

# **Kompresi Gambar dengan Metode Quadtree**

Laporan Tugas Kecil 2  
IF2211 - Strategi Algoritma



**Disusun Oleh:**

12821046 - Fardhan Indrayesa

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

**2025**

## **DAFTAR ISI**

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>1. Algoritma Divide and Conquer.....</b>	<b>3</b>
a. Penjelasan Algoritma.....	3
b. Pseudocode.....	3
<b>2. Implementasi.....</b>	<b>6</b>
a. Source Code.....	6
b. Implementasi.....	17
<b>3. Hasil dan Analisis.....</b>	<b>18</b>
a. Hasil.....	18
b. Analisis.....	23
<b>4. Kesimpulan.....</b>	<b>25</b>
<b>REFERENSI.....</b>	<b>26</b>
<b>LAMPIRAN.....</b>	<b>27</b>

## 1. Algoritma Divide and Conquer

### a. Penjelasan Algoritma

Algoritma yang digunakan dalam mengompres gambar adalah algoritma divide and conquer. Penggunaan algoritma ini memanfaatkan salah satu metode kompresi gambar, yaitu quadtree. Berikut merupakan penjelasan dari algoritma divide and conquer dalam permasalahan quadtree yang sudah diimplementasikan.

1. Fungsi divide and conquer (dnc) menerima input berupa matriks gambar RGB, metode eror (int), ambang batas eror, ukuran blok minimum, koordinat titik awal, lebar dan panjang blok, serta matriks gambar RGB (sebagai gambar pembanding untuk SSIM).
2. Program akan mengecek metode eror apa yang akan digunakan. Metode eror SSIM dibedakan dengan yang lain karena membutuhkan dua matriks, sebagai pembanding.
3. Lalu, gambar di-*slice* berdasarkan titik koordinat dan luas (panjang dan lebar) gambar dari data argumen yang sudah diberikan.
4. Hitung rata-rata setiap *channel* dalam satu blok gambar yang sudah di-*slice* tersebut.
5. Hitung erornya berdasarkan metode yang sudah diberikan.
6. Cek kondisi eror dan ukuran blok. Apabila eror lebih kecil dari *threshold* (atau *similarity* lebih besar dari *threshold*, untuk SSIM) atau ukuran blok sudah lebih kecil dari ukuran minimum blok, maka isi blok tersebut dengan rata-rata di setiap *channel*-nya (yang sudah dihitung pada langkah 4).
7. Apabila tidak memenuhi kondisi tersebut (eror masih lebih besar/*similarity* masih lebih kecil atau ukuran blok masih lebih besar dari ukuran blok minimum), maka bagi blok tersebut menjadi empat bagian.
8. Selesaikan masing-masing bagian blok tersebut sampai memenuhi salah satu kondisi pada langkah 6.

### b. Pseudocode

Berikut merupakan psudocode dari program ImageCompressor

```
START QuadtreeImageCompression

DECLARE nodes AS INTEGER ← 1
DECLARE ext AS STRING

FUNCTION Main(imPath, errorMethod, threshold, minBlock,
outPath)
    IF input is invalid THEN
        DISPLAY error message
        RETURN
    ENDIF
```

```

LOAD image from imPath INTO image
CONVERT image to RGB matrix (matrixIm) and alpha
matrix (a)
DISPLAY image size

IF errorMethod == 5 THEN
    Q ← Copy of matrixIm
    depth ← DNC(matrixIm, errorMethod, threshold,
minBlock, 0, 0, height, width, Q)
ELSE
    depth ← DNC(matrixIm, errorMethod, threshold,
minBlock, 0, 0, height, width)
ENDIF

SAVE compressed image to outPath
DISPLAY execution time, original size, compressed
size, compression %, tree depth, total nodes

FUNCTION DNC(P, errorMethod, threshold, minBlock, i, j,
height, width, Q = null) RETURNS INTEGER
    N ← height × width

    IF errorMethod == 5 THEN
        PSliced ← FillArray(P, i, j, height, width)
        QSliced ← FillArray(Q, i, j, height, width)
        mean ← ComputeMean(PSliced)
        REPLACE all pixels in PSliced with mean

        similarity ← ErrorMeasurement(5, PSliced,
QSliced)
        IF similarity > threshold OR N < minBlock THEN
            FILL block in P with mean
            RETURN 1
        ENDIF
    ELSE
        PSliced ← FillArray(P, i, j, height, width)
        mean ← ComputeMean(PSliced)
        error ← ErrorMeasurement(errorMethod, PSliced)

        IF error < threshold OR N < minBlock THEN
            FILL block in P with mean
            RETURN 1
        ENDIF
    ENDIF

// Divide the block into 4 quadrants
DECLARE h1 ← height / 2, h2 ← height - h1
DECLARE w1 ← width / 2, w2 ← width - w1

d1 ← DNC(P, errorMethod, threshold, minBlock, i, j,

```

```

h1, w1, Q)
    nodes ← nodes + 1
    d2 ← DNC(P, errorMethod, threshold, minBlock, i +
w1, j, h1, w2, Q)
    nodes ← nodes + 1
    d3 ← DNC(P, errorMethod, threshold, minBlock, i, j +
h1, h2, w1, Q)
    nodes ← nodes + 1
    d4 ← DNC(P, errorMethod, threshold, minBlock, i +
w1, j + h1, h2, w2, Q)
    nodes ← nodes + 1

    RETURN 1 + MAX(d1, d2, d3, d4)

FUNCTION FillArray(P, istart, jstart, height, width)
RETURNS block
    COPY sub-block from P starting at (istart, jstart)
with given height and width
    RETURN copied block

FUNCTION ComputeMean(block) RETURNS (Rmean, Gmean,
Bmean)
    FOR each channel R, G, B DO
        CALCULATE mean value
    RETURN mean for each channel

FUNCTION ErrorMeasurement(method, P, Q = null) RETURNS
number
    SWITCH method
        CASE 1: return Variance(P)
        CASE 2: return Mean Absolute Deviation (MAD) (P)
        CASE 3: return Max Pixel Difference (MPD) (P)
        CASE 4: return Entropy(P)
        CASE 5: return SSIM(P, Q)
    END SWITCH

FUNCTION ToImage(P, outputPath, alpha) RETURNS
fileSizeMB
    CONVERT P to image file, apply alpha if PNG
    RETURN file size in MB

FUNCTION CompressPercent(firstSize, secondSize) RETURNS
percentage
    RETURN (1 - secondSize / firstSize) × 100

END QuadtreeImageCompression

```

## 2. Implementasi

### a. Source Code

```
using System;
using System.IO;
using System.Diagnostics;
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.Formats;
using SixLabors.ImageSharp.PixelFormats;
using SixLabors.ImageSharp.Formats.Png;
using SixLabors.ImageSharp.Formats.Jpeg;

class Program
{
    static int nodes = 1; // root, untuk total simpul pohon
    static string ext = "";

    static void Main(string[] args) // {imPath, errorMethod,
    threshold, minblock, outPath}
    {
        if (args.Length < 5)
        {
            Console.WriteLine("Argumen tidak lengkap. Gunakan format
<imPath> <errorMethod> <threshold> <minBlock> <outPath>");
            return;
        }

        if (!File.Exists(args[0]))
        {
            Console.WriteLine("File gambar tidak ditemukan!");
            return;
        }

        string imPath = args[0];
        int errorMethod = int.Parse(args[1]);
        double threshold = double.Parse(args[2]);
        int minBlock = int.Parse(args[3]);
        string outPath = args[4];
        ext = Path.GetExtension(imPath).ToLower();
        int deep;

        if (minBlock < 0)
        {
            Console.WriteLine("Ukuran minimum blok tidak valid!
Masukkan angka > 4");
            return;
        }

        if ((errorMethod < 1) || (errorMethod > 5))
        {
            Console.WriteLine("Metode eror tidak valid! Masukkan
angka 1 sampai 5");
            return;
        }

        if (threshold < 0)
        {
            Console.WriteLine("Threshold tidak valid. Masukkan
```

```

threshold > 0");
    return;
}
else if ((errorMethod == 5) && (threshold > 1))
{
    Console.WriteLine("Threshold tidak valid. Untuk SSIM,
masukkan 0 < threshold < 1");
    return;
}

using (Image<Rgba32> image = Image.Load<Rgba32>(imPath))
{
    int w = image.Width;
    int h = image.Height;
    int[,] matrixIm = new int[h, w, 3]; // matriks RGB
    int[,] a = new int[h, w]; // simpan nilai alpha untuk
png, apabila terdapat piksel transparan

    for (int y = 0; y < h; y++)
    {
        for (int x = 0; x < w; x++)
        {
            Rgba32 px = image[x, y];
            matrixIm[y, x, 0] = px.R;
            matrixIm[y, x, 1] = px.G;
            matrixIm[y, x, 2] = px.B;
            a[y, x] = px.A;
        }
    }
    Console.WriteLine($"Gambar dimuat dengan ukuran: {w}*{h}
px");

    var fileInfo = new FileInfo(imPath);
    double sizeMB = fileInfo.Length / (1024.0*1024.0); // 
Ukuran awal gambar

    int height = matrixIm.GetLength(0);
    int width = matrixIm.GetLength(1);
    int channel = matrixIm.GetLength(2);

    Stopwatch duration = new Stopwatch();
    Console.WriteLine("Mengompres gambar...");
    if (errorMethod == 5)
    {
        int[,] Q = FillArray(matrixIm, 0, 0, height,
width); // untuk SSIM, Q sebagai matriks RGB yang tidak diubah

        duration.Start();
        deep = dnc(matrixIm, errorMethod, threshold,
minBlock, 0, 0, height, width, Q);
        duration.Stop();
    }
    else
    {
        duration.Start();
        deep = dnc(matrixIm, errorMethod, threshold,
minBlock, 0, 0, height, width);
        duration.Stop();
    }
}

```

```

        long ms = duration.ElapsedMilliseconds;
        double seconds = ms / 1000; // Konversi waktu
eksekusi ke detik

        double lastSize = ToImage(matrixIm, outPath, a); // Konversi gambar dari matriks RGB ke file gambar
        Console.WriteLine($"Waktu eksekusi: {seconds:F2} detik");
        Console.WriteLine($"Ukuran gambar sebelum di-compress: {sizeMB:F2} MB");
        Console.WriteLine($"Ukuran gambar setelah di-compress: {lastSize:F2} MB");
        Console.WriteLine($"Persentase kompresi: {CompressPercent(sizeMB, lastSize):F2}%");
        Console.WriteLine("Kedalaman pohon: " + deep);
        Console.WriteLine("Banyak simpul: " + nodes);
        Console.WriteLine($"Gambar tersimpan di {outPath}");
    }
}

// Algoritma divide and conquer Quadtree
public static int dnc(int[,] P, int errorMethod, double thresh,
int minblock, int i, int j, int height, int width, int[,]? Q =
null)
{
    int depth; // kedalaman pohon
    int x = i, y = j;
    int N = height * width;
    double error, sim;

    if (errorMethod == 5) // SSIM
    {
        Q ??= P;
        int[,] PSliced = FillArray(P, x, y, height, width);
        int[,] QSliced = FillArray(Q, x, y, height, width);

        // Hitung rata-rata tiap channel
        double rpSum = 0, rpMean = 0;
        double gpSum = 0, gpMean = 0;
        double bpSum = 0, bpMean = 0;
        for (int ii = 0; ii < width; ii++)
        {
            for (int jj = 0; jj < height; jj++)
            {
                rpSum += PSliced[jj, ii, 0];
                gpSum += PSliced[jj, ii, 1];
                bpSum += PSliced[jj, ii, 2];
            }
        }
        rpMean = rpSum / N;
        gpMean = gpSum / N;
        bpMean = bpSum / N;

        for (int ii = 0; ii < PSliced.GetLength(1); ii++)
        {
            for (int jj = 0; jj < PSliced.GetLength(0); jj++)
            {
                PSliced[jj, ii, 0] = (int)rpMean;
                PSliced[jj, ii, 1] = (int)gpMean;
                PSliced[jj, ii, 2] = (int)bpMean;
            }
        }
    }
}

```

```

        }
    }

    sim = ErrorMeasurement(errorMethod, PSliced, QSliced);

    if (sim > thresh) // berhenti apabila similarity
melebihi threshold
    {
        // Isi dengan rata-rata blok
        for (int ii = x; ii < width + x; ii++)
        {
            for (int jj = y; jj < height + y; jj++)
            {
                P[jj, ii, 0] = (int)rpMean;
                P[jj, ii, 1] = (int)gpMean;
                P[jj, ii, 2] = (int)bpMean;
            }
        }
        return 1;
    }
    else if (N < minblock) // berhenti apabila jumlah
piksel melebihi minblock
    {
        // Isi dengan rata-rata blok
        for (int ii = x; ii < width + x; ii++)
        {
            for (int jj = y; jj < height + y; jj++)
            {
                P[jj, ii, 0] = (int)rpMean;
                P[jj, ii, 1] = (int)gpMean;
                P[jj, ii, 2] = (int)bpMean;
            }
        }
        return 1;
    }
else
{
    // bagi blok menjadi empat bagian
    int heightDiv1 = height / 2;
    int heightDiv2 = height - heightDiv1;

    int widthDiv1 = width / 2;
    int widthDiv2 = width - widthDiv1;

    // Selesaikan masing-masing blok
    int d1 = dnc(P, errorMethod, thresh, minblock, x,
y, heightDiv1, widthDiv1, Q);
    nodes += 1;
    int d2 = dnc(P, errorMethod, thresh, minblock,
x+widthDiv1, y, heightDiv1, widthDiv2, Q);
    nodes += 1;
    int d3 = dnc(P, errorMethod, thresh, minblock, x,
y+heightDiv1, heightDiv2, widthDiv1, Q);
    nodes += 1;
    int d4 = dnc(P, errorMethod, thresh, minblock,
x+widthDiv1, y+heightDiv1, heightDiv2, widthDiv2, Q);
    nodes += 1;

    depth = 1 + Math.Max(Math.Max(d1, d2), Math.Max(d3,
d4));
}

```

```

        return depth;
    }
}
else // selain SSIM
{
    int[,] PSliced = FillArray(P, x, y, height, width);

    // Hitung rata-rata tiap channel
    double rSum = 0, rMean = 0;
    double gSum = 0, gMean = 0;
    double bSum = 0, bMean = 0;
    for (int ii = 0; ii < width; ii++)
    {
        for (int jj = 0; jj < height; jj++)
        {
            rSum += PSliced[jj, ii, 0];
            gSum += PSliced[jj, ii, 1];
            bSum += PSliced[jj, ii, 2];
        }
    }
    rMean = rSum /N;
    gMean = gSum /N;
    bMean = bSum /N;

    error = ErrorMeasurement(errorMethod, PSliced);

    if (error < thresh) // berhenti apabila error kurang dari threshold
    {
        for (int ii = x; ii < width + x; ii++)
        {
            for (int jj = y; jj < height + y; jj++)
            {
                P[jj, ii, 0] = (int)rMean;
                P[jj, ii, 1] = (int)gMean;
                P[jj, ii, 2] = (int)bMean;
            }
        }
        return 1;
    }
    else if (N < minblock) // berhenti apabila jumlah piksel melebihi minblock
    {
        for (int ii = x; ii < width + x; ii++)
        {
            for (int jj = y; jj < height + y; jj++)
            {
                P[jj, ii, 0] = (int)rMean;
                P[jj, ii, 1] = (int)gMean;
                P[jj, ii, 2] = (int)bMean;
            }
        }
        return 1;
    }
    else
    {
        // bagi blok menjadi empat bagian
        int heightDiv1 = height / 2;
        int heightDiv2 = height - heightDiv1;

```

```

        int widthDiv1 = width / 2;
        int widthDiv2 = width - widthDiv1;

        // Selesaikan masing-masing blok
        int d1 = dnc(P, errorMethod, thresh, minblock, x,
y, heightDiv1, widthDiv1);
        nodes += 1;
        int d2 = dnc(P, errorMethod, thresh, minblock,
x+widthDiv1, y, heightDiv1, widthDiv2);
        nodes += 1;
        int d3 = dnc(P, errorMethod, thresh, minblock, x,
y+heightDiv1, heightDiv2, widthDiv1);
        nodes += 1;
        int d4 = dnc(P, errorMethod, thresh, minblock,
x+widthDiv1, y+heightDiv1, heightDiv2, widthDiv2);
        nodes += 1;

        depth = 1 + Math.Max(Math.Max(d1, d2), Math.Max(d3,
d4));
    }
}

// Fungsi untuk konversi matriks RGB menjadi file gambar
public static double ToImage(int[,] P, string outputPath,
int[,] alpha)
{
    int height = P.GetLength(0);
    int width = P.GetLength(1);
    double sizeMB;

    using (Image<Rgba32> image = new Image<Rgba32>(width,
height))
    {
        for (int j = 0; j < height; j++)
        {
            for (int i = 0; i < width; i++)
            {
                int r = P[j, i, 0];
                int g = P[j, i, 1];
                int b = P[j, i, 2];

                if (ext == ".png")
                {
                    int a = alpha[j, i];
                    image[i, j] = new Rgba32((byte)r, (byte)g,
(byte)b, (byte)a);
                }
                else
                {
                    image[i, j] = new Rgba32((byte)r, (byte)g,
(byte)b);
                }
            }
        }
        image.Save(outputPath);
    }

    IImageEncoder encoder = ext == ".png" ? new PngEncoder()
: new JpegEncoder();
}

```

```

        // Mengambil ukuran gambar (dalam MB) setelah di-compress
        using var memoryStream = new MemoryStream();
        image.Save(memoryStream, encoder);
        long size = memoryStream.Length;
        sizeMB = size / (1024.0*1024.0);
    }
    return sizeMB;
}

// Fungsi untuk meng-copy array untuk satu blok
public static int[,] FillArray(int[,] P, int istart, int
jstart, int height, int width)
{
    int[,] PSliced = new int[height, width, P.GetLength(2)];

    for (int i = 0; i < P.GetLength(2); i++)
    {
        for (int j = istart; j < istart+width; j++)
        {
            for (int k = jstart; k < jstart+height; k++)
                PSliced[k-jstart, j-istart, i] = P[k, j, i];
        }
    }
    return PSliced;
}

// Fungsi logaritma dengan basis 2
public static double log2<T>(T x) where T : struct
{
    return Math.Log(Convert.ToDouble(x)) / Math.Log(2);
}

// Fungsi untuk menghitung persen gambar yang sudah dikompreksi
public static double CompressPercent(double firstByte, double
secondByte)
{
    return (1 - (secondByte / firstByte)) * 100;
}

// Perhitungan eror untuk satu blok gambar
public static double ErrorMeasurement(int method, int[,] P,
int[,]? Q = null)
{
    if (method == 1) // Variance
    {
        int height = P.GetLength(0); // tinggi piksel dalam
satu blok
        int width = P.GetLength(1); // lebar piksel dalam satu
blok
        int channel = P.GetLength(2);
        int N = width * height; // banyak piksel dalam satu blok
        double varianSum = 0;
        double miu;
        double sum;
        double varRGB;

        for (int i = 0; i < channel; i++)
        {
            // Menghitung rata-rata

```

```

        sum = 0;
        for (int j = 0; j < width; j++)
        {
            for (int k = 0; k < height; k++)
            {
                sum += P[k, j, i];
            }
        }

        miu = sum / N;

        // Menghitung variansi dalam satu channel
        sum = 0;
        for (int j = 0; j < width; j++)
        {
            for (int k = 0; k < height; k++)
            {
                sum += Math.Pow((P[k, j, i] - miu), 2);
            }
        }

        varianSum += sum / N;
    }

    varRGB = varianSum / 3; // Rata-rata variansi RGB

    return varRGB;
}
else if (method == 2) // MAD
{
    int height = P.GetLength(0); // tinggi piksel dalam
satu blok
    int width = P.GetLength(1); // lebar piksel dalam satu
blok
    int channel = P.GetLength(2);
    int N = width * height; // banyak piksel dalam satu blok
    double madSum = 0;
    double sum;
    double miu;
    double madRGB;

    for (int i = 0; i < channel; i++)
    {
        // Menghitung rata-rata
        sum = 0;
        for (int j = 0; j < width; j++)
        {
            for (int k = 0; k < height; k++)
            {
                sum += P[k, j, i];
            }
        }

        miu = sum / N;

        // Menghitung jumlah MAD dalam satu channel
        sum = 0;
        for (int j = 0; j < width; j++)
        {
            for (int k = 0; k < height; k++)
            {

```

```

        {
            sum += Math.Abs(P[k, j, i] - miu);
        }
    }

    madSum += sum / N;
}

madRGB = madSum / 3; // Rata-rata MAD RGB

return madRGB;
}
else if (method == 3) // Max Pixel Difference
{
    int height = P.GetLength(0); // tinggi piksel dalam
satu blok
    int width = P.GetLength(1); // lebar piksel dalam satu
blok
    int channel = P.GetLength(2);
    double dSum = 0;
    int maks;
    int min;
    double dRGB;
    double dc;

    for (int i = 0; i < channel; i++)
    {
        // Mencari nilai maks dan min dalam satu blok
        maks = P[0, 0, i];
        min = P[0, 0, i];
        for (int j = 0; j < width; j++)
        {
            for (int k = 0; k < height; k++)
            {
                if (maks < P[k, j, i])
                {
                    maks = P[k, j, i];
                }
                if (min > P[k, j, i])
                {
                    min = P[k, j, i];
                }
            }
        }
        dc = maks - min;
        dSum += dc;
    }
    dRGB = dSum / 3; // Rata-rata MPD RGB

    return dRGB;
}
else if (method == 4) // Entropy
{
    int height = P.GetLength(0); // tinggi piksel dalam
satu blok
    int width = P.GetLength(1); // lebar piksel dalam satu
blok
    int channel = P.GetLength(2);
    int N = width * height; // banyak piksel dalam satu blok
    double hSum = 0;
}

```

```

        double hRGBSum = 0;
        double prob;
        double hRGB;
        int[,] p = new int[height, width];

        for (int i = 0; i < channel; i++)
        {
            Dictionary<int, int> valueCount = new
Dictionary<int, int>{};
            for (int j = 0; j < width; j++)
            {
                for (int k = 0; k < height; k++)
                {
                    p[k, j] = P[k, j, i];
                }
            }

            // Mendapatkan jumlah nilai piksel dalam satu blok
            foreach (int val in p)
            {
                if (valueCount.ContainsKey(val))
                {
                    valueCount[val]++;
                }
                else
                {
                    valueCount[val] = 1;
                }
            }

            foreach (var values in valueCount)
            {
                prob = (double)values.Value / N; // Hitung
probabilitas kemunculan nilai piksel dalam satu blok
                hSum -= prob * log2(prob);
            }

            hRGBSum += hSum;
        }

        hRGB = hRGBSum / 3; // Rata-rata entropi RGB
        return hRGB;
    }
    else if (method == 5) // SSIM
    {
        Q ??= P;

        int height = P.GetLength(0); // tinggi piksel dalam
satu blok
        int width = P.GetLength(1); // lebar piksel dalam satu
blok
        int channel = P.GetLength(2);
        int N = width * height; // banyak piksel dalam satu blok
        double pSum, qSum, pqSum;
        double pMean, qMean;
        double pVar, qVar, pqVar;
        double ssim = 0, ssimC;
        double C1, C2, K1 = 0.01, K2 = 0.03; // (Zhou Wang,
dkk; 2004)
        double[] w = {0.299, 0.587, 0.114}; // ITU-R BT.601-7
    }
}

```

```

        double l, cs;

        for (int i = 0; i < channel; i++)
        {
            pSum = 0;
            qSum = 0;
            // mean
            for (int j = 0; j < width; j++)
            {
                for (int k = 0; k < height; k++)
                {
                    pSum += P[k, j, i];
                    qSum += Q[k, j, i];
                }
            }
            pMean = pSum / N;
            qMean = qSum / N;

            // covariance and variance
            pSum = 0;
            qSum = 0;
            pqSum = 0;
            for (int j = 0; j < width; j++)
            {
                for (int k = 0; k < height; k++)
                {
                    pSum += Math.Pow((P[k, j, i] - pMean), 2);
// variance P
                    qSum += Math.Pow((Q[k, j, i] - qMean), 2);
// variance Q
                    pqSum += (P[k, j, i] - pMean) * (Q[k, j, i]
- qMean); // covariance PQ
                }
            }
            pVar = pSum / (N - 1);
            qVar = qSum / (N - 1);
            pqVar = pqSum / (N - 1);

            C1 = Math.Pow(K1*255, 2); // 255 -> RGB 24-bit @
8-bit per channel
            C2 = Math.Pow(K2*255, 2);

            l = (2*pMean*qMean + C1) / (Math.Pow(pMean, 2) +
Math.Pow(qMean, 2) + C1);
            cs = (2*pqVar + C2) / (pVar + qVar + C2);
            ssimC = l*cs;

            ssim += w[i] * ssimC;
        }
        return ssim;
    }
    return 0.0;
}

```

## b. Implementasi

Berdasarkan source code di atas, berikut merupakan penjelasan dari masing-masing fungsi dan prosedur yang digunakan.

<b>Class</b>	
Program	Berisi program utama yang berfungsi sebagai kompresi gambar dengan memanfaatkan algoritma divide and conquer dalam metode quadtree.
<b>Fungsi dan Prosedur</b>	
Main	Berisi pesan eror/input yang tidak valid, load gambar, pemanggilan algoritma divide and conquer, penyimpanan gambar, serta konfigurasi gambar hasil kompresi. Menerima input alamat absolut gambar, metode perhitungan eror, threshold, ukuran blok minimum, alamat absolut gambar hasil kompresi.
dnc	Algoritma utama divide and conquer dalam kompresi gambar menggunakan metode quadtree. Fungsi ini mengembalikan kedalaman pohon.
ToImage	Mengkonversi matriks gambar menjadi file gambar sesuai dengan ekstensi file gambar input. Fungsi ini mengembalikan ukuran file gambar dalam MB.
FillArray	Fungsi untuk meng-copy array untuk satu blok, untuk keperluan perhitungan eror.
log2	Fungsi perhitungan log berbasis 2.
CompressPercent	Fungsi untuk menghitung persentase hasil kompresi.
ErrorMeasurement	Fungsi untuk menghitung eror berdasarkan metode yang sudah dipilih.

### 3. Hasil dan Analisis

#### a. Hasil

Konfigurasi	Input Gambar	Output Gambar
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor\dectrot run "D:/S11 MA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample1.jpg" 1 8 8 "D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample1_ compressed.jpg" Gambar dimuat dengan ukuran: 3744x5616 px Menghitung gambar... Waktu eksekusi: 65.00 detik Ukuran gambar sebelum di-compress: 4.73 MB Ukuran gambar setelah di-compress: 3.02 MB Persentase kompresi: 36.13% Kedalaman pohon: 12 Banyak simpul: 5592405 Gambar tersimpan di D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompresso r/images/sample1_compressed.jpg</pre> <ul style="list-style-type: none"> <li>- Eror: Variance (1)</li> <li>- Threshold: 0.1</li> <li>- Min. Block: 8</li> <li>- Waktu eksekusi: 65 s</li> <li>- Ukuran gambar sebelum dikompresi: 4.73 MB</li> <li>- Ukuran gambar setelah dikompresi: 3.02 MB</li> <li>- Persentase: 36.13%</li> <li>- Kedalaman pohon: 12</li> <li>- Banyak simpul: 5592405</li> </ul>		
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor\dectrot run "D:/S11 MA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample1.jpg" 1 50 8 "D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample1_ compressed2.jpg" Gambar dimuat dengan ukuran: 3744x5616 px Menghitung gambar... Waktu eksekusi: 65.00 detik Ukuran gambar sebelum di-compress: 4.73 MB Ukuran gambar setelah di-compress: 2.98 MB Persentase kompresi: 36.87% Kedalaman pohon: 12 Banyak simpul: 5384549 Gambar tersimpan di D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompresso r/images/sample1_compressed2.jpg</pre> <ul style="list-style-type: none"> <li>- Eror: Variance (1)</li> <li>- Threshold: 50</li> <li>- Min. Block: 8</li> <li>- Waktu eksekusi: 65 s</li> <li>- Ukuran gambar sebelum dikompresi: 4.73 MB</li> <li>- Ukuran gambar setelah dikompresi: 2.98 MB</li> <li>- Persentase: 36.87%</li> <li>- Kedalaman pohon: 12</li> <li>- Banyak simpul: 5384549</li> </ul>		
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor\dectrot run "D:/S11 MA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample2.jpeg" 1 4 16 "D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample2_ compressed.jpg" Gambar dimuat dengan ukuran: 1280x853 px Menghitung gambar... Waktu eksekusi: 1.00 detik Ukuran gambar sebelum di-compress: 0.32 MB Ukuran gambar setelah di-compress: 0.14 MB Persentase kompresi: 55.67% Kedalaman pohon: 10 Banyak simpul: 106785 Gambar tersimpan di D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompresso r/images/sample2_compressed.jpg</pre> <ul style="list-style-type: none"> <li>- Eror: MAD (2)</li> <li>- Threshold: 4</li> <li>- Min. Block: 16</li> <li>- Waktu eksekusi: 1 s</li> <li>- Ukuran gambar sebelum dikompresi: 0.32 MB</li> </ul>		

<ul style="list-style-type: none"> <li>- Ukuran gambar setelah dikompresi: 0.14 MB</li> <li>- Persentase: 55.67%</li> <li>- Kedalaman pohon: 10</li> <li>- Banyak simpul: 106785</li> </ul>		
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sample2.jpeg" 2 2 0 16 "D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sample2_compressed2.jpg" Gambar dimuat dengan ukuran: 1280x853 px Mengompres gambar... Waktu eksekusi: 0.00 detik Ukuran gambar sebelum di-compress: 0.32 MB Ukuran gambar setelah di-compress: 0.09 MB Persentase kompresi: 70.84% Kedalaman pohon: 10 Banyak simpul: 30293 Gambar tersimpan di D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sample2_compressed2.jpg</pre> <ul style="list-style-type: none"> <li>- Eror: MAD (2)</li> <li>- Threshold: 20</li> <li>- Min. Block: 16</li> <li>- Waktu eksekusi: 0 s</li> <li>- Ukuran gambar sebelum dikompresi: 0.32 MB</li> <li>- Ukuran gambar setelah dikompresi: 0.09 MB</li> <li>- Persentase: 70.84%</li> <li>- Kedalaman pohon: 10</li> <li>- Banyak simpul: 38293</li> </ul>		
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sample3.png" 3 2 0 16 "D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sample3_compressed.png" Gambar dimuat dengan ukuran: 2480x1680 px Mengompres gambar... Waktu eksekusi: 0.00 detik Ukuran gambar sebelum di-compress: 0.73 MB Ukuran gambar setelah di-compress: 0.62 MB Persentase kompresi: -13.27% Kedalaman pohon: 11 Banyak simpul: 30961 Gambar tersimpan di D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sample3_compressed.png</pre> <ul style="list-style-type: none"> <li>- Eror: MPD (3)</li> <li>- Threshold: 2</li> <li>- Min. Block: 16</li> <li>- Waktu eksekusi: 3 s</li> <li>- Ukuran gambar sebelum dikompresi: 0.73 MB</li> <li>- Ukuran gambar setelah dikompresi: 0.82 MB</li> <li>- Persentase: -13.27%</li> <li>- Kedalaman pohon: 11</li> <li>- Banyak simpul: 385961</li> </ul>		

<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STI MA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample3.png" 3 60 16 "D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample 3_compressed.png" Gambar dimuat dengan ukuran: 2400x1600 px Mengompres gambar... Waktu eksekusi: 2.00 detik Ukuran gambar sebelum di-compress: 0.73 MB Ukuran gambar setelah di-compress: 0.35 MB Persentase kompresi: 51.55% Kedalaman pohon: 11 Banyak simpul: 95841 Gambar tersimpan di D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompre ssor/images/sample3_compressed2.png</pre> <ul style="list-style-type: none"> <li>- Eror: MPD (3)</li> <li>- Threshold: 60</li> <li>- Min. Block: 16</li> <li>- Waktu eksekusi: 2 s</li> <li>- Ukuran gambar sebelum dikompresi: 0.73 MB</li> <li>- Ukuran gambar setelah dikompresi: 0.35 MB</li> <li>- Persentase: 51.55%</li> <li>- Kedalaman pohon: 11</li> <li>- Banyak simpul: 95841</li> </ul>		
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STI MA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample4.png" 4 1 8 "D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample 4_compressed.png" Gambar dimuat dengan ukuran: 2079x2571 px Mengompres gambar... Waktu eksekusi: 10.00 detik Ukuran gambar sebelum di-compress: 4.28 MB Ukuran gambar setelah di-compress: 1.33 MB Persentase kompresi: 69.01% Kedalaman pohon: 12 Banyak simpul: 742325 Gambar tersimpan di D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompre ssor/images/sample4_compressed.png</pre> <ul style="list-style-type: none"> <li>- Eror: Entropy (4)</li> <li>- Threshold: 1</li> <li>- Min. Block: 8</li> <li>- Waktu eksekusi: 10 s</li> <li>- Ukuran gambar sebelum dikompresi: 4.28 MB</li> <li>- Ukuran gambar setelah dikompresi: 1.33 MB</li> <li>- Persentase: 69.01%</li> <li>- Kedalaman pohon: 12</li> <li>- Banyak simpul: 742325</li> </ul>		
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STI MA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample4.png" 4 4 8 "D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompressor/images/sample 4_compressed.png" Gambar dimuat dengan ukuran: 2079x2571 px Mengompres gambar... Waktu eksekusi: 8.00 detik Ukuran gambar sebelum di-compress: 4.28 MB Ukuran gambar setelah di-compress: 1.24 MB Persentase kompresi: 71.01% Kedalaman pohon: 12 Banyak simpul: 682605 Gambar tersimpan di D:/STIMA/tucil_2/repo/ImageCompressor/ImageCompre ssor/images/sample4_compressed2.png</pre> <ul style="list-style-type: none"> <li>- Eror: Entropy (4)</li> <li>- Threshold: 4</li> <li>- Min. Block: 8</li> <li>- Waktu eksekusi: 8 s</li> <li>- Ukuran gambar sebelum dikompresi: 4.28 MB</li> <li>- Ukuran gambar setelah dikompresi: 1.24 MB</li> <li>- Persentase: 71.01%</li> <li>- Kedalaman pohon: 12</li> <li>- Banyak simpul: 682605</li> </ul>		

<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STI MA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sample5.jpeg" 5 0 7 8 10 /D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sampl e5_compressed.jpg" Gambar diimut dengan ukuran: 400x300 px Mengkompres gambar... Waktu eksekusi: 0.00 detik Ukuran gambar sebelum dikompress: 0.12 MB Ukuran gambar setelah dikompress: 0.03 MB Percentase kompresi: 76.58% Kedalaman pohon: 10 Banyak simpul: 19 Gambar versiimpai di D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompresso r/images/sample5_compressed.jpeg</pre> <ul style="list-style-type: none"> <li>- Eror: SSIM (5)</li> <li>- Threshold: 0.7</li> <li>- Min. Block: 4</li> <li>- Waktu eksekusi: 0 s</li> <li>- Ukuran gambar sebelum dikompresi: 0.12 MB</li> <li>- Ukuran gambar setelah dikompresi: 0.03 MB</li> <li>- Percentase: 76.58%</li> <li>- Kedalaman pohon: 10</li> <li>- Banyak simpul: 90589</li> </ul>		
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STI MA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sample1.jpg" 5 0 7 8 10 /D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sampl e1_compressed2.jpg" Gambar diimut dengan ukuran: 400x300 px Mengkompres gambar... Waktu eksekusi: 0.00 detik Ukuran gambar sebelum dikompress: 0.12 MB Ukuran gambar setelah dikompress: 0.02 MB Percentase kompresi: 80.56% Kedalaman pohon: 10 Banyak simpul: 35781 Gambar versiimpai di D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompresso r/images/sample1_compressed2.jpg</pre> <ul style="list-style-type: none"> <li>- Eror: SSIM (5)</li> <li>- Threshold: 0.2</li> <li>- Min. Block: 4</li> <li>- Waktu eksekusi: 0 s</li> <li>- Ukuran gambar sebelum dikompresi: 0.12 MB</li> <li>- Ukuran gambar setelah dikompresi: 0.02 MB</li> <li>- Percentase: 80.56%</li> <li>- Kedalaman pohon: 10</li> <li>- Banyak simpul: 35781</li> </ul>		
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STI MA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sample6.jpg" 5 0 7 8 10 /D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor/images/sampl e6_compressed.jpg" Gambar diimut dengan ukuran: 1920x1462 px Mengkompres gambar... Waktu eksekusi: 14.00 detik Ukuran gambar sebelum dikompress: 1.03 MB Ukuran gambar setelah dikompress: 0.70 MB Percentase kompresi: 32.48% Kedalaman pohon: 11 Banyak simpul: 1077313 Gambar versiimpai di D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompresso r/images/sample6_compressed.jpg</pre> <ul style="list-style-type: none"> <li>- Eror: SSIM (5)</li> <li>- Threshold: 0.7</li> <li>- Min. Block: 8</li> <li>- Waktu eksekusi: 14 s</li> <li>- Ukuran gambar sebelum dikompresi: 1.03 MB</li> <li>- Ukuran gambar setelah dikompresi: 0.7 MB</li> <li>- Percentase: 32.48%</li> <li>- Kedalaman pohon: 11</li> <li>- Banyak simpul: 1077313</li> </ul>		

<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STI MA\tucil_2\repo\ImageCompressor\ImageCompressor\images\sample6.jpg" 5 0 8 128 "D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor\images\sample 6_compressed.jpg" Gambar dimut dengan ukuran: 1920x1462 px Mengkompres gambar... Waktu eksekusi: 11.00 detik Ukuran gambar sebelum dikompress: 1.03 MB Ukuran gambar setelah di-kompress: 0.42 MB Percentase kompresi: 59.41% Kedalaman pohon: 9 Banyak simpul: 79541 Gambar tersimpan di D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompre ssor\images\sample6_compressed2.jpg</pre> <ul style="list-style-type: none"> <li>- Eror: SSIM (5)</li> <li>- Threshold: 0.7</li> <li>- Min. Block: 128</li> <li>- Waktu eksekusi: 11 s</li> <li>- Ukuran gambar sebelum dikompresi: 1.03 MB</li> <li>- Ukuran gambar setelah dikompresi: 0.42 MB</li> <li>- Percentase: 59.41%</li> <li>- Kedalaman pohon: 9</li> <li>- Banyak simpul: 79541</li> </ul>		
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STI MA\tucil_2\repo\ImageCompressor\ImageCompressor\images\sample7.jpg" 4 1 8 128 "D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor\images\sample 7_compressed.jpg" Gambar dimut dengan ukuran: 5000x3333 px Mengkompres gambar... Waktu eksekusi: 49.00 detik Ukuran gambar sebelum dikompress: 10.40 MB Ukuran gambar setelah di-kompress: 2.01 MB Percentase kompresi: 80.64% Kedalaman pohon: 12 Banyak simpul: 5517049 Gambar tersimpan di D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompre ssor\images\sample7_compressed.jpg</pre> <ul style="list-style-type: none"> <li>- Eror: Entropy (4)</li> <li>- Threshold: 1</li> <li>- Min. Block: 8</li> <li>- Waktu eksekusi: 49 s</li> <li>- Ukuran gambar sebelum dikompresi: 10.40 MB</li> <li>- Ukuran gambar setelah dikompresi: 2.01 MB</li> <li>- Percentase: 80.64%</li> <li>- Kedalaman pohon: 12</li> <li>- Banyak simpul: 5517049</li> </ul>		
<pre>D:\STIMA\tucil_2\repo\ImageCompressor\ImageCompressor&gt;dotnet run "D:/STI MA\tucil_2\repo\ImageCompressor\ImageCompressor\images\sample7.jpg" 4 1 8 128 "D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompressor\images\sample 7_compressed.jpg" Gambar dimut dengan ukuran: 5000x3333 px Mengkompres gambar... Waktu eksekusi: 49.00 detik Ukuran gambar sebelum dikompress: 10.40 MB Ukuran gambar setelah di-kompress: 1.61 MB Percentase kompresi: 85.46% Kedalaman pohon: 10 Banyak simpul: 349337 Gambar tersimpan di D:/STIMA\tucil_2\repo\ImageCompressor\ImageCompre ssor\images\sample7_compressed2.jpg</pre> <ul style="list-style-type: none"> <li>- Eror: Entropy (4)</li> <li>- Threshold: 1</li> <li>- Min. Block: 128</li> <li>- Waktu eksekusi: 37 s</li> <li>- Ukuran gambar sebelum dikompresi: 10.40 MB</li> <li>- Ukuran gambar setelah dikompresi: 1.51 MB</li> <li>- Percentase: 85.46%</li> <li>- Kedalaman pohon: 10</li> <li>- Banyak simpul: 349337</li> </ul>		

## b. Analisis

Berdasarkan hasil yang diperoleh pada bagian Hasil, didapatkan bahwa rata-rata gambar yang dihasilkan memiliki ukuran yang lebih kecil dari ukuran gambar aslinya. Untuk gambar pertama, digunakan metode eror variance dengan threshold 0.1 dan 50 dengan minblock 8. Waktu eksekusi untuk mengkompresi gambar ini adalah 65 detik. Berdasarkan gambar pertama, didapatkan bahwa semakin besar threshold variance, banyak simpul quadtree yang semakin sedikit sehingga ukuran gambarnya semakin kecil (persentase kompresinya membesar) atau gambarnya semakin kasar. Hal ini terjadi karena ketika gambar memiliki eror yang besar, gambar semakin tidak homogen (gambar masih memiliki struktur/piksel yang kompleks) sehingga ketika program berhenti pada keadaan tersebut, gambar yang dihasilkan akan semakin kasar. Begitupun untuk gambar sampel 2, 4, dan 5 menunjukkan hasil yang sama. Pada gambar sampel 5, semakin besar nilai SSIM, maka gambar hasil kompresi akan semakin mirip dengan gambar aslinya. Oleh karena itu, ketika nilai SSIM semakin besar, maka gambarnya akan semakin detail sehingga ukuran file-nya semakin besar atau mendekati file aslinya.

Namun, pada gambar sampel 3, ukuran hasil gambar kompresi lebih besar dari gambar aslinya. Hal ini dapat terjadi karena metrik eror yang digunakan adalah maximum pixel difference (MPD) dan threshold, serta jumlah minimum blok yang digunakan terlalu kecil. Hal ini dapat terjadi karena gambar memiliki variasi piksel tinggi atau gambar yang sangat detail (berdimensi besar). Ketika suatu blok gambar memiliki variasi piksel yang tinggi, maka nilai MPD-nya semakin besar. Apabila nilai MPD pada blok ini masih di atas threshold, gambar akan dibagi terus-menerus hingga nilainya di bawah threshold. Ketika jumlah blok terlalu banyak, informasi nilai rata-rata setiap blok yang harus disimpan semakin banyak sehingga ukuran file-nya menjadi semakin besar (karena terlalu detail).

Pada gambar sampel 6 dan 7 dilakukan variasi jumlah blok minimum pada masing-masing gambar, dengan konfigurasi eror yang digunakan berdasarkan persentase tertinggi pertama dan kedua pada gambar hasil kompresi untuk sampel 1 - 5. Berdasarkan hasil kedua sampel tersebut, dapat dilihat bahwa semakin besar jumlah minimum ukuran blok, maka akan semakin kecil ukuran gambarnya (semakin besar persentase kompresi). Selain itu, pohon quadtree juga menjadi semakin sederhana ketika jumlah minimum blok dinaikkan. Hal ini dapat terjadi karena ketika eror setiap blok masih di atas (di bawah, untuk SSIM) threshold dan ukuran blok sudah di bawah ukuran blok minimum, pembagian blok tidak akan dilakukan lagi sehingga blok tersebut akan langsung diisi dengan rata-rata nilai piksel di blok tersebut. Dengan kata lain, menaikkan nilai minimum blok akan membatasi kedalaman pohon sehingga blok yang dibagi menjadi lebih sedikit. Hal ini dapat menyebabkan pohon quadtree semakin sederhana, yang pada akhirnya berkontribusi pada ukuran file yang semakin kecil, meskipun kualitas gambarnya menjadi sedikit lebih kasar.

Kompleksitas algoritma pada implementasi quadtree ini adalah sebagai berikut.

- Kasus dibagi menjadi tiga, yaitu kasus terburuk (piksel gambar sangat variatif), kasus rata-rata, dan kasus terbaik (piksel gambar sangat sederhana).
- Asumsikan gambar yang digunakan berbentuk persegi  $n \times n = n^2$ .
- Setiap pemanggilan algoritma divide and conquer, gambar dibagi menjadi 4 blok dan terdapat pemanggilan rekursif sebanyak 4 kali, untuk kasus terburuk.
- Asumsikan pada fungsi ErrorMeasurement dan FillArray dalam satu blok memiliki kompleksitas algoritma  $O(wxh)$  atau  $O(n^2)$  dalam kasus terburuk. Maka, untuk kasus terburuk

$$T(n) = 4T(n/2) + O(n^2)$$

- Dengan  $a = 4$ ,  $b = 2$ ,  $f(n) = O(n^2)$ , berdasarkan Master Theorem. Maka untuk kasus terburuk

$$T(n) = O(n^2 \cdot \log n)$$

- Untuk kasus terbaik, hanya terjadi satu kali pengecekan, tanpa ada rekursi. Oleh karena itu kompleksitas algoritmanya

$$T(n) = O(n^2)$$

- Untuk kasus rata-rata, umumnya mendekati kasus terburuk, tetapi tergantung pada nilai threshold (eror atau minimum blok) atau kedalaman pohon

$$T(n) \approx O(n^2 \cdot \log k)$$

Dengan  $k$  adalah kedalaman pohon.

Pada perhitungan SSIM, pada saat pembagian blok, akan dihitung nilai SSIM berdasarkan blok yang sudah dirata-ratakan dengan blok gambar yang asli. Apabila nilai SSIM besar (mendekati 1), maka kedua blok sudah cukup mirip sehingga tidak akan lagi dilakukan pembagian blok. Sebaliknya, apabila nilai SSIM masih kecil (kurang dari 1 dan lebih besar dari 0), maka kedua blok masih belum cukup mirip sehingga akan dilakukan pembagian blok (apabila jumlah minimum bloknya masih memenuhi). Pada perhitungan SSIM terdapat tiga komponen penting, yaitu luminance, contrast, dan structure (biasanya contrast dan structure disatukan secara matematis). Luminance pada SSIM membandingkan kecerahan rata-rata antara kedua gambar, apabila nilai ini mendekati 1, maka kedua blok memiliki kecerahan yang mirip. Contrast pada SSIM membandingkan perbedaan variasi antara kedua gambar, apabila nilai ini mendekati 1, maka kedua blok memiliki kontras yang mirip. Structure pada SSIM membandingkan pola struktur antar gambar, yaitu mengukur perubahan intensitas piksel antara kedua gambar, apabila nilai ini semakin mendekati 1, maka hal ini menunjukkan kedua gambar saling berkorelasi. Nilai SSIM merupakan perkalian antara ketiga komponen tersebut.

#### **4. Kesimpulan**

Pada tugas ini, penulis telah berhasil membuat program kompresi gambar quadtree dengan algoritma divide and conquer. Berdasarkan sampel gambar yang dihasilkan, rata-rata gambar hasil kompresi memiliki ukuran file lebih kecil dari file aslinya, yang berarti proses kompresi berhasil dilakukan. Pada implementasi ini, terdapat lima perhitungan metrik yang digunakan, yaitu variance, mean absolute error (MAD), maximum pixel difference (MPD), entropy, dan SSIM. Berdasarkan hasil yang diperoleh, semakin besar threshold atau jumlah blok minimum, maka ukuran file gambar semakin kecil. Begitupun sebaliknya, apabila threshold atau jumlah minimum blok semakin kecil, maka ukuran file gambar semakin besar hingga mendekati ukuran file aslinya. Selain itu, kompleksitas algoritma untuk kasus terburuk adalah  $O(n^2 \cdot \log n)$ , kasus terbaik  $O(n^2)$ , dan kasus rata-rata mendekati  $O(n^2 \cdot \log k)$ . Oleh karena itu, penulis telah berhasil mengimplementasikan kompresi gambar quadtree dengan algoritma divide and conquer menggunakan bahasa C#.

## **REFERENSI**

International Telecommunication Union. ITU-R BT.601-7. 2017

York, T. (2020, July 15). *Quadtrees for Image Processing*. Medium.  
<https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>

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

## LAMPIRAN

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