



Institut Teknologi Bandung

Sekolah Teknik Elektro dan Informatika

Program Studi Informatika Semester II 2023/2024

Laporan Tugas Kecil 2

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

IF2211 Strategi Algoritma

Denise Felicia Tiowanni 13522013

Zahira Dina Amalia 13522085

Daftar Isi

Daftar Isi.....	2
BAB I: Analisis dan Implementasi dalam Algoritma Brute Force.....	3
BAB II: Analisis dan Implementasi dalam Algoritma Divide and Conquer.....	5
BAB III: Source Code Program.....	7
III. 1 Algoritma Brute Force.....	7
III. 2 Algoritma Divide & Conquer.....	9
III. 3 GUI.....	17
BAB IV: Tangkapan Layar.....	26
IV. 1 Test Case 1.....	26
IV. 2 Test Case 2.....	26
IV. 3 Test Case 3.....	27
IV. 4 Test Case 4.....	28
IV. 5 Test Case 5.....	28
IV. 6 Test Case 6.....	29
BAB V: Analisis Perbandingan Solusi Brute Force dan Divide and Conquer.....	30
BAB VI: Implementasi Bonus.....	31
VI. 1 Generalisasi untuk N Titik Kontrol.....	31
VI. 2 Visualisasi proses pembentukan kurva (GUI).....	33
Lampiran.....	35

BAB I

Analisis dan Implementasi dalam Algoritma

Brute Force

Dalam analisis ini, dievaluasi algoritma *brute force* sebagai pendekatan untuk menghitung titik-titik pada kurva Bézier. Algoritma *brute force* adalah pendekatan yang sederhana dan langsung, di mana setiap kemungkinan solusi dieksplorasi secara berurutan tanpa memanfaatkan struktur masalah.

Kurva Bézier adalah kurva parametrik yang dihasilkan oleh satu atau lebih titik kontrol yang dikenal sebagai titik-titik Bézier. Algoritma *brute force* untuk menghasilkan titik-titik pada kurva Bézier melibatkan iterasi melalui semua kemungkinan nilai parameter t dalam rentang $[0, 1]$. Pada setiap iterasi, titik kurva dihitung menggunakan rumus kurva Bézier, yang melibatkan perhitungan berulang untuk setiap titik kontrol.

Keuntungan utama dari algoritma *brute force* adalah kesederhanaannya. Ini mudah dipahami dan diimplementasikan, membuatnya cocok untuk pemecahan masalah yang sederhana dan kebutuhan implementasi cepat. Namun, kelemahannya adalah kompleksitas waktu yang meningkat secara eksponensial dengan jumlah titik kontrol dan jumlah iterasi. Hal ini dapat membuat algoritma *brute force* tidak praktis untuk jumlah titik kontrol yang besar, karena membutuhkan waktu komputasi yang signifikan.

Selain itu, mempertimbangkan aspek-aspek lain dari algoritma *brute force*, seperti fleksibilitasnya, algoritma ini memiliki kemampuan untuk menangani masalah dengan solusi yang tidak unik. Meskipun algoritma *brute force* mungkin tidak selalu memberikan solusi optimal, pendekatannya yang langsung dapat menjadi pilihan yang layak terutama untuk masalah dengan batasan waktu dan sumber daya yang terbatas.

Pada implementasinya, pendekatan *brute force* dalam membentuk kurva Bézier menerapkan rumus-rumus sebagai berikut. Kasus ketika kurva dikontrol oleh dua titik kontrol maka akan terbentuklah kurva Bézier linier dengan perumusan sebagai berikut.

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

Adapun ketika ditambahkan satu titik kontrol lagi, persamaan yang akan dihasilkan sebagai berikut,

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

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

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

yang apabila disubstitusikan akan menghasilkan,

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

Implementasi algoritma brute force dimulai dengan mencari koefisien binomial untuk perhitungan kurva Bézier. Fungsi `binomial_coefficient(n,k)` menghitung koefisien binomial berdasarkan pada rumus kombinasi, yang diperlukan untuk menentukan bobot setiap titik kontrol pada kurva Bézier. Ini merupakan langkah penting dalam algoritma brute force karena setiap titik kurva Bézier dihitung dengan menggunakan bobot yang sesuai.

Kemudian, kita melangkah ke penghitungan titik-titik kurva Bézier itu sendiri. Dalam definisi fungsi `bezier_curve_points(points, iterations)`, yang menerima daftar titik kontrol dan jumlah iterasi sebagai parameter input. Fungsi ini kemudian mengiterasi melalui semua nilai parameter t dalam rentang $[0, 1]$, dan pada setiap iterasi, titik kurva dihitung menggunakan rumus kurva Bézier yang sesuai. Hasilnya adalah daftar titik-titik yang menggambarkan kurva Bézier. Setelah semua titik kurva Bézier terkumpul, algoritma menggunakan matplotlib untuk memvisualisasikan hasilnya.

Dengan pendekatan brute force ini, kita dapat menggambar kurva Bézier dengan akurat menggunakan metode yang sederhana namun efektif. Ini memberikan pemahaman yang mendalam tentang cara kerja kurva Bézier dan algoritma brute force, serta menggambarkan kemampuan algoritma tersebut dalam menyelesaikan masalah geometris kompleks.

BAB II

Analisis dan Implementasi dalam Algoritma

Divide and Conquer

Pendekatan *divide and conquer* telah terbukti dapat menjadi strategi yang kuat dalam menangani masalah kompleks dengan cara yang terstruktur. Dalam pembuatan kurva Bézier, pendekatan ini memecah masalah menjadi bagian-bagian yang lebih kecil, menyelesaikan setiap bagian secara terpisah, dan kemudian menggabungkan solusi-solusi ini menjadi solusi untuk masalah aslinya. Kurva Bézier, sebuah kurva parametrik yang penting dalam bidang grafika komputer, desain komputer, dan animasi, dapat dibuat menggunakan pendekatan *divide and conquer* ini. Dengan membagi kurva menjadi segmen-segmen kecil, kompleksitas perhitungan dapat dikurangi, dan proses pembuatan kurva dapat dilakukan secara efisien.

Langkah pertama dalam implementasi algoritma *divide and conquer* adalah menentukan tiga titik kontrol awal untuk kurva Bézier, yaitu P0, P1, dan P2. Titik-titik ini menentukan bentuk umum dari kurva Bézier. Selanjutnya, sebuah parameter jumlah iterasi ditentukan untuk mengontrol seberapa banyak kurva Bézier akan dibagi menjadi segmen-segmen kecil. Fungsi `midpoint(point1, point2)` dibuat untuk menghitung titik tengah dari dua titik dimana pada kasus ini akan digunakan untuk memperhitungkan koordinat dari kedua titik kontrol dan mengembalikan titik tengahnya.

Fungsi inti dari algoritma *divide and conquer* adalah fungsi rekursif `draw_bezier_curve(P0, P1, P2, iterations, current_iteration)`. Fungsi ini mengambil tiga titik kontrol P0, P1, dan P2, jumlah iterasi, dan iterasi saat ini sebagai input. Dalam fungsi ini, kurva Bézier dibagi menjadi segmen-segmen kecil. Titik tengah dari setiap segmen garis kurva dihitung menggunakan fungsi `midpoint(point1, point2)`, dan kurva dibagi menjadi dua segmen yang lebih kecil. Fungsi `draw_bezier_curve(P0, P1, P2, iterations, current_iteration)` dipanggil secara rekursif untuk kedua segmen kurva yang lebih kecil, dan proses ini berlanjut hingga mencapai iterasi terakhir.

Jika iterasi saat ini sama dengan jumlah iterasi yang ditentukan, sebuah garis putus-putus digambar untuk memvisualisasikan pembagian kurva pada iterasi tersebut. Setelah langkah-langkah rekursif selesai dieksekusi, kurva Bézier lengkap ditampilkan dengan semua segmen-segmen kecil yang telah digabungkan kembali. Ini menunjukkan efektivitas algoritma *divide and conquer* dalam menangani masalah pembuatan kurva secara efisien, terstruktur, dan mudah dimengerti. Dengan demikian, pendekatan ini membawa manfaat signifikan dalam pengembangan aplikasi grafis dan desain komputer.

BAB III

Source Code Program

III. 1 *Algoritma Brute Force*

→ Algoritma berikut dapat digunakan untuk 3 hingga n buah titik kontrol.

```
import numpy as np
import matplotlib.pyplot as plt
import math
import timeit
from matplotlib.animation import FuncAnimation

# menghitung bobot tiap titik kontrol
def binomial_coefficient(n, k):
    if 0 <= k <= n:
        return math.factorial(n) // (math.factorial(k) * math.factorial(n
- k))
    else:
        return 0

# secara langsung menentukan titik dengan menghitung menggunakan rumus
kurva bezier
def bezier_curve_points(points, iterations):
    curve_points = []
    for t_value in range(iterations):
        t = t_value / (iterations - 1)
        curve_point = [0, 0]
        for point_index in range(len(points)):
            coefficient = binomial_coefficient(len(points) - 1,
point_index)
            bezier_term = coefficient * ((1 - t) ** (len(points) - 1 -
point_index)) * (t ** point_index)
            curve_point[0] += bezier_term * points[point_index][0]
            curve_point[1] += bezier_term * points[point_index][1]
        curve_points.append(curve_point)
    return np.array(curve_points)
```

```

# VISUALISASI
def update(frame, ax, points, curves, iterations, execution_time):
    ax.clear()
    ax.plot([point[0] for point in points], [point[1] for point in
points], 'ro-', label='Control Points')
    curve = curves[:frame+1] # Select curve points up to the current
iteration
    ax.plot(curve[:, 0], curve[:, 1], 'b-', label=f'Iteration {frame}')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_title(f'Bezier Curve Iteration {frame}')
    ax.legend()
    ax.axis('equal')

    execution_time_info = f'Execution Time: {execution_time:.2f} seconds'
    ax.text(0.95, 0.05, execution_time_info, transform=ax.transAxes,
ha='right', va='bottom')

# Input n titik
n = int(input("Masukkan jumlah titik yang hendak dimasukkan: "))
while n < 2:
    print("\nUntuk membuat kurva masukkan 2 atau lebih titik!")
    n = int(input("Masukkan jumlah titik yang hendak dimasukkan: "))

# Input koordinat poin-poin
x_start, y_start = map(float, input("Masukkan start point (x,y):
").split(","))
points = [(x_start, y_start)]

# control points
for i in range(n-2):
    x, y = map(float, input("Masukkan control point {}: (x,y):
".format(i+1)).split(","))
    points.append((x, y))

# end points
x_end, y_end = map(float, input("Masukkan end point (x,y): ").split(","))
points.append((x_end, y_end))

```



```

# banyak iterasi
iterations = int(input("Masukkan jumlah iterasi: "))

start_time = timeit.default_timer()
curves = bezier_curve_points(points, iterations+1)

fig, ax = plt.subplots()
execution_time = timeit.default_timer() - start_time

ani = FuncAnimation(fig, update, frames=iterations+1, fargs=(ax, points,
curves, iterations, execution_time), interval=150, repeat=False)
plt.show()

```

III. 2 Algoritma *Divide & Conquer*

→ Algoritma yang digunakan untuk 3 buah titik kontrol.

```

import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import time

# mencari mid point 2 titik
def midpoint(point1, point2):
    return ((point1[0] + point2[0]) / 2, (point1[1] + point2[1]) / 2)

# menggambar kurva bezier
def draw_bezier_curve(P0, P1, P2, iterations, current_iteration):
    if iterations == 0:
        plt.plot([P0[0], P1[0], P2[0]], [P0[1], P1[1], P2[1]], 'r-')
    else:
        Q0 = midpoint(P0, P1)
        Q1 = midpoint(P1, P2)
        R0 = midpoint(Q0, Q1)

        draw_bezier_curve(P0, Q0, R0, iterations - 1, current_iteration)
        draw_bezier_curve(R0, Q1, P2, iterations - 1, current_iteration)

    if iterations == current_iteration:
        plt.plot([P0[0], P1[0], P2[0]], [P0[1], P1[1], P2[1]], '--',
color='gray')

```

```

print("\n-- Kurva Bezier dengan 3 titik kontrol --")

# # titik awal dan akhir
# P0 = (0, 0)
# P1 = (4, 4)
# P2 = (8, 0)

# Input koordinat poin-poin
x_start, y_start = map(float, input("Masukan start point (x,y): ").split(","))
P0 = (x_start, y_start)

x_control, y_control = map(float, input("Masukan control point (x,y): ").split(","))
P1 = (x_control, y_control)

x_end, y_end = map(float, input("Masukan end point (x,y): ").split(","))
P2 = (x_end, y_end)

points = [P0, P1, P2]

# buat kurva animasi
def animate(iteration):

    # plot titik-titik awal
    plt.plot([point[0] for point in points], [point[1] for point in points], 'ro-', label='Control Points')
    ax.clear()
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_title(f'Beziér Curve Iteration {iteration+1}')
    ax.grid()
    ax.axis('equal')
    for i in range(1, iteration + 1):
        draw_bezier_curve(P0, P1, P2, iteration, i)
    ax.autoscale()

# tampilkan waktu eksekusi
execution_time = time.time() - start_time
ax.text(0.95, 0.05, f'Execution Time: {execution_time:.2f} seconds',

```

```

transform=ax.transAxes, ha='right', va='bottom', fontsize=10,
bbox=dict(facecolor='white', alpha=0.5))

# inisialisasi plot
fig, ax = plt.subplots()
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title('Beziér Curve')
ax.grid()
ax.axis('equal')

# buat animasi
iterations = int(input("Masukkan jumlah iterasi: "))
start_time = time.time()
ani = FuncAnimation(fig, animate, frames=iterations, interval=350,
repeat=False)

# tampilkan animasi
plt.draw()
plt.show()

```

→ Algoritma berikut dapat digunakan untuk 4 hingga n buah titik kontrol.

```

import time
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# untuk mencari titik tengah
def mid(point1, point2):
    return ((point1[0] + point2[0]) / 2, (point1[1] + point2[1]) / 2)

# secara rekursif mencari titik tengah dari list of points
def recursive_midpoint(points, next_points):
    if len(points) == 1:
        return points[0], next_points

```

```

else:
    midpoints = []
    for i in range(len(points) - 1):
        midpoints.append(mid(points[i], points[i+1]))
    if len(midpoints) == 1:
        to = len(next_points) // 2
        next_points.insert(to, midpoints[0])
    else:
        to = len(next_points) // 2
        next_points.insert(to, midpoints[-1])
        next_points.insert(to, midpoints[0])
    return recursive_midpoint(midpoints, next_points)

# mengambil titik-titik terluar untuk proses kurva selanjutnya
def make_the_next(points, next_points, middle):
    going = [points[0]] + next_points + [points[-1]]
    if len(going) % 2 == 0:
        to = len(going) // 2
        going.insert(to, middle)
    return going

# mengumpulkan titik-titik untuk menggambar kurva
def gather_graph_points(graph_points, middle_points):
    if len(middle_points) == 1 and len(graph_points) == 2:
        to = len(graph_points) // 2
        graph_points.insert(to, middle_points[0])
        return graph_points
    else:
        mid_graph = len(graph_points) // 2
        left_half_graph = graph_points[:mid_graph+1]
        right_half_graph = graph_points[mid_graph:]
        mid_middle = len(middle_points) // 2

```

```

        left_half_middle = middle_points[:mid_middle]
        right_half_middle = middle_points[mid_middle:]

        return gather_graph_points(left_half_graph,
left_half_middle[:-1] + gather_graph_points(right_half_graph,
right_half_middle)

# secara rekursif membagi list of points menjadi sesuai panjang 'length'
def divide_list_of_points(points, length):
    num_points = len(points)
    if num_points <= length:
        return [points]
    else:
        midpoint = num_points // 2
        left_half = points[:midpoint+1]
        right_half = points[midpoint:]
        return divide_list_of_points(left_half, length) +
divide_list_of_points(right_half, length)

# memproses kurva bezier sesuai iterasi yang dimasukkan dan menyimpan
titik-titik kurva untuk tiap iterasinya
def iterations(points, next_points, graph_points, iteration, i_now,
save_graphs):
    if iteration == 0:
        save_graphs.append(graph_points)
        return save_graphs
    elif len(next_points) <= len(points):
        new_mids = []
        new_midpoints, new_next_points = recursive_midpoint(points,
next_points)
        new_mids.append(new_midpoints)
        graph_points = gather_graph_points(graph_points, new_mids)
        save_graphs.append(graph_points)
        return iterations(points, new_next_points, graph_points,
iteration-1, i_now+1, save_graphs)

```

```

else:
    top = graph_points[len(graph_points)//2]
    next_points = make_the_next(points, next_points, top)
    divided_next_points = divide_list_of_points(next_points,
len(points))
    each_next = []
    new_mids = []
    for part in divided_next_points:
        new_midpoints, the_next = recursive_midpoint(part, [])
        new_mids.append(new_midpoints)
        for next in the_next:
            each_next.append(next)
    graph_points = gather_graph_points(graph_points, new_mids)
    save_graphs.append(graph_points)

    return iterations(points, each_next, graph_points, iteration-1,
i_now+1, save_graphs)

# Input n titik
n = int(input("Masukan jumlah titik yang hendak dimasukkan: "))
while (n < 4):
    print("\nUntuk membuat kurva masukan 4 atau lebih titik!")
    n = int(input("Masukan jumlah titik yang hendak dimasukkan: "))

# Input koordinat poin-poin
x_start, y_start = map(float, input("Masukan start point (x,y):
").split(","))
points = [(x_start, y_start)]

# Input control points
for i in range(n-2):
    x, y = map(float, input("Masukan control point {}: (x,y):
".format(i+1)).split(","))
    points.append((x, y))

```

```

x_end, y_end = map(float, input("Masukan end point (x,y): ").split(","))
points.append((x_end, y_end))

print("Iterasi maksimum ialah n+1! (dengan n jumlah titik yang
dimasukkan)")

i = int(input("Masukan jumlah iterasi: "))
while i > n+1:
    print("Iterasi maksimum ialah n+1! (dengan n jumlah titik yang
dimasukkan)")
    i = int(input("Masukan jumlah iterasi: "))

# Untuk penggunaan otomatis
# points = [(1.0, 0.0), (0.0, 3.0), (3.0, 6.0), (6.0, 6.0), (9.0, 3.0),
(8.0, 0.0)]
# points = [(0.0, 0.0), (2.0, 5.0), (6.0, 7.0), (10.0, 5.0), (12.0, 0.0)]

# Memulai program
start_time = time.perf_counter()

# Inisialisasi
next_points = []
graph_points = [points[0], points[-1]]
previous_curves = []
saved_graphs = []

# Mengambil titik-titik untuk membentuk kurva bezier
saved_graphs = iterations(points, next_points, graph_points, i, 0,
saved_graphs)

# Akhir program
end_time = time.perf_counter()

# Waktu eksekusi
time_execution = (end_time - start_time) * 1000

```

```

# VISUALISASI
fig, ax = plt.subplots()
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.grid()
ax.axis('equal')

# Untuk memperlihatkan kurva tiap iterasi
def animate(iteration, time_execution, control_points):
    ax.clear()
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_title(f'Bezier Curve {iteration} Iteration ')
    ax.grid()
    ax.axis('equal')

    # kurva yang bukan current curve akan berwarna abu garis-garis
    for i in range(iteration):
        points = saved_graphs[i]
        x_points, y_points = zip(*points)
        ax.plot(x_points, y_points, '--', color='gray', label='Curve
Before' if i == 0 else '')

    # kurva yang merupakan current curve akan berwarna biru
    points = saved_graphs[iteration]
    x_points, y_points = zip(*points)
    ax.plot(x_points, y_points, 'b-', label='Current Curve')
    ax.plot([point[0] for point in control_points], [point[1] for point
in control_points], 'ro-', label='Control Points')
    ax.legend(loc='lower right', fontsize=10)
    execution_time_info = f'Execution Time: {time_execution:.2f}
milliseconds'
    ax.text(0.5, 0.965, execution_time_info, transform=ax.transAxes,
ha='center', fontsize=10)

```



```

iterations = len(saved_graphs)
ani = FuncAnimation(fig, animate, frames=iterations,
fargs=(time_execution, points), interval=300, repeat=False)

plt.show()

```

III.3 GUI

→ Algoritma berikut dapat digunakan untuk 3 buah titik kontrol dengan visualisasi.

```

import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk
import os, sys
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import time

##### ALGORITMA #####
def midpoint(point1, point2):
    return ((point1[0] + point2[0]) / 2, (point1[1] + point2[1]) / 2)

def draw_bezier_curve(P0, P1, P2, iterations, current_iteration, ax):
    if iterations == 0:
        ax.plot([P0[0], P1[0], P2[0]], [P0[1], P1[1], P2[1]], 'b-')
    else:
        Q0 = midpoint(P0, P1)
        Q1 = midpoint(P1, P2)
        R0 = midpoint(Q0, Q1)

        draw_bezier_curve(P0, Q0, R0, iterations - 1, current_iteration, ax)
        draw_bezier_curve(R0, Q1, P2, iterations - 1, current_iteration, ax)

```

```

        if iterations == current_iteration:
            ax.plot([P0[0], P1[0], P2[0]], [P0[1], P1[1], P2[1]], '--',
                    color='grey')

def start_animation():
    global ani
    start_time = time.time()

    # Mengambil nilai input dari entry pada GUI
    update_point_text()
    x_start, y_start = float(start_x_entry.get()), float(start_y_entry.get())
    P0 = (x_start, y_start)

    x, y = float(control_x_entry.get()), float(control_y_entry.get())
    P1 = (x, y)

    x_end, y_end = float(end_x_entry.get()), float(end_y_entry.get())
    P2 = (x_end, y_end)

def animate(iteration):
    ax.clear()
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    current_time = time.time() - start_time
    ax.set_title(f'Bezier Curve Iteration {iteration} (Waktu eksekusi:
{current_time:.2f} s)')
    ax.grid()
    ax.axis('equal')

    # Plot titik-titik awal
    plt.plot(*P0, 'ro')
    plt.text(P0[0], P0[1], 'P0')
    plt.plot(*P1, 'ro')
    plt.text(P1[0], P1[1], 'P1')

```

```

plt.plot(*P2, 'ro')
plt.text(P2[0], P2[1], 'P2')

# Menggambar kurva bezier
for i in range(1, iteration + 1):
    draw_bezier_curve(P0, P1, P2, iteration, i, ax)

fig, ax = plt.subplots()
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title('Bezier Curve')
ax.grid()
ax.axis('equal')

iterations = int(iterations_entry.get())
ani = FuncAnimation(fig, animate, frames=iterations + 1, interval=350,
repeat=False)

canvas2.fig_photo = draw_figure(canvas2_animation, fig)

def draw_figure(canvas, figure):
    figure_canvas_agg = FigureCanvasTkAgg(figure, canvas)
    figure_canvas_agg.draw()
    figure_canvas_agg.get_tk_widget().pack(side='top', fill='both', expand=1)
    return figure_canvas_agg

##### BANTUAN FUNGSI UNTUK GUI #####

def resource_path(relative_path):
    base_path = getattr(sys, '_MEIPASS',
os.path.dirname(os.path.abspath(__file__)))
    return os.path.join(base_path, relative_path)

def resize_image(image_path, width, height):
    img = Image.open(image_path)

```

```

    img = img.resize((width, height))
    return ImageTk.PhotoImage(img)

# load gambar png untuk resize
def load_image_transparent(image_path, width, height):
    img = Image.open(image_path).resize((width, height), Image.NEAREST)
    return ImageTk.PhotoImage(img)

# switch notebook tab
def switch_tab(tab_index):
    notebook.select(tab_index)

def update_point_text():
    # fetch points
    x_start, y_start = start_x_entry.get(), start_y_entry.get()
    x_control, y_control = control_x_entry.get(), control_y_entry.get()
    x_end, y_end = end_x_entry.get(), end_y_entry.get()

    text_x = 668
    start_y = 300
    line_height = 35

    canvas2.create_text(text_x, start_y, text=f"{x_start}
{y_start}", fill="black", anchor="center")
    canvas2.create_text(text_x, start_y + line_height, text=f"{x_control}
{y_control}", fill="black", anchor="center")
    canvas2.create_text(text_x, start_y + 2 * line_height, text=f"{x_end}
{y_end}", fill="black", anchor="center")

def exit_program():
    root.quit() # quit tkinter application
    root.destroy() # destroy tkinter window
    sys.exit() # exit python

```

```

##### TKINTER GUI #####

root = tk.Tk()
root.title("Bezier Curve Maker")
root.geometry("850x768")

# create notebook
notebook = ttk.Notebook(root)
notebook.pack(fill='both', expand=True)

##### page 1 #####
page1 = ttk.Frame(notebook)
notebook.add(page1, text='Input')

# create canvas
canvas1 = tk.Canvas(page1, width=850, height=768)
canvas1.pack(fill="both", expand=True)

# background
background_image =
ImageTk.PhotoImage(file=resource_path("assets/page-1/Dashboard.png"))
canvas1.create_image(0, 0, image=background_image, anchor="nw")

# header
header_image =
load_image_transparent(resource_path("assets/page-1/header.png"), 920, 40)
canvas1.create_image(425, 28, image=header_image, anchor="center")

# logo
logo_image = load_image_transparent(resource_path("assets/page-1/logo.png"),
920, 254)
canvas1.create_image(425, 175, image=logo_image, anchor="center")

# control
control_image =
load_image_transparent(resource_path("assets/page-1/control.png"), 130, 35)

```

```

canvas1.create_image(78, 75, image=control_image, anchor="center")

# text
text_image = load_image_transparent(resource_path("assets/page-1/title.png"),
630, 50)
canvas1.create_image(410, 160, image=text_image, anchor="center")

canvas1.images = [background_image, header_image, logo_image, control_image,
text_image]

# frame untuk input table
entry_frame = tk.Frame(page1, bg='#dabecb', bd=5)
entry_frame.place(relx=0.5, rely=0.45, relwidth=0.85, relheight=0.5,
anchor='n')
entry_frame.grid_rowconfigure(0, minsize=40)

# input table
x_label = tk.Label(entry_frame, text="x", font=('Arial', 18), bg='#dabecb')
x_label.grid(row=1, column=1, padx=0, pady=0, sticky='nsew')
y_label = tk.Label(entry_frame, text="y", font=('Arial', 18), bg='#dabecb')
y_label.grid(row=1, column=2, padx=0, pady=0, sticky='nsew')

start_label = tk.Label(entry_frame, text="Start Point:", font=('Arial', 18),
bg='#dabecb')
start_label.grid(row=2, column=0, padx=5, pady=5, sticky='e')
start_x_entry = tk.Entry(entry_frame, font=('Arial', 18), borderwidth=0,
highlightthickness=0)
start_x_entry.grid(row=2, column=1, padx=(0, 5), pady=5, sticky='we')
start_y_entry = tk.Entry(entry_frame, font=('Arial', 18), borderwidth=0,
highlightthickness=0)
start_y_entry.grid(row=2, column=2, padx=5, pady=5, sticky='we')

control_label = tk.Label(entry_frame, text="Control Point:", font=('Arial',
18), bg='#dabecb')
control_label.grid(row=3, column=0, padx=5, pady=5, sticky='e')
control_x_entry = tk.Entry(entry_frame, font=('Arial', 18), borderwidth=0,

```

```

highlightthickness=0)

control_x_entry.grid(row=3, column=1, padx=(0, 5), pady=5, sticky='we')

control_y_entry = tk.Entry(entry_frame, font=('Arial', 18), borderwidth=0,
highlightthickness=0)

control_y_entry.grid(row=3, column=2, padx=5, pady=5, sticky='we')


end_label = tk.Label(entry_frame, text="End Point:", font=('Arial', 18),
bg='#dabecb')

end_label.grid(row=4, column=0, padx=5, pady=5, sticky='e')

end_x_entry = tk.Entry(entry_frame, font=('Arial', 18), borderwidth=0,
highlightthickness=0)

end_x_entry.grid(row=4, column=1, padx=(0, 5), pady=5, sticky='we')

end_y_entry = tk.Entry(entry_frame, font=('Arial', 18), borderwidth=0,
highlightthickness=0)

end_y_entry.grid(row=4, column=2, padx=5, pady=5, sticky='we')


iterations_label = tk.Label(entry_frame, text="Iteration:", font=('Arial',
18), bg='#dabecb')

iterations_label.grid(row=5, column=0, padx=5, pady=5, sticky='e')

iterations_entry = tk.Entry(entry_frame, font=('Arial', 18), borderwidth=0,
highlightthickness=0)

iterations_entry.grid(row=5, column=1, padx=(0, 5), pady=5, sticky='we')


# generate curve button
button_image_path = resource_path("assets/page-1/curve.png")
button_image = load_image_transparent(button_image_path, 180, 38)

# button ke page 2
start_button = tk.Button(entry_frame, image=button_image, command=lambda:
switch_tab(1), borderwidth=0, highlightthickness=0, relief='flat')

start_button.image = button_image # keep a reference to prevent
garbage-collection

start_button.place(relx=0.5, rely=0.75, anchor='center')


##### page 2 #####
page2 = ttk.Frame(notebook)

notebook.add(page2, text='Animation')

```

```
# create canvas
canvas2 = tk.Canvas(page2, width=850, height=768)
canvas2.pack(fill="both", expand=True)

# background
background_image2 =
ImageTk.PhotoImage(file=resource_path("assets/page-2/Dashboard.png"))
canvas2.create_image(0, 0, image=background_image2, anchor="nw")

# header
header_image2 =
load_image_transparent(resource_path("assets/page-2/header.png"), 920, 40)
canvas2.create_image(425, 28, image=header_image2, anchor="center")

# control
control_image2 =
load_image_transparent(resource_path("assets/page-2/control.png"), 130, 35)
canvas2.create_image(78, 75, image=control_image2, anchor="center")

# text
text_image2 =
load_image_transparent(resource_path("assets/page-2/title.png"), 400, 70)
canvas2.create_image(400, 110, image=text_image2, anchor="center")

# result
result_image2 =
load_image_transparent(resource_path("assets/page-2/result.png"), 710, 510)
canvas2.create_image(400, 410, image=result_image2, anchor="center")

# exit
exit_image2 = load_image_transparent(resource_path("assets/page-2/exit.png"),
60, 30)
canvas2.create_image(390, 680, image=exit_image2, anchor="center")

# exit button
exit_button = tk.Button(page2, image=exit_image2, command=exit_program,
```



```
borderwidth=0)

exit_button_window = canvas2.create_window(390, 680, anchor="center",
window=exit_button)

canvas2.images = [background_image2, header_image2, control_image2,
result_image2, exit_image2]

# canvas untuk si animasi di page 2
canvas2_animation = tk.Canvas(page2, width=700, height=500)
canvas2_animation.place(x=68, y=190, width=440, height=430)

def on_show_page2(event):
    if notebook.index("current") == 1:
        start_animation() # panggil fungsi ketika udah di page 2 aja

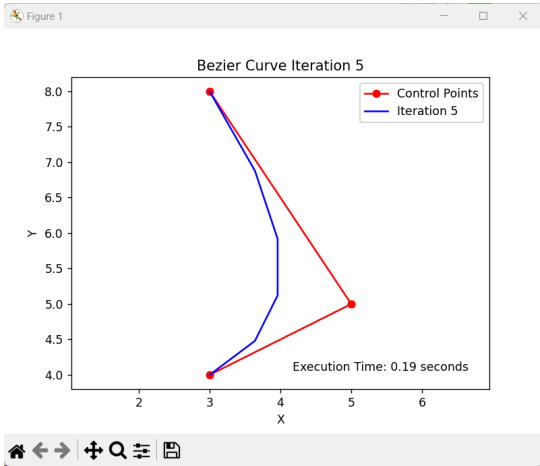
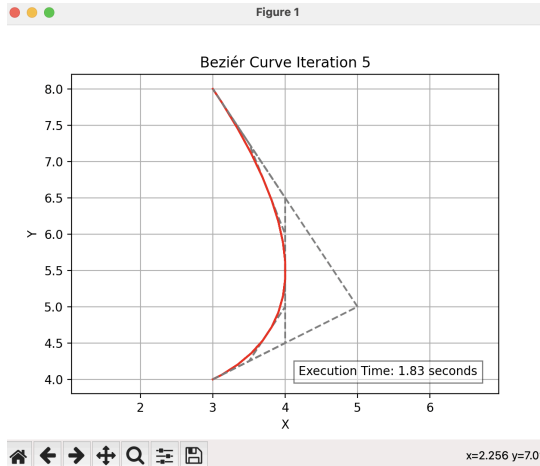
notebook.bind("<<NotebookTabChanged>>", on_show_page2)

root.mainloop()
```

BAB IV

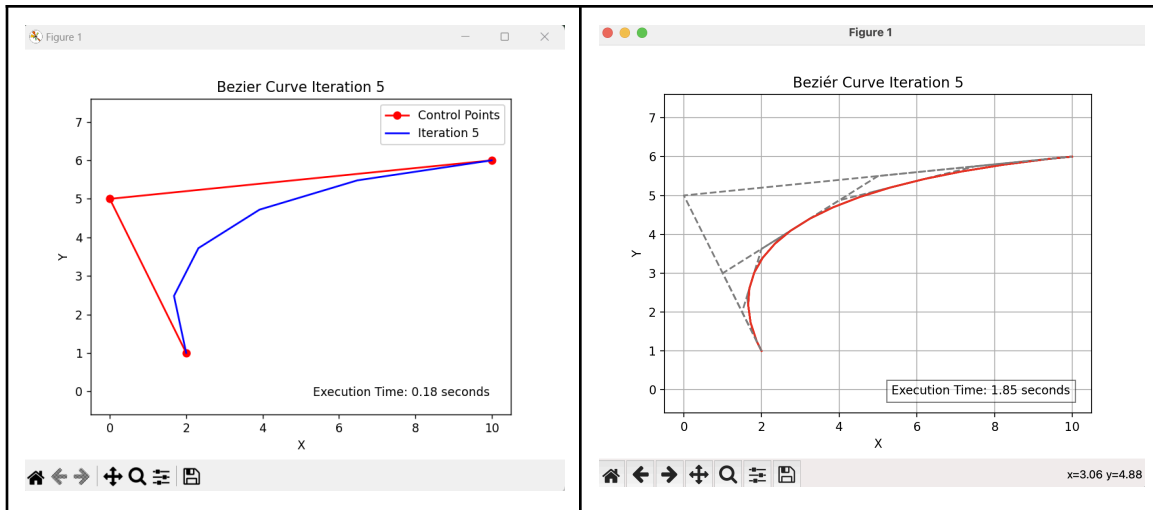
Tangkapan Layar

IV. 1 Test Case 1

Masukan	
Masukkan jumlah titik yang hendak dimasukkan: 3 Masukkan start point (x,y): 3,4 Masukkan control point 1: (x,y): 5,5 Masukkan end point (x,y): 3,8 Masukkan jumlah iterasi: 5	
Hasil	
Brute Force	Divide and Conquer
	

IV. 2 Test Case 2

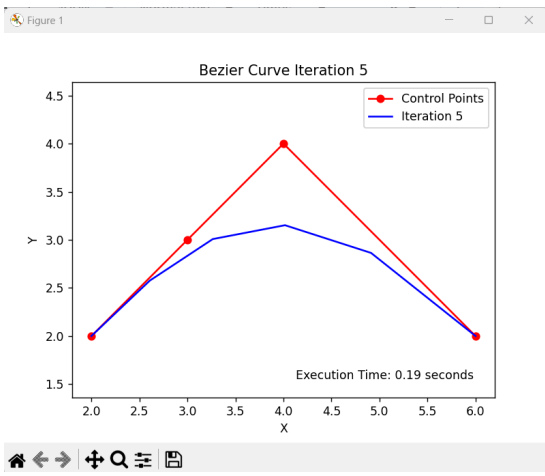
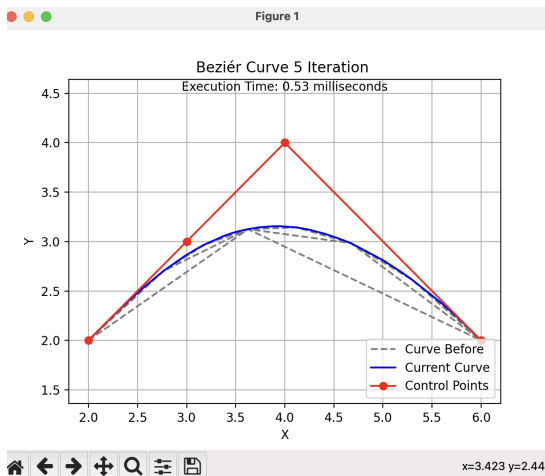
Masukan	
Masukkan jumlah titik yang hendak dimasukkan: 3 Masukkan start point (x,y): 2,1 Masukkan control point 1: (x,y): 0,5 Masukkan end point (x,y): 10,6 Masukkan jumlah iterasi: 5	
Hasil	
Brute Force	Divide and Conquer



IV. 3 Test Case 3

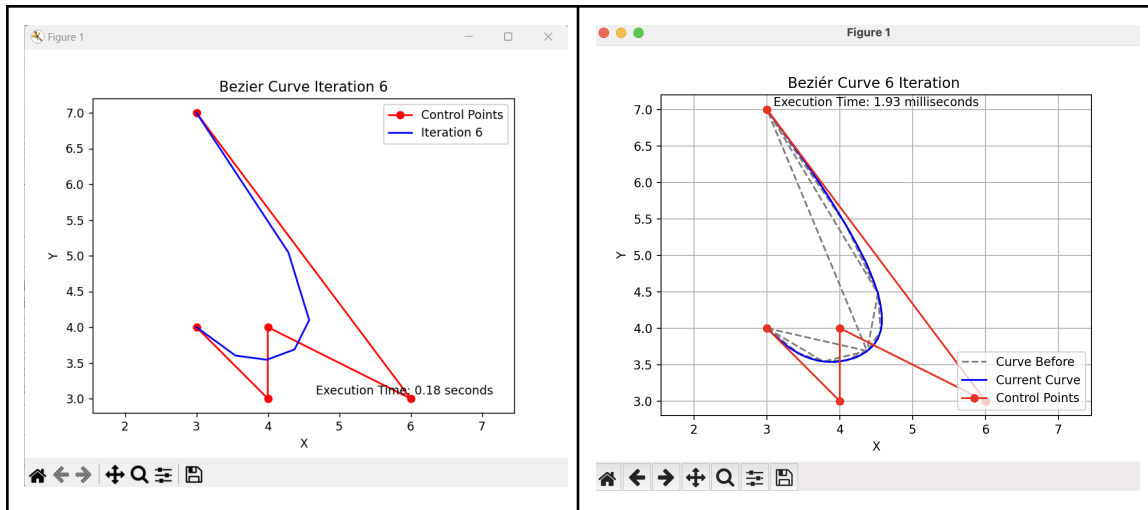
Masukan	
Masukkan jumlah titik yang hendak dimasukkan: 3 Masukkan start point (x,y): 3,5 Masukkan control point 1: (x,y): 1,3 Masukkan end point (x,y): 10,8 Masukkan jumlah iterasi: 8	
Hasil	
Brute Force	Divide and Conquer
<p>Figure 1</p> <p>Bezier Curve Iteration 8</p> <p>Execution Time: 0.43 seconds</p>	<p>Figure 1</p> <p>Beziér Curve Iteration 8</p> <p>Execution Time: 3.10 seconds</p> <p>x=3.015 y=9.081</p>

IV. 4 Test Case 4

Masukan	
Masukkan jumlah titik yang hendak dimasukkan: 4 Masukkan start point (x,y): 2,2 Masukkan control point 1: (x,y): 3,3 Masukkan control point 2: (x,y): 4,4 Masukkan end point (x,y): 6,2 Masukkan jumlah iterasi: 5	
Hasil	
Brute Force	Divide and Conquer
	

IV. 5 Test Case 5

Masukan	
Masukkan jumlah titik yang hendak dimasukkan: 5 Masukkan start point (x,y): 3,4 Masukkan control point 1: (x,y): 4,3 Masukkan control point 2: (x,y): 4,4 Masukkan control point 3: (x,y): 6,3 Masukkan end point (x,y): 3,7 Masukkan jumlah iterasi: 6	
Hasil	
Brute Force	Divide and Conquer



IV. 6 Test Case 6

Masukan	
Masukkan jumlah titik yang hendak dimasukkan: 6 Masukkan start point (x,y): 2,3 Masukkan control point 1: (x,y): 2,5 Masukkan control point 2: (x,y): 2,7 Masukkan control point 3: (x,y): 5,1 Masukkan control point 4: (x,y): 5,3 Masukkan end point (x,y): 5,5 Masukkan jumlah iterasi: 7	
Hasil	
Brute Force	Divide and Conquer

BAB V

Analisis Perbandingan Solusi *Brute Force* dan *Divide and Conquer*

Perbandingan antara solusi brute force dan solusi divide and conquer dalam konteks kurva Bezier ini sangat penting untuk dipertimbangkan dari segi efisiensi dan kompleksitas algoritma.

Solusi brute force memproses kurva Bezier dengan cara yang langsung menghitung titik-titik kurva pada setiap iterasi. Ini berarti untuk setiap nilai t dalam iterasi, perlu dilakukan perhitungan untuk setiap titik kontrol yang ada. Kompleksitas waktu solusi brute force ini secara kasar dapat dianggap sebagai $O(n * m)$, di mana n adalah jumlah iterasi dan m adalah jumlah titik kontrol. Meskipun sederhana dan mudah dipahami, solusi ini menjadi tidak efisien saat jumlah iterasi atau jumlah titik kontrol meningkat.

Di sisi lain, solusi divide and conquer membagi masalah menjadi submasalah yang lebih kecil dan lebih mudah dipecahkan. Dalam algoritma ini, setiap iterasi membagi kurva menjadi segmen-segmen kecil dengan menggunakan titik-titik tengah antara titik kontrol. Kemudian, titik-titik ini digunakan untuk menggambar kurva pada iterasi tersebut. Dengan menggunakan pendekatan ini, kompleksitas waktu algoritma divide and conquer secara kasar adalah $O(m * \log(n))$, di mana m adalah jumlah titik kontrol dan n adalah jumlah iterasi. Hal ini terjadi karena pada setiap iterasi, jumlah titik kontrol berkurang sekitar setengahnya. Meskipun solusi ini membutuhkan lebih banyak perhitungan pada setiap iterasi dibandingkan dengan solusi brute force, namun karena mengurangi jumlah titik kontrol secara eksponensial, algoritma ini menjadi lebih efisien ketika jumlah iterasi meningkat.

Secara keseluruhan, meskipun solusi brute force sederhana dan mudah dipahami, solusi divide and conquer menawarkan keseimbangan yang lebih baik antara efisiensi dan kompleksitas, terutama ketika jumlah titik kontrol meningkat. Solusi divide and conquer lebih disukai karena mampu menangani masalah dengan skala yang lebih besar dengan kompleksitas waktu yang lebih baik.

BAB VI

Implementasi Bonus

VI. 1 Generalisasi untuk N Titik Kontrol

Generalisasi untuk n titik kontrol dilakukan pada algoritma dengan pendekatan *brute force* maupun *divide and conquer*. Pembuatan algoritma *brute force* untuk n titik kontrol cenderung lebih mudah dibandingkan dengan algoritma *divide and conquer* untuk n titik kontrol karena pendekatan yang lebih langsung dan sederhana. Dalam algoritma *brute force*, langkah-langkah matematis untuk menghitung titik-titik kurva Bézier dapat diikuti secara langsung tanpa memerlukan banyak konseptualisasi tambahan. Setiap titik pada kurva dapat dihitung secara langsung dengan menggunakan rumus Bézier, membuatnya lebih intuitif untuk diimplementasikan. Selain itu, kode untuk algoritma *brute force* biasanya lebih sederhana dan lebih *direct* karena hanya memerlukan pengulangan sederhana untuk setiap titik pada setiap iterasi. Hal ini berbeda dengan algoritma *divide and conquer* yang melibatkan pemecahan masalah menjadi submasalah, menggunakan rekursi, dan mengelola titik-titik tengah yang memerlukan lebih banyak pemikiran dan perencanaan. Meskipun algoritma *brute force* memiliki kompleksitas yang lebih rendah dan risiko kesalahan yang lebih rendah, algoritma ini mungkin tidak efisien untuk masalah dengan skala yang lebih besar. Di sinilah kelebihan algoritma DNC muncul, karena meskipun memerlukan kerumitan dalam implementasi, dapat memberikan kinerja yang lebih baik dalam skala masalah yang lebih besar.

Generalisasi n titik kontrol dengan pendekatan *divide and conquer* ini dilakukan dengan melibatkan enam buah fungsi dengan penjelasan masing-masing, sebagai berikut.

1. `mid(point1, point2)`

Fungsi ini digunakan untuk mengambil titik tengah antara dua titik. Titik tengah ini diambil dengan rumus

$$\left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

2. ``recursive_midpoint(points, next_points)``

Fungsi ini akan secara rekursif mencari titik tengah dari serangkaian titik kontrol. Ini membagi setiap segmen antara dua titik kontrol dan memasukkan titik tengah ke dalam daftar ``next_points``.

3. ``make_the_next(points, next_points, middle)``

Fungsi ini digunakan untuk mengambil titik-titik terluar dan menambahkan titik tengah yang telah dihitung pada iterasi sebelumnya.

4. ``gather_graph_points(graph_points, middle_points)``

Fungsi digunakan untuk mengumpulkan titik-titik untuk menggambar kurva. Ini membagi daftar titik-titik menjadi dua bagian dan menyisipkan titik tengah di antara keduanya.

5. ``divide_list_of_points(points, length)``

Fungsi ini secara rekursif membagi daftar titik-titik menjadi sub-daftar dengan panjang tertentu.

6. ``iterations(points, next_points, graph_points, iteration, i_now, save_graphs)``

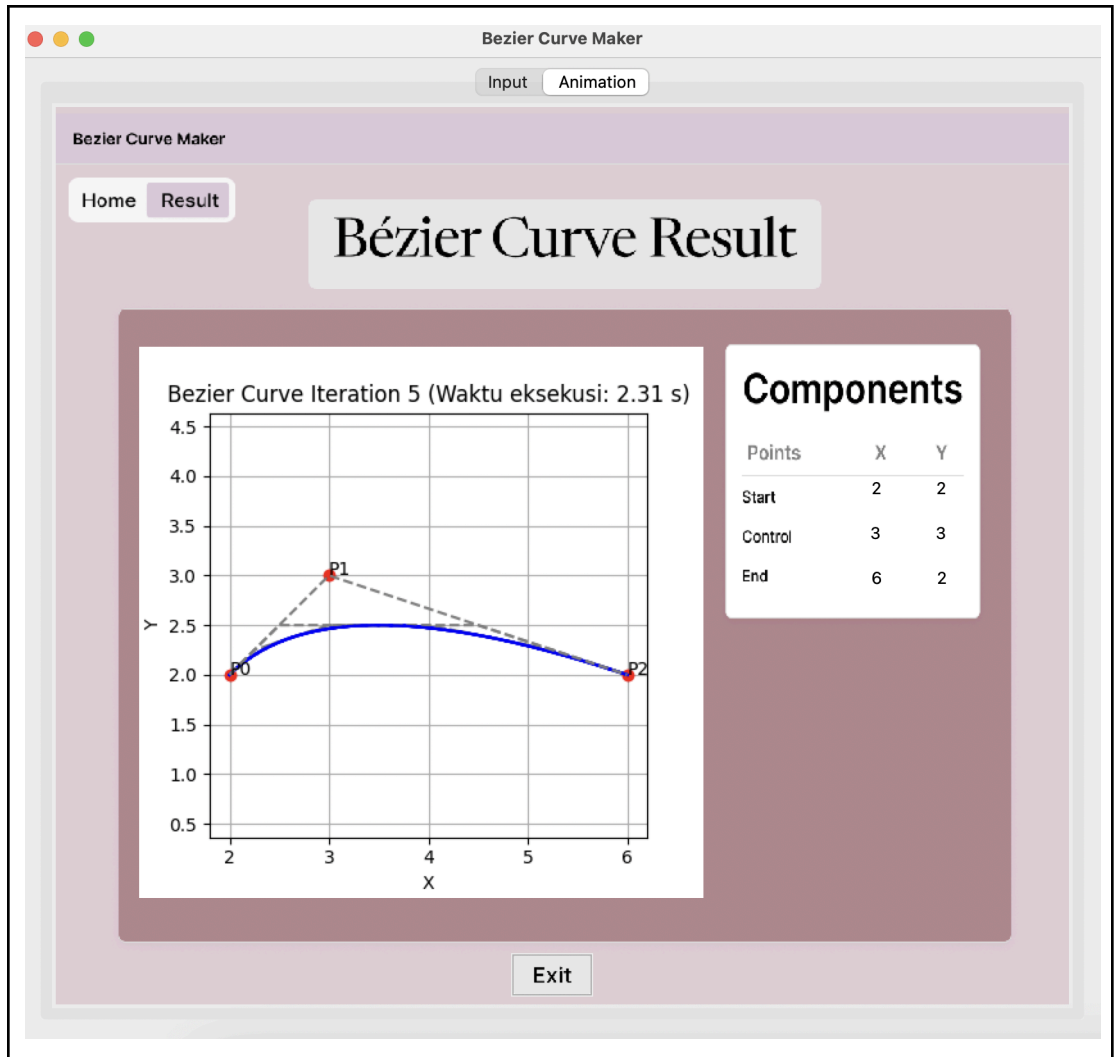
Fungsi ini memproses kurva Bézier sesuai dengan jumlah iterasi yang diberikan dan menghasilkan daftar titik-titik yang akan digunakan untuk menggambar kurva Bézier. Fungsi ini membagi daftar titik-titik menjadi segmen-segmen lebih kecil, mencari titik tengah untuk setiap segmen, dan mengumpulkan titik-titik untuk menggambar kurva. Proses ini berlanjut secara rekursif hingga mencapai iterasi terakhir.

Dengan menggunakan pendekatan *divide and conquer* seperti ini, kurva Bezier dapat dihasilkan dengan membagi masalah menjadi submasalah yang lebih kecil, menghitung solusi untuk setiap submasalah, dan menggabungkan solusi untuk mendapatkan kurva Bezier keseluruhan. Pendekatan ini memungkinkan pembuatan kurva Bezier dengan efisiensi yang lebih baik daripada pendekatan brute force, terutama ketika jumlah titik kontrol sangat besar. Akan tetapi pada saat ini, implementasi dibatasi hanya untuk pemakaian titik 4 hingga n dengan jumlah iterasi maksimal sebanyak $n + 1$ (dengan n jumlah titik masukan) untuk menghindari error pada kedalaman rekursif.

VI. 2 Visualisasi proses pembentukan kurva (GUI)


Pada GUI ini, digunakan algoritma *divide and conquer* untuk 3 buah titik (*points*), yang mana program akan meminta pengguna untuk memasukkan start point, control point, end point, serta jumlah iterasi yang diinginkan. Format masukannya adalah pengguna memasukkan poin dalam bentuk x dan y secara terpisah pada sel kolom yang berbeda. Program kemudian akan menghasilkan kurva animasi setelah pengguna menekan tombol “Generate Curve” pada halaman yang berbeda.

Masukan																
<div><div>Bezier Curve Maker</div><div>Input Animation</div><div><div>Bezier Curve Maker</div><div>Home Result</div><div><h1>Bézier Curve Maker</h1></div><div><table><thead><tr><th></th><th>x</th><th>y</th></tr></thead><tbody><tr><td>Start Point:</td><td>2</td><td>2</td></tr><tr><td>Control Point:</td><td>3</td><td>3</td></tr><tr><td>End Point:</td><td>6</td><td>2</td></tr><tr><td>Iteration:</td><td colspan="2">5</td></tr></tbody></table><div>Generate Curve</div></div></div></div>			x	y	Start Point:	2	2	Control Point:	3	3	End Point:	6	2	Iteration:	5	
	x	y														
Start Point:	2	2														
Control Point:	3	3														
End Point:	6	2														
Iteration:	5															
Hasil																



Lampiran

1. Pranala *Repository*

- Tautan Github *Repository* Tugas Kecil 2 dapat diakses pada https://github.com/hiirrs/Tucil2_13522013_13522085.
- Tautan dokumen laporan ini dapat diakses pada  Tucil2_13522013_13522085

2. 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	✓	