

IF2211 Strategi Algoritma
Pengembangan Kurva Bézier Menggunakan Algoritma *Divide and Conquer*

Laporan Tugas Kecil 2

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma
pada Semester 2 (dua) Tahun Akademik 2023/2024



Disusun oleh:

Rizqika Mulia Pratama (13522126)

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

**BANDUNG
2024**

DAFTAR ISI

DAFTAR ISI	2
BAB I	3
1.1. Abstrak	3
1.2. Pendahuluan	3
1.3. Landasan Teori	4
1.3.1. Kurva Bézier	4
1.3.2. Algoritma Divide and Conquer	4
1.3.3. Algoritma De Casteljau	4
1.3.4. Algoritma Brute Force	5
BAB II	5
2.1. Spesifikasi Tugas	5
BAB III	6
3.1. Implementasi Algoritma Divide and Conquer	6
3.2. Implementasi Algoritma Brute Force	7
3.3. Implementasi Algoritma Divide and Conquer Pada Bonus Dengan Pendekatan Algoritma De Casteljau	7
3.4. Implementasi Visualisasi Kurva Bézier	8
BAB IV	10
4.1. Source Code	10
4.2. Pengetesan Kode	17
BAB V	20
5.1. Analisis Perbandingan Algoritma Divide and Conquer dengan Algoritma Brute Force dalam Pembentukan Kurva Bézier	20
5.2. Analisis Penggunaan Pendekatan Algoritma De Casteljau Terhadap Pembuatan Kurva Bézier dengan N Titik Kontrol Menggunakan Algoritma Divide and Conquer	21
BAB VI	23
6.1. Kesimpulan	23
6.2. Saran	23
BAB VII	24
7.1. Sumber Pustaka	24
7.2. Lampiran	24

BAB I

PENDAHULUAN

1.1. Abstrak

Laporan ini membahas pengembangan dan visualisasi Kurva Bézier, yang merupakan elemen penting dalam desain grafis komputer, animasi, dan industri manufaktur, menggunakan dua pendekatan algoritma yang berbeda: algoritma *divide and conquer* dan algoritma *brute force* sebagai pembandingan. Algoritma *divide and conquer* menggunakan algoritma *de casteljau*. Melalui analisis dan implementasi kedua metode ini, laporan ini bertujuan untuk membandingkan efektivitas dan efisiensi mereka dalam pembentukan kurva bézier yang halus dan presisi.

1.2. Pendahuluan



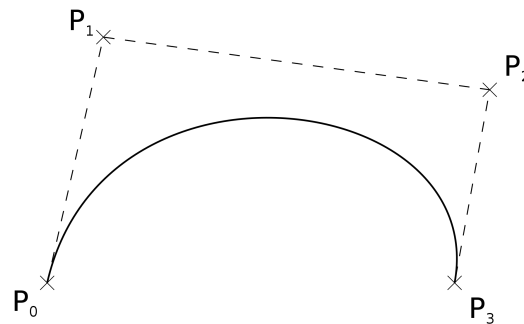
Gambar 1 Pierre Bézier

(Sumber: https://en.wikipedia.org/wiki/Pierre_B%C3%A9zier)

Kurva Bézier, diperkenalkan oleh Pierre Bézier untuk digunakan dalam desain bodi mobil, adalah kurva parametrik yang banyak digunakan dalam grafik komputer untuk menggambar gambar yang halus. Kurva ini dibangun berdasarkan serangkaian titik kontrol yang menentukan bentuknya. Dalam laporan ini, penulis fokus pada implementasi dan analisis dua pendekatan algoritma untuk membangun kurva bézier: *brute force* dan *divide and conquer*, dengan mencakup penerapan algoritma *de casteljau* dalam konteks *divide and conquer*.

1.3. Landasan Teori

1.3.1. Kurva Bézier



Gambar 1 Kurva Bézier

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

Kurva Bézier dibangun menggunakan serangkaian titik kontrol P_0, P_1, \dots, P_n , di mana kurva dimulai dari P_0 dan berakhir di P_n . Kurva Bézier linier, kuadratik, kubik, dan seterusnya dibangun berdasarkan jumlah titik kontrol.

Kurva Linier: Dibangun dari dua titik kontrol dan merupakan garis lurus antara titik-titik tersebut.

Kurva Kuadratik: Dibangun dari tiga titik kontrol, menghasilkan kurva halus yang melewati titik awal dan akhir tetapi biasanya tidak melewati titik kontrol tengah.

Kurva Kubik: Dibangun dari empat titik kontrol, memberikan fleksibilitas lebih dalam pembentukan kurva.

1.3.2. Algoritma Divide and Conquer

Algoritma Divide and Conquer adalah teknik algoritma yang digunakan dalam pemrograman komputer untuk memecahkan masalah yang kompleks. Prinsip dari algoritma ini adalah memecah-mecah masalah yang ada menjadi beberapa masalah yang lebih kecil sehingga mudah untuk diselesaikan.

Algoritma divide and conquer mempunyai kelebihan dapat mengurangi kompleksitas pencarian solusi suatu masalah. Kelebihan tersebut banyak menguntungkan dari segi waktu, tenaga, dan sumber daya.

1.3.3. Algoritma De Casteljau

Algoritma De Casteljau adalah metode numerik untuk menghitung dan memvisualisasi kurva Bézier. Algoritma ini bekerja dengan cara membagi secara rekursif segmen garis antara titik-titik kontrol menjadi bagian yang lebih kecil, memungkinkan pembentukan kurva pendekatan yang sangat akurat.

1.3.4. Algoritma Brute Force

Algoritma brute force adalah algoritma yang memiliki pendekatan yang lempang untuk memecahkan masalah, algoritma brute force memecahkan masalah dengan cara yang sangat sederhana, langsung, dan jelas caranya.

BAB II SPESIFIKASI

2.1. Spesifikasi Tugas

- Buatlah program sederhana dalam bahasa C++/Python/JavaScript/Go yang dapat membentuk sebuah **kurva Bézier kuadratik** dengan menggunakan algoritma titik tengah berbasis divide and conquer.
- Selain implementasi dalam algoritma divide and conquer, program **juga diminta untuk diimplementasikan dalam algoritma brute force**. Solusi ini ditujukan sebagai pembandingan dengan solusi sebelumnya.
- Tugas dapat dikerjakan individu atau berkelompok dengan anggota **maksimal 2 orang** (sangat disarankan). Boleh lintas kelas dan lintas kampus, tetapi **tidak boleh sama** dengan anggota kelompok pada tugas-tugas Strategi Algoritma sebelumnya.
- **Input :**
Format masukan **dibebaskan**, dengan catatan dijelaskan pada README dan laporan. Komponen yang perlu menjadi masukan yaitu:
 1. **Tiga buah pasangan titik**. Sebagai catatan, titik yang paling awal dimasukkan akan menjadi titik awal kurva, begitu juga dengan titik yang paling akhir.
 2. **Jumlah iterasi** yang ingin dilakukan.
- **Output:**
Berikut adalah luaran dari program yang diekspektasikan:
 1. Hasil **kurva Bézier yang terbentuk** pada iterasi terkait. Kakas yang digunakan untuk visualisasi dibebaskan.
 2. **Waktu eksekusi** program pembentukan kurva.
- **Bonus:**
Pastikan sudah mengerjakan spesifikasi wajib sebelum mengerjakan bonus.
 1. Melakukan generalisasi algoritma, ide, serta melakukan implementasinya sehingga program dapat membentuk kurva Bézier kubik, kuartik, dan selanjutnya dengan 4, 5, 6, hingga n titik kontrol.
 2. Memberikan visualisasi proses pembentukan kurva, sehingga tidak hanya hasil akhirnya saja. Tentu saja proses visualisasinya perlu melibatkan Graphical User Interface (GUI) yang kakasnya dibebaskan.

BAB III

ANALISIS DAN IMPLEMENTASI ALGORITMA

3.1. Implementasi Algoritma *Divide and Conquer*

Dalam implementasi algoritma divide and conquer untuk pembentukan kurva bézier, dilakukan pembagian masalah menjadi submasalah yang lebih kecil, menyelesaikan setiap submasalah secara rekursif, dan menggabungkan solusi dari submasalah untuk membentuk solusi akhir. Berikut adalah deskripsi langkah-langkahnya:

1. Pembagian masalah
 - Masalah utama adalah pembentukan kurva bézier antara tiga titik kontrol: P_0 , P_1 , dan P_2 .
 - Langkah pertama adalah membagi dua kurva menjadi submasalah yang lebih kecil. Untuk kurva bézier kuadratik, digunakan dua titik kontrol tambahan, yaitu P_{01} dan P_{12} , yang merupakan titik tengah dari P_0 dan P_1 , serta P_1 dan P_2 .
 - Titik tambahan tersebut digunakan untuk membagi kurva menjadi dua segmen: P_0 - P_{01} - P_{12} dan P_{12} - P_{12} - P_2 .
2. Penyelesaian subkurva
 - Setelah membagi kurva menjadi dua subkurva, setiap subkurva diselesaikan secara terpisah.
 - Hal ini dilakukan dengan memanggil fungsi rekursif untuk setiap subkurva yang baru terbentuk.
 - Pada kasus ini, dengan memanggil fungsi *bezier_quadratic_divide_and_conquer* untuk menghitung segmen P_0 - P_{01} - P_{12} dan segmen P_{12} - P_{12} - P_2 secara terpisah.
3. Penggabungan solusi
 - Setelah solusi untuk setiap subkurva ditemukan, solusi-solusi tersebut digabungkan untuk membentuk solusi akhir.
 - Untuk kurva bézier kuadratik, hasil dari kedua subkurva digabungkan menjadi satu array yang berisi semua titik dari kedua subkurva.

Dengan mengikuti langkah-langkah tersebut, algoritma divide and conquer dapat digunakan untuk secara efisien membangun kurva Bézier antara tiga titik kontrol yang diberikan. Algoritma ini dapat dengan cepat menghasilkan kurva yang halus dan akurat dengan membagi masalah menjadi submasalah yang lebih kecil dan menyelesaikannya secara terpisah sebelum menggabungkan solusinya.

3.2. Implementasi Algoritma *Brute Force*

Dalam implementasi algoritma *brute force* untuk pembentukan kurva bézier, semua kemungkinan kombinasi titik kontrol dicoba untuk menemukan kurva yang optimal. Berikut adalah analisis dan implementasi langkah-langkahnya:

1. Persiapan variabel
 - Menetapkan titik kontrol P_0 , P_1 , dan P_2 .
 - Menentukan iterasi atau jumlah titik yang akan dievaluasi untuk membentuk kurva bézier.
 - Setiap nilai t dievaluasi untuk menghitung koordinat (x,y) titik pada kurva bézier.
2. Iterasi titik pada kurva bézier
 - Membuat serangkaian nilai parameter t dalam domainnya, dari 0 hingga 1
 - Untuk setiap nilai t , menghitung koordinat titik pada kurva Bézier menggunakan rumus matematika kurva bézier kuadratik:
$$x = (1 - t)^2 * P0[0] + 2 * (1 - t) * t * P1[0] + t^2 * P2[0]$$
$$y = (1 - t)^2 * P0[1] + 2 * (1 - t) * t * P1[1] + t^2 * P2[1]$$
 - Menyimpan koordinat (x,y) dari setiap titik yang dievaluasi pada kurva bézier.
3. Penyimpanan hasil
 - Setelah semua titik pada kurva bézier dievaluasi, hasilnya disimpan dalam bentuk array.
 - Data ini dapat digunakan untuk visualisasi kurva bézier atau untuk analisis lebih lanjut.

Setelah langkah-langkah di atas selesai, kurva bézier lengkap telah terbentuk berdasarkan titik kontrol yang diberikan.

3.3. Implementasi Algoritma *Divide and Conquer* Pada Bonus Dengan Pendekatan Algoritma *De Casteljau*

Algoritma *divide and conquer* diimplementasikan dengan pendekatan *algoritma de casteljau* untuk membentuk kurva Bézier dengan jumlah titik kontrol yang dapat bervariasi, sesuai dengan input pengguna. Kurva Bézier yang dihasilkan akan memiliki ketelitian yang disesuaikan dengan jumlah titik kontrol yang diberikan. Berikut adalah analisis dan implementasi langkah-langkahnya:

1. Persiapan Variabel
 - Mendefinisikan fungsi *de_casteljau* untuk menghitung titik pada kurva Bézier menggunakan pendekatan algoritma *de casteljau*.
 - Menginisialisasi cache untuk menyimpan hasil perhitungan titik kontrol yang telah dievaluasi sebelumnya.

2. Pendekatan algoritma *de casteljau*
 - Algoritma *de casteljau* digunakan untuk membagi kurva Bézier menjadi segmen-segmen yang lebih kecil.
 - Fungsi *de_casteljau* menerima titik kontrol dan parameter t , kemudian mengembalikan titik pada kurva bézier pada titik tersebut.
 - Pada setiap iterasi, algoritma *de casteljau* membagi kurva menjadi dua segmen dengan membagi setiap segmen pada titik kontrol tengah.
3. Algoritma *divide and conquer*
 - Fungsi *bezier_general_dnc* digunakan untuk menerapkan pendekatan algoritma *divide and conquer* secara keseluruhan.
 - Jumlah segmen dan titik yang dibutuhkan dihitung berdasarkan iterasi yang diberikan.
 - Untuk setiap nilai t , titik pada kurva bézier dihitung menggunakan fungsi *de_casteljau*.
4. Penyimpanan hasil
 - Hasil dari setiap iterasi disimpan dalam *cache* untuk menghindari pengulangan perhitungan yang tidak perlu.
 - *Cache* menggunakan struktur data *map* yang memungkinkan pencarian cepat berdasarkan titik kontrol yang diberikan.

Adapun rasionalisasi penggunaan pendekatan algoritma *de casteljau*:

- **Keuntungan pemisahan**, Algoritma *de casteljau* memungkinkan pemisahan yang efisien dari kurva bézier menjadi segmen-segmen yang lebih kecil. Dalam algoritma ini, setiap segmen dibagi menjadi dua bagian, yang memungkinkan penyelesaian masalah secara rekursif.
- **Reduksi kompleksitas**: Dibandingkan dengan algoritma lain, seperti interpolasi polinomial, pendekatan *de casteljau* cenderung memiliki kompleksitas waktu yang lebih rendah. Ini disebabkan oleh sifat rekursifnya yang memungkinkan penggunaan ulang perhitungan yang sama pada setiap level rekursi, mengurangi jumlah perhitungan yang harus dilakukan.
- **Ketelitian dan kelancaran**: Algoritma *de casteljau* mempertahankan ketelitian kurva bézier yang tinggi dan kelancaran interpolasinya. Dengan membagi kurva menjadi segmen-segmen yang lebih kecil, algoritma ini dapat mencapai representasi yang lebih akurat dari kurva yang kompleks seperti kurva bézier.

3.4. Implementasi Visualisasi Kurva Bézier

Implementasi visualisasi kurva Bézier menggunakan Python dan beberapa alat yang digunakan, seperti NumPy untuk perhitungan numerik, Tkinter untuk pembuatan antarmuka grafis, serta matplotlib untuk melakukan plotting. Berikut adalah langkah-langkah mekanisme implementasi visualisasi kurva bézier:

1. Pembuatan titik kontrol
 - Titik kontrol adalah titik-titik yang akan digunakan untuk mendefinisikan kurva bézier. Pada implementasi ini, titik kontrol dapat dimasukkan oleh pengguna melalui antarmuka grafis menggunakan Tkinter.
 - Titik kontrol dapat dimasukkan dalam format x,y dan dipisahkan oleh tanda koma (,). Misalnya, "2.5, 3.0; 4.0, 1.5; 6.5, 4.0".
2. Metode perhitungan kurva bézier
 - Ada dua metode perhitungan kurva Bézier yang digunakan dalam kode yang diberikan: metode brute force dan metode divide and conquer.
 - Metode brute force (*bezier_quadratic_brute_force*) menghitung setiap titik pada kurva bézier dengan menggunakan rumus eksplisit dari kurva bézier.
 - Metode divide and conquer (*bezier_quadratic_divide_and_conquer*) membagi kurva bézier menjadi segmen-segmen yang lebih kecil dan menghitung setiap titik pada kurva menggunakan algoritma de casteljau.
3. Perhitungan kurva bézier
 - Setelah mendapatkan titik kontrol dan memilih iterasi, program akan menghitung titik-titik pada kurva bézier menggunakan metode yang dipilih (brute force atau divide and conquer).
4. Pembuatan antarmuka grafis
 - *Interface* grafis dibuat menggunakan Tkinter, yang memungkinkan pengguna untuk memasukkan titik kontrol dan mengatur iterasi.
 - *Interface* termasuk slider untuk memilih iterasi maksimum, entri teks untuk memasukkan titik kontrol, dan area plot untuk menampilkan kurva bézier.
5. Memperbarui plot
 - Saat *user* memasukkan atau mengubah titik kontrol atau iterasi maksimum, plot kurva bézier diperbarui.
 - Metode *update_plot* digunakan untuk menggambar plot baru berdasarkan titik kontrol dan iterasi yang dimasukkan.
6. Pengelolaan *cache*
 - Untuk mengoptimalkan kinerja, program menggunakan cache untuk menyimpan hasil perhitungan kurva bézier sehingga tidak perlu menghitung ulang jika parameter yang sama digunakan lagi.
7. Visualisasi dan interaksi pengguna
 - Setelah plot diperbarui, kurva bézier beserta titik kontrolnya ditampilkan pada *interface* grafis.
 - Pengguna dapat berinteraksi dengan antarmuka untuk mengubah titik kontrol, mengatur iterasi, dan melihat hasil kurva bézier yang diperbarui.

BAB IV

SOURCE CODE DAN PENGETESAN KODE

4.1. Source Code

fixed_points.py

```
1 import numpy as np
2 from tkinter import messagebox
3 import time
4
5 cache_brute_force = {}
6 cache_dnc = {}
7
8 def calculate_num_points(iteration):
9     return 2 ** (iteration) + 1
10
11 def bezier_quadratic_brute_force(P0, P1, P2, iteration):
12     P0, P1, P2 = np.array(P0), np.array(P1), np.array(P2)
13     for _ in range(10):
14         num_points = calculate_num_points(iteration)
15         t_values = np.linspace(0, 1, num_points)
16         curve_points = []
17         for t in t_values:
18             x = (1 - t)**2 * P0[0] + 2 * (1 - t) * t * P1[0] + t**2 * P2[0]
19             y = (1 - t)**2 * P0[1] + 2 * (1 - t) * t * P1[1] + t**2 * P2[1]
20             curve_points.append([x, y])
21         curve_points = np.array(curve_points)
22     return curve_points
23
24 import time
25
26 def bezier_quadratic_divide_and_conquer(P0, P1, P2, iteration):
27     P0, P1, P2 = np.array(P0), np.array(P1), np.array(P2)
28     if iteration == 0:
29         return np.array([P0, P2])
30     else:
31         P01 = (P0 + P1) / 2
32         P12 = (P1 + P2) / 2
33         P0112 = (P01 + P12) / 2
34         left_points = bezier_quadratic_divide_and_conquer(P0, P01, P0112, iteration - 1)
35         right_points = bezier_quadratic_divide_and_conquer(P0112, P12, P2, iteration - 1)
36         return np.concatenate((left_points, right_points))
37
38 def parse_control_points(entry):
39     if not entry.strip():
40         return None
41     try:
42         x, y = map(float, entry.strip().split(','))
43         return [x, y]
44     except ValueError:
45         messagebox.showerror("Error", "Control points are invalid or not in expected format.")
46     return None
```

```

1 def update_plot(fig, canvas, iteration_slider, entry_P0, entry_P1, entry_P2):
2     iteration = int(iteration_slider.get())
3     P0 = parse_control_points(entry_P0.get())
4     P1 = parse_control_points(entry_P1.get())
5     P2 = parse_control_points(entry_P2.get())
6     if None in (P0, P1, P2):
7         return
8
9     fig.clear()
10
11     ax1 = fig.add_subplot(121)
12     if iteration not in cache_brute_force:
13         start_time_brute_force = time.perf_counter()
14         points_brute_force = bezier_quadratic_brute_force(P0, P1, P2, iteration)
15         brute_force_time = time.perf_counter() - start_time_brute_force
16         cache_brute_force[iteration] = {'points': points_brute_force, 'time': brute_force_time}
17     else:
18         points_brute_force = cache_brute_force[iteration]['points']
19         brute_force_time = cache_brute_force[iteration]['time']
20     ax1.plot(points_brute_force[:, 0], points_brute_force[:, 1], 'go-', label=f"Brute Force (Iter: {iteration})")
21     ax1.plot([P0[0], P1[0], P2[0]], [P0[1], P1[1], P2[1]], 'ro-', label="Control Points")
22     ax1.legend()
23     ax1.set_title(f"Brute Force\nExecution Time: {brute_force_time:.6f} sec")
24
25     ax2 = fig.add_subplot(122)
26     if iteration not in cache_dnc:
27         start_time_dnc = time.perf_counter()
28         points_dnc = bezier_quadratic_divide_and_conquer(P0, P1, P2, iteration)
29         dnc_time = time.perf_counter() - start_time_dnc
30         cache_dnc[iteration] = {'points': points_dnc, 'time': dnc_time}
31     else:
32         points_dnc = cache_dnc[iteration]['points']
33         dnc_time = cache_dnc[iteration]['time']
34     ax2.plot(points_dnc[:, 0], points_dnc[:, 1], 'bo-', label=f"Divide & Conquer (Iter: {iteration})")
35     ax2.plot([P0[0], P1[0], P2[0]], [P0[1], P1[1], P2[1]], 'ro-', label="Control Points")
36     ax2.legend()
37     ax2.set_title(f"Divide & Conquer\nExecution Time: {dnc_time:.6f} sec")
38
39     canvas.draw()
40
41 def initialize_visualization(fig, canvas, iteration_slider, entry_max_iterations, entry_P0,
42 entry_P1, entry_P2):
43     global cache_brute_force, cache_dnc
44     try:
45         max_iterations = int(entry_max_iterations.get())
46     except ValueError:
47         messagebox.showerror("Error", "Iteration must be an integer")
48         return
49     iteration_slider.config(to=max_iterations)
50     iteration_slider.set(max_iterations)
51     cache_brute_force = {}
52     cache_dnc = {}
53     update_plot(fig, canvas, iteration_slider, entry_P0, entry_P1, entry_P2)

```

n_points.py

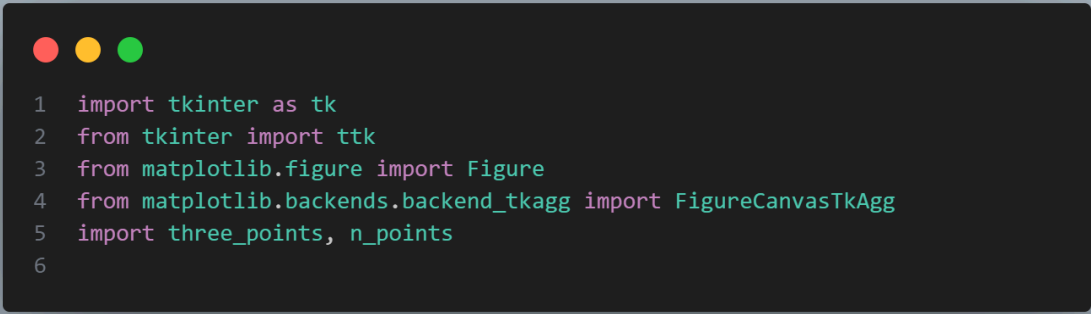
```
1 import numpy as np
2 from tkinter import messagebox
3 import time
4
5 cache_n_point = {}
6
7 def de_casteljau(control_points, t):
8     n = len(control_points)
9     if n == 1:
10         return control_points[0]
11     else:
12         new_points = []
13         for i in range(n - 1):
14             new_point = (1 - t) * np.array(control_points[i]) + t * np.array(control_points[i + 1])
15             new_points.append(new_point)
16         return de_casteljau(new_points, t)
17
18 def bezier_general_dnc(control_points, iteration):
19     num_segments = 2 ** (iteration - 1)
20     num_points = num_segments * (len(control_points) - 1) + 1
21     num_points = int(num_points)
22     t_values = np.linspace(0, 1, num_points)
23     curve_points = [de_casteljau(control_points, t) for t in t_values]
24     return np.array(curve_points)
25
26
27 def parse_control_points_general(entry):
28     try:
29         points = entry.strip().split(';')
30         control_points = [list(map(float, p.split(','))) for p in points if p.strip() != ""]
31         if not control_points or any(len(p) != 2 for p in control_points):
32             return None
33         return control_points
34     except ValueError:
35         messagebox.showerror("Error", "Control points are invalid or not in expected format.")
36         return None
```

```

1 def update_plot(entry_control_points, fig, canvas, iteration_slider):
2     iteration = int(iteration_slider.get())
3     control_points = parse_control_points_general(entry_control_points.get())
4     if not control_points:
5         return
6
7     fig.clear()
8
9     ax = fig.add_subplot(111)
10    control_points_str = str(control_points)
11    if control_points_str not in cache_n_point:
12        cache_n_point[control_points_str] = {}
13
14    if iteration == 0:
15        points_dnc = np.array([control_points[0], control_points[-1]])
16        dnc_time = 0
17    else:
18        if iteration not in cache_n_point[control_points_str]:
19            start_time_dnc = time.perf_counter()
20            points_dnc = bezier_general_dnc(control_points, iteration)
21            dnc_time = time.perf_counter() - start_time_dnc
22            cache_n_point[control_points_str][iteration] = {'points': points_dnc, 'time': dnc_time}
23        else:
24            points_dnc = cache_n_point[control_points_str][iteration]['points']
25            dnc_time = cache_n_point[control_points_str][iteration]['time']
26
27    ax.plot(points_dnc[:, 0], points_dnc[:, 1], 'go-', label=f"Divide & Conquer (Iter: {iteration})")
28    ax.plot([p[0] for p in control_points], [p[1] for p in control_points], 'ro-', label="Control Points")
29    ax.legend()
30    ax.set_title(f"Divide & Conquer\nExecution Time: {dnc_time:.5f} sec")
31
32    canvas.draw()
33
34
35 def visualize_with_max_iteration(entry_iteration, iteration_slider, entry_control_points, fig, canvas):
36     try:
37         max_iterations = int(entry_iteration.get())
38     except ValueError:
39         messagebox.showerror("Error", "Iteration must be an integer")
40         return
41     iteration_slider.config(to=max_iterations)
42     iteration_slider.set(max_iterations)
43     update_plot(entry_control_points, fig, canvas, iteration_slider)
44

```

main.py



```
1  import tkinter as tk
2  from tkinter import ttk
3  from matplotlib.figure import Figure
4  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
5  import three_points, n_points
6
```



```

1  def setup_three_points_tab(tab):
2      fig = Figure(figsize=(12, 6), dpi=100)
3      canvas = FigureCanvasTkAgg(fig, master=tab)
4      widget = canvas.get_tk_widget()
5      widget.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
6
7      controls_frame = tk.Frame(tab)
8      controls_frame.pack(side=tk.TOP, fill=tk.X)
9
10     label_P0 = tk.Label(controls_frame, text="P0 (x,y):")
11     label_P0.pack(side=tk.LEFT, padx=5)
12     entry_P0 = tk.Entry(controls_frame)
13     entry_P0.pack(side=tk.LEFT, padx=5)
14     entry_P0.insert(0, "")
15
16     label_P1 = tk.Label(controls_frame, text="P1 (x,y):")
17     label_P1.pack(side=tk.LEFT, padx=5)
18     entry_P1 = tk.Entry(controls_frame)
19     entry_P1.pack(side=tk.LEFT, padx=5)
20     entry_P1.insert(0, "")
21
22     label_P2 = tk.Label(controls_frame, text="P2 (x,y):")
23     label_P2.pack(side=tk.LEFT, padx=5)
24     entry_P2 = tk.Entry(controls_frame)
25     entry_P2.pack(side=tk.LEFT, padx=5)
26     entry_P2.insert(0, "")
27
28     label_max_iterations = tk.Label(controls_frame, text="No of Iterations:")
29     label_max_iterations.pack(side=tk.LEFT, padx=5)
30
31     entry_max_iterations = tk.Entry(controls_frame, width=5)
32     entry_max_iterations.pack(side=tk.LEFT, padx=5)
33     entry_max_iterations.insert(0, "0")
34
35     button_init = tk.Button(controls_frame, text="Visualize", command=lambda:
three_points.initialize_visualization(fig, canvas, iteration_slider, entry_ma
x_iterations, entry_P0, entry_P1, entry_P2))
36     button_init.pack(side=tk.LEFT, padx=5)
37
38     iteration_slider_label = tk.Label(controls_frame, text="Iteration:")
39     iteration_slider_label.pack(side=tk.LEFT, padx=5)
40
41     iteration_slider = ttk.Scale(controls_frame, from_=0, to=5, orient='horiz
ontal', command=lambda event=None: three_points.update_plot(fig, canvas, iter
ation_slider, entry_P0, entry_P1, entry_P2))
42     iteration_slider.pack(side=tk.LEFT, padx=5, fill='x', expand=True)
43     iteration_slider.set(0)
44
45     pass

```



```
1 def setup_n_points_tab(tab):
2     fig = Figure(figsize=(12, 6), dpi=100)
3     canvas = FigureCanvasTkAgg(fig, master=tab)
4     widget = canvas.get_tk_widget()
5     widget.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
6
7     input_frame = tk.Frame(tab)
8     input_frame.pack(side=tk.TOP, fill=tk.X)
9
10    label_control_points = tk.Label(input_frame, text="Control Points (x1,y1;
x2,y2;...):")
11    label_control_points.pack(side=tk.LEFT, padx=2)
12    entry_control_points = tk.Entry(input_frame, width=50)
13    entry_control_points.pack(side=tk.LEFT, padx=2)
14    entry_control_points.insert(0, "")
15
16    label_iteration = tk.Label(input_frame, text="No of Iteration:")
17    label_iteration.pack(side=tk.LEFT, padx=2)
18    entry_iteration = tk.Entry(input_frame)
19    entry_iteration.pack(side=tk.LEFT, padx=2)
20    entry_iteration.insert(0, "0")
21
22
23    button_visualize = tk.Button(input_frame, text="Visualize", command=lambda
a: n_points.visualize_with_max_iteration(entry_iteration, iteration_slider, e
ntry_control_points, fig, canvas))
24    button_visualize.pack(side=tk.LEFT, padx=4)
25
26    iteration_slider_label = tk.Label(input_frame, text="Iteration:")
27    iteration_slider_label.pack(side=tk.LEFT, padx=5)
28
29    iteration_slider = ttk.Scale(input_frame, from_=0, to=5, orient='horizont
al', command=lambda event=None: n_points.update_plot(entry_control_points, fi
g, canvas, iteration_slider))
30    iteration_slider.pack(side=tk.LEFT, padx=5, fill=tk.X, expand=True)
31    iteration_slider.set(0)
32    pass
```

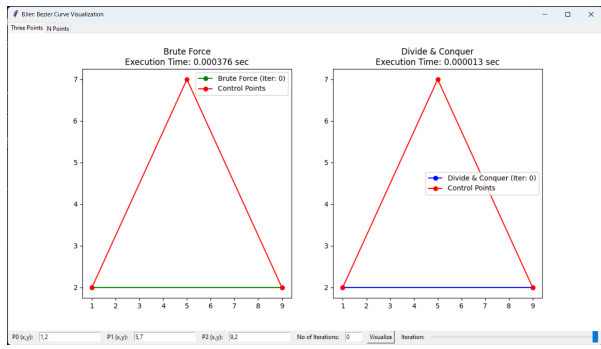


```

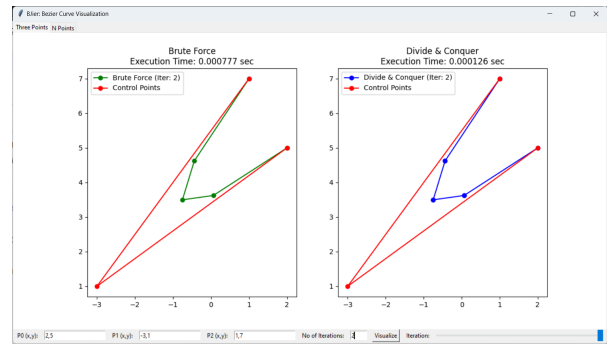
1  def main():
2      root = tk.Tk()
3      root.title("BJier: Bezier Curve Visualization")
4
5      notebook = ttk.Notebook(root)
6      notebook.pack(fill='both', expand=True)
7
8      tab_three_points = ttk.Frame(notebook)
9      tab_n_points = ttk.Frame(notebook)
10
11     notebook.add(tab_three_points, text='Three Points')
12     notebook.add(tab_n_points, text='N Points')
13
14     setup_three_points_tab(tab_three_points)
15     setup_n_points_tab(tab_n_points)
16
17     root.mainloop()
18
19 if __name__ == "__main__":
20     main()
21

```

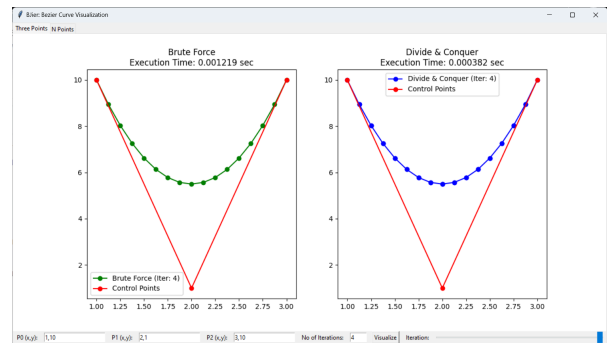
4.2. Pengetesan Kode

Input	Output
<p> $P_0 = (1,2)$ $P_1 = (5,7)$ $P_2 = (9,2)$ Iterasi = 0 Tipe = Kuadratik </p>	

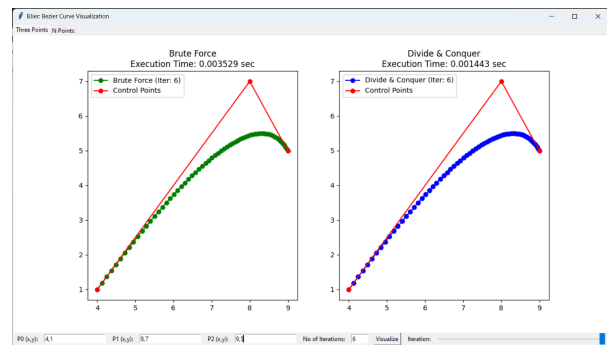
$P_0 = (2,5)$
 $P_1 = (-3,1)$
 $P_2 = (1,7)$
 Iterasi = 2
 Tipe = Kuadratik



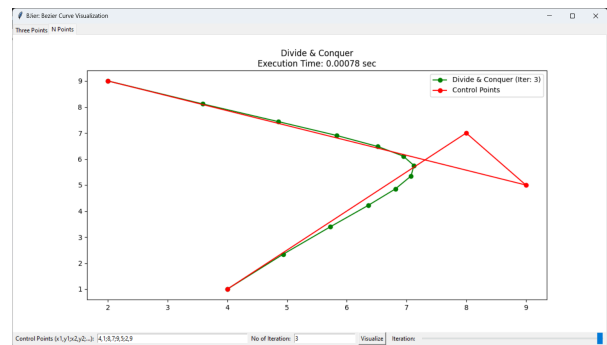
$P_0 = (1,10)$
 $P_1 = (2,1)$
 $P_2 = (3,10)$
 Iterasi = 4
 Tipe = Kuadratik



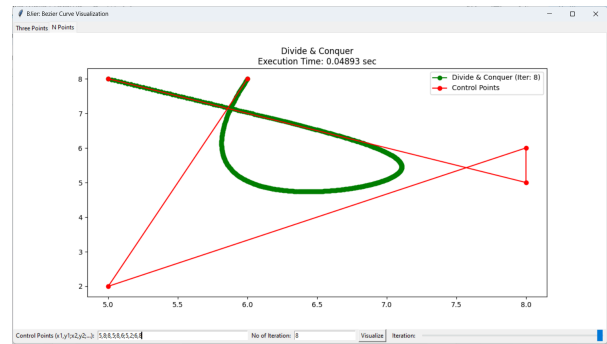
$P_0 = (4,1)$
 $P_1 = (8,7)$
 $P_2 = (9,5)$
 Iterasi = 6
 Tipe = Kuadratik



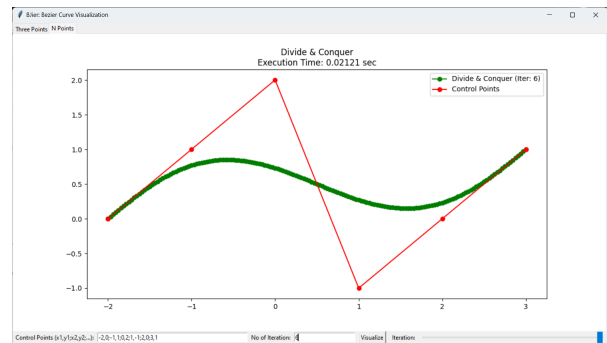
$P_0 = (4,1)$
 $P_1 = (8,7)$
 $P_2 = (9,5)$
 $P_3 = (2,9)$
 Iterasi = 3
 Tipe = 4 Titik



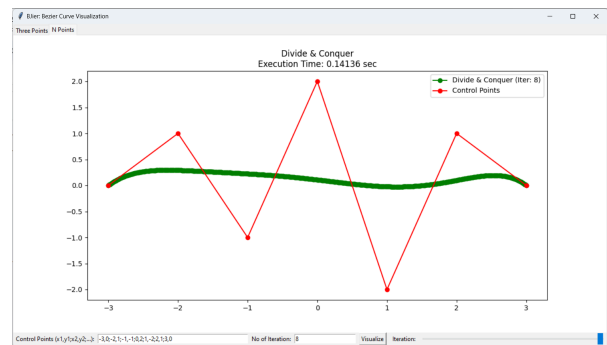
$P_0 = (5,8)$
 $P_1 = (8,5)$
 $P_2 = (8,6)$
 $P_3 = (5,2)$
 $P_4 = (6,8)$
 Iterasi = 8
 Tipe = 5 Titik



$P_0 = (-2,0)$
 $P_1 = (-1,1)$
 $P_2 = (0,2)$
 $P_3 = (1,-1)$
 $P_4 = (2,0)$
 $P_5 = (3,1)$
 Iterasi = 6
 Tipe = 6 Titik



$P_0 = (-3,0)$
 $P_1 = (-2,1)$
 $P_2 = (-1,-1)$
 $P_3 = (0,2)$
 $P_4 = (1,-2)$
 $P_5 = (2,1)$
 $P_6 = (3,0)$
 Iterasi = 8
 Tipe = 7 Titik



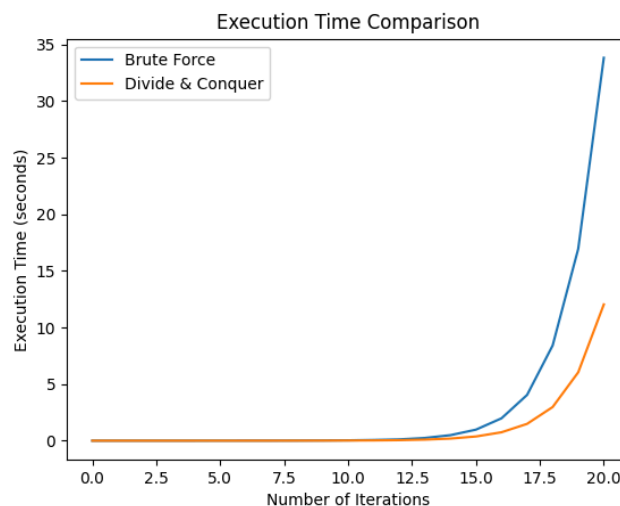
BAB V

HASIL ANALISIS ALGORITMA

5.1. Analisis Perbandingan Algoritma Divide and Conquer dengan Algoritma Brute Force dalam Pembentukan Kurva Bézier

Algoritma *brute force* dan *divide and conquer* merupakan dua pendekatan yang umum digunakan dalam menyelesaikan masalah-masalah algoritmik. Dalam konteks implementasi kurva bézier, keduanya dapat diaplikasikan untuk menghasilkan kurva yang serupa namun dengan perbedaan dalam pendekatan komputasinya. Analisis perbandingan antara solusi *brute force* dan *divide and conquer* untuk implementasi kurva bézier ini akan melibatkan beberapa aspek, termasuk kompleksitas algoritma, kinerja, dan kemungkinan *trade-off* yang terjadi.

Pertama-tama, tinjau kompleksitas algoritma kedua algoritma. Algoritma *brute force* dalam implementasi kurva bézier memerlukan iterasi melalui sejumlah titik pada kurva sesuai dengan parameter iterasi yang diberikan. Kompleksitas waktu algoritma *brute force* dapat dianggap sebagai $O(n)$, di mana n adalah jumlah titik yang dihasilkan oleh kurva pada setiap iterasi. Hal ini karena pada setiap iterasi, algoritma harus mengevaluasi polinomial bézier pada sejumlah titik tertentu untuk membangun kurva. Di sisi lain, metode *divide and conquer* memecah masalah menjadi submasalah yang lebih kecil, dengan membagi kurva menjadi dua bagian pada setiap iterasi. Kompleksitas waktu algoritma *divide and conquer* untuk implementasi kurva Bézier ini dapat dianggap sebagai $O(\log n)$, di mana n adalah jumlah titik yang dihasilkan pada setiap iterasi. Hal ini disebabkan karena kurva bézier dipartisi menjadi subkurva yang semakin kecil secara eksponensial dengan setiap iterasi, sehingga membutuhkan jumlah operasi yang lebih sedikit dibandingkan dengan pendekatan *brute force*.



Gambar Perbandingan Kompleksitas Waktu Algoritma *Brute Force* dan *Divide and Conquer*
(Sumber: Dokumentasi Pribadi)

Selanjutnya, dari segi kinerja, hasil percobaan menunjukkan bahwa algoritma *divide and conquer* lebih cepat daripada *brute force* dalam menghasilkan kurva bézier. Ini bisa dijelaskan oleh fakta bahwa pendekatan *divide and conquer* secara efisien memanfaatkan pola rekursif dari struktur kurva bézier, yang memungkinkan untuk menghindari beberapa evaluasi yang tidak perlu dibandingkan dengan metode *brute force*. Kompleksitas algoritma *brute force* secara teoritis sama dengan atau lebih buruk daripada *divide and conquer*, pada praktiknya, pendekatan *divide and conquer* menunjukkan kinerja yang lebih baik karena pengoptimalan yang dilakukan dalam pengelolaan submasalah yang lebih kecil.

Terakhir, meskipun metode *divide and conquer* menawarkan kinerja yang lebih baik dalam konteks implementasi kurva bézier, penting untuk mempertimbangkan *trade-off* yang mungkin terjadi. Algoritma *brute force* mungkin lebih sederhana untuk diimplementasikan dan lebih mudah dimengerti bagi pemula, sementara algoritma *divide and conquer* membutuhkan pemahaman yang lebih dalam tentang rekursi dan pembagian masalah. Selain itu, dalam beberapa kasus di mana jumlah titik kurva bézier sangat kecil, perbedaan kinerja antara kedua metode mungkin tidak signifikan secara praktis, dan dalam situasi seperti itu, keunggulan kinerja dari metode *divide and conquer* mungkin tidak terasa.

Kesimpulannya, analisis perbandingan solusi *brute force* dengan *divide and conquer* dalam implementasi kurva bézier menunjukkan bahwa meskipun kedua pendekatan tersebut memiliki kompleksitas algoritma yang berbeda, metode *divide and conquer* secara umum menawarkan kinerja yang lebih baik dalam menghasilkan kurva bézier.

5.2. Analisis Penggunaan Pendekatan Algoritma De Casteljau Terhadap Pembuatan Kurva Bézier dengan N Titik Kontrol Menggunakan Algoritma Divide and Conquer

Algoritma de casteljau merupakan pendekatan yang umum digunakan dalam pembuatan kurva bézier dengan jumlah titik kontrol yang tidak terbatas. Dalam konteks implementasi ini, algoritma de casteljau digunakan untuk menghitung posisi titik-titik pada kurva bézier pada setiap parameter t yang diberikan. Selanjutnya, algoritma *divide and conquer* digunakan untuk memecah masalah secara rekursif dalam pembuatan kurva bézier dengan membagi kurva menjadi segmen-segmen yang lebih kecil.

Pertama-tama, algoritma de casteljau bekerja dengan menghitung posisi titik-titik pada kurva bézier dengan membagi kurva menjadi segmen-segmen yang lebih kecil berdasarkan parameter t . Setiap segmen tersebut kemudian dievaluasi menggunakan rumus de casteljau, yang secara iteratif menghitung posisi titik pada kurva dengan interpolasi linier antara titik kontrol yang ada. Hal ini memungkinkan algoritma untuk secara efisien menghasilkan titik-titik kurva bézier dengan menggunakan teknik pencarian yang efisien.

Kemudian, algoritma *divide and conquer* digunakan untuk memecah masalah pembuatan kurva bézier menjadi submasalah yang lebih kecil. Dalam konteks ini, setiap submasalah terdiri dari segmen-segmen kurva bézier yang lebih kecil, yang kemudian dievaluasi menggunakan algoritma de casteljau untuk menghasilkan titik-titik pada kurva. Proses ini terus berlanjut secara rekursif hingga mencapai titik terminasi, di mana kurva sudah cukup dekat dengan bentuk yang diinginkan atau iterasi telah mencapai batas tertentu.

Dari segi kompleksitas algoritma, pendekatan algoritma de casteljau memiliki kompleksitas waktu yang cenderung lebih rendah daripada pendekatan *brute force* dalam implementasi kurva bézier dengan jumlah titik kontrol yang tidak terbatas. Hal ini disebabkan oleh efisiensi algoritma de casteljau dalam menghitung posisi titik pada kurva dengan memanfaatkan interpolasi linier antara titik kontrol yang ada. Di sisi lain, algoritma *divide and conquer* membagi masalah menjadi submasalah yang lebih kecil, yang memungkinkan pengurangan kompleksitas secara eksponensial dengan setiap iterasi.

Namun, walaupun algoritma de casteljau dapat bekerja efisien dalam pembuatan kurva bézier dengan jumlah titik kontrol yang tidak terbatas, penggunaan algoritma *divide and conquer* dalam konteks ini dapat memberikan keuntungan tertentu. Terutama dalam kasus ketika jumlah titik kontrol sangat besar, algoritma *divide and conquer* dapat membagi masalah menjadi submasalah yang lebih kecil, sehingga memungkinkan untuk mengurangi beban komputasi secara signifikan dibandingkan dengan algoritma de casteljau. Hal ini terutama terlihat saat mencapai iterasi yang lebih tinggi, di mana pembagian kurva menjadi segmen-segmen yang lebih kecil secara efektif mempercepat proses evaluasi.

Kesimpulannya, penggunaan pendekatan algoritma de casteljau dalam pembuatan kurva bézier dengan jumlah titik kontrol yang tidak terbatas menggunakan algoritma *divide and conquer* menunjukkan potensi untuk meningkatkan efisiensi komputasi. Kombinasi antara efisiensi algoritma de casteljau dalam menghitung posisi titik pada kurva dengan strategi pembagian masalah secara rekursif dari algoritma *divide and conquer* dapat menghasilkan kinerja yang optimal dalam pembuatan kurva bézier dengan jumlah titik kontrol yang besar.

BAB VI

KESIMPULAN DAN SARAN

6.1. Kesimpulan

Dalam analisis perbandingan antara pendekatan brute force dan divide and conquer dalam pembuatan kurva bézier dengan tiga titik kontrol, ditemukan bahwa algoritma divide and conquer menunjukkan kinerja yang lebih baik dalam sebagian besar kasus. Secara teoritis algoritma brute force memiliki kompleksitas waktu yang sama atau lebih buruk daripada *divide and conquer*, dalam praktiknya, algoritma *divide and conquer* mampu memberikan hasil lebih cepat. Faktor ini mungkin dipengaruhi oleh implementasi khusus dari kedua algoritma, karakteristik masukan, dan faktor-faktor lingkungan.

6.2. Saran

Untuk membuat kurva bézier, algoritma *divide and conquer* mungkin bisa dikombinasikan dengan algoritma lain supaya hasilnya lebih optimal, seperti kombinasi antara algoritma *divide and conquer* dengan algoritma de casteljau yang diimplementasikan di bonus tugas ini.

BAB VII

DAFTAR PUSTAKA

7.1. Sumber Pustaka

- [1]R. Munir and Bagian, “Algoritma Divide and Conquer Bahan Kuliah IF2211 Strategi Algoritma,” 2024. Accessed: Mar. 18, 2024. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf)
- [2]O. Nur, U. Maulidevi, R. Munir, and Bagian, “Algoritma Divide and Conquer Bahan Kuliah IF2211 Strategi Algoritma,” 2024. Accessed: Mar. 18, 2024. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf)
- [3]R. Munir, “Bahan Kuliah IF2211 Strategi Algoritma Algoritma Brute Force.” Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-\(2022\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf)
- [4]JavaScript.Info, “Bezier curve,” *javascript.info*, Nov. 30, 2022. <https://javascript.info/bezier-curve>
- [5]“Finding a Point on a Bézier Curve: De Casteljau’s Algorithm,” *pages.mtu.edu*. <https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/de-casteljau.html>

7.2. Lampiran

1. Github: https://github.com/rizqikapratamaa/Tucil2_13522126
- 2.

Poin	Ya	Tidak
1. Program berhasil dijalankan	✓	
2. Program dapat menjalankan 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.	✓	