

Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

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

Semester II Tahun 2023/2024



disusun oleh:

Nicholas Reymond Sihite (13522144)

INSTITUT TEKNOLOGI BANDUNG

2024

BAB I

Deskripsi Masalah

Kurva Bézier adalah kurva yang digunakan pada bidang grafika komputer untuk menggambar berbagai bentuk abstrak, salah satu penerapannya adalah pembuatan *font* baru. Kurva Bézier dibuat dengan cara menghubungkan titik-titik yang dibuat dari beberapa titik kontrol dengan iterasi tertentu di mana semakin banyak iterasi yang dilakukan, semakin mulus kurva Bézier yang dihasilkan.

Pada konteks informatika, kurva Bézier dapat dibuat dengan setidaknya dua algoritma, yaitu *Divide and Conquer* serta *Brute Force*. Pada Tugas Kecil 2 Strategi Algoritma 2024 ini, mahasiswa diminta untuk menerapkan algoritma *Divide and Conquer* dan *Brute Force* untuk membuat kurva Bézier dengan *Divide and Conquer* sebagai algoritma utama dan *Brute Force* sebagai algoritma pembanding.

BAB II

Analisis Algoritma

2.1 Algoritma *Brute Force*

Titik-titik yang akan dibuat menjadi kurva Bézier pada algoritma *Brute Force* untuk 3 titik dapat dicari dengan menggunakan rumus

$$B(t) = 1(1 - t)^2 P_0 + 2(1 - t)tP_1 + t^2 P_2$$

dengan P_0 , P_1 , dan P_2 adalah titik-titik kontrol dan t adalah nilai yang bervariasi pada rentang 0 sampai 1

Pada pembuatan kurva Bézier, banyak titik yang digambarkan menjadi kurva pada setiap iterasi bertambah dengan pola

$$U_i = 2U_{i-1} - 1$$

dengan $U_0 = 2$ (iterasi ke-0)

Fungsi rekursi tersebut dapat dicari solusinya sehingga berubah menjadi sebuah fungsi baru sebagai berikut

$$U_i = 2^i + 1$$

Untuk menentukan nilai t , perlu diketahui berapa banyak titik yang dibutuhkan pada iterasi tertentu. Setelah mengetahui berapa titik yang dibutuhkan (misal U_i titik), nilai t ditentukan sedemikian sehingga ada U_i buah nilai t pada perhitungan. Contohnya, pada iterasi ke-1, akan ada 3 titik. Dibutuhkan 3 nilai t yang bervariasi antara 0 sampai 1 dengan jarak antar nilai t tetap. Maka, nilai-nilai t yang memenuhi adalah 0, 0.5, dan 1. Mencari jarak antar nilai t dapat dilakukan dengan kalkulasi berikut

$$\Delta t = \frac{1}{U_i - 1}$$

Maka, untuk mencari titik-titik pada iterasi ke-1 untuk 3 titik kontrol, akan dilakukan 3 kali penggunaan rumus $B(t)$, yaitu $B(0)$, $B(0.5)$, dan $B(1)$ dengan $B(0)$ mewakili titik pertama, $B(0.5)$ mewakili titik kedua, dan $B(1)$ mewakili titik ketiga.

Untuk menerapkan algoritma yang sama untuk n buah titik kontrol, perlu dilakukan pengamatan terhadap rumus yang digunakan. Berikut merupakan rumus yang digunakan untuk 4 dan 5 titik kontrol

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3$$

$$B(t) = (1-t)^4 P_0 + 4(1-t)^3 t P_1 + 6(1-t)^2 t^2 P_2 + 4(1-t)t^3 P_3 + t^4 P_4$$

Perhatikan bahwa koefisien-koefisien pada setiap persamaan merupakan perwakilan angka-angka pada segitiga Paskal: pada 3 titik kontrol, koefisien mewakili segitiga pascal tingkat 3, yaitu 1, 2, 1; pada 4 titik kontrol, koefisien mewakili segitiga pascal tingkat 4, yaitu 1, 3, 3, 1; dan pada 5 titik kontrol, koefisien mewakili segitiga pascal tingkat 5, yaitu 1, 4, 6, 4, 1. Perhatikan pula bahwa nilai pangkat pada $(1-t)$ dan t masing-masing menurun dan meningkat dengan skala 0 sampai $n - 1$. Oleh sebab itu, algoritma untuk n titik dapat digeneralisasi menjadi sebagai berikut

$$B(t) = C_{n,1}(1-t)^{n-1}t^0 P_0 + C_{n,2}(1-t)^{n-2}t^1 P_1 + \cdots + C_{n,n}(1-t)^0 t^{n-1} P_{n-1}$$

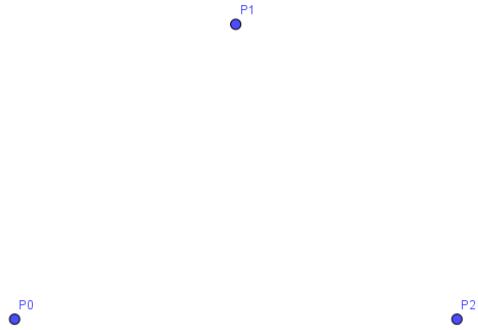
dengan $C_{n,x}$ adalah koefisien ke- x pada segitiga Paskal tingkat n dan nilai t bervariasi antara 0 sampai 1 dengan jarak antara nilai t adalah Δt .

Karena setiap iterasi memiliki banyak titik yang berbeda-beda, nilai Δt harus dikalkulasikan ulang per iterasi berdasarkan banyak titik pada iterasi tersebut.

Algoritma tersebut diimplementasikan dalam dua fungsi pada program, yaitu `createPascalTriangle` untuk mencari nilai koefisien dan `calculate` untuk menghitung titik-titik yang akan digambar menjadi kurva.

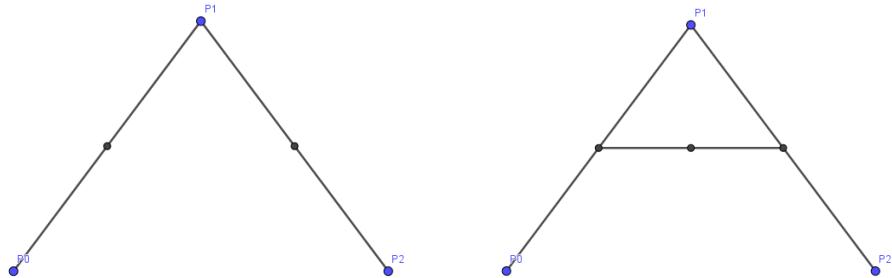
2.2 Algoritma *Divide and Conquer*

Berbeda dengan *Brute Force*, algoritma *Divide and Conquer* tidak memiliki rumus tertentu untuk mencari titik-titik yang akan dibuat menjadi kurva. Algoritma ini berfokus pada pencarian titik tengah dari titik-titik kontrol untuk dijadikan titik kontrol baru.



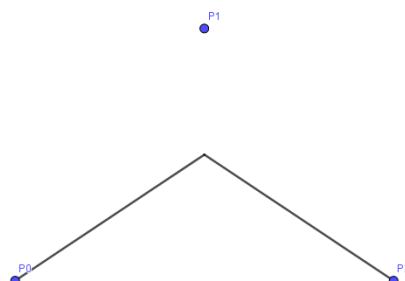
Gambar 2.2.1 Titik-Titik Kontrol

Pada kasus 3 titik kontrol, iterasi ke-0 hanya akan menghubungkan 2 titik kontrol paling ujung (P_0 dan P_2) menjadi sebuah garis lurus. Untuk iterasi ke-1, sesuai dengan rumus U_n pada bagian 2.1, perlu 3 titik untuk dijadikan kurva. Maka, perlu dicari titik ke-3 dari titik-titik tengah titik kontrol iterasi ke-0. Berikut merupakan ilustrasi-nya.



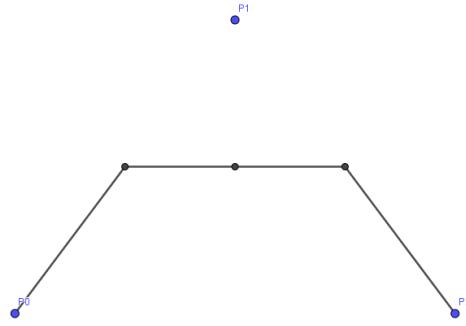
Gambar 2.2.2 Mencari Titik Baru Iterasi ke-1

Titik yang berada di tengah akan menjadi titik baru yang akan digambarkan menjadi kurva iterasi ke-1. Kurva untuk iterasi ke-1 dapat digambarkan sebagai berikut.



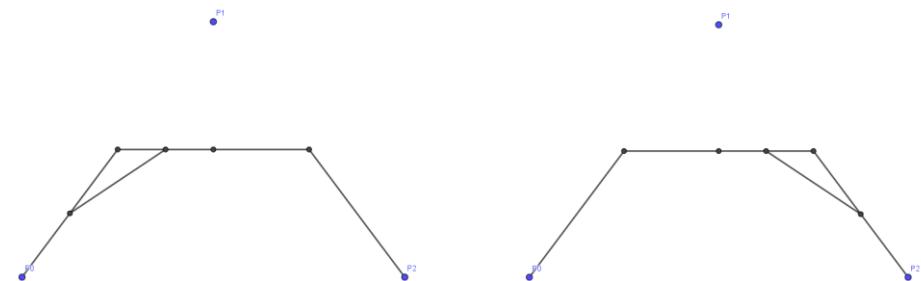
Gambar 2.2.3 Kurva Iterasi ke-1 3 Titik Kontrol

Titik-titik tengah pada iterasi ini akan menjadi titik-titik kontrol pada iterasi berikutnya. Maka, untuk iterasi ke-2, berikut merupakan ilustrasi titik kontrolnya.

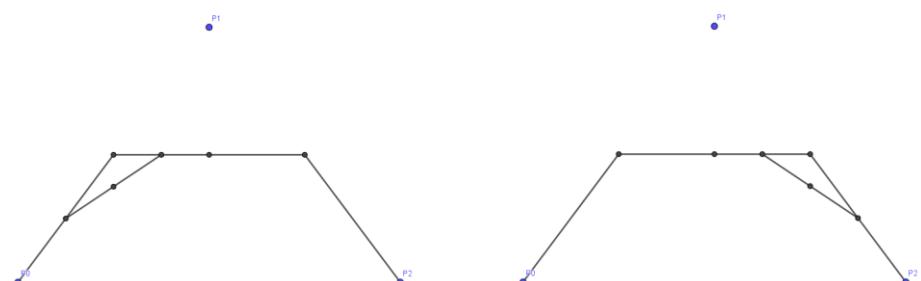


Gambar 2.2.4 Titik Kontrol Iterasi ke-2

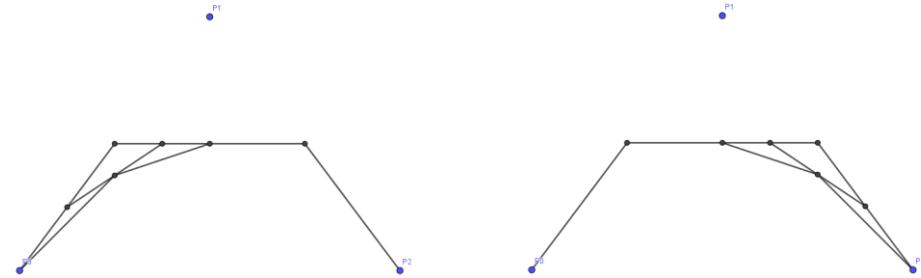
Mulai iterasi ke-2, algoritma *Divide and Conquer* akan mulai terlihat. Perhatikan bahwa 5 titik kontrol baru dapat dibagi (*divide*) menjadi 2 bagian yang masing-masing berisi 3 titik (titik ke-3 berada pada kedua bagian). Karena setiap bagian memiliki banyak titik kontrol yang sama dengan titik kontrol pada iterasi pertama, algoritma yang serupa dapat diterapkan pada kedua bagian untuk mencari solusi (*conquer*). Berikut merupakan ilustrasinya.



Gambar 2.2.5 Penerapan *Divide and Conquer* pada Iterasi ke-2 (Bagian 1)

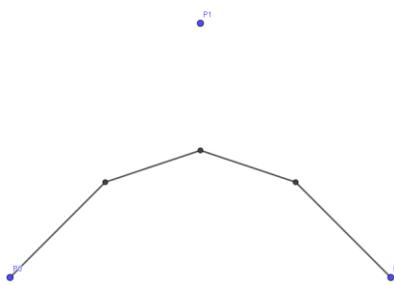


Gambar 2.2.6 Penerapan *Divide and Conquer* pada Iterasi ke-2 (Bagian 2)



Gambar 2.2.7 Penerapan *Divide and Conquer* pada Iterasi ke-2 (Bagian 3)

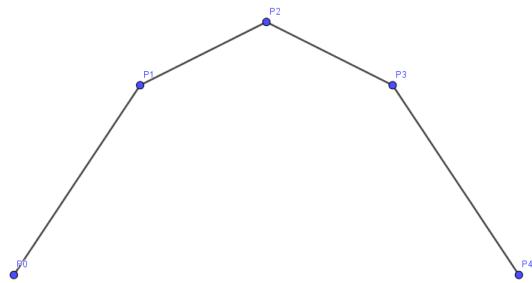
Karena algoritma sudah selesai diterapkan, solusi dapat digabung kembali membentuk sebuah kurva baru sebagai berikut.



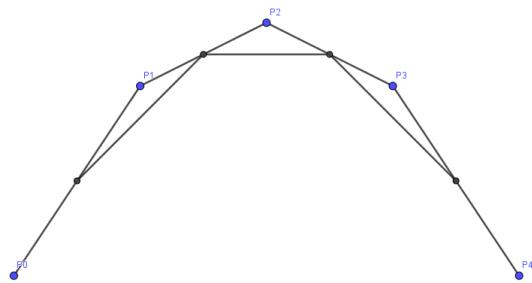
Gambar 2.2.8 Kurva Iterasi ke-2 Titik Kontrol

Titik-titik tengah pada iterasi ke-2 akan kembali disimpan untuk melakukan algoritma *Divide and Conquer* pada iterasi ke-3. Banyak titik kontrol hasil iterasi ke-2 adalah 9 buah. Semua titik tersebut akan dibagi-bagi kembali menjadi bagian yang masing-masing berisi 3 titik ([1, 2, 3], [3, 4, 5], dst). Solusi dari setiap bagian kemudian akan digabung menjadi kurva iterasi ke-3 dan seterusnya sampai berapapun iterasi yang diinginkan.

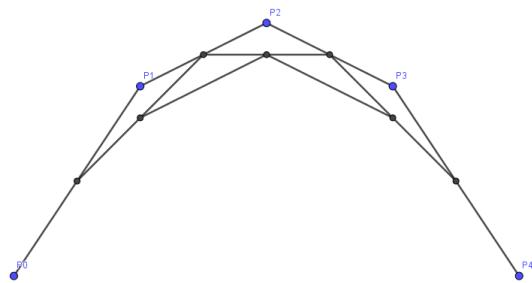
Agar dapat menerapkan algoritma ini untuk n titik kontrol, perlu dilakukan pengamatan pola untuk titik-titik di atas 3. Contohnya, berikut merupakan pencarian titik baru untuk 5 titik kontrol.



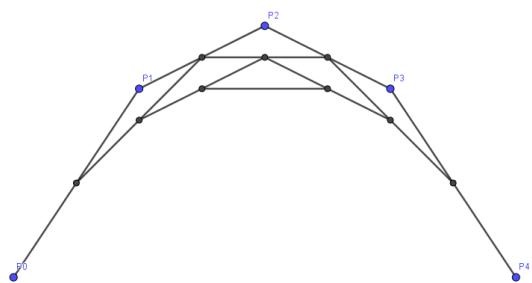
Gambar 2.2.9 Mencari Pola n Titik Kontrol (Bagian 1)



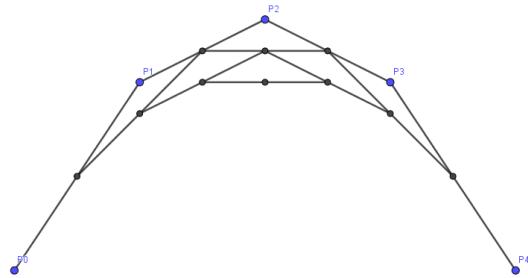
Gambar 2.2.10 Mencari Pola n Titik Kontrol (Bagian 2)



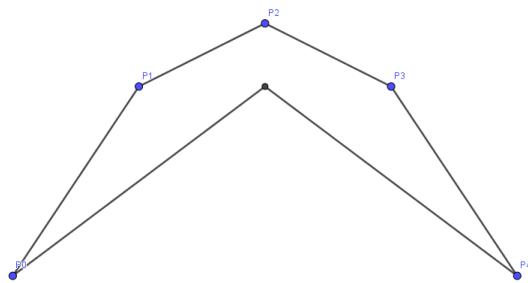
Gambar 2.2.11 Mencari Pola n Titik Kontrol (Bagian 3)



Gambar 2.2.12 Mencari Pola n Titik Kontrol (Bagian 4)



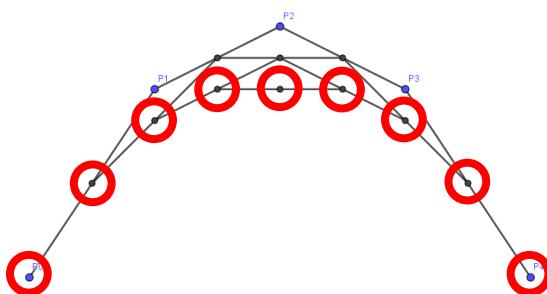
Gambar 2.2.13 Mencari Pola n Titik Kontrol (Bagian 5)



Gambar 2.2.14 Kurva Iterasi ke-1 5 Titik Kontrol

Perhatikan bahwa terjadi 4 kali pengulangan pencarian titik tengah. Pengulangan dilakukan sampai hanya ada 1 titik tengah yang didapat dari kalkulasi titik tengah sebelumnya. Maka, banyak pengulangan pencarian titik tengah dapat digeneralisasi menjadi sebanyak $n - 1$ kali.

Selain itu, perhatikan bahwa titik tengah pertama dan terakhir pada setiap pengulangan akan menjadi titik kontrol baru pada iterasi berikutnya. Berikut merupakan titik-titik yang dimaksud.



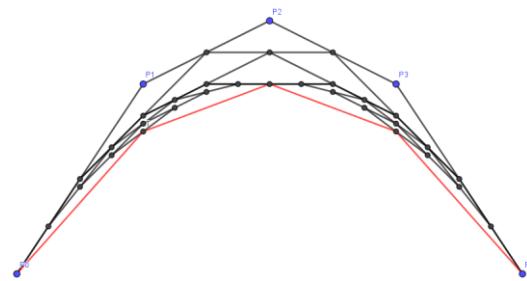
Gambar 2.2.15 Titik Kontrol untuk Iterasi ke-2

Dengan menggunakan metode yang mirip dengan metode pada 3 titik kontrol, titik-titik ini dapat dibagi menjadi bagian-bagian yang masing-masingnya terdiri dari 5 titik. Maka, dapat digeneralisasi, untuk setiap pengolahan n titik kontrol, akan terbentuk $2n -$

1 titik kontrol baru yang dapat dibagi menjadi bagian-bagian yang masing-masing berisi n titik. Selain itu, banyaknya bagian untuk setiap iterasi dapat digeneralisasi dengan melihat pola per iterasi: 1, 2, 4, 8, 16, dst. Maka, banyaknya bagian untuk iterasi ke- i dapat dihitung dengan rumus berikut.

$$\text{section}_i = 2^i$$

Setelah itu, *Divide and Conquer* dapat kembali diterapkan dengan mencari solusi masing-masing bagian lalu menggabungkan semua hasilnya. Berikut merupakan hasil iterasi ke-2 untuk 5 titik kontrol dengan algoritma yang sudah digeneralisasi.



Gambar 2.2.16 Kurva Iterasi ke-2 5 Titik Kontrol

Algoritma tersebut diimplementasikan pada dalam satu fungsi pada program, yaitu `createNewControl` untuk mencari nilai titik-titik kontrol baru dengan memecah titik-titik kontrol lama menjadi bagian-bagian yang berisi n titik kontrol.

2.3 Implementasi Algoritma dalam Bahasa Python

2.3.1 File “bezier.py” tanpa komentar

```
import time as t
import functions as f
import divandconq as dnc
import bruteforce as bf
import io
from PIL import Image

print("\n\nSELAMAT DATANG DI PROGRAM PEMBUAT KURVA BEZIER DENGAN ALGORITMA DIVIDE AND CONQUER / BRUTE FORCE\n\n")

arrIterations = []
arrIterationsBF = []
arrPoints = []
arrControlIterations = []
arrControl = []

valid = False
while not (valid):
    try:
        print("Masukkan banyaknya titik")
        countPoints = int(input(">> "))
        valid = True
    except ValueError:
        print("Tolong masukkan bilangan bulat (integer) !")
```

```

print("\nMasukkan titik-titik dengan format:\nx y")
for i in range (countPoints):
    valid = False
    while not (valid):
        try:
            x, y = input().split()
            x = float(x)
            y = float(y)
            if (i == 0):
                min = x
                max = x
            else:
                if (x < min):
                    min = x
                if (y < min):
                    min = y
                if (x > max):
                    max = x
                if (y > max):
                    max = y
            if (i == 0) or (i == countPoints - 1):
                arrPoints.append((x, y))
                arrControl.append((x, y))
            valid = True
        except ValueError:
            print("Tolong masukkan titik sesuai format")
extremePoint = [min, max]

arrIterations.append(arrPoints)
arrIterationsBF.append(arrPoints)
arrControlIterations.append(arrControl)

valid = False
while not (valid):
    try:
        print("\nMasukkan banyaknya iterasi")
        iterations = int(input(">> "))
        valid = True
    except ValueError:
        print("Tolong masukkan bilangan bulat (integer)!")
arrMidPoints = [[] for i in range (iterations + 1)]

startTime = t.time()

for i in range (iterations):
    arrPoints = []

    temp = dnc.createNewControl(arrControl, countPoints)
    arrControl = temp[0].copy()
    arrMidPoints[i + 1] = temp[1].copy()

    for j in range (0, len(arrControl), countPoints - 1):
        arrPoints.append(arrControl[j])

    arrIterations.append(arrPoints)
    arrControlIterations.append(arrControl)

endTime = t.time()
dncTime = endTime - startTime

startTime = t.time()

for i in range (iterations):
    arrPoints = []
    amountPoints = 2** (i + 1) + 1
    tVal = 1 / (amountPoints - 1)
    coeff = bf.createPascalTriangle(countPoints - 1)
    arrPoints = bf.calculate(arrControlIterations[0], coeff, tVal)
    arrIterationsBF.append(arrPoints)

endTime = t.time()
bfTime = endTime - startTime

print(f"\nWaktu eksekusi algoritma Divide and Conquer: {dncTime} detik")
print(f"Waktu eksekusi algoritma Brute Force: {bfTime} detik\n")

```

```

f.plot.ion()

i = 0
for points in arrIterations:
    f.plot.clf()
    f.plot_points_with_coordinates(arrControlIterations[0], extremePoint[1],
extremePoint[0])
    if (i != 0):
        f.plot_points_only(arrControlIterations[i - 1], extremePoint[1],
extremePoint[0])
        f.plot_control_points(arrControlIterations[i - 1], extremePoint[1],
extremePoint[0])
    else:
        f.plot_points_only(arrControlIterations[i], extremePoint[1],
extremePoint[0])

    for j in range (1, len(arrMidPoints[i])):
        f.plot_midpoints(arrMidPoints[i][j], countPoints - j, extremePoint[1],
extremePoint[0])

    f.plot_points_only(points, extremePoint[1], extremePoint[0])
    f.plot_bezier_curve(points, i, extremePoint[1], extremePoint[0])
    f.plot.clf()
    f.plot_points_with_coordinates(arrControlIterations[0], extremePoint[1],
extremePoint[0])
    f.plot_bezier_curve(points, i, extremePoint[1], extremePoint[0])
    i = i + 1

f.plot.ioff()

berhenti = False
while not (berhenti):
    try:
        print("Masukkan nomor iterasi yang ingin ditampilkan, tutup kurva sebelumnya
untuk memeriksa iterasi lain (-1 untuk berhenti)")
        number = int(input(">> "))
        if (number == -1):
            berhenti = True
        else:
            pointsDNC = arrIterations[number]
            pointsBF = arrIterationsBF[number]
            f.draw_two_curve(pointsDNC, pointsBF, extremePoint[1],
extremePoint[0])

    except IndexError:
        print(f"Masukkan nomor iterasi pada range 0 - {iterations}")
    except ValueError:
        print("Input tidak valid.")

valid = False
while not (valid):
    print("\nApakah Anda ingin menyimpan hasil dalam format .gif? (Ya/Tidak)")
    strInput = str(input(">> "))
    if ((strInput == "Ya") or (strInput == "Tidak")):
        valid = True
    else:
        print("Masukan tidak valid!")

if (strInput == "Ya"):
    frames = []

    for i, points in enumerate(arrIterations):
        f.plot.plot(*zip(*points))
        f.plot.xlabel('Sumbu x')
        f.plot.ylabel('Sumbu y')
        f.plot.title(f'Iterasi ke-{i}')

        buf = io.BytesIO()
        f.plot.savefig(buf, format='png')
        buf.seek(0)

        img = Image.open(buf)
        frames.append(img)

    f.plot.clf()

```

```

print("Masukkan (HANYA) nama file yang diinginkan: ")
namaFile = str(input(">> "))

frames[0].save(f'../test/{namaFile}.gif', format = 'GIF', append_images =
frames[1:], save_all = True, duration = 1000, loop = 0)

t.sleep(0.5)
print("\n\nTERIMA KASIH TELAH MENGGUNAKAN PROGRAM PEMBUAT KURVA BEZIER :)\n\n")
t.sleep(0.5)

print("Menutup program", end = "")
for i in range (3):
    t.sleep(1)
    print(".", end = "")
t.sleep(1)

```

2.3.2 File “divandconq.py” tanpa komentar

```

def createNewControl(prevControl, cntControl):
    newControl = []
    arrMidPoints = [[] for i in range (cntControl)]
    arrMidPoints[0] = prevControl.copy()
    i = 0

    while (i < (len(prevControl) - cntControl + 1)):
        idx = 0
        temp = []
        saveControl = []

        for j in range (i, i + cntControl):
            temp.append(prevControl[j])

        saveControl.append(prevControl[i])
        saveControl.append(prevControl[i + (cntControl - 1)])
        idx = idx + 1

        while (len(temp) != 1):
            temp2 = []

            for k in range (len(temp) - 1):
                x = float((temp[k][0] + temp[k + 1][0]) / 2)
                y = float((temp[k][1] + temp[k + 1][1]) / 2)
                temp2.append((x, y))

            if ((x, y)) not in (arrMidPoints[idx]):
                arrMidPoints[idx].append((x, y))

            temp = temp2.copy()

            saveControl.insert(idx, temp[0])
            saveControl.insert(len(saveControl) - idx, temp[-1])

            idx = idx + 1

        for j in range (len(saveControl)):
            if (saveControl[j]) not in (newControl):
                newControl.append(saveControl[j])

        i = i + (cntControl - 1)

    return [newControl, arrMidPoints]

```

2.3.3 File “bruteforce.py” tanpa komentar

```

def createPascalTriangle(level):
    if (level == 0):
        return [1]
    elif (level == 1):
        return [1, 1]
    else:

```

```
prev = createPascalTriangle(level - 1)
curr = [1]
for i in range (len(prev) - 1):
    curr.append(prev[i] + prev[i + 1])
curr.append(1)
return curr

def calculate(arrControl, coefficients, tVal):
    i = 0.0
    savePoints = []
    while (i <= 1.0):
        tempx = 0
        tempy = 0
        for j in range (len(arrControl)):
            x = arrControl[j][0]
            y = arrControl[j][1]
            tempx = tempx + (coefficients[j] * ((1 - i)**(len(arrControl) -
1 - j)) * ((i)**(j)) * x)
            tempy = tempy + (coefficients[j] * ((1 - i)**(len(arrControl) -
1 - j)) * ((i)**(j)) * y)
        savePoints.append((tempx, tempy))

        i = i + tVal

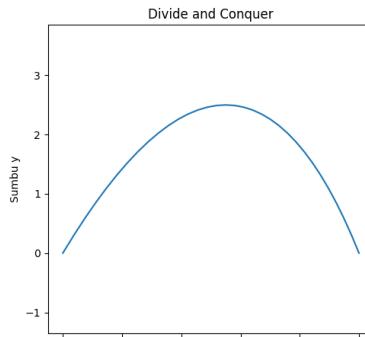
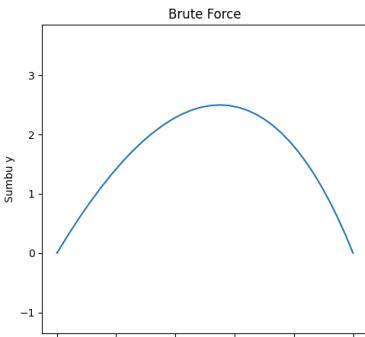
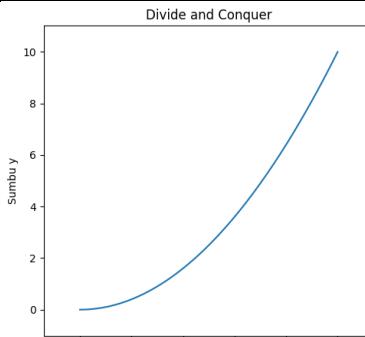
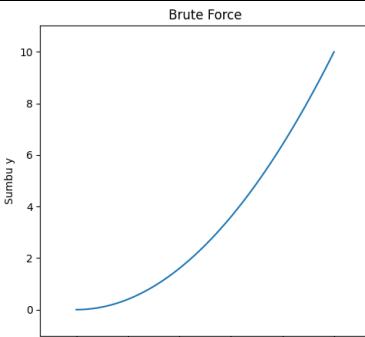
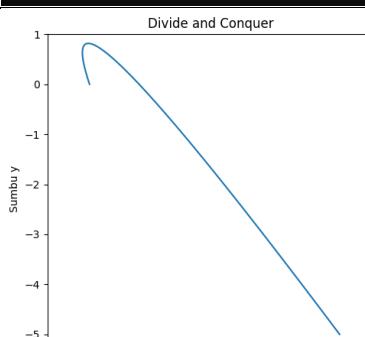
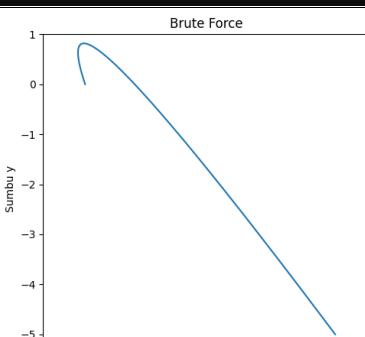
    return savePoints
```

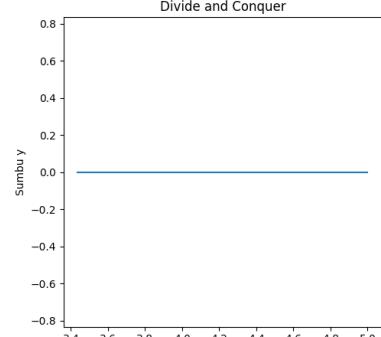
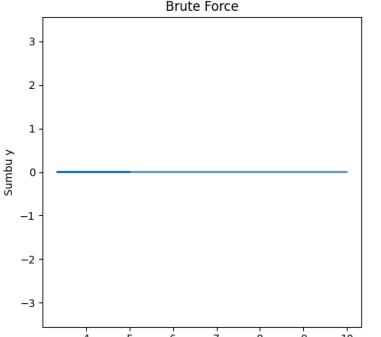
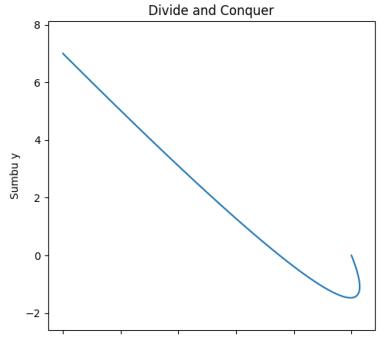
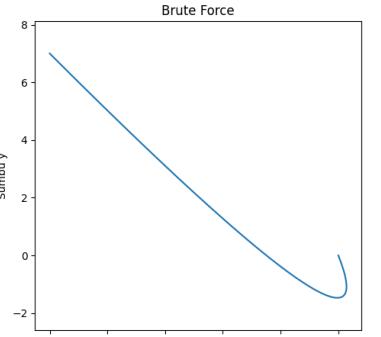
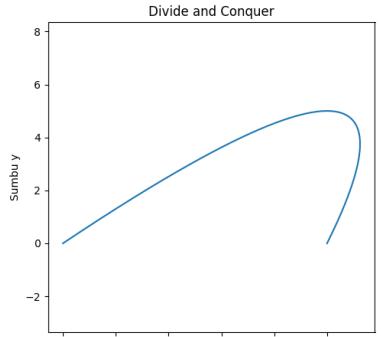
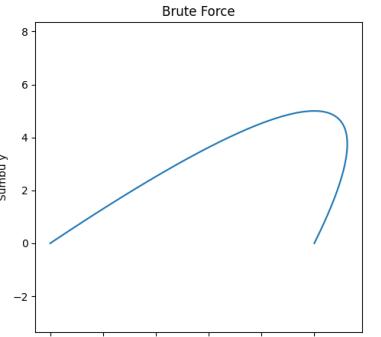
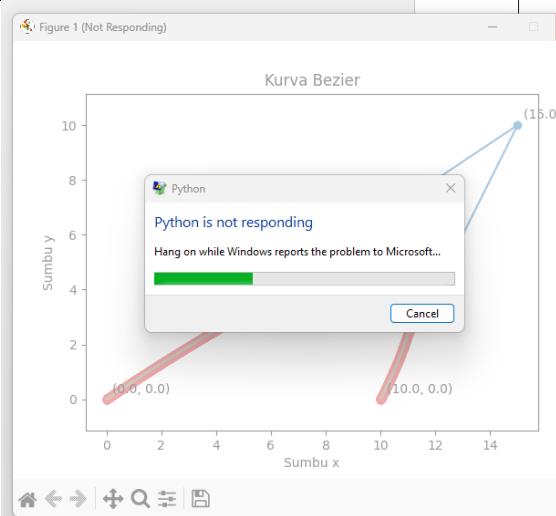
BAB III

Uji Coba Program

3.1 Uji Coba 3 Titik Kontrol

Tabel 3.1.1 Uji Coba 3 Titik Kontrol

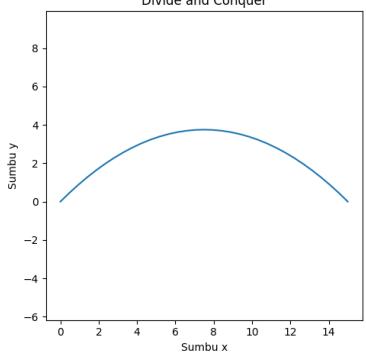
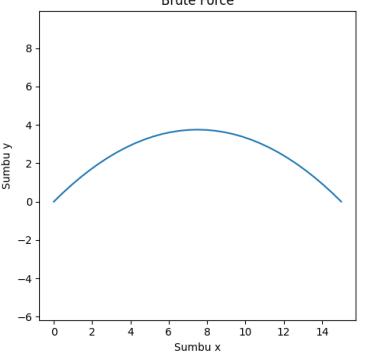
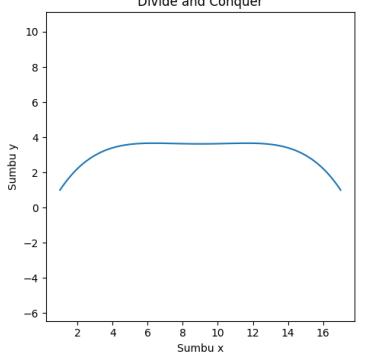
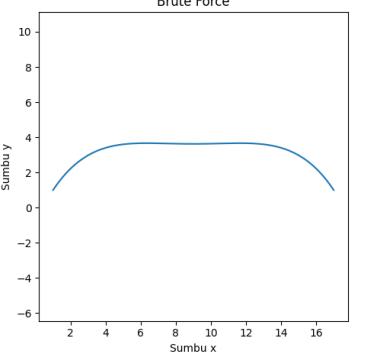
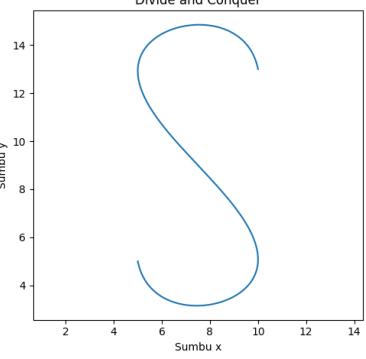
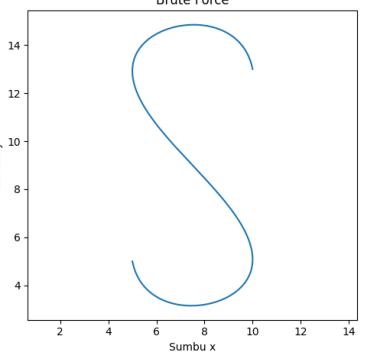
No.	Titik Kontrol	Iterasi	Perbandingan Hasil <i>Divide and Conquer</i> dengan <i>Brute Force</i>
1	(0, 0), (3, 5), (5, 0)	5	<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;">  <p>Divide and Conquer</p> </div> <div style="text-align: center;">  <p>Brute Force</p> </div> </div>
Waktu eksekusi		Waktu eksekusi algoritma Divide and Conquer: 0.0 detik Waktu eksekusi algoritma Brute Force: 0.0 detik	
2	(0, 0), (5, 0), (10, 10)	6	<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;">  <p>Divide and Conquer</p> </div> <div style="text-align: center;">  <p>Brute Force</p> </div> </div>
Waktu eksekusi		Waktu eksekusi algoritma Divide and Conquer: 0.0 detik Waktu eksekusi algoritma Brute Force: 0.0 detik	
3	(0, 0), (-1, 3), (5, -5)	8	<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;">  <p>Divide and Conquer</p> </div> <div style="text-align: center;">  <p>Brute Force</p> </div> </div>
Waktu eksekusi		Waktu eksekusi algoritma Divide and Conquer: 0.0 detik Waktu eksekusi algoritma Brute Force: 0.0 detik	

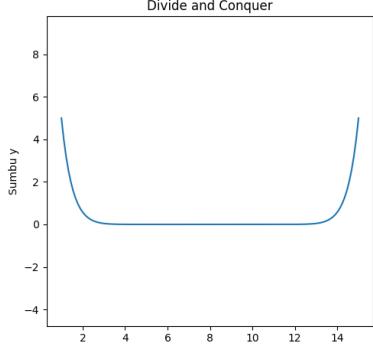
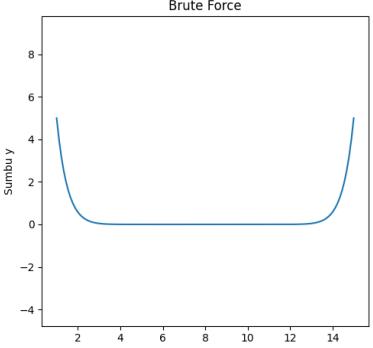
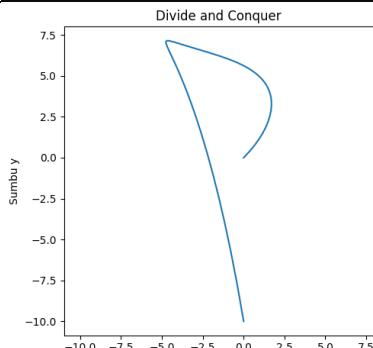
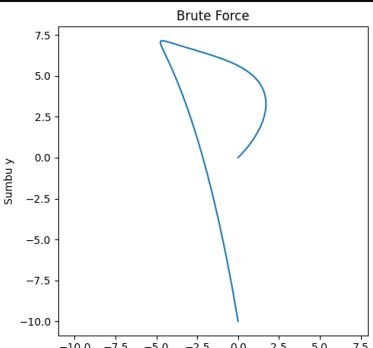
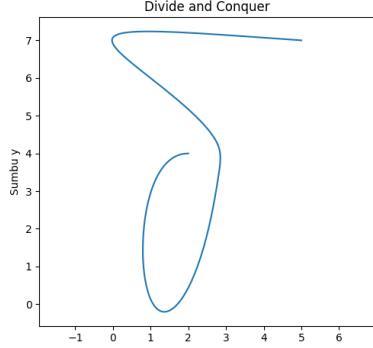
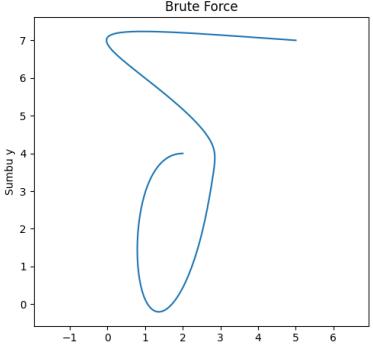
4	$(5, 0), (0, 0), (10, 0)$	10								
			Waktu eksekusi algoritma Divide and Conquer: 0.015530824661254883 detik Waktu eksekusi algoritma Brute Force: 0.0 detik							
5	$(10, 0), (12, -5), (0, 7)$	10								
			Waktu eksekusi algoritma Divide and Conquer: 0.04688000679016113 detik Waktu eksekusi algoritma Brute Force: 0.0 detik							
6	$(0, 0), (15, 10), (10, 0)$	10								
			Waktu eksekusi algoritma Divide and Conquer: 0.04300117492675781 detik Waktu eksekusi algoritma Brute Force: 0.0030019283294677734 detik							
7	$(0, 0), (15, 10), (10, 0)$	15		<table border="1"> <tr> <td>Waktu eksekusi</td> </tr> <tr> <td>$(0, 0), (12, -5), (0, 7)$</td> </tr> <tr> <td>Waktu eksekusi</td> </tr> <tr> <td>$(0, 0), (15, 10), (10, 0)$</td> </tr> <tr> <td>Waktu eksekusi</td> </tr> <tr> <td>Waktu eksekusi</td> </tr> </table>	Waktu eksekusi	$(0, 0), (12, -5), (0, 7)$	Waktu eksekusi	$(0, 0), (15, 10), (10, 0)$	Waktu eksekusi	Waktu eksekusi
Waktu eksekusi										
$(0, 0), (12, -5), (0, 7)$										
Waktu eksekusi										
$(0, 0), (15, 10), (10, 0)$										
Waktu eksekusi										
Waktu eksekusi										

	Waktu eksekusi	Waktu eksekusi algoritma Divide and Conquer: 48.60536193847656 detik Waktu eksekusi algoritma Brute Force: 0.07800936698913574 detik
--	----------------	---

3.2 Uji Coba n Titik Kontrol

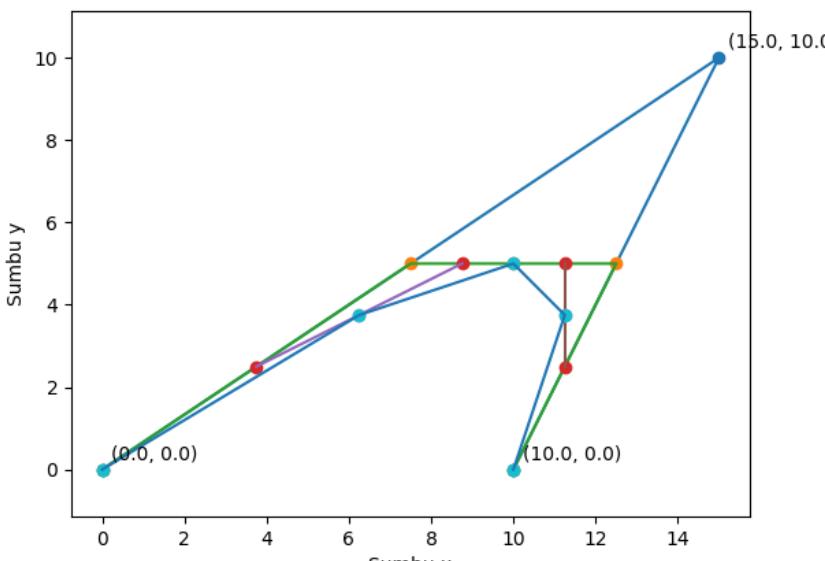
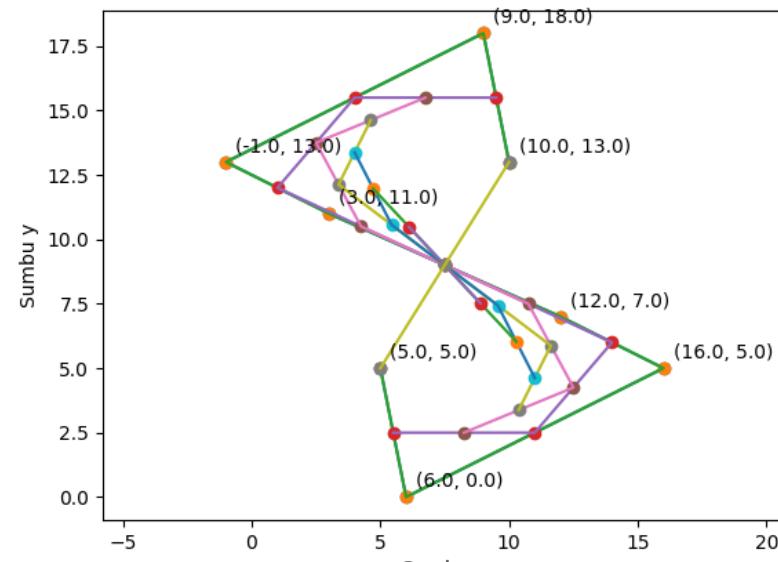
Tabel 3.2.1 Uji Coba n Titik Kontrol

No.	Titik Kontrol	Iterasi	Perbandingan Hasil Divide and Conquer dengan Brute Force
1	(0, 0), (5, 5), (10, 5), (15, 0)	5	 
	Waktu eksekusi		Waktu eksekusi algoritma Divide and Conquer: 0.0010008811950683594 detik Waktu eksekusi algoritma Brute Force: 0.0 detik
2	(1, 1), (5, 7), (9, 0), (13, 7), (17, 1)	7	 
	Waktu eksekusi		Waktu eksekusi algoritma Divide and Conquer: 0.003999471664428711 detik Waktu eksekusi algoritma Brute Force: 0.0 detik
3	(5, 5), (6, 0), (16, 5), (12, 7), (3, 11), (-1, 13), (9, 18), (10, 13)	10	 
	Waktu eksekusi		Waktu eksekusi algoritma Divide and Conquer: 0.547844648361206 detik Waktu eksekusi algoritma Brute Force: 0.00697636604309082 detik

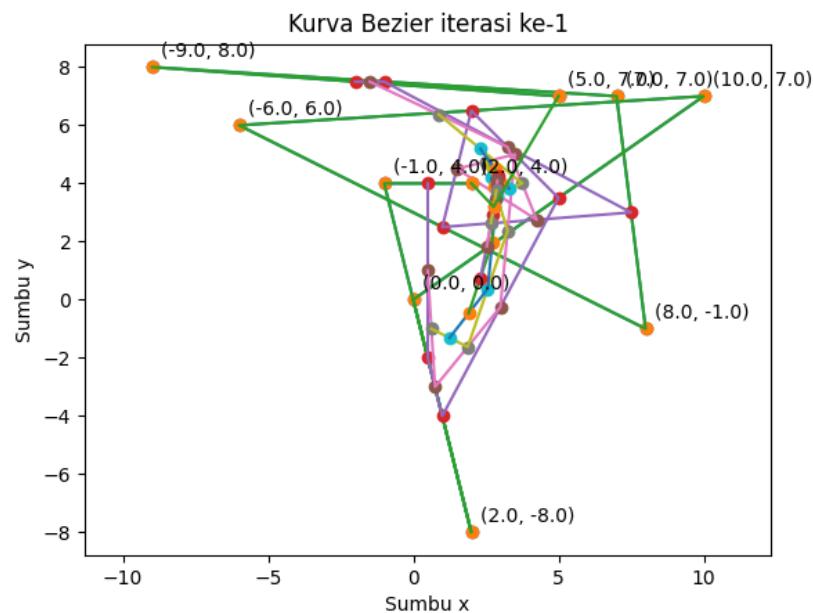
4	(1, 5), (2, -5), (3, 5), (4, -5), (5, 5), (6, -5), (7, 5), (8, -5), (9, 5), (10, -5), (11, 5), (12, -5), (13, 5), (14, -5), (15, 5)	7	 
	Waktu eksekusi		Waktu eksekusi algoritma Divide and Conquer: 0.046105384826660156 detik Waktu eksekusi algoritma Brute Force: 0.0018935203552246094 detik
5	(0, 0), (5, 5), (0, 10), (-10, 0), (-5, -20), (0, -10)	6	 
	Waktu eksekusi		Waktu eksekusi algoritma Divide and Conquer: 0.002000093460083008 detik Waktu eksekusi algoritma Brute Force: 0.0 detik
6	(2, 4), (-1, 4), (2, -8), (0, 0), (10, 7), (-6, 6), (8, -1), (7, 7), (-9, 8), (5, 7)	8	 
	Waktu eksekusi		Waktu eksekusi algoritma Divide and Conquer: 0.062000274658203125 detik Waktu eksekusi algoritma Brute Force: 0.0019998550415039062 detik

3.3 Uji Coba Bonus Visualisasi

Tabel 3.3.1 Uji Coba Bonus Visualisasi

Referensi	Contoh Visualisasi																												
3.3.1 Uji Coba 6	<p style="text-align: center;">Kurva Bezier iterasi ke-2</p>  <table border="1"> <caption>Data for Kurva Bezier iterasi ke-2</caption> <thead> <tr> <th>Sumbu x</th> <th>Sumbu y</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td></tr> <tr><td>10</td><td>0</td></tr> <tr><td>12</td><td>5</td></tr> <tr><td>15</td><td>10</td></tr> <tr><td>6</td><td>4</td></tr> <tr><td>8</td><td>5</td></tr> <tr><td>11</td><td>3.8</td></tr> <tr><td>4</td><td>2.5</td></tr> </tbody> </table>	Sumbu x	Sumbu y	0	0	10	0	12	5	15	10	6	4	8	5	11	3.8	4	2.5										
Sumbu x	Sumbu y																												
0	0																												
10	0																												
12	5																												
15	10																												
6	4																												
8	5																												
11	3.8																												
4	2.5																												
3.3.2 Uji Coba 3	<p style="text-align: center;">Kurva Bezier iterasi ke-1</p>  <table border="1"> <caption>Data for Kurva Bezier iterasi ke-1</caption> <thead> <tr> <th>Sumbu x</th> <th>Sumbu y</th> </tr> </thead> <tbody> <tr><td>-1</td><td>13</td></tr> <tr><td>0</td><td>12.5</td></tr> <tr><td>5</td><td>15</td></tr> <tr><td>10</td><td>13</td></tr> <tr><td>16</td><td>5</td></tr> <tr><td>0</td><td>12.5</td></tr> <tr><td>5</td><td>11.0</td></tr> <tr><td>10</td><td>7.0</td></tr> <tr><td>15</td><td>6.0</td></tr> <tr><td>5</td><td>5.0</td></tr> <tr><td>10</td><td>3.0</td></tr> <tr><td>15</td><td>2.0</td></tr> <tr><td>6</td><td>0.0</td></tr> </tbody> </table>	Sumbu x	Sumbu y	-1	13	0	12.5	5	15	10	13	16	5	0	12.5	5	11.0	10	7.0	15	6.0	5	5.0	10	3.0	15	2.0	6	0.0
Sumbu x	Sumbu y																												
-1	13																												
0	12.5																												
5	15																												
10	13																												
16	5																												
0	12.5																												
5	11.0																												
10	7.0																												
15	6.0																												
5	5.0																												
10	3.0																												
15	2.0																												
6	0.0																												

3.3.2 Uji Coba 6



3.4 Analisis Hasil Uji Coba

Secara kompleksitas waktu, algoritma *Brute Force* memiliki kompleksitas $T(t \times c)$ di mana t adalah banyaknya variasi nilai t pada rentang 0 sampai 1 dan c adalah banyaknya titik kontrol. Berdasarkan analisis pada bagian 2.1, banyaknya variasi nilai t akan sama dengan banyaknya titik kontrol. Namun, perhatikan bahwa untuk iterasi yang besar, nilai t akan semakin besar, tetapi nilai c tetap. Kompleksitas algoritma *Brute Force* akan meningkat secara eksponensial (mengikuti pola $2^i + 1$) sehingga dalam notasi Big-O, kompleksitasnya menjadi $O(2^n)$. Sementara itu, algoritma *Divide and Conquer* memiliki kompleksitas $T(\text{section} \times c! \times (c - 1)!)$ di mana *section* adalah banyaknya bagian titik kontrol dan c adalah banyaknya titik kontrol. Berdasarkan analisis pada bagian 2.2, banyaknya *section* akan meningkat setiap iterasi dengan pola 2^i dengan i adalah iterasi yang ke berapa. Karena nilai c selalu konstan, untuk iterasi yang besar, $c! \times (c - 1)!$ akan sangat kecil jika dibandingkan 2^i sehingga kompleksitasnya dapat dinyatakan sebagai $O(2^n)$.

Berdasarkan analisis sebelumnya, kompleksitas kedua algoritma adalah sama, yaitu $O(2^n)$. Namun, ada perbedaan pada konstanta pengali, yaitu c pada algoritma *Brute Force* dan $c! \times (c - 1)!$ pada algoritma *Divide and Conquer*. Cukup jelas bahwa $c! \times (c - 1)! > c$ sehingga waktu eksekusi pada bagian 3.1 dan 3.2 sudah sesuai dengan analisis kompleksitas.

Secara intuisi juga sebenarnya dapat diketahui bahwa algoritma *Brute Force* membutuhkan waktu yang lebih singkat untuk dieksekusi dibanding algoritma *Divide and Conquer* pada konteks pembuatan kurva Bézier. Perhatikan bahwa pada algoritma *Brute Force*, kalkulasi dilakukan dengan rumus yang tetap, sementara untuk *Divide and Conquer*, perlu dilakukan pencarian titik tengah untuk banyak bagian hasil pembagian titik-titik kontrol untuk iterasi di atas 1.

Faktor lain yang membuat algoritma *Divide and Conquer* pada program ini lebih kompleks dibanding algoritma *Brute Force* adalah kompleksitas ruang yang besar. Karena adanya pengajaran bonus, setiap titik tengah untuk setiap iterasi harus disimpan untuk menggambarkan proses pembuatan kurva Bézier secara *step-by-step* sehingga membutuhkan ruang penyimpanan yang lebih besar.

Bab IV

Kesimpulan

4.1 Kesimpulan

Baik algoritma *Brute Force* maupun algoritma *Divide and Conquer* dapat digunakan untuk menggambar kurva Bézier. Algoritma *Brute Force* menggunakan rumus hasil penurunan persamaan garis untuk mencari titik-titik pembentuk kurva. Algoritma *Divide and Conquer* menggunakan prinsip titik tengah untuk mencari titik-titik kontrol dan titik-titik pembentuk kurva. Secara kompleksitas waktu, algoritma *Divide and Conquer* lebih kompleks dibanding algoritma *Brute Force*.

4.2 Saran

1. Dalam pembuatan kurva Bézier, lebih baik menggunakan algoritma *Brute Force* dibanding algoritma *Divide and Conquer* karena kompleksitas algoritma *Brute Force* yang lebih kecil.
2. Program hasil implementasi kedua algoritma dapat digunakan di masa yang akan datang untuk membuat kurva Bézier sendiri.

Referensi

Kantor, Ilya. (2022). *Bezier curve*. Diakses pada 18 Maret 2024, dari <https://javascript.info/bezier-curve>

Melo, Mateius. (2021). *Understanding Bézier Curves*. Diakses pada 17 Maret 2024, dari <https://mmrndev.medium.com/understanding-b%C3%A9zier-curves-f6eaa0fa6c7d>

Munir, R. (2024). *Algoritma Divide and Conquer (Bagian 1)*. Diakses pada 17 Maret 2024, dari [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)

Munir, R. (2024). *Algoritma Divide and Conquer (Bagian 2)*. Diakses pada 17 Maret 2024, dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf)

Munir, R. (2024). *Algoritma Divide and Conquer (Bagian 3)*. Diakses pada 17 Maret 2024, dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf)

Munir, R. (2024). *Algoritma Divide and Conquer (Bagian 4)*. Diakses pada 17 Maret 2024, dari [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)

Lampiran

Link menuju *repository* Github:

https://github.com/nicholasrs05/Tucil2_13522144

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	✓	