

LAPORAN TUGAS KECIL II
IF2211 STRATEGI ALGORITMA

**MEMBANGUN KURVA BEZIER DENGAN ALGORITMA TITIK
TENGAH BERBASIS DIVIDE AND CONQUER**



Disusun oleh :

13522081 Albert

13522113 William Glory Henderson

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG 2023**

DAFTAR ISI

DAFTAR ISI	2
Daftar Gambar	4
BAB I	5
BAB II	7
2.1 Implementasi Algoritma Bruteforce pada kurva Bézier	7
2.2 Implementasi Algoritma Divide and Conquer pada kurva Bézier	8
BAB III	12
3.1 Algoritma Bruteforce untuk 3 titik	12
3.2 Algoritma Divide and Conquer untuk 3 titik	13
BAB IV	14
4.1 Test Case 1	14
4.2 Test Case 2	15
4.3 Test Case 3	16
4.4 Test Case 4	17
4.5 Test Case 5	18
4.6 Test Case 6	19
4.7 Test Case 7	20
4.8 Test Case 8	21
4.9 Test Case 9	22
4.10 Test Case 10	23
4.11 Test Case 11	24
4.12 Test Case 12	25
BAB V	26
5.1 Algoritma Bruteforce untuk n titik	26
5.2 Algoritma Divide and Conquer untuk n titik	27
5.3 Algoritma Untuk membuat Gui	28
BAB VI	33
6.1 Analisis Kompleksitas Algoritma Bruteforce untuk 3 Titik	33
6.2 Analisis Kompleksitas Algoritma Divide and Conquer untuk 3 Titik	33
6.3 Analisis Perbandingan Kompleksitas Algoritma antara Algoritma Bruteforce dengan Algoritma Divide and Conquer dengan 3 titik kontrol	34
6.4 Analisis Kompleksitas Algoritma Bruteforce untuk n Titik (Bonus)	35
6.5 Analisis Kompleksitas Algoritma Divide and Conquer untuk n Titik (Bonus)	35
BAB VII	36

7.1 Kesimpulan	37
7.2 Saran	37
LAMPIRAN	39
Pranala repository	39
Checklist	39
DAFTAR PUSTAKA	40

Daftar Gambar

Gambar 2.2.1. Hasil Divide yang dilakukan terhadap persoalan utama.....	12
Gambar 2.2.2. Hasil Conquer yang dilakukan terhadap persoalan utama.....	12
Gambar 2.2.3. Hasil Combine dari persoalan kecil yang sudah diselesaikan.....	13
Gambar 3.1. Source code fungsi bruteforce 3 titik.....	14
Gambar 3.2. Source code fungsi divide and conquer 3 titik.....	15
Gambar 4.1.1. Test case 1 dengan algoritma bruteforce.....	16
Gambar 4.1.2. Test case 1 dengan algoritma divide and conquer.....	16
Gambar 4.2.1. Test case 2 dengan algoritma bruteforce.....	17
Gambar 4.2.2. Test case 2 dengan algoritma divide and conquer.....	17
Gambar 4.3.1. Test case 3 dengan algoritma bruteforce.....	18
Gambar 4.3.2. Test case 3 dengan algoritma divide and conquer.....	18
Gambar 4.4.1. Test case 4 dengan algoritma bruteforce.....	19
Gambar 4.4.2. Test case 4 dengan algoritma divide and conquer.....	19
Gambar 4.5.1. Test case 5 dengan algoritma bruteforce.....	20
Gambar 4.5.2. Test case 5 dengan algoritma divide and conquer.....	20
Gambar 4.6.1. Test case 6 dengan algoritma bruteforce.....	21
Gambar 4.6.2. Test case 6 dengan algoritma divide and conquer.....	21
Gambar 4.7.1. Test Case 7 dengan algoritma bruteforce.....	22
Gambar 4.7.2. Test Case 7 dengan algoritma divide and conquer.....	22
Gambar 4.8.1. Test case 8 dengan algoritma bruteforce.....	23
Gambar 4.8.2. Test case 8 dengan algoritma divide and conquer.....	23
Gambar 4.9.1. Test case 9 dengan algoritma bruteforce.....	24
Gambar 4.9.2. Test case 9 dengan algoritma divide and conquer.....	24
Gambar 4.10.1. Test case 10 dengan algoritma bruteforce.....	25
Gambar 4.10.1. Test case 10 dengan algoritma divide and conquer.....	25
Gambar 4.11.1. Test case 11 dengan algoritma bruteforce.....	26
Gambar 4.11.2. Test case 11 dengan algoritma divide and conquer.....	26
Gambar 4.12.1. Test case 12 dengan algoritma bruteforce.....	27
Gambar 4.12.2. Test case 12 dengan algoritma divide and conquer.....	27
Gambar 5.1. Source code fungsi bruteforce n titik.....	28
Gambar 5.2. Source code fungsi divide and conquer n titik.....	29
Gambar 5.3.1. Fungsi Parse Input.....	30
Gambar 5.3.2. Plot Bezier Curve.....	31
Gambar 5.3.3. Plot Bezier Curve dengan algoritma Bruteforce.....	32
Gambar 5.3.4. Plot Bezier Curve dengan algoritma Divide and Conquer.....	33

BAB I

PENDAHULUAN

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 realistik, 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.

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P_0 sampai P_n , dengan n disebut order ($n = 1$ 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.

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P_0 dan P_1 yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan **kurva Bézier linier**. Misalkan terdapat sebuah titik Q_0 yang berada pada garis yang dibentuk oleh P_0 dan P_1 , maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

$$Q_0 = B(t) = (1 - t)P_0 + t \cdot P_1, \quad t \in [0, 1]$$

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh $B(t)$ dari P_0 ke P_1 . Misalnya ketika $t = 0.25$, maka $B(t)$ adalah seperempat jalan dari titik P_0 ke P_1 . sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan $B(t)$ membentuk sebuah garis lurus dari P_0 ke P_1 .

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P_2 , dengan P_0 dan P_2 sebagai titik kontrol awal dan akhir, dan P_1 menjadi titik kontrol antara. Dengan

menyatakan titik Q_1 terletak diantara garis yang menghubungkan P_1 dan P_2 , dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q_0 berada, maka dapat dinyatakan sebuah titik baru, R_0 yang berada diantara garis yang menghubungkan Q_0 dan Q_1 yang bergerak membentuk **kurva Bézier kuadratik** terhadap titik P_0 dan P_2 . Berikut adalah uraian persamaannya.

$$Q_0 = B(t) = (1 - t) P_0 + t \cdot P_1, \quad t \in [0, 1]$$

$$Q_1 = B(t) = (1 - t) P_1 + t \cdot P_2, \quad t \in [0, 1]$$

$$R_0 = B(t) = (1 - t) Q_0 + t \cdot Q_1, \quad t \in [0, 1]$$

dengan melakukan substitusi nilai Q_0 dan Q_1 , maka diperoleh persamaan sebagai berikut.

$$R_0 = B(t) = (1 - t)^2 P_0 + (1 - t) t \cdot P_1 + t^2 \cdot P_2, \quad t \in [0, 1]$$

Proses ini dapat juga diaplikasikan untuk jumlah titik yang lebih dari tiga, misalnya empat titik akan menghasilkan kurva Bézier kubik, lima titik akan menghasilkan kurva Bézier kuartik, dan seterusnya. Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik.

Dalam Tugas Kecil II IF2211 Strategi Algoritma ini, kami akan mengidentifikasi titik-titik yang membentuk kurva Bézier dengan titik awal, titik kontrol, dan titik akhir yang ditentukan sebelumnya, menggunakan pendekatan Divide and Conquer. Selain itu, kami akan melakukan perbandingan kinerja antara algoritma brute force dan algoritma Divide and Conquer dalam hal waktu eksekusi untuk menentukan metode yang lebih efisien. Kami juga memperluas kemampuan algoritma Divide and Conquer agar dapat menerima jumlah titik kontrol yang tidak terbatas (n titik kontrol), sehingga memungkinkan pembuatan kurva Bézier yang lebih kompleks dan fleksibel. Dengan demikian, program yang kami kembangkan dapat menyelesaikan berbagai jenis kurva Bézier, mulai dari linear hingga kuartik, menggunakan algoritma Divide and Conquer yang telah disesuaikan.

BAB II

IMPLEMENTASI ALGORITMA

2.1 Implementasi Algoritma *Bruteforce* pada kurva Bézier

Algoritma *bruteforce* adalah algoritma yang menggunakan pendekatan yang dibuat secara langsung hanya untuk memecahkan persoalan yang diminta. Algoritma brute force mengandalkan kekuatan komputasi untuk mengeksplorasi setiap kemungkinan solusi dalam ruang pencarian. Algoritma ini tidak memilih-milih dan tidak menggunakan prasangka atau pengetahuan sebelumnya untuk mengoptimalkan pencarian. Oleh karena itu, algoritma *bruteforce* bukan merupakan algoritma yang optimal karena algoritma tersebut seringkali menyelesaikan masalah dengan memeriksa semua kemungkinan solusi yang ada dan pada akhirnya baru mengambil solusi yang terbaik. Pendekatan ini sering digunakan ketika tidak ada solusi yang lebih efisien dan pendekatan ini seringkali tidak praktis karena inefisiensi waktu yang diperlukan untuk penyelesaian.

Dalam pembuatan kurva *Bezier*, algoritma *bruteforce* dipakai dalam menentukan titik-titik pada kurva *Bezier* dengan mencoba berbagai nilai parameter t yaitu 0 sampai 1 sehingga rumus untuk bruteforce adalah

$$Q_0 = B(t) = (1 - t) P_0 + t \cdot P_1, \quad t \in [0, 1]$$

$$Q_1 = B(t) = (1 - t) P_1 + t \cdot P_2, \quad t \in [0, 1]$$

$$R_0 = B(t) = (1 - t) Q_0 + t \cdot Q_1, \quad t \in [0, 1]$$

dengan melakukan substitusi nilai Q_0 dan Q_1 , maka diperoleh persamaan sebagai berikut.

$$R_0 = B(t) = (1 - t)^2 P_0 + 2(1 - t)t \cdot P_1 + t^2 \cdot P_2, \quad t \in [0, 1]$$

Dimana:

- $B(t)$ adalah posisi titik pada kurva untuk nilai parameter t .
- t adalah parameter yang nilainya berada dalam rentang 0 hingga 1.
- P_0 adalah titik awal (start point).
- P_1 adalah titik kontrol (control point).
- P_2 adalah titik akhir (end point).

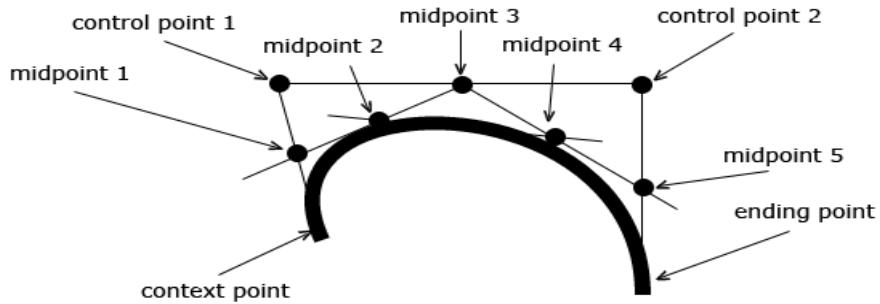
Dalam implementasi algoritma brute force untuk kurva Bézier, posisi titik-titik pada kurva dihitung secara iteratif untuk setiap nilai t yang diambil dari interval yang telah ditentukan. Misalnya, jika ingin menghitung 10 titik pada kurva, maka akan dihitung posisi titik untuk $t = 0$, $t = 0.1$, $t = 0.2$, dan seterusnya hingga $t = 1$. Proses ini akan menghasilkan serangkaian titik yang jika digambarkan akan membentuk kurva Bézier. Semakin banyak iterasi maka kurva yang dihasilkan akan semakin presisi tetapi membutuhkan waktu yang lebih lama.

2.2 Implementasi Algoritma *Divide and Conquer* pada kurva Bézier

Algoritma *Divide and Conquer* adalah algoritma pemecahan masalah yang menggunakan strategi membagi sebuah permasalahan besar menjadi bagian-bagian permasalahan yang lebih kecil secara rekursif. Permasalahan yang lebih kecil tersebut kemudian dicari solusinya kemudian digabungkan dengan solusi dari bagian permasalahan kecil lainnya sehingga menjadi sebuah solusi akhir. Algoritma Divide and Conquer merupakan strategi pemrograman yang penting dan sering digunakan dalam ilmu komputer. Pendekatan ini melibatkan tiga langkah utama: membagi, menaklukkan, dan menggabungkan.

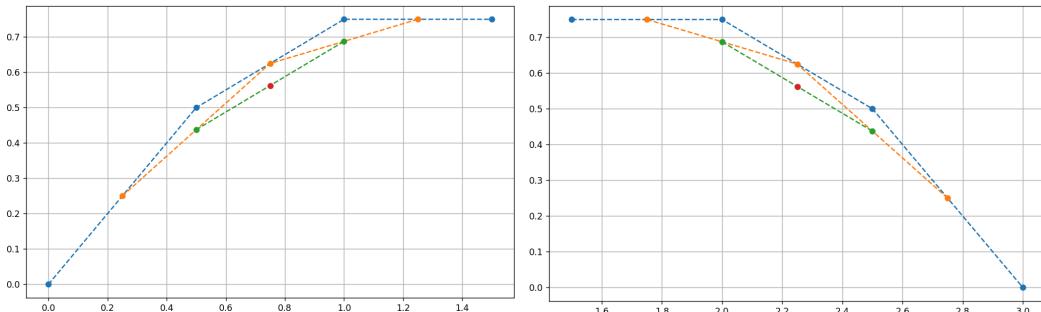
Penyusunan kurva Bezier dapat menggunakan konsep divide and conquer dengan membagi titik titik untuk dilakukan perhitungan lalu digabungkan kembali. Berikut adalah penjelasan lebih mendalam tentang masing-masing langkah:

1. **Memecah (*Divide*):** Dalam implementasi algoritma *Divide and Conquer* untuk kurva Bézier, langkah pemecahan dilakukan dengan membagi kurva menjadi dua segmen yang lebih kecil. Ini diwujudkan dalam kode dengan menghitung titik tengah antara dua titik yang diberikan, yang kemudian digunakan untuk membagi kurva menjadi dua bagian. Proses ini dilakukan secara rekursif, di mana setiap segmen kurva yang lebih kecil dibagi lagi hingga mencapai tingkat iterasi yang ditentukan. Pada setiap iterasi, titik tengah yang dihasilkan dari pembagian kurva dihitung dan ditambahkan ke daftar titik yang akhirnya akan membentuk aproksimasi kurva Bézier. Dengan membagi kurva menjadi segmen-segmen yang lebih kecil dan menghitung titik-titik tengah ini, algoritma secara bertahap membangun kurva Bézier dari titik awal ke titik akhir. Tujuan dari langkah ini adalah untuk menyederhanakan masalah yang kompleks menjadi bagian-bagian yang lebih mudah dikelola.



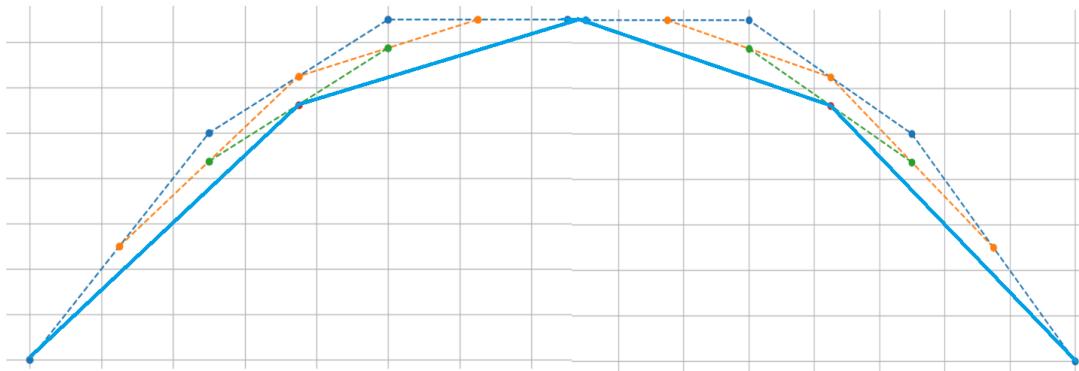
Gambar 2.2.1. Hasil *Divide* yang dilakukan terhadap persoalan utama

2. **Menaklukkan (*Conquer*):** Dalam konteks algoritma Divide and *Conquer* yang diterapkan pada kurva Bézier, langkah "Menaklukkan" (*Conquer*) melibatkan penyelesaian sub-masalah yang telah dibagi pada langkah sebelumnya. Setiap sub-masalah ini didefinisikan oleh segmen kurva yang lebih kecil, yang masing-masing merupakan masalah yang lengkap dan dapat diselesaikan secara independen. Penyelesaian sub-masalah dilakukan dengan menggunakan pendekatan rekursif. Pada setiap panggilan rekursif, titik-titik tengah baru dihitung untuk membagi segmen kurva lebih lanjut menjadi bagian yang lebih kecil. Hal ini dicapai dengan menghitung titik tengah antara pasangan titik yang berurutan. Misalnya, titik tengah antara titik awal dan titik kontrol, dan antara titik kontrol dan titik akhir. Kemudian, titik tengah utama dihitung antara kedua titik tengah ini. Titik tengah utama ini menjadi bagian dari solusi sub-masalah dan akhirnya merupakan salah satu titik yang membentuk kurva Bézier yang diaproksimasi. Proses rekursif ini terus berlanjut, dengan setiap iterasi rekursif lebih lanjut membagi segmen kurva yang lebih kecil dan menghitung titik-titik tengah baru, sampai jumlah iterasi yang ditentukan tercapai. Dengan demikian, dalam langkah "*Conquer*", algoritma secara efektif menyelesaikan setiap sub-masalah dengan menghitung titik-titik yang diperlukan untuk membentuk kurva Bézier yang diinginkan, dan menggabungkan solusi-solusi ini untuk membangun solusi akhir dari masalah utama.



Gambar 2.2.2. Hasil *Conquer* yang dilakukan terhadap persoalan utama

3. **Menggabungkan (Combine):** Dalam konteks kurva Bézier yang diimplementasikan melalui algoritma Divide and Conquer, langkah "Menggabungkan (Combine)" dalam konteks ini terjadi melalui pengumpulan titik-titik tengah yang dihitung selama proses rekursif. Setiap titik tengah yang dihitung dan ditambahkan ke daftar `bezier_points` merupakan bagian dari solusi gabungan yang membentuk kurva Bézier. Karena itu, penggabungan dalam kasus ini terjadi dengan menyatukan titik-titik tengah yang dihitung dari setiap sub-masalah untuk membentuk satu set titik yang mewakili kurva Bézier yang diinginkan. Lebih lanjut, semakin banyak iterasi yang dilakukan, semakin banyak titik tengah yang dihitung dan semakin banyak segmen kurva yang dibagi. Hal ini menghasilkan kurva yang lebih halus dan lebih dekat ke bentuk kurva Bézier yang sebenarnya. Pada dasarnya, setiap iterasi rekursif menambah detail dan kehalusan pada kurva yang dihasilkan dengan menambahkan titik-titik baru yang mendekati kurva Bézier yang sebenarnya. Ini adalah inti dari algoritma Divide and Conquer yang diterapkan pada masalah kurva Bézier, di mana setiap iterasi rekursif memecah, menaklukkan, dan menggabungkan segmen-segmen kurva untuk mencapai solusi akhir yang lebih halus dan akurat.



Gambar 2.2.3. Hasil *Combine* dari persoalan kecil yang sudah diselesaikan

Algoritma Divide and Conquer sering digunakan dalam berbagai aplikasi, termasuk pengurutan data (misalnya, QuickSort dan MergeSort), pencarian elemen (misalnya, Binary Search), dan operasi pada struktur data (misalnya, membangun dan menelusuri pohon pencarian biner). Salah satu keuntungan utama dari algoritma Divide and Conquer adalah kemampuannya untuk mengurangi kompleksitas waktu dari beberapa masalah. Dengan memecah masalah menjadi bagian yang lebih kecil, sering kali lebih mudah untuk menemukan solusi yang efisien. Selain itu, pendekatan ini memungkinkan penggunaan paralelisme dalam pemrosesan, karena sub-masalah dapat diselesaikan secara independen.

BAB III

SOURCE CODE

3.1 Algoritma *Bruteforce* untuk 3 titik

```
# Fungsi untuk mencari titik pada kurva bezier dengan bruteforce untuk 3 titik
def bezier_points_with_bruteforce(points, iteration):
    points = np.array(points)
    total = 2 ** iteration + 1
    t = np.linspace(0, 1, total)
    bezier_points = np.zeros((total, 2))

    for i in range(total):
        bezier_points[i] = (1 - t[i])**2 * points[0] + 2 * \
            (1 - t[i]) * t[i] * points[1] + t[i]**2 * points[2]
    return bezier_points
```

Gambar 3.1. *Source code* fungsi *bruteforce* 3 titik

Algoritma *bruteforce* untuk 3 titik ini memakai rumus yang ada di spek yaitu $R_0 = B(t) = (1 - t)^2 P_0 + 2(1 - t)t.P_1 + t^2.P_2$, $t \in [0, 1]$. Untuk total iterasi dipakai rumus $2^{iterasi} + 1$ agar jumlah iterasi sama dengan algoritma *divide and conquer*. Pemakaian *np.linspace* untuk mengembalikan array nilai yang berjarak sama dalam interval yang ditentukan dengan *np.linspace(batas awal, batas akhir, total elemen)*. Pemakaian *np.zeros* untuk membuat array baru dengan bentuk dan tipe tertentu dan diisi nol. Program melakukan *looping* sebanyak “total” untuk menghitung titik dengan rumus spek.

3.2 Algoritma *Divide and Conquer* untuk 3 titik

```
# Fungsi untuk mencari titik tengah
def find_mid_point(point1, point2):
    return ((point1[0] + point2[0]) / 2, (point1[1] + point2[1]) / 2)

# Fungsi untuk mencari titik pada kurva bezier dengan dnc untuk 3 titik
def bezier_points_with_dnc(point1, point2, point3, current_iteration, iteration, bezier_points):
    if current_iteration < iteration:
        mid_point1 = find_mid_point(point1, point2)
        mid_point2 = find_mid_point(point2, point3)
        mid_point = find_mid_point(mid_point1, mid_point2)
        current_iteration += 1

        # Left side
        bezier_points_with_dnc(point1, mid_point1, mid_point,
                               current_iteration, iteration, bezier_points)
        bezier_points.append(mid_point)

        # Right side
        bezier_points_with_dnc(mid_point, mid_point2, point3,
                               current_iteration, iteration, bezier_points)

# Fungsi untuk menggabungkan titik awal, titik akhir, dan hasil titik dari dnc
def generate_bezier(point1, point2, point3, iterations):
    bezier_points = [point1]
    bezier_points_with_dnc(point1, point2, point3, 0,
                           iterations, bezier_points)
    bezier_points.append(point3)
    return bezier_points
```

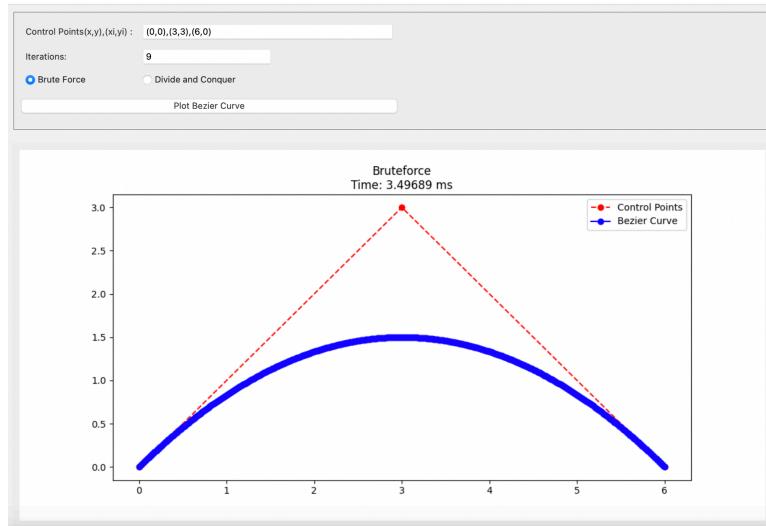
Gambar 3.2. *Source code* fungsi *divide and conquer* 3 titik

Terdapat fungsi *find_mid_point* untuk mencari titik tengah dari 2 titik. Algoritma *divide and conquer* untuk 3 titik menggunakan cara rekursif. Untuk awalnya akan dicari 3 titik tengah terlebih dahulu kemudian baru melakukan rekursif dengan dibagi 2 yaitu sisi kanan dan kiri kemudian akan melakukan rekursif lagi untuk masing-masing sisi (dibagi 2 lagi) sampai kedalaman rekursif sebanyak jumlah iterasi yang diinput. Fungsi *generate_bezier* untuk menggabungkan titik awal dan titik akhir dari *input* oleh pengguna serta memanggil fungsi *divide and conquer* untuk mendapatkan titik pada kurva bezier dan menggabungkannya menjadi *list of bezier_points*.

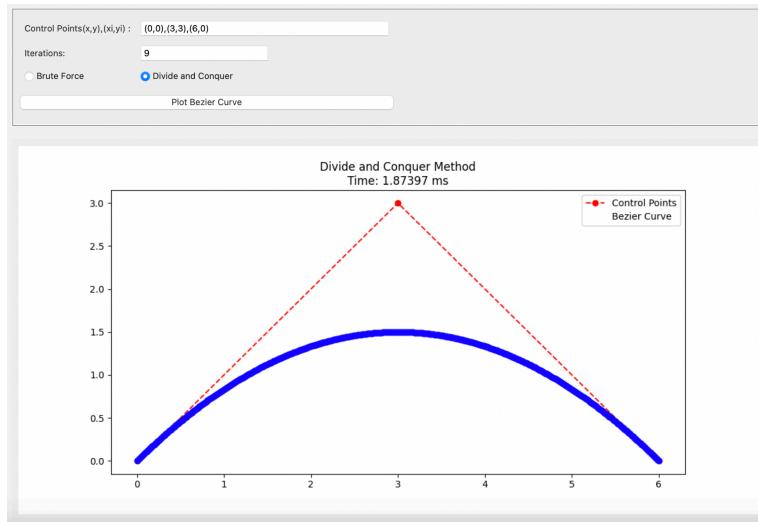
BAB IV

TESTING PROGRAM

4.1 Test Case 1

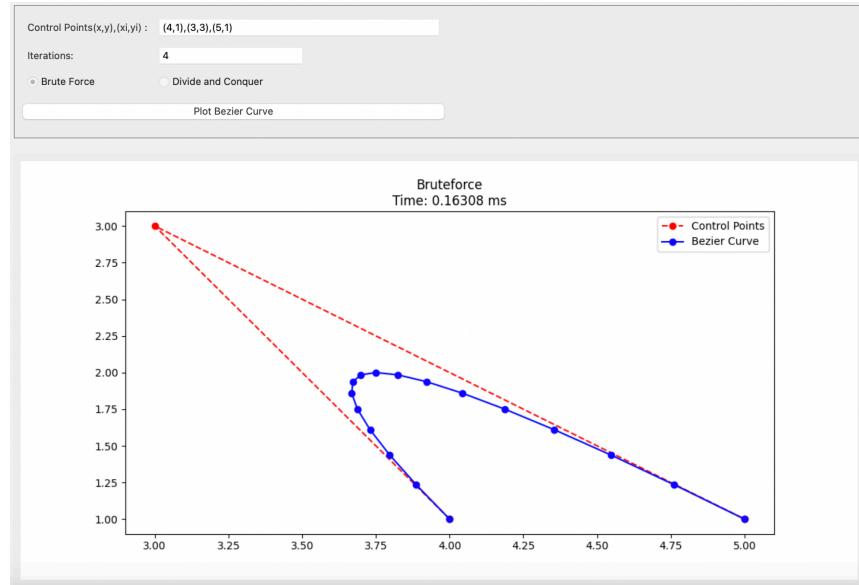


Gambar 4.1.1. Test case 1 dengan algoritma *bruteforce*

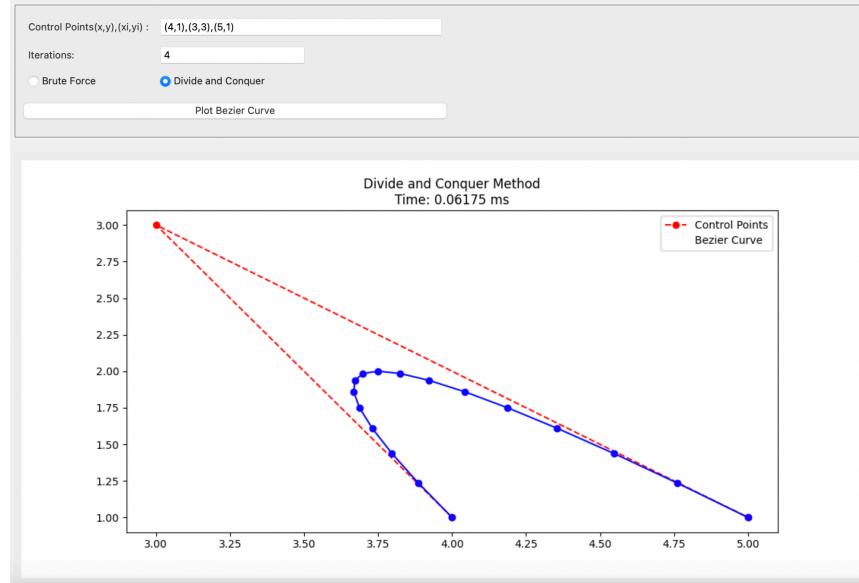


Gambar 4.1.2. Test case 1 dengan algoritma *divide and conquer*

4.2 Test Case 2

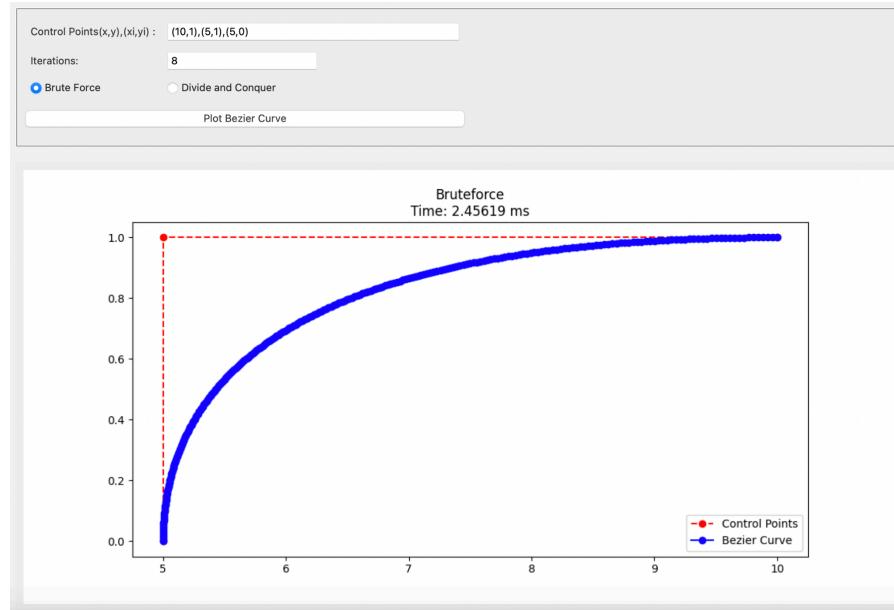


Gambar 4.2.1. Test case 2 dengan algoritma *bruteforce*

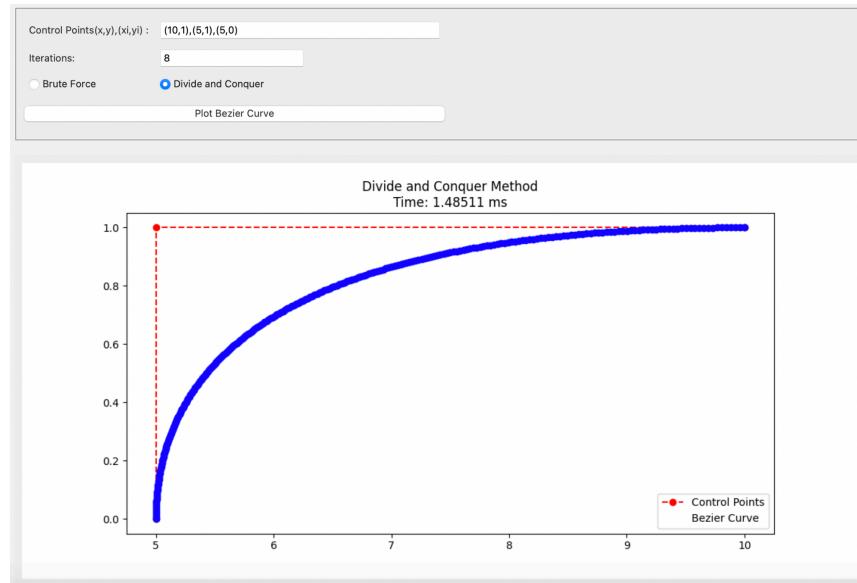


Gambar 4.2.2. Test case 2 dengan algoritma *divide and conquer*

4.3 Test Case 3

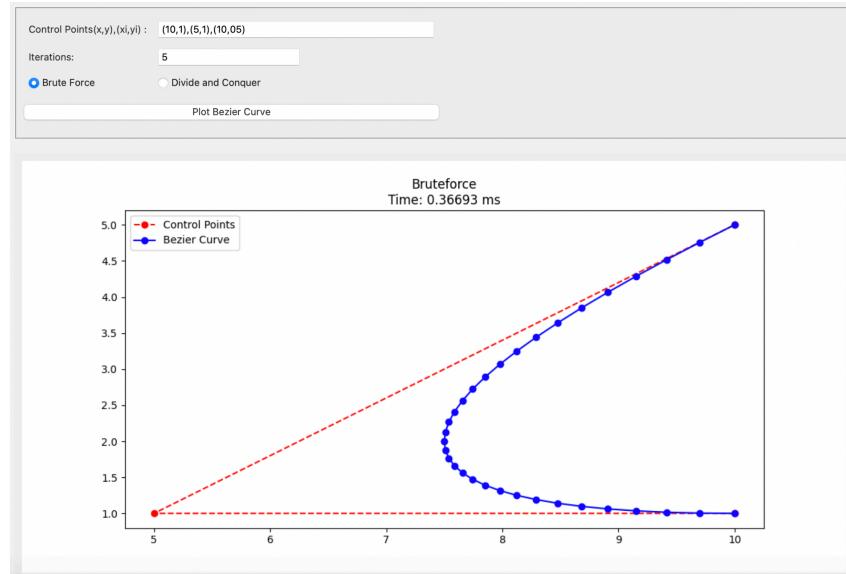


Gambar 4.3.1. Test case 3 dengan algoritma *bruteforce*

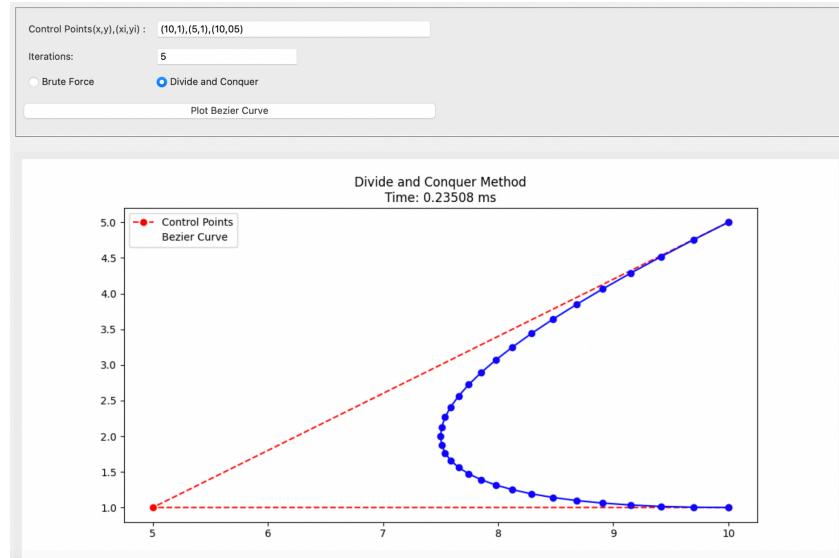


Gambar 4.3.2. Test case 3 dengan algoritma *divide and conquer*

4.4 Test Case 4

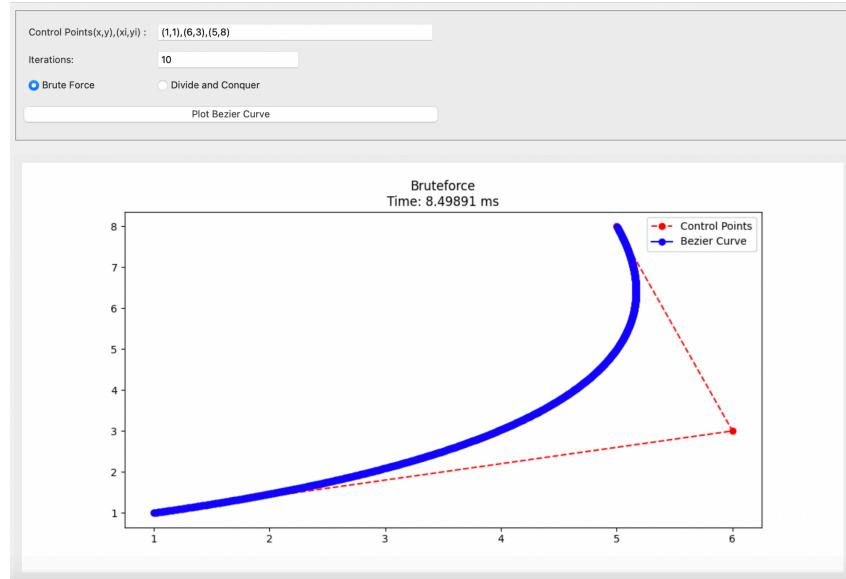


Gambar 4.4.1. Test case 4 dengan algoritma *bruteforce*

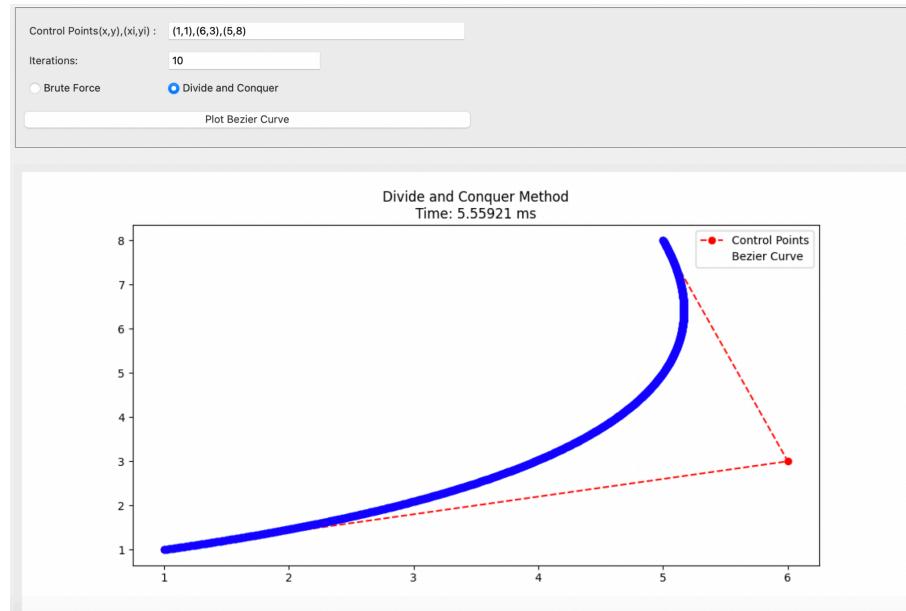


Gambar 4.4.2. Test case 4 dengan algoritma *divide and conquer*

4.5 Test Case 5

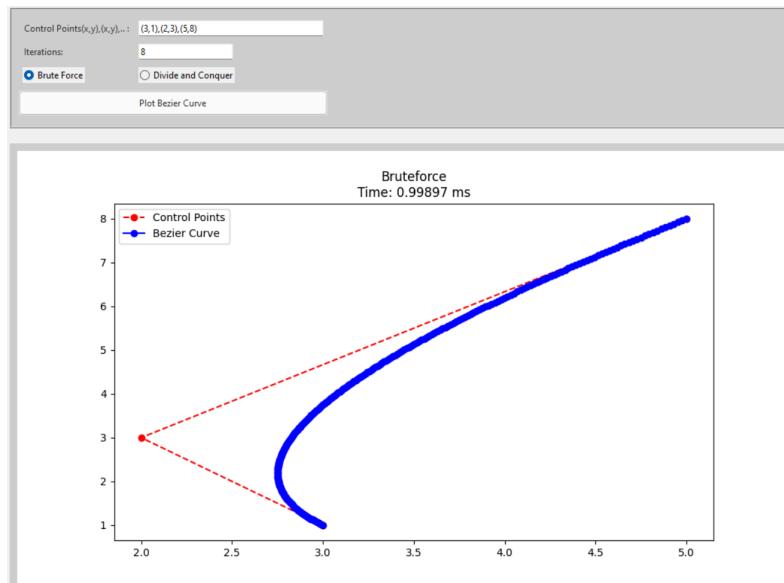


Gambar 4.5.1. Test case 5 dengan algoritma *bruteforce*

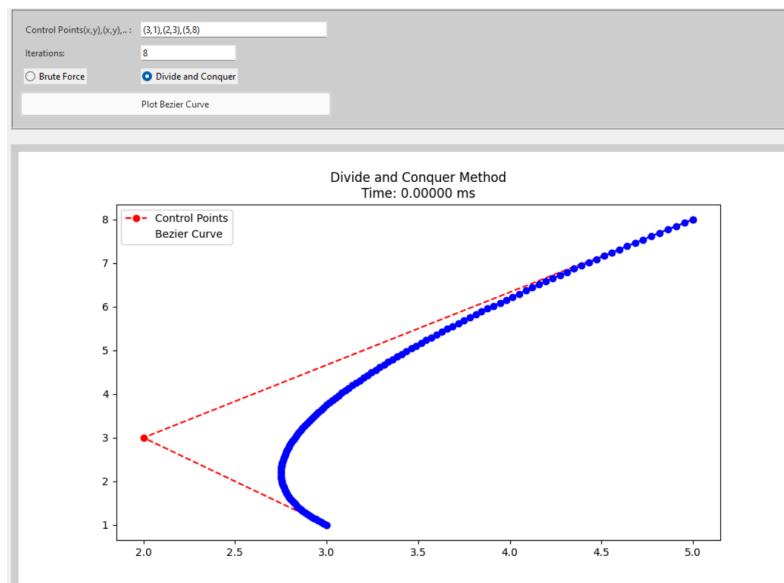


Gambar 4.5.2. Test case 5 dengan algoritma *divide and conquer*

4.6 Test Case 6

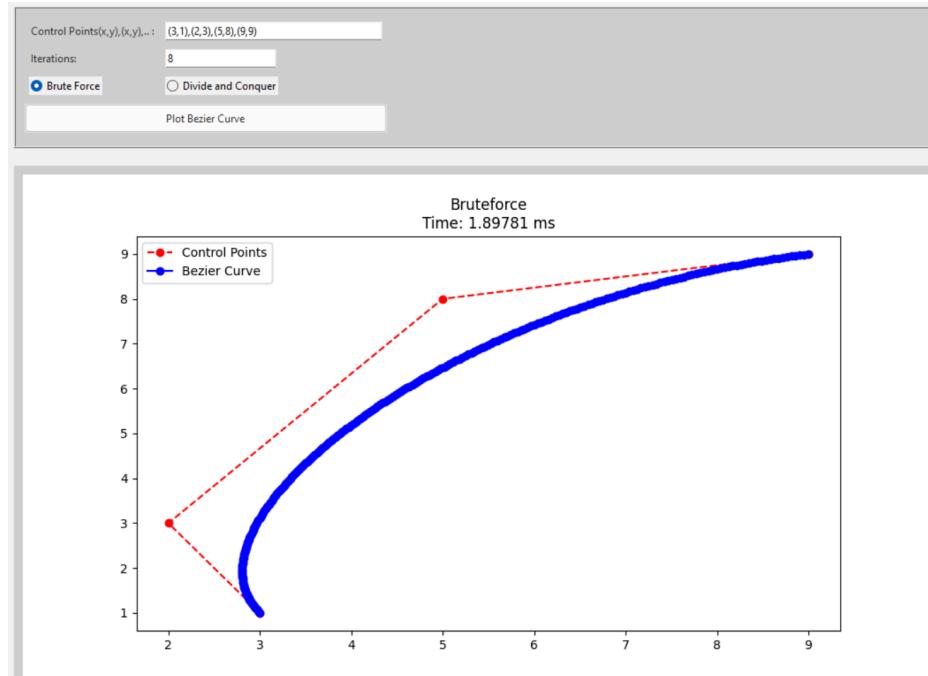


Gambar 4.6.1. Test case 6 dengan algoritma *bruteforce*

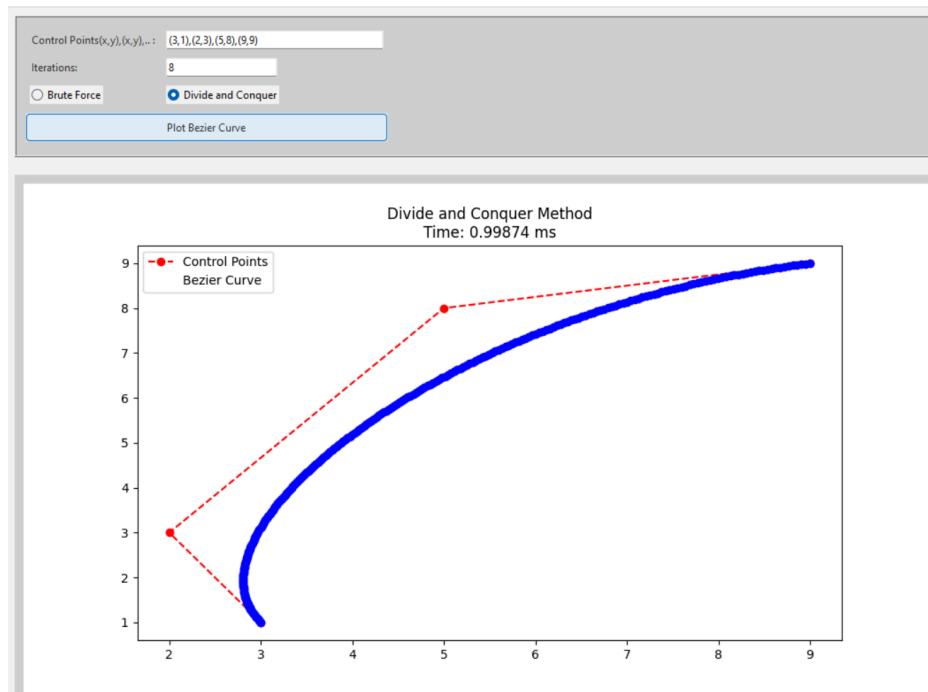


Gambar 4.6.2. Test case 6 dengan algoritma *divide and conquer*

4.7 Test Case 7

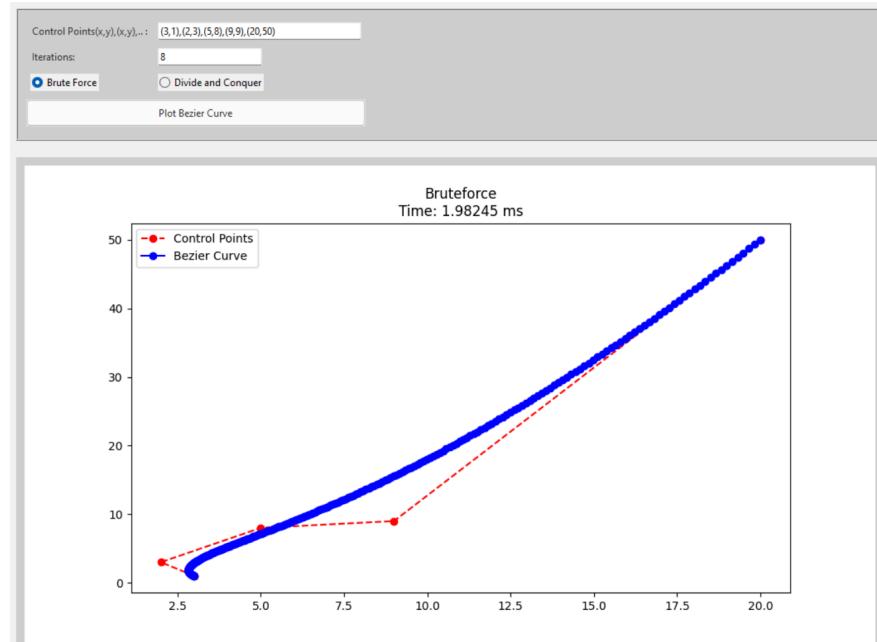


Gambar 4.7.1. Test Case 7 dengan algoritma *bruteforce*

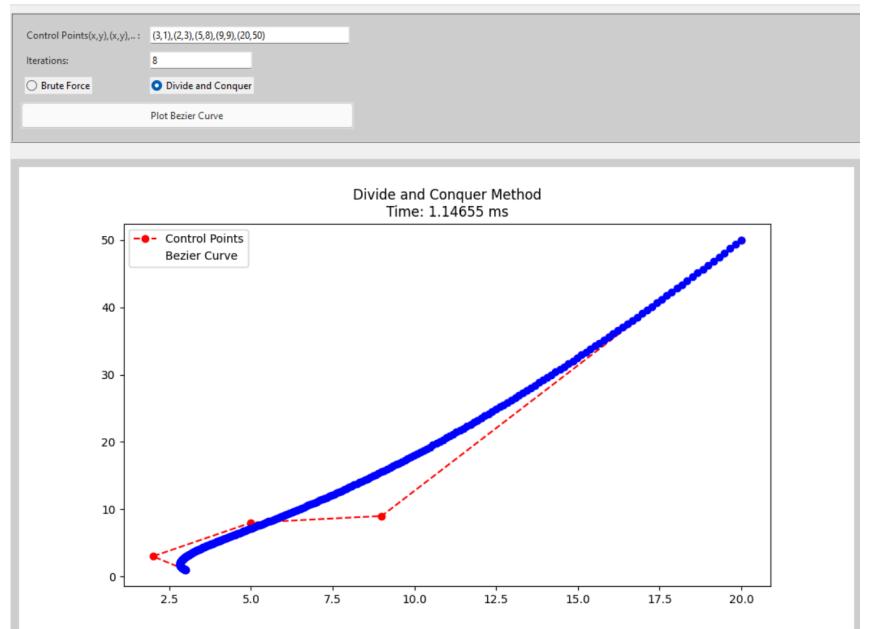


Gambar 4.7.2. Test Case 7 dengan algoritma *divide and conquer*

4.8 Test Case 8

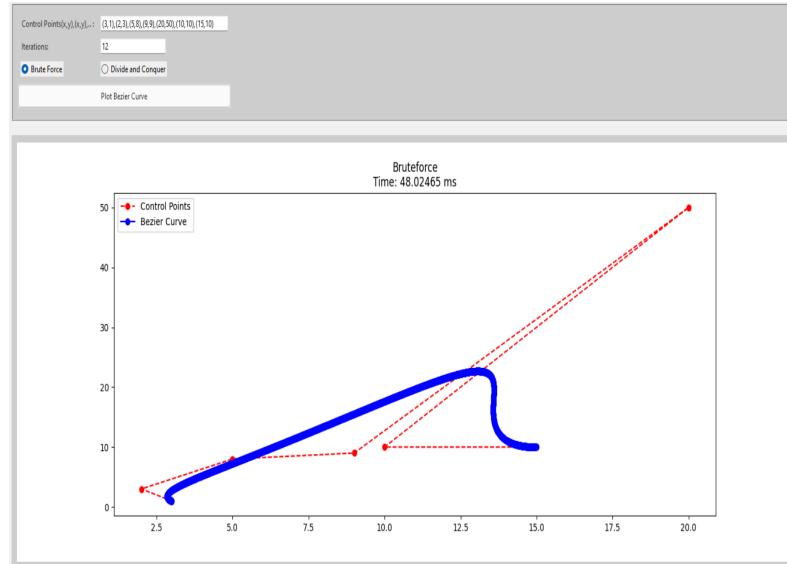


Gambar 4.8.1. Test case 8 dengan algoritma *bruteforce*

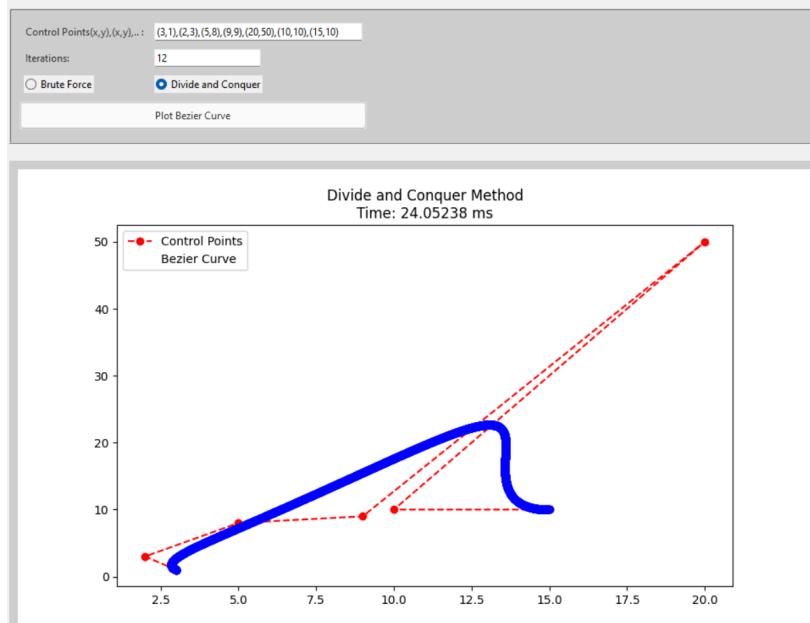


Gambar 4.8.2. Test case 8 dengan algoritma *divide and conquer*

4.9 Test Case 9

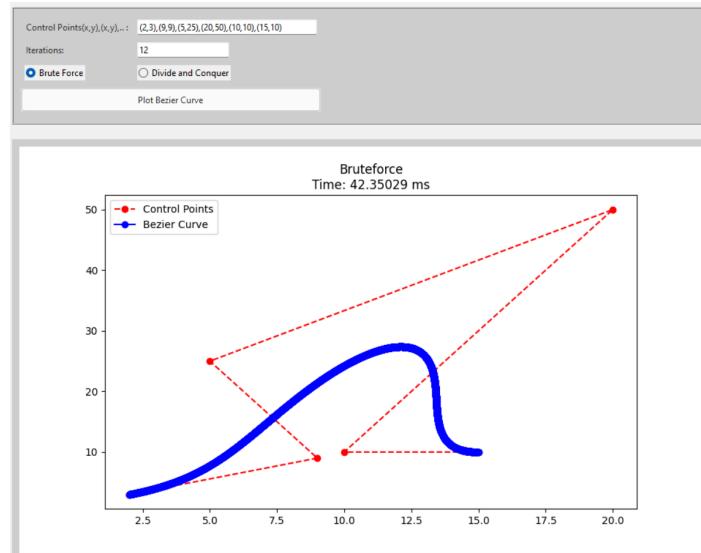


Gambar 4.9.1. Test case 9 dengan algoritma *bruteforce*

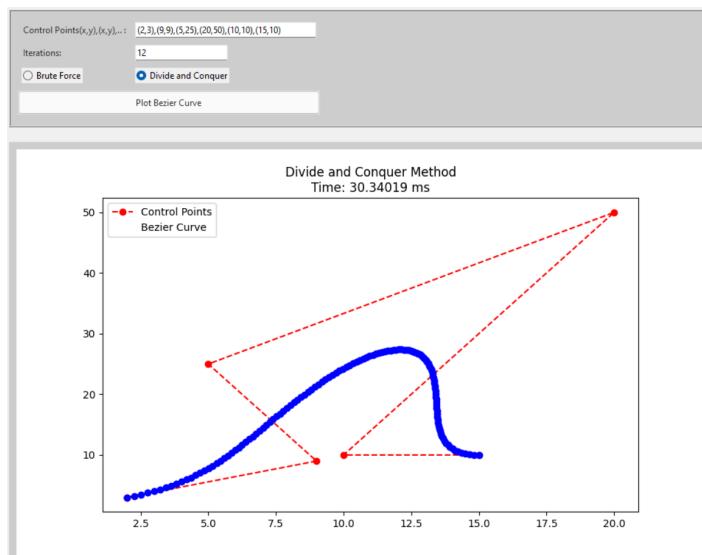


Gambar 4.9.2. Test case 9 dengan algoritma *divide and conquer*

4.10 Test Case 10

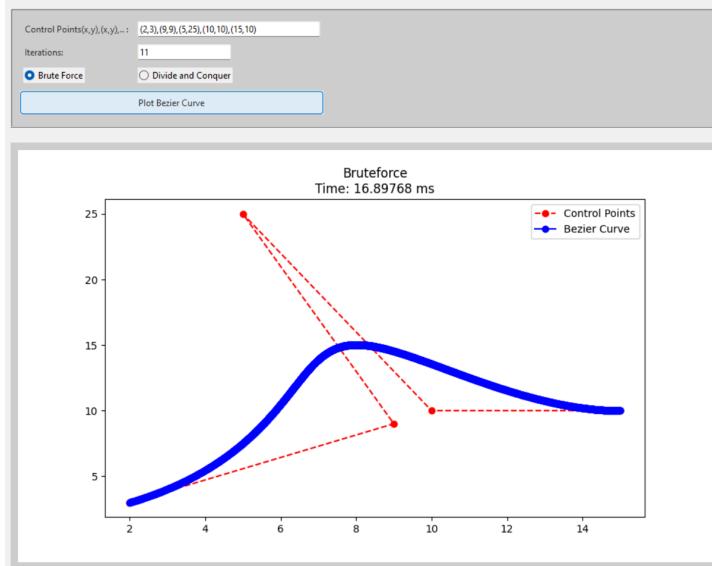


Gambar 4.10.1. Test case 10 dengan algoritma *bruteforce*

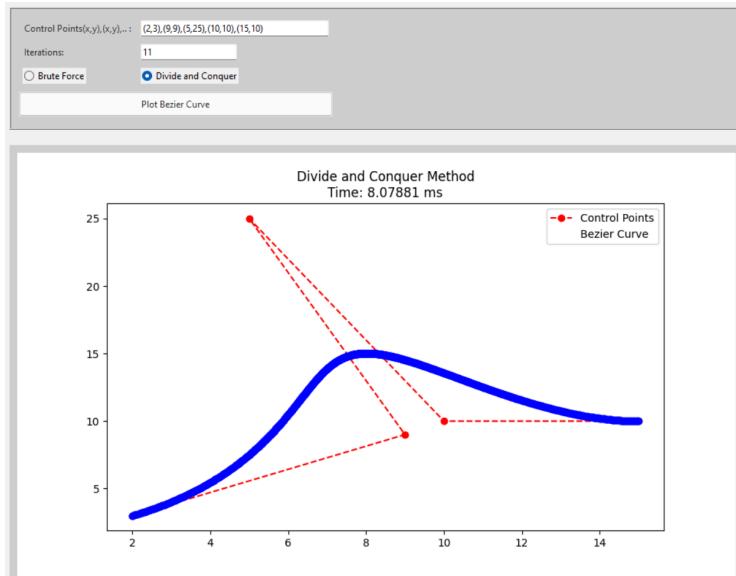


Gambar 4.10.1. Test case 10 dengan algoritma *divide and conquer*

4.11 Test Case 11

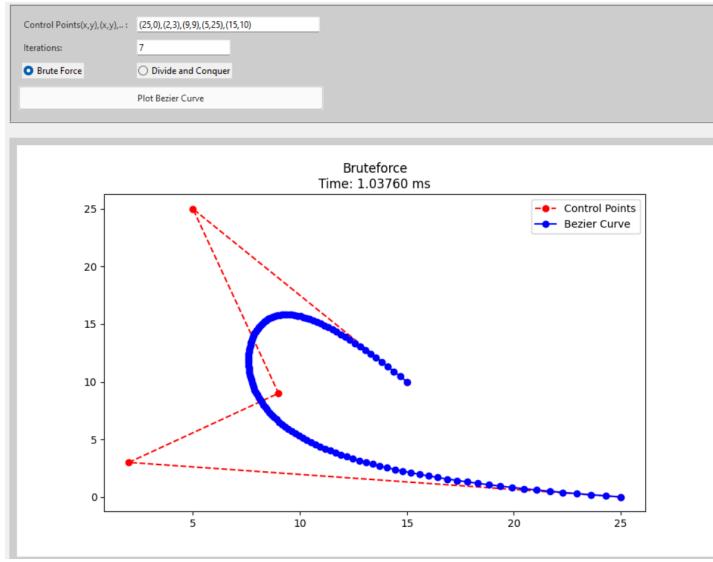


Gambar 4.11.1. Test case 11 dengan algoritma *bruteforce*

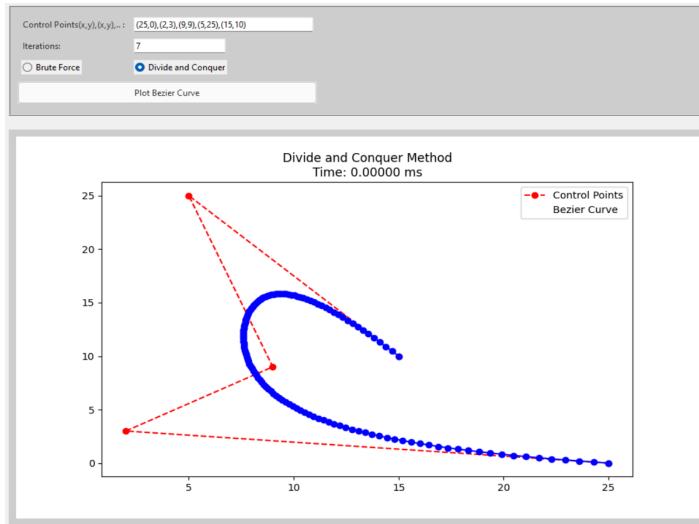


Gambar 4.11.2. Test case 11 dengan algoritma *divide and conquer*

4.12 Test Case 12



Gambar 4.12.1. Test case 12 dengan algoritma *bruteforce*



Gambar 4.12.2. Test case 12 dengan algoritma *divide and conquer*

BAB V

IMPLEMENTASI BONUS

5.1 Algoritma *Bruteforce* untuk n titik

```
def binomial_coef(n, c) :
    if c < 0 or c > n:
        return 0
    if c == 0 or c == n:
        return 1
    return binomial_coef(n - 1, c - 1) + binomial_coef(n - 1, c)

# Fungsi untuk mencari titik pada kurva bezier dengan bruteforce untuk n titik
def bezier_points_with_bruteforce_n(points, iteration):
    total = 2 ** iteration + 1
    n = len(points) - 1
    t = np.linspace(0, 1, total)
    bezier_points = np.zeros((total, 2))
    # Memakai rumus polinomial bernstein
    for i in range(n + 1):
        coef = binomial_coef(n, i)
        for j in range(total):
            formula = coef * (t[j] ** i) * ((1 - t[j]) ** (n - i))
            bezier_points[j] += np.array(points[i]) * formula
    return bezier_points
```

Gambar 5.1. *Source code* fungsi *bruteforce* n titik

Terdapat fungsi *binomial_coef* untuk menghitung coef binomial dengan memakai rekursi. Untuk total iterasi dipakai rumus $2^{iterasi} + 1$ agar jumlah iterasi sama dengan algoritma *divide and conquer*. Algoritma *bruteforce* untuk n titik memakai rumus polinomial *bernstein* yaitu dengan memanfaatkan rumus distribusi binomial kemudian dikalikan dengan titiknya untuk menentukan titik-titik pada kurva bezier.

5.2 Algoritma *Divide and Conquer* untuk n titik

```
def find_mid_point(point1, point2):
    return ((point1[0] + point2[0]) / 2, (point1[1] + point2[1]) / 2)

# Ini untuk menyimpan semua titik yang dihasilkan
def bezier_points_with_dnc_n(point, iteration):
    iteration_points = []
    save_point = []

    def recursive(points, current_iteration):
        if current_iteration < iteration:
            new_points = [points]
            for _ in range(len(points) - 1):
                new_mid = [find_mid_point(points[i], points[i + 1])
                           for i in range(len(points) - 1)]
                new_points.append(new_mid)
            points = new_mid

            left = [p[0] for p in new_points] # Left side
            right = [p[-1] for p in reversed(new_points)] # Right side

            recursive(left, current_iteration + 1)
            iteration_points.append(new_points[-1][0])
            save_point.append([new_points[-1]])
            recursive(right, current_iteration + 1)

    recursive(point, 0)
    return save_point
```

Gambar 5.2. *Source code* fungsi *divide and conquer* n titik

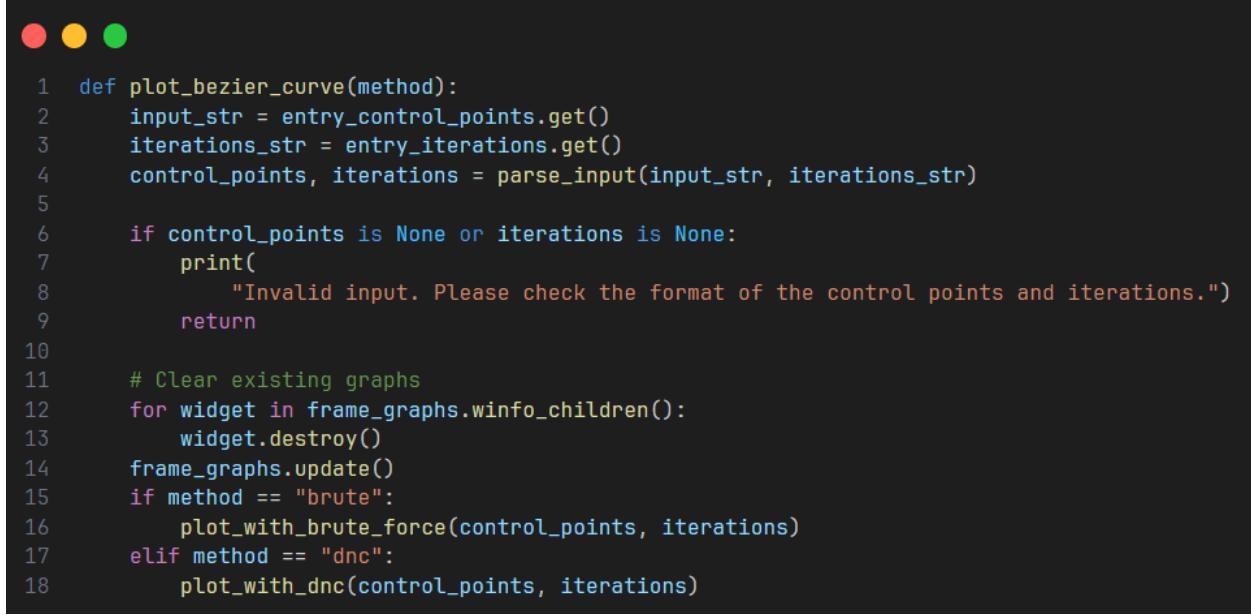
Fungsi *bezier_points_with_dnc_n* menggunakan fungsi bantuan *find_mid_point* untuk menghitung titik tengah antara dua titik yang diberikan. Fungsi *find_mid_point* ini akan digunakan secara berulang dalam proses pembagian kurva menjadi segmen-segmen yang lebih kecil. Algoritma *divide and conquer* untuk n titik ini juga menggunakan rekursi sama seperti *divide and conquer* untuk 3 titik. Dalam proses rekursif yang dilakukan oleh *bezier_points_with_dnc_n*, kurva dibagi menjadi dua bagian (kiri dan kanan) pada setiap iterasi. Titik-titik tengah yang dihasilkan pada setiap iterasi kemudian disimpan dalam daftar *iteration_points*. Kemudian dari tiap poin yang dihasilkan dicari lagi *midpoint* sampai *midpoint* dari point yang dicari hanya bersisa 1. Selain itu, fungsi ini juga menyimpan setiap titik tengah yang dihasilkan dalam daftar *save_point* untuk setiap iterasi. Daftar *save_point* ini berisi titik-titik tengah pada setiap iterasi, yang nantinya dapat digunakan untuk menganalisis proses pembentukan kurva Bézier secara lebih detail.

5.3 Algoritma Untuk membuat Gui

```
● ● ●
1 def parse_input(input_str, iterations_str):
2     if not input_str.startswith("(") or not input_str.endswith(")") or "(" not in input_str:
3         messagebox.showerror(
4             "Input Error", "Error Control Point: Invalid format")
5         return None, None
6     try:
7         control_points_str = input_str.split(",")
8         control_points = [tuple(map(float, point.strip("(").split(",")))
9                            for point in control_points_str]
10    except ValueError as e:
11        messagebox.showerror("Input Error", f"Error Control Point: {e}")
12        return None, None
13    try:
14        iterations = int(iterations_str)
15        return control_points, iterations
16    except ValueError as e:
17        messagebox.showerror("Input Error", f"Error Iteration : {e}")
18        return None, None
```

Gambar 5.3.1. Fungsi Parse Input

Fungsi *parse_input* bertujuan untuk mengolah dan memvalidasi input yang diberikan pengguna dalam bentuk *string* untuk titik kontrol dan jumlah iterasi pada kurva Bézier. Fungsi ini menerima dua parameter, yaitu *input_str* yang merupakan *string* yang berisi titik kontrol dan *iterations_str* yang merupakan *string* yang berisi jumlah iterasi. Pada awalnya, fungsi ini memeriksa apakah format *string* untuk titik kontrol sudah sesuai, yaitu dimulai dan diakhiri dengan tanda kurung dan memiliki separator antar titik kontrol. Jika format *string* titik kontrol sudah sesuai, fungsi akan mencoba mengurai *string* tersebut menjadi daftar *tuple* yang berisi titik kontrol dengan mengonversi setiap elemen *string* menjadi nilai *float*. Selanjutnya, fungsi ini mencoba mengonversi *string* yang berisi jumlah iterasi menjadi bilangan bulat. Jika konversi berhasil, fungsi akan mengembalikan daftar *tuple* titik kontrol dan jumlah iterasi sebagai output. Jika format tidak sesuai, fungsi akan menampilkan pesan kesalahan dan mengembalikan nilai *None* untuk kedua output. Namun, jika terjadi kesalahan dalam konversi titik kontrol atau jumlah iterasi, seperti nilai yang bukan angka, fungsi akan menampilkan pesan kesalahan dan mengembalikan nilai *None* untuk kedua output.



```
1 def plot_bezier_curve(method):
2     input_str = entry_control_points.get()
3     iterations_str = entry_iterations.get()
4     control_points, iterations = parse_input(input_str, iterations_str)
5
6     if control_points is None or iterations is None:
7         print(
8             "Invalid input. Please check the format of the control points and iterations.")
9         return
10
11    # Clear existing graphs
12    for widget in frame_graphs.winfo_children():
13        widget.destroy()
14    frame_graphs.update()
15    if method == "brute":
16        plot_with_brute_force(control_points, iterations)
17    elif method == "dnc":
18        plot_with_dnc(control_points, iterations)
```

Gambar 5.3.2. Plot Bezier Curve

Fungsi *plot_bezier_curve* bertujuan untuk menggambar kurva Bézier berdasarkan metode yang dipilih, yaitu *bruteforce* atau *divide and conquer*. Fungsi ini menerima satu parameter, yaitu *method*, yang menentukan metode yang digunakan untuk menghitung titik-titik pada kurva Bézier. Fungsi *parse_input* dipanggil untuk memvalidasi dan mengurai input menjadi daftar titik kontrol dan jumlah iterasi. Jika input tidak valid, fungsi akan menampilkan pesan kesalahan dan menghentikan eksekusi lebih lanjut. Jika input valid, fungsi akan melanjutkan dengan menghapus grafik yang ada sebelumnya di antarmuka pengguna untuk mempersiapkan tempat bagi grafik baru yang akan digambar. Bergantung pada nilai parameter *method*, fungsi akan memanggil *plot_with_brute_force* atau *plot_with_dnc* untuk menggambar kurva Bézier menggunakan metode *bruteforce* atau *divide and conquer*, masing-masing. Fungsi-fungsi ini bertanggung jawab untuk menghitung titik-titik pada kurva dan menampilkannya pada antarmuka pengguna.



```

1 def plot_with_brute_force(control_points, iterations):
2     fig, ax = plt.subplots(figsize=(5, 5))
3
4     start_time_brute = time.time()
5     if len(control_points) == 3:
6         bezier_points = bezier_points_with_bruteforce(
7             control_points, iterations)
8
9     else:
10        bezier_points = bezier_points_with_bruteforce_n(
11            control_points, iterations)
12
13    end_time_brute = time.time()
14    time_taken_brute = (end_time_brute - start_time_brute) * 1000
15    print(f'Bruteforce\nTime: {time_taken_brute:.5f} ms')
16    # print(bezier_points)
17    ax.plot(*zip(*control_points), 'ro--', label='Control Points')
18    bezier_line, = ax.plot(
19        *zip(*bezier_points[:1]), 'b-', marker='o', label='Bezier Curve')
20
21    def update_brute(frame):
22        bezier_line.set_data(*zip(*bezier_points[:frame+1]))
23        return bezier_line,
24
25    ani_brute = FuncAnimation(fig, update_brute, frames=range(
26        len(bezier_points)), interval=1, blit=True, repeat=False)
27    ax.set_title(f'Bruteforce\nTime: {time_taken_brute:.5f} ms')
28    ax.legend()
29
30    # Embed the graph into the Tkinter GUI
31    canvas = FigureCanvasTkAgg(fig, master=frame_graphs)
32    canvas.draw()
33    canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
34

```

Gambar 5.3.3. Plot Bezier Curve dengan algoritma *Bruteforce*

Fungsi *plot_with_brute_force* bertujuan untuk menggambar kurva Bézier menggunakan metode *bruteforce*. Fungsi ini menerima dua parameter, yaitu *control_points* yang merupakan daftar titik kontrol dan *iterations* yang merupakan jumlah iterasi. Pertama, fungsi ini membuat figur dan sumbu untuk grafik menggunakan *plt.subplots*. Kemudian, fungsi ini mencatat waktu mulai perhitungan dengan *time.time()*. Bergantung pada jumlah titik kontrol, fungsi ini memanggil *bezier_points_with_bruteforce* atau *bezier_points_with_bruteforce_n* untuk menghitung titik-titik pada kurva Bézier. Titik kontrol dan kurva Bézier yang dihasilkan kemudian

digambarkan pada sumbu. Fungsi ini menggunakan *FuncAnimation* dari *matplotlib.animation* untuk menganimasikan pembentukan kurva Bézier titik demi titik.



```
1 def plot_with_dnc(control_points, iterations):
2     fig, ax = plt.subplots(figsize=(5, 5))
3
4     start_time_dnc = time.time()
5     a = bezier_points_with_dnc_n(control_points, iterations)
6     end_time_dnc = time.time()
7     time_taken_dnc = (end_time_dnc - start_time_dnc) * \
8         1000 # Convert to milliseconds
9     print(f'dnc\nTime: {time_taken_dnc:.5f} ms')
10    ax.plot(*zip(*control_points), 'ro--', label='Control Points')
11    bezier_lines_dnc = []
12    for i in range(iterations + 1):
13        bezier_points_dnc = bezier_points_with_dnc_n(control_points, i)
14        bezier_points_flat = [
15            item for sublist in bezier_points_dnc for subsublist in sublist for item in subsublist]
16        bezier_points_flat = [control_points[0]] + \
17            bezier_points_flat + [control_points[-1]]
18        line, = ax.plot(*zip(*bezier_points_flat), 'b-',
19                         label='Bezier Curve' if iterations == i else None, marker='o')
20        bezier_lines_dnc.append(line)
21    # print(bezier_points_flat)
22
23    def update_dnc(frame):
24        if frame == iterations:
25            for line in bezier_lines_dnc:
26                line.set_visible(False)
27            bezier_lines_dnc[-1].set_visible(True)
28        else:
29            for i, line in enumerate(bezier_lines_dnc):
30                line.set_visible(i == frame)
31    return bezier_lines_dnc
32
33 ani_dnc = FuncAnimation(fig, update_dnc, frames=range(
34     iterations + 1), interval=500, blit=True, repeat=False)
35 ax.set_title(f'Divide and Conquer Method\nTime: {time_taken_dnc:.5f} ms')
36 ax.legend()
37
38 # Embed the graph into the Tkinter GUI
39 canvas = FigureCanvasTkAgg(fig, master=frame_graphs)
40 canvas.draw()
41 canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
```

Gambar 5.3.4. Plot Bezier Curve dengan algoritma *Divide and Conquer*

Fungsi *plot_with_dnc* bertujuan untuk menggambar kurva Bézier menggunakan *metode divide and conquer*. Fungsi ini menerima dua parameter, yaitu *control_points* yang merupakan daftar titik kontrol dan *iterations* yang merupakan jumlah iterasi. Pertama, fungsi ini membuat figur dan sumbu untuk grafik menggunakan *plt.subplots*. Kemudian, fungsi ini mencatat waktu mulai

perhitungan dengan `time.time()`. Fungsi ini memanggil `bezier_points_with_dnc_n` untuk menghitung titik-titik pada kurva Bézier untuk setiap iterasi. Titik kontrol dan kurva Bézier yang dihasilkan untuk setiap iterasi kemudian digambarkan pada sumbu. Fungsi ini menggunakan `FuncAnimation` dari `matplotlib.animation` untuk menganimasikan pembentukan kurva Bézier untuk setiap iterasi secara berurutan.

Dalam laporan ini, terdapat beberapa batasan yang diberlakukan untuk memastikan kelancaran proses pembuatan kurva Bézier:

1. Format Input Titik Kontrol: Input titik kontrol dari pengguna harus dimasukkan dalam format $(x_1, y_1), (x_2, y_2), \dots$ tanpa adanya spasi di belakang atau di depan.
2. Jumlah Titik Kontrol Minimum: Input titik kontrol harus terdiri dari minimal tiga titik. Hal ini untuk memastikan bahwa kurva Bézier dapat terbentuk dengan benar.
3. Input Iterasi: Jumlah iterasi yang dimasukkan harus lebih dari 0 dan merupakan bilangan bulat. Iterasi digunakan untuk menentukan ketelitian pembentukan kurva Bézier.
4. Batas Iterasi: Jika jumlah iterasi yang dilakukan lebih dari 20 kali, proses pembentukan grafik dan pemrosesan titik akan membutuhkan waktu yang lebih lama. Hal ini disebabkan oleh peningkatan kompleksitas perhitungan yang diperlukan untuk iterasi yang lebih banyak.

Dengan memperhatikan batasan-batasan tersebut, diharapkan proses pembuatan kurva Bézier dapat berjalan dengan lancar dan menghasilkan kurva yang akurat sesuai dengan spesifikasi yang diberikan.

BAB VI

HASIL ANALISIS

6.1 Analisis Kompleksitas Algoritma *Bruteforce* untuk 3 Titik

Dalam analisis kompleksitas algoritma *bruteforce* untuk penyusunan kurva Bézier, kita memfokuskan pada jumlah iterasi dan langkah-langkah yang diperlukan untuk menghitung titik-titik pada kurva. Kompleksitas waktu dari algoritma *bruteforce* ini berkaitan langsung dengan jumlah iterasi yang dimasukkan oleh pengguna. Dalam proses penyusunan kurva Bézier menggunakan algoritma *bruteforce*, kompleksitas algoritmanya adalah $O(2^n)$ dengan n sebagai jumlah iterasi dalam algoritma. Untuk iterasi pada *bruteforce* akan dilakukan sebanyak $2^{iterasi} + 1$ agar titik yang dihasilkan setara dengan algoritma *divide and conquer*. Kemudian untuk jumlah titik dianggap konstan yaitu 3 titik karena tujuan utamanya membandingkan algoritma *bruteforce* dan *divide and conquer*. Sehingga jumlah titik dianggap tidak mempengaruhi kompleksitas.

Untuk kompleksitas waktu eksekusi perhitungan total iterasi adalah $T(n) = 1$, untuk pembuatan array yang menampung interval adalah $T(n) = 2^n + 1$, untuk inisialisasi array bezier adalah $T(n) = 2^n + 1$, dan untuk perulangan dalam menghitung hasil akhir $T(n) = 1 \cdot (2^n + 1)$ dengan n sebagai jumlah iterasi dalam algoritma. Sehingga untuk keseluruhan $T(n) = 3 \cdot (2^n + 1) + 2$ tetapi karena 2^n tumbuh lebih cepat daripada konstanta dan penambahan maka dalam notasi *bigO* dapat disederhanakan menjadi $T(n) \approx O(2^n)$.

6.2 Analisis Kompleksitas Algoritma *Divide and Conquer* untuk 3 Titik

Dalam analisis proses penyusunan kurva Bézier menggunakan algoritma *divide and conquer*, kompleksitas algoritmanya adalah $O(2^n)$ dengan n sebagai jumlah iterasi yang dimasukkan oleh pengguna. Kemudian untuk jumlah titik dianggap konstan yaitu 3 titik karena tujuan utamanya membandingkan algoritma *bruteforce* dan *divide and conquer*. Sehingga jumlah titik dianggap tidak mempengaruhi kompleksitas.

Untuk kompleksitas waktu eksekusi perhitungan 3 midpoint adalah $T(n) = 3$, untuk langkah rekursif adalah $T(n) = 2^n - 1$ karena pada iterasi 1 ada 1 pemanggilan rekursif untuk membagi 2 kurva dan ketika iterasi 2 akan ada 2 pemanggilan dan ketika iterasi 3 akan ada 4 pemanggilan dan seterusnya. Sehingga didapat $1 + 2 + 2^2 + 2^3 + 2^{n-1}$ dan karena ini adalah deret geometri maka dapat diubah menjadi $2^n - 1$. Sehingga untuk keseluruhan $T(n) = (2^n - 1) + 3$ dengan n sebagai jumlah iterasi yang dimasukkan oleh pengguna. Tetapi karena 2^n tumbuh lebih cepat daripada konstanta dan penambahan maka dalam notasi *bigO* dapat disederhanakan menjadi $T(n) \approx O(2^n)$.

6.3 Analisis Perbandingan Kompleksitas Algoritma antara Algoritma *Bruteforce* dengan Algoritma *Divide and Conquer* dengan 3 titik kontrol

Dalam analisis perbandingan kompleksitas algoritma antara algoritma *bruteforce* dan algoritma *divide and conquer* untuk penyusunan kurva Bézier dengan tiga titik kontrol, kita dapat mengamati bahwa keduanya memiliki kompleksitas secara teoritis yang serupa yaitu $O(2^n)$, di mana n adalah jumlah iterasi yang dimasukkan oleh pengguna.

Dari hasil testing untuk 3 titik kontrol, dapat disimpulkan bahwa algoritma *divide and conquer* lebih cepat dibandingkan algoritma *bruteforce*. Meskipun kompleksitas sama, tetapi hal ini tetap terjadi karena $T(n)$ dari kedua algoritma berbeda. Algoritma *bruteforce* membutuhkan lebih banyak langkah untuk mencapai hasil iterasi dibandingkan *divide and conquer*. Untuk perbedaan kecepatan tidak begitu signifikan (perbedaannya kecil) karena secara pertumbuhan sama-sama eksponensial.

Namun, pemilihan algoritma yang tepat dapat bergantung pada konteks penggunaan spesifik, termasuk batasan memori dan kebutuhan akan kecepatan perhitungan. Hal ini disebabkan oleh kemampuan algoritma *divide and conquer* untuk memecah masalah menjadi sub-masalah yang lebih kecil dan menyelesaiakannya secara rekursif, sehingga mengurangi redundansi perhitungan tetapi memakai memori yang lebih banyak. Sedangkan *bruteforce* memakai memori yang lebih sedikit tetapi perhitungannya sangat banyak.

Contoh perhitungan jika titik 3 kontrol dengan jumlah iterasi 3:

$$T(n) = 3 \cdot (2^3 + 1) + 2 = 29 \text{ (bruteforce)}$$

$$T(n) = (2^3 - 1) + 3 = 10 \text{ (divide and conquer)}$$

Dapat dilihat bahwa secara $T(n)$ bahwa algoritma *bruteforce* lebih besar sehingga ini menjadikan alasan mengapa *runtime algoritma bruteforce* lebih lama dibandingkan algoritma *divide and conquer*.

6.4 Analisis Kompleksitas Algoritma *Bruteforce* untuk n Titik (Bonus)

Dalam analisis kompleksitas algoritma *bruteforce* untuk penyusunan kurva Bézier, kompleksitas algoritmanya adalah $O(2^n)$ dengan n sebagai jumlah iterasi dalam algoritma. Untuk iterasi pada bruteforce akan dilakukan sebanyak $2^{iterasi} + 1$ agar titik yang dihasilkan setara dengan algoritma *divide and conquer*. Kemudian untuk jumlah titik dianggap konstan yaitu 3 titik karena tujuan utamanya membandingkan algoritma *bruteforce* dan *divide and conquer*. Sehingga jumlah titik dianggap tidak mempengaruhi kompleksitas.

Untuk kompleksitas waktu pembuatan array yang menampung interval adalah $T(n) = 1$, untuk inisialisasi array bezier adalah $T(n) = 1 \cdot (2^{iterasi} + 1)$, untuk eksekusi perhitungan binomial coef adalah $T(n) = n$, untuk perhitungan looping coef adalah $T(n) = n$, dan untuk perhitungan looping rumus adalah. Sehingga untuk keseluruhan $T(n) = n \cdot n \cdot (2^{iterasi} + 1) + 1 + 2^{iterasi} + 1$ dengan n adalah jumlah titik. Tetapi karena jumlah titik dianggap konstan dan $2^{iterasi}$ tumbuh lebih cepat daripada penambahan maka dalam notasi *bigO* dapat disederhanakan menjadi $T(n) \approx O(2^{iterasi})$.

6.5 Analisis Kompleksitas Algoritma *Divide and Conquer* untuk n Titik (Bonus)

Dalam analisis proses penyusunan kurva bérzier menggunakan algoritma *divide and conquer* untuk n titik kontrol, kompleksitas algoritmanya adalah $O(2^n)$ dengan n sebagai jumlah iterasi yang dimasukkan oleh pengguna. Kemudian untuk jumlah titik dianggap konstan yaitu 3 titik karena tujuan utamanya membandingkan algoritma *bruteforce* dan *divide and conquer*. Sehingga jumlah titik dianggap tidak mempengaruhi kompleksitas. Kompleksitasnya $O(2^n)$

karena pada iterasi 1 akan memanggil rekursif untuk membagi 2 kurva dan ketika iterasi 2 akan menjadi 4 dan seterusnya hingga iterasi ke n.

Untuk kompleksitas waktu langkah rekursif adalah $T(n) = 2^{\text{iterasi}} - 1$ sama seperti *divide and conquer* 3 titik, untuk pencarian *midpoint* hingga bersisa 1 adalah $T(n) = (n - 1)^2 \cdot 1$ sehingga untuk keseluruhan $T(n) = 2^{\text{iterasi}} - 1 + (n - 1)^2$ dengan n adalah jumlah titik. Tetapi karena jumlah titik dianggap konstan dan 2^{iterasi} tumbuh lebih cepat daripada penambahan maka dalam notasi *bigO* dapat disederhanakan menjadi $T(n) \approx O(2^{\text{iterasi}})$.

BAB VII

KESIMPULAN DAN SARAN

7.1 Kesimpulan

Dalam tugas ini, kami telah melakukan analisis tentang penerapan algoritma *Divide and Conquer* dalam proses penyusunan kurva Bézier. Berdasarkan hasil perbandingan yang telah dilakukan, baik melalui perhitungan kompleksitas algoritma maupun pengujian praktis, kami dapat menyimpulkan bahwa algoritma *Divide and Conquer* menunjukkan kinerja yang lebih unggul dibandingkan dengan algoritma brute force. Dari segi kompleksitas algoritma, $T(n)$ untuk algoritma *Divide and Conquer* terbukti lebih efisien daripada algoritma *bruteforce*. Hal ini menunjukkan bahwa algoritma *Divide and Conquer* membutuhkan langkah yang lebih sedikit dalam proses perhitungan, sehingga dapat menghemat waktu eksekusi secara signifikan, terutama ketika jumlah titik kontrol dan iterasi meningkat.

Selain itu, hasil pengujian juga mendukung kesimpulan ini. Dalam berbagai skenario pengujian dengan jumlah titik kontrol dan iterasi yang bervariasi, algoritma *Divide and Conquer* secara konsisten menunjukkan waktu eksekusi yang lebih cepat dibandingkan dengan algoritma *bruteforce*. Ini membuktikan keefektifan algoritma *Divide and Conquer* dalam menyusun kurva Bézier, terutama untuk kasus dengan banyak titik kontrol dan iterasi.

Kesimpulannya, algoritma Divide and Conquer merupakan pilihan yang tepat dan efektif dalam proses penyusunan kurva Bézier. Dengan keunggulan dalam kompleksitas algoritma dan hasil pengujian praktis, algoritma ini dapat diandalkan untuk menghasilkan kurva Bézier dengan akurasi dan efisiensi yang tinggi. Dengan demikian, kami merekomendasikan penggunaan algoritma *Divide and Conquer* dalam aplikasi-aplikasi yang memerlukan penyusunan kurva Bézier, terutama ketika dihadapkan pada tantangan jumlah titik kontrol dan iterasi yang besar.

7.2 Saran

Dalam proses penggerjaan tugas kecil ini, sangat penting untuk memperhatikan detail spesifikasi tugas secara cermat, terutama pada bagian bonus yang mungkin kurang jelas. Hal ini dapat mencegah perlunya revisi kode berulang kali yang bisa mengganggu proses penyelesaian tugas. Dianjurkan juga agar tidak menunda pekerjaan mengingat batas waktu pengumpulan yang

telah ditetapkan. Memahami masalah dengan baik sebelum memulai pengembangan program akan membantu dalam menyelesaikan tugas dengan lebih efisien dan menghindari pekerjaan yang sia-sia akibat kurangnya pemahaman terhadap spesifikasi yang diberikan.

LAMPIRAN

Pranala repository

https://github.com/AlbertChoe/Tucil2_13522081_13522113

Checklist

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.	✓	

DAFTAR PUSTAKA

- Rinaldi, M. (2024). Algoritma Divide and Conquer (2024) Bagian 1. Retrieved from [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-\(2024\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-(2024)-Bagian1.pdf).
- Rinaldi, M. (2024). Algoritma Divide and Conquer (2024) Bagian 2. Retrieved from [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-\(2024\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-(2024)-Bagian2.pdf).
- Rinaldi, M. (2024). Algoritma Divide and Conquer (2024) Bagian 3. Retrieved from [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-\(2024\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-(2024)-Bagian3.pdf).
- Rinaldi, M. (2024). Algoritma Divide and Conquer (2024) Bagian 4. Retrieved from [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-\(2024\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-andConquer-(2024)-Bagian4.pdf).