

LAPORAN TUGAS KECIL

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

*Disajikan untuk memenuhi salah satu tugas kecil Mata Kuliah IF2211 Strategi Algoritma
yang diampu oleh:*

Dr. Nur Ulfa Maulidevi, S.T., M.Sc.



Disusun oleh:

Nelsen Putra (13520130)

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

2024

A. Analisis dan Implementasi Algoritma *Brute Force*

Algoritma *brute force* adalah penyelesaian suatu masalah dengan pendekatan yang sederhana, langsung, jelas, dan mudah dipahami. Dengan algoritma *brute force*, penyelesaian suatu masalah cenderung dilakukan dengan mencari dan meninjau semua kasus yang ada. Dengan demikian, algoritma *brute force* dapat dikatakan sebagai algoritma yang lempang (*straight-forward*). Kelebihan dari algoritma ini adalah cukup mudah untuk dipahami dan dapat diterapkan untuk hampir semua permasalahan komputasi. Akan tetapi, kekurangan dari algoritma ini adalah tidak efisien karena membutuhkan langkah yang banyak dalam penyelesaiannya sehingga membutuhkan waktu yang relatif lama dibandingkan dengan kebanyakan algoritma lainnya.

Dalam Tugas Kecil 2 IF2211 Strategi Algoritma ini, mahasiswa diminta untuk mengimplementasikan algoritma *brute force* dalam membuat kurva Bezier yang nantinya akan digunakan sebagai pembanding terhadap pembuatan kurva Bezier menggunakan algoritma *divide and conquer*. Sebelum menerapkan algoritma *Brute Force* dalam membuat kurva Bezier, program pertama – tama akan menerima dan membaca *input* dari *user* berupa sebuah *file .txt*. File tersebut berisi input data dalam bentuk format seperti berikut ini.

```
number_of_points (integer larger than 1)
point_1 (real, format: x y)
point_2
...
point_N
number_of_iterations (integer larger than 0)
cartesian_plane_size_range (format: x1 x2 y1 y2, x1 < x2 and y1 < y2)
```

Kurva Bezier dapat didefinisikan sebagai bentuk polinomial Bernstein, yaitu sebagai berikut.

$$Bézier(n, t) = \sum_{i=0}^n \underbrace{\binom{n}{i}}_{\text{binomial term}} \cdot \underbrace{\frac{(1-t)^{n-i} \cdot t^i}{\text{polynomial term}}}_{\text{polynomial term}} \cdot \underbrace{w_i}_{\text{weight}}$$

Pada kasus ini, w_i adalah titik kontrol ke- i dari sebuah kurva bezier, dengan w_0 adalah titik mulai dan w_n adalah titik akhir. Sebagai contoh, untuk menghitung Kurva Bezier yang mulai pada titik (110,150), titik kontrol tengah (25,190) dan (210,250), serta berakhir di titik (210,30), kurva Bezier dihitung sebagai berikut.

$$\begin{cases} x = 110 \cdot (1-t)^3 + 25 \cdot 3 \cdot (1-t)^2 \cdot t + 210 \cdot 3 \cdot (1-t) \cdot t^2 + 210 \cdot t^3 \\ y = 150 \cdot (1-t)^3 + 190 \cdot 3 \cdot (1-t)^2 \cdot t + 250 \cdot 3 \cdot (1-t) \cdot t^2 + 30 \cdot t^3 \end{cases}$$

Range nilai t adalah $[0, 1]$, dan setiap nilai t tersebut akan menghasilkan titik yang unik pada kurva Bezier.

B. Analisis dan Implementasi Algoritma *Divide and Conquer*

Algoritma Divide and Conquer adalah teknik algoritma yang digunakan dalam pemrograman komputer untuk memecahkan masalah yang kompleks menjadi beberapa bagian yang lebih kecil dan lebih mudah dipecahkan secara terpisah. Konsep dasarnya adalah memecahkan masalah besar menjadi beberapa sub-masalah yang lebih kecil, menyelesaikan setiap sub-masalah secara terpisah dan kemudian menggabungkan solusi sub-masalah menjadi solusi akhir untuk masalah yang lebih besar. Cara kerja dari algoritma ini terbagi menjadi 3 bagian, yakni *divide*, *conquer*, dan *combine*. Pada tahap *divide*, masalah besar dibagi menjadi dua atau lebih sub-masalah yang lebih kecil dan serupa dengan masalah asli. Proses pembagian ini dilakukan sampai tidak dapat dibagi lagi atau sampai ukuran sub-masalah sudah cukup kecil untuk dapat dipecahkan dengan mudah. Setelah masalah dibagi menjadi beberapa sub-masalah yang lebih kecil, langkah berikutnya ialah tahap *conquer*, yakni memecahkan setiap sub-masalah secara terpisah. Pemecahan ini dapat dilakukan dengan menggunakan algoritma yang sama atau dengan algoritma yang berbeda, tergantung pada masalah yang dihadapi. Pada akhirnya, memasuki tahap *combine*, solusi untuk setiap sub-masalah akan digabungkan untuk menghasilkan solusi akhir yang dapat menyelesaikan masalah besar. Proses penggabungan ini juga disebut sebagai *conquering back* atau *merging*.

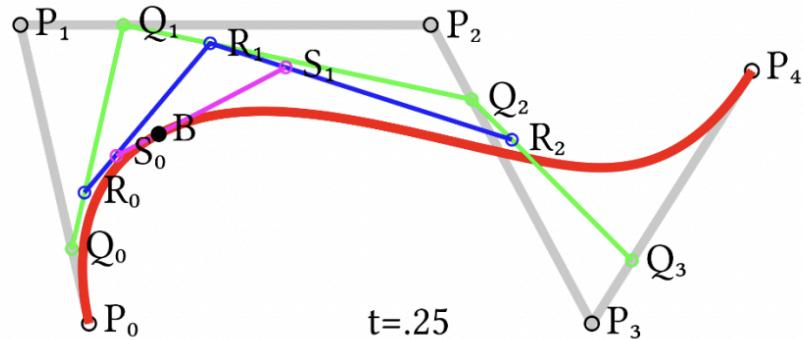
Dalam Tugas Kecil 2 IF2211 Strategi Algoritma ini, mahasiswa diminta untuk mengimplementasikan algoritma titik tengah berbasis *divide and conquer* dalam membuat kurva Bezier. Serupa dengan yang terjadi pada algoritma *Brute Force*, sebelum menerapkan algoritma *Divide and Conquer* dalam membuat kurva Bezier, program pertama – tama akan menerima dan membaca *input* dari *user* berupa sebuah *file .txt*. File tersebut berisi input data dalam bentuk format seperti berikut ini.

```
number_of_points (integer larger than 1)
point_1 (real, format: x y)
point_2
...
point_N
number_of_iterations (integer larger than 0)
cartesian_plane_size_range (format: x1 x2 y1 y2, x1 < x2 and y1 < y2)
```

Titik-titik pada kurva Bezier dapat dihitung dan digambar menggunakan algoritma de Casteljau. Algoritma ini bersifat rekursif sehingga implementasinya dapat dilakukan menggunakan pendekatan divide and conquer. Berikut langkah-langkah mendapatkan titik titik tersebut menurut algoritma ini.

1. Ambil semua garis di antara titik-titik penentu kurva. Terdapat n garis untuk kurva berorde n.
2. Ambil titik tengah untuk setiap garis, pada jarak t. Pada implementasi divide and conquer, diambil nilai t paling tengah, yaitu 0.5.
3. Tarik garis untuk setiap titik ke-i dan titik ke-(i+1) (setiap dua titik). Ini menghasilkan n-1 garis.
4. Ambil titik tengah untuk setiap garis yang terbentuk.
5. Bentuklah garis-garis di antara titik-titik tersebut. Ini akan menjadi n-2 baris.

6. Ambil titik tengah, bentuk garis, ambil titik tengah, bentuk garis, dst.
7. Ulangi ini sampai tersisa satu titik. Titik ini, sebut B, adalah titik tengah kurva Bezier.
8. Kemudian tarik garis dari titik awal ke titik B, kemudian dari titik B ke titik akhir.
9. Untuk iterasi berikutnya, lakukan proses yang sama untuk mendapatkan upakurva Bezier dari titik awal ke titik B serta dari titik B ke titik akhir. Titik kontrol masing-masing upakurva tersebut adalah, secara berturut-turut kumpulan titik tengah terkiri dan titik tengah terkanan. Berikut ilustrasi untuk memperjelas.



Pada gambar di atas, sebuah kurva Bezier orde 4 dibentuk dengan 5 titik. beziercurve(P_0, P_1, P_2, P_3, P_4) adalah hasil penggabungan beziercurve(P_0, Q_0, R_0, S_0, B) dan beziercurve (B, S_1, R_2, Q_3, P_4). Dengan ini, pada masing-masing upakurva, dapat dilakukan lagi langkah-langkah yang sudah disebutkan sehingga jelas bahwa dapat digunakan konsep divide and conquer dalam implementasi algoritme.

C. Source Code Program

Source code program *Bézier Curve Generator* ditulis dalam bahasa Python dengan isi kode sebagai berikut.

1. File timeFunction.py

```
from functools import wraps
from time import perf_counter

def timeHere(func):
    def wrapFunc(*args, **kwargs):
        t1 = perf_counter()
        result = func(*args, **kwargs)
        t2 = perf_counter()
        print(f'\nExecution time: {1000*(t2-t1):.2f}ms')
        return result
    return wrapFunc
```

2. File bezierDivideAndConquer.py

```

from dataclasses import dataclass
from module.timeFunc import timeHere

@dataclass
class Point :

    x : float
    y : float

    def __add__(self, other) :
        return Point(self.x + other.x, self.y + other.y)

    def __sub__(self, other) :
        return Point(self.x - other.x, self.y - other.y)

    def __mul__(self, k: float) :
        return Point(self.x * k, self.y * k)

    def __truediv__(self, k: float) :
        return Point(self.x / k, self.y / k)

    def __str__(self) :
        return f"({self.x}, {self.y})"

def midPoint(p1: Point, p2: Point) -> Point :
    return (p1 + p2) / 2

def weightedPoint(p1: Point, p2: Point, t: float) -> Point :
    return (p1 * t) + (p2 * (1 - t))

def quadraticBezier(p0, p2, p1, iter: int) -> list[Point]:
    # p2 is control point

    if iter == 0 :
        return [p0,p2,p1]

    elif iter == 1 :
        q0 = midPoint(p0, p2)
        q1 = midPoint(p2, p1)
        r0 = midPoint(q0, q1)

```

```

        return [p0,r0,p1]

    else :
        p02 = midPoint(p0, p2)
        p21 = midPoint(p2, p1)
        r0 = midPoint(p02, p21)

        # divide and conquer
        l = (quadraticBezier(p0, p02, r0, iter - 1))[:-1]
        m = [r0]
        r = (quadraticBezier(r0, p21, p1, iter - 1))[1:]

        return l + m + r

def nDegreeBezier(points: list[Point], iter: int) -> list[Point] :

    if iter == 0 :
        return points

    elif iter == 1 :

        prevPoints = points.copy()
        for i in range(len(points) - 1) :
            newPoints = list()
            for j in range(1, len(prevPoints)) :
                newPoints.append(midPoint(prevPoints[j],
prevPoints[j-1]))
            prevPoints = newPoints

        p0 = points[0]
        r0 = prevPoints[0]
        p1 = points[-1]

        return [p0,r0,p1]

    else :

        p0 = points[0]
        p1 = points[-1]

        leftPoints = [p0]

```

```

rightPoints = [p1]

# Process all bezier points up to nth degree
prevPoints = points.copy()
for i in range(len(points) - 1) :
    newPoints = list()
    for j in range(1, len(prevPoints)) :
        newPoints.append(midPoint(prevPoints[j],
prevPoints[j-1]))

    leftPoints.append(newPoints[0])
    rightPoints.append(newPoints[-1])

    prevPoints = newPoints

rightPoints.reverse()
r0 = prevPoints[0]

# divide and conquer
l = nDegreeBezier(leftPoints, iter - 1)[-1]
m = [r0]
r = nDegreeBezier(rightPoints, iter - 1)[1:]

return l + m + r

# nDegreeBezier is a recursive function so we need a helper to time it
from the beginning to the end
@timeHere
def helper(points: list[Point], iter: int) :
    return nDegreeBezier(points, iter)

```

3. File bezierBruteForce.py

```

from module.bezierDivideAndConquer import Point
from module.inputAndOutput import showGraph
from math import comb
from module.timeFunc import timeHere

@timeHere

def nDegreeBruteForceBezier(points: list[Point], iter) :

```

```

n = len(points) - 1
diff = 1 / (2 ** iter)
t = 0
ans = list()

while t <= 0.999 :

    p = Point(0, 0)
    for i in range(n + 1) :

        p += points[i] * (pow(t, i)) * (pow(1 - t, n - i)) * comb(n,
i)

    ans.append(p)

    t += diff

return ans

if __name__ == "__main__":
    p0 = Point(0,0)
    p2 = Point(20,30)
    p3 = Point(0,30)
    p1 = Point(30,0)
    points = [p0,p2,p3,p1]
    points = nDegreeBruteForceBezier(points)
    showGraph(points, zf = 30)

```

4. File inputAndOutput.py

```

from module.bezierDivideAndConquer import Point
import matplotlib.pyplot as plt
import os
import pyfiglet

def ui() -> str :

    title = pyfiglet.figlet_format("Bezier Curve Generator", font="big")
    print(title)

    print("=====")

```

```

    print("Hello, this program will help you to generate Bézier
curves!")

print("====")
print()
print("Let's have some fun! What are you gonna do?")
print("1. Start using the Bézier Curve Generator")
print("2. Help")

choice = input("(1/2): ")
if not choice : return 1

try :
    choice = int(choice)
    if choice not in [1,2] :
        raise ValueError
    else :
        return choice

except ValueError:
    print("Please input correctly.")
    quit()

def txtInput() -> list[Point] :

    fileName = input("Input filename (you can leave this blank to
default to \"input.txt\"): ")
    fileName = fileName if fileName else "input.txt"
    error = ""
    try :
        if not os.path.isfile(fileName) :
            filePath = os.path.join("Tucil2_13520130/test", fileName)

        with open(filePath, 'r') as f :

            points = list()

            p = f.readline().split()

            if len(p) != 1 :
                error = "Wrong format at line 1. Make sure the amount of
                "

```

```

points is a positive integer larger than 1."
    raise ValueError

try :
    p = int(p[0])
except ValueError :
    error = "Wrong format at line 1. Make sure you have put
in an integer."
    raise ValueError

for i in range(p) :

    line = f.readline().split()

    if len(line) != 2 :
        error = f"Wrong format at line {2 + i}. Make sure
you put in exactly two real numbers for each point."
        raise ValueError

    try :
        x,y = map(float, line)
        points.append(Point(x,y))
    except ValueError :
        error = f"Wrong format at line {2 + i}. Make sure
you have put in real values."
        raise ValueError

    t = f.readline().split()
    if len(t) != 1 :
        error = f"Wrong format at line {p + 2}. Make sure the
iteration count is a non negative integer."
        raise ValueError

    try :
        t = int(t[0])
    except ValueError :
        error = f"Wrong format at line {p + 2}. Make sure you
have put in a non negative integer."
        raise ValueError

x1,x2,y1,y2 = 0,0,0,0

```

```

        try :
            x1,x2,y1,y2 = map(float, f.readline().split())
            if x1 >= x2 or y1 >= y2 :
                raise ValueError

        except ValueError:
            error = f"Wrong format at line {p + 3}. Make sure the
last line contains 4 real numbers x1 x2 y1 y2, where x1 < x2 and y1 <
y2"
            raise ValueError

        return p, points, t, x1, x2, y1, y2

    except ValueError :
        error = "Failed to read from input file."
        print(error)
        print()
        quit()

    except FileNotFoundError :
        error = "Couldn't find the file. Please check your file name or
the directory of your file."
        print(error)
        print()
        quit()

def showGraph(points, x1, x2, y1, y2, controls) :

    x = [p.x for p in points]
    y = [p.y for p in points]

    plt.plot(x, y)
    plt.axis([x1,x2,y1,y2])

    x = [p.x for p in controls]
    y = [p.y for p in controls]
    plt.scatter(x,y, color = "red")
    plt.plot(x,y, color = 'black', alpha=0.1)

    plt.show()

```

```

def help() :
    print()
    print("How to use this Bézier Curve Generator?")
    print("1. In the menu section, you will be given the 2 options to choose. To start using this Bézier Curve Generator, you will have to enter 1.")
    print("2. Then, you will be prompted to enter a file name as an input to this program. The input file must be directly inside the src/module folder. You can actually leave this blank, in which case 'input.txt' will be used by default.")
    print("3. Press enter in the command line and the program will process to generate your Bézier curve based on your file input.")
    print("4. And... voila! Your Bézier curve is ready to serve!")
    print("5. Close the window to end the program.")
    print()
    quit()

```

5. File main.py

```

import os
import sys

moduledirectory =
os.path.abspath(os.path.join(os.path.dirname(__file__), 'module\\'))
testdirectory = os.path.abspath(os.path.join(os.path.curdir,
os.path.pardir))
sys.path.append(moduledirectory)

from module.bezierBruteForce import nDegreeBruteForceBezier
from module.bezierDivideAndConquer import nDegreeBezier, helper
from module.inputAndOutput import *

def cls() :
    os.system("cls" if os.name == "nt" else "clear")

def main() :

    # Initialize
    cls()
    points = []

```

```

# Input
choice = ui()
if choice == 1:
    p, points, t, x1, x2, y1, y2 = txtInput()

elif choice == 2:
    help()

# Solve
result = nDegreeBezier(points = points, iter = t)
execTime = helper(points = points, iter = t)

# For comparing the execution time of generating Bezier curve using
Brute Force and DnC
# result = nDegreeBruteForceBezier(points, iter = t)

# Output
print("Your Bézier Curve is ready!")
showGraph(result, x1, x2, y1, y2, controls=points)

main()

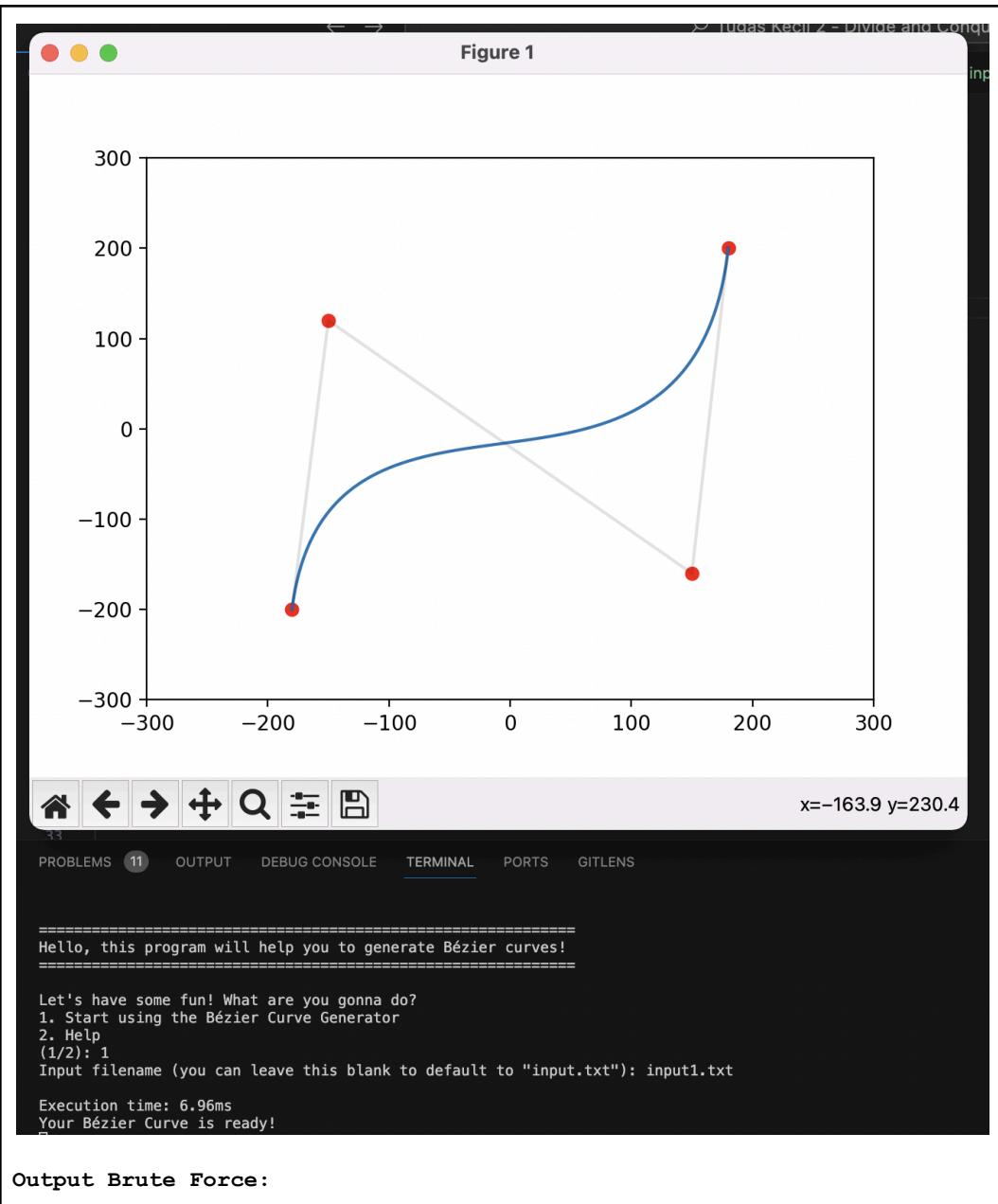
```

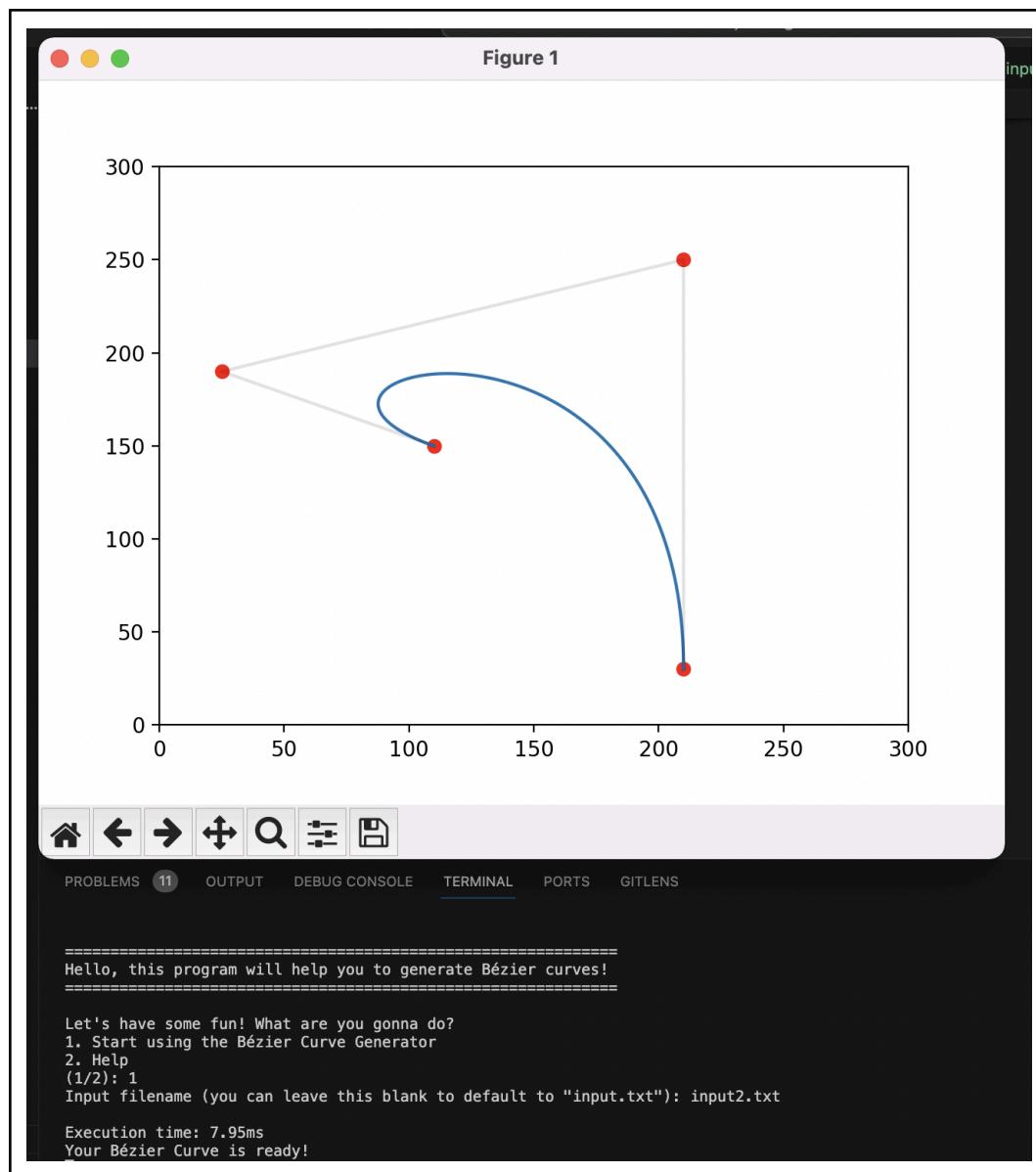
D. Screenshot Program (Input dan Output)

Berikut ini adalah hasil uji coba kode program berupa input yang digunakan dan output yang dihasilkan pada program.

- Uji Coba 1

| |
|-----------------------------------|
| Input: |
| 4 |
| -180 -200 |
| -150 120 |
| 150 -160 |
| 180 200 |
| 10 |
| -300 300 -300 300 |
| Output Divide and Conquer: |





2. Uji Coba 2

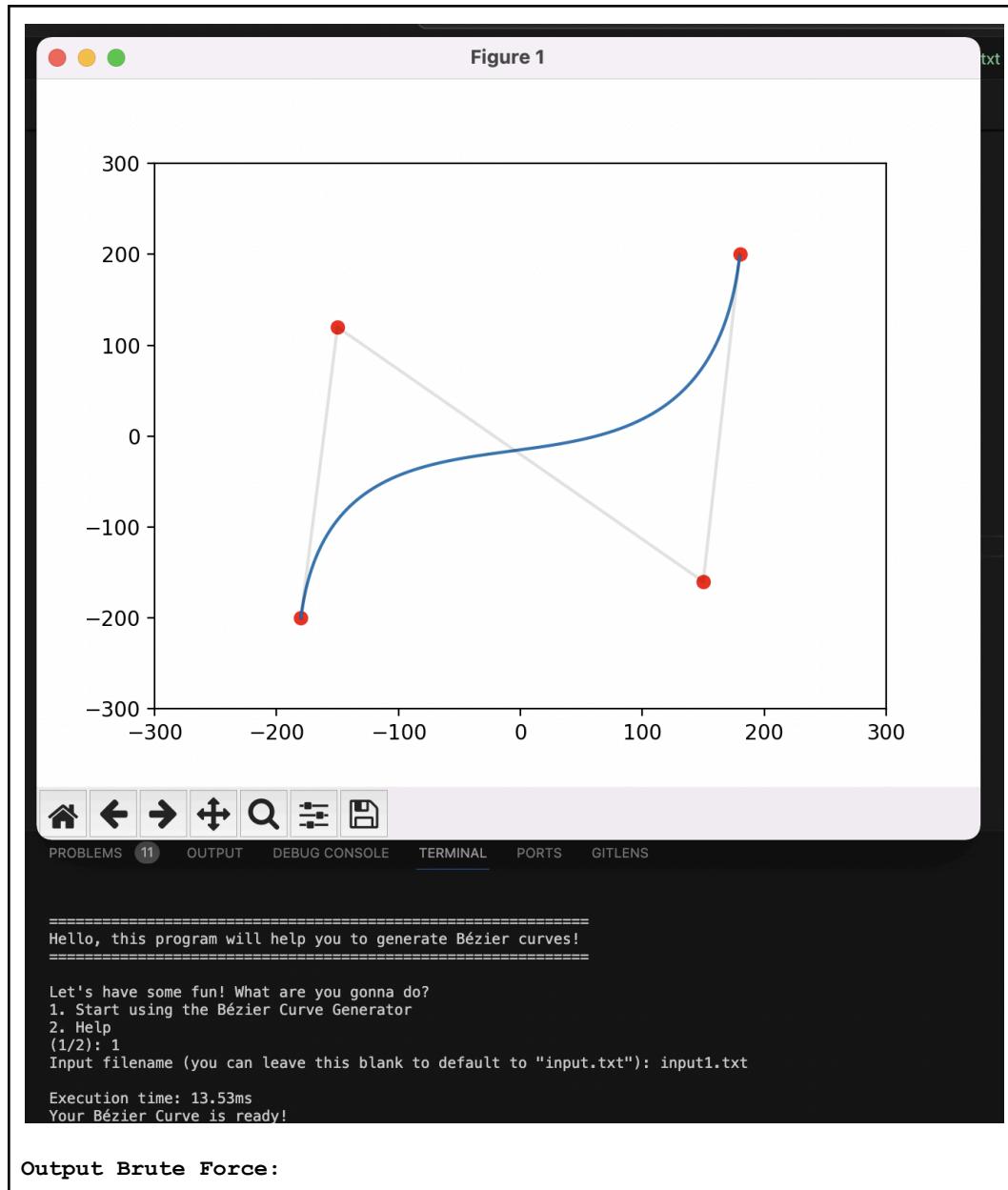
Input:

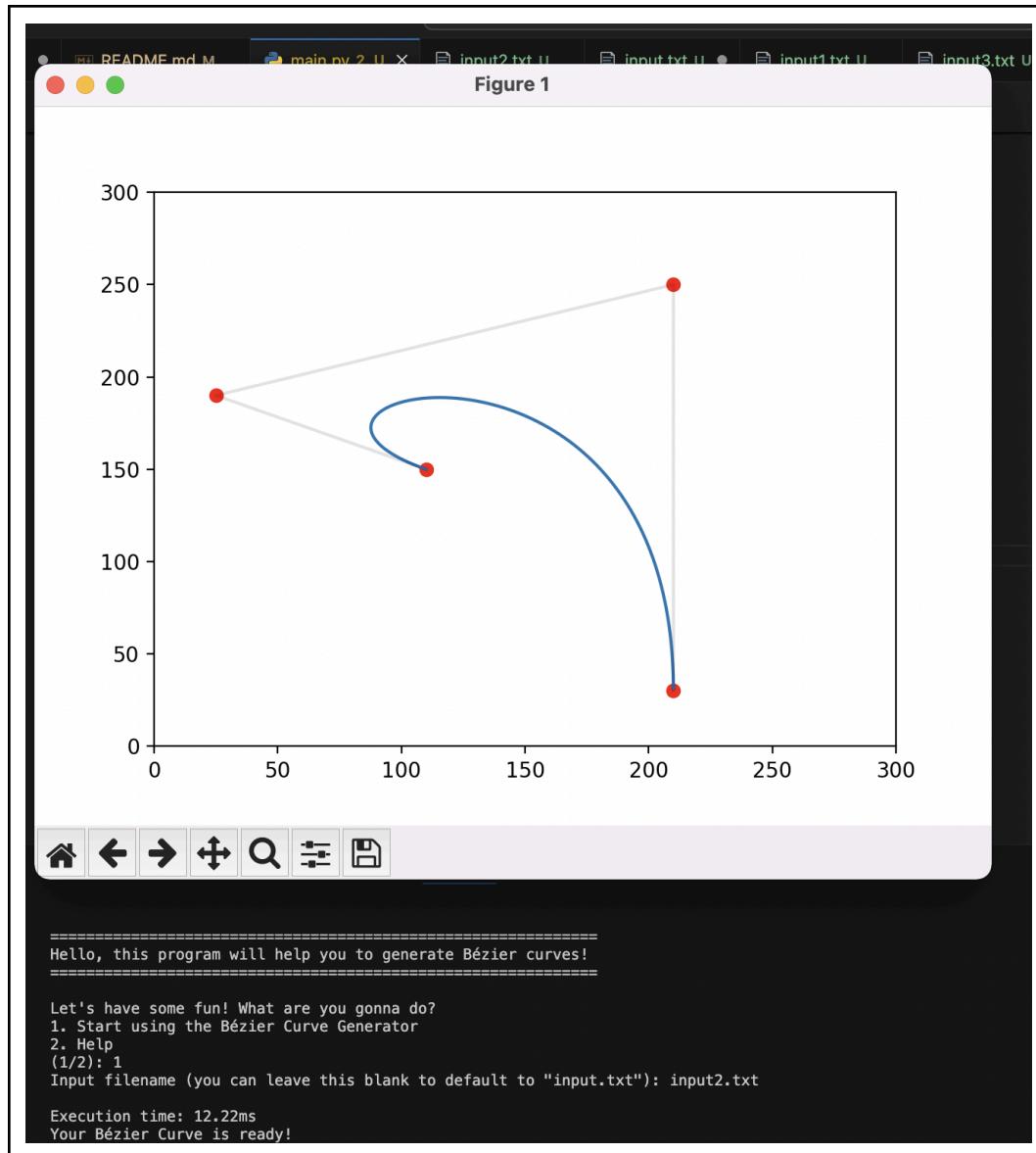
```

4
110 150
25 190
210 250
210 30
10
0 300 0 300

```

Output Divide and Conquer:

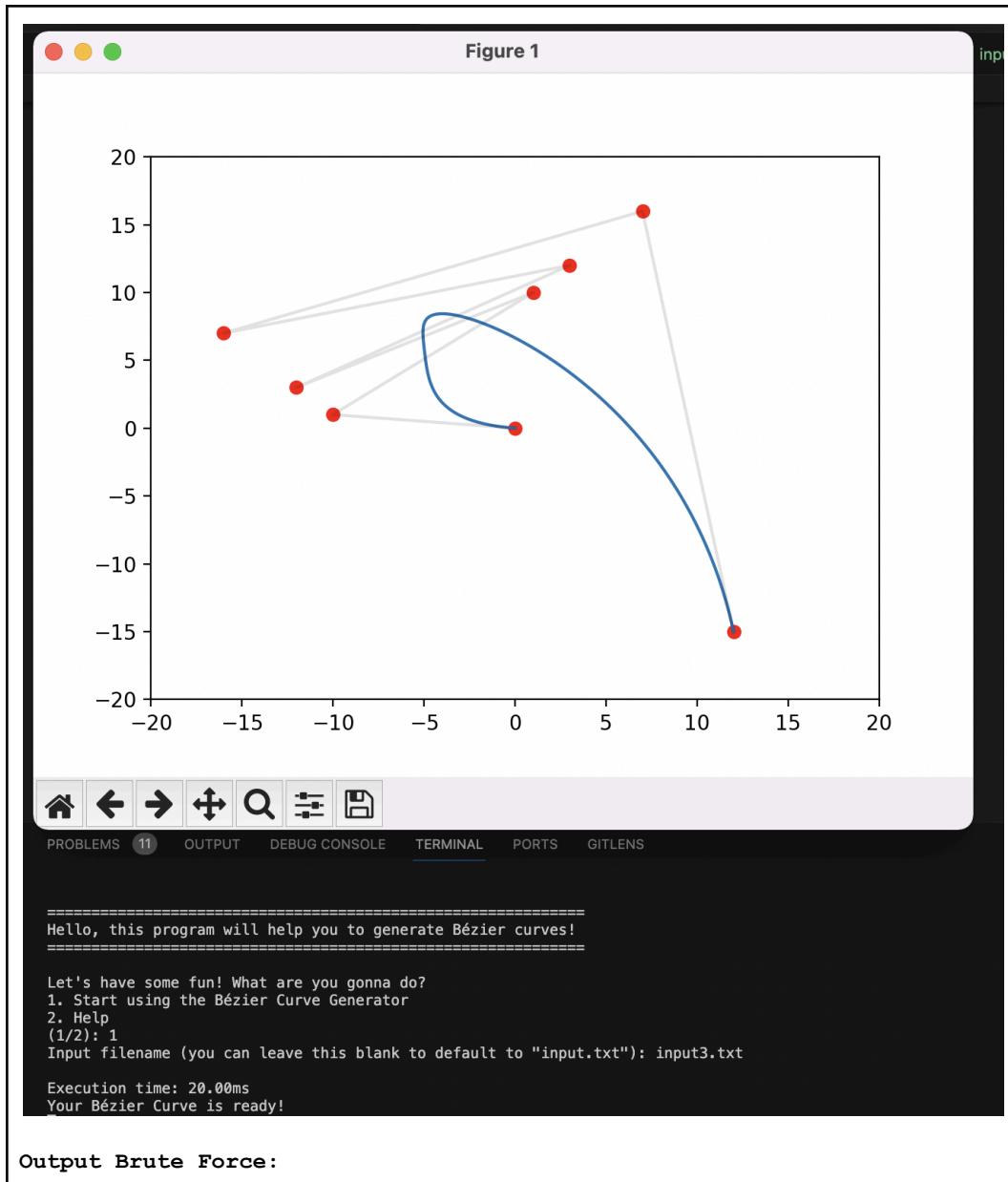


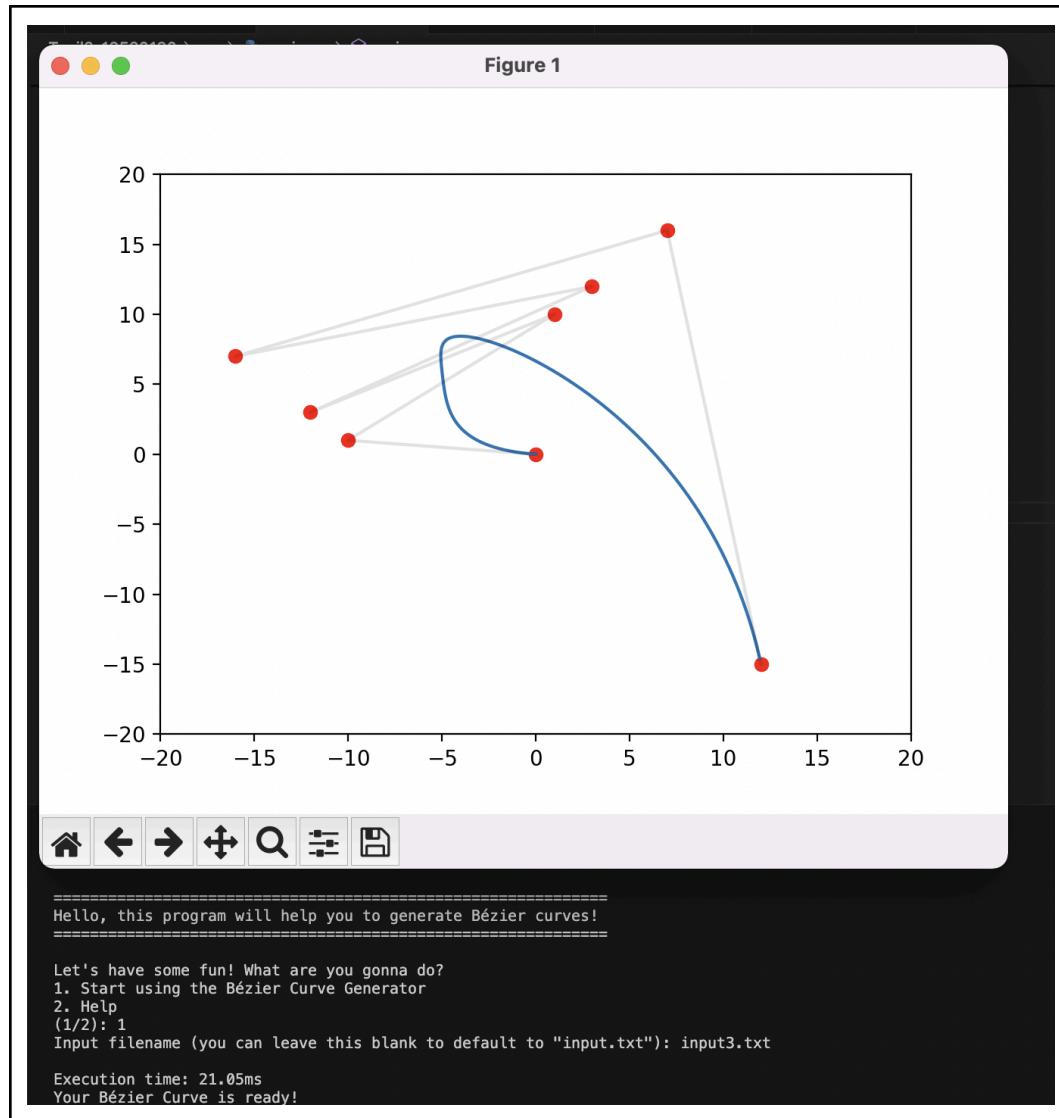


3. Uji Coba 3

```
Input:
8
0 0
-10 1
1 10
-12 3
3 12
-16 7
7 16
12 -15
10
-20 20 -20 20
```

```
Output Divide and Conquer:
```



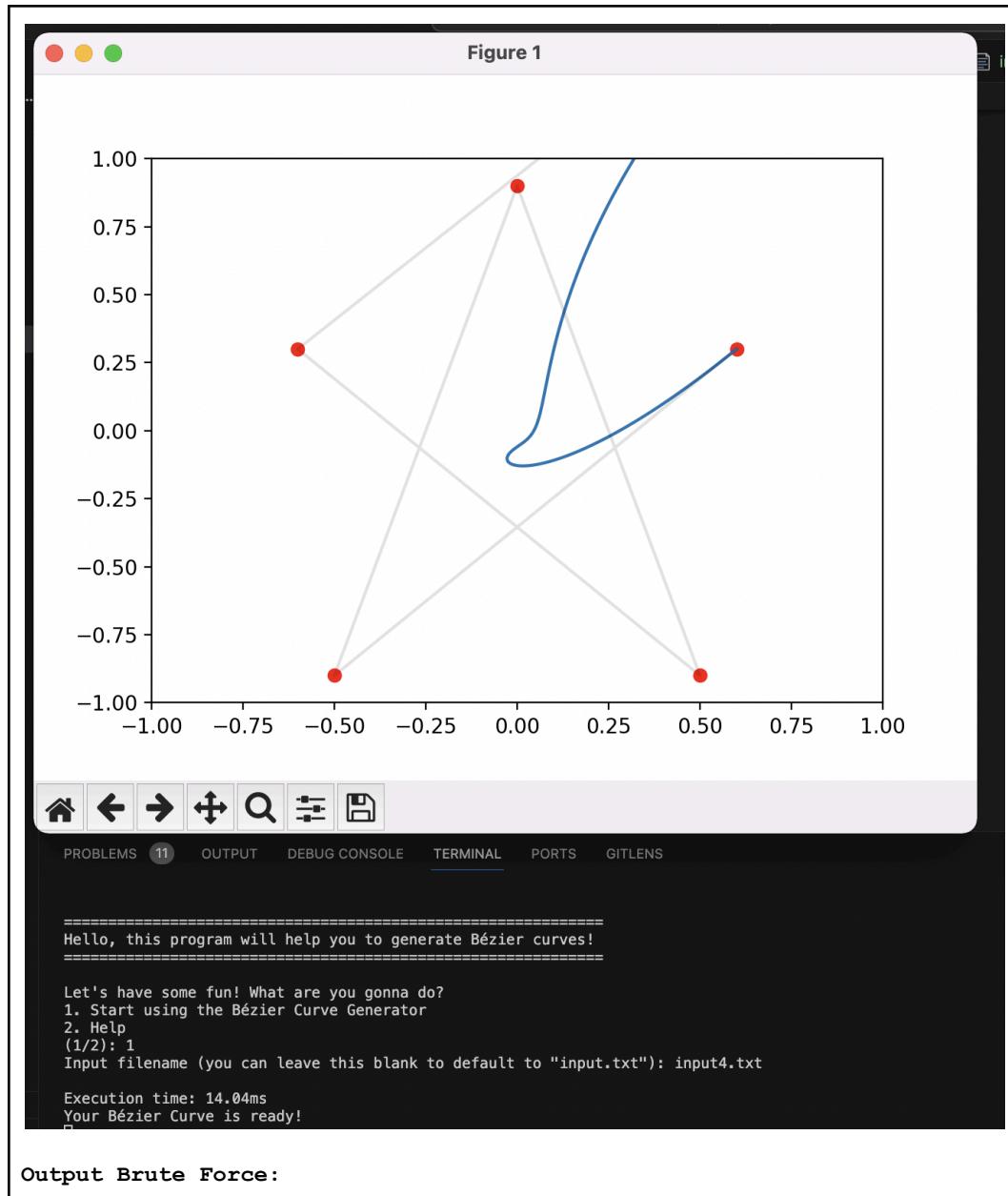


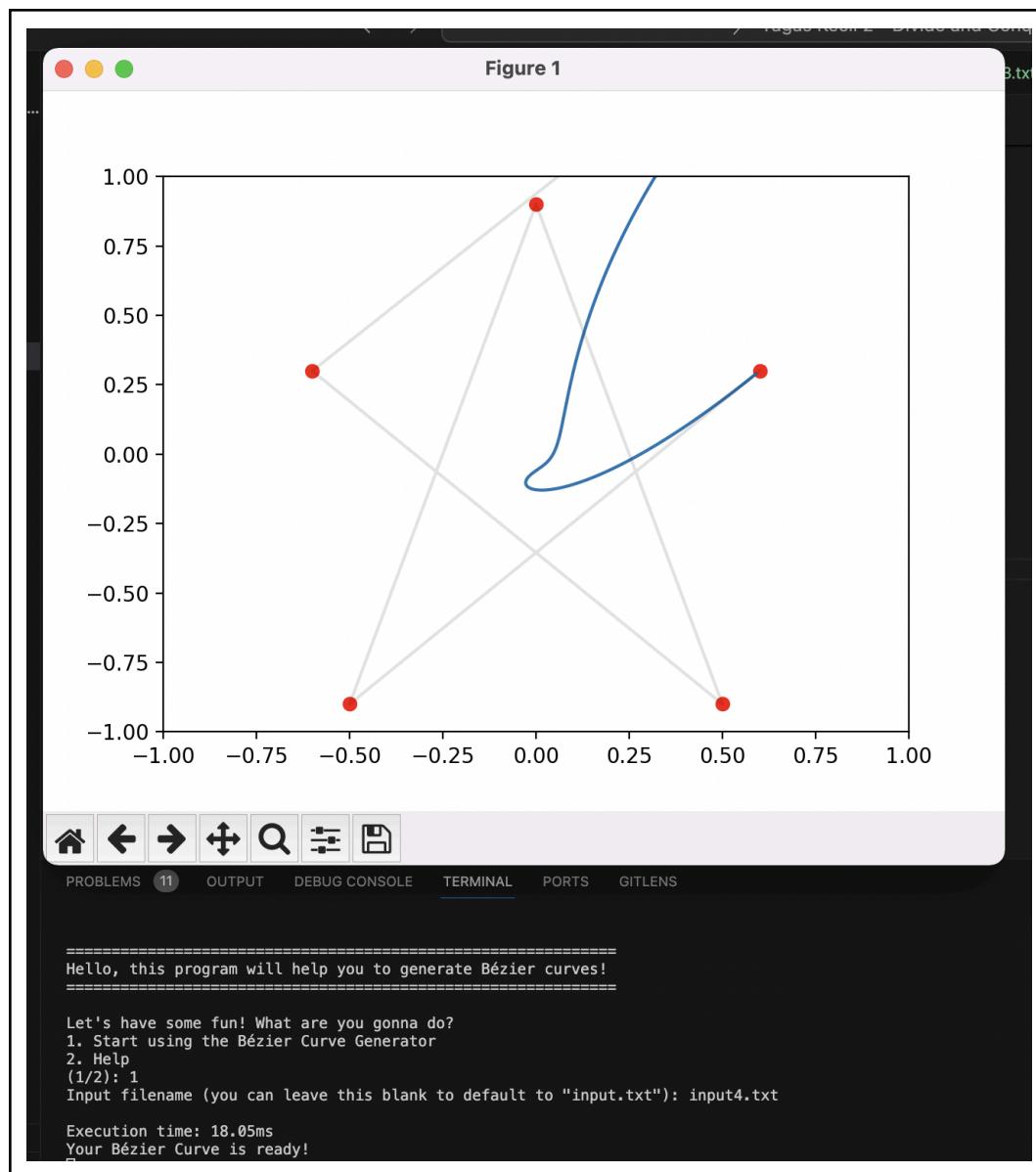
4. Uji Coba 4

Input:

```
6
1 2
-0.6 0.3
0.5 -0.9
0 0.9
-0.5 -0.9
0.6 0.3
10
-1 1 -1 1
```

Output Divide and Conquer:



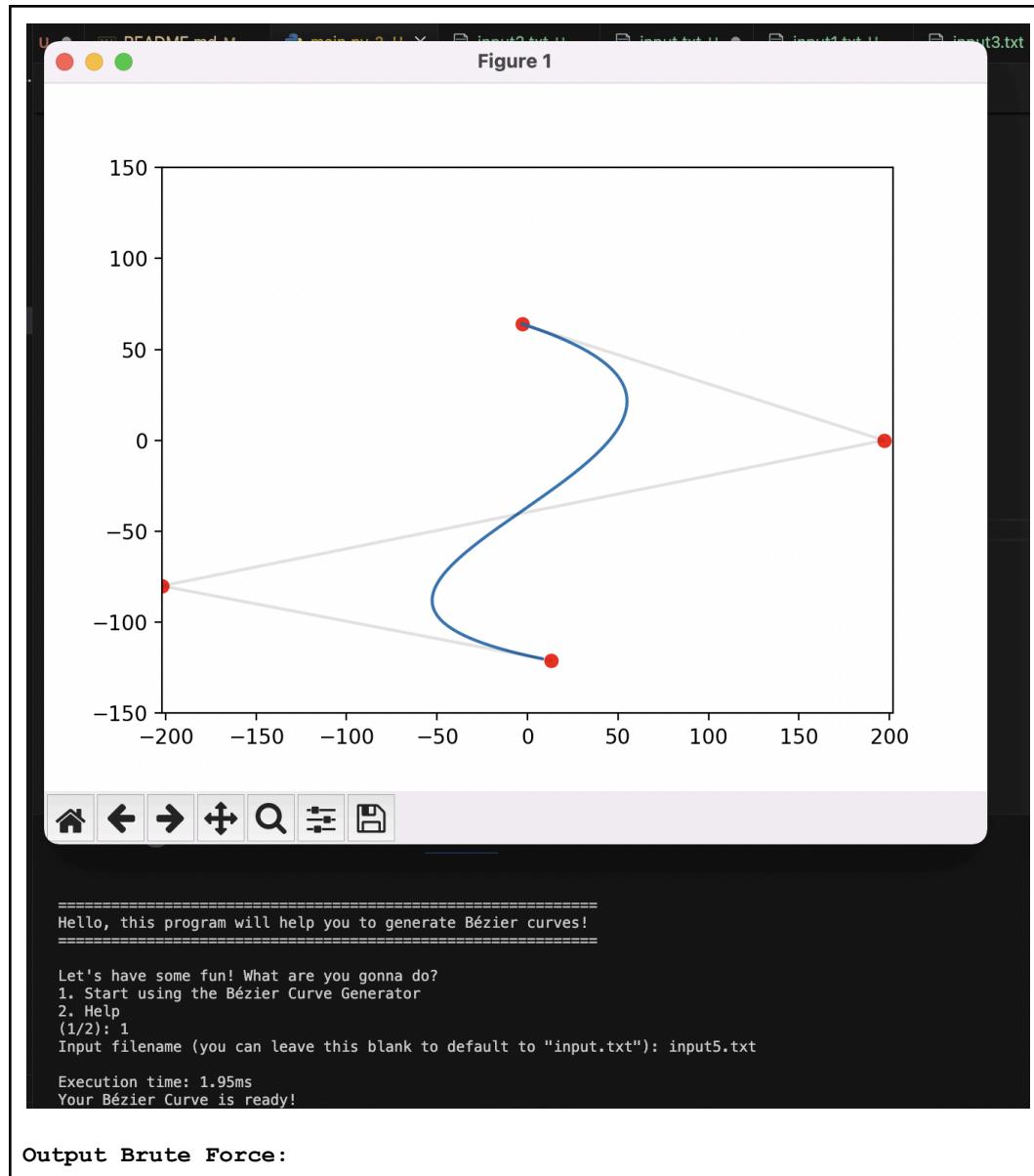


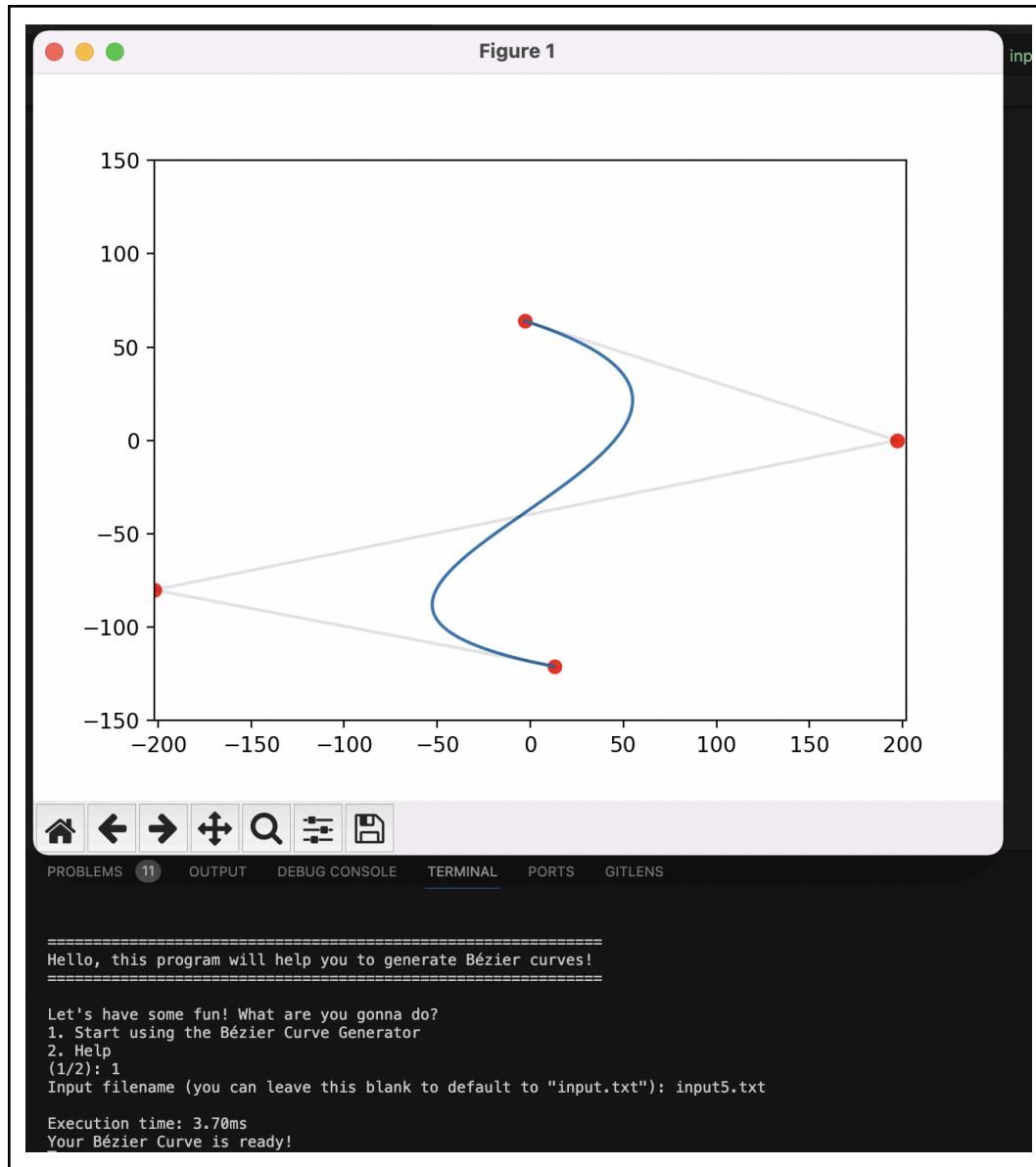
5. Uji Coba 5

Input:

```
4
-3 64
197 0
-202 -80
13 -121
7
-202 202 -150 150
```

Output Divide and Conquer:





6. Uji Coba 6

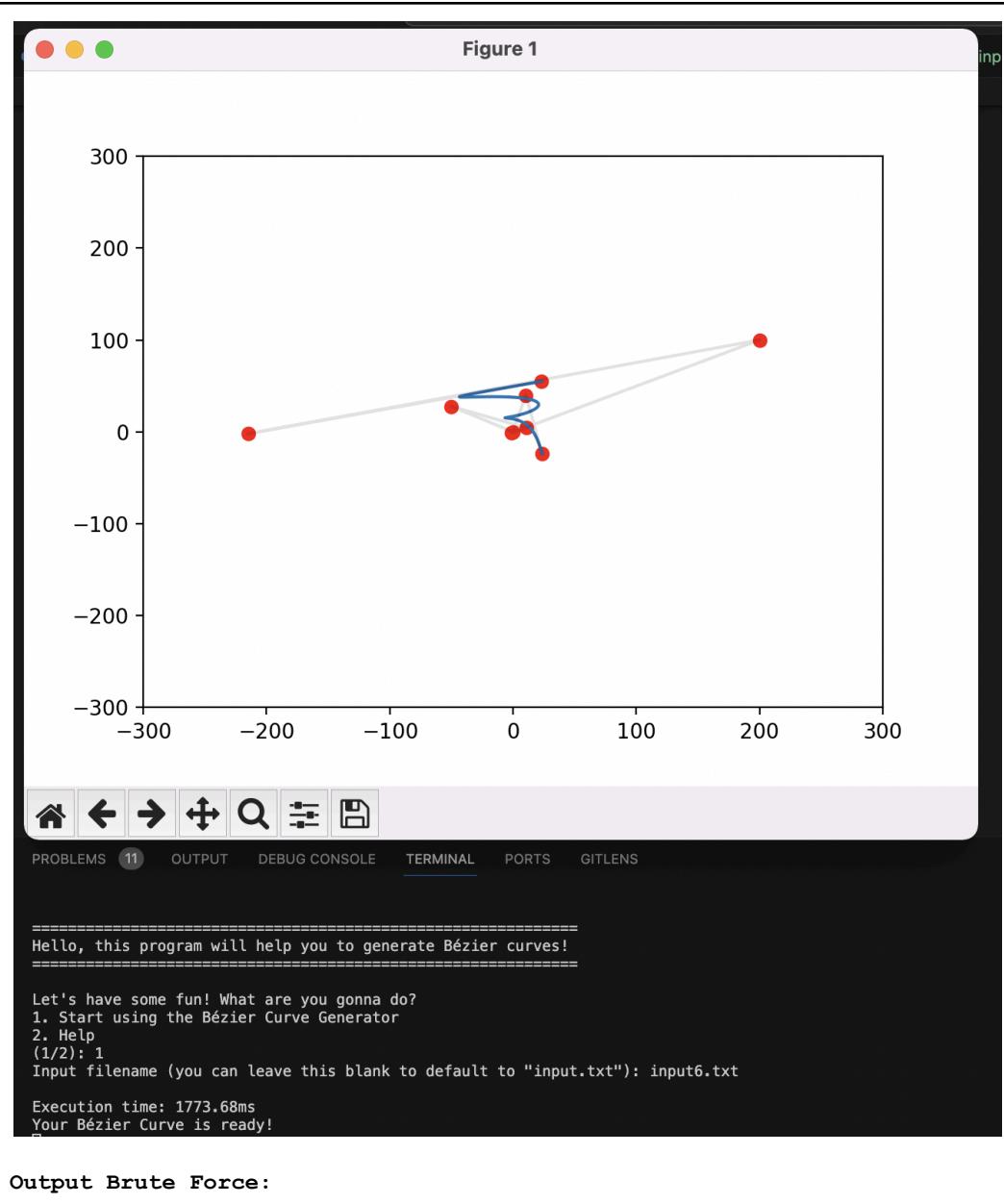
Input:

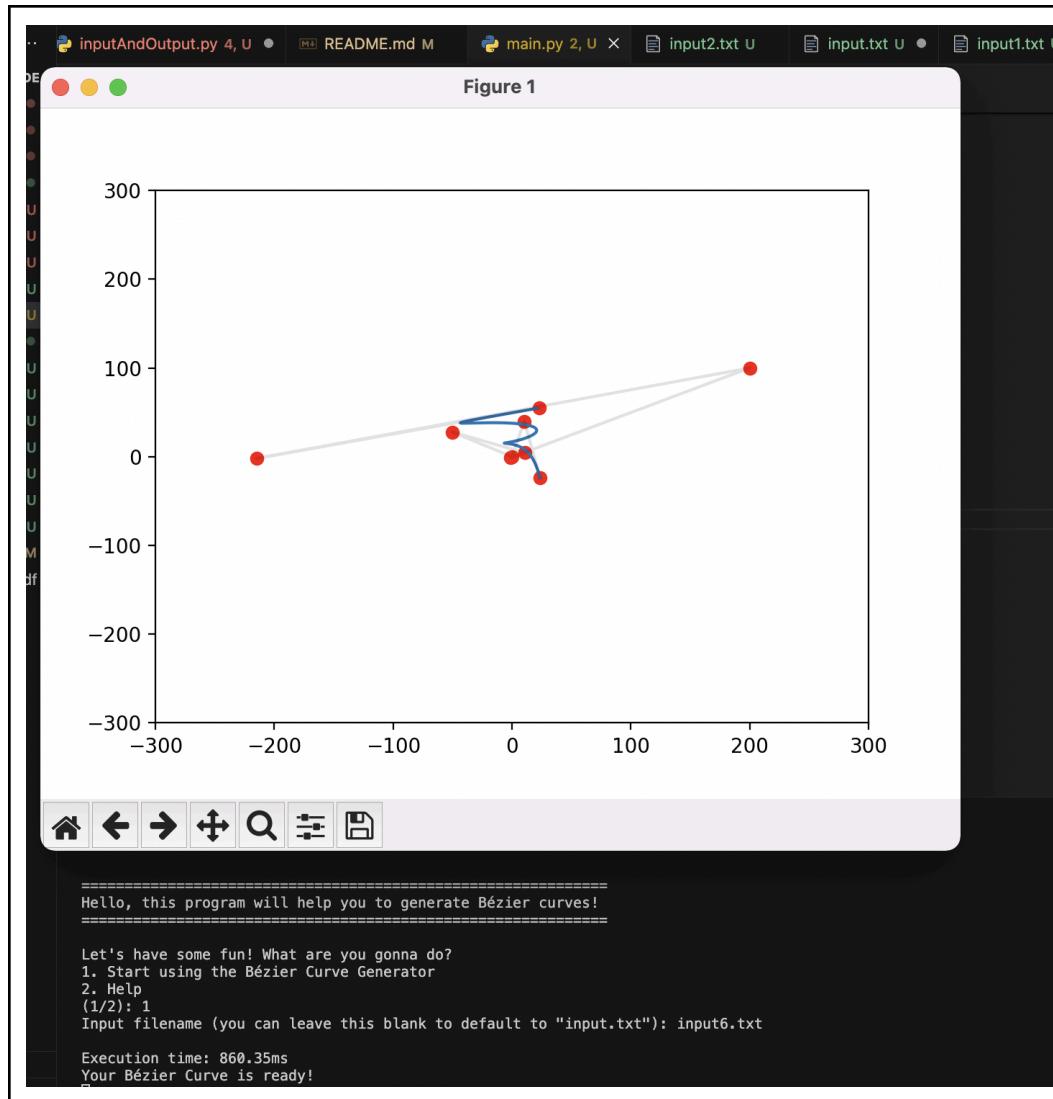
```

9
23.6 -23.6
10.4 40
0 0
-50.2 27.8
11.1111 4.32
-1 -1
200 100
-215 -2
23 55
17
-300 300 -300 300

```

Output Divide and Conquer:





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

Berdasarkan uji kasus di atas, tidak ada perbedaan dalam bentuk kurva untuk kedua pendekatan. Kedua pendekatan sudah memberikan bentuk yang benar. Terkecuali untuk uji kasus 4 yang mempunyai jumlah iterasi dua. Akibat jumlah iterasi yang sedikit, nilai t pada algoritma brute force kurang banyak sehingga titik kurva yang terhitung hanya tiga. Namun, hal ini persoalan trivial yang bisa diperbaiki dengan memulai nilai t dari 0, bukan dari inkremennya. Secara teori, berdasarkan kode implementasi, pendekatan brute force mengulang perhitungan polinom Bernstein sebanyak t kali. Setiap perhitungan mengandung n suku dengan masing-masing suku melakukan n perkalian. Dengan mengasumsikan fungsi `math.comb()` adalah $O(N)$, kompleksitas brute force yang diimplementasikan adalah $O(t N^2)$ dan $T(2t N^2)$. Secara teori, berdasarkan kode implementasi, pada pendekatan Divide and Conquer, fungsi terpanggil sebanyak 2^m kali dengan m adalah jumlah iterasi. Dalam setiap pemanggilan, berdasarkan langkah-langkah

pada bab 2, jika n adalah orde kurva, terjadi perhitungan titik tengah sebanyak $0.5 n^2$ kali. Dengan mengasumsi bahwa perhitungan titik tengah membutuhkan 4 operasi $((x_1 + x_2) / 2 \text{ dan } (y_1 + y_2) / 2)$, maka kompleksitas algoritma adalah $O(2^m N^2)$ dan $T(2^m \cdot 2n^2)$. Kedua pendekatan (secara teori) mempunyai kompleksitas yang sama. Hal ini karena diasumsikan jumlah titik yang dihitung kedua pendekatan sama sehingga berlaku $t = 2^m$. Namun, dari hasil pengujian, dapat dilihat bahwa kompleksitas waktu dari pendekatan divide and conquer yang saya implementasikan lebih besar dari brute force, terutama pada uji kasus 6 dan 7 yang mempunyai jumlah iterasi yang lebih besar dari uji kasus lainnya. Pada uji kasus 7, algoritma divide and conquer berjalan dua kali lebih pelan dari brute force. Hal ini mungkin disebabkan beberapa hal, tetapi yang menurut saya paling berpengaruh ada dua, yaitu penggunaan rekursi dan penggunaan array/list. Rekursi dalam Python berjalan lebih pelan daripada perulangan biasa, dan dalam kasus ini sangat berpengaruh karena rekursi diulang ratusan kali. Untuk array, dapat Anda lihat bahwa pada implementasi saya, terdapat banyak sekali operasi array dinamis, termasuk penambahan, slicing, dan reverse(). Apabila diperhatikan, pertambahan waktu yang dilakukan oleh operasi-operasi tersebut bersifat linier sehingga untuk notasi Big-O akan tetap sama. Akan tetapi, karena operasi cukup banyak dilakukan, muncul biaya overhead yang cukup signifikan. Biaya overhead operasi array ini tidak terdapat pada pendekatan brute force.

F. Pranala

Berikut ini adalah alamat *repository* yang berisi kode program Tugas Kecil 2 IF2211 Strategi Algoritma milik Nelsen Putra (13520130):

https://github.com/nelsenputra/Tucil2_13520130

G. Checklist

| No | Poin | Ya | Tidak |
|----|---|----|-------|
| 1 | Program berhasil dijalankan. | V | |
| 2 | Program dapat melakukan visualisasi kurva Bézier. | V | |
| 3 | Solusi yang diberikan program optimal. | V | |
| 4 | [Bonus] Program dapat membuat kurva untuk n titik kontrol. | V | |
| 5 | [Bonus] Program dapat melakukan visualisasi proses pembuatan kurva. | | V |

