

IF2240 - Strategi Algoritma

Tugas Kecil 2

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

18 Maret 2024



Hugo Sabam Augusto

13522129

Muhammad Dzaki Arta

13522149

**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024**

Bab 1

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. Kurva Bézier memiliki banyak kegunaan dalam kehidupan nyata, seperti pen tool, animasi yang halus dan realistis, membuat desain produk yang kompleks dan presisi, dan membuat font yang indah dan unik. Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan.

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P_0 sampai P_n , dengan n disebut order ($n = 1$ untuk linier, $n = 2$ untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) umumnya tidak terletak pada kurva. Pada gambar 1 diatas, titik kontrol pertama adalah P_0 , sedangkan titik kontrol terakhir adalah P_3 . Titik kontrol P_1 dan P_2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P_0 dan P_1 yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan kurva Bézier linier. Misalkan terdapat sebuah titik Q_0 yang berada pada garis yang dibentuk oleh P_0 dan P_1 , maka posisinya dapat dinyatakan dengan persamaan parametrik berikut

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

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh $B(t)$ dari P_0 ke P_1 . Misalnya ketika $t = 0.25$, maka $B(t)$ adalah seperempat jalan dari titik P_0 ke P_1 . sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan $B(t)$ membentuk sebuah garis lurus dari P_0 ke P_1 .

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P_2 , dengan P_0 dan P_2 sebagai titik kontrol awal dan akhir, dan P_1 menjadi titik kontrol antara. Dengan menyatakan titik Q_1 terletak diantara garis yang menghubungkan P_1 dan P_2 , dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q_0 berada, maka dapat dinyatakan sebuah titik baru, R_0 yang berada diantara garis yang menghubungkan Q_0 dan Q_1 yang bergerak membentuk kurva Bézier kuadratik terhadap titik P_0 dan P_2 . Berikut adalah uraian persamaannya.

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

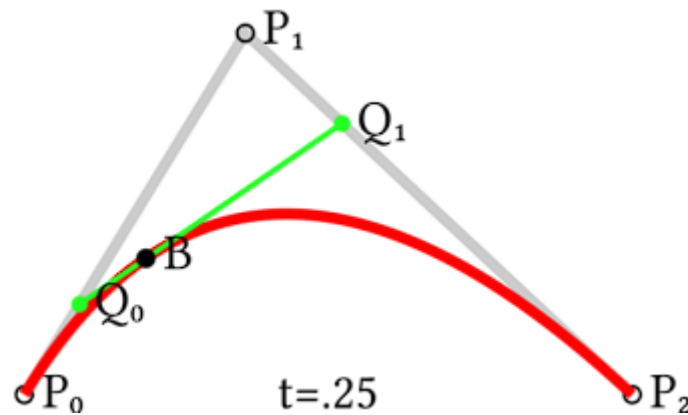
$$Q_1 = B(t) = (1 - t)P_1 + tP_2, \quad t \in [0, 1]$$

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

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

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

Berikut adalah ilustrasi dari kasus diatas



Gambar 2. Pembentukan Kurva Bézier Kuadrat.

(Sumber: <https://simonhalliday.com/2017/02/15/quadratic-bezier-curve-demo/>)

Proses ini dapat juga diaplikasikan untuk jumlah titik yang lebih dari tiga, misalnya empat titik akan menghasilkan kurva Bézier kubik, lima titik akan menghasilkan kurva Bézier kuartik, dan seterusnya. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya.

$$S_0 = B(t) = (1 - t)^3 P_0 + 3(1 - t)^2 t P_1 + 3(1 - t) t^2 P_2 + t^3 P_3, \quad t \in [0, 1]$$

$$T_0 = B(t) = (1 - t)^4 P_0 + 4(1 - t)^3 t P_1 + 6(1 - t)^2 t^2 P_2 + 4(1 - t) t^3 P_3 + t^4 P_4, \quad t \in [0, 1]$$

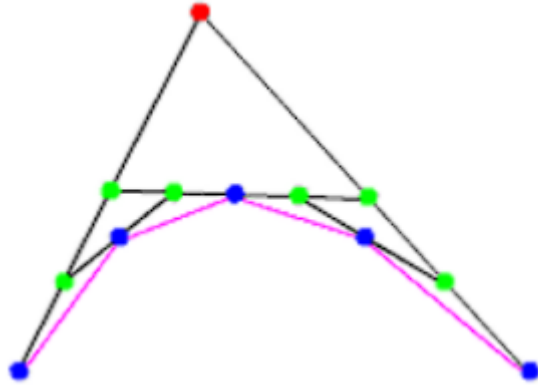
Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka Anda diminta untuk mengimplementasikan pembuatan kurva Bézier dengan algoritma titik tengah berbasis divide and conquer.

Ilustrasi Kasus:

Idenya cukup sederhana, relatif mirip dengan pembahasan sebelumnya, dan dilakukan secara iteratif. Misalkan terdapat tiga buah titik, P0, P1, dan P2, dengan titik P1 menjadi titik kontrol antara, maka:

- Buatlah sebuah titik baru Q0 yang berada di tengah garis yang menghubungkan P0 dan P1, serta titik Q1 yang berada di tengah garis yang menghubungkan P1 dan P2.
- Hubungkan Q0 dan Q1 sehingga terbentuk sebuah garis baru.
- Buatlah sebuah titik baru R0 yang berada di tengah Q0 dan Q1.
- Buatlah sebuah garis yang menghubungkan P0 - R0 - P2.

Melalui iterasi kedua akan tampak semakin mendekati sebuah kurva, dengan aproksimasi 5 buah titik. Anda dapat membuat visualisasi atau gambaran secara mandiri terkait hal ini sehingga dapat diamati dan diterka dengan jelas bahwa semakin banyak iterasi yang dilakukan, maka akan membentuk sebuah kurva yang tidak lain adalah kurva Bézier.



Gambar 3. Hasil pembentukan Kurva Bézier Kuadratik dengan divide and conquer setelah iterasi ke-2

Bab II

Analisis Algoritma

2.1 Algoritma Brute Force

Algoritma Brute Force yang digunakan dalam tugas ini digunakan sebagai pembanding untuk algoritma Divide and Conquer. Algoritma yang digunakan menggunakan rumus $B(t) = (1 - t)P_0 + tP_1$, $t \in [0,1]$ dan digunakan secara rekursif agar didapatkan titik untuk kurva beziernya. Fungsi brute force yang kami buat menerima dua parameter: *control_points*, yang merupakan array dari titik-titik kontrol kurva Bezier dan *t*, parameter yang menentukan titik mana yang akan dievaluasi pada kurva Bezier. Algoritma beroperasi dengan membagi setiap segmen garis antara titik kontrol secara iteratif (menggunakan fungsi dari module *numpy* yaitu *linspace*), kemudian melakukan pemanggilan rekursif untuk menghitung titik-titik di sepanjang kurva Bezier yang terbentuk dari titik-titik yang terbagi tersebut. Hasilnya adalah titik pada kurva Bezier yang dievaluasi pada parameter *t*.

Penjelasan secara Detailnya adalah:

1. Pertama, diperiksa dahulu jumlah titik kontrolnya (*n*). Apabila hanya ada 2 titik, maka kurva Bezier menjadi 1 garis lurus antara 2 titik kontrol tsb. Dalam kasus ini, fungsi langsung menghitung dan mengembalikan titik pada garis lurus tersebut dengan rumus interpolasi linier
2. Kedua, jika jumlah titik kontrol(*n*) > 2, maka fungsi akan melakukan langkah algoritma berikut:
 - a. fungsi membagi setiap segmen garis antara tiap 2 titik kontrol (*i* dan *i+1*) pada parameter '*t*' dan dilakukan rumus $B(t) = (1 - t)P_0 + tP_1$, $t \in [0,1]$.
 - b. Nanti akan didapatkan titik baru { *B(t)* } tiap diantara 2 titik kontrol
 - c. Setelah itu, akan dilakukan rekursif untuk menghitung titik di sepanjang kurva Bezier yang terbentuk oleh titik-titik baru.
 - d. Pemanggilan rekursif ini dilakukan 2 kali, masing-masing untuk dua setengah kurva yang telah dibagi.
 - e. Hasil dari kedua panggilan rekursif tersebut adalah dua titik, yang masing-masing titiknya adalah titik di sepanjang kurva bezier yang dibentuk oleh dua setengah kurva yang terbagi. Di fungsi ditulis dalam bentuk variabel '*left_curve*' dan '*right_curve*'.
 - f. Akhir, fungsi menghubungkan kedua titik dengan rumus yang sama $B(t) = (1 - t)P_0 + tP_1$, $t \in [0,1]$ sesuai dengan bobot parameter '*t*'

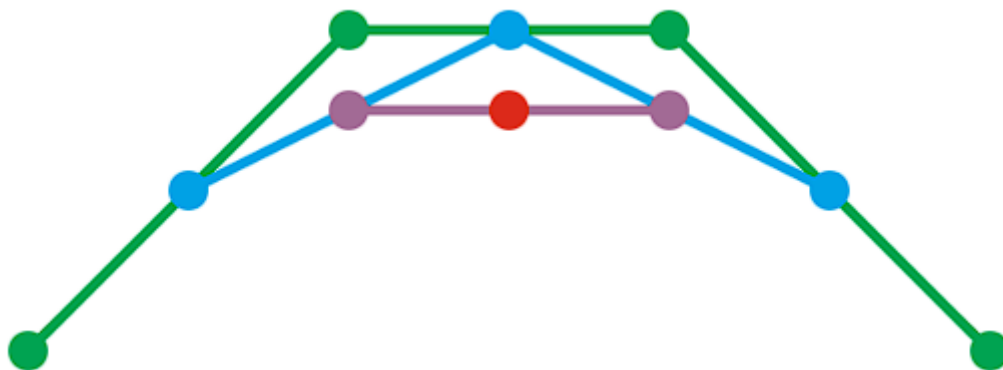
Untuk Penghitungan Kompleksitas Algoritma, 1 evaluasi titik tunggal '**bf**' memiliki kompleksitas $O(n^2)$ dan tiap titik dimasukkan ke dalam fungsi '**bf**' lalu direkursikan. Untuk menghasilkan kurva dengan banyak titik, '**plot_bezier_curve**' (karena tiap iterasi

membutuhkan $(2^{\text{iterasi}} + 1)$ titik) menunjukkan kompleksitas eksponensial-kuadrat, sehingga kurang efisien untuk nilai iterasi tinggi atau titik kontrol dalam jumlah besar.

Untuk visualisasi proses pembuatan kurva bezier, kami membuat *loop* untuk iterasi 1..x, dimana x adalah banyaknya iterasi yang diinginkan oleh input. dalam *loop* tersebut, sudah kami rancang dengan library *matplotlib* agar pergantian kurva tiap iterasi berganti setiap 1 detik.

2.2 Algoritma Divide and Conquer

Algoritma divide and conquer merupakan strategi yang efektif untuk menghasilkan kurva Bezier, terutama untuk kurva dengan jumlah iterasi tinggi. Algoritma ini bekerja dengan membagi kurva menjadi sub-kurva yang lebih kecil secara rekursif, kemudian menggabungkan hasil sub-kurva tersebut untuk menghasilkan kurva Bezier final.



Gambar 4. Kurva Bezier dengan algoritma Divide and Conquer

Sumber : https://en.wikipedia.org/wiki/De_Casteljau's_algorithm

Terdapat 2 fungsi yang kami gunakan untuk algoritma divide and conquer ini yaitu `divide_points` dan `bezier_curve`

fungsi `divide_points` menerima 3 argumen yaitu:

1. `points` : yang merupakan array of point yang berisi titik titik control point
2. `left_ctrl` : yang berisi titik control kiri untuk iterasi selanjutnya yang akan dihitung juga pada fungsi `divide point`
3. `right_ctrl` : yang berisi titik control kanan untuk iterasi selanjutnya yang akan dihitung juga pada fungsi `divide point`

fungsi `divide_points` digunakan untuk menentukan titik akhir berdasarkan titik-titik control dengan cara:

1. Jika panjang array lebih dari 1 maka program akan membuat array kosong yang dinamakan `new_points` yang nantinya akan diisi dengan titik titik tengah dari array of point serta melakukan prosedur selanjutnya
2. kemudian program akan melakukan looping sebanyak panjang array - 1 untuk menentukan titik tengah dengan rumus

$$X_{baru} = (X_i + X_{i+1})/2$$

$$Y_{baru} = (Y_i + Y_{i+1})/2$$

- dengan i merupakan iterasi ke- i pada looping
3. Kemudian menambahkan point pertama pada array of point ke `left_ctrl` pada index terakhir dan menambahkan point terakhir pada array of point ke `right_ctrl` pada index 0
 4. selanjutnya program akan melakukan rekursi dengan me return fungsi `divide_points` dengan parameter `new_points, left_ctrl, right_ctrl`
 5. jika panjang array of point tidak lebih dari 1 maka program akan mengembalikan element array of point pertama, `left_ctrl`, dan `right_ctrl`.

fungsi `bezier_curve` memiliki 3 argumen yaitu

1. `points` : yang merupakan array of point yang berisi titik-titik control point
2. `iterations` : yang merupakan banyak iterasi yang akan dilakukan
3. `main_points` yang berisi point-point akhir yang digunakan untuk membentuk kurva bezier

fungsi `bezier_curve` digunakan untuk membentuk kurva bezier menggunakan algoritma `divide and conquer` dengan memanfaatkan fungsi `divide_points` dengan cara :

1. jika iterasi tidak sama dengan 0 dan panjang `points` lebih dari 1 maka program akan menjalankan prosedur selanjutnya
2. me assign elemen pertama pada `points` ke dalam array ke variabel `left_ctrl`
3. me assign elemen terakhir pada `points` ke dalam array ke variabel `right_ctrl`
4. mencari `point, left_ctrl, right_ctrl` dengan memanggil fungsi `divide_points` dengan parameter `points, left_ctrl, right_ctrl`
5. mencari index elemen pertama pada `points` di `main_points` dan assign ke variabel `index`
6. memasukan point ke dalam `main_points` pada `index+1`
7. melakukan rekursi dengan memanggil fungsi `bezier_curve` dengan parameter `left_ctrl, iteration-1, main_points`
8. melakukan rekursi dengan memanggil fungsi `bezier_curve` dengan parameter `right_ctrl, iteration-1, main_points`

Kompleksitas Waktu :

Fungsi `divide_points` memiliki kompleksitas waktu $O(n)$, di mana n adalah jumlah titik kontrol. Hal ini karena fungsi tersebut mengulangi seluruh daftar titik sekali untuk menghitung titik tengah dan berpotensi mengubah daftar `left_ctrl` dan `right_ctrl`.

Fungsi `bezier_curve` menggunakan rekursi, yang menyebabkan kompleksitas waktu $O(n \log n)$. Berikut uraiannya:

- Pada setiap iterasi:
 - `divide_points` dipanggil, yang berkontribusi pada kompleksitas $O(n)$.
 - Fungsi tersebut secara rekursif memanggil dirinya sendiri pada dua sub-masalah dengan ukuran kira-kira setengahnya (karena pembagian dalam `divide_points`). Hal ini menghasilkan jumlah panggilan rekursif logaritmik ($\log n$) dalam kasus terburuk.
- Kompleksitas waktu keseluruhan menjadi $O(n) * \log n = O(n \log n)$.

Kompleksitas Ruang:

- Fungsi `divide_points` tidak membuat struktur data tambahan yang signifikan, sehingga kompleksitas ruangnya dianggap $O(1)$ (konstan).

- Namun, rekursi dalam `bezier_curve` menimbulkan overhead ruang karena tumpukan pemanggilan fungsi. Dalam kasus terburuk (kedalaman rekursi maksimum), hal ini dapat mencapai $O(\log n)$.

Penjelasan:

- Fungsi `divide_points` menangani satu tingkat pembagian titik kontrol. Fungsi tersebut mengulangi daftar sekali, melakukan operasi waktu konstan seperti perhitungan titik tengah dan modifikasi daftar. Oleh karena itu, kompleksitasnya adalah $O(n)$.
- Fungsi `bezier_curve` menggunakan rekursi untuk membagi titik kontrol menjadi sub-masalah dan menggabungkan hasilnya. Pendekatan rekursif ini menghasilkan faktor logaritmik dalam kompleksitas waktu ($\log n$) karena pembagian berulang pada setiap pemanggilan rekursif.
- Meskipun rekursi tidak membuat salinan tambahan dari seluruh daftar titik kontrol, ia memperkenalkan tumpukan pemanggilan fungsi untuk setiap tingkat rekursif. Overhead ini berkontribusi pada kompleksitas ruang logaritmik ($O(\log n)$).

Bab III

Uji Coba

3.1 Source Code Algoritma Brute Force

#BruteBezier.py

```
import numpy as np
import matplotlib.pyplot as plt
import time
import numpy as np
import matplotlib.pyplot as plt

def bf(control_points, t):
    n = len(control_points) - 1

    if n == 1:
        return (1 - t) * control_points[0] + t * control_points[1]

    points = [(1 - t) * control_points[i] + t * control_points[i + 1]
for i in range(n)]
    left_curve = bf(points, t)
    right_curve = bf(points, t)

    result = (1 - t) * left_curve + t * right_curve
    return result

def plot_bezier_curve(control_points, iteration, ax):
    tot_point = 2 ** iteration + 1
    t_values = np.linspace(0, 1, tot_point)
    curve_points = np.array([bf(control_points, t) for t in
t_values])

    ax.plot(control_points[:, 0], control_points[:, 1], 'ro-',
label='Control Points')
    ax.plot(curve_points[:, 0], curve_points[:, 1], 'b-',
label='Bezier Curve')
    ax.grid(True)
    ax.axis('equal')

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.legend()
```

```

def show_curve(control_points_coord, iteration, ax):
    control_points = np.array(control_points_coord, dtype=np.float64)
    for i in range(1, iteration+1):
        ax.cla()
        ax.set_title("Brute Force")
        tstart = time.time()
        plot_bezier_curve(control_points, i, ax)
        tend = time.time()
        if i == iteration:
            ax.text(0.5, -0.1, f"Execution Time in {i} iterations:
{round(tend - tstart, 5)} seconds", horizontalalignment='center',
verticalalignment='center', transform=ax.transAxes)

        print(i, " | Waktu Eksekusi : ", round(tend - tstart,5), "
s")

    plt.pause(1)

```

3.2 Source Code Algoritma Divide And Conquer

```

import matplotlib.pyplot as plt
import time

# Fungsi pembagian titik-titik
def divide_points(points, left_ctrl, right_ctrl):
    if(len(points) > 1):
        new_points = []
        for i in range(len(points) - 1):
            x = (points[i][0] + points[i+1][0]) / 2
            y = (points[i][1] + points[i+1][1]) / 2
            new_points.append((x, y))
        left_ctrl.append(new_points[0])
        right_ctrl.insert(0, new_points[-1])
        return divide_points(new_points, left_ctrl, right_ctrl)
    else:
        return points[0], left_ctrl, right_ctrl

```

```

# Fungsi pembentukan kurva Bezier dengan algoritma divide and conquer
def bezier_curve(points, iterations, main_points):
    if(iterations != 0 and len(points) > 1):
        left_ctrl = [points[0]]
        right_ctrl = [points[-1]]
        point, left_ctrl, right_ctrl = divide_points(points,
left_ctrl, right_ctrl)
        index = main_points.index(points[0])
        main_points.insert(index+1, point)
        bezier_curve(left_ctrl, iterations-1, main_points)
        bezier_curve(right_ctrl, iterations-1, main_points)

# Fungsi untuk menampilkan plot
def plot_curve_and_points(curve_points, control_points, ax):
    ax.plot([p[0] for p in curve_points], [p[1] for p in
curve_points], 'r-', label='Kurva Bezier')
    ax.plot([p[0] for p in control_points], [p[1] for p in
control_points], 'bo-', label='Titik-titik kontrol')
    for i in range(len(control_points) - 1):
        ax.plot([control_points[i][0], control_points[i+1][0]],
[control_points[i][1], control_points[i+1][1]], 'g--')
    ax.legend()
    # ax.title('Kurva Bezier dan Titik-titik Kontrol')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.grid(True)
    ax.axis('equal')

# fungsi membuat list yang unique
def unique(list1):
    list_set = set(list1)
    unique_list = (list(list_set))
    return unique_list

# mengurutkan list berdasarkan nilai x
def sortX(list1):
    list1.sort(key=lambda x: x[0])
    return list1

def sortY(list1):
    list1.sort(key=lambda x: x[1])
    return list1

def show_curve(control_points, iteration, ax):
    for i in range(1, iteration + 1):
        curve_points = [control_points[0], control_points[-1]]
        ax.cla()
        ax.set_title("Divide and Conquer")

```

```

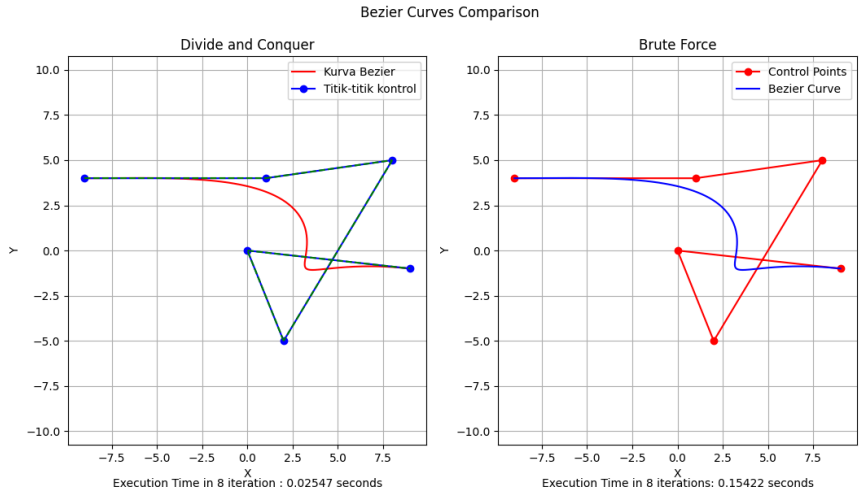
tstart = time.time()
bezier_curve(control_points, i, curve_points)
plot_curve_and_points(curve_points, control_points, ax)
tend = time.time()
if i == iteration:
    ax.text(0.5, -0.1, f"Execution Time in {i} iteration :
{round(tend - tstart, 5)} seconds", horizontalalignment='center',
verticalalignment='center', transform=ax.transAxes)

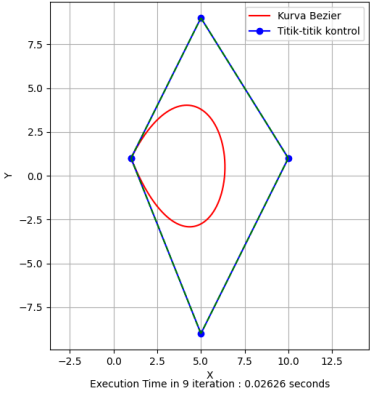
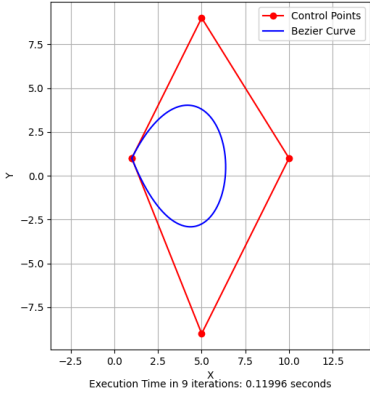
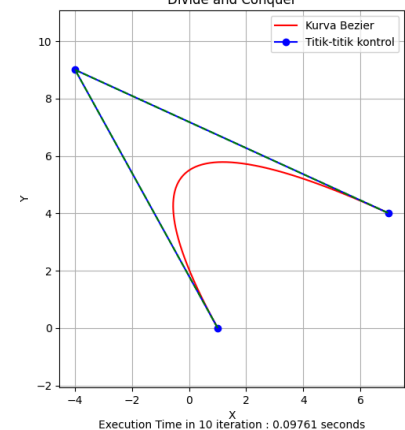
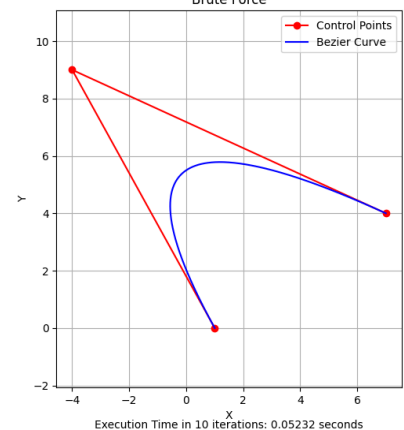
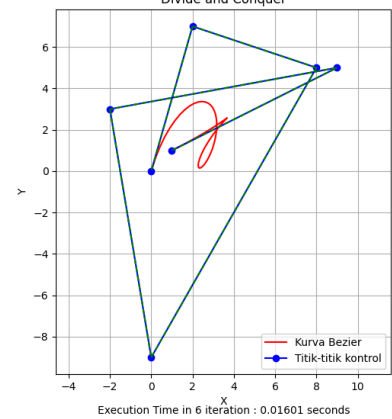
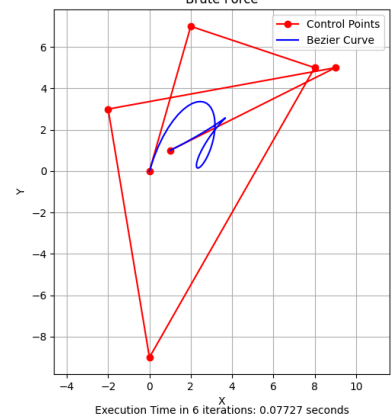
    print(i, " | Waktu Eksekusi : ", round(tend - tstart, 5),
"s")

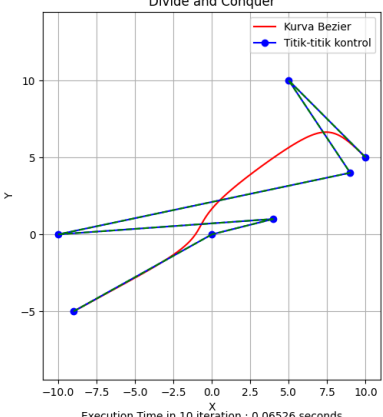
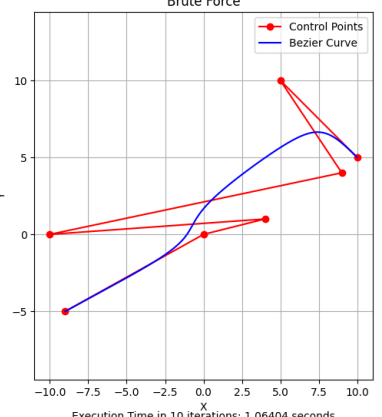
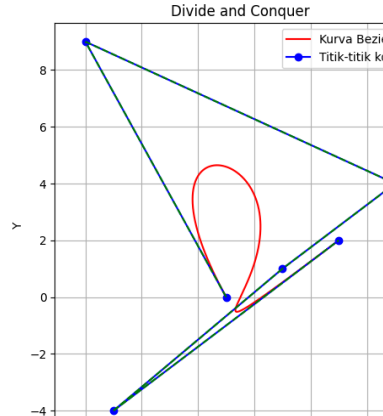
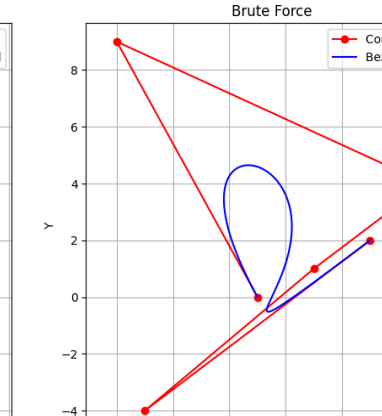
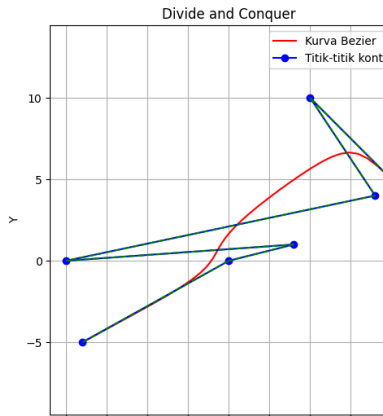
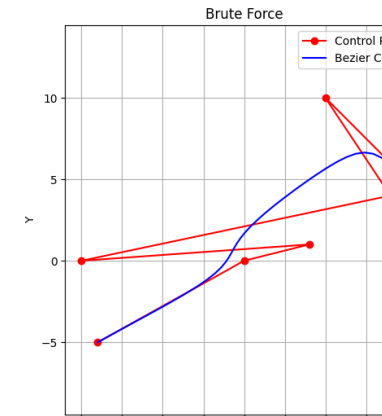
plt.pause(1)

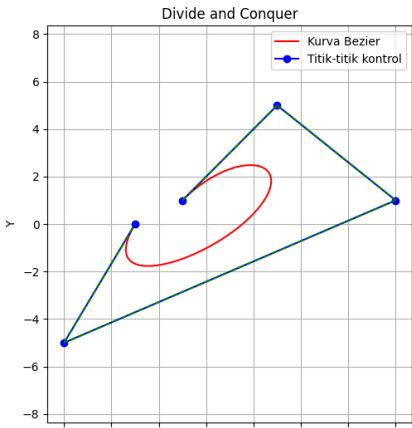
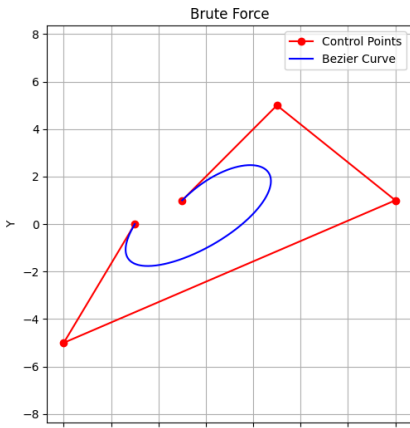
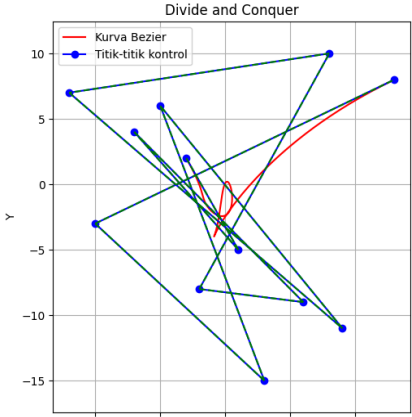
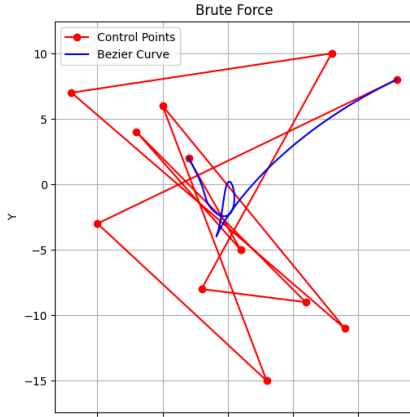
```

3.3 Hasil Beberapa Uji Coba

1.	<p>Titik:</p> <p>-9 4</p> <p>1 4</p> <p>8 5</p> <p>2 -5</p> <p>0 0</p> <p>9 -1</p> <p>Iterasi</p> <p>8</p> <p>DNC : 0.02547</p> <p>S</p> <p>BF : 0.15422 s</p>	<p>Bezier Curves Comparison</p> 
----	--	--

2.	<p>Titik: 1 1 5 9 10 1 5 -9 1 1</p> <p>Iterasi: 9</p> <p>DNC: 0.02626 s BF:0.11996 s</p>	<p>Bezier Curves Comparison</p> <div> <div> <p>Divide and Conquer</p>  <p>Execution Time in 9 iteration : 0.02626 seconds</p> </div> <div> <p>Brute Force</p>  <p>Execution Time in 9 iterations: 0.11996 seconds</p> </div> </div>
3.	<p>Titik: 1 0 -4 9 7 4</p> <p>Iterasi: 10</p> <p>DNC:0.09761 s BF:0.05232 s</p>	<p>Bezier Curves Comparison</p> <div> <div> <p>Divide and Conquer</p>  <p>Execution Time in 10 iteration : 0.09761 seconds</p> </div> <div> <p>Brute Force</p>  <p>Execution Time in 10 iterations: 0.05232 seconds</p> </div> </div>
4.	<p>Titik: 1 1 9 5 -2 3 0 -9 8 5 2 7 0 0</p> <p>Iterasi: 6</p> <p>DNC:0.01601 s BF:0.07727 s</p>	<p>Bezier Curves Comparison</p> <div> <div> <p>Divide and Conquer</p>  <p>Execution Time in 6 iteration : 0.01601 seconds</p> </div> <div> <p>Brute Force</p>  <p>Execution Time in 6 iterations: 0.07727 seconds</p> </div> </div>

<p>5.</p> <p>Titik: 10 5 5 10 9 4 -10 0 4 1 0 0 -9 -5</p> <p>Iterasi: 10</p> <p>DNC:0.06526 s BF:1.06404 s</p>		<p>Bezier Curves Comparison</p> <div> <div> <p>Divide and Conquer</p>  <p>Execution Time in 10 iteration : 0.06526 seconds</p> </div> <div> <p>Brute Force</p>  <p>Execution Time in 10 iterations: 1.06404 seconds</p> </div> </div>
<p>6.</p> <p>Titik: 1 0 -4 9 7 4 3 1 -3 -4 5 2</p> <p>Iterasi: 10</p> <p>DNC:0.06838 s BF: 0.60108 s</p>		<p>Bezier Curves Comparison</p> <div> <div> <p>Divide and Conquer</p>  <p>Execution Time in 10 Iteration : 0.06838 seconds</p> </div> <div> <p>Brute Force</p>  <p>Execution Time in 10 iterations: 0.60108 seconds</p> </div> </div>
<p>7.</p> <p>Titik: 10 5 5 10 9 4 -10 0 4 1 0 0 -9 -5</p> <p>Iterasi: 5</p> <p>DNC: 0.01907 s BF: 0.04572 s</p>		<p>Bezier Curves Comparison</p> <div> <div> <p>Divide and Conquer</p>  <p>Execution Time in 5 iteration : 0.01907 seconds</p> </div> <div> <p>Brute Force</p>  <p>Execution Time in 5 iterations: 0.04572 seconds</p> </div> </div>

8.	<p>Titik: 1 1 5 5 10 1 -4 -5 -1 0</p> <p>Iterasi: 12</p> <p>DNC:0.54882 s BF:0.94915 s</p>	<p>Bezier Curves Comparison</p> <div style="display: flex; justify-content: space-around;"> <div data-bbox="611 248 1023 719"> <p>Divide and Conquer</p>  <p>Execution Time in 12 iteration : 0.54882 seconds</p> </div> <div data-bbox="1050 248 1461 719"> <p>Brute Force</p>  <p>Execution Time in 12 iterations: 0.94915 seconds</p> </div> </div>
9.	<p>Titik: -3 2 1 -5 -7 4 6 -9 -2 -8 8 10 -12 7 9 -11 -5 6 3 -15 -10 -3 13 8</p> <p>Iterasi: 10</p> <p>DNC:0.05465 s BF: 9.28001 s</p>	<p>Bezier Curves Comparison</p> <div style="display: flex; justify-content: space-around;"> <div data-bbox="603 808 1015 1267"> <p>Divide and Conquer</p>  <p>Execution Time in 8 iteration : 0.05465 seconds</p> </div> <div data-bbox="1031 808 1442 1267"> <p>Brute Force</p>  <p>Execution Time in 8 iterations: 9.28001 seconds</p> </div> </div>

10.

Titik:

-3 0

3 0

-3 3

3 -3

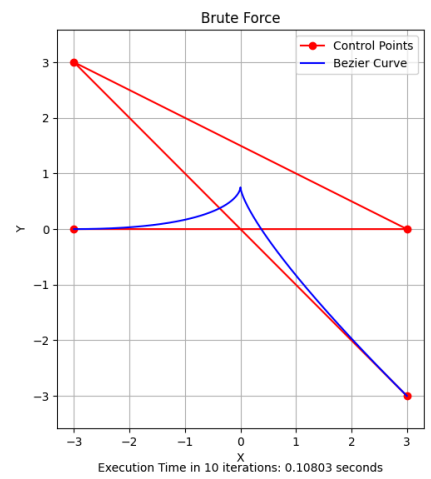
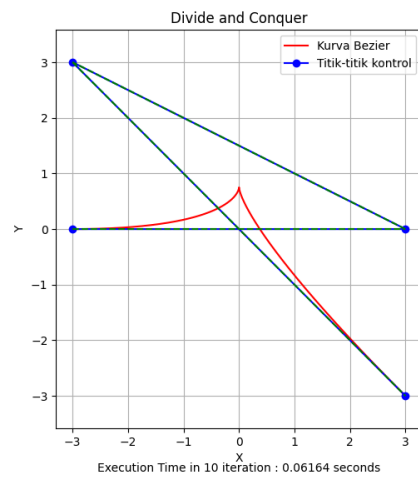
Iterasi:

10

DNC:0.06164 s

BF:0.10803 s

Bezier Curves Comparison



Bab IV

Pembahasan dan Kesimpulan

4.1 Pembahasan

Hampir dari seluruh percobaan dari 1-10, Algoritma Divide and Conquer **lebih unggul** dalam membentuk kurva Bezier dibandingkan Brute Force dalam segi waktu. Ada 1 pasang percobaan yang menarik perhatian kami yaitu dalam percobaan 3 dan 6. Dalam percobaan tersebut jumlah iterasi tetap sama, yang dibedakan hanyalah banyaknya control point. Tetapi apabila dilihat dari waktu kecepatan, Dalam percobaan 3 Brute Force lebih unggul sekian millisecond dibanding Divide and Conquer. Namun dalam percobaan 6, Divide and Conquer jauh lebih unggul dengan kecepatan hampir 10x lebih cepat dibanding Brute Force (DNC: 0.06838 s, BF: 0.60108 s). Hal ini sudah jelas dikarenakan kompleksitas algoritma yang berbeda untuk kedua algoritma tersebut, yang mana Algoritma DNC memiliki kompleksitas $O(n \log n)$, sedangkan untuk Algoritma BF memiliki kompleksitas $O(n^2)$. Bukti ini diperkuat dengan hasil percobaan 9. Dalam percobaan tersebut kami membentuk kurva Bezier dengan titik kontrol yang cukup banyak yaitu 12 titik dengan jumlah iterasi sebanyak 10 kali. Hasilnya sangat jomplang yaitu BF dengan waktu hampir 10 detik sedangkan DNC hanya sekitar 0.05 detik.

4.2 Kesimpulan

Dari hasil uji coba, terlihat bahwa ketika jumlah control point sedikit, brute force mungkin lebih cepat dalam membentuk kurva Bezier dibandingkan dengan algoritma Divide and Conquer. Namun, ketika jumlah control point cukup besar, algoritma Divide and Conquer menunjukkan kinerja yang jauh lebih baik dengan kompleksitas waktu yang tumbuh secara logaritmik, sementara brute force menjadi sangat tidak efisien karena kompleksitasnya yang meningkat secara kuadrat. Oleh karena itu, dalam pemilihan algoritma untuk pembentukan kurva Bezier, penting untuk mempertimbangkan jumlah control point yang akan digunakan untuk mendapatkan kinerja yang optimal.

Daftar Pustaka

<https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm-data-structure-and-algorithm-tutorials/>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2012-2013/Aplikasi%20Divide%20and%20Conquer.pptx>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Aplikasi-Divide-and-Conquer-2020.pdf>

Lampiran

1. Tabel Checklist

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.	✓	

2. Pranala Repository Program

https://github.com/miannetopokki/Tucil2_13522129_13522149