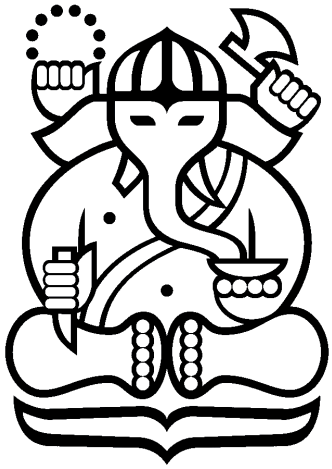
LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

Kompresi Gambar Dengan Metode Quadtree



Disusun oleh :

Ivan Wirawan 13523046

Carlo Angkisan 13523091

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESHA 10, BANDUNG 40132

2025

DAFTAR ISI

BAB I
DESKRIPSI MASALAH DAN ALGORITMA

1.1 Quadtree

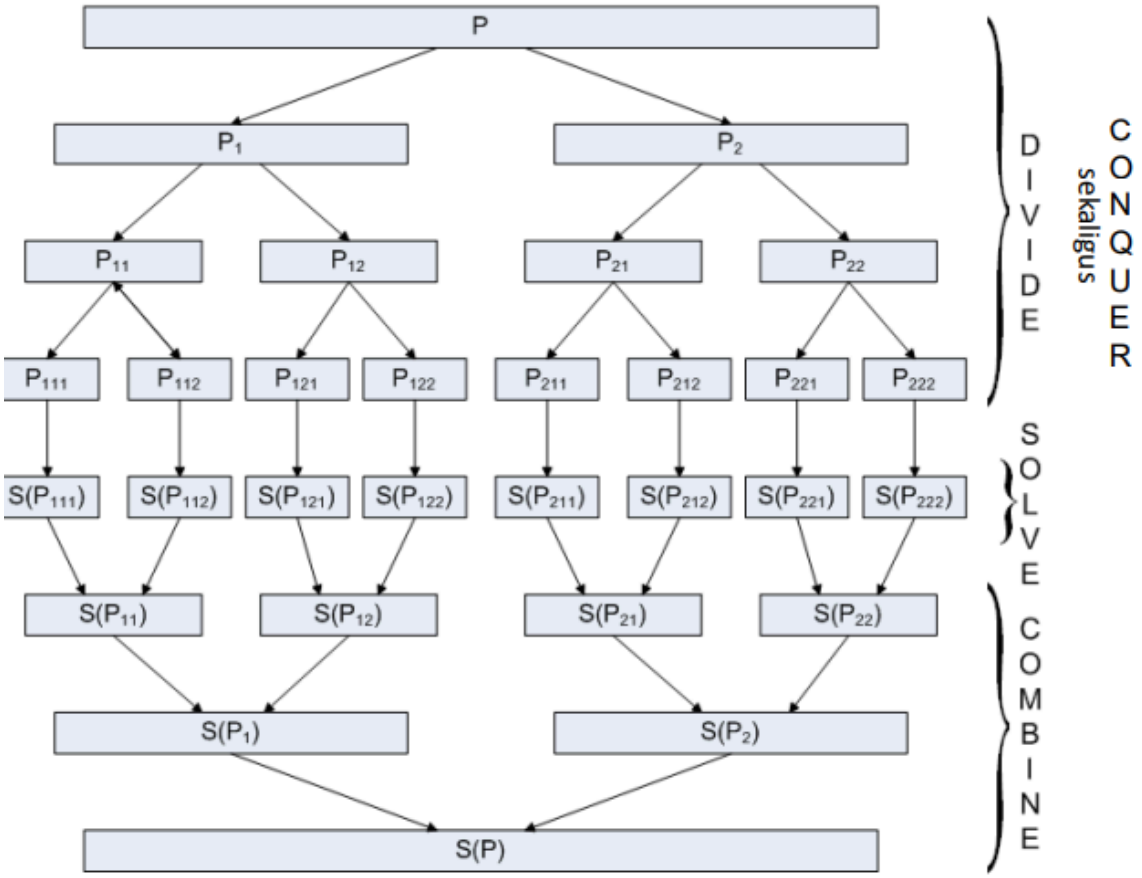
Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya 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.

1.2 Algoritma Divide and Conquer

Pada dasarnya, algoritma *Divide and Conquer* adalah sebuah konsep algoritma yang akan membagi sebuah persoalan besar menjadi beberapa persoalan kecil. Melalui pembagian tersebut, setiap upa-persoalan akan diselesaikan secara langsung maupun rekursif. Hasil dari penyelesaian tersebut akan digabungkan dengan solusi-solusi lainnya agar membentuk sebuah jawaban besar bagi permasalahan utamanya.

Tujuan utama dari algoritma *Divide and Conquer* adalah membagi sebuah persoalan besar menjadi beberapa upa-persoalan yang mirip dengan persoalan besar namun memiliki ukuran lebih kecil. Harapannya, dengan membagi sebuah persoalan besar menjadi kecil, maka persoalan kecil itu dapat diselesaikan langsung dengan mudah. Akan tetapi, apabila masih terlalu besar, maka dapat menggunakan penyelesaian secara rekursif. Setelah berhasil menyelesaikan seluruh upa-persoalan, masing-masing solusinya akan digabungkan hingga membentuk solusi utama.



Keterangan:
P = persoalan
S = solusi

Syarat dari penggunaan algoritma *Divide and Conquer* pada sebuah persoalan adalah objek dari masalah tersebut dapat dibagi menjadi beberapa bagian dengan karakteristik yang sama. Contohnya adalah tabel, matriks, eksponen, dan polinom. Apabila memiliki upa-persoalan yang dengan karakteristik yang sama namun lebih kecil, waktu dan sumber daya yang digunakan dalam menyelesaikan masalah akan lebih kecil sehingga meningkatkan efisiensi mesin.

1.3 Penerapan Algoritma Divide and Conquer pada Kompresi Gambar dengan Metode Quadtree

Adapun implementasi Algoritma Divide and Conquer pada kompresi gambar dengan metode quadtree menggunakan metode rekursif. Setelah program berhasil meminta masukan dari pengguna mengenai variabel-variabel yang akan digunakan seperti alamat gambar masukan dan keluaran, metode perhitungan, ambang batas, dan ukuran blok minimum, program akan menjalankan algoritma *Divide and Conquer* pada gambar yang telah diterima.

Sebelumnya, program akan mengubah bentuk gambar menjadi matriks berwarna. Dengan pengubahan ini, algoritma *Divide and Conquer* dapat dilakukan dengan pemecahan matriks menjadi blok-blok yang lebih kecil dari utamanya. Blok besar yang masih berkondisi tidak homogen (nilai error yang masih lebih besar daripada ambang batas yang diinginkan) dan ukurannya memungkinakan untuk pembagian, akan dibagi menjadi empat kuadran yaitu atas-kiri, atas-kanan, bawah-kiri, dan bawah-kanan. Dalam struktur Quadtree, awalnya satu matriks utama adalah sebuah simpul akar yang mewakili seluruh gambar, setelah pembagian, maka hasil pembagian itulah yang akan menjadi anak-anak dari simpul akar tersebut. Setelah itu, algoritma pembentukan Quadtree akan dipanggil kembali kepada pecahan-pecahan blok utama tadi. Hal ini akan membagi sebuah gambar utuh menjadi bagian-bagian yang dengan kehomogenan dan ukuran yang sesuai. Selagi pemecahan, data Quadtree akan terus mengalami pembaharuan sehingga terbentuk struktur data pohon yang berisikan alur pemecahan matriks.

Setelah kondisi pemecahan blok yaitu variasi warna dan ukuran yang kecil, maka proses pembagian akan berhenti pada setiap blok. Hasil pembagian tersebut akan direpresentasikan dengan sebuah nilai rata-rata warna. Hal ini akan melangsungkan sebuah kompresi pada gambar. Tujuannya adalah mengurangi variasi dari warna yang ada pada gambar sehingga ukurannya dapat lebih kecil.

Setelah kompresi dilakukan terhadap blok-blok yang memenuhi syarat, gambar akan dibentuk kembali. Proses penggabungan pada algoritma *Divide and Conquer* yang diaplikasikan dalam program ini memanfaatkan struktur pohon *Quadtree* yang ada. Rekonstruksi gambar akan dilakukan melalui himpunan blok-blok homogen yang terdapat pada simpul daun atau simpul paling bawah yang tidak memiliki anak lagi.

Sebagai sebuah kesimpulan, algoritma *Divide and Conquer* pada kompresi gambar dengan metode *Quadtree* bekerja dalam beberapa tahapan yaitu :

1. Inisialisasi : Setelah menghimpun seluruh data yang dibutuhkan melalui masukan dari pengguna, gambar akan dibentuk sebagai sebuah matriks. Selain itu, gambar juga akan dibentuk sebagai sebuah simpul akar yang mencakup seluruh gambar.
2. Pembagian (*Divide)* : Dalam membagi gambar menjadi 4 pecahan lainnya, algoritma akan menghitung nilai homogenitas dari setiap blok sebelum memecahnya. Apabila nilai errornya melebihi dari keinginan pengguna serta ukurannya memenuhi, maka blok tersebut akan dipecah menjadi 4 bagian selayaknya sebuah kuadran 1-4. Jika tidak memenuhi, maka blok tersebut akan menjadi simpul daun.. Apabila nilai error yang terdapat pada sebuah blok besar melebihi keinginan pengguna dan ukuran bloknya masih cukup besar. Proses inilah yang akan membagi sebuah masalah besar yaitu gambar besar, menjadi 4 bagian gambar yang lebih kecil. Proses pembagian ini akan dipanggil kembali untuk setiap pecahan blok dari gambar utamanya.
3. Penyelesaian (*Conquer*) : Setelah berhasil memecah gambar menjadi banyak blok kecil yang sudah tidak bisa diganti atau memiliki nilai error yang rendah, maka

algoritma ini akan menghitung nilai rata-rata warna dari blok tersebut yang menjadi nilai representasi warna blok tersebut. Proses penggantian warna dari yang heterogen menjadi sebuah warna rata-rata inilah yang melangsungkan sebuah kompresi pada gambar sehingga mengurangi ukuran berkas gambar tersebut.

4. Penggabungan (*Combine*) : Apabila proses pemecahan dan proses kompresi setiap blok kecil sudah selesai dilaksanakan, maka gambar tersebut akan direkonstruksi ulang. Dengan struktur pohon *Quadtree* yang telah dibentuk, maka gambar ini dapat dibentuk kembali dengan simpul-simpul daun yang dimilikinya. Hasil akhirnya adalah sebuah gambar dengan ukuran panjang dan lebar yang sama, namun nilai warna yang lebih homogen berdasarkan masukan yang diberikan pengguna.

Algoritma tersebut dapat dituliskan dalam Pseudocode sebagai berikut.

```
procedure QuadTreeCompression (input G : image, T : real, minSize :
integer, width : integer, height : integer) → QuadTreeNode
{ Menyelesaikan kompresi citra G menggunakan metode Quadtree berbasis
  algoritma divide and conquer

  Masukan  :
     G          : gambar berukuran width × height
     T          : threshold error untuk menentukan homogenitas
     minSize    : ukuran minimum blok (dalam piksel) agar masih boleh
dibagi

  Luaran   :
     root       : simpul akar Quadtree, mewakili struktur pohon hasil
kompresi

  Variabel  :
     root            : QuadtreeNode
     errorBlock      : real
     halfWidth       : integer
     halfHeight      : integer
}
  Algoritma:
     root ← CreateNode(G, x, y, width, height)
     errorBlock ← CalculateError(G)

     halfWidth ← width / 2
     halfHeight ← height / 2

     if ( (errorBlock ≤ T)
       OR (width(G) * height(G) < minSize) )
         OR width(G) <= 1 OR height(G) <= 1
            OR (((width(G)/2) * (height(G)/2)) < minSize) then
        root.isLeaf ← true
        root.averageColor ← CalculateAverageColor(G)

     else
```

```
      root.children[1] ← QuadTreeCompression(G, x,y, halfWidth,
      halfHeight, T, minSize)
      root.children[2] ← QuadTreeCompression(G, x + halfWidth, y,
      width - halfWidth, halfHeight, T, minSize)
      root.children[3] ← QuadTreeCompression(G, x,             y +
      halfHeight, halfWidth,         height - halfHeight, T, minSize)
      root.children[4] ← QuadTreeCompression(G, x + halfWidth, y +
      halfHeight, width - halfWidth, height - halfHeight, T, minSize)

      root.isLeaf ← false
    endif

    → root
}
```

BAB II
IMPLEMENTASI PROGRAM DALAM BAHASA JAVA

Dalam implementasi program Kompresi Gambar dengan Metode Quadtree dalam bahasa Java, pendekatan paradigma berorientasi objek (OOP) digunakan karena Java sendiri berbasis OOP. Dengan pendekatan ini, setiap komponen diorganisir dalam bentuk kelas (*class*) yang memiliki tanggung jawab spesifik, sehingga meningkatkan modularitas dan kemudahan dalam pengelolaan kode. Berikut adalah kelas-kelas yang digunakan dalam program ini.

2.1 Class QuadtreeNode

- Attribute

Nama	Tipe	Deskripsi
x	private int	Koordinat x (posisi horizontal) dari blok gambar yang diwakili oleh node.
y	private int	Koordinat y (posisi vertikal) dari blok gambar yang diwakili oleh node.
width	private int	Lebar blok gambar pada node ini.
height	private int	Tinggi blok gambar pada node ini.
imageMatrix	private ImageMatrix	Referensi ke objek ImageMatrix yang memuat data gambar; digunakan untuk menghitung rata-rata warna dari blok yang diberikan.
children	private QuadtreeNode[]	Array anak (4 node) yang merupakan hasil pembagian blok jika node tidak termasuk daun (leaf).
averageColor	private int	Nilai rata-rata warna (dalam format integer RGB) yang dihitung dari blok gambar pada node ini.
isLeaf	private boolean	Menandakan apakah node merupakan daun (tidak terbagi lebih lanjut) atau tidak.

- Method

Nama	Tipe	Deskripsi
QuadtreeNode	public	Konstruktor yang menginisialisasi node

	constructor	dengan blok gambar tertentu berdasarkan posisi (x,y), ukuran (width & height) dan menghitung nilai rata-rata warnanya.
calculateAverageColor	private int	Metode privat untuk menghitung rata-rata warna (RGB) pada blok gambar yang diwakili oleh node.
getChildren	public QuadtreeNode[]	Mengembalikan array anak (children) node; jika null, maka node tersebut merupakan daun.
getAverageColor	public int	Mengembalikan nilai rata-rata warna dari node.
getX	public int	Mengembalikan nilai koordinat x untuk blok gambar.
getY	public int	Mengembalikan nilai koordinat y untuk blok gambar.
getWidth	public int	Mengembalikan lebar blok gambar.
getHeight	public int	Mengembalikan tinggi blok gambar.
isLeaf	public boolean	Mengembalikan nilai boolean yang menunjukkan apakah node merupakan daun (tidak terbagi).
setChildren	public void	Menetapkan anak-anak node dan mengubah status node menjadi non-daun (isLeaf = false).

2.2 Class QuadtreeCompression

- Attribute

Nama	Tipe	Deskripsi
originalImage	private ImageMatrix	Objek ImageMatrix yang menyimpan data gambar asli.
threshold	private double	Nilai threshold awal atau optimal yang digunakan sebagai batas error saat kompresi.
minimumBlockSize	private int	Ukuran blok minimum yang harus dipenuhi

		agar blok gambar bisa dibagi lebih lanjut.
errorMethod	private int	Metode error yang dipilih (1: Variance, 2: Mean Absolute Deviation, 3: Max Pixel Difference, 4: Entropy, 5: SSIM).
targetCompressionPercentage	private double	Persentase target kompresi (nilai yang diharapkan) dalam satuan persentase.
inputPath	package-private String	Path file dari gambar input. (Tidak dideklarasikan sebagai private, bersifat default/package-private)

- Method

Nama	Tipe	Deskripsi
QuadtreeCompression	public constructor	Konstruktor untuk menginisialisasi objek QuadtreeCompression dengan parameter gambar asli, metode error, threshold, ukuran blok minimum, target kompresi, dan path input.
compress	public QuadtreeNode	Melakukan proses kompresi gambar dengan membangun Quadtree berdasarkan error yang dihitung; jika target kompresi diaktifkan, mencari threshold optimal terlebih dahulu.
buildQuadtree	private QuadtreeNode	Metode rekursif untuk membangun struktur Quadtree dari gambar dengan membagi blok gambar apabila error melebihi threshold (dan memenuhi ukuran minimum jika berlaku).
findOptimalThreshold	private double	Melakukan pencarian threshold optimal secara biner (binary search) agar kompresi gambar mendekati target compression percentage.
calculateTreeDepth	public int	Menghitung dan mengembalikan kedalaman maksimum Quadtree yang telah dibangun.
calculateNodeCount	public int	Menghitung jumlah node dalam struktur Quadtree secara keseluruhan.

getThreshold	public double	Mengembalikan nilai threshold yang digunakan (setelah optimasi jika berlaku).

2.3 Class ErrorMetrics

- Method

Nama	Tipe	Deskripsi
calculateVariance	private static double	Menghitung variance (ragam) total untuk setiap kanal warna (R, G, B) dalam sebuah blok gambar, dengan hasil rata-rata dari ketiga kanal.
calculateChannelVariance	private static double	Menghitung variance untuk kanal tertentu (0: R, 1: G, 2: B) dalam blok gambar.
calculateMean	private static double	Menghitung rata-rata nilai piksel untuk kanal tertentu dalam blok gambar.
calculateMeanAbsoluteDeviation	private static double	Menghitung nilai deviasi absolut rata-rata (Mean Absolute Deviation) untuk masing-masing kanal, dan mengembalikan rata-ratanya.
calculateMaxPixelDifference	private static double	Menghitung perbedaan maksimum piksel (selisih antara nilai maksimum dan minimum) untuk tiap kanal, kemudian mengembalikan rata-ratanya.
calculateEntropy	private static double	Menghitung entropi (pengukuran ketidakteraturan/distribusi) untuk masing-masing kanal dan mengembalikan rata-ratanya.
calculateSSIM	public static double	Menghitung indeks SSIM (Structural Similarity Index) antara gambar asli dan gambar terkompresi dengan bobot masing-masing kanal (R, G, B).
calculateChannelSSIM	private static double	Menghitung SSIM untuk sebuah kanal, menggunakan perhitungan rata-rata, variance, dan covariance antara gambar asli dan terkompresi.

calculateCovariance	private static double	Menghitung kovarians antara dua blok gambar untuk kanal tertentu, yang digunakan dalam perhitungan SSIM.
calculateError	public static double	Menghitung error berdasarkan metode yang dipilih (1: Variance, 2: Mean Absolute Deviation, 3: Max Pixel Difference, 4: Entropy, 5: SSIM).
calculateBlockSSIM	public static double	Menghitung nilai SSIM blok gambar dengan membuat blok kompresi sederhana berdasarkan rata-rata warna, lalu membandingkannya dengan gambar asli.

2.4 Class ImageMatrix

- Attribute

Nama	Tipe	Deskripsi
pixels	private int[][]	Array 3 dimensi yang menyimpan nilai piksel untuk masing-masing kanal (R, G, B) dengan indeks [kanal][x][y].
width	private int	Lebar gambar (jumlah kolom piksel).
height	private int	Tinggi gambar (jumlah baris piksel).

- Method

Nama	Tipe	Deskripsi
ImageMatrix(BufferedImage image)	public constructor	Konstruktor yang menginisialisasi objek ImageMatrix dengan mengambil data dari objek BufferedImage dan memecah nilai RGB tiap piksel ke array pixels.
ImageMatrix(int width, int height)	public constructor	Konstruktor yang membuat ImageMatrix dengan ukuran lebar dan tinggi tertentu, dan menginisialisasi array pixels.
toBufferedImage	public BufferedImage	Mengonversi ImageMatrix kembali menjadi objek BufferedImage dengan merakit nilai RGB dari array pixels.
getPixels	public	Mengembalikan array tiga dimensi pixels.

	int[][][]	
getWidth	public int	Mengembalikan lebar gambar.
getHeight	public int	Mengembalikan tinggi gambar.
getPixel	public int	Mengambil nilai piksel untuk kanal tertentu pada koordinat (x, y).
setPixel	public void	Menetapkan nilai piksel untuk kanal tertentu pada koordinat (x, y).
getAverageColor	public int	Menghitung rata-rata warna dalam blok gambar yang ditentukan dengan menghitung rata-rata nilai untuk tiap kanal, kemudian menggabungkannya menjadi satu nilai integer RGB.

2.5 Class IO

- Method

Nama	Tipe	Deskripsi
readImage	public static ImageMatrix	Membaca file gambar dari path yang diberikan dan mengembalikan objek ImageMatrix.
calcFileSize	public static long	Menghitung dan mengembalikan ukuran file dalam byte berdasarkan path file yang diberikan.
calcCompressionPercentage	public static double	Menghitung persentase kompresi dengan membandingkan ukuran file asli dan file terkompresi.
reconstructImageFromQuadtree	public static ImageMatrix	Mereconstruct (menyusun kembali) gambar dari struktur Quadtree yang dihasilkan dengan menggunakan informasi posisi dan ukuran dari tiap node.
writeCompressedImage	public static void	Menulis (menyimpan) ImageMatrix sebagai file gambar ke output path yang ditentukan.
reconstructGIFFromQuadtree	public static List<BufferedImage>	Membuat list frame BufferedImage dengan merekonstruksi gambar dari Quadtree untuk tiap tingkat (depth) sebagai persiapan

		pembuatan file GIF animasi.
reconstructNodeForGIF	public static void	Versi modifikasi dari metode rekonstruksi node yang digunakan untuk pembuatan GIF, dengan mempertimbangkan batas depth maksimum.
createCompressionGif	public static void	Membuat file GIF yang berisi frame-frame kompresi dengan pengaturan delay time antar frame.
isValidImagePath	public static boolean	Memvalidasi apakah path file gambar valid (ekstensi yang sesuai dan file tersebut ada).
getExtension	private static String	Mengambil ekstensi file dari path (dipanggil secara internal oleh metode validasi).
hasSameExtension	public static boolean	Memeriksa apakah dua file memiliki ekstensi yang sama.
isValidThreshold	public static boolean	Memvalidasi apakah nilai threshold berada dalam rentang yang sesuai berdasarkan metode error yang dipilih.

2.6 Class Main

- Attribute

Nama	Tipe	Deskripsi
RESET	private static final String	Kode warna ANSI untuk mereset warna output di terminal.
RED	private static final String	Kode warna ANSI untuk warna merah.
GREEN	private static final String	Kode warna ANSI untuk warna hijau.
YELLOW	private static final String	Kode warna ANSI untuk warna kuning.

BLUE	private static final String	Kode warna ANSI untuk warna biru.
PURPLE	private static final String	Kode warna ANSI untuk warna ungu.
WHITE	private static final String	Kode warna ANSI untuk warna putih.

- Method

Nama	Tipe	Deskripsi
main	public static void	Method utama yang mengeksekusi aplikasi, menangani input pengguna, validasi file, pemilihan metode error, pengaturan threshold, kompresi gambar melalui Quadtree, dan output hasil serta GIF jika diminta.

2.7 Class UI

- Attribute

Nama	Tipe	Deskripsi
RESET	private static final String	Kode warna ANSI untuk mereset warna output di terminal.
RED	private static final String	Kode warna ANSI untuk warna merah.
GREEN	private static final String	Kode warna ANSI untuk warna hijau.
YELLOW	private static final String	Kode warna ANSI untuk warna kuning.

BLUE	private static final String	Kode warna ANSI untuk warna biru.
PURPLE	private static final String	Kode warna ANSI untuk warna ungu.
WHITE	private static final String	Kode warna ANSI untuk warna putih.

- Method

Nama	Tipe	Deskripsi
printLogo	public static	Menampilkan logo aplikasi dengan efek warna dan format ASCII art.
printHeader	public static	Menampilkan header aplikasi dengan judul tool Quadtree Image Compression.
printProgressBar	public static void	Menampilkan progress bar di terminal untuk menunjukkan perkembangan proses kompresi berdasarkan persentase yang telah dicapai.
printErrorMethodInfo	public static void	Menampilkan informasi dan pilihan terkait metode pengukuran error (Variance, MAD, Max Pixel Difference, Entropy, SSIM).
printResultMenu	public static void	Menampilkan hasil kompresi, termasuk waktu eksekusi, ukuran file asli dan kompresi, kedalaman pohon, total node, dan informasi file output (gambar & GIF jika ada).
getThresholdRange	public static String	Mengembalikan string yang menunjukkan range nilai threshold yang valid berdasarkan metode error yang dipilih.
getMethodName	public static String	Mengembalikan nama metode error dalam bentuk string sesuai pilihan (misal: "Variance", "Mean Absolute Deviation", dll.).

BAB III
SOURCE CODE DAN STRUKTUR PROGRAM

3.1 Repositori Github

Berikut tautan repository github kelompok kami untuk Tugas Kecil 2 IF2211 Strategi Algoritma :
https://github.com/ivan-wirawan/Tucil2_13523046_13523091

3.2 Struktur Program

Program memiliki struktur folder sebagai berikut.

- **src** : berisi source code program dalam bentuk file **.java**.
- **bin** : berisi hasil kompilasi dalam bentuk file **.class** yang siap dijalankan.
- **test** : berisi hasil pengujian yang dicantumkan pada laporan.
- **doc** : berisi laporan tugas kecil dan dokumentasi program.

3.3 Source Code Program

3.3.1. Class Main

```java
public class Main {
    private static final String RESET = "\u001B[0m";
    private static final String RED = "\u001B[31m";
    private static final String GREEN = "\u001B[32m";
    private static final String YELLOW = "\u001B[33m";
    private static final String BLUE = "\u001B[34m";
    private static final String PURPLE = "\u001B[35m";
    private static final String WHITE = "\u001B[1;37m";

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            UI.printLogo();
            UI.printHeader();
            String inputPath;
            while (true) {
                System.out.print(WHITE + "\n 📁 Absolute path of input image: " + RESET);
                inputPath = scanner.nextLine().trim();
                if (IO.isValidImagePath(inputPath)) {
                    System.out.println(GREEN + " ✅ Valid image found!" + RESET);
                    break;
                }
                System.out.println(RED + " ❌ Invalid image path or format." + RESET);
            }

            int errorMethod = 0;
            while (true) {
                UI.printErrorMethodInfo();

                System.out.print(WHITE + "\n 🖍️  Choose method (1-5): " + RESET);

                try {
                    errorMethod = Integer.parseInt(scanner.nextLine());
                    if (errorMethod >= 1 && errorMethod <= 5) {
                        System.out.println(GREEN + " ✅ Method selected: " +
UI.getMethodName(errorMethod) + RESET);
                        break;
                    }
                } catch (Exception ignored) {
                }

                System.out.println(RED + " ❌ Please select a valid number (1-4)." +
RESET);
            }
```

```
double threshold;
            while (true) {
                System.out.print(
                        WHITE + "\n 🔍 Enter threshold " + YELLOW +
UI.getThresholdRange(errorMethod) + RESET + ": ");
                try {
                    threshold = Double.parseDouble(scanner.nextLine());
                    if (IO.isValidThreshold(threshold, errorMethod)) {
                        System.out.println(GREEN + " ✅ Threshold set!" + RESET);
                        break;
                    }
                } catch (Exception ignored) {
                }
                System.out.println(RED + " ❌ Threshold out of range for selected
method." + RESET);
            }

            int minBlock;
            while (true) {
                System.out.print(WHITE + "\n 🔪 Minimum block size: " + RESET);
                try {
                    minBlock = Integer.parseInt(scanner.nextLine());
                    if (minBlock > 0) {
                        System.out.println(GREEN + " ✅ Minimum block size set to " +
minBlock + RESET);
                        break;
                    }
                } catch (Exception ignored) {
                }
                System.out.println(RED + " ❌ Must be a positive integer." + RESET);
            }

            double compressionTarget;
            while (true) {
                System.out.print(WHITE + "\n 🎯 Target compression percentage " + YELLOW
+ "(0.0-1.0)" + RESET + ": ");
                try {
                    compressionTarget = Double.parseDouble(scanner.nextLine());
                    if (compressionTarget >= 0.0 && compressionTarget <= 1.0) {
                        if (compressionTarget == 0.0) {
                            System.out.println(BLUE + " ℹ️ Target compression disabled" +
RESET);
                        } else {
                            System.out.println(GREEN + " ✅ Target set to " +
(compressionTarget * 100) + "%" + RESET);
                        }
                        break;
                    }
                } catch (Exception ignored) {
                }
                System.out.println(RED + " ❌ Must be between 0.0 and 1.0." + RESET);
            }
```

```
String outputPath;
            while (true) {
                System.out.print(WHITE + "\n 💾 Absolute path for compressed image: " + RESET);
                outputPath = scanner.nextLine().trim();

                if (outputPath.equals(inputPath)) {
                    System.out.println(RED + " ❌ Output path cannot be the same as input path." + RESET);
                    continue;
                }

                File outputFile = new File(outputPath);
                File outputDir = outputFile.getParentFile();
                if (outputDir != null && !outputDir.exists()) {
                    System.out
                            .println(RED + " ❌ Output directory doesn't exist: " + outputDir.getAbsolutePath() + RESET);
                    continue;
                }

                if (outputFile.exists()) {
                    System.out.println(RED + " ❌ Output file already exists: " + outputPath + RESET);
                    continue;
                }

                if (IO.hasSameExtension(inputPath, outputPath)) {
                    System.out.println(GREEN + " ✅ Output path set!" + RESET);
                    break;
                }
                System.out.println(RED + " ❌ Output file must have the same extension as input." + RESET);
            }

            boolean saveGif = false;
            String gifPath = null;
            while (true) {
                System.out.print(WHITE + "\n 🎬 Do you want to save compression GIF? " + YELLOW + "(y/n)" + RESET + ": ");
                String gifChoice = scanner.nextLine().trim().toLowerCase();
                if (gifChoice.equals("y")) {
                    saveGif = true;
                    while (true) {
                        System.out.print(WHITE + "\n 📁 Enter absolute path to save GIF: " + RESET);
                        gifPath = scanner.nextLine().trim();

                        if (gifPath.equals(inputPath) || gifPath.equals(outputPath)) {
                            System.out.println(RED + " ❌ GIF path cannot be the same as input or output path." + RESET);
                            continue;
                        }
```

```
                        File gifFile = new File(gifPath);
                        File gifDir = gifFile.getParentFile();
                        if (gifDir != null && !gifDir.exists()) {
                            System.out.println(
                                    RED + " ❌ GIF directory doesn't exist: " +
gifDir.getAbsolutePath() + RESET);
                            continue;
                        }

                        if (gifFile.exists()) {
                            System.out.println(RED + " ❌ GIF file already exists: " +
gifPath + RESET);
                            continue;
                        }

                        if (gifPath.toLowerCase().endsWith(".gif")) {
                            System.out.println(GREEN + " ✅ GIF path set!" + RESET);
                            break;
                        }
                        System.out.println(RED + " ❌ Output file must have a .gif
extension." + RESET);
                    }
                    break;
                } else if (gifChoice.equals("n")) {
                    saveGif = false;
                    break;
                }
                System.out.println(RED + " ❌ Please type 'y' or 'n'." + RESET);
            }
```

```
System.out.println("\n" + PURPLE + "\n ⚙️  Processing image compression..." + RESET);
            UI.printProgressBar(0);

            long startTime = System.nanoTime();

            ImageMatrix original = IO.readImage(inputPath);
            int width = original.getWidth();
            int height = original.getHeight();
            UI.printProgressBar(20);

            QuadtreeCompression compressor = new QuadtreeCompression(
                    original, errorMethod, threshold, minBlock, compressionTarget*100,
inputPath);
            UI.printProgressBar(40);


            QuadtreeNode root = compressor.compress();
            UI.printProgressBar(70);

            int treeDepth = compressor.calculateTreeDepth(root);
            int nodeCount = compressor.calculateNodeCount(root);

            UI.printProgressBar(80);

            ImageMatrix compressed = IO.reconstructImageFromQuadtree(root, width, height);

            IO.writeCompressedImage(compressed, outputPath);
            UI.printProgressBar(90);

            if (saveGif && gifPath != null) {
                System.out.println(BLUE + "\n\n 🎬 Generating compression GIF..." +
RESET);
                List<BufferedImage> frames = IO.reconstructGIFFromQuadtree(root, width,
height,
                        treeDepth);
                IO.createCompressionGif(frames, gifPath, 500);
            }
            UI.printProgressBar(100);

            long endTime = System.nanoTime();
            double executionTime = (endTime - startTime) / 1_000_000.0;
            double compressionPercentage = IO.calcCompressionPercentage(inputPath,
outputPath);

            threshold = compressor.getThreshold();
            UI.printResultMenu(treeDepth, nodeCount, executionTime, compressionPercentage,
inputPath, outputPath, gifPath, saveGif, compressionTarget, threshold);

            System.out.println("\n" + PURPLE + "Thank you for using Quadtree Image
Compression!" + RESET);

        } catch (IOException e) {
            System.err.println(RED + " ❌ Error processing image: " + e.getMessage() +
RESET);
        } finally {
            scanner.close();
        }
    }
}
```

3.3.2. Class QuadtreeNode

```java
public class QuadtreeNode {
    private int x, y;
    private int width, height;
    private ImageMatrix imageMatrix;
    private QuadtreeNode[] children;
    private int averageColor;
    private boolean isLeaf;

    public QuadtreeNode(ImageMatrix imageMatrix, int x, int y, int width, int height) {
        this.imageMatrix = imageMatrix;
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.children = new QuadtreeNode[4];
        this.isLeaf = true;
        this.averageColor = calculateAverageColor();
    }
    private int calculateAverageColor() {
        return imageMatrix.getAverageColor(x, y, width, height);
    }

    public QuadtreeNode[] getChildren() {
        return children;
    }
    public int getAverageColor() {
        return averageColor;
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
    public int getWidth() {
        return width;
    }
    public int getHeight() {
        return height;
    }
    public boolean isLeaf() {
        return isLeaf;
    }
    public void setChildren(QuadtreeNode[] children) {
        this.children = children;
        this.isLeaf = false;
    }
}

```

3.3.4. Class QuadtreeCompression

```java
public class QuadtreeCompression {
    private ImageMatrix originalImage;
    private double threshold;
    private int minimumBlockSize;
    private int errorMethod;
    private double targetCompressionPercentage;
    String inputPath;

    public QuadtreeCompression(ImageMatrix originalImage, int errorMethod,
            double threshold, int minimumBlockSize, double targetCompressionPercentage,
String inputPath) {
        this.originalImage = originalImage;
        this.errorMethod = errorMethod;
        this.threshold = threshold;
        this.minimumBlockSize = minimumBlockSize;
        this.targetCompressionPercentage = targetCompressionPercentage;
        this.inputPath = inputPath;
    }

    public QuadtreeNode compress() {
        // Bonus: target Compression Percentage
        boolean useMinimumBlockSize = true;
        if (targetCompressionPercentage > 0) {
            this.threshold = findOptimalThreshold();
            useMinimumBlockSize = false;
        }

        return buildQuadtree(originalImage, 0, 0, originalImage.getWidth(),
originalImage.getHeight(),
                0, this.threshold, useMinimumBlockSize);
    }

private QuadtreeNode buildQuadtree(ImageMatrix originalImage, int x, int y, int width,int
height, int currentDepth, double threshold, boolean useMinimumBlockSize) {
        QuadtreeNode node = new QuadtreeNode(originalImage, x, y, width, height);

        if (width <= 1 || height <= 1) {
            return node;
        }

        double error = ErrorMetrics.calculateError(originalImage, x, y, width, height,
errorMethod);

        int halfWidth = width / 2;
        int halfHeight = height / 2;

        boolean shouldDivide;

        if (useMinimumBlockSize) {
            shouldDivide = (error > threshold) && (width * height >= minimumBlockSize)
                    && (halfWidth * halfHeight >= minimumBlockSize);
        } else {
            shouldDivide = (error > threshold);
        }
```

```
if (shouldDivide) {
            QuadtreeNode[] children = new QuadtreeNode[4];

            // Top-left
            children[0] = buildQuadtree(originalImage, x, y, halfWidth, halfHeight,
currentDepth + 1,
                    threshold, useMinimumBlockSize);

            // Top-right
            children[1] = buildQuadtree(originalImage, x + halfWidth, y, width -
halfWidth, halfHeight,
                    currentDepth + 1, threshold, useMinimumBlockSize);

            // Bottom-left
            children[2] = buildQuadtree(originalImage, x, y + halfHeight, halfWidth,
                    height - halfHeight, currentDepth + 1, threshold,
useMinimumBlockSize);

            // Bottom-right
            children[3] = buildQuadtree(originalImage, x + halfWidth, y + halfHeight,
                    width - halfWidth, height - halfHeight, currentDepth + 1, threshold,
useMinimumBlockSize);

            node.setChildren(children);
        }

        return node;
    }

    private double findOptimalThreshold() {
        double minThreshold = 0.0;
        double maxThreshold;

        if (errorMethod == 1) { // Variance
            maxThreshold = 16256.25;
        } else if (errorMethod == 2) { // Mean Absolute Deviation
            maxThreshold = 127.5;
        } else if (errorMethod == 3) { // Max Pixel Difference
            maxThreshold = 255.0;
        } else if (errorMethod == 4) { // Entropy
            maxThreshold = 8.0;
        } else if (errorMethod == 5) { // SSIM
            maxThreshold = 1.0;
        } else {
            maxThreshold = 255.0;
        }

        double currentThreshold;
        double currentCompression;
        double epsilon = 0.001;
        int maxIterations = 50;
        int iterations = 0;
```

```java
String originalExtension = inputPath.substring(inputPath.lastIndexOf('.'));
        String tempDirPath = "test/temp";
        String tempPath = tempDirPath + "/temp" + originalExtension;

        File tempDir = new File(tempDirPath);
        if (!tempDir.exists()) {
            if (!tempDir.mkdirs()) {
                System.err.println("Gagal membuat direktori: " + tempDirPath);
                return maxThreshold / 2;
            }
        }

        File tempFile = new File(tempPath);

        try {
            while (minThreshold < maxThreshold && iterations < maxIterations) {
                iterations++;
                currentThreshold = (minThreshold + maxThreshold) / 2;

                QuadtreeNode root = buildQuadtree(originalImage, 0, 0,
                        originalImage.getWidth(), originalImage.getHeight(), 0,
currentThreshold, false);

                ImageMatrix reconstructedImage = IO.reconstructImageFromQuadtree(root,
                        originalImage.getWidth(), originalImage.getHeight());

                IO.writeCompressedImage(reconstructedImage, tempPath);

                currentCompression = IO.calcCompressionPercentage(inputPath, tempPath);

                if (Math.abs(currentCompression - targetCompressionPercentage) < epsilon)
{
                    if (tempFile.exists()) {
                        tempFile.delete();
                    }
                    return currentThreshold;
                }

                if (currentCompression > targetCompressionPercentage) {
                    maxThreshold = currentThreshold;
                } else {
                    minThreshold = currentThreshold;
                }
            }

            double finalThreshold = (minThreshold + maxThreshold) / 2;

            if (tempFile.exists()) {
                tempFile.delete();
            }

            return finalThreshold;
```

```
} catch (Exception e) {
            System.err.println("Error during threshold optimization: " + e.getMessage());
            e.printStackTrace();

            if (tempFile.exists()) {
                tempFile.delete();
            }

            return (minThreshold + maxThreshold) / 2;
        } finally {
            if (tempFile.exists()) {
                tempFile.delete();
            }
        }
    }

    public int calculateTreeDepth(QuadtreeNode root) {
        if (root == null) {
            return -1;
        }

        if (root.getChildren() == null) {
            return 0;
        }

        int topLeftDepth = calculateTreeDepth(root.getChildren()[0]);
        int topRightDepth = calculateTreeDepth(root.getChildren()[1]);
        int bottomLeftDepth = calculateTreeDepth(root.getChildren()[2]);
        int bottomRightDepth = calculateTreeDepth(root.getChildren()[3]);

        int maxDepth = topLeftDepth;
        if (topRightDepth > maxDepth)
            maxDepth = topRightDepth;
        if (bottomLeftDepth > maxDepth)
            maxDepth = bottomLeftDepth;
        if (bottomRightDepth > maxDepth)
            maxDepth = bottomRightDepth;

        return maxDepth + 1;
    }

    public int calculateNodeCount(QuadtreeNode root) {
        if (root == null) {
            return 0;
        }

        int count = 1;

        if (root.getChildren() != null) {
            int topLeftCount = calculateNodeCount(root.getChildren()[0]);
            int topRightCount = calculateNodeCount(root.getChildren()[1]);
            int bottomLeftCount = calculateNodeCount(root.getChildren()[2]);
            int bottomRightCount = calculateNodeCount(root.getChildren()[3]);

            count += topLeftCount + topRightCount + bottomLeftCount + bottomRightCount;
        }

        return count;
    }

    public double getThreshold() {
        return threshold;
    }
}
```

3.3.5. Class ErrorMetrics

```java
public class ErrorMetrics {
    private static double calculateVariance(ImageMatrix image, int x, int y, int width, int height) {
        double totalVariance = 0.0;
        for (int channel = 0; channel < 3; channel++) {
            totalVariance += calculateChannelVariance(image, x, y, width, height, channel);
        }

        return totalVariance / 3.0;
    }

    private static double calculateChannelVariance(ImageMatrix image, int x, int y, int width, int height, int channel) {
        double mean = calculateMean(image, x, y, width, height, channel);
        double channelVariance = 0.0;

        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                int pixelValue = image.getPixel(channel, x + i, y + j);
                channelVariance += Math.pow(pixelValue - mean, 2);
            }
        }

        return channelVariance / (width * height);
    }

    private static double calculateMean(ImageMatrix image, int x, int y, int width, int height, int channel) {
        double sum = 0.0;
        int pixelCount = width * height;

        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                sum += image.getPixel(channel, x + i, y + j);
            }
        }

        return sum / pixelCount;
    }

    private static double calculateMeanAbsoluteDeviation(ImageMatrix image, int x, int y, int width, int height) {
        double totalMAD = 0.0;
        for (int channel = 0; channel < 3; channel++) {
            double mean = calculateMean(image, x, y, width, height, channel);
            double channelMAD = 0.0;

            for (int i = 0; i < width; i++) {
                for (int j = 0; j < height; j++) {
                    int pixelValue = image.getPixel(channel, x + i, y + j);
                    channelMAD += Math.abs(pixelValue - mean);
                }
            }

            totalMAD += channelMAD / (width * height);
        }

        return totalMAD / 3.0;
    }
```

```java
    private static double calculateMaxPixelDifference(ImageMatrix image, int x, int y, int width, int height) {
        double totalMaxDiff = 0.0;
        for (int channel = 0; channel < 3; channel++) {
            int maxValue = 0;
            int minValue = 255;

            for (int i = 0; i < width; i++) {
                for (int j = 0; j < height; j++) {
                    int pixelValue = image.getPixel(channel, x + i, y + j);

                    if (pixelValue > maxValue) {
                        maxValue = pixelValue;
                    }
                    if (pixelValue < minValue) {
                        minValue = pixelValue;
                    }
                }

            }

            totalMaxDiff += (maxValue - minValue);
        }

        return totalMaxDiff / 3.0;
    }

    private static double calculateEntropy(ImageMatrix image, int x, int y, int width, int height) {
        double totalEntropy = 0.0;

        for (int channel = 0; channel < 3; channel++) {
            int[] histogram = new int[256];

            for (int i = 0; i < width; i++) {
                for (int j = 0; j < height; j++) {
                    int pixelValue = image.getPixel(channel, x + i, y + j);
                    histogram[pixelValue]++;
                }
            }

            double channelEntropy = 0.0;
            int totalPixels = width * height;

            for (int count : histogram) {
                if (count > 0) {
                    double probability = (double) count / totalPixels;
                    channelEntropy -= probability * (Math.log(probability) / Math.log(2));
                }
            }

            totalEntropy += channelEntropy;
        }

        return totalEntropy / 3.0;
    }
```

```java
    public static double calculateSSIM(ImageMatrix original, ImageMatrix compressed, int x, int y, int width, int height) {
        double weightR = 0.299;
        double weightG = 0.587;
        double weightB = 0.114;
        double ssimR = calculateChannelSSIM(original, compressed, x, y, width, height, 0);

        double ssimG = calculateChannelSSIM(original, compressed, x, y, width, height, 1);

        double ssimB = calculateChannelSSIM(original, compressed, x, y, width, height, 2);


        return weightR * ssimR + weightG * ssimG + weightB * ssimB;
    }

    private static double calculateChannelSSIM(ImageMatrix original, ImageMatrix compressed, int x, int y, int width, int height, int channel) {
        double meanOriginal = calculateMean(original, x, y, width, height, channel);
        double meanCompressed = calculateMean(compressed, x, y, width, height, channel);
        double varianceOriginal = calculateChannelVariance(original, x, y, width, height, channel);
        double varianceCompressed = calculateChannelVariance(compressed, x, y, width, height, channel);


        double C1 = 6.5025; // (0.01 * 255)^2
        double C2 = 58.5225; // (0.03 * 255)^2

        double ssimC = ((2 * meanOriginal * meanCompressed + C1)
                * (2 * calculateCovariance(original, compressed, x, y, width, height, channel) + C2))
                / ((meanOriginal * meanOriginal + meanCompressed * meanCompressed + C1)
                        * (varianceOriginal + varianceCompressed + C2));

        return ssimC;
    }

    private static double calculateCovariance(ImageMatrix img1, ImageMatrix img2, int x, int y, int width, int height, int channel) {
        double meanImg1 = calculateMean(img1, x, y, width, height, channel);
        double meanImg2 = calculateMean(img2, x, y, width, height, channel);

        double covariance = 0.0;
        int pixelCount = width * height;

        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                double pixel1 = img1.getPixel(channel, x + i, y + j);
                double pixel2 = img2.getPixel(channel, x + i, y + j);

                covariance += (pixel1 - meanImg1) * (pixel2 - meanImg2);
            }
        }

        return covariance / pixelCount;
    }

```

```java
    public static double calculateError(ImageMatrix original, int x, int y, int width, int height, int errorMethod) {
        // 1: Variance
        // 2: Mean Absolute Deviation
        // 3: Max Pixel Difference
        // 4: Entropy
        // 5: SSIM (Not Fixed)

        double error = 0.0;
        if (errorMethod == 1){
            error = calculateVariance(original, x, y, width, height);
        } else if (errorMethod == 2){
            error = calculateMeanAbsoluteDeviation(original, x, y, width, height);
        } else if (errorMethod == 3){
            error = calculateMaxPixelDifference(original, x, y, width, height);
        } else if (errorMethod == 4){
            error = calculateEntropy(original, x, y, width, height);
        } else if (errorMethod == 5){
            error = 1.0 - calculateBlockSSIM(original, x, y, width, height);
        } else {
            throw new IllegalArgumentException("Invalid error method");
        }

        return error;
    }

    public static double calculateBlockSSIM(ImageMatrix original, int x, int y, int width, int height) {
        ImageMatrix compressedBlock = new ImageMatrix(width, height);
        int avgColor = original.getAverageColor(x, y, width, height);
        int avgR = (avgColor >> 16) & 0xFF;
        int avgG = (avgColor >> 8) & 0xFF;
        int avgB = avgColor & 0xFF;

        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                compressedBlock.setPixel(0, i, j, avgR);
                compressedBlock.setPixel(1, i, j, avgG);
                compressedBlock.setPixel(2, i, j, avgB);
            }
        }

        return calculateSSIM(original, compressedBlock, 0, 0, width, height);
    }
}

```

3.3.6. Class ImageMatrix

```java
public class ImageMatrix {
    private int[][][] pixels;
    private int width;
    private int height;

    public ImageMatrix(BufferedImage image) {
        this.width = image.getWidth();
        this.height = image.getHeight();
        this.pixels = new int[3][width][height];

        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                int rgb = image.getRGB(x, y);
                pixels[0][x][y] = (rgb >> 16) & 0xFF;
                pixels[1][x][y] = (rgb >> 8) & 0xFF;
                pixels[2][x][y] = rgb & 0xFF;
            }
        }
    }

    public ImageMatrix(int width, int height) {
        this.width = width;
        this.height = height;
        this.pixels = new int[3][width][height];
    }

    public BufferedImage toBufferedImage() {
        BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);

        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                int r = pixels[0][x][y];
                int g = pixels[1][x][y];
                int b = pixels[2][x][y];
                int rgb = (r << 16) | (g << 8) | b;
                image.setRGB(x, y, rgb);
            }
        }

        return image;
    }
```

```java
    public int[][][] getPixels() {
        return pixels;
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public int getPixel(int channel, int x, int y) {
        return pixels[channel][x][y];
    }

    public void setPixel(int channel, int x, int y, int value) {
        pixels[channel][x][y] = value;
    }

    public int getAverageColor(int x, int y, int blockWidth, int blockHeight) {
        long sumR = 0, sumG = 0, sumB = 0;
        int pixelCount = blockWidth * blockHeight;

        for (int i = 0; i < blockWidth; i++) {
            for (int j = 0; j < blockHeight; j++) {
                sumR += pixels[0][x + i][y + j];
                sumG += pixels[1][x + i][y + j];
                sumB += pixels[2][x + i][y + j];
            }
        }

        int avgR = (int) (sumR / pixelCount);
        int avgG = (int) (sumG / pixelCount);
        int avgB = (int) (sumB / pixelCount);

        return (avgR << 16) | (avgG << 8) | avgB;
    }
}
```

3.3.7. Class IO

```java
public class IO {
    public static ImageMatrix readImage(String imagePath) throws IOException {
        BufferedImage bufferedImage = ImageIO.read(new File(imagePath));
        return new ImageMatrix(bufferedImage);
    }

    public static long calcFileSize(String imagePath) {
        File file = new File(imagePath);
        return file.length();
    }

    public static double calcCompressionPercentage(String originalPath, String compressedPath) {
        long originalSize = calcFileSize(originalPath);
        long compressedSize = calcFileSize(compressedPath);

        if (originalSize == 0) {
            throw new IllegalArgumentException("[ERROR] : Original file size cannot be zero.");
        }

        double compressionRatio = 1.0 - ((double) compressedSize / originalSize);
        return compressionRatio * 100;
    }

    // Output Image
    public static ImageMatrix reconstructImageFromQuadtree(QuadtreeNode root, int width, int height) {
        ImageMatrix reconstructedImage = new ImageMatrix(width, height);
        reconstructNode(root, reconstructedImage);
        return reconstructedImage;
    }

    private static void reconstructNode(QuadtreeNode node, ImageMatrix image) {
        if (node.isLeaf()) {
            int avgColor = node.getAverageColor();
            for (int x = node.getX(); x < node.getX() + node.getWidth(); x++) {
                for (int y = node.getY(); y < node.getY() + node.getHeight(); y++) {
                    int r = (avgColor >> 16) & 0xFF;
                    int g = (avgColor >> 8) & 0xFF;
                    int b = avgColor & 0xFF;

                    image.setPixel(0, x, y, r);
                    image.setPixel(1, x, y, g);
                    image.setPixel(2, x, y, b);
                }
            }
        } else {
            // Top-left
            reconstructNode(node.getChildren()[0], image);

            // Top-right
            reconstructNode(node.getChildren()[1], image);

            // Bottom-left
            reconstructNode(node.getChildren()[2], image);

            // Bottom-right
            reconstructNode(node.getChildren()[3], image);
        }
    }
```

```java
    public static void writeCompressedImage(ImageMatrix image, String outputPath) throws IOException {
        BufferedImage bufferedImage = image.toBufferedImage();
        File outputFile = new File(outputPath);
        String format = outputPath.substring(outputPath.lastIndexOf('.') + 1);
        ImageIO.write(bufferedImage, format, outputFile);
    }

    // Output GIF
    public static List<BufferedImage> reconstructGIFFromQuadtree(QuadtreeNode root, int width, int height, int maxDepth) {
        List<BufferedImage> frames = new ArrayList<>();

        for (int i = 0; i <= maxDepth; i++) {
            ImageMatrix frameImage = new ImageMatrix(width, height);
            IO.reconstructNodeForGIF(root, frameImage, 0, i);
            frames.add(frameImage.toBufferedImage());
        }

        return frames;
    }

    public static void reconstructNodeForGIF(QuadtreeNode node, ImageMatrix image, int currentDepth, int maxDepth) {
        if (node.isLeaf() || currentDepth >= maxDepth) {
            int avgColor = node.getAverageColor();
            for (int x = node.getX(); x < node.getX() + node.getWidth(); x++) {
                for (int y = node.getY(); y < node.getY() + node.getHeight(); y++) {
                    int r = (avgColor >> 16) & 0xFF;
                    int g = (avgColor >> 8) & 0xFF;
                    int b = avgColor & 0xFF;

                    image.setPixel(0, x, y, r);
                    image.setPixel(1, x, y, g);
                    image.setPixel(2, x, y, b);
                }
            }
        } else {
            // Top-left
            reconstructNodeForGIF(node.getChildren()[0], image, currentDepth + 1, maxDepth);

            // Top-right
            reconstructNodeForGIF(node.getChildren()[1], image, currentDepth + 1, maxDepth);

            // Bottom-left
            reconstructNodeForGIF(node.getChildren()[2], image, currentDepth + 1, maxDepth);

            // Bottom-right
            reconstructNodeForGIF(node.getChildren()[3], image, currentDepth + 1, maxDepth);
        }
    }
```

```java
    public static void createCompressionGif(List<BufferedImage> frames, String outputPath,
int delayTime) throws IOException {
        ImageWriter gifWriter = ImageIO.getImageWritersBySuffix("gif").next();
        ImageWriteParam imageWriteParam = gifWriter.getDefaultWriteParam();
        ImageOutputStream output = ImageIO.createImageOutputStream(new File(outputPath));
        gifWriter.setOutput(output);

        gifWriter.prepareWriteSequence(null);

        for (int i = 0; i < frames.size(); i++) {
            BufferedImage frame = frames.get(i);
            IIOMetadata metadata = gifWriter.getDefaultImageMetadata(new
ImageTypeSpecifier(frame), imageWriteParam);
            String metaFormat = metadata.getNativeMetadataFormatName();
            IIOMetadataNode root = (IIOMetadataNode) metadata.getAsTree(metaFormat);
            IIOMetadataNode graphicsControlExtensionNode = new
IIOMetadataNode("GraphicControlExtension");
            graphicsControlExtensionNode.setAttribute("disposalMethod", "none");
            graphicsControlExtensionNode.setAttribute("userInputFlag", "FALSE");
            graphicsControlExtensionNode.setAttribute("transparentColorFlag", "FALSE");
            graphicsControlExtensionNode.setAttribute("delayTime",
String.valueOf(delayTime / 10));
            graphicsControlExtensionNode.setAttribute("transparentColorIndex", "0");
            root.appendChild(graphicsControlExtensionNode);

            // Loop forever
            if (i == 0) {
                IIOMetadataNode appExtensions = new
IIOMetadataNode("ApplicationExtensions");
                IIOMetadataNode appExtension = new
IIOMetadataNode("ApplicationExtension");
                appExtension.setAttribute("applicationID", "NETSCAPE");
                appExtension.setAttribute("authenticationCode", "2.0");

                byte[] loopForever = new byte[] { 0x1, 0x0, 0x0 };
                appExtension.setUserObject(loopForever);
                appExtensions.appendChild(appExtension);
                root.appendChild(appExtensions);
            }

            metadata.setFromTree(metaFormat, root);
            gifWriter.writeToSequence(new IIOImage(frame, null, metadata),
imageWriteParam);
        }

        gifWriter.endWriteSequence();
        output.close();
    }
```

```java
    // Validation IO
    public static boolean isValidImagePath(String path) {
        String ext = getExtension(path).toLowerCase();
        if (!ext.equals("jpg") && !ext.equals("jpeg") && !ext.equals("png")) {
            System.out.println(" ❌ Invalid image format! Allowed: .jpg, .jpeg, .png");
            return false;
        }

        File file = new File(path);
        if (!file.exists() || !file.isFile()) {
            System.out.println(" ❌ File does not exist.");
            return false;
        }

        return true;
    }


    private static String getExtension(String path) {
        int i = path.lastIndexOf('.');
        return (i > 0) ? path.substring(i + 1).toLowerCase() : "";
    }

    public static boolean hasSameExtension(String path1, String path2) {
        return getExtension(path1).equals(getExtension(path2));
    }

    public static boolean isValidThreshold(double value, int method) {
        if (method == 1) {
            return value >= 0 && value <= 16256.25;
        } else if (method == 2) {
            return value >= 0 && value <= 127.5;
        } else if (method == 3) {
            return value >= 0 && value <= 255;
        } else if (method == 4) {
            return value >= 0 && value <= 8;
        } else if (method == 5) {
            return value >= 0 && value <= 1;
        } else {
            return false;
        }
    }
}
```

3.3.8. Class UI

```
public class UI {
    private static final String RESET = "\u001B[0m";
    private static final String RED = "\u001B[31m";
    private static final String GREEN = "\u001B[32m";
    private static final String YELLOW = "\u001B[33m";
    private static final String BLUE = "\u001B[34m";
    private static final String PURPLE = "\u001B[35m";
    private static final String CYAN = "\u001B[36m";
    private static final String WHITE = "\u001B[1;37m";

    public static void printLogo() {
        System.out.println(CYAN + "  ┌──────────────────────────────────────┐" + RESET);

        System.out.println(CYAN + "  |                                      |" + RESET);
        System.out.println(CYAN + "  |    " + PURPLE + "[redacted]  " + GREEN + "[redacted]    " + BLU
[redacted]  "+ YELLOW + "[redacted]       " + CYAN + "|" + RESET);
        System.out.println(CYAN + "  |    " + PURPLE + "[redacted]  " + GREEN + "[redacted]    " + BLUE
[redacted] "+ YELLOW + "[redacted]      " + CYAN + "|" + RESET);
        System.out.println(CYAN + "  |    " + PURPLE + "[redacted]  " + GREEN + "[redacted]    " + BLUE
"[redacted] "+ YELLOW + "[redacted]      " + CYAN + "|" + RESET);
        System.out.println(CYAN + "  |    " + PURPLE + "[redacted]  " + GREEN + "[redacted]    " + BLUE
[redacted] "+ YELLOW + " [redacted]      " + CYAN + "|" + RESET);
        System.out.println(CYAN + "  |    " + PURPLE + "[redacted]  " + GREEN + "[redacted] " +
"[redacted]  [redacted] "+ YELLOW + "[redacted]       " + CYAN + "|" + RESET);
        System.out.println(CYAN + "  |    " + PURPLE + "     [redacted]  " + CYAN + "
RESET);
        System.out.println(CYAN + "  |                                      |" + RESET);
        System.out.println(CYAN + "  |    " + PURPLE + "[redacted] " + GREEN + "[redacted]    "
RED+ "[redacted]             " + CYAN + "|" + RESET);
        System.out.println(CYAN + "  |       " + PURPLE + "[redacted]    " + GREEN + "[redacted]  [redacted] " + BL
"[redacted]                " + CYAN + "|" + RESET);
        System.out.println(CYAN + "  |       " + PURPLE + "[redacted]    " + GREEN + "[redacted]  " + B
"[redacted]              " + CYAN + "|" + RESET);
        System.out.println(CYAN + "  |       " + PURPLE + "[redacted]    " + GREEN + "[redacted]  [redacted] " + BL
"[redacted]                " + CYAN + "|" + RESET);
        System.out.println(CYAN + "  |       " + PURPLE + "[redacted]    " + GREEN + "[redacted]  [redacted] " + BL
"[redacted]             " + CYAN + "|" + RESET);
        System.out.println(CYAN + "  |                                      |" + RESET);
        System.out.println(CYAN + "  |            " + YELLOW + "[ IMAGE COMPRESSION ]" + CYAN
RESET);
        System.out.println(CYAN + "  └──────────────────────────────────────┘" + RESET);

    }

    public static void printHeader() {
        System.out.println(CYAN + "╔═══════════════════════════════════════════════════════════════════════════════

        System.out.println(CYAN + "‖" + WHITE+ "                    QUADTREE IMAGE COMPRESSI
" + CYAN + "‖" + RESET);
        System.out.println(CYAN + "╚═══════════════════════════════════════════════════════════════════════════════


    }
```

```java
    public static void printProgressBar(int percent) {
        final int width = 40;
        int completed = width * percent / 100;
        String[] spinChars = new String[] { "⠷", "⠯", "⠟", "⠻", "⠽", "⠾", "⠷", "⠯", "⠟", "⠻" };
        String spinChar = spinChars[(percent / 5) % spinChars.length];

        StringBuilder bar = new StringBuilder();
        bar.append("\r").append(BLUE).append(spinChar).append(" ").append(WHITE).append("[");

        for (int i = 0; i < width; i++) {
            if (i < completed) {
                bar.append(GREEN).append("■").append(RESET);
            } else {
                bar.append(" ");
            }
        }

        bar.append(WHITE).append("] ").append(CYAN).append(String.format("%3d%%", percent)).append(RESET);
        System.out.print(bar);
        System.out.flush();
    }
```

```java
    public static void printErrorMethodInfo() {
        System.out.println("\n" + CYAN+ "╔═══════════════════════════════════════════════════════════

        System.out.println(CYAN + "║                      " + WHITE + "📊 ERROR MEASUREMENT ME
║" + RESET);
        System.out.println(CYAN + "╠═══════════════════════════════════════════════════════════

        System.out.println(CYAN + "║  " + BLUE + "1. " + RESET + "Variance
16256.25)" + CYAN + "                    ║" + RESET);
        System.out.println(CYAN + "║  " + BLUE + "2. " + RESET + "Mean Absolute Deviation
127.5)" + CYAN + "                       ║" + RESET);
        System.out.println(CYAN + "║  " + BLUE + "3. " + RESET + "Max Pixel Difference
255)" + CYAN + "                         ║" + RESET);
        System.out.println(CYAN + "║  " + BLUE + "4. " + RESET + "Entropy
+ CYAN + "                            ║" + RESET);
        System.out.println(CYAN + "║  " + BLUE + "5. " + RESET + "Structural Similarity Index
+ CYAN + "                            ║" + RESET);
        System.out.println(CYAN + "╚═══════════════════════════════════════════════════════════


    }

    public static void printResultMenu(int treeDepth, int nodeCount,double executionTime, doub
String inputPath, String outputPath, String gifPath, boolean saveGif, double compressionTarget
        String compressionColor = compressionPercentage > 50 ? GREEN : (compressionPercentage

        System.out.println("\n\n" + PURPLE + "╔═════════════════════════════════════╗" + RESET)

        System.out.println(PURPLE + "║" + WHITE + "           COMPRESSION RESULTS

        System.out.println(PURPLE + "╠════════════════════╦═════════════════╣" + RESET);

        System.out.printf(PURPLE + "║" + RESET + " %-22s" + PURPLE + "║" + CYAN + " %-18s" + P
RESET,"Execution Time", String.format("%.4f ms", executionTime));
        System.out.println(PURPLE + "╠════════════════════╬═════════════════╣" + RESET);

        System.out.printf(PURPLE + "║" + RESET + " %-22s" + PURPLE + "║" + YELLOW + " %-18s" +
RESET,"Original Size", String.format("%d bytes", IO.calcFileSize(inputPath)));
        System.out.println(PURPLE + "╠════════════════════╬═════════════════╣" + RESET);

        System.out.printf(PURPLE + "║" + RESET + " %-22s" + PURPLE + "║" + GREEN + " %-18s" +
RESET,"Compressed Size", String.format("%d bytes", IO.calcFileSize(outputPath)));
        System.out.println(PURPLE + "╠════════════════════╬═════════════════╣" + RESET);

        System.out.printf(PURPLE + "║" + RESET + " %-22s" + PURPLE + "║" + compressionColor +
RESET,"Compression Rate", String.format("%.4f%%", compressionPercentage));
        System.out.println(PURPLE + "╠════════════════════╬═════════════════╣" + RESET);

        System.out.printf(PURPLE + "║" + RESET + " %-22s" + PURPLE + "║" + BLUE + " %-18s" + P
RESET,"Tree Depth", String.format("%d", treeDepth));
        System.out.println(PURPLE + "╠════════════════════╬═════════════════╣" + RESET);

        System.out.printf(PURPLE + "║" + RESET + " %-22s" + PURPLE + "║" + BLUE + " %-18s" + P
RESET,"Total Nodes", String.format("%d", nodeCount));

        if (compressionTarget != 0){
            System.out.println(PURPLE + "╠════════════════════╬═════════════════╣" + RESET);

            System.out.printf(PURPLE + "║" + RESET + " %-22s" + PURPLE + "║" + BLUE + " %-18s"
RESET,"Optimal Threshold", String.format("%.4f", threshold));
        }

        System.out.println(PURPLE + "╚════════════════════╩═════════════════╝" + RESET);

        System.out.println(WHITE + "\n📄 Output Files:" + RESET);
        System.out.println(GREEN + " ✅ Compressed Image: " + RESET + outputPath);
        if (saveGif && gifPath != null) {
            System.out.println(GREEN + " ✅ Compression GIF: " + RESET + gifPath);
        }
    }
```

```
    public static String getThresholdRange(int method) {
        switch (method) {
            case 1:
                return "(0-16256.25)";
            case 2:
                return "(0-127.5)";
            case 3:
                return "(0-255)";
            case 4:
                return "(0.0-8.0)";
            case 5:
                return "(0.0-1.0)";
            default:
                return "";
        }
    }

    public static String getMethodName(int method) {
        switch (method) {
            case 1:
                return "Variance";
            case 2:
                return "Mean Absolute Deviation";
            case 3:
                return "Max Pixel Difference";
            case 4:
                return "Entropy";
            case 5:
                return "SSIM";
            default:
                return "Unknown";
        }
    }
}
```

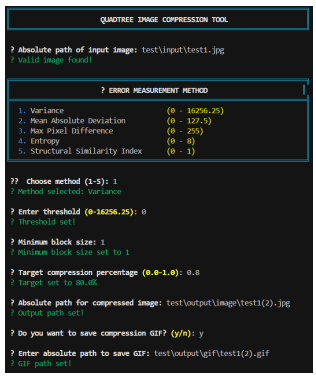
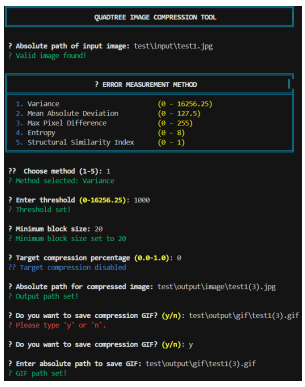
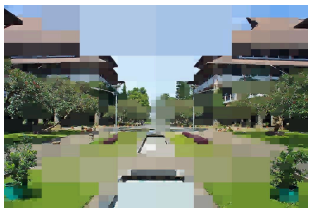
BAB IV
HASIL PENGUJIAN DAN ANALISIS

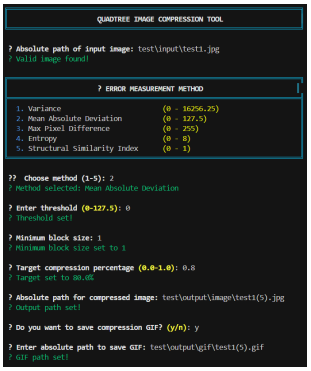
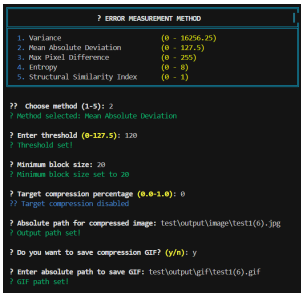
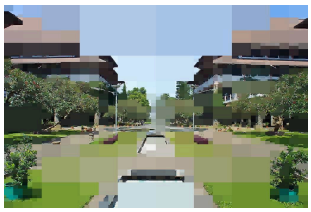
4.1 Hasil Pengujian

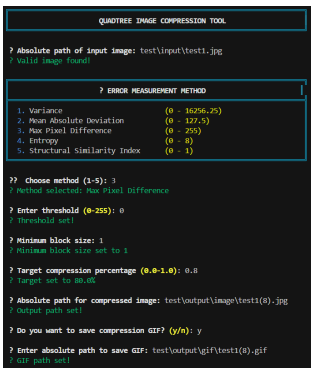
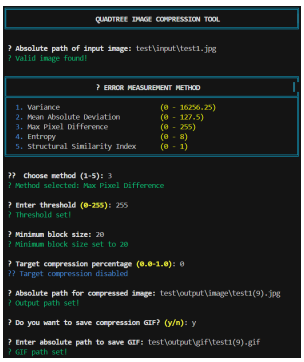
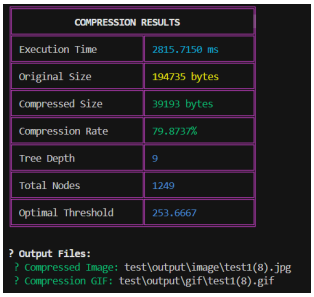
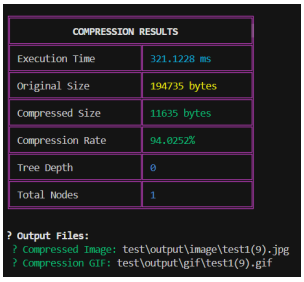
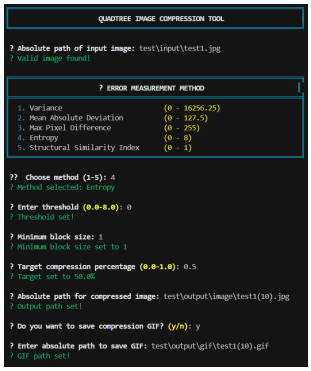
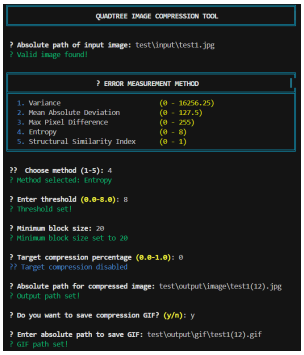
4.1.1 Kasus Eksperimen 1

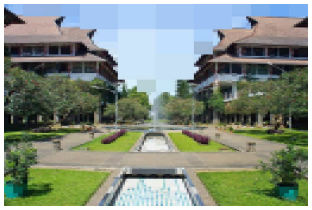
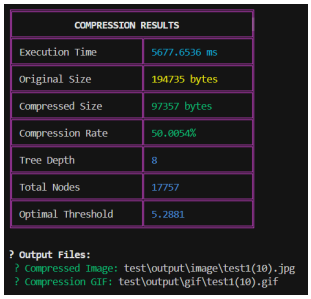
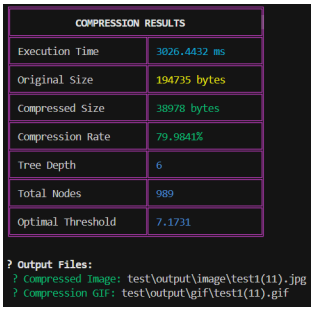
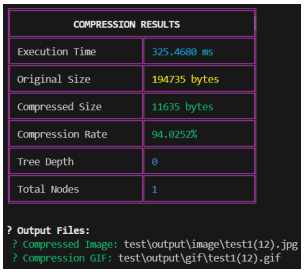
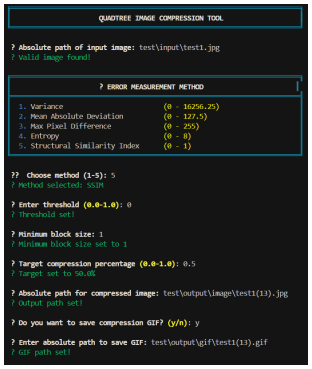
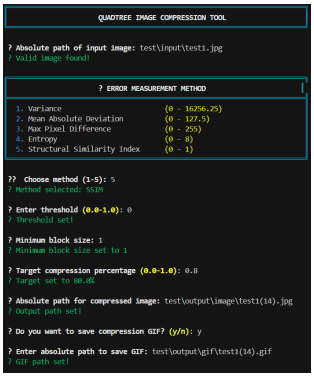
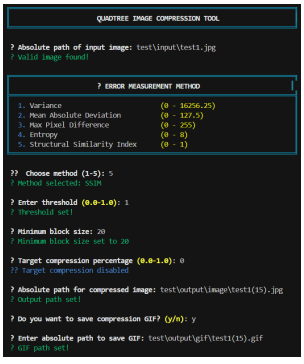
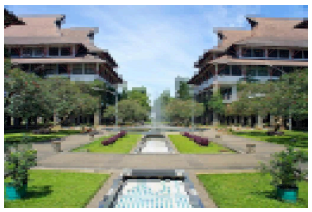
Gambar Asli :

Hasil Kompresi :

Metode	Ukuran Blok Minimum : 1	Ukuran Blok Minimum : 1	Ukuran Blok Minimum : 20
	Target Kompresi : 50%	Target Kompresi : 80%	Target Kompresi : 0%
Input :			
Variansi			

Nilai Ambang Batas :	0	0	1000
Output :	COMPRESSION RESULTS Execution Time: 1981.2501 ms Original Size: 194735 bytes Compressed Size: 97369 bytes Compression Rate: 49.9992% Tree Depth: 10 Total Nodes: 49525 Optimal Threshold: 2445.7798 ? Output Files: ? Compressed Image: test\output\image\test1(1).jpg ? Compression GIF: test\output\gif\test1(1).gif	COMPRESSION RESULTS Execution Time: 3322.7386 ms Original Size: 194735 bytes Compressed Size: 39033 bytes Compression Rate: 79.9558% Tree Depth: 10 Total Nodes: 4713 Optimal Threshold: 3675.4486 ? Output Files: ? Compressed Image: test\output\image\test1(2).jpg ? Compression GIF: test\output\gif\test1(2).gif	COMPRESSION RESULTS Execution Time: 1085.6454 ms Original Size: 194735 bytes Compressed Size: 88356 bytes Compression Rate: 54.6276% Tree Depth: 7 Total Nodes: 12525 ? Output Files: ? Compressed Image: test\output\image\test1(3).jpg ? Compression GIF: test\output\gif\test1(3).gif
Input :			
Mean Absolute Deviation			
Nilai Ambang Batas :	0	0	120
Output :	COMPRESSION RESULTS Execution Time: 3937.4134 ms Original Size: 194735 bytes Compressed Size: 97532 bytes Compression Rate: 49.9155% Tree Depth: 10 Total Nodes: 53809 Optimal Threshold: 39.2669 ? Output Files: ? Compressed Image: test\output\image\test1(4).jpg ? Compression GIF: test\output\gif\test1(4).gif	COMPRESSION RESULTS Execution Time: 3421.1890 ms Original Size: 194735 bytes Compressed Size: 38500 bytes Compression Rate: 80.2295% Tree Depth: 10 Total Nodes: 4681 Optimal Threshold: 49.8182 ? Output Files: ? Compressed Image: test\output\image\test1(5).jpg ? Compression GIF: test\output\gif\test1(5).gif	COMPRESSION RESULTS Execution Time: 338.7270 ms Original Size: 194735 bytes Compressed Size: 11635 bytes Compression Rate: 94.0252% Tree Depth: 0 Total Nodes: 1 ? Output Files: ? Compressed Image: test\output\image\test1(6).jpg ? Compression GIF: test\output\gif\test1(6).gif

Input :			
Max Pixel Difference			
Nilai Ambang Batas :	0	0	255
Output :			
Input :			

Entropy			
Nilai Ambang Batas :	0	0	8
Output :			
Input :			
Structural Similarity Index			
Nilai Ambang Batas :	0	0	1

Output :	COMPRESSION RESULTS Execution Time: 10889.1044 ms Original Size: 194735 bytes Compressed Size: 97409 bytes Compression Rate: 49.9787% Tree Depth: 10 Total Nodes: 43789 Optimal Threshold: 0.7032 ? Output Files: ? Compressed Image: test\output\image\test1(13).jpg ? Compression GIF: test\output\gif\test1(13).gif	COMPRESSION RESULTS Execution Time: 6797.0638 ms Original Size: 194735 bytes Compressed Size: 38004 bytes Compression Rate: 80.4842% Tree Depth: 6 Total Nodes: 1081 Optimal Threshold: 0.9873 ? Output Files: ? Compressed Image: test\output\image\test1(14).jpg ? Compression GIF: test\output\gif\test1(14).gif	COMPRESSION RESULTS Execution Time: 372.5254 ms Original Size: 194735 bytes Compressed Size: 11635 bytes Compression Rate: 94.0252% Tree Depth: 0 Total Nodes: 1 ? Output Files: ? Compressed Image: test\output\image\test1(15).jpg ? Compression GIF: test\output\gif\test1(15).gif

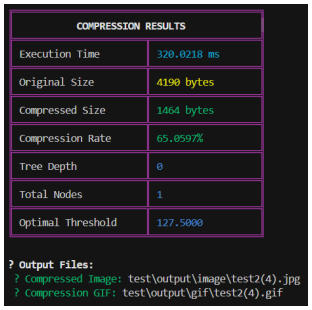
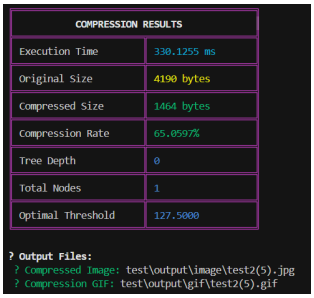
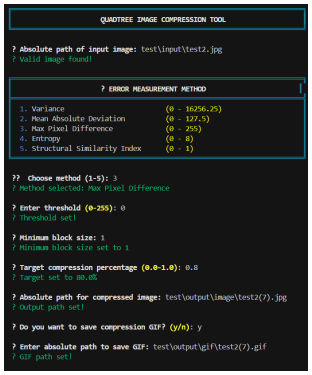
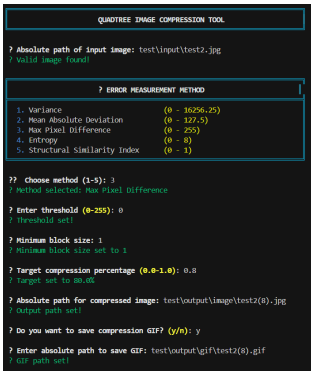
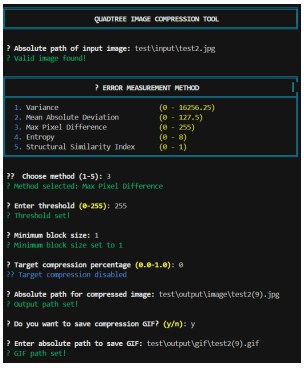
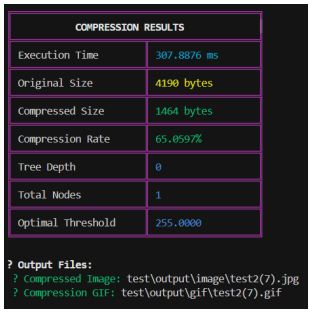
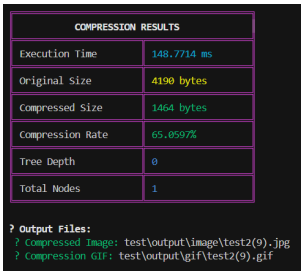
4.1.2 Kasus Eksperimen 2

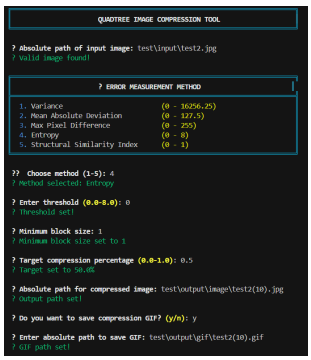
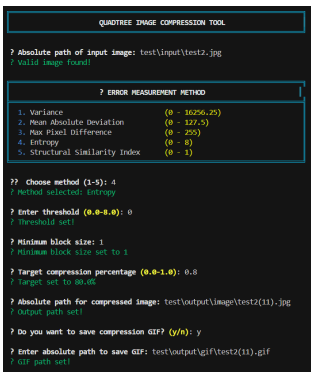
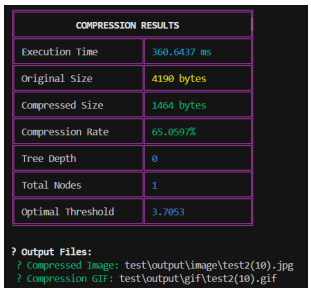
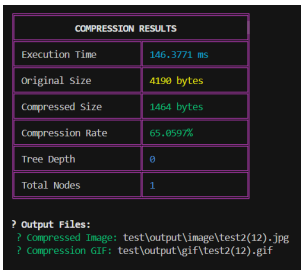
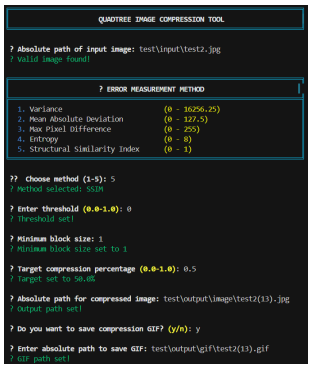
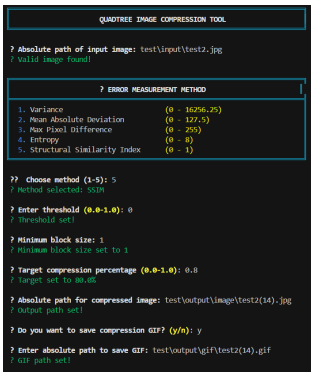
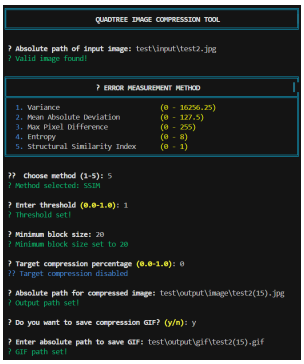
Gambar Asli :

Hasil Kompresi :

Metode	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 20
	Target Kompresi : 50%	Target Kompresi : 80%	Target Kompresi : 0%
Input :	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test2.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 1 ? Method selected: Variance ? Enter threshold (0-16256.25): 0 ? Threshold set! ? Minimum block size: 1 ? Minimum block size set to 1 ? Target compression percentage (0.0-1.0): 0.5 ? Target set to 50.0% ? Absolute path for compressed image: test\output\image\test2(1).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\image\test2(1).jpg ? GIF path cannot be the same as input or output path. ? Enter absolute path to save GIF: test\output\gif\test2(2).gif ? GIF path set!	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test2.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 1 ? Method selected: Variance ? Enter threshold (0-16256.25): 0 ? Threshold set! ? Minimum block size: 1 ? Minimum block size set to 1 ? Target compression percentage (0.0-1.0): 0.8 ? Target set to 80.0% ? Absolute path for compressed image: test\output\image\test2(2).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test2(2).gif ? GIF file already exists: test\output\gif\test2(2).gif ? Enter absolute path to save GIF: test\output\gif\test2(2).gif ? GIF path set!	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test2.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 1 ? Method selected: Variance ? Enter threshold (0-16256.25): 1000 ? Threshold set! ? Minimum block size: 20 ? Minimum block size set to 20 ? Target compression percentage (0.0-1.0): 0 ?? Target compression disabled ? Absolute path for compressed image: test\output\image\test2(3).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test2(3).gif ? GIF path set!

Variansi			
Nilai Ambang Batas :	0	0	1000
Output :	COMPRESSION RESULTS Execution Time: 352.8664 ms Original Size: 4190 bytes Compressed Size: 1464 bytes Compression Rate: 65.0597% Tree Depth: 0 Total Nodes: 1 Optimal Threshold: 2865.2359 ? Output Files: ? Compressed Image: test\output\image\test2(1).jpg ? Compression GIF: test\output\gif\test2(2).gif	COMPRESSION RESULTS Execution Time: 331.6970 ms Original Size: 4190 bytes Compressed Size: 1464 bytes Compression Rate: 65.0597% Tree Depth: 0 Total Nodes: 1 Optimal Threshold: 16256.2500 ? Output Files: ? Compressed Image: test\output\image\test2(2).jpg ? Compression GIF: test\output\gif\test2(2).gif	COMPRESSION RESULTS Execution Time: 191.4652 ms Original Size: 4190 bytes Compressed Size: 3909 bytes Compression Rate: 6.7064% Tree Depth: 5 Total Nodes: 181 ? Output Files: ? Compressed Image: test\output\image\test2(3).jpg ? Compression GIF: test\output\gif\test2(3).gif
Input :	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test2.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 2 ? Method selected: Mean Absolute Deviation ? Enter threshold (0-127.5): 0 ? Threshold set! ? Minimum block size: 1 ? Minimum block size set to 1 ? Target compression percentage (0.0-1.0): 0.8 ? Target set to 80.0% ? Absolute path for compressed image: test\output\image\test2(4).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test2(4).gif ? GIF path set!	? Absolute path of input image: test\input\test2.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 2 ? Method selected: Mean Absolute Deviation ? Enter threshold (0-127.5): 0 ? Threshold set! ? Minimum block size: 1 ? Minimum block size set to 1 ? Target compression percentage (0.0-1.0): 0.8 ? Target set to 80.0% ? Absolute path for compressed image: test\output\image\test2(5).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test2(5).gif ? GIF path set!	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test2.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 2 ? Method selected: Mean Absolute Deviation ? Enter threshold (0-127.5): 120 ? Threshold set! ? Minimum block size: 20 ? Minimum block size set to 20 ? Target compression percentage (0.0-1.0): 0 ?? Target compression disabled ? Absolute path for compressed image: test\output\image\test2(6).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test2(6).gif ? GIF path set!
Mean Absolute Deviation			
Nilai Ambang Batas :	0	0	120

Output :			
Input :			
Max Pixel Difference			
Nilai Ambang Batas :	0	0	255
Output :			

Input :			
Entropy			
Nilai Ambang Batas :	0	0	8
Output :			
Input :			

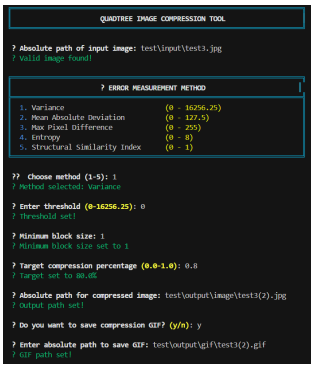
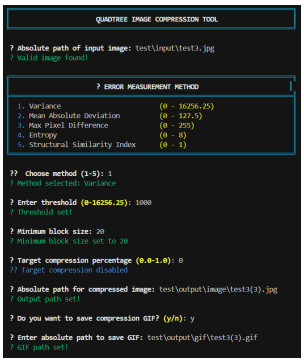
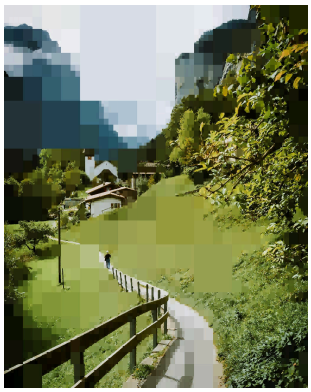
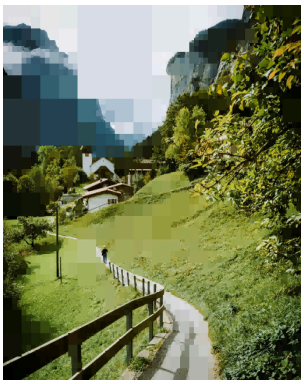
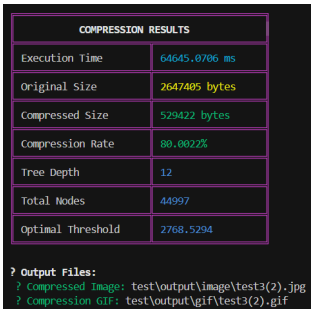
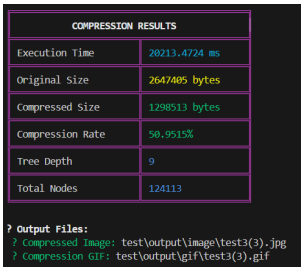
Structural Similarity Index			
Nilai Ambang Batas :	0	0	1
Output :	COMPRESSION RESULTS Execution Time: 503.7501 ms Original Size: 4190 bytes Compressed Size: 1847 bytes Compression Rate: 55.9189% Tree Depth: 2 Total Nodes: 9 Optimal Threshold: 0.9292 ? Output Files: ? Compressed Image: test\output\image\test2(13).jpg ? Compression GIF: test\output\gif\test2(13).gif	COMPRESSION RESULTS Execution Time: 393.4729 ms Original Size: 4190 bytes Compressed Size: 1464 bytes Compression Rate: 65.0597% Tree Depth: 0 Total Nodes: 1 Optimal Threshold: 1.0000 ? Output Files: ? Compressed Image: test\output\image\test2(14).jpg ? Compression GIF: test\output\gif\test2(14).gif	COMPRESSION RESULTS Execution Time: 143.2767 ms Original Size: 4190 bytes Compressed Size: 1464 bytes Compression Rate: 65.0597% Tree Depth: 0 Total Nodes: 1 ? Output Files: ? Compressed Image: test\output\image\test2(15).jpg ? Compression GIF: test\output\gif\test2(15).gif

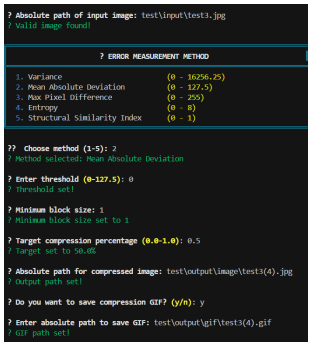
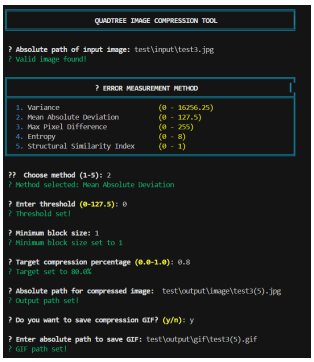
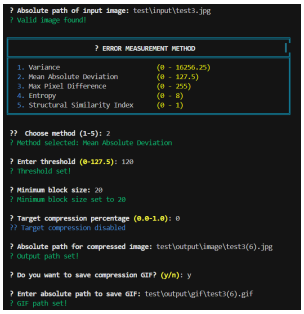
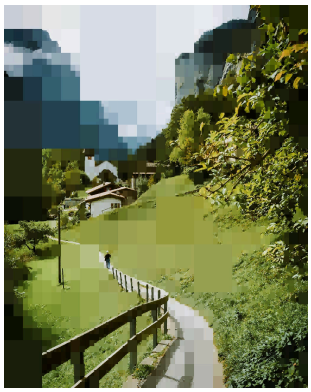
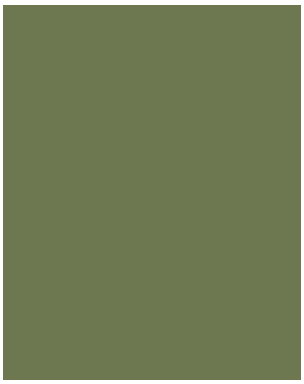
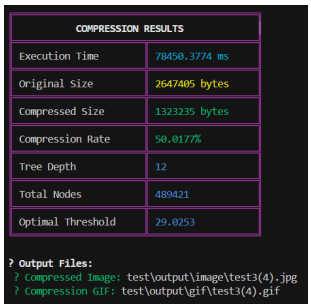
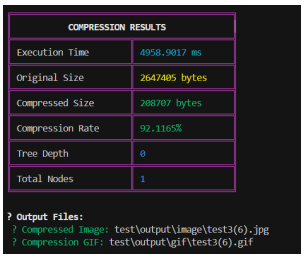
4.1.3 Kasus Eksperimen 3

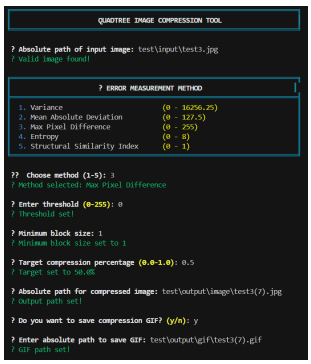
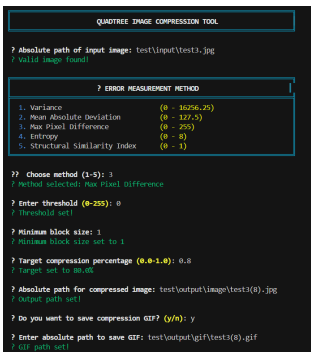
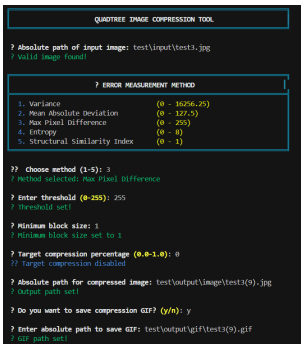
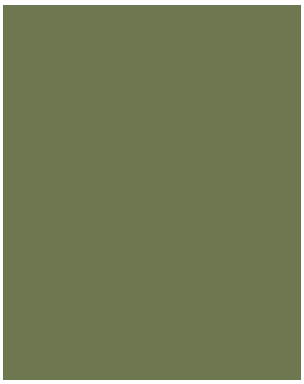
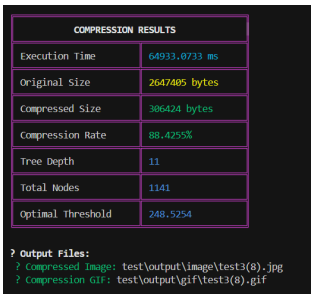
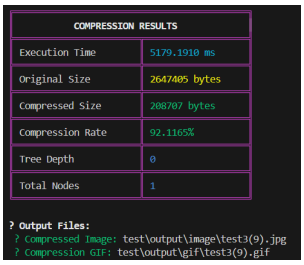
Gambar Asli :

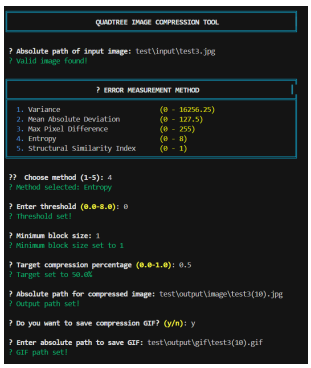
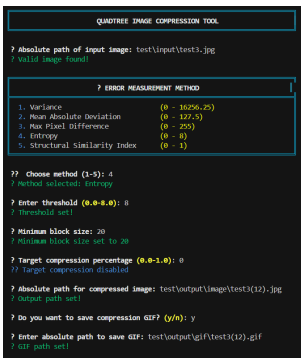
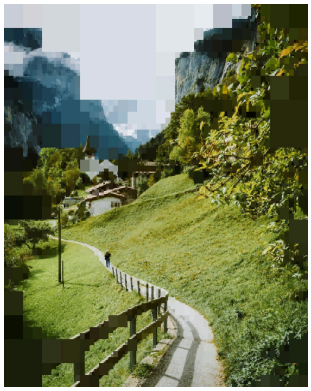
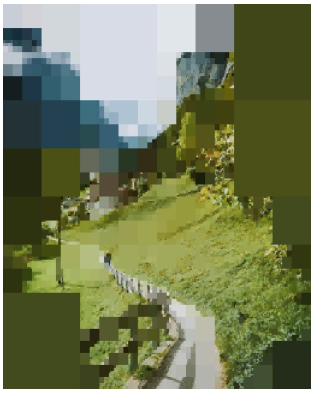
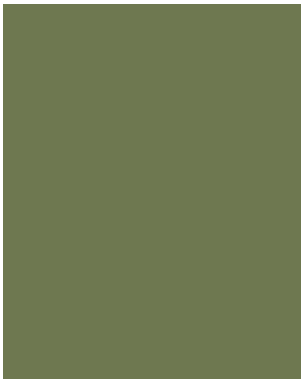
Hasil Kompresi :

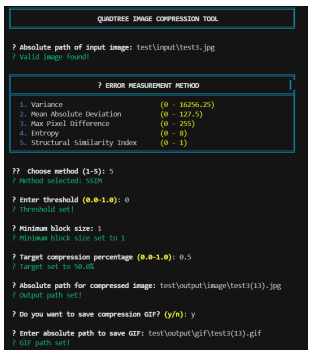
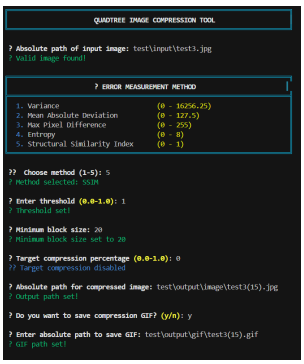
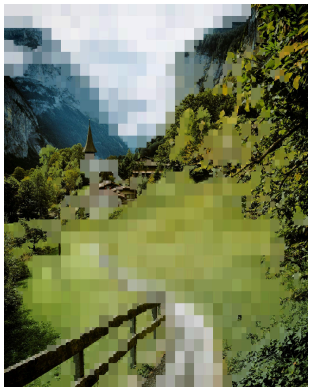
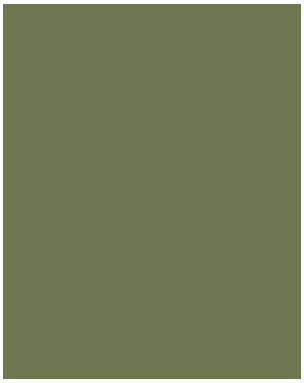
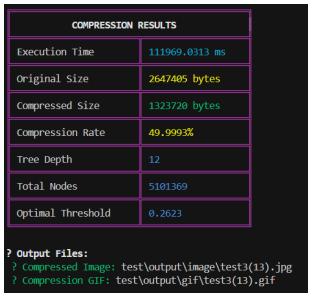
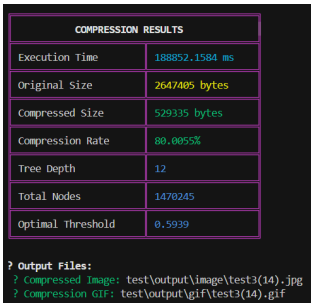
Metode	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 20

	Target Kompresi : 50%	Target Kompresi : 80%	Target Kompresi : 0%
Input :			
Variansi			
Nilai Ambang Batas :	0	0	1000
Output :			

Input :			
Mean Absolute Deviation			
Nilai Ambang Batas :	0	0	120
Output :			

Input :			
Max Pixel Difference			
Nilai Ambang Batas :	0	0	255
Output :			

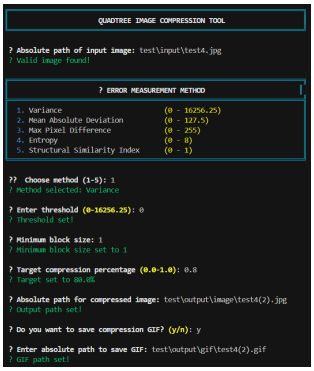
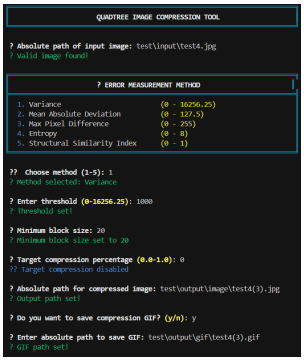
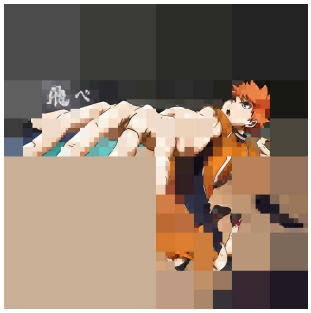
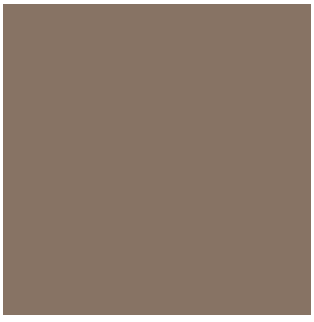
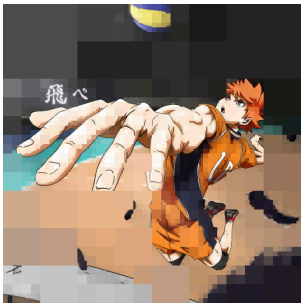
Input :			
Entropy			
Nilai Ambang Batas :	0	0	8
Output :	COMPRESSION RESULTS Execution Time 45414.2429 ms Original Size 2647405 bytes Compressed Size 1323706 bytes Compression Rate 49.9999% Tree Depth 9 Total Nodes 141417 Optimal Threshold 5.9818 ? Output Files: ✓ Compressed Image: test\output\image\test3(10).jpg ✓ Compression GIF: test\output\gif\test3(10).gif	COMPRESSION RESULTS Execution Time 62401.5356 ms Original Size 2647405 bytes Compressed Size 530034 bytes Compression Rate 79.9791% Tree Depth 9 Total Nodes 15577 Optimal Threshold 6.8491 ? Output Files: ✓ Compressed Image: test\output\image\test3(11).jpg ✓ Compression GIF: test\output\gif\test3(11).gif	COMPRESSION RESULTS Execution Time 5485.3369 ms Original Size 2647405 bytes Compressed Size 208707 bytes Compression Rate 92.1165% Tree Depth 0 Total Nodes 1 ? Output Files: ✓ Compressed Image: test\output\image\test3(12).jpg ✓ Compression GIF: test\output\gif\test3(12).gif

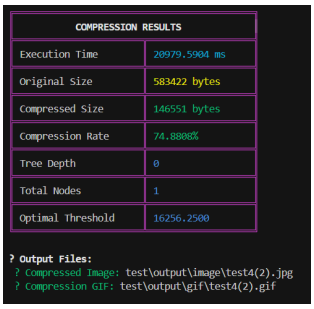
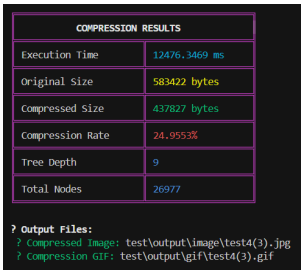
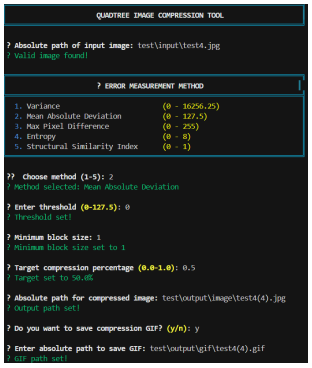
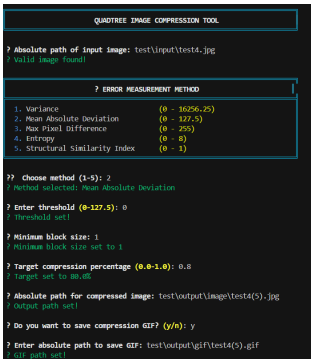
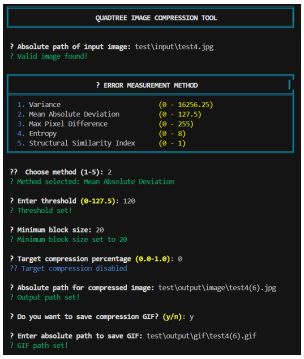
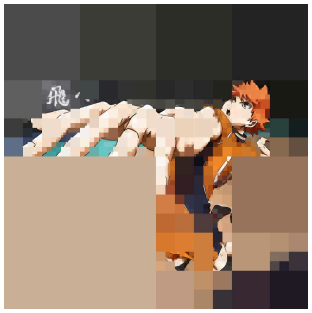
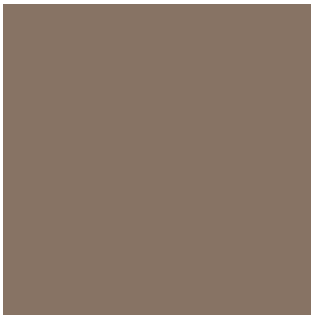
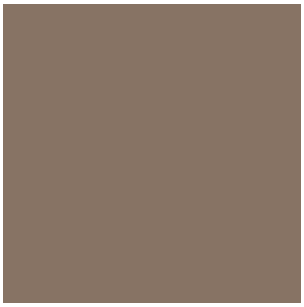
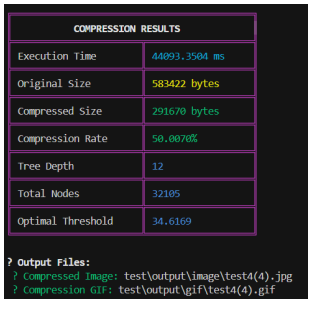
Input :			
Structural Similarity Index			
Nilai Ambang Batas :	0	0	1
Output :			

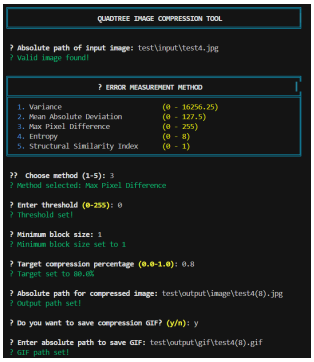
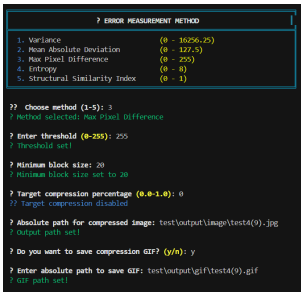
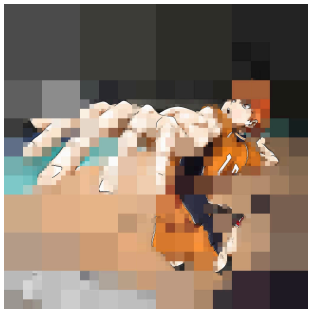
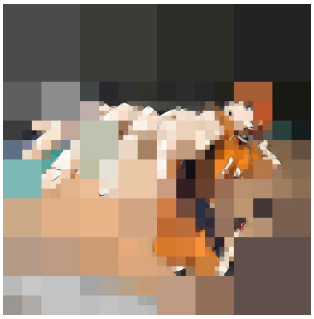
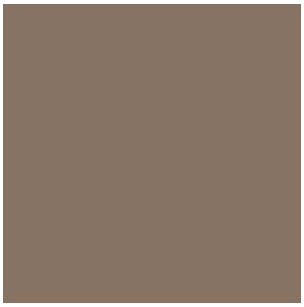
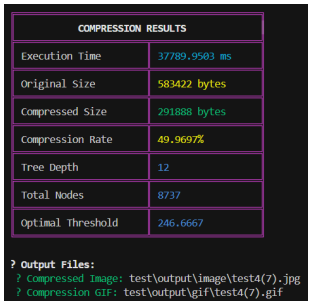
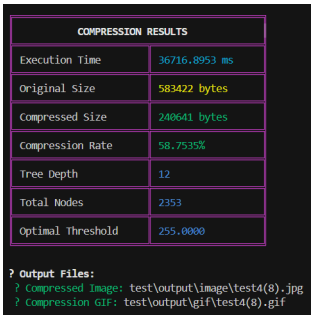
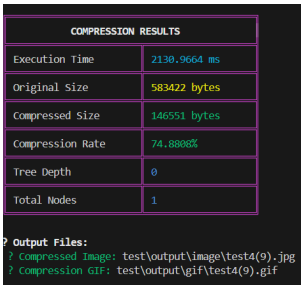
4.1.4 Kasus Eksperimen 4

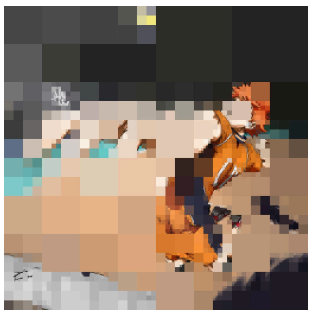
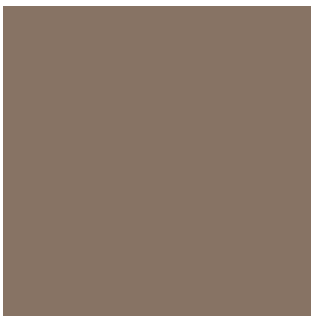
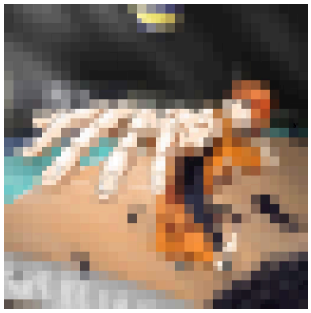
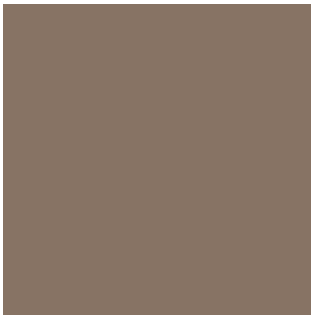
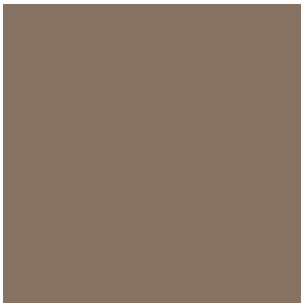
Gambar Asli :

Hasil Kompresi :

Metode	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 20
	Target Kompresi : 50%	Target Kompresi : 80%	Target Kompresi : 0%
Input :			
Variansi			
Nilai Ambang Batas :	0	0	1000

Output :			
Input :			
Mean Absolute Deviation			
Nilai Ambang Batas :	0	0	120
Output :			

Input :			
Max Pixel Difference			
Nilai Ambang Batas :	0	0	255
Output :			
Input :			

<table>
<tr><td>Entropy</td><td>  </td><td>  </td><td>  </td></tr>
<tr><td>Nilai Ambang Batas :</td><td>0</td><td>0</td><td>8</td></tr>
<tr><td>Output :</td><td>COMPRESSION RESULTS
Execution Time: 38050.8440 ms
Original Size: 583422 bytes
Compressed Size: 291839 bytes
Compression Rate: 49.9781%
Tree Depth: 9
Total Nodes: 4813
Optimal Threshold: 5.9375
? Output Files:
? Compressed Image: test\output\image\test4(10).jpg
? Compression GIF: test\output\gif\test4(10).gif</td><td>COMPRESSION RESULTS
Execution Time: 20118.7128 ms
Original Size: 583422 bytes
Compressed Size: 146551 bytes
Compression Rate: 74.8808%
Tree Depth: 0
Total Nodes: 1
Optimal Threshold: 8.0000
? Output Files:
? Compressed Image: test\output\image\test4(11).jpg
? Compression GIF: test\output\gif\test4(11).gif</td><td>COMPRESSION RESULTS
Execution Time: 2083.1717 ms
Original Size: 583422 bytes
Compressed Size: 146551 bytes
Compression Rate: 74.8808%
Tree Depth: 0
Total Nodes: 1
? Output Files:
? Compressed Image: test\output\image\test4(12).jpg
? Compression GIF: test\output\gif\test4(12).gif</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>Input :</td><td>QUADTREE IMAGE COMPRESSION TOOL
? Absolute path of input image: test\input\test4.jpg
? Valid image found!
? ERROR MEASUREMENT METHOD
1. Variance (0 - 16256.25)
2. Mean Absolute Deviation (0 - 127.5)
3. Max Pixel Difference (0 - 255)
4. Entropy (0 - 8)
5. Structural Similarity Index (0 - 1)
?? Choose method (1-5): 5
? Method selected: SSIM
? Enter threshold (0.0-1.0): 0
? Threshold set!
? Minimum block size: 1
? Minimum block size set to 1
? Target compression percentage (0.0-1.0): 0.5
? Target set to 50.0%
? Absolute path for compressed image: test\output\image\test4(13).jpg
? Output path set!
? Do you want to save compression GIF? (y/n): y
? Enter absolute path to save GIF: test\output\gif\test4(13).gif
? GIF path set!</td><td>QUADTREE IMAGE COMPRESSION TOOL
? Absolute path of input image: test\input\test4.jpg
? Valid image found!
? ERROR MEASUREMENT METHOD
1. Variance (0 - 16256.25)
2. Mean Absolute Deviation (0 - 127.5)
3. Max Pixel Difference (0 - 255)
4. Entropy (0 - 8)
5. Structural Similarity Index (0 - 1)
?? Choose method (1-5): 5
? Method selected: SSIM
? Enter threshold (0.0-1.0): 0
? Threshold set!
? Minimum block size: 1
? Minimum block size set to 1
? Target compression percentage (0.0-1.0): 0.8
? Target set to 80.0%
? Absolute path for compressed image: test\output\image\test4(14).jpg
? Output path set!
? Do you want to save compression GIF? (y/n): y
? Enter absolute path to save GIF: test\output\gif\test4(14).gif
? GIF path set!</td><td>QUADTREE IMAGE COMPRESSION TOOL
? Absolute path of input image: test\input\test4.jpg
? Valid image found!
? ERROR MEASUREMENT METHOD
1. Variance (0 - 16256.25)
2. Mean Absolute Deviation (0 - 127.5)
3. Max Pixel Difference (0 - 255)
4. Entropy (0 - 8)
5. Structural Similarity Index (0 - 1)
?? Choose method (1-5): 5
? Method selected: SSIM
? Enter threshold (0.0-1.0): 1
? Threshold set!
? Minimum block size: 20
? Minimum block size set to 20
? Target compression percentage (0.0-1.0): 0
?? Target compression disabled
? Absolute path for compressed image: test\output\image\test4(15).jpg
? Output path set!
? Do you want to save compression GIF? (y/n): y
? Enter absolute path to save GIF: test\output\gif\test4(15).gif
? GIF path set!</td></tr>
<tr><td>Structural Similarity Index</td><td>  </td><td>  </td><td>  </td></tr>
<tr><td>Nilai</td><td>0</td><td>0</td><td>1</td></tr>
</table>

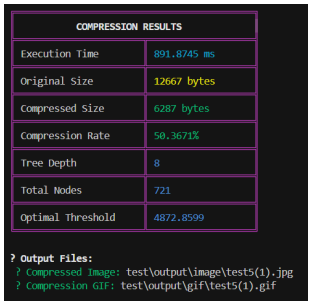
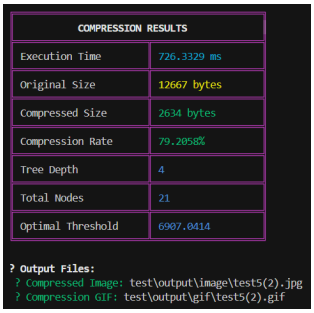
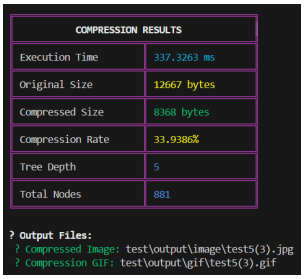
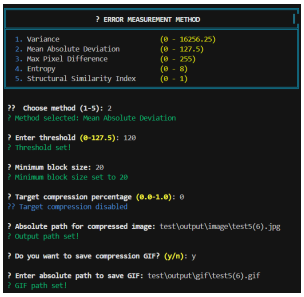
Ambang Batas :			
Output :	COMPRESSION RESULTS Execution Time: 109263.8604 ms Original Size: 583422 bytes Compressed Size: 291436 bytes Compression Rate: 50.0471% Tree Depth: 12 Total Nodes: 2929 Optimal Threshold: 0.6095 ? Output Files: ? Compressed Image: test\output\image\test4(13).jpg ? Compression GIF: test\output\gif\test4(13).gif	COMPRESSION RESULTS Execution Time: 37712.2924 ms Original Size: 583422 bytes Compressed Size: 146551 bytes Compression Rate: 74.8808% Tree Depth: 0 Total Nodes: 1 Optimal Threshold: 1.0000 ? Output Files: ? Compressed Image: test\output\image\test4(14).jpg ? Compression GIF: test\output\gif\test4(14).gif	COMPRESSION RESULTS Execution Time: 2346.9447 ms Original Size: 583422 bytes Compressed Size: 146551 bytes Compression Rate: 74.8808% Tree Depth: 0 Total Nodes: 1 ? Output Files: ? Compressed Image: test\output\image\test4(15).jpg ? Compression GIF: test\output\gif\test4(15).gif

4.1.5 Kasus Eksperimen 5

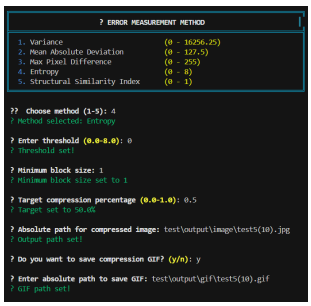
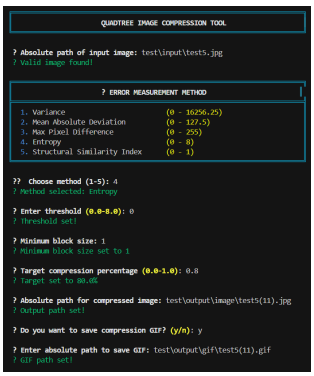
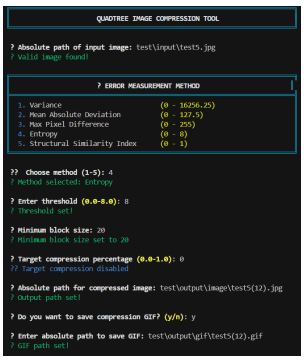
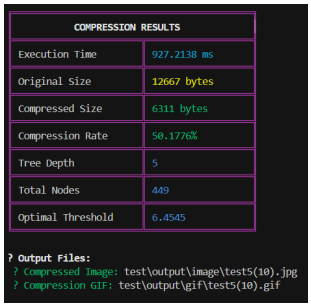
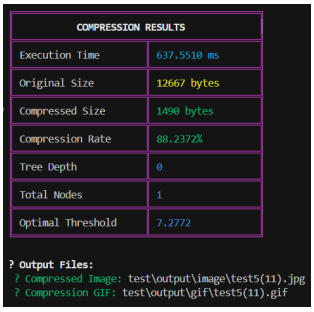
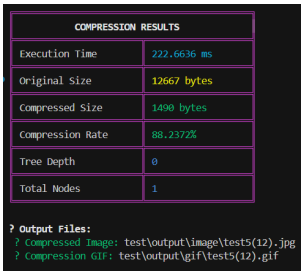
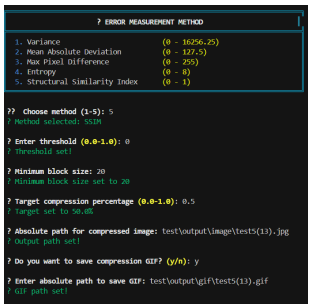
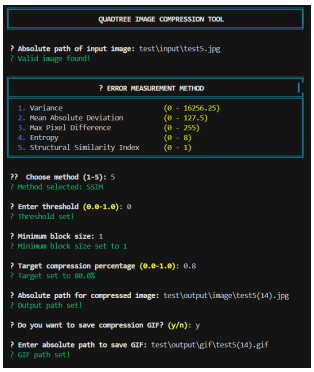
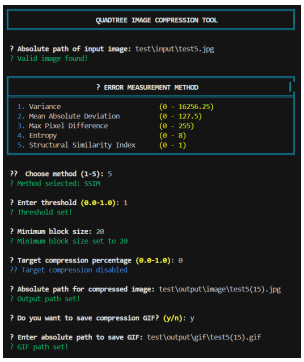
Gambar Asli :

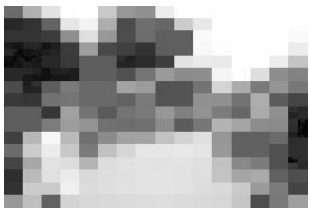
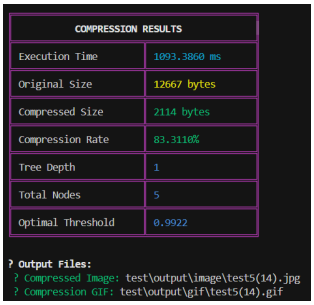
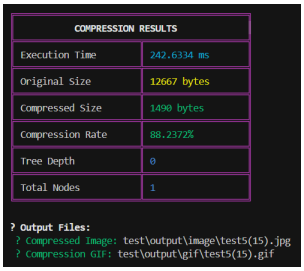
Hasil Kompresi :

Metode	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 20
	Target Kompresi : 50%	Target Kompresi : 80%	Target Kompresi : 0%
Input :	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test5.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 1 ? Method selected: Variance ? Enter threshold (0-16256.25): 0 ? Threshold set! ? Minimum block size: 1 ? Minimum block size set to 1 ? Target compression percentage (0.0-1.0): 0.5 ? Target set to 50.0% ? Absolute path for compressed image: test\output\image\test5(1).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test5(1).gif ? GIF path set!	? Absolute path of input image: test\input\test5.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 1 ? Method selected: Variance ? Enter threshold (0-16256.25): 0 ? Threshold set! ? Minimum block size: 1 ? Minimum block size set to 1 ? Target compression percentage (0.0-1.0): 0.8 ? Target set to 80.0% ? Absolute path for compressed image: test\output\image\test5(2).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test5(2).gif ? GIF path set!	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test5.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 1 ? Method selected: Variance ? Enter threshold (0-16256.25): 1000 ? Threshold set! ? Minimum block size: 20 ? Minimum block size set to 20 ? Target compression percentage (0.0-1.0): 0 ? Target compression disabled ? Absolute path for compressed image: test\output\image\test5(3).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test5(3).gif ? GIF path set!

Variansi			
Nilai Ambang Batas :	0	0	1000
Output :			
Input :			
Mean Absolute Deviation			
Nilai Ambang Batas :	0	0	120

Output :	COMPRESSION RESULTS Execution Time: 879.9418 ms Original Size: 12667 bytes Compressed Size: 6294 bytes Compression Rate: 50.3118% Tree Depth: 8 Total Nodes: 725 Optimal Threshold: 59.2477 ? Output Files: ? Compressed Image: test\output\image\test5(4).jpg ? Compression GIF: test\output\gif\test5(4).gif	COMPRESSION RESULTS Execution Time: 715.3608 ms Original Size: 12667 bytes Compressed Size: 2114 bytes Compression Rate: 83.3110% Tree Depth: 1 Total Nodes: 5 Optimal Threshold: 72.7513 ? Output Files: ? Compressed Image: test\output\image\test5(5).jpg ? Compression GIF: test\output\gif\test5(5).gif	COMPRESSION RESULTS Execution Time: 226.5652 ms Original Size: 12667 bytes Compressed Size: 1490 bytes Compression Rate: 88.2372% Tree Depth: 0 Total Nodes: 1 ? Output Files: ? Compressed Image: test\output\image\test5(6).jpg ? Compression GIF: test\output\gif\test5(6).gif
Input :	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test5.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 3 ? Method selected: Max Pixel Difference ? Enter threshold (0-255): 0 ? Threshold set! ? Minimum block size: 1 ? Minimum block size set to 1 ? Target compression percentage (0.0-1.0): 0.5 ? Target set to 50.0% ? Absolute path for compressed image: test\output\image\test5(7).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test5(7).gif ? GIF path set!	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test5.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 3 ? Method selected: Max Pixel Difference ? Enter threshold (0-255): 0 ? Threshold set! ? Minimum block size: 1 ? Minimum block size set to 1 ? Target compression percentage (0.0-1.0): 0.8 ? Target set to 80.0% ? Absolute path for compressed image: test\output\image\test5(8).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test5(8).gif ? GIF path set!	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test5.jpg ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 3 ? Method selected: Max Pixel Difference ? Enter threshold (0-255): 255 ? Threshold set! ? Minimum block size: 20 ? Minimum block size set to 20 ? Target compression percentage (0.0-1.0): 0 ?? Target compression disabled ? Absolute path for compressed image: test\output\image\test5(9).jpg ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test5(9).gif ? GIF path set!
Max Pixel Difference			
Nilai Ambang Batas :	0	0	255
Output :	COMPRESSION RESULTS Execution Time: 824.7805 ms Original Size: 12667 bytes Compressed Size: 6873 bytes Compression Rate: 45.7409% Tree Depth: 8 Total Nodes: 673 Optimal Threshold: 255.0000 ? Output Files: ? Compressed Image: test\output\image\test5(7).jpg ? Compression GIF: test\output\gif\test5(7).gif	COMPRESSION RESULTS Execution Time: 912.1869 ms Original Size: 12667 bytes Compressed Size: 6873 bytes Compression Rate: 45.7409% Tree Depth: 8 Total Nodes: 673 Optimal Threshold: 255.0000 ? Output Files: ? Compressed Image: test\output\image\test5(8).jpg ? Compression GIF: test\output\gif\test5(8).gif	COMPRESSION RESULTS Execution Time: 219.6406 ms Original Size: 12667 bytes Compressed Size: 1490 bytes Compression Rate: 88.2372% Tree Depth: 0 Total Nodes: 1 ? Output Files: ? Compressed Image: test\output\image\test5(9).jpg ? Compression GIF: test\output\gif\test5(9).gif

Input :			
Entropy			
Nilai Ambang Batas :	0	0	8
Output :			
Input :			

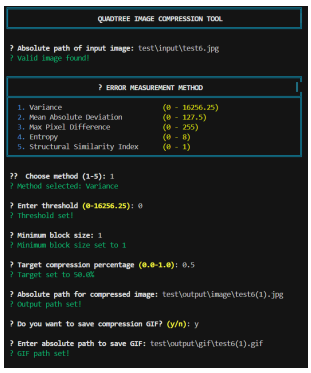
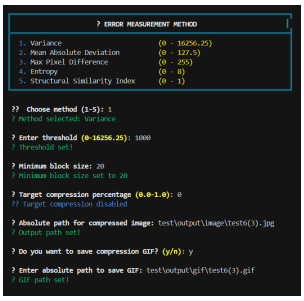
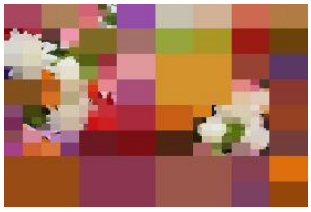
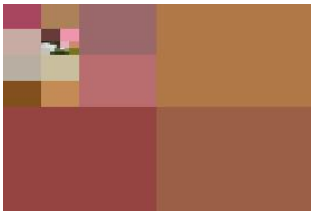
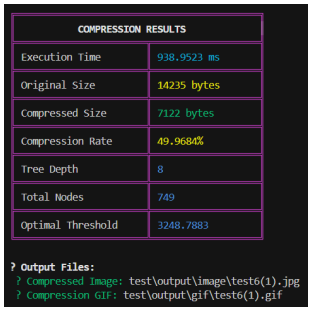
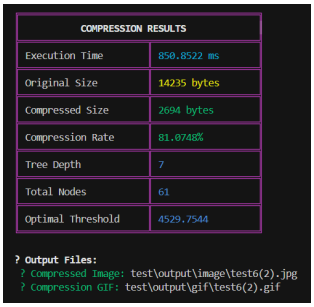
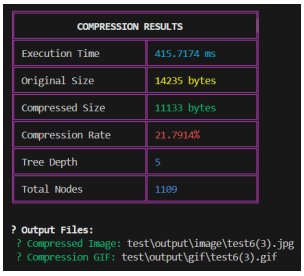
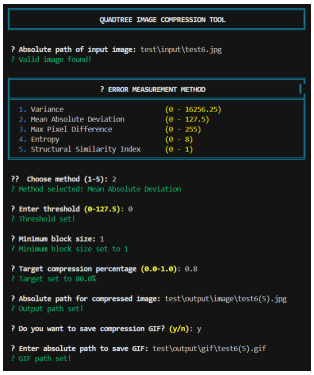
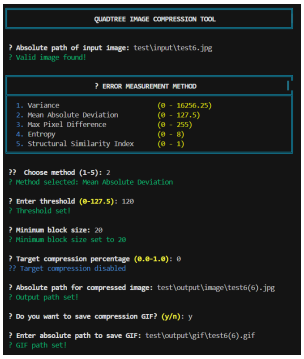
Structural Similarity Index			
Nilai Ambang Batas :	0	0	1
Output :			

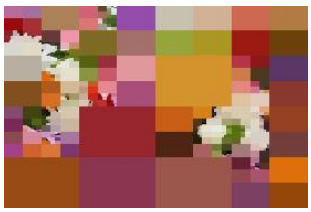
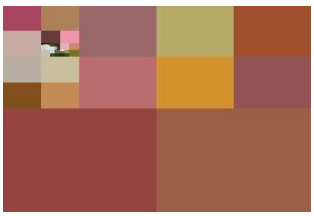
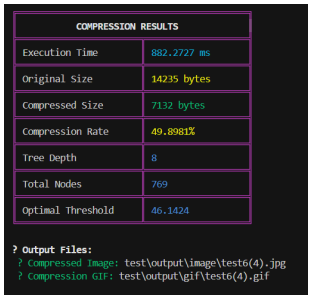
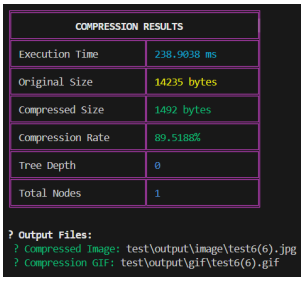
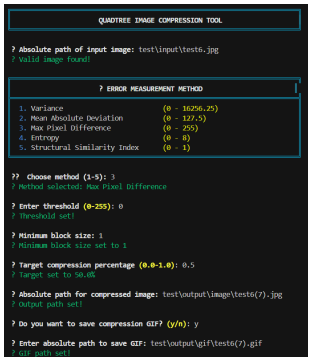
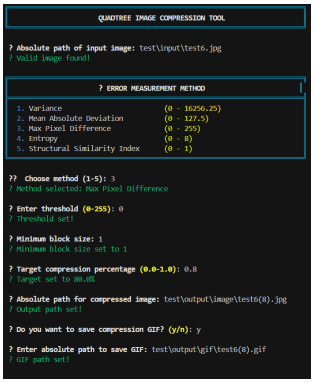
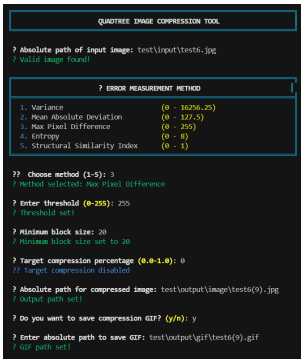
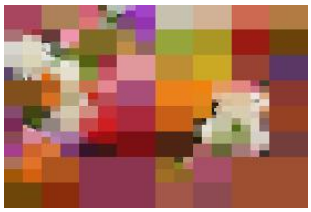
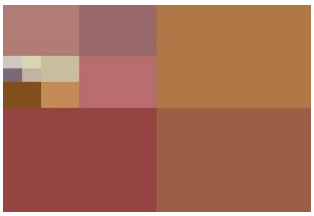
4.1.6 Kasus Eksperimen 6

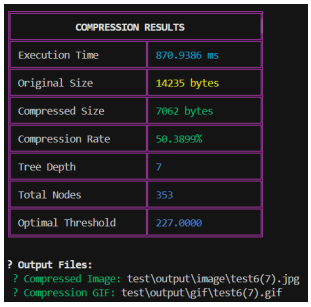
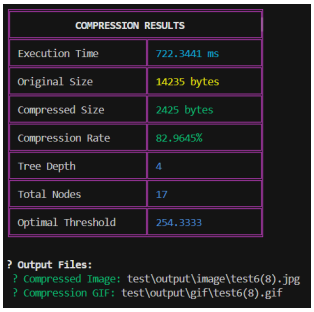
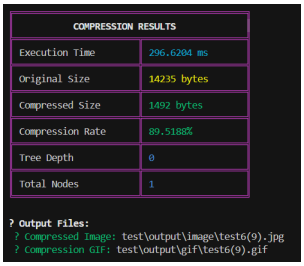
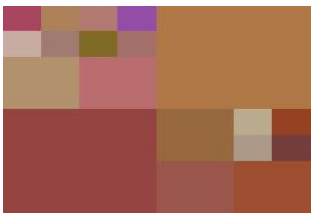
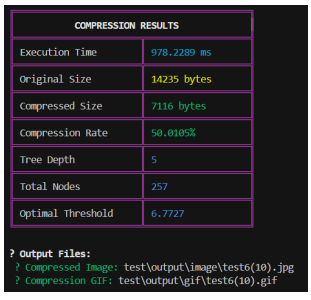
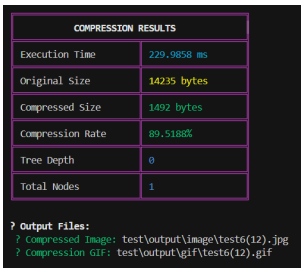
Gambar Asli :

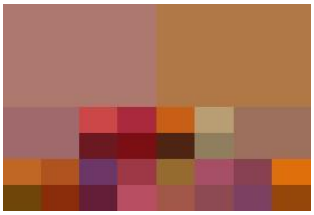
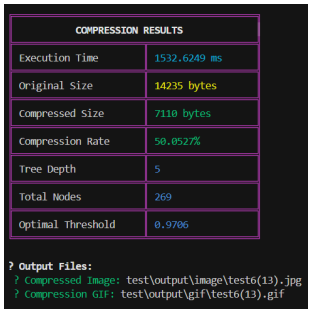
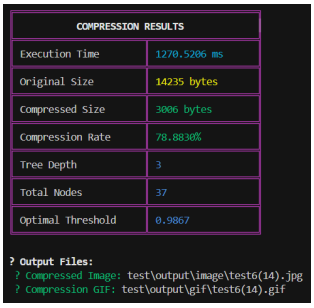
Hasil Kompresi :

Metode	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 20
	Target Kompresi : 50%	Target Kompresi : 80%	Target Kompresi : 0%

Input :			
Variansi			
Nilai Ambang Batas :	0	0	1000
Output :			
Input :			

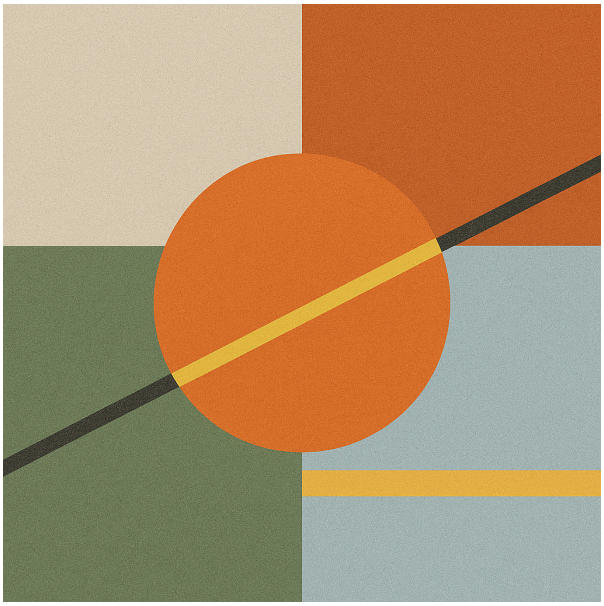
Mean Absolute Deviation			
Nilai Ambang Batas :	0	0	120
Output :			
Input :			
Max Pixel Difference			
Nilai Ambang Batas :	0	0	255

Output :			
Input :			
Entropy			
Nilai Ambang Batas :	0	0	8
Output :			

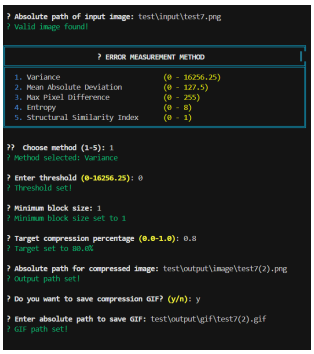
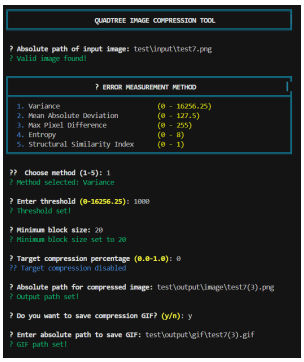
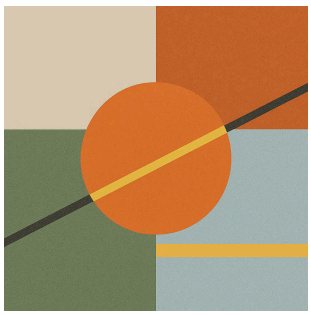
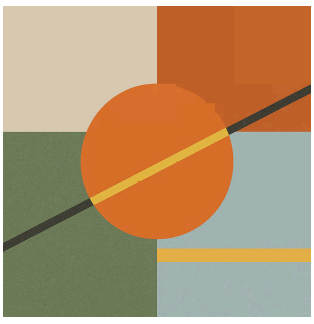
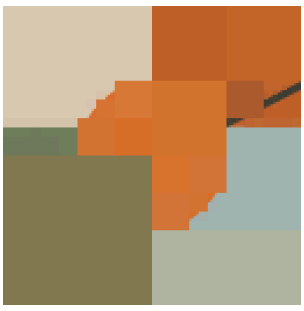
Input :			
Structural Similarity Index			
Nilai Ambang Batas :	0	0	1
Output :			

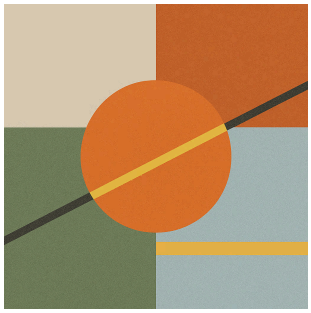
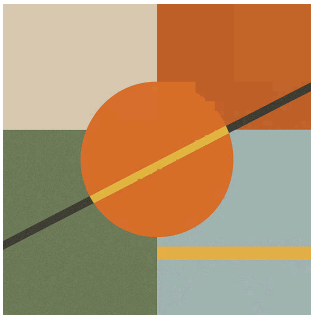
4.1.7 Kasus Eksperimen 7

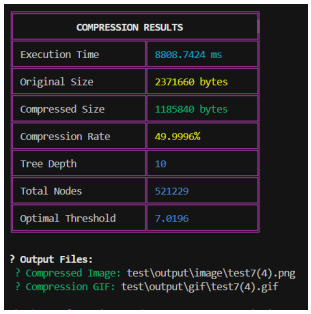
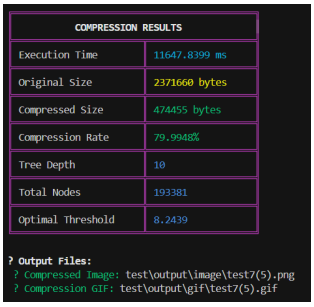
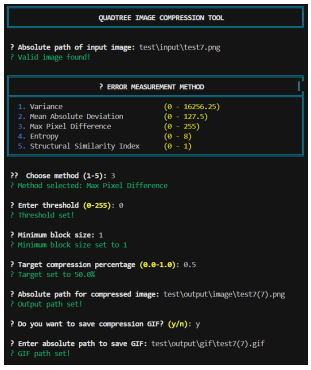
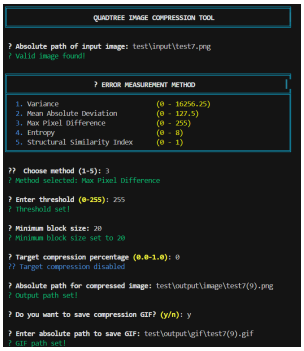
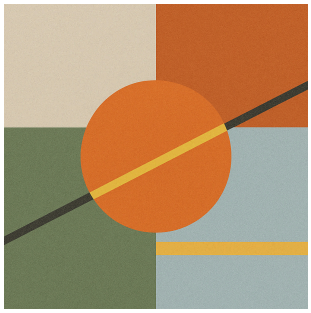
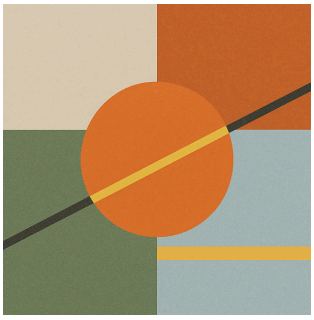
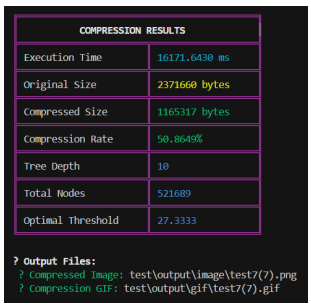
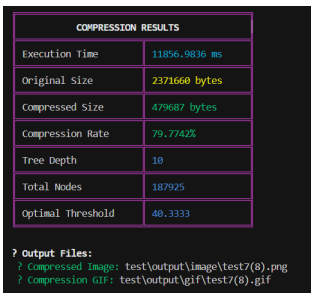
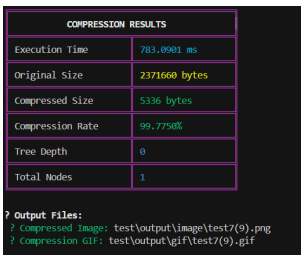
Gambar Asli :

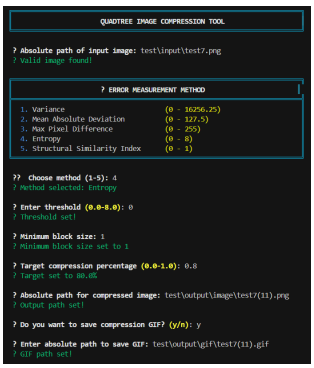
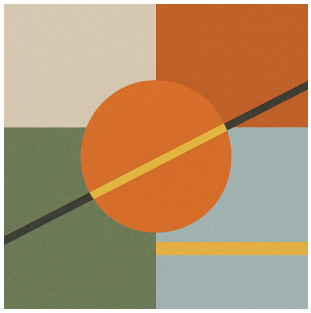
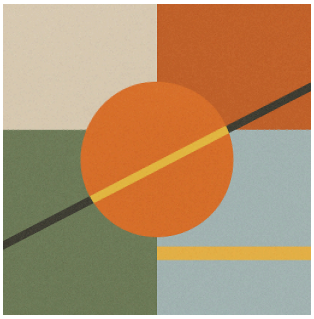
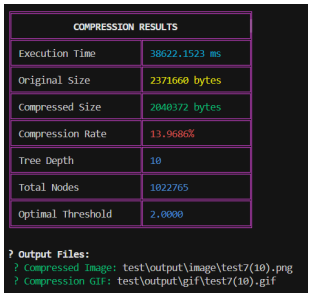
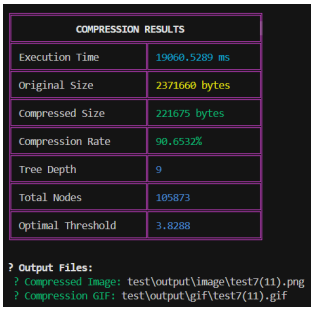
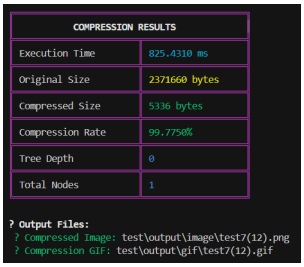
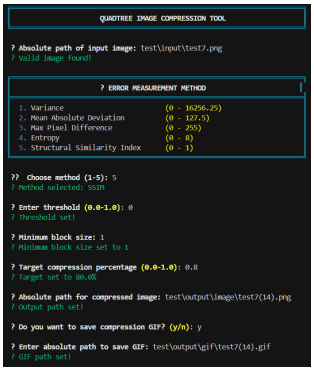
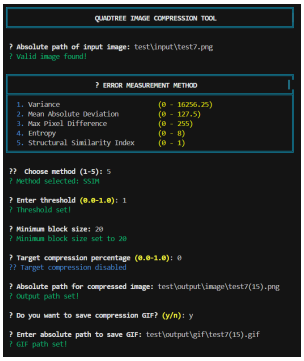


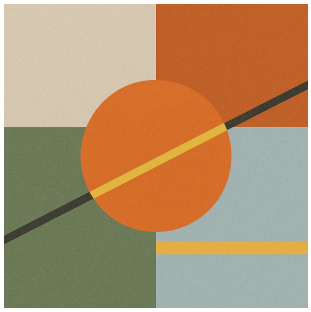
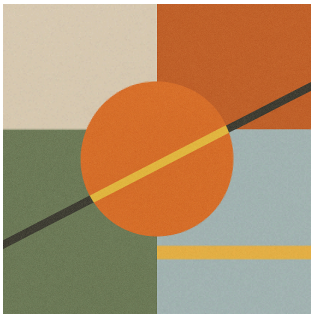
Hasil Kompresi :

Metode	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 0	Ukuran Blok Minimum : 20
	Target Kompresi : 50%	Target Kompresi : 80%	Target Kompresi : 0%
Input :			
Variansi			
Nilai	0	0	1000

Ambang Batas :			
Output :	COMPRESSION RESULTS Execution Time: 15412.4121 ms Original Size: 2371660 bytes Compressed Size: 1185975 bytes Compression Rate: 49.9939% Tree Depth: 10 Total Nodes: 519385 Optimal Threshold: 74.1250 ? Output Files: ? Compressed Image: test\output\image\test7(1).png ? Compression GIF: test\output\gif\test7(1).gif	COMPRESSION RESULTS Execution Time: 11603.6757 ms Original Size: 2371660 bytes Compressed Size: 473943 bytes Compression Rate: 80.0164% Tree Depth: 10 Total Nodes: 190305 Optimal Threshold: 104.7656 ? Output Files: ? Compressed Image: test\output\image\test7(2).png ? Compression GIF: test\output\gif\test7(2).gif	COMPRESSION RESULTS Execution Time: 2481.4069 ms Original Size: 2371660 bytes Compressed Size: 18628 bytes Compression Rate: 99.2146% Tree Depth: 7 Total Nodes: 285 ? Output Files: ? Compressed Image: test\output\image\test7(3).png ? Compression GIF: test\output\gif\test7(3).gif
Input :	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test7.png ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 2 ? Method selected: Mean Absolute Deviation ? Enter threshold (0-127.5): 0 ? Threshold set! ? Minimum block size: 1 ? Minimum block size set to 1 ? Target compression percentage (0.0-1.0): 0.5 ? Target set to 50.0% ? Absolute path for compressed image: test\output\image\test7(4).png ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test7(4).gif ? GIF path set!	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test7.png ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 2 ? Method selected: Mean Absolute Deviation ? Enter threshold (0-127.5): 0 ? Threshold set! ? Minimum block size: 1 ? Minimum block size set to 1 ? Target compression percentage (0.0-1.0): 0.8 ? Target set to 80.0% ? Absolute path for compressed image: test\output\image\test7(5).png ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test7(5).gif ? GIF path set!	QUADTREE IMAGE COMPRESSION TOOL ? Absolute path of input image: test\input\test7.png ? Valid image found! ? ERROR MEASUREMENT METHOD 1. Variance (0 - 16256.25) 2. Mean Absolute Deviation (0 - 127.5) 3. Max Pixel Difference (0 - 255) 4. Entropy (0 - 8) 5. Structural Similarity Index (0 - 1) ?? Choose method (1-5): 2 ? Method selected: Mean Absolute Deviation ? Enter threshold (0-127.5): 120 ? Threshold set! ? Minimum block size: 20 ? Minimum block size set to 20 ? Target compression percentage (0.0-1.0): 0 ?? Target compression disabled ? Absolute path for compressed image: test\output\image\test7(6).png ? Output path set! ? Do you want to save compression GIF? (y/n): y ? Enter absolute path to save GIF: test\output\gif\test7(6).gif ? GIF path set!
Mean Absolute Deviation			
Nilai Ambang Batas :	0	0	120

Output :			
Input :			
Max Pixel Difference			
Nilai Ambang Batas :	0	0	255
Output :			

Input :			
Entropy			
Nilai Ambang Batas :	0	0	8
Output :			
Input :			

Structural Similarity Index			
Nilai Ambang Batas :	0	0	1
Output :	COMPRESSION RESULTS Execution Time: 42084.4301 ms Original Size: 2371660 bytes Compressed Size: 1185342 bytes Compression Rate: 50.0206% Tree Depth: 10 Total Nodes: 622021 Optimal Threshold: 0.3915 ? Output Files: ? Compressed Image: test\output\image\test7(13).png ? Compression GIF: test\output\gif\test7(13).gif	COMPRESSION RESULTS Execution Time: 34186.1977 ms Original Size: 2371660 bytes Compressed Size: 474378 bytes Compression Rate: 79.9981% Tree Depth: 10 Total Nodes: 260541 Optimal Threshold: 0.5234 ? Output Files: ? Compressed Image: test\output\image\test7(14).png ? Compression GIF: test\output\gif\test7(14).gif	COMPRESSION RESULTS Execution Time: 695.0548 ms Original Size: 2371660 bytes Compressed Size: 5336 bytes Compression Rate: 99.7750% Tree Depth: 0 Total Nodes: 1 ? Output Files: ? Compressed Image: test\output\image\test7(15).png ? Compression GIF: test\output\gif\test7(15).gif

4.2 Analisis Pengujian dan Algoritma

Melalui berbagai eksperimen yang dilakukan. Seluruh jenis parameter dengan metode Quadtree efektif dalam melangsungkan sebuah kompresi terhadap gambar. Poin utama dari yang dilakukan dalam program ini adalah menghomogenisasi nilai warna dari blok-blok pecahan gambar yang lebih besar. Karena pemecahan sebuah blok gambar menjadi 4 bagian, dengan luas area gambar n yang memiliki lebar W dan tinggi H, nilai kompleksitas algoritmanya adalah $O(\log_4(n))$ (nilai asimptotik adalah $O(\log(n))$) . Hal ini didapatkan dari dalam pemecahan menjadi 4 blok kecil, setiap blok akan memiliki luas n/4, dalam pemecahan kedua terdapat 16 blok dengan luas n/16. Akan tetapi, waktu yang tetap dibutuhkan tetap sama yaitu n (jumlah piksel) dalam setiap tahapannya. Dengan mengasumsikan titik terjauh/terlama dari pemecahan adalah L, ukuran tujuan blok terkecil adalah 1 piksel (terburuk), maka pada level L, akan ada total piksel sejumlah 4^L. Berdasarkan konsep logaritma, makan nilai L adalah $\log_4(n)$. Dengan pengetahuan ini, kita dapat mencari kompleksitas algoritma total dari setiap metode. Kompleksitas waktu pemecahan ini merupakan nilai waktu yang dibutuhkan untuk mencapai kedalaman, sedangkan setiap metode merupakan kompleksitas algoritma yang dibutuhkan dalam pemecahan setiap levelnya.

Walaupun algoritma dasar yang digunakan sama yaitu pemecahan sebuah blok besar menjadi beberapa blok kecil, ada persyaratan yang berbeda-beda untuk setiap metode. Di dalam persyaratan yang ada, terdapat sebuah variabel yaitu nilai error atau nilai rata-rata warna yang dapat ditemukan dalam satu blok yang sama. Dengan metode variansi, nilai error akan diukur dengan seberapa jauh piksel-piksel dalam blok tersebar dari rata-ratanya. Sedangkan untuk metode *Mean Absolute Deviation* (MAD) mengukur rata-rata deviasi absolut dari tiap piksel dengan rata-ratanya. Metode *entropy* akan mengukur distribusi nilai piksel dalam suatu blok. Terakhir, metode *Structural Similarity Index* (SSIM) akan mengukur dengan kesamaan blok asli dan blok yang telah terkompresi.

Dalam metode variansi, formula pengukuran error yang digunakan adalah :

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

$$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$$

σ_c^2 = Variansi tiap kanal warna c (R, G, B) dalam satu blok

$P_{i,c}$ = Nilai piksel pada posisi *i* untuk kanal warna *c*

μ_c = Nilai rata-rata tiap piksel dalam satu blok

N = Banyaknya piksel dalam satu blok

Perhitungan yang dilakukan terdapat pada perhitungan mean, dan iterasi selisih kuadrat setiap piksel terhadap mean pada setiap piksel. Sehingga untuk perhitungan mean, dibutuhkan waktu O(W*H) dengan W adalah lebar gambar dan H adalah tinggi gambar. Perhitungan iterasi juga sebanyak O(W*H). Tidak hanya itu, karena setiap kanal warna dihitung, maka totalnya adalah O(3*2*W*H). Akan tetapi, dalam penulisan asimptotik, nilai kompleksitas waktunya adalah O(W*H) atau sama dengan O(n) (n adalah jumlah piksel total). Waktu inilah yang dibutuhkan dalam menghitung nilai variansi dalam setiap level. Sehingga dengan L kedalaman, dan O(log(n)) kedalaman, maka total kompleksitas waktunya adalah O(n) * O(log n) = O(n log n).

Sedangkan dalam metode *Mean Absolute Deviation* (MAD), perhitungan error dilakukan dengan rumus

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

$$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$$

MAD_c = Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok

$P_{i,c}$ = Nilai piksel pada posisi *i* untuk kanal warna *c*

μ_c = Nilai rata-rata tiap piksel dalam satu blok

N = Banyaknya piksel dalam satu blok

Proses waktu yang dibutuhkan oleh metode ini mirip dengan perhitungan menggunakan variansi. Dalam metode ini, dibutuhkan perhitungan rata-rata dengan waktu O(W*H). Kemudian iterasi jumlah deviasi absolut dengan waktu yang sama yaitu O(W*H). Sehingga totalnya adalah O(3*2*W*H). Dalam penulisan asimptotik tetap O(W*H) dengan total kompleksitas waktu adalah O(n) * O(log n) = O(n log n).

Metode *Max Pixel Difference* menggunakan rumus

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

$$D_{RGB} = \frac{D_R + D_G + D_B}{3}$$

D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok

$P_{i,c}$ = Nilai piksel pada posisi *i* untuk channel warna *c*

Sehingga pada eksekusinya, metode ini hanya membutuhkan iterasi sekali saja terhadap semua piksel untuk mencari nilai maksimum dan minimum sehingga kompleksitas waktu totalnya adalah O(W*H). Sehingga total kompleksitas waktunya adalah O(n) * O(log n) = O(n log n).

Pada metode entropi, rumus yang digunakan adalah

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

$$H_{RGB} = \frac{H_R + H_G + H_B}{3}$$

H_c = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok

$P_c(i)$ = Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)

Di dalam metode ini, dibutuhkan pembuatan sebuah histogram yang isinya merupakan luas gambar sehingga waktu yang dibutuhkan adalah O(W*H). Proses selanjutnya adalah perhitungan entropi yang membutuhkan waktu O(K) dengan K adalah 256 yaitu jumlah level piksel. Sehingga total waktu yang dibutuhkan adalah O(W*H + K). Akan tetapi, dengan penulisan asimptotik, total kompleksitas waktunya adalah O(n) * O (log n) = O (n log n).

Metode terakhir yaitu *Structural Similarity Index* (SSIM) memiliki rumus

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

$$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$$

Metode ini sebenarnya cukup kompleks karena harus menghitung rata-rata, variansi, dan kovariansi untuk dua buah gambar (biasa dan sudah terkompresi). Meskipun demikian, apabila ditulis dengan penulisan asimptotik, nilai kompleksitas waktunya masih pada O(n). Sehingga total kompleksitas waktunya adalah O(n) * O (log n) = O (n log n).

Sebagai sebuah kesimpulan, nilai total kompleksitas waktu dari setiap metode yang digunakan sama yaitu O(n log n). Hal ini bermakna bahwa dengan jumlah piksel total sebanyak n, maka terdapat log n operasi tambahan bagi setiap pikselnya. Nilai ini merupakan hasil dari algoritma *Divide and Conquer* yang diaplikasikan dalam program ini. Nilai log n sendiri berasal dari jumlah pembagian (kedalaman pohon) yang ditelusuri dan n adalah jumlah pemrosesan pada setiap level kedalamannya.

Walaupun dalam penulisan asimptotiknya sama, setiap metode memiliki jumlah operasi yang berbeda-beda. Metode yang paling berat adalah *Structural Similarity Index* karena harus menghitung berbagai variabel pada 2 gambar. Disusul oleh metode entropi yang disebabkan oleh perhitungan entropinya. Kemudian metode variansi dan *Mean Absolute Distribution* yang menghitung O(n) sebanyak 6 kali. Metode yang paling efisien adalah *Max Pixel Difference* karena hanya harus melakukan iterasi sebanyak sekali.

4.3 Analisis Fitur Bonus

Di dalam program ini, terdapat 3 fitur bonus yang telah diimplementasikan. Fitur tersebut mencakup target persentase kompresi, metode *Structural Similarity Index* (SSIM), dan pembuatan GIF proses kompresi. Ketiga fitur ini telah tersedia dalam program dan dapat diakses melalui *interface*.

Fitur target kompresi memungkinkan pengguna untuk mengatur nilai pengurangan ukuran hasil kompresi melalui nilai *floating point*. Tanpa fitur ini, pengguna hanya bisa mengatur nilai ambang batas error yang diterima dan ukuran blok minimum yang diinginkan. Dengan adanya fitur ini, algoritma akan menyesuaikan nilai ambang batas secara otomatis dan dinamis agar dapat memenuhi target yang diberikan oleh pengguna. Dalam aplikasinya, proses yang dilakukan adalah melakukan kompresi secara bertahap. Setelah melakukan kompresi, maka gambar hasil akan disimpan sementara dan nilai persentase kompresi akan dihitung. Proses ini akan dilakukan berulang kali hingga persentase kompresi mencapai target yang diinginkan. Implementasi fitur ini mencakup penggunaan fungsi compress dan findOptimalThreshold pada class QuadtreeCompression dan fungsi calcCompressionPercentage serta writeCompressesImage pada class IO.

Metode *Structural Similarity Index* (SSIM) adalah salah satu metode perhitungan error yang dapat digunakan. Pada dasarnya, metode ini akan menghitung perbedaan nilai warna antara blok gambar sebelum dan sesudah dikompresi. Di dalam setiap blok, akan dihitung nilai SSIM untuk setiap kanal warna yang pada akhirnya akan digabungkan. Perhitungan ini mencakup perhitungan rata-rata, variansi, dan kovariansi intensitas warna pada setiap gambar sebelum akan dibandingkan. Implementasinya dapat dilihat pada fungsi calculateSSIM pada class ErrorMetrics. Di dalam class ErrorMetrics juga dapat ditemukan fungsi-fungsi pembantu seperti calculateSSIM, calculateChannelSSIM, calculateCovariance, dan calculateBlockSSIM.

Fitur terakhir adalah pembuatan GIF proses kompresi. Melalui GIF ini, akan didapatkan visualisasi mengenai proses pembentukan Quadtree dalam kompresi gambar ini. Dalam berkas GIF, dapat dilihat proses secara bertahap dari kompresi dimulai dari gambar orisinil hingga ke hasil akhir berdasarkan kedalaman dari pohon Quadtree yang telah dibuat. Hal ini diimplementasikan pada method createCompressionGif pada class IO. Dalam pembuatannya, terdapat beberapa fungsi pembantu untuk rekonstruksi gambar yang dapat dilihat pada fungsi reconstructGIFFromQuadtree dan reconstructNodeforGIF.

LAMPIRAN

Tabel Kelengkapan Spesifikasi

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

DAFTAR PUSTAKA

Munir, Rinaldi. 2025. “Algoritma Divide and Conquer (Bagian 1)” (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf, diakses 8 April 2025)

Munir, Rinaldi. 2025. “Algoritma Divide and Conquer (Bagian 2)” (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/05-Algoritma-Greedy-(2025)-Bag2.pdf, diakses 8 April 2025)

Munir, Rinaldi. 2025. “Algoritma Divide and Conquer (Bagian 3)” (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-(2025)-Bagian3.pdf, diakses 8 April 2025)

Munir, Rinaldi. 2025. “Algoritma Divide and Conquer (Bagian 4)” (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/10-Algoritma-Divide-and-Conquer-(2025)-Bagian4.pdf, diakses 8 April 2025)