LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis *Divide and Conquer*

Disusun oleh:

Agil Fadillah Sabri (13522006)

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

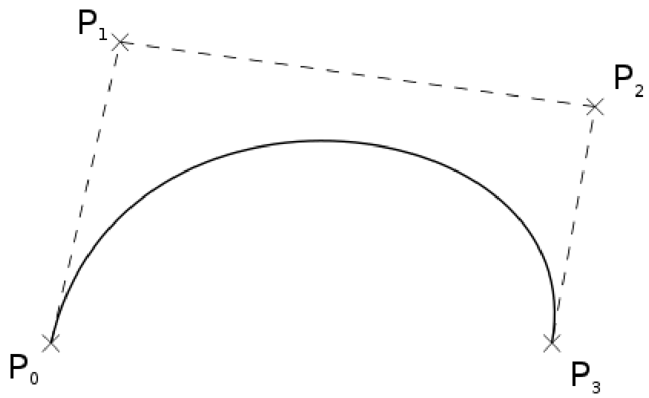
DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
DESKRIPSI MASALAH

Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol, yang menentukan bentuk dan arah kurva. Cara membuatnya cukup mudah, yaitu dengan menentukan titik-titik kontrol dan menghubungkannya dengan kurva. Kurva Bézier memiliki banyak kegunaan dalam kehidupan nyata, seperti *pen tool*, animasi yang halus dan realistis, membuat desain produk yang kompleks dan presisi, dan membuat font yang indah dan unik. Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan.



Gambar 1. Kurva Bézier Kubik

Sumber: https://id.wikipedia.org/wiki/Kurva_B%C3%A9zier

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P_0 sampai P_n, dengan n disebut orde (n = 1 untuk linier, n = 2 untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) umumnya tidak terletak pada kurva. Pada gambar 1 diatas, titik kontrol pertama adalah P_0, sedangkan titik kontrol terakhir adalah P_3. Titik kontrol P_1 dan P_2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Sebuah kurva Bézier berderajat n, dapat didefinisikan melalui persamaan berparameter sebagai berikut:

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} (1 - t)^{n-i} t^i \mathbf{P}_i , \quad 0 \le t \le 1$$

dimana : B(t) : Koordinat titik kurva Bézier pada saat t

P_i : Titik kontrol ke-i.

n : Derajat kurva Bézier

$\binom{n}{i}$: Koefisien binomial/kombinasi (${}_nC_i$)

Selain melalui persamaan di atas, sebuah kurva Bézier berderajat n juga dapat dibentuk dengan memanfaatkan algoritma titik tengah berbasis *divide and conquer*.

BAB II
LANGKAH PEMECAHAN MASALAH

2.1 Pembuatan Kurva Bézier dengan Pendekatan *Brute Force*

Pembuatan kurva Bézier menggunakan pendekatan *brute force*, dapat dilakukan dengan mencari satu-persatu titik-titik yang akan membentuk kurva Bézier menggunakan persamaan yang telah disebutkan pada bab sebelumnya, yaitu:

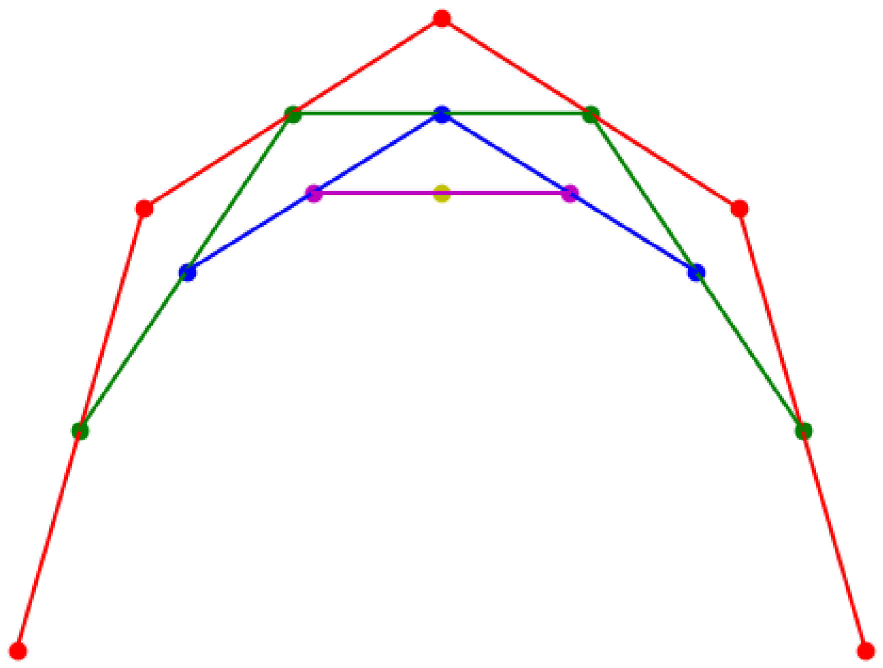
$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} (1 - t)^{n-i} t^i \mathbf{P}_i , \quad 0 \le t \le 1$$

Jika misal ingin dihasilkan kurva Bézier dengan banyak titik sebanyak *k* (termasuk titik kontrol pertama dan titik kontrol terakhir), maka dapat dilakukan dengan langkah-langkah sebagai berikut:

1. Hitung jarak antar titik dalam *t* sebagai *dt*, dengan:
$$dt = \frac{1}{k - 1}$$
2. Lakukan iterasi sebanyak *k* kali, dimulai dengan nilai *t* = 0 pada saat iterasi pertama.
3. Masukkan nilai *t* tersebut ke dalam persamaan di atas pada setiap iterasinya.
4. Simpan titik yang diperoleh ke dalam sebuah daftar/*list*.
5. Tambahkan nilai *t* sebesar *dt* di akhir setiap iterasi.
6. Pada iterasi yang terakhir, yaitu saat jumlah perulangan telah dilakukan sebanyak *k* kali, nilai *t* akan sama dengan 1.
7. Setelah semua titik dihasilkan, hubungkan setiap titik yang berdekatan dengan sebuah garis, maka terbentuklah kurve Bézier berderajat *n*.

2.2 Pembuatan Kurva Bézier dengan Pendekatan *Divide and Conquer*

Selain menggunakan metode *brute force*, pembuatan kurva Bézier juga dapat dilakukan dengan pendekatan *divide and conquer*, yaitu menggunakan algoritma titik tengah. Pada metode ini, pada setiap langkahnya akan dicari himpunan titik tengah di antara dua titik yang saling berdekatan (pada gambar di bawah, ditunjukkan oleh dua titik yang terhubung oleh sebuah garis secara langsung). Untuk himpunan titik yang baru, proses diulangi kembali berkali-kali hingga pada suatu waktu hanya satu titik yang dihasilkan. Satu titik terakhir inilah yang menjadi titik yang akan membentuk kurva Bézier. Adapun ilustrasinya diberikan oleh gambar dibawah ini.

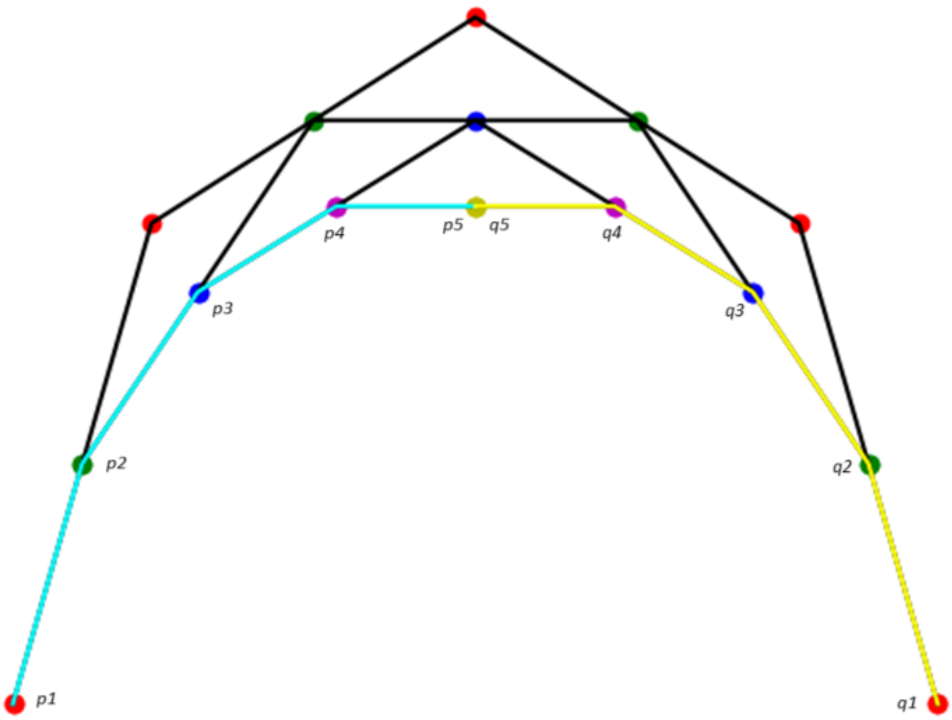


Gambar 2. Himpunan Titik Tengah (Orde = 4)

Pada awalnya, terdapat himpunan titik merah sebagai titik kontrol untuk pembentukan kurva Bézier. Dari himpunan titik merah tersebut, dilakukan pencarian titik tengah antara dua titik yang berdekatan, dihasilkanlah himpunan titik-titik hijau. Ulangi proses untuk himpunan titik hijau untuk membentuk himpunan titik biru, dan seterusnya. Pada satu waktu, hanya akan dihasilkan satu titik tengah, yang pada gambar di atas ditunjukkan oleh titik berwarna kuning. Titik berwarna kuning tersebutlah yang akan menjadi titik pembentuk kurva Bézier.

Melalui cara di atas, baru satu titik yang dihasilkan. Hal ini tentu tidak cukup untuk menghasilkan kurva Bézier yang cukup mulus. Diperlukan pencarian titik-titik yang lain agar dihasilkan kurva Bézier yang mulus. Di sinilah algoritma *divide and conquer* berperan.

Algoritma *divide and conquer* digunakan untuk mencari titik-titik kurva Bézier selanjutnya. Adapun titik kontrol yang digunakan tidak lagi himpunan titik kontrol sebelumnya, namun menggunakan titik kontrol yang baru. Adapun ilustrasi untuk mendapatkan titik kontrol yang baru diberikan oleh gambar di bawah.



Gambar 3. Himpunan Titik Kontrol Baru

Dari gambar di atas, titik yang berada pada garis berwarna biru dan kuning merupakan himpunan titik-titik yang akan menjadi titik kontrol untuk mendapatkan titik kurva Bézier selanjutnya. Disinilah proses *divide* dilakukan, yaitu dengan membagi himpunan titik tersebut menjadi dua himpunan berbeda, yaitu himpunan titik kiri (yaitu titik-titik yang berada pada garis berwarna biru) dan himpunan titik kanan (yaitu titik-titik yang berada pada garis berwarna kuning). Masing-masing himpunan titik ini akan menjadi titik kontrol untuk memperoleh titik kurva Bézier selanjutnya, melalui cara yang sama dengan yang dijelaskan pada bagian sebelumnya.

Adapun cara untuk memperoleh himpunan titik-titik tersebut dapat dilihat dari pola yang ditunjukkan pada gambar di atas. Himpunan titik kiri, yang beranggotakan titik *p1*, *p2*, *p3*, *p4*, dan *p5*, merupakan titik pertama dari setiap titik tengah pada proses sebelumnya (*p1* adalah titik pertama dari himpunan titik merah, *p2* merupakan titik pertama dari himpunan titik hijau, dan *p3* adalah titik pertama dari himpunan titik biru, dan seterusnya). Adapun himpunan titik kanan, yang beranggotakan titik *q1*, *q2*, *q3*, *q4*, *q5* diperoleh dengan cara yang hampir mirip dengan himpunan titik kiri, yang hanya berbeda dari titik yang diambil. Pada himpunan titik kanan, titik yang diambil merupakan titik terakhir dari setiap titik tengah pada proses sebelumnya (*q1* adalah titik terakhir dari himpunan titik merah, *q2* merupakan titik terakhir dari himpunan titik hijau, dan *q3* adalah titik terakhir dari himpunan titik biru, dan seterusnya). Ilustrasi ini mengasumsikan bahwa titik kontrol juga merupakan himpunan titik tengah.

Dalam pembuatan titik kontrol yang baru, urutan titik-titik yang dibentuk penting. Titik-titik yang bersebelahan harus tidak bisa ditukar dengan titik lain, karena akan mengubah bentuk kurva Bézier. Berdasarkan gambar di atas, maka urutan titik kontrol untuk himpunan titik kiri adalah *p1* → *p2* → *p3* → *p4* → *p5*. Urutan ini dapat dicerminkan, asalkan titik yang bersebelahan tetap bersebelahan, yaitu *p5* → *p4* → *p3* → *p2* → *p1*.

Adapun proses *conquer* dilakukan dengan mencari titik pembentuk kurva Bézier dari himpunan titik kontrol yang diberikan seperti yang telah dijelaskan pada halaman 3.

Proses ini secara keseluruhan dapat dilakukan dengan menggunakan pendekatan secara rekursif. Adapun langkah-langkah lengkapnya yaitu:

1. Diberikan himpunan titik kontrol awal yang akan menjadi titik kontrol untuk membentuk kurva Bézier.
2. Cari titik kurva Bézier untuk himpunan titik kontrol tersebut menggunakan cara yang telah dijelaskan pada halaman 3 (bagian *conquer*).
3. Cari himpunan titik kontrol baru sebagaimana yang dijelaskan pada halaman 4 dari himpunan titik kontrol saat ini. Pada bagian ini, akan diperoleh himpunan titik kontrol bagian kiri dan himpunan titik kontrol bagian kanan (bagian *divide*).
4. Untuk masing-masing himpunan titik kontrol yang baru, selesaikan secara terpisah dengan mengulangi proses 2 dan 3 (bagian *divide* sekaligus *conquer*).
5. Ulangi proses di atas sebanyak yang diinginkan.
6. Gabungkan solusi himpunan titik dengan urutan (bagian *combine*):
   (hasil dari himpunan titik kontrol baru bagian kiri) + (titik kurva Bézier dari himpunan titik kontrol saat ini) + (hasil dari himpunan titik kontrol baru bagian kanan).
7. Setelah itu, hubungkan setiap titik-titik yang berdekatan dengan garis untuk membentuk kurva Bézier secara utuh.

Untuk mengontrol kedalaman proses rekursif yang dilakukan, diperlukan sebuah pencatat, misal *i*, yang nilainya selalu berkurang 1 setiap kali prosedur sebelumnya dipanggil. Jika nilai *i* telah mencapai 1, maka prosedur tidak lagi dipanggil dan langsung mengembalikan titik kurva Bézier saat itu.

BAB III
IMPLEMENTASI PROGRAM

Program pembuatan kurva Bézier ini diimplementasikan menggunakan bahasa pemrograman python (.py). Adapun untuk proses visualisasi memanfaatkan *library* matplotlib.

3.1 Implementasi dalam Algoritma *Brute Force*

1. Kurva Bézier Kuadratik

```python
# Fungsi yang menghasilkan kurva bezier kuadratik menggunakan metode brute force
# Fungsi ini mengembalikan list of tuple yang berisi koordinat titik-titik yang membentuk kurva bezier
# Fungsi ini:
#   - menerima 4 parameter, yaitu
#       List of titik kontrol
#       parameter iterasi yang menentukan tingkat kedetailan kurva bezier
#       parameter animasikan yang menentukan apakah tiap iterasi akan di animasikan atau tidak
#       parameter pause yang menentukan lama jeda antar animasi
#   - mengembalikan list of tuple dengan
#       jumlah titik yang dihasilkan adalah (2^(iterasi) + 1) (sudah termasuk titik kontrol awal dan akhir)
def BF_Quadratik_Bezier(Titik_Kontrol : list, iterasi : int, animasikan : bool, pause : float) -> list:
    # Inisialisasi list of tuple yang akan menampung koordinat titik-titik kurva bezier
    hasil = []
    # Hitung jumlah titik yang dihasilkan, belum termasuk titik kontrol awal dan akhir
    n = 2 ** iterasi - 1
    # hitung interval nilai t
    dt = 1 / (n + 1)
    # Hitung koordinat tiap titik kurva bezier
    for i in range(n + 2):
        t = i * dt
        x = (1 - t) ** 2 * Titik_Kontrol[0][0] + 2 * (1 - t) * t * Titik_Kontrol[1][0] + t ** 2 * Titik_Kontrol[2][0]
        y = (1 - t) ** 2 * Titik_Kontrol[0][1] + 2 * (1 - t) * t * Titik_Kontrol[1][1] + t ** 2 * Titik_Kontrol[2][1]

        if animasikan:
            Animation.animate_with_pause([(x, y)], 1, pause)

        hasil.append((x, y))

    return hasil
```

Gambar 4. Implementasi Pembuatan Kurva Bézier Kuadratik dalam Bahasa Python dengan Algoritma *Brute Force*

2. Kurva Bézier N-Titik (N ≥ 1)

```python
# Fungsi yang menghasilkan generalisasi kurva bezier menggunakan metode brute force (banyak titik kontrol >= 1)
# Fungsi ini mengembalikan list of tuple yang berisi koordinat titik-titik yang membentuk kurva bezier
# Fungsi ini:
#   - menerima 4 parameter, yaitu
#       List of titik kontrol
#       parameter iterasi yang menentukan tingkat kedetailan kurva bezier
#       parameter animasikan yang menentukan apakah tiap iterasi akan di animasikan atau tidak
#       parameter pause yang menentukan lama jeda antar animasi
#   - mengembalikan list of tuple yang
#       jumlah mid-point yang dihasilkan adalah 2^(iterasi) + 1 (sudah termasuk titik kontrol awal dan akhir)
def BF_Generalized_Bezier(Titik_Kontrol : list, iterasi : int, animasikan : bool, pause : float) -> list:
    hasil = []
    n = 2 ** iterasi - 1 # banyak titik yang dihasilkan, belum termasuk titik kontrol awal dan akhir
    dt = 1 / (n + 1)

    for i in range(n + 2):
        t = i * dt
        x = 0
        y = 0
        for j in range(len(Titik_Kontrol)):
            x += (math.comb(len(Titik_Kontrol) - 1, j)) * ((1 - t) ** (len(Titik_Kontrol) - 1 - j)) * (t ** j) * (Titik_Kontrol[j][0])
            y += (math.comb(len(Titik_Kontrol) - 1, j)) * ((1 - t) ** (len(Titik_Kontrol) - 1 - j)) * (t ** j) * (Titik_Kontrol[j][1])

        if animasikan:
            Animation.animate_with_pause([(x, y)], 1, pause)

        hasil.append((x, y))

    return hasil
```

Gambar 5. Implementasi Pembuatan Kurva Bézier N Titik (N ≥ 1) dalam Bahasa Python dengan Algoritma *Brute Force*

3.2 Implementasi dalam Algoritma *Divide and Conquer*

1. Kurva Bézier Kuadratik

```python
# Fungsi yang menghasilkan kurva bezier kuadratik menggunakan metode mid-point (divide and conquer)
# Fungsi ini mengembalikan list of tuple yang berisi koordinat titik-titik yang membentuk kurva bezier
# Fungsi ini:
#   - menerima 5 parameter, yaitu
#       list of titik kontrol
#       parameter iterasi yang menentukan tingkat kedetailan kurva bezier
#       parameter warna yang menentukan warna titik tiap iterasi
#       parameter animasikan yang menentukan apakah tiap iterasi akan di animasikan atau tidak
#       parameter pause yang menentukan lama jeda antar animasi
#   - mengembalikan list of tuple yang
#       jumlah mid-point yang dihasilkan adalah 2^(iterasi) - 1 (belum termasuk titik kontrol awal dan akhir)
def DnC_Quadratik_Bezier(Titik_Kontrol : list, iterasi : int, warna : int, animasikan : bool, pause : float) -> list:
    if iterasi == 0:
        return []
    elif iterasi == 1:                  # Basis
        q0 = midpoint(Titik_Kontrol[0], Titik_Kontrol[1])
        q1 = midpoint(Titik_Kontrol[1], Titik_Kontrol[2])
        b = midpoint(q0, q1)
        if animasikan:
            Animation.animate_with_pause([q0,q1], warna+1, pause)
            Animation.animate_with_pause([b], warna+2, pause)
        return [b]
    else :                             # Rekurens
        q0 = midpoint(Titik_Kontrol[0], Titik_Kontrol[1])
        q1 = midpoint(Titik_Kontrol[1], Titik_Kontrol[2])
        b = midpoint(q0, q1)
        if animasikan:
            Animation.animate_with_pause([q0,q1], warna+1, pause)
            Animation.animate_with_pause([b], warna+2, pause)
    # Rekursi
        Kiri = DnC_Quadratik_Bezier([Titik_Kontrol[0], q0, b], iterasi - 1, warna, animasikan, pause)
        Kanan = DnC_Quadratik_Bezier([b, q1, Titik_Kontrol[2]], iterasi - 1, warna, animasikan, pause)
        return Kiri + [b] + Kanan
```

Gambar 6. Implementasi Pembuatan Kurva Bézier Kuadratik dalam Bahasa Python dengan Algoritma *Divide and Conquer*

2. Kurva Bézier N-Titik (N ≥ 1)

```python
# Fungsi yang menghasilkan generalisasi kurva bezier menggunakan metode mid-point (divide and conquer) (banyak titik kontrol >= 1)
# Fungsi ini mengembalikan list of tuple yang berisi koordinat titik-titik yang membentuk kurva bezier
# Fungsi ini:
#   - menerima 5 parameter, yaitu
#       list of titik kontrol
#       parameter iterasi yang menentukan tingkat kedetailan kurva bezier
#       parameter warna yang menentukan warna titik tiap iterasi
#       parameter animasikan yang menentukan apakah tiap iterasi akan di animasikan atau tidak
#       parameter pause yang menentukan lama jeda antar animasi
#   - mengembalikan list of tuple yang
#       jumlah mid-point yang dihasilkan adalah 2^(iterasi) - 1 (belum termasuk titik kontrol awal dan akhir)
def DnC_Generalized_Bezier(titik_kontrol : list, iterasi : int, warna : int, animasikan : bool, pause : float) -> list:
    if iterasi == 0:
        return []
    elif iterasi == 1:                  # Basis
        return Bezier_Point(titik_kontrol, warna, animasikan, pause)
    else :                             # Rekurens
        Kiri = Control_Point_Left(titik_kontrol)    # menghasilkan titik kontrol baru untuk iterasi selanjutnya (bagian kiri)
        Kiri.insert(0, titik_kontrol[0])            # menambahkan titik kontrol pertama untuk iterasi selanjutnya

        Kanan = Control_Point_Right(titik_kontrol)  # menghasilkan titik kontrol baru untuk iterasi selanjutnya (bagian kanan)
        Kanan.append(titik_kontrol[-1])             # menambahkan titik kontrol terakhir untuk iterasi selanjutnya

        current = Bezier_Point(titik_kontrol, warna, animasikan, pause)    # menghasilkan titik bezier pada iterasi sekarang

        return ( DnC_Generalized_Bezier(Kiri, iterasi - 1, warna, animasikan, pause)
                + current + DnC_Generalized_Bezier(Kanan, iterasi - 1, warna, animasikan, pause) )
```

Gambar 7. Implementasi Pembuatan Kurva Bézier N Titik (N ≥ 1) dalam Bahasa Python dengan Algoritma *Divide and Conquer* (1)

```python
# fungsi yang mengembalikan titik tengah antara dua titik
def midpoint(p1, p2) -> tuple:
    return ((p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2)

# fungsi yang mengembalikan list of midpoint diantara dua titik pada sebuah list of titik
def list_of_midpoint(p : list) -> list:
    hasil = []
    for i in range(len(p) - 1):
        hasil.append(midpoint(p[i], p[i + 1]))
    return hasil

# fungsi yang mengembalikan titik kontrol baru pada satu waktu iterasi
def Control_Point_Left(p : list) -> list:
    hasil = []
    while len(p) > 1:
        p = list_of_midpoint(p)
        hasil.append(p[0])
    return hasil

def Control_Point_Right(p : list) -> list:
    hasil = []
    while len(p) > 1:
        p = list_of_midpoint(p)
        hasil.insert(0, p[-1])
    return hasil

# fungsi yang mengembalikan titik kurva bezier pada satu waktu iterasi
def Bezier_Point(titik_kontrol : list, warna : int, animasikan: bool, pause : float) -> list:
    if len(titik_kontrol) == 1:
        return titik_kontrol
    else:
        middle_point = list_of_midpoint(titik_kontrol)
        if animasikan:
            Animation.animate_with_pause(middle_point, (warna + 1) % 7, pause)
        return Bezier_Point(middle_point, (warna + 1) % 7, animasikan, pause)
```

Gambar 8. Implementasi Pembuatan Kurva Bézier N Titik (N ≥ 1) dalam Bahasa Python dengan Algoritma *Divide and Conquer* (2)

3.3 Main Program

```python
src > main.py > ...
# Program Utama
import matplotlib.pyplot as plt
import time
import Animation
import Brute_Force_Bezier
import Divide_and_Conquer_Bezier
import Input
import Output

######################################################
# Menampilkan menu input dan meminta input dari user #
######################################################
Input.print_pembuka()
metode = Input.input_method()
jenis = Input.input_jenis(metode)
iterasi = Input.input_iterasi()
titik_kontrol = Input.input_titik_kontrol(jenis)
animasikan = Input.input_animate()
pause = Input.input_pause(animasikan)

# Memplot seluruh titik kontrol
Animation.animate_with_pause(titik_kontrol, 0, pause)

########################################################################
# Melakukan perhitungan kurva bezier sesuai dengan metode yang dipilih #
########################################################################
if metode == 1: # brute force
    # mencari titik-titik kurva bezier dan memplotnya
    start = time.time()
    if jenis == 1:
        kurva_bezier = Brute_Force_Bezier.BF_Quadratik_Bezier(titik_kontrol, iterasi, animasikan, pause)
        # hubungkan tiap titik bezier dengan garis
        Animation.animate_without_pause(kurva_bezier, 1)
        plt.pause(2)
        warna = 1
    elif jenis == 2:        # generalized
        kurva_bezier = Brute_Force_Bezier.BF_Generalized_Bezier(titik_kontrol, iterasi, animasikan, pause)
        # hubungkan tiap titik bezier dengan garis
        Animation.animate_without_pause(kurva_bezier, 1)
        plt.pause(2)
        warna = 1
    elif jenis == 3:        # animasikan dengan algoritma de casteljau
        kurva_bezier = Brute_Force_Bezier.BF_De_Casteljaus_algorithm(titik_kontrol, iterasi, animasikan, pause)
        warna = (len(titik_kontrol) % 7 - 1)
        if warna == 0:
            warna = 1
    end = time.time()

    # hapus semua plot, perbarui plot dengan kurva_bezier akhir (hanya garis tanpa titik)
    plt.clf()
    Animation.animate_without_pause(titik_kontrol, 0)
    Animation.animate_just_line(kurva_bezier, warna)
```

Gambar 9. *Main Program* (1)

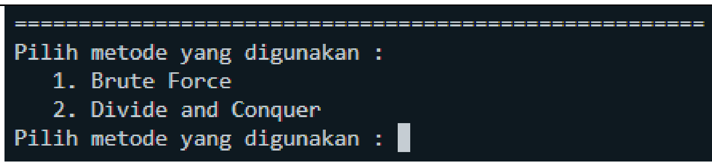
```python
else:           # divide and conquer
    # mencari titik-titik kurva bezier dan memplotnya
    start = time.time()
    if jenis == 1:          # kuadratik
        kurva_bezier = Divide_and_Conquer_Bezier.DnC_Quadratik_Bezier(titik_kontrol, iterasi, 0, animasikan, pause)
    elif jenis == 2:                 # generalized
        kurva_bezier = Divide_and_Conquer_Bezier.DnC_Generalized_Bezier(titik_kontrol, iterasi, 0, animasikan, pause)
    end = time.time()

    # tambahkan titik kontrol awal dan akhir ke dalam kurva bezier
    kurva_bezier.insert(0, titik_kontrol[0])
    kurva_bezier.append(titik_kontrol[-1])

    warna = (len(titik_kontrol) % 7 - 1)
    if warna == 0:          # membedakan warna titik bezier dengan titik kontrol
        warna = 1

    # animasikan kurva_bezier akhir dengan hanya titik kurva bezier yang diplot
    plt.pause(2)
    plt.clf()
    Animation.animate_without_pause(titik_kontrol, 0)
    Animation.animate_without_pause(kurva_bezier, warna)
    plt.pause(2)

    # hapus semua plot, perbarui plot dengan kurva_bezier akhir (hanya garis tanpa titik)
    plt.clf()
    Animation.animate_without_pause(titik_kontrol, 0)
    Animation.animate_just_line(kurva_bezier, warna)

plt.show()

# Menampilkan titik-titik kurva bezier pada terminal
is_print = Input.is_print_points_to_terminal()
if is_print:
    Output.print_points(kurva_bezier)

# Menampilkan waktu eksekusi
# Mulai dari saat pertama kali melakukan pencarian titik bezier pertama hingga titik bezier terakhir
# hanya menghitung waktu yang dibutuhkan untuk untuk menghasilkan list_of_titik_bezier
Input.print_batas()
print("Waktu eksekusi:", end - start, "detik")

#####################################
# Melakukan save hasil kurva bezier #
#####################################
is_save_png = Input.ingin_simpan_to_png()
if is_save_png:
    nama_file = input("Masukkan nama file (tanpa ekstensi): ")
    # memplot ulang untuk disimpan ke dalam file
    plt.clf()
    Animation.animate_without_pause(titik_kontrol, 0)
    Animation.animate_just_line(kurva_bezier, warna)
    # menyimpan plot ke dalam file
    Output.save_plot_to_png(nama_file + " Plot")                        # menyimpan hasil plot ke dalam file
    Output.save_points_to_txt(titik_kontrol, nama_file + " Titik Kontrol")   # menyimpan titik kontrol ke dalam file

is_save_txt = Input.ingin_simpan_to_txt()
if is_save_txt:
    nama_file = input("Masukkan nama file (tanpa ekstensi): ")
    Output.save_points_to_txt(kurva_bezier, nama_file + " Titik Bezier")     # menyimpan titik bezier ke dalam file

Input.print_batas()
```

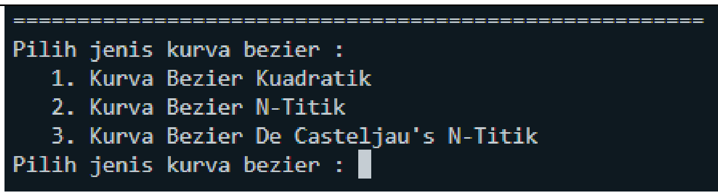
Gambar 10. *Main Program* (2)

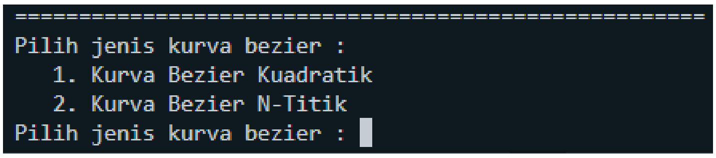
BAB IV
UJI COBA

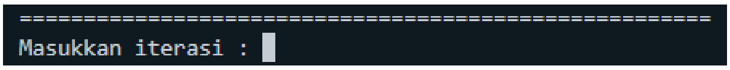
4.1 Keterangan Proses Input dan Output

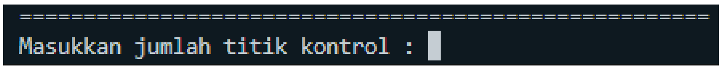
1. Input

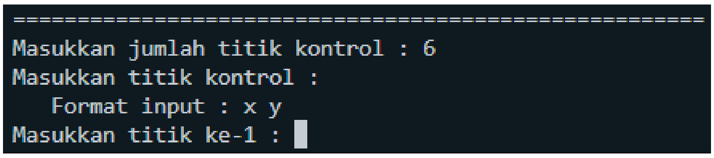
<table>
<tr><th>Input</th><th>Gambar</th></tr>
<tr><td>Meminta pengguna memasukkan metode pembentukan kurva Bézier.
Masukan: 1, 2.</td><td>

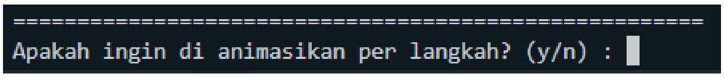
Gambar 11. Pilihan Metode</td></tr>
<tr><td>Meminta pengguna memasukkan jenis kurva Bézier yang akan dibuat. Bergantung pada pilihan sebelumnya.
Masukan: 1, 2, 3 (jika pilihan sebelumnya 1). 1, 2 (jika pilihan sebelumnya 2).</td><td>

Gambar 12. Jenis Kurva Bézier (Pilihan *Brute Force*)


Gambar 13. Jenis Kurva Bézier (Pilihan *Divide and Conquer*)</td></tr>
<tr><td>Meminta pengguna memasukkan jumlah iterasi yang diinginkan.
Masukan: int ≥ 0.</td><td>

Gambar 14. Input Jumlah Iterasi</td></tr>
<tr><td>Meminta pengguna memasukkan jumlah titik kontrol yang diinginkan. Akan dilewatkan jika seandainya memilih Kurva Bézier Kuadratik, karena jumlah titik pasti 3.
Masukan: int ≥ 1.</td><td>

Gambar 15. Input Jumlah Titik Kontrol</td></tr>
<tr><td>Meminta pengguna memasukkan koordinat titik kontrol.
Format: x y
Contoh: 12 34
Contoh: 12.5 34.7
Masukan: float float.</td><td>

Gambar 16. Input Koordinat Titik Kontrol</td></tr>
<tr><td>Meminta pengguna memasukkan pilihan untuk menganimasikan langkah per langkah pembentukan kurva.
Masukan: y/Y, n/N.</td><td>

Gambar 17. Input Pilihan Penganimasian</td></tr>
</table>

Meminta pengguna memasukkan jeda antar animasi/*frame*. Akan dilewatkan jika sebelumnya memilih n/N.
Masukan: float > 0.
	`==========================================================`
	`Masukkan lama jeda animasi (dalam detik) :`
	Gambar 18. Input Jeda Antar-*Frame*

Menanyakan pengguna apakah titik-titik koordinat kurva Bézier ingin ditampilkan di terminal.
Masukan: y/Y, n/N.
	`==========================================================`
	`Apakah ingin mencetak titik bezier ke layar? (y/n) :`
	Gambar 19. Input Ingin Tampilkan Titik Kurva Bézier ke Layar

Menanyakan pengguna apakah hasil visualisasi kurva Bézier ingin disimpan dalam file eksternal.
Masukan: y/Y, n/N.
	`==========================================================`
	`Apakah ingin menyimpan hasil plot ke dalam file? (y/n) :`
	Gambar 20. Input Ingin Menyimpan Hasil Plot ke File Eksternal

Menanyakan pengguna apakah titik-titik koordinat kurva Bézier ingin disimpan dalam file eksternal.
Masukan: y/Y, n/N.
	`==========================================================`
	`Apakah ingin menyimpan titik bezier ke dalam file? (y/n) :`
	Gambar 21. Input Ingin Menyimpan Titik Kurva Bézier ke File Eksternal

Tabel 1. Keterangan Input

2. Output

Output	Gambar

Daftar titik kurva Bézier yang dihasilkan (contoh dengan jumlah iterasi 3, jumlah titik kontrol 3).	`(1.0, 2.0)`
	`(1.421875, 2.40625)`
	`(1.6875, 2.625)`
	`(1.796875, 2.65625)`
	`(1.75, 2.5)`
	`(1.546875, 2.15625)`
	`(1.1875, 1.625)`
	`(0.671875, 0.90625)`
	`(0.0, 0.0)`
	Gambar 22. Keluaran Daftar Titik Kurva Bézier

Waktu yang dibutuhkan untuk mencari semua titik kurva Bézier (contoh dengan jumlah iterasi 3, jumlah titik kontrol 3).	`==========================================================`
	`Waktu eksekusi: 0.0 detik`
	Gambar 23. Keluaran Lama Waktu Eksekusi

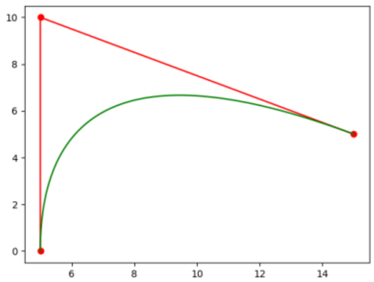
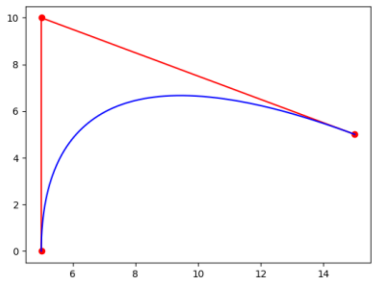
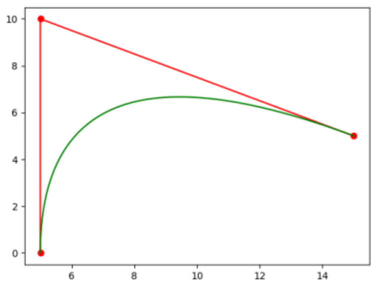
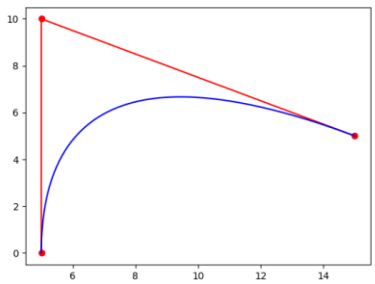
Meminta pengguna memasukkan nama file penyimpanan hasil kurva Bézier. Akan dilewatkan jika sebelumnya memilih n/N.	`Masukkan nama file (tanpa ekstensi): Contoh`
	`Plot berhasil disimpan ke dalam file Contoh Plot.png`
	`Koordinat titik berhasil disimpan ke dalam file Contoh Titik Kontrol.txt`
	Gambar 24. Keluaran Setelah Menyimpan Hasil Plot ke File Eksternal

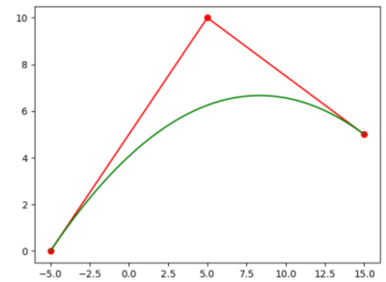
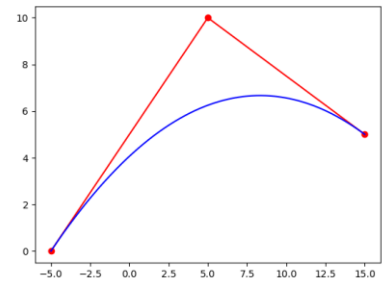
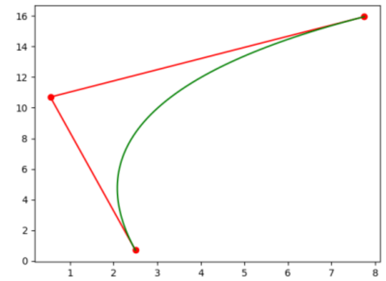
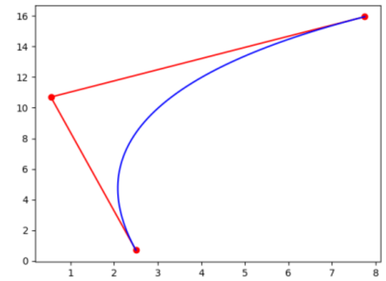
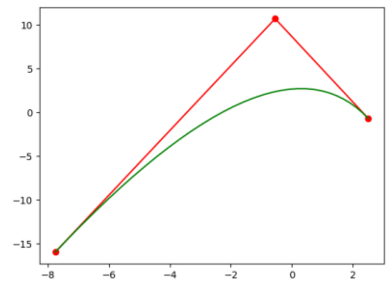
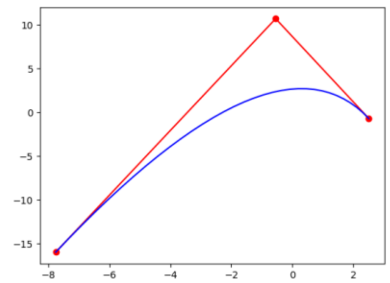
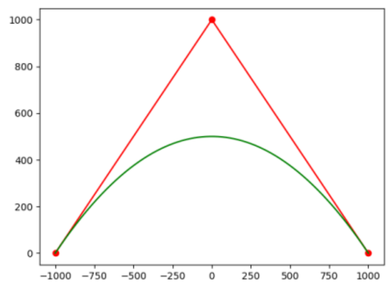
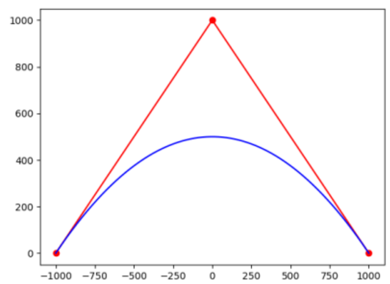
Meminta pengguna memasukkan nama file penyimpanan hasil kurva Bézier. Akan dilewatkan jika sebelumnya memilih n/N.	`Masukkan nama file (tanpa ekstensi): Contoh`
	`Koordinat titik berhasil disimpan ke dalam file Contoh Titik Bezier.txt`
	Gambar 25. Keluaran Setelah Menyimpan Titik-Titik Kurva Bézier ke File Eksternal

Lokasi penyimpanan berada pada folder test. Keterangan: - <nama file> Plot.png : File hasil visualisasi kurva Bézier. - <nama file> Titik Bézier.txt : File tempat menyimpan koordinat titik kurva Bézier yang dihasilkan. - <nama file> Titik Kontrol: File tempat menyimpan titik kontrol yang digunakan.	 **Gambar 26.** Lokasi Penyimpanan dan File Hasil Penyimpanan

Tabel 2. Keterangan Output

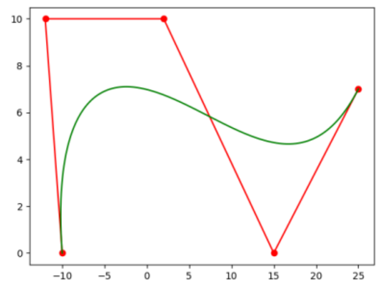
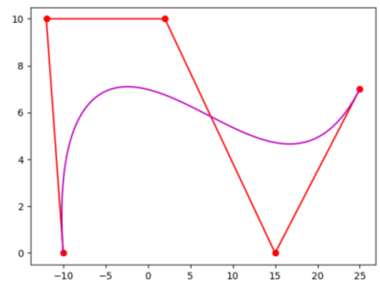
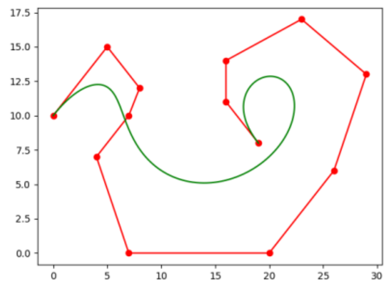
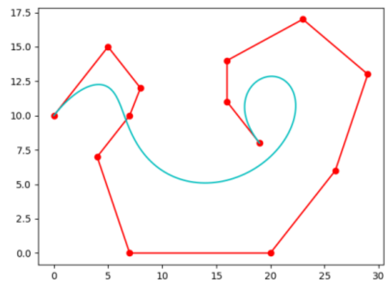
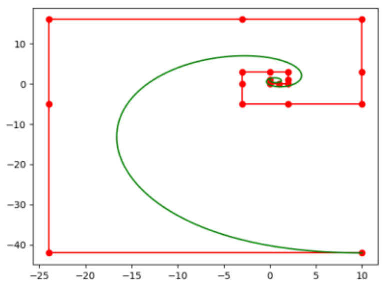
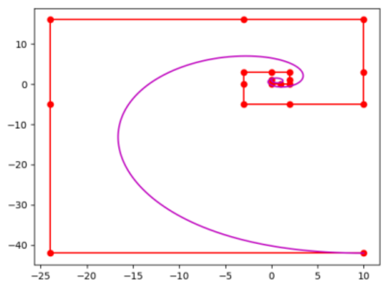
4.2 Kurva Bézier Kuadratik

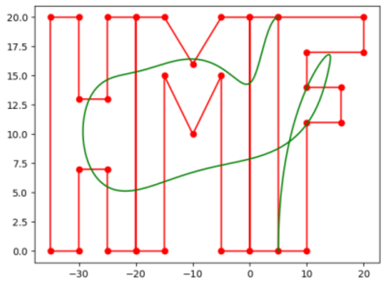
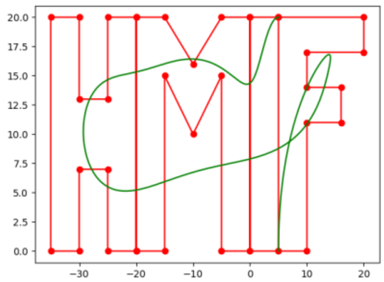
Keterangan Uji Coba	Hasil *Brute Force*	Hasil *Divide and Conquer*
Titik: 3 5 0 5 10 15 5 Iterasi: 20	 **Gambar 27.** Hasil Kurva Bézier Waktu eksekusi: 14.269880056381226 detik **Gambar 28.** Waktu Eksekusi	 **Gambar 29.** Hasil Kurva Bézier Waktu eksekusi: 2.100893497467041 detik **Gambar 30.** Waktu Eksekusi
Titik: 3 5 0 5 10 15 5 Iterasi: 15	 **Gambar 31.** Hasil Kurva Bézier Waktu eksekusi: 2.3251261711120605 detik **Gambar 32.** Waktu Eksekusi	 **Gambar 33.** Hasil Kurva Bézier Waktu eksekusi: 0.06463336944580078 detik **Gambar 34.** Waktu Eksekusi

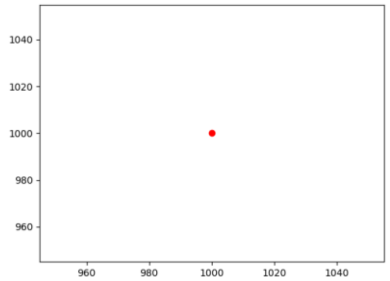
Titik: 3 -5 0 5 10 15 5 Iterasi: 15	 **Gambar 35.** Hasil Kurva Bézier `Waktu eksekusi: 2.311784267425537 detik` **Gambar 36.** Waktu Eksekusi	 **Gambar 37.** Hasil Kurva Bézier `Waktu eksekusi: 0.05699491500854492 detik` **Gambar 38.** Waktu Eksekusi
Titik: 3 2.5 0.7 0.55 10.7 7.75 15.95 Iterasi: 10	 **Gambar 39.** Hasil Kurva Bézier `Waktu eksekusi: 2.0364997386932373 detik` **Gambar 40.** Waktu Eksekusi	 **Gambar 41.** Hasil Kurva Bézier `Waktu eksekusi: 0.001992940902709961 detik` **Gambar 42.** Waktu Eksekusi
Titik: 3 2.5 -0.7 -0.55 10.7 -7.75 -15.95 Iterasi: 5	 **Gambar 43.** Hasil Kurva Bézier `Waktu eksekusi: 2.009791135787964 detik` **Gambar 44.** Waktu Eksekusi	 **Gambar 45.** Hasil Kurva Bézier `Waktu eksekusi: 0.0 detik` **Gambar 46.** Waktu Eksekusi
Titik: 3 -1000 0 0 1000 1000 0 Iterasi: 20	 **Gambar 47.** Hasil Kurva Bézier `Waktu eksekusi: 6.514337778091431 detik` **Gambar 48.** Waktu Eksekusi	 **Gambar 49.** Hasil Kurva Bézier `Waktu eksekusi: 0.9383761882781982 detik` **Gambar 50.** Waktu Eksekusi

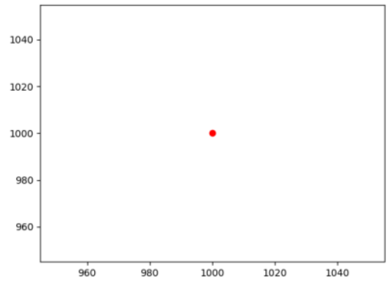
Tabel 3. Perbandingan Hasil Pembentukan Kurva Bézier Kuadratik Menggunakan Metode *Brute Force* dengan *Divide and Conquer*

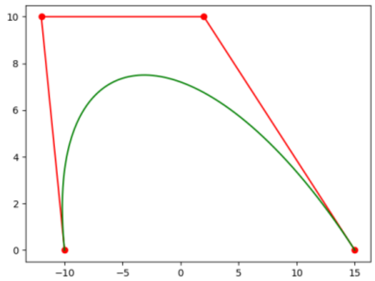
4.3 Kurva Bézier N Titik

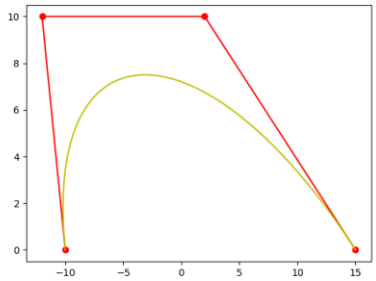
Keterangan Uji Coba	Hasil *Brute Force*	Hasil *Divide and Conquer*
Titik: 5 -10 0 -12 10 2 10 15 0 25 7 Iterasi: 22 * Koordinat Kurva Bezier untuk tes ini tidak disimpan dikarenakan ukuran file besar (> 100 KB)	 **Gambar 51.** Hasil Kurva Bézier `Waktu eksekusi: 85.24080872535706 detik` **Gambar 52.** Waktu Eksekusi	 **Gambar 53.** Hasil Kurva Bézier `Waktu eksekusi: 60.80336284637451 detik` **Gambar 54.** Waktu Eksekusi
Titik: 13 0 10 5 15 8 12 7 10 4 7 7 0 20 0 26 6 29 13 23 17 16 14 16 11 19 8 Iterasi: 20	 **Gambar 55.** Hasil Kurva Bézier `Waktu eksekusi: 37.43517994880676 detik` **Gambar 56.** Waktu Eksekusi	 **Gambar 57.** Hasil Kurva Bézier `Waktu eksekusi: 92.4105486869812 detik` **Gambar 58.** Waktu Eksekusi
Titik: 19 0 1 0 0 1 0 2 0 2 1 2 3 0 3 -3 3 -3 0 -3 -5 2 -5 10 -5 10 3 10 16	 **Gambar 59.** Hasil Kurva Bézier `Waktu eksekusi: 7.044555425643921 detik` **Gambar 60.** Waktu Eksekusi	 **Gambar 61.** Hasil Kurva Bézier `Waktu eksekusi: 24.562450885772705 detik` **Gambar 62.** Waktu Eksekusi

-3 16 -24 16 -24 -5 -24 -42 10 -42 Iterasi: 17		
Titik: 36 5 0 5 20 20 20 20 17 10 17 10 14 16 14 16 11 10 11 10 0 -5 0 -5 15 -10 10 -15 15 -15 0 -20 0 -25 0 -25 7 -30 7 -30 0 -35 0 -35 20 -30 20 -30 13 -25 13 -25 20 -20 20 -20 0 -20 20 -15 20 -10 16 -5 20 0 20 0 0 0 20 5 20 Iterasi: 20	 **Gambar 63.** Hasil Kurva Bézier Waktu eksekusi: 33.81191897392273 detik **Gambar 64.** Waktu Eksekusi	 **Gambar 65.** Hasil Kurva Bézier Waktu eksekusi: 318.5977973937988 detik **Gambar 66.** Waktu Eksekusi

<table>
<tr>
<td>Titik: 1
1000 1000
Iterasi: 20</td>
<td>

Gambar 67. Hasil Kurva Bézier
Waktu eksekusi: 14.57087779045105 detik
Gambar 68. Waktu Eksekusi</td>
<td>

Gambar 69. Hasil Kurva Bézier
Waktu eksekusi: 0.759089469909668 detik
Gambar 70. Waktu Eksekusi</td>
</tr>
<tr>
<td>Titik: 4
-10 0
-12 10
2 10
15 0
Iterasi: 20</td>
<td>

Gambar 71. Hasil Kurva Bézier
Waktu eksekusi: 19.837205410003662 detik
Gambar 72. Waktu Eksekusi</td>
<td>

Gambar 73. Hasil Kurva Bézier
Waktu eksekusi: 10.382862567901611 detik
Gambar 74. Waktu Eksekusi</td>
</tr>
</table>

Tabel 4. Perbandingan Hasil Pembentukan Kurva Bézier N Titik Menggunakan Metode *Brute Force* dengan *Divide and Conquer*

BAB V
ANALISIS SOLUSI

5.1 Perbandingan Solusi Algoritma Pembentuk Kurva Bézier Kuadratik

1. Kompleksitas Algoritma Pembentuk Kurva Bézier Kuadratik

a. Metode *Brute Force*

```python
# Fungsi yang menghasilkan kurva bezier kuadratik menggunakan metode brute force
# Fungsi ini mengembalikan list of tuple yang berisi koordinat titik-titik yang membentuk kurva bezier
# Fungsi ini:
#   - menerima 4 parameter, yaitu
#       List of titik kontrol
#       parameter iterasi yang menentukan tingkat kedetailan kurva bezier
#       parameter animasikan yang menentukan apakah tiap iterasi akan di animasikan atau tidak
#       parameter pause yang menentukan lama jeda antar animasi
#   - mengembalikan list of tuple dengan
#       jumlah titik yang dihasilkan adalah (2^(iterasi) + 1) (sudah termasuk titik kontrol awal dan akhir)
def BF_Quadratik_Bezier(Titik_Kontrol : list, iterasi : int, animasikan : bool, pause : float) -> list:
    # Inisialisasi list of tuple yang akan menampung koordinat titik-titik kurva bezier
    hasil = []
    # Hitung jumlah titik yang dihasilkan, belum termasuk titik kontrol awal dan akhir
    n = 2 ** iterasi - 1
    # hitung interval nilai t
    dt = 1 / (n + 1)
    # Hitung koordinat tiap titik kurva bezier
    for i in range(n + 2): # ditambah 2 untuk untuk mengikutsertakan titik kontrol awal dan akhir
        t = i * dt
        x = (1 - t) ** 2 * Titik_Kontrol[0][0] + 2 * (1 - t) * t * Titik_Kontrol[1][0] + t ** 2 * Titik_Kontrol[2][0]
        y = (1 - t) ** 2 * Titik_Kontrol[0][1] + 2 * (1 - t) * t * Titik_Kontrol[1][1] + t ** 2 * Titik_Kontrol[2][1]

        if animasikan:
            Animation.animate_with_pause([(x, y)], 1, pause)

        hasil.append((x, y))

    return hasil
```

Gambar 75. Algoritma Pembuatan Kurva Bézier Kuadratik dengan Pendekatan *Brute Force*

Perhatikan potongan kode di atas. Untuk menghitung nilai kompleksitas waktu $T(n)$ dari algoritma pembuatan kurva Bézier kuadratik di atas, maka yang menjadi fokus adalah pada bagian kalang/*looping*. Dengan menggunakan banyaknya titik yang ingin dihasilkan sebagai n, maka:

- Terdapat n kali perulangan yang dilakukan (dalam kode $n + 2$ karena n masih belum mengikutsertakan titik kontrol awal dan akhir).
- Untuk setiap perulangan, terdapat 8 kali operasi penjumlahan (termasuk pengurangan), 11 kali operasi perkalian, dan 4 kali operasi perpangkatan 2.

Sehingga, untuk menghasilkan n buah titik, $T(n)$ akan sama dengan:

$$T(n) = n(8 \text{ kali operasi penjumlahan}) + n(11 \text{ kali operasi perkalian}) + n(4 \text{ kali operasi perpangkatan } 2)$$

Operasi perpangkatan 2 sama dengan 1 kali operasi perkalian, sehingga

$$T(n) = 8n + 11n + 4n$$

$$T(n) = 23n$$

Diperoleh $T(n) = 23n$, sehingga kompleksitas algoritmanya adalah:

$$O(n) = n$$

b. Metode Divide and Conquer

```python
# Fungsi yang menghasilkan kurva bezier kuadratik menggunakan metode mid-point (divide and conquer)
# Fungsi ini mengembalikan list of tuple yang berisi koordinat titik-titik yang membentuk kurva bezier
# Fungsi ini:
#   - menerima 5 parameter, yaitu
#       List of titik kontrol
#       parameter iterasi yang menentukan tingkat kedetailan kurva bezier
#       parameter warna yang menentukan warna titik tiap iterasi
#       parameter animasikan yang menentukan apakah tiap iterasi akan di animasikan atau tidak
#       parameter pause yang menentukan lama jeda antar animasi
#   - mengembalikan list of tuple yang
#       jumlah mid-point yang dihasilkan adalah 2^(iterasi) - 1 (belum termasuk titik kontrol awal dan akhir)
def DnC_Quadratik_Bezier(Titik_Kontrol : list, iterasi : int, warna : int, animasikan : bool, pause : float) -> list:
    if iterasi == 0:
        return []
    elif iterasi == 1:                # Basis
        q0 = midpoint(Titik_Kontrol[0], Titik_Kontrol[1])
        q1 = midpoint(Titik_Kontrol[1], Titik_Kontrol[2])
        b = midpoint(q0, q1)
        if animasikan:
            Animation.animate_with_pause([q0,q1], warna+1, pause)
            Animation.animate_with_pause([b], warna+2, pause)
        return [b]
    else :                           # Rekurens
        q0 = midpoint(Titik_Kontrol[0], Titik_Kontrol[1])
        q1 = midpoint(Titik_Kontrol[1], Titik_Kontrol[2])
        b = midpoint(q0, q1)
        if animasikan:
            Animation.animate_with_pause([q0,q1], warna+1, pause)
            Animation.animate_with_pause([b], warna+2, pause)
    # Rekursi
        Kiri = DnC_Quadratik_Bezier([Titik_Kontrol[0], q0, b], iterasi - 1, warna, animasikan, pause)
        Kanan = DnC_Quadratik_Bezier([b, q1, Titik_Kontrol[2]], iterasi - 1, warna, animasikan, pause)
        return Kiri + [b] + Kanan
```

Gambar 76. Algoritma Pembuatan Kurva Bézier Kuadratik dengan Pendekatan *Divide and Conquer*

```python
# fungsi yang mengembalikan titik tengah antara dua titik
def midpoint(p1, p2) -> tuple:
    return ((p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2)
```

Gambar 77. Implementasi Fungsi midpoint

Perhatikan potongan kode di atas. Untuk menghitung nilai kompleksitas waktu $T(n)$ dari algoritma pembuatan kurva Bézier kuadratik di atas, maka yang menjadi fokus adalah operasi penjumlahan (dan pengurangan) dan perkalian (dan pembagian). Untuk menghasilkan jumlah titik yang sama dengan metode *brute force,* maka cukup dengan melakukan iterasi sebanyak i kali, dengan banyaknya titik (n) sama dengan ($2^i + 1$).

Untuk melihat nilai $T(n)$ algoritma pembuatan kurva Bézier kuadratik, maka terlebih dahulu harus dicari $T(n)$ untuk fungsi midpoint. Dari potongan kode di atas, diperoleh nilai $T_{\text{midpoint}}(n) = 4$.

Untuk iterasi sebanyak i kali, atau dalam hal ini i menyatakan tingkat kedalaman rekursif yang dilakukan, akan dihasilkan banyak titik sebanyak (2^i - 1), karena untuk titik kontrol awal dan akhir tidak masuk dalam perhitungan fungsi di atas. Untuk mendapatkan 1 titik, akan dilakukan sebanyak 3 pemanggilan fungsi midpoint, sehingga 1 titik akan memerlukan 3×4 = 12 kali operasi. Oleh karena itu, untuk menghasilkan n buah titik, diperlukan n×12 operasi, sehingga kompleksitas waktu fungsi di atas adalah:

$$T(n) = 12n$$

Nilai $T(n)$ lebih baik dari pada nilai $T(n)$ untuk pendekatan *brute force*. Adapun untuk kompleksitas waktunya adalah:

$O(n) = n$, yang mana sama dengan kompleksitas waktu pendekatan *brute force*.

2. Perbandingan Hasil Perhitungan dengan Hasil Uji Coba

Berdasarkan hasil perhitungan kompleksitas waktu kedua algoritma, dapat terlihat bahwa kedua algoritma sama-sama memiliki notasi Big-O yang sama, yaitu $O(n) = n$. Akan tetapi jika kita memperhatikan nilai $T(n)$, terlihat bahwa pendekatan *divide and conquer* lebih baik daripada pendekatan *brute force*, sehingga dapat dikatakan bahwa untuk n yang sama, yaitu saat diinginkan jumlah titik yang dihasilkan sama, pendekatan *divide and conquer* akan lebih cepat dari pada pendekatan *brute force*. Hal inilah yang menyebabkan mengapa untuk setiap uji coba yang dilakukan pada subbab 4.2, pendekatan *divide and conquer* selalu memakan waktu eksekusi yang lebih cepat dari pada pendekatan *brute force*.

5.2 Perbandingan Solusi Algoritma Pembentuk Kurva Bézier N Titik

1. Kompleksitas Algoritma Pembentuk Kurva Bézier N Titik

a. Metode *Brute Force*

```python
# Fungsi yang menghasilkan generalisasi kurva bezier menggunakan metode brute force (banyak titik kontrol >= 1)
# Fungsi ini mengembalikan list of tuple yang berisi koordinat titik-titik yang membentuk kurva bezier
# Fungsi ini:
#   - menerima 4 parameter, yaitu
#       List of titik kontrol
#       parameter iterasi yang menentukan tingkat kedetailan kurva bezier
#       parameter animasikan yang menentukan apakah tiap iterasi akan di animasikan atau tidak
#       parameter pause yang menentukan lama jeda antar animasi
#   - mengembalikan list of tuple yang
#       jumlah mid-point yang dihasilkan adalah 2^(iterasi) + 1 (sudah termasuk titik kontrol awal dan akhir)
def BF_Generalized_Bezier(Titik_Kontrol : list, iterasi : int, animasikan : bool, pause : float) -> list:
    hasil = []
    n = 2 ** iterasi - 1 # banyak titik yang dihasilkan, belum termasuk titik kontrol awal dan akhir
    dt = 1 / (n + 1)

    for i in range(n + 2): # ditambah 2 untuk untuk mengikutsertakan titik kontrol awal dan akhir
        t = i * dt
        x = 0
        y = 0
        for j in range(len(Titik_Kontrol)):
            x += (math.comb(len(Titik_Kontrol) - 1, j)) * ((1 - t) ** (len(Titik_Kontrol) - 1 - j)) * (t ** j) * (Titik_Kontrol[j][0])
            y += (math.comb(len(Titik_Kontrol) - 1, j)) * ((1 - t) ** (len(Titik_Kontrol) - 1 - j)) * (t ** j) * (Titik_Kontrol[j][1])

        if animasikan:
            Animation.animate_with_pause([(x, y)], 1, pause)

        hasil.append((x, y))

    return hasil
```

Gambar 78. Algoritma Pembuatan Kurva Bézier Kuadratik dengan Pendekatan *Brute Force*

Perhatikan potongan kode di atas. Untuk menghitung nilai kompleksitas waktu fungsi di atas, maka terdapat dua parameter penting yang menentukan, yaitu banyaknya titik yang ingin dihasilkan, misal n, dan banyaknya titik kontrol yang diberikan, misal m. Oleh karena itu, kompleksitas waktu fungsi di atas dinyatakan dalam $T(n,m)$.

Untuk mencari nilai $T(n,m)$, maka yang perlu diperhatikan adalah pada bagian kalang/*looping*. Fokus terlebih dahulu untuk kalang yang berada pada bagian dalam. Dalam hal ini, jumlah perulangan dilakukan sebanyak m kali. Untuk satu perulangan, dilakukan:

- 10 kali operasi penjumlahan (termasuk pengurangan).
- 6 kali operasi perkalian.
- 4 kali operasi perpangkatan.
- 2 kali operasi kombinatorial.

Berdasarkan persamaan untuk kombinatorial, yaitu ${}_{m}C_{k} = m!/((m-k)!k!)$, maka satu operasi kombinatorial sama dengan 2m kali operasi perkalian, sehingga untuk fungsi di atas, 2 kali operasi kombinatorial akan sama dengan 2×2m = 4m operasi perkalian.

Adapun untuk operasi perpangkatan, jika dilihat dari persaman untuk mendapatkan kurva Bézier, yaitu:

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^{i} \mathbf{P}_i , \quad 0 \leq t \leq 1$$

untuk satu kali perulangan, 2 operasi perpangkatan tersebut sama dengan m kali operasi perkalian, dalam fungsi di atas, 4 operasi perpangkatan sama dengan 2m kali operasi perkalian.

Dari penjelasan di atas, dapat dicari kompleksitas waktu untuk kalang bagian dalam, yaitu:

$T(m) = 16m + 6m^2$

Adapun untuk kalang bagian luar, terdapat *n* kali perulangan, yang masing-masing perulangan melakukan $T(m)$ kali operasi, sehingga:

$T(n) = n(T(m))$

$T(n,m) = n(16m + 6m^2)$

$T(n,m) = 6m^2n + 16mn$

Diperoleh $T(n,m) = 6m^2n + 16mn$. Adapun kompleksitas algoritmanya adalah:

$O(n,m) = m^2n$

b. Metode Divide and Conquer

```python
# Fungsi yang menghasilkan generalisasi kurva bezier menggunakan metode mid-point (divide and conquer) (banyak titik kontrol >= 1)
# Fungsi ini mengembalikan list of tuple yang berisi koordinat titik-titik yang membentuk kurva bezier
# Fungsi ini:
#   - menerima 5 parameter, yaitu
#       List of titik kontrol
#       parameter iterasi yang menentukan tingkat kedetailan kurva bezier
#       parameter warna yang menentukan warna titik tiap iterasi
#       parameter animasikan yang menentukan apakah tiap iterasi akan di animasikan atau tidak
#       parameter pause yang menentukan lama jeda antar animasi
#   - mengembalikan list of tuple yang
#       jumlah mid-point yang dihasilkan adalah 2^(iterasi) - 1 (belum termasuk titik kontrol awal dan akhir)
def DnC_Generalized_Bezier(titik_kontrol : list, iterasi : int, warna : int, animasikan : bool, pause : float) -> list:
    if iterasi == 0:
        return []
    elif iterasi == 1:                # Basis
        return Bezier_Point(titik_kontrol, warna, animasikan, pause)
    else :                           # Rekurens
        Kiri = Control_Point_Left(titik_kontrol)    # menghasilkan titik kontrol baru untuk iterasi selanjutnya (bagian kiri)
        Kiri.insert(0, titik_kontrol[0])            # menambahkan titik kontrol pertama untuk iterasi selanjutnya

        Kanan = Control_Point_Right(titik_kontrol)  # menghasilkan titik kontrol baru untuk iterasi selanjutnya (bagian kanan)
        Kanan.append(titik_kontrol[-1])             # menambahkan titik kontrol terakhir untuk iterasi selanjutnya

        current = Bezier_Point(titik_kontrol, warna, animasikan, pause)    # menghasilkan titik bezier pada iterasi sekarang

        return ( DnC_Generalized_Bezier(Kiri, iterasi - 1, warna, animasikan, pause)
                + current + DnC_Generalized_Bezier(Kanan, iterasi - 1, warna, animasikan, pause) )
```

Gambar 79. Algoritma Pembuatan Kurva Bézier Kuadratik dengan Pendekatan *Divide and Conquer*

```python
# fungsi yang mengembalikan titik tengah antara dua titik
def midpoint(p1, p2) -> tuple:
    return ((p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2)

# fungsi yang mengembalikan list of midpoint diantara dua titik pada sebuah list of titik
def list_of_midpoint(p : list) -> list:
    hasil = []
    for i in range(len(p) - 1):
        hasil.append(midpoint(p[i], p[i + 1]))
    return hasil

# fungsi yang mengembalikan titik kontrol baru pada satu waktu iterasi
def Control_Point_Left(p : list) -> list:
    hasil = []
    while len(p) > 1:
        p = list_of_midpoint(p)
        hasil.append(p[0])
    return hasil

def Control_Point_Right(p : list) -> list:
    hasil = []
    while len(p) > 1:
        p = list_of_midpoint(p)
        hasil.insert(0, p[-1])
    return hasil

# fungsi yang mengembalikan titik kurva bezier pada satu waktu iterasi
def Bezier_Point(titik_kontrol : list, warna : int, animasikan: bool, pause : float) -> list:
    if len(titik_kontrol) == 1:
        return titik_kontrol
    else:
        middle_point = list_of_midpoint(titik_kontrol)
        if animasikan:
            Animation.animate_with_pause(middle_point, (warna + 1) % 7, pause)
        return Bezier_Point(middle_point, (warna + 1) % 7, animasikan, pause)
```

Gambar 80. Fungsi Tambahan

Untuk dapat mencari kompleksitas waktu $T(m,n)$ fungsi pada gambar 80, terlebih dahulu harus dicari kompleksitas waktu masing-masing fungsi bantuannya.

Untuk $T(n,m)$ dari fungsi midpoint telah diperoleh pada bagian sebelumnya, pada subbab 5.1 bagian 1 bagian b, yaitu:

$T_{midpoint}(n,m) = 4$.

Untuk fungsi list_of_midpoint, $T(n,m)$ hanya bergantung pada jumlah titik kontrol yang diberikan, yaitu m. Di dalam fungsi tersebut, terdapat m-1 kali perulangan dengan masing-masing perulangan memanggil fungsi midpoint. Oleh karena itu,

$T_{list\_of\_midpoint}(n,m) = 4(m\text{-}1) = 4m - 4$.

Untuk fungsi Control_Point_Left dan Control_Point_Right memiliki algoritma yang mirip, hanya sedikit perbedaan pada indeks list yang diambil. Oleh karena itu, nilai $T(n,m)$ keduanya sama, sehingga cukup dicari salah satu saja. Pada kedua fungsi di atas, parameter input p adalah daftar titik kontrol, oleh karena itu, banyak-nya perulangan yang dilakukan di dalam kedua fungsi di atas adalah sebanyak m. untuk masing-masing perulangan memanggil fungsi list_of_midpoint, sehingga:

$T_{Control\_Point\_Left}(n,m) = T_{Control\_Point\_Right}(n,m) = m(4m\text{-}4) = 4m^2 + 4m$.

Terakhir untuk fungsi Bezier_Point, melakukan proses rekursif. Pada bagian basis, fungsi hanya mengembalikan parameter yang diberikan, sehingga tidak ada operasi (penjumlahan dan perkalian) yang dilakukan. Untuk bagian rekurens, fungsi memanggil fungsi list_of_midpoint dan untuk kemudian hasil kembaliannya dijadikan parameter masukan untuk pemanggilan fungsi Bezier_Point. Pada saat ini, jumlah titik kontrol telah berkurang satu, sehingga $T(n,m)$ dapat dirumuskan sebagai:

$$T(n,m) = \begin{cases} 0 & , m = 1 \\ T(n,m-1) + (4m-4) & , m > 1 \end{cases}$$

Adapun untuk $m > 1$, nilai $T(n,m)$ adalah:

$$\begin{aligned} T(n,m) &= T(n,m-1) + (4m-4) \\ &= T(n,m-2) + (4m-4) + (4m-4) \\ &= T(n,m-3) + (4m-4) + (4m-4)) + (4m-4) \\ &= \ldots \\ &= T(1) + \underbrace{(4m-4) + (4m-4) + \ldots + (4m-4)}_{(m-1)} \end{aligned}$$

$$T(n,m) = (m-1)(4m-4)$$

$$T_{Bezier\_Point}(n,m) = 4m^2 - 8m + 4$$

Sekarang, kita telah dapat menghitung kompleksitas waktu $T(n,m)$ untuk algoritma utama, yaitu algoritma pencari titik-titik kurva Bézier, yang diberikan oleh gambar 79.

Dari gambar terlihat bahwa fungsi menerima parameter iterasi. Karena diinginkan kompleksitas waktu dalam jumlah titik yang dihasilkan agar dapat dibandingkan dengan algoritma pendekatan *brute force*, maka perlu dicari hubungan antara iterasi yang dilakukan dengan banyaknya titik yang dihasilkan. Hubungan antara banyaknya iterasi (i) dengan banyaknya titik yang dihasilkan (n) yaitu $n = 2^i - 1$ (belum mengikutsertakan titik kontrol awal dan akhir).

Implementasi fungsi dilakukan secara rekursif, dengan terdapat dua basis. Pertama saat iterasi = 0, yang artinya titik yang dihasilkan alias n = 0, maka banyak operasi yang dilakukan adalah 0. Untuk basis kedua, iterasi = 1 sehingga n = 1, melakukan pemanggilan fungsi Bezier_Point, sehingga banyaknya operasi yang dilakukan adalah $4m^2 - 8m + 4$. Adapun untuk kasus iterasi > 1 alias n > 1, melakukan pemanggilan fungsi Control_Point_Left, Control_Point_Right, dan Bezier_Point dan kemudian kembali memanggil dirinya sendiri dengan parameter iterasi yang berkurang 1. Adapun $T(n,m) = (4m^2 + 4m) + (4m^2 + 4m) + (4m^2 - 8m + 4) = 12m^2 + 4$. Adapun detailnya adalah sebagai berikut:

$$T(n,m) = \begin{cases} 0 & , n = 0 \\ 4m^2 - 8m + 4 & , n = 1 \\ T(n-1,m) + (12m^2 + 4) & , n > 1 \end{cases}$$

Adapun untuk $n > 1$, nilai $T(n,m)$ adalah:

$$T(n,m) = T(n - 1, m) + (12m^2 + 4)$$
$$= T(n - 2, m) + (12m^2 + 4) + (12m^2 + 4)$$
$$= T(n - 3, m) + (12m^2 + 4) + (12m^2 + 4) + (12m^2 + 4)$$
$$= . . .$$
$$= T(1) + \underbrace{(12m^2 + 4) + (12m^2 + 4) + \ldots + (12m^2 + 4)}_{(n - 1)}$$

$$T(n,m) = (4m^2 – 8m + 4) + (n-1)(12m^2 + 4)$$
$$T (n,m) = 4m^2 – 8m + 4 + 12m^2n + 4n – 12m^2 – 4$$
$$T (n,m) = 12m^2n + 4n – 8m^2 – 8m$$

Diperoleh $T(n,m) = 12m^2n + 4n – 8m^2 – 8m$. Adapun kompleksitas algoritmanya adalah:
$O(n, m) = m^2n$, yang mana sama dengan kompleksitas algoritma pada pendekatan *brute force*.

2. Perbandingan Hasil Perhitungan dengan Hasil Uji Coba

Karena kedua algoritma memiliki notasi Big-O yang sama, maka $O(n,m)$ tidak dapat dijadikan acuan untuk membandingkan kedua algoritma tersebut. Perlu dilihat nilai $T(n,m)$ masing-masing algoritma. Perhatikan kembali kompleksitas waktu kedua algoritma tersebut.

$T(n,m) = 6m^2n + 16mn$, *brute force*
$T(n,m) = 12m^2n + 4n – 8m^2 – 8m$, *divide and conquer*

Dari dua fungsi di atas, suku yang paling mendominasi adalah suku m^2n. namun koefisien pada *brute force* lebih rendah dari pada koefisien pada *divide and conquer*. Hal ini menyebabkan untuk nilai m dan n yang sangat besar, pendekatan *brute force* jauh lebih cepat dibandingkan pendekatan *divide and conquer*. Hal inilah yang menjelaskan mengapa pada contoh uji coba yang dilakukan pada subbab 4.3, pada baris 2, 3, dan 4, yang masing-masing nilai n dan m nya yaitu (2^{20}-1, 13), (2^{17}-1,19), dan (2^{20}-1,36), pendekatan *brute force* memberikan waktu eksekusi yang jauh lebih cepat dibandingkan pendekatan *divide and conquer*.

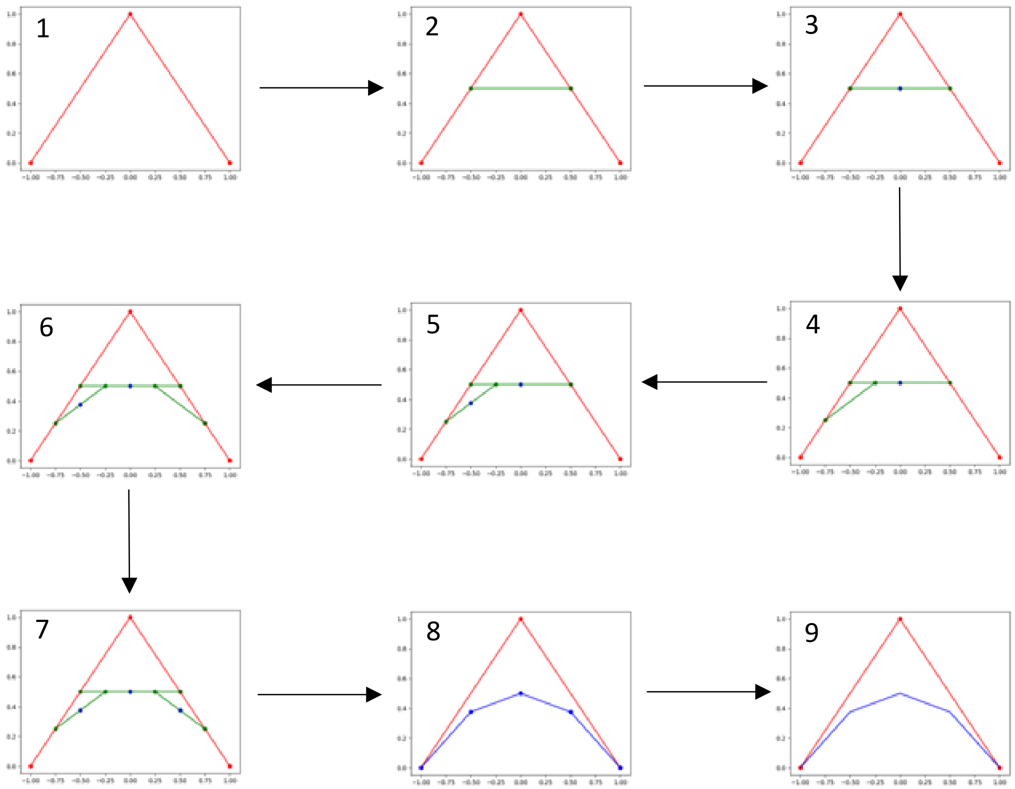
Namun, untuk nilai m yang kecil, suku ($4n + –8m^2 – 8m$) pada kompleksitas waktu *divide and conquer*, bisa memberikan efek yang cukup untuk mengurangi nilai suku $12m^2n$. Sebaliknya, suku $6m^2n$ pada kompleksitas waktu *brute force* justru menambah nilai suku $6m^2n$. Hal ini menyebabkan algoritma *divide and conquer* bisa lebih cepat dari pada *brute force* pada kasus tersebut. Hal inilah yang menjelaskan mengapa pada contoh uji coba yang dilakukan pada subbab 4.3, pada baris 1, 5, dan 6 dengan nilai m masing-masing yaitu 5, 1, dan 4, pendekatan *divide and conquer* memberikan waktu eksekusi yang lebih cepat dibandingkan pendekatan *brute force*.

BAB VI
IMPLEMENTASI BONUS

6.1 Visualisasi Pembangkitan Kurva

Proses visualisasi pembangkitan kurva pada program “Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer” memanfaatkan *library* matplotlib yang tersedia pada bahasa pemrograman python.

Proses visualisasi pembangkitan kurva dilakukan dengan memplot satu-persatu setiap langkah yang dilakukan untuk mendapatkan titik kurva Bézier. Berikut merupakan salah satu ilustrasi proses pembangkita kurva Bézier kuadratik dengan iterasi sebanyak 2 kali.



Gambar 81. Proses Pembentukan Kurva Bézier Kuadratik (Iterasi = 2)

Setiap proses visualisasi per *frame*-nya dapat dilakukan jeda, yang durasi jedanya dapat diatur sendiri dengan memasukkan lama jeda yang diinginkan pada saat proses input saat program pertama kali dijalankan (lihat kembali tabel 1 pada halaman 10). Namun perlu diperhatikan bahwa untuk jeda waktu yang sangat singkat (kurang dari 0,1) tidak akan memberikan efek apapun karena waktu yang diperlukan untuk memplot kurva tersebut akan memakan waktu lebih lama dari waktu jeda yang diberikan. Selain itu, berdasarkan percobaan yang dilakukan, proses animasi akan semakin melambat seiring berjalannya waktu penganimasian (jeda antar *frame* saat akhir animasi lebih lama dari pada jeda antar *frame* pada awal animasi). Hal ini kemungkinan disebabkan oleh keterbatasan dari perangkat ataupun dari bahasa yang digunakan.

Berikut merupakan kode program yang merupakan prosedur untuk melakukan proses visualisasi di atas.

```
# Module for animation
import matplotlib.pyplot as plt
COLOUR = ['r', 'g', 'b', 'y', 'm', 'c', 'k']

# prosedur yang melakukan animasi pembentukan titik kurva bezier pada saat t / waktu iterasi tertentu
def animate_with_pause(list_of_point : list, warna : int, pause : float) -> None:
    plt.scatter([i[0] for i in list_of_point], [i[1] for i in list_of_point], c = COLOUR[warna])
    plt.plot([i[0] for i in list_of_point], [i[1] for i in list_of_point], c = COLOUR[warna])
    plt.pause(pause)

def animate_without_pause(list_of_point : list, warna : int) -> None:
    plt.scatter([i[0] for i in list_of_point], [i[1] for i in list_of_point], c = COLOUR[warna])
    plt.plot([i[0] for i in list_of_point], [i[1] for i in list_of_point], c = COLOUR[warna])

def animate_just_line(list_of_point : list, warna : int) -> None:
    plt.plot([i[0] for i in list_of_point], [i[1] for i in list_of_point], c = COLOUR[warna])
```

Gambar 82. Fungsi-Fungsi untuk Visualisasi Kurva

Terdapat 3 fungsi untuk melakukan visualisasi. Yang pertama adalah fungsi animate_with_pause yang melakukan proses plot titik kurva, yang setiap melakukan plot satu *frame* akan melakukan jeda sesuai dengan parameter "*pause*". Yang kedua adalah fungsi animate_without_pause yang melakukan proses visualisasi kurva yang antar-*frame* tidak memiliki jeda, sehingga dari sudut pandang pengguna, keseluruhan *frame* akan tampak seperti hanya satu *frame*. Kedua fungsi ini akan memplot titik koordinat sekaligus menghubungkan setiap titik yang terbentuk dengan garis. Yang terakhir adalah animate_just_line yang hanya menggambar kurva dengan garis tanpa titik.

Selain itu, terdapat daftar warna yang digunakan, yang disimpan dalam variabel global bernama COLOUR, yang masing-masing isi di dalamnya melambangkan satu warna berbeda ('r' untuk merah/*red*, 'g' untuk hijau/*green*, 'b' untuk biru/*blue*, 'y' untuk kuning/*yellow*, 'm' untuk magenta/*magenta*, 'c' untuk sian/*cyan*, dan 'k' untuk hitam/*black*). Untuk setiap lapisan titik tengah yang diplot akan memiliki warna yang berbeda, dengan bagian terluar (alias titik kontrol) akan berwarna merah (warna pertama pada daftar). Untuk lapisan yang di bawahnya akan mengambil warna selanjutnya pada daftar, dan begitu seterusnya. Jika seandainya lapisan titik tengah lebih banyak daripada warna yang tersedia, maka jika telah sampai pada warna hitam, akan kembali lagi ke warna merah.

Pada akhir proses visualisasi, seluruh titik tengah yang digunakan untuk memperoleh titik kurva Bézier akan dihapus sehingga hanya menyisakan titik kontrol awal dan hasil dari kurva Bézier itu sendiri.

6.2 Generalisasi Algoritma

Generalisasi algoritma dilakukan dengan memanfaatkan prinsip yang sama dengan prinsip untuk menghasilkan kurva Bézier kuadratik, yiatu dengan terus mencari titik tengah antara 2 titik kontrol yang berdekatan sehingga menghasilkan himpunan titik yang baru. Proses ini kemudian diulangi kembali untuk himpunan titik yang baru hingga pada akhirnya hanya dihasilkan satu titik tengah. Satu titik tengah inilah yang akan menjadi titik kurva Bézier.

Untuk iterasi selanjutnya akan memanfaatkan titik kontrol baru yang diperoleh dari titik-titik tengah yang didapat pada proses sebelumnya. Adapun penjelasan lebih lengkap mengenai cara memperoleh kurva Bézier telah dijelaskan pada subbab 2.2 (lihat halaman 2).

Untuk generalisasi algoritma ini, jumlah titik kontrol yang valid adalah mulai dari 1, 2, 3, dan seterusnya. Jumlah titik kontrol 1 hanya menghasilkan kurva Bézier yang berupa titik. Jumlah titik kontrol 2 akan menghasilkan kurva Bézier berupa garis lurus yang menghubungkan kedua titik kontrol tersebut. Untuk jumlah titik kontrol yang lebih tinggi, bentuk kurva Bézier yang dihasilkan akan bergantung pada posisi relatif masing-masing titik kontrol satu sama lain.

Berikut merupakan potongan kode untuk generalisasi algoritma pembangun kurva Bézier.

```python
# Fungsi yang menghasilkan generalisasi kurva bezier menggunakan metode brute force (banyak titik kontrol >= 1)
# Fungsi ini mengembalikan list of tuple yang berisi koordinat titik-titik yang membentuk kurva bezier
# Fungsi ini:
#   - menerima 4 parameter, yaitu
#       List of titik kontrol
#       parameter iterasi yang menentukan tingkat kedetailan kurva bezier
#       parameter animasikan yang menentukan apakah tiap iterasi akan di animasikan atau tidak
#       parameter pause yang menentukan lama jeda antar animasi
#   - mengembalikan list of tuple yang
#       jumlah mid-point yang dihasilkan adalah 2^(iterasi) + 1 (sudah termasuk titik kontrol awal dan akhir)
def BF_Generalized_Bezier(Titik_Kontrol : list, iterasi : int, animasikan : bool, pause : float) -> list:
    hasil = []
    n = 2 ** iterasi - 1 # banyak titik yang dihasilkan, belum termasuk titik kontrol awal dan akhir
    dt = 1 / (n + 1)

    for i in range(n + 2):
        t = i * dt
        x = 0
        y = 0
        for j in range(len(Titik_Kontrol)):
            x += (math.comb(len(Titik_Kontrol) - 1, j)) * ((1 - t) ** (len(Titik_Kontrol) - 1 - j)) * (t ** j) * (Titik_Kontrol[j][0])
            y += (math.comb(len(Titik_Kontrol) - 1, j)) * ((1 - t) ** (len(Titik_Kontrol) - 1 - j)) * (t ** j) * (Titik_Kontrol[j][1])

        if animasikan:
            Animation.animate_with_pause([(x, y)], 1, pause)

        hasil.append((x, y))

    return hasil
```

Gambar 83. Generalisasi Algoritma Kurva Bézier Metode *Brute Force*

```python
# Fungsi yang menghasilkan generalisasi kurva bezier menggunakan metode mid-point (divide and conquer) (banyak titik kontrol >= 1)
# Fungsi ini mengembalikan list of tuple yang berisi koordinat titik-titik yang membentuk kurva bezier
# Fungsi ini:
#   - menerima 5 parameter, yaitu
#       List of titik kontrol
#       parameter iterasi yang menentukan tingkat kedetailan kurva bezier
#       parameter warna yang menentukan warna titik tiap iterasi
#       parameter animasikan yang menentukan apakah tiap iterasi akan di animasikan atau tidak
#       parameter pause yang menentukan lama jeda antar animasi
#   - mengembalikan list of tuple yang
#       jumlah mid-point yang dihasilkan adalah 2^(iterasi) - 1 (belum termasuk titik kontrol awal dan akhir)
def DnC_Generalized_Bezier(titik_kontrol : list, iterasi : int, warna : int, animasikan : bool, pause : float) -> list:
    if iterasi == 0:
        return []
    elif iterasi == 1:                  # Basis
        return Bezier_Point(titik_kontrol, warna, animasikan, pause)
    else :                              # Rekurens
        Kiri = Control_Point_Left(titik_kontrol)    # menghasilkan titik kontrol baru untuk iterasi selanjutnya (bagian kiri)
        Kiri.insert(0, titik_kontrol[0])            # menambahkan titik kontrol pertama untuk iterasi selanjutnya

        Kanan = Control_Point_Right(titik_kontrol)  # menghasilkan titik kontrol baru untuk iterasi selanjutnya (bagian kanan)
        Kanan.append(titik_kontrol[-1])             # menambahkan titik kontrol terakhir untuk iterasi selanjutnya

        current = Bezier_Point(titik_kontrol, warna, animasikan, pause)    # menghasilkan titik bezier pada iterasi sekarang

        return ( DnC_Generalized_Bezier(Kiri, iterasi - 1, warna, animasikan, pause)
                + current + DnC_Generalized_Bezier(Kanan, iterasi - 1, warna, animasikan, pause) )
```

Gambar 84. Generalisasi Algoritma Kurva Bézier Metode *Divide and Conquer*

DAFTAR PUSTAKA

Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf (itb.ac.id) (Diakses pada 15 Maret 2024).

Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf (itb.ac.id) (Diakses pada 15 Maret 2024).

Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf (itb.ac.id) (Diakses pada 15 Maret 2024).

Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf (itb.ac.id) (Diakses pada 15 Maret 2024).

Bézier curve - Wikipedia (Diakses pada 13 Maret 2024).

www.codeproject.com (Diakses pada 13 Maret 2024).

LAMPIRAN

1. Link Repository

Link : https://github.com/Agil0975/Tucil2_13522006.git

2. Tabel Checkpoint Program

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✓	
2. Program dapat melakukan visualisasi kurva Bézier.	✓	
3. Solusi yang diberikan program optimal.	✓	
4. **[Bonus]** Program dapat membuat kurva untuk n titik kontrol.	✓	
5. [Bonus] Program dapat melakukan visualisasi proses pembuatan kurva.	✓	