Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Visualisasi Kurva Bezier dengan Algoritma Divide and Conquer

Disusun oleh:
Ahmad Farid Mudrika 13522008
Muhamad Rifki Virziadeili Harisman 13522120

Daftar Isi

Daftar Gambar

A. Penjelasan algoritma pembentukan kurva Bezier

a. Menggunakan pendekatan bruteforce

Dalam upaya pembuatan kurva Bezier menggunakan algoritma *brute force*, secara garis besar adalah mencari jumlah titik yang akan digunakan, menghitung koordinat dari titik tersebut menggunakan rumus, lalu menggambarkannya.

Rumus yang digunakan untuk mencari titik-titik dalam kurva Bezier kuadratik adalah sebagai berikut:

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

dengan t menggambarkan seberapa jauh $B(t)$ dalam kurva , dan P_0, P_1, P_2 sebagai vektor koordinat titik. Misalnya ketika t=0.25, maka $B(t)$ adalah seperempat jalan di kurva.

Dengan rumus yang telah diberikan, kita hanya perlu menentukan berapa banyak titik referensi yang kita inginkan dalam kurva, lalu kita dapat membagi kurva sama rata dengan jumlah titik tersebut, lalu mencari koordinat titik-titik tersebut. Dikarenakan program ini dibuat dengan tujuan menjadi pembanding dari program yang sama dengan algoritma *divide and conquer*, setelah kita menerima jumlah iterasi, kita akan menghitung jumlah titik yang akan dihasilkan dari jumlah iterasi tersebut dalam algoritma *divide and conquer*, lalu menggunakannya untuk algoritma *brute force*.

Setelah jumlah titik yang diinginkan diketahui sebagai n, program akan mencari koordinat-koordinat titik di kurva, dengan$0,\ 1/(n-1),\ 2/(n-1),\ \dots,\ (n-1)/(n-1)$ sebagai t.

Setelah koordinat-koordinat titik disimpan dalam suatu *list*, program akan menggunakan kakas matplotlib untuk menggambarkan kurva Bezier yang dihasilkan.

b. Menggunakan pendekatan Divide and Conquer

Pada dasarnya pendekatan algoritma menggunakan divide and conquer adalah dengan membagi-bagi persoalan menjadi persoalan yang kecil, kemudian diselesaikan dan digabungkan lagi dalam bentuk solusi. Hal tersebut akan dicoba dalam tugas kecil stima 2 ini, mengingat pada pembuatan kurva bezier untuk titik kontrol yang semakin banyak, akan semakin banyak besar pula kompleksitas program.

Mari kita mengulas cara pembuatan kurva bezier lebih mendalam sebelum masuk ke cara pembuatannya (Dalam kasus 3 titik kontrol).

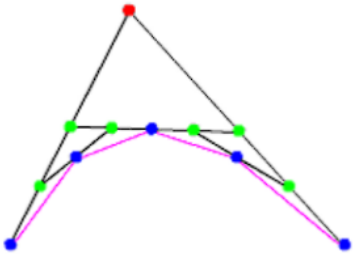
$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$

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

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

Asumsikan kita memiliki P_0, P_1, P_2 sebagai titik kontrol kurva. Q_0 merupakan titik tengah dari P_0 dan P_1, sementara Q_1 merupakan titik tengah dari P_1 dan P_2. Kemudian dari Q_0 dan Q_1 akan menghasilkan titik tengah R_0. Dengan melakukan substitusi persamaan 1 dan persamaan 2 ke persamaan 3 akan dihasilkan persamaan kurva bezier kuadratik yang sudah ditampilkan di penjelasan algoritma menggunakan pendekatan bruteforce.

Dari sini bisa kita amati bahwa sebetulnya persamaan kurva bezier kuadratik ini terbentuk dari persamaan yang lebih kecil dan selalu sama proses perhitungannya. Maka dengan memecah-mecah perhitungan pencarian koordinat R_0, kita selalu bisa mendapatkan nilai R_0 sesuai jumlah iterasi yang kita inginkan.



Gambar 1. Hasil pembentukan Kurva Bézier Kuadratik dengan divide and conquer setelah iterasi ke-2.

Dengan skema pembagian 2 proses (kiri dan kanan), kita akan mendapatkan sekumpulan koordinat koordinat R_0 yang kita inginkan. Setelah jumlah iterasi menjadi nol (pengurangan dalam proses rekursi) maka akan menemui basis yang menghasilkan P_0 dan P_n (dalam kasus ini P_2) sebagai titik kontrol awal dan akhir.

Dalam implementasinya, kami membuat suatu fungsi untuk mencari titik tengah antar titik kontrol maupun titik kontrol sementara. Fungsi ini bernama *get_Q*, ia akan mengeksekusi perhitungan terkecil dari pencarian R_0 dengan nilai $t = 0.5$.

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

Kemudian, dalam pembentukan kumpulan koordinat R_0 kami mengimplementasikan fungsi *curve*. Fungsi ini merupakan pemanggilan rekursif yang merupakan inti dari pendekatan algoritma menggunakan *divide and conquer*. Dalam fungsi ini akan terjadi pencarian koordinat R_0 dalam setiap iterasi dengan memanfaatkan *get_Q* untuk menghasilkan titik tengah yang diinginkan. Semakin banyak iterasi dilakukan maka pembagian kasus

menjadi lebih banyak dan lebih banyak pula R_0 yang dihasilkan. Ini membuat kurva yang dihasilkan akan semakin mulus.

Supaya algoritma ini dapat melakukan pembentukan kurva bezier dari n jumlah titik kontrol, maka perlu ditangani dengan method lain dalam pemanggilannya. Hal ini disebabkan oleh fungsi *curve* yang hanya dapat menghasilkan sekumpulan R_0 dari 3 titik kontrol saja. Oleh karena itu kami mengimplementasikan prosedur *draw* untuk skema pemanggilan *curve* sekaligus mengimplementasikan visualisasi hasil dari titik titik koordinat yang terbentuk.

Untuk skema pemanggilan *curve*, pertama kami membuat list kosong untuk menampung koordinat yang akan dihasilkan. Lalu kami akan melakukan pemanggilan fungsi *curve* dengan memasukkan titik untuk setiap tiga bergantian hingga semua titik telah melakukan pemanggilan fungsi curve.

Setelah itu, prosedur akan melakukan plotting garis dari koordinat yang telah dibentuk menggunakan *library matplotlib*. Kami juga mengimplementasikan animasi proses pembentukan kurva Bezier.

B. Source code program

a. Menggunakan pendekatan Bruteforce

```
class Bezier_bf(): # brute force
    def coordinates(self, points, iteration):# Brute force akan
membagi kurva menjadi jumlah iterasi, lalu menghitung nilai
titik di tiap bagian, lalu menghubungkannya
        x=[]
        y=[]
        if iteration<=1:
            x,y=zip(*points)
        else:
            for i in range (iteration):
                t= i / (iteration-1)
                 newx = (1 - t)**2 * points[0][0] + 2 * (1 - t)
* t * points[1][0] + (t)**2 * points[2][0]
                 newy = (1 - t)**2 * points[0][1] + 2 * (1 - t)
* t * points[1][1] + (t)**2 * points[2][1]
                x.append(newx);y.append(newy)
        return x,y

    def draw_coordinates(self, points, iteration):
        start=time.time()
            # a=iteration
               # for i in range (a): #supaya jumlah titik per
iterasi sama dengan DnC
            #     if i==0:
            #         iteration=3
            #     else:
            #         iteration=iteration*2 -1
            iteration=(2**iteration)+1#supaya jumlah titik per
iterasi same dengan DnC
        x,y = self.coordinates(points, iteration)
        end=time.time()-start
        print(list(zip(x,y)))
        plt.scatter(*zip(*points), color='red')
        plt.plot(*zip(*points), color='green')
        plt.plot(x, y)
        plt.title(f'Bezier Curve - Brute Force ({end} detik)')
        plt.show()
```

Gambar 2. Implementasi algoritma kurva Bezier dengan pendekatan bruteforce

b. Menggunakan pendekatan Divide and Conquer

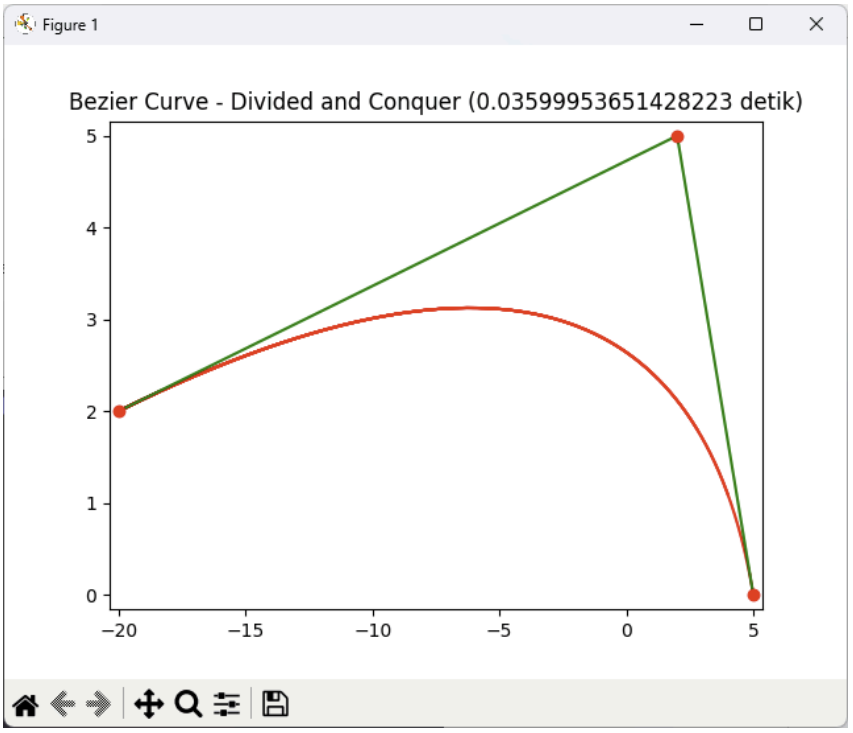
```
class Bezier_dac(): # devided and conquer
    def get_Q(point1, point2, t=0.5):
        return [(1-t)*x + t*y for x,y in zip(point1, point2)]

    def curve(p0, p1, p2, n):
        if (n==0):
            return [p0, p2]
        else:
            q0 = Bezier_dac.get_Q(p0,p1)
            q1 = Bezier_dac.get_Q(p1,p2)
            r0 = Bezier_dac.get_Q(q0,q1)
            curve1 = Bezier_dac.curve(p0, q0, r0, n-1)
            curve2 = Bezier_dac.curve(r0, q1, p2, n-1)
             # line = plt.plot([p0[0], p2[0]], [p0[1], p2[1]],
'ro-')
            # plt.pause(1)  # Jeda untuk animasi
            # line[0].remove()
            return curve1 + curve2[1:]

    def count_digit(n):
        count = 0
        while(n//10!=0):
            n//=10
            count+=1
        return count

    def draw(n, control_points, iteration):
        curve = []
        start = time.time()
        for i in range(n-2):
            print(i)
               curve+=(Bezier_dac.curve(*control_points[i:i+3],
iteration))

        print(curve)
        end = time.time() - start
        x = [x[0] for x in curve]
        y = [y[1] for y in curve]
        # print(x)
        xmax = max([x[0] for x in control_points])
        ymax = max([y[1] for y in control_points])
        xmin = min([x[0] for x in control_points])
        ymin = min([y[1] for y in control_points])
```

```
        # plt.figure()
        countx = (xmax-xmin)*0.01
        county = (ymax-ymin)*0.01
        if(len(x)>1000):
            n = 10**(Bezier_dac.count_digit(len(x))-1)
        elif(len(x)>100):
            n=5
        else:
            n=1
        for i in range(len(x)):
            if(i%n==0):
                countx +=0.01
                county +=0.01
                plt.xlim(xmin-countx,xmax+countx)
                plt.ylim(ymin-county,ymax+county)
                plt.plot(
                    x[:i+1],y[:i+1],'r-'
                )
                plt.pause(0.00001)
                in_end = i
        # print(in_end)
        plt.plot(x[in_end+1:-1],y[in_end+1:-1],'r-')
         plt.title(f'Bezier Curve - Divided and Conquer ({end}
detik)')
        plt.plot(
            [x[0] for x in control_points],
            [y[1] for y in control_points],
            'green',
        )
        plt.plot(
            [x[0] for x in control_points],
            [y[1] for y in control_points],
            'ro',
        )
        plt.show()
```

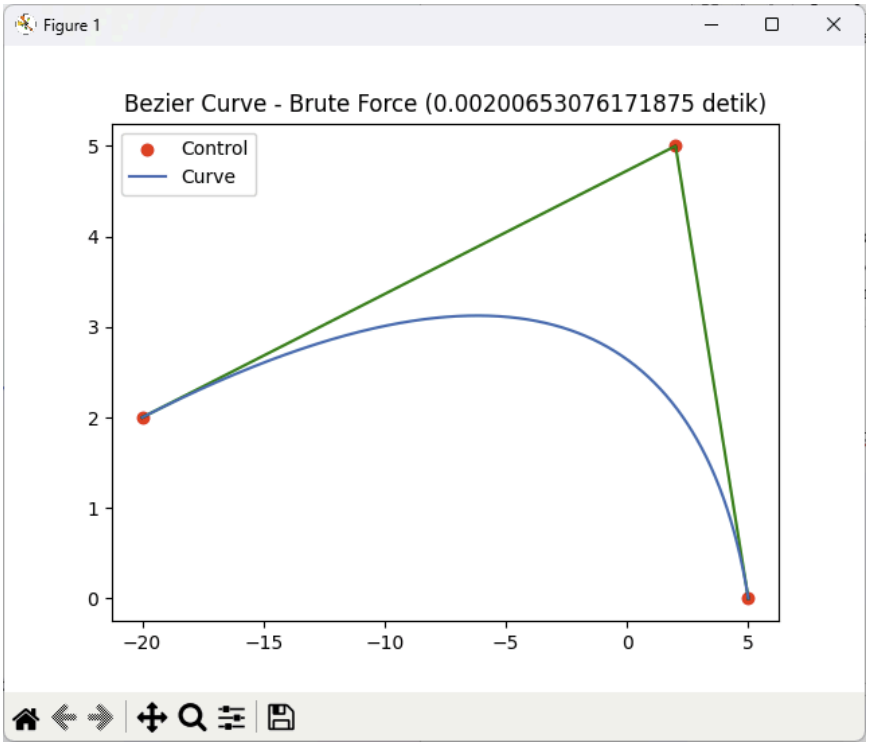
Gambar 3. Implementasi algoritma kurva Bezier dengan pendekatan divide and conquer

C. Hasil dalam tangkapan layar

a. Titik kontrol {(-20, 2), (2, 5), (5,0)}, iterasi 10

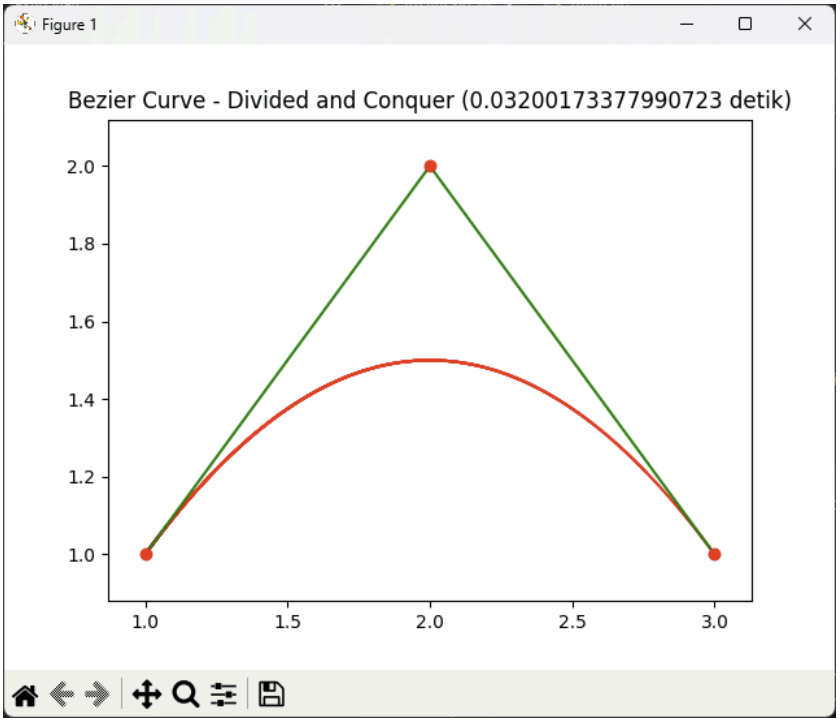


Gambar 4. Kurva Bezier menggunakan algoritma *divide and conquer*

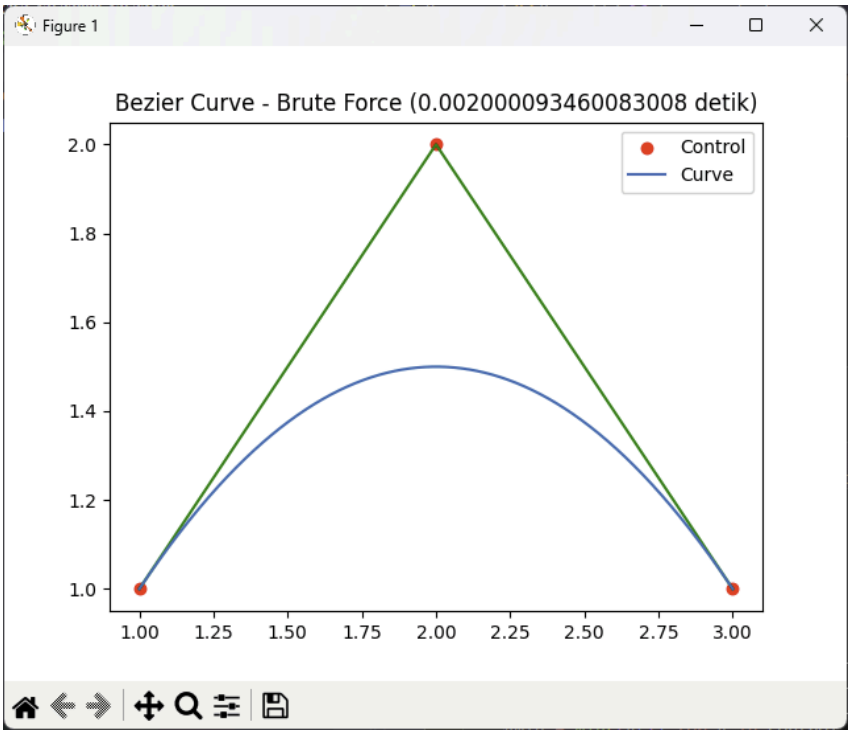


Gambar 5. Kurva Bezier menggunakan algoritma *brute force*

b. Titik kontrol {(1, 1), (2, 2), (3, 1)}, iterasi 10

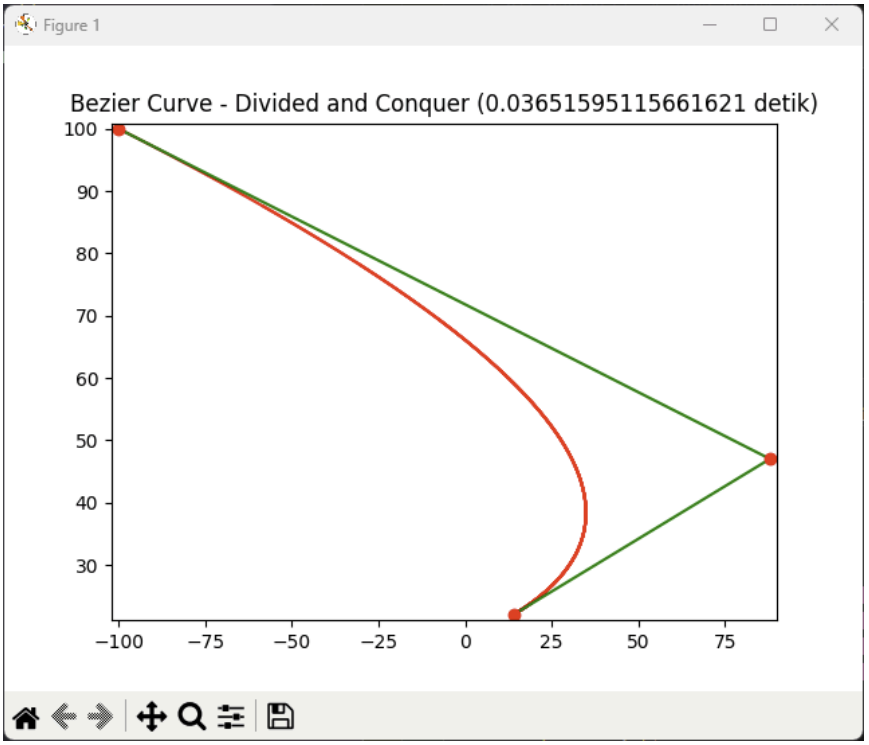


Gambar 6. Kurva Bezier menggunakan algoritma *divide and conquer*

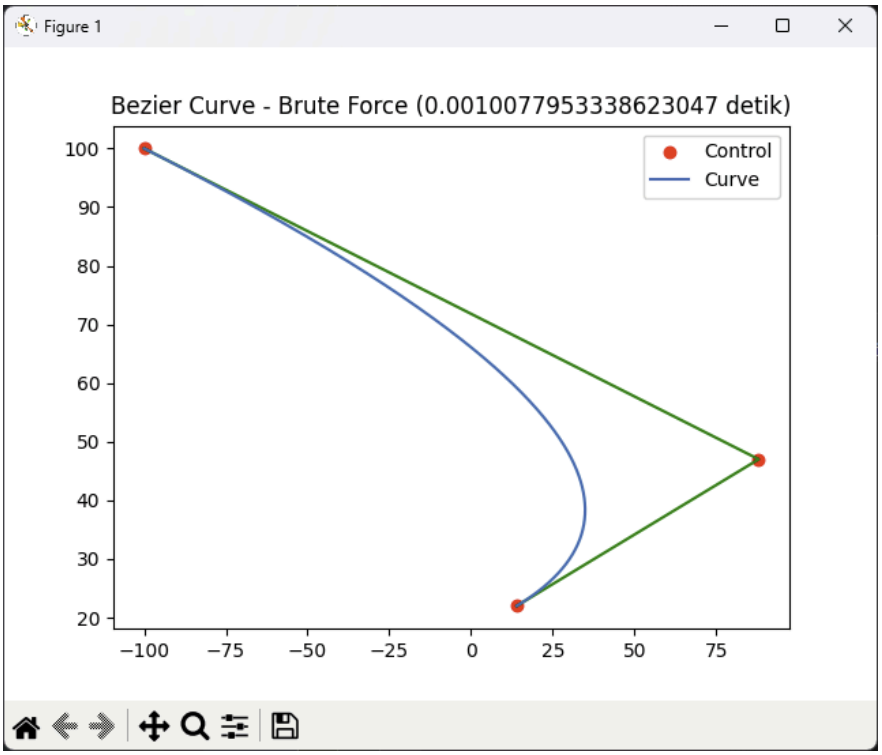


Gambar 7. Kurva Bezier menggunakan algoritma *brute force*

c. Titik kontrol {(14, 22), (88, 47), (-100,100)}, iterasi 10

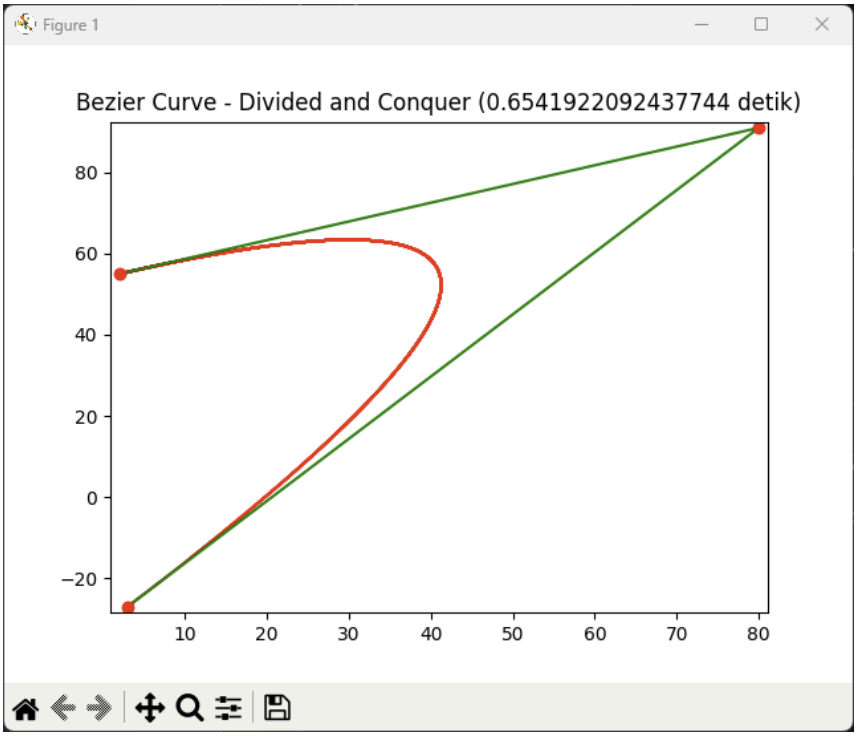


Gambar 8. Kurva Bezier menggunakan algoritma *divide and conquer*

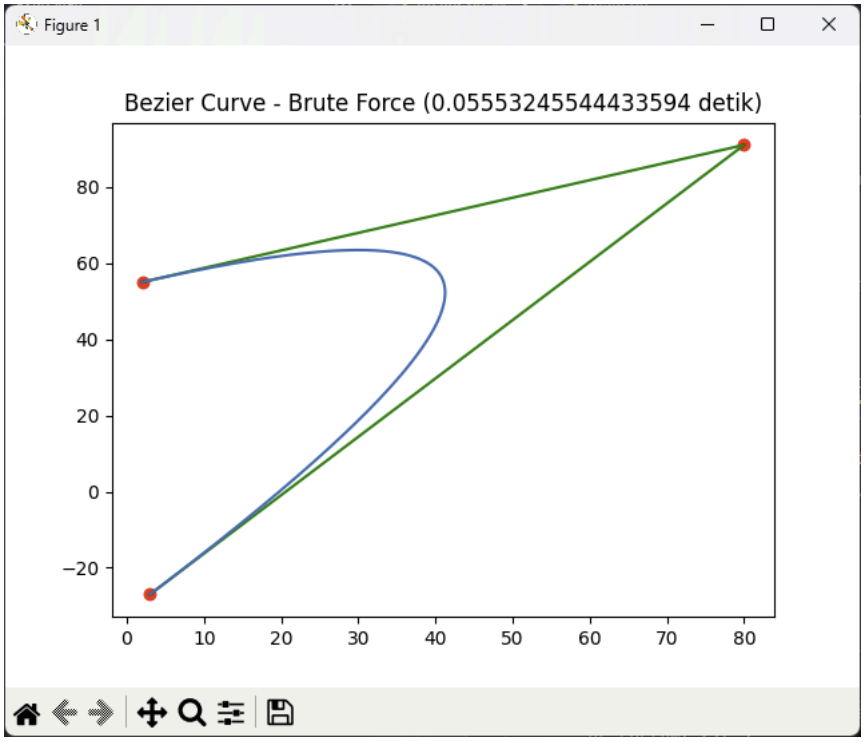


Gambar 9. Kurva Bezier menggunakan algoritma *brute force*

d. Titik kontrol {(2, 55), (80, 91), (3, -27)} , iterasi 15

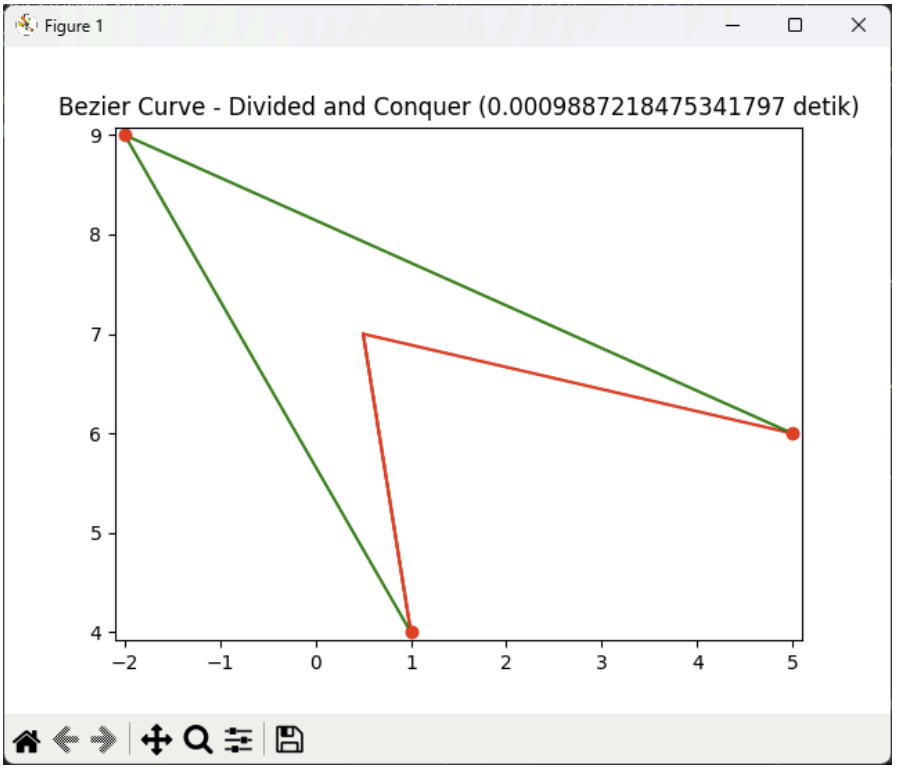


Gambar 10. Kurva Bezier menggunakan algoritma *divide and conquer*

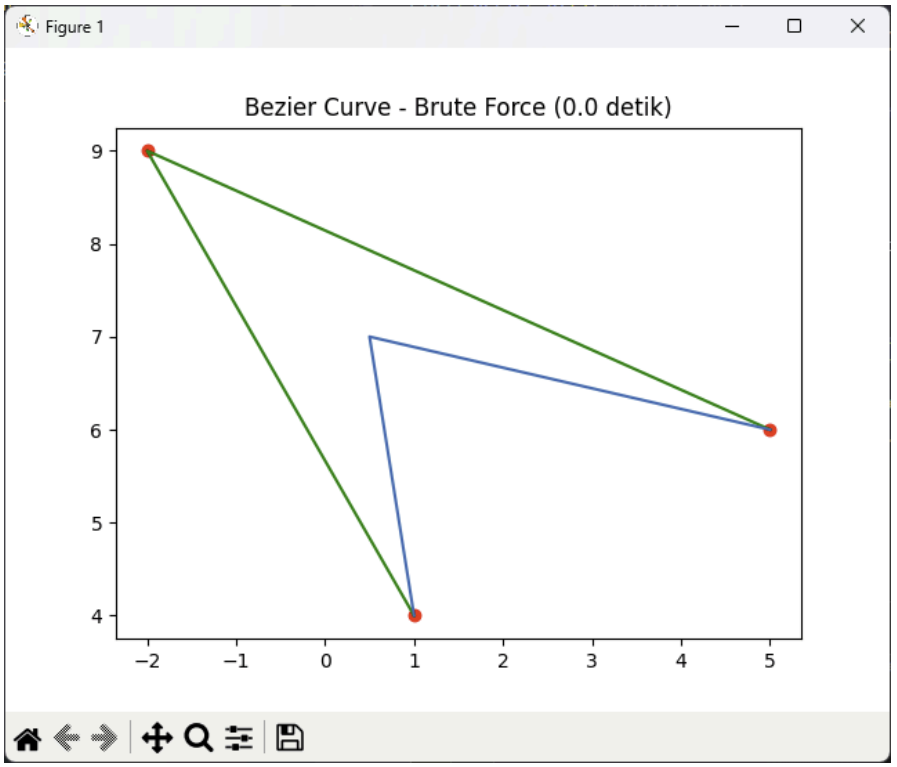


Gambar 11. Kurva Bezier menggunakan algoritma *brute froce*

e. Titik kontrol {(1, 4), (-2, 9), (5, 6)} , iterasi 1

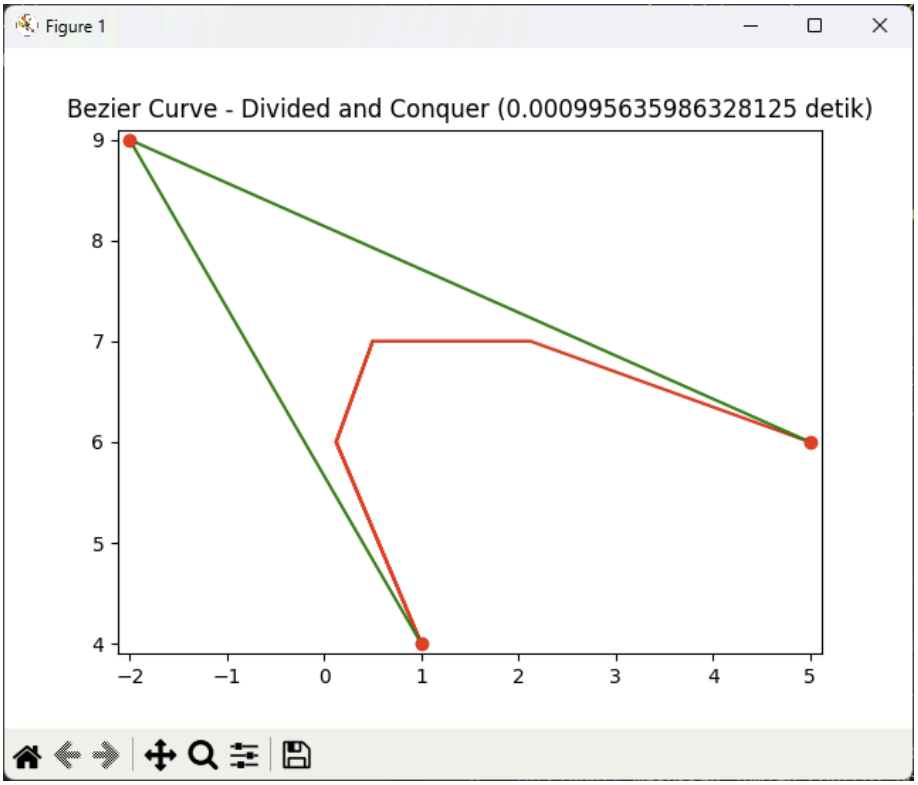


Gambar 12 Kurva Bezier menggunakan algoritma *divide and conquer*

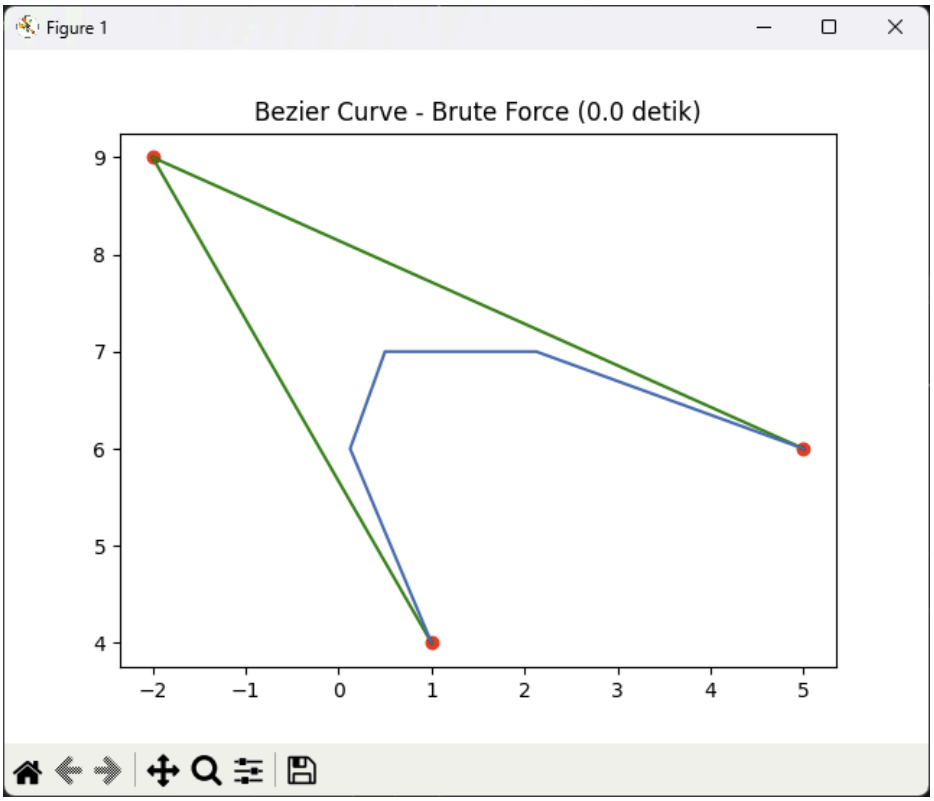


Gambar 13 Kurva Bezier menggunakan algoritma *brute force*

f. Titik kontrol {(1,4), (-2,9), (5,6)}, iterasi 2



Gambar 14. Kurva Bezier menggunakan algoritma *divide and conquer*



Gambar 15. Kurva Bezier menggunakan algoritma *brute force*

D. Analisis dan Pembahasan

Algoritma *divide and conquer* membagi kurva menjadi dua pada tiap iterasi. Sehingga, kompleksitas waktu yang didapat di di iterasi terakhir adalah $O(2^n)$. Ditambah dengan total semua iterasi sebelumnya sebesar $O(2^n)$. Maka, total kompleksitas waktu pada algoritma *divide and conquer* adalah $O(2^{n+1})$. Pada algoritma *brute force*, dilakukan $iteration = iteration^2 - 1$ sebelum dilakukan *loop* berbasis *iteration*, untuk mendapat hasil kurva yang sama dalam iterasi yang sama pada *divide and conquer*. Oleh karena itu, kompleksitas waktu dapat ditulis sebagai $O(2^n - 1) = O(2^n)$.

Sesuai data di tangkapan layar, didapat bahwa algoritma *brute force* malah menggunakan waktu yang lebih sedikit dari algoritma *divide and conquer*. Hal ini karena kompleksitas waktu dari algoritma *divide and conquer* lebih besar dari algoritma *brute force*. Algoritma *brute force* yang diimplementasikan hanya melibatkan mencari koordinat titik-titik dalam kurva menggunakan rumus yang telah diberikan. Di sisi lain, algoritma *divide and conquer* mencari semua titik di semua iterasi, yang menyebabkan kompleksitas waktu meningkat.

Link Repository Github:
https://github.com/rifchzschki/Tucil2_13522008_13522120

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	✓	