LAPORAN TUGAS KECIL 2

STRATEGI ALGORITMA

Nama:

Hafizh Hananta Akbari
(13522132)

INSTITUT TEKNOLOGI BANDUNG

2023

BAB I
TEORI DASAR

1.1. KURVA BEZIER

Kurva Bezier adalah kurva yang biasa digunakan dalam animasi dan media grafis lainnya. Kurva Bezier mampu dibuat dengan menentukan beberapa titik kontrol serta menghubungkannya dengan suatu kurva. Kurva yang dibuat tergantung dengan titik kontrol yang digunakan dengan jumlahnya yang mampu menentukan kehalusan atau ketajaman kurva yang dihasilkan. Kurva yang dihasilkan mampu memiliki karakteristik yang sedikit berbeda tergantung dengan kebutuhan.

Kurva Bezier tidak menghubungkan semua titik kontrol namun menggunakan beberapa titik kontrol untuk memanipulasi bentuk kurva yang pada akhirnya menghubungi beberapa titik kontrol lainnya. Kurva Bezier selalu melewati titik kontrol awal dan titik kontrol akhir. Selain titik kontrol awal dan akhir, titik kontrol yang digunakan banyaknya adalah titik kontrol yang memanipulasi bentuk kurva yang belum tentu akan dilewati kurva. Titik-titik kontrol tersebut memanipulasi bentuk kurva dengan menjadi panduan untuk pembuatan kurva. Panduan tersebut dapat terlihat dengan menggambarkan beberapa garis yang menghubungkan dua titik kontrol tertentu sehingga dapat memunculkan panduan untuk menentukan kurva.

Kurva Bezier memiliki beberapa jenis yang memiliki ciri-ciri dan kegunaan yang berbeda. Berikut adalah jenis-jenis dari kurva Bezier:

1. Kurva Bezier Linier

   Kurva Bezier Linier adalah kurva yang ditentukan dengan hanya dua titik kontrol yaitu titik awal dan titik akhir. Kurva ini digambarkan dengan garis lurus. Berikut ini adalah perumusannya.

   $$Q_0 = B(t) = (1 - t)P_0 + tP_1$$

   Di rumus ini, *t* dalam fungsi kurva Bezier linier menggambarkan seberapa jauh fungsi *B(t)* dari titik awal yaitu P_0 ke titik akhir yaitu P_1.

2. Kurva Bezier Kuadratik

Kurva Bezier Kuadratik adalah kurva yang ditentukan dengan tiga titik kontrol yaitu titik awal, titik akhir, dan sebuah titik kontrol bebas. Kurva ini memiliki bentuk kurva yang fleksibel. Berikut ini adalah perumusannya.

$$Q_0 = B(t) = (1-t)P_0 + tP_1$$
$$Q_1 = B(t) = (1-t)P_1 + tP_2$$
$$R_0 = B(t) = (1-t)Q_0 + tQ_1$$

Di rumus ini, *t* dalam fungsi kurva Bezier linier menggambarkan seberapa jauh fungsi *B(t)* dari titik awal ke titik akhir sesuai dengan rumus. P_0, P_1, dan P_2 adalah titik-titik kontrol awal, antara, dan akhir. Dengan substitusi nilai Q_0 dan Q_1, didapat rumus dibawah ini.

$$R_0 = B(t) = (1-t)^2P_0 + (1-t)P_1 + t^2P_2$$

3. Kurva Bezier Kubik

Kurva Bezier Kubik adalah kurva yang ditentukan dengan empat titik kontrol yaitu titik awal, titik akhir, dan dua buah titik kontrol bebas. Kurva ini memiliki bentuk kurva yang lebih fleksibel dibanding Kurva Bezier Kuadratik sehingga lebih biasa digunakan.

4. Kurva Bezier Derajat Tinggi

Kurva Bezier Derajat Tinggi adalah kurva yang ditentukan dengan lebih dari tiga titik kontrol yaitu titik awal, titik akhir, dan beberapa titik kontrol bebas. Kurva ini memiliki bentuk kurva yang lebih fleksibel dibanding Kurva Bezier Kubik namun lebih sulit untuk dikelola.

1.2. ALGORITMA DIVIDE AND CONQUER

Algoritma *divide and conquer* adalah suatu algoritma yang bisa diaplikasikan di pemrograman. Algoritma ini berfungsi dengan membagi suatu persoalan menjadi beberapa subbagian. Lalu, setiap subbagian diproses secara terpisah sebelum digabung kembali untuk membuat solusi untuk persoalan yang dikerjakan. Seperti nama dari algoritmanya,

algoritma ini memiliki unsur *divide* dan *conquer* dimana unsur *divide* berupa pembagian persoalan menjadi beberapa subbagian dan unsur *conquer* berupa penyelesaian subbagian tersebut sebelum digabung kembali.

Algoritma divide and conquer memiliki beberapa kelebihan. Pertama adalah kemampuan algoritma ini untuk menyelesaikan persoalan skala besar dalam waktu yang relatif lebih cepat. Dengan pembagian suatu persoalan menjadi beberapa subbagian berbeda, setiap persoalan mampu dipecahkan secara paralel sehingga persoalan tersebut berhasil diselesaikan dengan lebih cepat. Kemudian, algoritma ini juga mampu digunakan secara berulang karena bentuknya yang cenderung memiliki struktur yang rekursif. Semua hal tersebut memungkinkan algoritma divide and conquer untuk menjadi pilihan algoritma yang bisa efisien, cepat, dan mudah digunakan.

Selain kelebihan, algoritma divide and conquer juga memiliki beberapa kekurangan. Kekurangan tersebut adalah pengimplementasian algoritma yang apabila buruk, dapat mempengaruhi kinerja program secara negatif. Contoh pengaruh tersebut adalah penggunaan rekursi yang berlebihan yang mampu mengurangi kinerja program. Selain itu, penggabungan solusi-solusi subbagian menjadi solusi persoalan secara keseluruhan mampu mengurangi kinerja program apabila tidak diimplementasikan dengan baik. Lalu, tidak semua persoalan mampu diselesaikan dengan sempurna dengan menggunakan algoritma divide and conquer.

BAB 2
IMPLEMENTASI ALGORITMA

2.1. ALGORITMA BRUTE FORCE

Pada program kurva Bezier kuadratik, algoritma *brute force* diaplikasikan dengan membuat titik-titik kontrol awal. Lalu, setiap segmen dibagi dengan menggunakan titik tengah sebagai titik pembaginya. Kemudian, setiap posisi titik pada suatu segmen dievaluasi dengan suatu parameter yang bervariasi agar mampu dibangun kurva Bezier dengan menghubungkan titik-titik yang didapat dengan mengubah parameter yang digunakan. Metode yang sama digunakan untuk setiap segmen hingga semua segmen telah diproses.

Inti dari pengimplementasian algoritma ini adalah pada pengevaluasian kurva. Pada implementasi algoritma ini, kurva dibagi dengan menambah titik kontrol yang baru di titik tengah setiap segmen sebelum kurva dievaluasi dengan melihat titik kontrol yang awal dengan titik kontrol yang baru. Proses ini dilakukan secara berulang hingga semua iterasi selesai.

2.2. ALGORITMA DIVIDE AND CONQUER

Pada program kurva Bezier kuadratik, algoritma *divide and conquer* diaplikasikan dengan membuat titik-titik kontrol awal. Lalu, setiap segmen dibagi dengan menggunakan titik tengah sebagai titik pembaginya. Kemudian, kurva dibagi lagi berdasarkan posisi-posisi titik yang didapat dari langkah sebelumnya dengan menggunakan titik tengah dari titik kontrol awal dan titik hasil pembagian. Setelah pembagian lebih lanjut dari kurva, setiap segmen dievaluasi dengan parameter yang bervariasi untuk mendapat titik-titik dari kurva Bezier sebelum menggabungkan semua segmen di langkah akhir untuk membuat kurva Bezier secara keseluruhan.

Pendekatan *divide and conquer* terlihat memiliki beberapa kesamaan dengan *brute force*. Meskipun itu, pendekatan divide and conquer berbeda dengan pendekatan *brute force*. Perbedaannya terletak di pengevaluasian kurva dimana setelah pembagian segmen kurva dengan lebih kecil, setiap segmen diproses secara terpisah dengan menggunakan dua

titik kontrol yang digunakan di segmen tersebut. Setelah pemrosesan setiap segmen, segmen-segmen tersebut digabung untuk membentuk kurva Bezier yang utuh.

BAB 3
PROGRAM

3.1. Penjelasan Singkat Program dan Source Code

Program ini adalah program yang menggambar kurva Bezier kuadratik dengan menggunakan dua algoritma berbeda yaitu algoritma *divide and conquer* dan algoritma *brute force*. Algoritma ini meminta empat inputan berbeda yaitu jumlah iterasi dan 3 titik kontrol yang digunakan. Algoritma ini mengeluarkan *output* yaitu gambar kurva Bezier kuadratik dan waktu pemrosesan yang dibutuhkan.

Berikut ini adalah *source code* dari program yang telah dibuat.

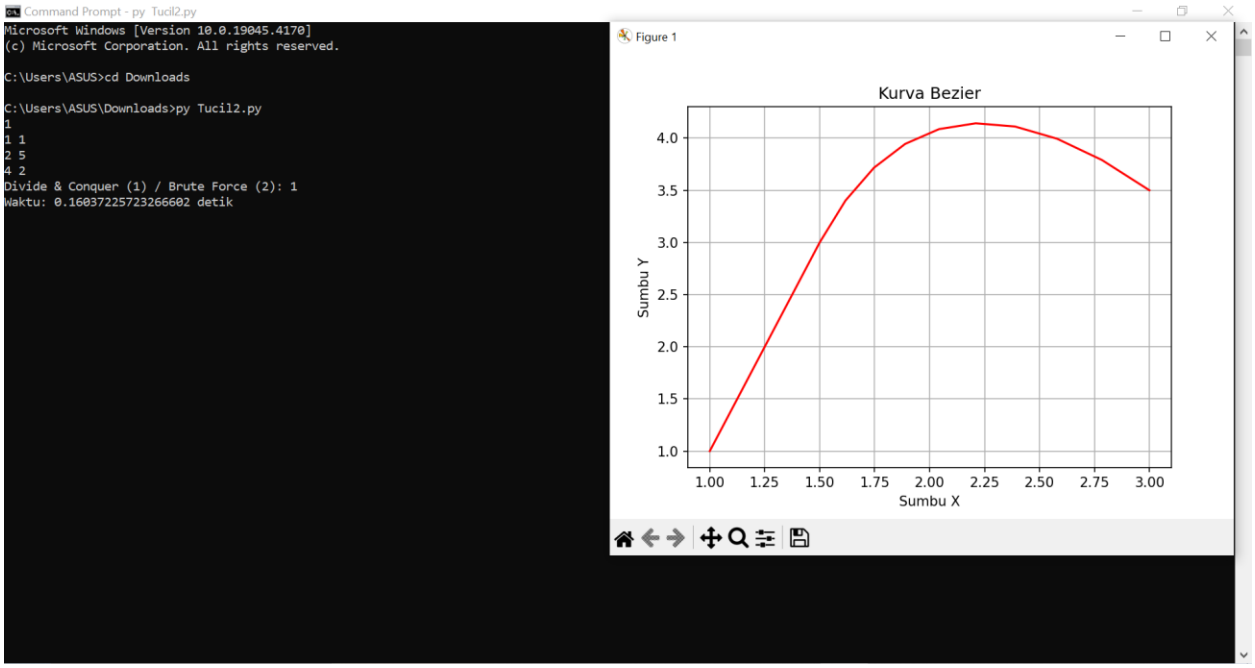
```
#Kurva Bezier di Python

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

# Fungsi-Fungsi Pembantu
def plot_control_points(control_points):
    x_points = [point[0] for point in control_points]
    y_points = [point[1] for point in control_points]
    plt.plot(x_points, y_points, 'bo-')

def evaluate_bezier_segment(segment_points, t):
    # Titik kontrol kurva Bezier
    P0, P1, P2 = segment_points

    # Rumus Bezier kuadratik
    x = (1 - t) ** 2 * P0[0] + 2 * (1 - t) * t * P1[0] + t ** 2 * P2[0]
    y = (1 - t) ** 2 * P0[1] + 2 * (1 - t) * t * P1[1] + t ** 2 * P2[1]

    return x, y

# Fungsi Divide and Conquer
def bezierDivideConquer(control_points, iterations):
    # Inisialisasi array posisi titik kurva Bezier
    bezier_points = []

    for _ in range(iterations):
```

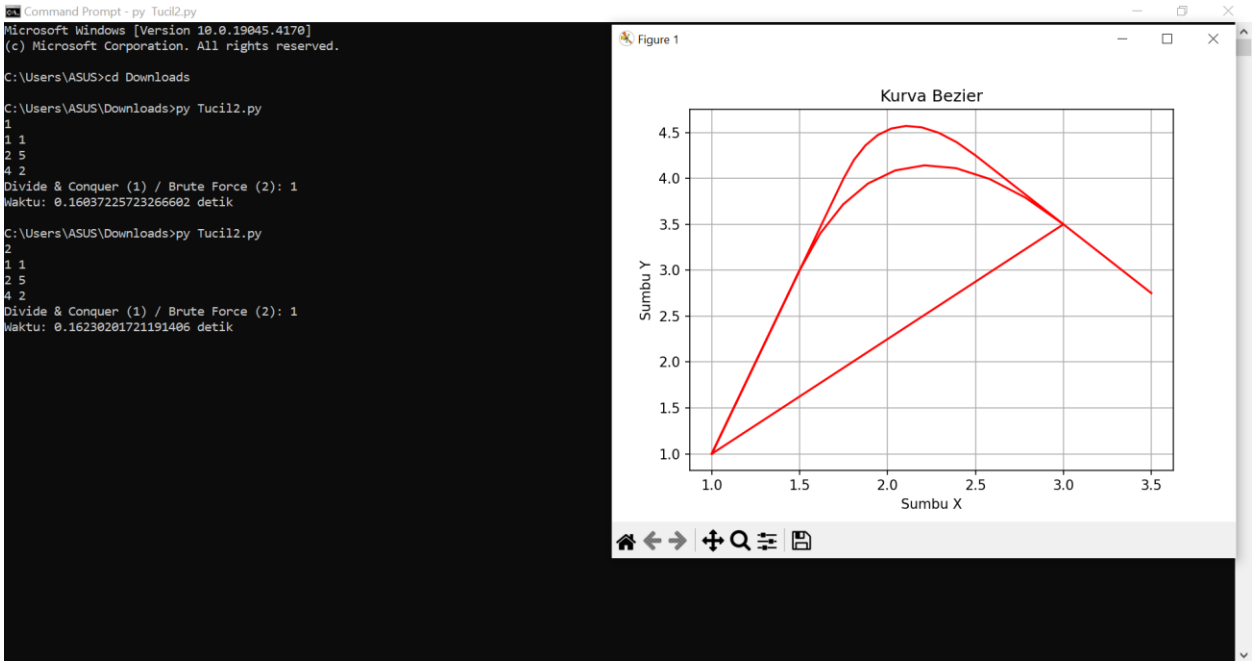
```
        new_control_points = [control_points[0]]
        for i in range(len(control_points) - 1):
            # Titik kontrol awal
            new_control_points.append(control_points[i])
            # Perhitungan titik tengah antara titik kontrol
            new_point = ((control_points[i][0] + control_points[i + 1][0]) / 2,
                         (control_points[i][1] + control_points[i + 1][1]) / 2)
            new_control_points.append(new_point)
        # Titik kontrol akhir
        new_control_points.append(control_points[-1])

        control_points = new_control_points.copy()

        # Evaluasi kurva Bezier antara dua titik kontrol secara berurutan
        for i in range(0, len(new_control_points) - 3, 2):
            points = new_control_points[i:i+3]
            for t in np.linspace(0, 1, 10):
                point = evaluate_bezier_segment(points, t)
                bezier_points.append(point)

    # Plot kurva Bezier
    x_values = [point[0] for point in bezier_points]
    y_values = [point[1] for point in bezier_points]
    plt.plot(x_values, y_values, 'r-')

# Fungsi Brute Force
def bezierBruteForce(control_points, iterations):
    # Inisialisasi array posisi titik kurva Bezier
    bezier_points = []

    for _ in range(iterations):
        new_control_points = [control_points[0]]
        for i in range(len(control_points) - 1):
            # Tambahkan titik kontrol baru di tengah di tiap segmen
            new_point = ((control_points[i][0] + control_points[i + 1][0]) / 2,
(control_points[i][1] + control_points[i + 1][1]) / 2)
            new_control_points.append(new_point)
            new_control_points.append(control_points[i + 1])
        control_points = new_control_points.copy()

    # Plot kurva Bezier
    for i in range(0, len(control_points) - 2, 2):
        # Mengambil tiap segmen dari tiga titik kontrol secara berurutan
        points = control_points[i:i+3]
        bezier_points = []
```

```
        for t in np.linspace(0, 1, 1000):
            point = evaluate_bezier_segment(points, t)
            bezier_points.append(point)
        x_values = [point[0] for point in bezier_points]
        y_values = [point[1] for point in bezier_points]
        plt.plot(x_values, y_values, 'b-')

# Input titik kontrol dan jumlah iterasi
iterations = int(input())
control_points = []

for i in range(3):
    x, y = map(int, input().split())
    control_points.append((x , y))

# Input pemilihan algoritma
j = int(input("Divide & Conquer (1) / Brute Force (2): " ))

# Pemrosesan tiap algoritma serta waktu
if (j == 1):
    start_time = time.time()
    bezierDivideConquer(control_points, iterations)
    end_time = time.time()
else:
    start_time = time.time()
    bezierBruteForce(control_points, iterations)
    end_time = time.time()

end_time = time.time()
execution_time = end_time - start_time

# Output
print("Waktu:", execution_time, "detik")
plt.title('Kurva Bezier')
plt.xlabel('Sumbu X')
plt.ylabel('Sumbu Y')
plt.grid(True)
plt.show()
```

3.2. Tangkapan Layar

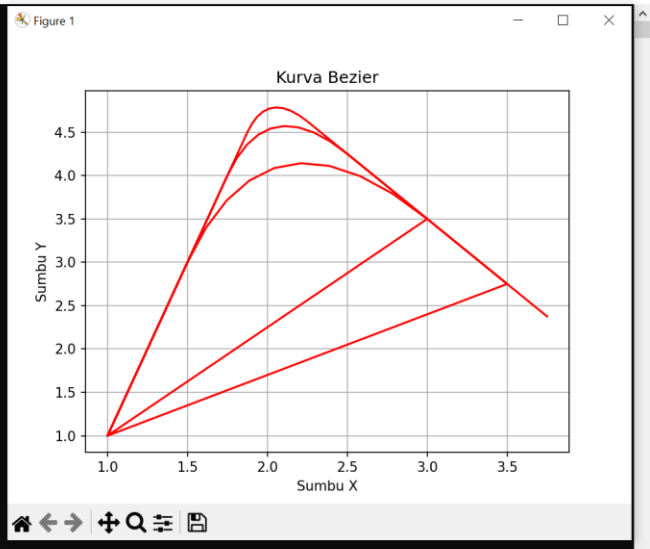
3.2.1. Algoritma Divide and Conquer





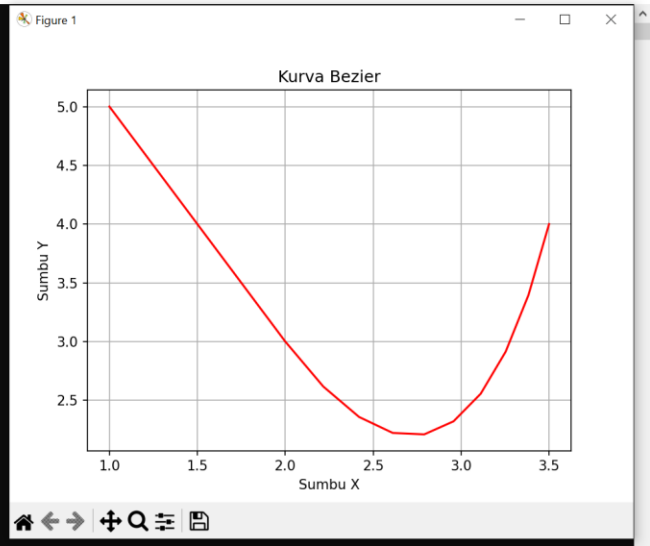
Command Prompt - py Tucil2.py

```
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS>cd Downloads

C:\Users\ASUS\Downloads>py Tucil2.py
1
1 1
2 5
4 2
Divide & Conquer (1) / Brute Force (2): 1
Waktu: 0.16037225723266602 detik

C:\Users\ASUS\Downloads>py Tucil2.py
2
1 1
2 5
4 2
Divide & Conquer (1) / Brute Force (2): 1
Waktu: 0.16230201721191406 detik

C:\Users\ASUS\Downloads>py Tucil2.py
3
1 1
2 5
4 2
Divide & Conquer (1) / Brute Force (2): 1
Waktu: 0.16422462463378906 detik
```



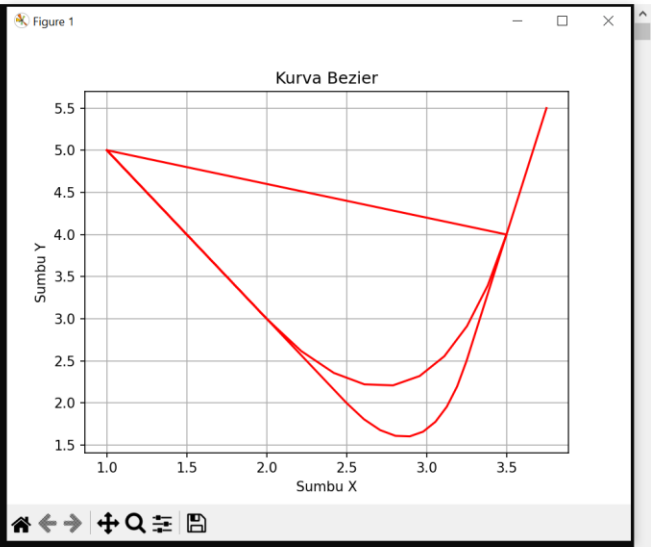
Command Prompt - py Tucil2.py

```
C:\Users\ASUS\Downloads>py Tucil2.py
1
1 5
3 1
4 7
Divide & Conquer (1) / Brute Force (2): 1
Waktu: 0.1693587303161621 detik
```



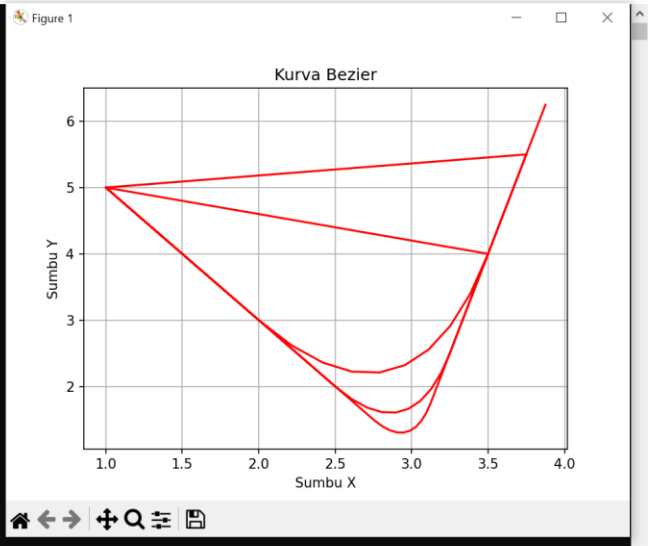
Command Prompt - py Tucil2.py

```
C:\Users\ASUS\Downloads>py Tucil2.py
1
1 5
3 1
4 7
Divide & Conquer (1) / Brute Force (2): 1
Waktu: 0.1693587303161621 detik

C:\Users\ASUS\Downloads>py Tucil2.py
2
1 5
3 1
4 7
Divide & Conquer (1) / Brute Force (2): 1
Waktu: 0.15619754791259766 detik
```

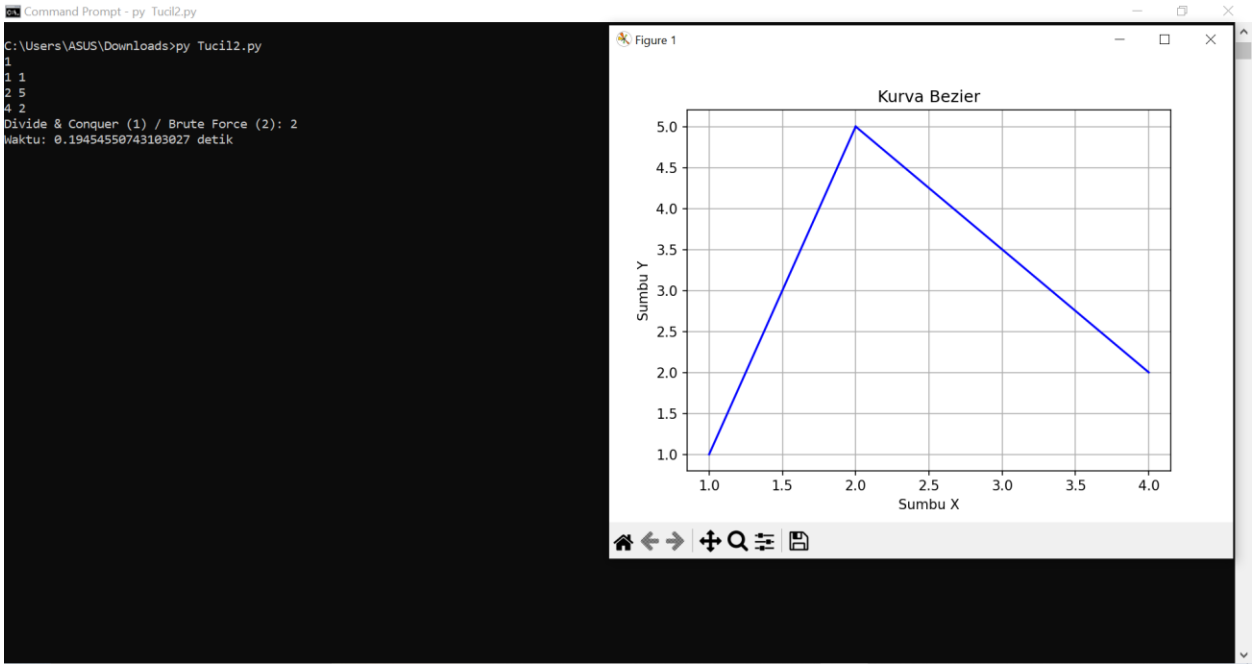


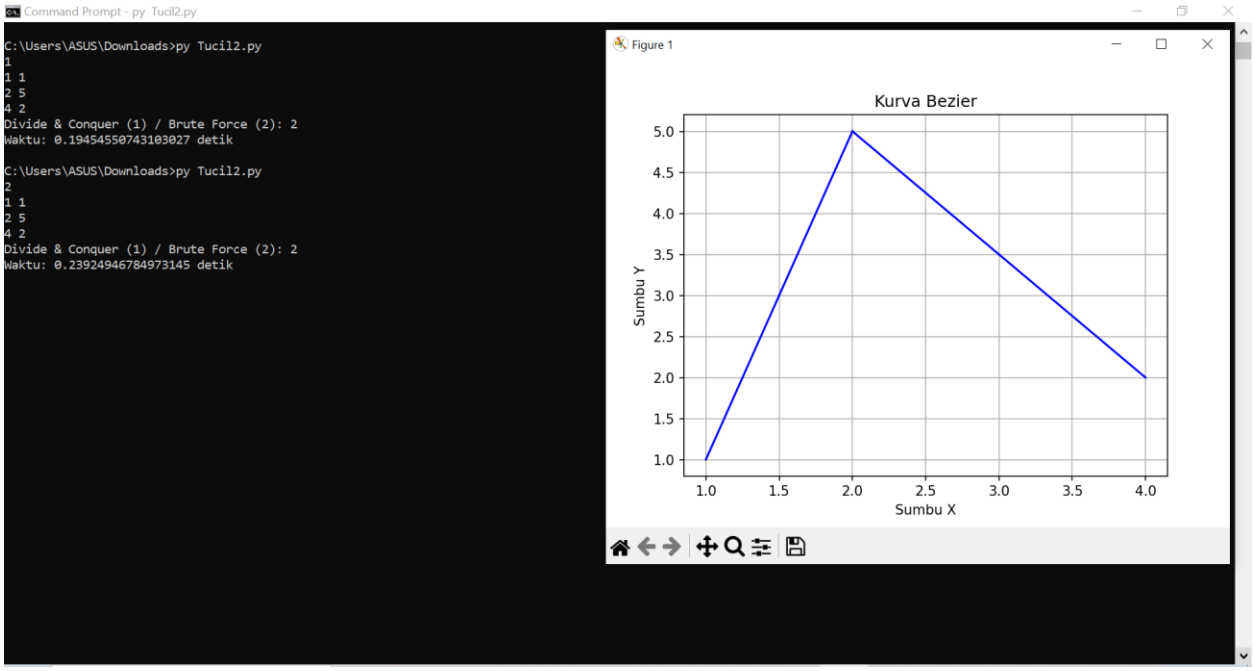
Command Prompt - py Tucil2.py

```
C:\Users\ASUS\Downloads>py Tucil2.py
1
1 5
3 1
4 7
Divide & Conquer (1) / Brute Force (2): 1
Waktu: 0.1693587303161621 detik

C:\Users\ASUS\Downloads>py Tucil2.py
2
1 5
3 1
4 7
Divide & Conquer (1) / Brute Force (2): 1
Waktu: 0.15619754791259766 detik

C:\Users\ASUS\Downloads>py Tucil2.py
3
1 5
3 1
4 7
Divide & Conquer (1) / Brute Force (2): 1
Waktu: 0.1520454883754395 detik
```



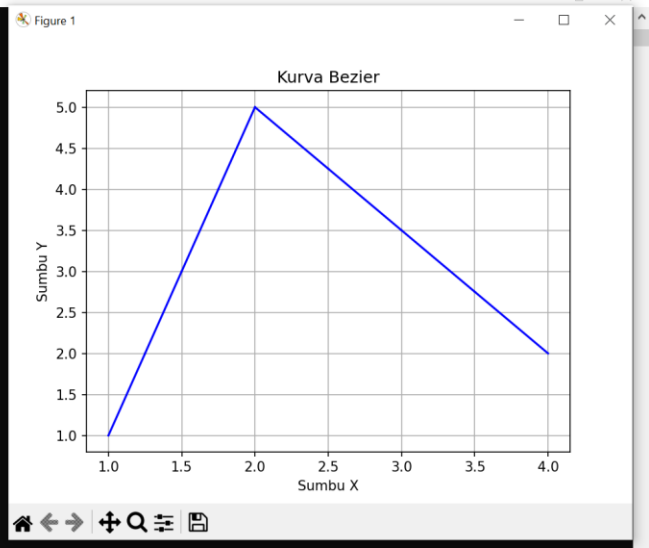
3.2.2. Algoritma Brute Force





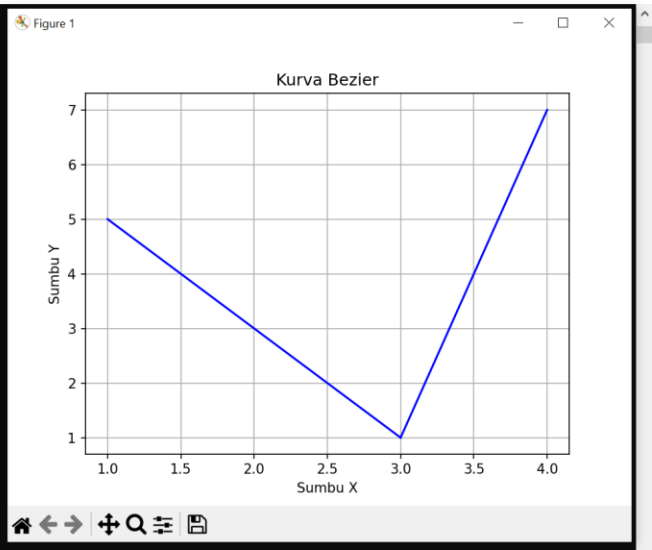
Command Prompt - py Tucil2.py

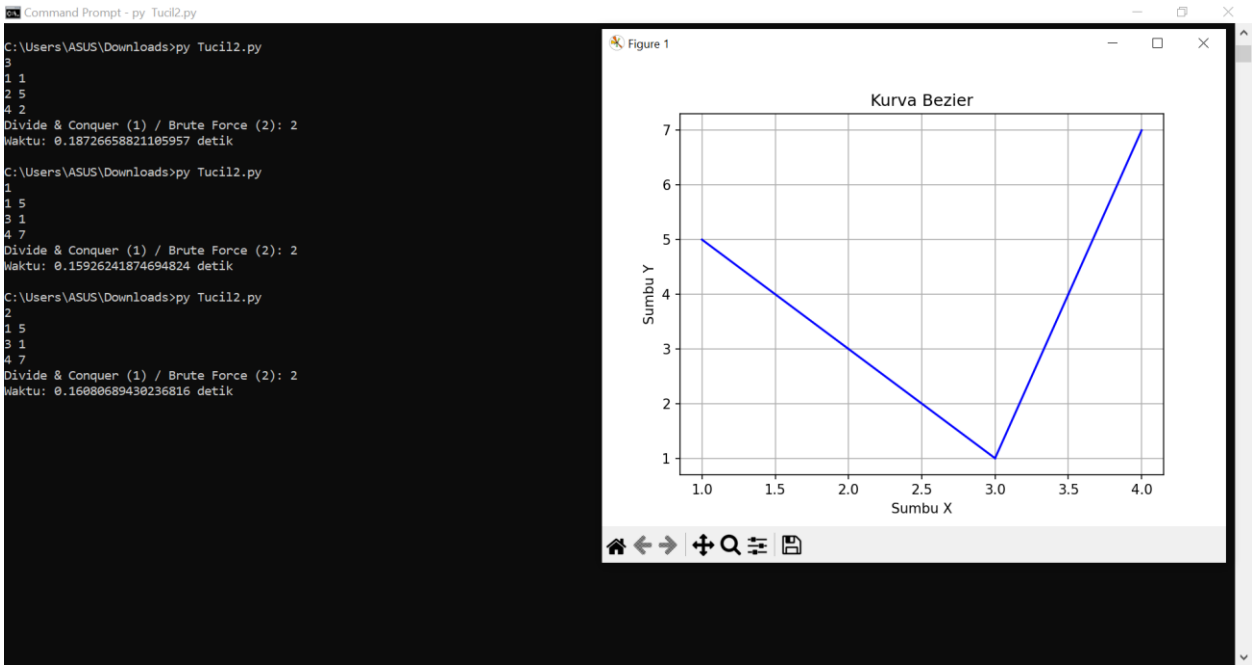
```
C:\Users\ASUS\Downloads>py Tucil2.py
3
1 1
2 5
4 2
Divide & Conquer (1) / Brute Force (2): 2
Waktu: 0.18726658821105957 detik
```

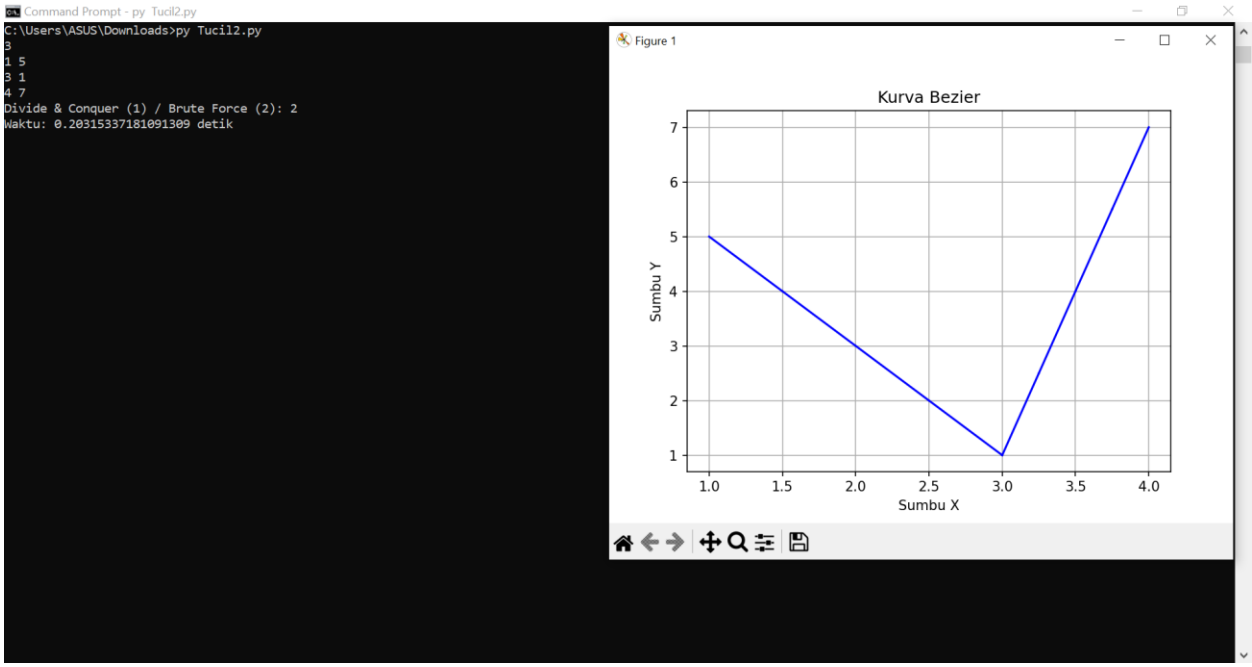


Command Prompt - py Tucil2.py

```
C:\Users\ASUS\Downloads>py Tucil2.py
3
1 1
2 5
4 2
Divide & Conquer (1) / Brute Force (2): 2
Waktu: 0.18726658821105957 detik

C:\Users\ASUS\Downloads>py Tucil2.py
1
1 5
3 1
4 7
Divide & Conquer (1) / Brute Force (2): 2
Waktu: 0.15926241874694824 detik
```







3.3. Analisis Perbandingan

Program ini menggunakan dua jenis algoritma yaitu algoritma *brute force* dan algoritma *divide and conquer*. Kedua algoritma ini memiliki perbedaan-perbedaan sehingga dapat menghasilkan kinerja yang mungkin berbeda. Secara teori, penggunaan algoritma *divide and conquer* memiliki kinerja yang lebih baik dibanding algoritma *brute*

force. Hal ini karena cara kerja algoritma *divide and conquer* dibanding algoritma *brute force*. Algoritma *divide and conquer* membagi kurva menjadi segmen-segmen yang lebih kecil serta memproses nilai-nilai pada setiap segmen tersebut. Sementara itu, algoritma *brute force* membagi setiap segmen secara berulang sehingga menghasilkan pengulangan yang lebih banyak dan menambah waktu pemrosesan.

Pada kenyataan, analisis tidak bisa dilakukan dengan baik karena kesalahan *programmer* dalam membuat koding yang sesuai sehingga menghasilkan hasil yang memuaskan. Kesalahan tersebut berupa kesalahan penggambaran untuk kedua algoritma, terutama algoritma *brute force* yang hanya berbentuk garis lurus yang menghubungkan dua titik kontrol berbeda. Meskipun hasil yang kurang memuaskan, masih didapat penghitungan waktu yang dilakukan oleh program. Berdasarkan dari hasil tangkapan layar, penggunaan algoritma *brute force* memiliki kecenderungan untuk menggunakan waktu yang sedikit lebih lama dibanding penggunaan algoritma *divide and conquer*. Meskipun itu, hasil yang kurang sesuai membuka kemungkinan bahwa hasil perbandingan tersebut diakibatkan karena ketidakmampuan *programmer*, bukan karena perbedaan dari kedua algoritma tersebut.

3.4. Repository

https://github.com/Hapish/Tucil2_13522132