Tugas Kecil 1 IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Semester II Tahun 2023/2024

Disusun oleh

Shafiq Irvansyah	13522003
Muhammad Dava Fathurrahman	13522114

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

I. Deskripsi Tugas

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 mengenai 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.

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

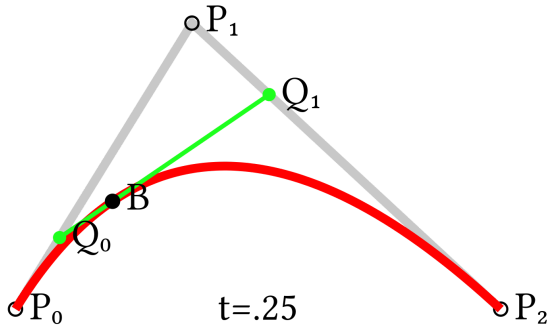
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

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

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

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

Berikut adalah ilustrasi dari kasus diatas.



Gambar 2. Pembentukan Kurva Bézier Kuadratik.
(Sumber: https://simonhalliday.com/2017/02/15/quadratic-bezier-curve-demo/)

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.

$$S_0 = B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3, \qquad t \in [0,1]$$

$$T_0 = B(t) = (1-t)^4 P_0 + 4(1-t)^3 t P_1 + 6(1-t)^2 t^2 P_2 + 4(1-t)t^3 P_3 + t^4 P_4, \qquad t \in [0,1]$$

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.

II. Landasan Teori

A. Algoritma Divide and Conquer

Divide and Conquer adalah sebuah paradigma algoritma yang umum digunakan dalam pemecahan masalah. Pendekatan ini menguraikan masalah yang kompleks menjadi sub-masalah yang lebih kecil, lebih mudah dipecahkan, dan kemudian menggabungkan solusi dari sub-masalah tersebut untuk membentuk solusi dari masalah asli. Ide dasar dari Divide and Conquer adalah untuk membagi masalah menjadi dua atau lebih bagian yang serupa dan independen secara rekursif, menyelesaikan masing-masing bagian secara terpisah, dan kemudian menggabungkan solusi-solusi tersebut. Langkah-langkah ini dapat diulang hingga mencapai kasus dasar yang langsung dapat dipecahkan. Ketika membagi masalah, langkah ini memastikan bahwa sub-masalah memiliki ukuran yang lebih kecil dan lebih mudah dikelola. Pada tahap pembagian ini, kompleksitas waktu algoritma sering kali meningkat hanya secara linier atau logaritmik terhadap ukuran masalah asli, yang membuatnya efisien. Setelah itu, solusi dari sub-masalah digabungkan dengan kompleksitas waktu yang relatif rendah, karena sub-masalah tersebut berukuran lebih kecil.

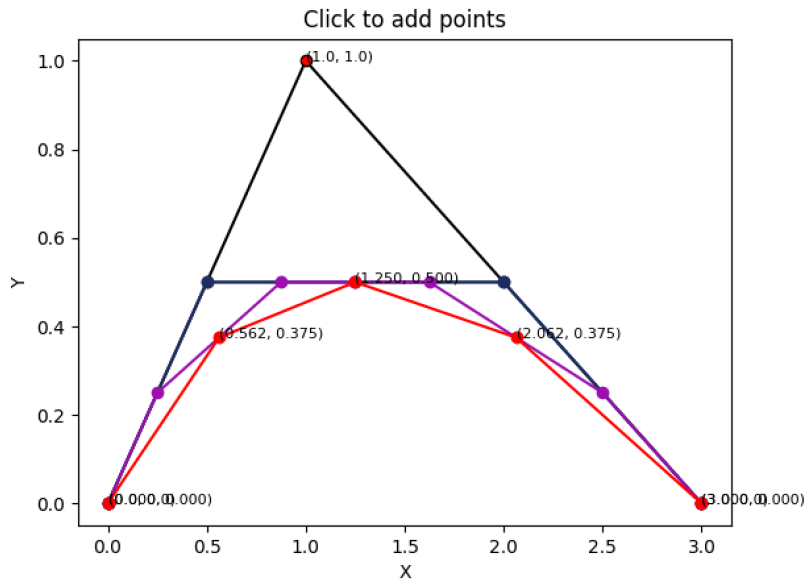
Pendekatan Divide and Conquer banyak diterapkan dalam berbagai bidang seperti algoritma pengurutan (seperti merge sort, quicksort), pencarian biner, perhitungan eksponensial (seperti perpangkatan cepat), dan optimisasi komputasi paralel. Keunggulan utama dari pendekatan ini adalah kemampuannya untuk menangani masalah yang kompleks dengan efisien dan memanfaatkan struktur rekursif untuk mencapai solusi yang optimal.

B. Algoritma Bruteforce

Algoritma Brute Force adalah metode pencarian solusi yang langsung dan sistematis untuk memecahkan suatu persoalan. Pendekatan ini berdasarkan pada deskripsi problematika dan penerapan definisi atau konsep yang terlibat tanpa memanfaatkan informasi kontekstual atau struktur yang mungkin ada. Algoritma ini secara eksplisit mencoba semua kemungkinan solusi, seperti uji coba seluruh kombinasi kata sandi, kunci enkripsi, atau input lainnya, tanpa memanfaatkan pengetahuan tambahan yang dapat mempercepat proses pencarian.

III. Implementasi Algoritma Divide and Conquer

A. Deskripsi Langkah



Gambar 3. Pembentukan Kurva Bezier Kuadratik dengan Iterasi kedua

Kurva bezier kuadratik di atas dapat dibentuk dengan langkah sebagai berikut,

1. Untuk 3 titik kontrol p_0, p_1, p_2, jadikan p_0 sebagai titik kontrol awal, p_2 sebagai titik kontrol akhir, dan p_1 sebagai titik kontrol antara
2. Hitung titik tengah setiap garis yang terletak di antara tiga titik kontrol. Untuk gambar di atas, akan didapatkan dua titik tengah. Sebut saja p_{01} untuk titik tengah antara p_0 dan p_1 dan p_{12} untuk titik tengah antara p_1 dan p_2.

3. Hitung titik tengah dari titik tengah yang baru dibuat sebelumnya, yaitu p_{01}
4. Hubungkan titik tengah yang baru dibuat, yaitu p_{01} dan p_{12} sehingga mendapatkan titik tengah baru bernama p_{0112}.
5. Jika kita menghubungkan p_0 - p_{0112} - p_2, kita akan mendapatkan kurva bezier kuadratik dengan iterasi kesatu.
6. Selanjutnya, kita akan membagi titik menjadi dua bagian, yaitu titik segmen kiri dan titik segmen kanan.
7. Dari iterasi kesatu, kita akan mendapatkan titik segmen kiri beranggotakan p_0, p_{01}, dan p_{0112} sedangkan titik segmen kanan beranggotakan p_{0112}, p_{12}, dan p_2.
8. Lakukan divide and conquer sesuai banyaknya iterasi dengan memanggil fungsi pembentuk kurva bezier secara rekursif dengan argumen titik segmen kiri dan titik segmen kanan sehingga menghasilkan titik tengah baru pada masing-masing sisi.
9. Dari titik tengah baru pada masing-masing sisi, akan dicari lagi titik tengah di antaranya sehingga menghasilkan titik penyusun kurva bezier.

Dengan melakukan rekursi ini, kurva Bezier kuadratik akan terbentuk secara iteratif hingga tingkat presisi yang diinginkan. Metode ini dapat digunakan untuk membuat kurva Bezier dengan kontrol yang lebih halus dan lebih kompleks dari yang mungkin dicapai dengan titik kontrol terbatas.

B. Kode Sumber

Kode sumber berikut hanya terdiri dari algoritma utama, tidak termasuk kode pembangun GUI.

```
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np
import time


# DATA
controlPoints = []
bezierPoints = []
lc = []
nIter = None

def getMidPoint(P1, P2):
    return ((P1[0] + P2[0]) / 2), ((P1[1] + P2[1]) / 2)


def quadraticBezier(ctrl1, ctrl2, ctrl3, limIter, currIter=0):
    if (currIter < limIter):
        midP1 = getMidPoint(ctrl1, ctrl2)
        midP2 = getMidPoint(ctrl2, ctrl3)

        midP3 = getMidPoint(midP1, midP2)

        # Left segment
```

```
        quadraticBezier(ctrl1, midP1, midP3, limIter, currIter+1)

        bezierPoints.append(midP3)

        # Right segment
        quadraticBezier(midP3, midP2, ctrl3, limIter, currIter+1)


def animateQuadraticBezier(ctrl1, ctrl2, ctrl3, limIter, currIter=0):
    if (currIter < limIter):
        midP1 = getMidPoint(ctrl1, ctrl2)
        midP2 = getMidPoint(ctrl2, ctrl3)

        midP3 = getMidPoint(midP1, midP2)

        ax.plot([ctrl1[0], midP1[0], midP2[0], ctrl3[0]], [ctrl1[1], midP1[1],
midP2[1], ctrl3[1]], color=lc[currIter], marker='o', linestyle='-')

        animateQuadraticBezier(ctrl1, midP1, midP3, limIter, currIter+1)

        bezierPoints.append(midP3)

        animateQuadraticBezier(midP3, midP2, ctrl3, limIter, currIter+1)

        fig.canvas.draw_idle()
        fig.canvas.start_event_loop(0.00005)


def NthBezier(controlPoints, limIter, currIter=0):
    if (currIter < limIter):
        midPoints = controlPoints.copy()

        left = [controlPoints[0]]
        right = [controlPoints[-1]]


        for i in range(len(controlPoints) -1):
            midMidPoints = []
            for j in range(1, len(midPoints)):
                midMidPoints.append(getMidPoint(midPoints[j], midPoints[j-1]))

            left.append(midMidPoints[0])
            right.append(midMidPoints[-1])

            midPoints = midMidPoints

        right = right[::-1]

        # LEFT SEGMENT
        NthBezier(left, nIter, currIter+1)

        bezierPoints.append(midPoints[0])

        # RIGHT SEGMENT
        NthBezier(right, nIter, currIter+1)
```

```
def animateNthBezier(controlPoints, limIter, currIter=0):
    if (currIter < limIter):
        midPoints = controlPoints.copy()

        left = [controlPoints[0]]
        right = [controlPoints[-1]]


        for i in range(len(controlPoints) -1):
            midMidPoints = []
            for j in range(1, len(midPoints)):
                midMidPoints.append(getMidPoint(midPoints[j], midPoints[j-1]))


            if i == 0:
                X = [controlPoints[0][0]] + [point[0] for point in midMidPoints] +
[controlPoints[-1][0]]
                Y = [controlPoints[0][1]] + [point[1] for point in midMidPoints] +
[controlPoints[-1][1]]
                ax.plot(X, Y, marker="o", markerfacecolor=lc[currIter],
color="black", ls="-")

            left.append(midMidPoints[0])
            right.append(midMidPoints[-1])

            midPoints = midMidPoints

        right = right[::-1]

        # LEFT SEGMENT
        animateNthBezier(left, nIter, currIter+1)

        bezierPoints.append(midPoints[0])

        # RIGHT SEGMENT
        animateNthBezier(right, nIter, currIter+1)
        fig.canvas.draw_idle()
        fig.canvas.start_event_loop(0.00005)


def runDNC():
    global isProcessing
    isProcessing = True

    BtnAddPoint.config(state='disabled')
    bezierPoints.clear()
    global nIter, lc

    nIter = int(EntryIteration.get())

    if (nIter > 0 and len(controlPoints) >= 2):
        lc = [(np.random.random(), np.random.random(), np.random.random()) for i
in range(nIter+1)]
```

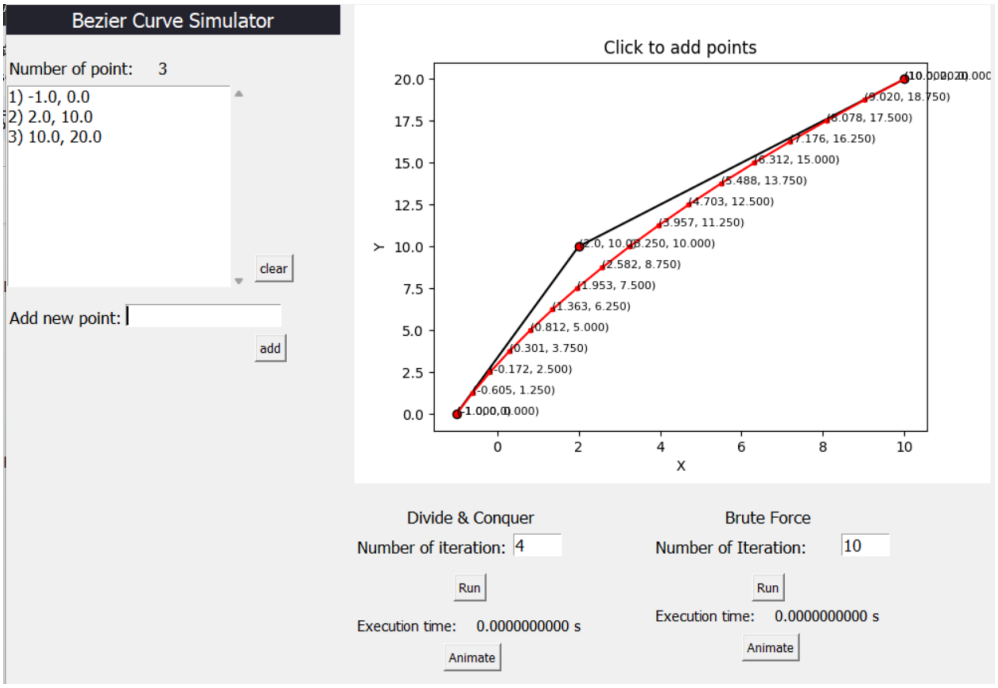
```
        start = time.time()

        if (len(controlPoints) == 3):
            bezierPoints.append(controlPoints[0])
            quadraticBezier(controlPoints[0], controlPoints[1], controlPoints[2],
nIter)
            bezierPoints.append(controlPoints[-1])
        else:
            bezierPoints.append(controlPoints[0])
            NthBezier(controlPoints, nIter)
            bezierPoints.append(controlPoints[-1])

        end = time.time()
        LabelDNCTimeVal.config(text=f"{(end - start):.10f} s")

        ax.clear()

        ax.plot([point[0] for point in controlPoints], [point[1] for point in
controlPoints], 'ko-', markerfacecolor='red')

        for point in controlPoints:
            ax.text(point[0], point[1], f'({point[0]}, {point[1]})', fontsize=8)

        ax.plot([point[0] for point in bezierPoints], [point[1] for point in
bezierPoints], 'r.-')
        for point in bezierPoints:
            ax.text(point[0], point[1], f'({point[0]:.3f}, {point[1]:.3f})',
fontsize=8)

        ax.set_title('Click to add points')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        fig.canvas.draw()

    BtnAddPoint.config(state='normal')

    isProcessing = False


def animateDNC():
    global isProcessing
    isProcessing = True

    BtnAddPoint.config(state='disabled')
    bezierPoints.clear()
    global nIter, lc
    nIter = int(EntryIteration.get())

    if (nIter > 0 and len(controlPoints) >= 2):
        lc = [(np.random.random(), np.random.random(), np.random.random()) for i
in range(nIter+1)]

        ax.clear()

        ax.plot([point[0] for point in controlPoints], [point[1] for point in
controlPoints], 'ko-', markerfacecolor='red')
```

```
        for point in controlPoints:
            ax.text(point[0], point[1], f'({point[0]}, {point[1]})', fontsize=8)

        if (len(controlPoints) == 3):
            for i in range(nIter):
                bezierPoints.append(controlPoints[0])
                animateQuadraticBezier(controlPoints[0], controlPoints[1],
controlPoints[2], i+1)
                bezierPoints.append(controlPoints[-1])

                if i != nIter-1:
                    bezierPoints.clear()
        else:
            for i in range(nIter):
                bezierPoints.append(controlPoints[0])
                animateNthBezier(controlPoints, i+1)
                bezierPoints.append(controlPoints[-1])

                if i != nIter-1:
                    bezierPoints.clear()

        ax.plot([point[0] for point in bezierPoints], [point[1] for point in
bezierPoints], 'ro-')
        for point in bezierPoints:
            ax.text(point[0], point[1], f'({point[0]:.3f}, {point[1]:.3f})',
fontsize=8)

        ax.set_title('Click to add points')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        fig.canvas.draw()


    BtnAddPoint.config(state='normal')

    isProcessing = False
```

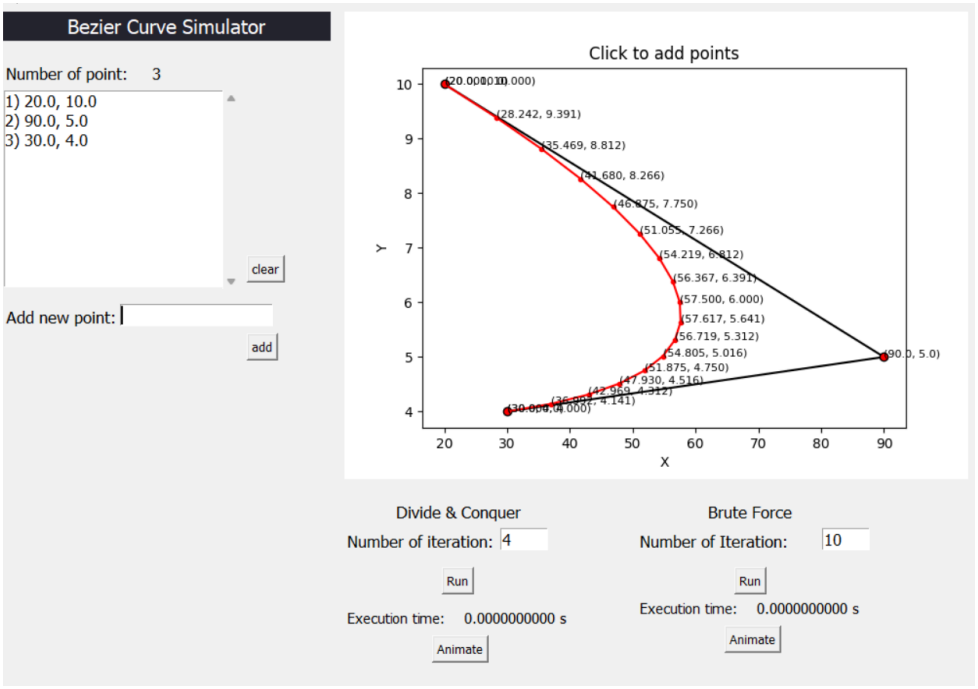
C. Pengujian

Test Case Kuadratik

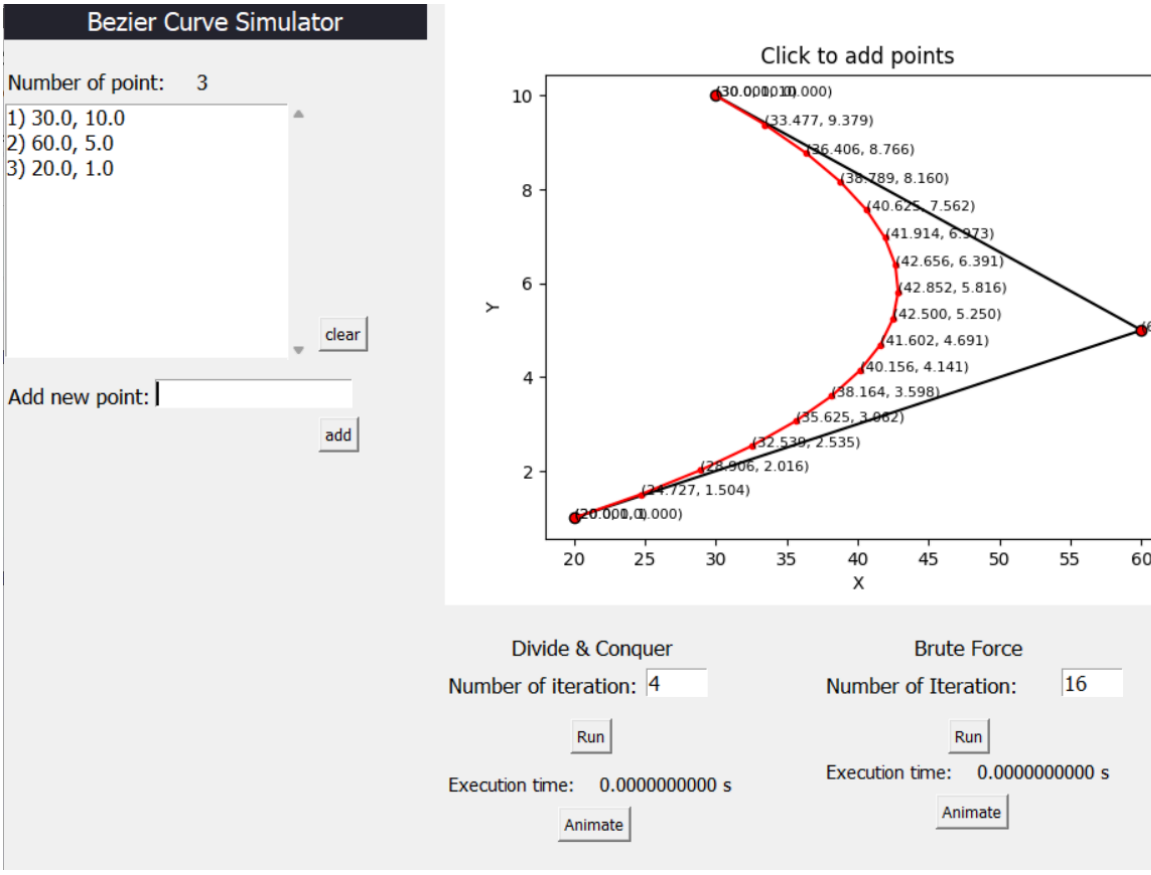
1. Test Case 1



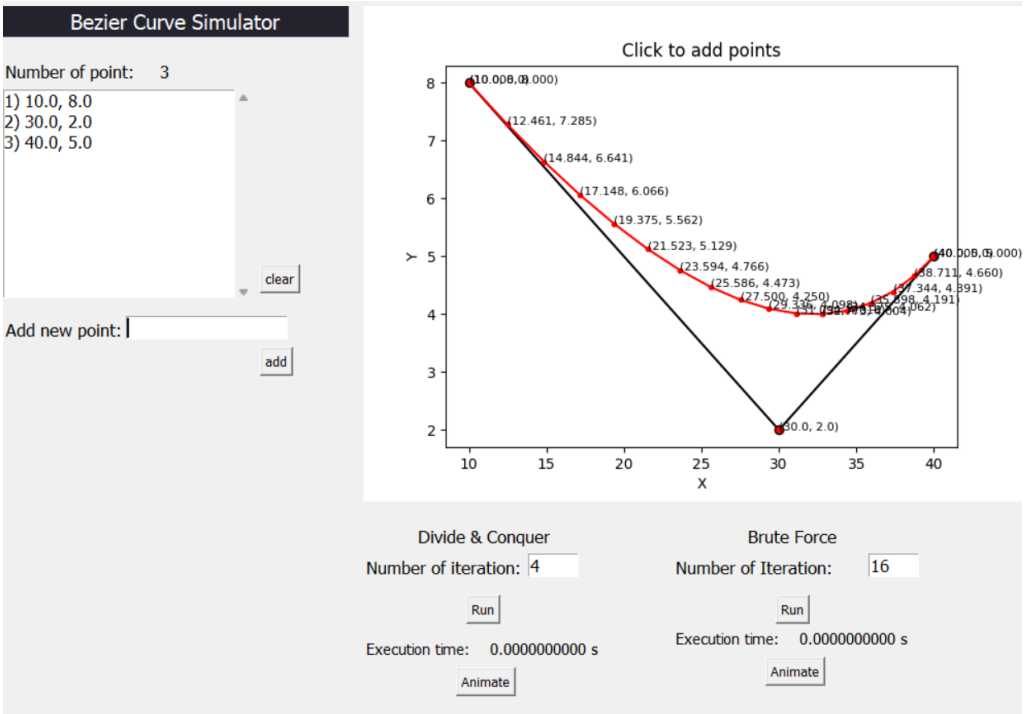
2. Test Case 2



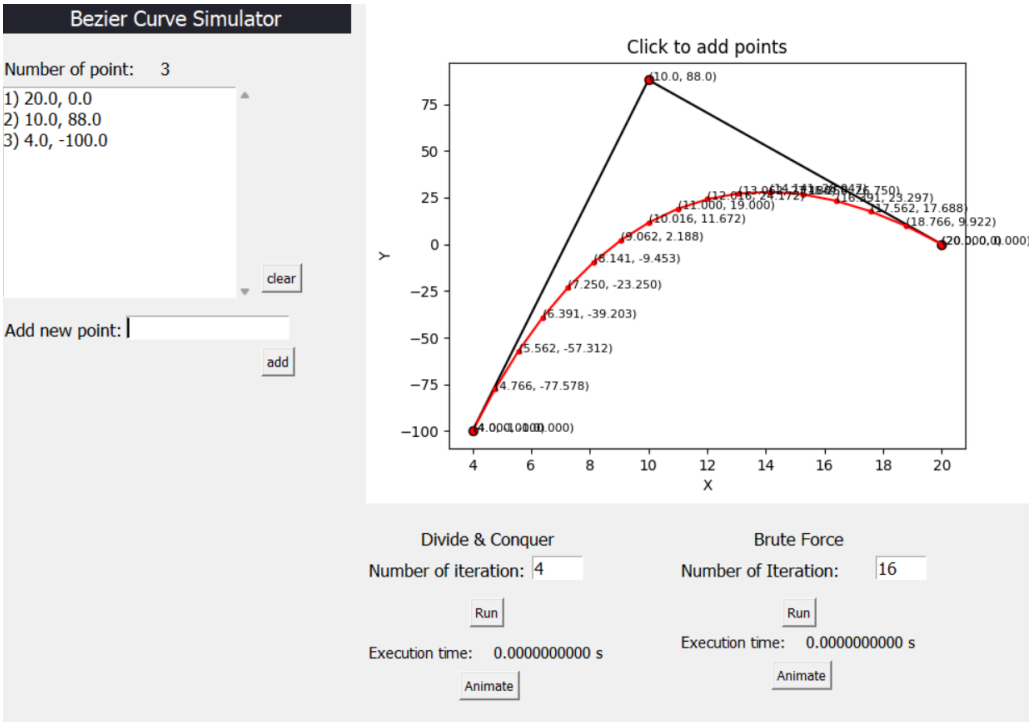
3. Test Case 3



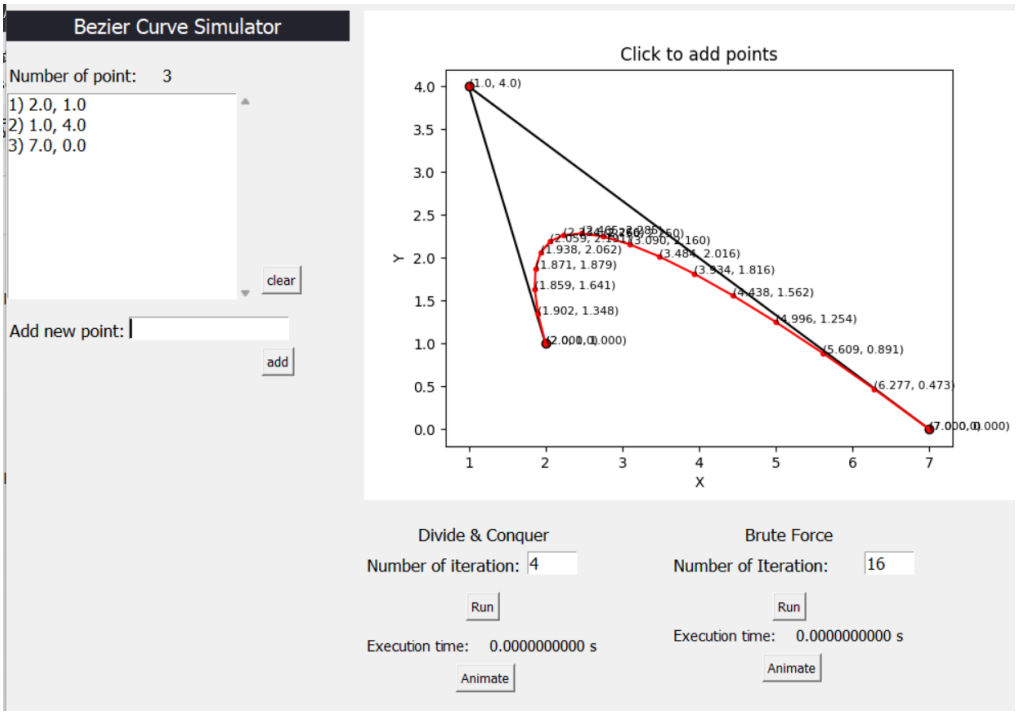
4. Test Case 4



5. Test Case 5

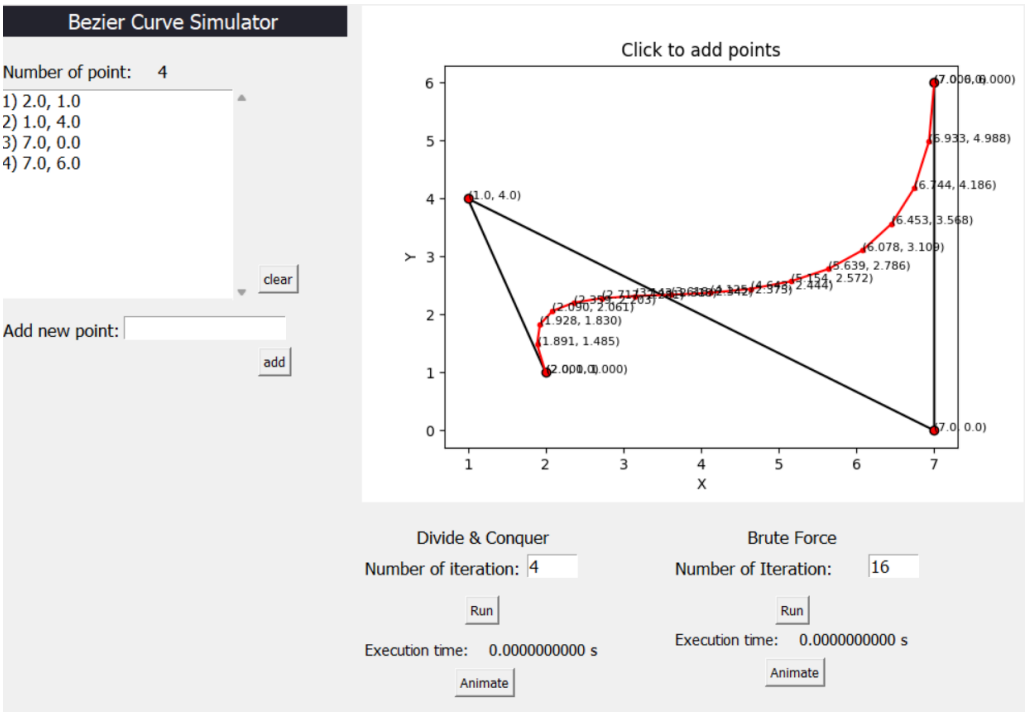


6. Test Case 6

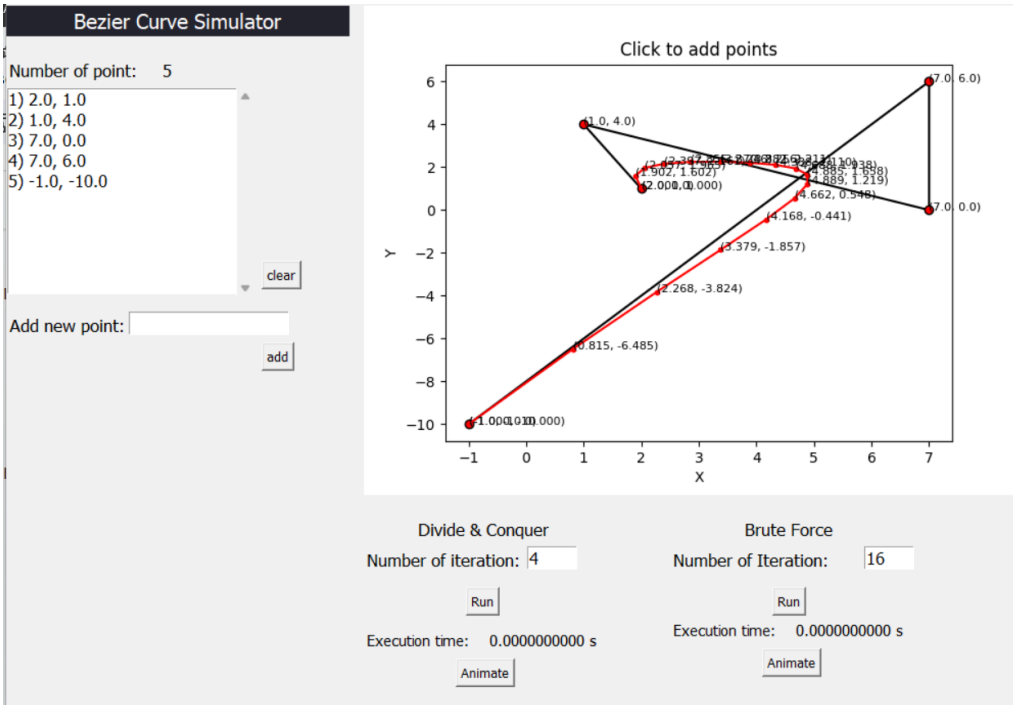


Test Case n > 3

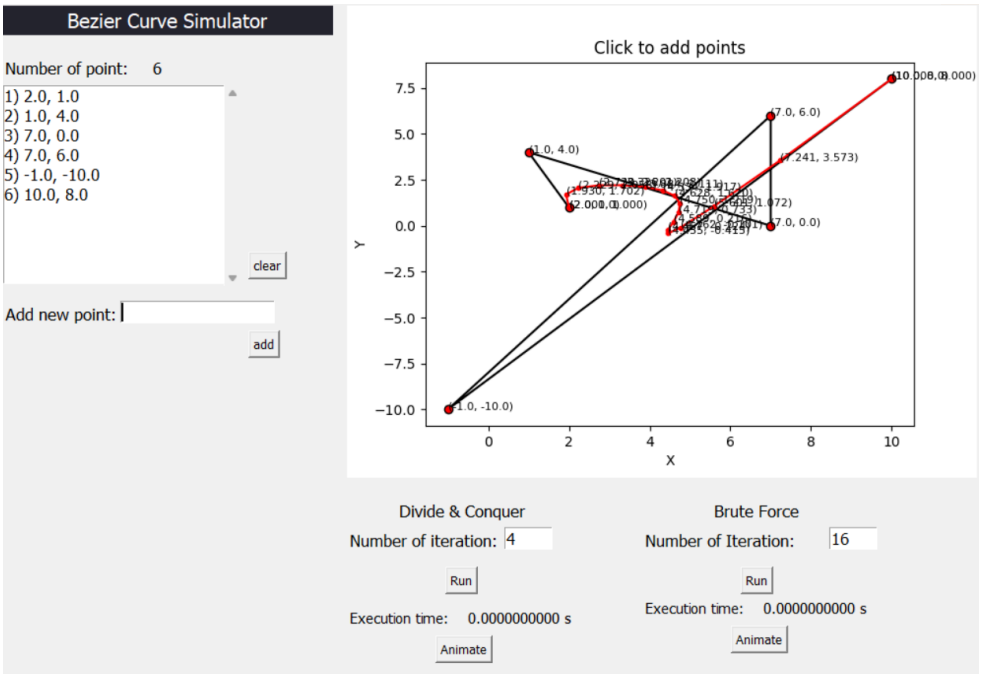
1. **Test Case 1**



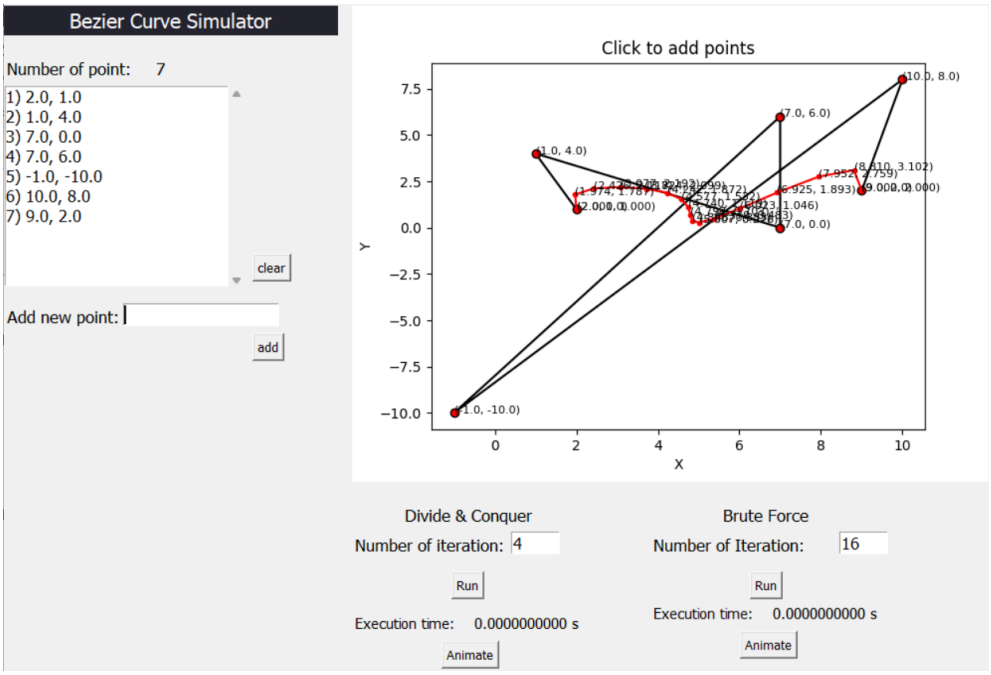
2. Test Case 2



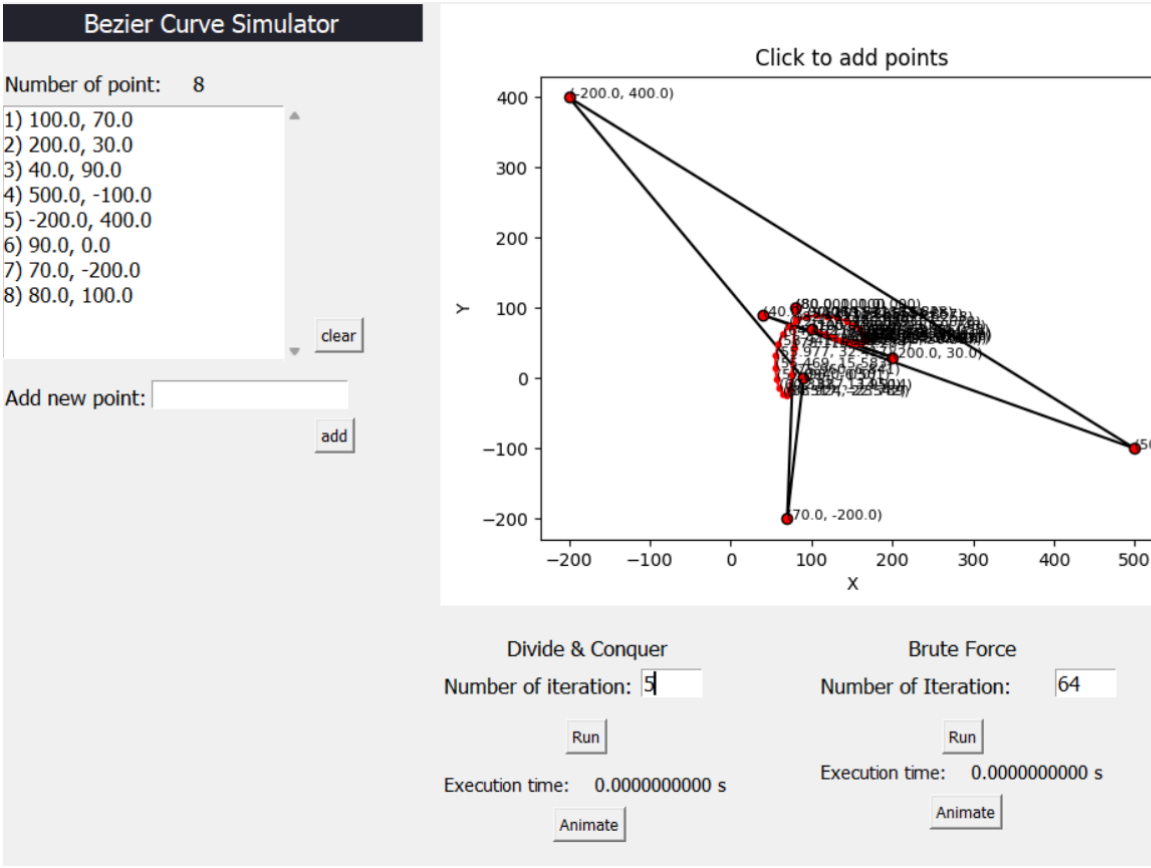
3. Test Case 3



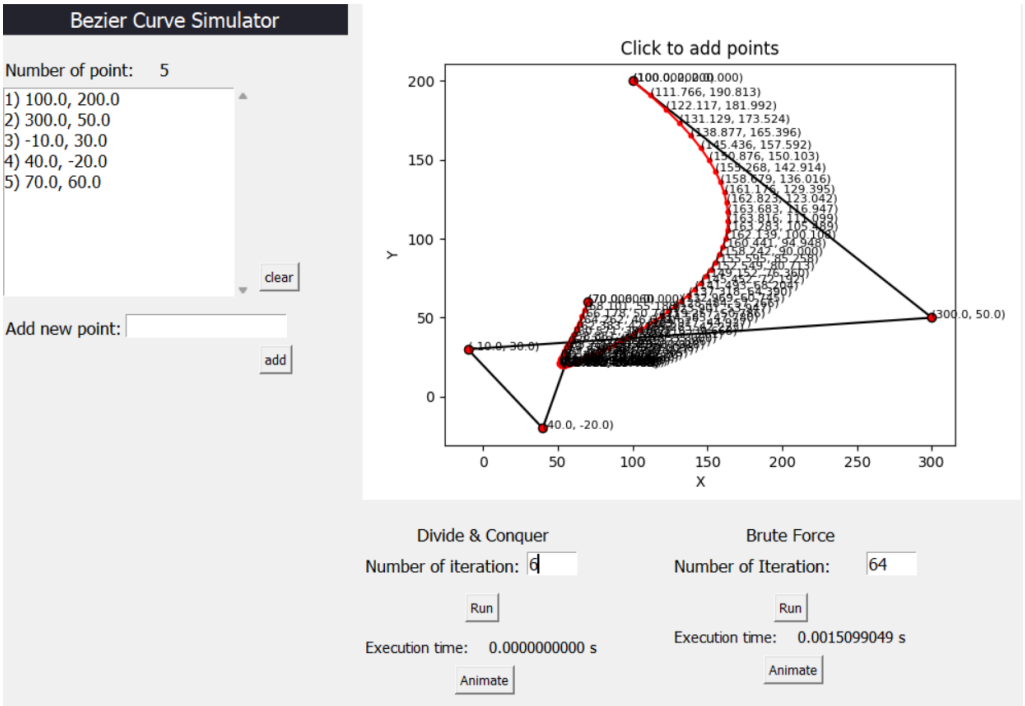
4. Test Case 4



5. Test Case 5


Bezier Curve Simulator
Number of point: 8
1) 100.0, 70.0
2) 200.0, 30.0
3) 40.0, 90.0
4) 500.0, -100.0
5) -200.0, 400.0
6) 90.0, 0.0
7) 70.0, -200.0
8) 80.0, 100.0
clear
Add new point:
add
Click to add points
(-200.0, 400.0)
(200.0, 30.0)
(70.0, -200.0)
X
Y
Divide & Conquer
Number of iteration: 5
Run
Execution time: 0.0000000000 s
Animate
Brute Force
Number of Iteration: 64
Run
Execution time: 0.0000000000 s
Animate

6. Test Case 6


Bezier Curve Simulator
Number of point: 5
1) 100.0, 200.0
2) 300.0, 50.0
3) -10.0, 30.0
4) 40.0, -20.0
5) 70.0, 60.0
clear
Add new point:
add
Click to add points
(300.0, 50.0)
(-10.0, 30.0)
(40.0, -20.0)
X
Y
Divide & Conquer
Number of iteration: 6
Run
Execution time: 0.0000000000 s
Animate
Brute Force
Number of Iteration: 64
Run
Execution time: 0.0015099049 s
Animate

IV. Implementasi Algoritma Bruteforce

A. Deskripsi Langkah

Program berikut memanfaatkan algoritma de casteljau untuk pencarian bezier curve.

$$P = (1 - t)P_0 + tP_1$$

Rumus tersebut kemudian dilakukan secara rekursif hingga *control points* turunannya tersisa 1. Proses tersebut kemudian dilakukan sebanyak N kali untuk semua t = 1/N dengan 0 < t <=1.

1. Program menerima jumlah *control points* beserta dengan *control points,* dan jumlah iterasi N
2. Program mencari P untuk setiap titik *control points* yang bersebelahan dan di simpan dalam array yang berisi P tersebut
3. Array dari P dianggap sebagai *control points* baru kemudian direkursi pada definisi bezier_curve() hingga panjang array dari P berjumlah 1
4. Program mengembalikan Array dari P
5. Langkah 1-4 diulangi sebanyak N kali untuk semua t = 1/N dengan 0 < t <=1.

B. Kode Sumber

Kode sumber berikut hanya terdiri dari algoritma utama, tidak termasuk kode pembangun GUI.

```
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np
import time


def de_casteljau(control_points, t):
    if len(control_points) == 1:
        return control_points[0]
    else:
        new_points = []
        for i in range(len(control_points) - 1):
            x = (1 - t) * control_points[i][0] + t * control_points[i + 1][0]
            y = (1 - t) * control_points[i][1] + t * control_points[i + 1][1]
            new_points.append((x, y))
        return de_casteljau(new_points, t)


def bezier_curve(control_points, n):
    curve = []
    for t in [i / n for i in range(n + 1)]:
        curve.append(de_casteljau(control_points, t))
    return curve


def runBF():
    global isProcessing
    isProcessing = True
```

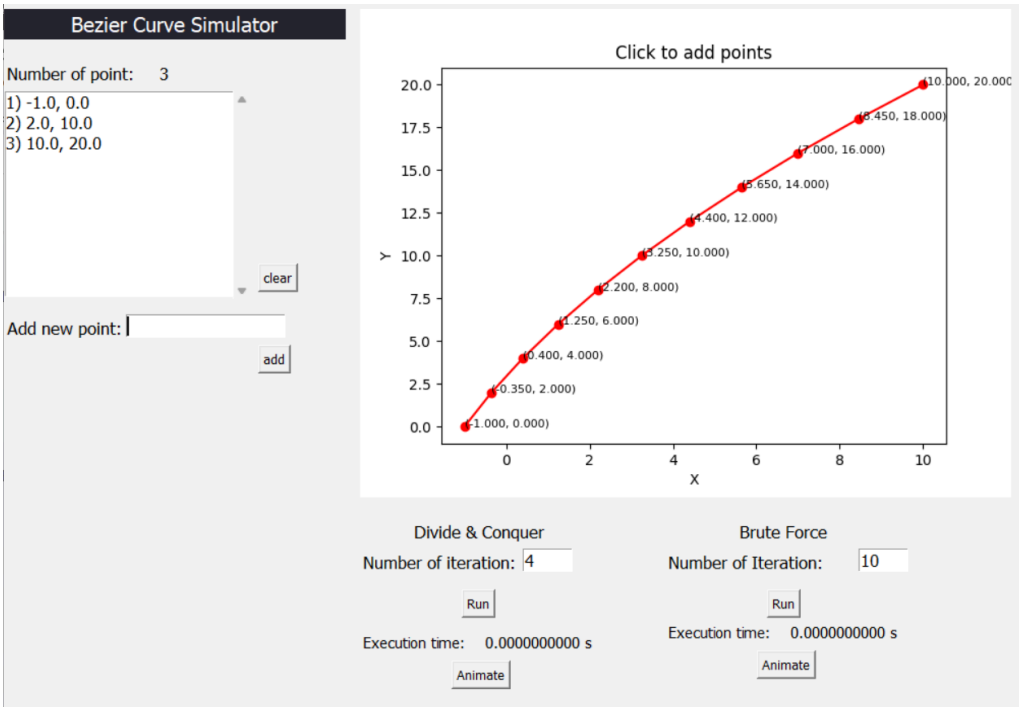
```
    BtnAddPoint.config(state='disabled')
    bezierPoints.clear()
    ax.clear()

    N = int(EntryBFIteration.get())

    if (N > 0 and len(controlPoints) >= 2):
        start = time.time()
        result = bezier_curve(controlPoints, N)
        end = time.time()

        LabelBFTimeVal.config(text=f"{(end - start):.10f} s")


        # Plot points
        ax.plot([point[0] for point in result], [point[1] for point in result],
'ro-', markerfacecolor='red')

        # Annotate each point with its coordinates formatted to three decimal
places
        for point in result:
            ax.text(point[0], point[1], f'({point[0]:.3f}, {point[1]:.3f})',
fontsize=8)


        ax.set_title('Click to add points')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')

        fig.canvas.draw()

    BtnAddPoint.config(state='normal')

    isProcessing = False


def animateBF():
    global isProcessing
    isProcessing = True

    BtnAddPoint.config(state='disabled')
    bezierPoints.clear()
    ax.clear()

    N = int(EntryBFIteration.get())

    if (N > 0 and len(controlPoints) >= 2):
        result = bezier_curve(controlPoints, N)

        for i in range(len(result) - 1):
            ax.plot(
                [result[i][0], result[i + 1][0]],
                [result[i][1], result[i + 1][1]],
                'ro-'
            )
```

```
            ax.text(result[i][0], result[i][1], f'({result[i][0]:.3f},
{result[i][1]:.3f})', fontsize=8)
            fig.canvas.draw_idle()
            fig.canvas.start_event_loop(0.05)

        ax.text(result[-1][0], result[-1][1], f'({result[-1][0]:.3f},
{result[-1][1]:.3f})', fontsize=8)

        ax.set_title('Click to add points')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        fig.canvas.draw()
```

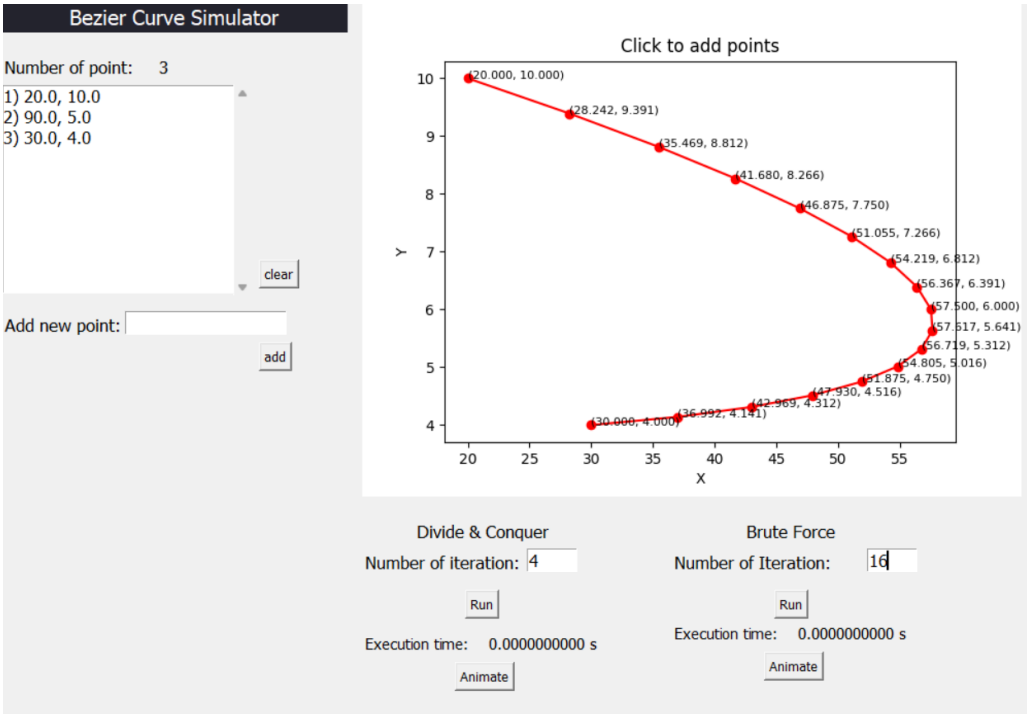
C. Pengujian

Test case kuadratik

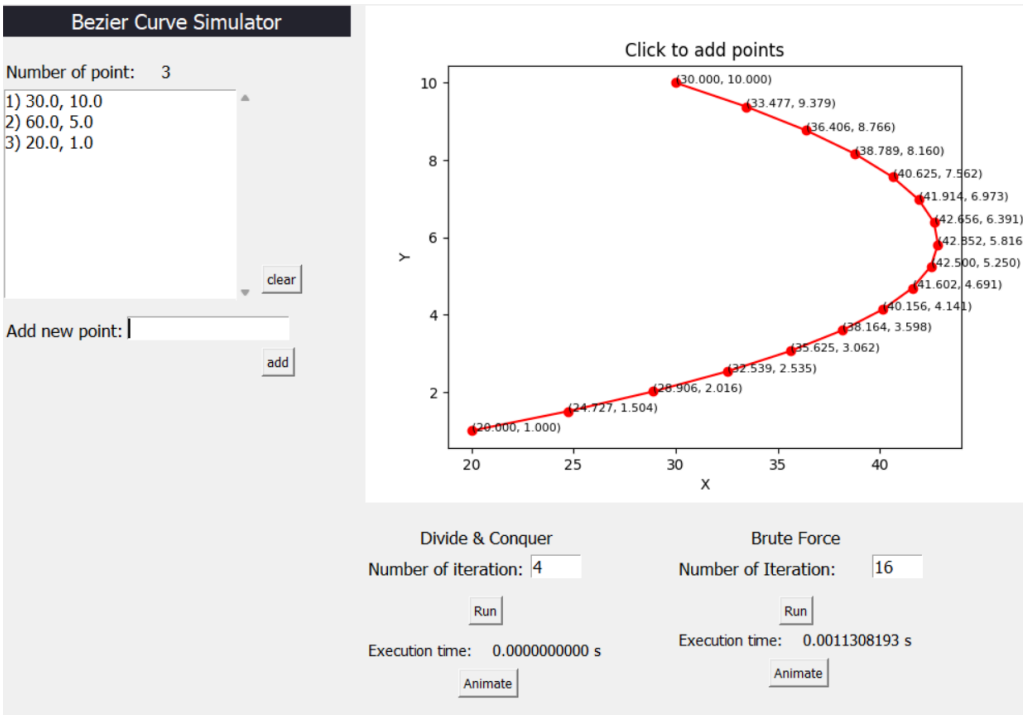
1. Test Case 1:



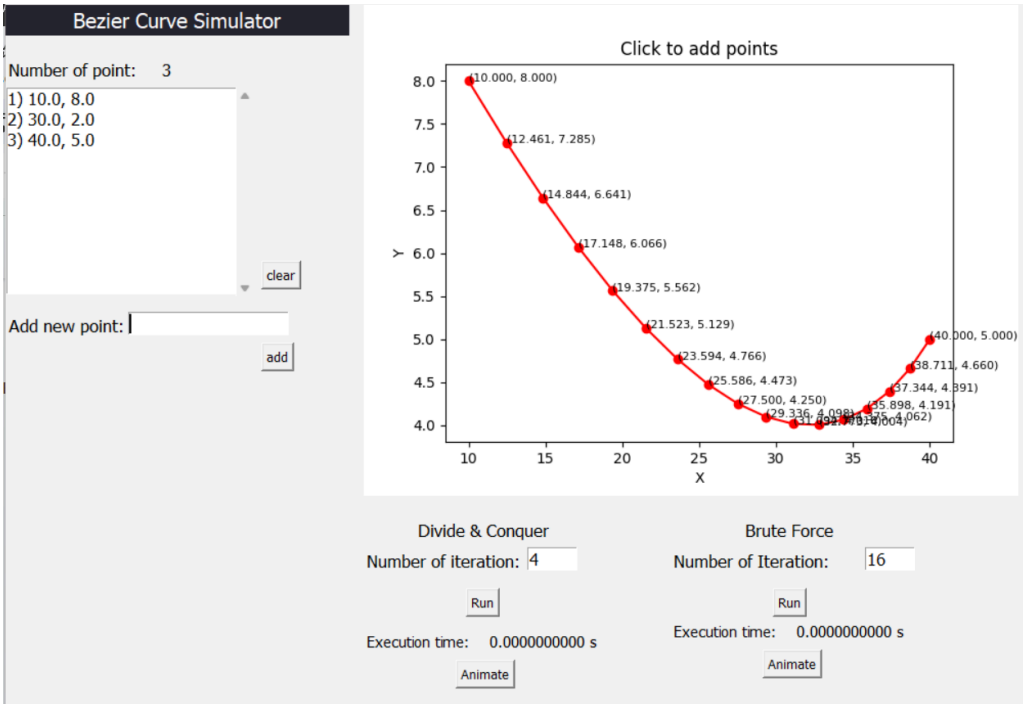
2. Test Case 2



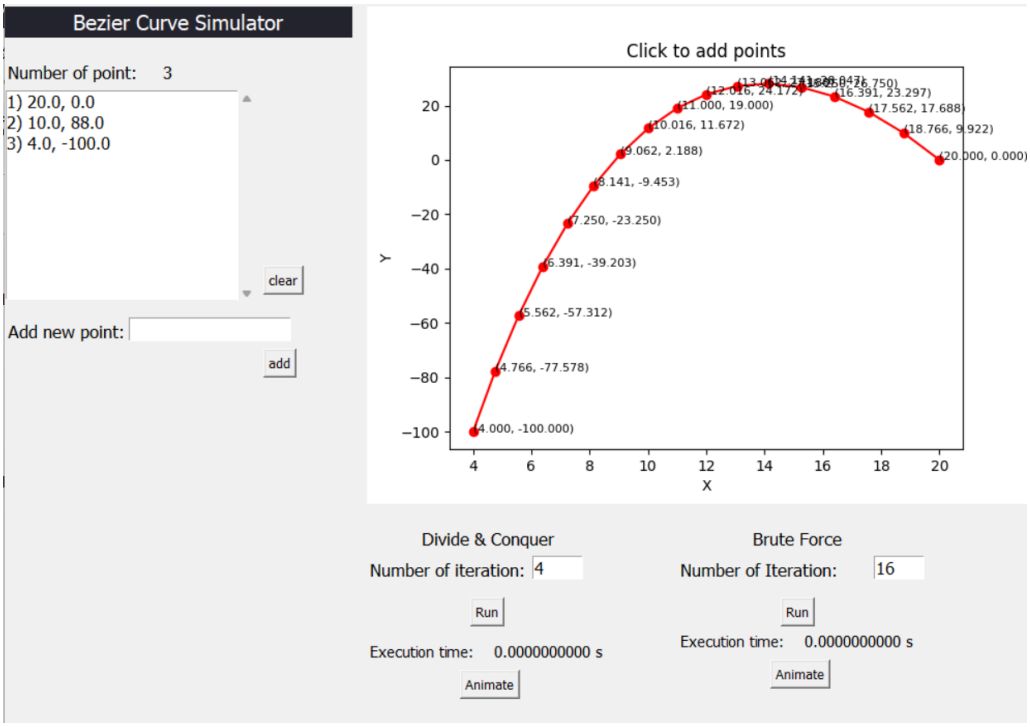
3. Test Case 3



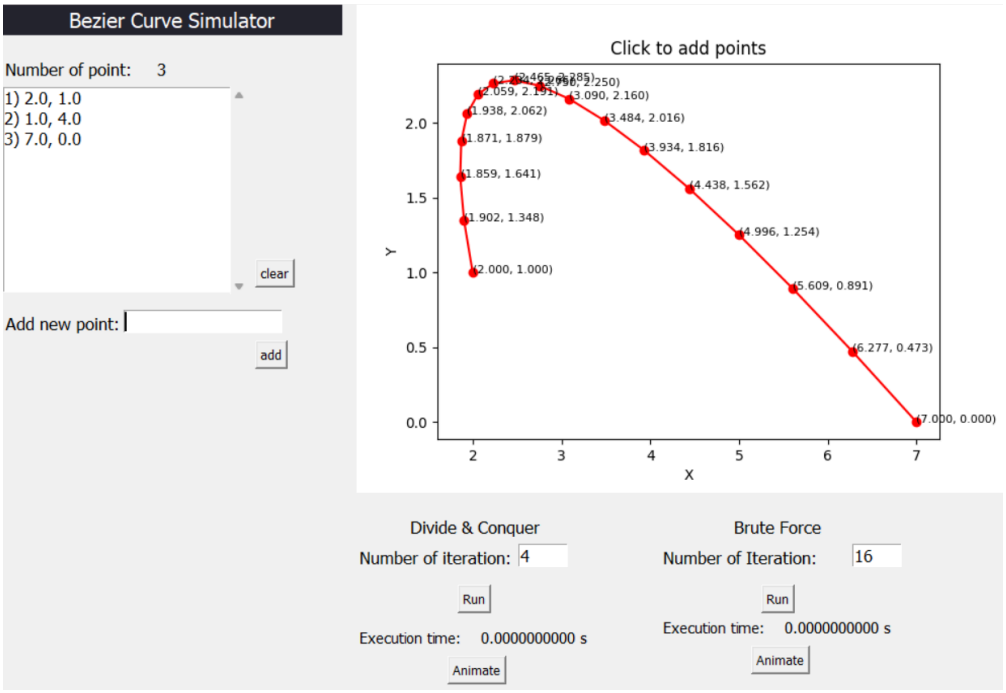
4. Test Case 4



5. Test Case 5

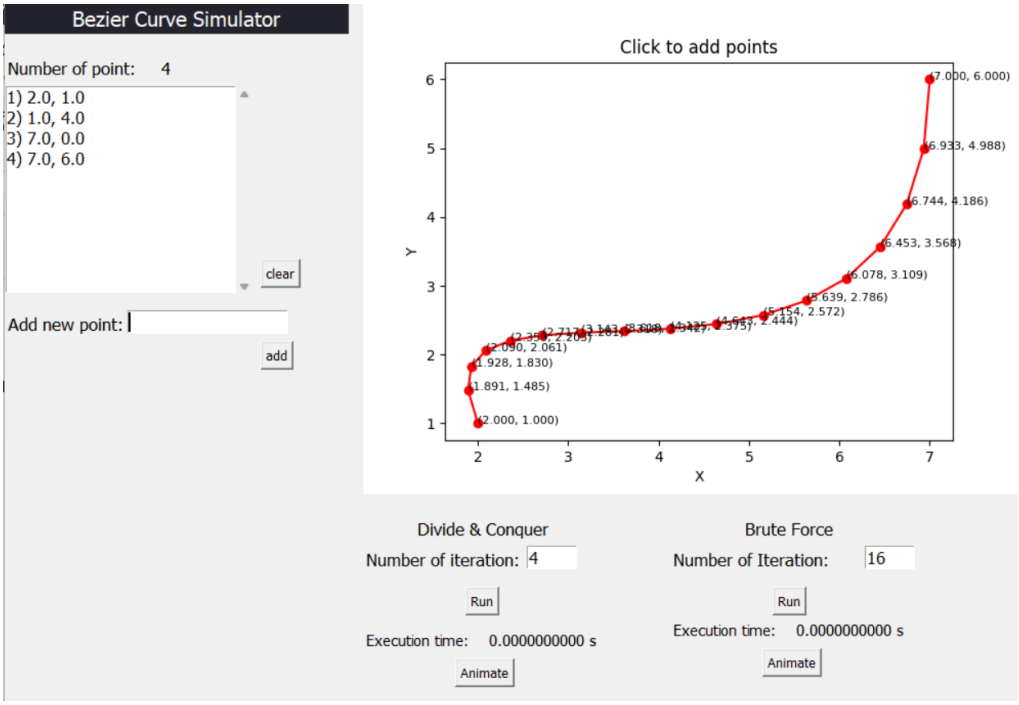


6. Test Case 6

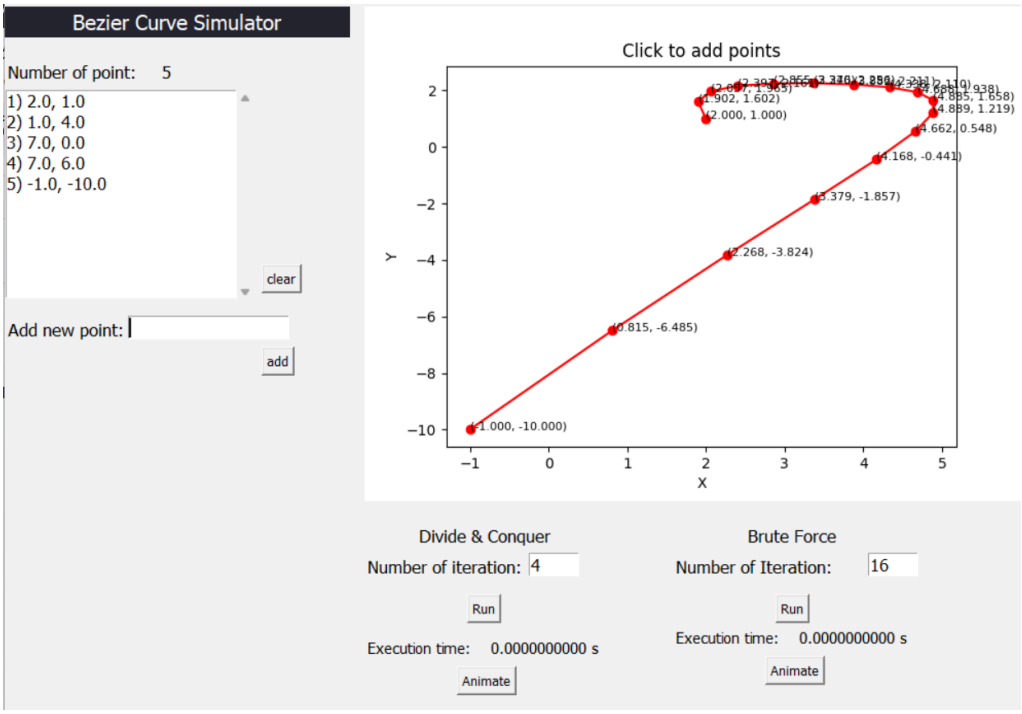


Test case n > 3

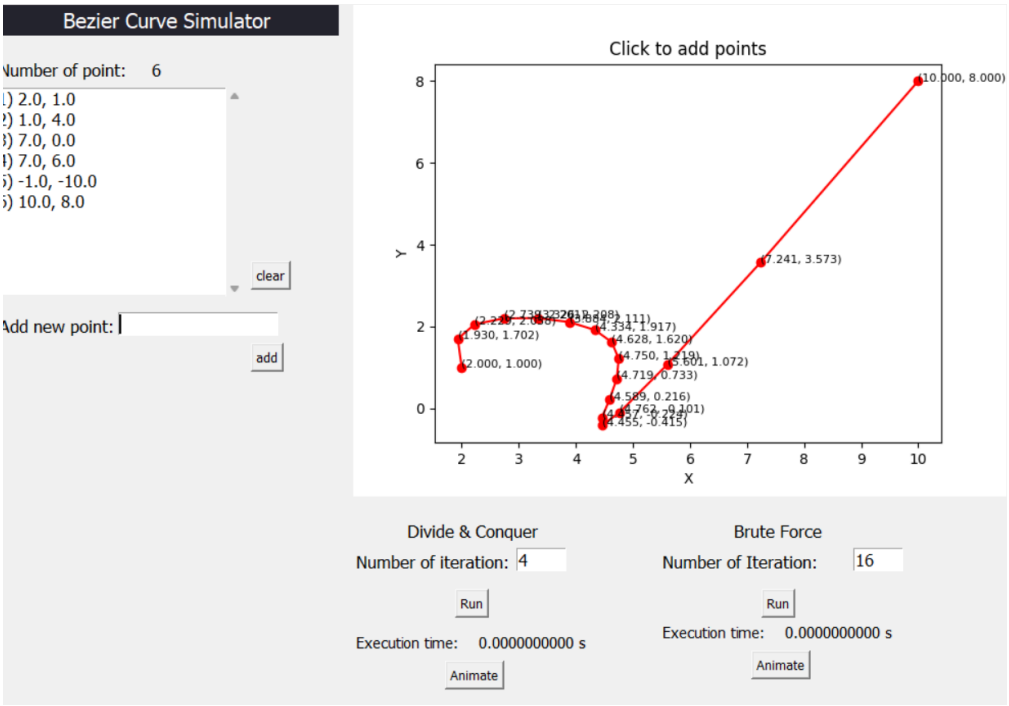
7. **Test Case 1**



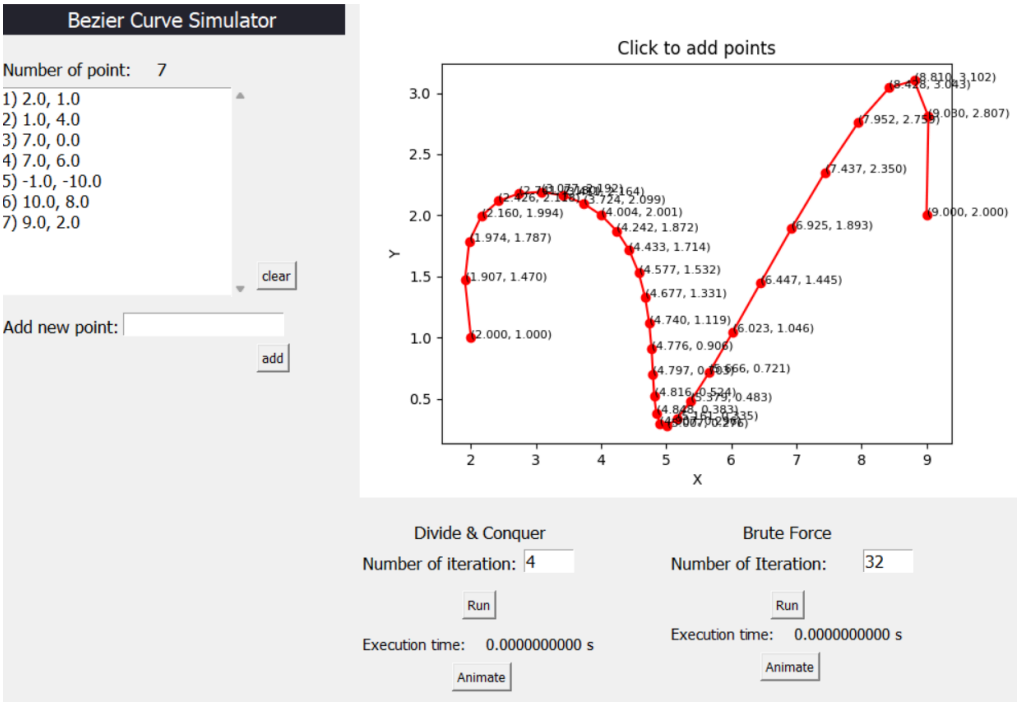
8. **Test Case 2**



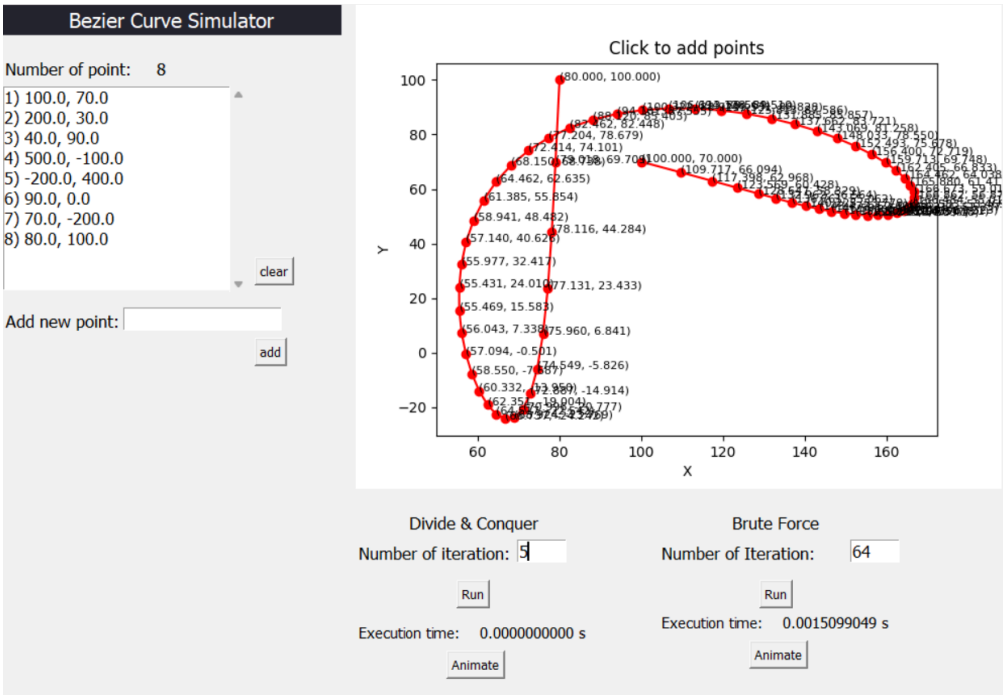
9. Test Case 3



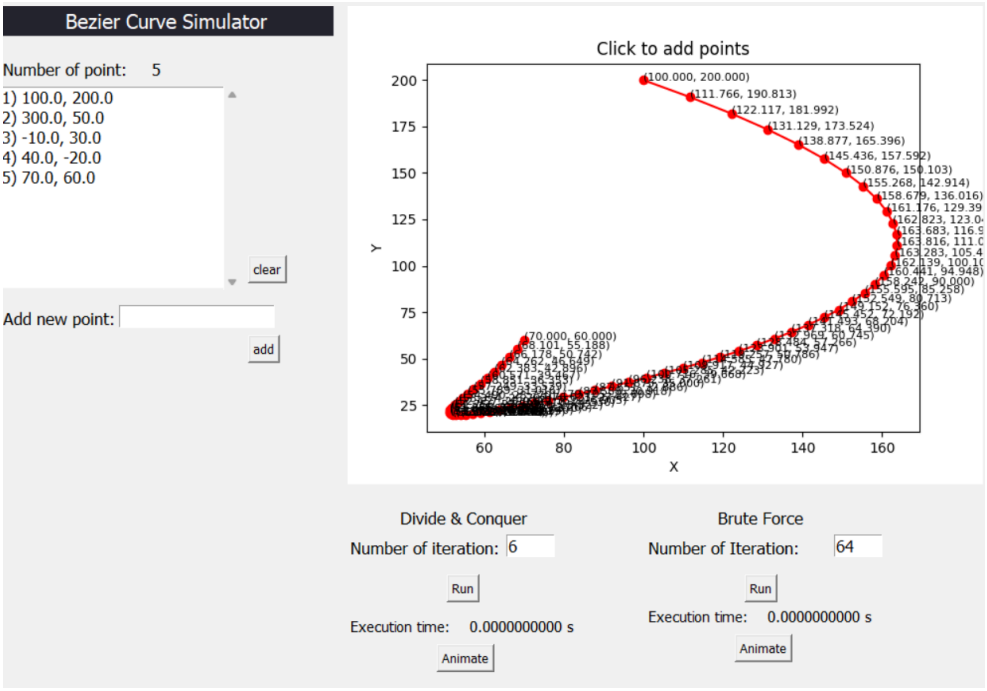
10. Test Case 4



11. Test Case 5



12. Test Case 6



V. Perbandingan Algoritma DnC dan Bruteforce

A. Kompleksitas Algoritma

1. Divide and Conquer

Dengan menggunakan Teorema Master, pertama-tama dilakukan identifikasi terhadap parameter kunci dari algoritma. Ukuran masalah diasumsikan sebagai jumlah titik kontrol (n), dengan faktor pembagian (b = 2) mengindikasikan bahwa setiap pemanggilan fungsi menghasilkan dua submasalah. Kedalaman rekursi ditentukan oleh jumlah iterasi maksimum (m). Dengan parameter ini, kompleksitas waktu algoritma diwakili sebagai

$(T(n) = a \,.\, T(n/b) + f(n))$, di mana (a) adalah jumlah submasalah per pemanggilan, dan (f(n)) merupakan kompleksitas waktu untuk menyelesaikan masalah ukuran (n) dan menggabungkan solusinya.

Dengan (a = 2), (b = 2), dan asumsi bahwa kompleksitas penggabungan solusi tidak dominan, kompleksitas waktu algoritma ini dapat disimpulkan sebagai $O(n \,.\, 2^m)$, di mana (n) adalah jumlah titik kontrol dan (m) adalah jumlah iterasi maksimum.

2. Brute Force

Pada bagian rekursif, proses pencarian Array dari P memiliki kompleksitas O(n), di mana n adalah panjang array control points. Kemudian, dalam rekursi, Array dari P digunakan sebagai control points baru, dan rekursi dilakukan sebanyak n-1 kali. Kemudian dilakukan iterasi sebanyak k kali dengan k merupakan konstanta. Dengan demikian, kompleksitas algoritma pada bagian rekursifnya adalah O(k(n(n-1))), yang menyebabkan kompleksitas keseluruhan menjadi $O(n^2)$.

Berdasarkan uji coba, metode Divide and Conquer (DnC) menunjukkan bahwa tidak dibutuhkan iterasi yang banyak dibandingkan dengan metode Bruteforce untuk menghasilkan kurva bezier yang mulus. Percobaan menunjukkan bahwa jumlah titik pada kurva bezier dalam algoritma Bruteforce sebanding dengan jumlah titik dalam algoritma DnC ketika $n = 2^m$, di mana n adalah jumlah iterasi pada Bruteforce, dan m adalah jumlah iterasi pada DnC. Waktu pemrosesan keduanya juga tidak berbeda jauh jika jumlah titik pada kurva bezier sama. Dilihat dari kompleksitas waktu, jika menggantikan n dengan 2^m dalam kompleksitas algoritma Bruteforce, maka kompleksitasnya akan sebanding dengan algoritma Divide and Conquer.

VI. Implementasi Bonus

A. Generalisasi Orde Kurva Bezier

Berdasarkan deskripsi langkah algoritma divide and conquer pada bab III dan eksplorasi terhadap pembentukan kurva bezier kuadratik dan kubik, generalisasi algoritma pembentukan kurva bezier berorde N dapat dibuat dengan mengetahui fakta berikut.

1. Iterasi kesatu selalu menghasilkan 1 titik tengah sehingga kurva bezier yang terbentuk adalah tiga titik.
2. Titik kontrol akhir pada segmen kiri sama dengan titik kontrol awal pada segmen kanan.

Dengan fakta di atas, diperoleh algoritma divide and conquer untuk kurva bezier berorde N sebagai berikut,

```
def NthBezier(controlPoints, limIter, currIter=0):
  if (currIter < limIter):
    midPoints = controlPoints.copy()

    left = [controlPoints[0]]
    right = [controlPoints[-1]]


    for i in range(len(controlPoints) -1):
      midMidPoints = []
      for j in range(1, len(midPoints)):
        midMidPoints.append(getMidPoint(midPoints[j], midPoints[j-1]))

      left.append(midMidPoints[0])
      right.append(midMidPoints[-1])

      midPoints = midMidPoints

    right = right[::-1]

    # LEFT SEGMENT
    NthBezier(left, nIter, currIter+1)
```

```
    bezierPoints.append(midPoints[0])

    # RIGHT SEGMENT
    NthBezier(right, nIter, currIter+1)
```

S
s

B. Visualisasi Proses Pembuatan Kurva
Perlu diketahui bahwa graphical user interface (GUI) dibentuk dengan pustaka Tkinter dan grafik dibuat dengan pustaka Matplotlib. Pengguna dapat memasukkan titik kontrol melalui kolom “Add new point” atau dengan mengklik langsung pada koordinat yang tersedia pada grafik.
Visualisasi proses pembuatan kurva dilakukan dengan cara melakukan plotting pada grafik untuk titik kontrol awal, titik tengah baru, dan titik kontrol akhir yang berasal dari argumen fungsi rekursif.

VII. Lampiran

Repositori GitHub : https://github.com/shafiqIrv/Tucil2_13522003_13522114

VIII. Daftar Pustaka

Munir, Rinaldi. 2024. “Aplikasi Divide and Conquer pada Grafika Komputer”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Aplikasi-Divide-and-Conquer-2020.pdf
(Diakses pada 16 Maret 2024)

Kumar, K. Ganesh. 2012. "Midpoint Algorithm: Divide and Conquer Method for Drawing Ellipse." CodeProject. https://www.codeproject.com/Articles/223159/Midpoint-Algorithm-Divide-and-Conquer-Method-for-D
(Diakses pada 15 Maret 2024)

No	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.	☑	☐