LAPORAN TUGAS KECIL
IF2211 - STRATEGI ALGORITMA

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Kelas Mahasiswa K01
Dosen Pengampu:
Dr. Ir. Rinaldi Munir, M.T.

Disusun Oleh:

Mesach Harmasendo 13522117
Farel Winalda 13522047

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024

DAFTAR ISI

BAB 1
ALGORITMA BRUTEFORCE

Algoritma Brute Force, juga dikenal sebagai pendekatan kekuatan penuh atau pencarian lengkap, adalah metode pemecahan masalah yang sederhana namun kuat, yang bekerja dengan mencoba semua kemungkinan solusi hingga menemukan jawaban yang benar. Algoritma ini tidak memerlukan pengetahuan khusus tentang masalah yang dihadapi dan mengandalkan kekuatan komputasi murni untuk menelusuri setiap kemungkinan solusi satu per satu.

Dalam kasus ini, pembuatan kurva bezier dengan pendekatan brute force dilakukan dengan cara menggunakan persamaan polinomial yang terbentuk berdasarkan titik-titik kontrol yang diberikan. Persamaan polinomial yang terbentuk akan bergantung pada variabel t yang memiliki rentang nilai antara 0 dan 1 (inklusif). Pembentukan kurva dengan cara ini akan dilakukan dengan cara mencoba berbagai macam nilai t pada rentang tersebut untuk mendapatkan titik-titik pada kurva bezier. Satu nilai t akan mewakili satu titik pada kurva bezier oleh sebab itu semakin banyak nilai t yang dipilih maka kurva yang terbentuk juga akan semakin halus. Pada algoritma brute force ini juga terdapat parameter iterasi untuk menyesuaikan dengan algoritma DnC.

Untuk kasus 3 titik kontrol rumus yang digunakan untuk menghitung titik pada kurva bezier adalah sebagai berikut

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$
$$R_0 = B(t) = (1-t)Q_0 + tQ_1, \qquad t \in [0,1]$$

Dengan mensubstitusikan nilai Q_0 dan Q_1 maka persamaan di atas bisa diubah menjadi persamaan seperti di bawah ini

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

Variabel P di atas menunjukkan titik kontrol yang diberikan sedangkan variabel R menunjukkan titik pada kurva bezier yang didapatkan berdasarkan nilai t tertentu.

Untuk kasus n buah titik kontrol dimana n > 1 maka rumus yang digunakan adalah sebagai berikut

$$B(t) = \sum_{i=0}^{n} C(n,i).(1-t)^{n-i} t^{i} P_i - 1$$

$$= (1-t)^{n} P_0 + C(n,1)(1-t)^{n-1} t P_1 + \ldots\ldots + t^{n} P_n$$

$$untuk\ nilai\ t \in [0,1]$$

Selain menggunakan rumus di atas bisa juga dengan cara yang lebih manual yaitu dengan menggunakan persamaan di bawah ini untuk setiap 2 titik yang saling terhubung.

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

Setelah didapatkan titik baru dari persamaan tersebut maka titik baru tersebut akan kembali disubstitusikan ke dalam persamaan tersebut dan proses ini akan dilakukan berulang kali hingga didapatkan titik akhir yang sudah tidak bisa disubstitusikan lagi. Titik akhir itulah yang nantinya akan menjadi titik pada kurva bezier.

Penerapan algoritma brute force pada pembuatan kurva bezier adalah sebagai berikut:

1. Menentukan banyak titik pada kurva yang akan dicari berdasarkan jumlah iterasi yang diberikan. **Hal ini diperlukan agar jumlah titik pada kurva hasil dari perhitungan brute force selalu sama dengan jumlah titik pada kurva hasil dari perhitungan divide and conquer**. Setelah jumlah titik sudah diketahui maka akan dicari interval atau jarak antar nilai t yang mewakili titik-titik tersebut.
2. Setelah interval ditemukan maka hanya tinggal melakukan perulangan untuk nilai t dengan interval yang sudah dihitung sebelumnya. Sebagai contoh misalnya akan dicari 10 titik pada kurva bezier maka akan dilakukan perulangan untuk nilai t = 0, t = 0.1, t = 0.2, …, t = 1. Pada setiap perulangan nilai t tersebut akan dimasukkan ke dalam rumus

persamaan seperti yang sudah tertulis di atas menyesuaikan dengan jumlah titik kontrol yang diberikan. Jumlah perulangan yang dilakukan adalah sejumlah jumlah titik pada kurva bezier yang ingin dicari.

3. Setelah titik-titik pada kurva sudah ditemukan maka titik-titik tersebut hanya tinggal dihubungkan dengan garis dan kurva bezier akan terbentuk.

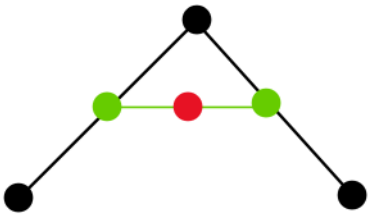
Pada kasus pembentukan kurva bezier dengan pendekatan brute force ini secara umum akan perlu dilakukan perulangan sebanyak 3 tingkat dimana perulangan tingkat pertama akan berfungsi untuk melacak jumlah titik pada kurva bezier yang ingin dicari sedangkan perulangan tingkat 2 dan 3 akan digunakan untuk mencari titik pada kurva berdasarkan nilai t yang diberikan. Untuk membuat algoritma yang umum/general maka pencarian titik pada kurva akan menggunakan cara yang manual yaitu dengan hanya menggunakan persamaan 2 titik seperti yang sudah dituliskan di atas. Perulangan perlu dilakukan dengan 2 tingkat dikarenakan persamaan dua titik itu perlu dipanggil dan disubstitusikan secara terus menerus hingga mendapatkan titik pada kurva yang diinginkan. Untuk kasus hanya 3 titik kontrol saja Perulangan bisa disederhanakan menjadi satu tingkat saja dimana perulangan akan dilakukan sebanyak titik pada kurva yang ingin dicari dan pada tiap perulangan hanya tinggal memasukkan nilai t pada persamaan untuk tiga titik kontrol yang sudah ditampilkan di atas. Algoritma brute force ini dibuat hanya untuk digunakan sebagai pembanding dengan algoritma Divide and Conquer yang akan dibuat selanjutnya.

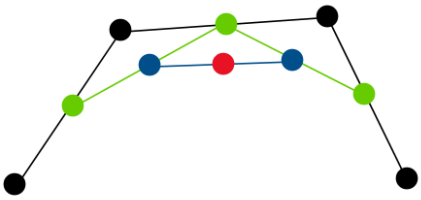
BAB 2
ALGORITMA DIVIDE AND CONQUER

Algoritma Divide and Conquer merupakan sebuah paradigma pemrograman yang penting dalam pengembangan algoritma komputasi, yang bekerja dengan memecah masalah besar menjadi sub-masalah yang lebih kecil, lebih mudah dipecahkan, dan kemudian menggabungkan solusi dari sub-masalah tersebut untuk mendapatkan solusi masalah utama. Konsep utama dari Divide and Conquer adalah membagi masalah menjadi dua atau lebih bagian yang *disjunct* (tidak tumpang tindih) atau lebih sering dibagi menjadi dua bagian yang hampir sama ukurannya, menyelesaikan setiap bagian secara rekursif, dan akhirnya menggabungkan solusi dari setiap bagian untuk membentuk solusi akhir.

Berdasarkan penjelasan di atas bisa dikatakan bahwa pendekatan Divide and Conquer ini bisa digunakan untuk menyelesaikan masalah pembuatan kurva bezier dengan 3 atau bahkan lebih dari 3 titik kontrol. Dengan menggunakan pendekatan Divide and Conquer maka algoritma yang akan digunakan untuk pembentukan kurva bezier adalah algoritma titik tengah. Dengan pendekatan ini kurva nantinya akan secara terus menerus dibagi dua, kemudian akan dicari titik tengahnya dan pada tahap akhir titik-titik tengah tersebut akan disatukan kembali membentuk sebuah kurva yang diinginkan. Pembagian kurva menjadi dua bagian akan dilakukan sebanyak iterasi yang diinginkan dan semakin banyak iterasi yang dilakukan maka bentuk kurva yang terbentuk akan semakin halus dan rapi. Berikut adalah langkah-langkah pembentukan kurva bezier dengan pendekatan Divide and Conquer:

1. Mencari titik tengah dari garis terakhir yang bisa terbentuk berdasarkan titik-titik tengah sebelumnya. Pada tahap pertama ini titik tengah akan dicari dari garis-garis yang terbentuk dari titik-titik kontrol yang diberikan. Untuk memperjelas titik tengah mana yang dicari maka bisa melihat ilustrasi di bawah ini:

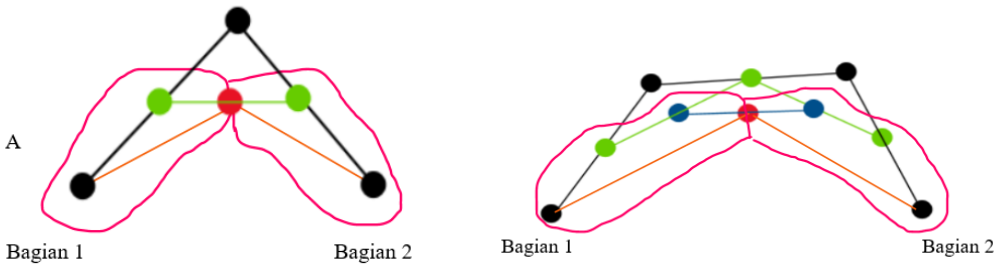




Titik berwarna merah pada gambar adalah titik tengah yang dimaksud. Titik tengah ini nantinya akan dijadikan pedoman untuk melakukan pembagian kurva menjadi dua bagian. Titik tengah ini juga akan menjadi salah satu titik yang terdapat pada kurva bezier yang akan terbentuk. Seperti yang terlihat pada gambar semakin banyak titik kontrolnya maka titik tengah yang ingin didapatkan akan berada pada posisi yang semakin dalam. Secara umum untuk dapat memperoleh titik tengah tersebut setidaknya perlu dilakukan perulangan sebanyak dua tingkat dimana perulangan pertama dilakukan untuk melacak sedalam apa posisi dari titik tengah dari garis terakhir tersebut akan terbentuk sedangkan perulangan kedua dilakukan untuk menghitung titik-titik tengah dari setiap garis yang terbentuk pada suatu kedalaman/tingkat tertentu. Untuk lebih memahami makna dari kedalaman atau tingkat disini mungkin bisa melihat kembali gambar diatas. Garis berwarna hijau adalah garis pada kedalaman atau tingkat 1 sedangkan garis berwarna biru adalah garis yang berada pada kedalaman atau tingkat 2 dan begitu seterusnya. Perulangan tingkat pertama akan dilakukan sebanyak n - 1 dimana n menunjukkan banyak titik kontrol yang diberikan sedangkan perulangan tingkat kedua akan dilakukan sebanyak m - 1 dimana m menunjukkan banyaknya titik tengah pada suatu kedalaman atau tingkat tertentu. Untuk kasus 3 titik kontrol perulangan bisa dibuat lebih sederhana menjadi satu tingkat saja dikarenakan pada kasus 3 titik kontrol titik tengah yang ingin dicari hanya berada pada kedalaman satu. Pada kasus tersebut perulangan hanya perlu dilakukan sebanyak n - 1 kali dimana n menunjukkan banyak titik kontrol yang diberikan. Tahap awal ini bisa dibilang sebagai tahap conquer.

2. Langkah yang berikutnya adalah membagi 2 kurva menjadi 2 bagian berdasarkan titik tengah utama yang telah didapatkan sebelumnya. Bagian satu akan terdiri dari titik-titik tengah dan titik kontrol pertama (jika bagian kurva paling kiri) yang berada di di sebelah

kiri titik tengah utama yang didapatkan sebelumnya sedangkan bagian dua akan terdiri dari titik-titik tengah dan titik kontrol terakhir (jika bagian kurva paling kanan) yang berada di sebelah kanan titik tengah utama yang didapatkan sebelumnya. Titik tengah utama yang didapat pada perhitungan sebelumnya akan masuk ke dalam bagian satu maupun bagian dua. Banyak titik pada setiap bagian yang baru terbentuk akan sama dengan banyak titik-titik kontrol yang diberikan sehingga bisa memanfaatkan ulang fungsi-fungsi yang telah dibuat sebelumnya. Tahap kedua ini bisa disebut juga sebagai tahap divide dimana sebuah masalah dipecah menjadi masalah-masalah yang lebih kecil namun tetap identik. Untuk memperjelas proses divide ini mungkin bisa melihat ilustrasi dibawah ini



3. Tahap pertama dan kedua yang telah dilakukan sebelumnya adalah langkah yang dilakukan untuk iterasi pertama. Jika iterasi dilakukan lebih dari satu kali maka akan dilakukan tahap ketiga dimana pada tahap ini akan dilakukan proses conquer kembali seperti yang dilakukan pada tahap pertama. Pada tahap ini proses conquer akan dilakukan pada bagian-bagian dari kurva yang telah dibagi sebelumnya. Setelah proses conquer selesai bagian-bagian dari kurva tersebut akan dibagi lagi (divide) menjadi dua bagian seperti yang dilakukan pada tahap dua dan proses ini akan berulang secara terus menerus sejumlah iterasi yang diinginkan. Secara singkat tahap ketiga adalah pengulangan dari tahap pertama dan kedua sejumlah iterasi yang diinginkan pada setiap bagian-bagian dari kurva yang telah terbagi-bagi. Proses pada tahap ketiga ini bersifat opsional bergantung jumlah iterasi yang diinginkan. Semakin banyak jumlah iterasi yang dilakukan tentu saja akan membuat konsentrasi titik akan semakin meningkat dan kepadatan titik pada kurva bezier yang terbentuk juga akan semakin rapat yang akan menghasilkan kurva bezier

yang halus. Proses iterasi ini dalam penerapannya akan dilakukan secara rekursif untuk mempermudah pembuatannya,

4. Setelah melalui proses iterasi tahap terakhir dari pembentukan kurva bezier ini adalah menyatukan kembali (combine) bagian-bagian dari kurva yang terbagi-bagi sebelumnya. Pada setiap bagian dari kurva akan didapatkan satu titik tengah dari sebuah garis terakhir yang merupakan hasil dari proses conquer di setiap bagian-bagian kurva. Titik-titik tengah tersebut kemudian akan disatukan dan garis yang menghubungkan titik-titik tengah tersebut akan membentuk sebuah kurva bezier berdasarkan titik kontrol yang diberikan.

BAB 3

SOURCE CODE

Algoritma Utama 3 Titik Kontrol

```python
class BezierCurve3:
    #inisialisasi objek
    def __init__(self, control_points, num_iterate):
        self.control_points = control_points
        self.num_iterate = num_iterate
        self.bezier_points = [] # titik-titik pada kurva (solusi)
        self.time_execution = 0 # waktu eksekusi algoritma murni
        self.num_points=0  # banyak titik di kurva pada suatu iterasi
        self.data_bruteforce=[] # data bruteforce untuk membuat animasi
        self.data_dnc = [[] for _ in range(5)] # data dnc untuk membuat animasi

    # fungsi untuk menghitung titik tengah diantara dua buah titik
    def mid_point(self,point1, point2):
        mid_x = (point1[0] + point2[0]) / 2
        mid_y = (point1[1] + point2[1]) / 2

        return mid_x, mid_y

    # fungsi rekursif untuk menghitung titik-titik pada kurva bezier dengan pendekatan dnc
    def calculate_bezier_points_dnc(self,point1, point2, point3, currIterations):
        if (currIterations < self.num_iterate):
            # tahap conquer
            midPoint1 = self.mid_point(point1, point2)
            midPoint2 = self.mid_point(point2, point3)
            midPoint3 = self.mid_point(midPoint1, midPoint2)

            if currIterations < 5:
                self.data_dnc[currIterations].extend([midPoint1, midPoint2])
            currIterations = currIterations + 1

            #tahap divide, pemanggilan rekursif, dan penggabungan solusi
            self.calculate_bezier_points_dnc(point1, midPoint1, midPoint3, currIterations)
            self.bezier_points.append(midPoint3)
            self.calculate_bezier_points_dnc(midPoint3, midPoint2, point3, currIterations)

    # fungsi yang menjadi entry point untuk pemanggilan fungsi rekursif dnc
    def calc_dnc(self):
        start_time = timeit.default_timer()
        self.bezier_points.append(self.control_points[0])
        self.calculate_bezier_points_dnc(self.control_points[0], self.control_points[1], self.control_points[2], 0)
        self.bezier_points.append(self.control_points[2])
        end_time = timeit.default_timer()
        self.time_execution = end_time - start_time
```

```python
# fungsi untuk menghitug berapa banyak titik yang akan dicari pada suatu iterasi
def count_points(self):
    count = 2 ** self.num_iterate + 1

    return count

# fungsi untuk menghitung titik-titik pada kurva bezier secara bruteforce
def calculate_bezier_points_bruteforce(self, num_points):
    curve_points = []

    for t in range(1, num_points - 1):
        t /= num_points - 1

        # persamaan bertahap
        x1 = (1-t) * self.control_points[0][0] + t* self.control_points[1][0]
        y1 = (1-t) * self.control_points[0][1] + t* self.control_points[1][1]
        x2 = (1-t) * self.control_points[1][0] + t* self.control_points[2][0]
        y2 = (1-t) * self.control_points[1][1] + t* self.control_points[2][1]

        x = (1-t) * x1 + t*x2
        y = (1-t) * y1 + t*y2

        # persamaan langsung jadi
        # x = (1 - t)**2 * self.control_points[0][0] + 2 * (1 - t) * t * self.control_points[1][0] + t**2 * self.control_points[2][0]
        # y = (1 - t)**2 * self.control_points[0][1] + 2 * (1 - t) * t * self.control_points[1][1] + t**2 * self.control_points[2][1]
        self.data_bruteforce.append([(x1,y1), (x2,y2)])
        curve_points.append((x, y))

    self.bezier_points = curve_points.copy()

# fungsi yang menjadi entry points untuk melakukan pembentukan kurva bezier secara bruteforce
def calc_bruteforce(self):
    start_time = timeit.default_timer()
    self.num_points=self.count_points()
    self.calculate_bezier_points_bruteforce(self.num_points)
    self.bezier_points.append(self.control_points[-1])
    self.bezier_points.insert(0, self.control_points[0])
    end_time = timeit.default_timer()
    self.time_execution = end_time-start_time
```

Code ini terdapat pada file src/algo/bezier_3_titik.py

Algoritma Utama n Titik Kontrol

```python
class BezierCurveN:
    #inisialisasi objek
    def __init__(self, control_points, num_iterate):
        self.control_points = control_points
        self.num_iterate = num_iterate
        self.bezier_points = [] # titik-titik pada kurva (solusi)
        self.time_execution = 0 # waktu eksekusi algoritma murni
        self.num_points = 0 # banyak titik di kurva pada suatu iterasi
        self.data_bruteforce=[] # data bruteforce untuk membuat animasi
        self.data_dnc = [[] for _ in range(5)] # data dnc untuk membuat animasi

    # fungsi untuk menghitung titik tengah diantara dua buah titik
    def mid_point(self, point1, point2):
        mid_x = (point1[0] + point2[0]) / 2
        mid_y = (point1[1] + point2[1]) / 2
        return mid_x, mid_y

    # fungsi rekursif untuk menghitung titik-titik pada kurva bezier dengan pendekatan dnc
    def calculate_bezier_points_dnc(self, points, current_iteration):
        if current_iteration < self.num_iterate:
            # tahap conquer
            all = [] # menyimpan semua data titik tengah yang didapatkan
            all.append(points)

            # pencarian titik tengah yang berada pada garis terakhir yang dibentuk
            for j in range(0, len(points) - 1):
                mid_points = []
                for i in range(0, len(all[j]) - 1):
                    mid_point = self.mid_point(all[j][i], all[j][i+1])
                    mid_points.append(mid_point)

                all.append(mid_points)

            #tahap divide
            left = [all[i][0] for i in range(len(all))]
            right = [all[len(all) - i - 1][-1] for i in range(len(all))]

            # penyimpanan data untuk mempermudah pembuatan animasi
            if current_iteration + 1 < 5:
                self.data_dnc[current_iteration + 1].append(all)

            # pemanggilan rekursif dan penggabungan solusi
            self.calculate_bezier_points_dnc(left, current_iteration + 1)
            self.bezier_points.append(all[-1][0])
            self.calculate_bezier_points_dnc(right, current_iteration + 1)
```

```python
# fungsi yang menjadi entry point untuk pemanggilan fungsi rekursif dnc
def calc_dnc(self):
    start_time = timeit.default_timer()
    self.bezier_points.append(self.control_points[0])
    self.calculate_bezier_points_dnc(self.control_points, 0)
    self.bezier_points.append(self.control_points[-1])
    end_time = timeit.default_timer()
    self.time_execution = end_time - start_time

# fungsi untuk menghitug berapa banyak titik yang akan dicari pada suatu iterasi
def count_points(self):
    count = 2 ** self.num_iterate + 1

    return count

# fungsi untuk menghitung titik-titik pada kurva bezier secara bruteforce
def calculate_bezier_points_bruteforce(self, num_points):
    curve_points = []

    for t in range(1, num_points - 1):
        t /= num_points - 1
        data = []
        data.append(self.control_points)
        for i in range(len(self.control_points) - 1):
            value = []
            for j in range(len(data[i]) - 1):
                # menggunakan persamaan dua titik
                x = (1-t) * data[i][j][0] + t *data[i][j+1][0]
                y = (1-t) * data[i][j][1] + t *data[i][j+1][1]
                value.append((x, y))

            data.append(value)
        curve_points.append(data[-1][0])
        self.data_bruteforce.append(data)

    self.bezier_points = curve_points.copy()

# fungsi yang menjadi entry points untuk melakukan pembentukan kurva bezier secara bruteforce
def calc_bruteforce(self):
    start_time = timeit.default_timer()
    self.num_points=self.count_points()
    self.calculate_bezier_points_bruteforce(self.num_points)
    self.bezier_points.append(self.control_points[-1])
    self.bezier_points.insert(0, self.control_points[0])
    end_time = timeit.default_timer()
    self.time_execution = end_time - start_time
```

Code ini terdapat pada file src/algo/bezier_n_titik.py

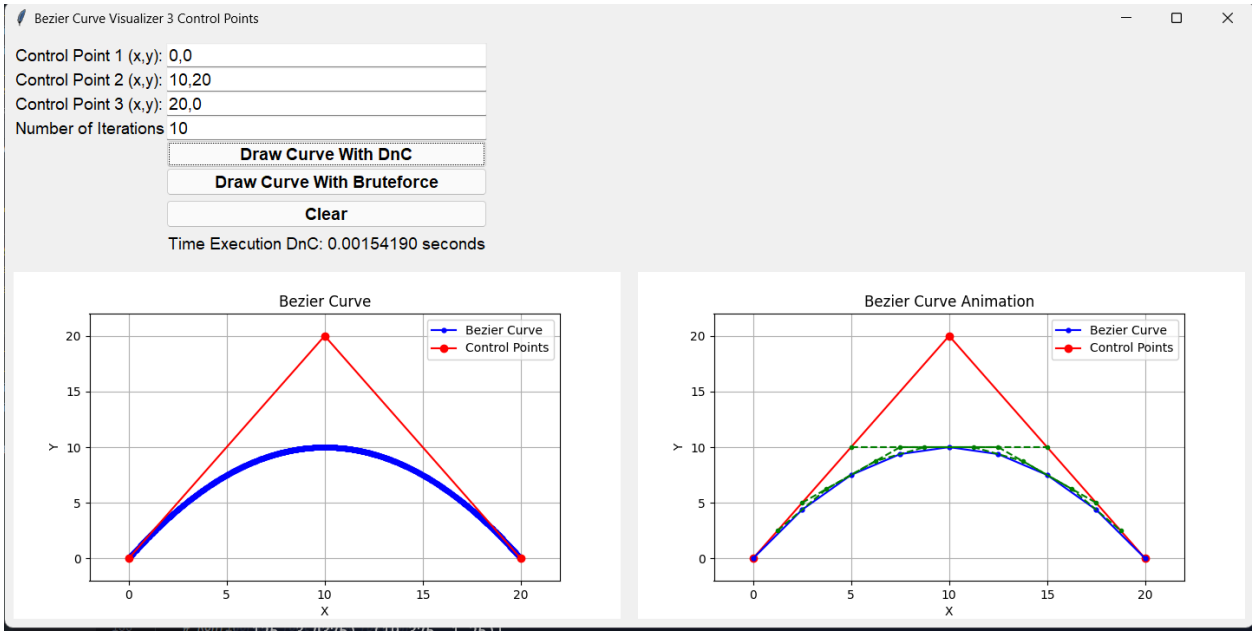
Dalam folder src/algo juga terdapat folder try yang mungkin bisa diabaikan saja. Folder itu berisi semua kode yang kami buat selama tucil ini sebagai proses dalam pembuatan algoritma yang kami gunakan sekarang.

BAB 4

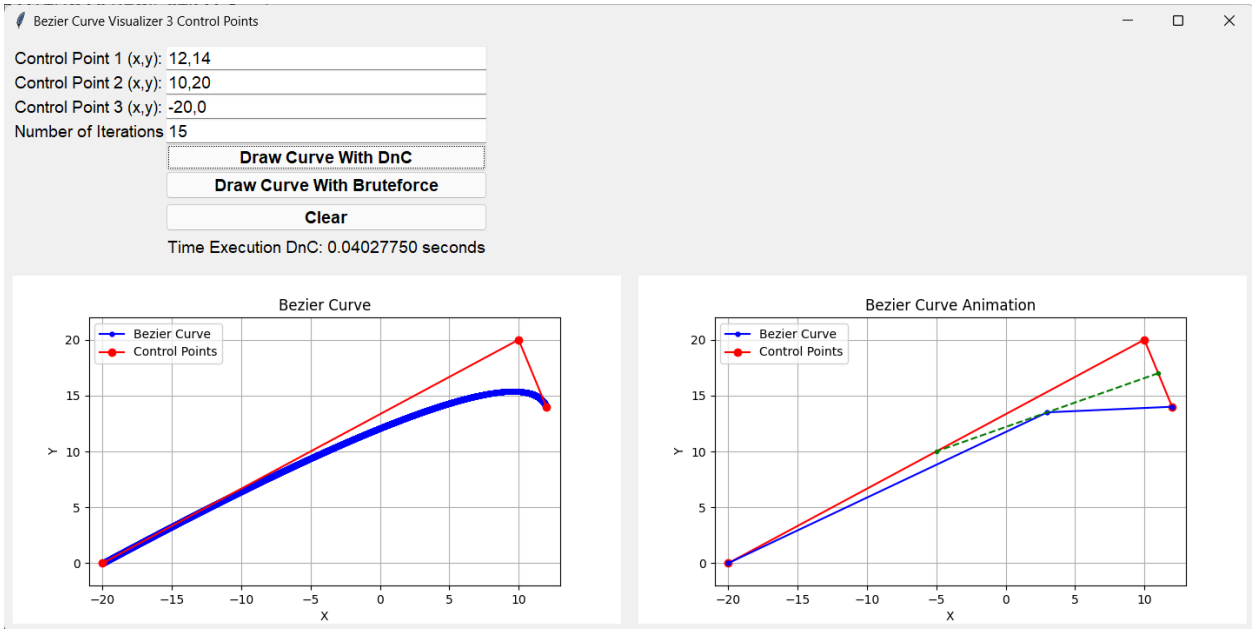
TEST CASE

a. Algoritma Divide and Conquer 3 Titik Kontrol

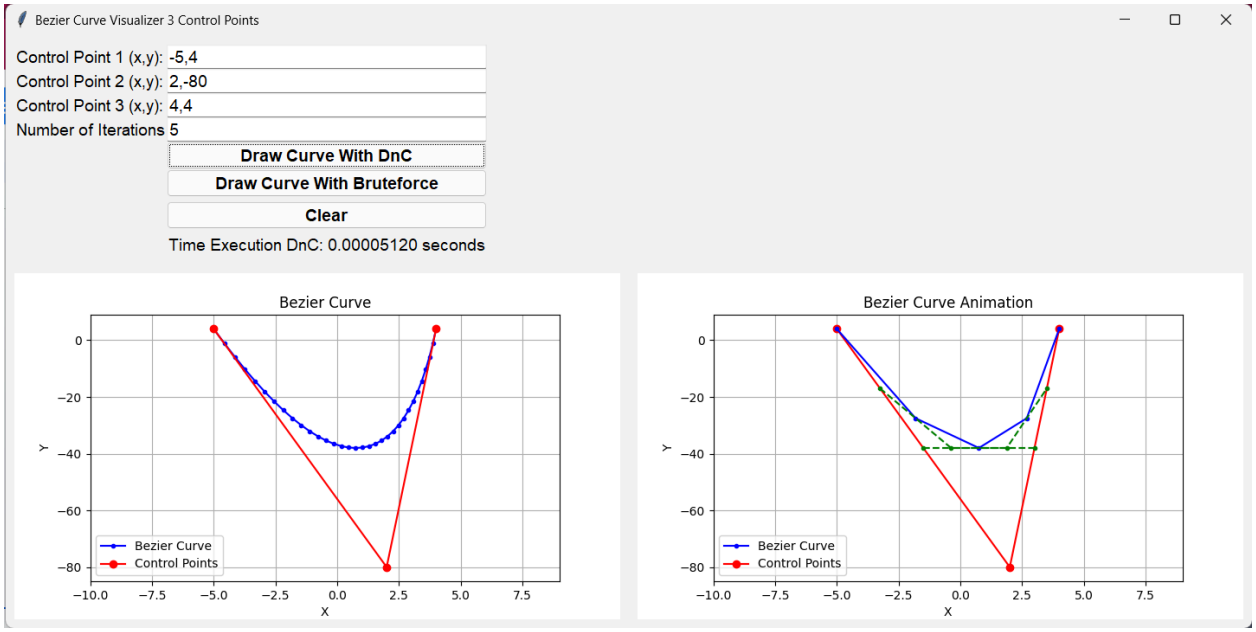
Test case 1



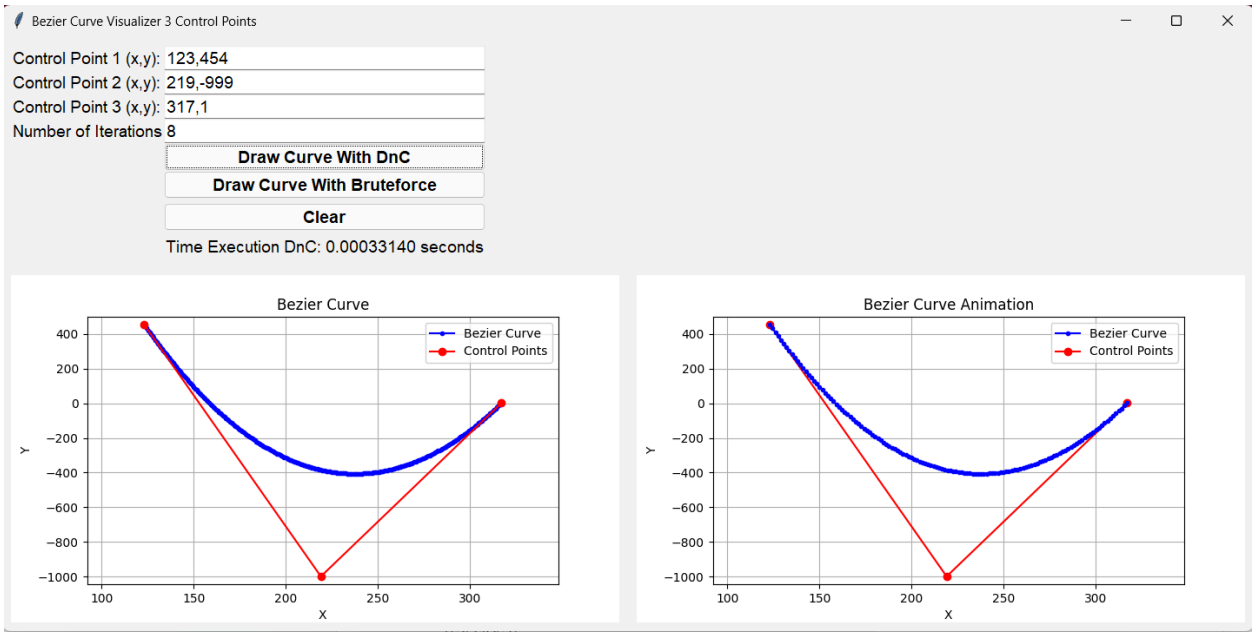
Test case 2



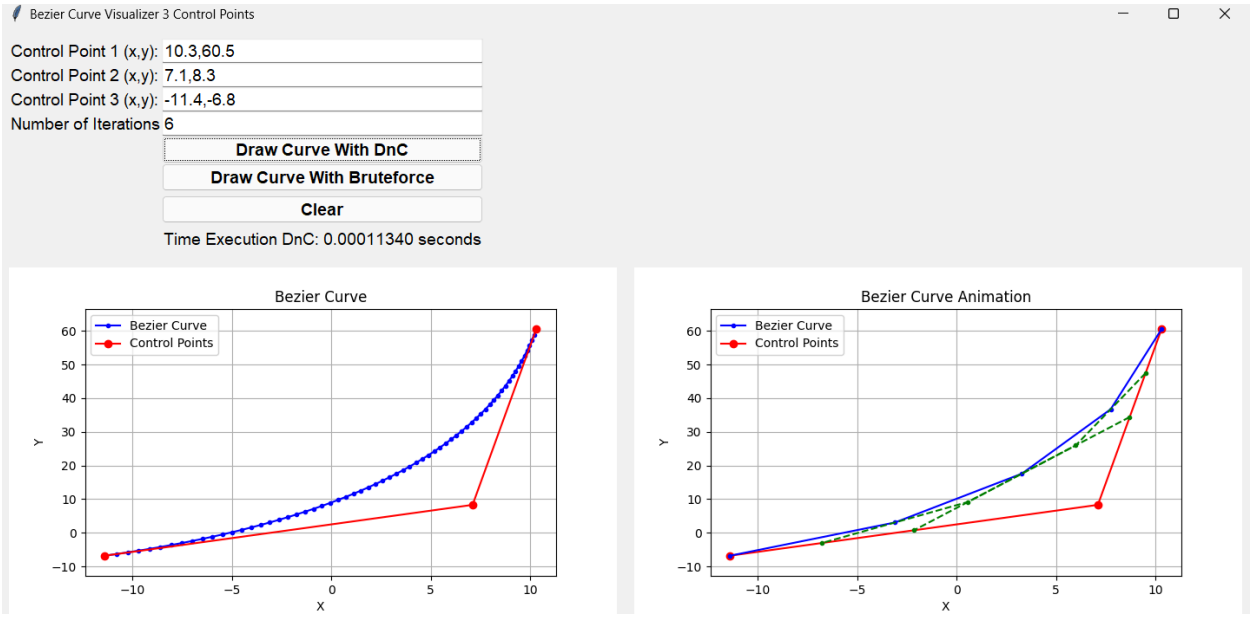
Test case 3



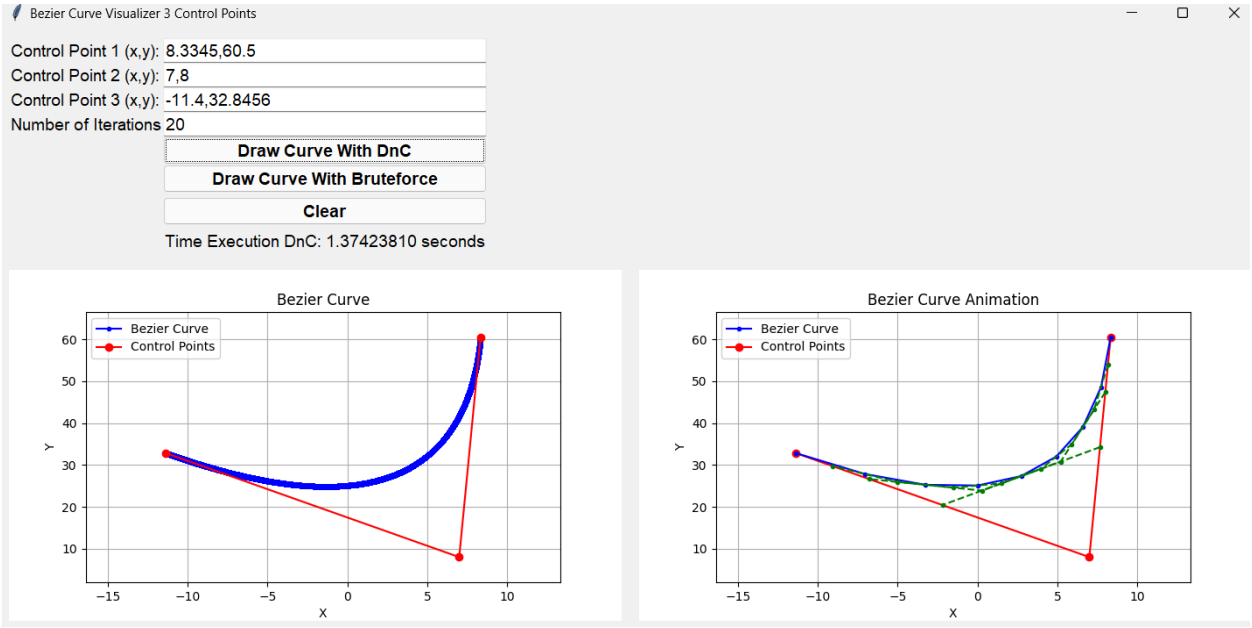
Test case 4



Test case 5

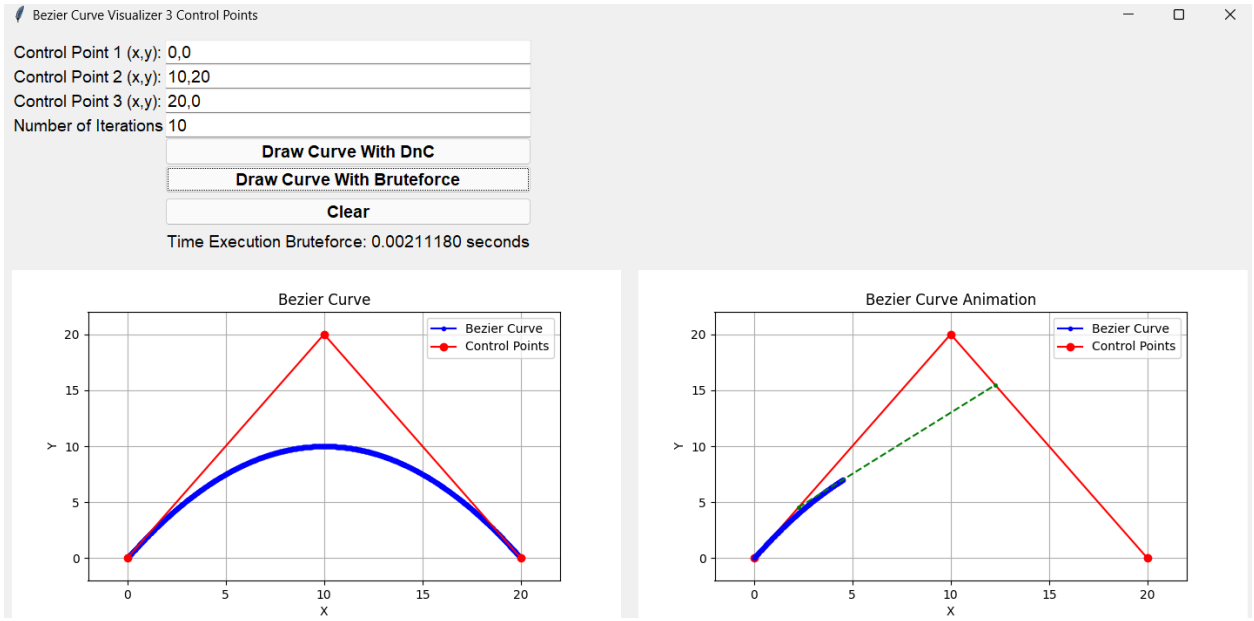


Test case 6

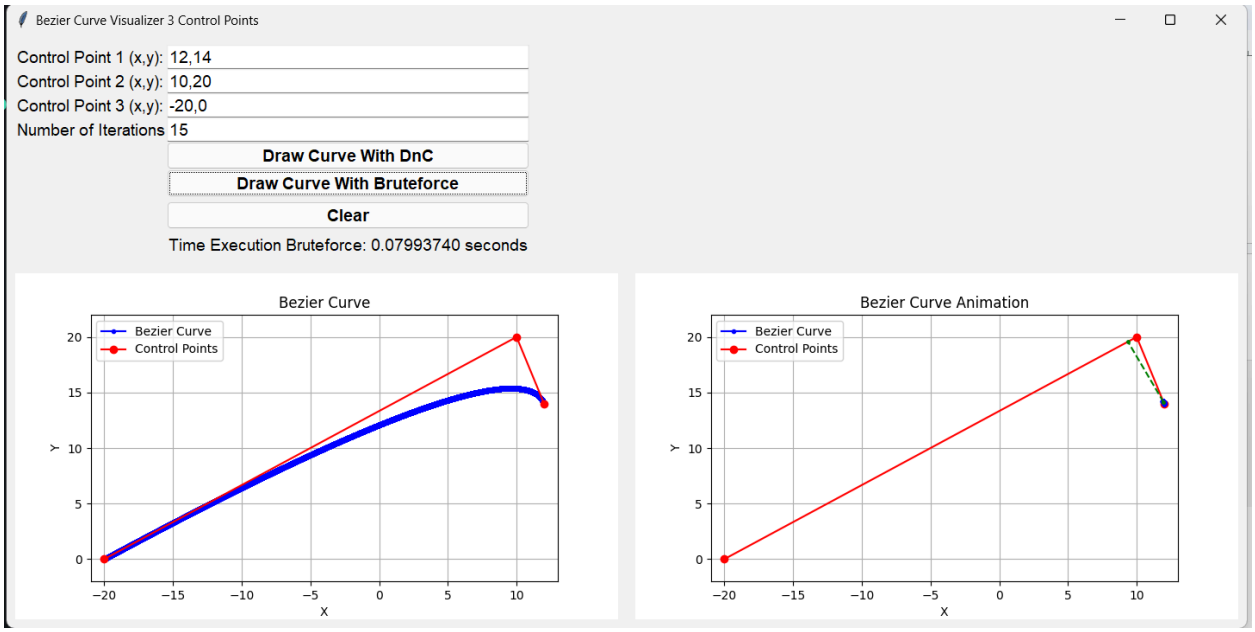


b. Algoritma Bruteforce 3 Titik Kontrol

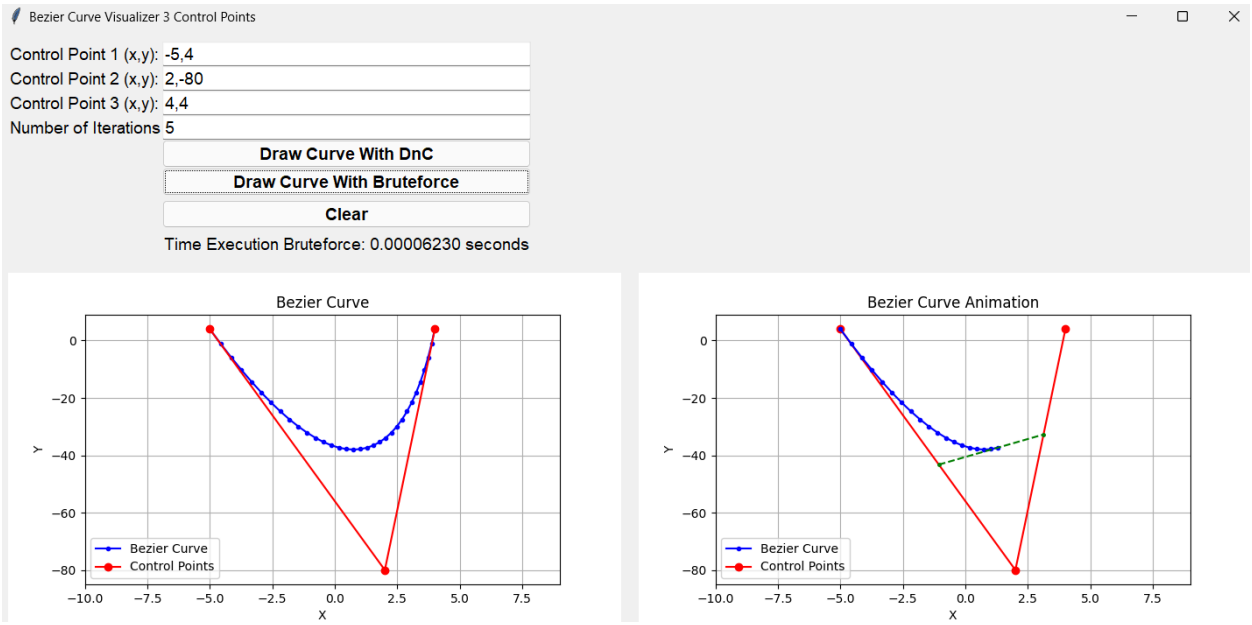
Test case 1



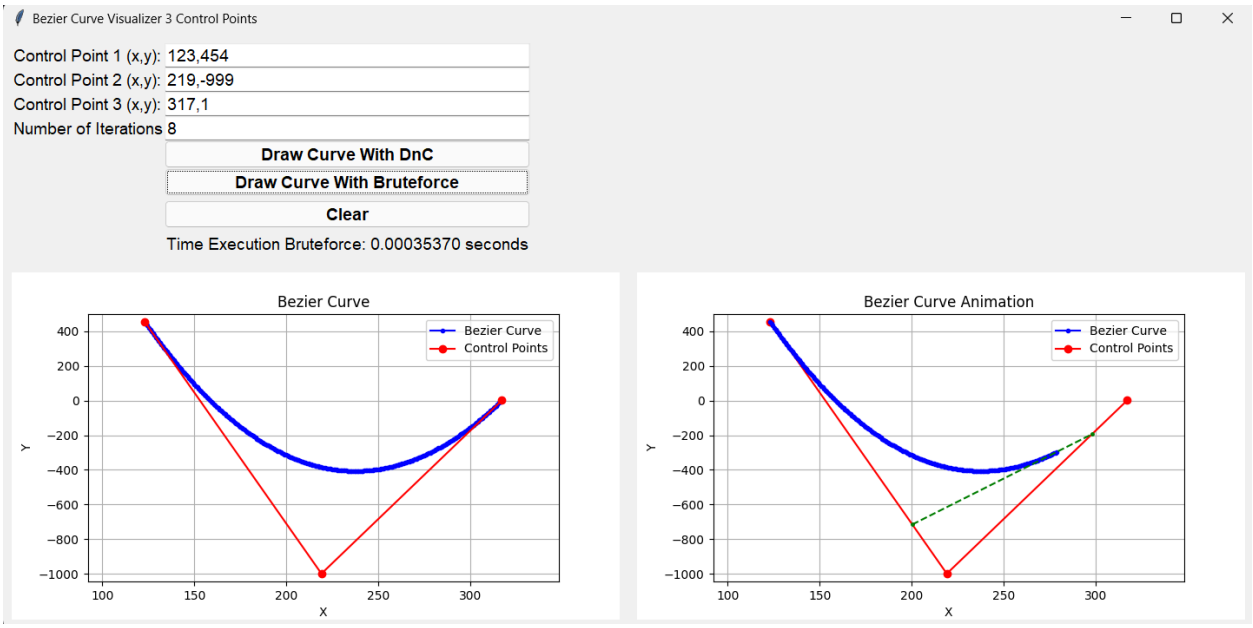
Test case 2



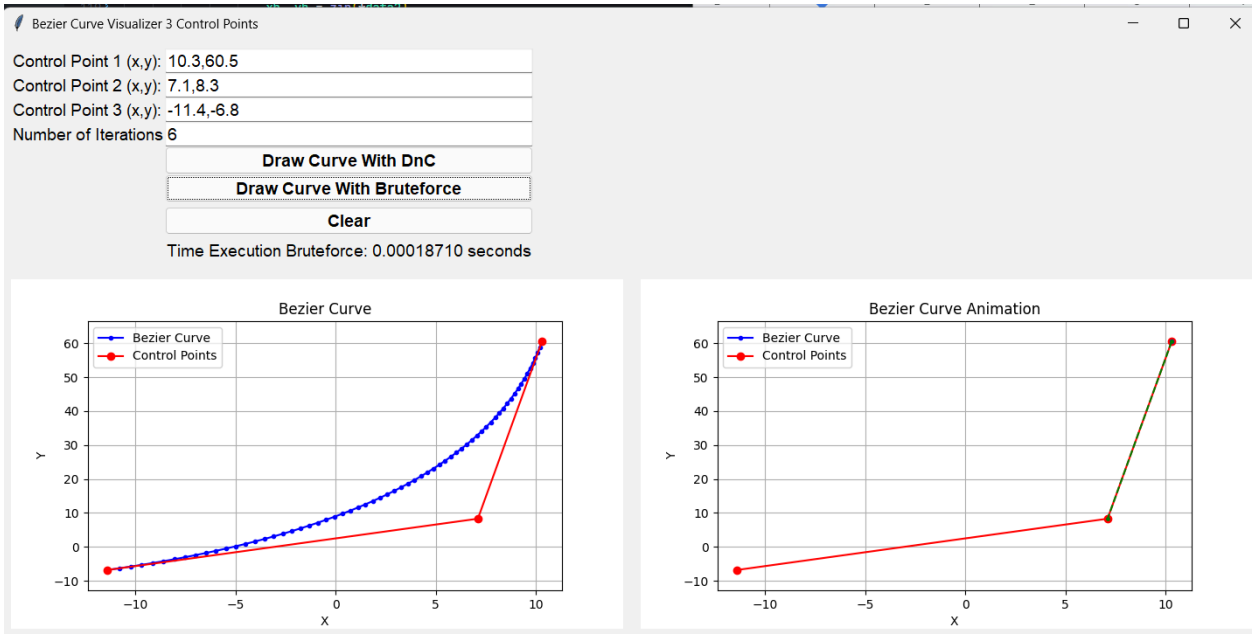
Test case 3



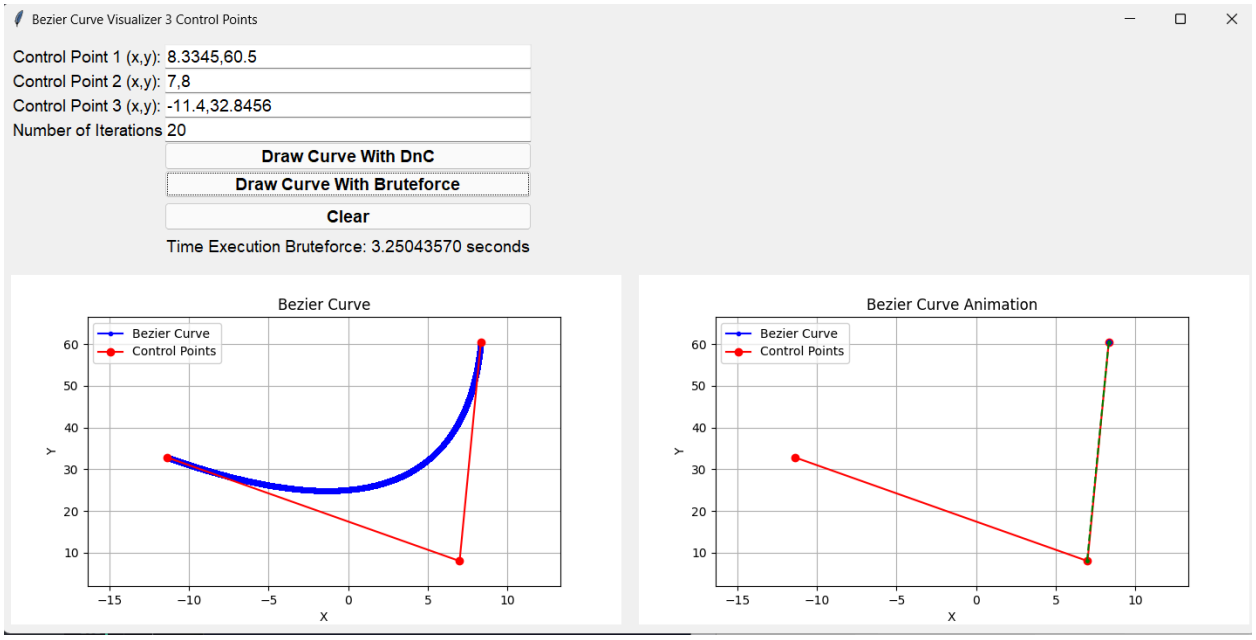
Test case 4



Test case 5

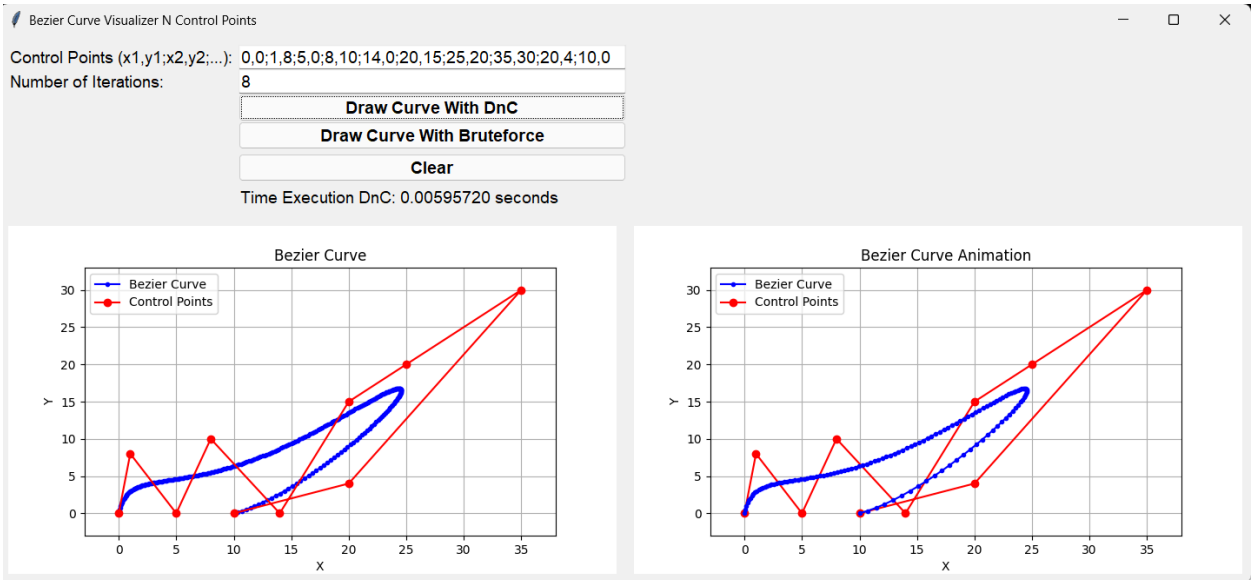


Test case 6

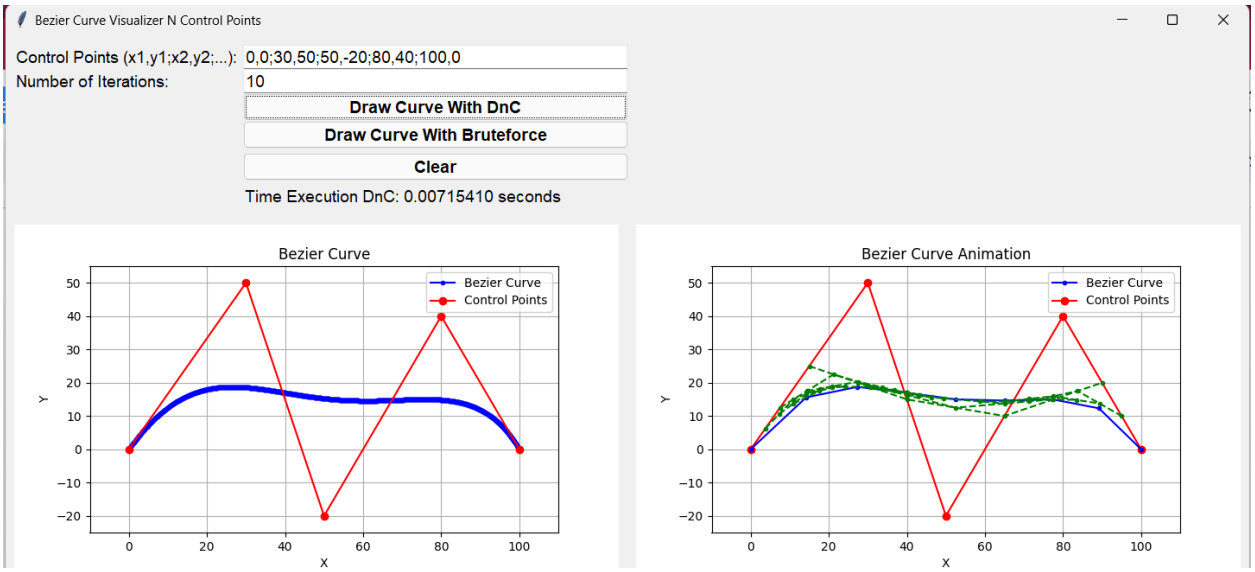


c. Algoritma Divide and Conquer n Titik Kontrol

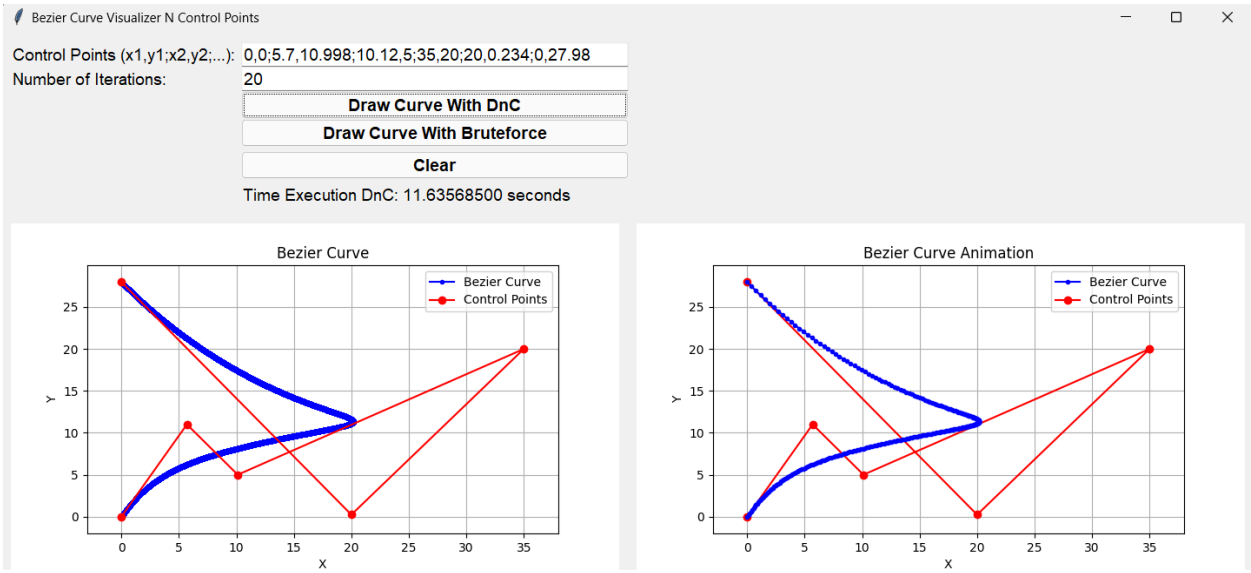
Test case 1 ((0,0),(1,8),(5,0),(8,10),(14,0),(20,15),(25,20),(35,30),(20,4),(10,0))



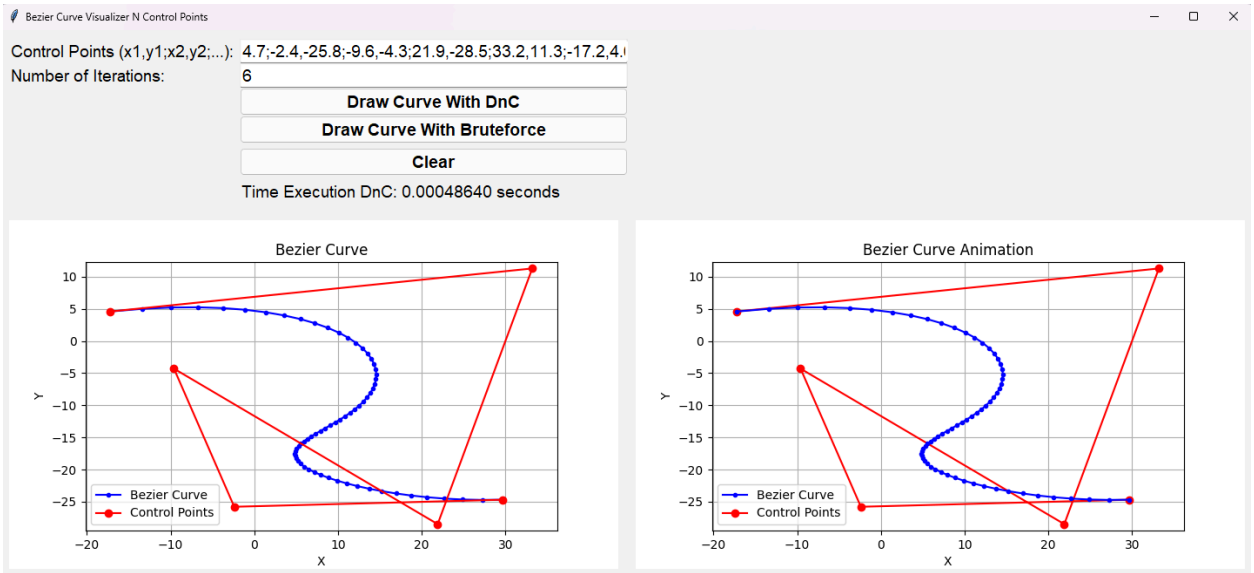
Test case 2 ((0, 0), (30, 50), (50, -20), (80, 40), (100, 0))



Test case 3 ((0, 0), (5.7, 10.998), (10.12, 5), (35,20), (20, 0.234), (0, 27.98))

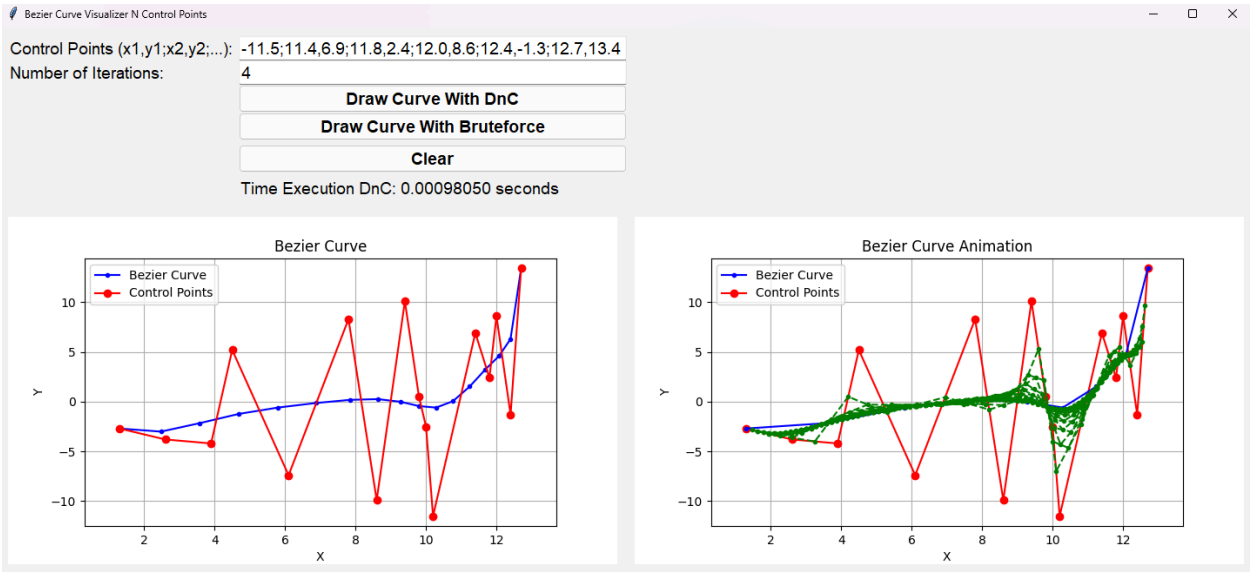


Test case 4 ((29.68,-24.7),(-2.4,-25.8),(-9.6,-4.3),(21.9,-28.5),(33.2,11.3),(-17.2,4.6))

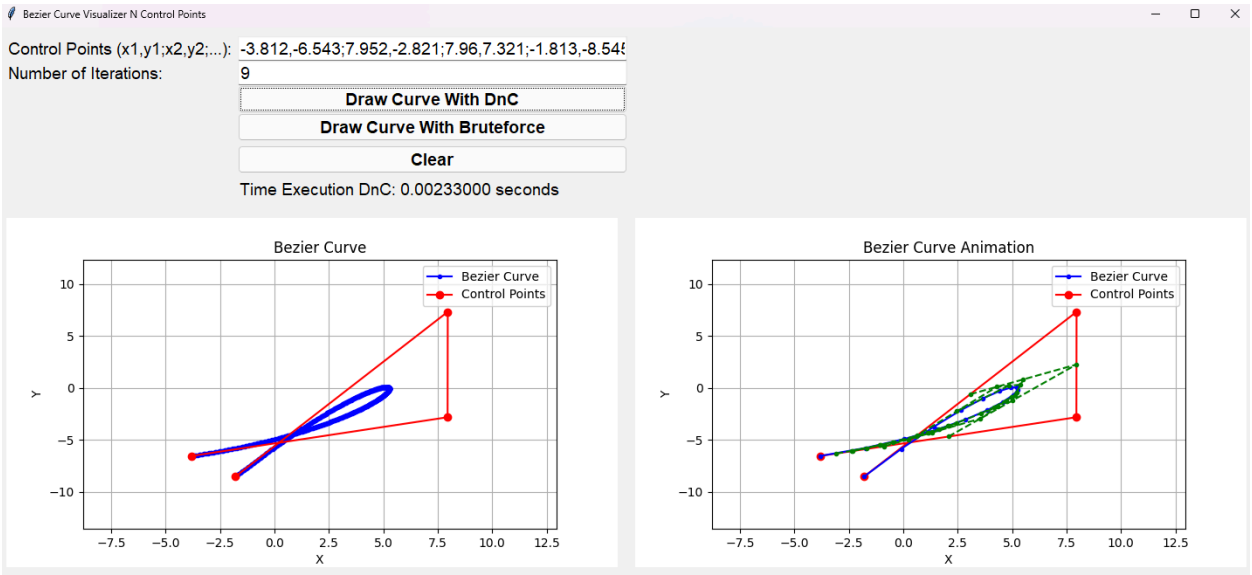


Test case 5

((1.3,-2.7),(2.6,-3.8),(3.9,-4.2),(4.5,5.2),(6.1,-7.4),(7.8,8.3),(8.6,-9.9),(9.4,10.1),(9.8,0.5),(10.0,-2.5),(10.2,-11.5),(11.4,6.9),(11.8,2.4),(12.0,8.6),(12.4,-1.3),(12.7,13.4))

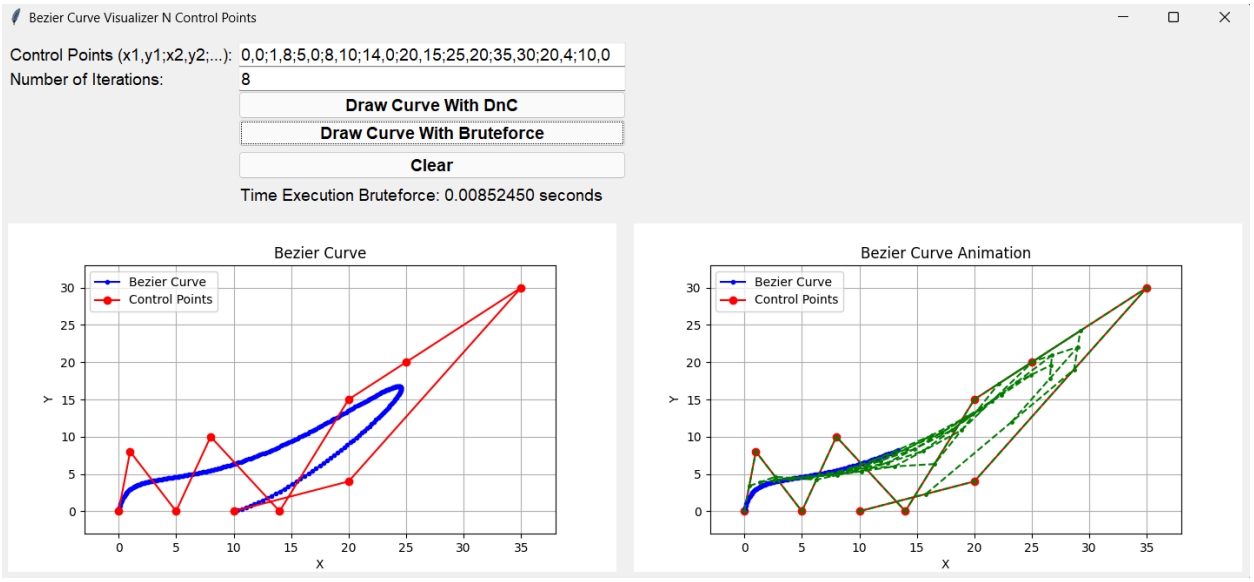


Test case 6 ((-3.812,-6.543),(7.952,-2.821),(7.96,7.321),(-1.813,-8.545))

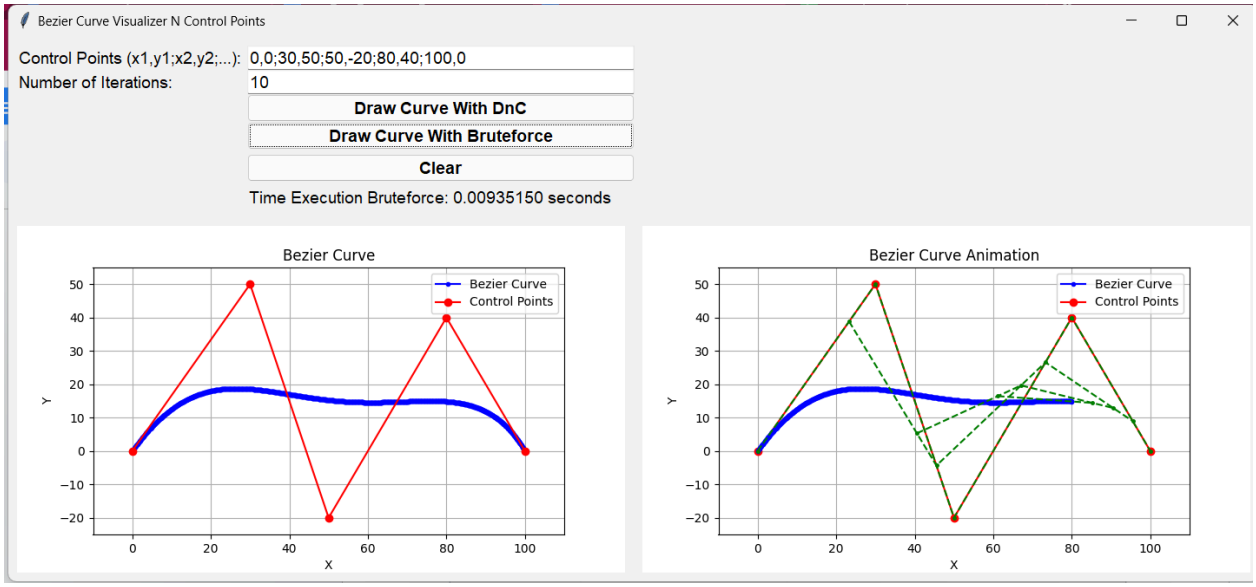


d. Algoritma Bruteforce n Titik Kontrol

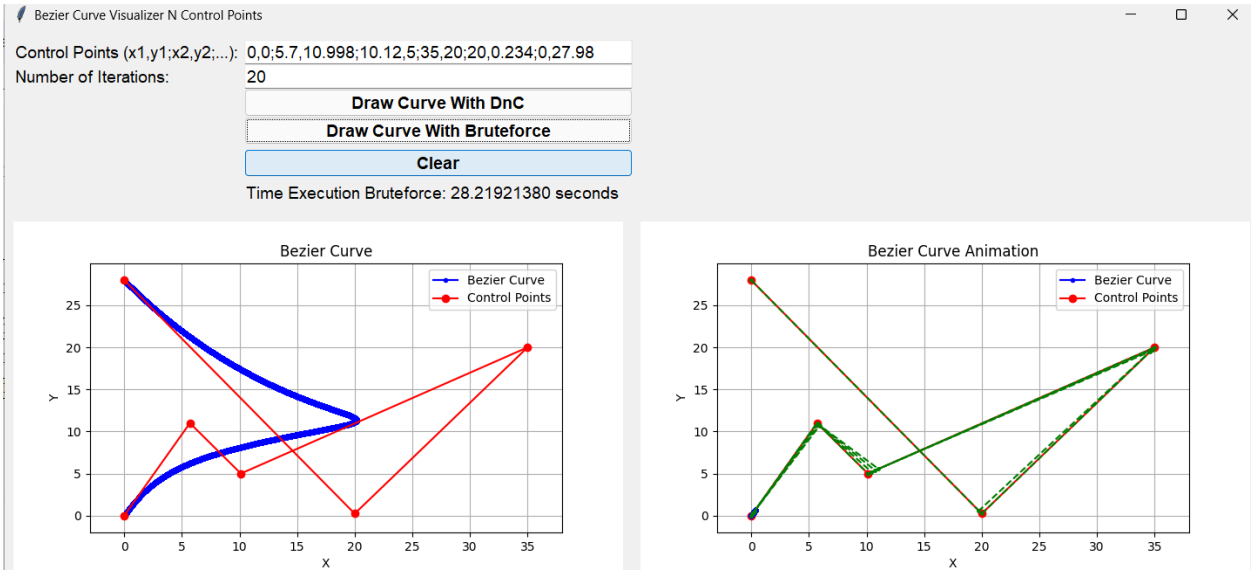
Test case 1((0,0),(1,8),(5,0),(8,10),(14,0),(20,15),(25,20),(35,30),(20,4),(10,0))



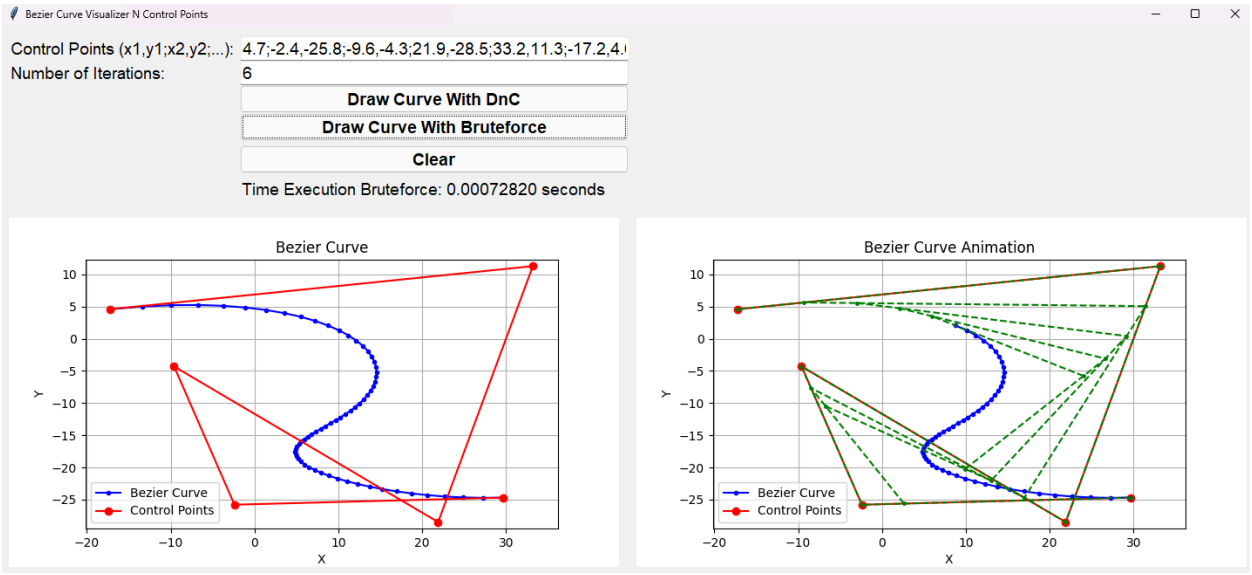
Test case 2 ((0, 0), (30, 50), (50, -20), (80, 40), (100, 0))



Test case 3 ((0, 0), (5.7, 10.998), (10.12, 5), (35,20), (20, 0.234), (0, 27.98))

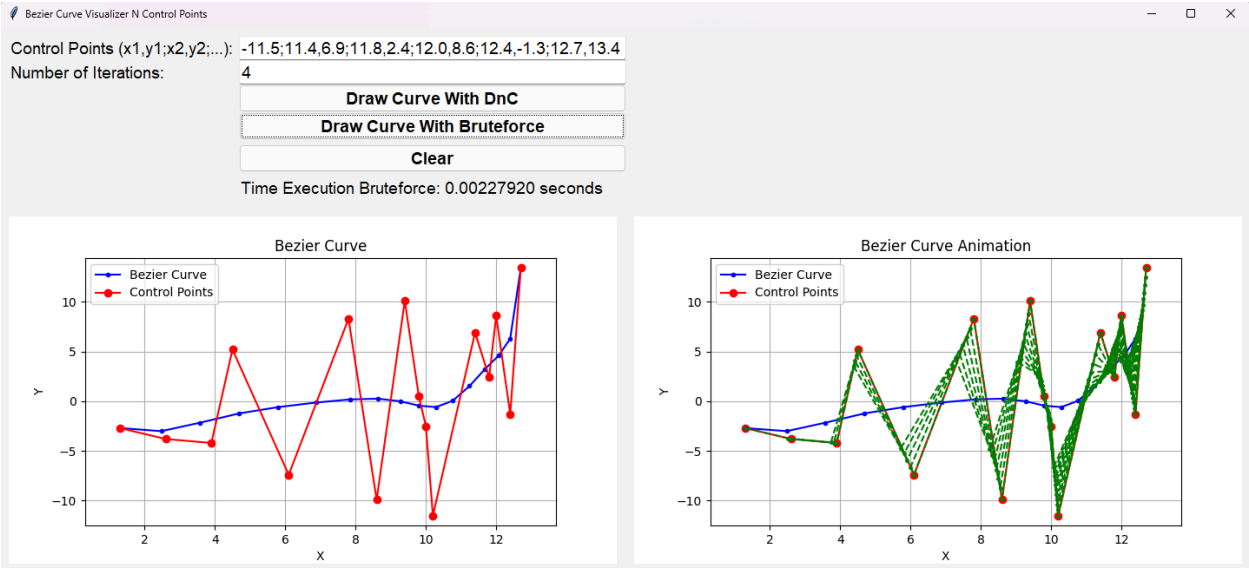


Test case 4 ((29.68,-24.7),(-2.4,-25.8),(-9.6,-4.3),(21.9,-28.5),(33.2,11.3),(-17.2,4.6))

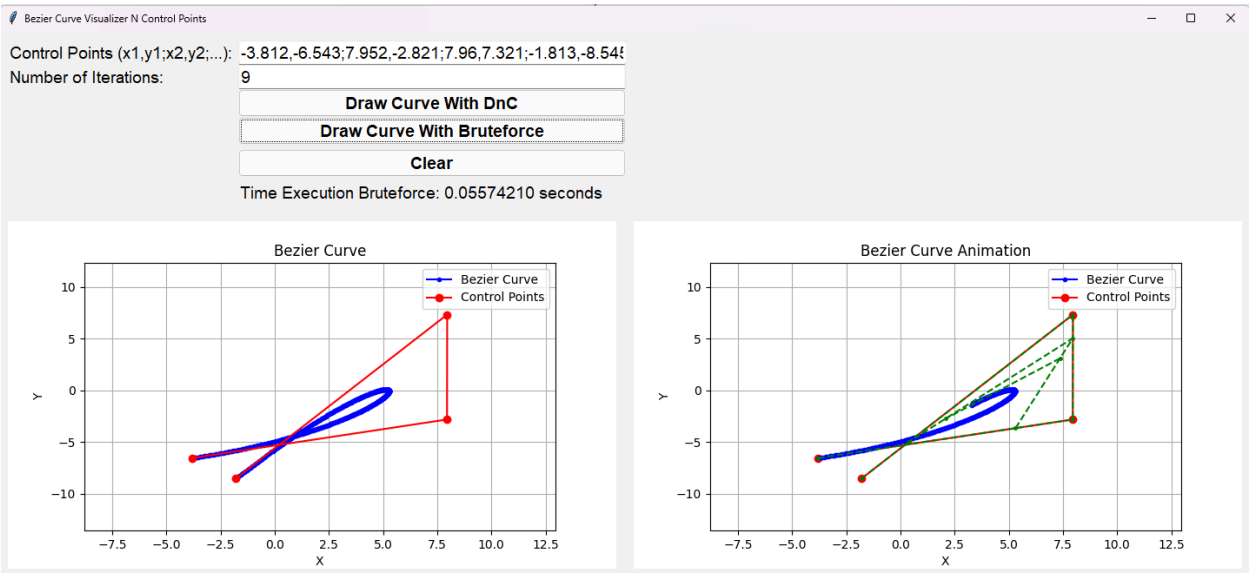


Test case 5

((1.3,-2.7),(2.6,-3.8),(3.9,-4.2),(4.5,5.2),(6.1,-7.4),(7.8,8.3),(8.6,-9.9),(9.4,10.1),(9.8,0.5),(10.0,-2.5),(10.2,-11.5),(11.4,6.9),(11.8,2.4),(12.0,8.6),(12.4,-1.3),(12.7,13.4))

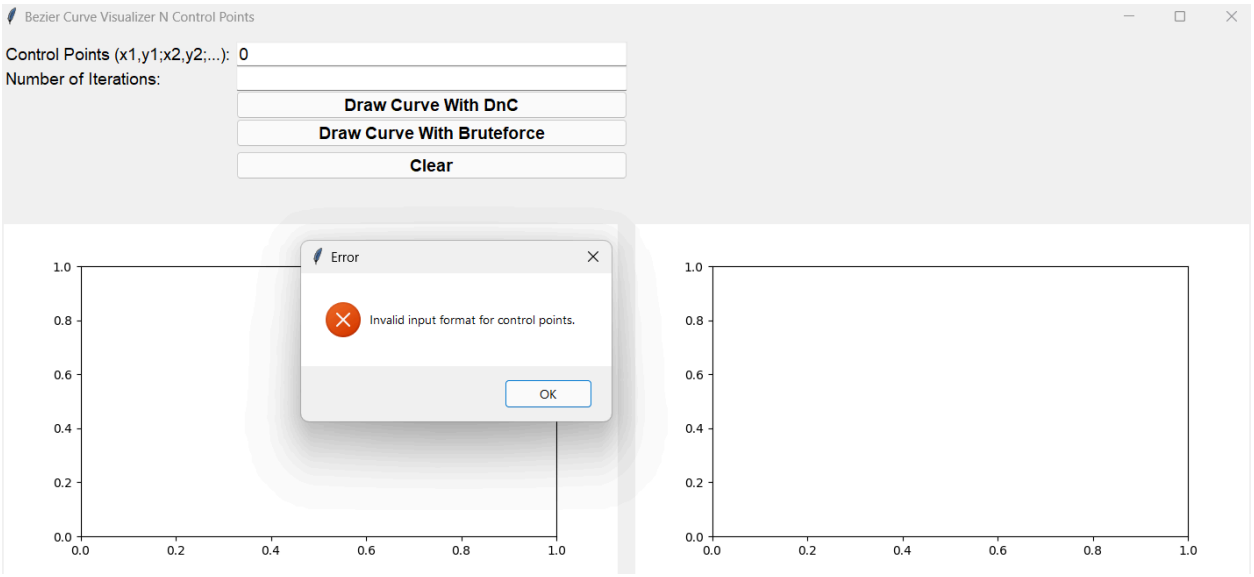


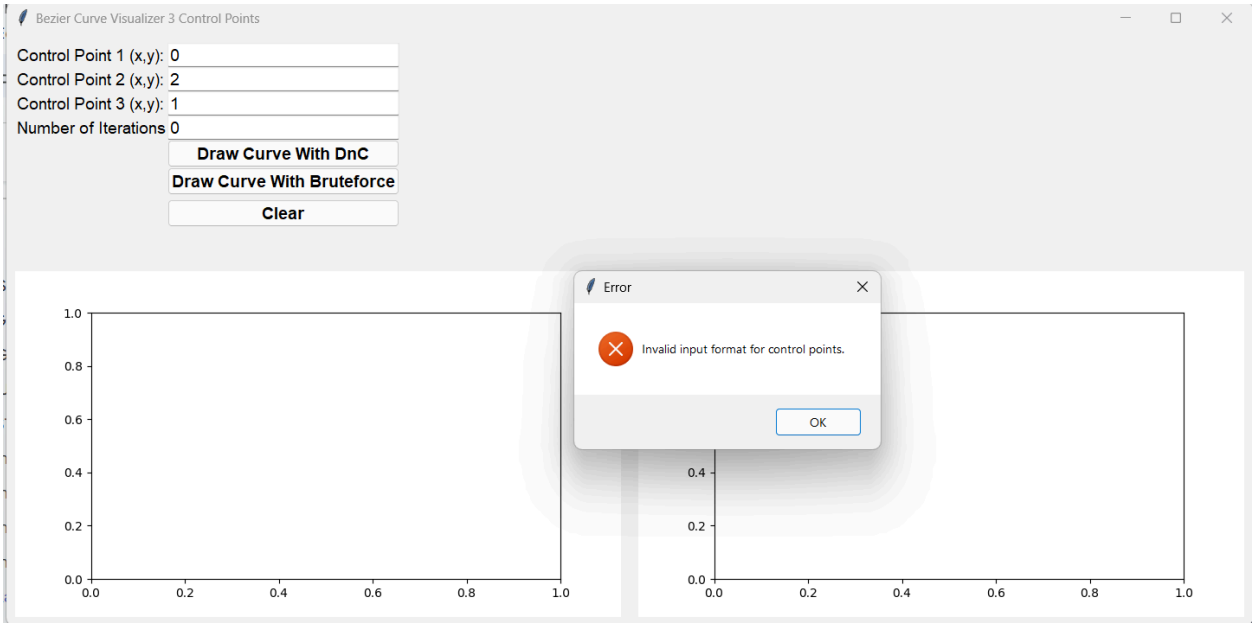
Test case 6 ((-3.812,-6.543),(7.952,-2.821),(7.96,7.321),(-1.813,-8.545))

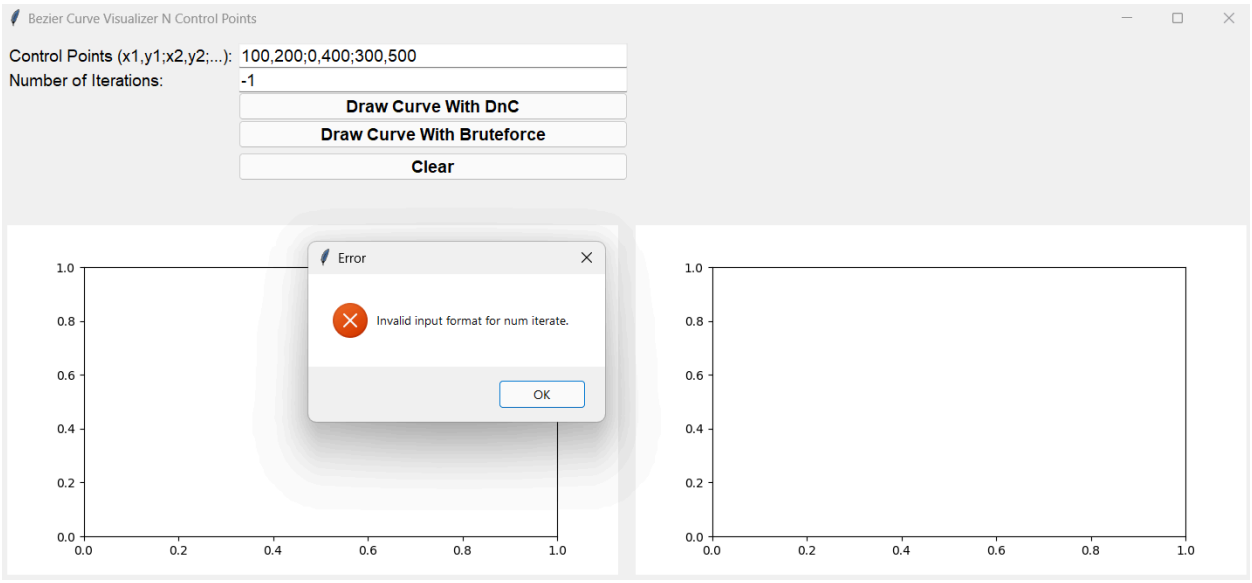


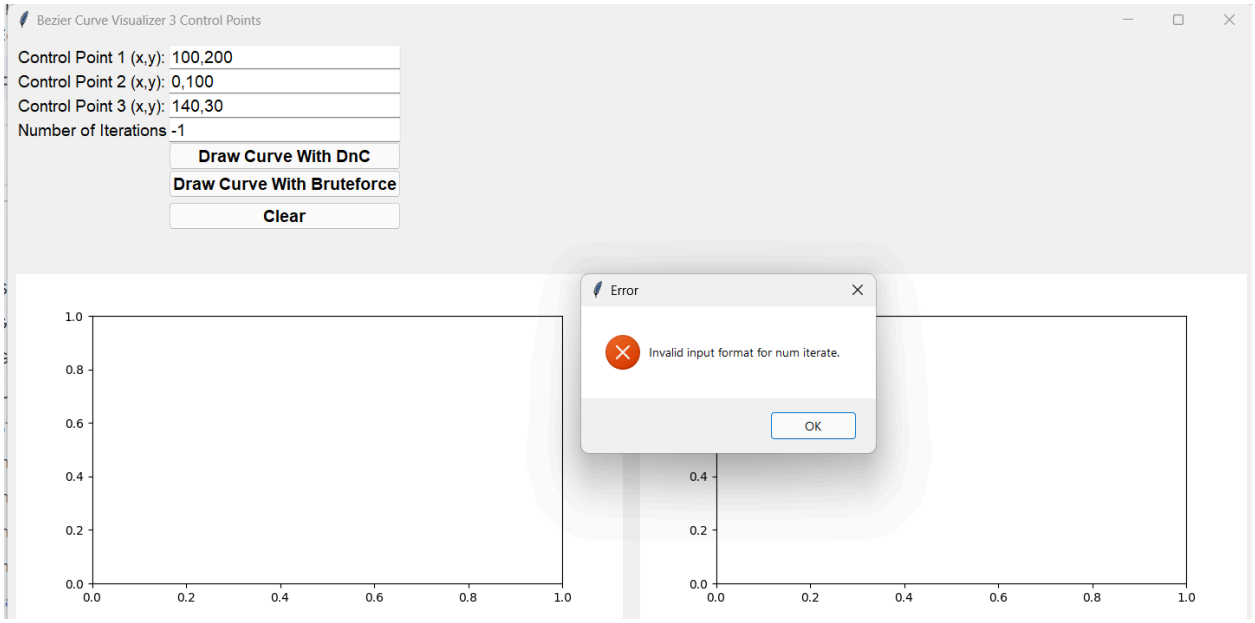
e. Kasus-kasus lain

Masukan tidak valid

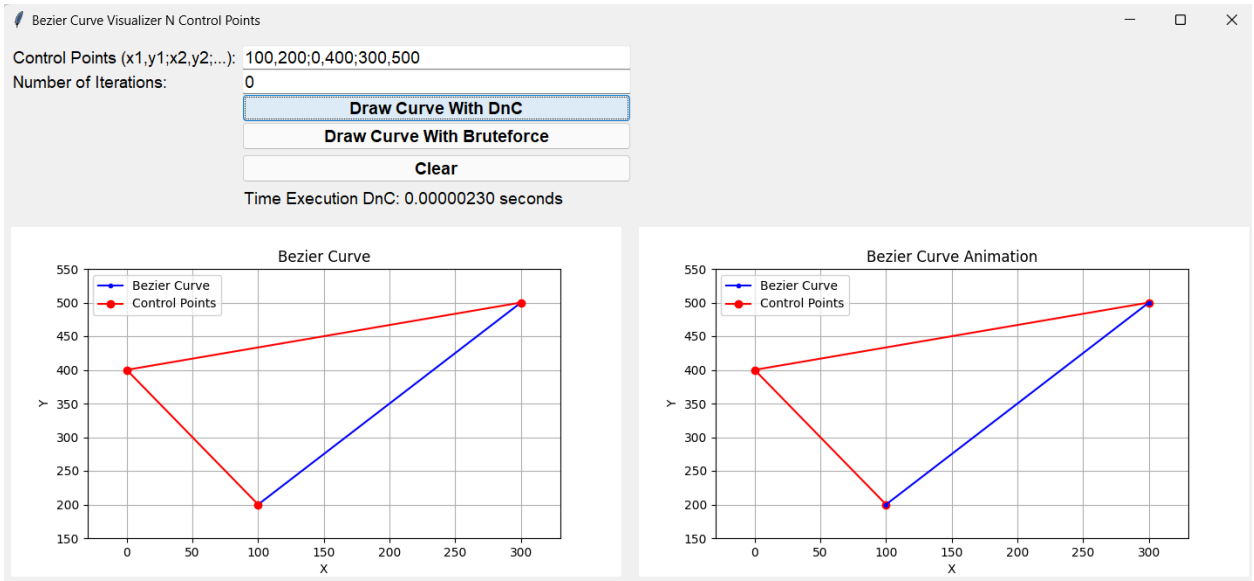

Bezier Curve Visualizer N Control Points
Control Points (x1,y1;x2,y2;...): 0
Number of Iterations:
Draw Curve With DnC
Draw Curve With Bruteforce
Clear
Error
Invalid input format for control points.
OK

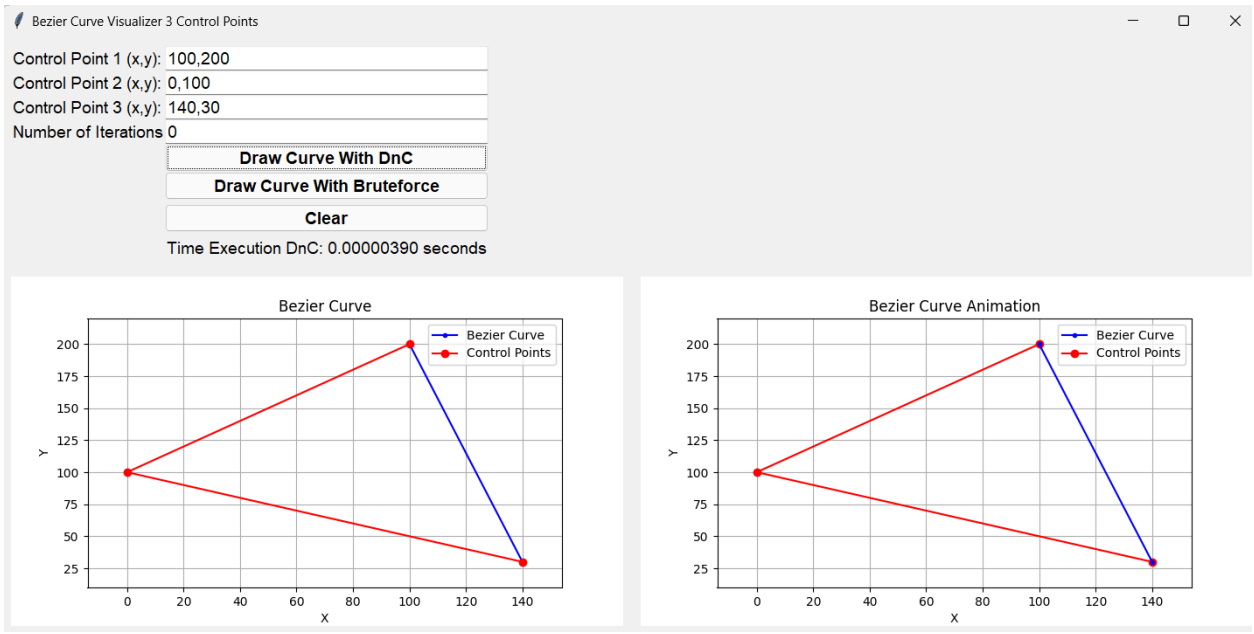

Bezier Curve Visualizer 3 Control Points
Control Point 1 (x,y): 0
Control Point 2 (x,y): 2
Control Point 3 (x,y): 1
Number of Iterations 0
Draw Curve With DnC
Draw Curve With Bruteforce
Clear
Error
Invalid input format for control points.
OK


Bezier Curve Visualizer N Control Points
Control Points (x1,y1;x2,y2;...): 100,200;0,400;300,500
Number of Iterations: -1
Draw Curve With DnC
Draw Curve With Bruteforce
Clear
Error
Invalid input format for num iterate.
OK


Bezier Curve Visualizer 3 Control Points
Control Point 1 (x,y): 100,200
Control Point 2 (x,y): 0,100
Control Point 3 (x,y): 140,30
Number of Iterations -1
Draw Curve With DnC
Draw Curve With Bruteforce
Clear
Error
Invalid input format for num iterate.
OK

Iterasi 0


Bezier Curve Visualizer N Control Points
Control Points (x1,y1;x2,y2;...): 100,200;0,400;300,500
Number of Iterations: 0
Draw Curve With DnC
Draw Curve With Bruteforce
Clear
Time Execution DnC: 0.00000230 seconds
Bezier Curve
Bezier Curve
Control Points
X
Y
Bezier Curve Animation
Bezier Curve
Control Points
X
Y


Bezier Curve Visualizer 3 Control Points
Control Point 1 (x,y): 100,200
Control Point 2 (x,y): 0,100
Control Point 3 (x,y): 140,30
Number of Iterations 0
Draw Curve With DnC
Draw Curve With Bruteforce
Clear
Time Execution DnC: 0.00000390 seconds
Bezier Curve
Bezier Curve
Control Points
X
Y
Bezier Curve Animation
Bezier Curve
Control Points
X
Y

BAB 5
ANALISIS PERBANDINGAN

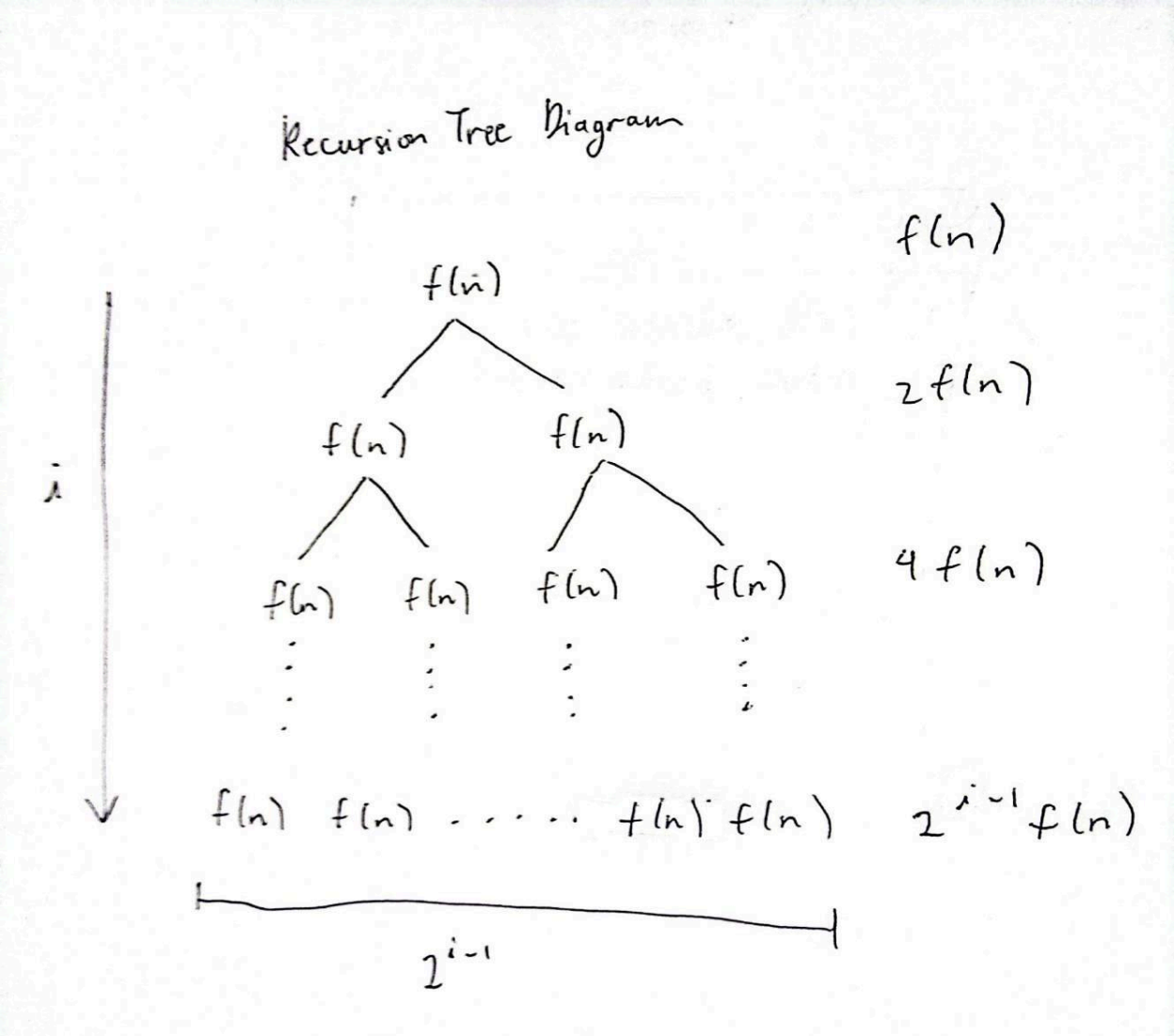
Seperti yang sudah bisa dilihat pada bab 4 test case, secara keseluruhan algoritma dnc dan brute force sudah menghasilkan bentuk kurva yang sama yang menandakan bahwa kedua algoritma tersebut sudah benar. Jika dilihat secara keseluruhan algoritma dnc memiliki kecenderungan untuk memiliki waktu eksekusi yang lebih cepat dibandingkan dengan algoritma brute force. Algoritma brute force terkadang bisa menjadi lebih cepat hanya untuk beberapa kondisi seperti ketika jumlah titik kontrol sedikit dan iterasi sedikit juga. Ketika jumlah iterasi dan titik kontrol semakin banyak maka algoritma DnC akan cenderung lebih cepat. DnC cenderung lebih cepat dikarenakan algoritma dnc membagi masalah menjadi masalah-masalah yang lebih kecil dan independen sehingga proses eksekusi bisa menjadi lebih singkat. Untuk lebih memperjelas algoritma mana yang lebih cepat maka akan dilakukan analisis kompleksitas waktu untuk masing-masing algoritma sebagai berikut

a. Kompleksitas Waktu Algoritma Divide and Conquer

Secara umum, untuk mencari kompleksitas dari algoritma divide and conquer dengan pendekatan rekursif bisa dengan menggunakan bantuan *Teorema Master* namun pada kasus ini *Teorema Master* tidak bisa digunakan dikarenakan kasus DnC pada tugas ini sedikit berbeda dibandingkan dengan kasus-kasus DnC pada umumnya. Pada *Teorema Master* terdapat sebuah syarat dimana b > 1 namun untuk kasus ini nilai dari b adalah satu sehingga bisa dipastikan bahwa *Teorema Master* tidak bisa digunakan disini. Dikarenakan *Teorema Master* tidak bisa digunakan disini maka akan digunakan cara lain untuk menghitung kompleksitas algoritma dari algoritma ini yaitu dengan menggunakan *Recursive Tree Diagram*.

Seperti yang sudah dijelaskan pada bab 2 di setiap pemanggilan fungsi rekursif ini akan terdapat perhitungan titik tengah akhir yang memerlukan 2 kalang loop dalam prosesnya. Pada loop pertama akan dilakukan perulangan sebanyak $n - 1$ kali sedangkan loop kedua akan dilakukan perulangan sebanyak $n - k - 1$ kali. Secara total 2 kalang loop itu akan melakukan perulangan sebanyak $f(n) = n - 1 + n - 2 + n - 3 + ... + 1 = (n(n + 1))/2$ kali. Nilai n disini bermakna jumlah dari titik kontrol yang diberikan. Setelah didapatkan harga atau beban yang

harus dibayarkan untuk menjalankan satu fungsi rekursif ini maka sudah bisa dibuat *Recursive Tree Diagram* yang gambarnya sudah bisa dilihat di bawah ini



Berdasarkan diagram tersebut bisa didapatkan bahwa kompleksitas waktu total dari algoritma DnC ini adalah sebagai berikut

$$T(n,i) = \sum_{a=1}^{i} 2^{a-1} f(n) = (2^i - 1) f(n) = (2^i - 1) \left(\frac{n(n+1)}{2}\right)$$

Dengan nilai *n* adalah banyaknya titik kontrol yang diberikan dan nilai *i* adalah banyaknya jumlah iterasi yang diinginkan. Untuk memperjelas perbandingan kompleksitas antara algoritma

DnC ini dengan algoritma brute force maka kompleksitas waktu akan dibiarkan dalam bentuk *T(n)* dan tidak akan diubah ke dalam bentuk *O(n)*.

Kompleksitas waktu di atas merupakan kompleksitas waktu secara umum (general). Untuk kasus spesifik seperti pada spesifikasi wajib yaitu banyak titik kontrol sudah pasti 3 maka kemungkinan besar hanya nilai *i* saja yang akan berpengaruh besar terhadap kompleksitas waktunya sedangkan nilai *n* tidak akan berpengaruh besar terhadap kompleksitas waktunya dikarenakan nilai *n* yang selalu konstan yaitu 3. Berdasarkan penjelasan tersebut kompleksitas waktu untuk kasus hanya tiga titik kontrol saja adalah sebagai berikut

$$T(i) = 2^i - 1$$

b. Kompleksitas Waktu Algoritma Brute Force

Algoritma brute force dilakukan dengan cara melakukan perulangan rumus persamaan sebanyak titik-titik pada kurva bezier yang ingin didapatkan. Seperti yang sudah dijelaskan pada bab 1 secara umum algoritma brute force ini akan memerlukan 3 kalang loop dalam proses pencarian titik-titik pada kurva beziernya. Loop pertama akan dilakukan sebanyak *m* dimana *m* menunjukkan banyaknya titik yang akan dicari, loop kedua akan dilakukan sebanyak *n - 1* kali dimana *n* mewakili banyaknya titik kontrol yang diberikan, sedangkan untuk loop ketiga akan dilakukan sebanyak *n - k - 1* kali. Secara total banyak loop yang akan dilakukan adalah sebanyak *m((n(n-1))/2)* kali. Nilai *m* dipengaruhi oleh nilai *i* dimana nilai *i* mewakili banyaknya jumlah iterasi yang diberikan. Hubungan antara *m* dan *i* adalah sebagai berikut $m = 2^i + 1$. Berdasarkan persamaan sebelumnya maka bisa dituliskan bahwa kompleksitas waktu total dalam algoritma ini adalah sebagai berikut

$$T(n,i) = (2^i + 1)\left(\frac{n(n+1)}{2}\right)$$

Untuk kasus hanya diberikan 3 titik kontrol saja (spesifikasi wajib) nilai *n* tidak akan berpengaruh terhadap perhitungan kompleksitasnya dikarenakan nilai *n* selalu konstan. Oleh sebab itu, untuk kasus pada spesifikasi wajib algoritma brute force untuk membentuk kurva bezier akan memiliki kompleksitas waktu yaitu sebagai berikut

$$T(i) = 2^i + 1$$

Pada kasus t = 0 dan t = 1 itu sebenarnya adalah titik kontrol pertama dan titik kontrol terakhir. Oleh sebab itu kedua titik itu tidak perlu dihitung lagi dan sudah bisa langsung digunakan. Dikarenakan kedua titik awal dan akhir pada kurva bezier tidak perlu diperhitungkan lagi maka kompleksitas waktu di atas bisa menjadi lebih kecil dengan cara dikurangi 2 langkah iterasi sehingga bentuk persamaannya akan menjadi seperti berikut

$$T(n, i) = (2^i - 1)\left(\frac{n(n+1)}{2}\right) \qquad T(i) = 2^i - 1$$

c. Analisis lanjutan

Jika kita lihat hasil dari perhitungan kompleksitas algoritma di atas algoritma dnc maupun algoritma brute force memiliki kompleksitas waktu yang sama baik secara general maupun hanya untuk kasus tiga titik kontrol saja. Jika kita bandingkan hasil dari test case dengan hasil dari perhitungan kompleksitas algoritma disini maka akan ditemukan anomali dimana pada kasus nyata algoritma DnC cenderung lebih cepat dibandingkan dengan brute force sedangkan secara teoritis bruteforce dan DnC seharusnya mempunyai kecepatan yang relatif sama.

Hal tersebut bisa terjadi dikarenakan terdapat faktor-faktor lain dalam kasus nyata yang tidak bisa dimasukkan ke dalam perhitungan teoritis. Algoritma brute force bisa saja lebih lama dikarenakan terdapat lebih banyak overhead yang terjadi dibandingkan dengan algoritma dnc. Overhead yang dimaksud disini adalah waktu tambahan yang diperlukan untuk melakukan sesuatu seperti inisialisasi memori, waktu akses memori, jeda pemanggilan fungsi, dan lain-lain. Hal-hal seperti ini juga sangat dipengaruhi oleh bahasa yang kita gunakan, tentang bagaimana cara sebuah bahasa ini memperlakukan array, tentang bagaimana sistem caching pada suatu

bahasa bekerja, tentang bagaimana interpreter/compiler dari suatu bahasa melakukan optimalisasi terhadap kode yang dibuat dan masih banyak lagi.

Dalam perhitungan kompleksitas yang dilakukan diatas juga tidak dipertimbangkan kompleksitas untuk operasi-operasi matematika yang digunakan dalam bahasa python. Semua operasi matematika yang digunakan baik perkalian, pertambahan, maupun perpangkatan diasumsikan hanya memiliki kompleksitas *O(1)* namun pada kenyataannya belum tentu kompleksitasnya memang benar segitu, bisa saja berbeda tergantung dari bagaimana cara python mengurus permasalahan tersebut. Waktu eksekusi juga sangat bergantung dengan device atau mesin yang digunakan untuk menjalankan program, keadaan CPU, GPU, RAM, dan lain-lain yang terdapat pada sebuah device atau mesin juga akan mempengaruhi waktu eksekusi pada suatu program.

Perbedaan data secara teoritis dan kasus nyata juga bisa terjadi dikarenakan implementasi teori ke dalam bentuk kode yang sesungguhnya masih kurang tepat. Pemilihan struktur data, bahasa pemrograman, dan struktur program yang akan digunakan dalam implementasi teori juga akan sangat berpengaruh terhadap hasil dari program yang dibuat. Dalam tugas ini mungkin implementasi kode yang dibuat hanya berorientasi pada hasil dimana yang penting hasil secara teori dan secara implementasi kode sudah sama namun proses untuk mencapai hasil tersebut mungkin akan sedikit berbeda dan masih kurang tepat sehingga membuat perbedaan data waktu.

BAB 6
BONUS

a. Bonus Algoritma Divide and Conquer dengan N titik kontrol

Algoritma Divide and Conquer untuk N titik kontrol mengikuti pola umum dari pendekatan divide and conquer yang digunakan pada kurva Bezier dengan tiga titik kontrol atau jumlah titik yang lebih sedikit. Namun, fokusnya lebih pada langkah penentuan titik tengah akhir dari serangkaian titik-titik tengah yang berasal dari titik kontrol. Walaupun semakin banyaknya titik kontrol, proses ini dapat diatasi dengan menggunakan strategi brute force berdasarkan pola yang ditemukan dalam analisis yang dilakukan. Strategi brute force yang dimaksud disini adalah brute force untuk mencari titik tengah akhir pada setiap bagian kurva. Secara garis besar algoritma yang digunakan tetaplah divide and conquer.

Meskipun masuk akal bahwa kompleksitasnya akan meningkat seiring dengan bertambahnya titik kontrol, algoritma tetap mengikuti prinsip dasar divide, combine, dan rekurensi dari algoritma divide and conquer untuk menyusun kurva Bezier dengan titik kontrol tertentu. Ini memungkinkan algoritma untuk diimplementasikan secara efisien, bahkan dengan jumlah titik kontrol yang besar.

Dengan demikian, algoritma ini memberikan solusi yang efektif untuk menghadapi permasalahan kurva Bezier dengan N titik kontrol, dengan mengambil manfaat dari pola dan strategi matematis yang ditemukan dalam analisis algoritma divide and conquer.

b. Bonus Visualisasi Proses Pembentukan kurva Bezier

Visualisasi dari proses pembentukan kurva Bezier ini diimplementasikan dengan menggunakan library tkinter dalam pembuatan GUI dan animasi dibuat dengan menggunakan library matplotlib.

Proses pembuatan kurva dimulai dengan menerima masukan / input dari user yaitu dengan menginput titik kontrol sebanyak 3 atau n titik, lalu user memasukan input berupa jumlah iterasi yang diinginkan. Semakin banyak jumlah iterasi, maka semakin bagus juga kurva yang akan dibentuk. Setelah itu, user dapat memilih untuk membuat kurva dengan 2 metode, yaitu *divide and conquer* atau dengan menggunakan *bruteforce*. User juga bisa mengosongkan kurva yang telah dibentuk sebelumnya dengan menekan tombol *clear*.

Setelah menekan pilihan antara 2 tombol tersebut, maka 2 kurva akan di generate, yang pertama berupa kurva statik yang memiliki beberapa titik bezier sesuai jumlah iterasi dan kurva kedua berupa animasi cara pembentukan kurva bezier. Proses pembuatan animasi kurva dibuat dengan menggunakan library *matplotlib.animation* dengan menggunakan fungsi *FuncAnimation*. Animasi yang diberikan juga akan menampilkan satu per satu pembuatan titik bezier baik pada algoritma *divide and conquer* maupun *bruteforce*

Setelah user menekan tombol pembuatan kurva, program juga akan menampilkan waktu eksekusi dari pencarian titik bezier. Semakin banyak jumlah iterasi yang dimasukkan maka akan semakin lama pula waktu eksekusi program yang dijalankan.

BAB 7
PENUTUP

a. Kesimpulan

Pembuatan kurva bezier bisa dilakukan baik dengan dengan cara brute force maupun divide and conquer. Berdasarkan analisis yang dilakukan penggunaan algoritma divide and conquer tidak selalu membuat pembentukan kurva bezier menjadi lebih cepat jika dibandingkan dengan menggunakan algoritma brute force. Dalam prakteknya penggunaan DnC memang cenderung lebih cepat namun hal tersebut dipengaruhi oleh terlalu banyak faktor yang masih tidak bisa dipastikan kebenarannya. Melalui tugas ini dapat diketahui bahwa tidak semua masalah bisa dibuat lebih cepat dengan algoritma DnC terkadang untuk kasus-kasus tertentu penggunaan algoritma brute force bisa menjadi pilihan yang lebih baik. Dalam kasus ini mungkin saja DnC bisa menjadi pilihan yang lebih cepat jika proses yang sudah dibagi bisa dieksekusi secara paralel.

b. Saran

1. Pengujian bisa dilakukan dengan contoh kasus yang lebih banyak untuk lebih dapat memastikan efisiensi dari program.
2. Bisa dilakukan pengujian dengan membuat program dengan bahasa yang berbeda dan melihat hasilnya pada suatu bahasa tersebut.

LAMPIRAN

Spesifikasi Tugas

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.	✓	

Link Repository

https://github.com/Otzzu/Tucil2_13522047_13522117

DAFTAR PUSTAKA

Munir, Rinaldi. “Algoritma Brute Force (Bagian 1).” Informatika. Diakses 17 Maret 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

Munir, Rinaldi. “Algoritma Divide and Conquer (Bagian 1).” Informatika. Diakses 17 Maret 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf

Munir, Rinaldi. “Algoritma Divide and Conquer (Bagian 2).” Informatika. Diakses 17 Maret 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf

Munir, Rinaldi. “Algoritma Divide and Conquer (Bagian 3).” Informatika. Diakses 17 Maret 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf

Munir, Rinaldi. “Algoritma Divide and Conquer (Bagian 4).” Informatika. Diakses 15 Maret 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf