LAPORAN TUGAS KECIL
IF2211 STRATEGI ALGORITMA

Disusun Oleh :
M Athaullah Daffa Kusuma M (13522044)

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

Daftar Isi

Daftar Isi **2**
Bab 1 : Deskripsi Masalah **3**
1.1. Kurva Bézier 3
Bab 2 : Analisis Landasan Teori **5**
2.1. Algoritma Divide and Conquer 5
2.2. Implementasi Divide and Conquer dalam menyelesaikan Kurva Bézier 5
2.3. Implementasi Brute Force dalam menyelesaikan Kurva Bézier sebagai Algoritma pembanding 5
2.4. Implementasi N-buah titik kontrol Membentuk Kurva Bézier (Bonus) 6
Bab 3 : Implementasi **7**
3.1. DivideNConquer.py 7
3.2. BruteForce.py 8
3.3. Main.py 9
Bab 4 : Uji Coba **12**
4.1. Test Case 1 – Input dengan 3 garis 12
4.2. Test Case 2 – Input dengan 4 garis 15
4.3. Test Case 3 – Input dengan 3 garis dan iterasi banyak 19
4.4. Test Case 4 – Input dengan 5 titik 23
4.5. Test Case 5 – Input dengan 9 titik 27
4.6. Test Case 6 – Input dengan 20 titik 31
Bab 5 : Analisis Hasil Uji Coba **37**
Lampiran **38**
Link Repository 38
Sheets Poin 38

Bab 1 : Deskripsi Masalah

1.1. Kurva Bézier

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.

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.

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

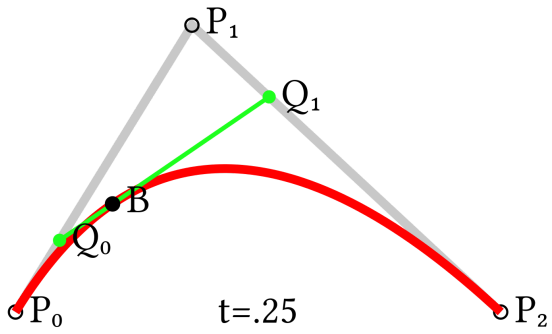
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

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

dengan melakukan substitusi nilai Q_0 dan Q_1, maka diperoleh persamaan sebagai berikut.

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

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.

$$S_0 = B(t) = (1-t)^3 P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P_2 + t^3 P_3, \qquad t \in [0,1]$$

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

Persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik. Maka dari itu diperlukan suatu cara yang dapat men-generalisasikan kurva bezier agar mudah dihitung oleh komputer.

Bab 2 : Analisis Landasan Teori

2.1. Algoritma *Divide and Conquer*

Divide and conquer merupakah salah satu algoritma dari sekian banyaknya jenis algoritma yang ada. Divide and conquer merupakan algoritma yang cukup mudah dan berguna untuk men-generalisasi beberapa masalah. Ide utama dari algoritma ini yaitu membagi sebuah masalah ke beberapa masalah yang lebih kecil, dan dari situ kita solve menggunakan penyelesaian tertentu saat masalah tersebut sudah terbagi menjadi masalah yang cukup kecil. Akurasi dari algoritma ini dibilang cukup akurat karena solusi solve nya mirip seperti algoritma penyelesaian masalah lainnya, akan tetapi algoritma ini membagi masalah besar menjadi masalah-masalah kecil terlebih dahulu. Algoritma ini mempunyai satu masalah yang cukup besar yaitu waktunya yang sangat lambat, dikarenakan ia akan memanggil fungsi ia sendiri secara rekursif untuk masalah yang lebih kecil.

2.2. Implementasi *Divide and Conquer* dalam menyelesaikan Kurva Bézier

Algoritma Divide and conquer dapat menyelesaikan masalah kurva Bézier ini dengan tepat dan akurat tergantung dengan berapa kali iterasi yang dilakukan. Ide dari penyelesaian masalah kurva Bézier yaitu sebagai berikut:

1. Misalkan terdapat tiga buah titik P0, P1, P2
2. Buatlah sebuah titik baru Q0 di tengah P0 dan P1, lalu buat juga sebuah titik baru Q1 di tengah P1 dan P2.
3. Hubungkan Q0 dan Q1 menjadi sebuah garis terbaru
4. Buatlah sebuah titik baru R0 yang berada di tengah Q0 dan Q1
5. Hubungkan garis P0 - R0 - P1

Akan tetapi, proses di atas hanya melakukan iterasi sebanyak 1 kali, untuk membuat kurva yang lebih mulus, idenya yaitu membuat titik tengah lagi di antara P0 dan Q0, lalu di ambil titik tengahnya, kemudian kembali dihubungkan dengan titik tengah Q0 dan R0. Proses ini akan kita lakukan sebanyak iterasi yang kita inginkan. Semakin banyak iterasi yang dilakukan, semakin mulus pula kurva Bézier yang terbentuk.

2.3. Implementasi *Brute Force* dalam menyelesaikan Kurva Bézier sebagai Algoritma pembanding

Brute force merupakan salah satu algoritma yang paling mendasar. Brute force sendiri berarti kita menyelesaikan masalah secara *straightforward* dan sesuai dengan soal atau permasalahannya. Ide dari menggunakan Brute force untuk menyelesaikan kurva Bézier yaitu:

1. Buat sebuah garis Q0 yang menghubungkan P0 dan P1
2. Buat juga garis Q1 yang menghubungkan P1 dan P2

3. Potong Q0 dan Q1 menjadi beberapa titik dan buat menjadi fungsi terhadap t, dimana t merupakan iterasi dari 0 sampai 1, tergantung berapa kali iterasi yang diinginkan
4. Potong Q1 dan Q2 seperti nomor 4
5. Hubungkan titik untuk t di iterasi sekarang dari nomor 3 dan 4 menjadi garis baru, buat juga garis tersebut menjadi fungsi terhadap t
6. Masukkan titik dari hasil nomor 5 ke sebuah lera, iterasi t dari 0 sampai 1 sebanyak iterasi yang diinginkan
7. Hubungkan titik-titik dari nomor 6

Brute force memberikan solusi yang optimal, akan tetapi idenya tidak mudah dipahami. Walaupun sepertinya kompleksitas waktu Divide and Conquer jauh lebih buruk daripada brute force, Algoritma Divide and Conquer sangat mudah dipahami dan mudah dibuat.

2.4. Implementasi N-buah titik kontrol Membentuk Kurva Bézier (Bonus)

Dalam program ini, telah diimplementasikan bonus dimana kita dapat memasukkan N-buat titik kontrol untuk membentuk Kurva Bezier. Ide umum untuk algoritma Divide and conquernya mirip seperti konsep Divide and conquer di 3 titik kontrol (dijelaskan di 2.2.), akan tetapi kita akan membentuk garis dalamnya sampai menjadi 1 buah garis. Sebagai contoh, untuk 4 titik kontrol, jika kita hubungkan keempat titik kontrol tersebut, kita akan mendapat 3 garis utama. Jika kita hubungkan tiap titik di tengah 3 garis utama tersebut, kita akan mendapat 2 garis utama. Sampai pada 1 garis utama lah rekursif ini menjadi basis, dimana kita akan mengiterasi N-buah titik kontrol sebelah kiri, titik tunggal tengah di bentuk ini, dan iterasi N-buah titik kontrol sebelah kanan.

Untuk Algoritma brute force nya, kita hanya mengrekursifkan lerp nya, dimana sebagai contoh jika kita mempunyai 4 titik kontrol, kita mendapat 3 garis utama dari menghubungkan 4 titik kontrol, lalu kita dapat membentuk 2 garis utama dari menghubungkan 3 garis sebelumnya. Proses ini akan dilakukan terus menerus sampai terdapat hanya 1 garis utama, dimana kita akan mengiterasi garis tersebut terhadap nilai t pada fungsi garis2 sebelumnya.

Bab 3 : Implementasi

3.1. DivideNConquer.py

Source code berikut merupakan source code yang memberikan array dari titik-titik hasil Algoritma Divide and Conquer

```
from typing import List
from typing import Optional

class Point:
    def __init__(self, x:float, y:float):
        self.x = x; self.y = y

def midPoint(p1:Point, p2:Point) -> Point:
    return Point((p1.x+p2.x)/2, (p1.y+p2.y)/2)

def bagiBezierDNC(arrBez:List[Point]) -> List[Point]:
    if (len(arrBez) > 2):
        arrRes = [arrBez[0]]
        arrBezNew = []
        for i in range(len(arrBez)-1):
            arrBezNew.append(midPoint(arrBez[i], arrBez[i+1]))
        arrRes = arrRes + bagiBezierDNC(arrBezNew)
        arrRes.append(arrBez[len(arrBez)-1])
        return arrRes
    else:
        return [arrBez[0], midPoint(arrBez[0], arrBez[1]),
arrBez[1]]

def bezierDNC(arrBez:List[Point], currentIterations:int,
wantedIterations:int) -> Optional[Point]:
    if (currentIterations < wantedIterations):
        # declare var dan masukkan bahan iterasi selanjutnya dengan
midpoint2 tertentu
        arrBezRes:List[Point] = []
        arrBezNew = bagiBezierDNC(arrBez)

        iterate1 = arrBezNew[:((len(arrBezNew)//2)+1)]
        arrTemp1:Optional[Point] = bezierDNC(iterate1,
currentIterations+1, wantedIterations)
        if arrTemp1 != None:
            arrBezRes = arrBezRes + arrTemp1

        arrBezRes.append(arrBezNew[len(arrBezNew)//2])

        iterate2 = arrBezNew[len(arrBezNew)//2:]
```

```
        arrTemp2:Optional[Point] = bezierDNC(iterate2,
currentIterations+1, wantedIterations)
        if arrTemp2 != None:
            arrBezRes = arrBezRes + arrTemp2

        return arrBezRes
    else:
        return None
```

Fungsi utama dari source code ini adalah bezierDNC, dimana ia menerima 3 argument, yaitu array dari titik awal yang hendak dihitung bezier-nya, currentIteration yang dimulai dari 0, dan wantedIteration yang merupakan jumlah iterasi yang diinginkan pengguna. Program ini akan mereturn array dari titik-titik yang nantinya dihubungkan menjadi kurva bezier.

3.2. BruteForce.py

Source code berikut merupakan source code yang memberikan array dari titik-titik hasil Algoritma Brute force

```
from typing import List

class Point:
    def __init__(self, x:float, y:float):
        self.x = x; self.y = y

def midPoint(p1:Point, p2:Point) -> Point:
    return Point((p1.x+p2.x)/2, (p1.y+p2.y)/2)

def bagiBezierBF(arrBez: List[Point], t:int) -> Point:
    if (len(arrBez) > 2):
        arrLerp:List[Point] = []
        for i in range(len(arrBez)-1):
            arrLerp.append(Point((t*arrBez[i].x +
(1-t)*arrBez[i+1].x), (t*arrBez[i].y + (1-t)*arrBez[i+1].y)))
        return bagiBezierBF(arrLerp, t)
    else:
        return Point((t*arrBez[0].x + (1-t)*arrBez[1].x),
(t*arrBez[0].y + (1-t)*arrBez[1].y))


def bezierBF(arrBez: List[Point], itrTimes: int) -> List[Point]:
    arrBezRes:List[Point] = []
    for t in range(itrTimes+1):
        arrBezRes.append(bagiBezierBF(arrBez, (1/itrTimes)*t))
    return arrBezRes
```

Fungsi utama dari source code ini adalah bezierBF, dimana ia menerima 2 argument, yaitu array dari titik awal yang hendak dihitung beziernya, dan itrTimes yang merupakan berapa kali iterasi yang diinginkan. Program ini akan mereturn array dari titik-titik yang nantinya dihubungkan menjadi kurva bezier

3.3. Main.py

Source code berikut merupakan source code yang memberikan tampilan menu kepada pengguna.

```
from DivideNConquer import *
from BruteForce import *
import matplotlib.pyplot as plt
import time

print("===BEZIER CALCULATOR===")
print("program untuk membandingkan algoritma Divide and Conquer
dengan Brute Force")

itrTimes = int(input("Masukkan jumlah iterasi yang diinginkan
(untuk Divide and Conquer): "))
while (itrTimes < 1):
    print("Mohon masukkan jumlah iterasi lebih dari atau sama
dengan 1 !")
    itrTimes = int(input("Masukkan jumlah iterasi yang diinginkan
(untuk Divide and Conquer): "))

itrTimesBF = int(input("Masukkan jumlah iterasi yang diinginkan
(untuk Brute Force): "))
while (itrTimesBF < 1):
    print("Mohon masukkan jumlah iterasi lebih dari atau sama
dengan 1 !")
    itrTimesBF = int(input("Masukkan jumlah iterasi yang diinginkan
(untuk Brute Force): "))

jumlPoint = int(input("Masukkan berapa banyak titik yang diinginkan
(n > 2) : "))
while (jumlPoint <= 2):
    print("Mohon masukkan jumlah titik di atas 2 !")
    jumlPoint = int(input("Masukkan berapa banyak titik yang
diinginkan (n > 2) : "))

arrBezIt: List[Point] = []
xOrigin: List[float] = []
yOrigin: List[float] = []
```

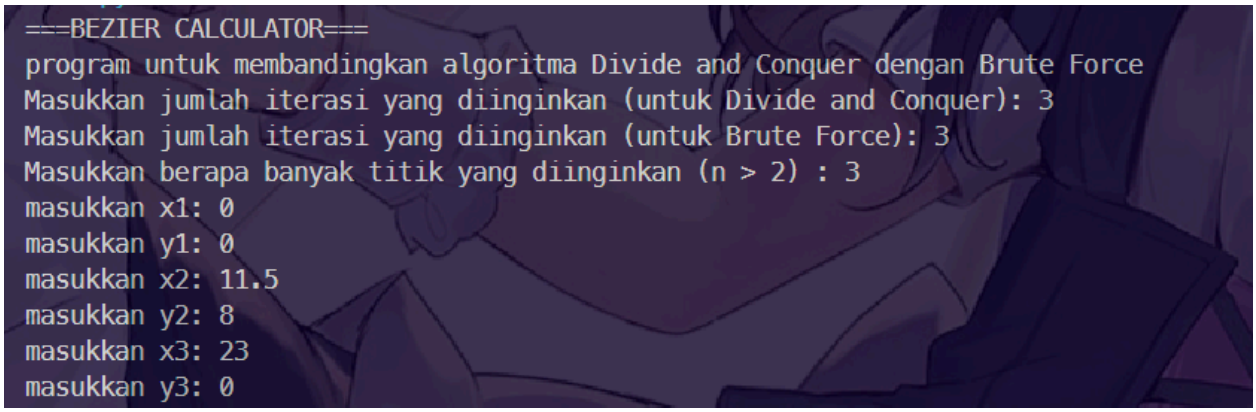
```
for i in range(jumlPoint):
    x = float(input(f"masukkan x{i+1}: "))
    y = float(input(f"masukkan y{i+1}: "))
    xOrigin.append(x)
    yOrigin.append(y)
    arrBezIt.append(Point(x, y))

startBF = time.perf_counter()
resBF : List[Point] = bezierBF(arrBezIt, itrTimesBF)
endBF = time.perf_counter()

startDNC = time.perf_counter()
resDNC : List[Point] = [arrBezIt[0]]
resDNC = resDNC + bezierDNC(arrBezIt, 0, itrTimes)
resDNC.append(arrBezIt[len(arrBezIt)-1])
endDNC = time.perf_counter()

xBF: List[float] = []; yBF: List[float] = []
for i in resBF:
    xBF.append(i.x)
    yBF.append(i.y)

xDNC: List[float] = []; yDNC: List[float] = []
for i in resDNC:
    xDNC.append(i.x)
    yDNC.append(i.y)

fig, axes = plt.subplots(1, 3)

axes[0].plot(xOrigin, yOrigin, marker='o', markersize=5)
axes[0].plot(xBF, yBF, marker='o', markersize=1)
axes[0].plot(xDNC, yDNC, marker='o', markersize=1)

axes[1].plot(xOrigin, yOrigin, marker='o', markersize=5)
axes[1].plot(xDNC, yDNC, marker='o', markersize=1)

axes[2].plot(xOrigin, yOrigin, marker='o', markersize=5)
axes[2].plot(xBF, yBF, marker='o', markersize=1)

axes[0].legend(["Titik Aseli", "Algoritma Brute Force", "Algoritma
Divide n Conquer"])
axes[1].legend(["Titik Aseli", "Algoritma Divide n Conquer"])
axes[2].legend(["Titik Aseli", "Algoritma Brute Force"])

print(f"Waktu yang dibutuhkan untuk menghitung kurva bezier dengan
algoritma Divide and Conquer: {endDNC-startDNC} sekon")
print(f"Waktu yang dibutuhkan untuk menghitung kurva bezier dengan
algoritma Brute Force: {endBF-startBF} sekon")

plt.show()
```

Source code di atas merupakan tampilan utama program. Pengguna akan menginput argumen-argumennya di sini. Program ini juga mengimplementasikan penggambaran kurva dengan matplotlib untuk memperlihatkan hasil kurva bezier sesuai dengan permintaan pengguna.

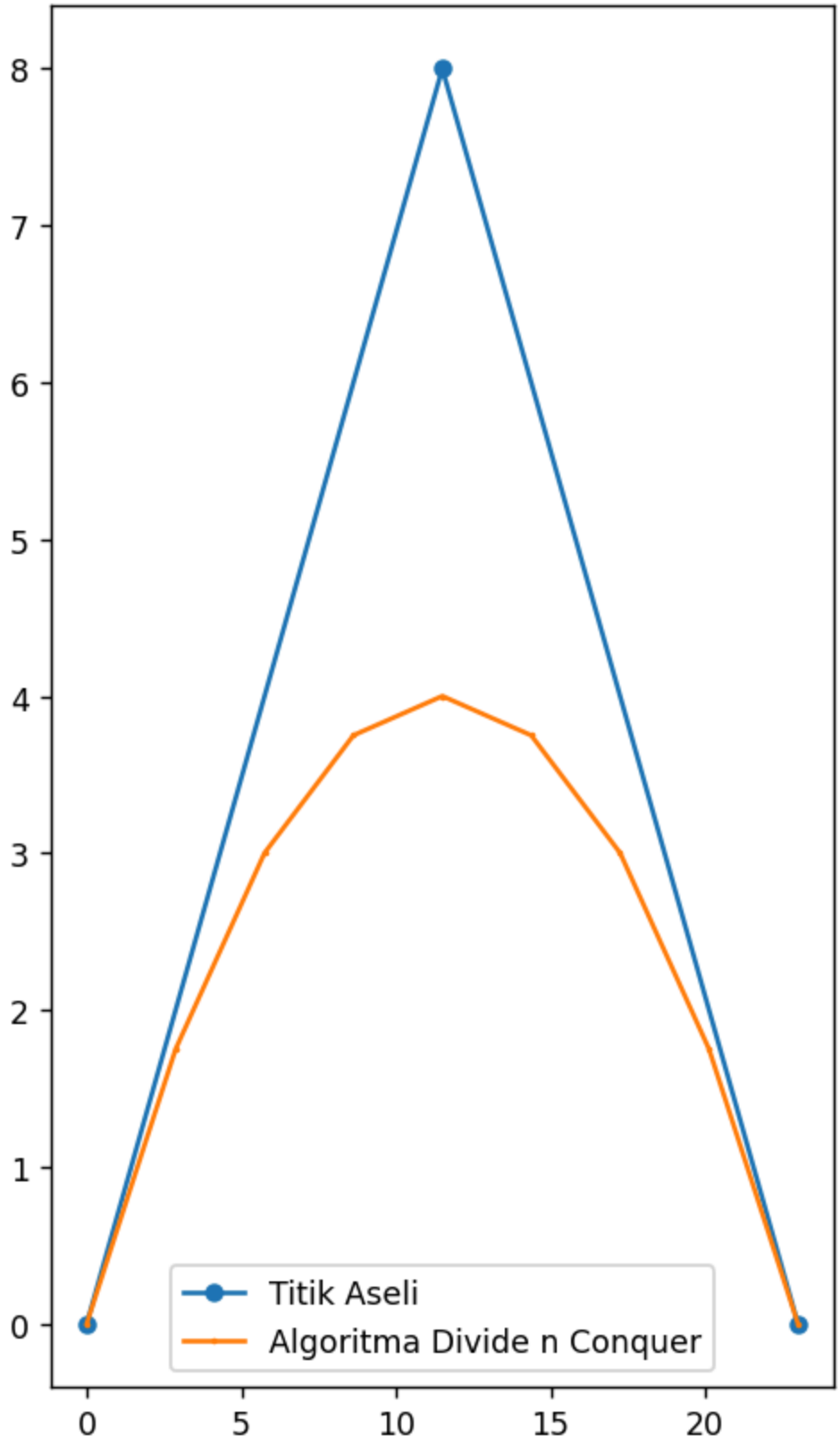
Bab 4 : Uji Coba

4.1. Test Case 1 – Input dengan 3 garis

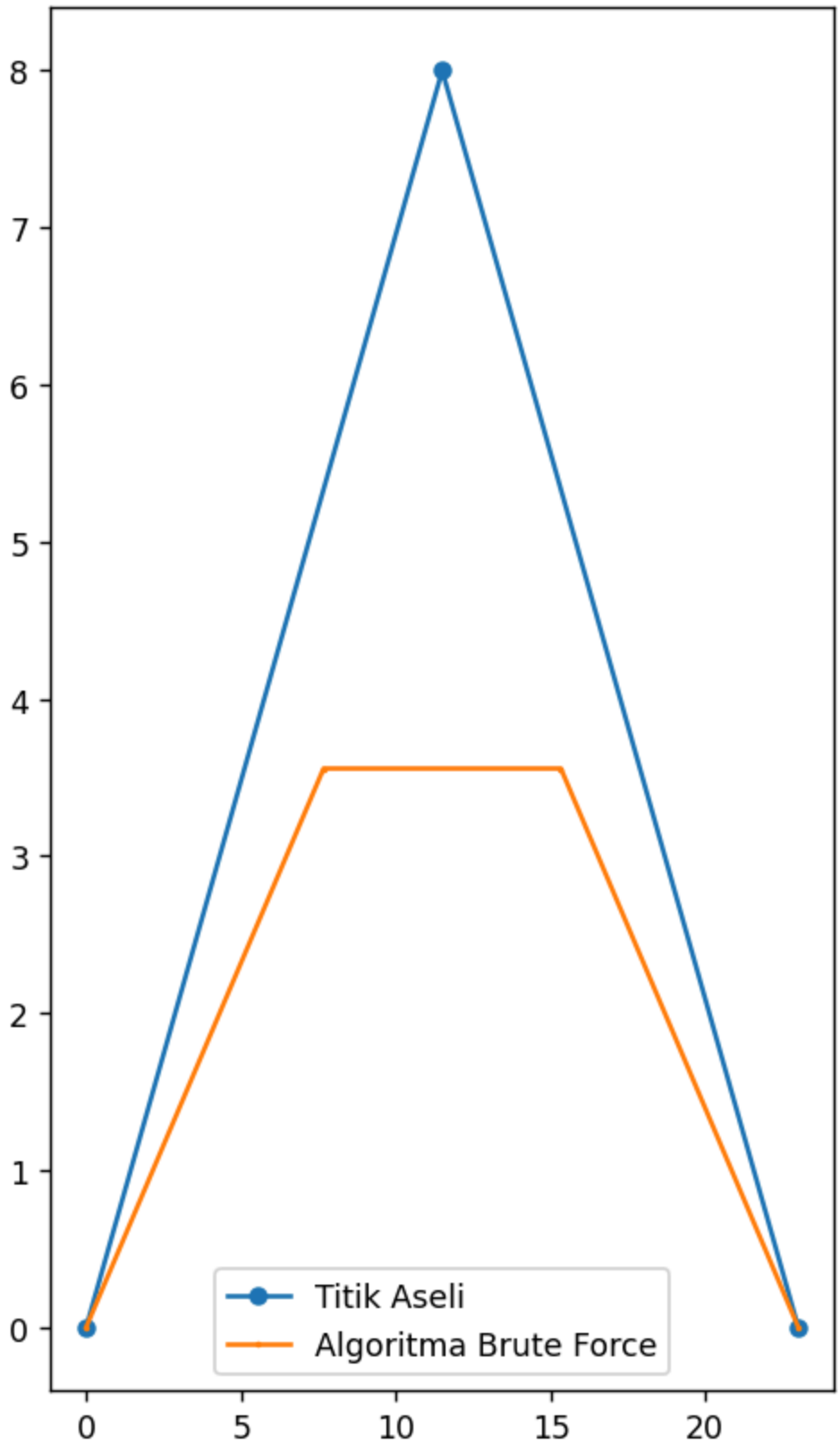
Dengan input berupa



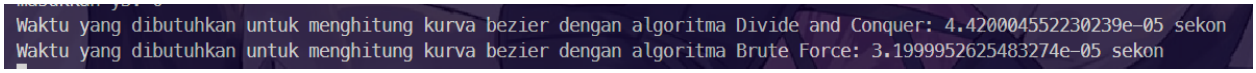
Didapat solusi Divide and Conquer berupa



Dan solusi algoritma brute force berupa

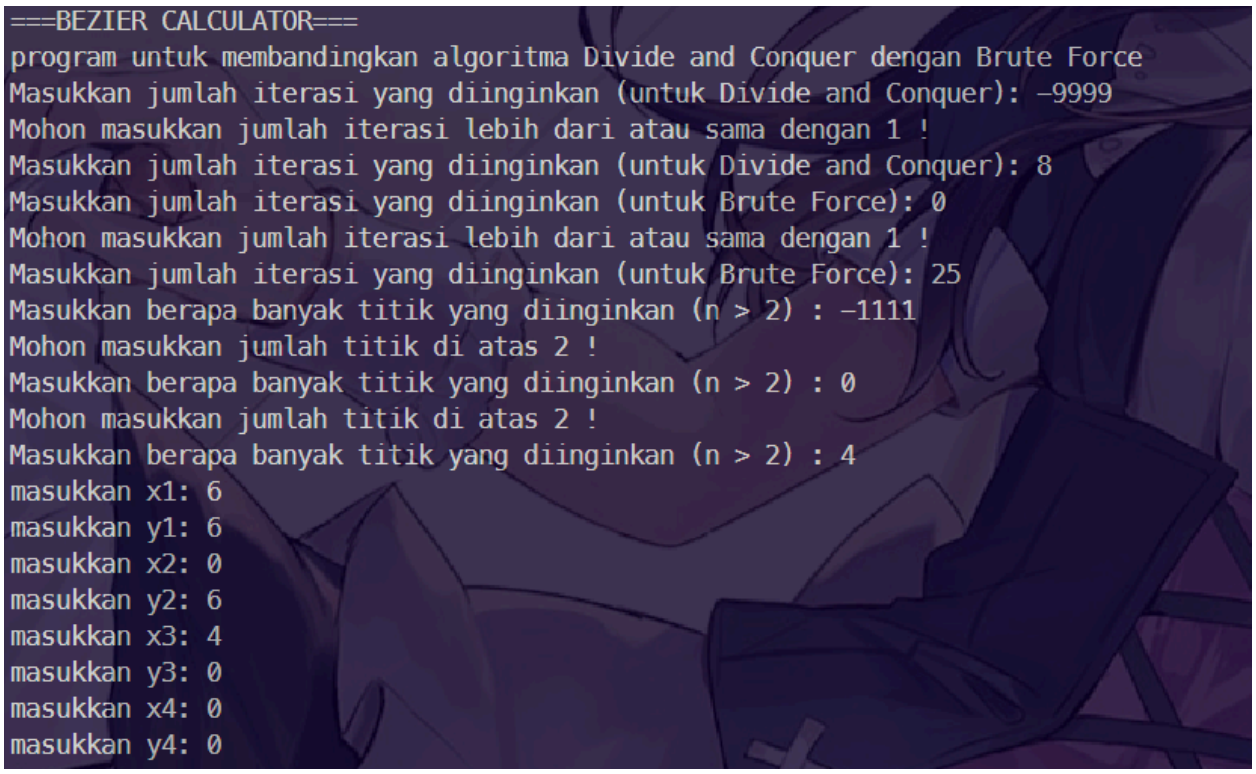

8
7
6
5
4
3
2
1
0
0
5
10
15
20
Titik Aseli
Algoritma Brute Force

Dan didapat waktu eksekusi sebagai berikut

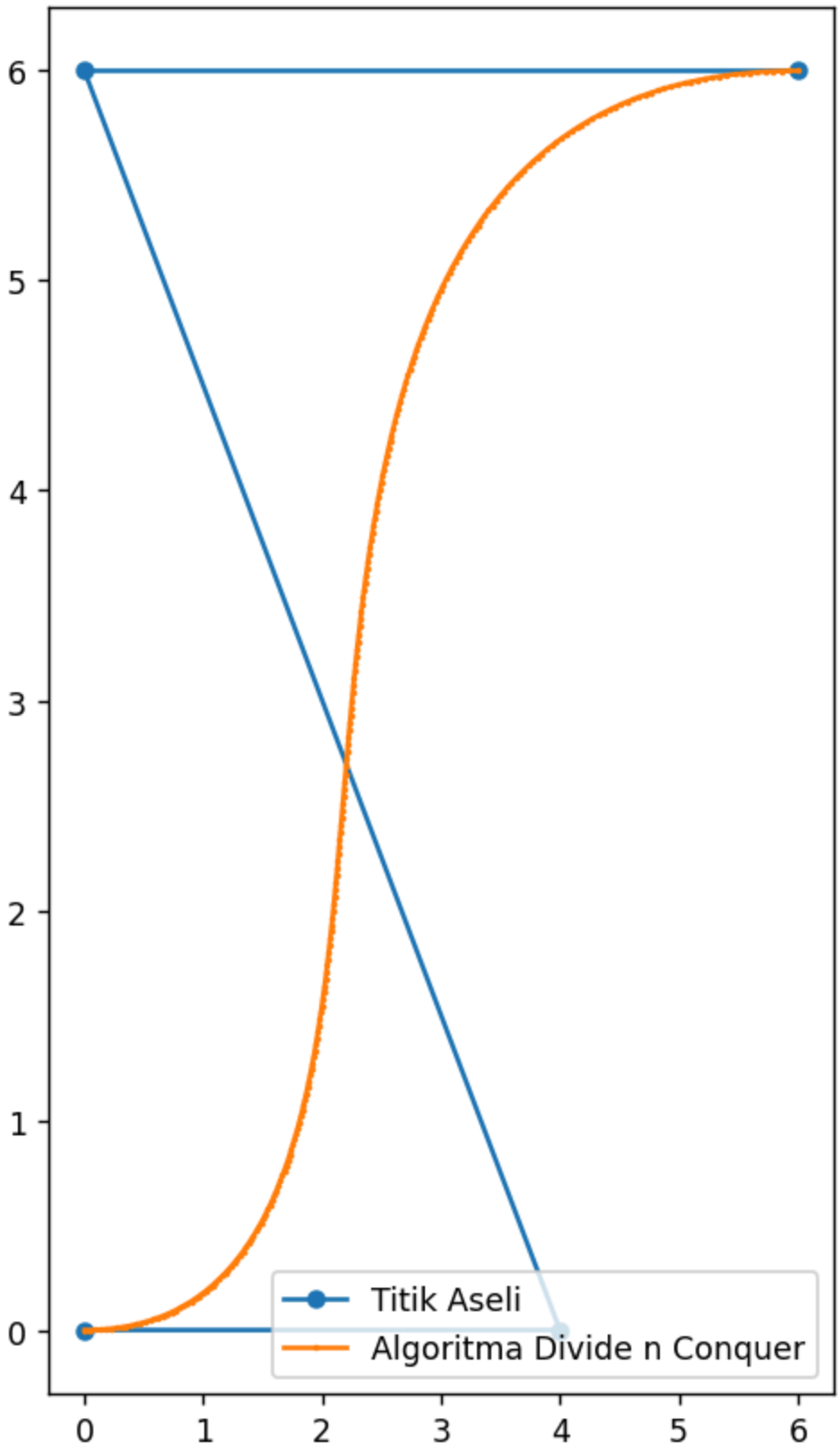


4.2. Test Case 2 – Input dengan 4 garis

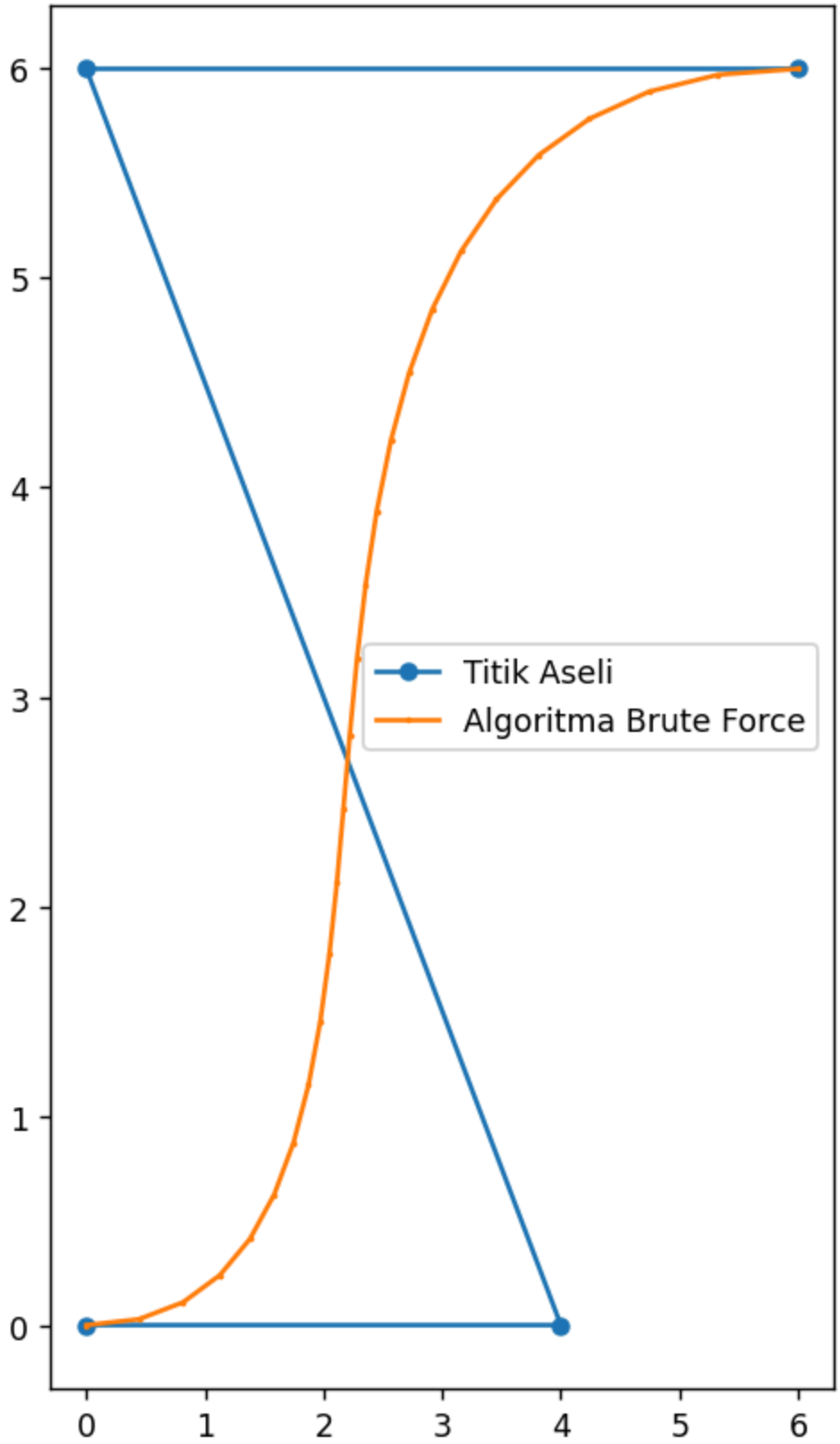
Dengan input berupa 4 garis, lalu di dalamnya diselingi dengan contoh error input yang sudah dihandling



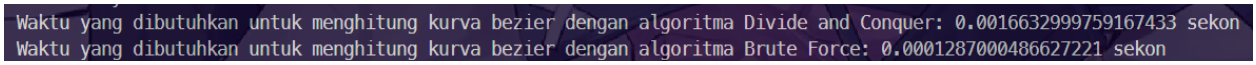
Didapat solusi Divide and conquer sebagai berikut


6
5
4
3
2
1
0
0
1
2
3
4
5
6
Titik Aseli
Algoritma Divide n Conquer

Dan solusi algoritma brute force sebagai berikut

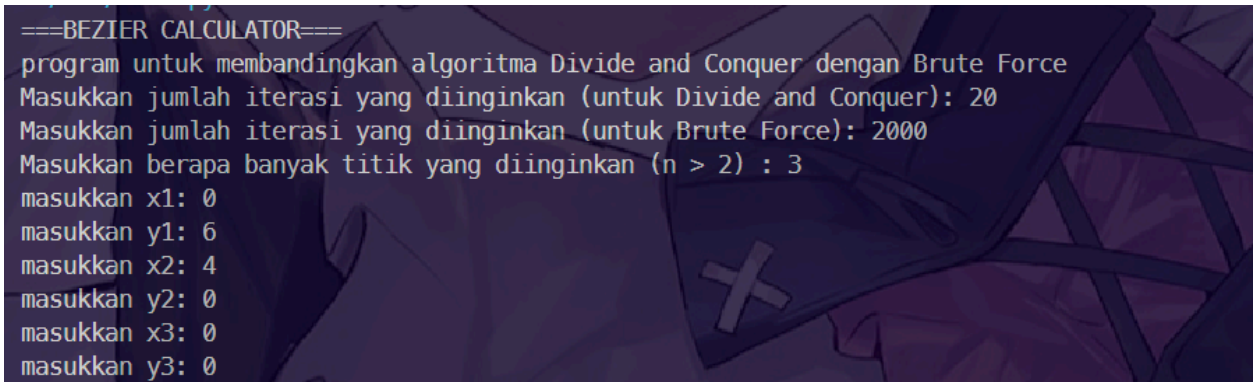

Titik Aseli
Algoritma Brute Force
0
1
2
3
4
5
6
0
1
2
3
4
5
6

Dan waktu eksekusi sebagai berikut

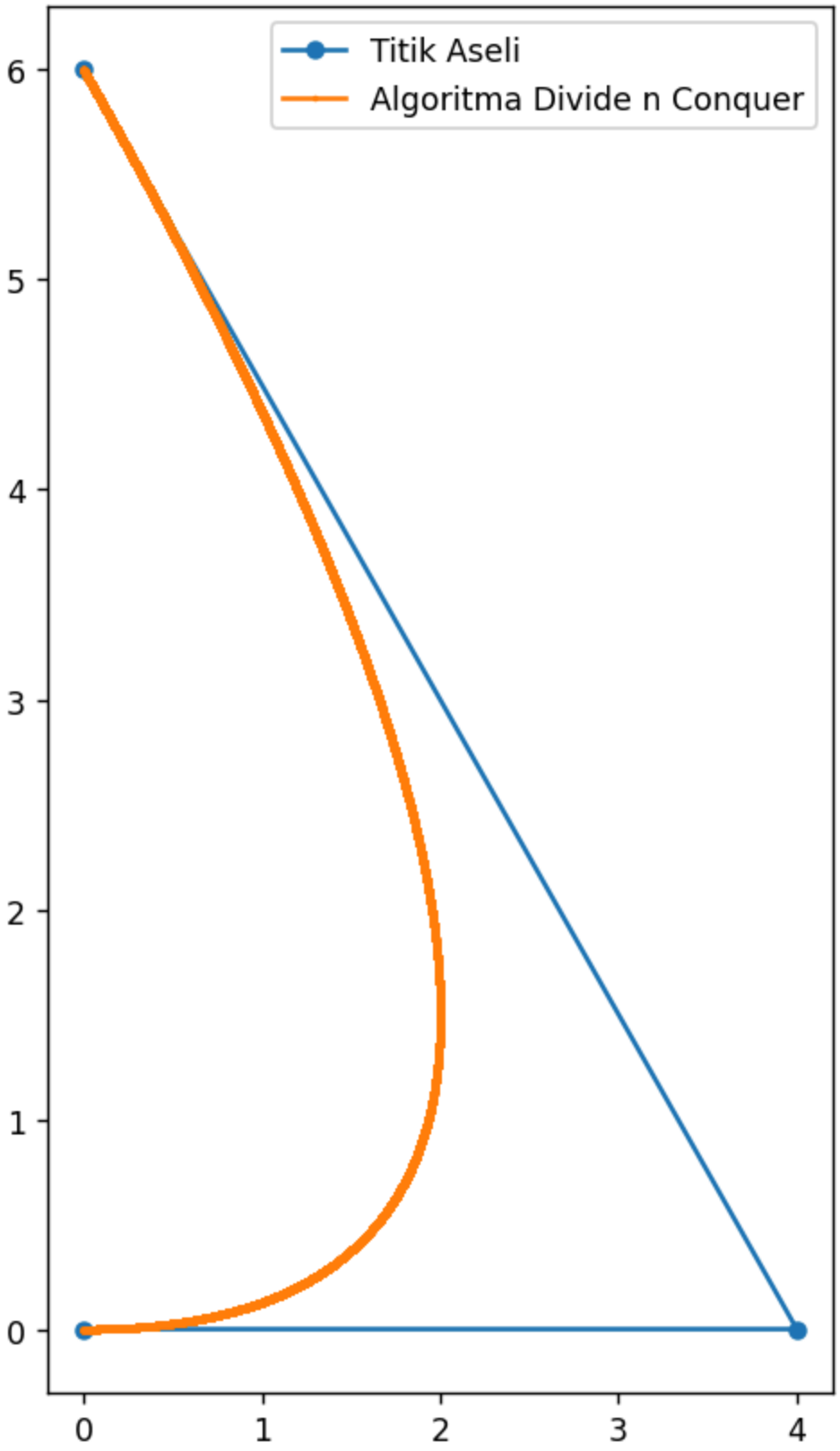


4.3. Test Case 3 – Input dengan 3 garis dan iterasi banyak

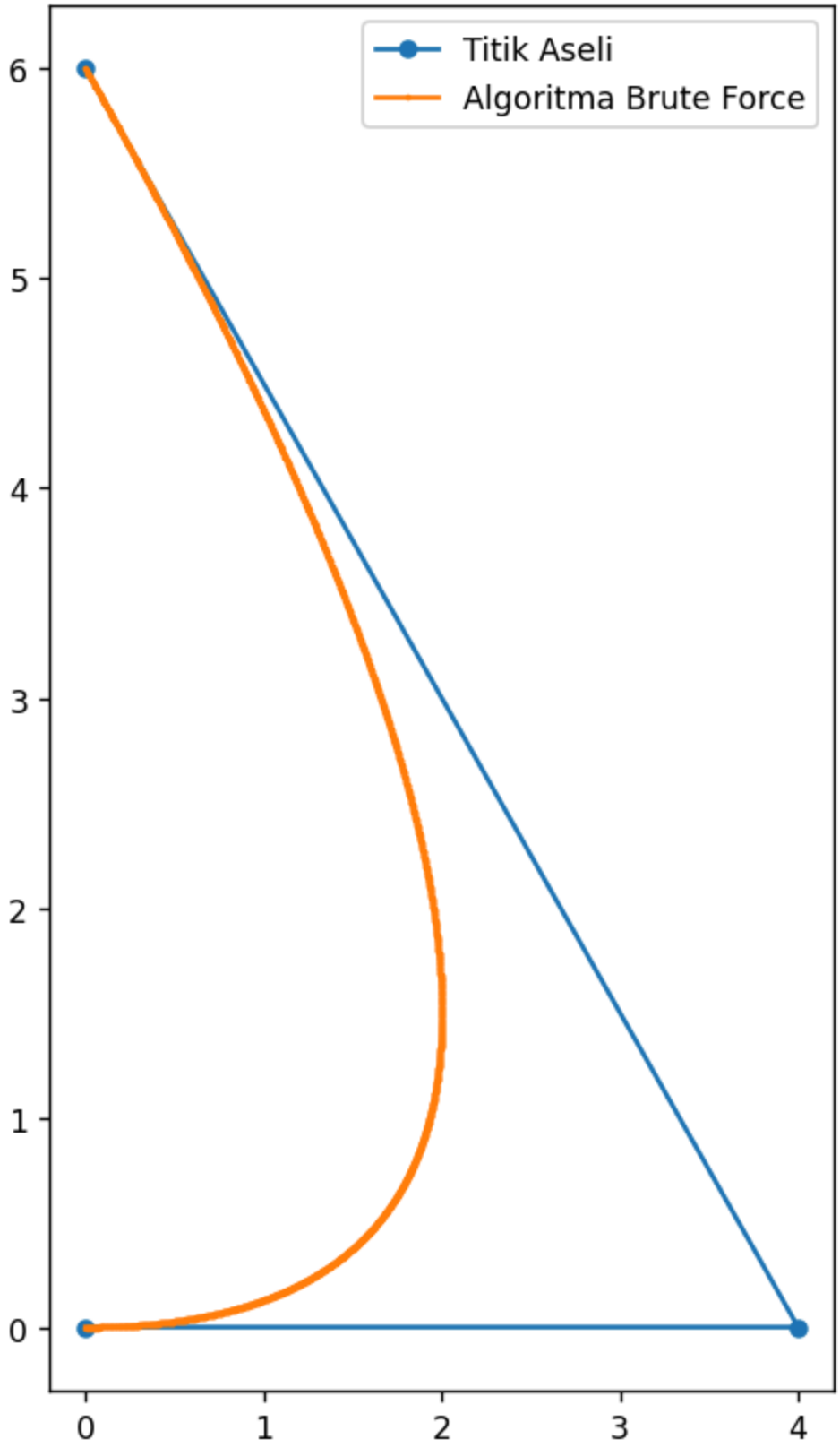
Dengan input berupa



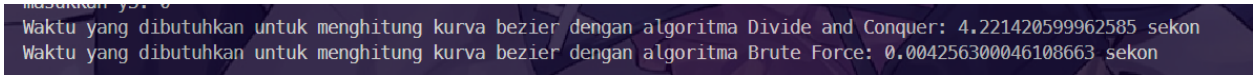
Didapat solusi dari Algoritma Divide and Conquer


Titik Aseli
Algoritma Divide n Conquer
6
5
4
3
2
1
0
0
1
2
3
4

Dan juga hasil algoritma brute force berupa

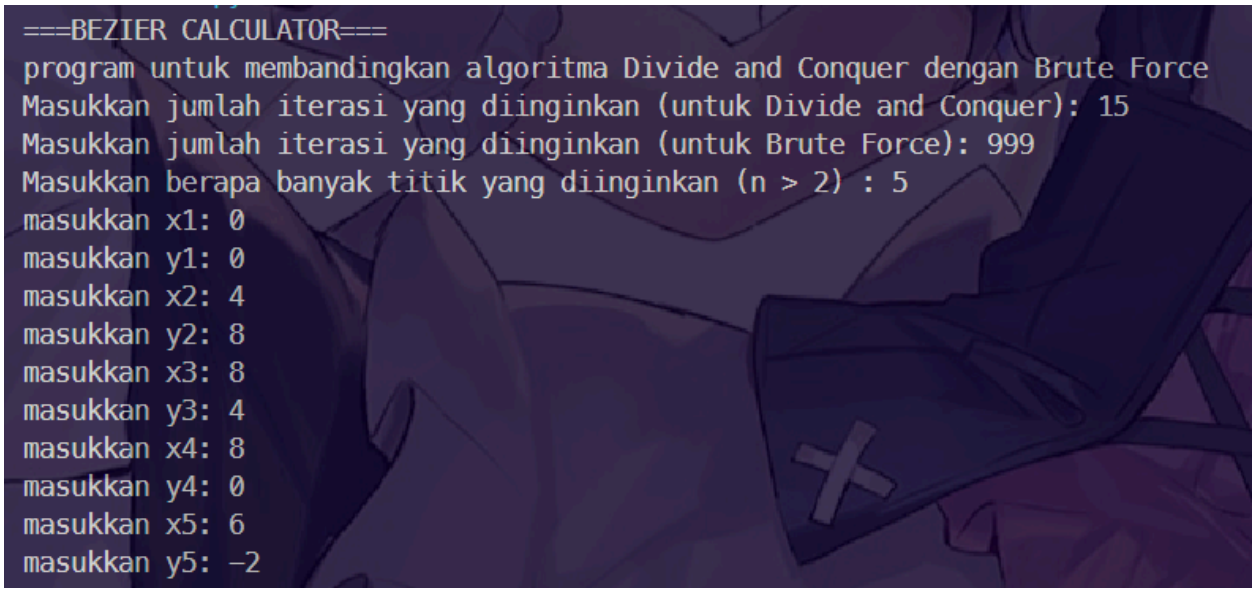

Titik Aseli
Algoritma Brute Force
6
5
4
3
2
1
0
0
1
2
3
4

Dengan waktu eksekusi

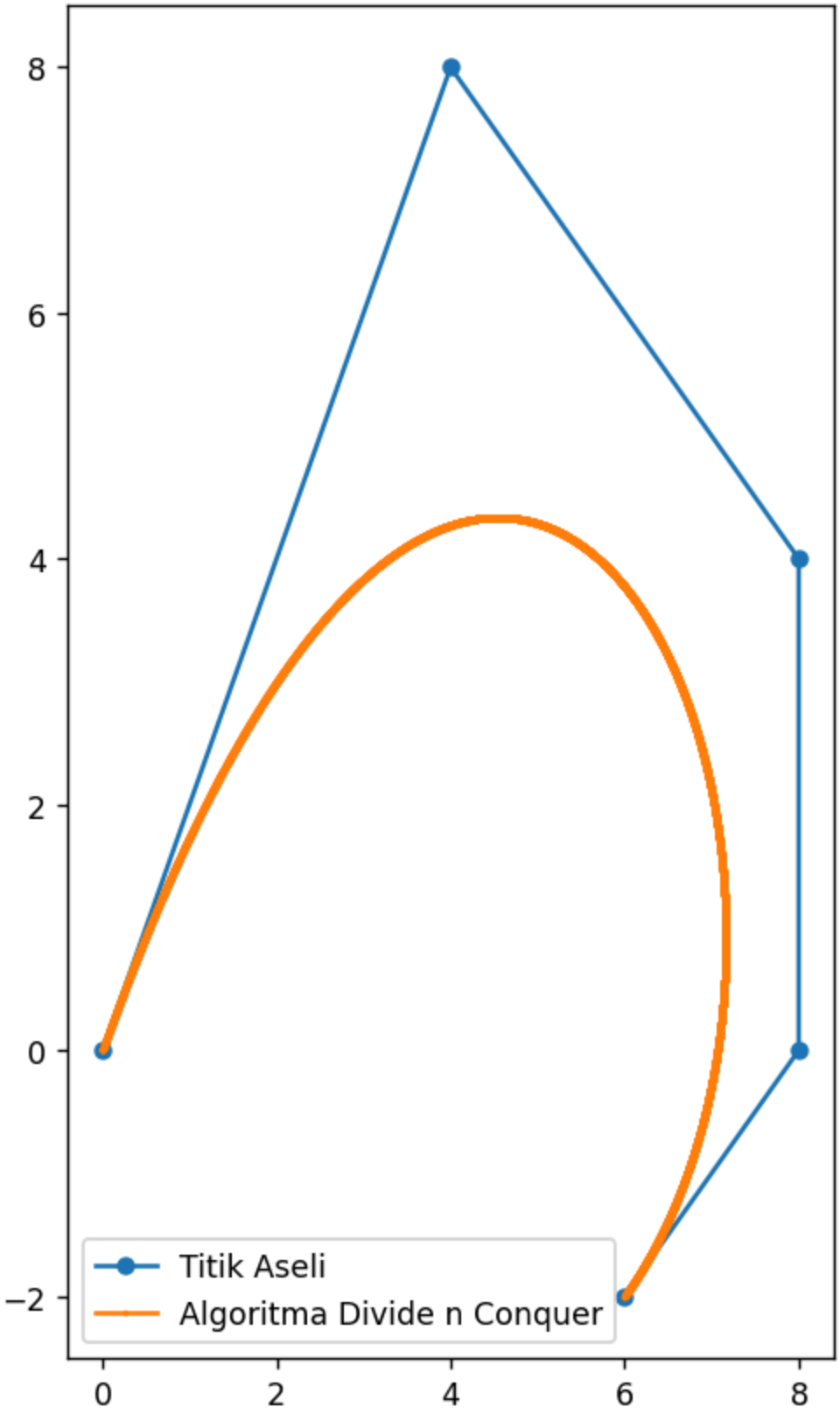


4.4. Test Case 4 – Input dengan 5 titik

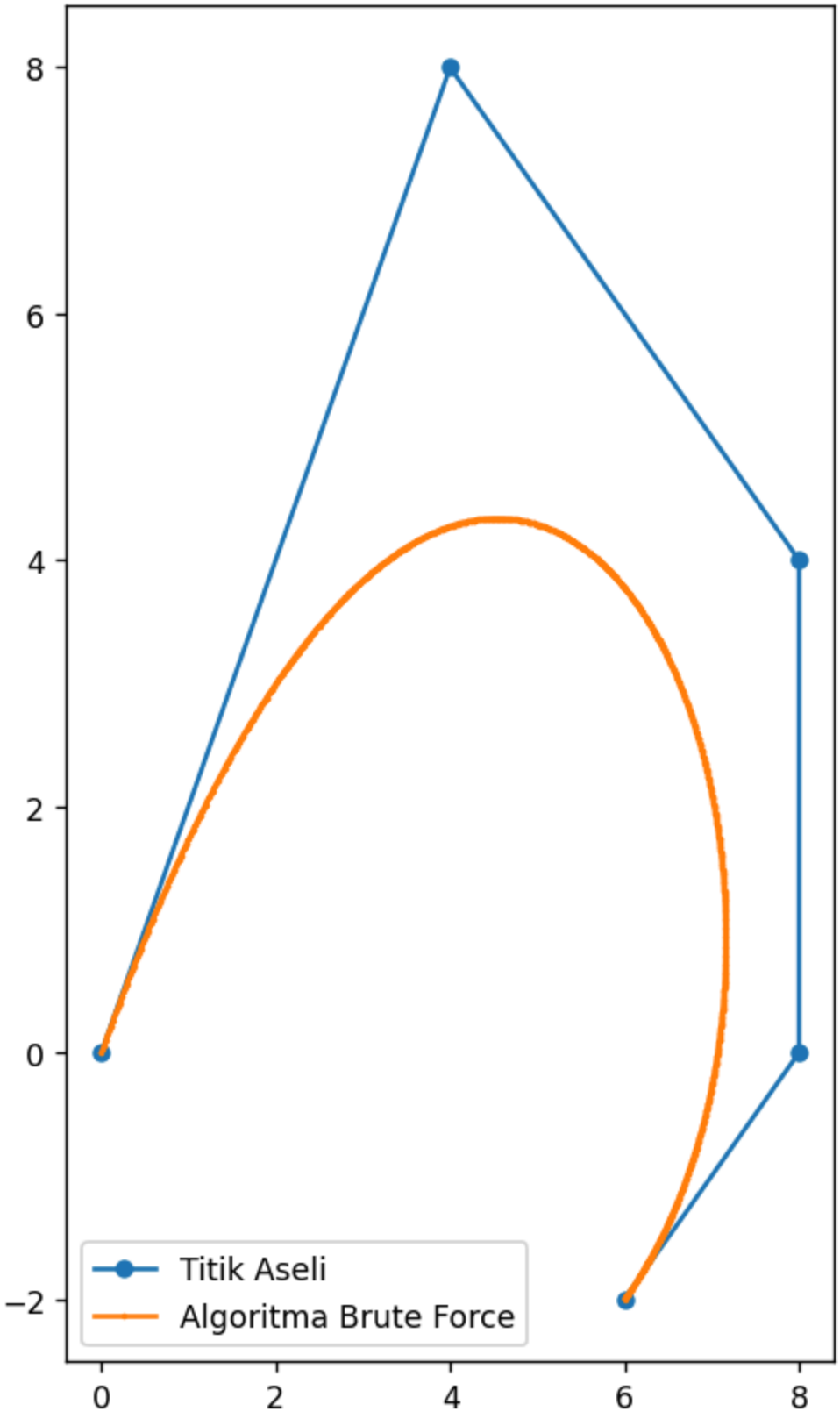
Dengan input berupa



Didapat hasil dari algoritma Divide and Conquer


8
6
4
2
0
−2
0
2
4
6
8
Titik Aseli
Algoritma Divide n Conquer

Dan juga hasil dari algoritma Brute Force


8
6
4
2
0
−2
0
2
4
6
8
Titik Aseli
Algoritma Brute Force

Dengan waktu eksekusi

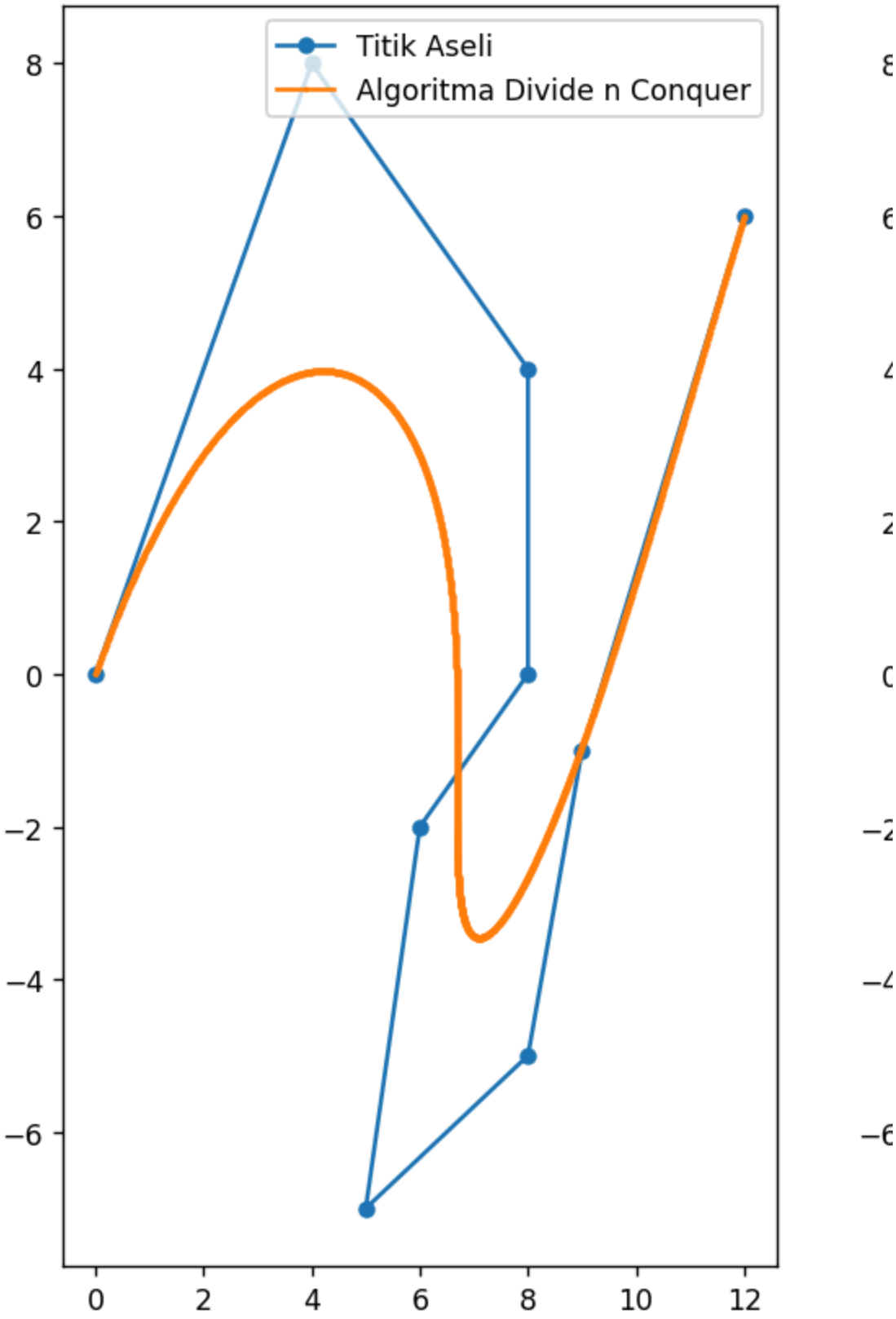
```
masukkan y5: -2
Waktu yang dibutuhkan untuk menghitung kurva bezier dengan algoritma Divide and Conquer: 0.19071689998963848 sekon
Waktu yang dibutuhkan untuk menghitung kurva bezier dengan algoritma Brute Force: 0.005689100013114512 sekon
```

4.5. Test Case 5 – Input dengan 9 titik

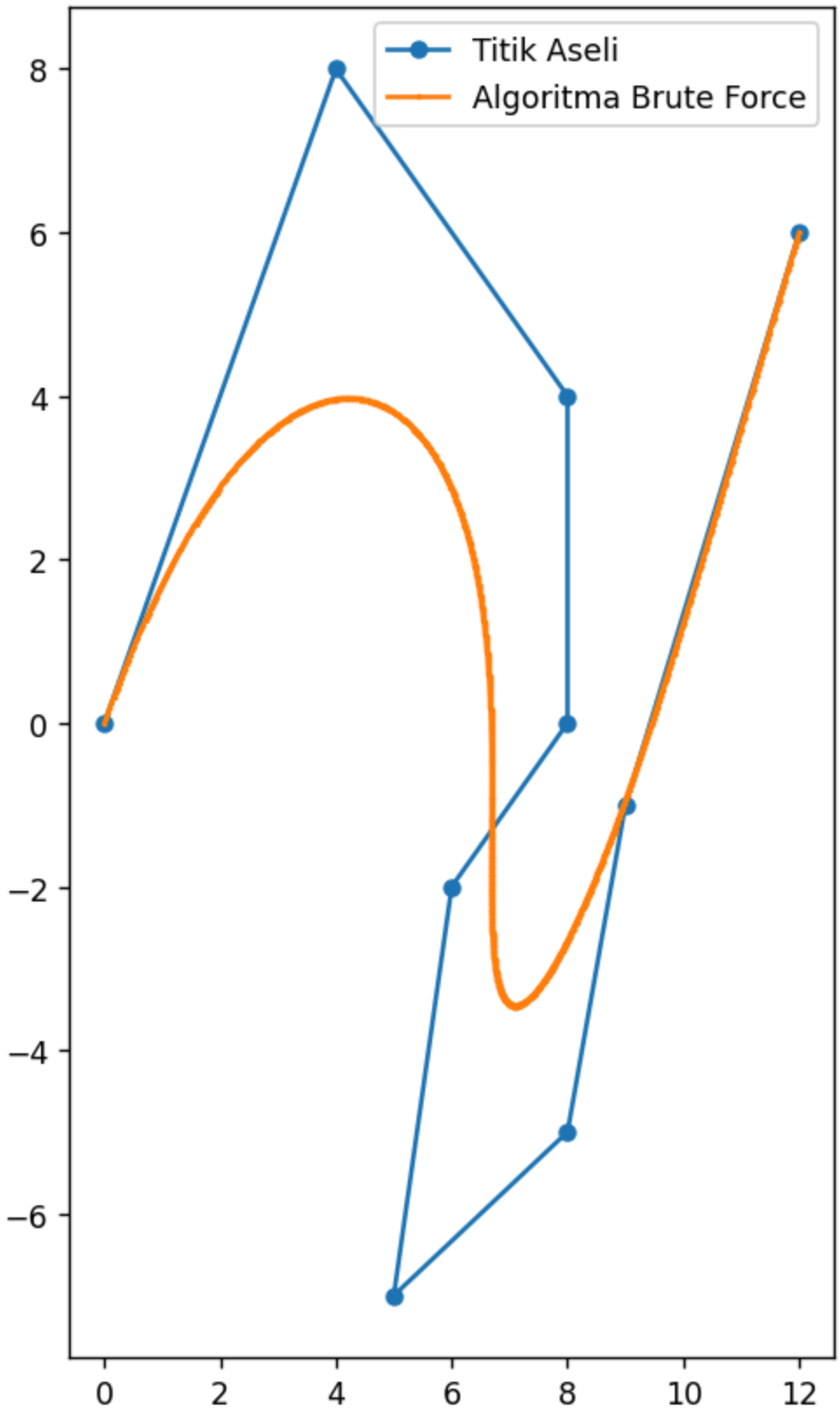
Berikut input yang dimasukkan

```
===BEZIER CALCULATOR===
program untuk membandingkan algoritma Divide and Conquer dengan Brute Force
Masukkan jumlah iterasi yang diinginkan (untuk Divide and Conquer): 12
Masukkan jumlah iterasi yang diinginkan (untuk Brute Force): 999
Masukkan berapa banyak titik yang diinginkan (n > 2) : 9
masukkan x1: 0
masukkan y1: 0
masukkan x2: 4
masukkan y2: 8
masukkan x3: 8
masukkan y3: 4
masukkan x4: 8
masukkan y4: 0
masukkan x5: 6
masukkan y5: -2
masukkan x6: 5
masukkan y6: -7
masukkan x7: 8
masukkan y7: -5
masukkan x8: 9
masukkan y8: -1
masukkan x9: 12
masukkan y9: 6
```

Didapat hasil dari algoritma divide and conquer berupa


Titik Aseli
Algoritma Divide n Conquer
8
6
4
2
0
−2
−4
−6
0
2
4
6
8
10
12

Dan juga hasil dari algoritma brute force berupa


Titik Aseli
Algoritma Brute Force
8
6
4
2
0
−2
−4
−6
0
2
4
6
8
10
12

Dengan waktu eksekusi

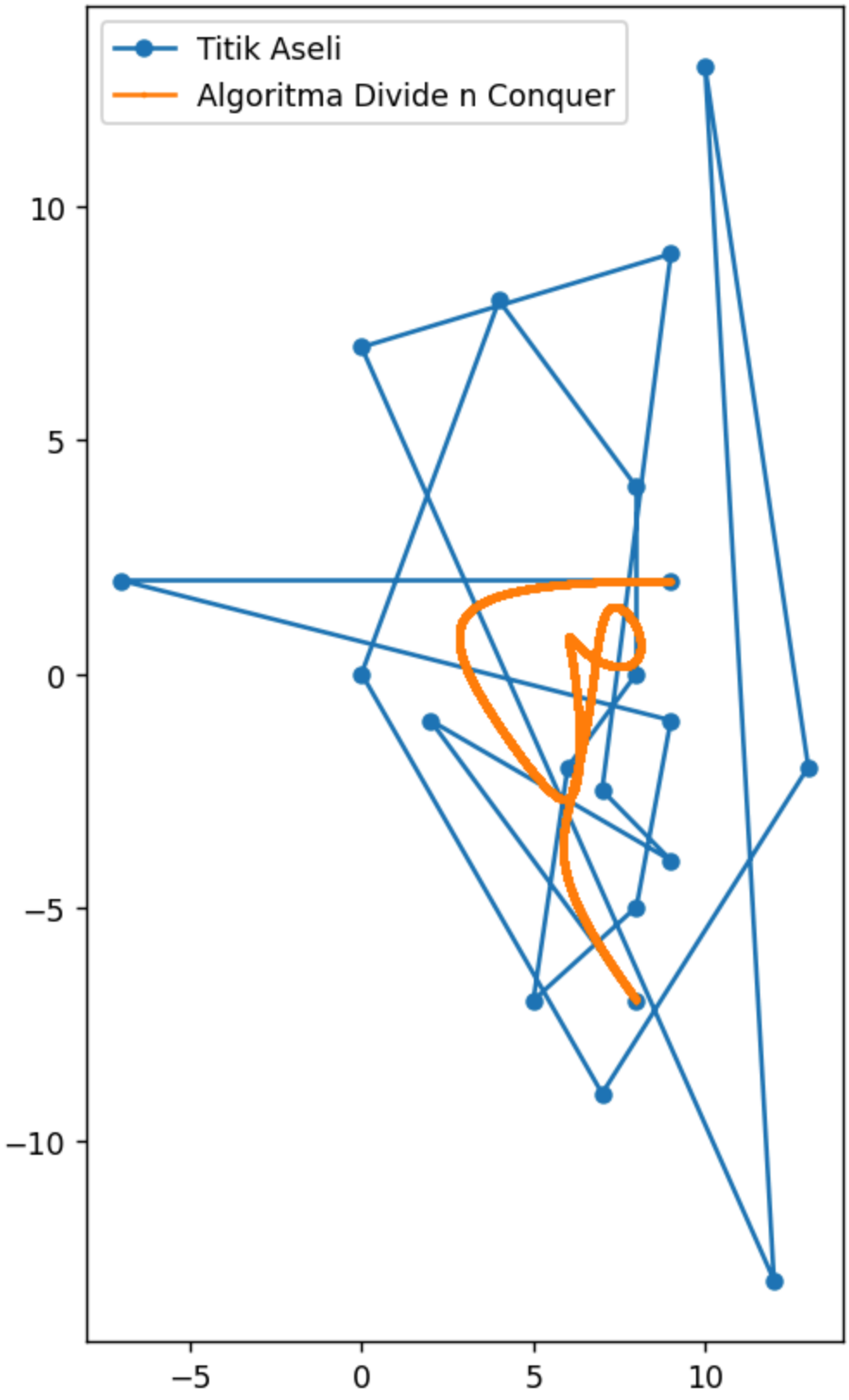
```
Waktu yang dibutuhkan untuk menghitung kurva bezier dengan algoritma Divide and Conquer: 0.07256100000813603 sekon
Waktu yang dibutuhkan untuk menghitung kurva bezier dengan algoritma Brute Force: 0.01808900001924485 sekon
```

4.6. Test Case 6 – Input dengan 20 titik

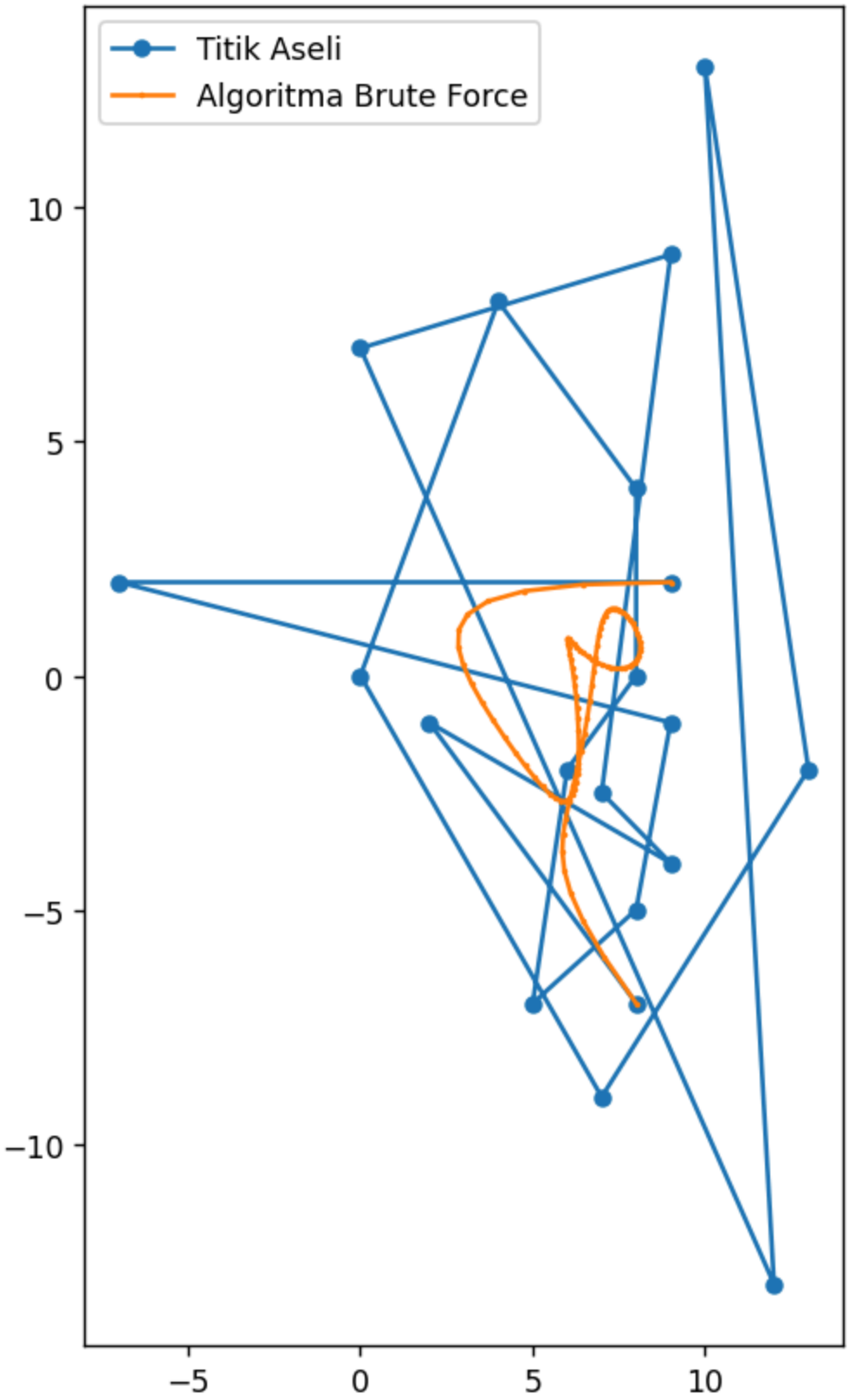
Berikut input dimasukkan

```
program untuk membandingkan algoritma Divide and Conquer dengan Brute Force
Masukkan jumlah iterasi yang diinginkan (untuk Divide and Conquer): 18
Masukkan jumlah iterasi yang diinginkan (untuk Brute Force): 100
Masukkan berapa banyak titik yang diinginkan (n > 2) : 20
masukkan x1: 8
masukkan y1: -7
masukkan x2: 2
masukkan y2: -1
masukkan x3: 9
masukkan y3: -4
masukkan x4: 7
masukkan y4: -2.5
masukkan x5: 9
masukkan y5: 9
masukkan x6: 0
masukkan y6: 7
masukkan x7: 12
masukkan y7: -13
masukkan x8: 10
masukkan y8: 13
masukkan x9: 13
masukkan y9: -2
masukkan x10: 7
masukkan y10: -9
masukkan x11: 0
masukkan y11: 0
masukkan x12: 4
masukkan y12: 8
masukkan x13: 8
masukkan y13: 4
masukkan x14: 8
masukkan y14: 0
masukkan x15: 6
masukkan y15: -2
masukkan x16: 5
masukkan y16: -7
masukkan x17: 8
masukkan y17: -5
```

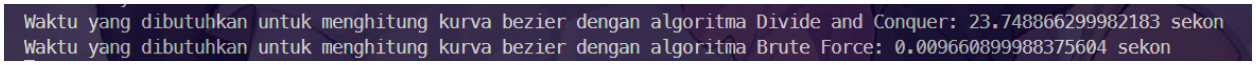
didapat hasil dari algoritma Divide and conquer


Titik Aseli
Algoritma Divide n Conquer
10
5
0
−5
−10
−5
0
5
10

Dan juga hasil algoritma brute force


Titik Aseli
Algoritma Brute Force
10
5
0
−5
−10
−5
0
5
10

Dan waktu eksekusi



Bab 5 : Analisis Hasil Uji Coba

Dari 6 percobaan dari masing-masing algoritma, didapat bahwa Hasil dari kedua Algoritma sangat mirip, bahkan hampir sama. Yang membedakan diantara keduanya yaitu jumlah iterasi yang dibutuhkan, dimana rata-rata Algoritma Divide and Conquer membutuhkan lebih dari 10 iterasi sampai 20 iterasi untuk membuat kurva yang mulus dalam waktu yang singkat. Sementara untuk Algoritma Brute Force, ia membutuhkan iterasi di atas 100 untuk membuat kurva yang mulus, dan ia tetap berjalan secara singkat walaupun jumlah iterasinya mencapai ribuan bahkan puluh ribuan. Hal ini disebabkan karena perbedaan kompleksitas waktunya, dimana kompleksitas Algoritma Divide and Conquer di kasus ini, untuk fungsi bezierDNC, ia memakan O(2^n) karena ia memanggil rekursif setiap kali fungsi sebanyak 2, dan setiap kali fungsi dipanggil, juga memanggil fungsi rekursif yang memanggil menuju basis, yaitu O(n) dan juga setiap fungsi ini dipanggil, ia mengiterasi arraynya, menjadi O(n) lagi, semuanya dikali, jadilah kompleksitas total dari Algoritma Divide and Conquer kasus ini berupa O(2^n * n^2).

Beda dengan Algoritma brute force, yang di fungsi utamanya ia hanya mengiterasi sebanyak iterasi yang diinginkan, menjadi O(n), dan setiap iterasi tersebut, memanggil fungsi rekursif yang menuju basis, O(n), dan setiap fungsi juga mengiterasi arraynya, menjadi O(n) lagi. Maka total dari kompleksitas total dari Algoritma Brute Force kasus ini berupa O(n^3). Dapat disimpulkan bahwa di sini kompleksitas waktu Algoritma Brute Force jauh lebih baik daripada kompleksitas waktu Algoritma Divide and Conquer

Lampiran

Link Repository

https://github.com/NameLessAth/Tucil2_13522044

Sheets Poin

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.		✓