LAPORAN TUGAS KECIL 2

IF2211 STRATEGI ALGORITMA

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Disusun oleh:

Imam Hanif Mulyarahman 13522030

Muhammad Syarafi Akmal 13522076

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

I. Analisis dan Implementasi Algoritma *Brute Force*

Pada algoritma *brute force*, persoalan dilakukan secara iteratif menggunakan rumus yang disediakan yaitu,

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

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

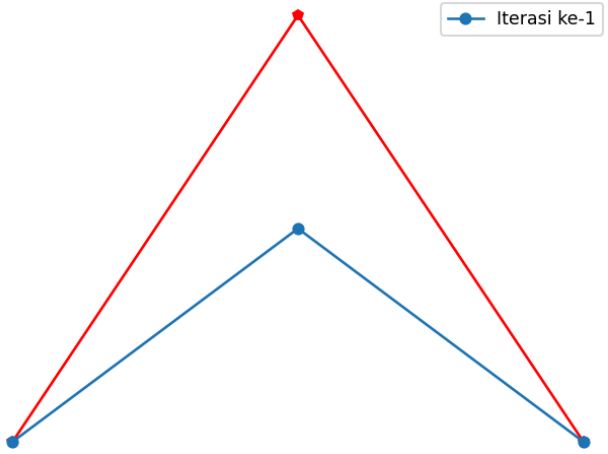
$$R_0 = B(t) = (1-t)Q_0 + tQ_1, \qquad t \in [0,1]$$

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2 \ , t \in [0,1].$$

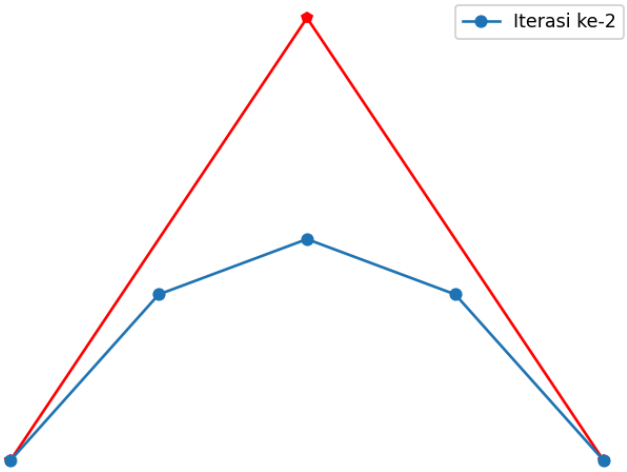
Gambar 1.1. Rumus Bezier

Dengan melakukan perumusan diatas sebanyak partisi, himpunan titik R_0, Q_0 sebagai titik mula, dan Q_1 sebagai titik akhir akan menyusun kurva bezier.

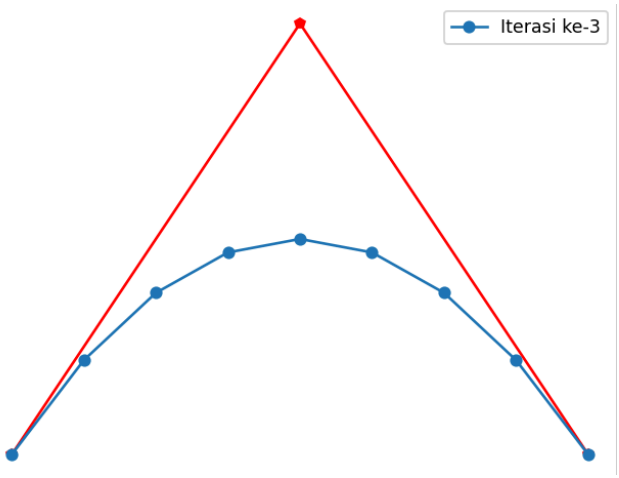
Spesifikasi program memiliki input yaitu iterasi melainkan partisi. Dengan menganalisis proses iterasi pada kurva bezier, kita dapat menentukan hubungan dari partisi dengan iterasi.



Gambar 1.2. Kurva bezier iterasi-1



Gambar 1.3. Kurva bezier iterasi-2



Gambar 1.4. Kurva bezier iterasi-3

Pada ketiga gambar diatas dapat dilihat bahwa jumlah garis yang membentuk kurva ,selain Q_0 dan Q_1, mengartikan bahwa terdapat partisi sebanyak n+1 dengan n adalah jumlah titik, maka untuk iterasi-1 terdapat 2 partisi, iterasi-2 terdapat 4 partisi, dan iterasi-3 terdapat 8 partisi. Dari pola tersebut didapatkan bahwa hubungan partisi dengan garis yaitu,

$$\text{Jumlah partisi (n)} = 2^n, \quad n = \text{jumlah iterasi}$$

Sehingga, proses iterasi dengan rumus di atas dapat dilakukan sebanyak perumusan jumlah partisi ,melalui jumlah iterasi, untuk mengumpulkan semua himpunan titik kurva bezier.

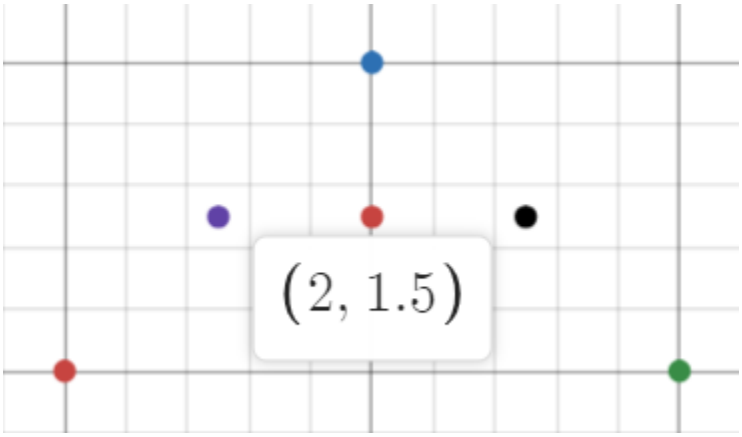
Berikut pseudocode dari algoritma *brute force* kurva bezier 3 titik.

```
function Brute_Force_Bezier(input integer: iterasi, point: p0, point: cp, point: p1,
    input/output arrayOfPoints : Hasil)

KAMUS
    {tidak ada}

ALGORITMA
   t iterate [0 …  2^n+1]
      append(Hasil, Bezier_Points(p0, cp, p1, t)) //Bezier_Points mengembalikan nilai
R0 per nilai t
```

II. Analisis dan Implementasi Algoritma *Divide and Conquer*

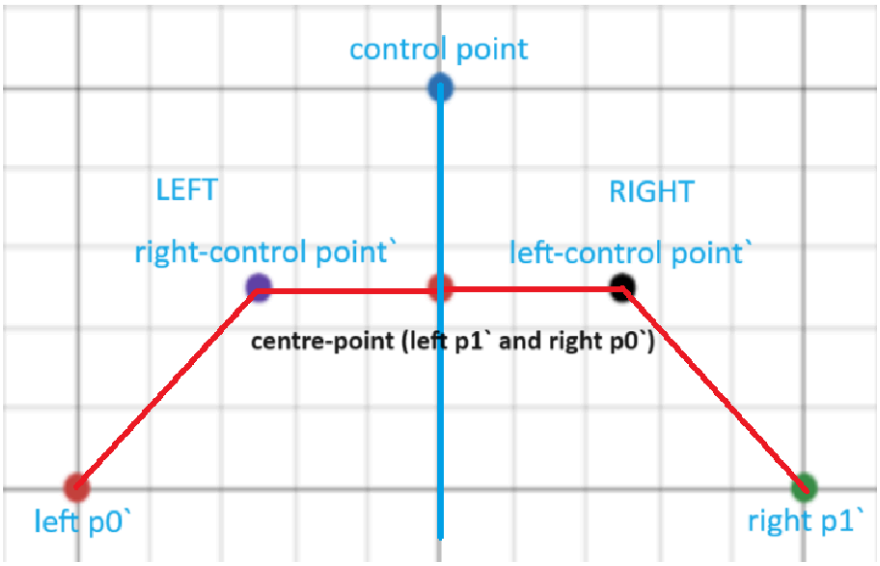
Pada algoritma *divide and conquer*, persoalan dibagi menjadi dua bagian yaitu kiri dan kanan berdasarkan pada daerah tiga titik (p0, *controlling point*, p1). Lalu pada masing-masing daerah dilakukan lagi pembagian sub-persoalan hingga mencapai batas tertentu agar bisa diselesaikan dan dapat digabung menjadi satu solusi.

Algoritma ini menggunakan konsep rekursif dengan basis yaitu kondisi dimana persoalan berada pada kondisi dasar, yaitu pada saat iterasi yang dilakukan hanya sekali dan ‘kurva’ yang terbentuk merupakan p0, middle-section dan p1. Middle-section adalah titik pusat dari segitiga yang terbentuk oleh tiga titik, terbentuk dengan mencari titik tengah dari titik tengah yang dihasilkan dari (p1, *controlling point*) dan (p2, *controlling point*) (ilustrasi middle-section terdapat pada gambar 2.1.).



Gambar 2.1. (middle-section).

Rekurens dari persoalan ini adalah saat membagi daerah menjadi dua bagian, terdapat sub-persoalan baru berupa daerah tiga titik (p0`, *controlling point*`,* p1`) yang lebih kecil di masing-masing daerah kiri dan kanan. Berikut ilustrasi dari sub-persoalan yang terbentuk.



Gambar 2.2. (ilustrasi sub-persoalan)

Sehingga pada fungsi, pemanggilan rekursif akan dilakukan sebanyak 2 kali untuk bagian kanan dan kiri.

Penyatuan solusi dilakukan dengan cara mengurutkan titik-titik dari semua middle section yang didapatkan pada proses rekursi dan menghubungkannya dengan titik p0 (di paling awal) dan p1 awal (di paling akhir) yang akan menjadi kurva bezier. Berikut adalah himpunan titik-titik kurva bezier yang akan dihasilkan.

pBezier = {p0, left-middle-section1, right-middle-section1, left-middle-section2, right-middle-section2, **…** , left-middle-sectionN, right-middle-sectionN/2, **center-point**, left-middle-sectionN/2+1, center-pointN/2+1, right-middle-sectionN/2+1, **…** , left-middle-sectionN, right-middle-sectionN p1}

Berikut adalah pseudocode dari algoritma *divide and conquer* kurva bezier 3 titik.

```
procedure Divide_and_Conquer_Bezier(input integer: iterasi, point: p0, point: cp,
    point: p1,  input/output arrayOfPoints : Hasil)

KAMUS
    {tidak ada}

ALGORITMA
   if iterasi = 1 then
       append(Hasil, p0) //fungsi append(array, element)
       append(Hasil, mid_section(p0, cp))
    else
       Divide_and_Conquer_Bezier(iterasi-1, p0, mid_point(p0, cp), mid_section(p0,
       cp, p1), Hasil) //Rekursi bagian kiri
       Divide_and_Conquer_Bezier(iterasi-1, mid_section(p0, cp, p1), mid_point(cp,
        p1), p1, Hasil) //Rekursi bagian kanan
```

III. Source Code Algoritma *Divide and Conquer* dan *Brute Force* (Bahasa Pemrograman Python)

a. Source Code Algoritma Divide and Conquer

```python
#Main Function
def Divide_n_Conquer(n, p0, cp, p2, res):
    if (n==1):
        res.append(p0)
        res.append(mid_section(p0, cp, p2))
    else:
        Divide_n_Conquer(n-1, p0, mid_point(p0, cp), mid_section(p0, cp, p2), res) #left
        Divide_n_Conquer(n-1, mid_section(p0, cp, p2), mid_point(p2, cp), p2, res) #right
    res.append(p2)
```

Gambar 3.a.1. (main dnc function)

```python
#Helping Functions
def Insert_Point(pn):
    point = [float(x) for x in input(f"Masukkan titik {pn} (x,y): ").split(",")]
    return Point(point[0], point[1])

def mid_point(p0: Point, p2: Point):
    return Point((p0.getAbsis()+p2.getAbsis())/float(2), (p0.getOrdinat()+p2.getOrdinat())/float(2))

def mid_section(p0: Point, cp: Point, p2: Point):
    return mid_point(mid_point(p0, cp), mid_point(cp, p2))
```

Gambar 3.a.2. (helping functions)

```python
class Point:

    #Constructor
    def __init__(self, x, y):
        self.x = x
        self.y = y

    #Methods
    def setAbsis(self,x):
        self.x = x
    def setOrdinat(self,y):
        self.y = y
    def getAbsis(self):
        return self.x
    def getOrdinat(self):
        return self.y
    def printPoint(self):
        print(f"({self.x}, {self.y})")
```

Gambar 3.a.3. (point class)

b. Source Code Algoritma Brute Force

```python
def bezierBF(p0,p1,p2,iterasi) :
    t = 2**iterasi
    arraybezier = []
    for i in range(0,t+1) :
        b = rumusbezier(p0,p1,p2,(i/t))
        arraybezier.append(Point(b[0],b[1]))
    return arraybezier
```

Gambar 3.b.1. (main brute force function)

```python
def Insert_Point(pn):
    point = [float(x) for x in input(f"Masukkan titik {pn} (x,y): ").split(",")]
    return Point(point[0], point[1])

def rumusbezier(p0:Point, p1:Point, p2:Point, t) :
    return [(1-t)*(1-t)*p0.getAbsis() + 2*(1-t)*t*p1.getAbsis() + t*t*p2.getAbsis(), (1-t)*(1-t)*p0.getOrdinat() + 2*(1-t)*t*p1.getOrdinat() + t*t*p2.getOrdinat()]
```

Gambar 3.b.2. (helping functions)

```python
class Point:

    #Constructor
    def __init__(self, x, y):
        self.x = x
        self.y = y

    #Methods
    def setAbsis(self,x):
        self.x = x
    def setOrdinat(self,y):
        self.y = y
    def getAbsis(self):
        return self.x
    def getOrdinat(self):
        return self.y
    def printPoint(self):
        print(f"({self.x}, {self.y})")
```

Gambar 3.a.3. (point class)

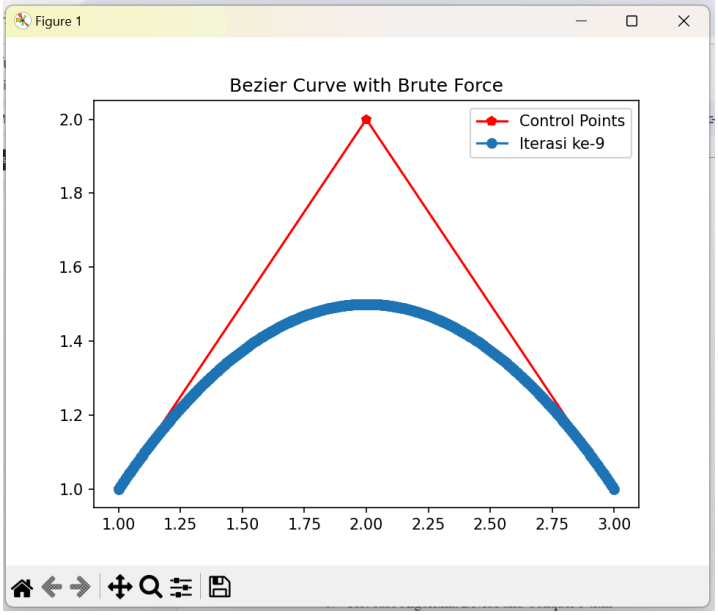
IV. Test Case

a. Test case Algoritma Brute Force 3 titik

Input 1:

```
Masukkan jumlah iterasi : 9
Masukkan titik p0 (x,y): 1,1
Masukkan titik p1 (x,y): 2,2
Masukkan titik p2 (x,y): 3,1
Time taken: 1.358ms
```

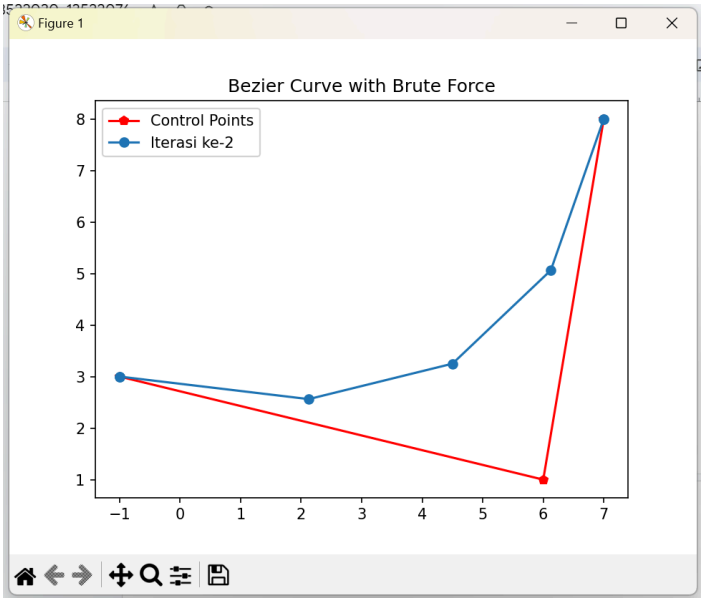
Output 1:



Input 2 :

```
Masukkan jumlah iterasi : 2
Masukkan titik p0 (x,y): -1,3
Masukkan titik p1 (x,y): 6,1
Masukkan titik p2 (x,y): 7,8
Time taken: 0.000ms
```

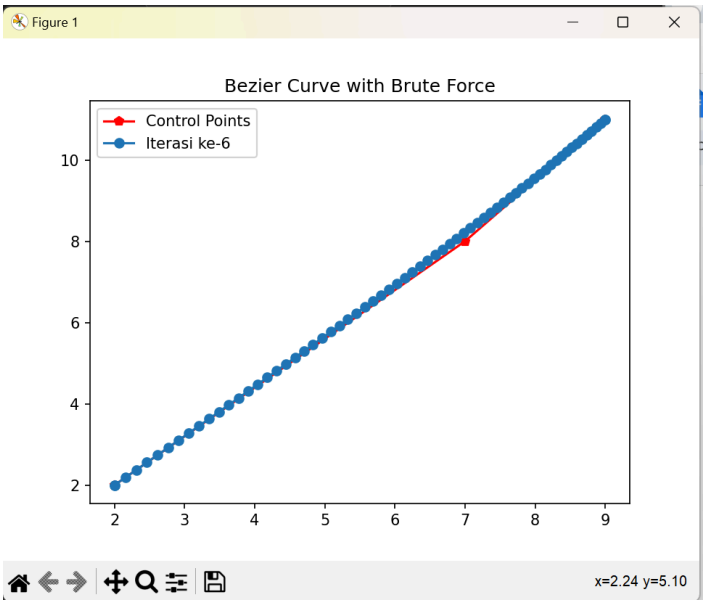
Output 2 :



Input 3 :

```
Masukkan jumlah iterasi : 6
Masukkan titik p0 (x,y): 2,2
Masukkan titik p1 (x,y): 7,8
Masukkan titik p2 (x,y): 9,11
Time taken: 0.000ms
```

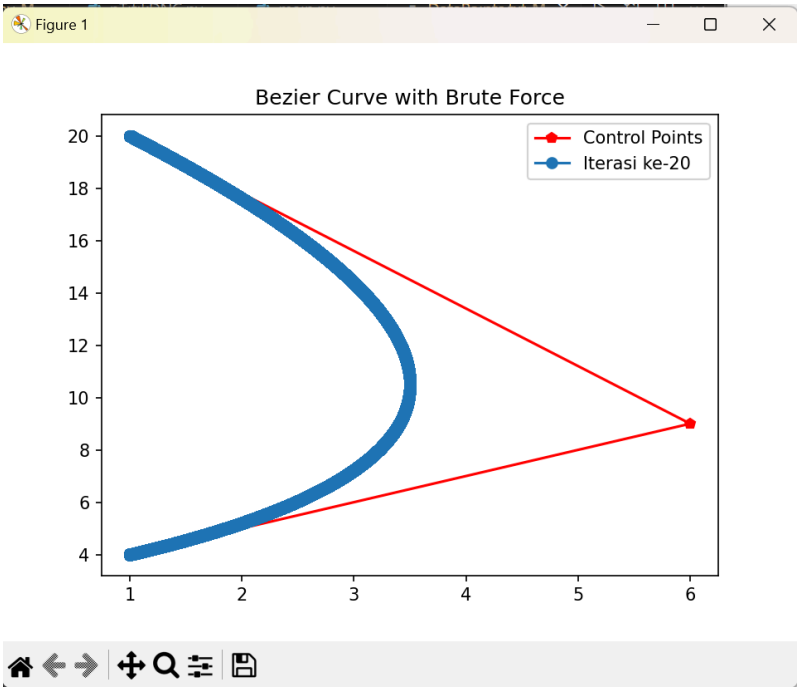
Output 3 :



Input 4 :

```
Masukkan jumlah iterasi : 20
Masukkan titik p0 (x,y): 1,4
Masukkan titik p1 (x,y): 6,9
Masukkan titik p2 (x,y): 1,20
Time taken: 1284.418ms
```

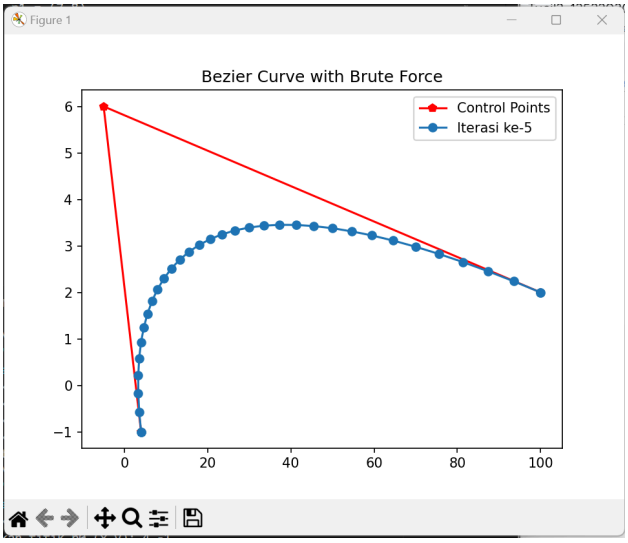
Output 4 :



Input 5 :

```
Masukkan jumlah iterasi : 5
Masukkan titik p0 (x,y): 4,-1
Masukkan titik p1 (x,y): -5,6
Masukkan titik p2 (x,y): 100,2
Time taken: 0.000ms
```

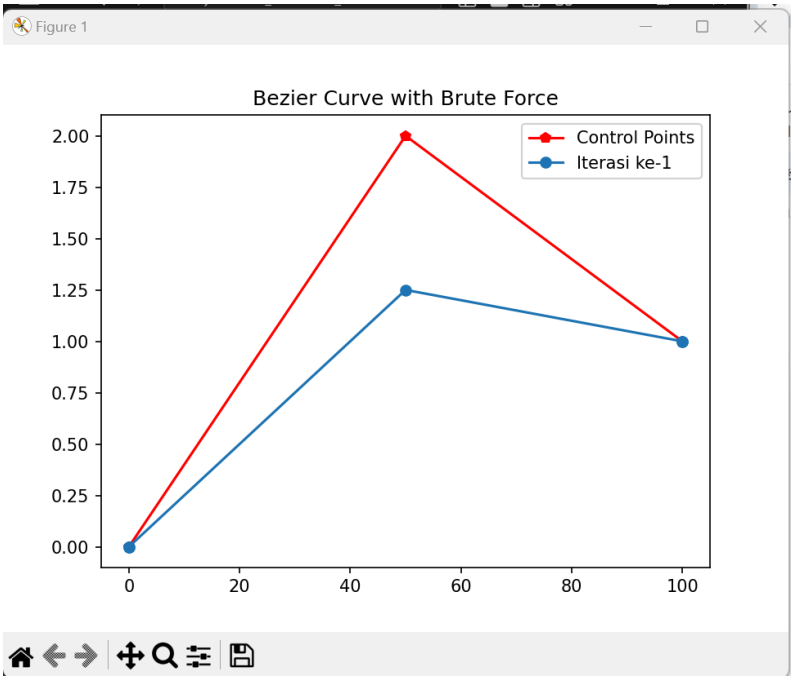
Output 5 :



Input 6 :

```
Masukkan jumlah iterasi : 1
Masukkan titik p0 (x,y): 0,0
Masukkan titik p1 (x,y): 50,2
Masukkan titik p2 (x,y): 100,1
Time taken: 0.000ms
```

Output 6 :

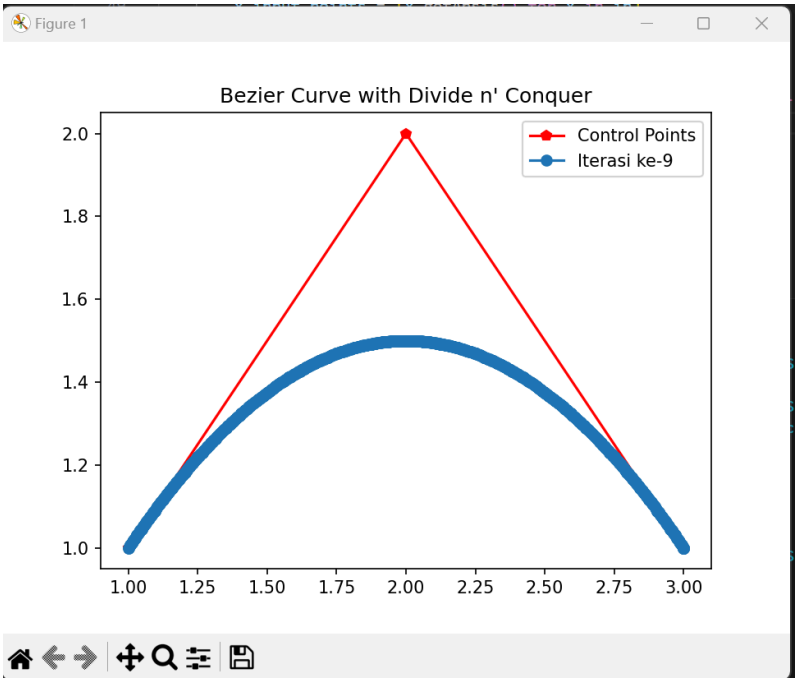


b. Test case Algoritma Divide and Conquer 3 titik

Input 1 :

```
Masukkan jumlah iterasi: 9
Masukkan titik p0 (x,y): 1,1
Masukkan titik p1 (x,y): 2,2
Masukkan titik p2 (x,y): 3,1
Time taken: 2.525ms
```

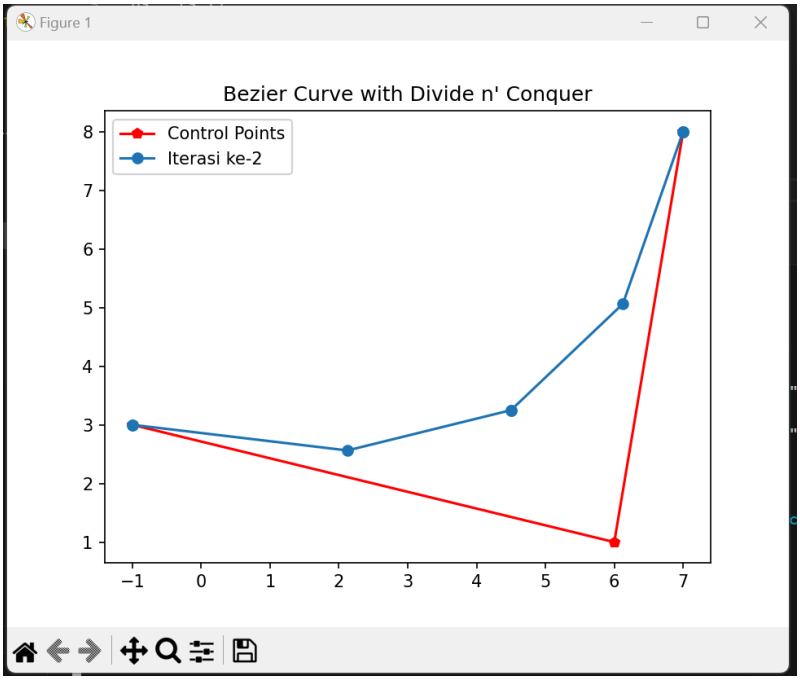
Output 1 :



Input 2:

```
Masukkan jumlah iterasi: 2
Masukkan titik p0 (x,y): -1,3
Masukkan titik p1 (x,y): 6,1
Masukkan titik p2 (x,y): 7,8
Time taken: 0.000ms
```

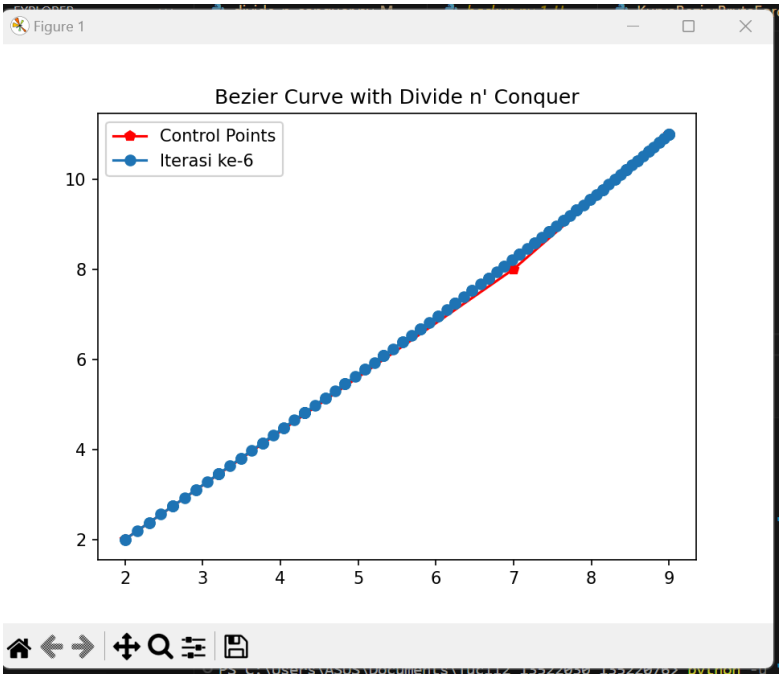
Output 2 :



Input 3 :

```
Masukkan jumlah iterasi: 6
Masukkan titik p0 (x,y): 2,2
Masukkan titik p1 (x,y): 7,8
Masukkan titik p2 (x,y): 9,11
Time taken: 0.000ms
```

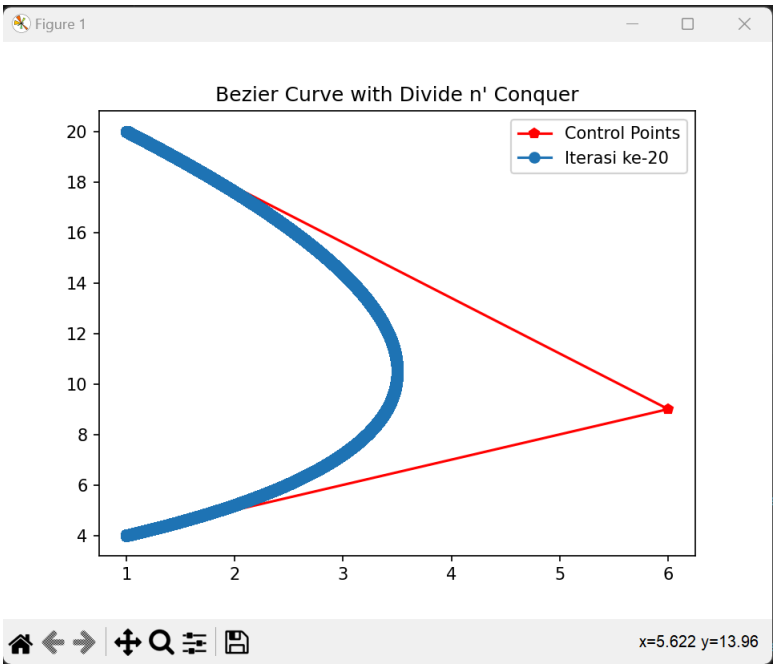
Output 3 :



Input 4 :

```
Masukkan jumlah iterasi: 20
Masukkan titik p0 (x,y): 1,4
Masukkan titik p1 (x,y): 6,9
Masukkan titik p2 (x,y): 1,20
Time taken: 4658.015ms
```

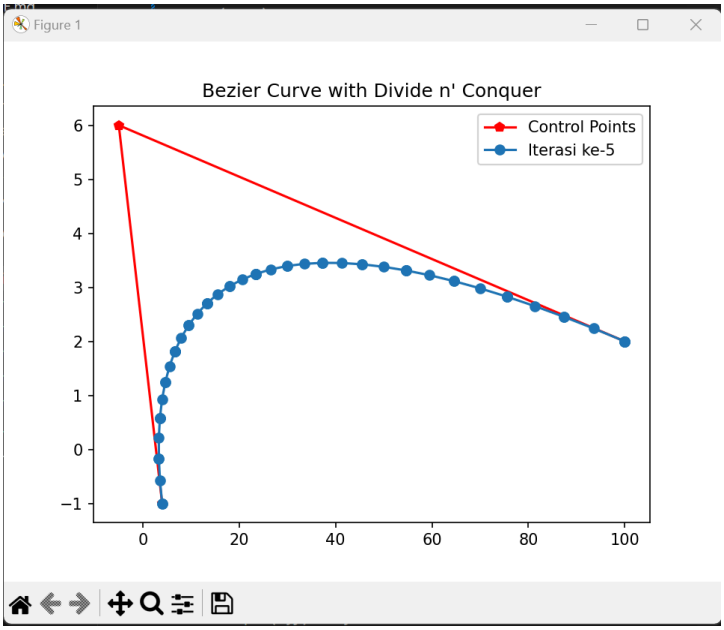
Output 4 :



Input 5 :

```
Masukkan jumlah iterasi: 5
Masukkan titik p0 (x,y): 4,-1
Masukkan titik p1 (x,y): -5,6
Masukkan titik p2 (x,y): 100,2
Time taken: 0.000ms
```

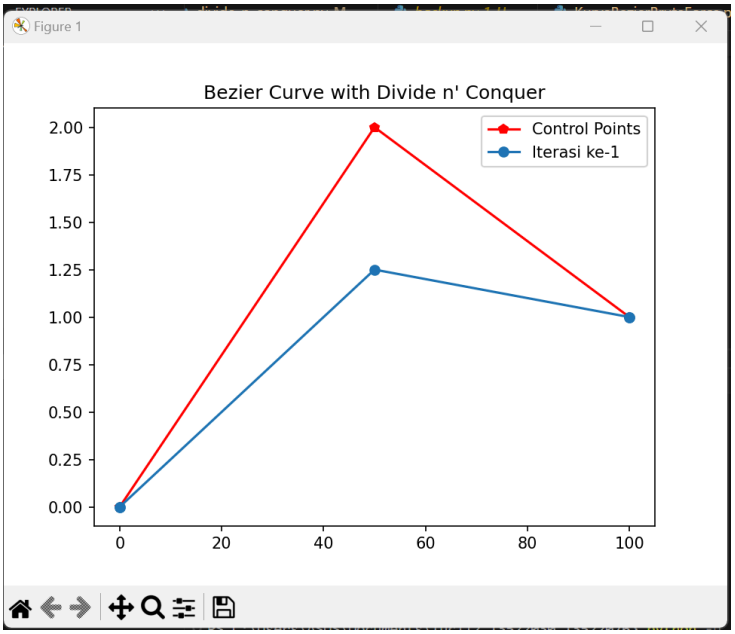
Output 5 :



Input 6 :

```
Masukkan jumlah iterasi: 1
Masukkan titik p0 (x,y): 0,0
Masukkan titik p1 (x,y): 50,2
Masukkan titik p2 (x,y): 100,1
Time taken: 0.000ms
```

Output 6 :

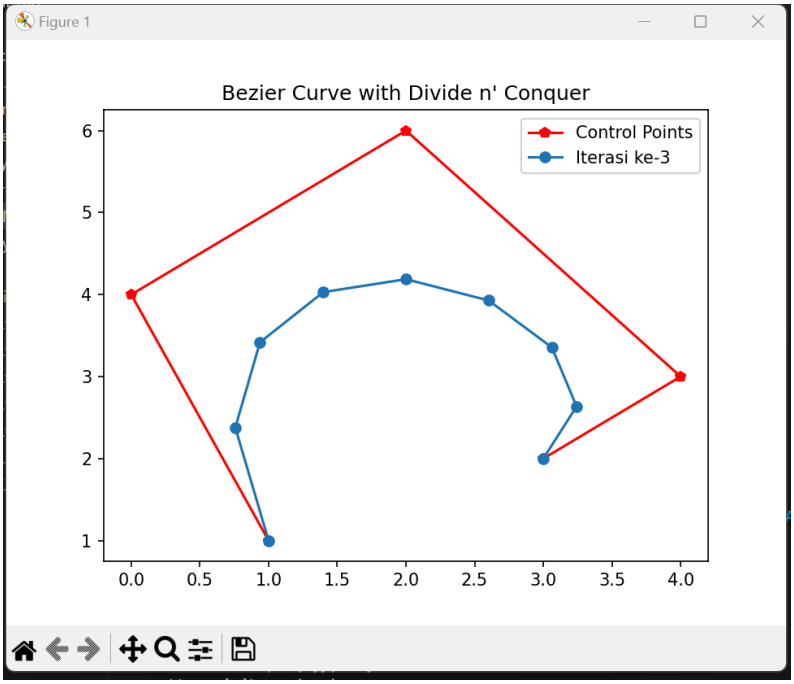


c. Test case Algoritma Divide and Conquer n titik (Bonus)

Input 1 :

```
Masukkan jumlah iterasi : 3
Masukkan jumlah titik yang diinginkan : 5
Masukkan titik p1 (x,y): 1,1
Masukkan titik p2 (x,y): 0,4
Masukkan titik p3 (x,y): 2,6
Masukkan titik p4 (x,y): 4,3
Masukkan titik p5 (x,y): 3,2
Time taken: 0.000ms
```

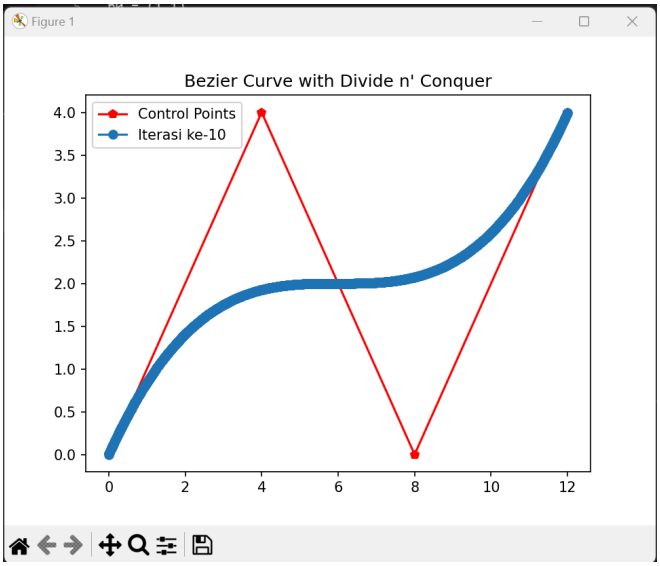
Output 1 :



Input 2 :

```
Masukkan jumlah iterasi : 10
Masukkan jumlah titik yang diinginkan : 4
Masukkan titik p1 (x,y): 0,0
Masukkan titik p2 (x,y): 4,4
Masukkan titik p3 (x,y): 8,0
Masukkan titik p4 (x,y): 12,4
Time taken: 19.845ms
```

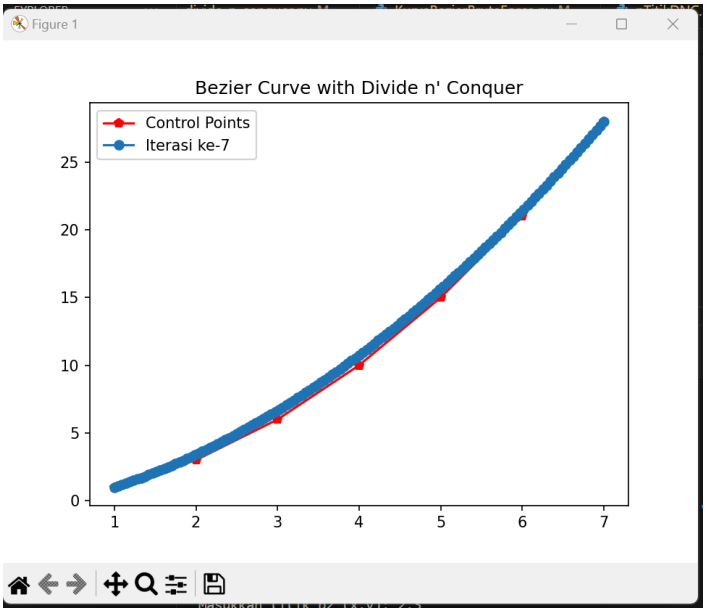
Output 2 :



Input 3 :

```
Masukkan jumlah iterasi : 7
Masukkan jumlah titik yang diinginkan : 7
Masukkan titik p1 (x,y): 1,1
Masukkan titik p2 (x,y): 2,3
Masukkan titik p3 (x,y): 3,6
Masukkan titik p4 (x,y): 4,10
Masukkan titik p5 (x,y): 5,15
Masukkan titik p6 (x,y): 6,21
Masukkan titik p7 (x,y): 7,28
Time taken: 5.000ms
```

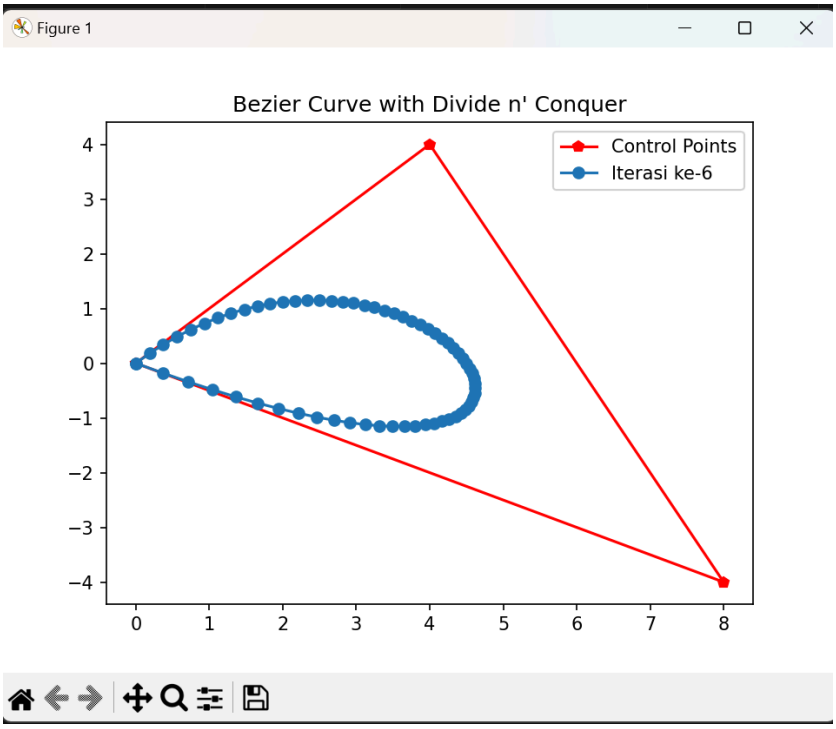
Output 3 :



Input 4 :

```
Masukkan jumlah iterasi : 6
Masukkan jumlah titik yang diinginkan : 4
Masukkan titik p1 (x,y): 0,0
Masukkan titik p2 (x,y): 4,4
Masukkan titik p3 (x,y): 8,-4
Masukkan titik p4 (x,y): 0,0
Time taken: 3.510ms
```

Output 4 :



V. Perbandingan Algoritma *Brute Force* dan *Divide and Conquer*

Algoritma *Brute Force* dan *Divide and Conquer* memiliki pendekatan berbeda untuk penyelesaian persoalan, masing-masing memiliki kelebihan dan kekurangan. Secara umum, pendekatan *Divide and Conquer* lebih efisien daripada *Brute Force* karena karakteristik *Divide and Conquer* yaitu memecah persoalan menjadi sub-persoalan yang lebih mudah dan menggabungkan semua penyelesaian sub-persoalan menjadi satu solusi global. *Brute Force* memiliki algoritma, yang di banyak kasus, kurang efisien karena biasanya menggunakan pendekatan *exhaustive search*.

Pada konteks program ini, kita akan mencari tahu apakah benar algoritma *Divide and Conquer* akan lebih efisien dibandingkan *Brute Force*. Pada algoritma *Brute Force* akan dilakukan proses append R0 ke array hasil sebanyak $2^n + 1$ kali sehingga kompleksitas waktunya adalah,

$$T(n) = 2^n + 1$$

Maka didapatkan **kompleksitas *brute force* = O(T(n)) = O(2^n)**.

Algoritma *Divide and Conquer* memiliki bentuk kompleksitas sebagai berikut,

$$T(n) = aT(n-1) + cn$$

cn = 1, karena komparasi, pemanggilan mid_point, dan mid_section semua memiliki nilai O(1).

a = 2, karena memanggil rekursi dua kali (left dan right)

T(1) = 1, karena basis

T(2) = 2T(1) + 1 = 2.1 + 1 = 3

T(3) = 2T(2) + 1 = 2.3 + 1 = 7

T(4) = 2T(3) + 1 = 2.7 + 1 = 15

T(5) = 2T(4) + 1 = 2.15 + 1 = 31

Didapatkan sebuah pola yaitu $T(n) = 2^n - 1$, sehingga didapatkan **kompleksitas *divide and conquer* = $O(n) = O(2^n - 1) = O(2^n)$**.

Dari hasil kompleksitas kedua algoritma, didapatkan bahwa kedua algoritma memiliki kompleksitas O(n) yang sama. Terdapat sedikit perbedaan pada kompleksitas T(n) dimana kompleksitas T(n) milik *divide and conquer* lebih kecil atau cepat, tapi tidak signifikan.

VI. Implementasi Bonus

A. *Divide and Conquer* untuk n titik control

Penyelesaian kurva bezier dengan lebih dari 3 titik kontrol menggunakan algoritma Divide and Conquer memiliki prinsip yang sama dengan dengan kasus yang memiliki hanya 3 titik kontrol. Perbedaannya terletak pada cara untuk mendapatkan titik tengahnya.

Pada implementasi bonus ini digunakan fungsi tambahan untuk mencari titik setiap iterasi. Fungsi ini menerima masukan berupa control point dan array kosong yang digunakan untuk melakukan pemanggilan secara rekursif. Fungsi ini mengembalikan titik hasil beserta control point baru untuk melakukan iterasi berikutnya apabila dibutuhkan. Berikut merupakan code untuk fungsi tersebut.

```
def titikAproksimasi(control_point, pinggiran) :

    if len(control_point) <= 1 :

        return control_point, control_point
```

```
    else :

        titik_aproksimasi = []

        for i in range(len(control_point)-1) :


titik_aproksimasi.append(mid_point(control_point[i],control_po
int[i+1]))

        titik, pinggir =
titikAproksimasi(titik_aproksimasi,pinggiran)

        pinggir = np.concatenate(([control_point[0]], pinggir,
[control_point[-1]]))

        return titik, pinggir
```

Adapun untuk fungsi utamanya sebagai berikut.

```
def nAryDNC(control_point, iterasi) :

    new_control_point, old_control_point =
titikAproksimasi(control_point,[])

    if iterasi == 1 :

        return new_control_point

    else :

        mid = len(old_control_point)//2

        awal = old_control_point[:mid+1]

        akhir = old_control_point[mid:]

        res = np.array(nAryDNC(awal,iterasi-1))

        res = np.concatenate((res, new_control_point))
```

```
        res = np.concatenate((res,nAryDNC(akhir, iterasi-1)))

        return res
```

Fungsi diatas menerima masukan control point dan jumlah iterasi dan menghasilkan titik-titik kurva bezier kecuali 2 titik kontrol point bagian pinggir kiri dan kanan. Sehingga penggunaan keseluruhan fungsi sebagai berikut. (Catatan : arr_point adalah array dari control point)

```
res = nAryDNC(arr_point, iterasi)

res = np.concatenate(([arr_point[0]], res, [arr_point[-1]]))
```

B. Visualisasi Kurva

Proses visualisasi hasil kurva bezier dilakukan dengan menggunakan library matplotlib yang tersedia di python. Visualisasi yang diimplementasikan berupa penampilan kurva dari iterasi 1 hingga n, dengan n sebagai input iterasi. Visualisasi ini bertujuan untuk melihat proses penghalusan kurva seiring bertambahnya nilai iterasi. Proses penampilan iterasi 1 hingga n dilakukan dengan menggunakan for loop. Berikut adalah fungsi visualisasi,

```python
#Visual Bonus Function
def plotAnimation(ip, n):
    for i in range(n):
        res = []
        Divide_n_Conquer(i+1, ip[0], ip[1], ip[2], res)
        res = np.concatenate(([ip[0]], res, [ip[-1]]))
        x_res = [x.getAbsis() for x in res]
        y_res = [y.getOrdinat() for y in res]
        x_input_points = [x.getAbsis() for x in ip]
        y_input_points = [y.getOrdinat() for y in ip]
        plt.clf()
        plt.plot(x_input_points, y_input_points, marker="p", c="r")
        plt.plot(x_res, y_res, marker="o", label=f"Iterasi ke-{i+1}")
        plt.title("Bezier Curve with Divide n' Conquer")
        plt.legend()
        plt.pause(1.25)
```

Pada fungsi plotAnimation() di atas, hasil grafik ditampilkan secara iteratif dengan memanggil fungsi dengan parameter iterasi berupa variabel looping (i). Dalam looping, plot.clf() dilakukan untuk mengosongkan tampilan grafik tiap iterasi lalu setelah itu dilakukan plotting titik-titik yang ditampilkan secara iteratif. Iterasi ditambah plt.pause() dengan tujuan agar setiap iterasi tampilan di matplotlib terdapat jeda yang dapat diatur.

VII. Lampiran

Link Github : https://github.com/Akmal2205/Tucil2_13522030_13522076

Poin	Ya	Tidak
1. Program berhasil dijalankan	✅	
2. Program dapat melakukan visualisasi kurva Bezier	✅	
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	✅	