

Tugas Kecil 2 IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis

Divide and Conquer



Disusun oleh :

Erdianti Wiga Putri Andini (13522053)

Muhammad Neo Clcero Koda (13522108)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

DAFTAR ISI.....	1
BAB I	
DESKRIPSI MASALAH.....	3
1.1 Algoritma Divide and Conquer.....	3
1.2 Kurva Bézier.....	4
BAB II	
IMPLEMENTASI ALGORITMA DALAM BAHASA PYTHON.....	6
2.1 File bezier.py.....	6
2.2 File bruteforce.py.....	6
2.3 File dnc.py.....	7
2.4 File dnc_n.py.....	7
2.5 File main.py.....	8
2.6 File input.py.....	9
2.7 File util.py.....	9
2.8 File GUI.py.....	9
BAB III	
SOURCE CODE PROGRAM.....	12
3.1 Repository Program.....	12
3.2 Kurva Bézier.....	12
3.2.1 bezier.py.....	12
3.2.2 bruteforce.py.....	14
3.2.3 dnc.py.....	14
3.2.4 dnc_n.py.....	15
3.2.5 main.py.....	16
3.2.6 input.py.....	18
3.2.7 util.py.....	19
3.2.8 GUI.py.....	19
BAB IV	
ANALISIS ALGORITMA.....	26
4.1 Pembuatan Kurva Bézier Kuadratik dengan Algoritma Divide and Conquer.....	26
4.2 Pembuatan Kurva Bézier Kuadratik dengan Algoritma Brute Force.....	26
4.3 Pembuatan Kurva Bézier N Titik dengan Algoritma Divide and Conquer (Bonus).....	27
4.4 Analisis Perbandingan Solusi Brute Force dengan Divide And Conquer.....	28

4.4.1 Analisis Solusi Brute Force.....	28
4.4.2 Analisis Solusi Divide and Conquer.....	29
4.4.3 Hasil Perbandingan Solusi Brute Force dengan Divide And Conquer.....	30
BAB V	
MASUKAN DAN LUARAN PROGRAM.....	31
5.1 Input CLI.....	31
5.1.1 Test Case 1 (Brute Force).....	31
5.1.2 Test Case 2 (Brute Force).....	32
5.1.3 Test Case 3 (Brute Force).....	33
5.1.4 Test Case 4 (Brute Force).....	34
5.1.5 Test Case 5 (Brute Force).....	35
5.1.6 Test Case 6 (Brute Force).....	36
5.1.7 Test Case 7 (Divide and Conquer).....	37
5.1.8 Test Case 8 (Divide and Conquer).....	38
5.1.9 Test Case 9 (Divide and Conquer).....	38
5.1.10 Test Case 10 (Divide and Conquer).....	40
5.1.11 Test Case 11 (Divide and Conquer).....	41
5.1.12 Test Case 12 (Divide and Conquer).....	42
5.1.13 Test Case 13 (Invalid Input).....	44
5.2 Menggunakan GUI.....	45
5.2.1 Test Case 1.....	45
5.2.2 Test Case 2.....	47
5.2.3 Test Case 3.....	49
5.2.4 Test Case 4.....	51
5.2.5 Test Case 5.....	52
5.2.6 Test Case 6.....	53
5.2.7 Test Case 7.....	54
5.2.8 Test Case 8.....	55
LAMPIRAN.....	58
DAFTAR PUSTAKA.....	59

BAB I

DESKRIPSI MASALAH

1.1 Algoritma *Divide and Conquer*

Divide and Conquer adalah strategi pemecahan masalah yang melibatkan pembagian masalah kompleks menjadi bagian-bagian yang lebih kecil dan mudah dikelola, menyelesaikan setiap bagian secara terpisah, dan kemudian menggabungkan solusi-solusi tersebut untuk memecahkan masalah aslinya. Metode ini melibatkan tiga langkah utama. Pertama, persoalan utama dibagi menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan aslinya, namun ukurannya lebih kecil. Idealnya, setiap upa-persoalan memiliki ukuran yang hampir sama. Kedua, setiap upa-persoalan diselesaikan secara langsung jika sudah berukuran kecil, atau secara rekursif jika masih berukuran besar. Terakhir, solusi dari masing-masing upa-persoalan digabungkan kembali sehingga membentuk solusi dari persoalan utama. Metode ini berguna dalam menangani masalah kompleks dengan memecahkannya menjadi bagian-bagian yang lebih mudah diatasi, menyelesaikan masing-masing bagian, dan kemudian menggabungkan solusi-solusi tersebut untuk memperoleh solusi akhir.

Keunggulan dari algoritma *Divide and Conquer* terletak pada efisiensi penyelesaian masalah yang kompleks. Sebagai contoh, kompleksitas untuk perkalian dua matriks menggunakan metode biasa adalah $O(n^3)$, sedangkan menggunakan pendekatan divide and conquer seperti perkalian matriks Strassen memiliki kompleksitas $O(n^{2.8074})$. Keunggulan lainnya adalah kecocokan algoritma ini untuk sistem multiproses yang memungkinkan penerapan paralelisme untuk meningkatkan kinerja. Selain itu, algoritma ini menggunakan memori cache secara efisien, sehingga mengoptimalkan penggunaan sumber daya memori yang tersedia.

1.2 Kurva Bézier

Kurva Bézier adalah kurva parametrik (dengan parameter t bervariasi dari 0 hingga 1) yang ditentukan oleh serangkaian titik kontrol. Posisi titik-titik ini satu sama lainnya menentukan bentuk dari kurva tersebut. Kurva Bézier banyak digunakan dalam berbagai aplikasi seperti desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol yang menentukan bentuk dan arah kurva tersebut. Dalam pembuatan kurva Bézier, perlu menentukan titik-titik kontrol dan menghubungkannya dengan kurva. Kurva Bézier memiliki banyak manfaat dalam kehidupan sehari-hari, seperti dalam useran *pen tool*, pembuatan animasi yang halus dan realistik, desain produk yang kompleks dan presisi, serta pembuatan font yang unik dan menarik. Salah satu keunggulan utama dalam menggunakan kurva Bézier adalah kemampuannya untuk dengan mudah diubah dan dimanipulasi, sehingga memungkinkan untuk menciptakan desain yang presisi dan sesuai dengan kebutuhan.

Suatu kurva Bézier didefinisikan oleh kumpulan titik kontrol, dari P_0 sampai P_n , dimana n menunjukkan orde ($n = 1$ untuk linear, $n = 2$ untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva tersebut, sementara titik kontrol di antara keduanya, jika ada, umumnya tidak berada langsung pada kurva yang terbentuk. Sebagai contoh, dalam gambar 1, titik kontrol pertama adalah P_0 dan titik kontrol terakhir adalah P_3 . Titik kontrol di antara keduanya, yaitu P_1 dan P_2 , disebut sebagai titik kontrol antara yang tidak terletak pada kurva yang terbentuk.

Dalam pembentukan kurva Bézier, jika diberikan dua titik kontrol, misalnya P_0 dan P_1 , kurva Bézier yang terbentuk akan menjadi garis lurus antara kedua titik tersebut, yang dikenal sebagai kurva Bézier linier. Posisi pada kurva ini ditentukan

oleh persamaan parametrik, di mana t dalam fungsi tersebut menunjukkan seberapa jauh titik $B(t)$ dari P_0 ke P_1 . Ketika t bernilai 0.25 misalnya, titik $B(t)$ akan berada seperempat jalan dari titik P_0 ke P_1 , membentuk garis lurus di sepanjang rentang nilai t dari 0 hingga 1.

Dengan penambahan titik kontrol lainnya, seperti P_2 di antara P_0 dan P_1 , dan P_1 menjadi titik kontrol antara P_0 dan P_2 , kita dapat membentuk kurva Bézier kuadratik yang berbeda. Titik Q_1 terletak di antara garis yang menghubungkan P_1 dan P_2 , membentuk kurva Bézier linier baru dengan posisi Q_0 . Kemudian, kita dapat menentukan titik R_0 di antara garis yang menghubungkan Q_0 dan Q_1 , yang membentuk kurva Bézier kuadratik terhadap titik P_0 dan P_2 .

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

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

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

Proses ini dapat diterapkan untuk jumlah titik kontrol yang lebih dari tiga, menghasilkan kurva Bézier kubik jika terdapat empat titik, kurva Bézier kuartik jika terdapat lima titik, dan seterusnya. Persamaan untuk kurva-kurva ini dapat diturunkan menggunakan prosedur yang sama seperti sebelumnya, meskipun persamaan yang dihasilkan menjadi lebih panjang dan kompleks seiring bertambahnya jumlah titik kontrol.

BAB II

IMPLEMENTASI ALGORITMA DALAM BAHASA PYTHON

2.1 File bezier.py

File ini berisi program untuk memproses pembuatan kurva bezier dan menampilkan kurva bezier berdasarkan iterasinya.

Nama Fungsi	Deskripsi
kurva_bezier	Fungsi yang mengembalikan hasil kurva bezier dan total waktu eksekusi yang diperlukan.
show_kurva_bezier	Fungsi untuk menampilkan kurva bezier yang terbentuk dengan menggunakan animasi yang menyertakan warna berbeda untuk masing-masing iterasi.
update	Fungsi ini digunakan sebagai callback untuk setiap frame animasi. Fungsi ini memanggil ‘kurva_bezier’ untuk setiap iterasi, mengambil hasilnya, dan memplot kurva pada iterasi tersebut.

2.2 File bruteforce.py

File ini berisi program untuk memproses pembuatan kurva bezier dengan algoritma brute force.

Nama Fungsi	Deskripsi
bf_kurva	Fungsi yang mencari titik-titik kurva bezier dengan menggunakan algoritma <i>brute force</i> .

2.3 File dnc.py

File ini berisi program untuk memproses pembuatan kurva bezier 3 titik dengan algoritma *divide and conquer*.

Nama Fungsi	Deskripsi
dnc_kurva	Fungsi yang mencari titik-titik kurva bezier dengan menggunakan algoritma <i>divide and conquer</i> . Fungsi ini hanya bisa digunakan untuk input 3 titik (titik awal, satu titik kontrol, titik akhir).

2.4 File dnc_n.py

File ini berisi program untuk memproses pembuatan kurva bezier n titik dengan algoritma *divide and conquer*.

Nama Fungsi	Deskripsi
dnc_kurva_n	Fungsi yang mencari titik-titik kurva bezier dengan menggunakan algoritma <i>divide and conquer</i> . Fungsi ini bisa digunakan untuk input titik dengan

	jumlah berapapun.
--	-------------------

2.5 File main.py

File ini merupakan driver utama dari program ini, sehingga tidak terdapat fungsi di dalamnya, hanya berisi menu utama dari program ini. Hal-hal yang dilakukan dalam file ini hanyalah memanggil fungsi-fungsi dari file lain untuk dijalankan.

2.6 File input.py

File ini berisi program untuk input dari CLI.

Nama Fungsi	Deskripsi
input_points	Fungsi yang menerima input titik berupa <i>tuple</i> dan disimpan ke dalam array.
input_iteration_and_t	Fungsi yang menerima input banyaknya iterasi dan nilai t. Nilai t hanya diinput bila user memilih metode <i>brute force</i> .

2.7 File util.py

File ini berisi program umum, pada file ini hanya terdiri satu fungsi yaitu untuk mencari titik tengah di antara 2 titik.

Nama Fungsi	Deskripsi
mid_point	Fungsi yang mengembalikan titik tengah di antara 2 titik.

2.8 File GUI.py

File ini berisi program untuk user interface menggunakan tkinter.

Nama Fungsi	Deskripsi
create_input	Fungsi ini digunakan untuk membuat label dan input teks dalam frame.

save_curve	Fungsi ini digunakan untuk menyimpan hasil kurva dalam format pdf.
generate_bezier_curve	Fungsi ini dipanggil ketika tombol "Show the Bézier Curve" ditekan. Ini mengambil jumlah titik, titik kontrol, dan jumlah iterasi dari entri teks yang sesuai, kemudian memanggil fungsi 'show_kurva_bezier' pada bezier.py dengan parameter yang sesuai.
back_to_initial_display	Fungsi ini digunakan untuk kembali dari main display ke initial display.
main_display	Fungsi ini mengatur tata letak utama dari GUI. Fungsi menciptakan dan menempatkan label, entri teks, dan tombol pada frame utama.
create_entry_points	Fungsi ini menciptakan entri teks untuk setiap titik kontrol berdasarkan jumlah titik yang dimasukkan user. Jika jumlah titik adalah 2, hanya entri teks untuk titik awal dan akhir yang dibuat. Jika lebih dari 2, entri teks juga dibuat untuk titik kontrol di antara titik awal dan akhir. Fungsi ini diikat ke peristiwa "FocusOut" pada entri teks jumlah titik

	untuk memastikan bahwa entri teks titik kontrol diperbarui setelah user memasukkan jumlah titik.
root,mainloop	Fungsi ini menjaga jendela GUI tetap terbuka dan merespon user seperti klik tombol dan masukan keyboard.

BAB III

SOURCE CODE PROGRAM

3.1 Repository Program

Berikut adalah pranala ke *repository* program:

https://github.com/wigaandini/Tucil2_13522053_13522108

3.2 Kurva Bézier

3.2.1 bezier.py

```
from bruteforce import bf_kurva
from dnc import dnc_kurva
from dnc_n import dnc_kurva_n
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib.animation import FuncAnimation

def kurva_bezier(points, i, t, dnc):
    start_time = time.time()
    if (dnc):
        titik_kurva, titik_tengah = dnc_kurva_n(points, i)
    else:
        titik_kurva = bf_kurva(points, i, t)
    end_time = time.time()
    waktu_eksekusi = (end_time - start_time) * 1000
    return titik_kurva, waktu_eksekusi

# with animation
def show_kurva_bezier(points, iterations, t, dnc):
    def update(frame):
        plt.cla()
        colors = plt.cm.viridis(np.linspace(0, 1, frame + 1))
```

```

for i in range(frame + 1):
    titik_kurva, waktu_eksekusi = kurva_bezier(points, i, t, dnc)
    x_kurva = [titik[0] for titik in titik_kurva]
    y_kurva = [titik[1] for titik in titik_kurva]

    if i == frame:
        plt.plot(x_kurva, y_kurva, color='red', label=f"Iterasi ke-{i} ({waktu_eksekusi:.2f} ms)")
    else:
        plt.plot(x_kurva, y_kurva, color=colors[i % len(colors)], label=f"Iterasi ke-{i} ({waktu_eksekusi:.2f} ms)")

    if len(titik_kurva) <= 1025:
        for point in titik_kurva:
            plt.scatter(point[0], point[1], color='#a83d57', zorder=5)

    plt.scatter([point[0] for point in points], [point[1] for point in points], color='red', label="Control Points")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Kurva Bézier')
    plt.legend()
    plt.grid(True)
    plt.axis('equal')

fig = plt.figure(num='Kurva Bézier Result', figsize=(8, 6))
update(iterations)
ani = FuncAnimation(fig, update, frames=iterations + 1, interval=1000, repeat=False)
plt.show()

```

3.2.2 bruteforce.py

```
from util import mid_point

def bf_kurva(points, i, t):
    result_points = [points[0], points[-1]]
    temp_result = result_points.copy()
    intermediate_points = points[1:-1]

    for j in range(i):
        temp_intermediate = []
        while (len(intermediate_points) != 0):
            temp_intermediate.append(mid_point(temp_result[0],
intermediate_points[0], t))
            if (len(temp_intermediate) % 2 == 1):
                temp_result.pop(0)
            else:
                intermediate_points.pop(0)
        intermediate_points = temp_intermediate.copy()

        temp_result = []
        temp_result.append(result_points[0])
        for l in range(len(intermediate_points) - 1):
            temp_result.append(mid_point(intermediate_points[l],
intermediate_points[l + 1], t))
        temp_result.append(result_points[-1])
        result_points = temp_result.copy()

    return result_points
```

3.2.3 dnc.py

```
from util import mid_point

def dnc_kurva(points, i):
    titik_tengah = []
```

```

if i == 0:
    return [points[0], points[-1]], titik_tengah

q0 = mid_point(points[0], points[1], 0.5)
q1 = mid_point(points[1], points[2], 0.5)
r0 = mid_point(q0, q1, 0.5)

kurva_kiri, tengah_kiri = dnc_kurva([points[0], q0, r0], i - 1)
kurva_kanan, tengah_kanan = dnc_kurva([r0, q1, points[2]], i - 1)

titik_tengah.extend(tengah_kiri)
titik_tengah.append(r0)
titik_tengah.extend(tengah_kanan)

return kurva_kiri + kurva_kanan[1:], titik_tengah

```

3.2.4 dnc_n.py

```

from util import mid_point

def dnc_kurva_n(points, i):
    titik_tengah = []
    if i == 0:
        return [points[0], points[-1]], titik_tengah

    intermediate_points = points.copy()
    left_points = [points[0]]
    right_points = [points[-1]]
    temp = []
    for j in range(1, len(points)):
        for k in range(len(intermediate_points) - 1):
            temp.append(mid_point(intermediate_points[k],
intermediate_points[k + 1], 0.5))
        intermediate_points = temp.copy()
        left_points.append(intermediate_points[0])
        right_points.insert(0, intermediate_points[-1])

```

```
temp = []

kurva_kiri, tengah_kiri = dnc_kurva_n(left_points, i - 1)
kurva_kanan, tengah_kanan = dnc_kurva_n(right_points, i - 1)

titik_tengah.extend(tengah_kiri)
titik_tengah.extend(intermediate_points)
titik_tengah.extend(tengah_kanan)

return kurva_kiri + kurva_kanan[1:], titik_tengah
```

3.2.5 main.py

```

print("13522053 - 13522108")
print("-----\n")

while True:
    print("Choose algorithm to use ")
    print("1. Brute Force")
    print("2. Divide and Conquer")
    choice = input("Choice: ")
    if choice.isdigit():
        choice = int(choice)
        if choice in [1, 2]:
            break
    print("Invalid input. Please input 1 or 2.\n")

points = input_points(choice)
i, t = input_iteration_and_t(choice)

if choice == 1:
    # bruteforce
    titik_kurva, waktu_eksekusi = kurva_bezier(points, i, t, False)
    print("Execution Time:", waktu_eksekusi, "ms")
    if (i <= 7) :
        print("Final Points Bézier Curve", titik_kurva)
    else :
        print("Total Points Bézier Curve:", len(titik_kurva), "points")
        show_kurva_bezier(points, i, t, False)
else :
    # dnc
    titik_kurva, waktu_eksekusi = kurva_bezier(points, i, t, True)
    print("Execution Time:", waktu_eksekusi, "ms")
    if (i <= 7) :
        print("Final Points Bézier Curve", titik_kurva)
    else :
        print("Total Points Bézier Curve:", len(titik_kurva), "points")
        show_kurva_bezier(points, i, t, True)

```

3.2.6 input.py

```
def input_points(choice):
    try:
        n = int(input("Number of points (minimum 2): "))
        if choice == 1:
            if n not in [2, 3]:
                raise ValueError("Number of points max 3 for this
method.")
        else:
            if n < 2:
                raise ValueError("Number of points must be at least 2.")

        points = []
        for i in range(n):
            if i == 0:
                point = tuple(map(float, input(f"Start point coordinate
(separate with space): ").split())))
            elif i == n - 1:
                point = tuple(map(float, input(f"Final point coordinate
(separate with space): ").split())))
            else:
                point = tuple(map(float, input(f"Coordinate of control
point {i} (separate with space): ").split()))

            if len(point) != 2:
                raise ValueError("Invalid point input.")
            points.append(point)

    return points
except ValueError as ve:
    print("Invalid input:", ve)
    raise SystemExit
except:
    print("Something went wrong.")
    raise SystemExit
```

```

def input_iteration_and_t(choice):
    try:
        i = int(input("Number of iteration: "))
        if (i < 0):
            raise ValueError("Invalid iteration count. ")

        if choice == 1:
            t = float(input("t value (0 <= t <= 1): "))
            if (t < 0 or t > 1):
                raise ValueError("Invalid t value.")
        else :
            t = 0.5
    return i, t
    except ValueError as ve:
        print("Invalid input.", ve)
        raise SystemExit
    except:
        print("Something went wrong.")
        raise SystemExit

```

3.2.7 util.py

```

def mid_point(p1, p2, t):
    mid = ((1 - t) * p1[0] + t * p2[0], (1 - t) * p1[1] + t * p2[1])
    return mid

```

3.2.8 GUI.py

```

import tkinter as tk
from tkinter import messagebox, filedialog
import matplotlib.pyplot as plt
from PIL import Image, ImageTk
from bezier import *

```

```

from matplotlib.backends.backend_pdf import PdfPages

entry_points = None
entry_iterations = None
input_points = []
waktu_eksekusi_label = None
save_button = None


def create_input(frame, label, row, column):
    label = tk.Label(frame, text=label, bg="#f9e9f3", anchor='center',
justify='center')
    label.grid(row=row, column=column, padx=5, pady=5, sticky='w')
    entry = tk.Entry(frame, justify='center')
    entry.grid(row=row, column=column+1, sticky='e')
    return entry


def save_curve(waktu_eksekusi_label, points, iterations, t, dnc):
    try:
        filename = filedialog.asksaveasfilename(defaultextension=".pdf",
filetypes=[("PDF files", "*.pdf")])
        if filename:
            with open(filename, "a") as file:
                file.write(f"\n\nFinal Execution Time:\n{waktu_eksekusi_label['text']}")

            plt.figure()
            colors = plt.cm.viridis(np.linspace(0, 1, iterations + 1))
            for i in range(iterations + 1):
                titik_kurva, waktu_eksekusi = kurva_bezier(points, i, t,
dnc)
                x_kurva = [titik[0] for titik in titik_kurva]
                y_kurva = [titik[1] for titik in titik_kurva]

                if i == iterations:

```

```

                plt.plot(x_kurva, y_kurva, color='red',
label=f"Iteration {i} ({waktu_eksekusi:.2f} ms)")
            else:
                plt.plot(x_kurva, y_kurva, color=colors[i %
len(colors)], label=f"Iteration {i} ({waktu_eksekusi:.2f} ms)")

        for point in titik_kurva:
            plt.scatter(point[0], point[1], color='#a83d57',
zorder=5)

        plt.scatter([point[0] for point in points], [point[1] for
point in points], color='red', label="Control Points")
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.title('Bézier Curve Result')
        plt.legend()
        plt.grid(True)

    with PdfPages(filename) as pdf:
        pdf.savefig()
    plt.close()

    messagebox.showinfo("Success", "Bézier Curve saved
successfully.")
except Exception as e:
    messagebox.showerror("Error", f"An error occurred while saving:
{str(e)}")

def generate_bezier_curve():
    global waktu_eksekusi_label, save_button
    try:
        n = int(entry_points.get())
        if n < 2:
            raise ValueError("Number of points must be at least 2.")

```

```

        points = []
        for i in range(n):
            point_str = input_points[i].get().strip()
            if not point_str:
                raise ValueError(f"Point {i+1} is empty.")
            point = tuple(map(float, point_str.split()))
            if len(point) != 2:
                raise ValueError(f"Invalid format for point {i+1}. Points
should be separated by space.")
            points.append(point)

        iterations = int(entry_iterations.get())
        if iterations < 0:
            raise ValueError("Number of iterations must be at least 0.")

        titik_kurva, waktu_eksekusi = kurva_bezier(points, iterations,
0.5, True)

        waktu_eksekusi_label.config(text=f"Execution Time:
{waktu_eksekusi:.2f} ms")

        show_kurva_bezier(points, iterations, 0.5, dnc=True)

        save_button = tk.Button(frame, text="Save Curve", command=lambda:
save_curve(waktu_eksekusi_label, points, iterations, 0.5, True),
width=20, height=2, bg="#cc98aa")
        save_button.grid(row=102, column=0, columnspan=2, pady=5)

    except ValueError as e:
        messagebox.showerror("Error", str(e))
    except Exception as e:
        messagebox.showerror("Error", str(e))

def back_to_initial_display():
    global background_label, start_button, back_button

```

```

label_title.destroy()
frame.destroy()
entry_points.destroy()
entry_iterations.destroy()
show_button.destroy()
back_button.destroy()

background_label = tk.Label(root, image=background_photo)
background_label.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
start_button = tk.Button(root, text="Let's Make Bézier Curve!",
command=main_display, width=20, height=2, bg="#cc98aa")
start_button.place(relx=0.5, rely=0.75, anchor=tk.CENTER)

def main_display():
    global entry_points, entry_iterations, input_points, label_title,
frame, show_button, back_button, waktu_eksekusi_label

    background_label.destroy()
    start_button.destroy()

    root.configure(background="#f9e9f3")

    label_title = tk.Label(root, text="Bézier Curve Generator",
font=(16), pady=20, padx=20, bg="#f9e9f3", anchor='center',
justify='center')
    label_title.pack()

    frame = tk.Frame(root, padx=20, pady=20, bg="#f9e9f3")
    frame.pack()

    label_points = tk.Label(frame, text="Number of points (minimum 2):",
bg="#f9e9f3", anchor='center', justify='center')
    label_points.grid(row=0, column=0)

```

```

entry_points = tk.Entry(frame, justify='center')
entry_points.grid(row=0, column=1)

input_points.clear()

def create_entry_points():
    try:
        num_points = int(entry_points.get())
        if num_points < 2:
            messagebox.showerror("Error", "Number of points must be at least 2.")
        return
        for widget in input_points:
            widget.destroy()
        input_points.clear()
        for i in range(num_points):
            if i == 0:
                entry = create_input(frame, "Start Point:", i + 1, 0)
            elif i == num_points - 1:
                entry = create_input(frame, "Final Point:", i + 1, 0)
            else:
                entry = create_input(frame, f"Control Point {i}:", i + 1, 0)
            input_points.append(entry)
    except ValueError:
        messagebox.showerror("Error", "Invalid input for number of points.")

entry_points.bind("<FocusOut>", lambda event: create_entry_points())

entry_iterations = create_input(frame, "Number of Iteration:", 20, 0)

show_button = tk.Button(frame, text="Show the Bézier Curve",
command=generate_bezier_curve, width=20, height=2, bg="#cc98aa")
show_button.grid(row=100, column=0, columnspan=2, pady=10)

```

```
waktu_eksekusi_label = tk.Label(frame, text="", bg="#f9e9f3")
waktu_eksekusi_label.grid(row=101, column=0, columnspan=2, pady=5)

back_button = tk.Button(root, text="\u2190", font=('Arial', 12),
command=back_to_initial_display, width=3, height=1, bg="#cc98aa")
back_button.place(relx=0, rely=0, anchor=tk.NW)

# Initial display
root = tk.Tk()
root.title("B\u00e9zier Curve Generator")
background_image = Image.open("assets/background.png")
background_photo = ImageTk.PhotoImage(background_image)
background_label = tk.Label(root, image=background_photo)
background_label.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
root.geometry("1280x720")

# Main display
start_button = tk.Button(root, text="Let's Make B\u00e9zier Curve!",
command=main_display, width=20, height=2, bg="#cc98aa")
start_button.place(relx=0.5, rely=0.75, anchor=tk.CENTER)

root.mainloop()
```

BAB IV

ANALISIS ALGORITMA

4.1 Pembuatan Kurva Bézier Kuadratik dengan Algoritma *Divide and Conquer*

Pembuatan kurva Bézier kuadratik dengan algoritma *Divide and Conquer* dilakukan secara rekursif. Berikut adalah langkah-langkah untuk menentukan titik-titik pada kurva dengan tiga titik kontrol (misalkan p_0 , p_1 , dan p_2) dan jumlah iterasi sebanyak i :

1. Jika iterasi yang dicari merupakan iterasi ke-0 atau hanya terdapat dua titik kontrol, titik-titik yang dihasilkan pada kurva Bézier adalah p_0 dan p_2 .
2. Untuk iterasi pertama, tentukan titik antara q_0 , yaitu titik tengah dari p_0 dan p_1 dan tentukan titik antara q_1 , yaitu titik tengah dari p_1 dan p_2 .
3. Tentukan r_0 , yaitu titik tengah dari q_0 dan q_1 .
4. Tentukan titik-titik yang dihasilkan oleh kurva dengan titik kontrol berupa p_0 , q_0 , dan r_0 serta iterasi sebanyak $(i - 1)$. Titik-titik tersebut akan membentuk bagian kiri kurva.
5. Tentukan titik-titik yang dihasilkan oleh kurva dengan titik kontrol berupa r_0 , q_1 , dan p_2 serta iterasi sebanyak $(i - 1)$. Titik-titik tersebut akan membentuk bagian kanan kurva.
6. Gabungan titik-titik yang dihasilkan pada langkah 4 dan 5 merupakan kurva Bézier yang dihasilkan.

4.2 Pembuatan Kurva Bézier Kuadratik dengan Algoritma *Brute Force*

Pembuatan kurva Bézier kuadratik dengan algoritma *Brute Force* dilakukan secara iteratif. Berikut adalah langkah-langkah untuk menentukan titik-titik pada kurva dengan tiga titik kontrol dan jumlah iterasi sebanyak i :

1. Jika iterasi yang dicari merupakan iterasi ke-0, titik-titik yang dihasilkan pada kurva Bézier adalah titik pertama dan titik terakhir.
2. Untuk iterasi pertama, tentukan daftar titik antara dengan menentukan titik tengah dari semua pasangan titik kontrol yang berurutan (titik tengah dari titik kontrol ke- j dengan titik kontrol ke- $(j + 1)$).
3. Tentukan titik-titik tengah dari daftar titik antara yang dihasilkan pada langkah 2.
4. Urutan titik-titik hasil pada kurva yang dihasilkan pada iterasi pertama adalah titik kontrol pertama, hasil titik-titik dari langkah 3, dan titik kontrol terakhir.
5. Untuk iterasi selanjutnya, tentukan daftar titik antara dengan menentukan titik tengah dari daftar titik antara iterasi sebelumnya dan titik kurva iterasi sebelumnya.
6. Tentukan titik-titik tengah dari daftar titik antara yang dihasilkan pada langkah 5.
7. Urutan titik-titik hasil pada kurva yang dihasilkan adalah titik kontrol pertama, hasil titik-titik dari langkah 6, dan titik kontrol terakhir.
8. Ulangi langkah 5 hingga 7 sebanyak $(i - 1)$ kali untuk mendapatkan titik-titik kurva Bézier ke- i .

4.3 Pembuatan Kurva Bézier N Titik dengan Algoritma *Divide and Conquer* (Bonus)

Pembuatan kurva Bézier n titik dengan algoritma *Divide and Conquer* dilakukan secara rekursif. Ide dari pembentukan kurva hampir sama dengan pembentukan kurva bázier kuadratik, namun terdapat generalisasi dalam penentuan titik tengah dan pembagian kurva menjadi bagian titik kiri dan bagian titik kanan. Berikut

adalah langkah-langkah untuk menentukan titik-titik pada kurva dengan n titik kontrol dan jumlah iterasi sebanyak i :

1. Jika iterasi yang dicari merupakan iterasi ke-0 atau hanya terdapat dua titik kontrol, titik-titik yang dihasilkan pada kurva Bézier adalah titik pertama dan titik terakhir.
2. Jika terdapat tiga titik kontrol atau lebih, maka akan ditentukan daftar titik kiri dan daftar titik kanan dari kurva tersebut. Daftar titik kiri awalnya berisi titik kontrol pertama dan daftar titik kanan awalnya berisi titik kontrol terakhir.
3. Tentukan titik tengah dari semua pasangan titik kontrol yang berurutan (titik tengah dari titik kontrol ke- j dengan titik kontrol ke- $(j + 1)$). Titik tengah yang dihasilkan akan berjumlah $(n - 1)$. Tambahkan titik tengah pertama yang dihasilkan ke daftar titik kiri dan titik tengah terakhir yang dihasilkan ke daftar titik kanan.
4. Ulangi pencarian titik tengah dari titik-titik yang dihasilkan pada langkah 3 hingga menghasilkan satu titik.
5. Tentukan kurva Bézier yang dibuat oleh iterasi ke- $(i - 1)$ dari daftar titik kiri dan daftar titik kanan.
6. Gabungan titik-titik yang dihasilkan pada langkah 5 merupakan kurva Bézier yang dihasilkan.

4.4 Analisis Perbandingan Solusi Brute Force dengan Divide And Conquer

4.4.1 Analisis Solusi Brute Force

Algoritma *brute force* yang diimplementasikan dapat dilihat pada bagian 3.2.2. Algoritma tersebut bersifat iteratif dan tiap iterasi memiliki dua tahap, yaitu penentuan titik antara dan penentuan titik kurva pada iterasi tersebut.

Pada tahap penentuan titik antara, titik-titik antara didapatkan dari titik kurva dan titik antara pada solusi sebelumnya. Sebagai contoh, iterasi ke-0 memiliki dua titik kurva dan satu titik antara, yaitu titik kontrol. Iterasi ke-1 memiliki tiga titik kurva dan dua titik antara. Iterasi ke-2 memiliki lima titik kurva dan empat titik antara. Iterasi ke-3 memiliki sembilan titik kurva dan delapan titik antara. Terlihat bahwa ada pola berupa penggandaan jumlah titik antara pada tiap iterasi. Tahap penentuan titik antara akan memanggil fungsi *mid_point* sebanyak 2^n kali, dengan n merupakan jumlah iterasi yang dilakukan.

Pada tahap penentuan titik kurva, titik-titik didapatkan dari titik antara yang dihasilkan pada iterasi tersebut serta titik awal dan akhir pada titik kontrol. Karena terdapat 2^n titik antara pada suatu iterasi, akan terdapat $2^n - 1$ pemanggilan fungsi *mid_point* dan $2^n + 1$ titik kurva pada suatu iterasi.

Berdasarkan jumlah pemanggilan fungsi *mid_point*, kompleksitas waktu algoritma *brute force* adalah:

$$T(n) = 0, \text{ untuk } n = 0$$

$$T(n) = \sum_{i=1}^n (2^n + 2^n - 1)$$

Persamaan tersebut dapat disederhanakan sehingga menjadi:

$$T(n) = 4(2^n) - n - 4 = O(2^n)$$

4.4.2 Analisis Solusi Divide and Conquer

Algoritma *divide and conquer* yang diimplementasikan dapat dilihat pada bagian 3.2.3. Algoritma tersebut bersifat rekursif dengan cara membagi

kurva menjadi bagian kiri dan kanan lalu menentukan dan menggabungkan kurva yang dibentuk oleh kedua bagian tersebut.

Pembagian kurva dilakukan dengan menentukan titik tengah dari ketiga titik kontrol terlebih dahulu (q_0 dan q_1) dan menentukan r_0 yang merupakan titik tengah dari q_0 dan q_1 . Proses tersebut akan menggunakan fungsi *mid_point* sebanyak tiga kali. Setelah mendapatkan q_0 , q_1 , dan r_0 , titik kurva bagian kiri dan titik kurva bagian kanan ditentukan dengan memanggil fungsi *dnc_kurva* secara rekursif sebanyak dua kali.

Berdasarkan jumlah pemanggilan fungsi *mid_point*, jika n adalah jumlah iterasi, kompleksitas waktu algoritma *divide and conquer* adalah:

$$T(n) = 0, \text{ untuk } n = 0$$

$$T(n) = 2T(n - 1) + 3, \text{ untuk } n > 0$$

Persamaan tersebut dapat disederhanakan menjadi:

$$T(n) = 3(2^n) - 3 = O(2^n)$$

4.4.3 Hasil Perbandingan Solusi Brute Force dengan Divide And Conquer

Walaupun hasil perbandingan menunjukkan bahwa kedua algoritma sama-sama memiliki kompleksitas waktu $O(2^n)$, dapat dilihat bahwa solusi dengan metode *divide and conquer* memiliki nilai $T(n)$ yang lebih kecil. Hal tersebut terjadi karena pada setiap iterasi, metode *brute force* menghasilkan kembali titik pada iterasi sebelumnya sehingga terdapat operasi yang dilakukan secara mubazir. Dapat disimpulkan bahwa pembangunan kurva Bézier menggunakan algoritma *divide and conquer* lebih efisien daripada menggunakan algoritma *brute force*.

BAB V

MASUKAN DAN LUARAN PROGRAM

5.1 Input CLI

5.1.1 Test Case 1 (*Brute Force*)

Data Input
Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 1 Number of points (minimum 2): 4

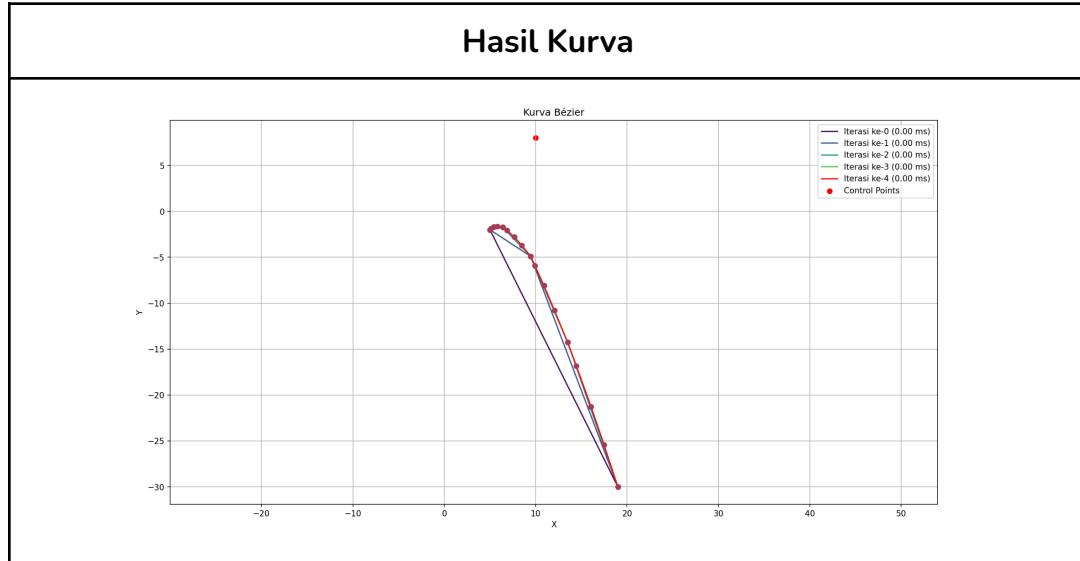
Hasil pada CLI
PS D:\ITB\Tugas Stima\Tucil\Tucil 2\Tucil2_13522053_13522108\src> python main.py Welcome to Bézier Curve Generator!  This program is used to create Bézier Curve using divide and conquer algorithm. ----- Wiga - Neo 13522053 - 13522108 ----- Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 1 Number of points (minimum 2): 4 Invalid input: Number of points max 3 for this method.

Hasil Kurva
-

5.1.2 Test Case 2 (Brute Force)

Data Input
<p>Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 1 Number of points (minimum 2): 3 Start point coordinate (separate with space): 5 -2 Coordinate of control point 1 (separate with space): 10 8 Final point coordinate (separate with space): 19 -30 Number of iteration: 4 t value (0 <= t <= 1): 0.3</p>

Hasil pada CLI
<pre>PS D:\ITB\Tugas Stima\Tucill\Tucill_13522053_13522108\src> python main.py Welcome to Bézier Curve Generator! ----- Wiga - Neo 13522053 - 13522108 ----- Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 1 Number of points (minimum 2): 3 Start point coordinate (separate with space): 5 -2 Coordinate of control point 1 (separate with space): 10 8 Final point coordinate (separate with space): 19 -30 Number of iteration: 4 t value (0 <= t <= 1): 0.3 Execution Time: 0.0 ms Final Points Bézier Curve [(5.0, -2.0), (5.129813839999999, -1.849281679999998), (5.425303999999998, -1.668007999999998), (5.829506959999998, -1.632339199999993), (6.382399999999998, -1.704799999999996), (6.850106959999997, -2.068739199999996), (7.677175999999998, -2.794951999999994), (8.496836239999997, -3.69956647999998), (9.439999999999994, -4.87999999999999), (9.934166059999995, -5.92389991999999), (10.91117599999998, -8.086952), (12.05364823999998, -10.76615848), (13.505599999999998, -14.27119999999997), (14.46461623999998, -16.84888648), (16.064743999999997, -21.266877999999994), (17.48824559999998, -25.41297512), (19.0, -30.0)]</pre>



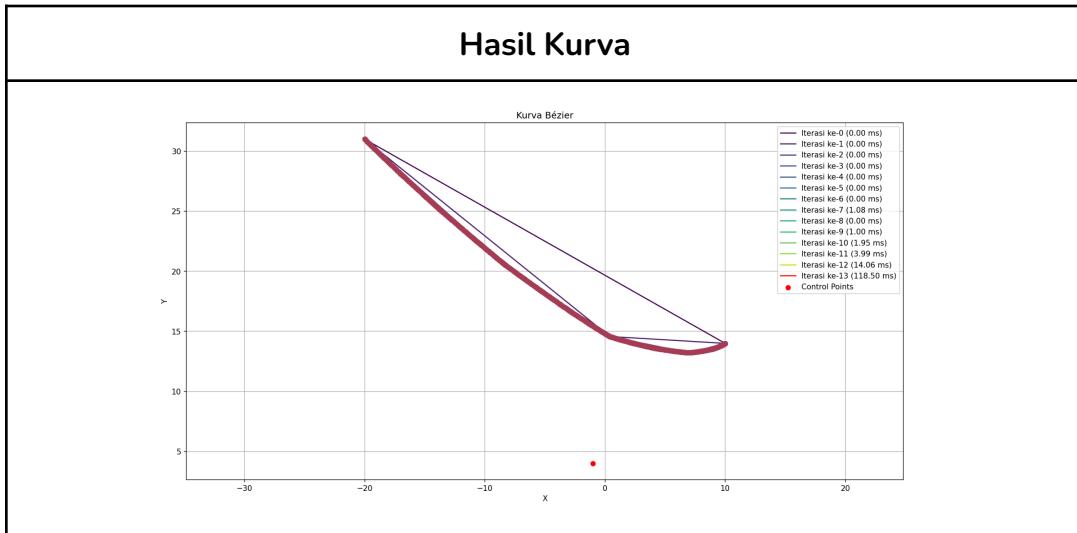
5.1.3 Test Case 3 (Brute Force)

Data Input

```
Choose algorithm to use
1. Brute Force
2. Divide and Conquer
Choice: 1
Number of points (minimum 2): 3
Start point coordinate (separate with space): 10 14
Coordinate of control point 1 (separate with space): -1 4
Final point coordinate (separate with space): -20 31
Number of iteration: 15
t value (0 <= t <= 1): 0.3
```

Hasil pada CLI

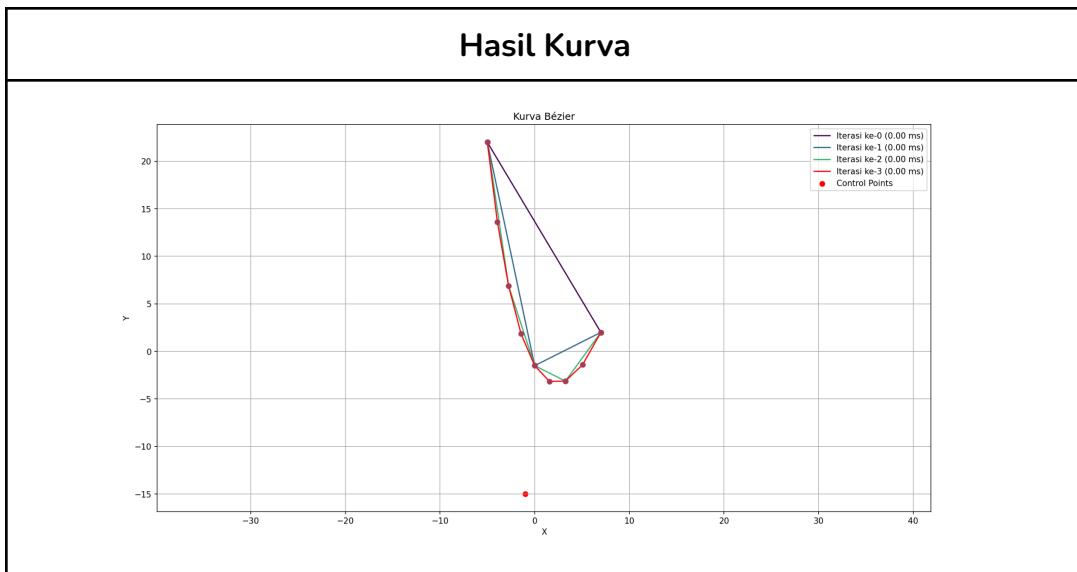
```
PS D:\ITB\Tugas Stima\Tucil\Tucil 2\Tucil2_13522053_13522108\src> python main.py
Welcome to Bézier Curve Generator!
-----  
BEZIER CURVE
-----  
This program is used to create Bézier Curve using divide and conquer algorithm.  
-----  
Wiga - Neo  
13522053 - 13522108  
-----  
Choose algorithm to use
1. Brute Force
2. Divide and Conquer
Choice: 1
Number of points (minimum 2): 3
Start point coordinate (separate with space): 10 14
Coordinate of control point 1 (separate with space): -1 4
Final point coordinate (separate with space): -20 31
Number of iteration: 15
t value (0 <= t <= 1): 0.3
Execution Time: 790.8677239998234 ms
Total Points Bézier Curve: 32769 points
```



5.1.4 Test Case 4 (Brute Force)

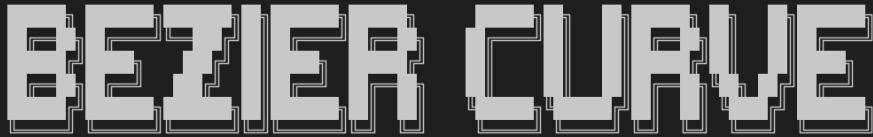
Data Input
Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 1 Number of points (minimum 2): 3 Start point coordinate (separate with space): 7 2 Coordinate of control point 1 (separate with space): -1 -15 Final point coordinate (separate with space): -5 22 Number of iteration: 3 t value (0 <= t <= 1): 0.5

Hasil pada CLI
PS D:\ITB\Tugas Stima\Tucill\Tucill_2\Tucill_13522053_13522108\src> python main.py Welcome to Bézier Curve Generator! BEZIER CURVE This program is used to create Bézier Curve using divide and conquer algorithm. ----- Wiga - Neo 13522053 - 13522108 ----- Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 1 Number of points (minimum 2): 3 Start point coordinate (separate with space): 7 2 Coordinate of control point 1 (separate with space): -1 -15 Final point coordinate (separate with space): -5 22 Number of iteration: 3 t value (0 <= t <= 1): 0.5 Execution Time: 0.0 ms Final Points Bézier Curve [(7.0, 2.0), (5.0625, -1.40625), (3.25, -3.125), (1.5625, -3.15625), (0.0, -1.5), (-1.4375, 1.84375), (-2.75, 6.875), (-3.9375, 13.59375), (-5.0, 22.0)]



5.1.5 Test Case 5 (Brute Force)

Data Input
Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 1 Number of points (minimum 2): 1

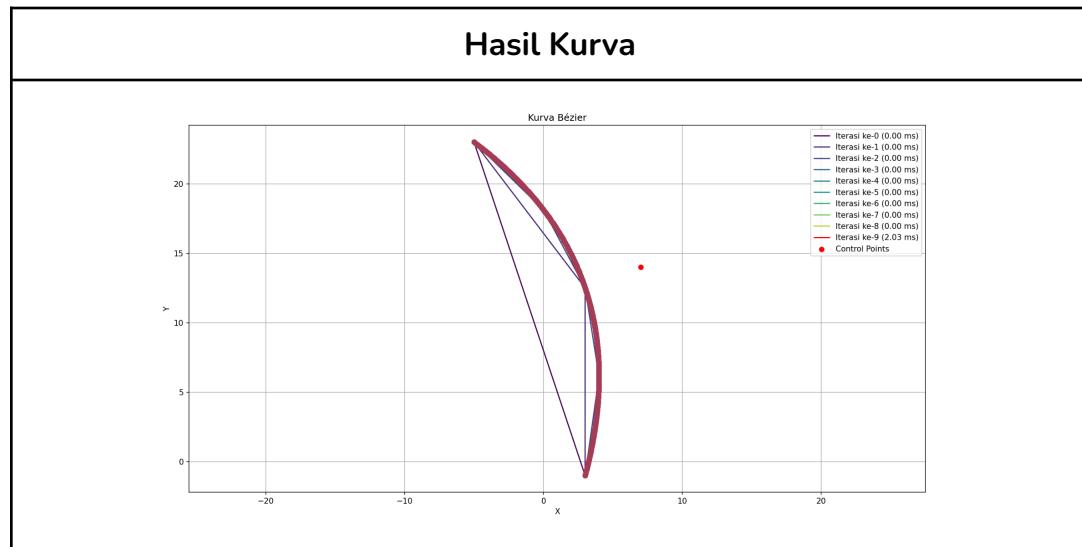
Hasil pada CLI
PS D:\ITB\Tugas Stima\Tucil\Tucil 2\Tucil2_13522053_13522108\src> python main.py Welcome to Bézier Curve Generator!  This program is used to create Bézier Curve using divide and conquer algorithm. ----- Wiga - Neo 13522053 - 13522108 ----- Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 1 Number of points (minimum 2): 1 Invalid input: Number of points max 3 for this method.

Hasil Kurva
-

5.1.6 Test Case 6 (Brute Force)

Data Input
Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 1 Number of points (minimum 2): 3 Start point coordinate (separate with space): 3 -1 Coordinate of control point 1 (separate with space): 7 14 Final point coordinate (separate with space): -5 23 Number of iteration: 13 t value (0 <= t <= 1): 0.4

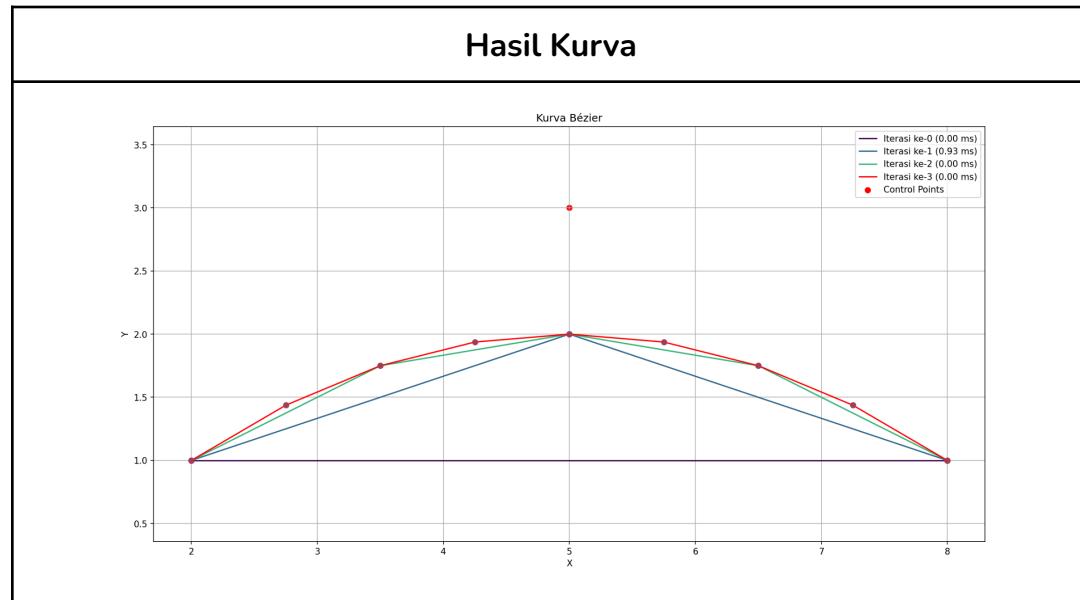
Hasil pada CLI
<pre>PS D:\ITB\Tugas Stima\Tucil\Tucil 2\Tucil2_13522053_13522108\src> python main.py Welcome to Bézier Curve Generator! <big>BEZIER CURVE</big> This program is used to create Bézier Curve using divide and conquer algorithm. ----- Wiga - Neo 13522053 - 13522108 ----- Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 1 Number of points (minimum 2): 3 Start point coordinate (separate with space): 3 -1 Coordinate of control point 1 (separate with space): 7 14 Final point coordinate (separate with space): -5 23 Number of iteration: 13 t value (0 <= t <= 1): 0.4 Execution Time: 66.07341766357422 ms Total Points Bézier Curve: 8193 points</pre>



5.1.7 Test Case 7 (Divide and Conquer)

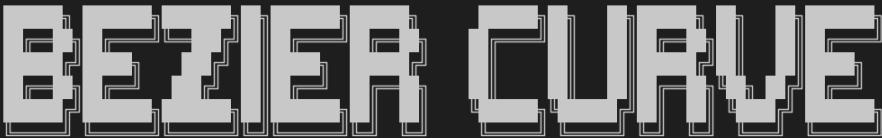
Data Input
Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 2 Number of points (minimum 2): 3 Start point coordinate (separate with space): 2 1 Coordinate of control point 1 (separate with space): 5 3 Final point coordinate (separate with space): 8 1 Number of iteration: 3

Hasil pada CLI
<pre>PS D:\ITB\Tugas Stima\Tucil\Tucil 2\Tucil2_13522053_13522108\src> python main.py Welcome to Bézier Curve Generator! <big>BÉZIER CURVE</big> This program is used to create Bézier Curve using divide and conquer algorithm. ----- Wiga - Neo 13522053 - 13522108 ----- Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 2 Number of points (minimum 2): 3 Start point coordinate (separate with space): 2 1 Coordinate of control point 1 (separate with space): 5 3 Final point coordinate (separate with space): 8 1 Number of iteration: 3 Execution Time: 0.0 ms Final Points Bézier Curve [(2.0, 1.0), (2.75, 1.4375), (3.5, 1.75), (4.25, 1.9375), (5.0, 2.0), (5.75, 1.9375), (6.5, 1.75), (7.25, 1.4375), (8.0, 1.0)]</pre>



5.1.8 Test Case 8 (*Divide and Conquer*)

Data Input
Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 2 Number of points (minimum 2): 1

Hasil pada CLI
PS D:\ITB\Tugas Stima\Tucil\Tucil 2\Tucil2_13522053_13522108\src> python main.py Welcome to Bézier Curve Generator!  This program is used to create Bézier Curve using divide and conquer algorithm. ----- Wiga - Neo 13522053 - 13522108 ----- Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 2 Number of points (minimum 2): 1 Invalid input: Number of points must be at least 2.

Hasil Kurva
-

5.1.9 Test Case 9 (*Divide and Conquer*)

Data Input
Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 2 Number of points (minimum 2): 3

```

Start point coordinate (separate with space): 10 14
Coordinate of control point 1 (separate with space): -1 4
Final point coordinate (separate with space): -20 31
Number of iteration: 15

```

Hasil pada CLI

```

PS D:\ITB\Tugas Stima\Tucil\Tucil 2\Tucil2_13522053_13522108\src> python main.py

```

Welcome to Bézier Curve Generator!



This program is used to create Bézier Curve using divide and conquer algorithm.

Wiga - Neo
13522053 - 13522108

Choose algorithm to use

1. Brute Force
2. Divide and Conquer

Choice: 2

Number of points (minimum 2): 3

Start point coordinate (separate with space): 10 14

Coordinate of control point 1 (separate with space): -1 4

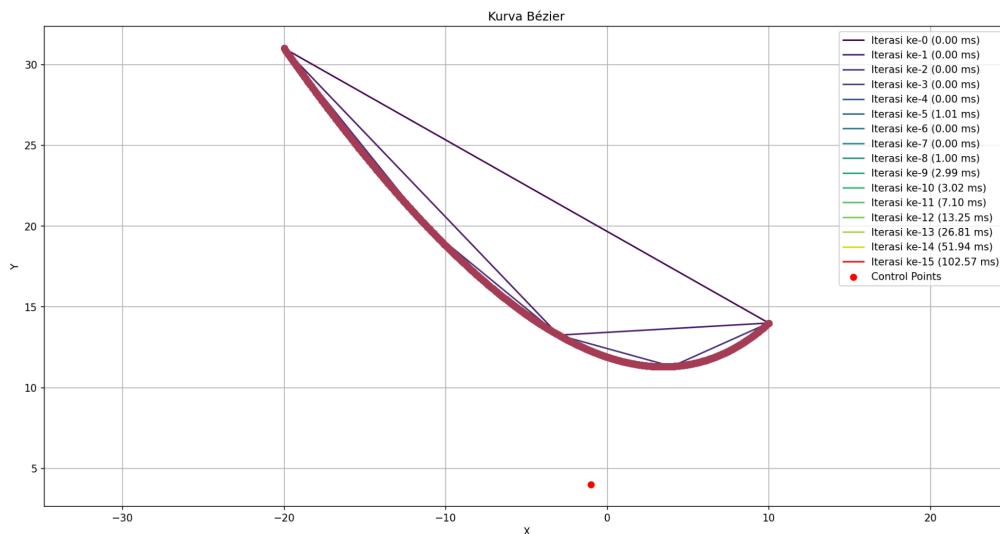
Final point coordinate (separate with space): -20 31

Number of iteration: 15

Execution Time: 184.30209159851074 ms

Total Points Bézier Curve: 32769 points

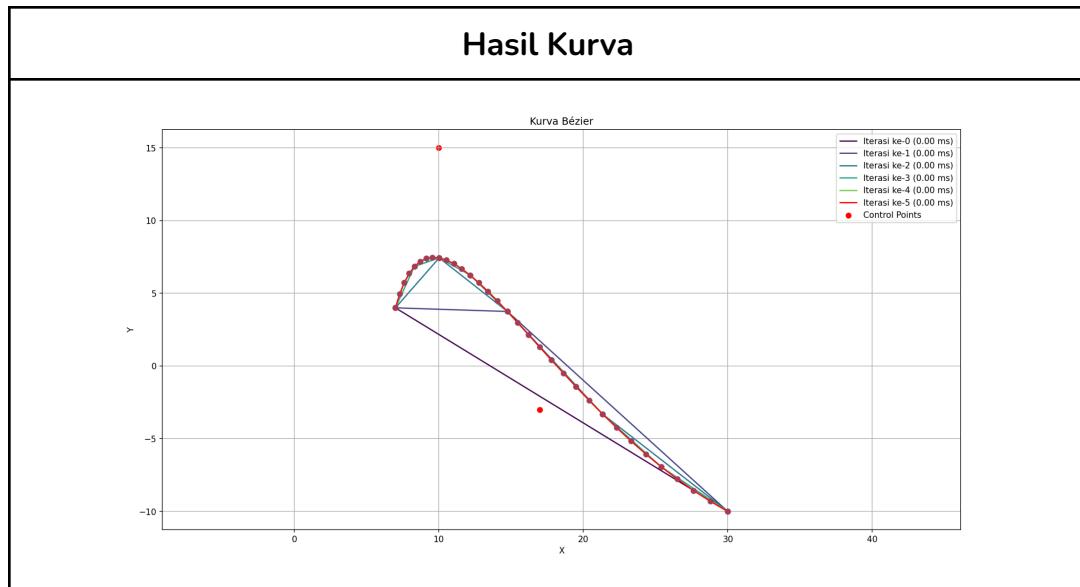
Hasil Kurva



5.1.10 Test Case 10 (Divide and Conquer)

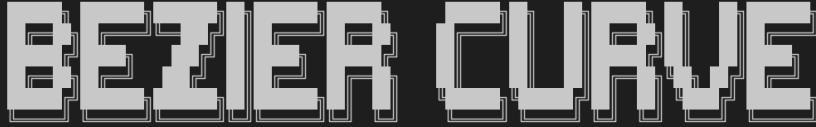
Data Input
<p>Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 2 Number of points (minimum 2): 4 Start point coordinate (separate with space): 7 4 Coordinate of control point 1 (separate with space): 10 15 Coordinate of control point 2 (separate with space): 17 -3 Final point coordinate (separate with space): 30 -10 Number of iteration: 5</p>

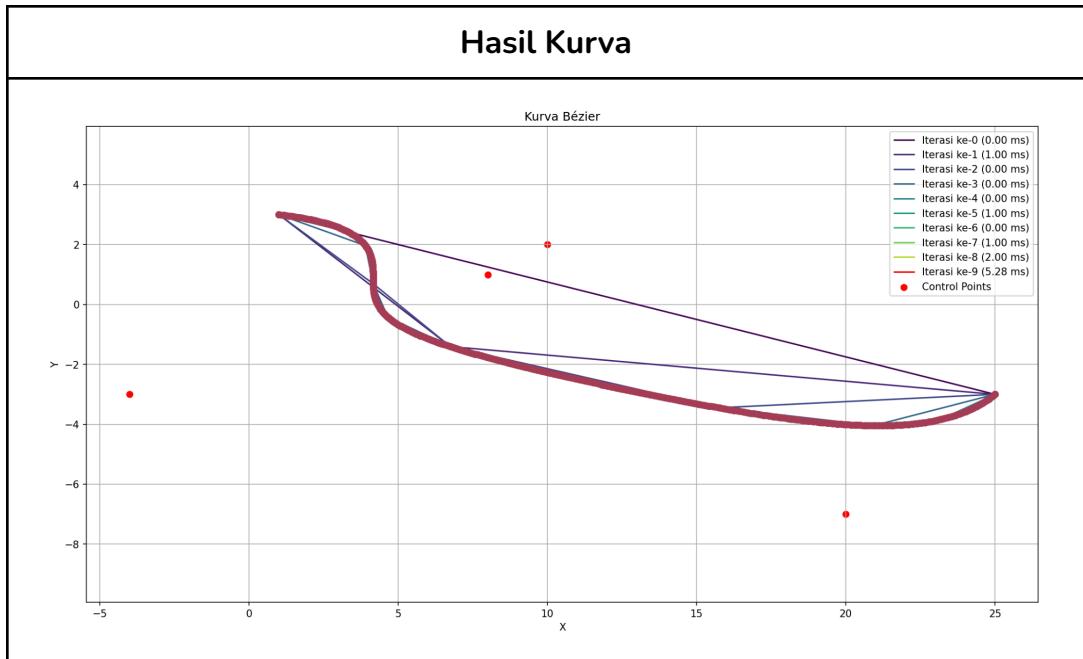
Hasil pada CLI
<pre>PS D:\ITB\Tugas Stima\Tucill\Tucill_13522059_13522108\src> python main.py Welcome to Bézier Curve Generator! BEZIER CURVE This program is used to create Bézier Curve using divide and conquer algorithm. Wiga - Neo 13522059 - 13522108 Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 2 Number of points (minimum 2): 4 Start point coordinate (separate with space): 7 4 Coordinate of control point 1 (separate with space): 10 15 Coordinate of control point 2 (separate with space): 17 -3 Final point coordinate (separate with space): 30 -10 Number of iteration: 5 Execution Time: 0.0 ms Final Points of the Curve: [(7, 0, 4.0), (7.125983973151925, 4.247209705625), (7.50985328125, 5.73421975), (7.9588669321875, 6.362869461975), (8.31048625, 6.84375), (8.76894140515), (9.2220645125), (9.6823228375), (10.14375), (10.6053125), (11.0673125), (11.52925), (12.0893125), (12.551359375, 7.393279125), (9.56390308059375, 7.474265234375), (10.43125, 7.4375), (10.526633780625, 7.267206646925), (11.04541015625, 7.037100375), (11.59225), (12.1390625, 6.886875), (12.167964875, 6.25), (12.77981298878125), (13.40185546875, 5.13476525), (14.061462449234375, 4.472412198975), (14, 75, 75), (15.4678344, 7265625, 2.974835315625), (16.21533208125, 3.15429875), (16.9928588671875, 1.29565429875), (17.80878125, 0.40625), (18.65946533203125, -0.506591796875), (19.5927734375, 6875), (20.4108534969375, -2.373291015625), (21.34375, -3.3125), (22.30914386648625, -4.245849609375), (23.30712898625, -5.166815625), (24.33887373046875, -6.065673828125), (25.402, 34375, -6.9375), (26.50898517578125, -7.774169921875), (27.63232421875, -8.56859375), (28.7987678984375, -9.312744140625), (30.0, -10.0)]</pre>



5.1.11 Test Case 11 (*Divide and Conquer*)

Data Input
Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 2 Number of points (minimum 2): 6 Start point coordinate (separate with space): 1 3 Coordinate of control point 1 (separate with space): 10 2 Coordinate of control point 2 (separate with space): -4 -3 Coordinate of control point 3 (separate with space): 8 1 Coordinate of control point 4 (separate with space): 20 -7 Final point coordinate (separate with space): 25 -3 Number of iteration: 9

Hasil pada CLI
○ <code>python main.py</code> Welcome to Bézier Curve Generator!  This program is used to create Bézier Curve using divide and conquer algorithm. ----- Wiga - Neo 13522053 - 13522108 ----- Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 2 Number of points (minimum 2): 6 Start point coordinate (separate with space): 1 3 Coordinate of control point 1 (separate with space): 10 2 Coordinate of control point 2 (separate with space): -4 -3 Coordinate of control point 3 (separate with space): 8 1 Coordinate of control point 4 (separate with space): 20 -7 Final point coordinate (separate with space): 25 -3 Number of iteration: 9 Execution Time: 5.971193313598633 ms Total Points Bézier Curve: 513 points



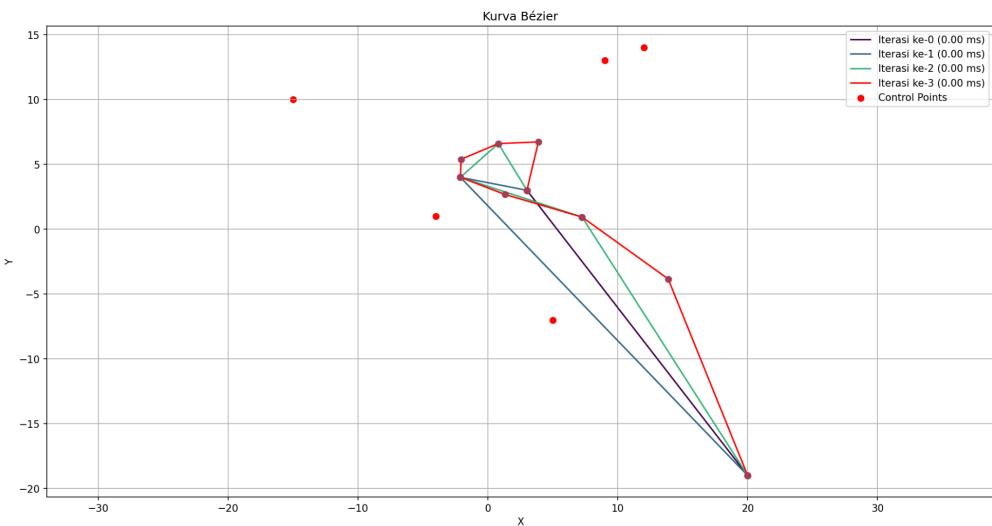
5.1.12 Test Case 12 (*Divide and Conquer*)

Data Input
Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 2 Number of points (minimum 2): 7 Start point coordinate (separate with space): 3 3 Coordinate of control point 1 (separate with space): 9 13 Coordinate of control point 2 (separate with space): -4 1 Coordinate of control point 3 (separate with space): -15 10 Coordinate of control point 4 (separate with space): 5 -7 Coordinate of control point 5 (separate with space): 12 14 Final point coordinate (separate with space): 20 -19 Number of iteration: 3

Hasil pada CLI

```
PS D:\ITB\Tugas Stima\Tucil2\Tucil 2\Tucil2_13522053_13522108\src> python main.py
Welcome to Bézier Curve Generator!
BEZIER CURVE
This program is used to create Bézier Curve using divide and conquer algorithm.
-----
Wiga      Neo
13522053 - 13522108
-----
Choose algorithm to use
1. Brute Force
2. Divide and Conquer
Choice: 2
Number of points (minimum 2): 7
Start point coordinate (separate with space): 3 3
Coordinate of control point 1 (separate with space): 9 13
Coordinate of control point 2 (separate with space): -4 1
Coordinate of control point 3 (separate with space): -15 18
Coordinate of control point 4 (separate with space): 5 -7
Coordinate of control point 5 (separate with space): 12 14
Final point coordinate (separate with space): 20 -19
Number of iteration: 3
Execution Time: 0.0 ms
Final Points Bézier Curve [(3.0, 3.0), (3.8824615478515625, 6.7288665771484375), (0.7958984375, 6.6025398025), (-2.0711517333984375, 5.3904876708984375), (-12.125, 4.8), (1.3090972960390625, 2.679183959609375), (.72451171875, 0.9345703125), (13.876724243164862, -3.8366851806540625), (20.0, -19.0)]
```

Hasil Kurva



5.1.13 Test Case 13 (Invalid Input)

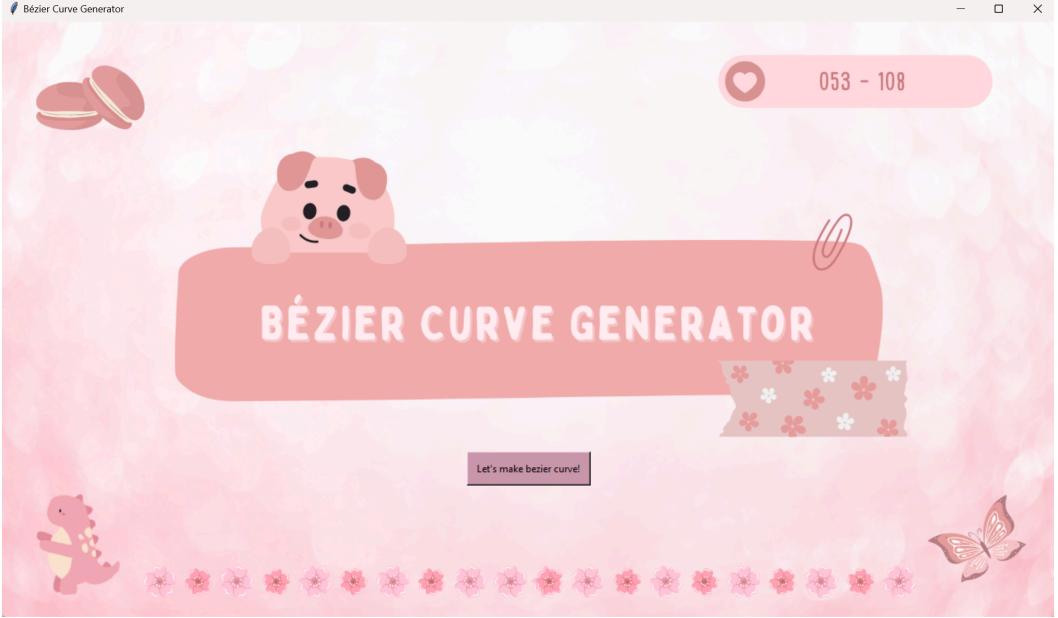
Data Input
Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 5

Hasil pada CLI
<pre>PS D:\ITB\Tugas Stima\Tucil\Tucil 2\Tucil2_13522053_13522108\src> python main.py Welcome to Bézier Curve Generator! BÉZIER CURVE This program is used to create Bézier Curve using divide and conquer algorithm. ----- Wiga - Neo 13522053 - 13522108 ----- Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: 5 Invalid input. Please input 1 or 2. Choose algorithm to use 1. Brute Force 2. Divide and Conquer Choice: </pre>

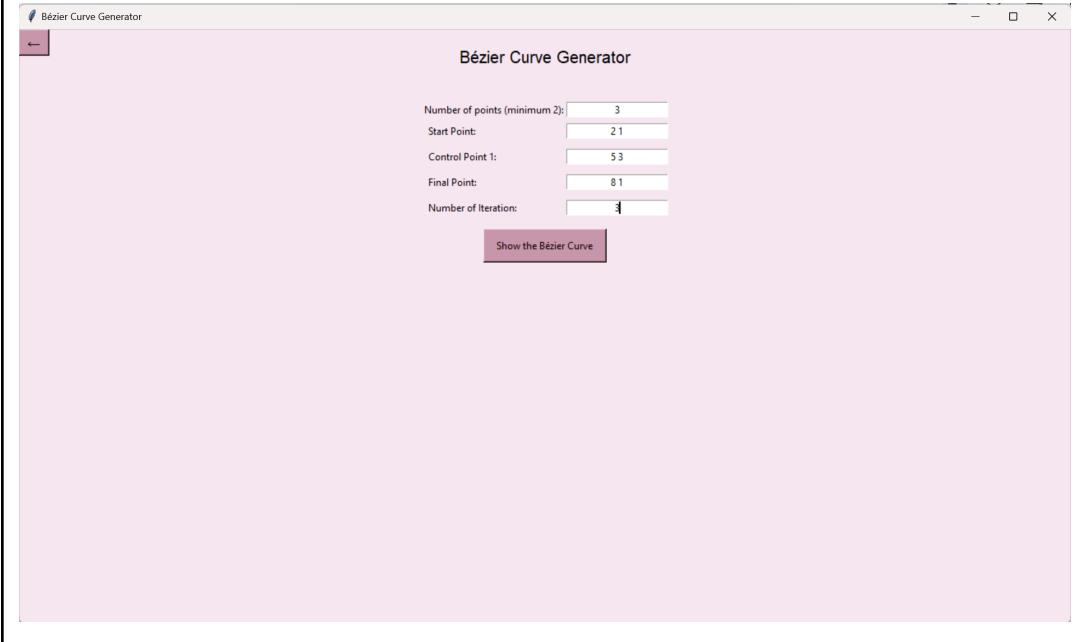
Hasil Kurva
-

5.2 Menggunakan GUI

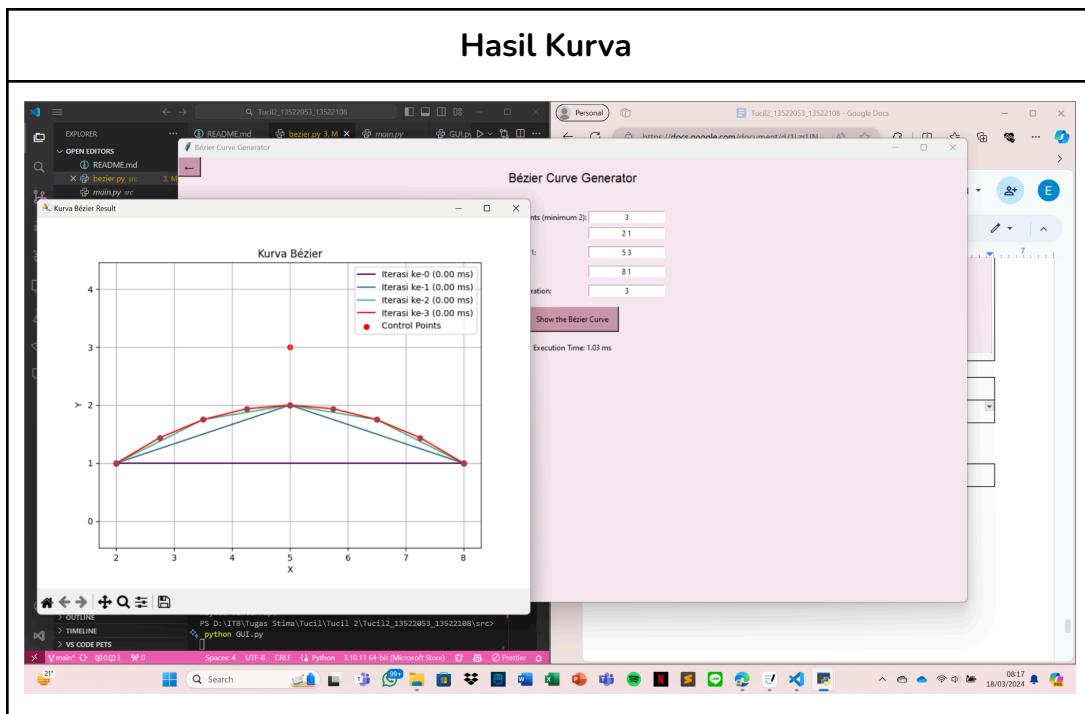
5.2.1 Test Case 1

Halaman Utama	
	
Data Input	
<pre>Number of points (minimum 2): 3 Start point: 2 1 Control point 1: 5 3 Final point: 8 1 Number of iteration: 3</pre>	

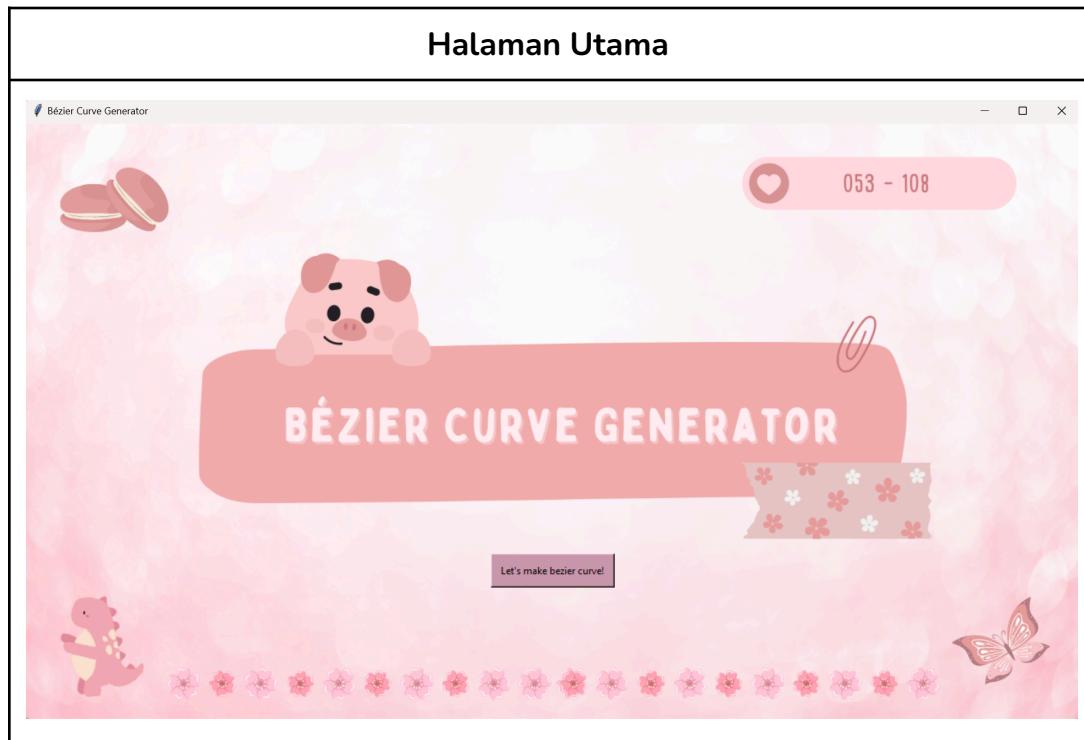
Halaman Input GUI



Hasil Kurva



5.2.2 Test Case 2



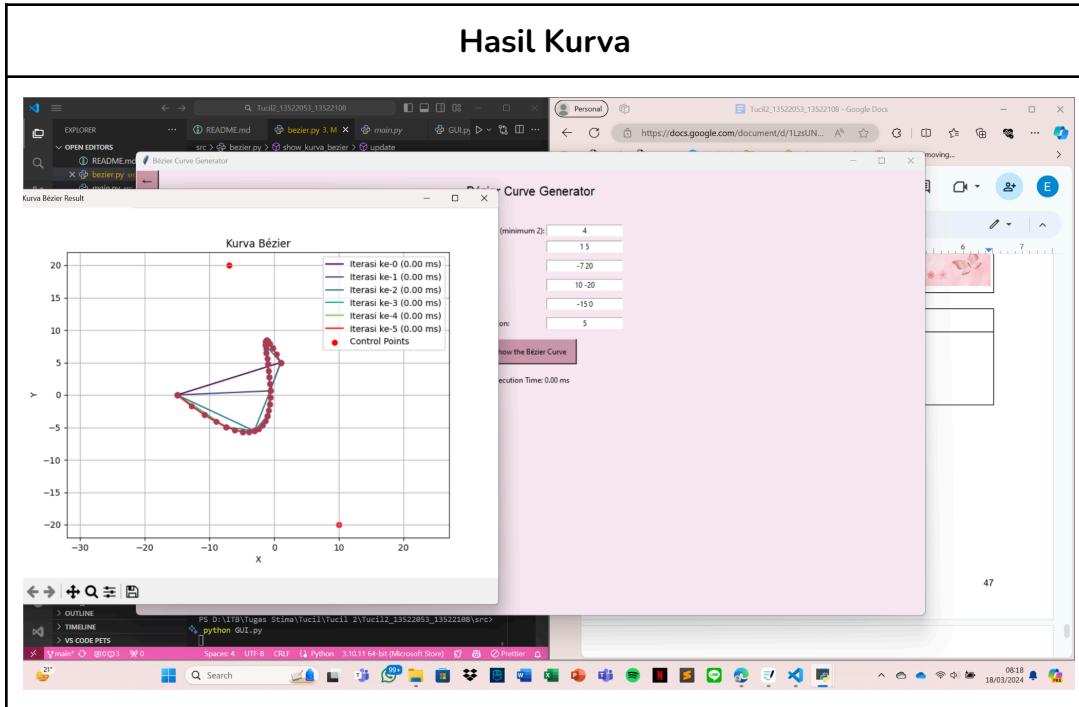
Data Input

Number of points (minimum 2): 4
Start point: 1 5
Control point 1: -7 20
Control point 2: 10 -20
Final point: -15 0
Number of iteration: 5

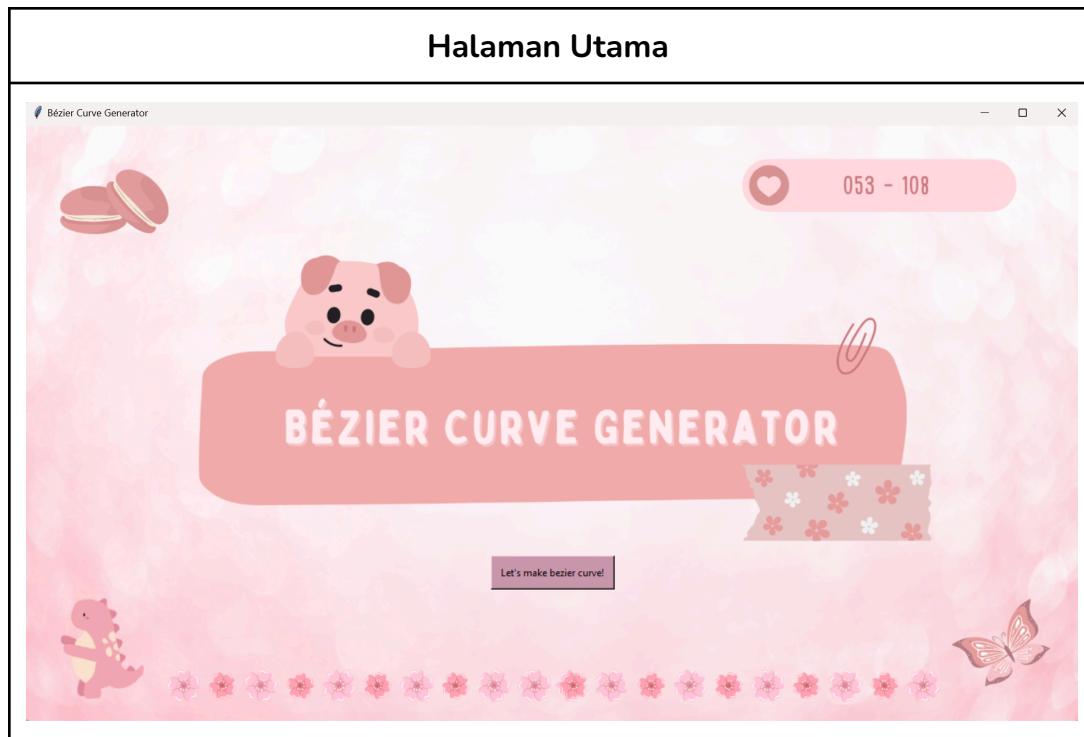
Halaman Input GUI



Hasil Kurva



5.2.3 Test Case 3

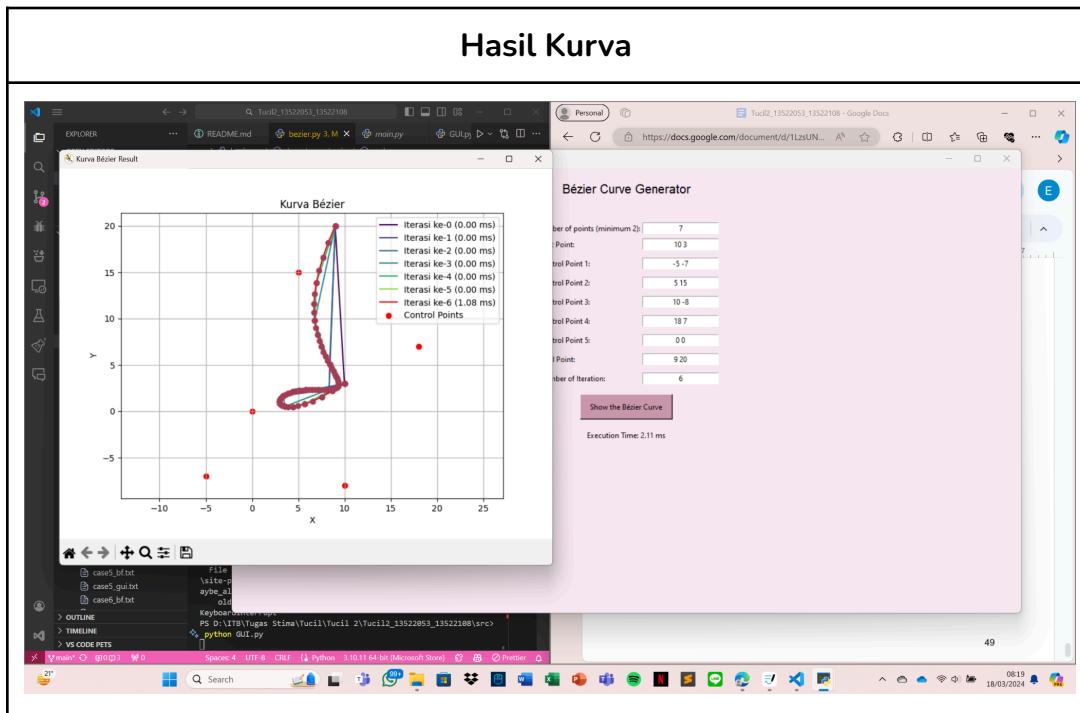


Data Input	
Number of points (minimum 2):	7
Start point:	10 3
Control point 1:	-5 -7
Control point 2:	5 15
Control point 3:	10 -8
Control point 4:	18 7
Control point 5:	0 0
Final point:	9 20
Number of iteration:	6

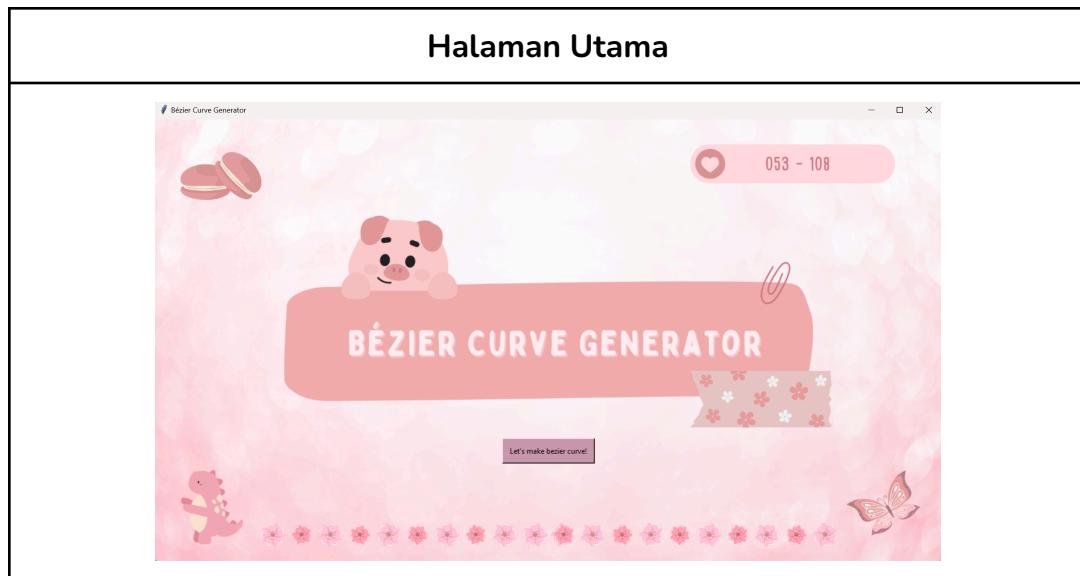
Halaman Input GUI



Hasil Kurva

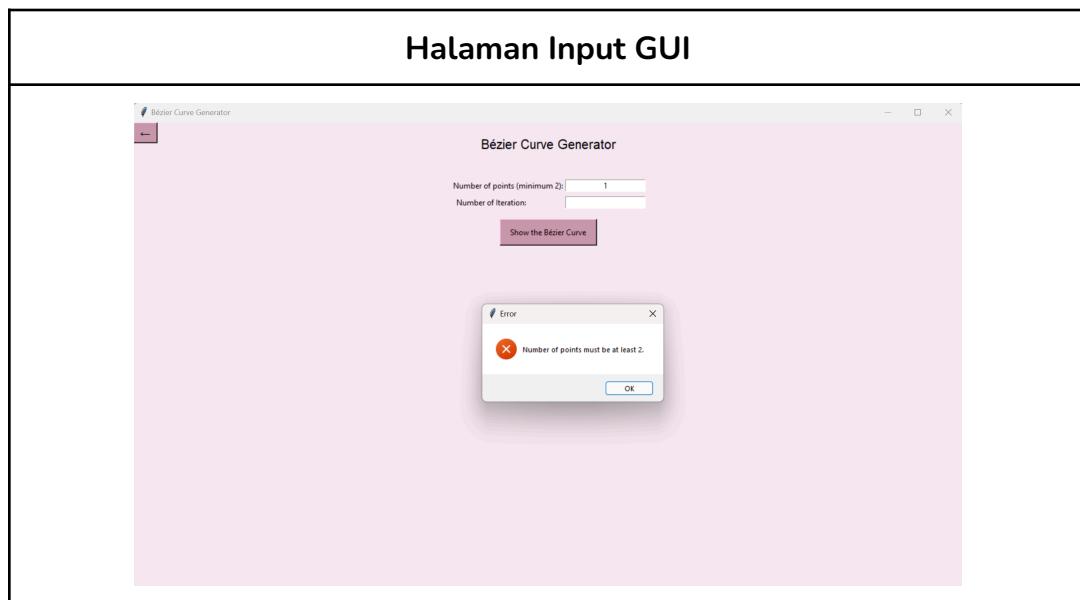


5.2.4 Test Case 4

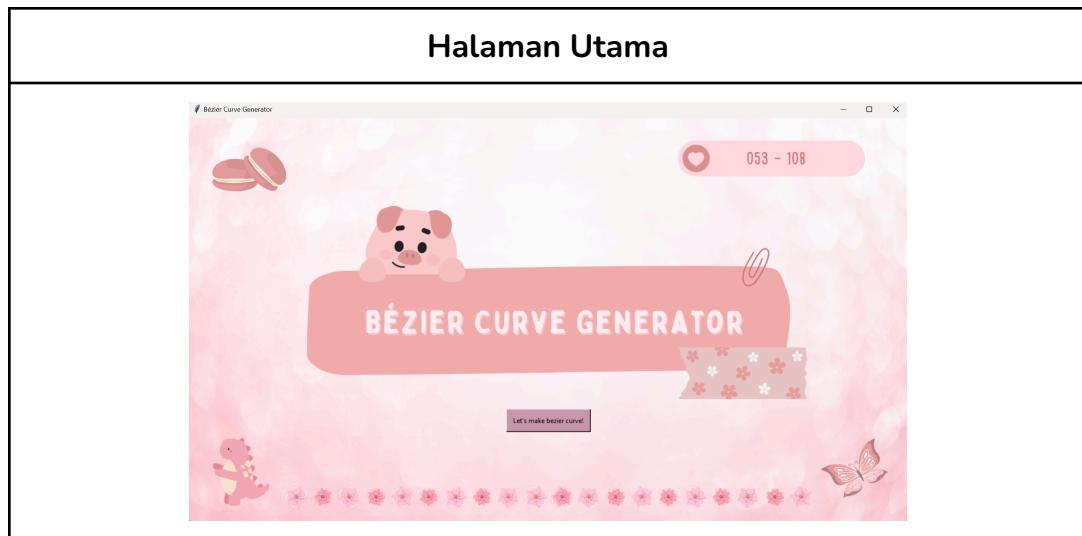


Data Input

Number of points (minimum 2): 1
Number of iteration:

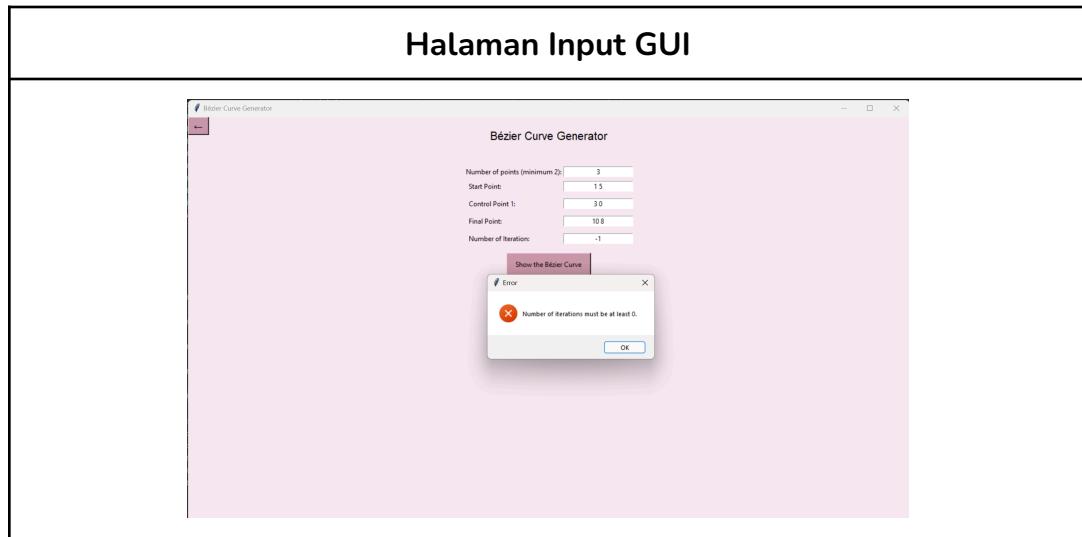


5.2.5 Test Case 5

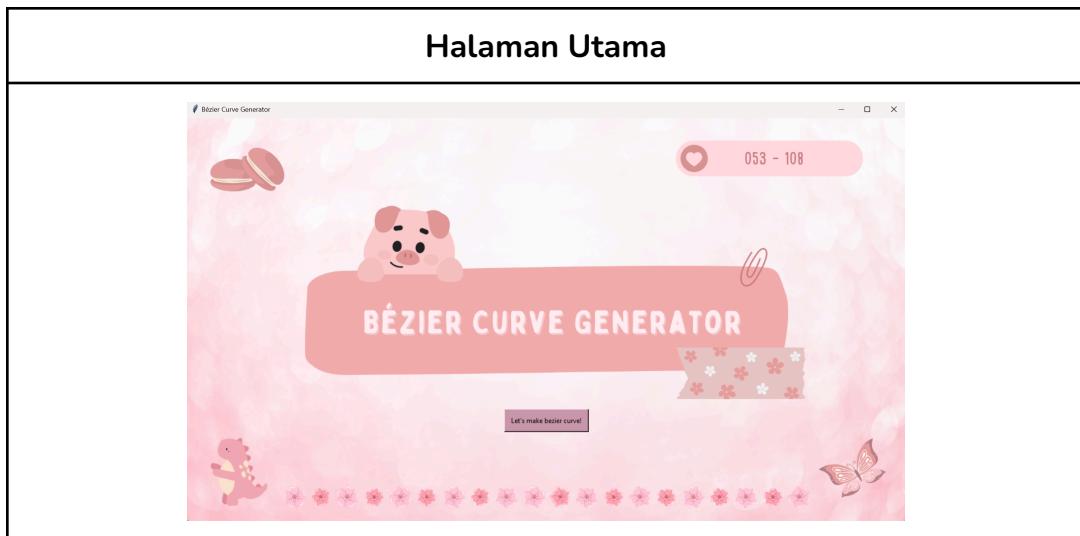


Data Input

Number of points (minimum 2): 3
Start point: 1 5
Control point 1: 3 0
Final point: 10 8
Number of iteration: -1

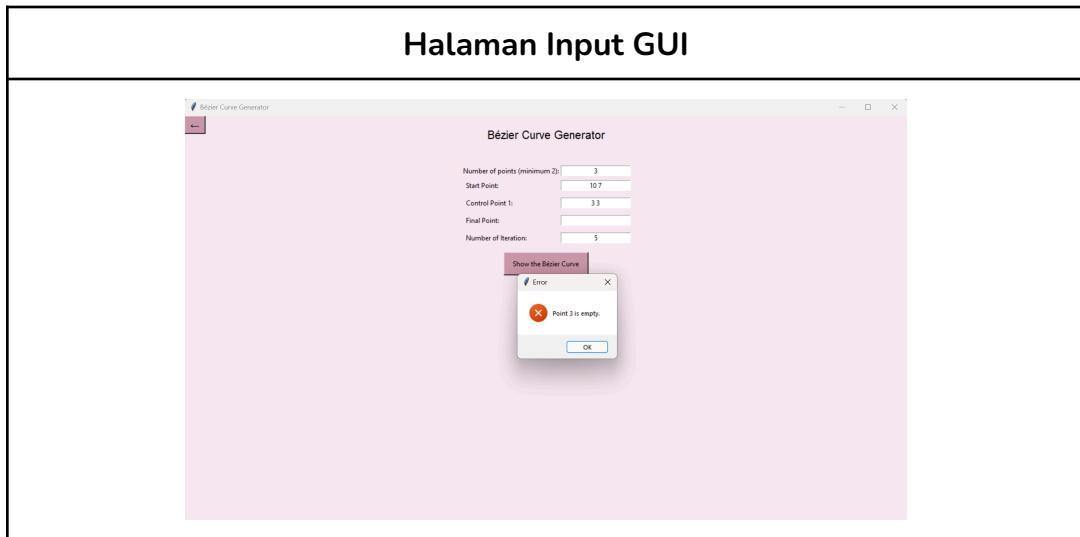


5.2.6 Test Case 6

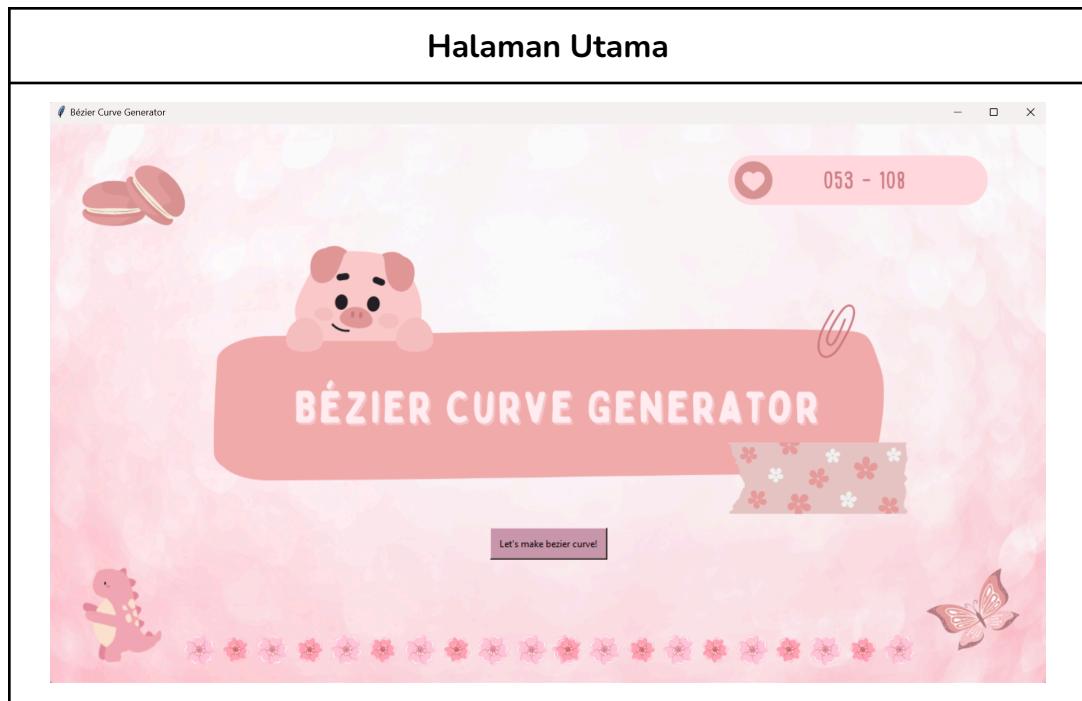


Data Input

Number of points (minimum 2): 3
Start point: 10 7
Control point 1: 3 3
Final point:
Number of iteration: 5

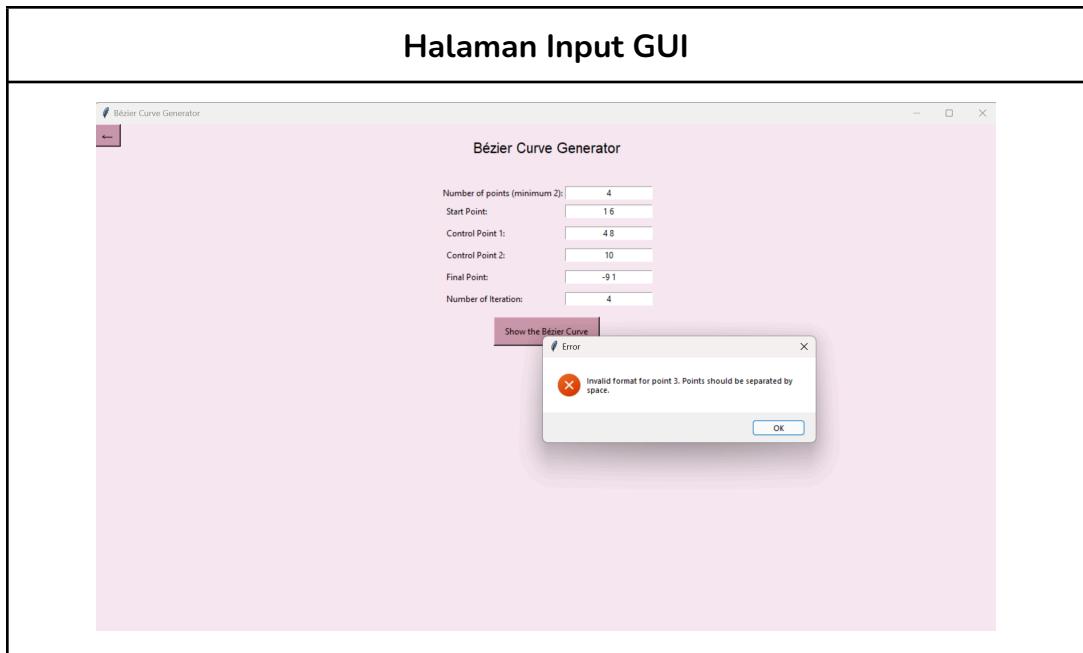


5.2.7 Test Case 7

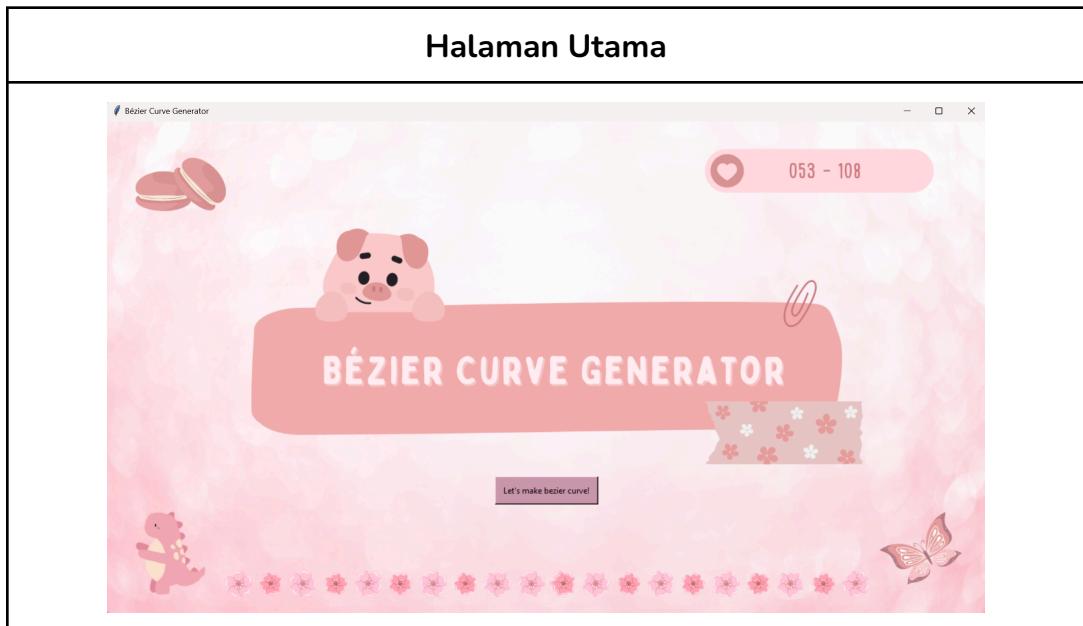


Data Input

Number of points (minimum 2): 4
Start point: 1 6
Control point 1: 4 8
Control point 2: 10
Final point: -9,1
Number of iteration: 4



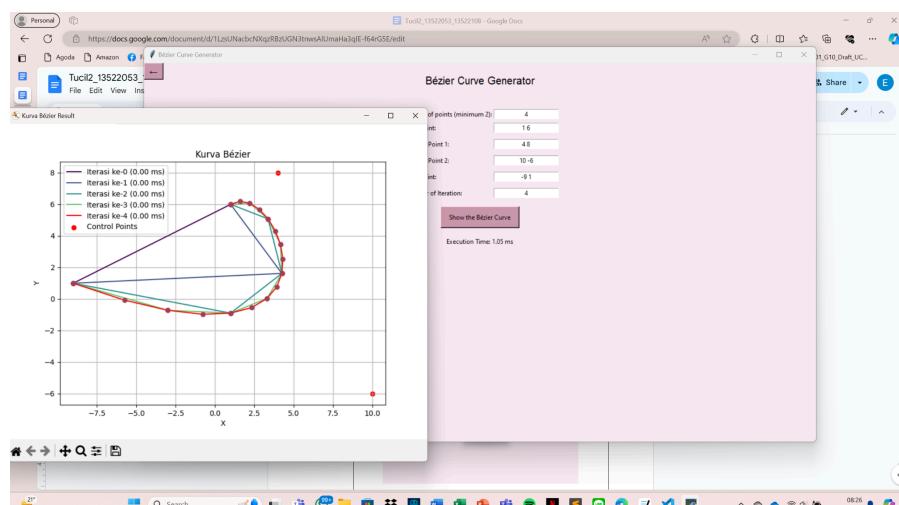
5.2.8 Test Case 8



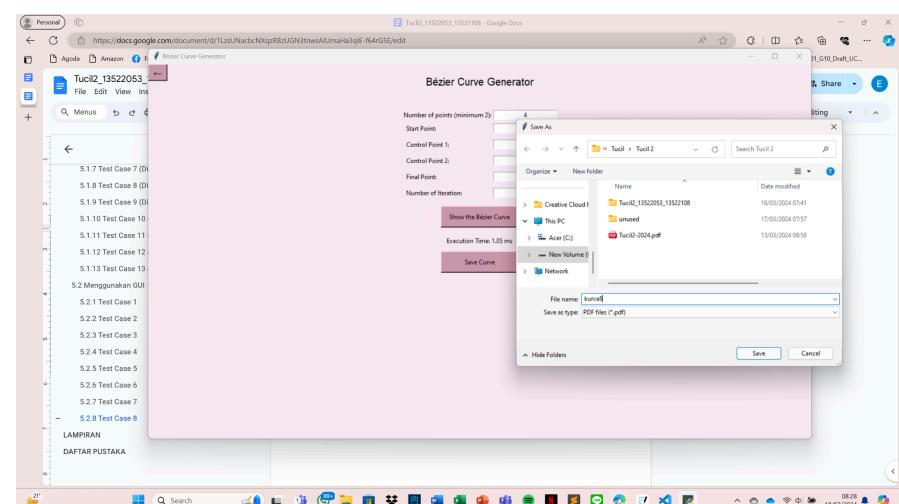
Data Input

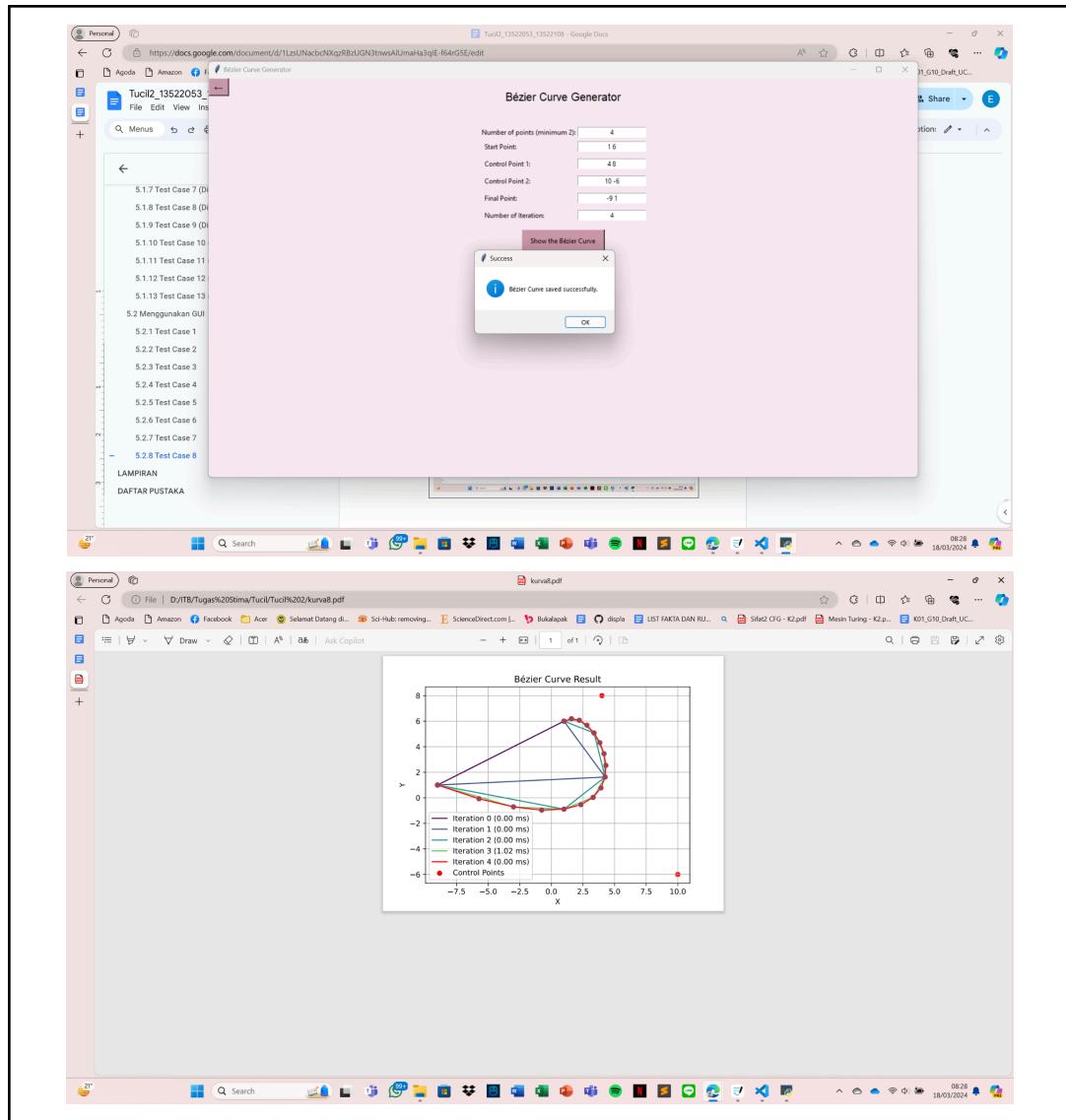
Number of points (minimum 2): 4
Start point: 1 6
Control point 1: 4 8
Control point 2: 10 -6
Final point: -9 1
Number of iteration: 4

Halaman Kurva



Hasil Saved





LAMPIRAN

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.	✓	

DAFTAR PUSTAKA

- [1] [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Baqian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Baqian1.pdf)
- [2] GfG. (2024, February 22). Divide and conquer. GeeksforGeeks.
<https://www.geeksforgeeks.org/divide-and-conquer/#what-is-divide-and-conquer>
- [3] *Divide and conquer algorithm.* (n.d.). <https://www.programiz.com/dsa/divide-and-conquer>
- [4] Melo, M. (2022, January 5). Understanding Bézier curves - Mateus Melo - medium. Medium.
<https://mmrndev.medium.com/understanding-b%C3%A9zier-curves-f6eaa0fa6c7d>
- [5] Halliday, S. (2021, March 25). Quadratic Bezier Curve demo. An Invert Look.
<https://simonhalliday.com/2017/02/15/quadratic-bezier-curve-demo/>