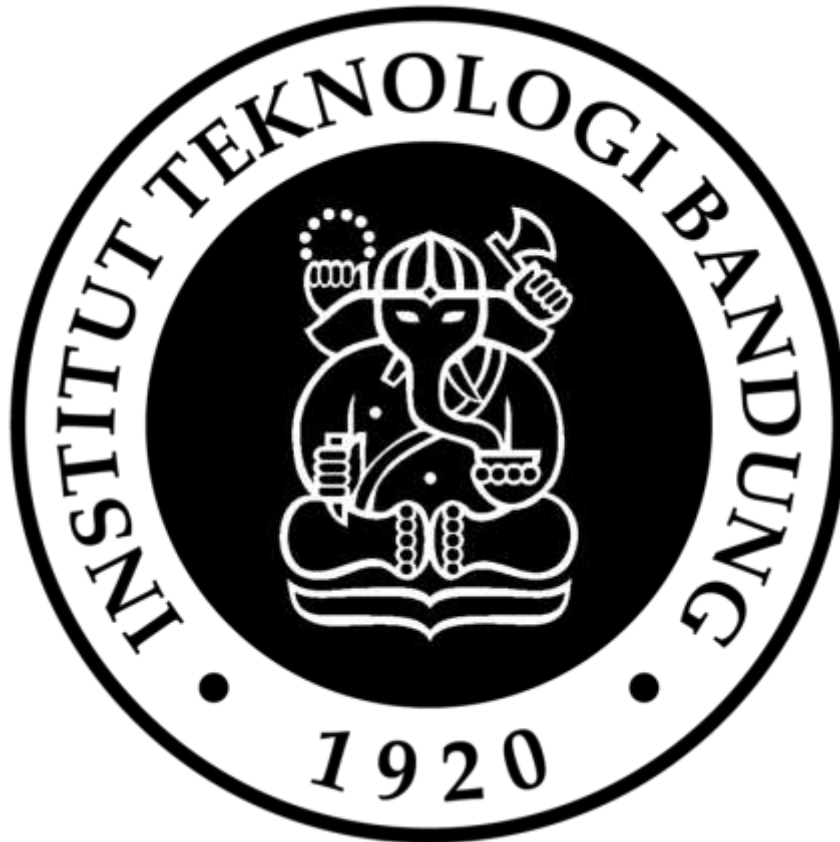


Laporan Tugas Kecil 2

IF2211 Strategi Algoritma

Aplikasi Divide and Conquer Dalam Pembentukan Kurva Bezier



Disusun oleh:

Akbar Al Fattah 13522036

Devinzen 13522064

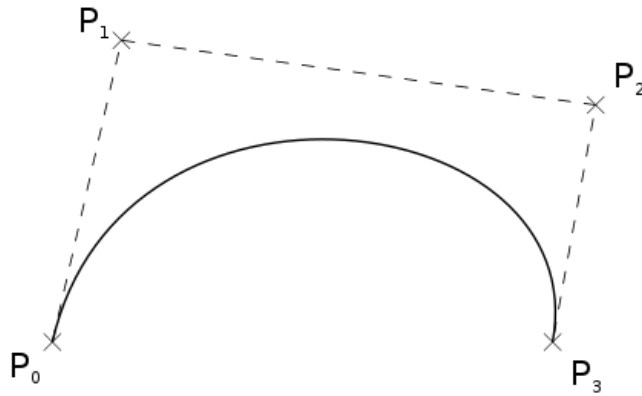
PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

Bab 1: Deskripsi Masalah



Gambar 1. Kurva Bézier Kubik

(Sumber: https://id.wikipedia.org/wiki/Kurva_B%C3%A9zier)

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.

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

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.

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

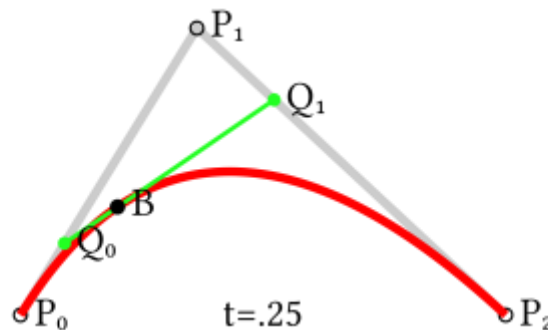
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \quad t \in [0, 1]$$

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

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

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

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.

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

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

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 akan diimplementasikan **pembuatan kurva Bézier** dengan algoritma titik tengah berbasis **divide and conquer**.

Bab 2: Algoritma Pembentukan Kurva Bezier

A. Algoritma Brute Force

1. Tentukan persamaan kurva bezier dengan parameter t.

Untuk kuadratik, persamaan kurva beziernya adalah

$$R_0 = B(t) = (1-t)^2 P_0 + (1-t)t P_1 + t^2 P_2, \quad t \in [0, 1]$$

Untuk n buah control point, persamaan kurva beziernya adalah

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

Dengan P_i adalah titik ke-i, nC_i adalah kombinasi (n,i) , dan n adalah banyak control point.

2. Tentukan p buah titik yang akan dicari berdasarkan persamaan kurva bezier di atas..
3. Hitung nilai titik ke-i dengan cara menginput t dari 0 sampai 1 dengan pertambahan nilai t sebesar 1/p sebanyak p kali dengan p adalah banyak titik yang dibuat.

B. Algoritma Divide and Conquer

Untuk Kurva Bezier Kuadratik

1. Awalnya dimulai dari parameter P0,P1,P2,dan iterasi
2. Tentukan midpoint kiri (Midpoint dari P0 dan P1)
3. Tentukan midpoint kanan (Midpoint dari P1 dan P2)
4. Tentukan midpoint dari kedua midpoint di atas sebelumnya.
5. Masukkan midpoint di nomor 1 dan nomor 2 ke list midpoint
6. Ulangi proses ini lagi namun parameternya adalah P0, Midpoint di nomor 1, Midpoint di nomor 3, dan kurangi iterasi dengan 1 (divide and conquer ke kiri)
7. Masukkan midpoint dari nomor 3 menjadi titik di kurva bezier
8. Ulangi proses ini lagi namun parameternya adalah Midpoint di nomor 3, Midpoint di nomor 2, P2, dan kurangi iterasi dengan 1 (divide and conquer ke kanan)
9. Ulangi divide and conquer sampai iterasi mencapai 0

Untuk Kurva Bezier dengan n buah control point

1. Awalnya dimulai dari parameter ControlPointsList, iterasi, dan MidpointList
 2. Bagi list menjadi dua, yaitu ListKiri dan ListKanan
 3. Lakukan iterasi mundur dari derajat kurva bezier dengan indeks i
 - a. Jika i = 1
 - i. Jika derajat kurva bezier + 1 adalah 2, maka buat midpoint baru dari P0 dan P1 (control point)
 - ii. Jika bukan, maka buat midpoint baru dari midpoint ke-(m-2) dan midpoint ke-(m-1) dengan m adalah panjang (keterangan: midpoint ke-n didapat dari MidpointList)
- Lalu tambahkan midpoint baru tersebut ke ListKiri dari belakang dan ke ListKanan dari depan.

- b. Jika $i = \text{derajat kurva bezier}$
 - Lakukan iterasi dengan indeks j dari 0 ke derajat kurva bezier-1
 - Buat midpoint baru dari control point ke j dan control point ke $j+1$
 - 1. Jika $j = 0$, tambahkan midpoint baru tersebut ke ListKiri dari belakang
 - 2. Jika $j = \text{derajat kurva bezier} - 1$, tambahkan midpoint baru tersebut ke ListKanan dari depan
 - Tambahkan midpoint baru tersebut ke MidpointList
 - c. Jika i bernilai selain kedua nilai di atas
 - Lakukan iterasi dengan indeks j dari 0 ke $i-1$
 - Buat midpoint baru dari midpoint ke $(m-i-1)$ ke midpoint ke $(m-i)$ dengan m adalah panjang dari MidpointList
 - 1. Jika $j = 0$, tambahkan midpoint baru tersebut ke ListKiri dari belakang
 - 2. Jika $j = i-1$, tambahkan midpoint baru tersebut ke ListKanan dari depan
 - Tambahkan midpoint baru tersebut ke MidpointList
- 4. Bagi permasalahan menjadi dua yaitu ke kiri dan ke kanan
 - 5. Panggil fungsi ini lagi dengan parameter ListKiri, iterasi-1, dan MidpointList
 - 6. Tambahkan midpoint dari bagian 3a ke MidpointList
 - 7. Panggil fungsi ini lagi dengan parameter ListKanan, iterasi-1, dan MidpointList
 - 8. Ulangi divide and conquer sampai nilai iterasi-1 ≤ 0

Bab 3: Source Code BezierFormulas.py

Screenshot source code

- Algoritma Brute Force (Kurva Bezier Kuadratik dan n buah control point)

```
1 from point import Point
2 def LinearBezier(ControlPoint0: Point, ControlPoint1: Point, t: int) -> Point: #Persamaan kurva Bezier Linear
3     Pt = Point((ControlPoint0.x + t * (ControlPoint1.x - ControlPoint0.x)), (ControlPoint0.y + t * (ControlPoint1.y - ControlPoint0.y)), "")
4     return Pt
5
6 def QuadraticBezier(ControlPoint0: Point, ControlPoint1: Point, ControlPoint2: Point, t: int) -> Point: #Persamaan kurva Bezier Kuadratik
7     Q1 = LinearBezier(ControlPoint0, ControlPoint1, t)
8     Q2 = LinearBezier(ControlPoint1, ControlPoint2, t)
9     R = LinearBezier(Q1, Q2, t)
10    return R
11
12 # algo brute force (khusus kurva bezier kuadratik)
13 def BruteForceQuadraticBezier(ControlPoint0: Point, ControlPoint1: Point, ControlPoint2: Point, iteration: int) -> list[Point]:
14     # termasuk titik ujungnya
15     return [QuadraticBezier(ControlPoint0, ControlPoint1, ControlPoint2, n/iteration) for n in range(iteration + 1)]
16
17 # kalau jadi ke-juin bonus satu
18 def Combination(n: int, r: int) -> int:
19     C = 1
20     for i in range(n, r, -1): C *= i
21     for i in range(2, (n - r + 1)): C //= i
22     return C
23
24 # untuk brute force
25 def GeneralBezierFormula(ControlPointsList: list[Point], t: float) -> Point: # sesuai dengan panjang list ControlPointsList
26     order = len(ControlPointsList) - 1 #derajat kurva bezier
27     X = Point(0, 0, "NW")
28     for i in range(order + 1):
29         #hitung nilai polinomial Bernstein untuk x dan y
30         X.x += Combination(order, i) * ((1 - t) ** (order - i)) * (t ** i) * ControlPointsList[i].x
31         X.y += Combination(order, i) * ((1 - t) ** (order - i)) * (t ** i) * ControlPointsList[i].y
32     return X
33
34 def GeneralBruteForceBezier(ControlPointsList, iteration) -> list[Point]: #algoritma pembentukan kurva bezier yang sudah digeneralisasi secara
35     return [GeneralBezierFormula(ControlPointsList, n/iteration) for n in range(iteration+1)] #kembalikan array yang berisi titik-titik di kur
```

- Algoritma Divide And Conquer (untuk Kurva Bezier Kuadratik)

```
36 # Divide and conquer (hanya untuk kurva bezier kuadratik)
37 def DivideAndConquerQuadraticBezier(ControlPoint0: Point, ControlPoint1: Point, ControlPoint2: Point, iteration: int, MidpointArray):
38     if(iteration > 0): #selama iterasinya belum mencapai 0
39         Midpoint1 = Midpoint(ControlPoint0, ControlPoint1, "KIRI") #cari titik tengah di antara garis di control point 0 dan control point 1
40         Midpoint2 = Midpoint(ControlPoint1, ControlPoint2, "KANAN") #cari titik tengah di antara garis di control point 1 dan control point 2
41         ExtraPoint = Midpoint(Midpoint1, Midpoint2, "TENGAH") #cari titik tengah di antara garis di midpoint 1 dan midpoint 2 (titik di kurva bezier)
42         #urutannya akan selalu "KIRI", "KANAN", "TENGAH". yang akan membentuk garis di visualisasi hanyalah titik yang diberi nama "KIRI"
43         #yang akan dianggap titik di kurva bezier adalah titik yang diberi nama "TENGAH"
44         MidpointArray.append(Midpoint1)
45         MidpointArray.append(Midpoint2)
46         DivideAndConquerQuadraticBezier(ControlPoint0, Midpoint1, ExtraPoint, iteration-1, MidpointArray) #divide permasalahan ke kiri
47         MidpointArray.append(ExtraPoint)
48         DivideAndConquerQuadraticBezier(ExtraPoint, Midpoint2, ControlPoint2, iteration-1, MidpointArray) #divide permasalahan ke kanan
```

- Algoritma Divide And Conquer (untuk n buah control point)

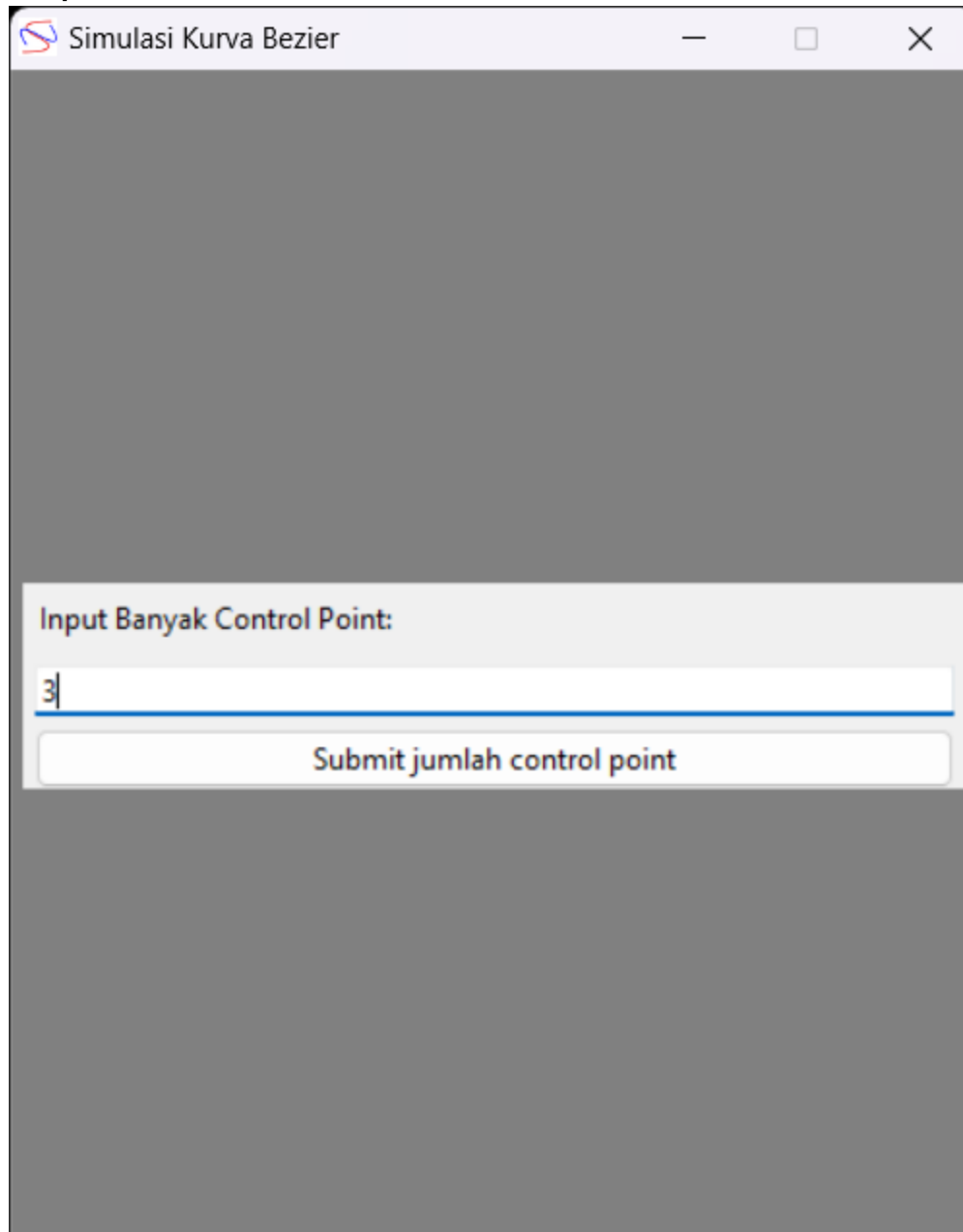
```

50 def GeneralDivideAndConquerBezier(ControlPointsList : list[Point], iteration: int, MidpointArray):
51     if (iteration > 0): #selesai iterasinya belum mencapai 0
52         order = len(ControlPointsList) - 1 #derajat kurva bezier
53         NewLeft = [ControlPointsList[0]]
54         NewRight = [ControlPointsList[order]]
55         ExtraPoint = Point(0, 0, "NEW")
56         for i in range (order, 0, -1): #iterasi mundur dari derajat
57             if (i == 1):
58                 if ((order + 1) == 2): # edge case jika banyak control point = 2
59                     ExtraPoint = Midpoint(ControlPointsList[0], ControlPointsList[1], "NEW")
60                 else:
61                     ExtraPoint = Midpoint(MidpointArray[len(MidpointArray) - 2], MidpointArray[len(MidpointArray) - 1], "NEW") #titik di kurva bezier
62                 NewLeft.append(ExtraPoint)
63                 NewRight = [ExtraPoint] + NewRight
64             elif (i == order):
65                 for j in range (1):
66                     if (j == 0):
67                         NewPoint = Midpoint(ControlPointsList[j], ControlPointsList[j + 1], "KIRI")
68                         NewLeft.append(NewPoint)
69                     elif (j == (i - 1)):
70                         NewPoint = Midpoint(ControlPointsList[j], ControlPointsList[j + 1], "KANAN")
71                         NewRight = [NewPoint] + NewRight
72                     else:
73                         NewPoint = Midpoint(ControlPointsList[j], ControlPointsList[j + 1], "TENGAH")
74                         MidpointArray.append(NewPoint)
75             else:
76                 for j in range (1):
77                     if (j == 0):
78                         NewPoint = Midpoint(MidpointArray[len(MidpointArray) - 1 - 1], MidpointArray[len(MidpointArray) - 1], "KIRI")
79                         NewLeft.append(NewPoint)
80                     elif (j == (i - 1)):
81                         NewPoint = Midpoint(MidpointArray[len(MidpointArray) - 1 - 1], MidpointArray[len(MidpointArray) - 1], "KANAN")
82                         NewRight = [NewPoint] + NewRight
83                     else:
84                         NewPoint = Midpoint(MidpointArray[len(MidpointArray) - 1 - 1], MidpointArray[len(MidpointArray) - 1], "TENGAH")
85                         MidpointArray.append(NewPoint)
86             GeneralDivideAndConquerBezier(NewLeft, iteration-1, MidpointArray) #divide permasalahan ke kiri
87             MidpointArray.append(ExtraPoint)
88             GeneralDivideAndConquerBezier(NewRight, iteration-1, MidpointArray) #divide permasalahan ke kanan
89     return

```

Bab 4: Uji Coba Program

Set point 1



Simulasi Kurva Bezier

Input Banyak Control Point:

3

Submit jumlah control point

Simulasi Kurva Bezier

Titik Kontrol P0

Nilai X:

-3

Nilai Y:

0

Titik Kontrol P1

Nilai X:

1

Nilai Y:

3

Titik Kontrol P2

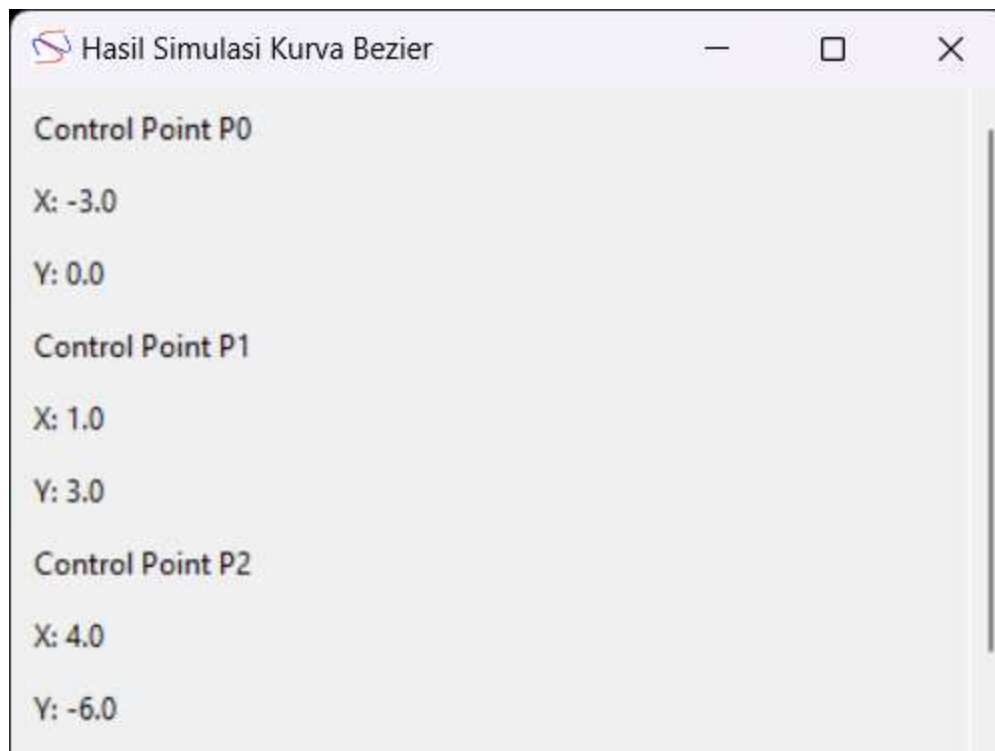
Nilai X:

4

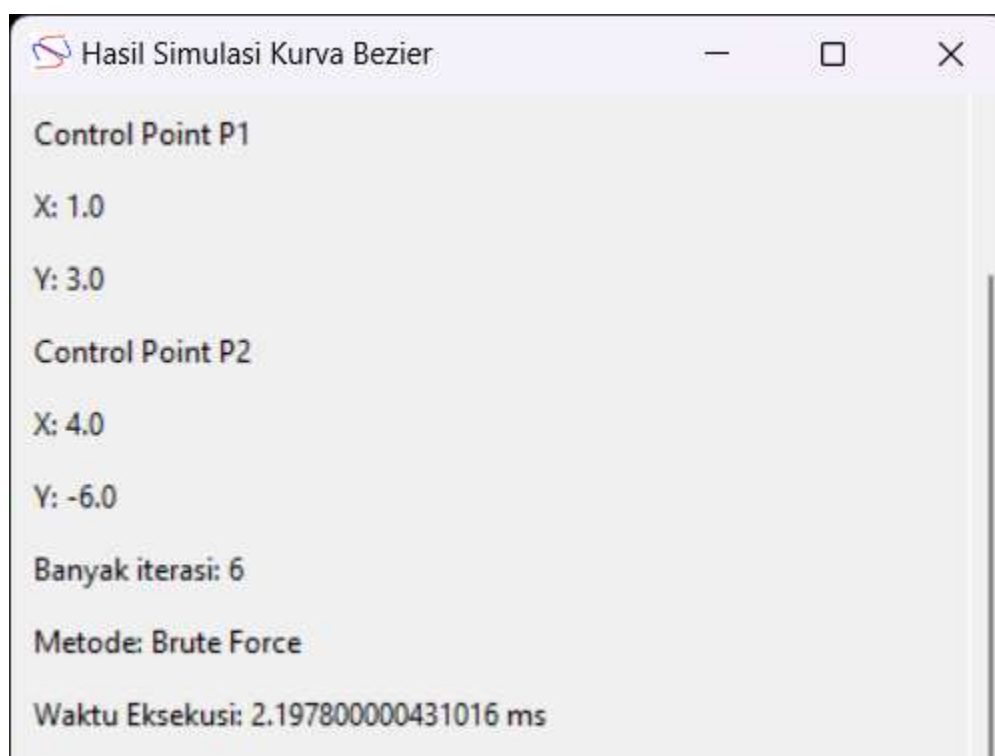
Nilai Y:

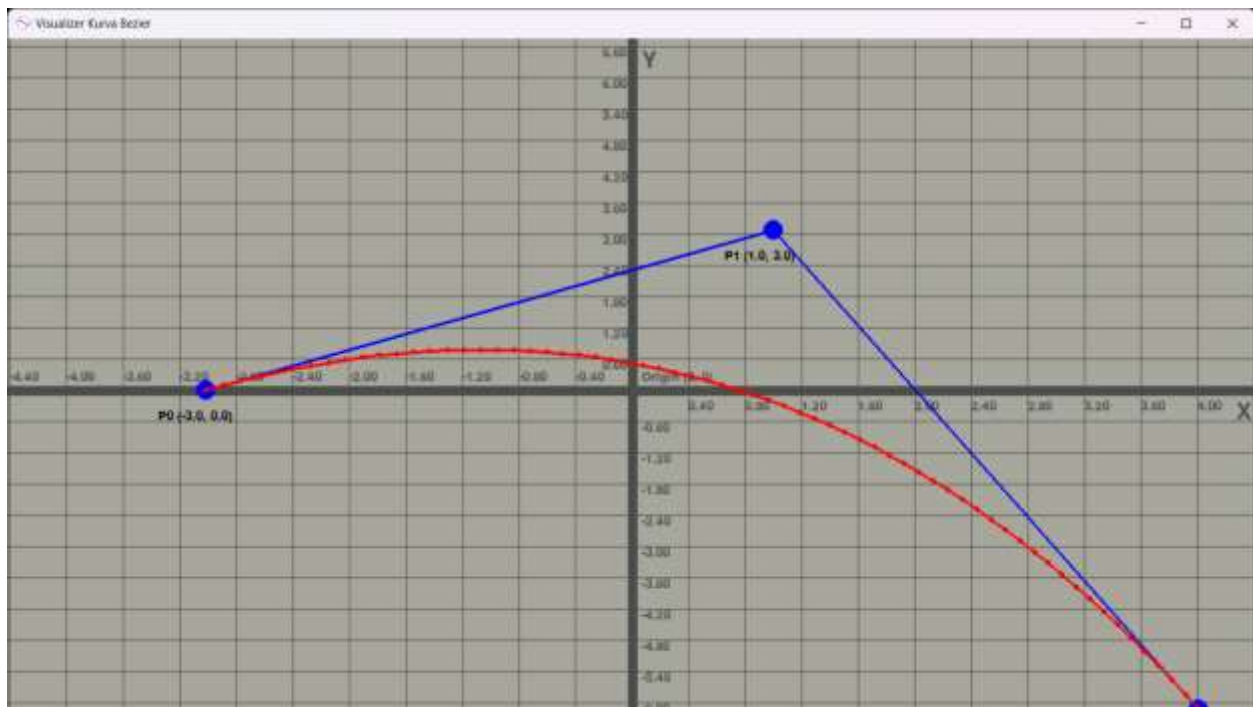
-6

Banyak Iterasi

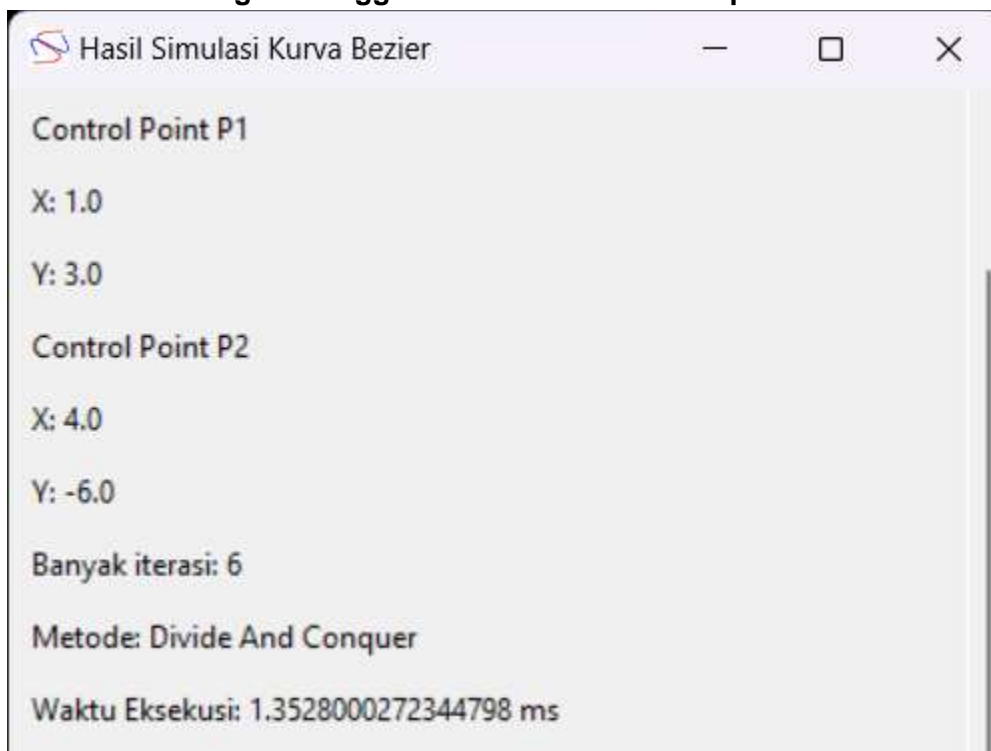


A. Hasil dengan menggunakan Metode Brute Force






B. Hasil dengan menggunakan Divide And Conquer






Set point 2

 Simulasi Kurva Bezier

Input Banyak Control Point:

 Simulasi Kurva Bezier

Titik Kontrol P0

Nilai X:

Nilai Y:

Titik Kontrol P1

Nilai X:

Nilai Y:

Titik Kontrol P2

Nilai X:

Nilai Y:

Titik Kontrol P3

Simulasi Kurva Bezier

0

Titik Kontrol P2

Nilai X:

3

Nilai Y:

0

Titik Kontrol P3

Nilai X:

0

Nilai Y:

-5

Banyak Iterasi

3

Pilih Metode Pembentukan Kurva Bezier

☒ Brute Force

☐ Divide And Conquer

Simulasi Kurva Bezier

0

Titik Kontrol P2

Nilai X:

3

Nilai Y:

0

Titik Kontrol P3

Nilai X:

0

Nilai Y:

-5

Banyak Iterasi

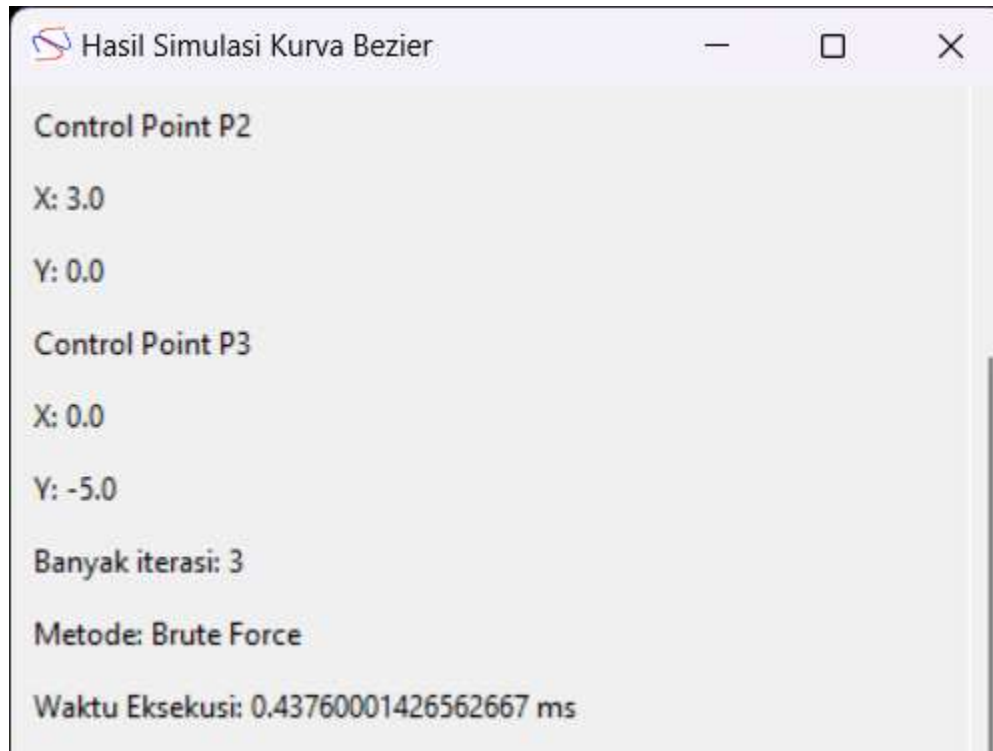
3

Pilih Metode Pembentukan Kurva Bezier

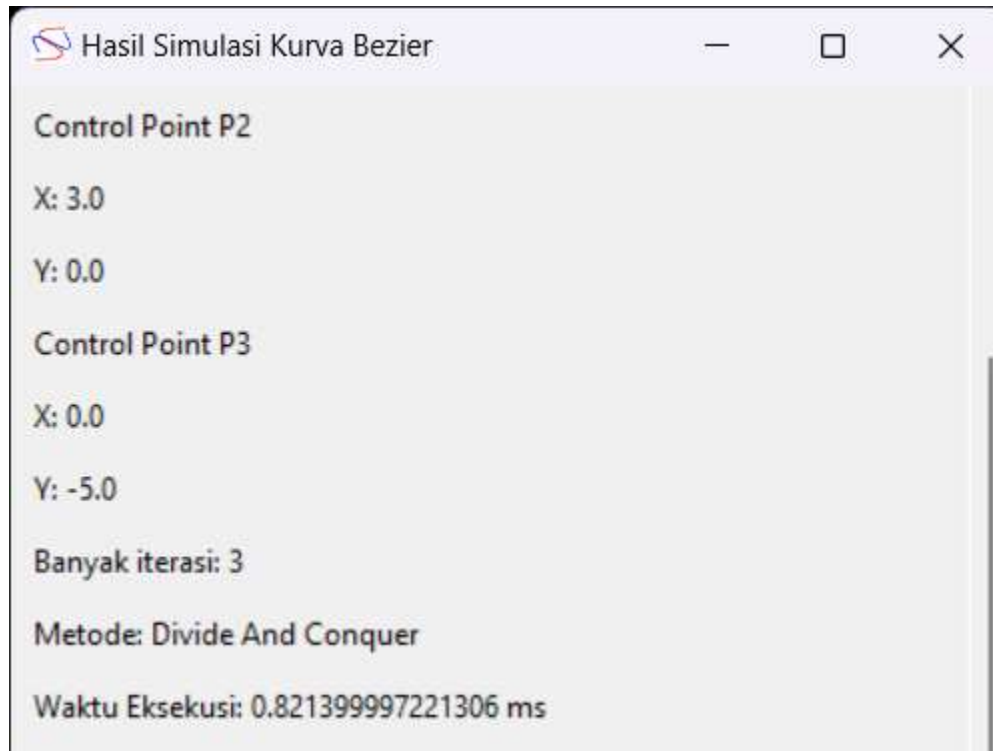
☐ Brute Force

☒ Divide And Conquer

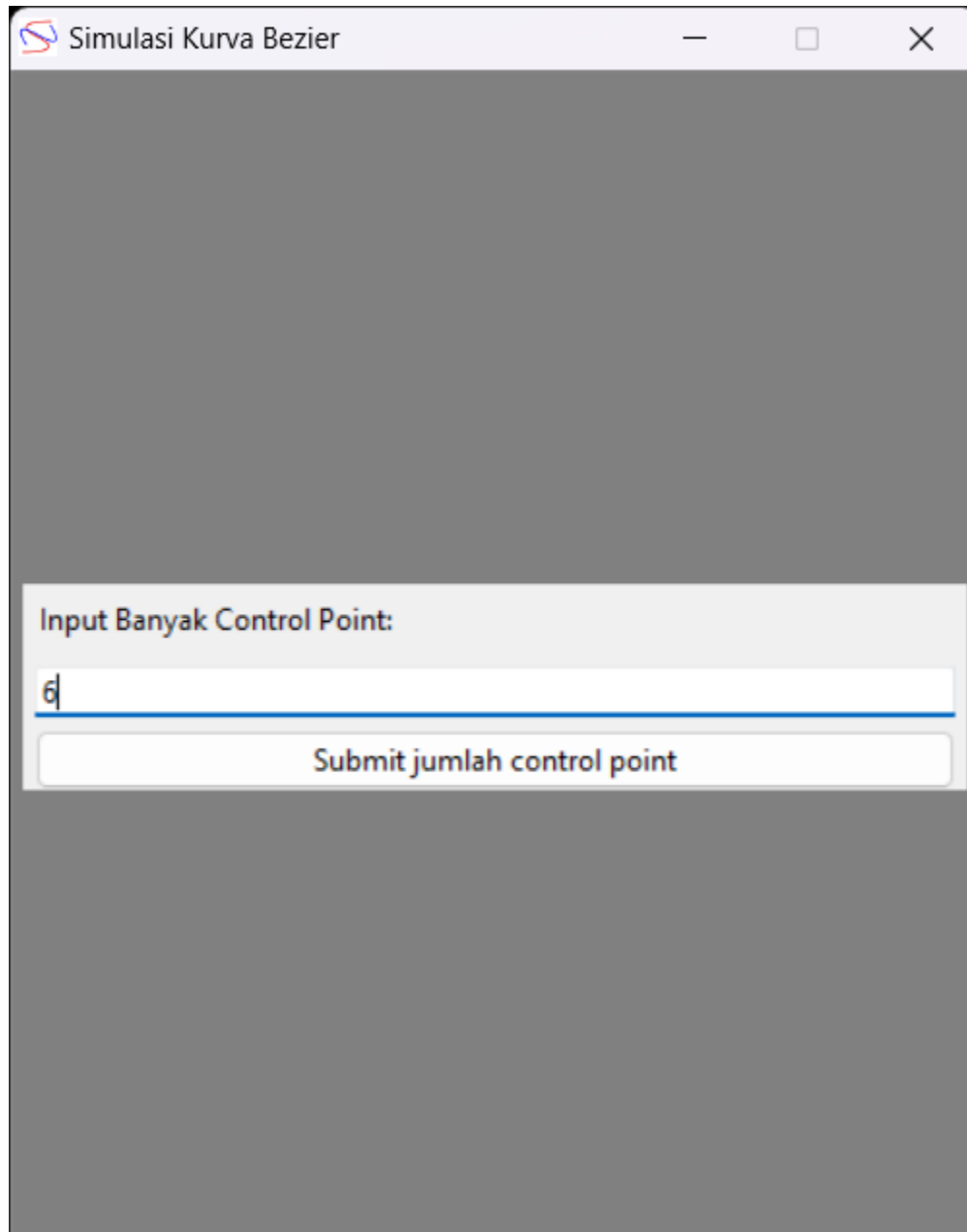
A. Hasil dengan menggunakan Metode Brute Force



B. Hasil dengan menggunakan Divide And Conquer



Set point 3




Simulasi Kurva Bezier

Input Banyak Control Point:

6

Submit jumlah control point

 Simulasi Kurva Bezier

—

□

×

Titik Kontrol P0

Nilai X:

Nilai Y:

Titik Kontrol P1

Nilai X:


Nilai Y:

Titik Kontrol P2

Nilai X:

Nilai Y:

Titik Kontrol P3

 Simulasi Kurva Bezier

Titik Kontrol P3

Nilai X:

-5

Nilai Y:

2.5

Titik Kontrol P4

Nilai X:

5

Nilai Y:

2.5

Titik Kontrol P5

Nilai X:

-3.5

Nilai Y:

-5

Banyak Iterasi

Simulasi Kurva Bezier

2.5

Titik Kontrol P4

Nilai X:

5

Nilai Y:

2.5

Titik Kontrol P5

Nilai X:

-3.5

Nilai Y:

-5

Banyak Iterasi

5

Pilih Metode Pembentukan Kurva Bezier

☒ Brute Force

☐ Divide And Conquer

Simulasi Kurva Bezier

2.5

Titik Kontrol P4

Nilai X:

5

Nilai Y:

2.5

Titik Kontrol P5

Nilai X:

-3.5

Nilai Y:

-5

Banyak Iterasi

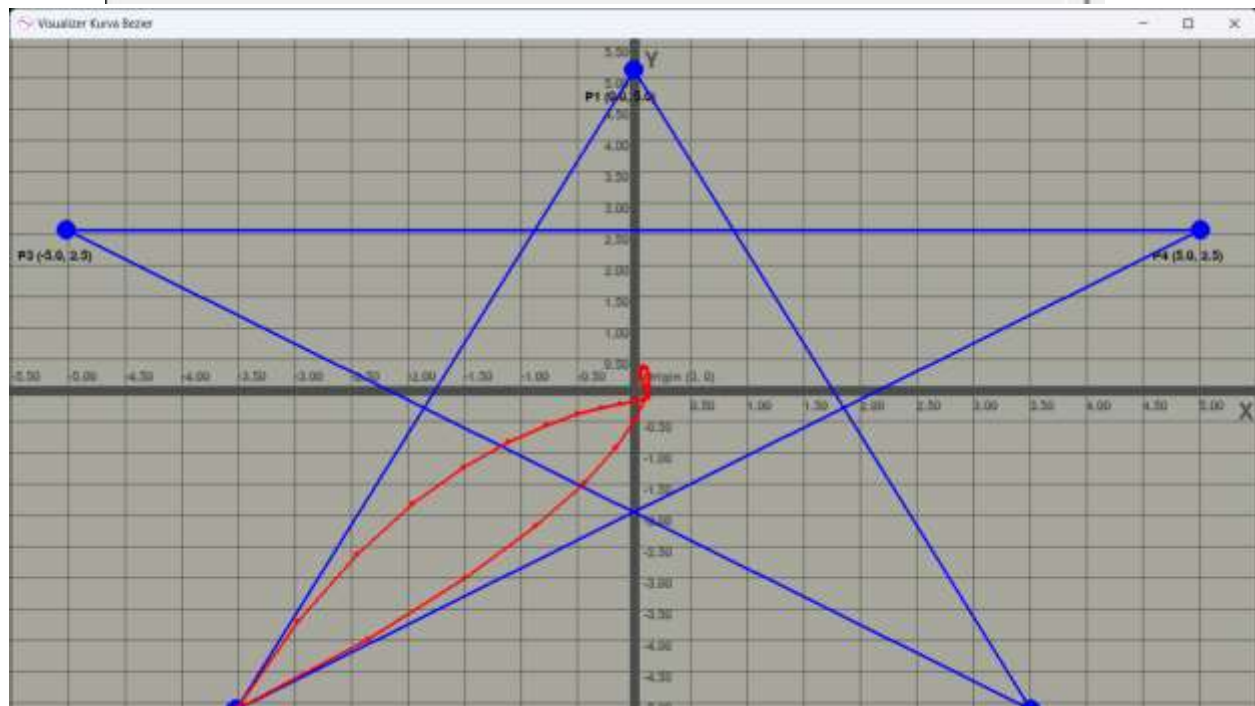
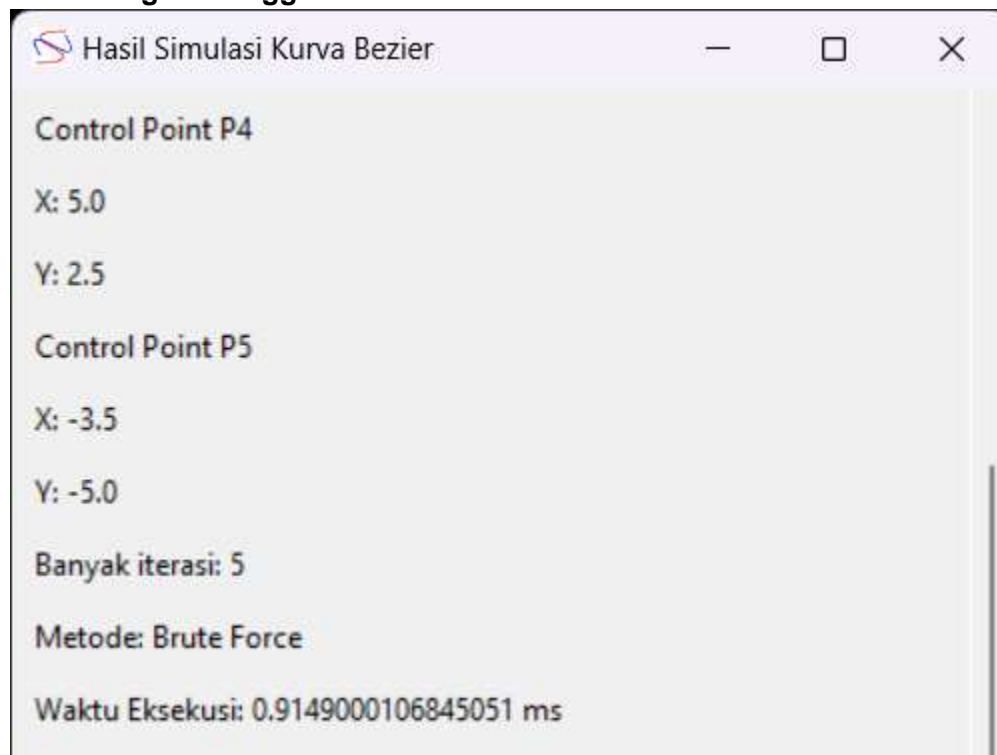
5

Pilih Metode Pembentukan Kurva Bezier

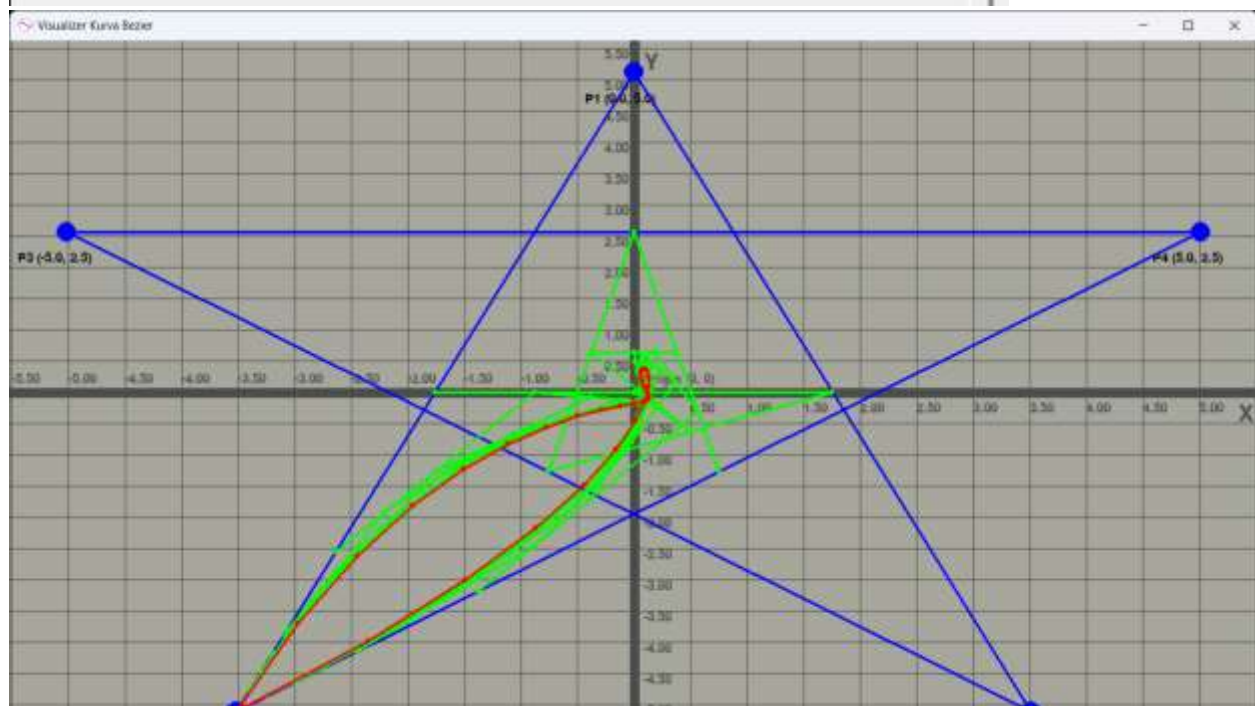
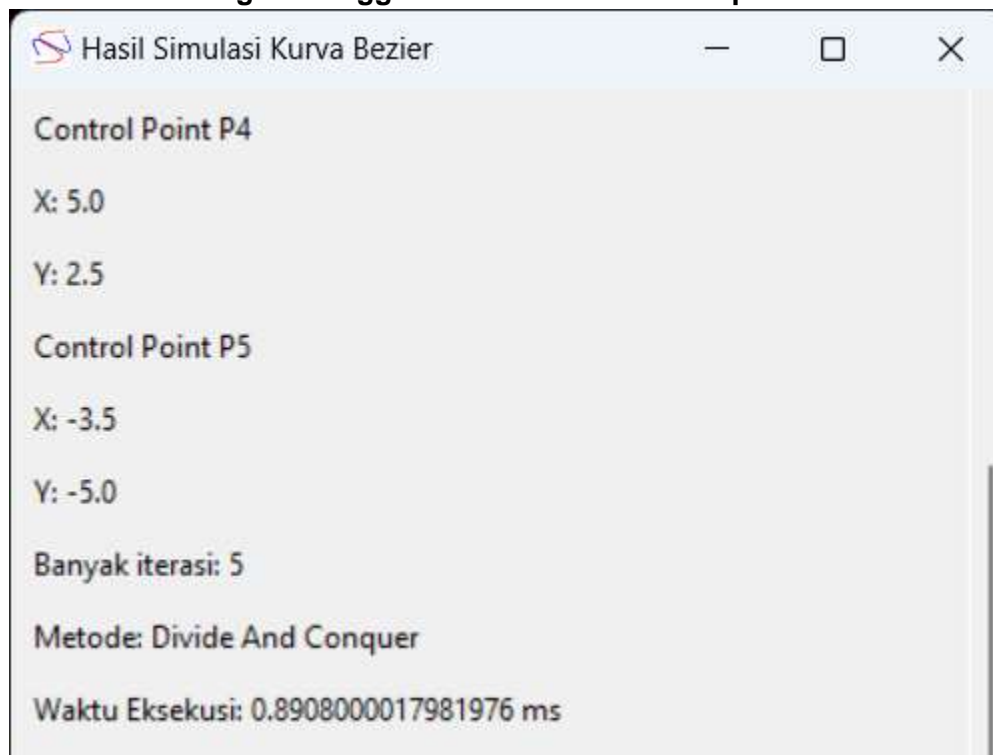
☐ Brute Force

☒ Divide And Conquer

A. Hasil dengan menggunakan Metode Brute Force



B. Hasil dengan menggunakan Divide And Conquer



Bab 5: Perbandingan dan Analisis Kompleksitas Waktu

Setelah melakukan beberapa uji coba, didapatkan bahwa metode brute force dan divide and conquer memiliki waktu eksekusi yang relatif sama (termasuk galat yang disebabkan oleh faktor eksternal seperti python itu sendiri atau device yang menjalankan program ini). Berikut ini adalah pembahasan mengapa hal tersebut terjadi.

1. Analisis Kompleksitas Waktu untuk Kurva Bezier Kuadratik

a. Divide And Conquer

Misalkan banyak iterasi pada divide and conquer adalah m . Di dalam source code tersebut ada pemanggilan rekursif sebanyak dua kali. Jadi, didapat

$$T(m) = 2T(m-1) + 1$$

Perhatikan bahwa setiap $T(m)$ akan memanggil 2 buah fungsi $T(m-1)$. Artinya, 1 buah fungsi $T(m)$ akan memanggil 2 buah fungsi $T(m-1)$ dan akan memanggil total 4 buah fungsi $T(m-2)$... dan seterusnya sampai $T(1)$.

Jadi, kompleksitas waktu untuk algoritma divide and conquer adalah $O(2^m + 1) = O(2^m)$ dengan m adalah banyak iterasi yang dilakukan dan akan terbentuk $2^m + 1$ buah titik.

b. Brute Force

Misalkan banyak **titik** yang akan dibuat adalah p . Di dalam source code tersebut akan dibuat titik sebanyak $p+1$ buah dan hanya ada 1 buah for loop yang mengiterasikan i dari 0 sampai $p+1$.

Jadi, kompleksitas waktu untuk algoritma brute force adalah $O(p)$ dengan p adalah banyak titik.

Akan tetapi, jika kita hanya mengartikan $p = m$ dengan m adalah banyak iterasi yang dilakukan, maka hasilnya tidak akan sama dengan kurva bezier yang dihasilkan oleh metode divide and conquer karena banyak titik di kurva bezier yang dibuat di metode divide and conquer adalah **$2^m + 1$ buah titik.**

Oleh karena itu, agar sebanding, maka p haruslah sama dengan **$2^m + 1$ ($p = 2^m + 1$)**. Akibatnya, **kompleksitas waktu untuk algoritma brute force adalah $O(p) = O(2^m + 1) = O(2^m)$ dengan p adalah banyak titik yang dibuat dan m adalah banyak iterasi (diseragamkan dengan iterasi di metode divide and conquer agar kedua kurva setara berdasarkan jumlah titik yang dihasilkan).**

2. Analisis Kompleksitas Waktu untuk Kurva Bezier dengan n buah control point

a. Divide And Conquer

Misalkan n adalah banyak control point dan m adalah banyak iterasi yang dilakukan. Di dalam source code tersebut, untuk melakukan iterasi pertama dengan indeks i dari derajat kurva bezier ke 0 membutuhkan n kali iterasi, dan terdapat nested loop dari 0 sampai i (maksimal adalah $i = \text{derajat kurva bezier}$)

Karena derajat kurva bezier = $n-1$ dengan n adalah banyak control point, didapatkan bahwa dari proses ini saja didapatkan **kompleksitas waktu sebesar $O((n-1) \cdot (n-1)) = O(n^2)$ dengan n adalah banyak control point.**

Kemudian, ada tahap divide and conquer lagi yang akan memanggil 2 fungsi lagi secara terus menerus sampai iterasi mencapai 0. Akibatnya, **kompleksitas**

waktu dari proses ini saja adalah $O(2^m + 1) = O(2^m)$ dengan m adalah banyak iterasi yang dilakukan.

Jadi, kompleksitas waktu dari metode divide and conquer adalah $O(n^2 \cdot 2^m)$ dengan n adalah banyak control point dan m adalah banyak iterasi yang dilakukan. Serta, banyak titik yang dihasilkan adalah

b. Brute Force

Misalkan n adalah banyak control point, p adalah banyak titik kurva bezier yang dibuat, dan m adalah banyak iterasi yang dilakukan. Di dalam metode brute force, kita harus memanggil fungsi **kombinasi yang memiliki kompleksitas $O(n-1) = O(n)$** . Kombinasi itu dibutuhkan untuk menghitung nilai fungsi Bernstein yang beriterasi dari 0 sampai n. Setiap kali iterasi fungsi Bernstein dilakukan, kita perlu memanggil fungsi kombinasi. Jadi, kompleksitas waktu untuk proses ini saja adalah $O(n^2)$.

Setelah itu, akan ada simulasi titik sebanyak p kali dari fungsi Bernstein tersebut. Jadi, **kompleksitas waktunya adalah $O(n^2 \cdot p)$**

Akan tetapi, jika kita hanya mengartikan $p = m$ dengan m adalah banyak iterasi yang dilakukan, maka hasilnya tidak akan sama dengan kurva bezier yang dihasilkan oleh metode divide and conquer karena banyak titik di kurva bezier yang dibuat di metode divide and conquer adalah **$2^m + 1$ buah titik**.

Oleh karena itu, agar sebanding, maka p haruslah sama dengan **$2^m + 1$ ($p = 2^m + 1$)**. Akibatnya, **kompleksitas waktu untuk algoritma brute force adalah $O(n^2 \cdot p) = O(n^2 \cdot (2^m + 1)) = O(n^2 \cdot 2^m)$** dengan n adalah banyak control point, p adalah banyak titik yang dibuat dan m adalah banyak iterasi (diseragamkan dengan iterasi di metode divide and conquer agar kedua kurva setara berdasarkan jumlah titik yang dihasilkan).

Kesimpulan yang didapat adalah metode brute force dan divide and conquer memiliki kompleksitas waktu yang sama, yaitu $O(n^2 \cdot 2^m)$ dengan n adalah banyak control point dan m adalah banyak iterasi yang dilakukan. Akibatnya, waktu eksekusinya juga mirip.

Lampiran

Link Repository

https://github.com/DeltDev/Tucil2_13522036_13522064

Check list program:

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	