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\input\input2.png Muat gambar... 1. Varians 2. Mean Absolute Difference (MAD) 3. Mean Pixel Difference (MPD) 4. Entropi 5. Structural Similarity Index Measure (SSIM) Metode: Threshold nilai error (0.0 - 127.5): 0 Ukuran blok minimum (1 - 399298): 1 Target persentase kompresi (0.0 - 1.0): 0.5 Threshold dan ukuran blok minimum akan diatur otomatis Alamat absolut gambar hasil: C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output4.png Alamat absolut GIF hasil (n untuk skip): C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output4.gif </pre>	 <pre> Kompresi berhasil Proses Waktu Memuat Gambar : 106.51 ms Build Quadtree : 28.83 ms Rebuild + Ekspor : 816.32 ms Ekspor GIF : 304.07 ms Total Waktu : 1255.73 ms Sebelum : 750.07 KB Sesudah : 370.86 KB Rasio Kompresi : 50.56% Kedalaman Pohon : 11 Jumlah Simpul : 17681 Alamat Gambar : C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output4.png Alamat GIF : C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output4.gif </pre>
5.	(input2.png–565x565, MPD, 0, 1, 0.5, GIF) → (output5.png output5.gif)	
		

No	Input	Output												
	<pre>> java -jar bin/ImageCompressor.jar Alamat absolut gambar: C:\Users\rizac\sigma\stima\tucil\tucil2_13523162\input\input2.png Memuat gambar... 1. Varians 2. Mean Absolute Difference (MAD) 3. Mean Pixel Difference (MPD) 4. Entropi 5. Structural Similarity Index Measure (SSIM) Metode: (1 - 5) Threshold nilai error (0.0 - 255.0): 0 Ukuran blok minimum (1 - 210225): 1 Target persentase kompresi (0.0 - 1.0): 0.5 Threshold dan ukuran blok minimum akan diatur otomatis Alamat absolut gambar hasil: C:\Users\rizac\sigma\stima\tucil\tucil2_13523162\test\output5.png Alamat absolut GIF hasil (n untuk skip): C:\Users\rizac\sigma\stima\tucil\tucil2_13523162\test\output5.gif</pre>	<p>Kompresi berhasil</p> <table> <thead> <tr> <th>Proses</th> <th>Waktu</th> </tr> </thead> <tbody> <tr> <td>Memuat Gambar</td> <td>: 107.86 ms</td> </tr> <tr> <td>Build Quadtree</td> <td>: 51.34 ms</td> </tr> <tr> <td>Rebuild + Ekspor</td> <td>: 675.27 ms</td> </tr> <tr> <td>Ekspor GIF</td> <td>: 347.39 ms</td> </tr> <tr> <td>Total Waktu</td> <td>: 1181.84 ms</td> </tr> </tbody> </table> <p>Sebelum : 750.00 KB Sesudah : 378.98 KB Rasio Kompresi : 49.49% Kedalaman Pohon : 11 Jumlah Simpul : 164237</p> <p>Alamat Gambar : C:\Users\rizac\sigma\stima\tucil\tucil2_13523162\test\output5.png Alamat GIF : C:\Users\rizac\sigma\stima\tucil\tucil2_13523162\test\output5.gif</p>	Proses	Waktu	Memuat Gambar	: 107.86 ms	Build Quadtree	: 51.34 ms	Rebuild + Ekspor	: 675.27 ms	Ekspor GIF	: 347.39 ms	Total Waktu	: 1181.84 ms
Proses	Waktu													
Memuat Gambar	: 107.86 ms													
Build Quadtree	: 51.34 ms													
Rebuild + Ekspor	: 675.27 ms													
Ekspor GIF	: 347.39 ms													
Total Waktu	: 1181.84 ms													
6.	(input2.png—565x565, entropi, 0, 1, 0.5, GIF) → (output6.png output6.gif)	  <pre>> java -jar bin/ImageCompressor.jar Alamat absolut gambar: C:\Users\rizac\sigma\stima\tucil\tucil2_13523162\input\input2.png Memuat gambar... 1. Varians 2. Mean Absolute Difference (MAD) 3. Mean Pixel Difference (MPD) 4. Entropi 5. Structural Similarity Index Measure (SSIM) Metode: (1 - 5) Threshold nilai error (0.0 - 255.0): 0 Ukuran blok minimum (1 - 210225): 1 Target persentase kompresi (0.0 - 1.0): 0.5 Threshold dan ukuran blok minimum akan diatur otomatis Alamat absolut gambar hasil: C:\Users\rizac\sigma\stima\tucil\tucil2_13523162\test\output6.png Alamat absolut GIF hasil (n untuk skip): C:\Users\rizac\sigma\stima\tucil\tucil2_13523162\test\output6.gif</pre> <p>Kompresi berhasil</p> <table> <thead> <tr> <th>Proses</th> <th>Waktu</th> </tr> </thead> <tbody> <tr> <td>Memuat Gambar</td> <td>: 106.37 ms</td> </tr> <tr> <td>Build Quadtree</td> <td>: 98.70 ms</td> </tr> <tr> <td>Rebuild + Ekspor</td> <td>: 871.25 ms</td> </tr> <tr> <td>Ekspor GIF</td> <td>: 287.28 ms</td> </tr> <tr> <td>Total Waktu</td> <td>: 1363.66 ms</td> </tr> </tbody> </table> <p>Sebelum : 750.07 KB Sesudah : 376.10 KB Rasio Kompresi : 49.86% Kedalaman Pohon : 10 Jumlah Simpul : 154505</p> <p>Alamat Gambar : C:\Users\rizac\sigma\stima\tucil\tucil2_13523162\test\output6.png Alamat GIF : C:\Users\rizac\sigma\stima\tucil\tucil2_13523162\test\output6.gif</p>	Proses	Waktu	Memuat Gambar	: 106.37 ms	Build Quadtree	: 98.70 ms	Rebuild + Ekspor	: 871.25 ms	Ekspor GIF	: 287.28 ms	Total Waktu	: 1363.66 ms
Proses	Waktu													
Memuat Gambar	: 106.37 ms													
Build Quadtree	: 98.70 ms													
Rebuild + Ekspor	: 871.25 ms													
Ekspor GIF	: 287.28 ms													
Total Waktu	: 1363.66 ms													
7.	(input2.png—565x565, SSIM, 0, 1, 0.5, GIF) → (output7.png output7.gif)													

No	Input	Output
	 <pre> > java -jar bin/ImageCompressor.jar Alamat absolut gambar: C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\input\input2.png Memuat gambar ... 1. Varians 2. Mean Absolute Difference (MAD) 3. Mean Pixel Difference (MPD) 4. Entropi 5. Structural Similarity Index Measure (SSIM) Metode: (1 - 5): 5 Threshold nilai error (0.0 - 1.0): 0 Ukuran blok minimum (1 - 319225): 1 Target persentase kompresi (0.0 - 1.0): 0.5 Threshold dan ukuran blok minimum akan diatur otomatis Alamat absolut gambar hasil: C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output7.png Alamat absolut GIF hasil (n untuk skip): C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test </pre>	 <pre> Kompresi berhasil Proses Waktu Memuat Gambar : 107.02 ms Build Quadtree : 63.35 ms Rebuild + Ekspor : 634.83 ms Ekspor GIF : 332.87 ms Total Waktu : 1138.06 ms Sebelum : 758.07 KB Sesudah : 372.46 KB Rasio Kompresi : 50.34% Kedalaman Pohon : 11 Jumlah Simpul : 170297 Alamat Gambar : C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output7.png Alamat GIF : C:\Users\rizac\sigma\stima\tucil\Tucil2_13523162\test\output7.gif </pre>
8.	[stress test] (input3.jpg—4000x6000, SSIM, 0, 1, 0.4, GIF) → (output8.jpg, output8.gif)	

No	Input	Output																						
	 <pre>> java -jar bin/ImageCompressor.jar Alamat absolut gambar: C:\Users\rizac\stima\tucil2_13523162\input\input3.jpg Memuat gambar... 1. Varians 2. Mean Absolute Difference (MAD) 3. Mean Pixel Difference (MPD) 4. Entropi 5. Structural Similarity Index Measure (SSIM) Metode: (1 - 5): 5 Threshold nilai error (0.0 - 1.0): 0 Ukuran blok minimum (1 - 24000000): 1 Target persentase kompresi (0.0 - 1.0): 0.4 Threshold dan ukuran blok minimum akan diatur otomatis Alamat absolut gambar hasil: C:\Users\rizac\stima\tucil2_13523162\test\output8.jpg Alamat absolut GIF hasil (n untuk skip): C:\Users\rizac\stima\tucil2_13523162\test\output8.gif</pre>	 <p>Kompresi berhasil</p> <table> <thead> <tr> <th>Proses</th> <th>Waktu</th> </tr> </thead> <tbody> <tr> <td>Memuat Gambar</td> <td>6176.78 ms</td> </tr> <tr> <td>Build Quadtree</td> <td>1105.02 ms</td> </tr> <tr> <td>Rebuild + Ekspor</td> <td>7001.50 ms</td> </tr> <tr> <td>Ekspor GIF</td> <td>12339.98 ms</td> </tr> <tr> <td>Total Waktu</td> <td>26623.28 ms</td> </tr> </tbody> </table> <table> <thead> <tr> <th>Sebelum</th> <th>: 4241.22 KB</th> </tr> </thead> <tbody> <tr> <td>Sesudah</td> <td>: 2517.19 KB</td> </tr> <tr> <td>Rasio Kompresi</td> <td>: 40.65%</td> </tr> <tr> <td>Kedajaman Pohon</td> <td>: 14</td> </tr> <tr> <td>Jumlah Simpul</td> <td>: 1139857</td> </tr> </tbody> </table> <p>Alamat Gambar : C:\Users\rizac\stima\tucil2_13523162\test\output8.jpg Alamat GIF : C:\Users\rizac\stima\tucil2_13523162\test\output8.gif</p>	Proses	Waktu	Memuat Gambar	6176.78 ms	Build Quadtree	1105.02 ms	Rebuild + Ekspor	7001.50 ms	Ekspor GIF	12339.98 ms	Total Waktu	26623.28 ms	Sebelum	: 4241.22 KB	Sesudah	: 2517.19 KB	Rasio Kompresi	: 40.65%	Kedajaman Pohon	: 14	Jumlah Simpul	: 1139857
Proses	Waktu																							
Memuat Gambar	6176.78 ms																							
Build Quadtree	1105.02 ms																							
Rebuild + Ekspor	7001.50 ms																							
Ekspor GIF	12339.98 ms																							
Total Waktu	26623.28 ms																							
Sebelum	: 4241.22 KB																							
Sesudah	: 2517.19 KB																							
Rasio Kompresi	: 40.65%																							
Kedajaman Pohon	: 14																							
Jumlah Simpul	: 1139857																							

6.2. Analisis Algoritma Program

Untuk menghitung kompleksitas keseluruhan program, perlu dilakukan analisis dalam menghitung bagian-bagian kecil program. Nantinya akan didapat kompleksitas waktu keseluruhan dengan menggabungkan kompleksitas waktu semua bagian program.

6.2.1. Analisis Algoritma Pembentukan Quadtree

Proses pembentukan *quadtree* mencakup beberapa proses. Dimulai dari perhitungan warna rata-rata untuk digunakan dalam perhitungan galat blok dan sebagai representasi warna simpul. Setelah normalisasi warna, perhitungan galat dilakukan sesuai dengan metode IQA yang dipilih oleh pengguna. Terakhir, dilakukan pengecekan kondisi; apabila galat di atas ambang batas dan blok maupun kuadran blok

masih diatas ukuran blok minimum, maka blok dipecah menjadi 4 upa-blok (empat kuadran) secara rekursif. Jika tidak, maka rekursi berhenti.

Proses normalisasi/perhitungan rata-rata warna dilakukan dalam waktu $O(n)$ dengan n adalah jumlah piksel atau luas blok gambar. Proses normalisasi warna memiliki kompleksitas waktu linier karena program hanya perlu melakukan iterasi sebanyak jumlah piksel untuk menjumlahkan warna merah, hijau, dan biru di blok gambar. Lalu diakhiri dengan membagi nilai warna dengan luas blok gambar.

Setelah mendapatkan rata-rata warna blok. Program menghitung galat sesuai metode IQA yang dipilih. Rumus dan algoritma yang digunakan oleh masing-masing metode berbeda. Namun, semua metode tersebut memiliki kompleksitas waktu $O(n)$.

Di akhir tiap iterasi, program akan melakukan perhitungan untuk pengecekan kondisi pembagian simpul. Pengecekan berdasarkan nilai galat (yang telah dihitung) beserta ukuran blok dan upa-blok. Proses ini memiliki kompleksitas waktu $O(1)$ karena tidak bergantung dengan jumlah input. Jika kondisi benar, maka dilakukan kembali proses yang sama sebanyak empat kali dengan ukuran n dibagi 4.

Bila keseluruhan proses tersebut dilambangkan dengan fungsi $T(n)$, maka didapat bentuk:

$$\begin{aligned} T(n) &= O(n) + O(n) + O(1) + 4T\left(\frac{n}{4}\right) \\ &= 4T\left(\frac{n}{4}\right) + O(n) \end{aligned}$$

Karena fungsi tersebut berbentuk rekurens, diperlukan basis fungsi. Didefinisikan fungsi rekurens membentuk *quadtree* sebagai fungsi *piecewise* berikut:

$$T(n) = \begin{cases} O(n), & n < C_1 \vee \text{error} \leq C_2 \\ 4T\left(\frac{n}{4}\right) + O(n), & n \geq C_1 \wedge \text{error} > C_2 \end{cases}$$

Dimana:

$T(n)$: Kompleksitas waktu pembuatan *quadtree*.

$O(n)$: Kompleksitas waktu algoritma tiap rekursi.

error : Nilai galat blok.

C_1 : Konstanta ukuran blok minimum.

C_2 : Konstanta ambang batas galat.

Bentuk fungsi rekurens tersebut serupa dengan bentuk yang dijelaskan oleh teorema Master:

$$T(n) = aT\left(\frac{n}{b}\right) + cn^d$$

Dengan melakukan substitusi, didapat nilai $a = 4$, $b = 4$, $c = 1$, dan $d = 1$. Menggunakan teorema Master, karena $a = b^d$ maka berlaku kasus ke-2 sehingga didapatkan persamaan kompleksitas waktu algoritma pembuatan *quadtree*:

$$\begin{aligned} T(n) &= O(n^d \log(n)), \text{ substitusikan } d = 1 \\ &= O(n \log(n)) \end{aligned}$$

Perlu diingat bahwa basis fungsi logaritma tidak berpengaruh di notasi *Big-O*.

6.2.2. Analisis Algoritma Pencarian Biner Nilai Ambang Batas

Sesuai pembahasan di [Bab 5.1](#), disebutkan bahwa pengaturan dinamis nilai ambang batas galat dilakukan dengan algoritma pencarian biner. Kompleksitas waktu pencarian biner juga dapat dicari menggunakan teorema Master.

Algoritma pencarian biner hanya mengambil titik tengah di tiap rekursi, kemudian membagi ruang sampel menjadi 2, sehingga dapat didefinisikan fungsi rekurens *piecewise*:

$$T(n) = \begin{cases} 1, & n = 1 \\ T\left(\frac{n}{2}\right) + 1, & n > 1 \end{cases}$$

Didapatkan konstanta a , b , c , dan d yang memenuhi teorema Master kasus ke-2, sehingga didapat kompleksitas waktu pencarian biner:

$$\begin{aligned} T(n) &= O(n^0 \log(n)) \\ &= O(\log(n)) \end{aligned}$$

Merujuk pada [Bab 5.1](#), pencarian biner dilakukan pada domain yang kontigu, yaitu rentang nilai galat metode IQA yang dipilih. Karena pengecekan solusi ditentukan oleh konstanta presisi yang membagi selang $[a, b]$ menjadi “diskrit”, maka didapat rumus kompleksitas untuk algoritma pencarian biner dalam domain kontigu:

$$\begin{aligned} T(n) &= O\left(\log\left(\frac{n}{\varepsilon}\right)\right) \\ &= O(\log(n)) \end{aligned}$$

Dimana:

n : Rentang galat metode IQA.

ε : Konstanta kecil. Dalam program, nilai ini adalah 0.01 (1%).

6.2.3. Analisis Algoritma Pembuatan GIF

Proses pembuatan GIF mencakup iterasi seluruh kedalaman pohon. Di setiap kedalaman pohon, program mengekspor gambar yang berbeda berukuran n . Namun kedalaman pohon ini bergantung pada banyak parameter seperti ukuran blok

minimum, ambang batas galat, dan kualitas gambar. Anggap kasus terburuk (kedalaman pohon maksimal), maka didapat kompleksitas waktu:

$$T(n) = O(n \log(n))$$

6.2.4. Kompleksitas Waktu Total Program Beserta Bonus

Untuk mendapatkan kompleksitas keseluruhan program, maka dilakukan penggabungan antara kompleksitas waktu pembentukan *quadtree*, pencarian biner nilai ambang batas, dan pembentukan GIF. Kompleksitas waktu pencarian biner dikali karena proses pembentukan *quadtree* dilakukan ulang setiap mendapat nilai ambang batas yang baru. Sedangkan proses pembentukan GIF dilakukan sekali di akhir. Sehingga didapat kompleksitas waktu program total lengkap dengan fitur-fitur bonus adalah:

$$\begin{aligned} T(m, n) &= O(\log(m) * n \log(n) + n \log(n)) \\ &= O(\log(m) * n \log(n)) \end{aligned}$$

Dimana:

n : Jumlah piksel atau luas gambar.

m : Rentang nilai galat sesuai metode IQA.

Lampiran

Tabel penyelesaian tugas besar

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

Pranala *repository*

https://github.com/L4mbads/Tucil2_13523162

Daftar Pustaka

Zhou Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," in IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, April 2004, doi: 10.1109/TIP.2003.819861.

International Telecommunication Union. "Recommendation ITU-R BT. 601-7." (2011).

Gunawan, Irwan Prasetya, Guson P. Kuntarto, and Berkah Iman Santoso. "Image Quality Assessment: Metode Standar dan Evaluasinya." (2022).

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

Jakob Jenkov, Java Fork and JoinForkPool, Java Concurrency
url:<https://jenkov.com/tutorials/java-util-concurrent/java-fork-and-join-forkjoinpool.html>