LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

Konstruksi Kurva Bezier Menggunakan Pendekatan *Divide and Conquer* dan *Brute Force*

NIM	: 13522022
Nama	: Renaldy Arief Susanto
Kelas	: K02

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO dan INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

1. Konstruksi Kurva Bezier Menggunakan Pendekatan *Brute Force*

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

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

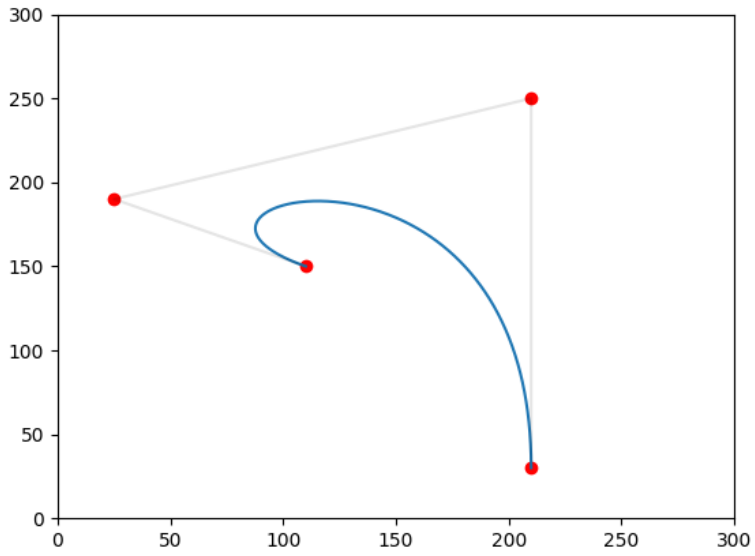
Sumber: https://pomax.github.io/bezierinfo/

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.

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

Sumber: https://pomax.github.io/bezierinfo/

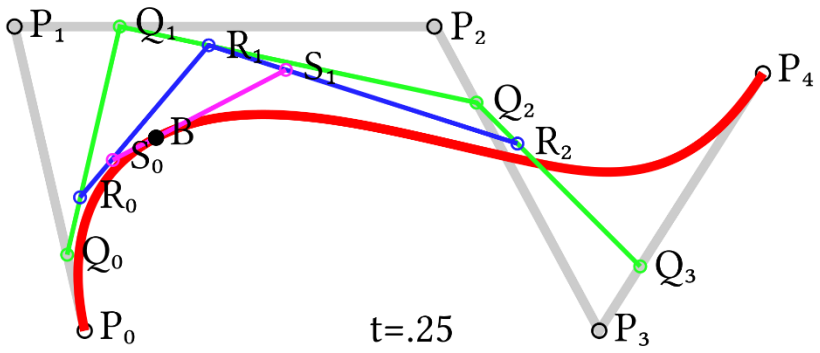
Range nilai t adalah [0, 1], dan setiap nilai t tersebut akan menghasilkan titik yang unik pada kurva Bezier. Berikut adalah hasil kurva jika kumpulan titik yang dihasilkan dari iterasi nilai t pada komputasi di atas digambar.



2. Konstruksi Kurva Bezier Menggunakan Pendekatan *Divide And Conquer*

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



Sumber: Wikipedia

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.

3. Implementasi

Algoritme ditulis dalam bahasa Python dan tidak digunakan *library* selain Python Standard Library. Library *matplotlib* digunakan hanya untuk menampilkan gambar kurva hasil. Berikut detail-detail implementasi.

3.1. Point Class

Didefinisikan objek *Point* atau titik untuk mempermudah penulisan operasi. Lebih spesifiknya, empat operator aritmetika di-*overload* dan didefinisikan juga fungsi `midpoint()` untuk mendapatkan titik tengah dari dua titik.

```python
@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))
```

3.2. Implementasi Algoritme *Brute Force*

Berikut implementasi langkah-langkah yang dijelaskan pada bab 1. Untuk memenuhi spesifikasi, diberikan juga parameter iterasi, yaitu mengaproksimasi nilai inkrementasi nilai t yang sesuai dengan jumlah titik yang dihasilkan algoritme *divide and conquer* pada iterasi tersebut (sehingga banyaknya titik yang dihasilkan sama).

```
def nDegreeBruteForceBezier(points: list[Point], iterations) :
    n = len(points) - 1
    diff = 1 / (2 ** iterations)
    t = diff
    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
```

3.3.1. Implementasi Algoritme *Divide and Conquer*

Berikut implementasi langkah-langkah yang dijelaskan pada bab 2.

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

    if iterations == 0 :
        return points

    elif iterations == 1 :

        prev_points = points.copy()
        for i in range(len(points) - 1) :
            new_points = list()
            for j in range(1, len(prev_points)) :
                new_points.append(midpoint(prev_points[j], prev_points[j-1]))
            prev_points = new_points

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

        return [p0,r0,p1]
```

```
else :

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

    left_points = [p0]
    right_points = [p1]

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

        left_points.append(new_points[0])
        right_points.append(new_points[-1])

        prev_points = new_points

    right_points.reverse()
    r0 = prev_points[0]

    # divide and conquer
    l = nDegreeBezier(left_points, iterations - 1)[:-1]
    m = [r0]
    r = nDegreeBezier(right_points, iterations - 1)[1:]

    return l + m + r
```

3.3.2. Implementasi Kurva Bezier Kuadratik (orde 2) dengan Divide And Conquer

Fungsi ini tidak digunakan dan dibuat untuk memenuhi spesifikasi tugas (tetapi sudah diuji dan hasilnya sama persis dengan fungsi pada subbab 3.3.1.). Konsep perhitungan sama persis dengan Kurva Bezier orde-n, tetapi jumlah titik kontrol tengah yang dihitung hanya satu untuk setiap iterasi sehingga penulisan kode dapat disederhanakan secara signifikan.

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

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

    elif iterations == 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, iterations - 1))[:-1]
        m = [r0]
        r = (quadraticBezier(r0, p21, p1, iterations - 1))[1:]

        return l + m + r
```

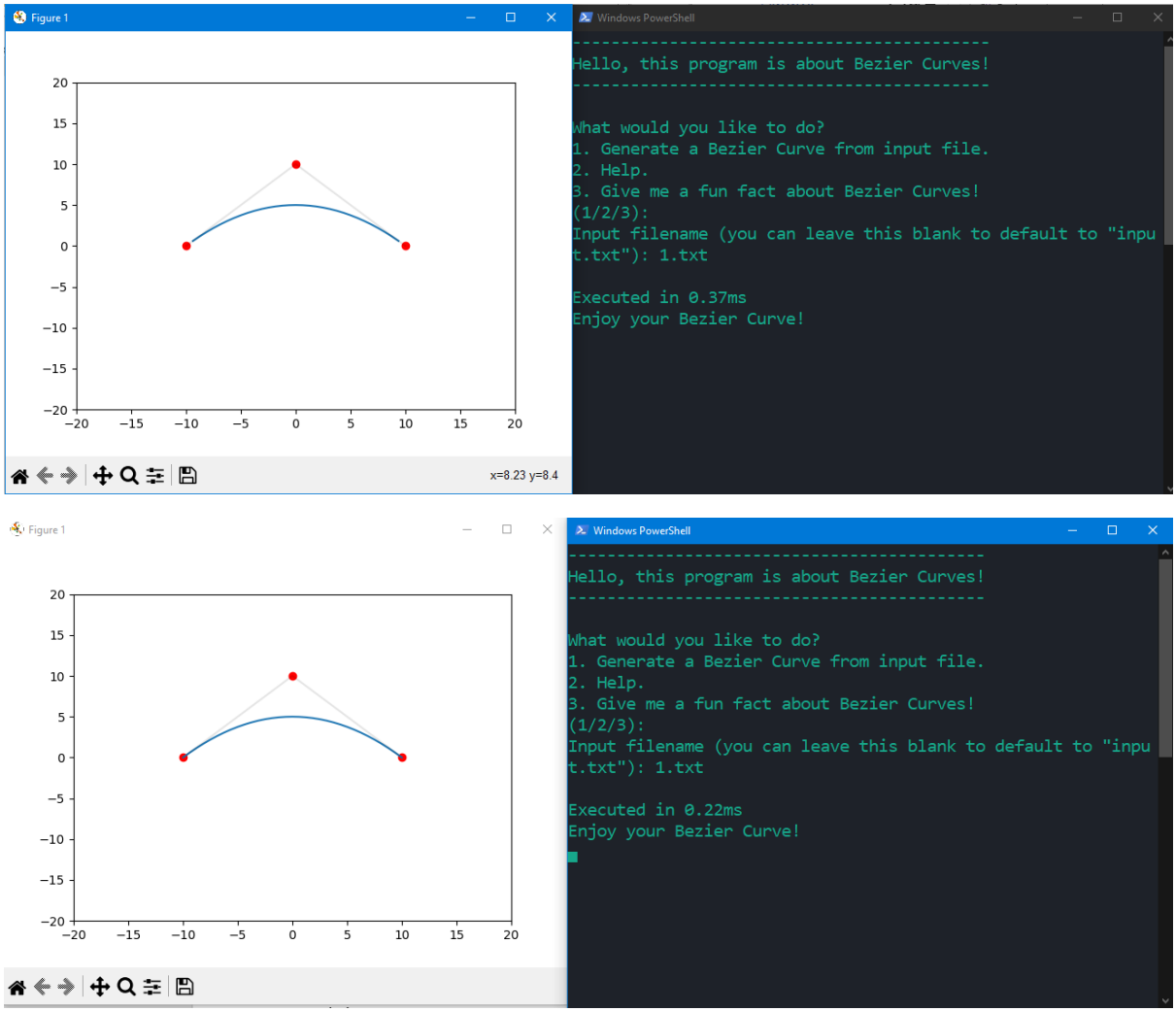
4. Pengujian

Untuk pengujian, sesuai dengan instruksi penggunaan program, akan digunakan file input dengan format yang sudah ditentukan. Namun, untuk mempermudah pembaca, ketentuan file input dituliskan kembali di sini.

1. Baris pertama berisi sebuah bilangan bulat p yang merupakan jumlah titik kontrol Kurva Bezier.
2. Sejumlah p baris berikutnya masing-masing berisi dua bilangan real x_i dan y_i yang merupakan nilai x dan y dari titik ke-i.
3. Baris berikutnya berisi satu bilangan bulat t yang menjelaskan berapa banyak iterasi yang akan dihitung untuk Kurva Bezier.
4. Baris terakhir terdiri dari 4 bilangan real x1, x2, y1, dan y2. Ini akan menjadi parameter untuk menampilkan grafik. Grafik akan menampilkan bidang kartesius untuk sumbu x pada rentang [x1, x2] dan sumbu y pada rentang [y1, y2].

4.1. Uji Kasus 1

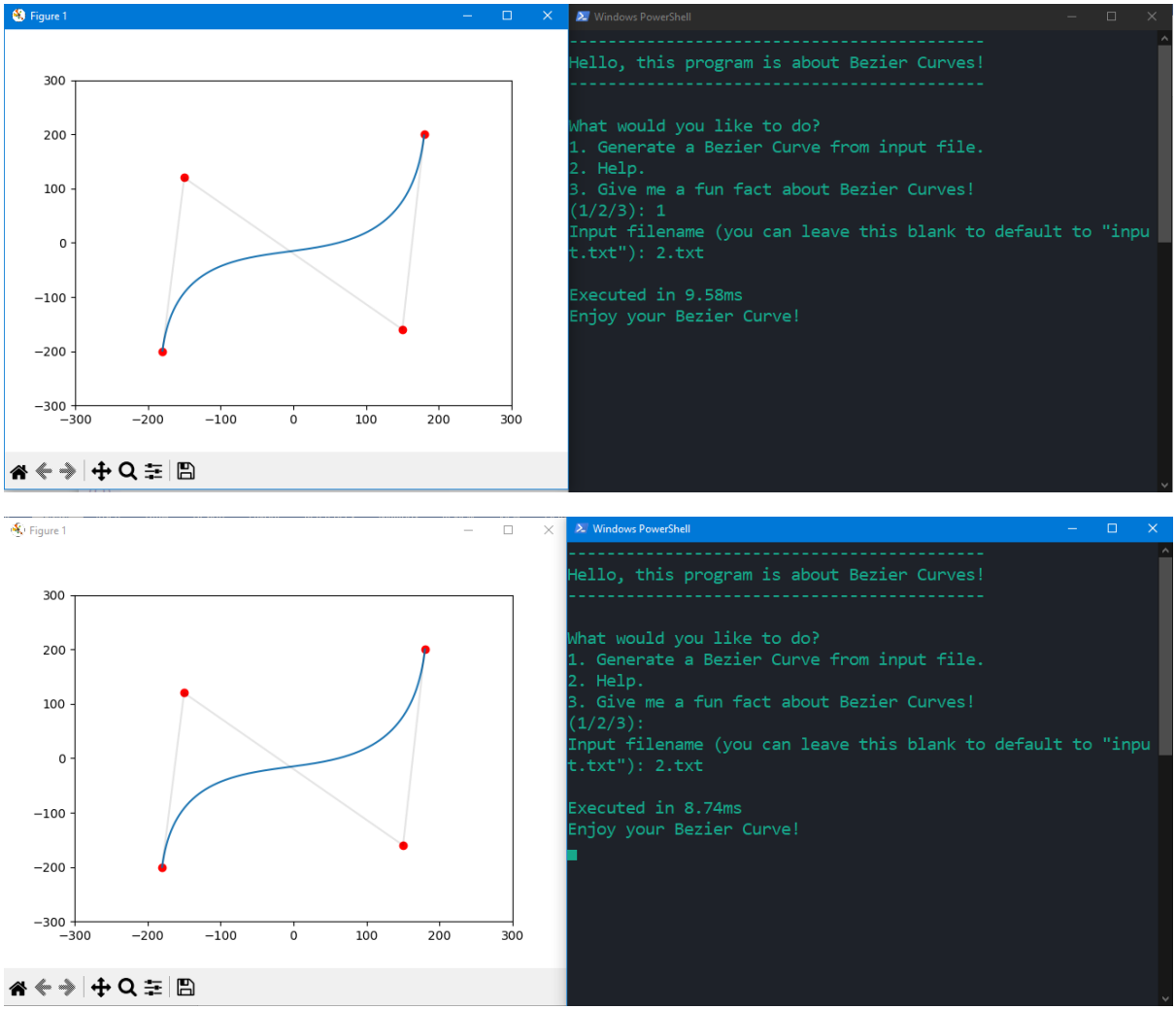
```
3
-10 0
0 10
10 0
5
-20 20 -20 20
```



Hasil uji kasus 1 algoritme Brute Force (atas) dan algoritme *Divide and Conquer* (bawah)

4.2. Uji Kasus 2

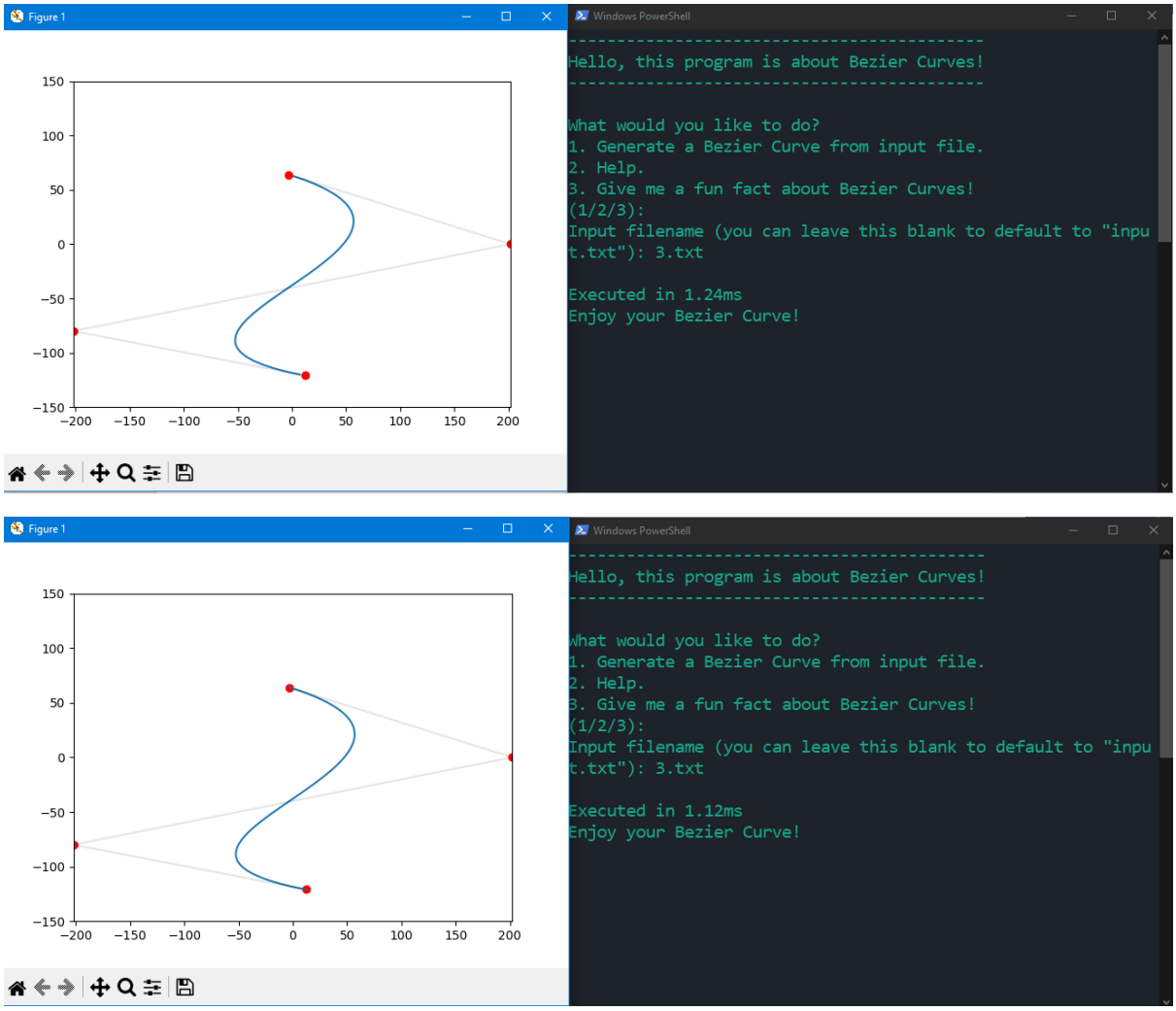
```
4
-180 -200
-150 120
150 -160
180 200
10
-300 300 -300 300
```



Hasil uji kasus 2 algoritme Brute Force (atas) dan algoritme *Divide and Conquer* (bawah)

4.3. Uji Kasus 3

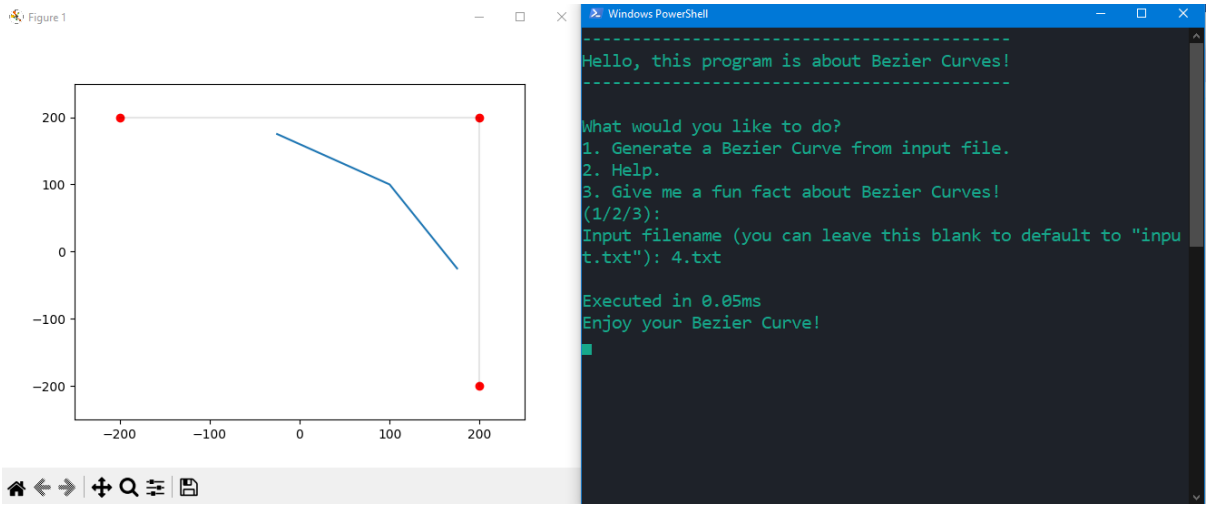
```
4
-3 64
202 0
-202 -80
12 -121
7
-202 202 -150 150
```

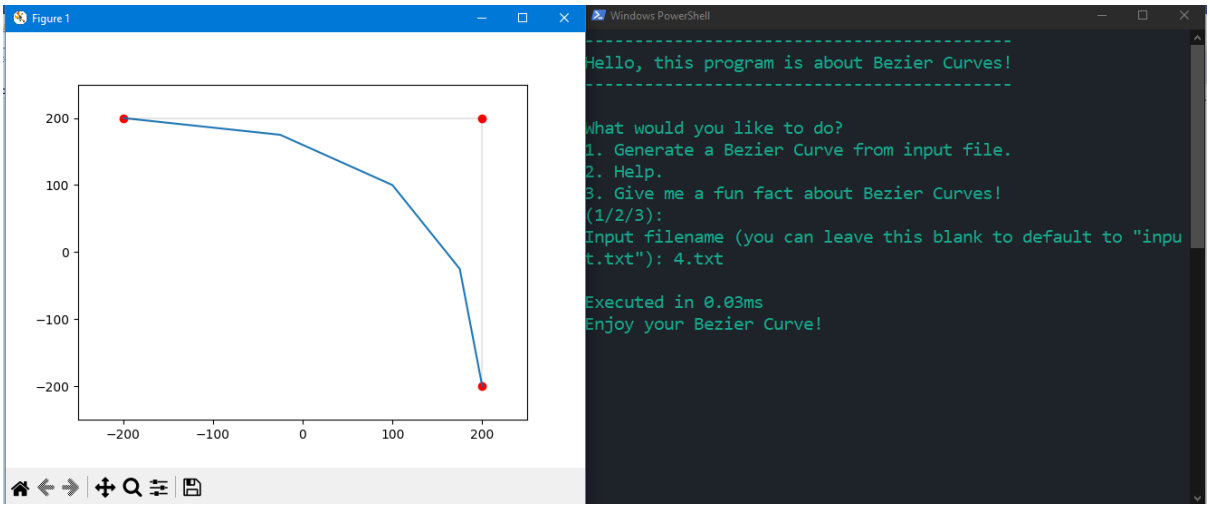


Hasil uji kasus 3 algoritme Brute Force (atas) dan algoritme *Divide and Conquer* (bawah)

4.4. Uji Kasus 4

```
3
-200 200
200 200
200 -200
2
-250 250 -250 250
```

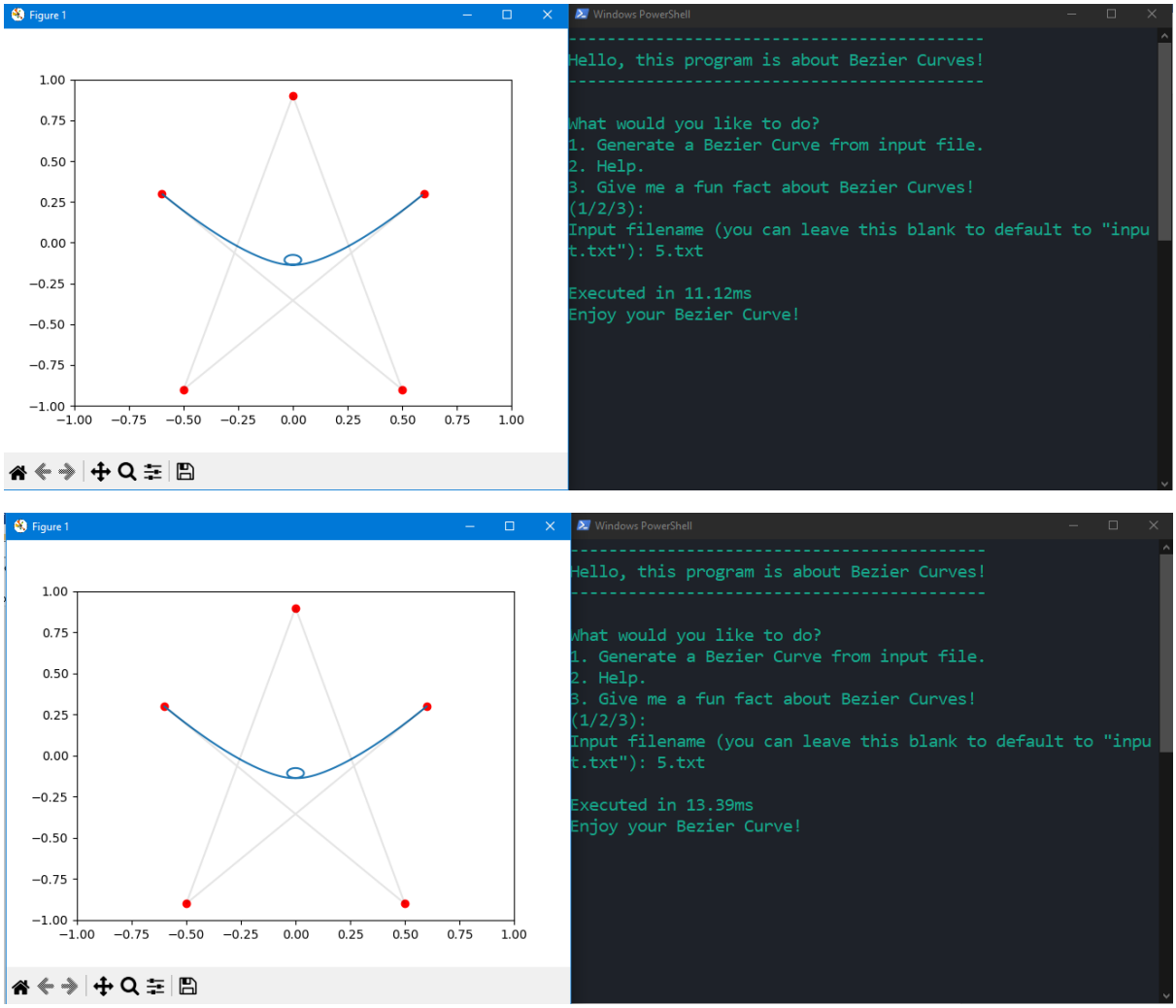




Hasil uji kasus 4 Brute Force (atas) dan algoritme *Divide and Conquer* (bawah)

4.5. Uji Kasus 5

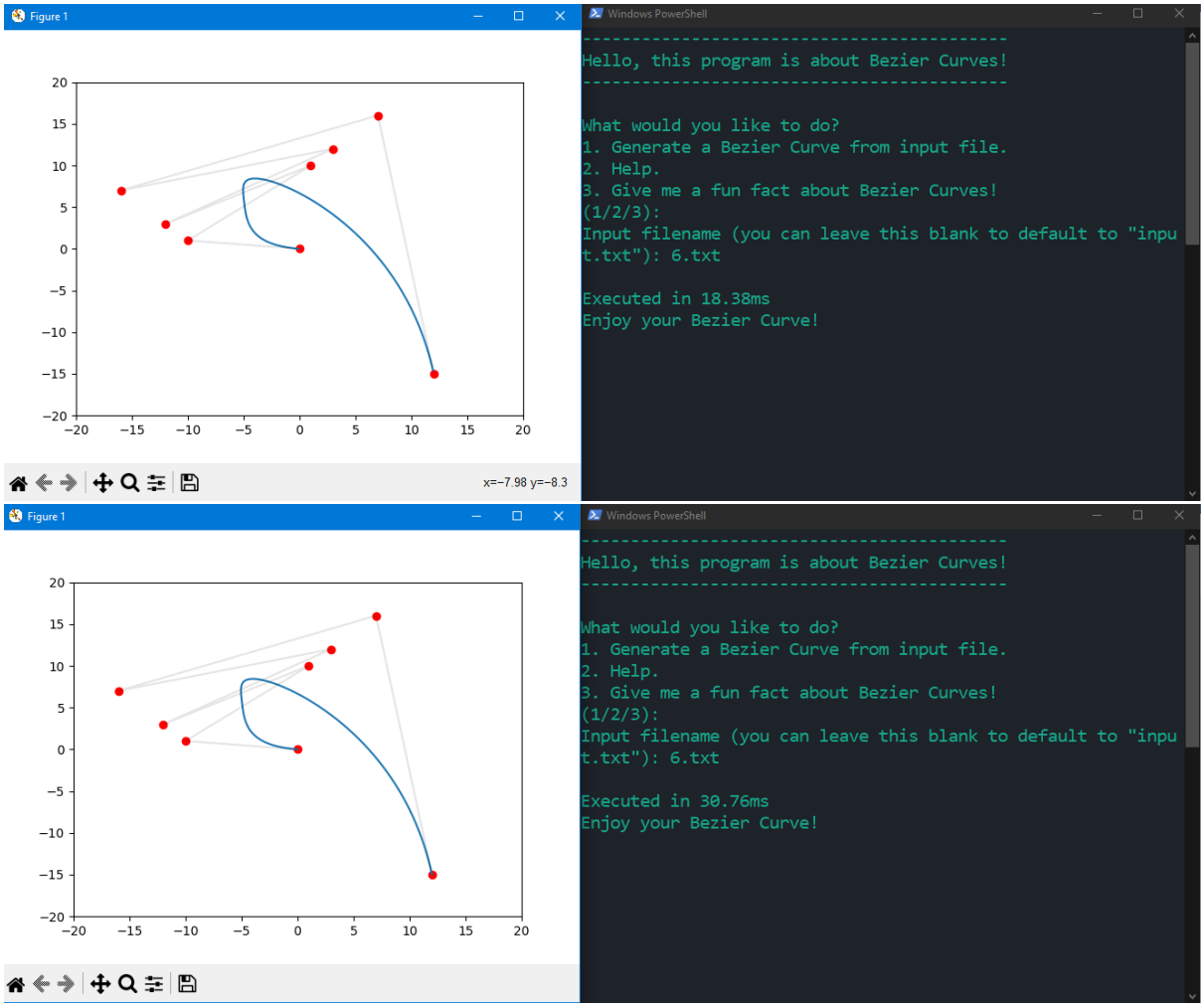
```
5
-0.6 0.3
0.5 -0.9
0 0.9
-0.5 -0.9
0.6 0.3
10
-1 1 -1 1
```



Hasil uji kasus 5 algoritme Brute Force (atas) dan algoritme *Divide and Conquer* (bawah)

4.6. Uji Kasus 6

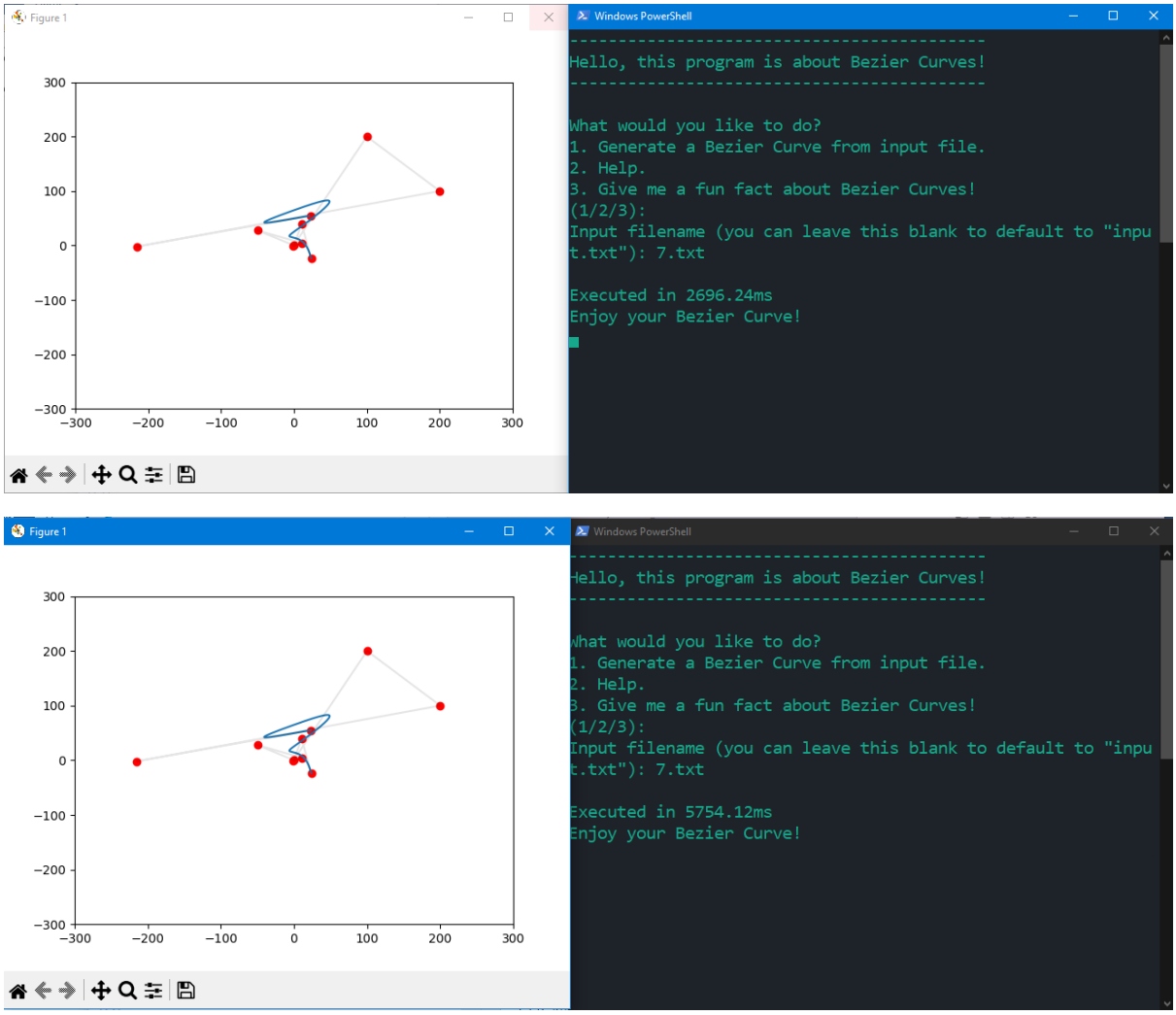
```
8
0 0
-10 1
1 10
-12 3
3 12
-16 7
7 16
12 -15
10
-20 20 -20 20
```



Hasil uji kasus 6 algoritme Brute Force (atas) dan algoritme *Divide and Conquer* (bawah)

4.7. Uji Kasus 7

```
10
23.6 -23.6
10.4 40
0 0
-50.2 27.8
11.1111 4.32
-1 -1
100 200
200 100
-215 -2
23 55
17
-300 300 -300 300
```



Hasil uji kasus 7 algoritme Brute Force (atas) dan algoritme *Divide and Conquer* (bawah)

5. Analisis Hasil dan Kompleksitas

Berdasarkan tujuh 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 ((x1 + x2) / 2 dan (y1 + y2) / 2), maka kompleksitas algoritma adalah $O(2^m N^2)$ dan $T(2^m 2n^2)$.

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, algoritme *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*.

6. *Checklist* dan Tautan *Repository* Github

6.1. *Checklist*

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✓	
2. Program dapat melakukan visualisasi kurva Bézier.	✓	
3. Solusi yang diberikan program optimal.	✓	
4. **[Bonus]** Program dapat membuat kurva untuk *n* titik kontrol.	✓	
5. **[Bonus]** Program dapat melakukan visualisasi proses pembuatan kurva.		✓

6.2. Tautan *Repository* Github

https://github.com/AldyDPP/Tucil2_13522022