

LAPORAN TUGAS KECIL II
IF2211 STRATEGI ALGORITMA

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



Dosen:

Ir. Rila Mandala, M. Eng, Ph. D.

Monterico Adrian, S. T, M. T.

Kelompok 45:

13522140 Yasmin Farisah Salma

13522148 Auralea Alvinia Syaikha

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
SEMESTER II TAHUN 2023/2024

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa penulis ucapkan atas keberhasilan dalam menyelesaikan Tugas Kecil II IF2211 Strategi Algoritma pada Semester II tahun ajaran 2023/2024 ini.

Laporan ini merupakan dokumentasi dan penjelasan atas program dan algoritma yang telah penulis implementasikan. Penulis berusaha untuk merancang dan mengimplementasikan salgoritma *brute force* dan *divide and conquer* ini sedemikian rupa untuk membuat Kurva Bézier ini.

Penyelesaian tugas kecil ini tidak lepas dari bimbingan dan arahan yang diberikan oleh dosen pengampu mata kuliah Strategi Algoritma. Penulis juga ingin mengucapkan terima kasih kepada rekan-rekan penulis yang telah memberikan masukan, saran, dan inspirasi selama proses pengembangan algoritma ini. Selain itu, penulis juga mengapresiasi dukungan yang diberikan oleh asisten dan semua pihak yang telah membantu, baik secara langsung maupun tidak langsung, dalam penyelesaian tugas kecil ini. Penulis berharap bahwa *output* dari tugas kecil ini dapat memberikan kontribusi bagi pengembangan ilmu pengetahuan.

Penulis menyadari bahwa tugas kecil ini masih jauh dari sempurna. Oleh karena itu, penulis sangat terbuka untuk menerima kritik dan saran yang konstruktif demi perbaikan di masa yang akan datang. Akhir kata, semoga tugas kecil ini dapat bermanfaat bagi semua pihak yang berkepentingan.

Sumedang, 13 Maret 2024,

Penulis.

DAFTAR ISI

KATA PENGANTAR.....	1
DAFTAR ISI.....	2
BAB I	
DESKRIPSI TUGAS.....	4
1.1. Deskripsi Tugas.....	4
BAB II	
LANDASAN TEORI.....	8
2.1 Algoritma Brute Force.....	8
2.2 Algoritma Divide and Conquer.....	10
BAB III	
SOURCE CODE & ANALISIS IMPLEMENTASI ALGORITMA.....	13
3.1 Source Code & Analisis Implementasi Algoritma Divide and Conquer.....	13
3.2 Source Code & Analisis Implementasi Algoritma Brute Force.....	15
3.3 Main Program.....	17
3.4 Implementasi Bonus 1 (Kurva Bézier dengan n Points).....	19
3.5 Implementasi Bonus 2 (Visualisasi Proses Pembentukan Kurva Bézier).....	20
BAB IV	
HASIL UJI COBA.....	22
4.1 Hasil Uji Coba Algoritma Divide and Conquer.....	22
4.2 Hasil Uji Coba Algoritma Brute Force.....	24

4.3 Analisis Perbandingan Solusi Brute Force dengan Divide and Conquer.....26

BAB V

PENUTUP.....27

 5.1 Kesimpulan..... 27

 5.2 Saran..... 27

DAFTAR PUSTAKA..... 29

LAMPIRAN..... 30

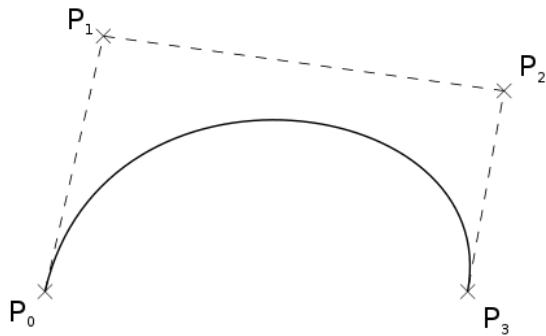
 6.1 GitHub Repository (Latest Release)..... 30

 6.2 Tabel Spesifikasi..... 30

BAB I

DESKRIPSI TUGAS

1.1. Deskripsi Tugas



(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 realistik, 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.

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.

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

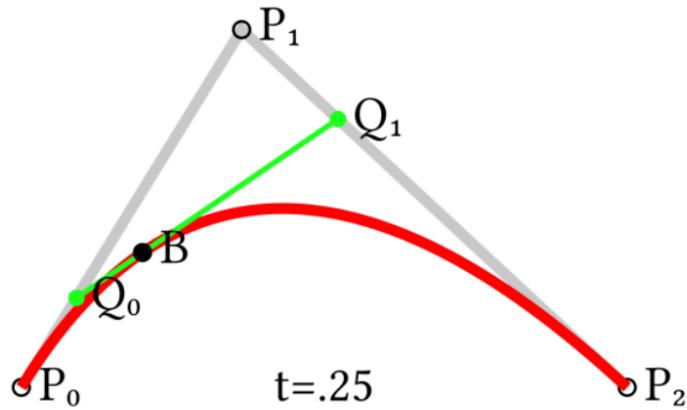
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \quad t \in [0, 1]$$

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

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.



(Sumber: [https://simonhalliday.com/2017/02/15/quadratic-Bézier-curve-demo/](https://simonhalliday.com/2017/02/15/quadratic-B%C3%A9zier-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 II

LANDASAN TEORI

2.1 Algoritma *Brute Force*

Algoritma *brute force* adalah sebuah pendekatan yang langsung (*straightforward*) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan. Algoritma brute force memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas (*obvious way*).

Kelebihan Algoritma *Brute Force*:

- 1) Algoritma *brute force* dapat digunakan untuk memecahkan hampir sebagian besar masalah.
- 2) Sederhana dan mudah dimengerti.
- 3) Menghasilkan algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan string, perkalian matriks.
- 4) Menghasilkan algoritma baku (standar) untuk tugas-tugas komputasi seperti penjumlahan/perkalian N buah bilangan, menentukan elemen minimum atau maksimum ditabel.

Kelemahan Algoritma *Brute Force*:

- 1) Jarang menghasilkan algoritma yang mangkus/efektif.
- 2) Lambat sehingga tidak dapat diterima.
- 3) Tidak sekreatif teknik pemecahan masalah lainnya.

Cara Kerja Algoritma *Brute Force*:

- 1) Definisikan masalah yang ingin diselesaikan dan parameter-parameter yang terlibat.
- 2) *Generate* semua kemungkinan solusi tanpa mempertimbangkan keefisiensian.
- 3) Evaluasi setiap solusi untuk memeriksa apakah memenuhi kriteria keberhasilan.
- 4) Pilih solusi yang paling optimal atau memenuhi kriteria terbaik dari semua solusi yang dihasilkan.
- 5) Implementasikan solusi yang dipilih dan analisis performa serta keefisiensinya.
- 6) Optimalkan algoritma jika diperlukan untuk meningkatkan performa atau mengurangi kompleksitasnya.
- 7) Uji coba algoritma pada berbagai kasus uji untuk memastikan hasil yang benar dan sesuai dengan ekspektasi.
- 8) Ulangi proses jika diperlukan untuk meningkatkan performa atau menangani masalah baru yang muncul.
- 9) Gunakan algoritma *brute force* secara efektif untuk menyelesaikan masalah, dengan memperhitungkan kebutuhan dan keterbatasan yang ada.

2.2 Algoritma Divide and Conquer

Algoritma *divide and conquer* adalah strategi pemecahan masalah yang efisien, yang bekerja dengan mekanisme membagi masalah utama menjadi sub-masalah yang lebih kecil, menyelesaikan sub-masalah tersebut (baik secara langsung atau melalui rekursi), dan kemudian menggabungkan solusi dari sub-masalah untuk membentuk solusi dari masalah utama. Proses ini terdiri dari tiga langkah utama:

1) *Divide (Membagi)*

Langkah pertama adalah membagi masalah utama menjadi beberapa sub-masalah yang lebih kecil. Sub-masalah ini idealnya harus memiliki karakteristik yang sama dengan masalah asli tapi berukuran lebih kecil, memudahkan penanganannya. Pembagian ini terus dilakukan hingga sub-masalah mencapai ukuran yang cukup kecil sehingga bisa diselesaikan dengan mudah dan langsung.

2) *Conquer (Menyelesaikan)*

Setelah masalah dibagi menjadi sub-masalah yang lebih kecil, langkah selanjutnya adalah menyelesaikan masing-masing sub-masalah tersebut. Penyelesaian ini dilakukan secara langsung jika sub-masalah sudah cukup kecil atau melalui rekursi lebih lanjut jika sub-masalah masih terlalu besar untuk dipecahkan secara langsung. Ini memungkinkan penanganan masalah kompleks dengan cara yang lebih terstruktur dan sistematis.

3) *Combine (Menggabungkan)*

Langkah terakhir dalam algoritma *divide and conquer* adalah menggabungkan solusi dari semua sub-masalah yang telah diselesaikan untuk membentuk solusi untuk masalah utama. Proses penggabungan ini harus dilakukan dengan cara yang mempertimbangkan bagaimana sub-masalah dibagi pada awalnya, memastikan bahwa solusi akhir memenuhi semua persyaratan dan kondisi dari masalah asli.

Kelebihan Algoritma *Divide and Conquer*

- 1) Algoritma ini sangat efektif untuk masalah skala besar karena memecah masalah menjadi bagian-bagian yang lebih kecil membuat masalah tersebut lebih mudah diatasi.
- 2) Proses pemecahan masalah dapat diparalelkan karena sub-masalah diatasi secara independen, yang meningkatkan efisiensi pada sistem dengan multi-prosesor.
- 3) Dalam banyak kasus, algoritma *Divide and Conquer* dapat mengurangi waktu komputasi dari polinomial menjadi logaritmik atau sub-linier.
- 4) Memberikan solusi yang optimal atau efisien untuk berbagai masalah komputasi, seperti sorting (*QuickSort*, *MergeSort*), pencarian (*Binary Search*), dan perhitungan matematika.

Kelemahan Algoritma *Divide and Conquer*

- 1) Proses membagi masalah dan menggabungkan solusi sub-masalah bisa menambah *overhead* tambahan yang tidak ada dalam pendekatan *brute force*.
- 2) Untuk beberapa masalah, mungkin sulit untuk menentukan cara membagi masalah tersebut menjadi sub-masalah yang lebih kecil.
- 3) Penggabungan solusi dari sub-masalah bisa menjadi tantangan tersendiri, terutama jika sub-masalah memiliki solusi yang sangat beragam.

Cara Kerja Algoritma *Divide and Conquer*

- 1) Identifikasi masalah yang ingin diselesaikan dan bagi menjadi beberapa sub-masalah yang lebih kecil dan lebih mudah diatasi.

- 2) Selesaikan setiap sub-masalah secara independen. Hal ini dapat dilakukan dengan rekursi, di mana sub-masalah dipecah lebih lanjut hingga mencapai kasus dasar yang mudah diatasi.
- 3) Gabungkan solusi dari sub-masalah untuk membentuk solusi untuk masalah utama.
- 4) Implementasikan solusi yang telah ditemukan dan analisis efisiensi serta performanya. Evaluasi apakah solusi memenuhi semua kriteria dan spesifikasi masalah.
- 5) Jika diperlukan, lakukan optimasi untuk meningkatkan efisiensi atau untuk mengurangi kompleksitas komputasi.
- 6) Uji coba algoritma pada berbagai kasus untuk memastikan keakuratan dan efisiensi. Gunakan *feedback* dari pengujian untuk melakukan peningkatan.
- 7) Ulangi proses jika diperlukan, untuk meningkatkan solusi atau untuk menangani masalah yang belum terpecahkan.

BAB III

SOURCE CODE & ANALISIS IMPLEMENTASI ALGORITMA

3.1 *Source Code & Analisis Implementasi Algoritma Divide and Conquer*

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def is_flat_enough(control_points, iterations, current_iteration):
5     if current_iteration >= iterations:
6         return True
7     return False
8
9 def subdivide(control_points, iterations):
10    left = [control_points[0]]
11    right = [control_points[-1]]
12    while len(control_points) > 1:
13        new_points = []
14        for i in range(len(control_points) - 1):
15            plt.plot([control_points[i][0], control_points[i + 1][0]], [control_points[i][1], control_points[i + 1][1]], 'yo--')
16            plt.draw()
17            if iterations <= 5:      # kalau iterasi lebih dari 5 dan ditambahkan pause, program akan menggambarkan prosesnya satu per satu
18                plt.pause(0.00001)   # sehingga akan memakan waktu yang lama
19                # execution time untuk jumlah iterasi yang <=5 otomatis akan bertambah karena adanya pause berikut
20
21        mid_point = (control_points[i] + control_points[i + 1]) / 2
22        new_points.append(mid_point)
23        if i == 0:
24            left.append(mid_point)
25        if i == len(control_points) - 2:
26            right.append(mid_point)
27        control_points = new_points
28
29    right.reverse()
30    return left, right
31
32 def bezier_divide_and_conquer(control_points, curve_points, iterations, current_iteration=0):
33     if is_flat_enough(control_points, iterations, current_iteration):
34         curve_points.append(control_points[0])
35         curve_points.append(control_points[-1])
36     else:
37         left, right = subdivide(control_points, iterations)
38         bezier_divide_and_conquer(left, curve_points, iterations, current_iteration + 1)
39         bezier_divide_and_conquer(right, curve_points, iterations, current_iteration + 1)
40
41 def plot_curve(control_points, curve_points):
42     # plt.figure(figsize=(10,5))
43     plt.plot(control_points[:, 0], control_points[:, 1], 'co-', label='Control Points')
44     plt.draw()
45     plt.pause(0.05)
46     plt.plot(curve_points[:, 0], curve_points[:, 1], 'mo-', label='Bezier Curve')
47     plt.draw()
48     plt.pause(0.05)
49     plt.legend()
50     plt.title('Bezier Curve Divide and Conquer')
51     plt.xlabel('X')
52     plt.ylabel('Y')
53     plt.show()
```

Program diatas menerapkan algoritma subdivisi kurva Bézier menggunakan pendekatan divide and conquer, yang terdiri dari dua fungsi utama, yaitu subdivide dan Bézier_divide_and_conquer.

Pertama, fungsi `subdivide` bertanggung jawab untuk membagi kurva Bézier menjadi segmen-semen garis yang terhubung antara titik kontrol. Proses ini dimulai dengan mengambil titik awal dan titik akhir dari kurva untuk membentuk dua bagian, yaitu *left* dan *right*. Selanjutnya, dilakukan sebuah *loop* yang akan berlangsung selama masih ada lebih dari satu titik kontrol yang perlu subdivisi. Pada setiap iterasi dalam *loop*, segmen garis antara setiap pasang titik kontrol yang berurutan akan digambarkan secara visual menggunakan matplotlib. Kemudian, titik tengah dari setiap segmen tersebut dihitung dan ditambahkan ke dalam list `'new_points'`. Selain itu, titik-titik tengah tersebut juga akan dimasukkan ke dalam list `'left'` dan `'right'` jika mereka merupakan titik pertama atau terakhir dari kurva. Setelah itu, list `'control_points'` diperbarui dengan list `'new_points'`. Proses ini berlanjut hingga hanya tersisa satu titik kontrol. Terakhir, titik-titik di dalam list *right* dibalikkan urutannya untuk memastikan urutan yang benar, dan kedua list *left* dan *right* dikembalikan sebagai hasil.

Kedua, fungsi `'Bézier_divide_and_conquer'` menggunakan pendekatan *divide and conquer* untuk membagi kurva Bézier menjadi dua bagian, yaitu kiri dan kanan, hingga kurva dianggap cukup datar. Fungsi ini memeriksa apakah kurva sudah cukup datar menggunakan suatu kriteria yang tidak diberikan dalam kode yang diberikan. Jika kurva belum cukup datar, fungsi akan memanggil fungsi `'subdivide'` untuk membagi kurva menjadi dua bagian. Selanjutnya, proses ini akan berlanjut secara rekursif untuk setiap bagian yang baru terbentuk, sampai kurva dianggap sudah cukup datar. Titik-titik hasil dari subdivisi akan ditambahkan ke dalam list `'curve_points'` yang merepresentasikan kurva Bézier akhir. Proses ini akan berlanjut hingga semua subdivisi selesai, dan kurva Bézier akhir terbentuk.

3.2 Source Code & Analisis Implementasi Algoritma Brute Force

```
1  import numpy as np
2  from scipy.special import comb # For binomial coefficients
3  import matplotlib.pyplot as plt
4  from matplotlib.animation import FuncAnimation
5
6  class BezierCurve:
7      def __init__(self):
8          self.bezier_points = []
9
10     def create_bezier(self, control_points, iterations):
11         n = len(control_points) - 1
12         t_values = np.linspace(0, 1, (2 ** iterations) + 1)
13         bezier_points = [self.calculate_bezier_point(t, control_points, n) for t in t_values]
14         return bezier_points
15
16     def calculate_bezier_point(self, t, control_points, n):
17         x, y = 0, 0
18         for i, (px, py) in enumerate(control_points):
19             bernstein_poly = comb(n, i) * (t ** i) * ((1 - t) ** (n - i))
20             x += px * bernstein_poly
21             y += py * bernstein_poly
22         return x, y
23
24     def animate_curve(self, control_points, iterations):
25         n = len(control_points) - 1
26         fig, ax = plt.subplots()
27         ax.set_title('Bezier Curve Animation')
28         line, = ax.plot([], [], 'o-', label='Bezier Curve')
29         ctrl_x, ctrl_y = zip(*control_points)
30         ax.plot(ctrl_x, ctrl_y, 'co--', label='Control Points')
31         ax.legend()
32
33     def init():
34         line.set_data([], [])
35         return line,
36
37     def update(frame):
38         x, y = zip(*self.create_bezier(control_points, frame))
39         line.set_data(x, y)
40         return line,
41
42     anim = FuncAnimation(fig, update, frames=range(1, iterations + 1), init_func=init, blit=True, repeat=False, interval=500)
43     plt.show()
44     return anim
```

Algoritma *brute force* untuk membentuk kurva Bézier menggunakan pendekatan langsung dengan menghitung setiap titik pada kurva secara terpisah. Pertama-tama, dibentuk objek kelas `BezierCurve` yang menyimpan titik-titik hasil kurva Bézier. Fungsi `create_Bezier` menerima titik kontrol dan jumlah iterasi sebagai argumen. Di dalamnya, dilakukan iterasi pada nilai parameter t dari 0 hingga 1 dengan jarak yang dihitung secara proporsional terhadap jumlah iterasi. Setiap nilai t digunakan untuk memanggil fungsi `calculate_Bezier_point` yang mengembalikan titik pada kurva Bézier sesuai dengan nilai t tersebut.

Di dalam fungsi 'calculate_Bézier_point', titik pada kurva Bézier dihitung menggunakan rumus interpolasi Bernstein. Variabel 'x' dan 'y' diinisialisasi terlebih dahulu sebagai 0. Kemudian, dilakukan iterasi pada setiap titik kontrol. Pada setiap iterasi, koefisien polinomial Bernstein untuk titik kontrol tersebut dihitung menggunakan rumus kombinatorial. Koefisien ini kemudian dikalikan dengan koordinat x dan y dari titik kontrol tersebut. Hasil perkalian ini ditambahkan ke nilai x dan y yang sedang diakumulasikan. Proses ini dilakukan untuk setiap titik kontrol, dan akhirnya, titik hasil kurva Bézier pada nilai parameter t yang diberikan dikembalikan dalam bentuk koordinat x dan y.

3.3 *Main Program*

```
46
47 if user_input == '1':
48     num_points = int(input("Enter the number of control points (at least 2): "))
49     control_points = []
50     print("Enter the control points (x, y):")
51     for i in range(num_points):
52         point = input(f"Point {i+1}: ")
53         x, y = map(float, point.split())
54         control_points.append([x, y])
55
56     iterations = int(input("Enter the number of iterations for curve refinement: "))
57
58     control_points_np = np.array(control_points)
59
60     curve_points = []
61     start_time = time.time()
62     bezier.divide_and_conquer(control_points_np, curve_points, iterations)
63     curve_points = np.array(curve_points)
64     end_time = time.time()
65     print()
66     print()
67     print("-----")
68     print("      EXECUTION TIME      ")
69     print("-----")
70     print(f"Execution time: {end_time - start_time} seconds")
71     plot_curve(control_points_np, curve_points)
72     print()
73     print()
74
75 else:
76     bezier = BezierCurve()
77     ctrl1 = tuple(map(int, input("Enter first control point as two space separated integers: ").split()))
78     ctrl2 = tuple(map(int, input("Enter second control point as two space separated integers: ").split()))
79     ctrl3 = tuple(map(int, input("Enter third control point as two space separated integers: ").split()))
80     iterations = int(input("Enter number of iterations: "))
81     start_time = time.time()
82     bezier.create_bezier(ctrl1, ctrl2, ctrl3, iterations)
83     end_time = time.time()
84     print()
85     print()
86     print("-----")
87     print("      EXECUTION TIME      ")
88     print("-----")
89     print(f"Execution time: {end_time - start_time} seconds")
90     bezier.plot_curve([ctrl1, ctrl2, ctrl3])
91     print()
92     print()
```

Proses utama dalam program ini adalah pengambilan opsi menu dari pengguna menggunakan fungsi `get_menu_option()`. Pengguna diminta untuk memilih antara dua opsi: menggunakan metode "*Divide & Conquer*" atau "*Brute Force*" untuk membuat kurva Bézier. Jika pengguna memilih opsi '1', program akan meminta input jumlah titik kontrol, titik kontrol tersebut, dan jumlah iterasi untuk penyempurnaan kurva. Setelah mendapatkan input tersebut, program akan memanggil fungsi `Bezier_divide_and_conquer()` dengan titik kontrol dan iterasi

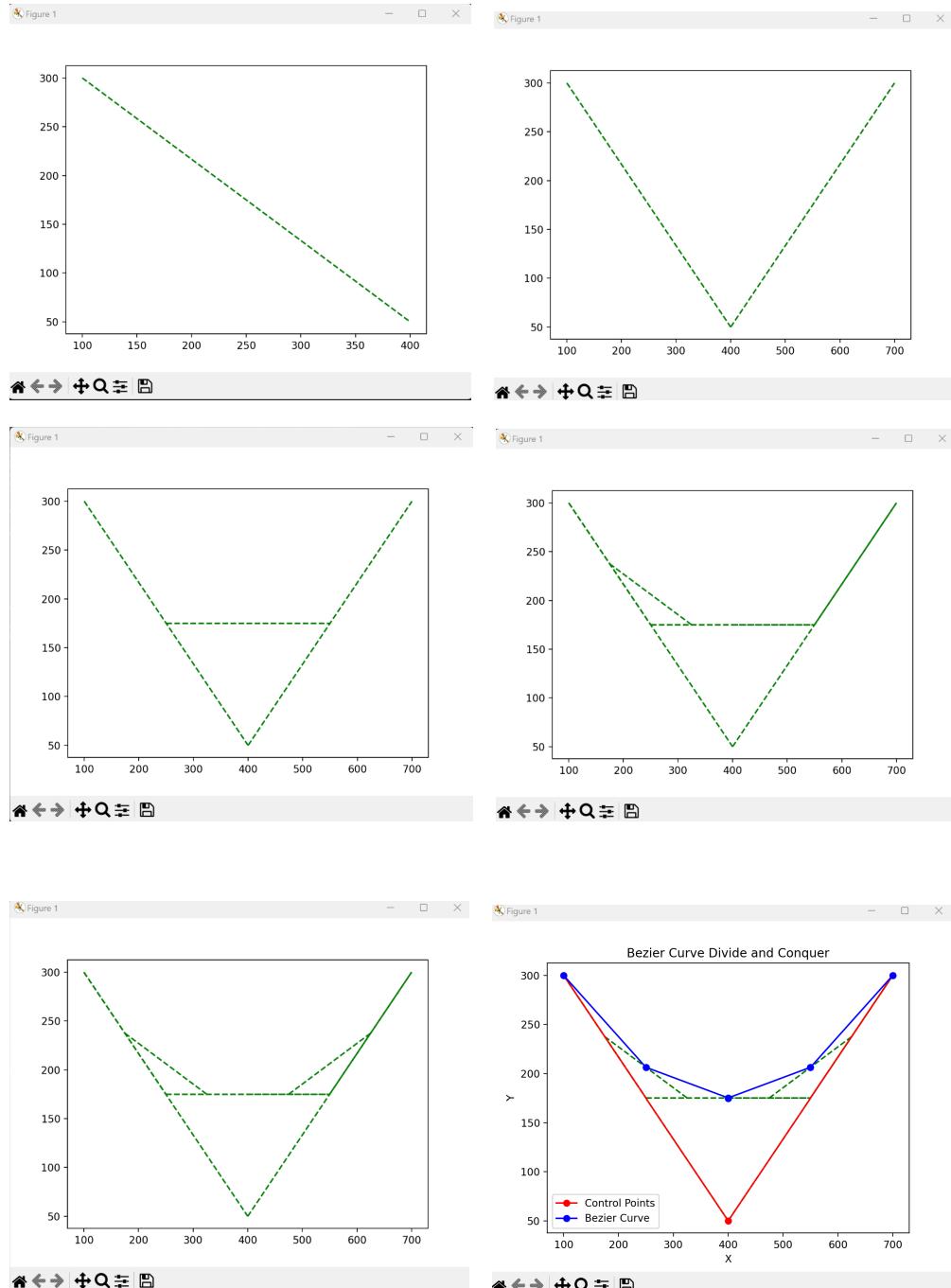
yang diberikan, dan mengukur waktu eksekusi. Selanjutnya, kurva Bézier hasil akan ditampilkan menggunakan fungsi `plot_curve()`.

Namun, jika pengguna memilih opsi '2', program akan meminta input tiga titik kontrol dan jumlah iterasi yang sama, lalu memanggil metode `create_Bezier()` dari objek `BezierCurve`. Waktu eksekusi akan diukur, dan kurva Bézier hasil akan ditampilkan menggunakan metode `plot_curve()`.

3.4 Implementasi Bonus 1 (Kurva Bézier dengan *n Points*)

Pada kurva Bézier dengan *n points*, implementasinya sama seperti implementasi program *Divide and Conquer* pada subbab 3.1 diatas. Program menggunakan algoritma subdivisi kurva Bézier dengan pendekatan divide and conquer, yang melibatkan dua fungsi utama: `subdivide` dan `Bezier_divide_and_conquer`. Fungsi `subdivide` bertugas membagi kurva Bézier menjadi segmen-segmen garis antara titik kontrol. Ini dimulai dengan membagi kurva menjadi dua bagian, kiri dan kanan, dengan mengambil titik awal dan akhir. Setiap segmen garis antara titik kontrol digambar secara visual menggunakan matplotlib, lalu titik tengahnya dihitung dan dimasukkan ke dalam list `new_points`. Titik-titik tengah yang sesuai juga ditambahkan ke dalam list `left` dan `right`, jika mereka merupakan titik awal atau akhir kurva. Proses ini terus berlanjut hingga hanya tersisa satu titik kontrol. Fungsi `Bezier_divide_and_conquer` menggunakan pendekatan *divide and conquer* untuk membagi kurva menjadi dua bagian hingga kurva dianggap cukup datar. Jika belum, fungsi akan memanggil `subdivide` secara rekursif untuk membagi kurva lebih lanjut. Titik-titik hasil subdivisi ditambahkan ke dalam list `curve_points`, merepresentasikan kurva Bézier akhir. Proses ini berlanjut hingga semua subdivisi selesai dan kurva Bézier akhir terbentuk.

3.5 Implementasi Bonus 2 (Visualisasi Proses Pembentukan Kurva Bézier)

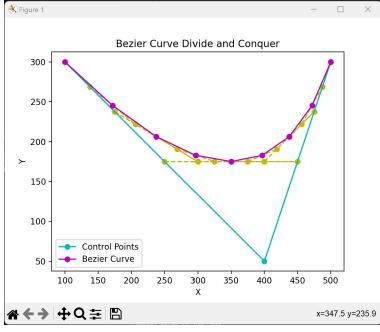
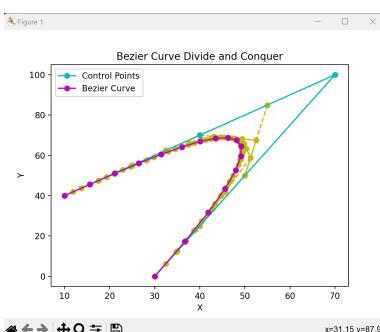
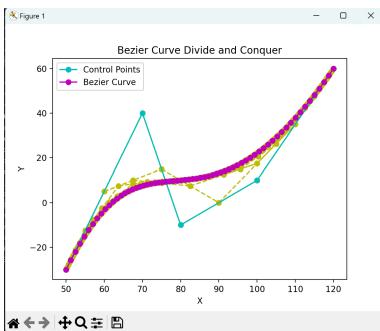


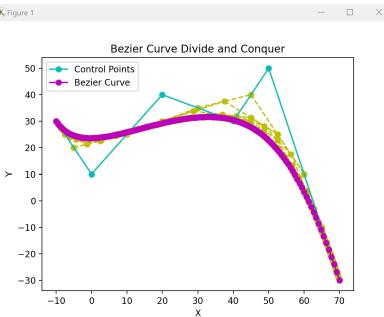
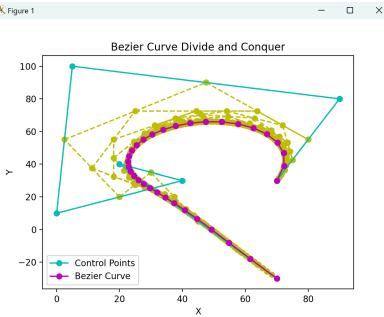
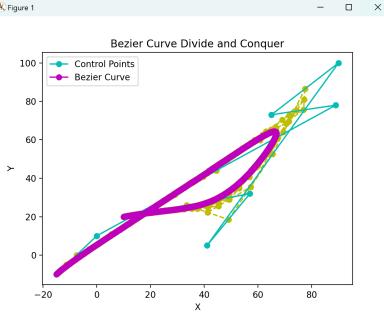
Gambar diatas adalah contoh visualisasi proses pembentukan kurva Bézier menggunakan algoritma titik tengah *divide and conquer* untuk 3 titik kontrol dan 2 iterasi. Untuk mengilustrasikan tahapan pembentukan kurva, penulis memanfaatkan pustaka *matplotlib* dengan memanggil `plt.draw()` dan `plt.pause()`. Ini memungkinkan penulis untuk secara dinamis memperbarui gambar kurva saat proses pembentukan sedang berlangsung. Ketika `plt.draw()` digunakan, grafik akan diperbarui sesuai dengan perubahan yang terjadi, sehingga pengguna dapat melihat secara visual bagaimana kurva berkembang dari satu tahap ke tahap berikutnya. Sedangkan `plt.pause()` digunakan untuk memberikan jeda singkat dalam proses visualisasi yang memungkinkan pengguna untuk mengamati setiap perubahan dengan lebih jelas tanpa kurva yang berubah terlalu cepat. Dengan demikian, pengguna dapat memahami lebih baik bagaimana kurva Bézier terbentuk dari titik kontrol yang diberikan.

BAB IV

HASIL UJI COBA

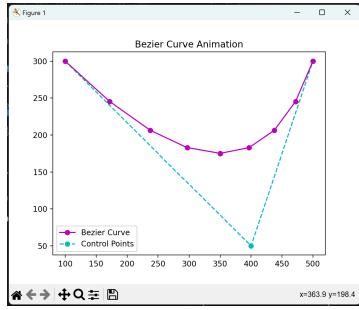
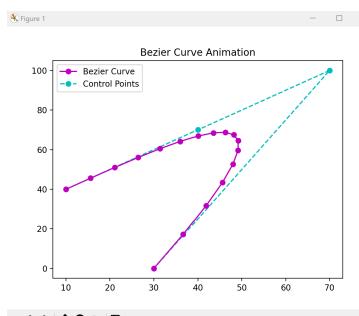
4.1 Hasil Uji Coba Algoritma *Divide and Conquer*

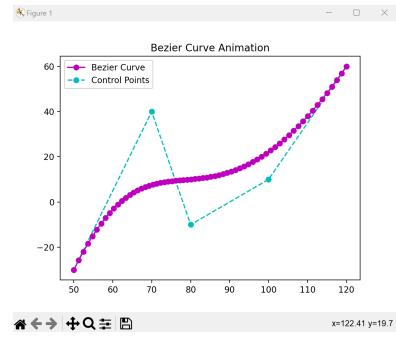
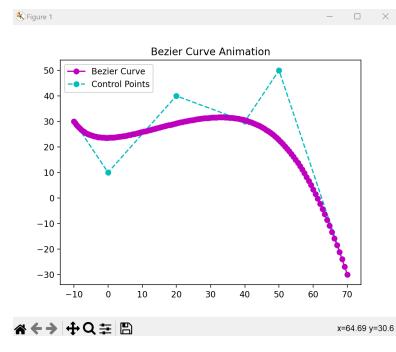
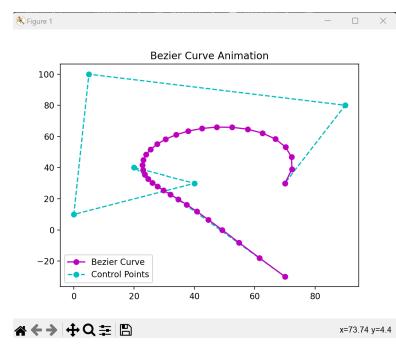
No	Input dan Output <i>Execution Time</i>	Output Kurva
1	<pre>Enter the number of control points (>1): 3 Enter the control points (x, y): Control point 1: 100, 300 Control point 2: 400, 50 Control point 3: 500, 300 Enter the number of iterations (>=0): 3 ----- EXECUTION TIME ----- Execution time: 0.30664896965026855 seconds</pre> <p>(di atas adalah waktu eksekusi tanpa animasi, jika ada animasi waktunya akan lebih lama)</p>	
2	<pre>Enter the number of control points (>1): 4 Enter the control points (x, y): Control point 1: 10, 40 Control point 2: 40, 70 Control point 3: 70, 100 Control point 4: 30, 0 Enter the number of iterations (>=0): 4 ----- EXECUTION TIME ----- Execution time: 0.347592830657959 seconds</pre> <p>(di atas adalah waktu eksekusi tanpa animasi, jika ada animasi waktunya akan lebih lama)</p>	
3	<pre>Enter the number of control points (>1): 5 Enter the control points (x, y): Control point 1: 50, -30 Control point 2: 70, 40 Control point 3: 80, -10 Control point 4: 100, 10 Control point 5: 120, 60 Enter the number of iterations (>=0): 6 ----- EXECUTION TIME ----- Execution time: 0.673229455947876 seconds</pre> <p>(di atas adalah waktu eksekusi tanpa animasi,</p>	

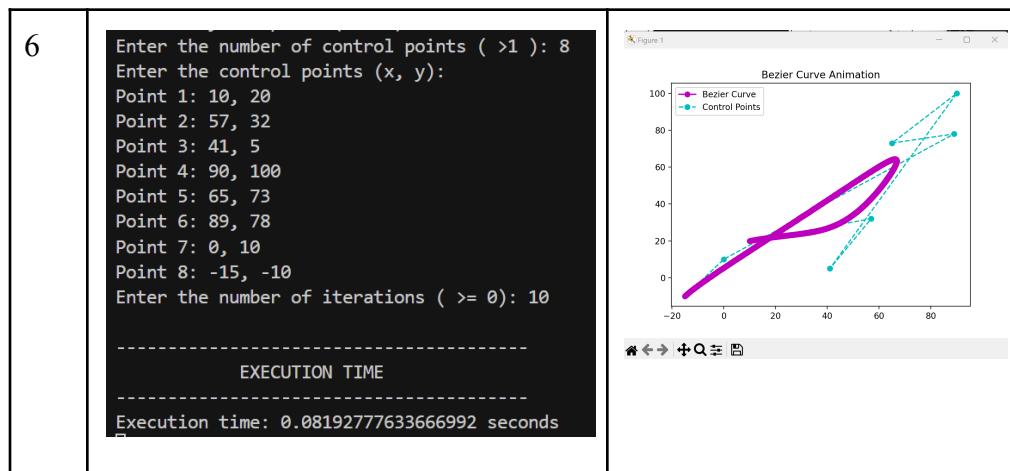
	<p>jika ada animasi waktunya akan lebih lama)</p>	
4	<pre>Enter the number of control points (>1): 6 Enter the control points (x, y): Control point 1: -10, 30 Control point 2: 0, 10 Control point 3: 20, 40 Control point 4: 40, 30 Control point 5: 50, 50 Control point 6: 70, -30 Enter the number of iterations (>=0): 7 ----- EXECUTION TIME ----- Execution time: 1.5050976276397705 seconds</pre> <p>(di atas adalah waktu eksekusi tanpa animasi, jika ada animasi waktunya akan lebih lama)</p>	
5	<pre>Enter the number of control points (>1): 7 Enter the control points (x, y): Control point 1: 70, -30 Control point 2: 20, 40 Control point 3: 40, 30 Control point 4: 0, 10 Control point 5: 5, 100 Control point 6: 90, 80 Control point 7: 70, 30 Enter the number of iterations (>=0): 5 ----- EXECUTION TIME ----- Execution time: 0.6976234912872314 seconds</pre> <p>(di atas adalah waktu eksekusi tanpa animasi, jika ada animasi waktunya akan lebih lama)</p>	
6	<pre>Enter the number of control points (>1): 8 Enter the control points (x, y): Control point 1: 10, 20 Control point 2: 57, 32 Control point 3: 41, 5 Control point 4: 90, 100 Control point 5: 65, 73 Control point 6: 89, 78 Control point 7: 0, 10 Control point 8: -15, -10 Enter the number of iterations (>=0): 10 ----- EXECUTION TIME ----- Execution time: 21.29399585723877 seconds</pre> <p>(di atas adalah waktu eksekusi tanpa animasi,</p>	

	jika ada animasi waktunya akan lebih lama)	
--	--	--

4.2 Hasil Uji Coba Algoritma *Brute Force*

No	Input dan Output <i>Execution Time</i>	Output Kurva
1	<pre>Enter the number of control points (>1): 3 Enter the control points (x, y): Point 1: 100, 300 Point 2: 400, 50 Point 3: 500, 300 Enter the number of iterations (>= 0): 3 ----- EXECUTION TIME ----- Execution time: 0.000980377197265625 seconds</pre>	
2	<pre>Enter the number of control points (>1): 4 Enter the control points (x, y): Point 1: 10, 40 Point 2: 40, 70 Point 3: 70, 100 Point 4: 30, 0 Enter the number of iterations (>= 0): 4 ----- EXECUTION TIME ----- Execution time: 0.001013040542602539 seconds</pre>	

3	<pre>Enter the number of control points (>1): 5 Enter the control points (x, y): Point 1: 50, -30 Point 2: 70, 40 Point 3: 80, -10 Point 4: 100, 10 Point 5: 120, 60 Enter the number of iterations (>= 0): 6 ----- EXECUTION TIME ----- Execution time: 0.003792285919189453 seconds</pre>	
4	<pre>Enter the number of control points (>1): 6 Enter the control points (x, y): Point 1: -10, 30 Point 2: 0, 10 Point 3: 20, 40 Point 4: 40, 30 Point 5: 50, 50 Point 6: 70, -30 Enter the number of iterations (>= 0): 7 ----- EXECUTION TIME ----- Execution time: 0.0075380802154541016 seconds</pre>	
5	<pre>Enter the number of control points (>1): 7 Enter the control points (x, y): Point 1: 70, -30 Point 2: 20, 40 Point 3: 40, 30 Point 4: 0, 10 Point 5: 5, 100 Point 6: 90, 80 Point 7: 70, 30 Enter the number of iterations (>= 0): 5 ----- EXECUTION TIME ----- Execution time: 0.0029740333557128906 seconds</pre>	



4.3 Analisis Perbandingan Solusi *Brute Force* dengan *Divide and Conquer*

Dari kedua pengujian yang telah dilakukan, terdapat kesulitan dalam membandingkan kecepatan kedua algoritma secara konsisten. Kompleksitas waktu algoritma, yang biasanya diukur dengan notasi Big O, juga tidak bisa digunakan untuk membandingkannya karena operasi yang dilakukan oleh keduanya berbeda. Meskipun algoritma *brute force* memiliki kompleksitas algoritma yang lebih tinggi, hasil penelitian menunjukkan bahwa algoritma *brute force* lebih cepat. Hal ini bisa disebabkan oleh faktor-faktor seperti *overhead* yang terlibat dalam rekursi dan pengelolaan memori tambahan dalam algoritma *divide and conquer*. Kinerja algoritma dapat dipengaruhi oleh implementasi masing-masing algoritma, termasuk cara pengoptimalan dan struktur data yang digunakan. Jadi, kesimpulannya meskipun algoritma *divide and conquer* biasanya dianggap lebih efisien dalam memecahkan masalah secara umum, hasilnya mungkin berbeda dalam kasus spesifik seperti pembuatan kurva Bézier.

BAB V

PENUTUP

5.1 Kesimpulan

Dalam laporan ini, penulis berhasil mendemonstrasikan implementasi algoritma pembangunan kurva Bézier dengan pendekatan titik tengah berbasis *divide and conquer*. Melalui eksperimen yang dilakukan, penulis dapat menyimpulkan bahwa pendekatan ini mampu membangun kurva Bézier dengan akurasi yang memadai serta kinerja yang baik dalam hal waktu eksekusi. Perbandingan waktu eksekusi antara program yang menggunakan algoritma *divide and conquer* dan algoritma *brute force* menunjukkan bahwa meskipun algoritma *brute force* memiliki kompleksitas algoritma yang lebih tinggi, hasil penelitian menunjukkan bahwa algoritma tersebut lebih cepat. Hal ini mungkin disebabkan oleh berbagai faktor seperti beban tambahan yang terjadi pada proses rekursi dan pengelolaan memori yang lebih banyak pada algoritma *divide and conquer*. Kinerja dari kedua algoritma juga bisa dipengaruhi oleh berbagai faktor lainnya, seperti implementasi spesifik dari masing-masing algoritma, strategi pengoptimalan yang digunakan, dan jenis struktur data yang dipilih. Oleh karena itu, meskipun algoritma *divide and conquer* sering dianggap lebih efisien dalam menyelesaikan masalah secara umum, namun dalam kasus tertentu seperti pembangunan kurva Bézier, hasilnya bisa jadi berbeda, seperti yang telah diimplementasikan dalam tugas ini.

5.2 Saran

Sebagai saran untuk penelitian selanjutnya, penulis merekomendasikan eksplorasi lebih lanjut terhadap optimasi algoritma divide and conquer dalam

konteks pembangunan kurva Bézier. Hal ini dapat melibatkan peningkatan efisiensi algoritma, penggunaan strategi pendekatan yang lebih canggih, atau penyesuaian kriteria untuk menentukan kapan kurva dianggap cukup halus. Selain itu, penulis juga mengusulkan untuk memperluas analisis perbandingan kinerja antara algoritma divide and conquer dengan algoritma bruteforce dengan menggunakan lebih banyak kasus uji dan variasi parameter. Pemahaman yang lebih mendalam tentang perbedaan kinerja antara kedua pendekatan ini dapat memberikan wawasan yang lebih baik bagi pengembangan aplikasi praktis dari kurva Bézier. Dengan demikian, penelitian lebih lanjut dalam hal ini dapat memberikan kontribusi yang berharga bagi pengembangan teknik pembangunan kurva Bézier yang efisien dan akurat.

DAFTAR PUSTAKA

- Munir, R. (2024). *Strategi Algoritma*. Retrieved from Homepage Rinaldi Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir/>
- Bandara, Ravimal. (2011). *Midpoint Algorithm (Divide and Conquer Method) for Drawing a Simple Bézier Curve*: [Midpoint Algorithm \(Divide and Conquer Method\) for Drawing a Simple Bézier Curve - CodeProject](#)
- Mateus, Melo. (2021). *Understanding Bézier Curves*: [Understanding Bézier Curves. A mathematical and intuitive approach | by Mateus Melo | Medium](#)

LAMPIRAN

6.1 GitHub Repository (Latest Release)

https://github.com/caernations/Tucil2_13522140_13522148

6.2 Tabel Spesifikasi

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.	✓	