

TUGAS KECIL 2 IF2211 STRATEGI ALGORITMA

MEMBANGUN KURVA BÉZIER DENGAN ALGORITMA TITIK

TENGAH BERBASIS *DIVIDE AND CONQUER*



Disusun oleh:

Salsabiila 13522062

Shazya Audrea Taufik 13522063

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

DAFTAR ISI

DAFTAR ISI.....	1
BAB I ANALISIS DAN IMPLEMENTASI.....	2
1.1 Algoritma Brute Force.....	2
1.2 Algoritma Divide and Conquer.....	2
BAB II ALUR PROGRAM.....	4
BAB III SOURCE CODE.....	7
3.1 Input.py.....	7
3.2 N_DnC.py.....	7
3.3 N_BF.py.....	8
3.4 Visualisation.py.....	9
3.5 main.py.....	11
BAB IV HASIL PENGUJIAN.....	13
4.1 Test Case 1.....	13
4.2 Test Case 2.....	14
4.3 Test Case 3.....	14
4.4 Test Case 4.....	15
4.5 Test Case 5.....	16
4.6 Test Case 6.....	16
4.7 Test Case 7 (Bonus_1).....	17
4.8 Test Case 8 (Bonus_2).....	18
4.9 Test Case 9 (Bonus_3).....	20
4.10 Test Case 10 (Bonus_4).....	21
4.11 Test Case 11 (Bonus_5).....	21
4.12 Test Case 12 (Bonus_6).....	23
BAB V ANALISIS PERBANDINGAN SOLUSI.....	24
BAB VI BONUS.....	30
6.1 N titik.....	30
6.2 Animasi Iterasi.....	30
BAB VII LAMPIRAN.....	32
7.1 Link Repository.....	32
7.2 Tabel Kelengkapan.....	32

BAB I

ANALISIS DAN IMPLEMENTASI

1.1 Algoritma Brute Force

Algoritma *brute force* merupakan algoritma yang menyelesaikan permasalahan dengan tahapan yang *straightforward*. Pada program yang menghasilkan kurva Bezier dengan menggunakan algoritma *brute force* ini, penentuan titik pada kurva akan dihitung dengan menggunakan rumus linear kurva Bezier secara berulang hingga terbentuk kurva akhir. Berikut merupakan persamaan linear kurva Bezier:

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

Pada rumus tersebut, t merupakan perbandingan antara jarak titik $B(t)$ dengan P_0 dibandingkan jarak P_0 dengan P_1 . Karena pada spesifikasi dari tugas ini tujuan dari pendekatan *brute force* adalah sebagai pembanding, maka garis linear yang dibentuk dari setiap *control point* akan dibagi sebanyak jumlah titik akhir pada iterasi ke- i *divide and conquer* dikurangi 1. Perhitungan jumlah titik pada iterasi ke- i dapat dilakukan dengan menggunakan rumus berikut:

$$\text{number of points} = 2^{\text{iteration}} + 1$$

Dalam implementasi algoritma *brute force*, berikut merupakan langkah-langkah yang digunakan antara lain:

1. Untuk setiap bagian t , akan dikalkulasikan titik interpolasi sejauh t dari *control point* ke- i ke *control point* ke- $(i+1)$.
2. Titik-titik yang dihasilkan dari hasil interpolasi pada langkah 1 akan menjadi *control point* dan dikalkulasikan kembali titik interpolasi sejauh t secara rekursif hingga mengembalikan satu titik yang menjadi titik pada kurva Bezier.
3. Setelah dikembalikan satu nilai titik yang menjadi titik akhir kurva, iterasi t akan berlanjut dan prosesnya akan berulang hingga t bernilai sama dengan satu.

Metode ini menggunakan algoritma *brute force* yang secara langsung menghitung titik akhir pada kurva Bezier dan implementasinya terdapat dalam file **N_BF.py** yang dapat dilihat pada bab III laporan ini.

1.2 Algoritma Divide and Conquer

Midpoint algorithm adalah suatu algoritma yang banyak digunakan dalam pembuatan grafik komputer. Algoritma ini secara efisien menentukan titik-titik yang membentuk garis/kurva.

Algoritma ini dianggap sebagai alternatif yang baik daripada menentukan posisi titik-titik secara presisi. Cara kerja dari algoritma ini sendiri adalah dengan:

1. Mengambil titik-titik yang termasuk titik awal, titik akhir, dan titik kontrol. Untuk pembuatan kurva dengan derajat 2, maka akan memerlukan 3 titik.
2. Pada setiap iterasi cari titik tengah dari antara setiap titik yang berurutan. Tarik garis di antara titik-titik tengah tersebut.

Semakin banyak iterasi, maka akan semakin mulus kurva yang dibentuk. Dalam membentuk kurva Bezier ini, dapat juga diimplementasikan algoritma *divide and conquer* sehingga memudahkan komputasi. Langkah-langkahnya adalah sebagai berikut:

1. Membuat basis rekursif, yaitu ketika iterasi = 0 atau iterasi = 1. Jika iterasi = 0, maka kode akan mengembalikan titik kontrol asli. Jika iterasi = 1, maka kode akan menghitung titik tengah antara titik kontrol yang berdekatan secara iteratif dan mengembalikan titik awal, titik-titik tengah, dan titik akhir.
2. Jika jumlah iterasi lebih dari 1, fungsi membagi titik kontrol menjadi segmen kiri, tengah, dan kanan.
3. Titik kendali dibagi menjadi segmen kiri (kiri), tengah (perhitungan titik tengah), dan kanan (kanan). Secara rekursif, fungsi dipanggil di segmen kiri dan kanan dengan iterasi yang telah dikurangi 1. (Mekanisme *divide*)
4. Terakhir, langkah *conquer* menggabungkan segmen kiri dan kanan beserta bagian tengah (titik tengah) untuk membentuk kurva Bezier.

Metode ini menggunakan algoritma titik tengah berbasis *divide and conquer* untuk membagi kurva secara rekursif menjadi segmen-segmen yang lebih kecil hingga tingkat detail yang diinginkan tercapai. Metode ini diimplementasikan dalam file **N_DnC.py** yang dapat dilihat pada Bab III laporan ini.

BAB II

ALUR PROGRAM

1. Program akan mengeluarkan tampilan utama

```
=====
        Bezier Curve
=====
1. Nth Degree
2. Exit
=====
```

2. Program akan menerima input pilihan yang diinginkan pengguna. Jika input tidak valid, maka program akan terus menerima input pilihan.

```
=====
        Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 3
=====
        Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 4
=====
        Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 5
=====
        Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 6
```

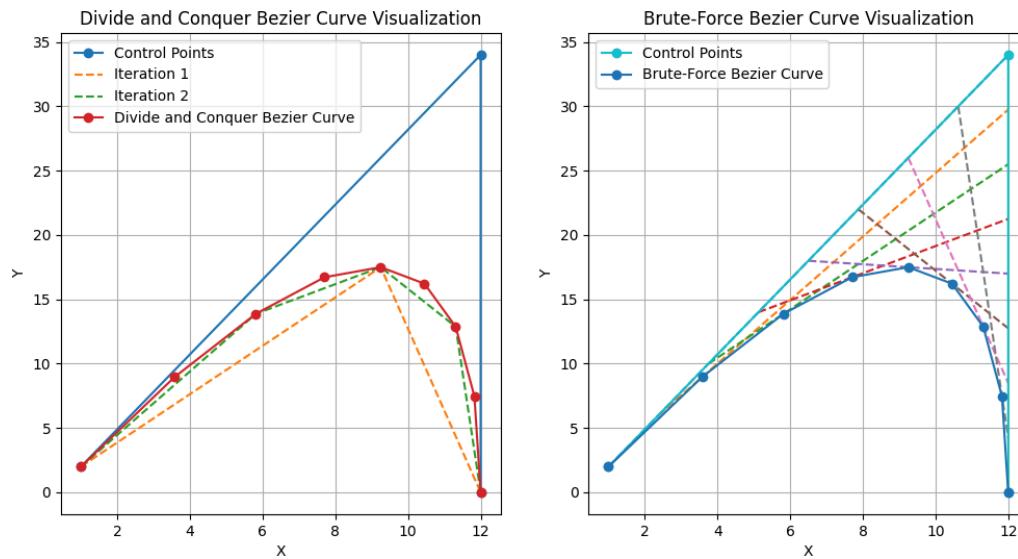
3. Jika pengguna memilih pilihan 1 (*Nth Degree*), maka program akan menerima input jumlah titik. Kemudian, program akan menerima titik-titik sesuai dengan jumlah yang sebelumnya telah diinput. Program juga akan menerima input jumlah iterasi. Jika input titik melebihi 2 angka atau bukan berupa angka, maka program akan meminta ulang input titik.

```
=====
        Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 3
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 1 2
Masukkan titik 2: 3
Input harus berupa dua angka yang dipisahkan oleh spasi.
Masukkan titik 2: m n
Input harus berupa dua angka yang dipisahkan oleh spasi.
Masukkan titik 2: 12 34 mn
Input harus berupa dua angka yang dipisahkan oleh spasi.
Masukkan titik 2: 12 34
Masukkan titik 3: 12 0
Masukkan jumlah iterasi: 3
```

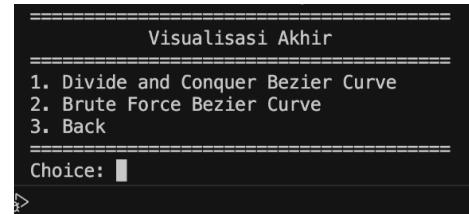
4. Program akan menjalankan pembuatan *Nth bezier curve* dengan metode *divide and conquer* serta metode *brute force*.
5. Program akan menampilkan waktu eksekusi dengan kedua metode dan juga perbandingan waktu eksekusinya.

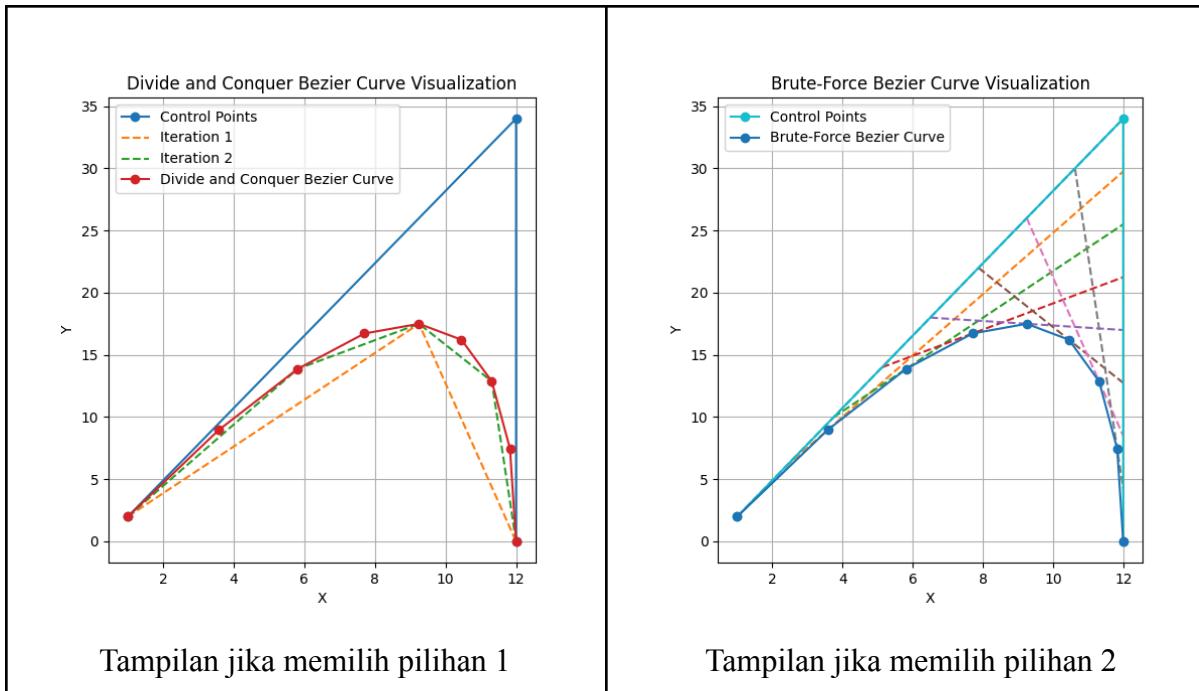
```
Waktu eksekusi Divide and Conquer: 0.1289844512939453 ms
Waktu eksekusi Brute Force: 0.2980232238769531 ms
Divide and Conquer lebih cepat dari Brute Force sebesar 0.1690387725830078 ms
```

6. Program akan menampilkan kurva yang dihasilkan dengan metode *divide and conquer* dan kurva dengan metode *brute force*.



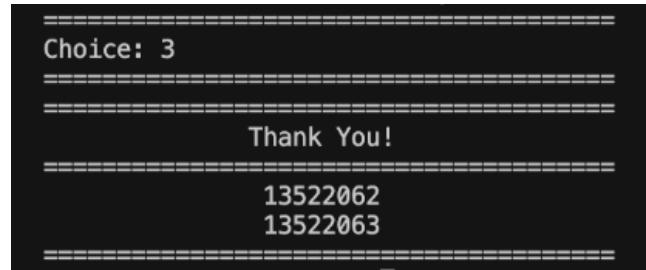
7. Kemudian, program akan mengeluarkan pilihan untuk menampilkan visualisasi akhir.





Jika pengguna memilih 3, maka akan muncul layar utama kembali.

8. Program akan menerima kembali input pilihan. Program hanya akan berhenti ketika pengguna memilih pilihan 2 (*exit*).



BAB III

SOURCE CODE

3.1 Input.py

```
def input_main_menu():
    print("=====")
    print("          Bezier Curve      ")
    print("=====")
    print("1. Nth Degree")
    print("2. Exit")
    print("=====")
    choice = int(input("Choice: "))
    print("=====")

    return choice

def input_visualisation_menu():
    print("=====")
    print("          Visualisasi Akhir      ")
    print("=====")
    print("1. Divide and Conquer Bezier Curve")
    print("2. Brute Force Bezier Curve")
    print("3. Back")
    print("=====")
    choice = int(input("Choice: "))
    print("=====")

    return choice
```

3.2 N_DnC.py

```
from collections import namedtuple
# inisialisasi new tuple dengan jenis point (x, y)
Point = namedtuple("Point", ["x", "y"])
# mencari midpoint di antara 2 titik
def mid_point(point1: Point, point2: Point) -> Point:
    return Point((point1.x + point2.x) / 2, (point1.y + point2.y) / 2)
# mencari titik-titik pembentuk kurva bezier dengan konsep midpoint algorithm
def dnc_bezier_curve(control_points: list[Point], iterations: int) ->
list[Point]:
    initial = control_points.copy()
    titik_awal = control_points[0]
    titik_akhir = control_points[-1]
    # recursion base
    if iterations == 0 :
        return control_points
    elif iterations == 1 :
        for i in range(len(control_points) - 1):
```

```

temp = []
for j in range(len(initial) - 1):
    mid = mid_point(initial[j], initial[j+1])
    temp.append(mid)
initial = temp
mid_points = initial[0]
return [titik_awal, mid_points, titik_akhir]
# recursion iteration
else :
    kanan = [titik_akhir]
    kiri = [titik_awal]
    for i in range(len(control_points) - 1):
        temp = []
        for j in range(len(initial) - 1):
            mid = mid_point(initial[j], initial[j+1])
            temp.append(mid)
        kanan.insert(0, temp[-1])
        kiri.append(temp[0])
        initial = temp
    mid_points = initial[0]

    # divide bagi kiri, tengah, kanan
    left = dnc_bezier_curve(kiri, iterations - 1)
    middle = [mid_points]
    right = dnc_bezier_curve(kanan, iterations - 1)

    # conquer (gabungin)
    return left[:-1] + middle + right[1:]

```

3.3 N_BF.py

```

from typing import List, Tuple
import numpy as np
Point_N = Tuple[float, float]
def interpolate(control_points: List[Point_N], t: float) -> Point_N:
    if len(control_points) == 1:
        return control_points[0]

    # Interpolate between 2 control points
    interpolated_points = []
    for i in range(len(control_points) - 1):
        x = (1 - t) * control_points[i][0] + t * control_points[i + 1][0]
        y = (1 - t) * control_points[i][1] + t * control_points[i + 1][1]
        interpolated_points.append((x, y))

    return interpolate(interpolated_points, t)
def bf_bezier_curve(control_points: List[Point_N], num_points: int) ->
List[Point_N]:
    bezier_points = []

```

```

for t in np.linspace(0, 1, num_points):
    bezier_point = interpolate(control_points, t)
    bezier_points.append(bezier_point)

return bezier_points

```

3.4 Visualisation.py

```

import matplotlib.pyplot as plt
from N_DnC import *
from N_BF import *
def interpolate_vis(control_points: List[Point_N], t: float) -> Point_N:
    if len(control_points) == 1:
        return control_points[0]

    # Interpolate between 2 control points
    interpolated_points = []
    for i in range(len(control_points) - 1):
        x = (1 - t) * control_points[i][0] + t * control_points[i + 1][0]
        y = (1 - t) * control_points[i][1] + t * control_points[i + 1][1]
        interpolated_points.append((x, y))
        plt.plot

    x_values = [point[0] for point in interpolated_points]
    y_values = [point[1] for point in interpolated_points]
    plt.plot(x_values, y_values, '--')

    return interpolate(interpolated_points, t)
def bf_bezier_curve_interpolate_vis(control_points: List[Point_N], num_points: int) -> List[Point_N]:
    bezier_points = []
    for t in np.linspace(0, 1, num_points):
        bezier_point = interpolate_vis(control_points, t)
        bezier_points.append(bezier_point)

    return bezier_points
def plot_dnc_bezier_curve(control_points: List[Point], iteration: int):
    fig, ax = plt.subplots()
    # Bezier Curve per Iteration
    for i in range(iteration + 1):
        curve_points = dnc_bezier_curve(control_points, i)
        curve_x = [point.x for point in curve_points]
        curve_y = [point.y for point in curve_points]
        if i == 0:
            ax.plot(curve_x, curve_y, 'o-', label='Control Points')
        elif i != iteration:
            ax.plot(curve_x, curve_y, '--', label=f'Iteration {i}')
        else:
            ax.plot(curve_x, curve_y, 'o-', label=f'Divide and Conquer Bezier'

```

```

Curve')
    ax.legend()
    ax.set_title('Divide and Conquer Bezier Curve Visualization')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    plt.grid(True)
    plt.show()
def plot_bf_bezier_curve(control_points: List[Point], num_points: int):
    bf_curve_points = bf_bezier_curve_interpolate_vis(control_points,
num_points)
    # Control points
    control_x = [point[0] for point in control_points]
    control_y = [point[1] for point in control_points]
    plt.plot(control_x, control_y, 'o-', label='Control Points')
    # Bezier curve
    bf_curve_x = [point[0] for point in bf_curve_points]
    bf_curve_y = [point[1] for point in bf_curve_points]
    plt.plot(bf_curve_x, bf_curve_y, 'o-', label='Brute-Force Bezier Curve')
    plt.title('Brute-Force Bezier Curve Visualization')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend()
    plt.grid(True)
    plt.show()
def animate(iteration, control_points, ax1, ax2):
    ax1.clear()
    ax2.clear()
    for i in range(iteration + 1):
        curve_points = dnc_bezier_curve(control_points, i)
        curve_x = [point.x for point in curve_points]
        curve_y = [point.y for point in curve_points]
        if i == 0:
            ax1.plot(curve_x, curve_y, 'o-', label='Control Points')
        elif i != iteration:
            ax1.plot(curve_x, curve_y, '--', label=f'Iteration {i}')
        else:
            ax1.plot(curve_x, curve_y, 'o-', label=f'Divide and Conquer Bezier
Curve')
    ax1.legend()
    ax1.set_title('Divide and Conquer Bezier Curve Visualization')
    ax1.set_xlabel('X')
    ax1.set_ylabel('Y')
    ax1.grid(True)
    bf_curve_points = bf_bezier_curve_interpolate_vis(control_points,
2**iteration + 1)
    bf_curve_x = [point[0] for point in bf_curve_points]
    bf_curve_y = [point[1] for point in bf_curve_points]
    control_x = [point.x for point in control_points]

```

```

control_y = [point.y for point in control_points]
ax2.plot(control_x, control_y, 'o-', label='Control Points')
ax2.plot(bf_curve_x, bf_curve_y, 'o-', label='Brute-Force Bezier Curve')
ax2.legend()
ax2.set_title('Brute-Force Bezier Curve Visualization')
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.grid(True)

```

3.5 main.py

```

from N_DnC import dnc_bezier_curve, Point
from N_BF import bf_bezier_curve
from Visualisation import *
from Input import *
from matplotlib.animation import FuncAnimation
import time
import numpy as np
choice = input_main_menu()
stop_program = False
while(stop_program == False):
    while(choice != 1 and choice != 2):
        choice = input_main_menu()
    if(choice == 1):
        jumlah_titik = int(input("Masukkan jumlah titik: "))
        print("Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).")
        print("Contoh: 0 0")
        control_points = []
        for i in range(jumlah_titik):
            while True:
                try:
                    x, y = input(f"Masukkan titik {i+1}: ").split()
                    temp_titik = Point(float(x), float(y))
                    control_points.append(temp_titik)
                    break # loop berhenti ketika sudah input benar
                except ValueError:
                    print("Input harus berupa dua angka yang dipisahkan oleh spasi.")
                    continue # ulangi minta input
        iteration = int(input("Masukkan jumlah iterasi: "))
        num_points = (2 ** iteration) + 1

        start_dnc = time.time()
        dnc_points = dnc_bezier_curve(control_points, iteration)
        end_dnc = time.time()
        start_bf = time.time()
        bf_points = bf_bezier_curve(control_points, num_points)
        end_bf = time.time()

        control_y = [point.y for point in control_points]
        ax2.plot(control_x, control_y, 'o-', label='Control Points')
        ax2.plot(bf_curve_x, bf_curve_y, 'o-', label='Brute-Force Bezier Curve')
        ax2.legend()
        ax2.set_title('Brute-Force Bezier Curve Visualization')
        ax2.set_xlabel('X')
        ax2.set_ylabel('Y')
        ax2.grid(True)
    
```

```

dnc_time = np.float16(end_dnc-start_dnc)*1000
bf_time = np.float16(end_bf-start_bf)*1000
print()
print("Waktu eksekusi Divide and Conquer:", dnc_time, "ms")
print("Waktu eksekusi Brute Force:", bf_time, "ms")
if bf_time > dnc_time:
    print("Divide and Conquer lebih cepat dari Brute Force sebesar",
bf_time-dnc_time, "ms")
else:
    print("Brute Force lebih cepat dari Divide and Conquer sebesar",
dnc_time-bf_time, "ms")

# Animation
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
ani = FuncAnimation(fig, animate, frames=range(iteration + 1),
fargs=(control_points, ax1, ax2), interval=350, repeat=False)
plt.show()
# Final Result Visualisation
vis_choice = input_visualisation_menu()
while(vis_choice != 1 and vis_choice != 2 and vis_choice != 3):
    vis_choice = input_visualisation_menu()
while(vis_choice == 1 or vis_choice == 2):
    if(vis_choice == 1):
        plot_dnc_bezier_curve(control_points, iteration)
        vis_choice = input_visualisation_menu()
    if(vis_choice == 2):
        plot_bf_bezier_curve(control_points, num_points)
        vis_choice = input_visualisation_menu()

choice = input_main_menu()
if(choice == 2):
    stop_program = True
    print("=====")
    print("          Thank You!          ")
    print("=====")
    print("          13522062          ")
    print("          13522063          ")
    print("=====")

```

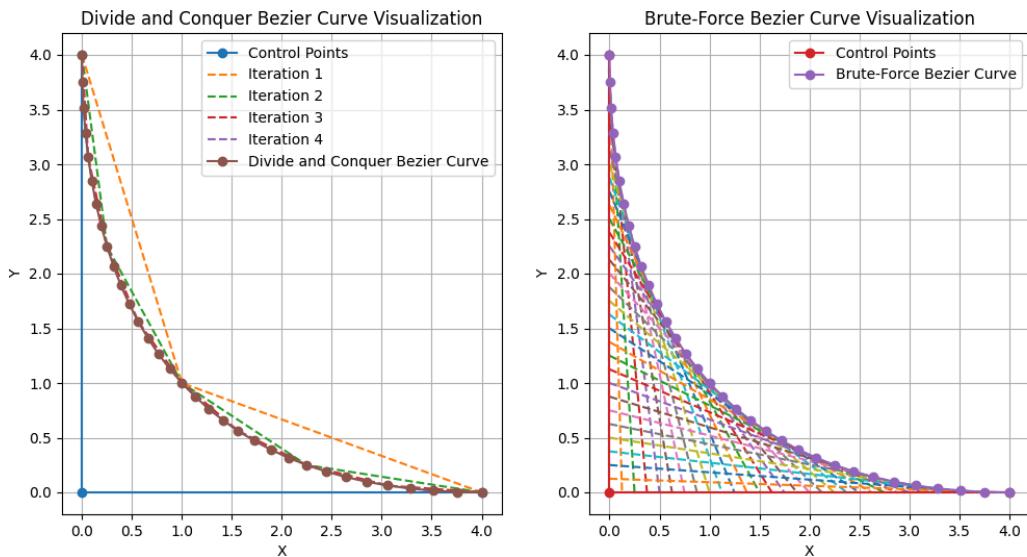
BAB IV

HASIL PENGUJIAN

4.1 Test Case 1

```
=====
Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 3
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 0 4
Masukkan titik 2: 0 0
Masukkan titik 3: 4 0
Masukkan jumlah iterasi: 5

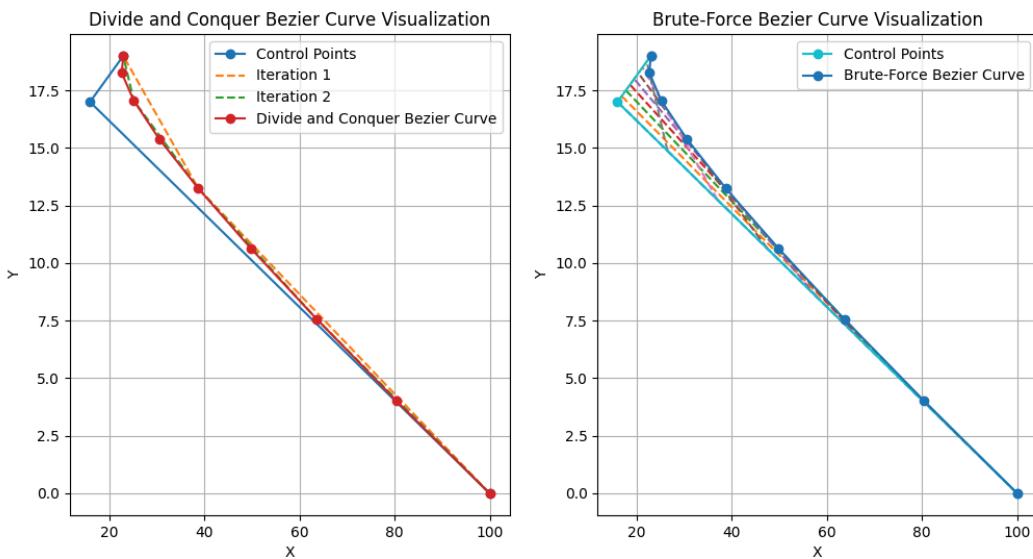
Waktu eksekusi Divide and Conquer: 0.2181529998779297 ms
Waktu eksekusi Brute Force: 0.3848075866699219 ms
Divide and Conquer lebih cepat dari Brute Force sebesar 0.1666545867919922 ms
```



4.2 Test Case 2

```
=====
        Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 3
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 100 0
Masukkan titik 2: 16 17
Masukkan titik 3: 23 19
Masukkan jumlah iterasi: 3

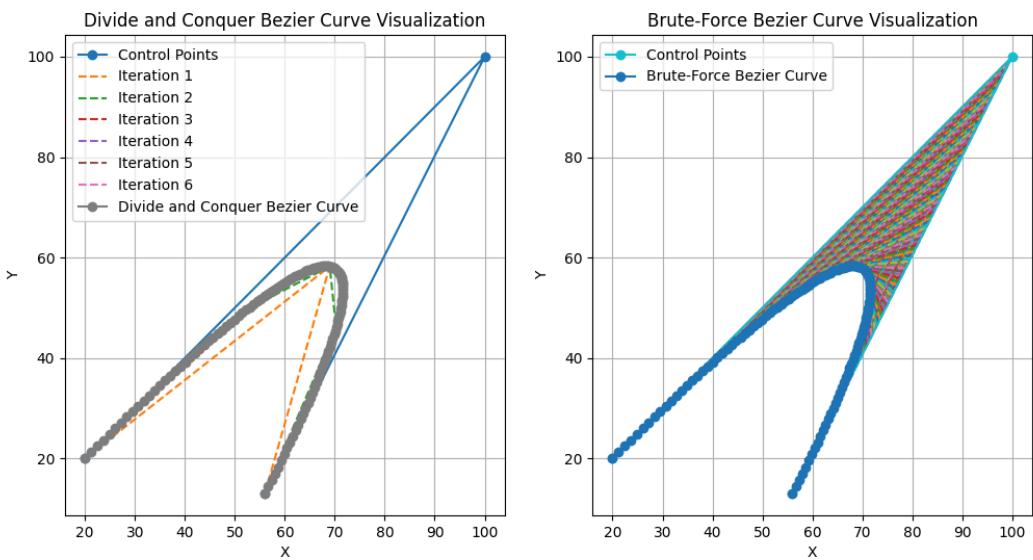
Waktu eksekusi Divide and Conquer: 0.11491775512695312 ms
Waktu eksekusi Brute Force: 0.30803680419921875 ms
Divide and Conquer lebih cepat dari Brute Force sebesar 0.19311904907226562 ms
```



4.3 Test Case 3

```
=====
        Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 3
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 56 13
Masukkan titik 2: 100 100
Masukkan titik 3: 20 20
Masukkan jumlah iterasi: 7

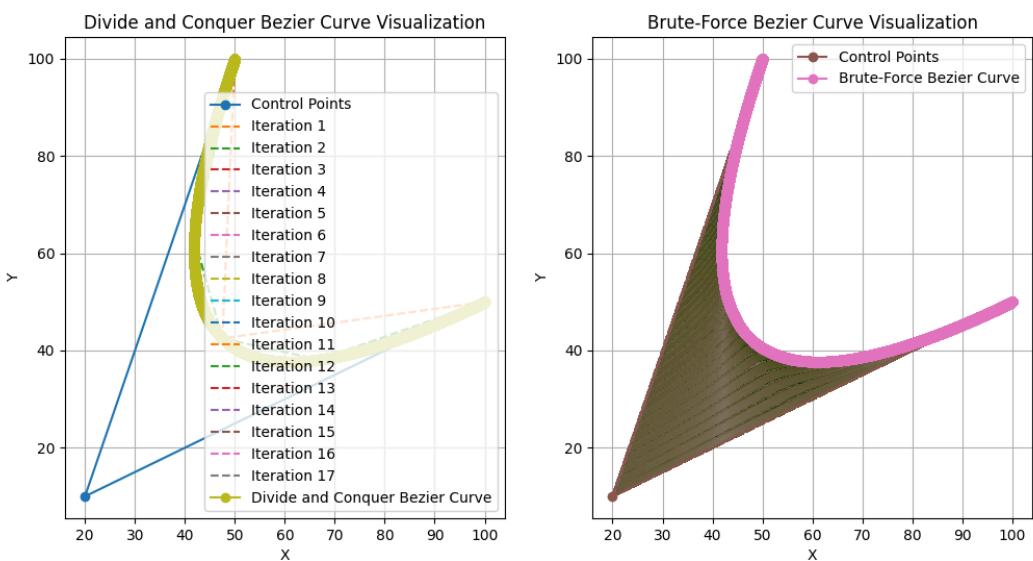
Waktu eksekusi Divide and Conquer: 0.9517669677734375 ms
Waktu eksekusi Brute Force: 1.1110305786132812 ms
Divide and Conquer lebih cepat dari Brute Force sebesar 0.15926361083984375 ms
```



4.4 Test Case 4

```
=====
Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 3
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 100 50
Masukkan titik 2: 20 10
Masukkan titik 3: 50 100
Masukkan jumlah iterasi: 18

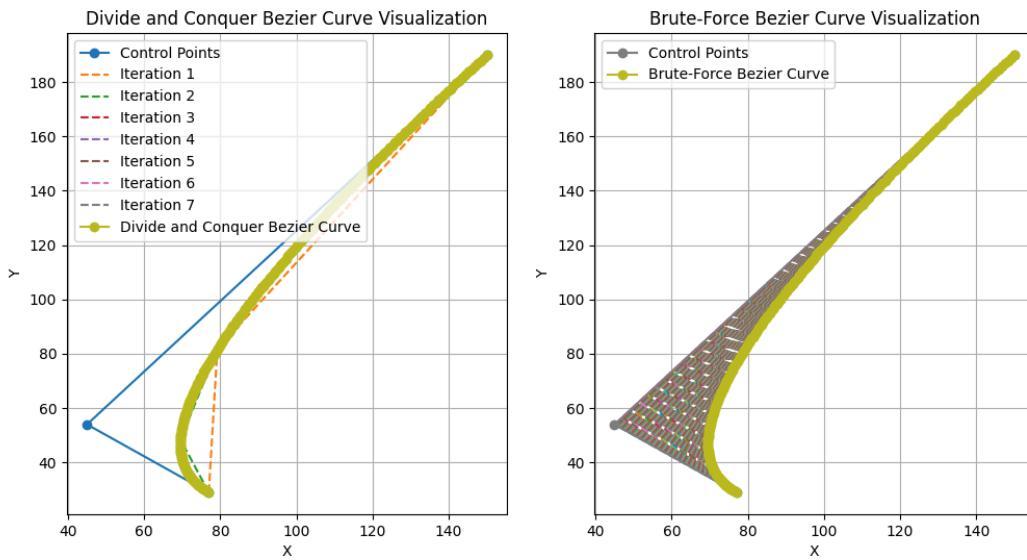
Waktu eksekusi Divide and Conquer: 458.0078125 ms
Waktu eksekusi Brute Force: 344.23828125 ms
Brute Force lebih cepat dari Divide and Conquer sebesar 113.76953125 ms
=====
```



4.5 Test Case 5

```
=====
Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 3
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 77 29
Masukkan titik 2: 45 54
Masukkan titik 3: 150 190
Masukkan jumlah iterasi: 8

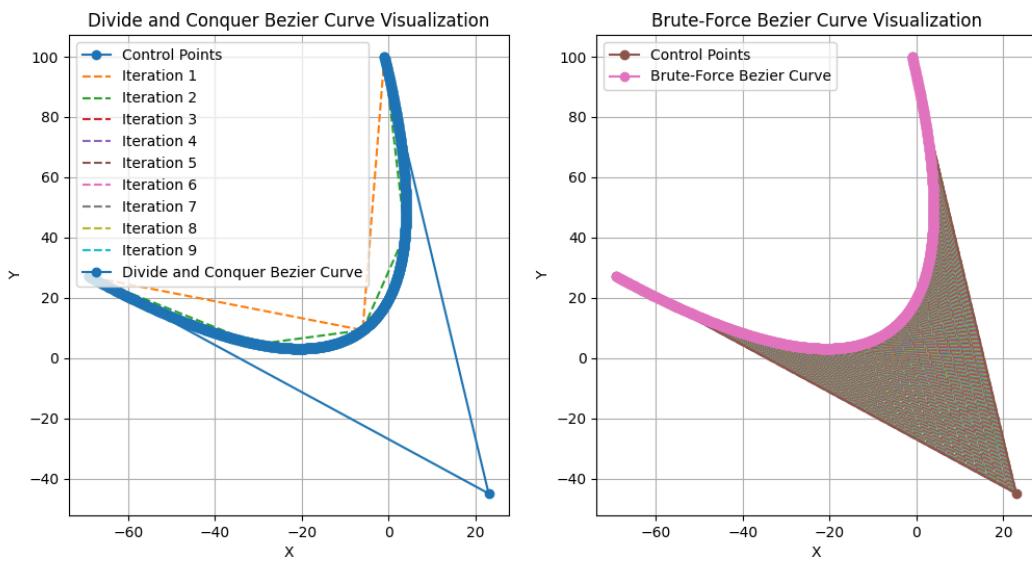
Waktu eksekusi Divide and Conquer: 0.3921985626220703 ms
Waktu eksekusi Brute Force: 0.4680156707763672 ms
Divide and Conquer lebih cepat dari Brute Force sebesar 0.07581710815429688 ms
=====
```



4.6 Test Case 6

```
=====
Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 3
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: -1 100
Masukkan titik 2: 23 -45
Masukkan titik 3: -69 27
Masukkan jumlah iterasi: 10

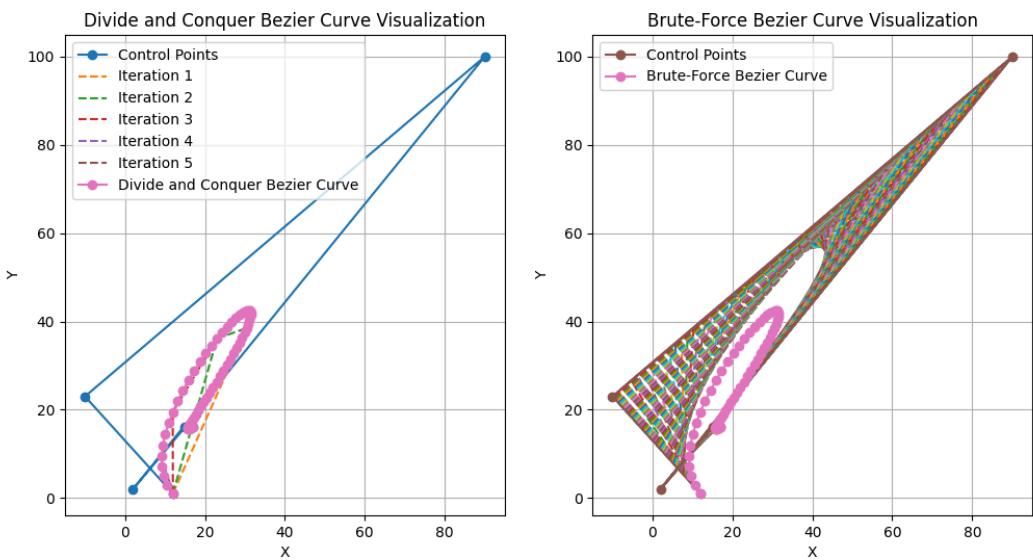
Waktu eksekusi Divide and Conquer: 3.8967132568359375 ms
Waktu eksekusi Brute Force: 3.11279296875 ms
Brute Force lebih cepat dari Divide and Conquer sebesar 0.7839202880859375 ms
=====
```



4.7 Test Case 7 (Bonus_1)

```
=====
Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 6
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 12 1
Masukkan titik 2: -10 23
Masukkan titik 3: 90 100
Masukkan titik 4: 2
Input harus berupa dua angka yang dipisahkan oleh spasi.
Masukkan titik 4: 2 2
Masukkan titik 5: 15 16
Masukkan titik 6: 17 16
Masukkan jumlah iterasi: 6

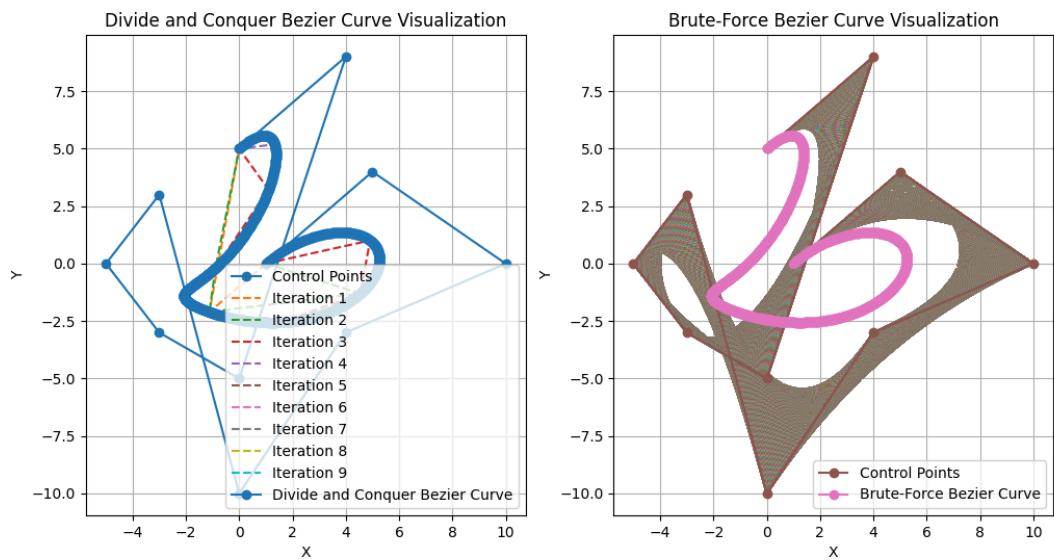
Waktu eksekusi Divide and Conquer: 1.392364501953125 ms
Waktu eksekusi Brute Force: 1.5840530395507812 ms
Divide and Conquer lebih cepat dari Brute Force sebesar 0.19168853759765625 ms
|
```



4.8 Test Case 8 (Bonus_2)

```
=====
        Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 11
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 1 0
Masukkan titik 2: 5 4
Masukkan titik 3: 10 0
Masukkan titik 4: 4 -3
Masukkan titik 5: 0 -10
Masukkan titik 6: -3 3
Masukkan titik 7: -5 0
Masukkan titik 8: -3 -3
Masukkan titik 9: 0 -5
Masukkan titik 10: 4 9
Masukkan titik 11: 0 5
Masukkan jumlah iterasi: 10

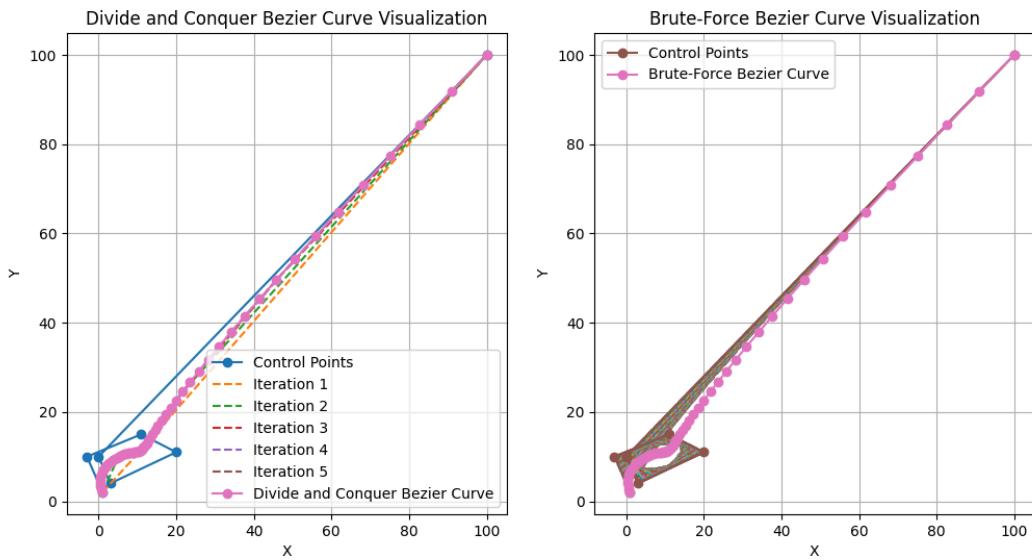
Waktu eksekusi Divide and Conquer: 30.4412841796875 ms
Waktu eksekusi Brute Force: 20.5841064453125 ms
Brute Force lebih cepat dari Divide and Conquer sebesar 9.857177734375 ms
=====
Ln 8, Col 1   Spaces: 4   UTF-8   LF   Python
```



4.9 Test Case 9 (Bonus_3)

```
=====
Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 7
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 1 2
Masukkan titik 2: -3 10
Masukkan titik 3: 11 15
Masukkan titik 4: 20 11
Masukkan titik 5: 3 4
Masukkan titik 6: 0 10
Masukkan titik 7: 100 100
Masukkan jumlah iterasi: 6

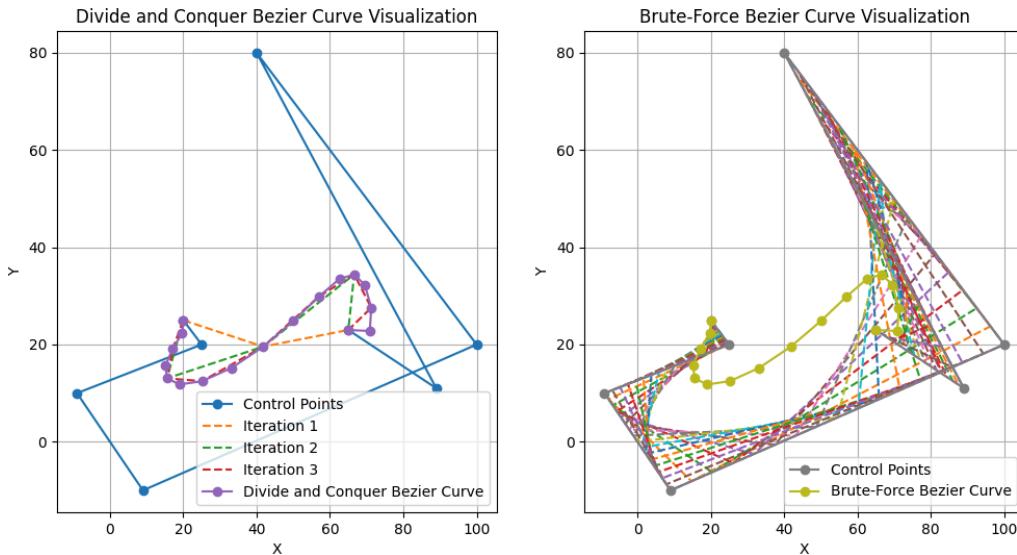
Waktu eksekusi Divide and Conquer: 15.2435302734375 ms
Waktu eksekusi Brute Force: 19.7906494140625 ms
Divide and Conquer lebih cepat dari Brute Force sebesar 4.547119140625 ms
=====
```



4.10 Test Case 10 (Bonus_4)

```
=====
Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 8
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 20 25
Masukkan titik 2: 25 20
Masukkan titik 3: -9 10
Masukkan titik 4: 9 -10
Masukkan titik 5: 100 20
Masukkan titik 6: 40 80
Masukkan titik 7: 89 11
Masukkan titik 8: 65 23
Masukkan jumlah iterasi: 4

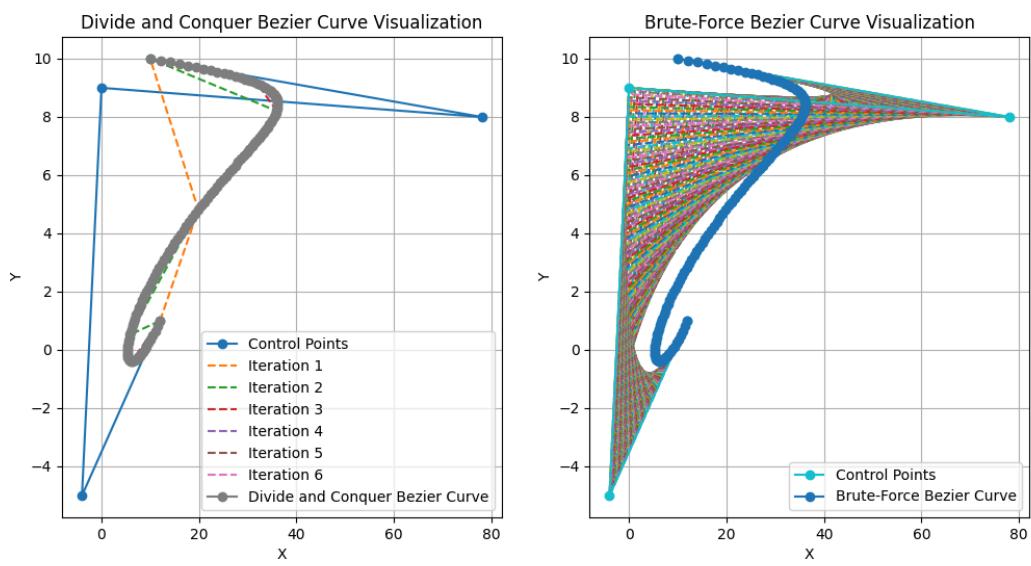
Waktu eksekusi Divide and Conquer: 0.6861686706542969 ms
Waktu eksekusi Brute Force: 0.8788108825683594 ms
Divide and Conquer lebih cepat dari Brute Force sebesar 0.1926422119140625 ms
```



4.11 Test Case 11 (Bonus_5)

```
=====
Bezier Curve
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 5
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: 12 1
Masukkan titik 2: -4 -5
Masukkan titik 3: 0 9
Masukkan titik 4: 78 8
Masukkan titik 5: 10 10
Masukkan jumlah iterasi: 7

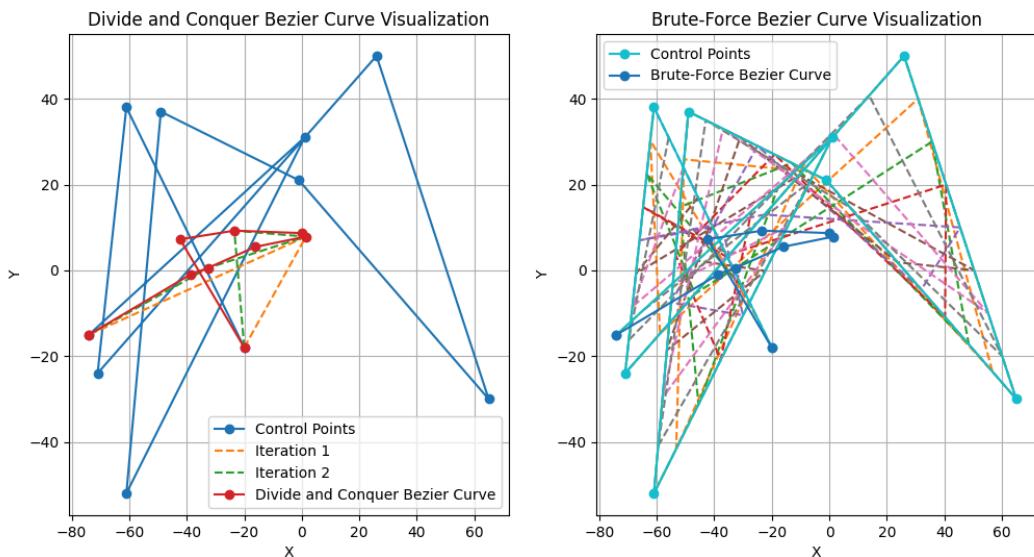
Waktu eksekusi Divide and Conquer: 1.3275146484375 ms
Waktu eksekusi Brute Force: 1.678466796875 ms
Divide and Conquer lebih cepat dari Brute Force sebesar 0.3509521484375 ms
```



4.12 Test Case 12 (Bonus_6)

```
=====
1. Nth Degree
2. Exit
=====
Choice: 1
=====
Masukkan jumlah titik: 10
Input titik (x dan y dipisahkan oleh spasi). Titik dimulai dengan titik awal dan diakhiri dengan titik akhir (berurutan).
Contoh: 0 0
Masukkan titik 1: -20 -18
Masukkan titik 2: -61 38
Masukkan titik 3: -71 -24
Masukkan titik 4: 26 50
Masukkan titik 5: 65
Input harus berupa dua angka yang dipisahkan oleh spasi.
Masukkan titik 5: 65 -30
Masukkan titik 6: -1 21
Masukkan titik 7: -49 37
Masukkan titik 8: -61 -52
Masukkan titik 9: 1 31
Masukkan titik 10: -74 -15
Masukkan jumlah iterasi: 3

Waktu eksekusi Divide and Conquer: 0.8182525634765625 ms
Waktu eksekusi Brute Force: 1.1854171752929688 ms
Divide and Conquer lebih cepat dari Brute Force sebesar 0.36716461181640625 ms
=====
```



BAB V

ANALISIS PERBANDINGAN SOLUSI

Berdasarkan 12 pengujian yang telah kami lakukan pada bab IV, kami menemukan bahwa pada beberapa percobaan, metode divide and conquer memiliki waktu eksekusi yang lebih kecil atau lebih cepat daripada metode brute force. Hal tersebut kami amati pada jumlah iterasi yang lebih kecil atau sama dengan 8. Jika melebihi iterasi tersebut, hasil waktu eksekusi brute force cenderung lebih cepat. Dari sini, kami menyimpulkan bahwa kesangkilan program ini bergantung dengan jumlah iterasi yang perlu dilakukan.

Kami juga mengamati bahwa perbedaan waktu terbesar atau paling signifikan terdapat pada percobaan 4 yang menggunakan 18 iterasi dengan perbedaan waktu 113,79 ms. Namun, mengingat bahwa pada percobaan 6 dengan 10 iterasi yang memiliki perbedaan waktu 0,78 ms dan percobaan 5 dengan 8 iterasi yang memiliki perbedaan waktu 0,079 ms, maka tidak dapat langsung disimpulkan bahwa semakin banyak iterasi, maka semakin besar perbedaan waktu.

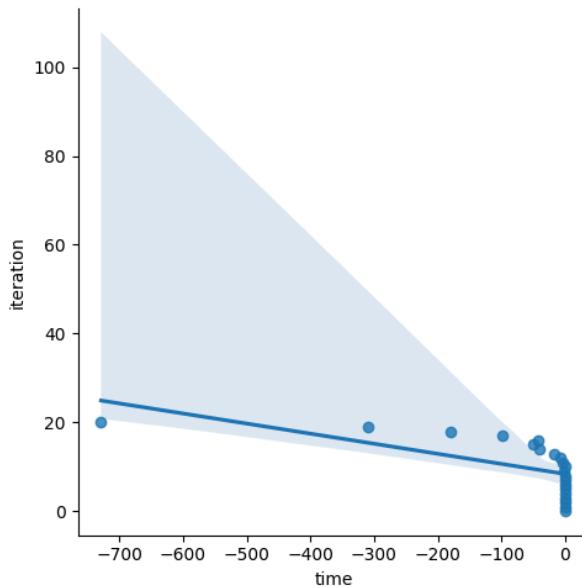
Ada beberapa hal yang kami rasa signifikan pada perbandingan dua algoritma tersebut mengenai mengapa ada kasus di mana *brute force* lebih cepat. Hal-hal tersebut meliputi:

1. Algoritma rekursif dapat menimbulkan *overhead* karena pemanggilan fungsi, manajemen tumpukan, dan penghitungan berulang.
2. Algoritma rekursif mungkin memerlukan memori tambahan untuk pemanggilan fungsi rekursif dan mempertahankan status.
3. Pendekatan brute force yang mencari nilai langsung mungkin melibatkan logika yang lebih sederhana dibandingkan dengan algoritma rekursif, sehingga menghasilkan eksekusi yang lebih cepat dan optimasi yang lebih mudah.

Namun, perlu diperhatikan bahwa performa suatu algoritma dapat bergantung pada berbagai faktor, termasuk implementasi spesifik, ukuran data input, platform perangkat keras, dan optimalisasi compiler. Dalam beberapa kasus, algoritma rekursif yang dioptimalkan dengan baik mungkin memiliki kinerja yang sama baiknya atau bahkan lebih baik daripada pendekatan iteratif pada metode *brute force*.

Untuk mengetahui korelasi antara jumlah titik kontrol dengan perbedaan waktu *processing* dan jumlah iterasi dengan perbedaan waktu *processing*, kami melakukan 2 percobaan untuk mendapatkan nilai korelasi antara variabel-variabel tersebut. Pada percobaan 1, jumlah iterasi merupakan variabel bebas, jumlah titik kontrol sebagai variabel kontrol, serta perbedaan waktu

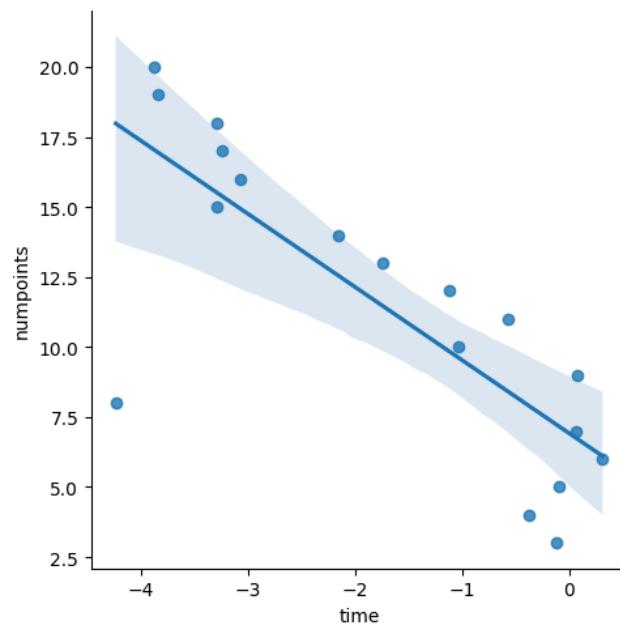
sebagai variabel terikat, sedangkan pada percobaan 2, jumlah titik kontrol merupakan variabel bebas dan jumlah iterasi berperan sebagai variabel kontrol.



	time	iteration
time	1.000000	-0.617886
iteration	-0.617886	1.000000

Iteration	Time Difference (ms)
0	0.5819797515869141
1	0.21982192993164062
2	0.18787384033203125
3	0.12159347534179688
4	0.17333030700683594
5	0.1537799835205078
6	0.16689300537109375
7	0.1468658447265625
8	-0.171661376953125
9	-0.7610321044921875

10	-0.598907470703125
11	-3.2501220703125
12	-7.14111328125
13	-16.7236328125
14	-41.259765625
15	-49.86572265625
16	-42.48046875
17	-99.12109375
0	-180.6640625
1	-309.5703125
2	-728.515625



	time	numpoints
time	1.000000	-0.790354
numpoints	-0.790354	1.000000

Control Points	Time Difference (ms)
3	-0.11539459228515625
4	-0.37479400634765625
5	-0.095367431640625
6	0.31280517578125
7	0.0629425048828125
8	-4.2362213134765625
9	-0.0782012939453125
10	-1.033782958984375
11	-0.57220458984375
12	-1.1138916015625
13	-1.739501953125
14	-2.1514892578125
15	-3.28826904296875
16	-3.0670166015625
17	-3.238677978515625
18	-3.2958984375
19	-3.83758544921875
20	-3.875732421875

Pada percobaan tersebut, *time difference* (perbedaan durasi kedua algoritma) yang negatif menyatakan bahwa *brute force* lebih cepat dan jika positif, maka *divide and conquer* lebih cepat. Berdasarkan percobaan tersebut, kami memperoleh nilai korelasi sebesar -0,617867 pada

hubungan antara *time difference* kedua algoritma dengan iterasi, yang mana hal ini menunjukkan bahwa semakin banyak iterasi, maka metode *brute force* akan memiliki perbandingan waktu *processing* yang semakin signifikan dibandingkan dengan *divide and conquer*. Hal yang sama juga terjadi pada hubungan antara *time difference* kedua algoritma dengan jumlah titik kontrol yang memiliki nilai korelasi sebesar -0.790354, yang mana hal ini menunjukkan bahwa apabila pengguna memberikan jumlah titik yang besar, maka menggunakan metode *brute force* akan lebih cepat daripada menggunakan metode *divide and conquer*.

Kami juga telah melakukan analisis time complexity dan space complexity dari kedua algoritma (brute force dan divide and conquer) dengan hasil sebagai berikut:

1. Brute Force

- Kompleksitas Fungsi interpolate:

Fungsi ini melakukan interpolasi linier antara titik kontrol yang berdekatan secara rekursif hingga hanya tersisa satu titik kontrol. Misalkan n adalah banyaknya titik kendali. Dalam setiap panggilan rekursif, ia melakukan operasi $O(n)$ untuk menghitung titik interpolasi. Jumlah panggilan rekursif adalah $\log_2(n)$ (dengan asumsi n adalah power of 2). Oleh karena itu, kompleksitas waktu dapat dinyatakan dengan $T(n) = O(n) + T(n/2)$. Fungsi ini memiliki kompleksitas waktu **$O(n)$** . Kompleksitas ruang fungsi ini juga **$O(n)$** karena ruang tumpukan panggilan rekursif.

- Kompleksitas Fungsi *bf_bezier_curve*:

Fungsi ini menghasilkan titik-titik pada kurva Bezier dengan melakukan interpolasi antar titik kontrol menggunakan fungsi interpolasi. Misalkan n adalah jumlah titik kontrol dan m adalah jumlah titik yang dihasilkan pada kurva Bezier. Dengan $m = 2^p + 1$, p adalah jumlah iterasi. Fungsi ‘*interpolate*’ memiliki kompleksitas waktu $O(n)$. Oleh karena itu, kompleksitas waktu *bf_bezier_curve* adalah **$O(2^p * n)$** . Kompleksitas ruang adalah **$O(2^p * n)$** karena penyimpanan titik Bezier yang dihasilkan.

2. Divide and Conquer

- Kompleksitas Fungsi *dnc_bezier_curve*:

Fungsi ini menggunakan pendekatan bagi-dan-taklukkan untuk menghitung titik-titik pada kurva Bezier. Misalkan jumlah titik kontrol sebagai n dan jumlah iterasi sebagai m . Dalam kasus dasar (iterasi = 0 atau 1), fungsi melakukan operasi $O(n)$

untuk menghitung titik tengah. Dalam kasus rekursif (iterasi > 1), fungsi membagi titik kontrol menjadi segmen kiri, tengah, dan kanan dan memanggil dirinya sendiri dua kali secara rekursif. Setiap panggilan rekursif mengurangi jumlah titik kontrol sekitar setengahnya. Jumlah total panggilan rekursif adalah 2^m . Oleh karena itu, kompleksitas waktu dapat dinyatakan sebagai $O(n * 2^m)$. Kompleksitas ruang juga dipengaruhi oleh panggilan rekursif dan $O(m)$ karena kedalaman tumpukan panggilan.

BAB VI

BONUS

6.1 N titik

Bonus ini kami implementasikan sekalian dengan implementasi untuk spesifikasi wajib, yaitu kurva Bezier kuadratik. Dengan mengimplementasikan ide yang sama pada keduanya, kami menggunakan algoritma midpoint untuk terus mencari titik tengah pada pada yang sudah dibagi menjadi segmen-segmen. Kemudian, hasil akhirnya menggunakan penggabungan semua hasil segmen-segmen tersebut. Secara lengkap, langkah dari ide tersebut adalah:

1. Membuat basis rekursif, yaitu ketika iterasi = 0 atau iterasi = 1. Jika iterasi = 0, maka kode akan mengembalikan titik kontrol asli. Jika iterasi = 1, maka kode akan menghitung titik tengah antara titik kontrol yang berdekatan secara iteratif dan mengembalikan titik awal, titik-titik tengah, dan titik akhir.
2. Jika jumlah iterasi lebih dari 1, fungsi membagi titik kontrol menjadi segmen kiri, tengah, dan kanan.
3. Titik kendali dibagi menjadi segmen kiri (kiri), tengah (perhitungan titik tengah), dan kanan (kanan). Secara rekursif, fungsi dipanggil di segmen kiri dan kanan dengan iterasi yang telah dikurangi 1. (Mekanisme *divide*)
4. Terakhir, langkah *conquer* menggabungkan segmen kiri dan kanan beserta bagian tengah (titik tengah) untuk membentuk kurva Bezier.

Metode ini menggunakan algoritma titik tengah berbasis *divide and conquer* untuk membagi kurva secara rekursif menjadi segmen-segmen yang lebih kecil hingga tingkat detail yang diinginkan tercapai. Metode ini diimplementasikan dalam file **N_DnC.py** yang dapat dilihat pada Bab III laporan ini.

6.2 Animasi Iterasi

Animasi per iterasi dari metode *divide and conquer* serta *brute force* menggunakan modul matplotlib dengan fungsi FuncAnimation. Untuk animasi yang menjadi parameter dari fungsi tersebut, implementasinya terdapat pada fungsi animate pada file **Visualisation.py**. Fungsi animate memiliki parameter iteration untuk menampilkan kurva pada tiap tahapan iterasi untuk metode *divide and conquer* dan digunakan untuk menghitung jumlah titik akhir pada metode

brute force, *control_points* yang merupakan *control point* awal masukkan pengguna, serta *ax1* dan *ax2* yang merupakan sumbu objek.

Pada visualisasi metode *divide and conquer*, akan dilakukan pengulangan dari iterasi ke-0 hingga ke-n dan pada setiap iterasinya, titik-titik yang dihasilkan akan diplot pada *graph*. Iterasi ke-0 akan dilabeli sebagai *control points* dan iterasi ke-n akan dilabeli sebagai kurva Bezier dengan metode *divide and conquer*.

Berbeda dengan visualisasi metode *divide and conquer*, visualisasi metode *brute force* menggunakan fungsi baru yang merupakan fungsi metode *brute force* ditambahkan dengan plotting garis pada setiap iterasi perhitungan titik interpolasinya. Setelah itu, *control points* dan titik akhir akan diplot dan dilabeli sesuai nama masing-masing.

BAB VII

LAMPIRAN

7.1 Link Repository

https://github.com/zyaaa-aaa/Tucil2_13522062_13522063.git

7.2 Tabel Kelengkapan

Poin	Ya	Tidak
1. Program berhasil dijalankan.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2. Program dapat melakukan visualisasi kurva Bézier.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3. Solusi yang diberikan program optimal.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4. [Bonus] Program dapat membuat kurva untuk n titik kontrol.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5. [Bonus] Program dapat melakukan visualisasi proses pembuatan kurva.	<input checked="" type="checkbox"/>	<input type="checkbox"/>