Tugas Kecil 2 IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah Berbasis *Divide and Conquer*

Disusun Oleh :
Miftahul Jannah (10023500)
Tazkirah Amaliah (10023608)

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

2024

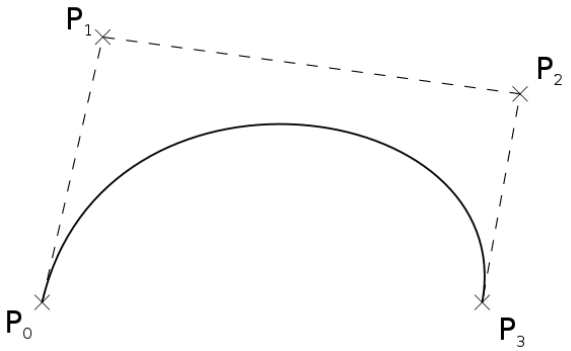
Daftar Isi

Daftar Isi **2**
Bab 1: Deskripsi Masalah **3**
Bab 2: Landasan Teori **6**
2.1 Algoritma Divide and Conquer 6
2.2 Algoritma Brute Force 7
Bab 3: Implementasi Program **8**
3.1 Analisis dan implementasi dalam algoritma brute force sebagai algoritma 8
3.2 Analisis dan implementasi dalam algoritma divide and conquer sebagai algoritma 8
3.3 Source Code Program 9
Bab 4: Eksperimen dan Hasil **12**
4.1 Input dan Output 12
4.2 Analisis Perbandingan 24
Bab 5: Kesimpulan dan Saran **26**
5.1 Kesimpulan 26
5.2 Saran 26
Bab 6: Lampiran **27**

Bab 1: Deskripsi Masalah

Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol, yang menentukan bentuk dan arah kurva. Cara membuatnya cukup mudah, yaitu dengan menentukan titik-titik kontrol dan menghubungkannya dengan kurva. Kurva Bézier memiliki banyak kegunaan dalam kehidupan nyata, seperti *pen tool*, animasi yang halus dan realistis, membuat desain produk yang kompleks dan presisi, dan membuat font yang indah dan unik. Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan.

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P_0 sampai P_n , 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 P_0, sedangkan titik kontrol terakhir adalah P_3. Titik kontrol P_1 dan P_2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.



Gambar 1. Kurva Bézier Kubik
(Sumber: https://id.wikipedia.org/wiki/Kurva_B%C3%A9zier)

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P_0 dan P_1 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 Q_0 yang berada pada garis yang dibentuk oleh P_0 dan P_1, 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 P_0 ke P_1. Misalnya ketika *t* = 0.25, maka *B*(*t*) adalah seperempat jalan dari titik P_0 ke P_1. sehingga seluruh rentang variasi nilai *t* dari 0 hingga 1 akan membuat persamaan *B*(*t*) membentuk sebuah garis lurus dari P_0 ke P_1.

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P_2, dengan P_0 dan P_2 sebagai titik kontrol awal dan akhir, dan P_1 menjadi titik kontrol antara. Dengan menyatakan titik Q_1 terletak diantara garis yang menghubungkan P_1 dan P_2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q_0 berada, maka dapat dinyatakan sebuah titik baru, R_0 yang berada diantara garis yang menghubungkan Q_0 dan Q_1 yang bergerak membentuk kurva Bézier kuadratik terhadap titik P_0 dan P_2. Berikut adalah uraian persamaannya.

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

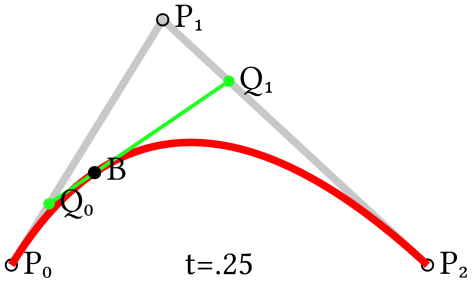
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

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

dengan melakukan substitusi nilai Q_0 dan Q_1, 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*.

Bab 2: Landasan Teori

2.1 Algoritma Divide and Conquer

Divide and conquer merupakan strategi algoritma yang digunakan dalam pemrograman untuk memecahkan persoalan atau masalah yang kompleks menjadi bagian-bagian yang lebih kecil, lebih mudah diatasi, dan kemudian menggabungkan solusi-solusi dari bagian-bagian tersebut untuk mendapatkan solusi dari masalah awal. Pendekatan ini terdiri dari tiga langkah utama:

1. Divide (Membagi)

   Masalah besar dibagi menjadi beberapa masalah kecil yang lebih mudah dipecahkan atau biasanya dalam beberapa kasus membagi dua persoalan tersebut. Proses pemecahan ini dilakukan hingga masalah mencapai ukuran yang cukup kecil sehingga dapat dipecahkan dengan mudah.

2. Conquer (Menaklukkan)

   Setelah membagi masalah menjadi bagian-bagian yang lebih kecil, langkah selanjutnya adalah menyelesaikan setiap bagian tersebut secara terpisah. Pemecahan masalah pada bagian-bagian kecil ini bisa dilakukan dengan menggunakan pendekatan lain yang lebih sederhana atau rekursif.

3. Combine (Menggabungkan)

   Setelah semua bagian kecil dari masalah diselesaikan, solusi dari setiap bagian tersebut digabungkan kembali menjadi solusi untuk masalah awal.

Keuntungan utama dari pendekatan Divide and Conquer adalah kemampuannya untuk mengurangi kompleksitas masalah dengan memecahnya menjadi bagian-bagian yang lebih kecil. Pendekatan ini seringkali digunakan untuk menyelesaikan masalah-masalah dalam algoritma pencarian, pengurutan, dan optimisasi. Beberapa contoh algoritma yang menggunakan pendekatan Divide and Conquer termasuk algoritma QuickSort, algoritma Binary Search, dan algoritma Strassen untuk perkalian matriks.

2.2 Algoritma Brute Force

Algoritma brute force adalah pendekatan dalam pemrograman komputer di mana sebuah masalah diselesaikan melalui metode langsung dan sederhana, tanpa menggunakan optimisasi atau teknik yang kompleks. Istilah "brute force" sendiri berasal dari bahasa Inggris yang secara harfiah berarti "kekerasan kasar" atau "kekuatan mentah".

Pendekatan brute force mencakup mencoba setiap kemungkinan solusi secara sistematis, bahkan jika itu berarti mencoba semua kombinasi yang mungkin. Meskipun pendekatan ini sering kali tidak efisien, terutama untuk masalah dengan ukuran atau kompleksitas yang besar, namun ia sering digunakan karena kesederhanaannya dan kepastiannya dalam menemukan solusi yang benar. Beberapa karakteristik algoritma brute force:

1. Pemecahan secara langsung

   Algoritma brute force menyelesaikan masalah dengan cara mencoba semua kemungkinan secara berurutan tanpa mengimplementasikan optimisasi atau strategi kompleks lainnya.

2. Penggunaan kekuatan komputasi

   Pendekatan ini mengandalkan kekuatan komputasi untuk menghitung dan memeriksa semua kemungkinan solusi. Ini bisa sangat memakan waktu dan sumber daya komputasi, terutama untuk masalah dengan ukuran yang besar.

3. Kemungkinan solusi yang benar

   Karena algoritma brute force mencoba semua kemungkinan, maka hasilnya sering kali benar. Namun, efisiensi waktu yang rendah bisa menjadi kelemahan utamanya.

Contoh penerapan algoritma brute force adalah dalam pencarian linier, pencarian string, atau pemecahan masalah optimisasi sederhana. Meskipun demikian, untuk masalah yang lebih kompleks, seperti masalah optimisasi kombinatorial yang besar, algoritma brute force sering kali tidak praktis karena kompleksitas waktu dan ruangnya yang tinggi.

Bab 3: Implementasi Program

3.1 Analisis dan implementasi dalam algoritma brute force sebagai algoritma

Algoritma brute force adalah suatu pendekatan yang mudah untuk memecahkan suatu masalah dengan sangat sederhana, langsung, dan dengan cara yang cukup jelas. Pada program ini terdapat fungsi “brute_force_bezier_kurva” yaitu implementasi algoritma brute force untuk menghitung titik-titik pada kurva Bézier kuadratik dengan menghitung nilai untuk parameter “t” yang dicantumkan. Kemudian algoritma brute force ini akan menghitung posisi tiitk pada kurva untuk setiap nilai “t”.

Pada fungsi brute force ini menggunakan empat parameter yaitu p0, p1, dan p2 yang merupakan titik kontrol yang mendefinisikan kurva Bézier kuadratik, dan “nilai_t” adalah daftar nilai parameter “t” yang ingin dievaluasi. Algoritma ini kemudian melakukan iterasi berdasarkan setiap nilai “t” dalam “nilai_t”. Untuk setiap nilai “t” di mana algoritma menghitung posisi titik pada kurva Bézier dengan menggunakan rumus dasar kurva Bézier.

Titik-titik yang dihasilkan dari perhitungan ini kemudian ditambahkan ke dalam daftar “kurva_point”. Kemudian fungsi mengembalikan nilai array pada NumPy yang berisi semua titik-titik yang dihitung. Pada “return np.array” untuk menghitung operasi penjumlahan dan perkalian pada vector titik dengan benar menggunakan NumPy. Untuk semua titik kontrol p0, p1, p2 serta titik-tiitk kurva yang dihasilkan oleh algoritma ini direpresentasikan sebagai vektor NumPy untuk memungkinkan operasi vektorisasi yang efisien.

3.2 Analisis dan implementasi dalam algoritma divide and conquer sebagai algoritma

Algoritma divide and conquer adalah suatu pendekatan dalam memecahkan suatu persoalan dengan membagi persoalan menjadi beberapa upaya persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil kemudian menyelesaikan masing-masing upaya persoalan untuk persoalan ukuran kecil dengan

cara secara langsung sedangkan untuk persoalan yang berukuran besar dengan cara rekursif. Pada program ini terdapat fungsi “divide_and_conquer_bezier_kurva” yaitu implementasi algoritma divide and conquer untuk menghitung titik-titik pada kurva Bézier kuadratik. Algoritma ini membagi interval nilai parameter “t” menjadi dua bagian kemudian secara rekursif memanggil dirinya sendiri untuk menghitung titik-titik pada setiap bagian yang kemudian hasil dari kedua bagian tersebut digabungkan menjadi satu array hasil.

Fungsi ini menggunakan empat parameter p0, p1, dan p2 adalah titik kontrol yang mendefinisikan kurva Bézier kuadratik, dan “nilai_t” adalah daftar nilai parameter “t” yang ingin dievaluasi. Algoritma ini pertama-tama memeriksa apakah daftar “nilai_t” hanya berisi satu nilai “t”. Jika iya, maka algoritma langsung memanggil fungsi “kuadrat_bezier_curve” untuk menghitung posisi titik pada kurva Bézier untuk nilai “t” tersebut. Jika daftar “nilai_t” memiliki lebih dari satu nilai, algoritma membagi daftar tersebut menjadi dua bagian (sekitar nilai tengahnya). Kemudian algoritma secara rekursif memanggil dirinya sendiri untuk menghitung titik-titik pada kurva Bézier untuk kedua bagian dari daftar “nilai_t”. Hasil dari kedua pemanggilan rekursif digabungkan menjadi satu array dengan fungsi “np.vstack”. Lalu fungsi akan mengembalikkan nilai array NumPy yang berisi semua titik-titik yang dihitung. Penggunaan “np.vstack” untuk penggabungan hasil dari kedua bagian kurva Bézier menjadi satu array hasil. Sama dengan algoritma brute force, algoritma divide and conquer menggunakan NumPy untuk operasi vektorisasi yang lebih efisien.

3.3 Source Code Program

```
import numpy as np
import matplotlib.pyplot as plt

def kuadrat_bezier_kurva(p0, p1, p2, t):
# Fungsi untuk menghitung titik pada kurva Bézier kuadratik
return (1 - t)**2 * p0 + (1 - t) * p1 + t**2 * p2

def brute_force_bezier_kurva(p0, p1, p2, nilai_t):
# Metode Brute Force untuk menghitung titik pada kurva Bézier kuadratik
```

```
kurva_point = []
for t in nilai_t:
q0 = (1 - t) * p0 + t * p1
q1 = (1 - t) * p1 + t * p2
r0 = (1 - t)**2 * p0 + (1 - t) *t * p1 + t**2 * p2
kurva_point.append(r0)
return np.array(kurva_point)

def divide_and_conquer_bezier_kurva(p0, p1, p2, nilai_t):
# Metode Divide and Conquer untuk menghitung titik pada kurva Bézier
kuadratik
if len(nilai_t) == 1:
return kuadrat_bezier_kurva(p0, p1, p2, nilai_t[0])
else:
t_mid = len(nilai_t) // 2 # Divide and conquer merupakan strategi
algoritma dengan solusi bagi dua
kurva_kiri = divide_and_conquer_bezier_kurva(p0, p1, p2,
nilai_t[:t_mid]) # Sehingga didapatlah titik tengah
kurva_kanan = divide_and_conquer_bezier_kurva(p0, p1, p2,
nilai_t[t_mid:])
return np.vstack((kurva_kiri, kurva_kanan))

# Titik-titik kontrol

import numpy as np

# Input untuk titik p0
# Silahkan ganti x dan y yang diinginkan
p0 = np.array([6, 6])
p1 = np.array([3, 2])
p2 = np.array([7, 9])

# Menentukan jumlah iterasi
# Anda dapat memasukkan jumlah iterasi yang anda inginkan
jlh_iterasi = 13

# Header tabel iterasi
print("Tabel Iterasi:")
print("i\tt\tR0\t\tp0\t\tp1\t\tp2")

# Membuat array untuk menyimpan titik-titik iterasi
```

```
iterasi_point = []

# Membuat array parameter t
nilai_t = np.linspace(0, 1, jlh_iterasi + 1)

# Melakukan iterasi dan mencetak nilai t dan R0 pada setiap iterasi
for i, t in enumerate(nilai_t):
r0 = (1 - t)**2 * p0 + (1 - t) * p1 + t**2 * p2
print(f"{i}\t{t}\t{r0}\t{p0}\t{p1}\t{p2}")
iterasi_point.append(r0)

# Membuat array parameter t untuk plot kurva Bézier kuadratik
nilai_t_plot = np.linspace(0, 1, 100)

# Menghitung titik pada kurva Bézier kuadratik dengan brute force
kurva_point_brute_force = brute_force_bezier_kurva(p0, p1, p2,
nilai_t_plot)

# Menghitung titik pada kurva Bézier kuadratik dengan divide and
conquer
kurva_point_divide_conquer = divide_and_conquer_bezier_kurva(p0, p1,
p2, nilai_t_plot)

# Plotting kurva Bézier kuadratik dan titik-titik iterasi
plt.plot(kurva_point_brute_force[:, 0], kurva_point_brute_force[:, 1],
label='Brute Force')
plt.plot(kurva_point_divide_conquer[:, 0],
kurva_point_divide_conquer[:, 1], label='Divide and Conquer')
plt.scatter([p0[0], p1[0], p2[0]], [p0[1], p1[1], p2[1]],
coloTr='green', label='Titik Kontrol')
for i, txt in enumerate([p0, p1, p2]):
plt.annotate(f'P{i}', (txt[0], txt[1]), textcoords="offset points",
xytext=(0, 10), ha='center')
for i, txt in enumerate(iterasi_point):
plt.annotate(f'R{i}', (txt[0], txt[1]), textcoords="offset points",
xytext=(0, 10), ha='center')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Kurva Bézier Kuadrat dengan Divide and Conquer dan Brute
Force') # Judul kurva
plt.grid(True)
```

```
plt.legend()
plt.axis('equal')
plt.show() # Menampilkan kurva
```

Bab 4: Eksperimen dan Hasil

4.1 Input dan Output

Berikut adalah enam hasil percobaan kurva Bézier dalam menggunakan algoritma brute force dan divide and conquer dengan menginputkan suatu nilai di dalam program

1. p0 = [6, 4]; p1 = [3, 2]; p2 = [4, 6], iterasi = 10

   Input 1:

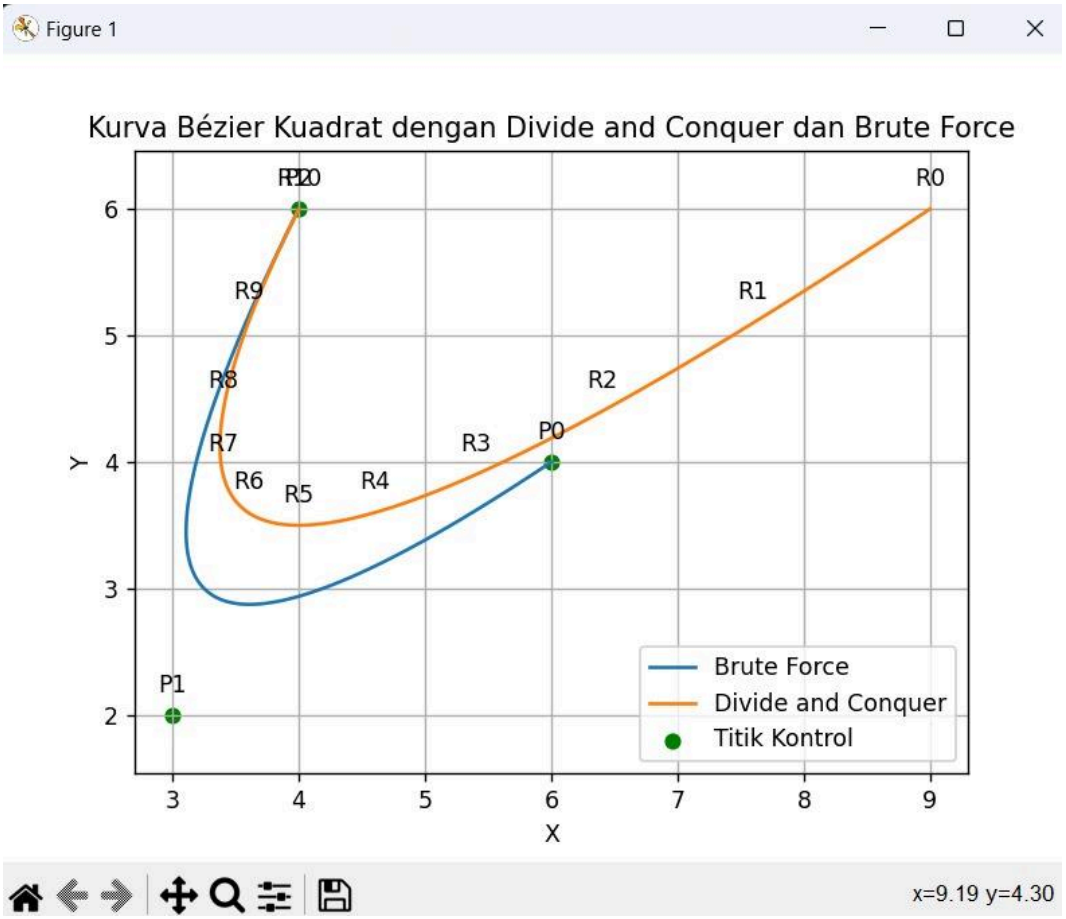
```
# Input untuk titik p0
# Silahkan ganti x dan y yang diinginkan
p0 = np.array([6, 4])
p1 = np.array([3, 2])
p2 = np.array([4, 6])

# Menentukan jumlah iterasi
# Anda dapat memasukkan jumlah iterasi yang anda inginkan
jlh_iterasi = 10
```

   Output 1:

```
Tabel Iterasi:
i t R0    p0    p1    p2
0 0.0 [9. 6.] [6 4] [3 2] [4 6]
1 0.1 [7.6 5.1] [6 4] [3 2] [4 6]
2 0.2 [6.4 4.4] [6 4] [3 2] [4 6]
3 0.30000000000000004 [5.4 3.9] [6 4] [3 2] [4 6]
4 0.4 [4.6 3.6] [6 4] [3 2] [4 6]
5 0.5 [4.  3.5] [6 4] [3 2] [4 6]
6 0.6000000000000001  [3.6 3.6] [6 4] [3 2] [4 6]
7 0.7000000000000001  [3.4 3.9] [6 4] [3 2] [4 6]
8 0.8 [3.4 4.4] [6 4] [3 2] [4 6]
9 0.9 [3.6 5.1] [6 4] [3 2] [4 6]
10  1.0 [4. 6.] [6 4] [3 2] [4 6]
```

Kurva 1:



2. p0 = [3, 7]; p1 = [2, 6]; p2 = [8, 2], iterasi = 10

Input 2:

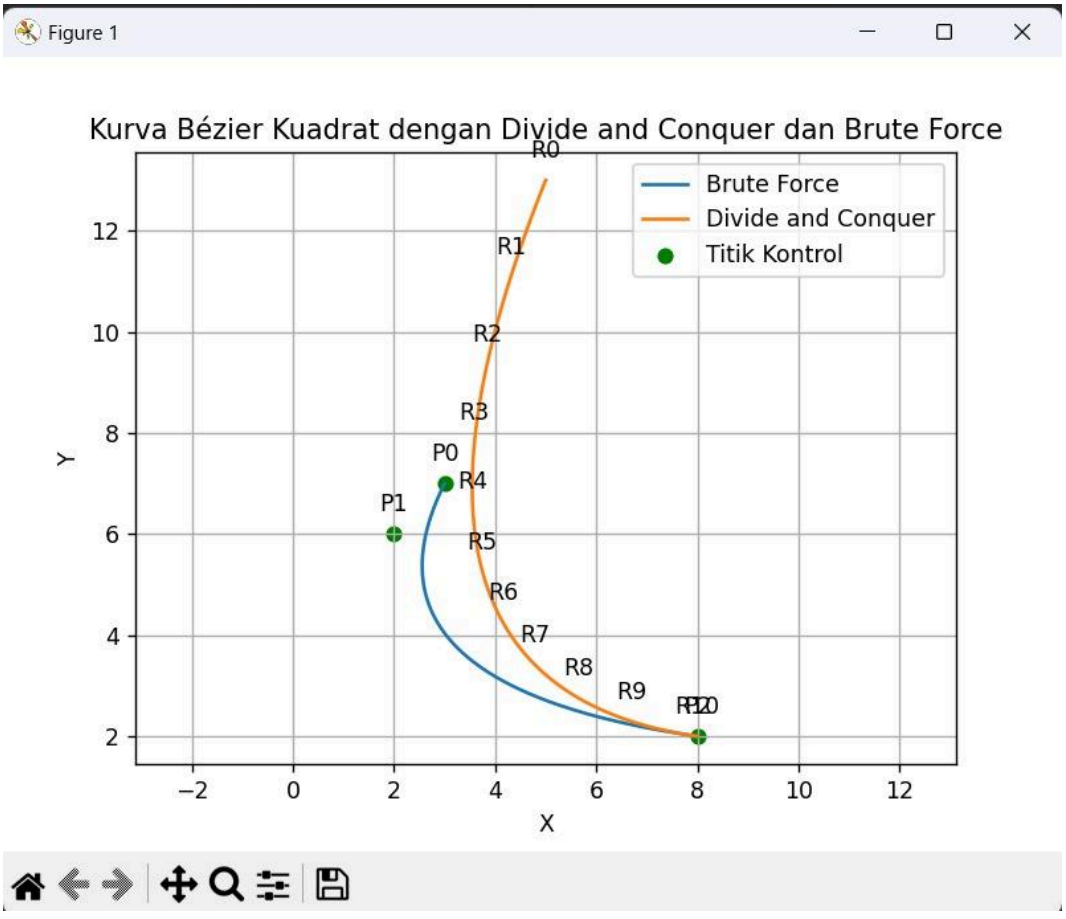
```
# Input untuk titik p0
# Silahkan ganti x dan y yang diinginkan
p0 = np.array([3, 7])
p1 = np.array([2, 6])
p2 = np.array([8, 2])

# Menentukan jumlah iterasi
# Anda dapat memasukkan jumlah iterasi yang anda inginkan
jlh_iterasi = 10
```

Output 2:

```
Tabel Iterasi:
i t R0    p0    p1    p2
0 0.0 [ 5. 13.] [3 7] [2 6] [8 2]
1 0.1 [ 4.31 11.09] [3 7] [2 6] [8 2]
2 0.2 [3.84 9.36] [3 7] [2 6] [8 2]
3 0.30000000000000004 [3.59 7.81] [3 7] [2 6] [8 2]
4 0.4 [3.56 6.44] [3 7] [2 6] [8 2]
5 0.5 [3.75 5.25] [3 7] [2 6] [8 2]
6 0.6000000000000001  [4.16 4.24] [3 7] [2 6] [8 2]
7 0.7000000000000001  [4.79 3.41] [3 7] [2 6] [8 2]
8 0.8 [5.64 2.76] [3 7] [2 6] [8 2]
9 0.9 [6.71 2.29] [3 7] [2 6] [8 2]
10  1.0 [8. 2.] [3 7] [2 6] [8 2]
```

Kurva 2:



3. p0 = [1, 4]; p1 = [3, 5]; p2 = [7, 9], iterasi = 10

Input 3:

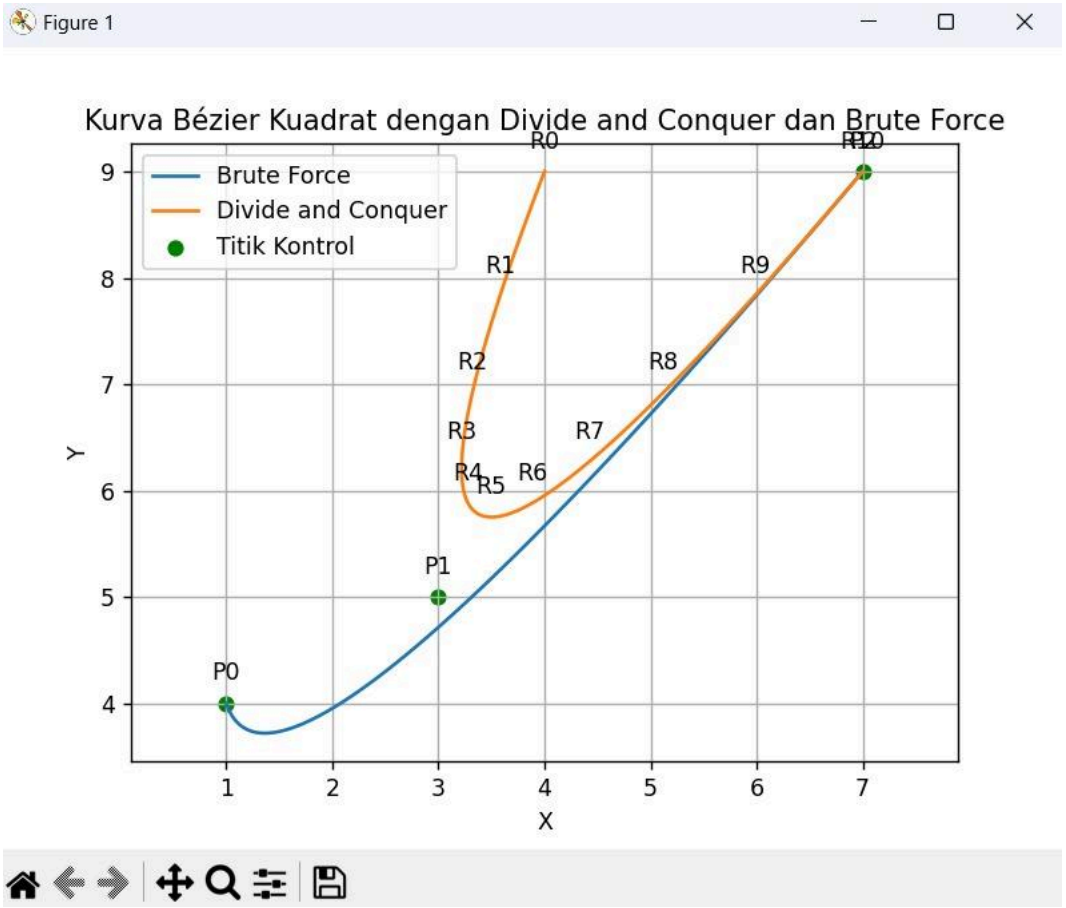
```
# Input untuk titik p0
# Silahkan ganti x dan y yang diinginkan
p0 = np.array([1, 4])
p1 = np.array([3, 5])
p2 = np.array([7, 9])

# Menentukan jumlah iterasi
# Anda dapat memasukkan jumlah iterasi yang anda inginkan
jlh_iterasi = 10
```

Output 3:

```
Tabel Iterasi:
i t R0    p0    p1    p2
0 0.0 [4. 9.] [1 4] [3 5] [7 9]
1 0.1 [3.58 7.83] [1 4] [3 5] [7 9]
2 0.2 [3.32 6.92] [1 4] [3 5] [7 9]
3 0.30000000000000004 [3.22 6.27] [1 4] [3 5] [7 9]
4 0.4 [3.28 5.88] [1 4] [3 5] [7 9]
5 0.5 [3.5  5.75] [1 4] [3 5] [7 9]
6 0.6000000000000001  [3.88 5.88] [1 4] [3 5] [7 9]
7 0.7000000000000001  [4.42 6.27] [1 4] [3 5] [7 9]
8 0.8 [5.12 6.92] [1 4] [3 5] [7 9]
9 0.9 [5.98 7.83] [1 4] [3 5] [7 9]
10  1.0 [7. 9.] [1 4] [3 5] [7 9]
```

Kurva 3:



4. p0 = [3, 7]; p1 = [5, 4]; p2 = [2, 6], iterasi = 10

   Input 4:

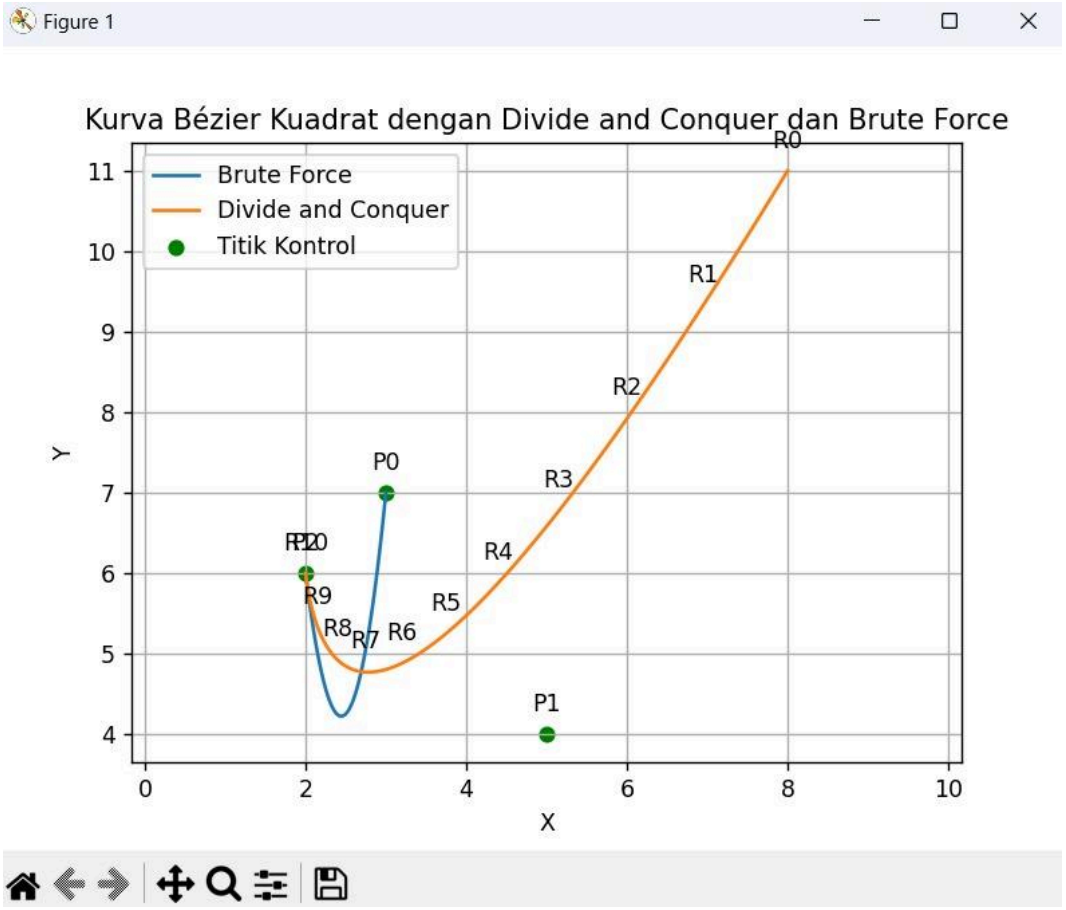
```
# Input untuk titik p0
# Silahkan ganti x dan y yang diinginkan
p0 = np.array([3, 7])
p1 = np.array([5, 4])
p2 = np.array([2, 6])

# Menentukan jumlah iterasi
# Anda dapat memasukkan jumlah iterasi yang anda inginkan
jlh_iterasi = 10
```

Output 4:

```
Tabel Iterasi:
i t R0    p0    p1    p2
0 0.0 [ 8. 11.] [3 7] [5 4] [2 6]
1 0.1 [6.95 9.33] [3 7] [5 4] [2 6]
2 0.2 [6.   7.92] [3 7] [5 4] [2 6]
3 0.30000000000000004 [5.15 6.77] [3 7] [5 4] [2 6]
4 0.4 [4.4  5.88] [3 7] [5 4] [2 6]
5 0.5 [3.75 5.25] [3 7] [5 4] [2 6]
6 0.6000000000000001  [3.2  4.88] [3 7] [5 4] [2 6]
7 0.7000000000000001  [2.75 4.77] [3 7] [5 4] [2 6]
8 0.8 [2.4  4.92] [3 7] [5 4] [2 6]
9 0.9 [2.15 5.33] [3 7] [5 4] [2 6]
10  1.0 [2. 6.] [3 7] [5 4] [2 6]
```

Kurva 4:



5. p0 = [3, 7]; p1 = [5, 9]; p2 = [8, 14], iterasi = 10

   Input 5:

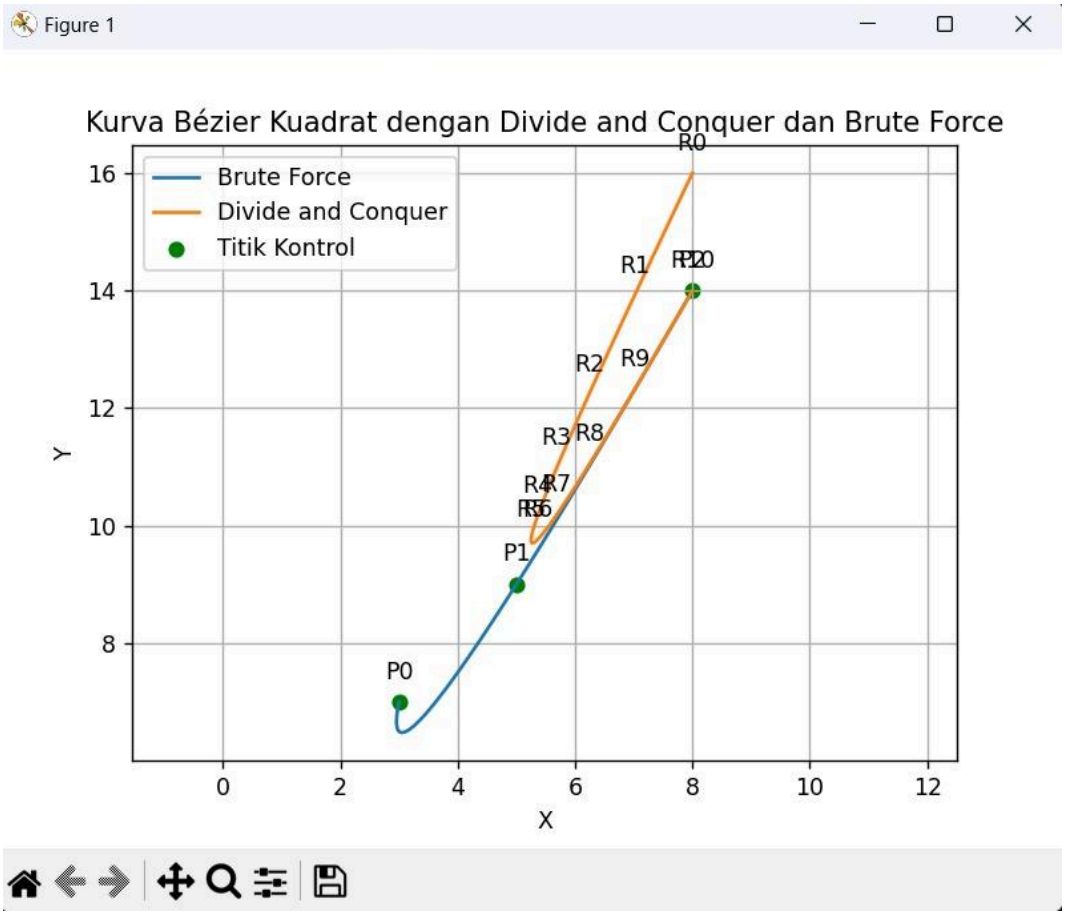
```
# Input untuk titik p0
# Silahkan ganti x dan y yang diinginkan
p0 = np.array([3, 7])
p1 = np.array([5, 9])
p2 = np.array([8, 14])

# Menentukan jumlah iterasi
# Anda dapat memasukkan jumlah iterasi yang anda inginkan
jlh_iterasi = 10
```

   Output 5:

```
Tabel Iterasi:
i t R0    p0    p1    p2
0 0.0 [ 8. 16.] [3 7] [5 9] [ 8 14]
1 0.1 [ 7.01 13.91] [3 7] [5 9] [ 8 14]
2 0.2 [ 6.24 12.24] [3 7] [5 9] [ 8 14]
3 0.30000000000000004 [ 5.69 10.99] [3 7] [5 9] [ 8 14]
4 0.4 [ 5.36 10.16] [3 7] [5 9] [ 8 14]
5 0.5 [5.25 9.75] [3 7] [5 9] [ 8 14]
6 0.6000000000000001  [5.36 9.76] [3 7] [5 9] [ 8 14]
7 0.7000000000000001  [ 5.69 10.19] [3 7] [5 9] [ 8 14]
8 0.8 [ 6.24 11.04] [3 7] [5 9] [ 8 14]
9 0.9 [ 7.01 12.31] [3 7] [5 9] [ 8 14]
10  1.0 [ 8. 14.] [3 7] [5 9] [ 8 14]
```

Kurva 5:



6. p0 = [16, 10]; p1 = [15, 15]; p2 = [2, 14], iterasi = 10

   Input 6:

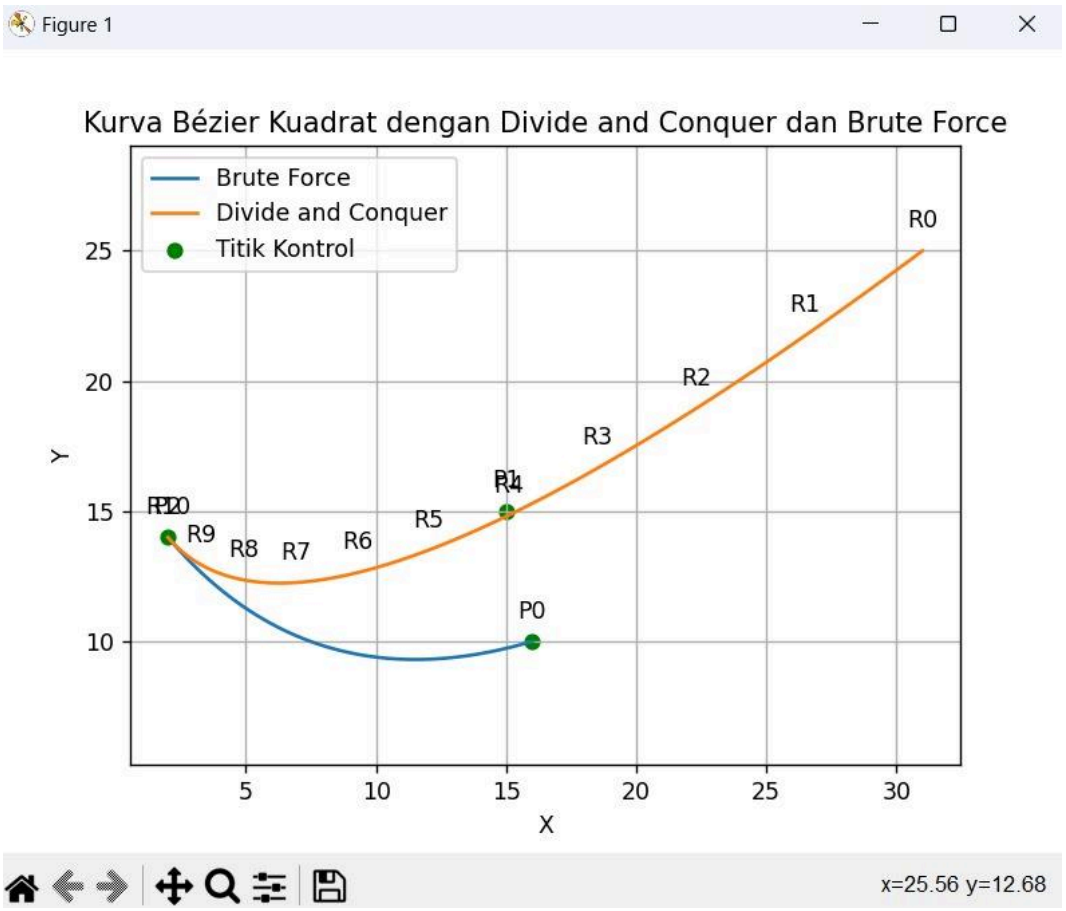
```
# Input untuk titik p0
# Silahkan ganti x dan y yang diinginkan
p0 = np.array([16, 10])
p1 = np.array([15, 15])
p2 = np.array([2, 14])

# Menentukan jumlah iterasi
# Anda dapat memasukkan jumlah iterasi yang anda inginkan
jlh_iterasi = 10
```

Output 6:

```
Tabel Iterasi:
i t R0    p0    p1    p2
0 0.0 [31. 25.] [16 10] [15 15] [ 2 14]
1 0.1 [26.48 21.74] [16 10] [15 15] [ 2 14]
2 0.2 [22.32 18.96] [16 10] [15 15] [ 2 14]
3 0.30000000000000004 [18.52 16.66] [16 10] [15 15] [ 2 14]
4 0.4 [15.08 14.84] [16 10] [15 15] [ 2 14]
5 0.5 [12.  13.5] [16 10] [15 15] [ 2 14]
6 0.6000000000000001  [ 9.28 12.64] [16 10] [15 15] [ 2 14]
7 0.7000000000000001  [ 6.92 12.26] [16 10] [15 15] [ 2 14]
8 0.8 [ 4.92 12.36] [16 10] [15 15] [ 2 14]
9 0.9 [ 3.28 12.94] [16 10] [15 15] [ 2 14]
10  1.0 [ 2. 14.] [16 10] [15 15] [ 2 14]
```

Kurva 6:



7. p0 = [6, 6]; p1 = [3, 2]; p2 = [7, 9], iterasi = 13

   Input 7:

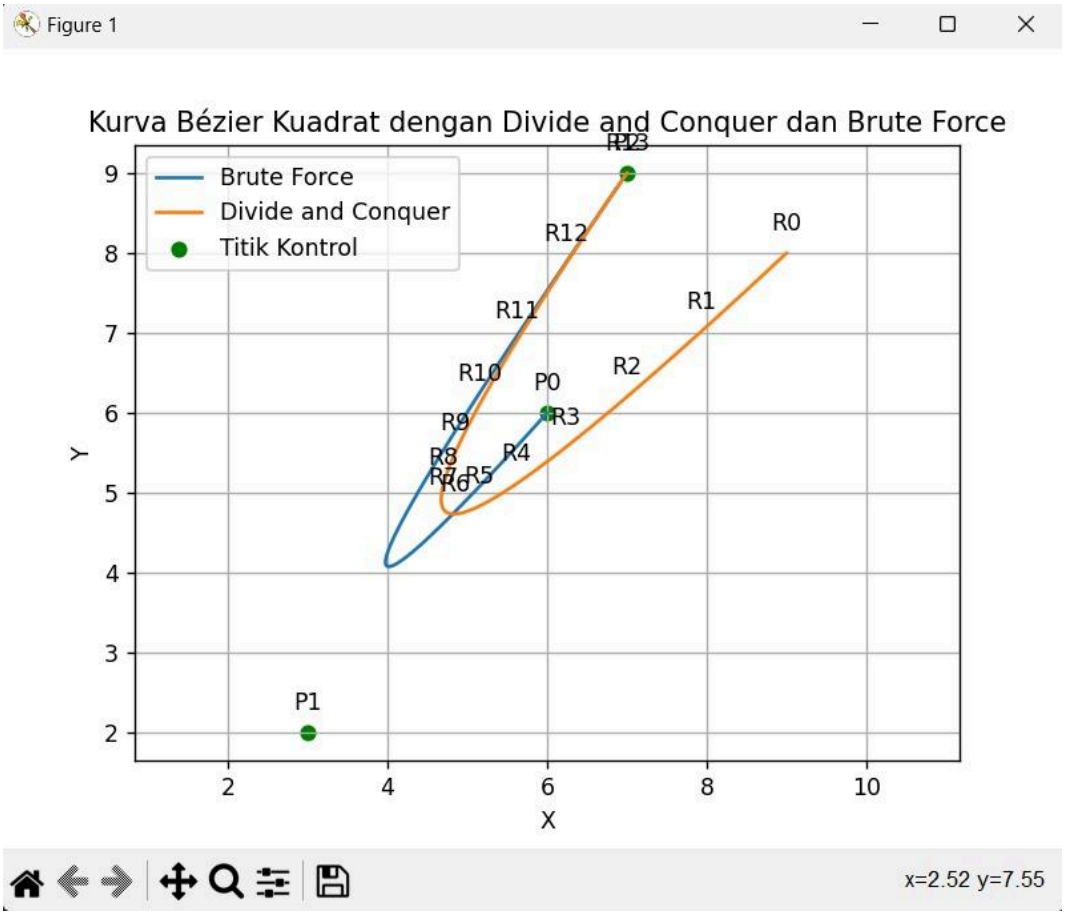
```
# Input untuk titik p0
# Silahkan ganti x dan y yang diinginkan
p0 = np.array([6, 6])
p1 = np.array([3, 2])
p2 = np.array([7, 9])

# Menentukan jumlah iterasi
# Anda dapat memasukkan jumlah iterasi yang anda inginkan
jlh_iterasi = 13
```

   Output 7:

```
Tabel Iterasi:
i t R0    p0    p1    p2
0 0.0 [9. 8.] [6 6] [3 2] [7 9]
1 0.07692307692307693 [7.92307692 7.01183432] [6 6] [3 2] [7 9]
2 0.15384615384615385 [7.         6.20118343] [6 6] [3 2] [7 9]
3 0.23076923076923078 [6.23076923 5.56804734] [6 6] [3 2] [7 9]
4 0.3076923076923077  [5.61538462 5.11242604] [6 6] [3 2] [7 9]
5 0.38461538461538464 [5.15384615 4.83431953] [6 6] [3 2] [7 9]
6 0.46153846153846156 [4.84615385 4.73372781] [6 6] [3 2] [7 9]
7 0.5384615384615385  [4.69230769 4.81065089] [6 6] [3 2] [7 9]
8 0.6153846153846154  [4.69230769 5.06508876] [6 6] [3 2] [7 9]
9 0.6923076923076923  [4.84615385 5.49704142] [6 6] [3 2] [7 9]
10  0.7692307692307693  [5.15384615 6.10650888] [6 6] [3 2] [7 9]
11  0.8461538461538463  [5.61538462 6.89349112] [6 6] [3 2] [7 9]
12  0.9230769230769231  [6.23076923 7.85798817] [6 6] [3 2] [7 9]
13  1.0 [7. 9.] [6 6] [3 2] [7 9]
```

Kurva 7:



8. p0 = [6, 6]; p1 = [3, 2]; p2 = [7, 9], iterasi = 10

   Input 8:

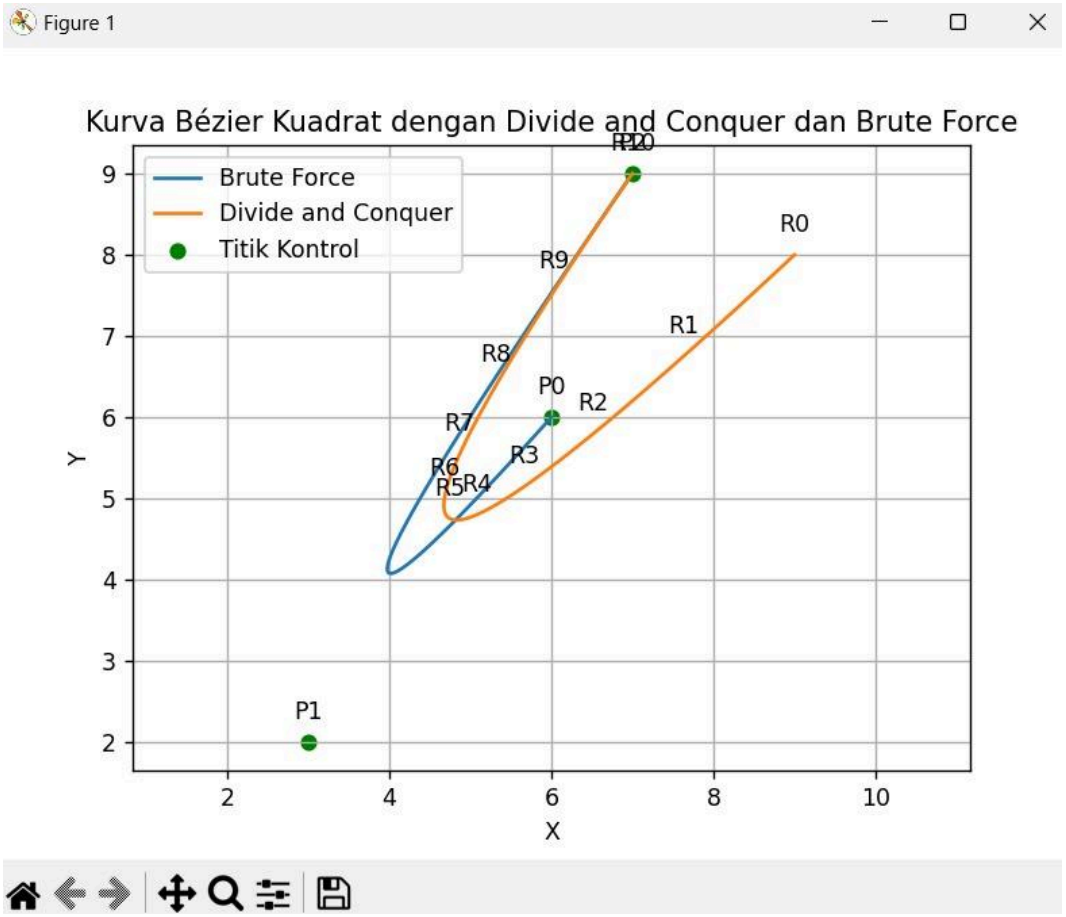
```
# Input untuk titik p0
# Silahkan ganti x dan y yang diinginkan
p0 = np.array([6, 6])
p1 = np.array([3, 2])
p2 = np.array([7, 9])

# Menentukan jumlah iterasi
# Anda dapat memasukkan jumlah iterasi yang anda inginkan
jlh_iterasi = 10
```

Output 8:

```
Tabel Iterasi:
i t R0    p0    p1    p2
0 0.0 [9. 8.] [6 6] [3 2] [7 9]
1 0.1 [7.63 6.75] [6 6] [3 2] [7 9]
2 0.2 [6.52 5.8 ] [6 6] [3 2] [7 9]
3 0.30000000000000004 [5.67 5.15] [6 6] [3 2] [7 9]
4 0.4 [5.08 4.8 ] [6 6] [3 2] [7 9]
5 0.5 [4.75 4.75] [6 6] [3 2] [7 9]
6 0.6000000000000001  [4.68 5.  ] [6 6] [3 2] [7 9]
7 0.7000000000000001  [4.87 5.55] [6 6] [3 2] [7 9]
8 0.8 [5.32 6.4 ] [6 6] [3 2] [7 9]
9 0.9 [6.03 7.55] [6 6] [3 2] [7 9]
10  1.0 [7. 9.] [6 6] [3 2] [7 9]
```

Kurva 8:



4.2 Analisis Perbandingan

Berdasarkan algoritma yang telah dibuat yaitu algoritma brute force dan divide and conquer, terdapat perbedaan yang signifikan dari hasil yang diperoleh. Untuk melakukan analisis perbandingan antara algoritma brute force dan divide and conquer pada konteks kurva Bézier kuadratik, kita perlu mempertimbangkan beberapa faktor, seperti kompleksitas waktu, kompleksitas ruang, dan kecepatan konvergensi.

1. Kompleksitas Waktu
   Algoritma brute force secara langsung menghitung setiap titik pada kurva Bézier untuk setiap nilai parameter t, sehingga memiliki kompleksitas waktu $O(n^2)$ di mana n adalah jumlah titik kontrol dan m adalah jumlah titik pada kurva yang dihasilkan. Sedangkan algoritma divide and conquer membagi masalah menjadi dua sub-masalah yang lebih kecil, yang kemudian dipecahkan secara rekursif. Hal ini menghasilkan kompleksitas waktu $O(n \log n)$, yang lebih efisien daripada pendekatan brute force, terutama ketika m cukup besar.
2. Kompleksitas Ruang
   Algoritma brute force memerlukan ruang memori yang proporsional dengan jumlah titik yang dihasilkan, sehingga kompleksitas ruangnya adalah $O(n)$. Sedangkan algoritma divide and conquer memerlukan ruang tambahan untuk menyimpan hasil dari setiap submasalah yang dipecahkan secara rekursif, yang akan meningkatkan kompleksitas ruangnya menjadi $O(n \log n)$ juga. Namun, algoritma divide and conquer mungkin lebih efisien dalam penggunaan ruang karena membagi masalah menjadi bagian-bagian yang lebih kecil.
3. Kecepatan Konvergensi
   Kecepatan konvergensi mengacu pada seberapa cepat algoritma mendekati solusi yang benar. Dalam konteks kurva Bézier, baik algoritma brute force maupun divide and conquer akan menghasilkan titik-titik yang mendekati kurva sebenarnya. Namun, algoritma divide and conquer mungkin lebih cepat mendekati solusi akhir karena pendekatan rekursifnya memungkinkan untuk menemukan solusi yang lebih akurat dengan jumlah iterasi yang lebih sedikit.

Dalam analisis perbandingan antara solusi brute force dan divide and conquer untuk kurva Bézier kuadratik, kompleksitas algoritma menjadi faktor utama yang harus dipertimbangkan. Algoritma brute force secara langsung menghitung setiap titik pada kurva Bézier untuk setiap nilai parameter t, menyebabkan kompleksitas waktu yang meningkat secara proporsional dengan jumlah titik kontrol dan jumlah titik pada kurva yang dihasilkan, atau $O(n^2)$, di mana n adalah jumlah titik kontrol dan m adalah jumlah titik pada kurva. Sebaliknya, algoritma divide and conquer membagi masalah menjadi dua sub-masalah yang lebih kecil, yang kemudian dipecahkan secara rekursif. Hal ini menghasilkan kompleksitas waktu $O(n \log n)$, yang lebih efisien daripada pendekatan brute force, terutama ketika m cukup besar.

Meskipun demikian, keduanya memiliki kompleksitas ruang yang sama $O(n)$, karena keduanya memerlukan ruang memori yang proporsional dengan jumlah titik yang dihasilkan. Dalam hal kecepatan konvergensi, keduanya cenderung mendekati solusi dengan akurasi yang serupa, namun algoritma divide and conquer mungkin lebih cepat mendekati solusi akhir karena pendekatan rekursifnya memungkinkan untuk menemukan solusi yang lebih akurat dengan jumlah iterasi yang lebih sedikit. Dengan demikian, meskipun algoritma divide and conquer memiliki kompleksitas waktu yang lebih baik, pemilihan antara kedua algoritma ini harus dipertimbangkan dengan cermat, tergantung pada ukuran dan kompleksitas masalah serta kebutuhan akan efisiensi waktu dan ruang.

Bab 5: Kesimpulan dan Saran

5.1 Kesimpulan

Berdasarkan algoritma yang telah dibuat, program ini berhasil mengimplementasikan dua algoritma untuk menghitung kurva Bézier kuadratik, yaitu brute force dan divide and conquer. Kemudian Program dapat menghitung titik pada kurva Bézier kuadratik dengan kedua pendekatan yang diberikan dan juga dapat memplot hasilnya menggunakan Matplotlib. Dari segi efisiensi, algoritma divide and conquer memiliki kompleksitas waktu yang lebih baik daripada pendekatan brute force, terutama ketika jumlah titik pada kurva cukup besar. Serta program memberikan visualisasi yang jelas tentang bagaimana kedua algoritma beroperasi dan hasil yang dihasilkan.

5.2 Saran

Beberapa pelajaran yang kami dapatkan selama melakukan pengerjaan tugas kecil ini yaitu pentingnya untuk melakukan optimisasi kode, terutama dalam hal penggunaan memori dan kompleksitas waktu. Ini dapat mencakup mempertimbangkan penggunaan struktur data yang lebih efisien atau algoritma yang lebih cepat untuk mencapai tujuan yang sama. Kami juga menyarankan untuk menggunakan penamaan variabel yang lebih deskriptif agar kode menjadi lebih mudah dipahami oleh pembaca, terutama bagi mereka yang baru mempelajari atau memodifikasi kode. Selalu meningkatkan penanganan error dalam program. Ini termasuk memeriksa input pengguna untuk memastikan bahwa mereka memasukkan nilai yang valid, serta memberikan pesan kesalahan yang informatif jika terjadi kesalahan. Disarankan untuk menambahkan dokumentasi yang lebih rinci dalam kode program, termasuk penjelasan tentang bagaimana setiap fungsi bekerja, apa yang diharapkan dari input, dan apa yang dihasilkan sebagai output. Serta tentunya sebagai makhluk sosial disarankan untuk menerima masukan dari pengguna program dan mempertimbangkan saran atau permintaan fitur yang dapat meningkatkan kualitas dan fungsionalitas program.

Bab 6: Lampiran

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.		✓

Tautan *Repository GitHub*

https://github.com/tazzkirahamaliah/Tucil2_10023500_10023608