TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA
Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Disusun Oleh :

Jason Fernando 13522156

PROGRAM STUDI TEKNIK INFORMATIKA SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT TEKNOLOGI BANDUNG 2024

DAFTAR ISI

BAB I
DESKRIPSI MASALAH DAN ALGORITMA

1.1 Algoritma Divide and Conquer

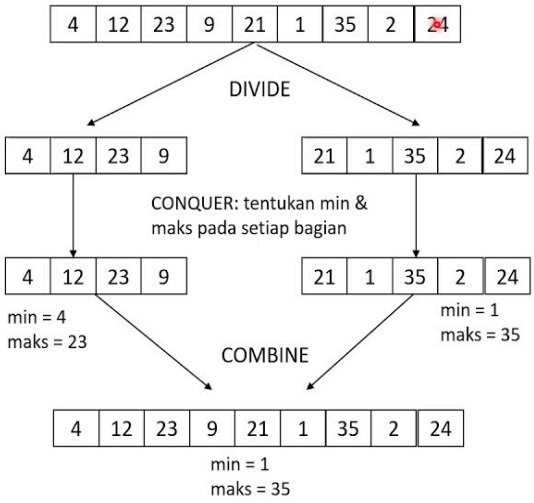
Algoritma Divide and Conquer merupakan pendekatan esensial dalam dunia pemrograman untuk menyelesaikan masalah-masalah yang kompleks. Metode ini terbagi menjadi tiga tahap utama: Divide, Conquer, dan Combine. Tahap pertama, Divide, melibatkan pemecahan masalah awal menjadi sub-masalah yang lebih kecil dan seragam. Penting untuk memastikan bahwa setiap sub-masalah memiliki karakteristik yang serupa agar pendekatan ini efektif. Tahap kedua, Conquer, menerapkan solusi untuk setiap sub-masalah secara terpisah. Sub-masalah yang sederhana dapat diselesaikan secara langsung, sedangkan yang lebih kompleks mungkin memerlukan pendekatan rekursif. Tahap terakhir, Combine, melibatkan penggabungan solusi dari setiap sub-masalah untuk membentuk solusi keseluruhan.

Metode Divide and Conquer secara alami mendukung pendekatan rekursif karena struktur masalah yang seragam dan terpecah secara bertahap. Proses ini memungkinkan pemecahan masalah secara efisien dengan membagi masalah besar menjadi sub-masalah yang lebih kecil dan lebih mudah dipecahkan. Banyak masalah klasik dalam ilmu komputer yang dapat diselesaikan menggunakan metode ini, termasuk pencarian nilai minimum dan maksimum, pengurutan data, mencari pasangan titik terdekat, dan banyak lagi. Pendekatan ini telah terbukti memberikan solusi yang efisien dan skalabel untuk berbagai masalah kompleks.

Dalam konteks laporan, metode Divide and Conquer memainkan peran penting dalam meningkatkan efisiensi dan kinerja pemecahan masalah komputasi. Dengan memanfaatkan pembagian masalah menjadi sub-masalah yang lebih kecil dan lebih mudah dipecahkan, algoritma ini dapat menangani masalah yang kompleks dengan lebih efisien. Keserbagunaan dan keefektifan metode ini telah terbukti dalam berbagai aplikasi praktis, memperkuat reputasinya sebagai salah satu pendekatan terdepan dalam dunia pemrograman. Dengan demikian, pemahaman yang mendalam tentang metode Divide and Conquer menjadi penting bagi para praktisi dan peneliti di bidang ilmu komputer.

Penyelesaian dengan *algoritma divide and conquer*

Ide dasar secara *divide and conquer*:



Gambar 1.1 algoritma divide and conquer

1.2 Algortima Brute Force

Algoritma Brute Force adalah pendekatan yang sederhana dan langsung dalam menyelesaikan masalah dengan mengeksplorasi setiap kemungkinan solusi tanpa memerlukan pengetahuan khusus tentang struktur masalah. Contohnya, dalam mencari elemen terbesar dalam sebuah daftar, setiap elemen dibandingkan satu per satu untuk menemukan yang terbesar. Namun, kekurangan utamanya terletak pada keterbatasan efisiensi karena membutuhkan komputasi besar dan waktu yang lama, terutama untuk ukuran masukan yang besar. Meskipun demikian, keunggulannya terletak pada kemampuannya untuk menyelesaikan sebagian besar masalah dengan pendekatan yang mudah dimengerti.

Karakteristik algoritma Brute Force mencerminkan kelebihan dan kelemahannya. Meskipun cenderung tidak memberikan solusi yang optimal atau efisien, algoritma ini dapat diterapkan pada berbagai macam masalah, terutama saat ukuran masukan kecil dan implementasinya sederhana. Namun, ketika masukan menjadi besar, kinerjanya seringkali tidak dapat diterima. Walaupun demikian, algoritma Brute Force tetap penting karena sering digunakan sebagai dasar perbandingan dengan algoritma yang lebih canggih dan efisien.

Untuk mengatasi keterbatasan efisiensi algoritma Brute Force, sering kali digunakan teknik heuristik. Teknik ini membantu meningkatkan kinerja dengan mengurangi

beberapa kemungkinan solusi tanpa harus mengeksplorasi semua kemungkinan secara menyeluruh. Meskipun bersifat intuitif dan non-formal, teknik heuristik dapat membantu meningkatkan efisiensi algoritma Brute Force dalam menyelesaikan masalah.

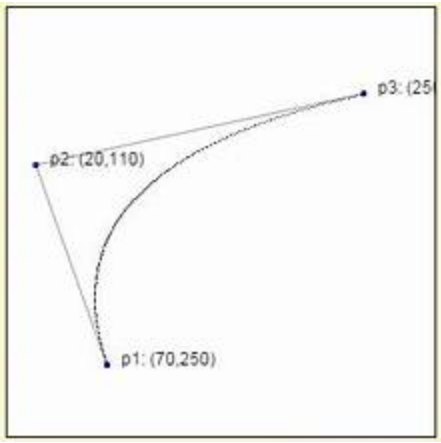
Dalam banyak kasus, algoritma Brute Force digunakan sebagai titik awal dalam memahami dan menyelesaikan masalah yang kompleks. Walaupun mungkin tidak memberikan solusi optimal, algoritma ini tetap bermanfaat dalam pemecahan berbagai masalah. Dengan pendekatan yang jelas dan sederhana, algoritma Brute Force membantu memahami dasar struktur masalah, yang kemudian dapat menjadi landasan untuk pengembangan solusi yang lebih canggih dan efisien.

1.3 Kurva Bezier

Kurva Bézier adalah sebuah kurva halus yang umumnya digunakan dalam berbagai aplikasi seperti desain grafis, animasi, dan manufaktur. Kurva ini terbentuk dengan menghubungkan beberapa titik kontrol yang menentukan bentuk dan arah kurva. Proses pembuatan kurva Bézier relatif mudah, di mana titik-titik kontrol ditentukan dan dihubungkan untuk membentuk kurva. Kegunaan kurva Bézier sangat luas dalam kehidupan sehari-hari, termasuk alat pen, animasi yang realistis, desain produk yang kompleks, dan pembuatan font yang unik.

Salah satu keuntungan utama menggunakan kurva Bézier adalah fleksibilitasnya dalam pengeditan dan manipulasi. Ini memungkinkan desainer untuk menciptakan desain dengan presisi dan sesuai dengan kebutuhan mereka. Kurva Bézier didefinisikan oleh serangkaian titik kontrol, di mana titik awal dan akhir selalu menjadi ujung dari kurva. Namun, titik kontrol di antara kedua ujung tersebut, yang disebut sebagai titik kontrol antara, umumnya tidak terletak langsung pada kurva yang terbentuk.

Dalam definisi kurva Bézier, setiap kurva memiliki satu set titik kontrol P0 hingga Pn, dengan n menunjukkan order kurva tersebut (misalnya, n = 1 untuk kurva linier, n = 2 untuk kurva kuadrat, dan seterusnya). Hal ini menunjukkan fleksibilitas kurva Bézier dalam menyesuaikan tingkat kompleksitas dan kehalusan kurva yang dihasilkan, tergantung pada jumlah titik kontrol yang digunakan.



Gambar 1.2 kurva bezier

Sebuah kurva Bézier didefinisikan oleh serangkaian titik kontrol dari P0 hingga Pn, dengan n mengindikasikan orde kurva (n = 1 untuk garis lurus, n = 2 untuk kuadrat, dan seterusnya). Titik kontrol awal dan akhir selalu menjadi titik ujung dari kurva, tetapi titik kontrol di antara keduanya (jika ada) biasanya tidak langsung berada pada jalur kurva tersebut. Dalam gambar 1, P0 menjadi titik kontrol pertama, sementara P3 adalah titik kontrol terakhir. P1 dan P2, disebut sebagai titik kontrol antara, tidak terletak pada jalur kurva yang terbentuk.

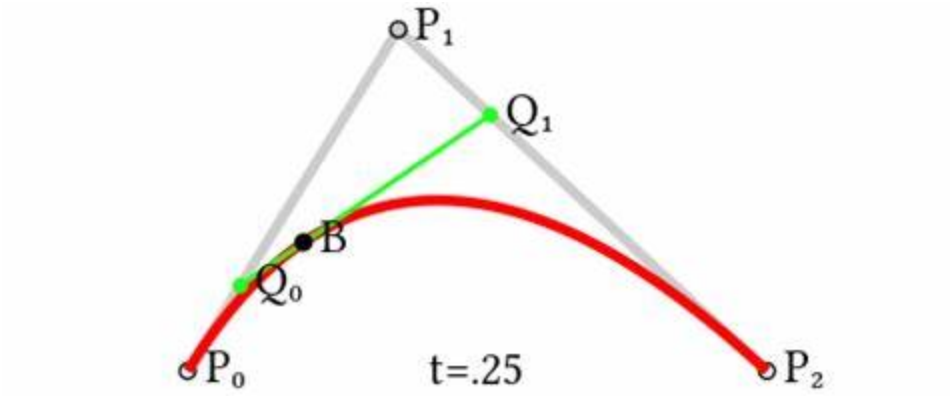
Penerapan kurva Bézier dengan algoritma brute-force pada dasarnya cukup sederhana dari perspektif pemrogram. Kurva Bézier dibentuk menggunakan persamaan:

$$Q0 = B(t) = (1 - t)P0 + tP1,\ t\epsilon[0,\ 1].$$

Di sini, Q0 adalah titik yang berada pada garis yang dibentuk oleh P0 dan P1. Dalam kurva Bézier linier, B(t) dari P0 ke P1 bergantung pada nilai t dalam rentang [0, 1], sehingga nilai B(t) membentuk garis lurus yang menghubungkan P0 dan P1 ketika t berubah dari 0 hingga 1.

Dalam situasi lain, ketika kita menambahkan titik baru, P2, antara P0 dan P1, dengan P0 dan P2 sebagai titik kontrol awal dan akhir, serta P1 sebagai titik kontrol antara, kita dapat membentuk kurva Bézier linier baru dengan P0 sebagai titik awal dan Q0 sebagai titik akhir. Dengan menemukan titik baru, R0, di antara garis yang menghubungkan Q0 dan Q1, kita dapat membentuk kurva Bézier kuadratik baru dengan P0 dan P2 sebagai titik awal dan akhir. Persamaan untuk ini adalah

$$R0 = B(t) = (1 - t)^2\ P0 + 2(1 - t)tP1 + t^2\ P2,\ t\epsilon[0,\ 1].$$



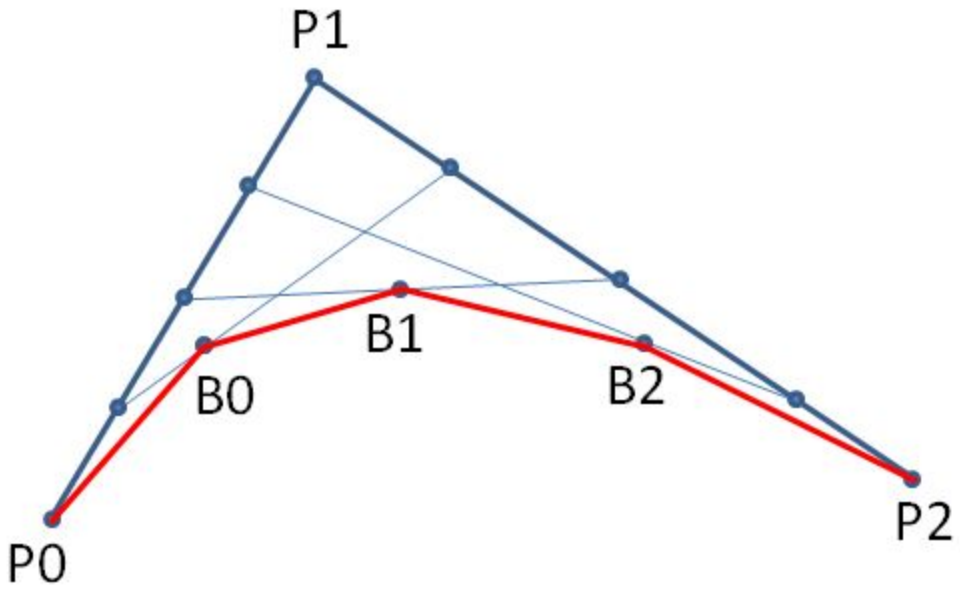
Gambar 1.3 Pembentukan Kurva Bezier Kuadratik.

Proses ini dapat diaplikasikan untuk jumlah titik yang lebih dari 3 titik kontrol. Misalkan, empat titik kontrol akan menghasilkan kurva bezier kubik, lima titik kontrol akan menghasilkan kurva bezier kuartik, dan seterusnya. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya.

$$S_0 = B(t) = (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + t^3P_3, \quad t\epsilon[0,1]$$
$$T_0 = B(t) = (1-t)^4P_0 + 4(1-t)^3tP_1 + 6(1-t)^2t^2P_2 + 4(1-t)t^3P_3 + t^4P_4, \quad t\epsilon[0,1]$$

Persamaan diatas tentunya akan semakin rumit seiring bertambahnya jumlah titik kontrol. Oleh sebab itu, dalam rangka melakukan efisiensi sehingga semakin mangkus dan sangkil dalam program pembuatan kurva Bézier yang sangat berguna ini, kami perlu mengimplementasikan pembuatan kurva Bézier dengan algoritma titik tengah berdasarkan prinsip dari algoritma divide and conquer. untuk algoritma divide and conquer prosesnya dimulai dengan tiga titik, yaitu P0, P1, dan P2, dengan P1 sebagai titik kontrol antara. Langkah pertama adalah menciptakan dua titik baru, Q0 dan Q1, yang masing-masing berada di tengah garis yang menghubungkan P0-P1 dan P1-P2. Kemudian, kita menghubungkan Q0 dan Q1 untuk membentuk garis baru. Titik baru, R0, dibuat di tengah garis Q0-Q1. Dengan garis yang dibuat dari P0 ke R0 dan dari R0 ke P2, kita telah melakukan satu iterasi dan mendapatkan sebuah kurva yang masih belum cukup mulus dengan tiga titik. Untuk meningkatkan kualitas kurva, dilakukan iterasi lanjutan. Titik-titik baru, S0, S1, S2, dan S3, dibuat di tengah-tengah segmen antara P0-Q0, Q0-R0, R0-Q1, dan Q1-P2. Kemudian, garis-garis baru dibuat menghubungkan S0-S1 dan S2-S3. Dua titik baru, T0 dan T1, dibuat di tengah segmen antara S0-S1 dan S2-S3. Terakhir, garis yang menghubungkan P0-T0-R0-T1-P2 dibuat.

Melalui iterasi kedua, kurva yang dihasilkan semakin mendekati kurva Bézier dengan aproksimasi lima titik. Proses ini dapat diulangi dengan iterasi tambahan untuk mendekati kurva yang semakin halus, dengan jumlah titik yang semakin banyak. Ini mengilustrasikan bahwa semakin banyak iterasi yang dilakukan, kurva yang dihasilkan akan semakin mendekati kurva Bézier.



Gambar 1.4 Pembentukan Kurva Bezier dengan iterasi 2

1.4 Penentuan Titik Bezier dengan Divide and Conquer

Algoritma Divide and Conquer bisa diterapkan untuk menentukan titik-titik dalam pembentukan kurva Bézier. Algoritma ini memecah proses pembuatan kurva menjadi submasalah yang lebih kecil, menyelesaikan setiap submasalah secara terpisah, dan kemudian menggabungkan solusi dari submasalah tersebut untuk membentuk kurva. Berikut adalah langkah-langkah untuk mendapatkan kurva Bézier yang diinginkan menggunakan algoritma Divide and Conquer :

1. User Input :
   a. User memasukan 3 input titik kontrol
   b. User memasukan banyak iterasi yang diinginkan
2. Perhitungan Kurva dengan metode Divide and Conquer :
   a. Buat titik kontrol baru bedasarkan titik tengah
   b. Proses ini berjalan sebanyak iterasi yang diinput user
   c. Setelah semua titik iterasi diproses, kurva digambarkan
3. Visualisasi :
   a. Penggabungan semua titik iterasi yang sudah dihitung
   b. Menampilkan titik kontrol yang diinput user
   c. Tampilkan waktu eksekusinya

1.5 Penentuan Titik Bezier dengan Brute Force

Metode brute force bisa digunakan untuk menentukan titik-titik dalam pembentukan Kurva Bézier. Pendekatan ini adalah algoritma yang langsung dan sederhana untuk menemukan titik-titik pada kurva dengan mencoba berbagai kemungkinan titik kontrol yang memenuhi kriteria tertentu. Berikut adalah langkah-langkah untuk mencapai kurva Bézier yang diinginkan menggunakan pendekatan brute force :

1. User Input :
    a. User memasukan 3 input titik kontrol
    b. User memasukan banyak iterasi yang diinginkan
2. Perhitungan Kurva dengan metode Divide and Conquer :
    a. Buat titik kontrol baru bedasarkan titik tengah
    b. Proses ini berjalan sebanyak $2^{\text{iterasi} + 1}$ yang diinput user
    c. Setelah semua titik iterasi diproses, kurva digambarkan
3. Visualisasi :
    a. Penggabungan semua titik iterasi yang sudah dihitung
    b. Menampilkan titik kontrol yang diinput user
    c. Tampilkan waktu eksekusinya

BAB II

IMPLEMENTASI DAN ANALISIS ALGORITMA DALAM BAHASA PYTHON

Dalam pembuatan program pembentukan kurva bezier ini, saya menggunakan bahasa pemograman python. Dalam program ini saya menggunakan modul ‘tkinter’ dari python. tkinter adalah toolkit GUI (Graphical User Interface) standar untuk Python. Ini menyediakan antarmuka untuk toolkit GUI Tk yang ditulis dalam bahasa pemrograman Tcl. Pada program ini algoritma dibagi ke 4 file yaitu src/main.py, src/quadratic_bezier.py, src/brute_force.py, dan Point.py.

2.1 main.py

File main.py adalah file utama dari program yang menggunakan modul tkinter untuk membuat aplikasi GUI. Ini adalah titik masuk utama ke dalam program dan berfungsi untuk membuat jendela aplikasi GUI serta mengatur elemen-elemen GUI dan fungsionalitasnya.

Tabel 2.1 main.py

METHOD	DESCRIPTION
frame1	Merupakan frame kanvas antara kurva dan juga waktu eksekusi untuk metode Divide and Conquer
frame2	Merupakan frame kanvas antara kurva dan juga waktu eksekusi untuk metode Brute Force
ctrl1, ctrl2, ctrl3	Merupakan titik kontrol yang user input pada program
iterations	Merupakan banyaknya iterasi yang user inginkan
draw_quadratic_bezier	Merupakan fungsi yang di dalamnya mengurus pengambilan titik kontrol serta iterasi dari user, mengubah input menjadi tuple koordinat, mengambar sumbu x, y dan pemanggilan algoritma Divide and Conquer
draw_brute_force	Merupakan fungsi yang di dalamnya mengurus pengambilan titik kontrol serta iterasi dari user, mengubah input menjadi

	tuple koordinat, mengambar sumbu x, y dan pemanggilan algoritma Brute Force
draw_button	Membuat tampilan button untuk memulai program perhitungan Divide and Conquer serta Brute Force

2.2 quadratic_bezier.py

File ini berisikan algoritma perhitungan menggunakan metode Divide and Conquer

Tabel 2.2 quadratic_bezier.py

METHOD	DESCRIPTION
create_bezier	Fungsi lengkap dari gabungan semua algoritma Divide and Conquer yang akan mengolah input user dan mengembalikan hasil ke main
start_time_quadratic	Memulai perhitungan waktu awal eksekusi
control_point	Mengubah titik kontrol menjadi sebuah array dengan 3 titik
populate_bezier_points	Fungsi rekursif untuk mengisi titik-titik pada kurva bezier
mid_point	Fungsi untuk menghitung titik tengah antara 2 titik kontrol
scaled_bezier_points	Mengubah koordinat titik-titik pada kurva agar sama dengan koordinat pada kanvas
scaled_control_points	Mengubah koordinat titik kontrol agar sama dengan koordinat pada kanvas
end_time_quadratic	Menghitung waktu akhir eksekusi
execution_time_quadratic	Mengubah waktu dari seconds menjadi milliseconds

2.3 brute_force.py

File ini berisikan algoritma perhitungan menggunakan metode Brute Force

Tabel 2.3 brute_force.py

METHOD	DESCRIPTION
generate_bezier_bruteforce	Fungsi algoritma Brute Force yang akan mengembalikan hasil dan waktu eksekusi dari algoritma brute force
start_time_brute	Memulai perhitungan waktu awal eksekusi
num_points	Jumlah titik yang dihasilkan
point	Array penyimpanan titik-tiitk kurva bezier
scaled_control_points	Mengubah koordinat titik kontrol disesuaikan dengan koordinat pada kanvas
scaled_brute_force_points	Mengubah koordinat titik iterasi disesuaikan dengan koordinat pada kanvas
end_time_brute	Menghitung waktu akhir eksekusi
execution_time_brute_force	Mengubah waktu dari seconds menjadi milliseconds

2.4 Point.py

Kelas Point yang didefinisikan di atas adalah kelas sederhana yang merepresentasikan sebuah titik dalam sistem koordinat dua dimensi.

Tabel 2.4 Point.py

METHOD	DESCRIPTION
self	Parameter pertama dalam setiap metode di dalam kelas Python yang merujuk pada objek itu sendiri.
x, y	Parameter-parameter ini merepresentasikan koordinat x dan y dari titik yang ingin dibuat.

BAB III

SOURCE CODE PROGRAM

3.1 Repository Program

Repository program dapat diakses melalui pranala *GitHub* berikut :
https://github.com/JasonFernandoo/Tucil2_13522156

3.2 Source Code Program

3.2.1 main.py

```python
import tkinter as tk
import time
import quadratic_bezier
import brute_force

root = tk.Tk()
root.title("Bezier Curves")

frame1 = tk.Frame(root)
frame1.pack(side=tk.LEFT)

frame2 = tk.Frame(root)
frame2.pack(side=tk.RIGHT)

canvas1 = tk.Canvas(frame1, width=700, height=400)
canvas1.pack()

canvas2 = tk.Canvas(frame2, width=700, height=400)
canvas2.pack()

ctrl1_label = tk.Label(root, text="Control Point 1 (x, y):")
ctrl1_label.pack()
ctrl1_entry = tk.Entry(root)
ctrl1_entry.pack()

ctrl2_label = tk.Label(root, text="Control Point 2 (x, y):")
ctrl2_label.pack()
ctrl2_entry = tk.Entry(root)
ctrl2_entry.pack()

ctrl3_label = tk.Label(root, text="Control Point 3 (x, y):")
ctrl3_label.pack()
ctrl3_entry = tk.Entry(root)
ctrl3_entry.pack()

iterations_label = tk.Label(root, text="Iterations:")
iterations_label.pack()
iterations_entry = tk.Entry(root)
iterations_entry.pack()
```

```python
def draw_quadratic_bezier(canvas1):
    canvas1.delete("all")
    ctrl1_str = ctrl1_entry.get()
    ctrl2_str = ctrl2_entry.get()
    ctrl3_str = ctrl3_entry.get()
    iterations_str = iterations_entry.get()

    try:
        ctrl1 = tuple(map(float, ctrl1_str.split(',')))
        ctrl2 = tuple(map(float, ctrl2_str.split(',')))
        ctrl3 = tuple(map(float, ctrl3_str.split(',')))
        iterations = int(iterations_str)
    except ValueError:
        tk.messagebox.showerror("Error", "Invalid input. Please enter numbers separated by commas.")
        return

    for widget in frame1.winfo_children():
        if isinstance(widget, tk.Label) and "Quadratic Bezier execution time" in widget.cget("text"):
            widget.destroy()

    canvas_width = canvas1.winfo_width()
    canvas_height = canvas1.winfo_height()

    start_time_quadratic = time.time()
    canvas1.create_line(30, 300, 650, 300, fill="black")
    for x in range(0, 13):
        canvas1.create_line((x * 50) + 30, 50, (x * 50) + 30, 300, fill="grey")
        canvas1.create_text((x * 50) + 30, 315, text=str(x), anchor=tk.N)
    canvas1.create_line(30, 300, 30, 40, fill="black")
    for y in range(20, 240, 20):
        canvas1.create_line(30, y + 60, 650, y + 60, fill="grey")
        canvas1.create_text(25, y + 60, text=str(12 - y // 20), anchor=tk.E)
    execution_time_quadratic = quadratic_bezier.create_bezier(ctrl1, ctrl2, ctrl3, iterations, canvas1, canvas_width, canvas_height)
    end_time_quadratic = time.time()
    execution_time_quadratic = (end_time_quadratic - start_time_quadratic) * 1000
    label_exec_time_quadratic = tk.Label(frame1, text=f"Quadratic Bezier execution time: {execution_time_quadratic:.6f} ms", fg="black")
    label_exec_time_quadratic.pack(side=tk.BOTTOM)
```

```python
def draw_brute_force(canvas2):
    canvas2.delete("all")
    ctrl1_str = ctrl1_entry.get()
    ctrl2_str = ctrl2_entry.get()
    ctrl3_str = ctrl3_entry.get()
    iterations_str = iterations_entry.get()

    try:
        ctrl1 = tuple(map(float, ctrl1_str.split(',')))
        ctrl2 = tuple(map(float, ctrl2_str.split(',')))
        ctrl3 = tuple(map(float, ctrl3_str.split(',')))
        iterations = int(iterations_str)
    except ValueError:
        tk.messagebox.showerror("Error", "Invalid input. Please enter numbers separated by commas.")
        return

    for widget in frame2.winfo_children():
        if isinstance(widget, tk.Label) and "Brute Force execution time" in widget.cget("text"):
            widget.destroy()

    canvas_width = canvas2.winfo_width()
    canvas_height = canvas2.winfo_height()

    start_time_brute = time.time()
    canvas2.create_line(30, 300, 650, 300, fill="black")
    for x in range(1, 13):
        canvas2.create_line((x * 50) + 30, 50, (x * 50) + 30, 300, fill="grey")
        canvas2.create_text((x * 50) + 30, 315, text=str(x), anchor=tk.N)
    canvas2.create_line(30, 300, 30, 40, fill="black")
    for y in range(20, 240, 20):
        canvas2.create_line(30, y + 60, 650, y + 60, fill="grey")
        canvas2.create_text(25, y + 60, text=str(12 - y // 20), anchor=tk.E)
    start_point = brute_force.Point(*ctrl1)
    control_point = brute_force.Point(*ctrl2)
    end_point = brute_force.Point(*ctrl3)
    execution_time_brute_force = brute_force.generate_bezier_bruteforce(start_point, control_point, end_point, iterations, canvas2, canvas_width, canvas_height)
    end_time_brute = time.time()
    execution_time_brute_force = (end_time_brute - start_time_brute) * 1000
    label_exec_time_brute_force = tk.Label(frame2, text=f"Brute Force execution time: {execution_time_brute_force:.6f} ms", fg="black")
    label_exec_time_brute_force.pack(side=tk.BOTTOM)
```

```python
draw_button = tk.Button(root, text="Draw Curve", command=lambda: (draw_quadratic_bezier(canvas1), draw_brute_force(canvas2)))
draw_button.pack()

root.mainloop()
```

3.2.2 quadratic_bezier.py

```python
import time

def create_bezier(ctrl1, ctrl2, ctrl3, iterations, canvas, canvas_width, canvas_height):
    start_time_quadratic = time.time()
    bezier_points = []
    control_points = [ctrl1, ctrl2, ctrl3]

    def populate_bezier_points(ctrl1, ctrl2, ctrl3, current_iteration, iterations):
        nonlocal bezier_points
        if current_iteration < iterations:
            mid_point1 = mid_point(ctrl1, ctrl2)
            mid_point2 = mid_point(ctrl2, ctrl3)
            mid_point3 = mid_point(mid_point1, mid_point2)
            current_iteration += 1
            populate_bezier_points(ctrl1, mid_point1, mid_point3, current_iteration, iterations)
            bezier_points.append(mid_point3)
            populate_bezier_points(mid_point3, mid_point2, ctrl3, current_iteration, iterations)
        else:
            bezier_points.extend([ctrl1, ctrl3])

    def mid_point(control_point1, control_point2):
        return ((control_point1[0] + control_point2[0]) / 2, (control_point1[1] + control_point2[1]) / 2)

    populate_bezier_points(ctrl1, ctrl2, ctrl3, 0, iterations)

    scaled_bezier_points = [((point[0] * canvas_width/14.07 + 30), (canvas_height - point[1] * canvas_height/20.25) - 105) for point in bezier_points if point[0] >= 0 and
    scaled_control_points = [((point[0] * canvas_width/14.07 + 30), (canvas_height - point[1] * canvas_height/20.25) - 105) for point in control_points if point[0] >= 0 a

    canvas.create_line(scaled_bezier_points, smooth=True, fill="blue")

    for point in scaled_bezier_points:
        canvas.create_oval(point[0]-2, point[1]-2, point[0]+2, point[1]+2, fill="red")

    for point in scaled_control_points:
        canvas.create_oval(point[0]-2, point[1]-2, point[0]+2, point[1]+2, fill="red")

    for i in range(len(scaled_control_points)-1):
        canvas.create_line(scaled_control_points[i], scaled_control_points[i+1], fill="black")
```

(parameter) current_iteration: Any

```python

    end_time_quadratic = time.time()
    execution_time_quadratic = (end_time_quadratic - start_time_quadratic) * 1000

    return execution_time_quadratic
```

3.2.3 brute_force.py

```python
from Point import Point
import time

def generate_bezier_bruteforce(start_point, control_point, end_point, iterations, canvas, canvas_width, canvas_height):
    start_time_brute = time.time()
    num_points = 2 ** (iterations + 1)
    points = [start_point]
    for i in range(1, num_points):
        t = i / num_points
        x = (1 - t)**2 * start_point.x + 2 * (1 - t) * t * control_point.x + t**2 * end_point.x
        y = (1 - t)**2 * start_point.y + 2 * (1 - t) * t * control_point.y + t**2 * end_point.y
        points.append(Point(x, y))
    points.append(end_point)

    canvas.create_line(((start_point.x * canvas_width/14.07) + 30, (canvas_height - start_point.y * canvas_height/20.25) - 105,
                        (control_point.x * canvas_width/14.07) + 30, (canvas_height - control_point.y * canvas_height/20.25) - 105,
                        (end_point.x * canvas_width/14.07) + 30, (canvas_height - end_point.y * canvas_height/20.25) - 105), fill="black")

    scaled_control_points = [((point.x * canvas_width/14.07) + 30, (canvas_height - point.y * canvas_height/20.25) - 105) for point in [start_point, control_point, end_p
    for point in scaled_control_points:
        canvas.create_oval(point[0]-2, point[1]-2, point[0]+2, point[1]+2, fill="blue")

    scaled_brute_force_points = [((point.x * canvas_width/14.07) + 30, (canvas_height - point.y * canvas_height/20.25) - 105) for point in points if point.x >= 0 and poi
    canvas.create_line(scaled_brute_force_points, smooth=True, fill="red")

    for point in scaled_brute_force_points:
        canvas.create_oval(point[0]-2, point[1]-2, point[0]+2, point[1]+2, fill="red")

    end_time_brute = time.time()
    execution_time_brute_force = (end_time_brute - start_time_brute) * 1000

    return execution_time_brute_force
```

3.2.4 Point.py

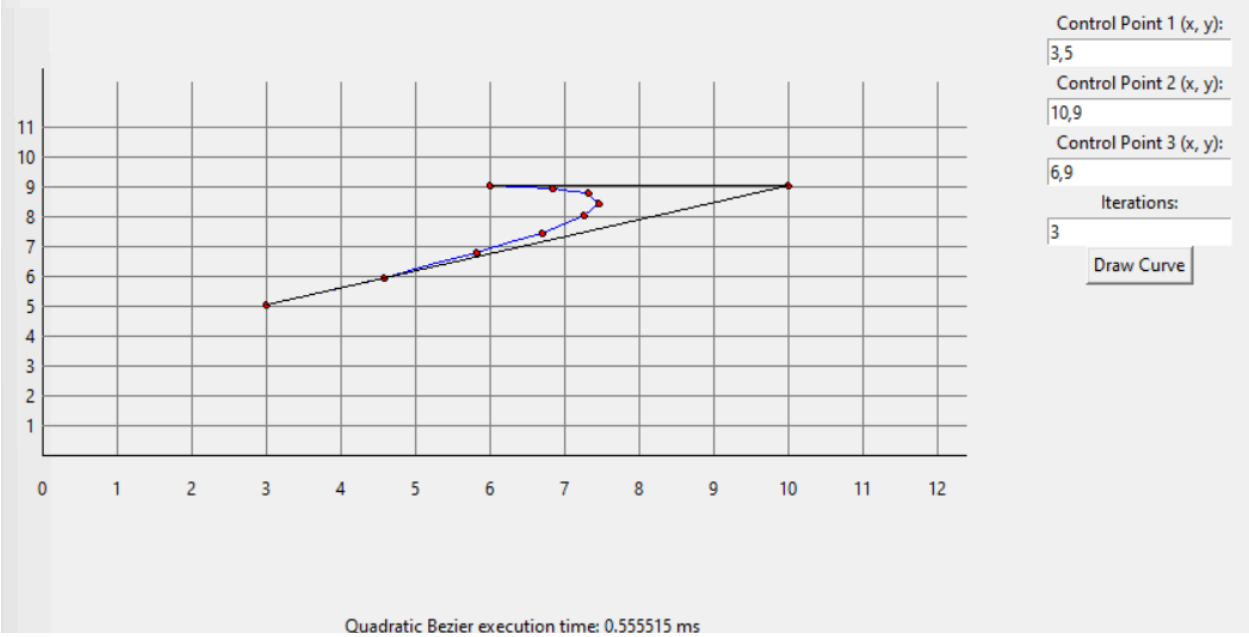
```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

BAB IV
UJI COBA PROGRAM

Pada uji coba program ini, saya akan melakukan 3 kali pecobaan pada spesifikasi wajib dengan 3 titik kontrol dan akan dilakukan 6 set percobaan. Saya akan lakukan dengan 3 dan 5 iterasi untuk masing-masing algoritma dalam setiap set. Berikut 6 titik set percobaanya :
Set 1: (3, 5), (10, 9), (6, 9)
Set 2: (8, 2), (10, 6), (2, 4)
Set 3: (7, 1), (2, 9), (5, 3)
Set 4: (1, 1), (8, 11), (10, 8)
Set 5: (3, 6), (4, 11), (11, 7)
Set 6: (0, 11), (3, 0), (12, 5)

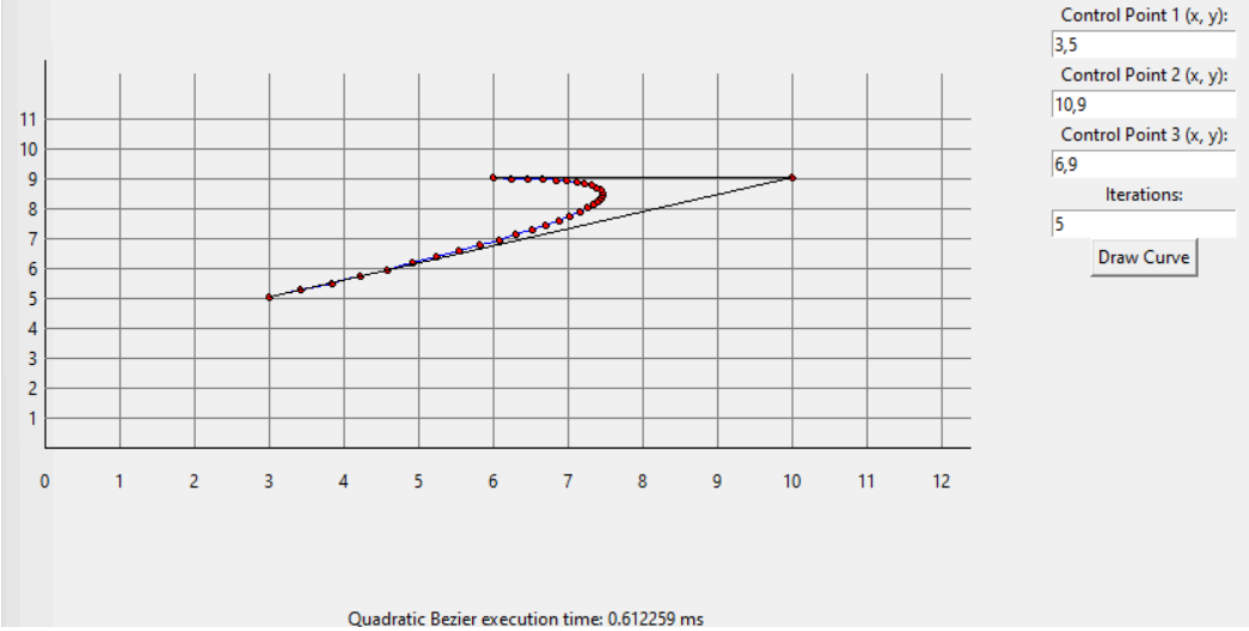
4.1 Set Percobaan ke-1

a. Algoritma Divide and Conquer
   i. 3 iterasi



Gambar 4.1 Percobaan set pertama divide and conquer iterasi 3

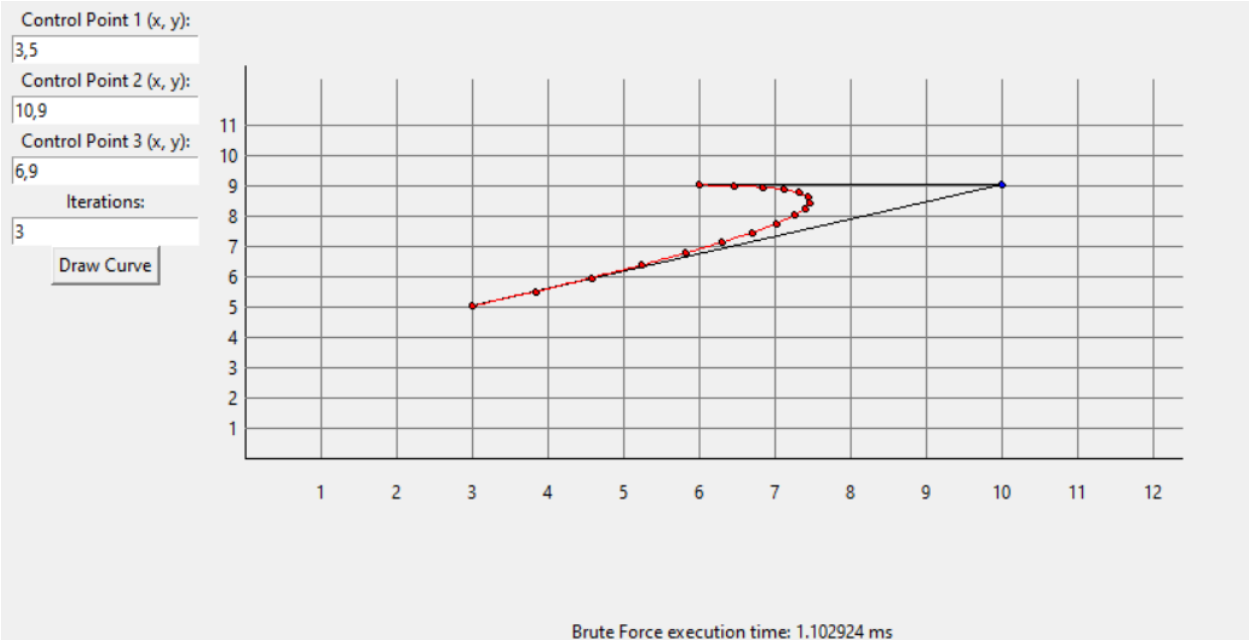
ii. 5 iterasi



Gambar 4.2 Percobaan set pertama divide and conquer iterasi 5

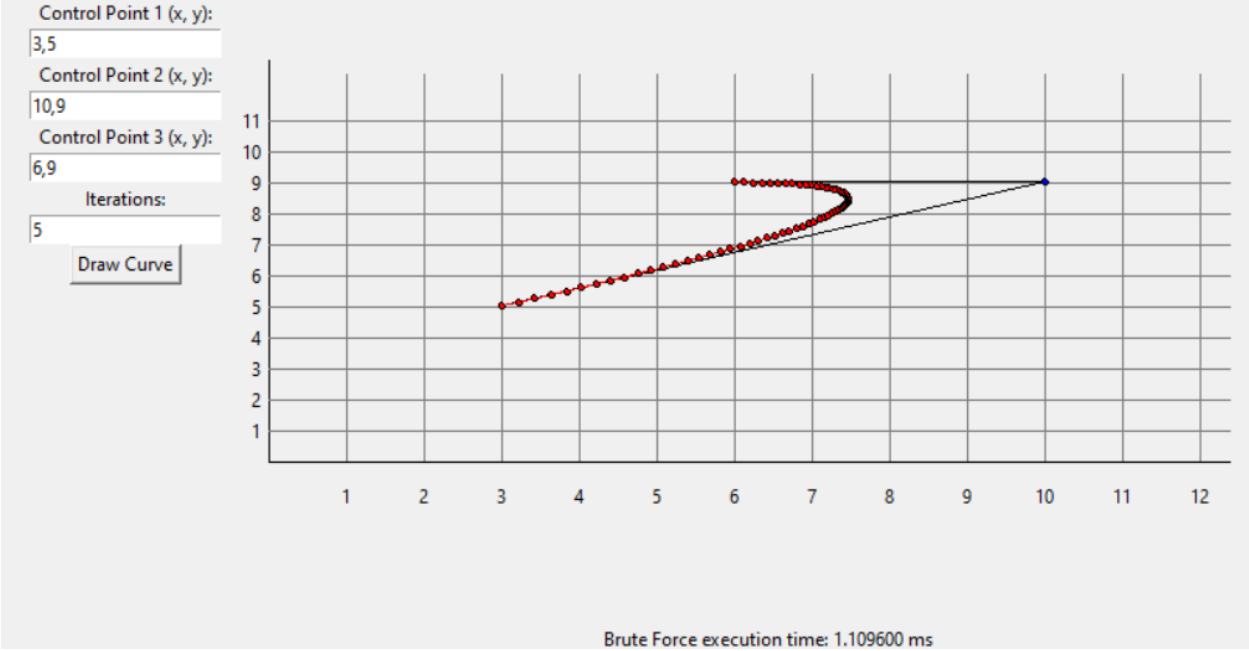
b. Algoritma Brute Force

i. 3 iterasi



Gambar 4.3 Percobaan set pertama brute force iterasi 3

ii. 5 iterasi

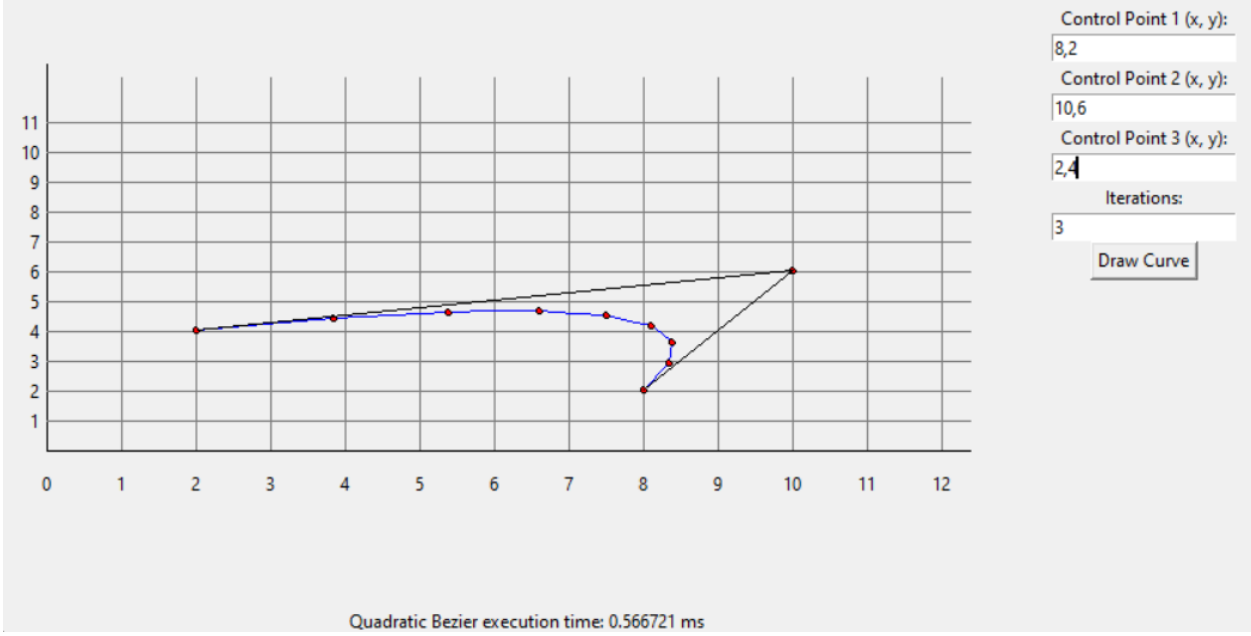


Gambar 4.4 Percobaan set pertama brute force iterasi 5

4.2 Set Percobaan ke-2

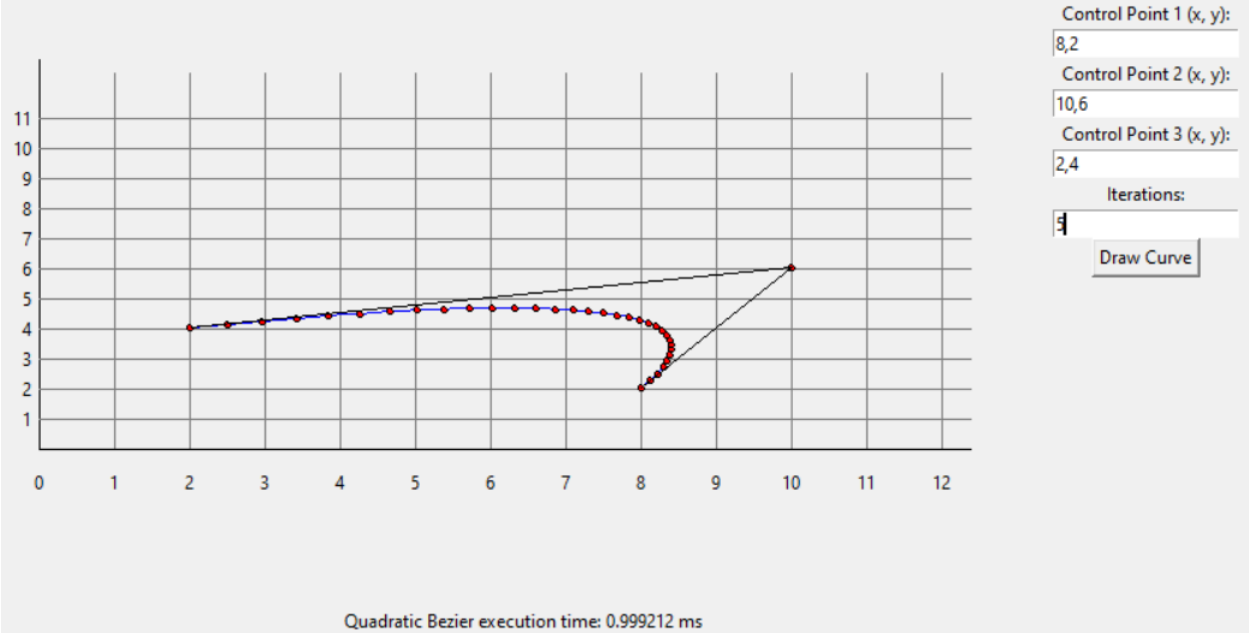
a. Algoritma Divide and Conquer

i. 3 iterasi



Gambar 4.5 Percobaan set kedua divide and conquer iterasi 3

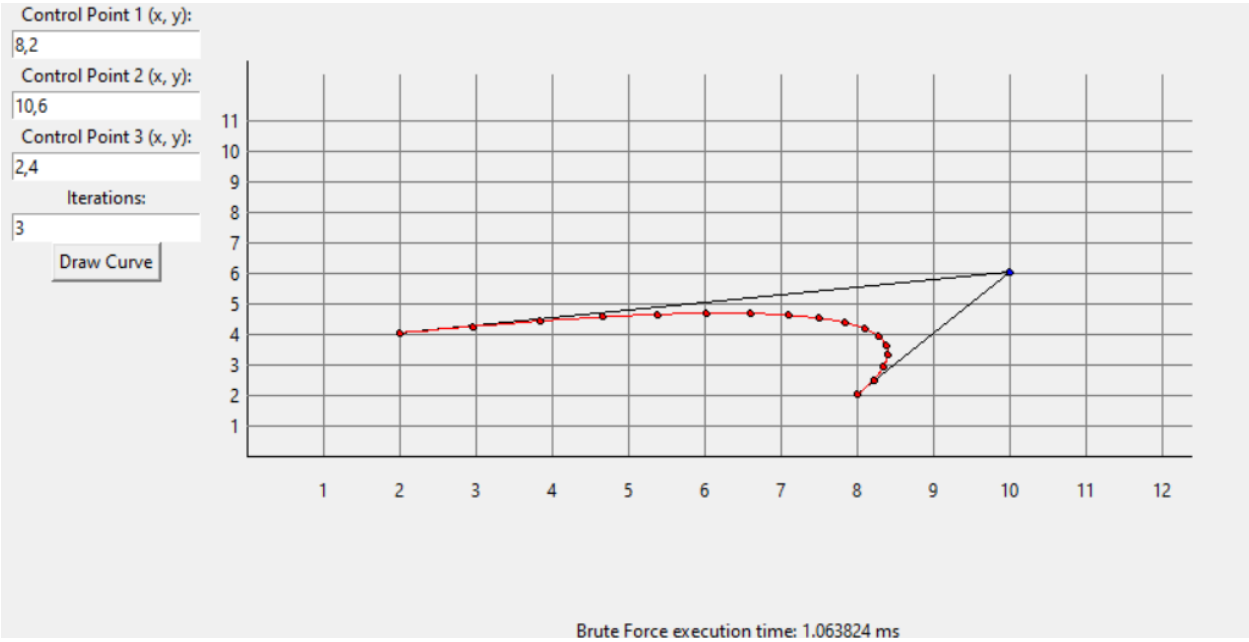
ii. 5 iterasi



Gambar 4.6 Percobaan set kedua divide and conquer iterasi 5

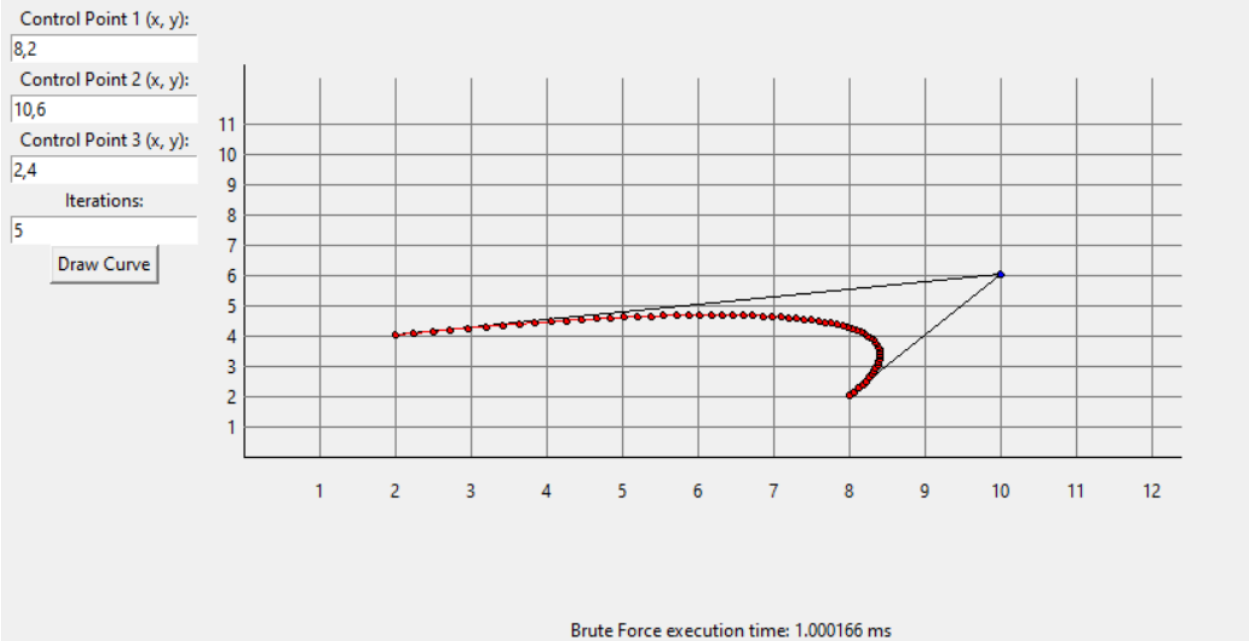
b. Algoritma Brute Force

i. 3 iterasi



Gambar 4.7 Percobaan set kedua brute force iterasi 3

ii. 5 iterasi

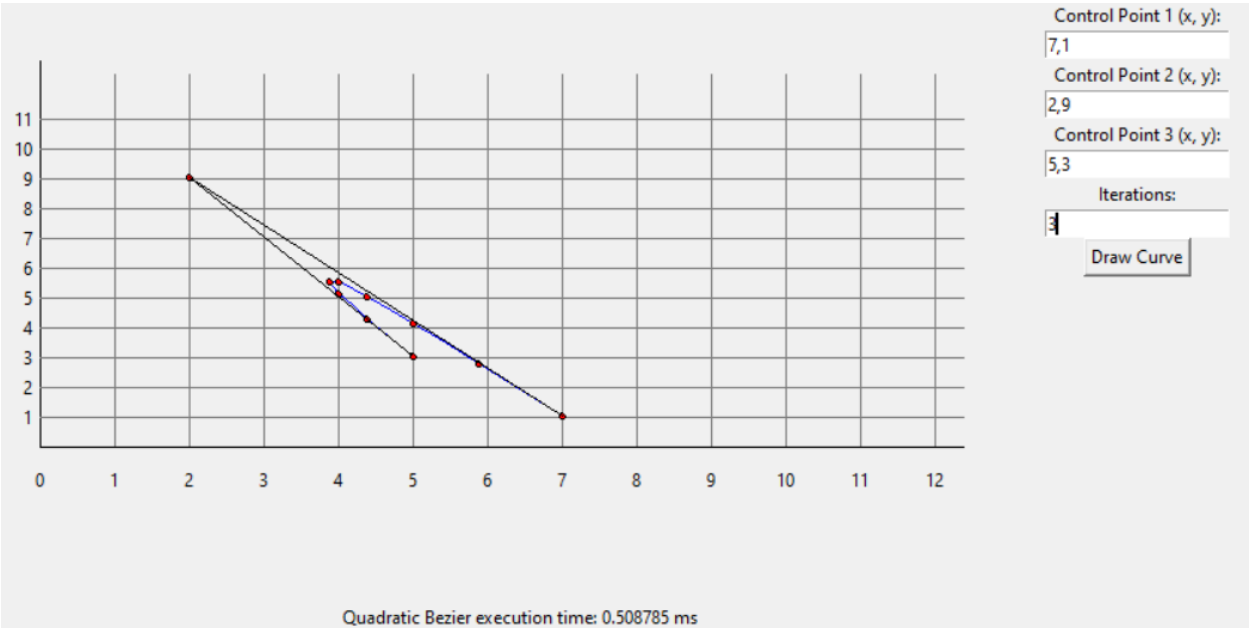


Gambar 4.8 Percobaan set kedua brute force iterasi 5

4.3 Set Percobaan ke-3

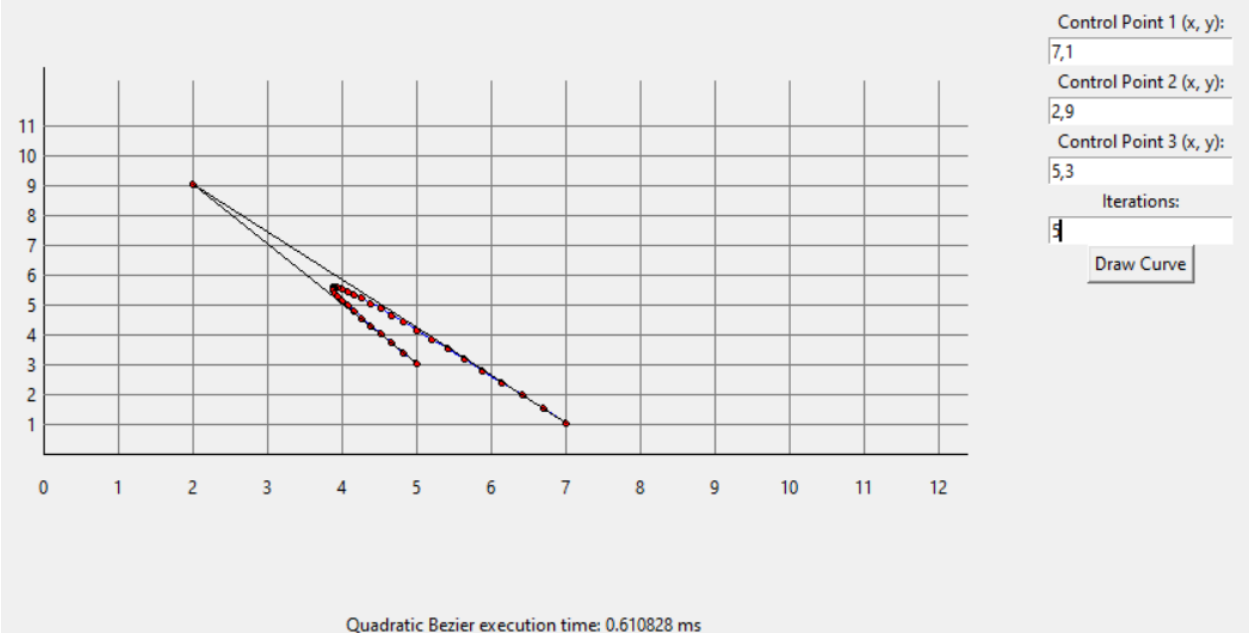
a. Algoritma Divide and Conquer

i. 3 iterasi



Gambar 4.9 Percobaan set ketiga divide and conquer iterasi 3

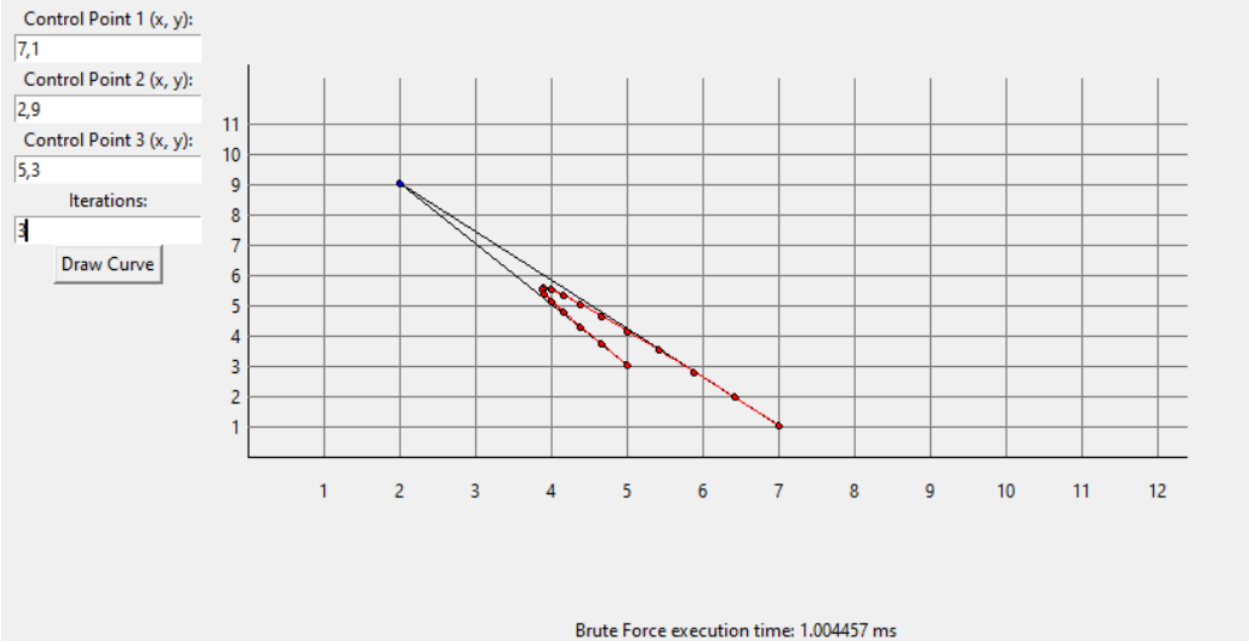
ii. 5 iterasi



Gambar 4.10 Percobaan set ketiga divide and conquer iterasi 5

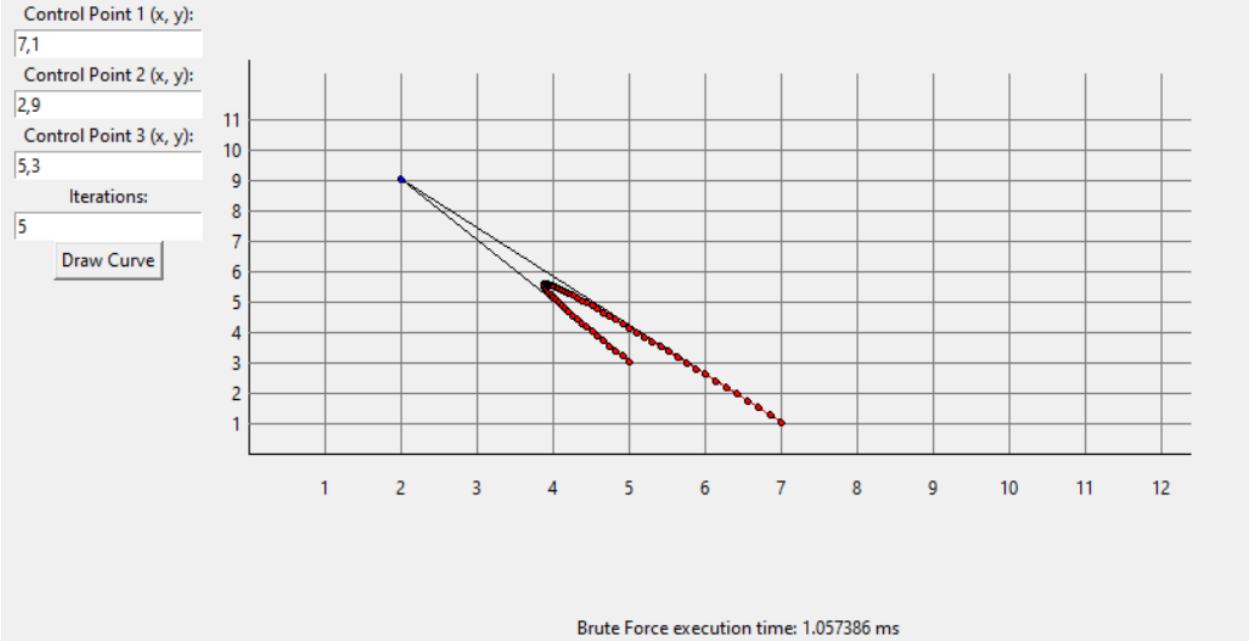
b. Algoritma Brute Force

i. 3 iterasi



Gambar 4.11 Percobaan set ketiga brute force iterasi 3

ii. 5 iterasi

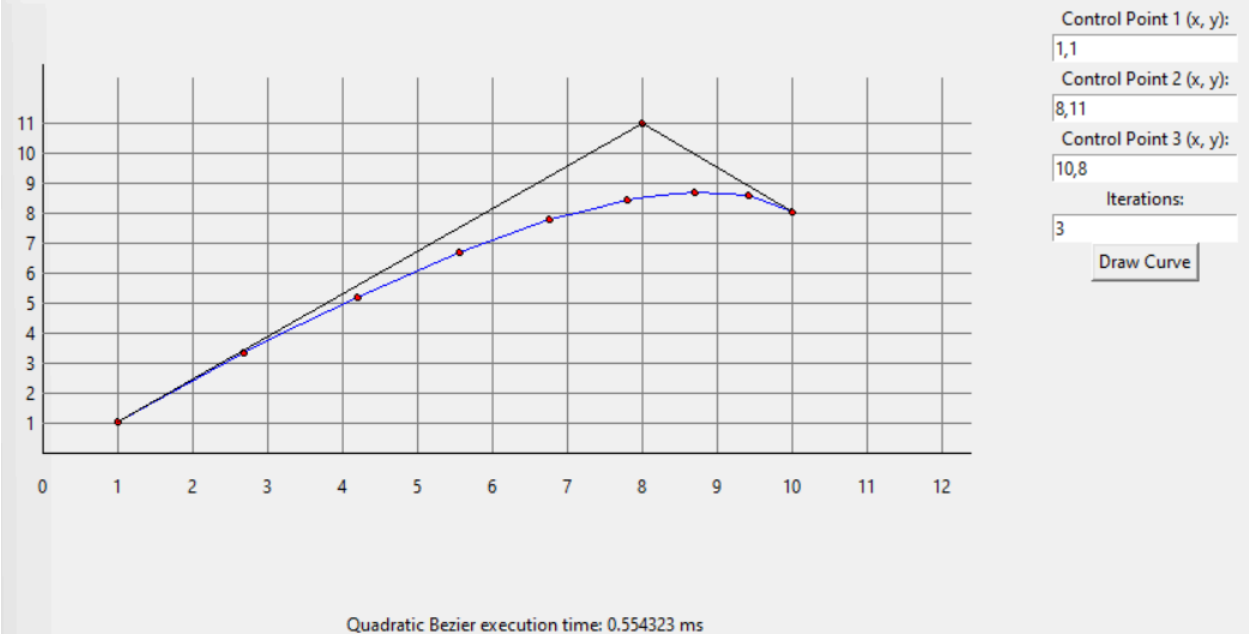


Gambar 4.12 Percobaan set ketiga brute force iterasi 5

4.4 Set Percobaan ke-4

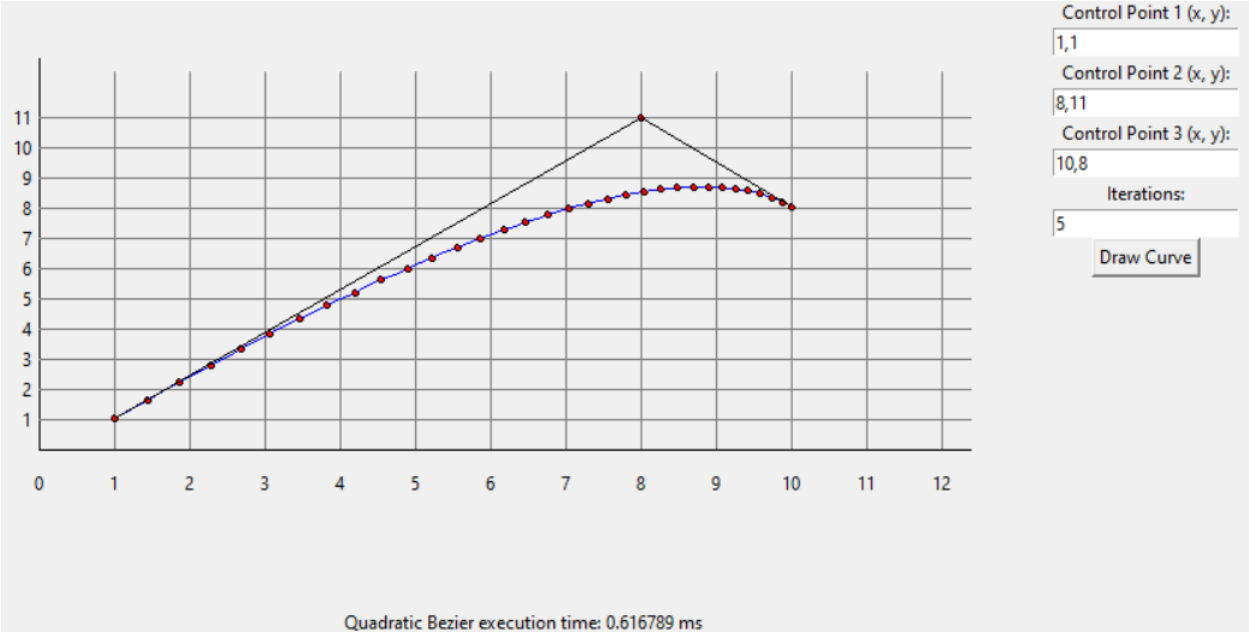
a. Algoritma Divide and Conquer

i. 3 iterasi



Gambar 4.13 Percobaan set keempat divide and conquer iterasi 3

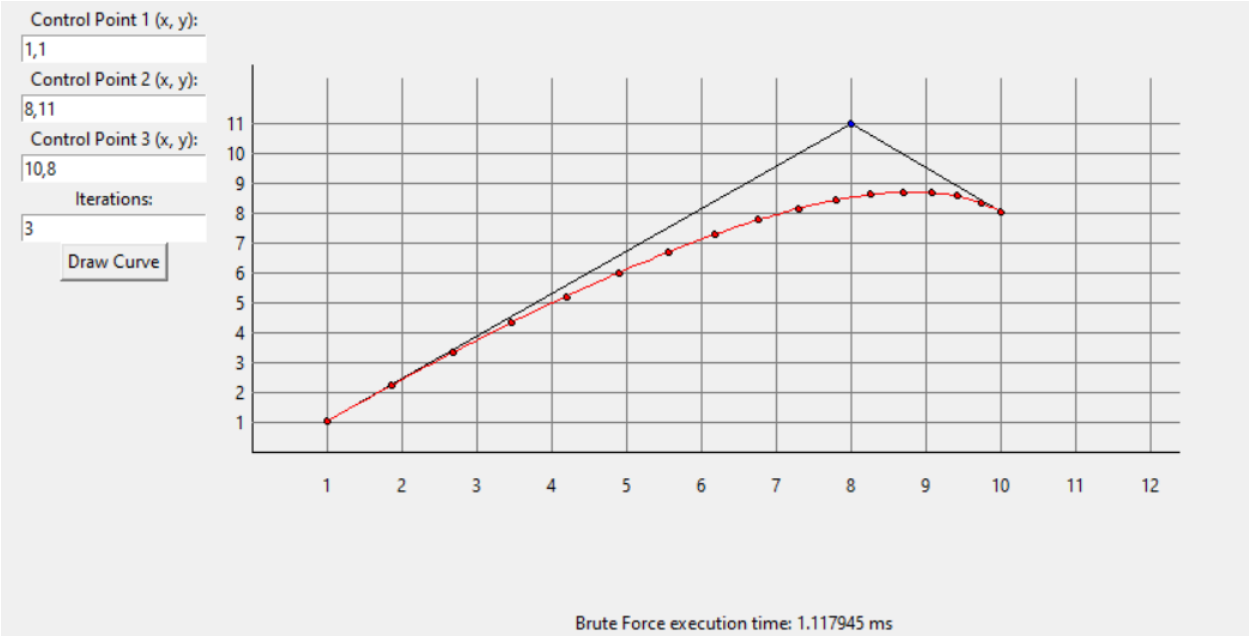
ii. 5 iterasi



Gambar 4.14 Percobaan set keempat divide and conquer iterasi 5

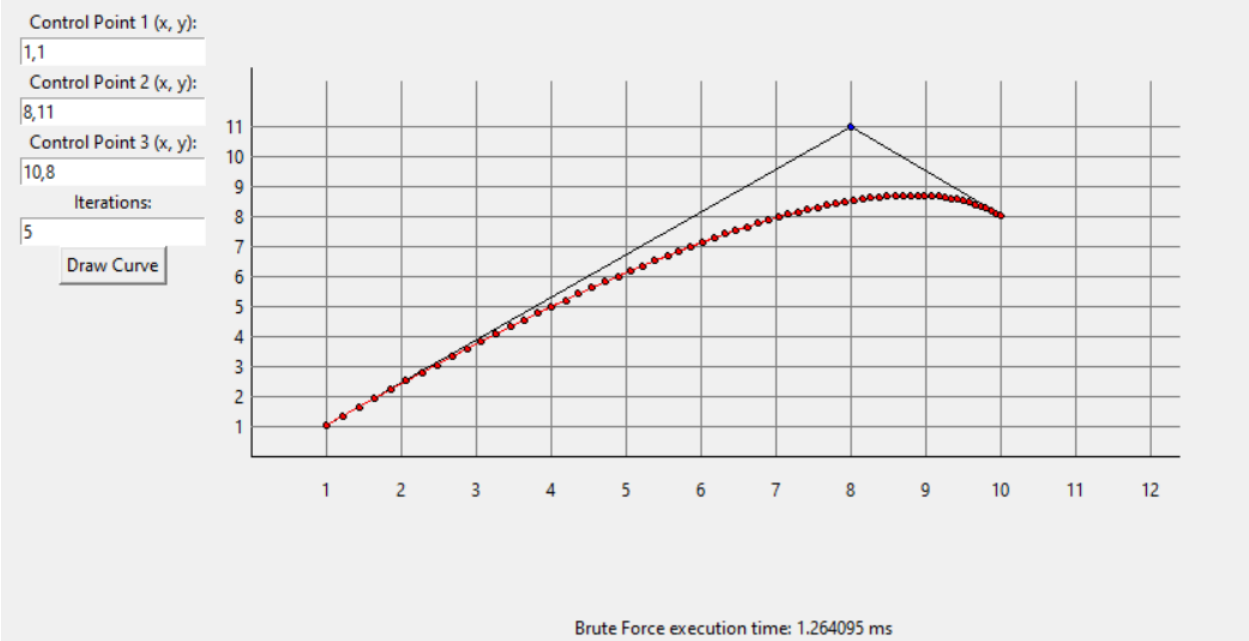
b. Algoritma Brute Force

i. 3 iterasi



Gambar 4.15 Percobaan set keempat brute force iterasi 3

ii. 5 iterasi

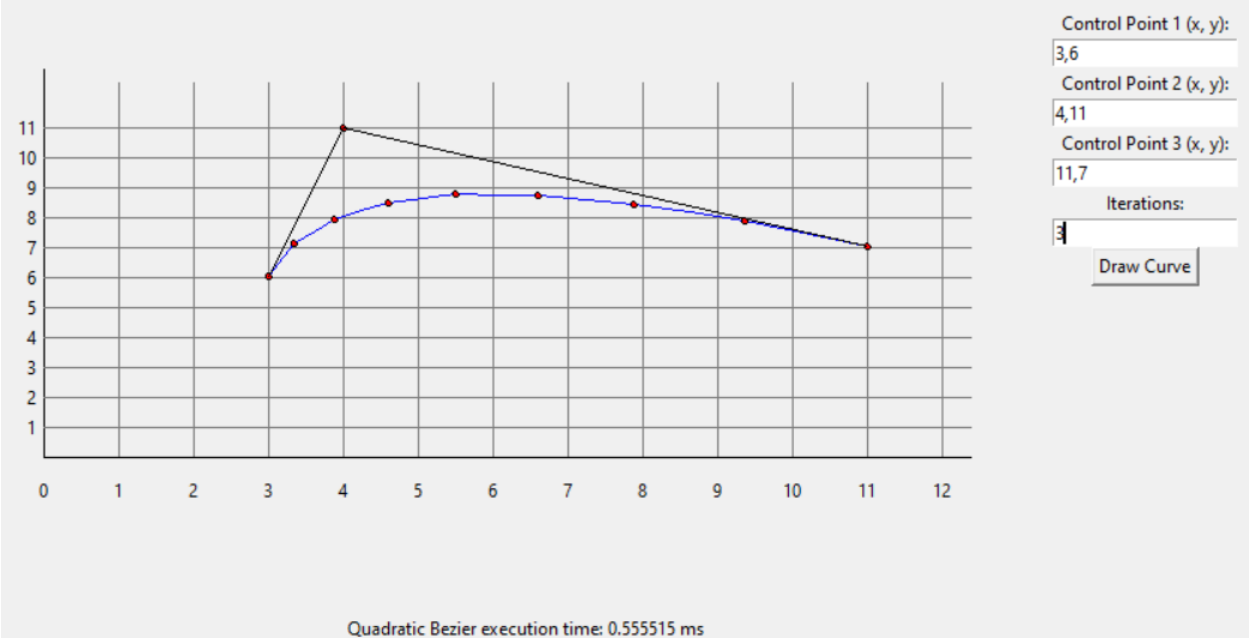


Gambar 4.16 Percobaan set keempat brute force iterasi 5

4.5 Set Percobaan ke-5

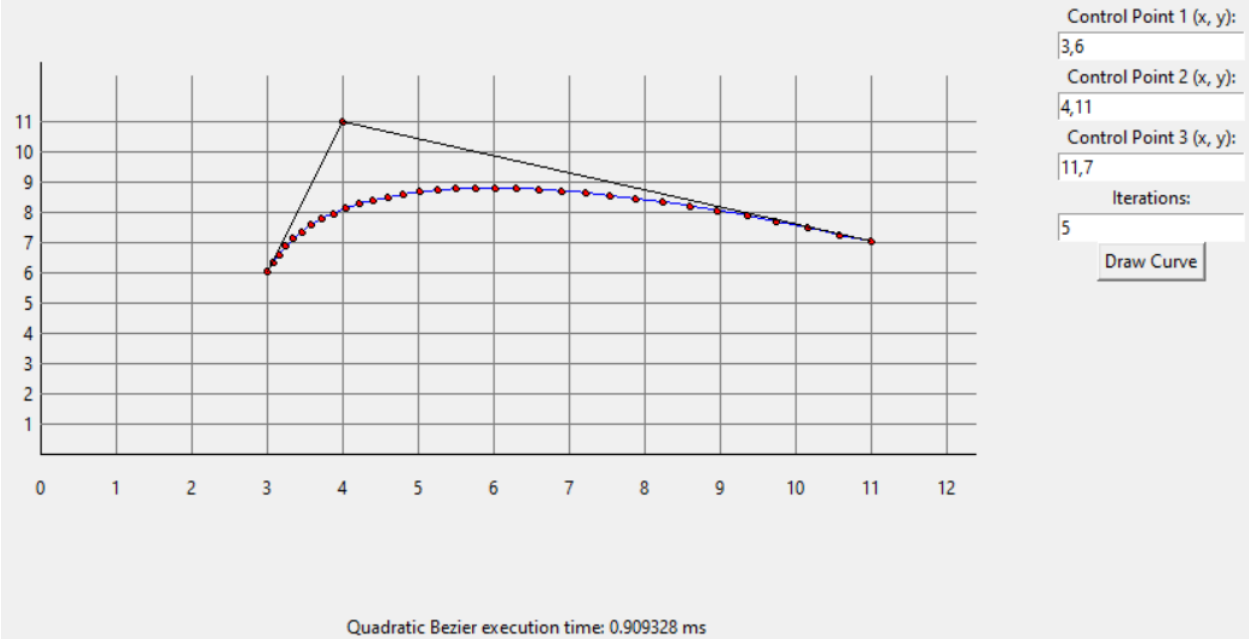
a. Algoritma Divide and Conquer

i. 3 iterasi



Gambar 4.17 Percobaan set kelima divide and conquer iterasi 3

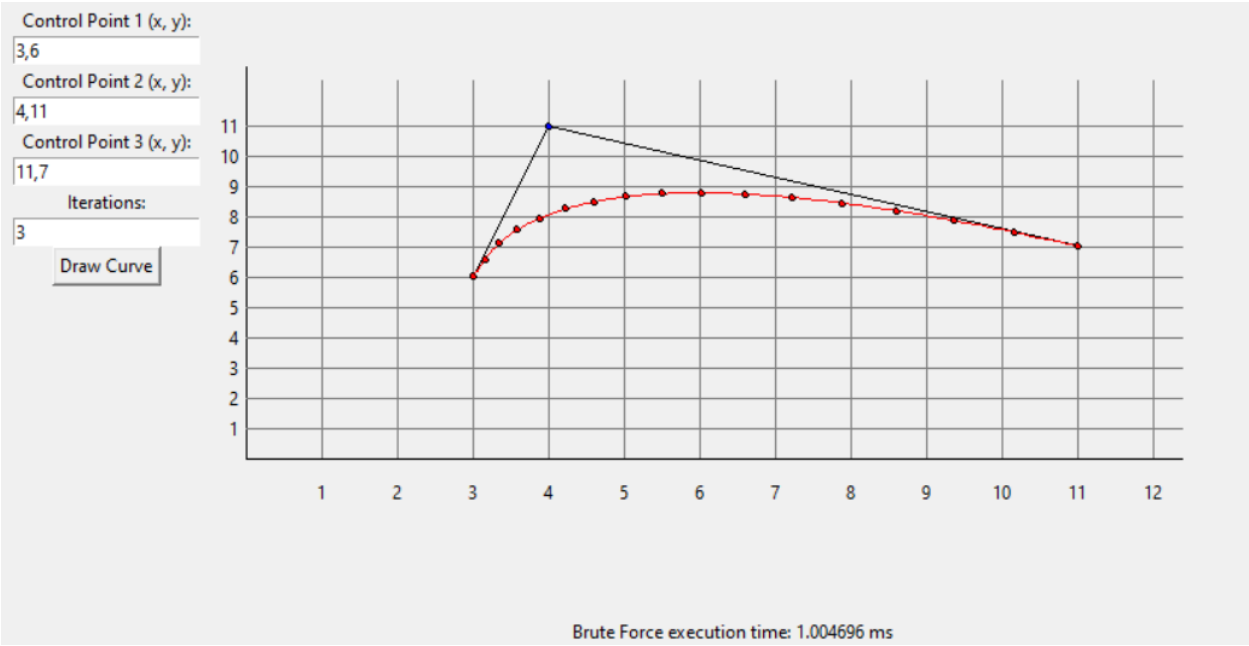
ii. 5 iterasi



Gambar 4.18 Percobaan set kelima divide and conquer iterasi 5

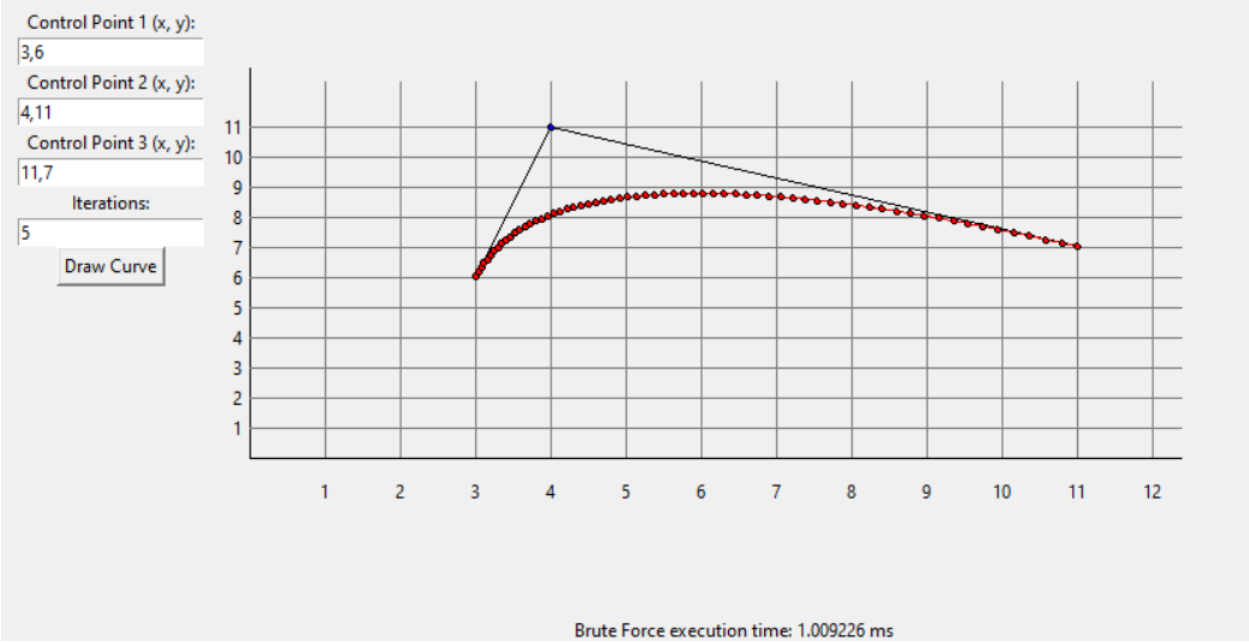
b. Algortima Brute Force

i. 3 iterasi



Gambar 4.19 Percobaan set kelima brute force iterasi 3

ii. 5 iterasi

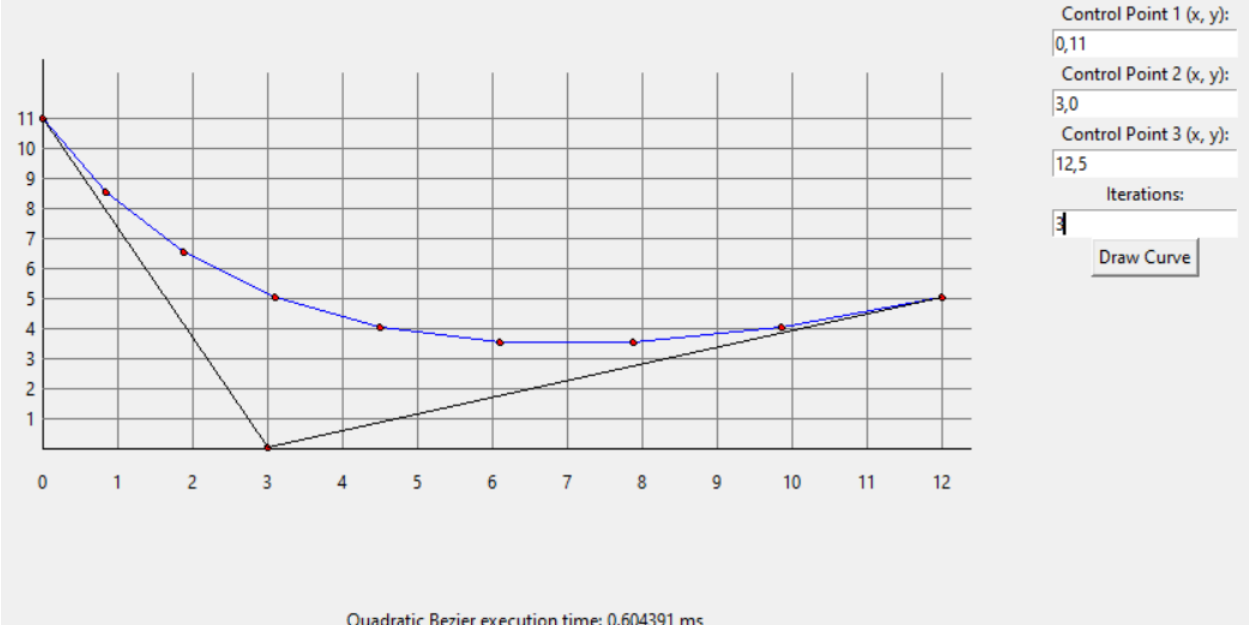


Gambar 4.20 Percobaan set kelima brute force iterasi 5

4.6 Set Percobaan ke-6

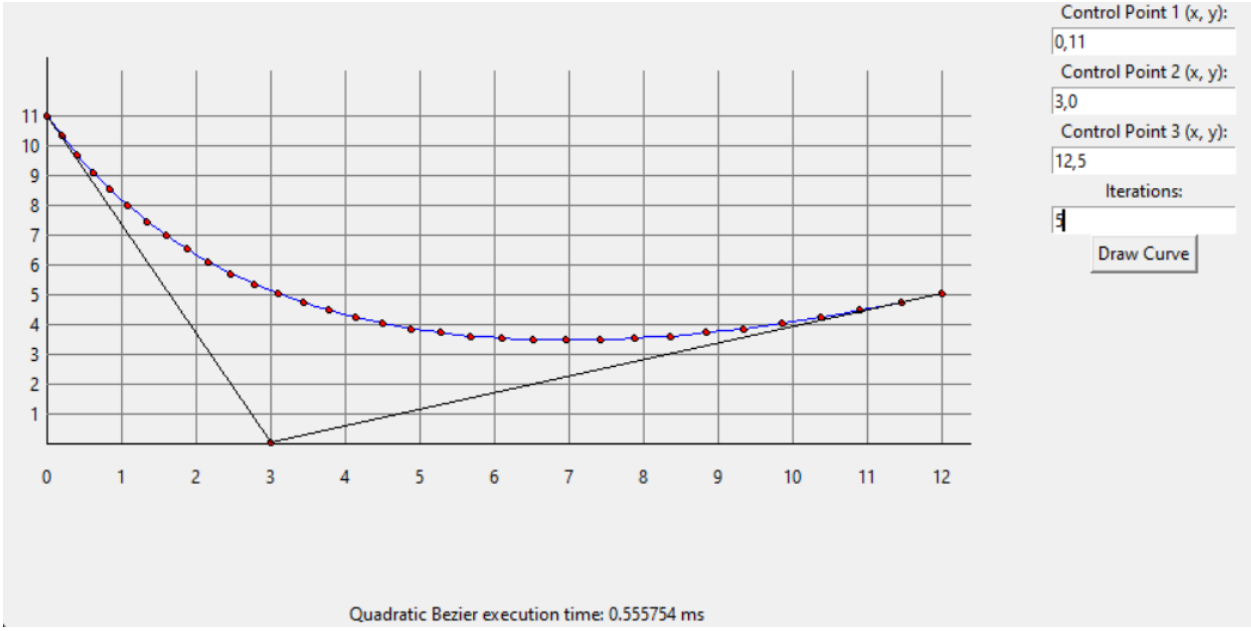
a. Algoritma Divide and Conquer

i. 3 iterasi



Gambar 4.21 Percobaan set keenam divide and conquer iterasi 3

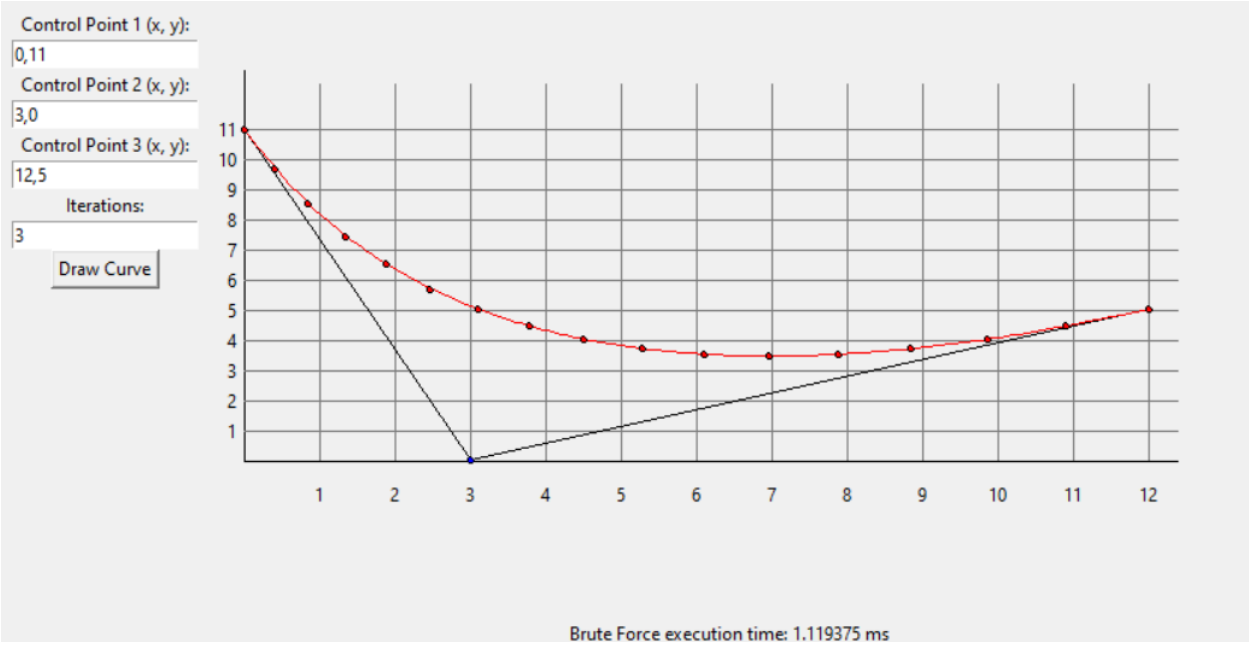
ii. 5 iterasi



Gambar 4.22 Percobaan set keenam divide and conquer iterasi 5

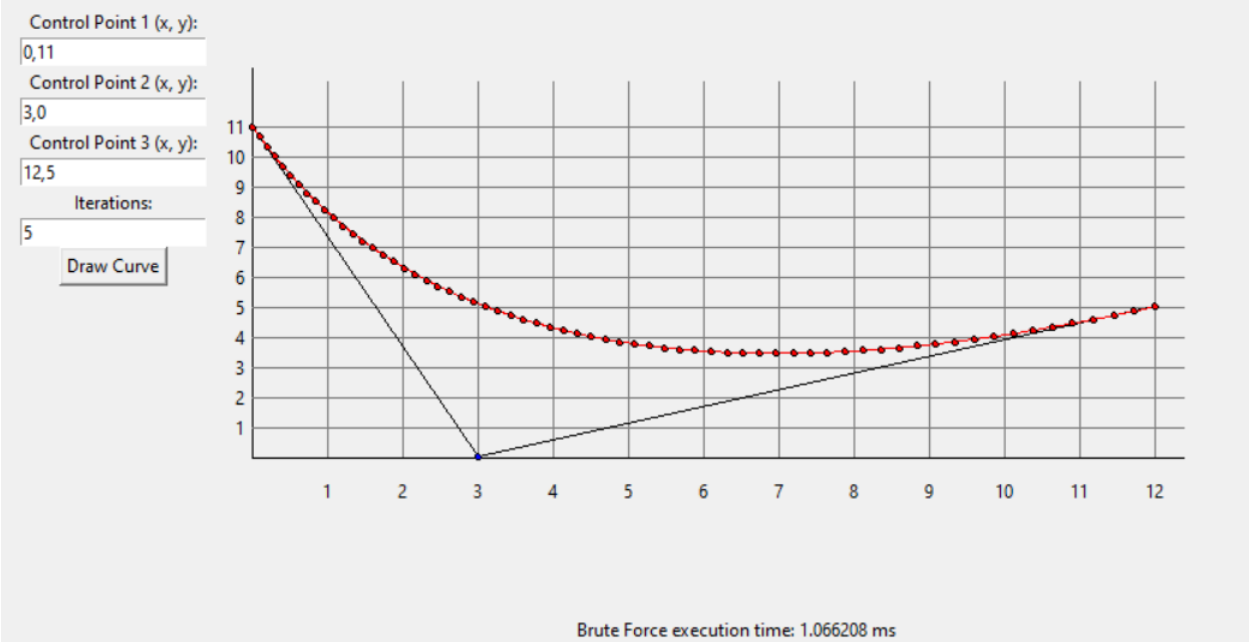
b. Algoritma Brute Force

i. 3 iterasi



Gambar 4.23 Percobaan set keenam brute force iterasi 3

ii. 5 iterasi



Gambar 4.24 Percobaan set keenam brute force iterasi 5

BAB V

KESIMPULAN

Dengan demikian, dapat disimpulkan bahwa algoritma Divide and Conquer merupakan pendekatan yang lebih efisien dan efektif dalam pembentukan kurva Bezier. Hal ini disebabkan oleh pendekatan Divide and Conquer yang membagi masalah menjadi submasalah yang lebih kecil, sehingga memungkinkan proses pembentukan kurva memiliki waktu eksekusi yang lebih singkat. Dengan memecah masalah secara terstruktur, algoritma ini mampu menangani pembentukan kurva dengan jumlah titik kontrol yang relatif sedikit secara lebih efisien. Sebaliknya, Brute Force membutuhkan lebih banyak iterasi karena mencoba semua kemungkinan titik kontrol, yang akhirnya memakan waktu lebih lama untuk menyelesaikan proses pembentukan kurva. Meskipun Brute Force tetap relevan untuk keperluan pembanding atau dalam kasus-kasus tertentu, algoritma Divide and Conquer tetap menjadi pilihan yang lebih optimal dalam sebagian besar situasi pembentukan kurva Bezier.

LAMPIRAN

Spesifikasi Tugas Kecil 2 IF2211 Strategi Algoritma tertera pada link di bawah ini
https://drive.google.com/file/d/1y16HU7ZPqmq7YwP5Y9HJVof8WGv0cNO2/view

Checklist Tugas Kecil 2 IF2211 Strategi Algoritma

POIN	YA	TIDAK
1. Program berhasil dijalankan.	V	
2. Program dapat melakukan visualisasi kurva Bézier	V	
3. Solusi yang diberikan program optimal.	V	
4. **[Bonus]** Program dapat membuat kurva untuk n titik kontrol.		V
5. **[Bonus]** Program dapat melakukan visualisasi proses pembuatan kurva.		V