

LAPORAN TUGAS KECIL 2

IF2211 STRATEGI ALGORITMA

“Membangun Kurva Bézier dengan Algoritma Titik Tengah
Berbasis Divide and Conquer”



Disusun oleh:

Andhita Naura Haryanto K02 13522060

Shulha K01 13522087

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

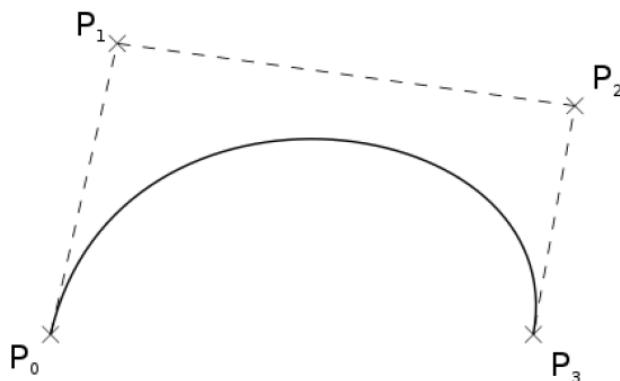
DAFTAR ISI

DAFTAR ISI	2
BAB I	3
DESKRIPSI MASALAH	3
BAB II	4
LANGKAH-LANGKAH ALGORITMA BRUTE FORCE	4
BAB III	6
LANGKAH-LANGKAH ALGORITMA DIVIDE AND CONQUER	6
BAB IV	9
IMPLEMENTASI PROGRAM DALAM PYTHON	9
BAB V	14
EKSPERIMEN	14
BAB VI	27
ANALISIS	27
BAB VII	29
PENUTUP	29

BAB I

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.



Gambar 1. Kurva Bézier

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P0 sampai Pn , 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 P0, sedangkan titik kontrol terakhir adalah P3. Titik kontrol P1 dan P2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Kurva Bézier linier dibentuk dari dua titik kontrol, kurva Bézier kuadratik dibentuk dari dua titik kontrol, kurva Bézier kubik dibentuk dari empat titik kontrol, begitu seterusnya dengan kurva Bézier kuartik atau kurva Bézier yang dibentuk dari lebih banyak titik kontrol. Misal dengan kurva Bézier Kuadratik dapat diperoleh titik baru R0 dari titik kontrol P0, P1, dan P3 yaitu dengan persamaan

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

yang kemudian dihubungkan membentuk garis P0-R0-P2 membentuk kurva Bézier iterasi pertama.

Pada tugas ini, penulis menggunakan algoritma titik tengah sehingga mensubsitusi nilai t dengan $\frac{1}{2}$. Pembentukan kurva Bézier dengan titik tengah ini dibuat dengan algoritma *brute force* maupun *divide and conquer* dengan bahasa Python.

BAB II

LANGKAH-LANGKAH ALGORITMA BRUTE FORCE

Algoritma yang digunakan untuk menyelesaikan permasalahan dengan *brute force* adalah dengan mengabstraksi persoalan seperti berikut:

1. Titik-titik tengah sementara yang penulis definisikan sebagai ‘*midpoint*’ adalah titik-titik tengah dari titik-titik tengah yang penulis definisikan sebagai titik-titik ‘antara’
2. Titik-titik yang menjadi titik Bézier baru atau ‘*new Bézier point*’ adalah titik tengah dari titik-titik tengah sementara ‘*midpoint*’
3. Hasil kurva Bézier untuk suatu iterasi adalah penyisipan ‘*new Bézier point*’ ke hasil kurva Bézier iterasi sebelumnya
4. Titik-titik ‘antara’ dari suatu iterasi kemudian diperbarui dengan menyisipkan ‘*midpoint*’ iterasi sebelumnya ke hasil kurva Bézier iterasi sebelumnya

Untuk mempermudah pemahaman algoritma tersebut, penulis akan mengilustrasikannya lebih lanjut. Misal, titik-titik kontrol awal adalah p_0 , p_1 , dan p_2 , maka kita sebut titik-titik tersebut sebagai ‘antara1’ dengan p_0 dan p_2 sebagai ‘hasil0’. Maka, dengan algoritma *brute force* yang sudah didefinisikan, dilakukan iterasi pertama sebagai berikut:

1. ‘*Midpoint 1*’ adalah titik-titik tengah dari ‘antara1’ yaitu q_0 antara p_0 dan p_1 serta q_1 antara p_1 dan p_2 . ‘*Midpoint 1*’ = $[q_0, q_1]$
2. ‘*New Bézier point 1*’ adalah titik-titik tengah dari ‘*Midpoint 1*’ yaitu r_0 antara q_0 dan q_1 sehingga ‘*New Bézier point 1*’ = $[r_0]$
3. Hasil kurva Bézier untuk iterasi pertama adalah penyisipan ‘*New Bézier point 1*’ yaitu r_0 ke ‘hasil0’ yaitu p_0 dan p_2 sehingga ‘hasil1’ = $[p_0, r_0, p_2]$
4. Titik-titik ‘antara’ kemudian diperbarui dengan menyisipkan ‘*Midpoint 1*’ = $[q_0, q_1]$ ke hasil kurva Bézier ‘hasil1’ = $[p_0, r_0, p_2]$ sehingga ‘antara2’ = $[p_0, q_0, r_0, q_1, p_2]$

Dengan algoritma yang sama, dilakukan iterasi kedua sebagai berikut:

1. ‘*Midpoint 2*’ adalah titik-titik tengah dari ‘antara2’ = $[p_0, q_0, r_0, q_1, p_2]$ yaitu s_0 antara p_0 dan q_0 kemudian s_1 antara q_0 dan r_0 dan selanjutnya sehingga diperoleh ‘*Midpoint 2*’ = $[s_0, s_1, s_2, s_3]$
2. ‘*New Bézier point 2*’ adalah titik-titik tengah dari ‘*Midpoint 2*’ = $[s_0, s_1, s_2, s_3]$ yaitu t_0 antara s_0 dan s_1 serta t_1 antara s_2 dan s_3 sehingga ‘*New Bézier point 2*’ = $[t_0, t_1]$

3. Hasil kurva Bézier untuk iterasi kedua adalah penyisipan ‘*New Bézier point 2*’ = [t0, t1] sehingga ‘hasil2’ = [p0, t0, r0, t1, p2]
4. Titik-titik ‘antara’ kemudian diperbarui dengan menyisipkan ‘*Midpoint 2*’ = [s0, s1, s2, s3] ke hasil kurva Bézier ‘hasil2’ = [p0, t0, r0, t1, p2] sehingga ‘antara3’ = [p0, s0, q0, s1, r0, s2, q1, s3, p2]

Ketika kita melanjutkan algoritma di atas untuk iterasi ketiga diperoleh ‘*Midpoint 3*’ = [u0, u1, u2, u3, u4, u5, u6, u7], ‘*New Bézier point 3*’ = [v0, v1, v2, v3], dan ‘hasil3’ = [p0, v0, t0, v1, r0, v2, t1, v3, p2] yang sudah terdiri dari tiga titik yang baik membentuk kurva. Iterasi dilakukan sebanyak yang diinginkan, semakin banyak iterasi yang dilakukan hasil kurva Bézier akan semakin mulus.

BAB III

LANGKAH-LANGKAH ALGORITMA DIVIDE AND CONQUER

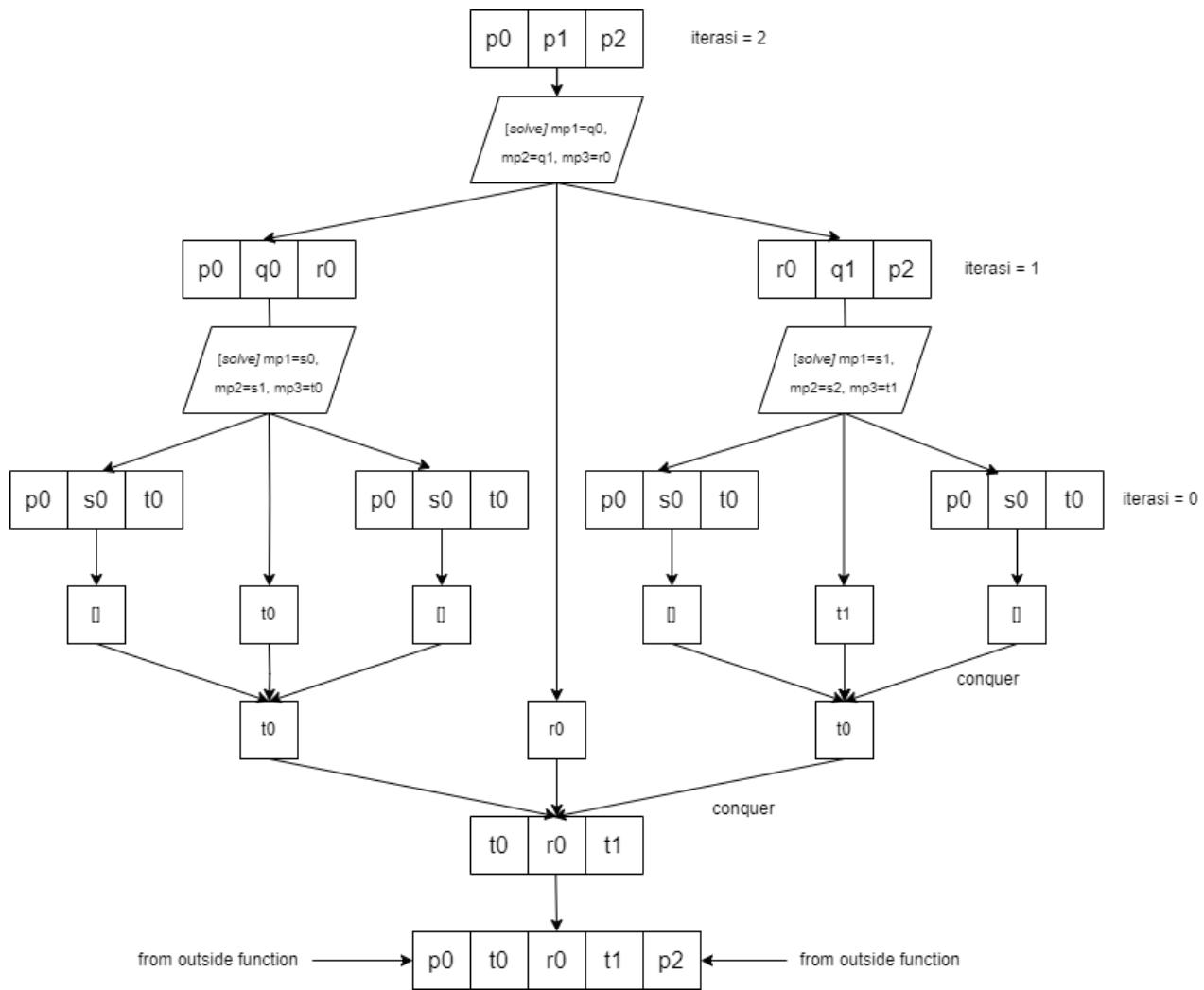
Algoritma yang digunakan untuk menyelesaikan permasalahan dengan *divide and conquer* adalah dengan memanfaatkan rekursivitas dengan mengabstraksi persoalan seperti penjelasan berikut.

Sebelumnya, ingat bahwa pada persoalan membangun kurva Bézier kuadratik dimulai dengan tiga titik kontrol. Fungsi *divide and conquer* menerima parameter berupa tiga titik dan jumlah iterasi. Titik-titik berjumlah tiga dibagi (*divide*) menjadi dua bagian yaitu kanan dan kiri juga yang tengah. Cara membaginya adalah sebagai berikut:

1. [solve] Fungsi akan mencari nilai dari titik tengah antara titik pertama dan kedua (disebut ‘*midpoint 1*’) dan titik tengah antara titik kedua dan ketiga (disebut ‘*midpoint 2*’)
2. [solve] Fungsi akan mencari titik tengah dari ‘*midpoint 1*’ dan ‘*midpoint 2*’ yang disebut ‘*midpoint 3*’
3. Nilai dari iterasi yang menjadi parameter fungsi akan dikurangi (*decrement*) satu
4. Sebuah variabel ‘BézierPoint’ diinisiasi sebagai array kosong
5. [*divide and conquer*] Variabel ‘BézierPoint’ akan diisi dengan hasil dari fungsi *divide and conquer* untuk titik-titik sebelah kiri yaitu titik pertama, ‘*midpoint 1*’, dan ‘*midpoint 3*’ dengan nilai iterasi yang sudah dikurangi
6. [*conquer*] Variabel ‘BézierPoint’ akan di-*append* dengan nilai titik tengah ‘*midpoint 3*’
7. [*divide and conquer*] Variabel ‘BézierPoint’ akan di-*extend* dengan hasil dari fungsi *divide and conquer* untuk titik-titik sebelah kanan yaitu titik tengah ‘*midpoint 3*’ , ‘*midpoint 2*’, dan ‘titik ketiga’ dengan nilai iterasi yang sudah dikurangi
8. Fungsi akan mengembalikan array kosong ketika mencapai basis yaitu pemanggilan fungsi dengan parameter iterasi bernilai 0

Fungsi yang dikembalikan akan diapit oleh titik kontrol awal dan titik kontrol akhir yang membangun kurva Bézier.

Untuk mempermudah pemahaman terhadap algoritma *divide and conquer* ini, penjelasan algoritma untuk dua kali iterasi diilustrasikan pada diagram berikut.



Gambar 2. Ilustrasi algoritma divide and conquer

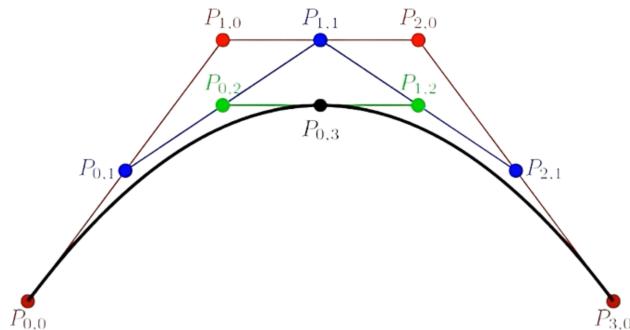
Algoritma yang telah dijelaskan di atas sangat berguna dan dapat membentuk kurva Bézier kuadratik dengan baik. Selain dengan menggunakan tiga titik kontrol (kurva Bézier kuadratik), kurva ini juga bisa dibuat dengan empat titik kontrol (kurva Bézier kubik), lima titik kontrol (kurva Bézier kuartik), hingga n buah titik kontrol. Dengan prinsip algoritma *divide and conquer*, membentuk kurva Bézier dengan n titik kontrol menjadi baik untuk diimplementasikan.

Algoritma *divide and conquer* untuk membentuk kurva Bézier dengan n titik kontrol dilakukan dengan mengabstraksi persoalan sebagai berikut:

1. **[solve]** Pencarian titik tengah paling dalam dari n titik kontrol

Titik tengah terdalam adalah titik tengah dari titik-titik tengah antara titik-titik kontrol dan dilakukan secara menurun dari n titik kontrol ke $n-1$ titik, $n-2$ titik hingga tersisa 2 titik. Pada ilustrasi pembentukan kurva Bézier iterasi pertama berikut, iterasi dilakukan dari empat titik

kontrol awal lalu mendapatkan titik-titik tengah dari empat titik tersebut menjadi tiga titik, kemudian dua titik dan mendapatkan titik tengah terdalam yaitu $P_{0,3}$ pada ilustrasi di bawah



Gambar 3. Pencarian titik tengah terdalam dari empat titik kontrol

2. [solve] Mendapatkan titik tengah kanan dan titik tengah kiri

Selama pencarian titik tengah terdalam seperti yang sudah dijelaskan sebelumnya, diambil juga nilai-nilai dari titik tengah paling kanan dan titik tengah paling kiri dari setiap iterasi. Pada ilustrasi Gambar 3 di atas, ‘titik tengah kiri’ adalah $P_{0,0}$, $P_{0,1}$, $P_{0,2}$, dan $P_{0,3}$. Sementara ‘titik tengah kanan’ adalah $P_{0,3}$, $P_{1,2}$, $P_{2,1}$, dan $P_{3,0}$. Masing-masing titik tengah kanan maupun kiri akan terdiri dari n buah titik sejumlah banyaknya titik kontrol. Hal ini akan dimanfaatkan untuk melakukan algoritma *divide and conquer* secara rekursif dengan membagi titik-titik kontrol menjadi kanan, tengah, dan kiri.

3. Untuk fungsi *divide and conquer*, setelah melakukan pencarian titik tengah terdalam, nilai dari iterasi yang menjadi parameter fungsi *divide and conquer* akan dikurangi (*decrement*) satu
4. Sebuah variabel ‘BézierPoint’ diinisiasi sebagai array kosong
5. [divide and conquer] Variabel ‘BézierPoint’ akan diisi dengan hasil dari fungsi *divide and conquer* untuk n titik-titik sebelah kiri pada ‘titik tengah kiri’ dengan nilai iterasi yang sudah dikurangi sebagai parameter
6. [conquer] Variabel ‘BézierPoint’ akan di-*append* dengan nilai titik tengah terdalam
7. [divide and conquer] Variabel ‘BézierPoint’ akan di-*extend* dengan hasil dari fungsi *divide and conquer* untuk n titik-titik sebelah kiri pada ‘titik tengah kanan’ dengan nilai iterasi yang sudah dikurangi sebagai parameter
8. Fungsi akan mengembalikan array kosong ketika mencapai basis yaitu pemanggilan fungsi dengan parameter iterasi bernilai 0

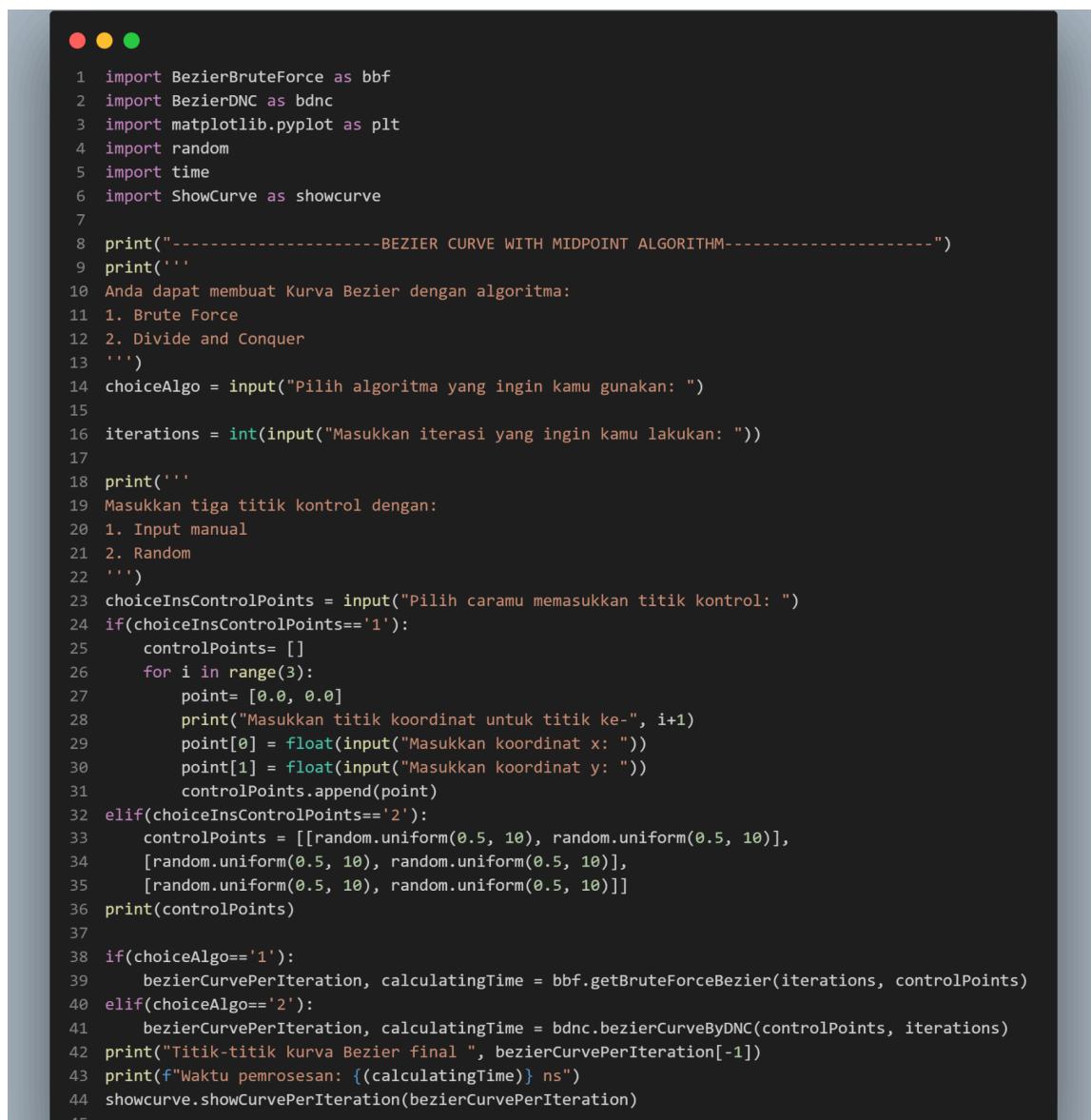
Fungsi yang dikembalikan akan diapit oleh titik kontrol awal dan titik kontrol akhir yang membangun kurva Bézier.

BAB IV

IMPLEMENTASI PROGRAM DALAM PYTHON

1. Antarmuka Awal

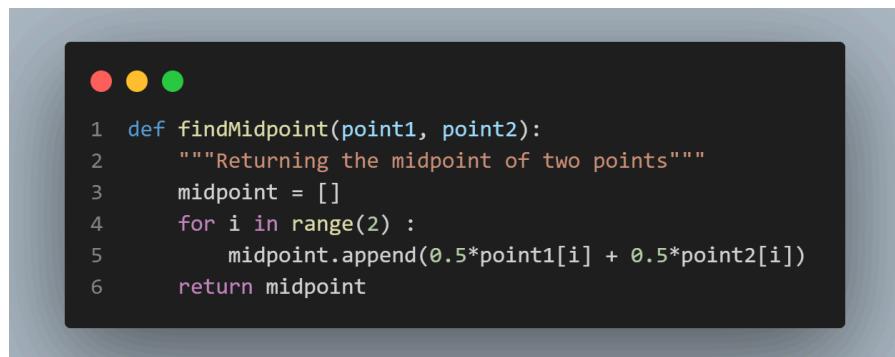
Kode program ini disimpan pada file `Bézier.py`. Program melakukan import berbagai library seperti time dan matplotlib. Antarmuka dimulai dengan meminta masukan algoritma apa yang ingin digunakan. Kemudian meminta masukan titik-titik kontrol dan jumlah iterasi yang ingin dilakukan. Program akan menampilkan titik-titik kurva Bézier dan waktu yang dibutuhkan untuk membangun kurva Bézier pada CLI dan menampilkan hasil kurva Bézier setiap iterasi pada jendela matplotlib.



```
 1 import BezierBruteForce as bbf
 2 import BezierDNC as bdnc
 3 import matplotlib.pyplot as plt
 4 import random
 5 import time
 6 import ShowCurve as showcurve
 7
 8 print("-----BEZIER CURVE WITH MIDPOINT ALGORITHM-----")
 9 print('')
10 Anda dapat membuat Kurva Bezier dengan algoritma:
11 1. Brute Force
12 2. Divide and Conquer
13 ''')
14 choiceAlgo = input("Pilih algoritma yang ingin kamu gunakan: ")
15
16 iterations = int(input("Masukkan iterasi yang ingin kamu lakukan: "))
17
18 print('')
19 Masukkan tiga titik kontrol dengan:
20 1. Input manual
21 2. Random
22 ''')
23 choiceInsControlPoints = input("Pilih caramu memasukkan titik kontrol: ")
24 if(choiceInsControlPoints=='1'):
25     controlPoints= []
26     for i in range(3):
27         point= [0.0, 0.0]
28         print("Masukkan titik koordinat untuk titik ke-", i+1)
29         point[0] = float(input("Masukkan koordinat x: "))
30         point[1] = float(input("Masukkan koordinat y: "))
31         controlPoints.append(point)
32 elif(choiceInsControlPoints=='2'):
33     controlPoints = [[random.uniform(0.5, 10), random.uniform(0.5, 10)],
34                     [random.uniform(0.5, 10), random.uniform(0.5, 10)],
35                     [random.uniform(0.5, 10), random.uniform(0.5, 10)]]
36 print(controlPoints)
37
38 if(choiceAlgo=='1'):
39     bezierCurvePerIteration, calculatingTime = bbf.getBruteForceBezier(iterations, controlPoints)
40 elif(choiceAlgo=='2'):
41     bezierCurvePerIteration, calculatingTime = bdnc.bezierCurveByDNC(controlPoints, iterations)
42 print("Titik-titik kurva Bezier final ", bezierCurvePerIteration[-1])
43 print(f"Waktu pemrosesan: {(calculatingTime)} ns")
44 showcurve.showCurvePerIteration(bezierCurvePerIteration)
45
```

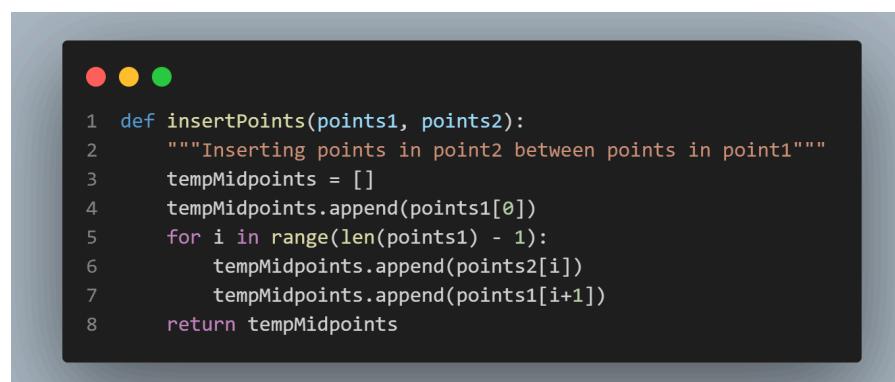
Gambar 4. Antarmuka Awal Program

2. Implementasi Algoritma Brute Force



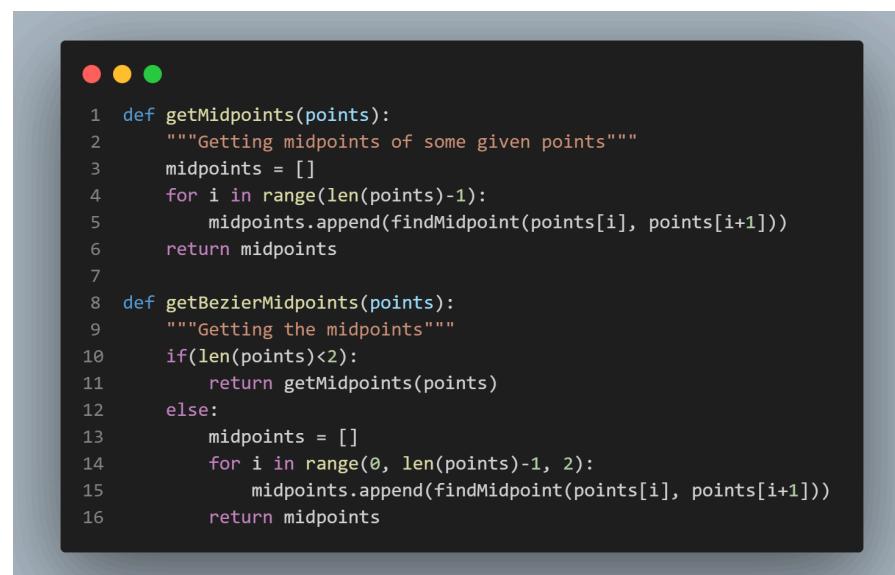
```
1 def findMidpoint(point1, point2):
2     """Returning the midpoint of two points"""
3     midpoint = []
4     for i in range(2) :
5         midpoint.append(0.5*point1[i] + 0.5*point2[i])
6     return midpoint
```

Gambar 5. Fungsi mencari titik tengah dua titik



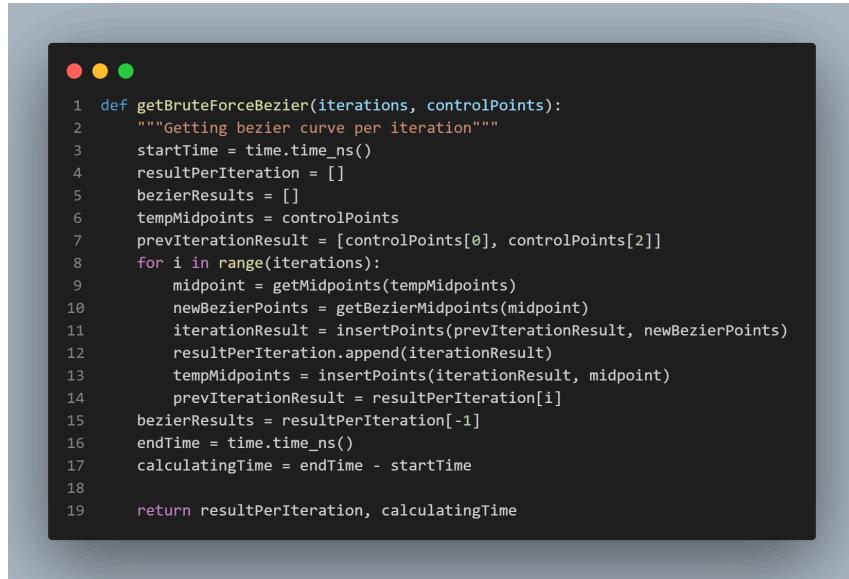
```
1 def insertPoints(points1, points2):
2     """Inserting points in point2 between points in point1"""
3     tempMidpoints = []
4     tempMidpoints.append(points1[0])
5     for i in range(len(points1) - 1):
6         tempMidpoints.append(points2[i])
7         tempMidpoints.append(points1[i+1])
8     return tempMidpoints
```

Gambar 6. Fungsi menyisipkan titik-titik ‘points2’ di antara titik-titik ‘points1’



```
1 def getMidpoints(points):
2     """Getting midpoints of some given points"""
3     midpoints = []
4     for i in range(len(points)-1):
5         midpoints.append(findMidpoint(points[i], points[i+1]))
6     return midpoints
7
8 def getBezierMidpoints(points):
9     """Getting the midpoints"""
10    if(len(points)<2):
11        return getMidpoints(points)
12    else:
13        midpoints = []
14        for i in range(0, len(points)-1, 2):
15            midpoints.append(findMidpoint(points[i], points[i+1]))
16    return midpoints
```

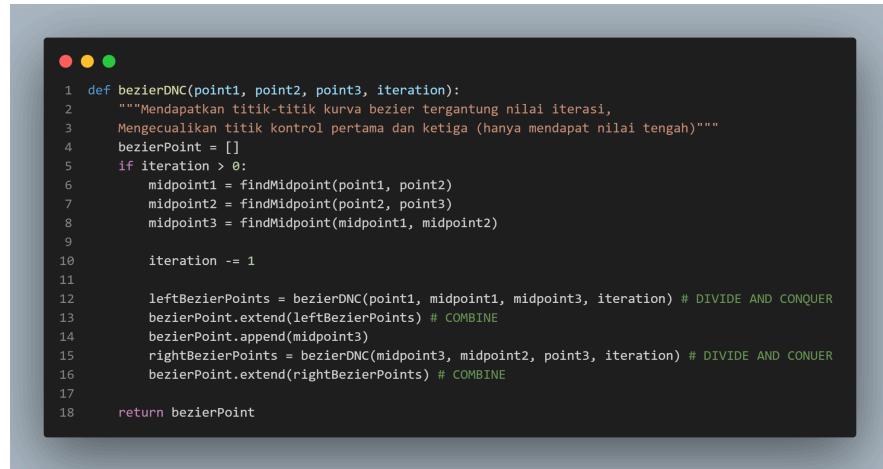
Gambar 7. Fungsi mendapatkan titik-titik tengah dari serangkaian titik



```
1 def getBruteForceBezier(iterations, controlPoints):
2     """Getting bezier curve per iteration"""
3     startTime = time.time_ns()
4     resultPerIteration = []
5     bezierResults = []
6     tempMidpoints = controlPoints
7     prevIterationResult = [controlPoints[0], controlPoints[2]]
8     for i in range(iterations):
9         midpoint = getMidpoints(tempMidpoints)
10        newBezierPoints = getBezierMidpoints(midpoint)
11        iterationResult = insertPoints(prevIterationResult, newBezierPoints)
12        resultPerIteration.append(iterationResult)
13        tempMidpoints = insertPoints(iterationResult, midpoint)
14        prevIterationResult = resultPerIteration[i]
15    bezierResults = resultPerIteration[-1]
16    endTime = time.time_ns()
17    calculatingTime = endTime - startTime
18
19    return resultPerIteration, calculatingTime
```

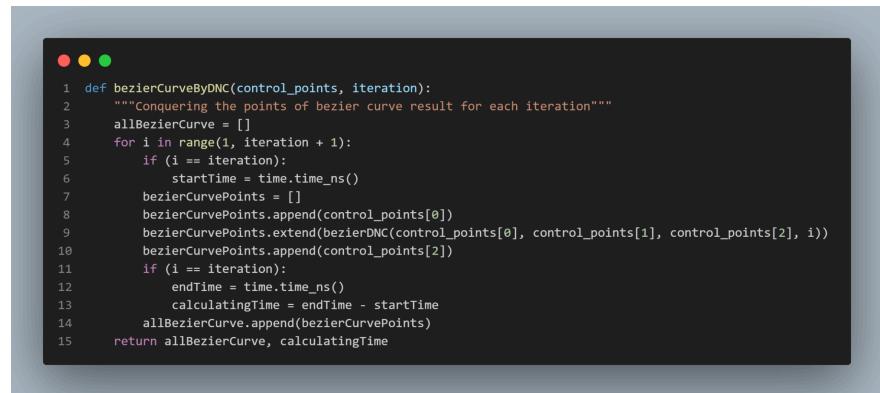
Gambar 8. Fungsi mendapatkan kurva Bézier setiap iterasi

3. Implementasi Algoritma Divide and Conquer



```
1 def bezierDNC(point1, point2, point3, iteration):
2     """Mendapatkan titik-titik kurva bezier tergantung nilai iterasi,
3     Mengecualikan titik kontrol pertama dan ketiga (hanya mendapat nilai tengah)"""
4     bezierPoint = []
5     if iteration > 0:
6         midpoint1 = findMidpoint(point1, point2)
7         midpoint2 = findMidpoint(point2, point3)
8         midpoint3 = findMidpoint(midpoint1, midpoint2)
9
10    iteration -= 1
11
12    leftBezierPoints = bezierDNC(point1, midpoint1, midpoint3, iteration) # DIVIDE AND CONQUER
13    bezierPoint.extend(leftBezierPoints) # COMBINE
14    bezierPoint.append(midpoint3)
15    rightBezierPoints = bezierDNC(midpoint3, midpoint2, point3, iteration) # DIVIDE AND CONQUER
16    bezierPoint.extend(rightBezierPoints) # COMBINE
17
18    return bezierPoint
```

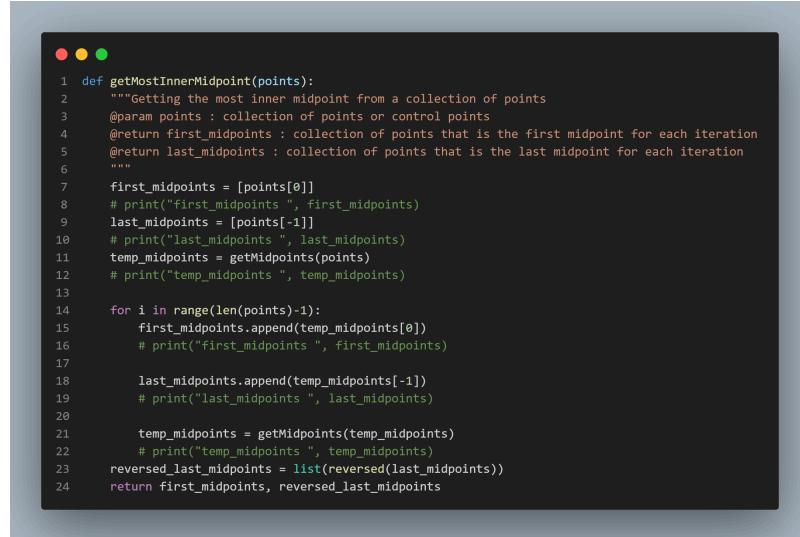
Gambar 9. Fungsi mendapatkan titik-titik untuk kurva Bézier



```
1 def bezierCurveByDNC(control_points, iteration):
2     """Conquering the points of bezier curve result for each iteration"""
3     allBezierCurve = []
4     for i in range(1, iteration + 1):
5         if (i == iteration):
6             startTime = time.time_ns()
7             bezierCurvePoints = []
8             bezierCurvePoints.append(control_points[0])
9             bezierCurvePoints.extend(bezierDNC(control_points[0], control_points[1], control_points[2], i))
10            bezierCurvePoints.append(control_points[2])
11        if (i == iteration):
12            endTime = time.time_ns()
13            calculatingTime = endTime - startTime
14            allBezierCurve.append(bezierCurvePoints)
15
16    return allBezierCurve, calculatingTime
```

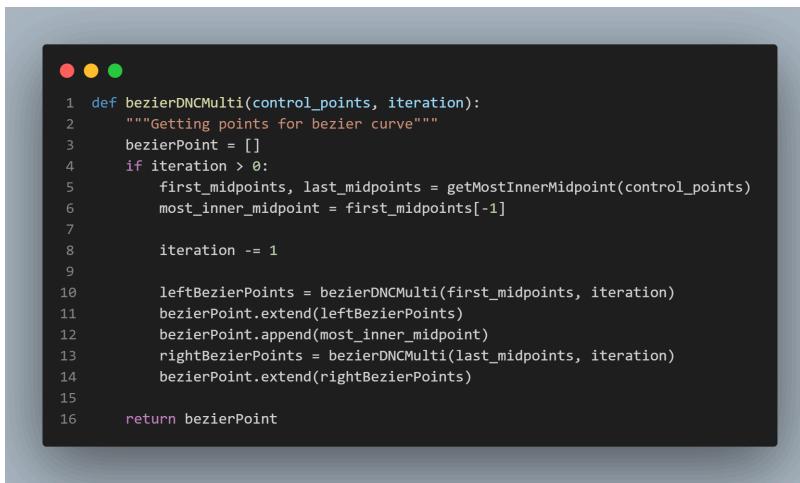
Gambar 10. Fungsi mendapatkan kurva Bézier setiap iterasi

4. Implementasi Algoritma Divide and Conquer N Titik Kontrol



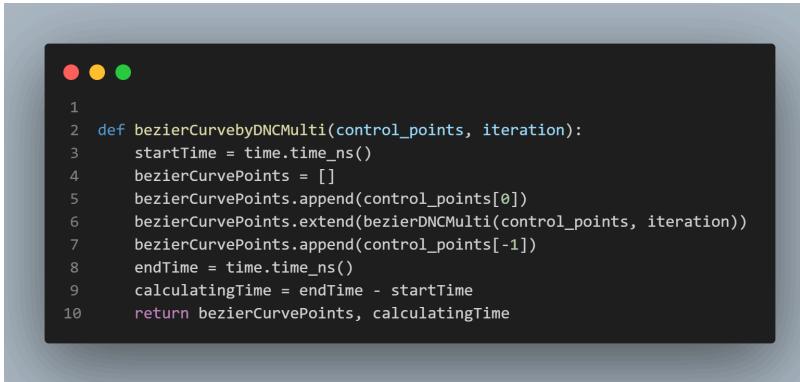
```
1 def getMostInnerMidpoint(points):
2     """Getting the most inner midpoint from a collection of points
3     @param points : collection of points or control points
4     @return first_midpoints : collection of points that is the first midpoint for each iteration
5     @return last_midpoints : collection of points that is the last midpoint for each iteration
6     """
7     first_midpoints = [points[0]]
8     # print("first_midpoints ", first_midpoints)
9     last_midpoints = [points[-1]]
10    # print("last_midpoints ", last_midpoints)
11    temp_midpoints = getMidpoints(points)
12    # print("temp_midpoints ", temp_midpoints)
13
14    for i in range(len(points)-1):
15        first_midpoints.append(temp_midpoints[0])
16        # print("first_midpoints ", first_midpoints)
17
18        last_midpoints.append(temp_midpoints[-1])
19        # print("last_midpoints ", last_midpoints)
20
21        temp_midpoints = getMidpoints(temp_midpoints)
22        # print("temp_midpoints ", temp_midpoints)
23    reversed_last_midpoints = list(reversed(last_midpoints))
24
25    return first_midpoints, reversed_last_midpoints
```

Gambar 11. Fungsi mendapatkan titik tengah terdalam, titik tengah kiri, dan titik tengah kanan



```
1 def bezierDNCMulti(control_points, iteration):
2     """Getting points for bezier curve"""
3     bezierPoint = []
4     if iteration > 0:
5         first_midpoints, last_midpoints = getMostInnerMidpoint(control_points)
6         most_inner_midpoint = first_midpoints[-1]
7
8         iteration -= 1
9
10        leftBezierPoints = bezierDNCMulti(first_midpoints, iteration)
11        bezierPoint.extend(leftBezierPoints)
12        bezierPoint.append(most_inner_midpoint)
13        rightBezierPoints = bezierDNCMulti(last_midpoints, iteration)
14        bezierPoint.extend(rightBezierPoints)
15
16    return bezierPoint
```

Gambar 12. Fungsi mendapatkan titik-titik untuk kurva Bézier



```
1
2 def bezierCurvebyDNCMulti(control_points, iteration):
3     startTime = time.time_ns()
4     bezierCurvePoints = []
5     bezierCurvePoints.append(control_points[0])
6     bezierCurvePoints.extend(bezierDNCMulti(control_points, iteration))
7     bezierCurvePoints.append(control_points[-1])
8     endTime = time.time_ns()
9     calculatingTime = endTime - startTime
10
11    return bezierCurvePoints, calculatingTime
```

Gambar 13. Fungsi mendapatkan kurva Bézier final

Kode program yang ditampilkan di atas adalah garis besar jalannya program, seiring laporan ini dibuat, kode program mungkin mengalami perubahan untuk pengimplementasian visualisasi animasi pembuatan kurva Bézier. Akibat perubahan tersebut, waktu calculatingTime pun menjadi berkali-kali lipat dari kode program tanpa proses visualisasi

BAB V

EKSPERIMENT

1. Eksperimen 1

Titik kontrol adalah (1,4.5) , (6, 4.2), dan (10, 7) dengan tiga kali iterasi.

```
-----BEZIER CURVE WITH MIDPOINT ALGORITHM-----
Anda dapat membuat Kurva Bezier dengan algoritma:
1. Brute Force
2. Quadratic Divide and Conquer
3. N Control Points Divide and Conquer

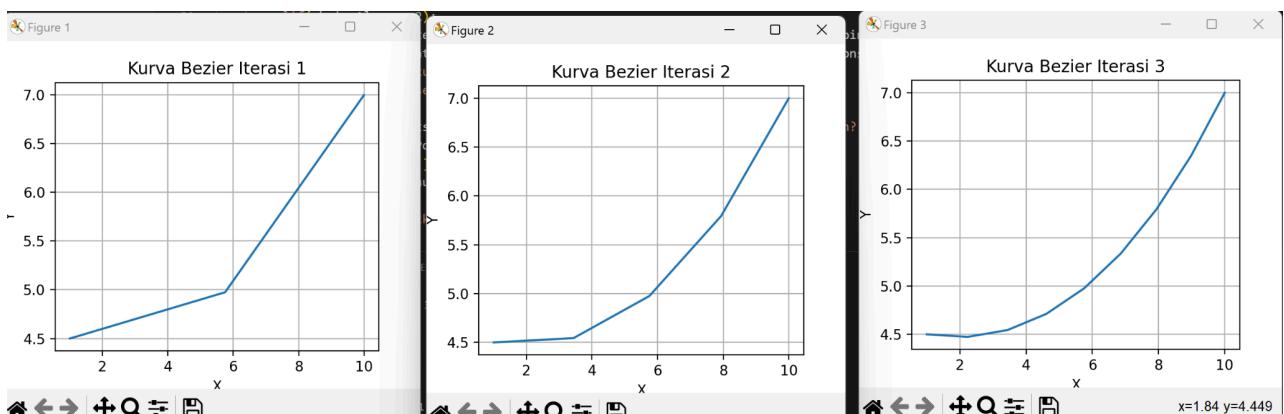
Pilih algoritma yang ingin kamu gunakan: 1
3. N Control Points Divide and Conquer

Pilih algoritma yang ingin kamu gunakan: 1
Masukkan iterasi yang ingin kamu lakukan: 3

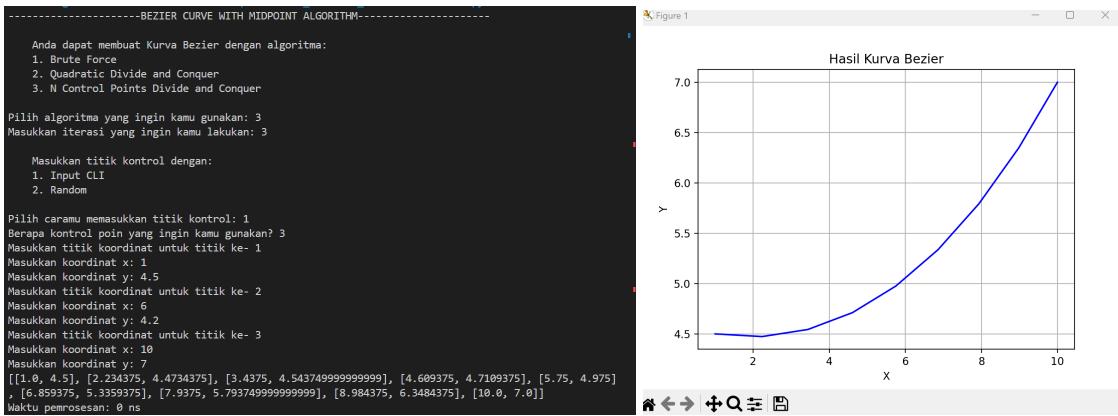
Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih cara kamu memasukkan titik kontrol: 1
Masukkan titik koordinat untuk titik ke- 1
Masukkan koordinat x: 1
Masukkan koordinat y: 4.5
Masukkan titik koordinat untuk titik ke- 2
Masukkan koordinat x: 6
Masukkan koordinat y: 4.2
Masukkan titik koordinat untuk titik ke- 3
Masukkan koordinat x: 10
Masukkan koordinat y: 7
[[1.0, 4.5], [6.0, 4.2], [10.0, 7.0]]
Titik-titik kurva Bezier final [[1.0, 4.5], [2.234375, 4.4734375], [3.4375, 4.543749999999999], [4.609375, 4.7109375], [5.75, 4.975], [6.859375, 5.3359375], [7.9375, 5.793749999999999], [8.984375, 6.3484375], [10.0, 7.0]]
Waktu pemrosesan: 0 ns
```

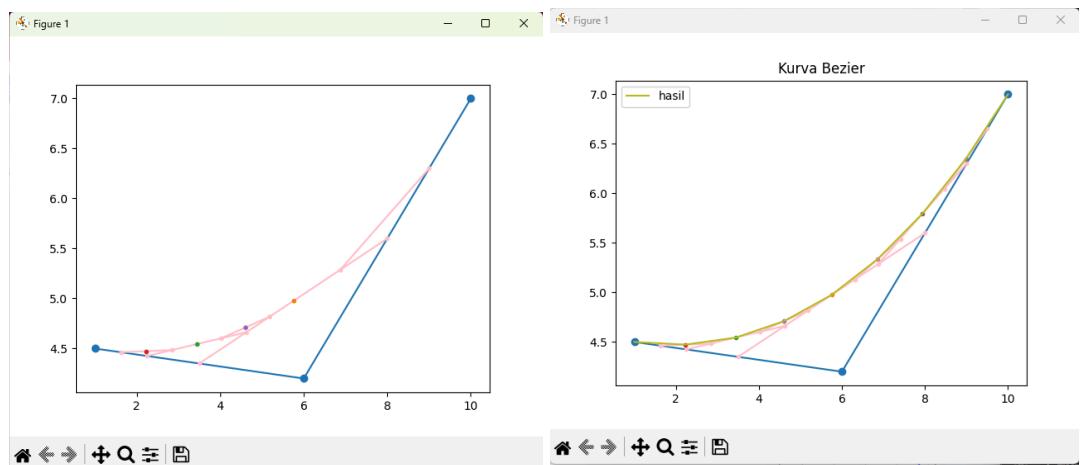
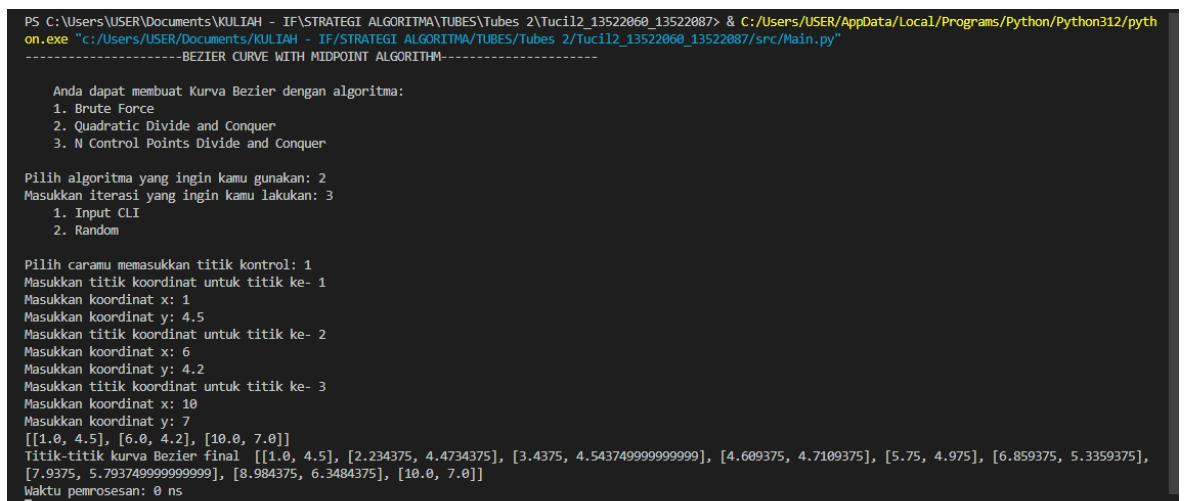
Gambar 14. Input untuk eksperimen 1 dengan brute force



Gambar 15. Eksperimen 1 dengan brute force



Gambar 16. Eksperimen 1 dengan algoritma divide and conquer untuk n titik kontrol



Gambar 17. Eksperimen 1 dengan divide and conquer untuk kurva Bézier kuadratik (tiga titik kontrol)

2. Eksperimen 2

Titik kontrol adalah (1,4.5) , (6, 4.2), dan (10, 7) dengan lima kali iterasi.

```
--BEZIER CURVE WITH MIDPOINT ALGORITHM--

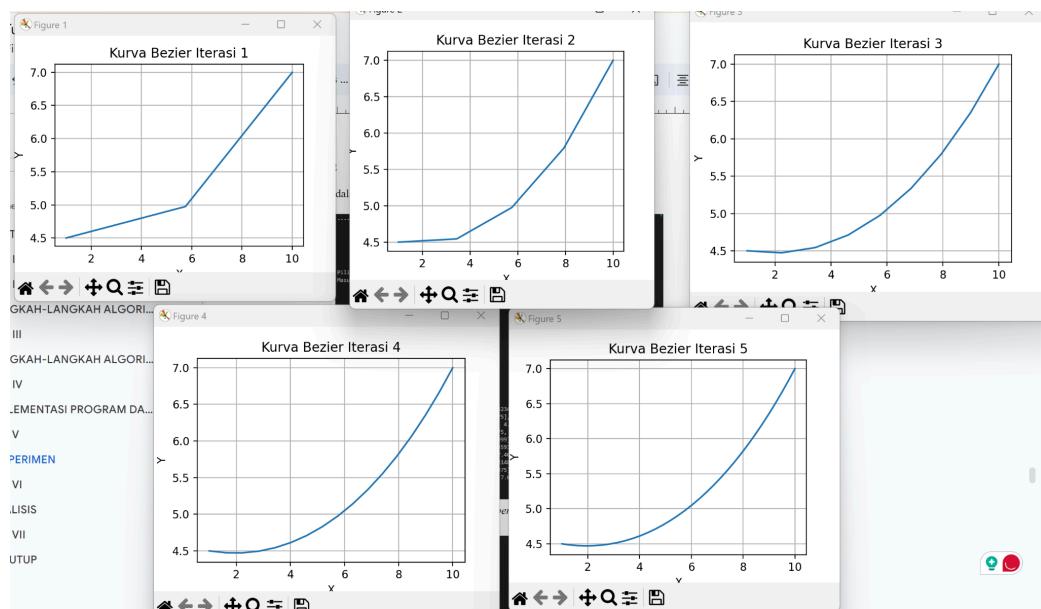
Anda dapat membuat Kurva Bezier dengan algoritma:
1. Brute Force
2. Quadratic Divide and Conquer
3. N Control Points Divide and Conquer

Pilih algoritma yang ingin kamu gunakan: 1
Masukkan iterasi yang ingin kamu lakukan: 5

Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih cara kamu memasukkan titik kontrol: 1
Masukkan titik koordinat untuk titik ke- 1
Masukkan koordinat x: 1
Masukkan koordinat y: 4.5
Masukkan titik koordinat untuk titik ke- 2
Masukkan koordinat x: 6
Masukkan koordinat y: 4.2
Masukkan titik koordinat untuk titik ke- 3
Masukkan koordinat x: 10
Masukkan koordinat y: 7
[[1.0, 4.5], [6.0, 4.2], [10.0, 7.0]]
Titik-titik kurva Bezier final [[1.0, 4.5], [1.3115234375, 4.48427734375], [1.62109375, 4.474609375], [1.9287109375, 4.47099609375], [2.234375, 4.4734375], [2.5380859375, 4.48193359375], [2.83984375, 4.496484375], [3.1396484375, 4.517688984375], [3.4375, 4.543749999999999], [3.7333984375, 4.576464843749999], [4.02734375, 4.615234374999999], [4.3193359375, 4.66005859375], [4.609375, 4.7109375], [4.8974609375, 4.76787109375], [5.18359375, 4.830859374999999], [5.4677734375, 4.89990234375], [5.75, 4.975], [6.0302734375, 5.05615234375], [6.30859375, 5.143359374999999], [6.5849609375, 5.23662109375], [6.859375, 5.3359375], [7.1318359375, 5.44130859375], [7.40234375, 5.552734374999999], [7.6708984375, 5.670214843749999], [7.9375, 5.793749999999999], [8.2021484375, 5.92333984375], [8.46484375, 6.058984375], [8.7255859375, 6.20068359375], [8.984375, 6.3484375], [9.2412109375, 6.50224609375], [9.49609375, 6.662109375], [9.7490234375, 6.82802734375], [10.0, 7.0]]
Waktu pemrosesan: 12047000 ns
```

Gambar 18. Input untuk eksperimen 2 dengan brute force



Gambar 19. Eksperimen 2 dengan brute force

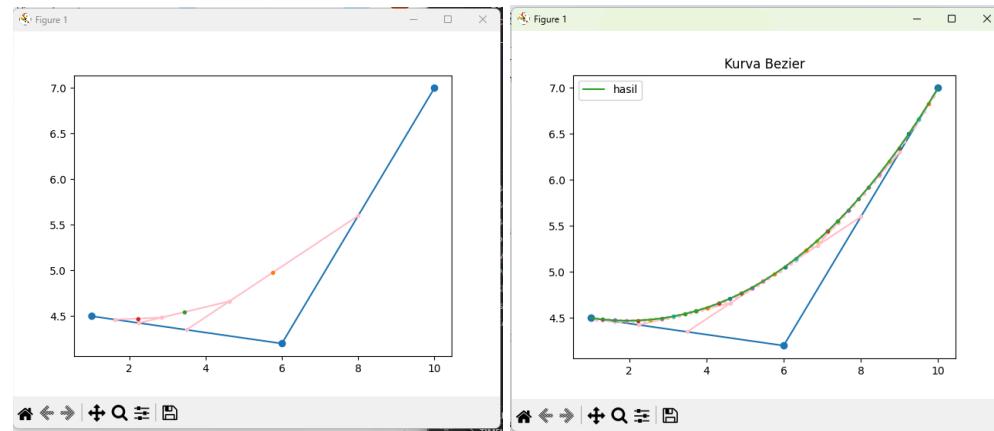
```

Masukkan iterasi yang ingin kamu lakukan: 5
Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih caramu memasukkan titik kontrol: 1
Masukkan titik koordinat untuk titik ke-1
Masukkan koordinat x: 1
Masukkan koordinat y: 4.5
Masukkan titik koordinat untuk titik ke-2
Masukkan koordinat x: 6
Masukkan koordinat y: 4.2
Masukkan titik koordinat untuk titik ke-3
Masukkan koordinat x: 10
Masukkan koordinat y: 7
[[1.0, 4.5], [6.0, 4.2], [10.0, 7.0]]
Titik-titik kurva Bezier final [[1.0, 4.5], [1.3115234375, 4.48427734375], [1.62109375, 4.474609375], [1.9287109375, 4.47099609375], [2.234375, 4.4734375], [2.5380859375, 4.48193359375], [2.83984375, 4.496484375], [3.1396484375, 4.51708894375], [3.4375, 4.543749999999999], [3.7333984375, 4.576464843749999], [4.02734375, 4.615234374999999], [4.319359375, 4.66005859375], [4.609375, 4.7109375], [4.8974609375, 4.76787109375], [5.18359375, 4.830859374999999], [5.4677734375, 4.89990234375], [5.75, 4.975], [6.0302734375, 5.05615234375], [6.30859375, 5.14335937499999], [6.5849609375, 5.23662109375], [6.859375, 5.3359375], [7.1318359375, 5.44130859375], [7.40234375, 5.55273437499999], [7.6708984375, 5.670214843749999], [7.9375, 5.79374999999999], [8.2021484375, 5.92333984375], [8.46484375, 6.058984375], [8.7255859375, 6.20068359375], [8.984375, 6.3484375], [9.2412109375, 6.50224609375], [9.49609375, 6.662109375], [9.7490234375, 6.828802734375], [10.0, 7.0]]
Waktu pemrosesan: 20228200 ns

```

Gambar 20. Input dan Output CLI untuk eksperimen 2 dengan Algoritma Divide and Conquer Kurva Bézier Kuadratik



Gambar 21. Output Visualisasi Bertahap dan Akhir untuk eksperimen 2 dengan Algoritma Divide and Conquer Kurva Bézier Kuadratik

3. Eksperimen 3

Titik kontrol adalah random dengan empat kali iterasi.

```

-----BEZIER CURVE WITH MIDPOINT ALGORITHM-----
Anda dapat membuat Kurva Bezier dengan algoritma:
1. Brute Force
2. Quadratic Divide and Conquer
3. N Control Points Divide and Conquer

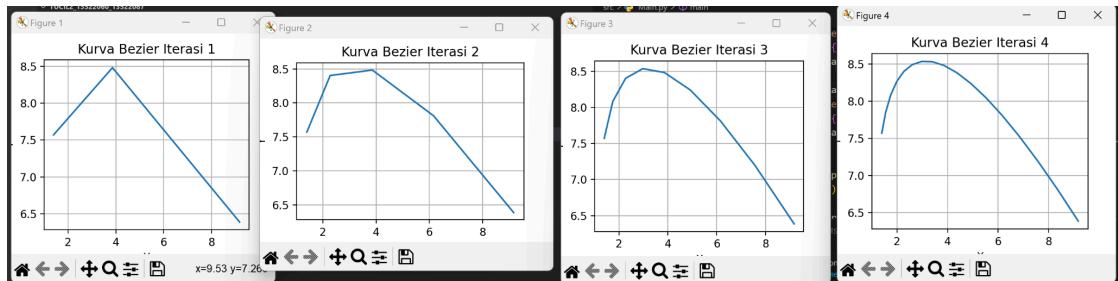
Pilih algoritma yang ingin kamu gunakan: 1
Masukkan iterasi yang ingin kamu lakukan: 4

Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih caramu memasukkan titik kontrol: 2
[[1.3974488504446345, 7.570884195965596], [2.434527201504997, 9.987347056123983], [9.170449345562226, 6.38629337855994]]
Titik-titik kurva Bezier final [[1.3974488504446345, 7.570884195965596], [1.5493447528935738, 7.849436129989075], [1.7457628724753012, 8.080976215103263], [1.986703209188165, 8.265564452745663], [2.27216576303712, 8.403820842437108], [2.6021505340172113, 8.493525384177598], [2.9766575221300806, 8.537018077967133], [3.395686727375768, 8.533498923805713], [3.8592381497542134, 8.482967921693335], [4.367311782654565, 8.385425071630003], [4.91998764590488, 8.240678373615717], [5.517025719686307, 8.049308276504761], [6.1586660105959155, 7.81072543734279], [6.84482851863811, 7.525135191867127], [7.57513423813495, 7.1925310204902], [8.350720186121467, 6.812919164279958], [9.170449345562226, 6.38629337855994]]
Waktu pemrosesan: 13470800 ns

```

Gambar 22. Input untuk eksperimen 3 dengan brute force



Gambar 23. Eksperimen 3 dengan brute force

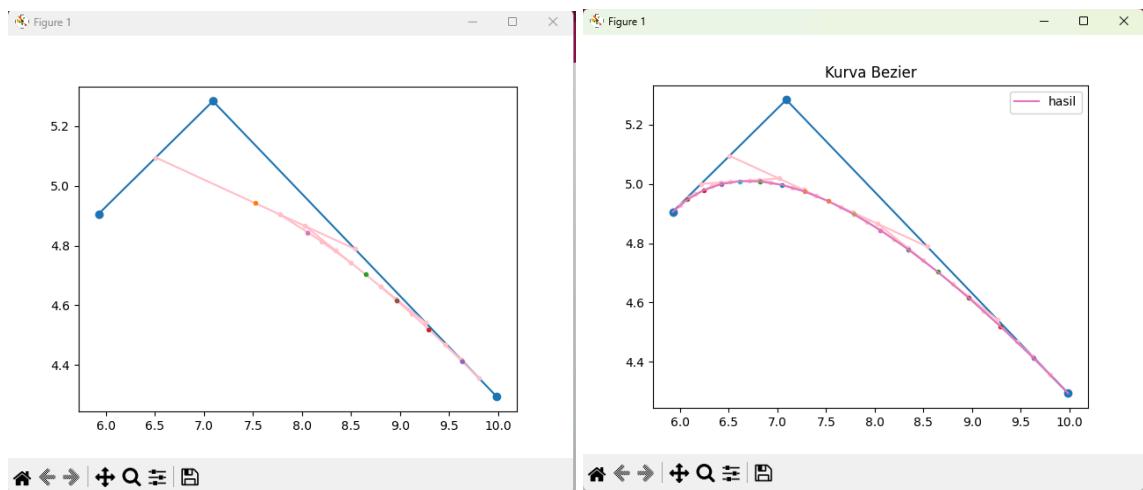
```
--BEZIER CURVE WITH MIDPOINT ALGORITHM-----
Anda dapat membuat Kurva Bezier dengan algoritma:
1. Brute Force
2. Quadratic Divide and Conquer
3. N Control Points Divide and Conquer

Pilih algoritma yang ingin kamu gunakan: 2
Masukkan iterasi yang ingin kamu lakukan: 4

Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih cara kamu memasukkan titik kontrol: 2
[[9.990500429641338, 4.29310083417419], [7.091262432360243, 5.284145525126286], [5.92375659161681, 4.906000345348873]]
Titik-titik kurva Bezier final [[9.990500429641338, 4.29310083417419], [9.634860258717676, 4.41163302618665], [9.292749245266965, 4.519468415182065], [8.9641673892895, 4.616687011888392], [8.649114690784394, 4.7030488127296435], [8.347591149752535, 4.778793817705822], [8.059596766193625, 4.843842026816924], [7.785131540107667, 4.898193440062954], [7.524195471494659, 4.941848057443996], [7.276788560354601, 4.97488587895979], [7.042910806687494, 4.9970669046105956], [6.8225621049337, 5.008631134396327], [6.615742771772131, 5.0094985683169851], [6.422452498523875, 4.999669206372569], [6.242691366748569, 4.979143048563078], [6.076459400446215, 4.947920094888512], [5.92375659161681, 4.906000345348873]]
Waktu pemrosesan: 0 ns
```

Gambar 24. Input dan Output CLI untuk eksperimen 3 dengan Algoritma Divide and Conquer Kurva Bézier Kuadratik



Gambar 25. Output Visualisasi Bertahap dan Akhir untuk eksperimen 3 dengan Algoritma Divide and Conquer Kurva Bézier Kuadratik

4. Eksperimen 4

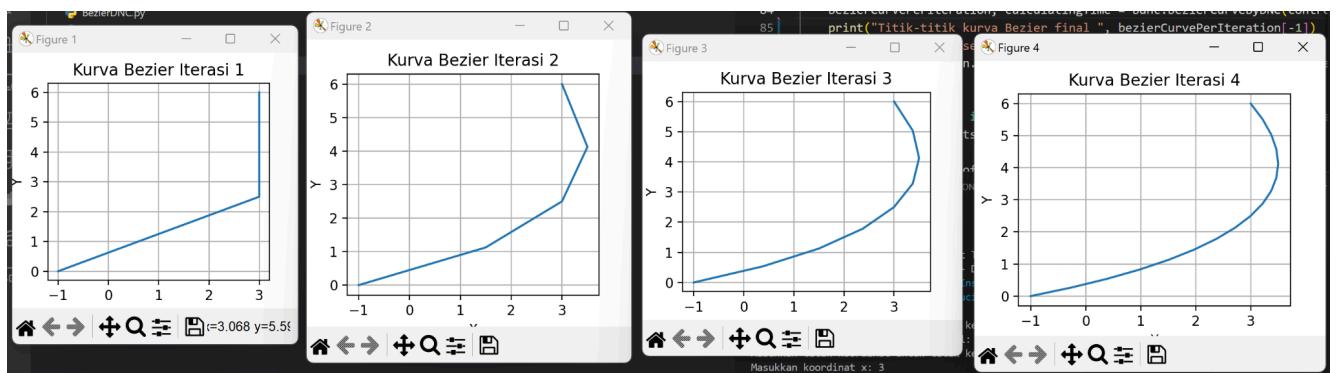
Titik kontrol adalah $(3, 6)$, $(5, 2)$, dan $(-1, 0)$ dengan empat kali iterasi.

```

Masukkan titik koordinat untuk titik ke- 1
Pilih caramu memasukkan titik kontrol: 1
Masukkan titik koordinat untuk titik ke- 1
Masukkan koordinat x: 3
Masukkan koordinat y: 6
Masukkan titik koordinat untuk titik ke- 2
Masukkan koordinat x: 5
Masukkan koordinat y: 2
Masukkan titik koordinat untuk titik ke- 3
Masukkan koordinat x: -1
Masukkan koordinat y: 0
[[3.0, 6.0], [5.0, 2.0], [-1.0, 0.0]]
Titik-titik kurva Bezier final [[3.0, 6.0], [3.21875, 5.5078125], [3.375, 5.03125], [3.46875, 4.5703125], [3.5, 4.125], [3.46875, 3.6953125], [3.375, 3.28125], [3.21875, 2.8828125], [3.0, 2.5], [2.71875, 2.1328125], [2.375, 1.78125], [1.96875, 1.4453125], [1.5, 1.125], [0.96875, 0.8203125], [0.375, 0.53125], [-0.28125, 0.2578125], [-1.0, 0.0]]
Waktu pemrosesan: 15643100 ns

```

Gambar 26. Input untuk eksperimen 4 dengan brute force



Gambar 23. Eksperimen 4 dengan brute force

```

-----BEZIER CURVE WITH MIDPOINT ALGORITHM-----
Anda dapat membuat Kurva Bezier dengan algoritma:
1. Brute Force
2. Quadratic Divide and Conquer
3. N Control Points Divide and Conquer

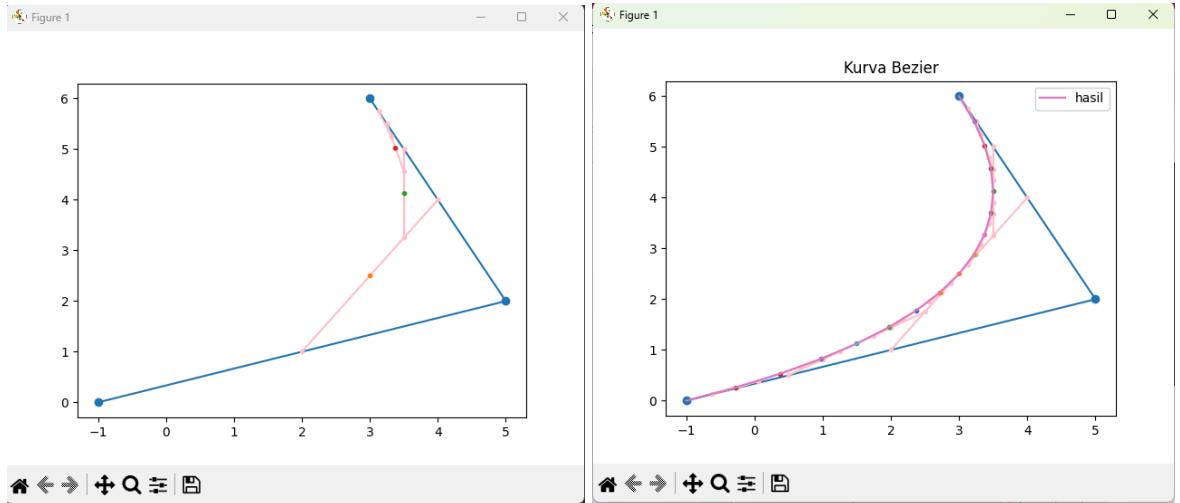
Pilih algoritma yang ingin kamu gunakan: 2
Masukkan iterasi yang ingin kamu lakukan: 4

Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih caramu memasukkan titik kontrol: 1
Masukkan titik koordinat untuk titik ke- 1
Masukkan koordinat x: 3
Masukkan koordinat y: 6
Masukkan titik koordinat untuk titik ke- 2
Masukkan koordinat x: 5
Masukkan koordinat y: 2
Masukkan titik koordinat untuk titik ke- 3
Masukkan koordinat x: -1
Masukkan koordinat y: 0
[[3.0, 6.0], [5.0, 2.0], [-1.0, 0.0]]
Titik-titik kurva Bezier final [[3.0, 6.0], [3.21875, 5.5078125], [3.375, 5.03125], [3.46875, 4.5703125], [3.5, 4.125], [3.46875, 3.6953125], [3.375, 3.28125], [3.21875, 2.8828125], [3.0, 2.5], [2.71875, 2.1328125], [2.375, 1.78125], [1.96875, 1.4453125], [1.5, 1.125], [0.96875, 0.8203125], [0.375, 0.53125], [-0.28125, 0.2578125], [-1.0, 0.0]]
Waktu pemrosesan: 0 ns

```

Gambar 24. Input dan Output CLI untuk eksperimen 4 dengan Algoritma Divide and Conquer Kurva Bézier Kuadratik



Gambar 25. Output Visualisasi Bertahap dan Akhir untuk eksperimen 4 dengan Algoritma Divide and Conquer Kurva Bézier Kuadratik

5. Eksperimen 5

Titik kontrol adalah (9, 5), (2, 1), dan (-4, 3) dengan tujuh kali iterasi. Sebelumnya memberikan input titik (9, #) yang meminta kembali validasi

```
--BEZIER CURVE WITH MIDPOINT ALGORITHM-----
Anda dapat membuat Kurva Bézier dengan algoritma:
1. Brute Force
2. Quadratic Divide and Conquer
3. N Control Points Divide and Conquer

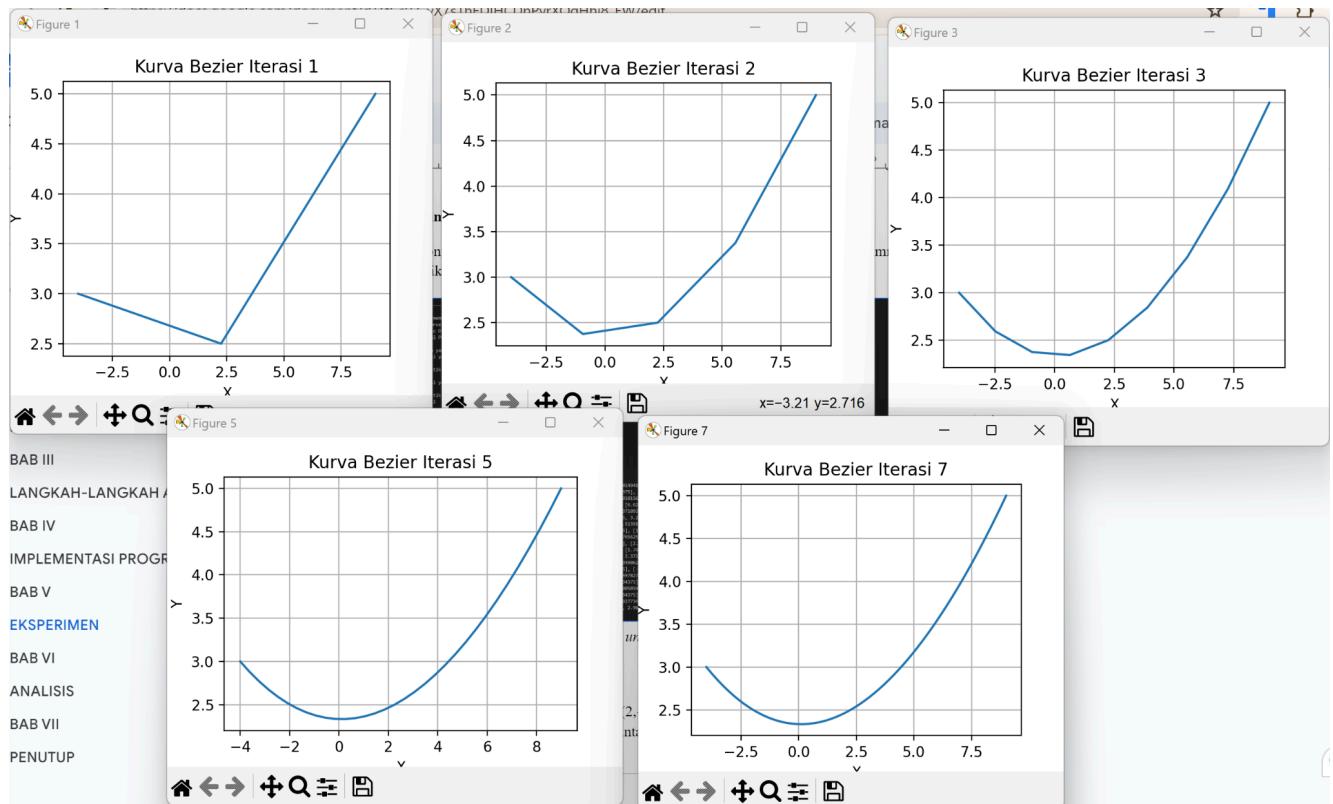
Pilih algoritma yang ingin kamu gunakan: 1
Masukkan iterasi yang ingin kamu lakukan: 7

Masukkan titik kontrol dengan:
mu gunakan: 1
Masukkan iterasi yang ingin kamu lakukan: 7

Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih caramu memasukkan titik kontrol: 1
Masukkan titik koordinat untuk titik ke- 1
~~~~~titik~~~~~
KeyboardInterrupt
Masukkan koordinat x: 2
Masukkan koordinat y: 1
Masukkan titik koordinat untuk titik ke- 3
Masukkan koordinat x: -4
Masukkan koordinat y: 3
[[9.0, 5.0], [2.0, 1.0], [-4.0, 3.0]]
Titik-titik kurva Bézier final [[9.0, 5.0], [8.89068603515625, 4.9378662189375], [8.781494140625, 4.87646484375], [8.6724231640625, 4.817958984375], [8.5634765625, 4.755895375], [8.45465987899625, 4.696652734375], [8.3459472565625, 4.63818359375], [8.23736572565625, 4.58044339375], [8.12896625, 4.5234375], [8.0205584765625, 4.4671638859375], [7.912351515625, 4.41162109375], [7.80426025390625, 4.3568115234375], [7.6952896625, 4.302734375], [7.58843994140625, 4.249389646484375], [7.480712899625, 4.1448974609375], [7.37318791015625, 4.09375], [7.15826416015625, 4.0433349689375], [7.051025390625, 3.99365234375], [6.94398869140625, 3.9458984375], [6.80426025390625, 3.896625484375], [6.662291815625, 3.80224699375], [6.516662597365625, 3.7562255859375], [6.41015625, 3.7109375], [6.30377197265625, 3.6663818359375], [6.197599765625, 3.62255859375], [6.09136962890625, 3.5794677734375], [5.98531515625, 3.537109375], [5.87945556640625, 3.495483984375], [5.773681640625, 3.45458984375], [5.66802978515625, 3.4144287109375], [5.5625, 3.375], [5.45769228515625, 3.3363037109375], [5.3158664640625, 3.29833984375], [5.24664386640625, 3.2611883984375], [5.1416015625, 3.2246699375], [5.03668212890625, 3.1888427734375], [4.931884765625, 3.15388859375], [4.82720947265625, 3.1056015625, 2.84375], [3.7871041015625, 2.8167724609375], [3.683837898625, 2.79052734375], [3.5862744140625, 2.7659146484375], [3.4775390625, 2.740234375], [3.37457275398625, 2.7161865234375], [3.271728515625, 2.69287109375], [3.1690634765625, 3.051005859375], [4.49972900398625, 3.0209609375], [4.40972900398625, 2.9896240234375], [4.3056640625, 2.958984375], [4.20172119140625, 2.9298771484375], [4.097906309625, 2.89990234375], [3.99420166015625, 2.8714599609375], [3.896625, 2.84375], [3.7871041015625, 2.8167724609375], [3.683837898625, 2.79052734375], [3.5862744140625, 2.7659146484375], [3.4775390625, 2.740234375], [3.37457275398625, 2.7161865234375], [3.271728515625, 2.69287109375], [3.1690634765625, 3.051005859375], [2.066933359375], [2.75933837898625, 2.5859375], [2.6572265625, 2.568359375], [2.55523681640625, 2.5501708984375], [2.453369140625, 2.53271484375], [2.3516235315625, 2.5159912109375], [2.25, 2.5], [2.1484985315625, 2.4847412109375], [2.047119140625, 2.47921484375], [1.94586181640625, 2.4564208994375], [1.8447265625, 2.443359375], [1.74371337890625, 2.4310382734375], [1.64282265625, 2.41943359375], [1.54205322265625, 2.4085693359375], [1.44146625, 2.3984375], [1.340881745625, 2.389388859375], [1.240478515625, 2.38037109375], [1.140197730625, 2.3723465234375], [1.0400390625, 2.365234375], [0.9400244140625, 2.35302734375], [0.74029541015625, 2.3480224609375], [0.646825, 2.34375], [0.54107666015625, 2.340209609375], [0.441658039625, 2.33740234375], [0.34234619140625, 2.3335271484375], [0.14414040390625, 2.3333740234375], [0.045166015625, 2.3334990234375], [0.05364990234375, 2.3343505859375], [0.-0.15234375, 2.3339975], [0.-0.25091552734375, 2.3382568359375], [0.-0.349365234375, 2.34138659375], [0.-0.44769287109375, 2.3458927734375], [0.-0.5458894375, 2.349], [0.609375, 2.41015625, 2.343984375], [0.513916015625, 2.33683984375], [0.-0.419433359375, 2.33675537109375], [0.-0.39375, 2.375], [1.-0.03598521484375, 2.393288984375], [1.-1.22991343359375, 2.4017333984375], [0.4017333984375, 2.412109375], [1.-1.22991343359375, 2.423217734375], [1.-1.521240234375, 2.43505859375], [1.-1.61810392734375, 2.4476138359375], [1.-1.71484375, 2.4699375], [1.-1.81162048034375, 2.4749755859375], [1.-1.90758984375, 2.48974609375], [0.-0.04333349609375, 2.5052409234375], [0.-2.19671630859375, 2.5384521484375], [0.-2.292724609375, 2.55615234375], [0.-2.38861083984375, 2.5745849699375], [0.-2.484375, 2.59375], [0.-2.58001708984375, 2.6136474609375], [0.-2.675537109375, 2.63427734375], [0.-2.77093505859375, 2.65656396484375], [0.-2.8662109375, 2.677734375], [0.-2.96136474609375, 2.7005615234375], [0.-3.056396484375, 2.74212109375], [0.-3.15136015234375, 2.7484130859375], [0.-3.24609375, 2.7734375], [0.-3.4075927734375, 2.7991943359375], [0.-3.435302734375, 2.82568359375], [0.-3.52972412109375, 2.8529052734375], [0.-3.6240234375, 2.880859375], [0.-3.71820668359375, 2.9095458894375], [0.-3.812255859375, 2.93896484375], [0.-3.90618896484375, 2.9691162109375], [0.-4.0, 3.0]]
```

Gambar 26. Input untuk eksperimen 5 dengan brute force



Gambar 27. Eksperimen 5 dengan brute force

```
--BEZIER CURVE WITH MIDPOINT ALGORITHM--

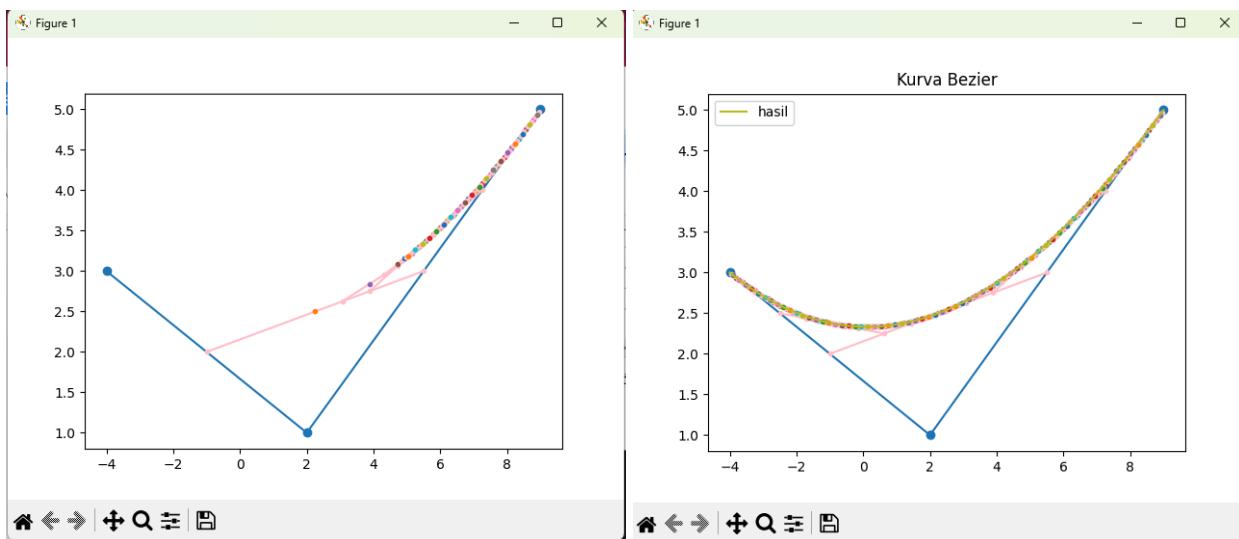
Anda dapat membuat Kurva Bezier dengan algoritma:
1. Brute Force
2. Quadratic Divide and Conquer
3. N Control Points Divide and Conquer

Pilih algoritma yang ingin kamu gunakan: 2
Masukkan iterasi yang ingin kamu lakukan: 7

Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih caramu memasukkan titik kontrol: 1
Masukkan titik koordinat untuk titik ke- 1
Masukkan koordinat x: 9
Masukkan koordinat y: #
Input harus berupa bilangan real. Coba lagi.
Masukkan koordinat y: 5
Masukkan titik koordinat untuk titik ke- 2
5, 2.443359375, [1.7437133789625, 2.4310302734375], [1.642822265625, 2.4085693359375], [1.44140625, 2.3984375], [1.34088134765625, 2.389038859375], [1.240478515625, 2.38037109375], [1.14019775390625, 2.3724365234375], [1.04000390625, 2.365234375], [0.94000244140625, 2.3587646484375], [0.8400087890625, 2.35802734375], [0.74029541015625, 2.3488224609375], [0.640625, 2.34375], [0.5410766615625, 2.3402899609375], [0.44165039625, 2.33740234375], [0.34234619140625, 2.3353271484375], [0.2431640625, 2.333984375], [0.14410400390625, 2.3333740234375], [0.045166015625, 2.33349609375], [-0.05364990234375], [0.34343505859375], [-0.15234375, 2.3359375], [-0.25091552734375, 2.3382568359375], [-0.349365234375, 2.34130859375], [-0.44769287109375, 2.34508927734375], [-0.5458984375, 2.349609375], [-0.64398193359375, 2.3548583994375], [-0.741943359375, 2.36083984375], [-0.83978271484375, 2.3675537109375], [-0.9375, 2.375], [-1.03509521484375, 2.3831787109375], [-1.132568359375, 2.39208984375], [-1.22991943359375, 2.4017333984375], [-1.3271484375, 2.412109375], [-1.42425537109375, 2.423217734375], [-1.521240234375, 2.43505859375], [-1.61810302734375, 2.4476318359375], [-1.71484375, 2.4609375], [-1.81146240234375, 2.4749755859375], [-1.907958984375, 2.48974609375], [-2.00433349609375, 2.5052496234375], [-2.1095859375, 2.521484375], [-2.19671630859375, 2.5384521484375], [-2.292724609375, 2.55615234375], [-2.38861083984375, 2.5745849609375], [-2.484375, 2.59375], [-2.58001708984375, 2.6136474609375], [-2.675537109375, 2.63427734375], [-2.77093505859375, 2.6556396484375], [-2.8662109375, 2.677734375], [-2.96136474609375, 2.7085615234375], [-3.05369484375, 2.72412109375], [-3.15138615234375, 2.7484130859375], [-3.24669375, 2.7734375], [-3.34075927734375, 2.7991943359375], [-3.435302734375, 2.82568359375], [-3.52972412109375, 2.8529652734375], [-3.6240234375, 2.880859375], [-3.71820068359375, 2.9095458984375], [-3.812255859375, 2.9386484375], [-3.98618896484375, 2.9691162109375], [-4.0, 3.0]]
Waktu pemrosesan: 17734988 ns
```

Gambar 28. Input dan Output CLI untuk eksperimen 5 dengan Algoritma Divide and Conquer Kurva Bézier Kuadratik



Gambar 29. Output Visualisasi Bertahap dan Akhir untuk eksperimen 5 dengan Algoritma Divide and Conquer Kurva Bézier Kuadratik

6. Eksperimen 6

Titik kontrol adalah (8,5), (-9,10), dan (2,-1) dengan 3 kali iterasi. Sebelumnya memberikan input A kali iterasi dan -3 kali iterasi yang meminta kembali validasi.

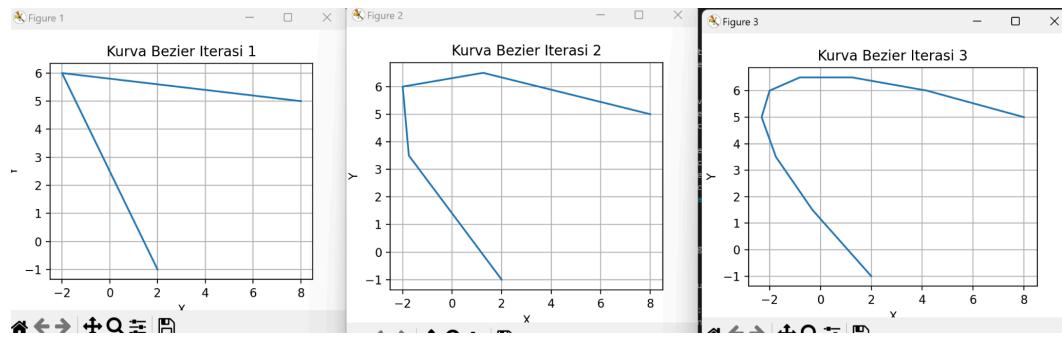
```
-----BEZIER CURVE WITH MIDPOINT ALGORITHM-----
Anda dapat membuat Kurva Bézier dengan algoritma:
1. Brute Force
2. Quadratic Divide and Conquer
3. N Control Points Divide and Conquer

Pilih algoritma yang ingin kamu gunakan: 1
Masukkan iterasi yang ingin kamu lakukan: A
Input harus berupa bilangan bulat. Coba lagi.
Masukkan iterasi yang ingin kamu lakukan: -3
Jumlah iterasi harus lebih besar dari atau sama dengan 0!
Masukkan iterasi yang ingin kamu lakukan: 3

Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih caramu memasukkan titik kontrol: 1
Masukkan titik koordinat untuk titik ke- 1
Masukkan koordinat x: 8
Masukkan koordinat y: 5
Masukkan titik koordinat untuk titik ke- 2
Masukkan koordinat x: -9
Masukkan koordinat y: 10
Masukkan titik koordinat untuk titik ke- 3
Masukkan koordinat x: 2
Masukkan koordinat y: -1
[[8.0, 5.0], [-9.0, 10.0], [2.0, -1.0]]
Titik-titik kurva Bézier final [[8.0, 5.0], [4.1875, 6.0], [1.25, 6.5], [-0.8125, 6.5], [-2.0, 6.0], [-2.3125, 5.0], [-1.75, 3.5], [-0.3125, 1.5], [2.0, -1.0]]
Waktu pemrosesan: 0 ns
[]
```

Gambar 30. Input untuk eksperimen 6 dengan brute force



Gambar 31. Eksperimen 6 dengan brute force

```
-----BEZIER CURVE WITH MIDPOINT ALGORITHM-----
Anda dapat membuat Kurva Bezier dengan algoritma:
1. Brute Force
2. Quadratic Divide and Conquer
3. N Control Points Divide and Conquer

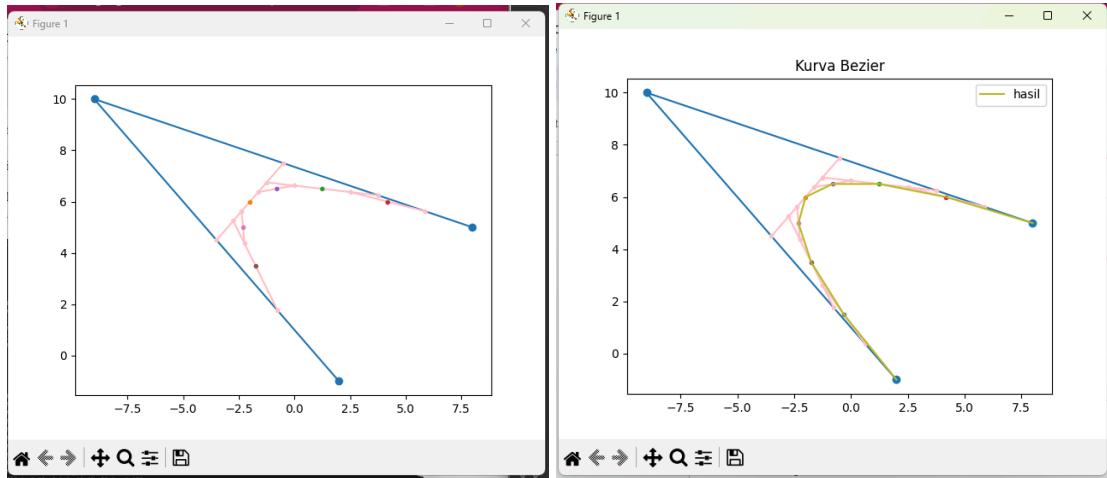
Pilih algoritma yang ingin kamu gunakan: 2
Masukkan iterasi yang ingin kamu lakukan: A
Input harus berupa bilangan bulat. Coba lagi.
Masukkan iterasi yang ingin kamu lakukan: -3
Jumlah iterasi harus lebih besar dari atau sama dengan 0!
Masukkan iterasi yang ingin kamu lakukan: 3

Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih caramu memasukkan titik kontrol: 1
Masukkan titik koordinat untuk titik ke- 1
Masukkan koordinat x: 8
Masukkan koordinat y: 5
Masukkan titik koordinat untuk titik ke- 2
Masukkan koordinat x: -9
Masukkan koordinat y: 10
Masukkan titik koordinat untuk titik ke- 3
Masukkan koordinat x: 2
Masukkan koordinat y: -1

Titik-titik kurva Bezier final [[8.0, 5.0], [4.1875, 6.0], [1.25, 6.5], [-0.8125, 6.5], [-2.0, 6.0], [-2.3125, 5.0], [-1.75, 3.5], [-0.3125, 1.5], [2.0, -1.0]]
Waktu pemrosesan: 0 ns
```

Gambar 32. Input dan Output CLI untuk eksperimen 6 dengan Algoritma Divide and Conquer Kurva Bézier Kuadratik



Gambar 33. Output Visualisasi Bertahap dan Akhir untuk eksperimen 6 dengan Algoritma Divide and Conquer Kurva Bézier Kuadratik

7. Eksperimen 7

Titik kontrol random berjumlah empat untuk 4 kali iterasi menghasilkan kurva Bézier kubik.

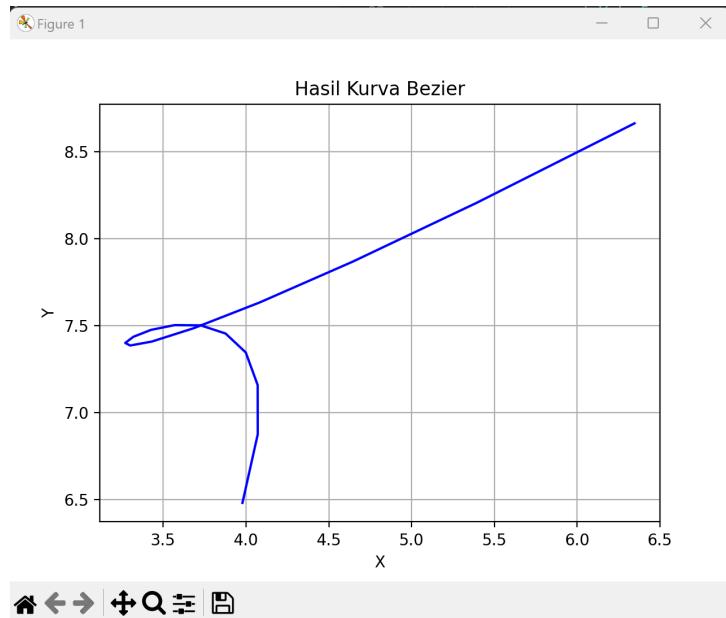
```
-----BEZIER CURVE WITH MIDPOINT ALGORITHM-----
Anda dapat membuat Kurva Bezier dengan algoritma:
1. Brute Force
2. Quadratic Divide and Conquer
3. N Control Points Divide and Conquer

Pilih algoritma yang ingin kamu gunakan: 3
Masukkan iterasi yang ingin kamu lakukan: 4

Masukkan titik kontrol dengan:
1. Input CLI
2. Random

Pilih caramu memasukkan titik kontrol: 2
Berapa kontrol poin yang ingin kamu gunakan? 4
Titik kontrol anda adalah [[6.3473177483724355, 8.662910053892562], [0.6512802289402141, 5.870149976832924], [4.757954037556466, 8.91180549818174], [3.270103090186926, 7.400882405628607], [3.319161372319411, 7.436349980874012], [3.425833176864889, 7.476852207760042], [3.5686013132888414, 7.503454889893315], [3.7259485910252312, 7.501987539680446], [3.8763578195557002, 7.455088742328051], [3.998311808329971, 7.346197083042746], [4.070293366805764, 7.1587511471], [4.07078530440803, 6.876189519499869], [3.978270430692809, 6.481950785655531]]
Waktu pemrosesan: 16279100 ns
```

Gambar 34. Input eksperimen 7 dengan algoritma divide and conquer untuk n titik kontrol

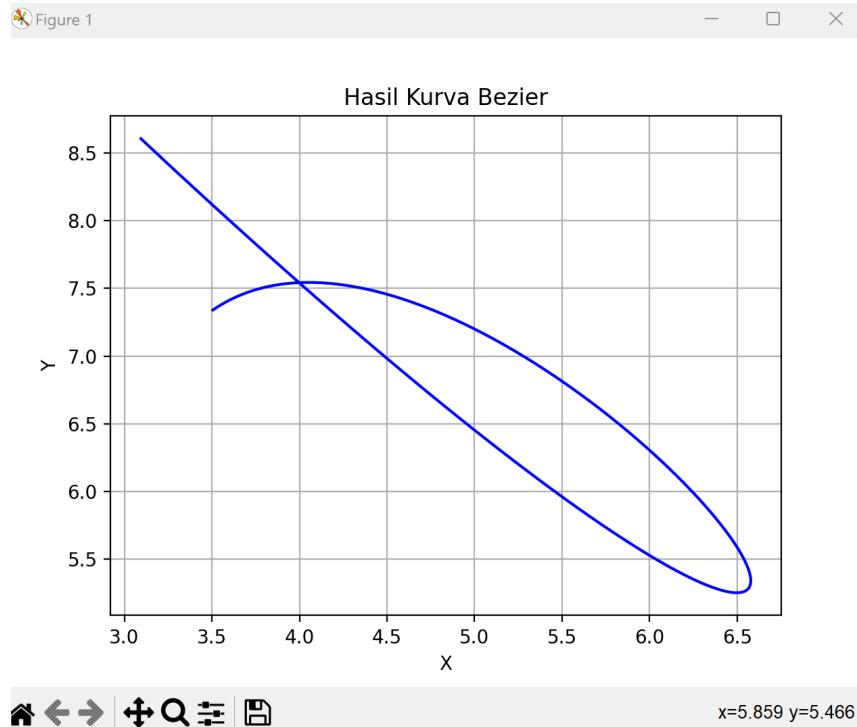


Gambar 35. Hasil eksperimen 7 dengan algoritma divide and conquer untuk n titik kontrol

8. Eksperimen 8

Titik kontrol random berjumlah lima untuk 7 kali iterasi.

Gambar 36. Input eksperimen 7 dengan algoritma divide and conquer untuk n titik kontrol



Gambar 37. Hasil eksperimen 8 dengan algoritma divide and conquer untuk n titik kontrol

BAB VI

ANALISIS

1. Analisis Algoritma Brute Force

Fungsi `findMidpoint` memiliki kompleksitas dengan empat kali perkalian dan dua kali penambahan apapun titiknya sehingga memiliki kompleksitas ($T(\text{findMidpoint})$) sebesar 1 dengan notasi big-O yaitu $O(1)$. Dalam proses iterasi untuk mendapatkan titik-titik yang akan dilalui oleh kurva Bézier, implementasi algoritma brute force akan Fungsi `getMidpoints` melakukan iterasi sebanyak titik kontrol yang masuk menjadi parameter. Misalkan banyaknya titik kontrol masukan parameter adalah p , maka $T(\text{getMidpoints})$ adalah c . Fungsi `getBézierMidpoints` melakukan iterasi sebanyak hasil dari fungsi `getMidpoints` kali. Karena fungsi `getMidpoints` akan menghasilkan $c - 1$ titik, maka $T(\text{getBézierMidpoints})$ adalah $c - 1$. Fungsi `insertPoints` akan menambahkan seluruh titik hasil fungsi ke dalam senarai yang berisi hasil iterasi dari iterasi. Fungsi akan melakukan proses konkatenasi sebanyak hasil dari fungsi `getBézierMidpoints`, yaitu sebanyak $c - 2$ kali. Sehingga, kompleksitas waktu proses `insertPoints` adalah $c - 2$. Kemudian, akan dilakukan proses penambahan hasil `insertPoints` ke dalam senarai `resultPerIteration` yang berarti memiliki kompleksitas waktu $c - 2$ pula. Kemudian proses iterasi dilanjutkan dengan memanggil kembali fungsi `insertPoints` yang kompleksitas waktunya $c - 2$. Sehingga, dalam satu kali iterasi, kompleksitas waktunya :

$$T(\text{iterasi}) = 1 + (c - 1) + (c - 1) + (c - 2) + (c - 2) + (c - 2) + (c - 2)$$

$$T(\text{iterasi}) = 6c - 9$$

Fungsi `getBruteForceBézier` melakukan iterasi sebanyak jumlah iterasi yang diinginkan yaitu suatu n kali. Generalisasi untuk banyaknya titik kontrol yang digunakan adalah N dan banyaknya iterasi yang dilakukan adalah N .

$$T(\text{iterasi}) = O(N)$$

Setiap iterasi memanggil fungsi `getMidpoints`, `getBézierMidpoints`, `insertPoints` sebanyak dua kali sehingga kompleksitas setiap iterasi adalah $O(N) + O(N) + 2*O(N) = O(N)$. Sehingga kompleksitas untuk membuat kurva Bézier dengan algoritma *brute force* adalah $O(N) * O(N)$ yaitu $O(N^2)$.

2. Analisis Algoritma Divide and Conquer Kurva Bézier Kuadratik

Fungsi `findMidpoint` terdiri dari dua kali proses penjumlahan dan dua kali proses pembagian apapun titiknya sehingga memiliki kompleksitas $O(1)$. Fungsi `BézierDNC` melakukan iterasi sebanyak jumlah iterasi yang diinginkan yaitu sebanyak N . Setiap iterasi memanggil fungsi `findMidpoint` sebanyak tiga kali (besar kompleksitas waktu proses ini $3 * O(1)$ yaitu $O(1)$). Kemudian, fungsi akan membagi titik-titik kontrol menjadi 3 bagian (bagian kiri, tengah, dan kanan) dan melakukan pemanggilan kembali fungsi `BézierDNC` (melakukan proses rekursi) sebanyak dua kali untuk titik kontrol bagian kiri dan kanan (besar kompleksitas waktu proses ini adalah $O(2^{\text{iterasi}})$). Setelah itu melakukan tiga kali proses

konkatenasi baik dengan fungsi append maupun extend). Oleh karena pada setiap rekursi, fungsi membagi dua permasalahan dan melakukan pemanggilan dua kali fungsi rekursi yang sama dengan, maka algoritma ini memiliki kompleksitas $O(2^N)$ dengan N adalah jumlah iterasi yang dilakukan.

Fungsi `BézierCurvebyDNC` akan menyatukan titik kontrol masukan pertama dan titik-titik yang dilewati kurva Bézier berdasarkan fungsi dengan fungsi extend dan titik kontrol masukan akhir dengan fungsi append. Proses ini memiliki kompleksitas waktu sebesar $O(1)$. Sehingga, keseluruhan proses pembentukan kurva Bézier dengan algoritma *divide and conquer* ini memiliki kompleksitas waktu sebesar $O(2^N)$.

3. Analisis Algoritma Divide and Conquer untuk N Titik Kontrol

Fungsi `getMostInnerMidpoint` melakukan iterasi sebanyak jumlah titik kontrol yang setiap iterasinya melakukan proses konkatenasi yang memiliki kompleksitas $O(1)$ dan `getMidpoints` yang memiliki kompleksitas $O(N)$ sehingga setiap pencarian titik tengah terdalam memiliki kompleksitas $O(N) * O(N)$ yaitu $O(N^2)$.

Fungsi *divide and conquer* yaitu `BézierDNCMulti` melakukan iterasi sebanyak jumlah iterasi yang diinginkan yaitu suatu n kali. Untuk setiap iterasi, fungsi ini memanggil fungsi `getMostInnerMidpoint` kompleksitas $O(N^2)$, melakukan tiga kali penambahan baik dengan fungsi append maupun extend, dan memanggil kembali fungsi dirinya sebanyak dua kali. Oleh karena pada setiap rekursi, fungsi membagi dua permasalahan dan melakukan pemanggilan dua kali fungsi rekursi yang sama dengan, maka algoritma ini memiliki kompleksitas $O(2^N) * O(N^2)$ atau $O(2^N * N^2)$.

4. Analisis Umum untuk Kurva Bézier Kuadratik

Berdasarkan analisis kompleksitas di atas, terlihat bahwa untuk pembentukan suatu kurva Bézier kuadratik, kompleksitas algoritma *brute force* yang dikembangkan penulis lebih baik daripada algoritma *divide and conquer* yang dikembangkan penulis. Hal tersebut tidak sesuai dengan hipotesis awal penulis di mana algoritma *divide and conquer* akan memiliki kompleksitas yang lebih baik. Hal ini, mungkin karena algoritma *brute force* yang dikembangkan penulis bukanlah algoritma *brute force* yang paling buruk kompleksitasnya (ada algoritma *brute force* lain yang mungkin lebih *brute force* dengan kompleksitas lebih buruk).

BAB VII

PENUTUP

5.1 Kesimpulan

Algoritma *divide and conquer* efektif untuk membuat kurva Bézier yang baik. Namun berdasarkan hasil eksperimen dan analisis, kompleksitas algoritma *brute force* untuk kasus kurva Bézier kuadratik lebih baik.

5.2 Pustaka

Bandara, R. (2011). Midpoint Algorithm (Divide and Conquer Method) for Drawing a Simple Bezier Curve.

<https://www.codeproject.com/Articles/223159/Midpoint-Algorithm-Divide-and-Conquer-Method-for-D> (Diakses pada 16 Maret 2024 pukul 12.31)

Rinaldi, M. (2021). Algoritma Divide and Conquer (2021) Bagian 1. Retrieved from

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-andConquer-\(2021\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-andConquer-(2021)-Bagian1.pdf). (Diakses pada 13 Maret 2024 at 15.41)

5.3 Link Repository

Link repository untuk tugas kecil 2 mata kuliah IF2211 Strategi Algoritma adalah sebagai berikut https://github.com/shulhajws/Tucil2_13522060_13522087

5.4 Tabel Checkpoint Program

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.	✓	