LAPORAN TUGAS KECIL 02

IF2211 STRATEGI ALGORITMA

“Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer”

Disusun oleh:

Mohammad Andhika Fadillah K-03 13522128

Rayhan Ridhar Rahman K-03 13522160

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

DAFTAR ISI

BAB 1

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.

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P0 sampai Pn , 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 P0, sedangkan titik kontrol terakhir adalah P3. Titik kontrol P1 dan P2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Mengulas lebih jauh menge nai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P0 dan P1 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 Q0 yang berada pada garis yang dibentuk oleh P0 dan P1, maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh B(t) dari P0 ke P1. Misalnya ketika t = 0.25, maka B(t) adalah seperempat jalan dari titik P0 ke P1. sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan B(t) membentuk sebuah garis lurus dari P0 ke P1.

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P2, dengan P0 dan P2 sebagai titik kontrol awal dan akhir, dan P1 menjadi titik kontrol antara. Dengan menyatakan titik Q1 terletak diantara garis yang menghubungkan P1 dan P2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q0 berada, maka dapat dinyatakan sebuah titik baru, R0 yang berada diantara garis yang menghubungkan Q0 dan Q1 yang bergerak membentuk kurva Bézier kuadratik terhadap titik P0 dan P2. Berikut adalah uraian persamaannya.

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$

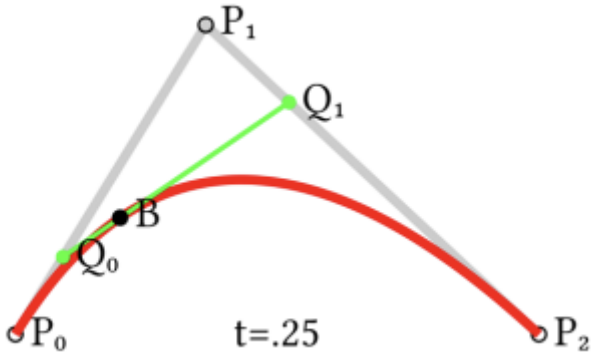
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

$$R_0 = B(t) = (1-t)Q_0 + tQ_1, \qquad t \in [0,1]$$

dengan melakukan substitusi nilai Q0 dan Q1, maka diperoleh persamaan sebagai berikut.

$$R_0 = B(t) = (1-t)^2 P_0 + (1-t)tP_1 + t^2 P_2, \qquad t \in [0,1]$$

Berikut adalah ilustrasi dari kasus diatas.



Gambar 2. Pembentukan Kurva Bézier Kuadratik.

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. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya.

$$S_0 = B(t) = (1-t)^3 P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P_2 + t^3 P_3, \qquad t \in [0,1]$$

$$T_0 = B(t) = (1-t)^4 P_0 + 4(1-t)^3 tP_1 + 6(1-t)^2 t^2 P_2 + 4(1-t)t^3 P_3 + t^4 P_4, \qquad t \in [0,1]$$

Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka Anda diminta untuk mengimplementasikan pembuatan kurva Bézier dengan algoritma titik tengah berbasis divide and conquer.

BAB 2

TEORI SINGKAT

2.1 Algoritma Brute Force

Algoritma *brute force* adalah metode yang digunakan untuk memecahkan masalah atau mencari solusi dengan cara mencoba semua kemungkinan secara berurutan hingga menemukan solusi yang benar. Metode ini sering digunakan ketika tidak ada cara yang lebih efisien untuk menyelesaikan masalah tersebut. Algoritma brute force adalah metode yang sederhana dan kuat, karena dia tidak mengandalkan pengetahuan sebelumnya atau optimasi khusus.

2.2 Algoritma *Divide and Conquer*

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.

BAB 3

ANALISIS & IMPLEMENTASI PROGRAM

3.1 Analisis & Implementasi dalam Algoritma *Brute Force*

Penggunaan algoritma *brute force* dalam menentukan kurva bezier memiliki kelemahan dan juga keuntungan. Algoritma *brute force* yang digunakan dalam pembuatan kurva Bézier memang memiliki kelemahan dalam hal efisiensi saat jumlah titik kontrol sangat besar, karena harus mencoba semua kemungkinan kombinasi. Hal ini dapat mengakibatkan kompleksitas waktu yang tinggi dan membutuhkan sumber daya komputasi yang besar. Meskipun demikian, algoritma ini menawarkan keunggulan dalam presisi yang tinggi dan fleksibilitas dalam menentukan bentuk kurva yang dihasilkan.

Dengan menggunakan algoritma *brute force*, kita dapat menciptakan kurva Bézier yang sangat akurat sesuai dengan titik-titik kontrol yang diberikan. Ini sangat berguna dalam konteks desain grafis, animasi, dan manufaktur di mana kepresisian dan kehalusan kurva menjadi sangat penting. Fleksibilitas algoritma brute force juga memungkinkan kita untuk menyesuaikan kurva yang dihasilkan dengan lebih leluasa, sehingga memungkinkan eksperimen dan iterasi yang lebih mudah dalam pengembangan desain.

3.2 Analisis & Implementasi dalam Algoritma *Divide & Conquer*

Penggunaan algoritma *divide & conquer* dalam menentukan kurva Bezier, memiliki ide yang cukup sederhana. Misalkan terdapat n buah titik yang telah ditentukan urutannya. Untuk iterasi selanjutnya kita mengambil semua titik tengah dari titik yang urutannya relatif dekat. Kemudian akan didapat n - 1 buah titik yang akan diambil kembali titik tengahnya, sampai pada akhirnya hanya 1 titik tersisa yang menjadi titik tengah dari semua titik. Pada iterasi selanjutnya, kita mengambil n buah titik kembali dari bagian kiri dan kanan antara titik-titik yang telah dibentuk pada iterasi sebelumnya. Pembagian titik-titik ini membuat persamaan bezier baru. Pada bagian kiri, dimulai dengan titik paling kiri dan berakhir pada titik tengah dari iterasi sebelumnya, sedangkan pada bagian kanan, diawali oleh titik tengah dan diakhiri titik paling kanan.

Dalam algoritma yang dikembangkan, kami menemukan suatu pola untuk mendapatkan titik tengah melalui suatu rumus, yaitu:

$$midpoint = \frac{\sum_{i=0}^{n} \binom{n}{i} P_i}{2^n}$$

Dengan n merupakan banyak titik yang terlibat. Kemudian untuk mencari titik yang terlibat pada bagian kiri dan kanan iterasi selanjutnya, titik-titik tersebut merupakan titik tengah dari titik terkiri sampai m yang iterasi dari 1 hingga m buah titik kemudian sebaliknya untuk bagian kanan.

3.3 Analisis Perbandingan Solusi dalam Algoritma *Divide & Conquer dengan Algoritma Brute Force*

Algoritma *Divide & Conquer* bekerja dengan cara selalu mencari titik tengah yang nilai iterasinya sama dengan ½ nilai iterasi dalam algoritma brute force untuk titik-titik kontrol yang sama. Berlaku juga dengan bagian turunannya, yang mendapat titik kontrol baru.

Algoritma *Brute Force* menggunakan semua nilai yang ada dan akan mengecek semua nilai tersebut sampai akhir, baru setelah itu dicari nilai yg paling optimal.

Kami melakukan percobaan untuk melihat perbandingan antara dua solusi kurva bezier dengan algoritma yang berbeda. Kami men-*input* 5 titik dengan koordinat (1,4), (2,3), (5,8), (7,4), (9,6) dan juga iterasi yang dilakukan berjumlah 6. Sesuai hasil dari percobaan tersebut, solusi kurva bezier dalam algoritma *brute force* memiliki waktu yang lebih cepat dibandingkan dengan solusi kurva bezier dalam algoritma *divide & conquer.* Berdasarkan analisis kami, hal tersebut dapat terjadi karena banyak faktor, salah satunya yaitu algoritma *divide & conquer* yang kami buat masih belum optimal dan pada algoritma *brute force*, kami menggunakan *library numpy* yang terdapat pada *python* yang memang sudah dioptimalkan untuk memanipulasi array sehingga waktu eksekusi menjadi lebih cepat.

Solusi kurva bezier yang menggunakan algoritma brute force memiliki Kompleksitas algoritma yaitu O(n + cell_amount). Solusi kurva bezier yang menggunakan algoritma *divide & conquer* memiliki Kompleksitas algoritma yaitu O(n * log(n)). Perbandingan kompleksitas algoritma kedua solusi tersebut menunjukkan bahwa O(n * log(n)) cenderung lebih efisien daripada O(n + cell_amount) saat nilai n sangat besar. Namun, perlu diingat bahwa efisiensi algoritma juga dipengaruhi oleh faktor-faktor lain seperti konstanta yang terlibat dalam operasi-operasi algoritma, sifat masukan (misalnya, apakah masukan sudah diurutkan atau tidak), dan implementasi algoritma secara keseluruhan.

3.4 Implementasi Bonus

Kami mengimplementasikan bonus yang diberikan yaitu program yang kami buat dapat menginput n buah titik.

3.5 Source code program implementasi keduanya dalam bahasa pemrograman yang dipilih

3.5.1 bf_bezier.py

```
def compute_bezier_curve(n, i, t):
    curvePoint = comb(n,i) * t ** i * (1 - t) ** (n - i)
    return curvePoint
```

```
def bf_bezier_curve(x, y, cell_amount):
    t = np.linspace(0,1,cell_amount)
    n = len(x)

    array = np.array([compute_bezier_curve(n - 1, i, t) for i in range(0, n)], dtype=np.float64)

    xBezier = np.dot(x, array)
    yBezier = np.dot(y, array)

    bezier_curve = np.vstack((xBezier, yBezier)).T

    return bezier_curve
```

3.5.2 dnc_bezier.py

```
def midpoint(points):
    npoints = len(points)
    n = npoints - 1
    coeffs = np.array([comb(n, i) for i in range(npoints)])
    result_x = np.sum(coeffs * points[:, 0]) / 2 ** n
```

```
    result_y = np.sum(coeffs * points[:, 1]) / 2 ** n
    return np.array([result_x, result_y])
```

```
def dnc_bezier_curve(points, npoints, iteration):
    if iteration == 0 or npoints == 2:
        return np.array([points[0], points[-1]])
    else:
        left_array = np.array([midpoint(points[:1])])
        right_array = np.array([midpoint(points[0:])])
        for i in range(1, npoints):
            left_array = np.append(left_array, [midpoint(points[:i+1])], axis=0)
            right_array = np.append(right_array, [midpoint(points[i:])], axis=0)

        left_curve = dnc_bezier_curve(left_array, npoints, iteration - 1)
        right_curve = dnc_bezier_curve(right_array, npoints, iteration - 1)

        return np.concatenate((left_curve, right_curve[1:]))
```

3.5.3 input_reader.py

```
def getNPoints():
    while True:
        try:
            npoints = int(input("Masukkan jumlah titik yang akan digunakan: "))
            if npoints < 2:
                print("Banyaknya titik yang digunakan harus lebih dari satu!")
            else:
                return npoints
        except:
            print("Masukkan angka yang benar!")
```

```
def setPoints(position, npoints):
    i = 0

    while i < npoints:
        try:
            print(f"Masukkan posisi titik ke-{i+1} : ", end="")
            current_point = np.array(list(map(np.float64, input().strip().split())))
            if len(current_point) != 2:
                raise Exception
            position[i] = current_point
            i += 1
        except:
            print("Masukan posisi Anda salah (dalam format x y, contoh: 1 2)")
```

```
def getNIterations():
    while True:
        try:
            iteration = int(input("Banyak iterasi yang dilakukan: "))
```

```
            if iteration < 0:
                print("Masukkan angka bernilai positif!")
            else:
                return iteration
        except:
            print("Masukkan angka!")
```

```
def getMethod():
    while True:
        print("|==== Metode yang dapat dipilih ====|")
        print("| 1. Divide and Conquer            |")
        print("| 2. Brute-force                   |")
        print("|==================================|")
        try:
            method = int(input("Metode penyelesaian yang dipilih "))
            if method != 1 and method != 2:
                print("Masukkan angka sesuai dengan penomoran metode!")
            else:
                return method
        except:
            print("Masukkan angka!")
```

3.5.4 visualize_graph.py

```
def draw_graph(Positions, BezierPositions):
    plt.cla()
    plt.plot(Positions[:,0], Positions[:,1], "b--", marker="o", label="Linear equation")
    for i in range(len(Positions)):
        plt.annotate(f'({Positions[i,0]:.2f}, {Positions[i,1]:.2f})',
                     xy=(Positions[i,0], Positions[i,1]),
                     xytext=(10, 10),
                     textcoords='offset pixels',
                     fontsize=7)
    plt.plot(BezierPositions[:,0], BezierPositions[:,1], color="red", label="Bezier curve")
    plt.legend()
    plt.tight_layout()
```

```
def animate_graph(positions, iterations, method):
    plt.figure(num="Bezier Curve")
    if method == 1:
        for i in range(iterations + 1):
            plt.suptitle(f"Divide and Conquer (iterasi {i})")
            start = time()
            BezierPositions = dnc_bezier_curve(positions, len(positions), i)
            print(f"Waktu proses iterasi {i} = ", time() - start, " detik")
            draw_graph(positions, BezierPositions)
            plt.pause(1.5)
    else:
```

```
    for i in range(iterations + 1):
        plt.suptitle(f"Brute-force (iterasi {i})")
        start = time()
        BezierPositions = bf_bezier_curve(positions[:,0], positions[:,1], 2 ** i + 1)
        print(f"Waktu proses iterasi {i} = ", time() - start, " detik")
        draw_graph(positions, BezierPositions)
        plt.pause(1.5)
```

3.5.5 main.py

```
def pBezierASCII():
    print("""=================================================================
 _______          _           ______
| ____ \\        (_)          /  __ \\
| |_/ / ____ ______  ____ _ ___  | /  \\/_   _ _ _____   ______
| ____ \\/ _ \\_  / |/ _ \\ '__| | |   | | | | '__\\ \ / / _ \\
| |_/ /  __// /| |  __/ |    | \\__/\\ |_| | |   \\ V /  __/
\\_____/ \\___/___|_|\\___|_|     \\_____/\\__,_|_|    \\_/ \\___|
=================================================================
        """)
```

```
def pMethodASCII(case):
    if case == 1:
print("""====================================================================================
=========
 ______ _     _   _                     _  _____          __
|  _  (_)    (_)  | |                  | | / __ \         / /
| | | |___   ___  __| | ___    __ _ _ __   __| | |/  \/ ___  _ __    / /_ _ _   _  ___ _ __
| | | | \ \ / / |/ _` |/ _ \  / _` | '_ \ / _` | | |    / _ \| '_ \  / / _` | | |/ _ \ '__|
| |/ /| |\ V /| | (_| |  __/ | (_| | | | | (_| | | \__/\ (_) | | | |/ / (_| | |_| |  __/ |
|___/ |_| \_/ |_|\__,_|\___|  \__,_|_| |_|\__,_|  \____/\___/|_| |_/_/ \__, |\__,_|\___|_|
                                                                  | |
                                                                  |_|
====================================================================================
===""")
    elif case == 2:
        print("====================================================")
        print("______          _       ______  ")
        print("| ___ \         | |      |  ___|  ")
        print("| |_/ /_ __ _   _| |_ ___  | |_ ___  _ __ ___ ___ ")
        print("| ___ \ '__| | | | __/ _ \ |  _/ _ \| '__/ __/ _ /")
        print("| |_/ / |  | |_| | ||  __/ | || (_) | | | (_|  __/")
        print("\____/|_|   \__,_|\__\___| \_| \___/|_|  \___\___|")
        print("====================================================")
```

```
def main():
    pBezierASCII()
    nPoints = getNPoints()

    positions = np.zeros((nPoints, 2), dtype=np.float64)
```

```
setPoints(positions, nPoints)

nIterations = getNIterations()

print()

method = getMethod()
if method == 1 :
    pMethodASCII(1)
else :
    pMethodASCII(2)

animate_graph(positions, nIterations, method)

plt.show()
```

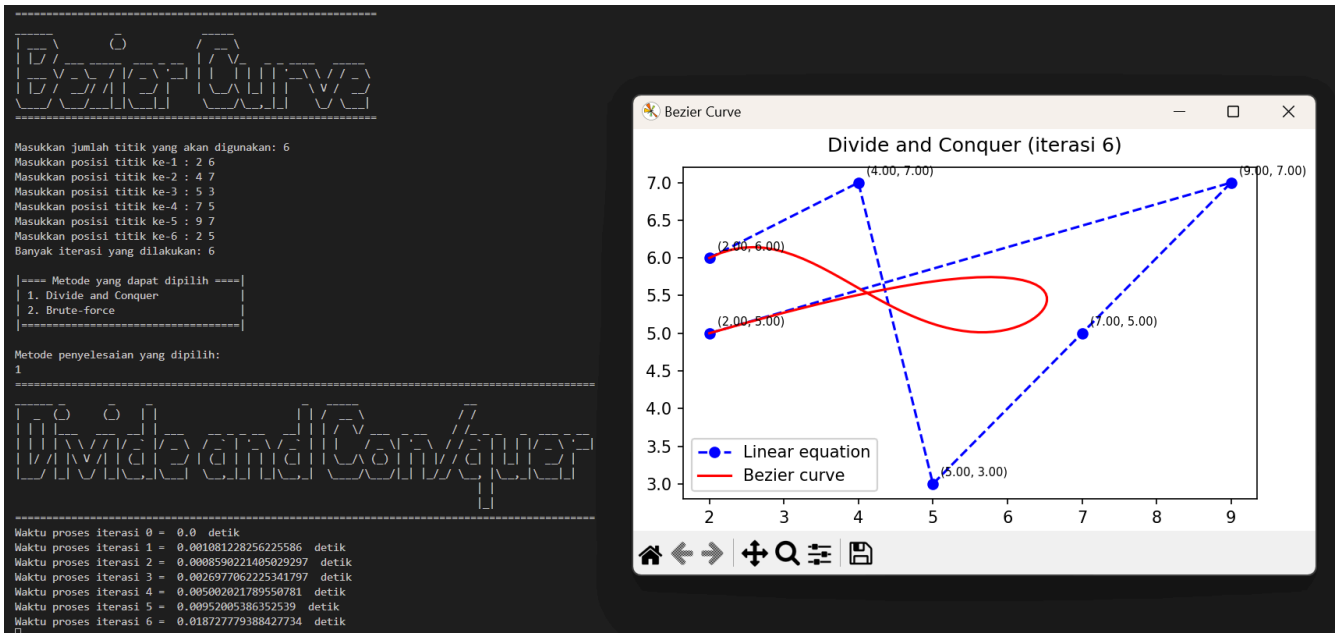
BAB 4

EKSPERIMEN

4.1 Divide & Conquer Algorithm

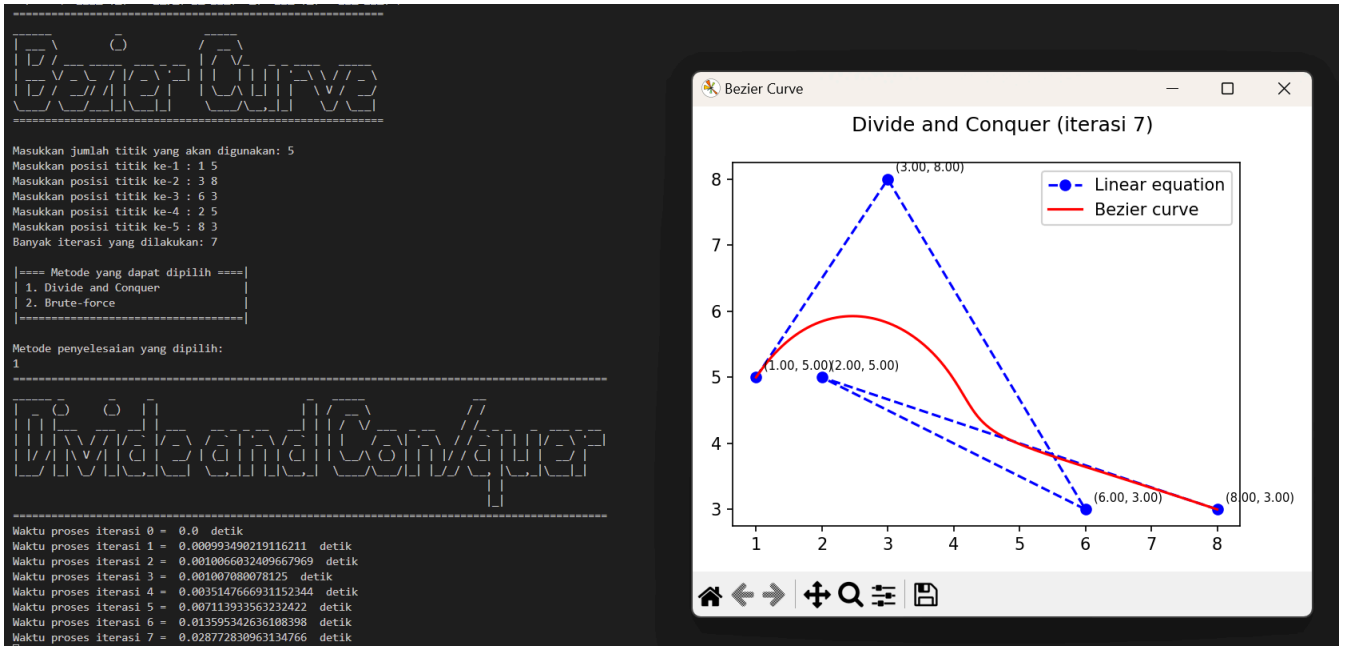
Jumlah Titik = 6

Jumlah Iterasi = 6



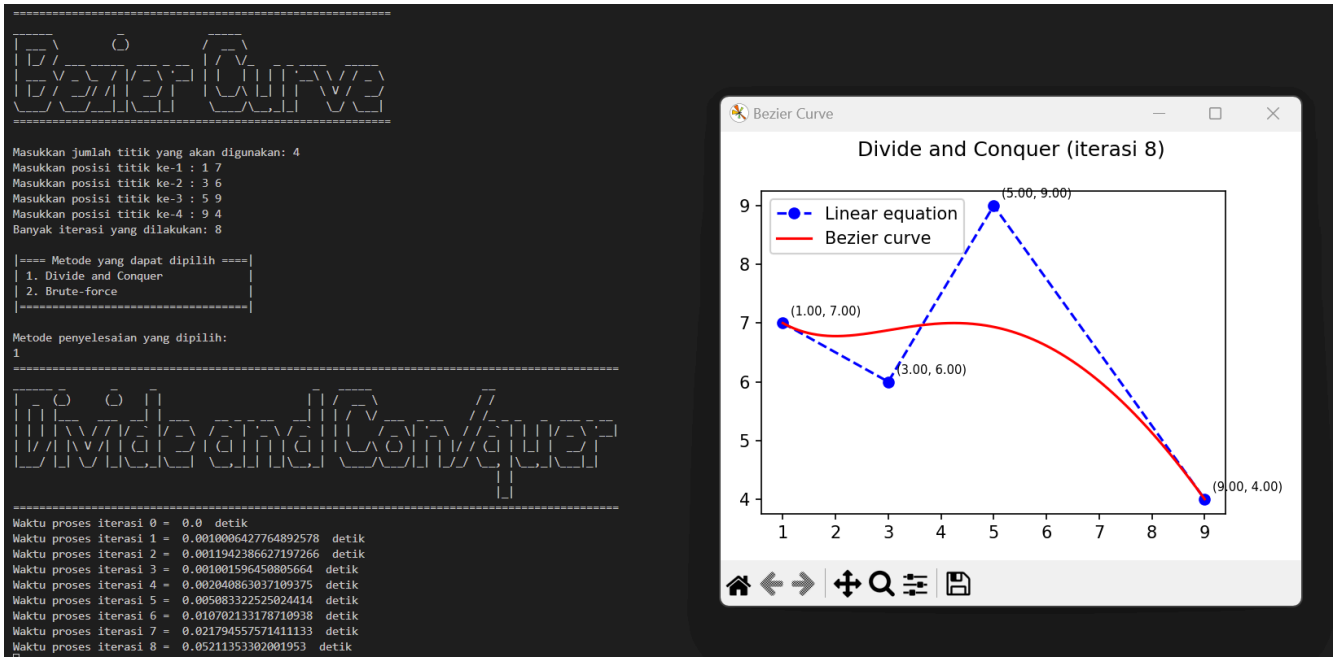
Jumlah Titik = 5

Jumlah Iterasi = 7



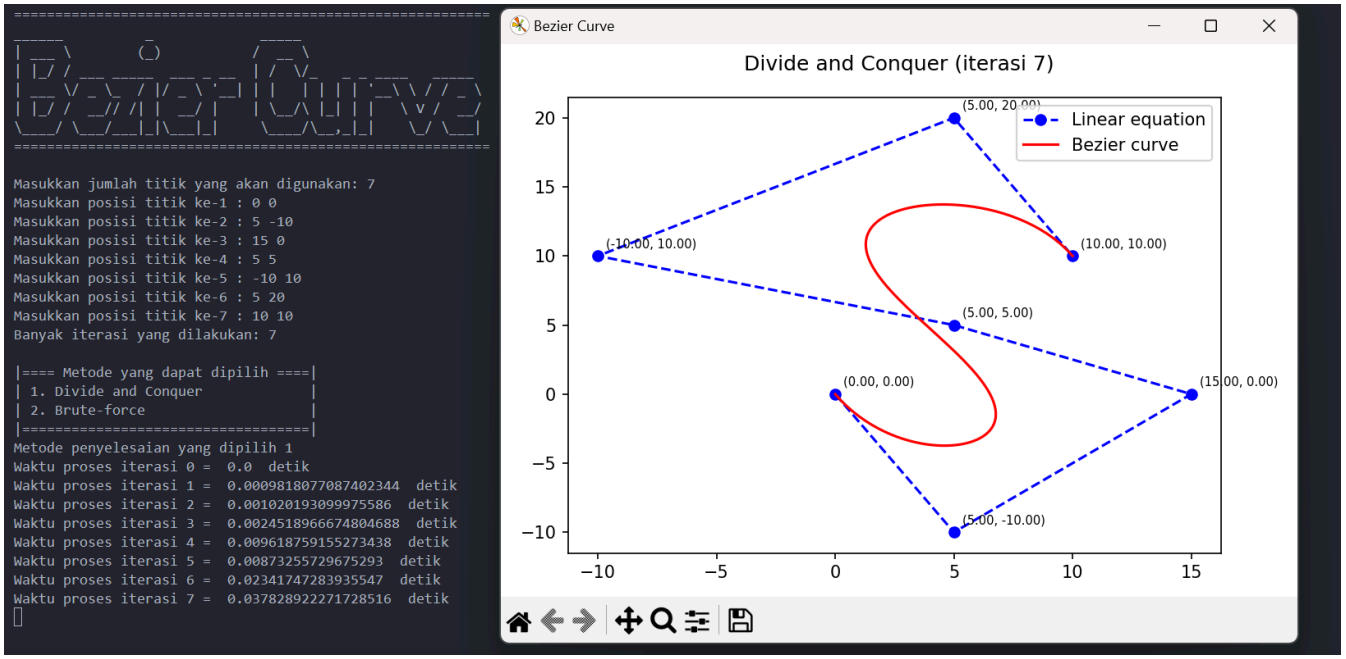
Jumlah Titik = 4

Jumlah Iterasi = 8



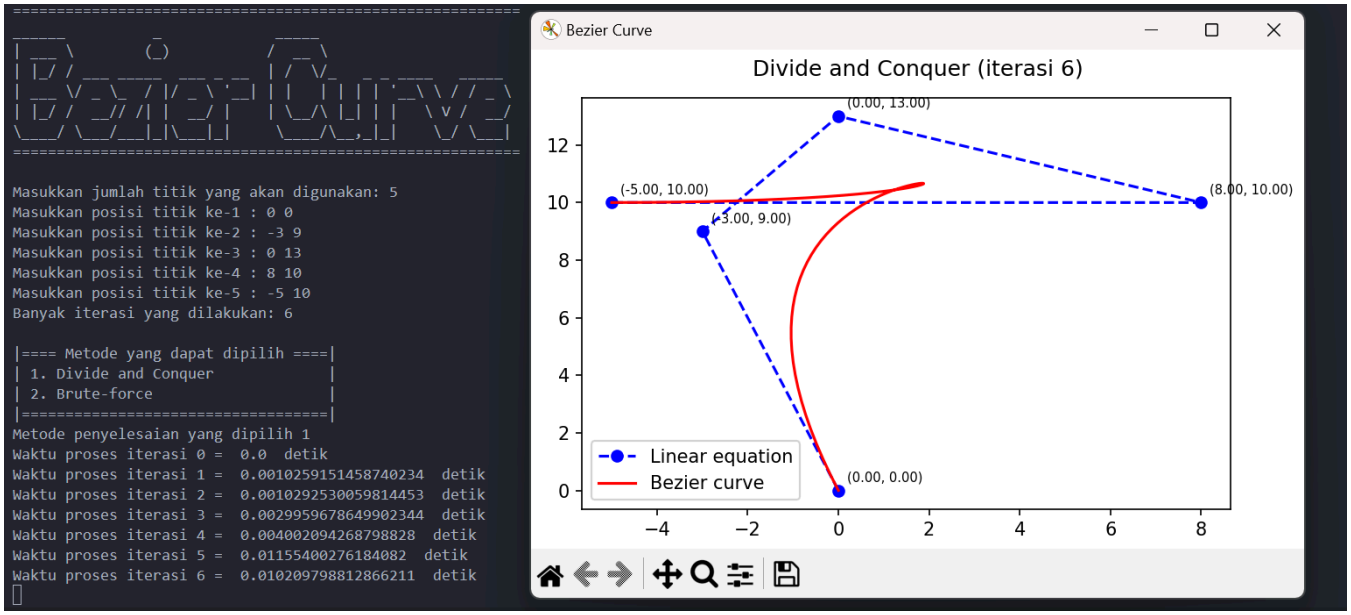
Jumlah Titik = 7

Jumlah Iterasi = 7



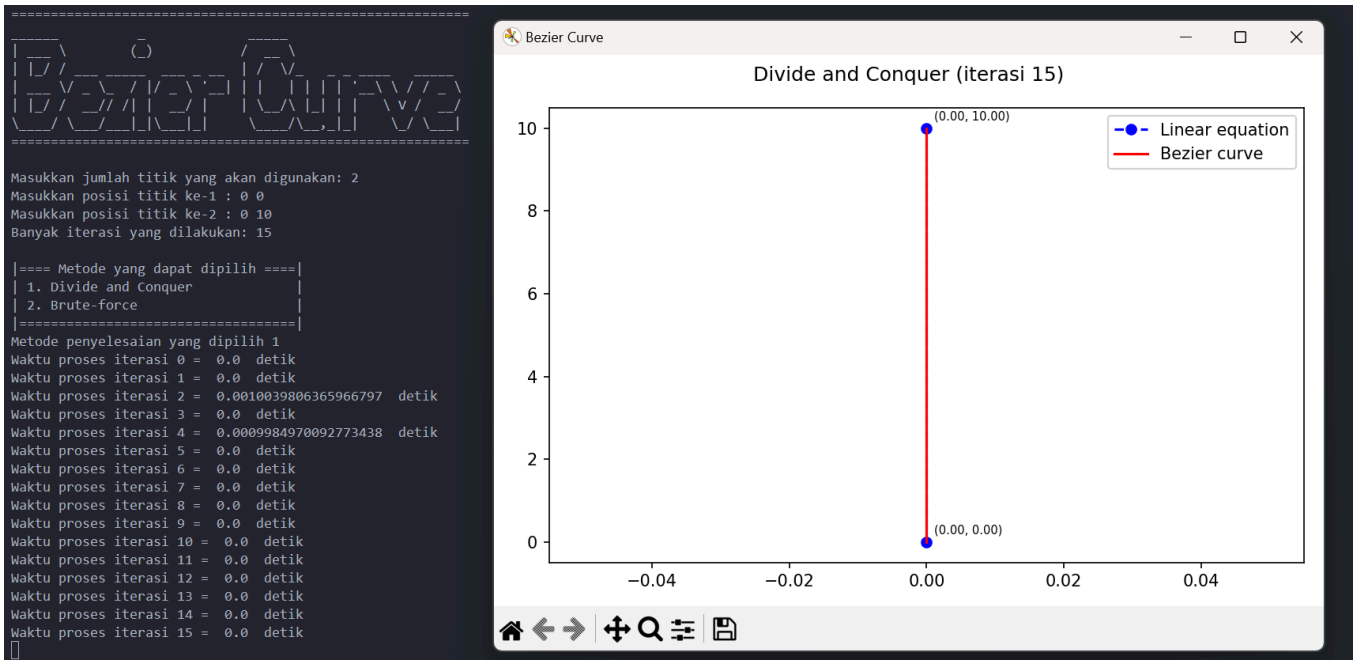
Jumlah Titik = 5

Jumlah Iterasi = 6



Jumlah Titik = 2

Jumlah Iterasi = 15

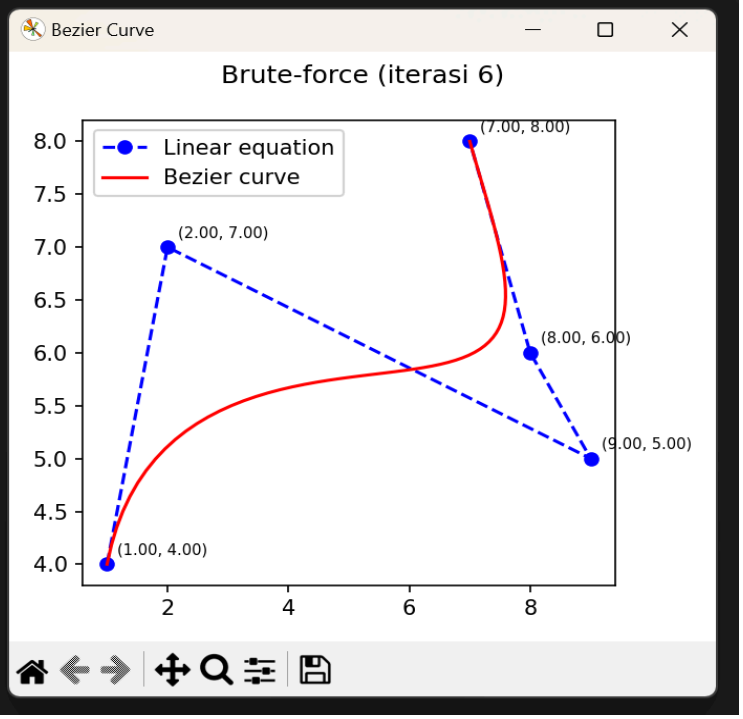


4.2 Brute Force Algorithm

Jumlah Titik = 5

Jumlah Iterasi = 6

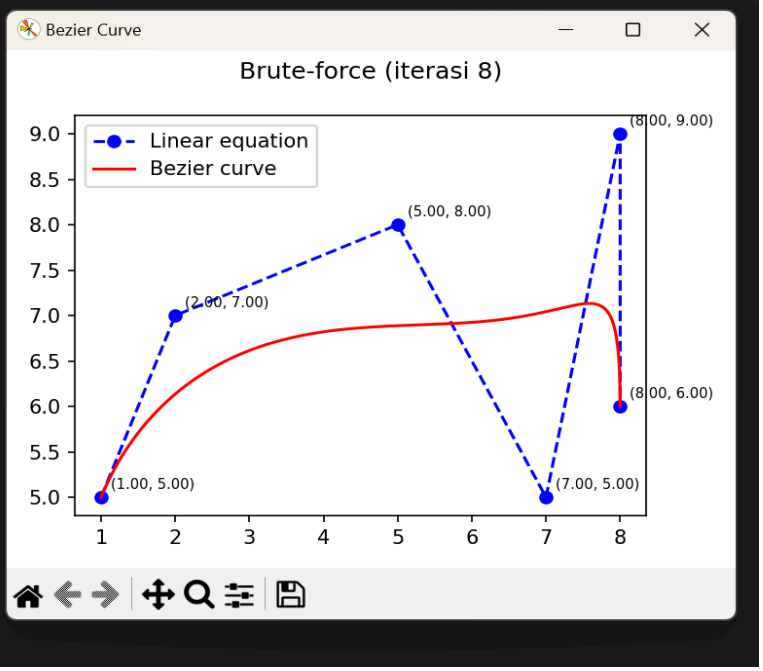
```
Masukkan jumlah titik yang akan digunakan: 5
Masukkan posisi titik ke-1 : 1 4
Masukkan posisi titik ke-2 : 2 7
Masukkan posisi titik ke-3 : 9 5
Masukkan posisi titik ke-4 : 8 6
Masukkan posisi titik ke-5 : 7 8
Banyak iterasi yang dilakukan: 6

|==== Metode yang dapat dipilih ====|
| 1. Divide and Conquer             |
| 2. Brute-force                    |
|===================================|

Metode penyelesaian yang dipilih:
2
================================================
Waktu proses iterasi 0 =  0.0  detik
Waktu proses iterasi 1 =  0.0  detik
Waktu proses iterasi 2 =  0.0  detik
Waktu proses iterasi 3 =  0.0  detik
Waktu proses iterasi 4 =  0.0  detik
Waktu proses iterasi 5 =  0.0009999275207519531  detik
Waktu proses iterasi 6 =  0.0  detik
```



Jumlah Titik = 6

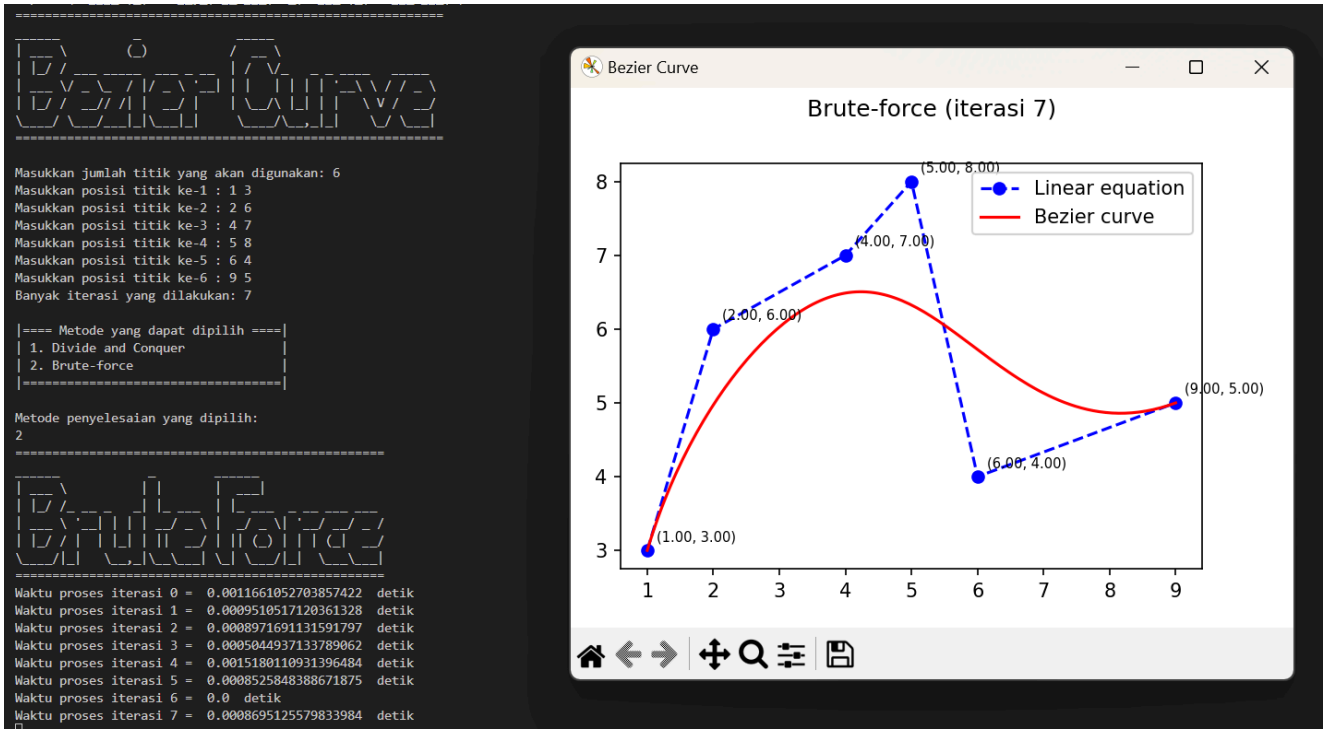
Jumlah Iterasi = 8

```
Masukkan jumlah titik yang akan digunakan: 6
Masukkan posisi titik ke-1 : 1 5
Masukkan posisi titik ke-2 : 2 7
Masukkan posisi titik ke-3 : 5 8
Masukkan posisi titik ke-4 : 7 5
Masukkan posisi titik ke-5 : 8 9
Masukkan posisi titik ke-6 : 8 6
Banyak iterasi yang dilakukan: 8

|==== Metode yang dapat dipilih ====|
| 1. Divide and Conquer             |
| 2. Brute-force                    |
|===================================|

Metode penyelesaian yang dipilih:
2
================================================
Waktu proses iterasi 0 =  0.0009186267852783203  detik
Waktu proses iterasi 1 =  0.0  detik
Waktu proses iterasi 2 =  0.0  detik
Waktu proses iterasi 3 =  0.0  detik
Waktu proses iterasi 4 =  0.0  detik
Waktu proses iterasi 5 =  0.0  detik
Waktu proses iterasi 6 =  0.0  detik
Waktu proses iterasi 7 =  0.0008709430694580078  detik
Waktu proses iterasi 8 =  0.0  detik
```



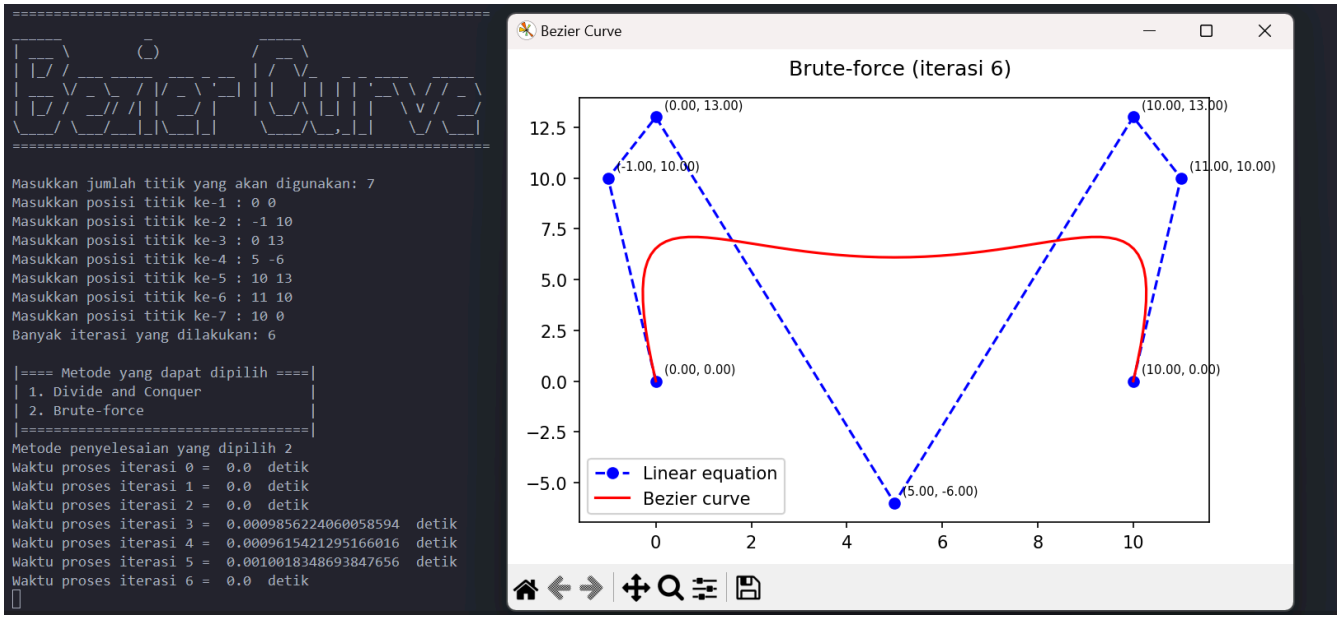
Jumlah Titik = 6

Jumlah Iterasi = 7



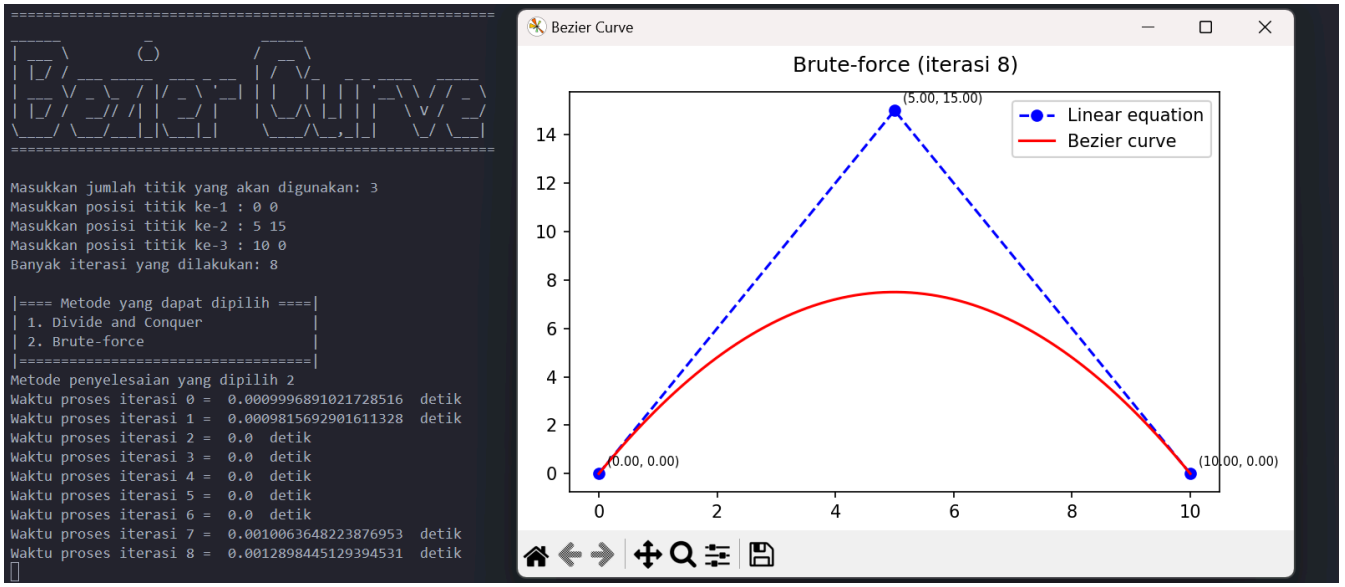
Jumlah Titik = 7

Jumlah Iterasi = 6



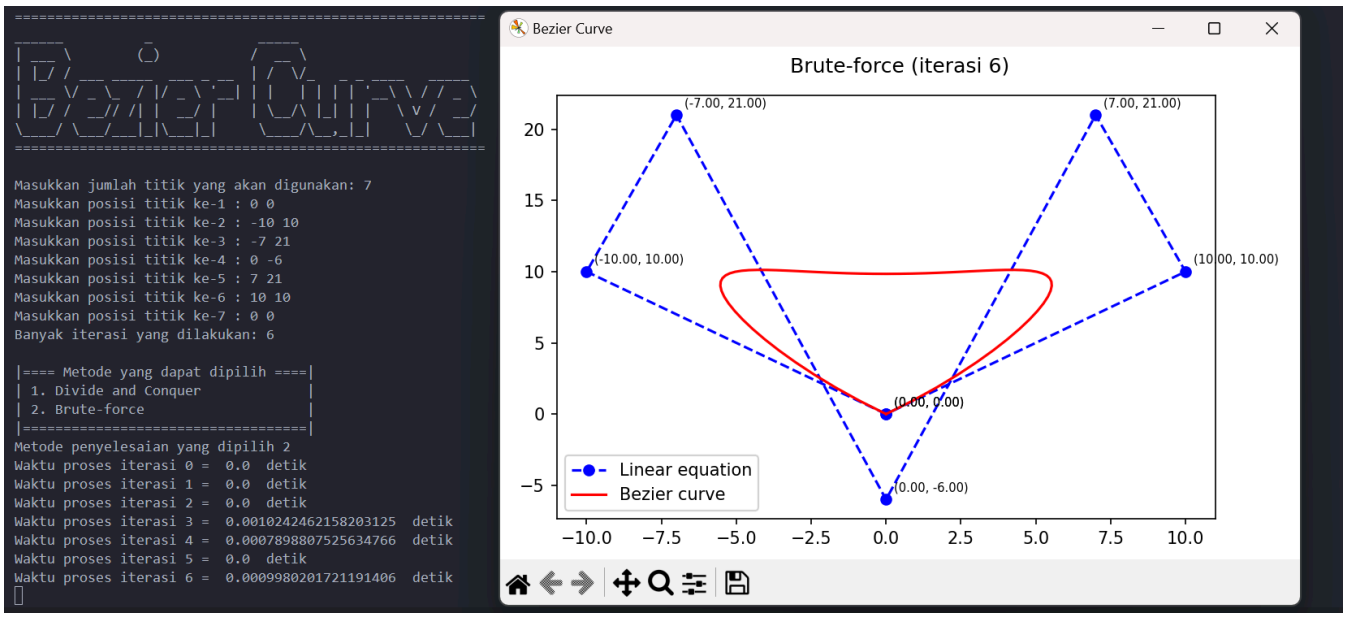
Jumlah Titik = 3

Jumlah Iterasi = 8



Jumlah Titik = 7

Jumlah Iterasi = 6



BAB 5

PENUTUP

5.1 Kesimpulan

Dalam mencari solusi kurva bezier yang optimal, dapat menggunakan algoritma *Brute Force* maupun algoritma *Divide and Conquer*. Dari analisis yang telah kami lakukan, didapat bahwa pencarian solusi kurva bezier yang menggunakan algoritma brute force memiliki waktu eksekusi yang lebih cepat dibandingkan dengan yang menggunakan algoritma divide & conquer. Perbedaan waktu eksekusi kedua solusi tersebut terlihat dengan jelas pada percobaan yang kami lakukan. Hal tersebut dapat terjadi karena banyak faktor yang mana telah kami jelaskan pada bagian bab 3 analisis dan implementasi.

5.2 Saran

Implementasi pencarian solusi kurva bezier yang kami buat masih dapat dikembangkan lebih lanjut. Dari segi performa, implementasi algoritma *divide and conquer* kami juga masih dapat dioptimalkan, seperti dengan menggantikan bahasa pemrograman yang digunakan.

DAFTAR PUSTAKA

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag2.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf

LAMPIRAN

LINK REPOSITORY

Link repository GitHub : https://github.com/Andhikafdh/Tucil2_13522128_13522160

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.		✓