Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Semester 2 Tahun 2023/2024

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

NIM : 13522026

Nama : Rici Trisna Putra

Kelas : K02

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

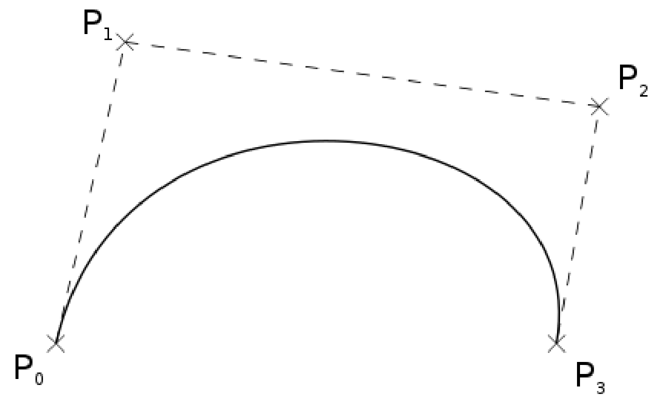
INSTITUT TEKNOLOGI BANDUNG

2024

IF2211 Strategi Algoritma
Tugas Kecil 2

BAB I

Deskripsi Masalah



Gambar 1.1 Kurva Bézier Kubik

(Sumber: https://id.wikipedia.org/wiki/Kurva_B%C3%A9zier)

Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol, yang menentukan bentuk dan arah kurva. Cara membuatnya cukup mudah, yaitu dengan menentukan titik-titik kontrol dan menghubungkannya dengan kurva. Kurva Bézier memiliki banyak kegunaan dalam kehidupan nyata, seperti pen tool, animasi yang halus dan realistis, membuat desain produk yang kompleks dan presisi, dan membuat font yang indah dan unik. Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan.

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P0 sampai Pn , dengan n disebut order (n = 1 untuk linier, n = 2 untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) umumnya tidak terletak pada kurva. Pada gambar 1 diatas, titik kontrol pertama adalah P0, sedangkan titik kontrol terakhir adalah P3. Titik kontrol P1 dan P2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P0 dan P1 yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan **kurva Bézier linier.** Misalkan terdapat sebuah titik Q0 yang berada pada garis yang dibentuk oleh P0 dan P1, maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh B(t) dari P0 ke P1. Misalnya ketika t = 0.25, maka B(t) adalah seperempat jalan dari titik P0 ke P1. sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan B(t) membentuk sebuah garis lurus dari P0 ke P1.

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P2, dengan P0 dan P2 sebagai titik kontrol awal dan akhir, dan P1 menjadi titik kontrol antara. Dengan menyatakan titik Q1 terletak diantara garis yang menghubungkan P1 dan P2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q0 berada, maka dapat dinyatakan sebuah titik baru, R0 yang berada diantara garis yang menghubungkan Q0 dan Q1 yang bergerak membentuk **kurva Bézier kuadratik** terhadap titik P0 dan P2. Berikut adalah uraian persamaannya.

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

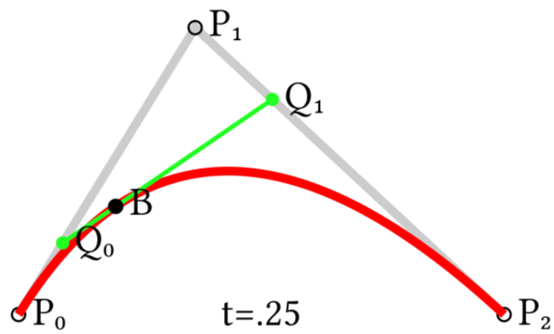
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

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

dengan melakukan substitusi nilai Q0 dan Q1, maka diperoleh persamaan sebagai berikut.

$$R_0 = B(t) = (1-t)^2 P_0 + (1-t)tP_1 \; + \; t^2 P_2, \qquad t \in [0,1]$$

Berikut adalah ilustrasi dari kasus diatas.



Gambar 1.2 Pembentukan Kurva Bézier Kuadratik.

(Sumber: https://simonhalliday.com/2017/02/15/quadratic-bezier-curve-demo/)

Proses ini dapat juga diaplikasikan untuk jumlah titik yang lebih dari tiga, misalnya empat titik akan menghasilkan **kurva Bézier kubik**, lima titik akan menghasilkan **kurva Bézier kuartik**, dan seterusnya. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya.

IF2211 Strategi Algoritma
Tugas Kecil 2

$$S_0 = B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3, \qquad t \in [0,1]$$

$$T_0 = B(t) = (1-t)^4 P_0 + 4(1-t)^3 t P_1 + 6(1-t)^2 t^2 P_2 + 4(1-t) t^3 P_3 + t^4 P_4, \qquad t \in [0,1]$$

Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka Anda diminta untuk mengimplementasikan **pembuatan kurva Bézier** dengan algoritma titik tengah berbasis ***divide and conquer.***

Bab II

Teori Singkat

Metode *Divide and Conquer* merupakan desain strategi yang menyarankan kita untuk membagi masalah berukuran n menjadi beberapa sub himpunan berbeda sehingga terbentuk k sub masalah. Sub masalah tersebut kemudian harus masing-masing diselesaikan dan kemudian solusinya digabung dengan suatu metode untuk mencapai solusi dari masalah awal. Pada umumnya sub masalah yang dihasilkan dari pembagian masalah di strategi ini memiliki tipe yang sama dengan masalah awal sehingga sub masalah tersebut dapat terus dibagi hingga mencapai sub masalah yang tidak dapat dibagi lagi yang lalu kemudian diselesaikan dan digabung untuk mendapat solusi masalah awal.

Dalam notasi algoritma strategi *Divide and Conquer* dapat digeneralisir menjadi seperti berikut:

```
Algorithm DAndC(P)
{
    if Small(P) then return S(P);
    else
    {
        divide P into smaller instances P1, P2, ..., Pk, k ≥ 1;
        Apply DAndC to each of these subproblems;
        return Combine(DAndC(P1),DAndC(P2),...,DAndC(Pk));
    }
}
```

Gambar 2.1 Strategi *Divide and Conquer* Secara General

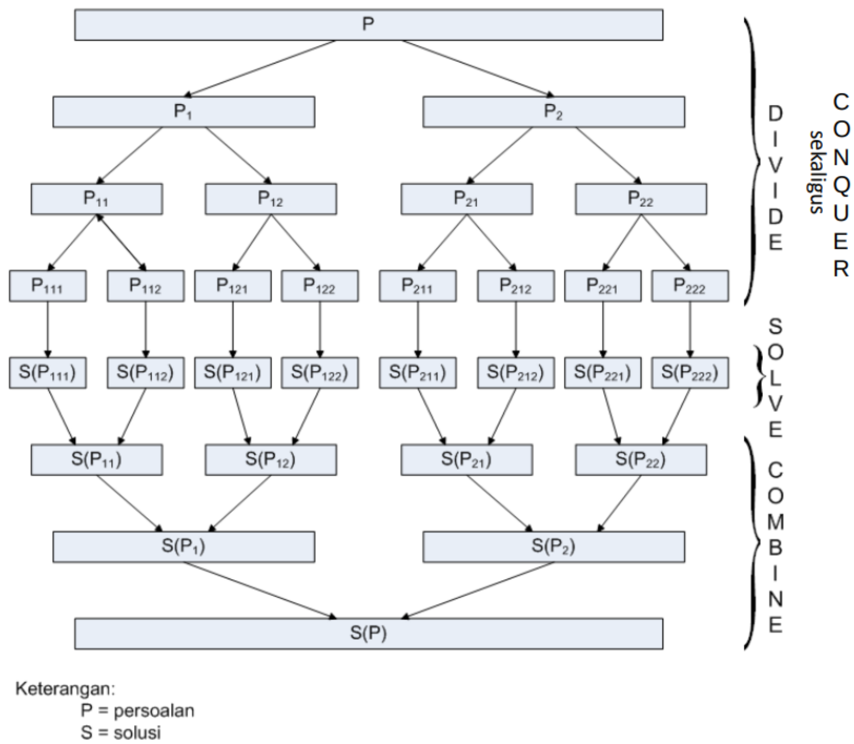
(Sumber: Computer Algorithms - Horowitz et al. - 1998 p.128)(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf)

Suatu algoritma *Divide and Conquer* memiliki sebuah fungsi untuk mengecek apakah suatu masalah P sudah terlalu kecil untuk bisa dibagi yakni Small(P). Apabila P benar sudah terlalu kecil maka Small(P) akan bernilai true dan algoritma akan melakukan fungsi S, yakni fungsi yang akan mencari solusi pada sub masalah terkecil tersebut. Apabila P masih dapat dibagi maka Small(P) bernilai false dan problem P akan dipecah menjadi beberapa bagian dan tiap bagian tersebut akan di aplikasikan pada algoritma tersebut kemudian solusi dari tiap sub masalah tersebut akan digabung dengan fungsi Combine untuk mendapatkan solusi dari P.

Dari penjelasan di atas kita dapat menyimpulkan bahwa strategi *Divide and Conquer* memiliki beberapa bagian penting yakni Divide, Conquer (Solve), dan Combine.(Sumber:

IF2211 Strategi Algoritma
Tugas Kecil 2

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf)



Gambar 2.2 Diagram Algoritma *Divide and Conquer*

(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf)

Adapun untuk kompleksitas waktu dari algoritma *Divide and Conquer* dapat dihitung dengan rumus sebagai berikut:

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \ldots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

Gambar 2.3 Rumus Kompleksitas Waktu Algoritma *Divide and Conquer*

(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf)

Dimana :

- T(n) : kompleksitas waktu penyelesaian persoalan P yang berukuran n

- g(n) : kompleksitas waktu untuk mencari solusi problem n yang merupakan problem terkecil

- T(n1) + T(n2) + … + T(nr) : kompleksitas waktu untuk memproses setiap sub masalah

- f(n) : kompleksitas waktu untuk melakukan penggabungan solusi sub masalah

Sehingga untuk algoritma *Divide and Conquer* yang memecah masalah menjadi dua masalah dengan ukuran yang sama maka kompleksitasnya:

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ 2T(n/2) + f(n) & , n > n_0 \end{cases}$$

Gambar 2.4 Rumus Kompleksitas Waktu Algoritma *Divide and Conquer* dengan sub masalah berukuran sama

(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf)

BAB III

Implementasi dan Analisis

- Algoritma Brute Force untuk Kurva Bezier Kuadratik dan General

Gambar 3.1

```python
def QuadraticBezierBF(initial_point_list,iteration):
    step = 2**(iteration) ##  Membuat jumlah step dari nilai t
    increment = 1 / step ## Increment dari nilai t

    ## Membuat daftar nilai t yang akan digunakan untuk mengenerasi titik dengan BF
    listOfTValue = [i * increment for i in range(1,step+1)]

    ## Membuat list penampung titik yang tergenerasi
    result = [initial_point_list[0]] ## Menambahkan titik awal ke dalam list result
    for i in listOfTValue:
        temp = Casteljau(initial_point_list,i) ## Melakukan linear interpolasi pada tiap nilai t
        result.append(temp)

    return result

def Casteljau(initial_point_list,t): ## De Casteljau's Algorithm

    if len(initial_point_list) == 1: ## Apabila telah didapat satu titik estimasi
        return initial_point_list[0]

    subPointList = [] ## List penampung subpoint sesuai hasil linear interpolation pada nilai t saat ini
    for i in range(0,len(initial_point_list)-1): ## Mengenerasi subpoint sesuai nilai t dan list subpoint saat ini
        Point1, Point2 = initial_point_list[i:i+2]
        subPoint = lerp(Point1,Point2,t)
        subPointList.append(subPoint)

    result = Casteljau(subPointList,t) ## Melakukan linear interpolation pada subpoint list hingga didapat satu titik
    return result

def lerp(Point1,Point2,t): ## Linear interpolation
    return ((1 - t) * Point1[0] + t * Point2[0], (1 - t) * Point1[1] + t * Point2[1])
```

Implementasi De Casteljau Untuk Pembentukan Kurva Bezier Kuadratik

Gambar 3.2

```python
def GeneralBezierBF(initial_point_list,iteration):
    step = 2**(iteration) ##  Membuat jumlah step dari nilai t
    increment = 1 / step ## Increment dari nilai t

    ## Membuat daftar nilai t yang akan digunakan untuk mengenerasi titik dengan BF
    listOfTValue = [i * increment for i in range(1,step+1)]

    ## Membuat list penampung titik yang tergenerasi
    result = [initial_point_list[0]] ## Menambahkan titik awal ke dalam list result
    for i in listOfTValue:
        temp = Casteljau(initial_point_list,i) ## Melakukan linear interpolasi pada tiap nilai t
        result.append(temp)

    return result

def Casteljau(initial_point_list,t): ## De Casteljau's Algorithm

    if len(initial_point_list) == 1: ## Apabila telah didapat satu titik estimasi
        return initial_point_list[0]

    subPointList = [] ## List penampung subpoint sesuai hasil linear interpolation pada nilai t saat ini
    for i in range(0,len(initial_point_list)-1): ## Mengenerasi subpoint sesuai nilai t dan list subpoint saat ini
        Point1, Point2 = initial_point_list[i:i+2]
        subPoint = lerp(Point1,Point2,t)
        subPointList.append(subPoint)

    result = Casteljau(subPointList,t) ## Melakukan linear interpolation pada subpoint list hingga didapat satu titik
    return result

def lerp(Point1,Point2,t): ## Linear interpolation
    return ((1 - t) * Point1[0] + t * Point2[0], (1 - t) * Point1[1] + t * Point2[1])
```

Implementasi De Casteljau Untuk Pembentukan Kurva Bezier General

IF2211 Strategi Algoritma
Tugas Kecil 2

Program menggunakan algoritma De Casteljau untuk dapat membentuk titik-titik estimasi dari kurva bezier. Dari jumlah iterasi yang diberikan pengguna program membuat list nilai t untuk selanjutnya digunakan untuk membuat titik estimasi dengan pemanggilan rekursi fungsi Casteljau yang merupakan implementasi dari algoritma De Casteljau. Algoritma De Casteljau ini memanfaatkan fungsi lerp yang merupakan fungsi linear interpolation untuk melakukan interpolasi antara dua titik apabila diberikan suatu nilai t tertentu yakni dengan rumus (1 – t) P0 + t * P1. Algoritma De Casteljau akan terus memanggil dirinya sendiri dan memanfaatkan fungsi lerp hingga mendapatkan satu buah titik estimasi yang berada pada kurva.

Untuk suatu iterasi sebanyak m akan terjadi 2^m pemanggilan algoritma De Casteljau (terdapat 2^m nilai t berbeda). Namun diketahui pula bahwa Algoritma De Casteljau memiliki kompleksitas $O(n^2)$ apabila dilihat dari jumlah penjumlahan dan perkalian, maka secara garis besar algoritma ini secara general memiliki kompleksitas $O(2^m n^2)$. Sehingga untuk versi Kuadratiknya kompleksitasnya adalah $(4 . 2^m)$.

- Algoritma Divide and Conquer untuk Kurva Bezier Kuadratik

```
def QuadraticBezierDAC(Point1,Point2,Point3,cur_iteration,iteration):
    ## Membentuk dua buah subpoint dan sebuah titik estimasi
    SubPoint1 = SubPoint(Point1,Point2)
    SubPoint2 = SubPoint(Point2,Point3)
    ResultPoint = SubPoint(SubPoint1,SubPoint2)

    ## Apabila iterasi masih berlanjut maka kembali panggil fungsi secara rekursi hingga iterasi selesai
    if cur_iteration < iteration:
        ## Mencari solusi dari dua buah pecahan masalah
        left = QuadraticBezierDAC(Point1, SubPoint1, ResultPoint, cur_iteration + 1, iteration)
        right = QuadraticBezierDAC(ResultPoint, SubPoint2, Point3, cur_iteration + 1, iteration)
        result = Merge(left,right) ## Menggabungkan kedua buah pecahan solusi
        return result

    result = ([Point1, ResultPoint, Point3])
    return result

## Fungsi untuk menggabung dua buah solusi
def Merge(list1,list2):
    result = list1[:-1]
    result.extend(list2)
    return result

## Fungsi untuk mencari midpoint dari dua buah point
def SubPoint(Point1,Point2):
    return ((Point1[0] + Point2[0]) / 2, (Point1[1] + Point2[1]) / 2)
```

Gambar 3.3 Implementasi *Divide and Conquer* Untuk Pembentukan Kurva Bezier Kuadratik

Sesuai gambar diatas program membuat dua buah subpoint (mid point) dengan memanfaatkan fungsi Subpoint() dari 3 buah poin awal yang kemudian kedua subpoint tersebut di cari subpointnya dan dihasilkanlah titik tengah yang merupakan titik estimasi dari kurva bezier (Divide). Lalu dengan adanya parameter iterasi dan iterasi pada saat instan tersebut program akan mengecek apakah iterasi pada instan tersebut merupakan iterasi terakhir, apabila belum maka akan direkursi dengan nilai iterasi yang lebih dalam sehingga mendapat solusi kiri dan solusi kanan (Conquer). Setelah itu kedua solusi tersebut akan digabung menjadi satu dengan menggunakan fungsi Merge() dan hasilnya akan direturn (Combine). Apabila saat itu merupakan iterasi terakhir maka program akan mengembalikan tiga buah poin yakni titik handle pertama, titik estimasi dan titik handle terakhir.

- Algoritma Divide and Conquer untuk Kurva Bezier General

```python
def GeneralBezierDAC(initial_point_list, cur_iteration, iteration):
    # Mengenerasi subpoint dan titik estimasi kemudian membagi masalah menjadi dua
    left_initial_point, right_initial_point = Subdivide(initial_point_list)

    left_initial_point.insert(0,initial_point_list[0]) ## Menambahkan titik handle kurva paling ujung kiri
    right_initial_point.extend([initial_point_list[-1]]) ## Menambahkan titik handle kurva ujung kanan

    if cur_iteration < iteration: ## Apabila iterasi masih belum selesai maka lanjutkan DAC
        ## Mencari solusi dari dua bagian masalah
        left_result = GeneralBezierDAC(left_initial_point,cur_iteration+1,iteration)
        right_result = GeneralBezierDAC(right_initial_point, cur_iteration+1,iteration)
        ## Tahap menggabungkan kedua solusi dari dua bagian yang berbeda
        result = Merge(left_result,right_result)
        return result

    result = [left_initial_point[0], left_initial_point[-1],right_initial_point[-1]]
    return result

def Subdivide(initial_point_list):

    subPointList = [] ## List yang menampung semua subpoint yang digenerasi
    for i in range(0,len(initial_point_list)-1): ## Mengenerasi subpoint
        Point1, Point2 = initial_point_list[i:i+2]
        subPoint = SubPoint(Point1,Point2)
        subPointList.append(subPoint)

    if(len(subPointList) == 1): ## Apabila sudah menghasilkan titik estimasi
        return subPointList, subPointList.copy()

    subPointListCopy = subPointList ## Membuat salinan Subpointlist untuk selanjutnya di cari subpoint untuk titik-titik pada list tersebut
    left, right = Subdivide(subPointListCopy)
    left.insert(0,subPointList[0])
    right.append(subPointList[-1])

    return left, right
      You, 5 hours ago • init
def Merge(list1,list2):
    result = list1 + list2[1:]
    return result

## Fungsi untuk mencari midpoint dari dua buah point
def SubPoint(Point1,Point2):
    return ((Point1[0] + Point2[0]) / 2, (Point1[1] + Point2[1]) / 2)
```

Gambar 3.4 Implementasi *Divide and Conquer* Untuk Pembentukan Kurva Bezier General

Program menggunakan prinsip yang mirip dengan Algoritma *Divide and Conquer* Kuadratik namun diperluas dengan cara menggunakan fungsi Subdivide yang berfungsi untuk memecah problem menjadi dua bagian (Divide). Subdivide menggunakan fungsi Subpoint untuk menemukan titik estimasi sekaligus menghasilkan dua buah list yang dapat di *Divide and Conquer* pada iterasi yang lebih dalam. Sama seperti *Divide and Conquer* Kuadratik maka perlu dicek apakah iterasi adalah yang paling akhir, apabila tidak maka panggil secara rekursif menggunakan dua buah list berbeda yang telah dihasilkan Subdivide(Conquer). Setelah itu hasilnya dapat digabung menjadi satu (Combine) untuk mendapatkan solusi dari masalah awal.

IF2211 Strategi Algoritma
Tugas Kecil 2

Untuk suatu nilai iterasi m maka algoritma ini memanggil fungsi Subdivide sebanyak 2^m yang kemudian Subdivide memanggil fungsi SubPoint. Karena fungsi subpoint merupakan bentuk khusus dari *linear interpolation* (t = 0,5) maka fungsi subdivide mirip dengan fungsi De Casteljau dalam tinjauan kompleksitas yakni $O(n^2)$ sehingga total kompleksitas dari algoritma ini adalah sama dengan *Brute Force* yakni $O(2^m n^2)$. Sehingga untuk versi kuadratnya kompleksitasnya yakni $(4 \cdot 2^m)$. Namun perlu diketahui bahwa fungsi SubPoint sedikit lebih sangkil daripada fungsi *linear interpolation* sehingga untuk algoritma Kurva Bezier yang menerapkan *Divide and Conquer* lebih sangkil daripada yang menerapkan *Brute Force*

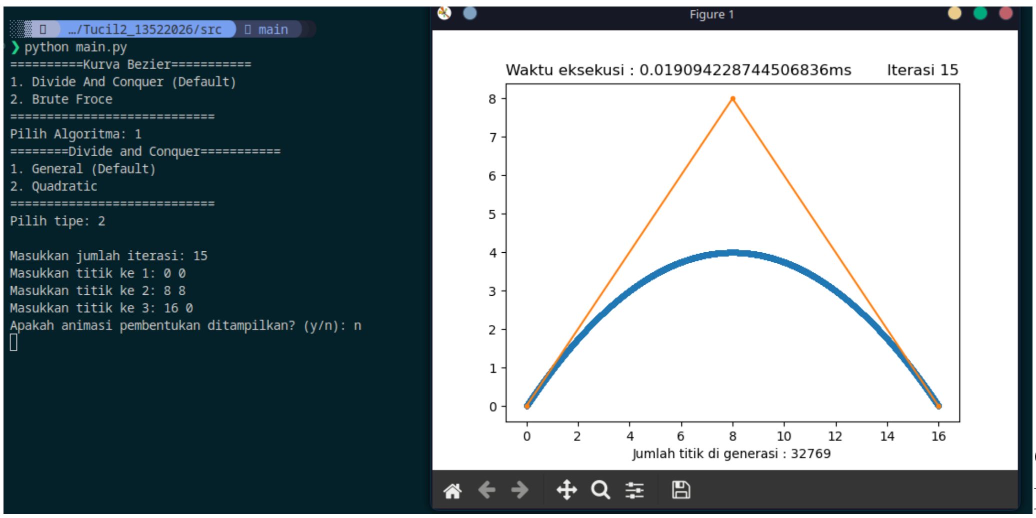
BAB IV

Eksperimen

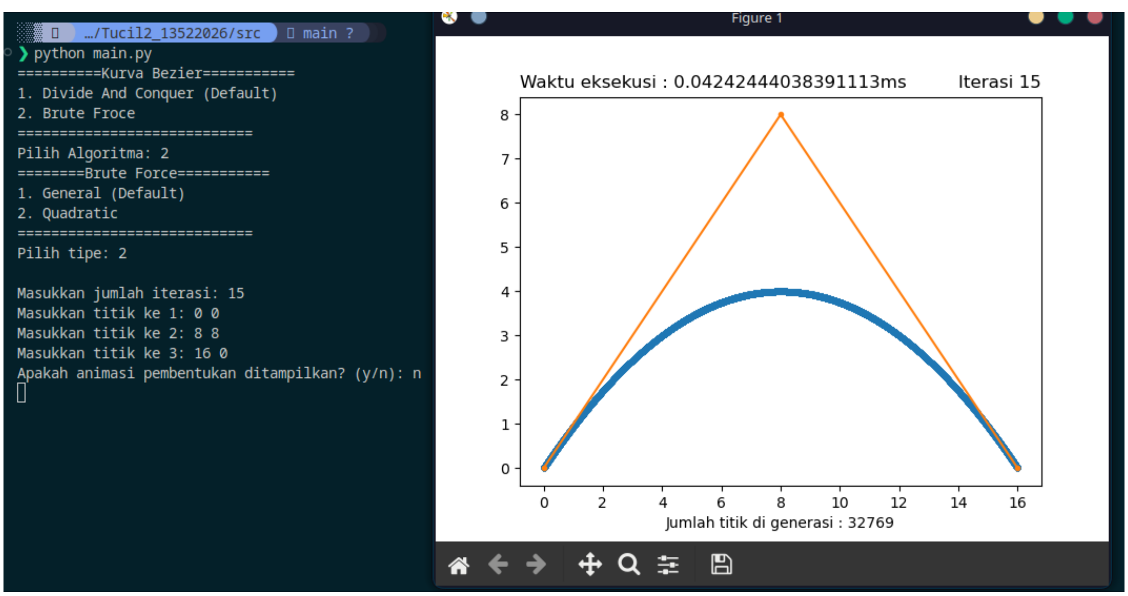
- Test case 1 :

Kuadratik dengan iterasi = 15

Titik = [(0,0),(8,8),(16,0)]



Gambar 4.1 Eksperimen Test Case 1 pada *Divide and Conquer* Bezier Kuadratik



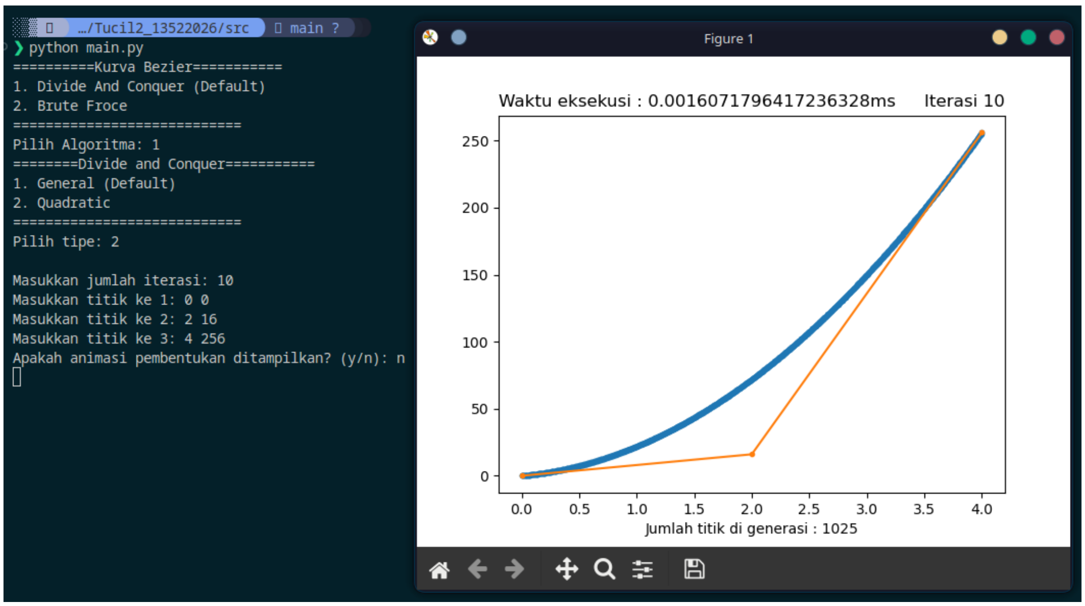
Gambar 4.2 Eksperimen Test Case 1 pada *Brute Force* Bezier Kuadratik

- Test case 2 :

Kuadratik dengan iterasi = 10

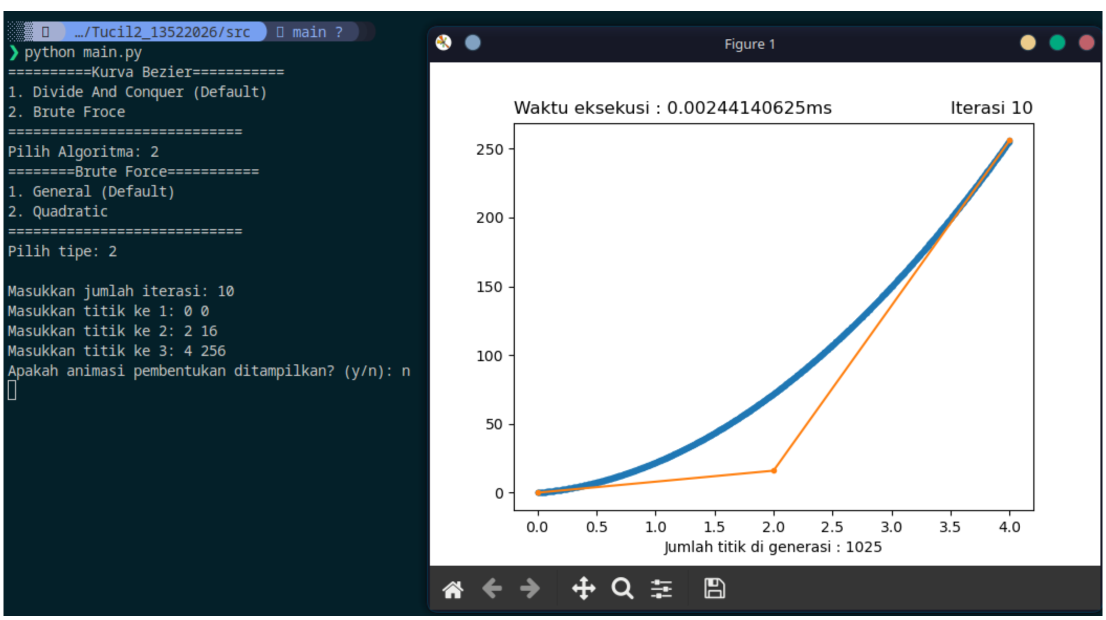
Titik = [(0,0),(2,16),(4,256)]

Gambar 4.3



Eksperimen Test Case 2 pada *Divide and Conquer* Bezier Kuadratik

Gambar 4.4

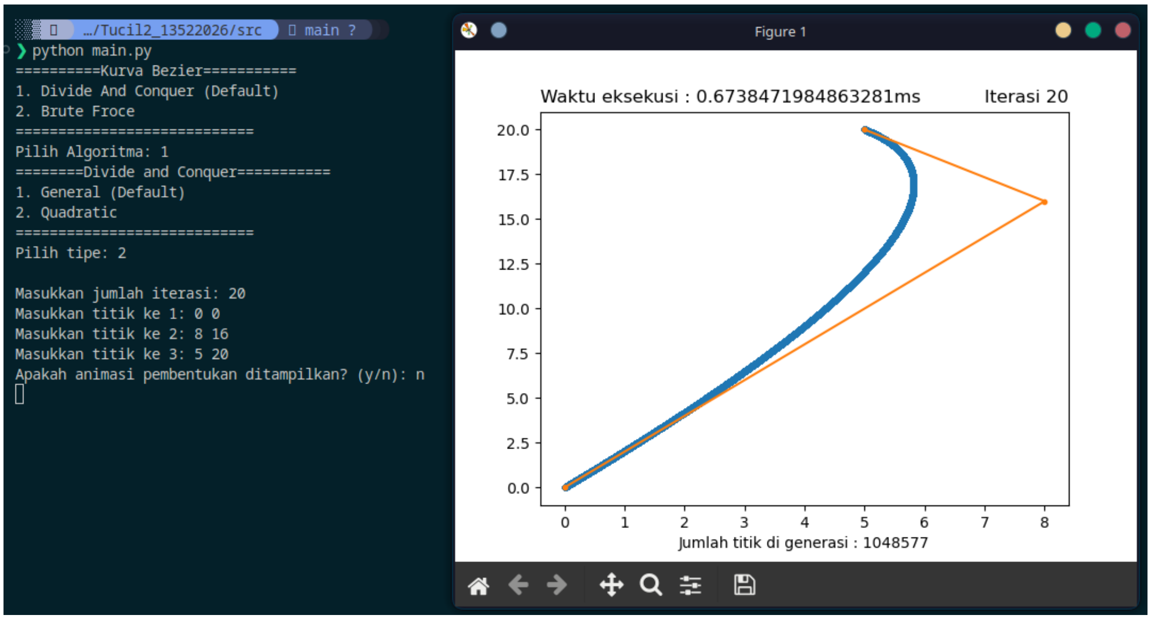


Eksperimen Test Case 2 pada *Brute Force* Bezier Kuadratik

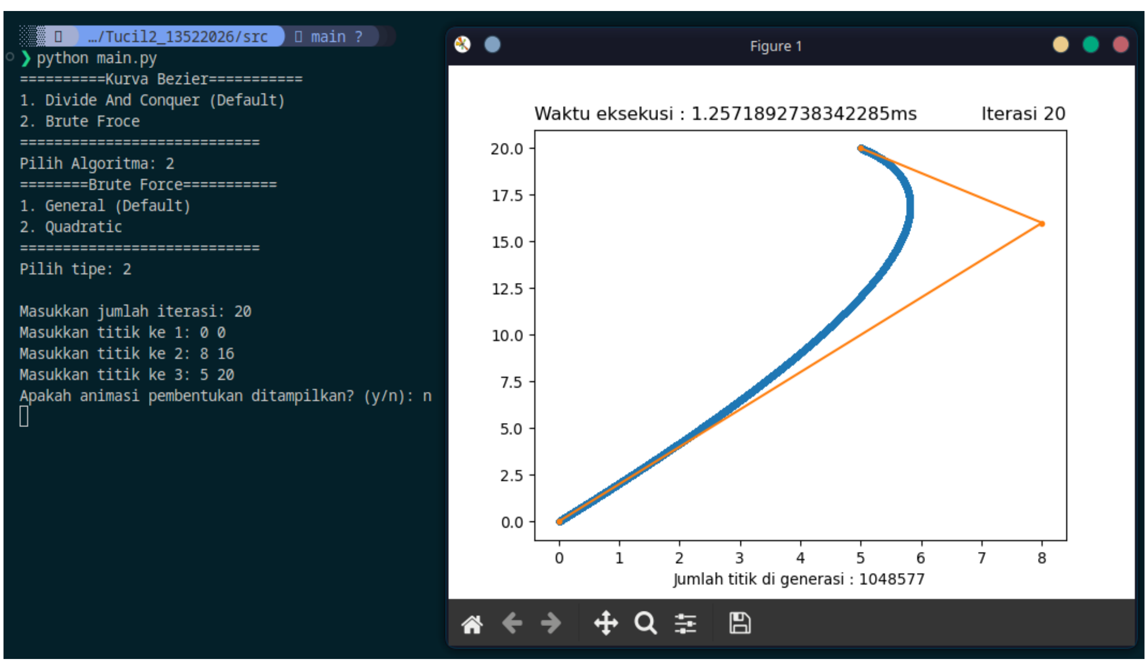
- Test case 3 :

Kuadratik dengan iterasi = 20

Titik = [(0,0),(8,16),(5,20)]



Gambar 4.5 Eksperimen Test Case 3 pada *Divide and Conquer* Bezier Kuadratik



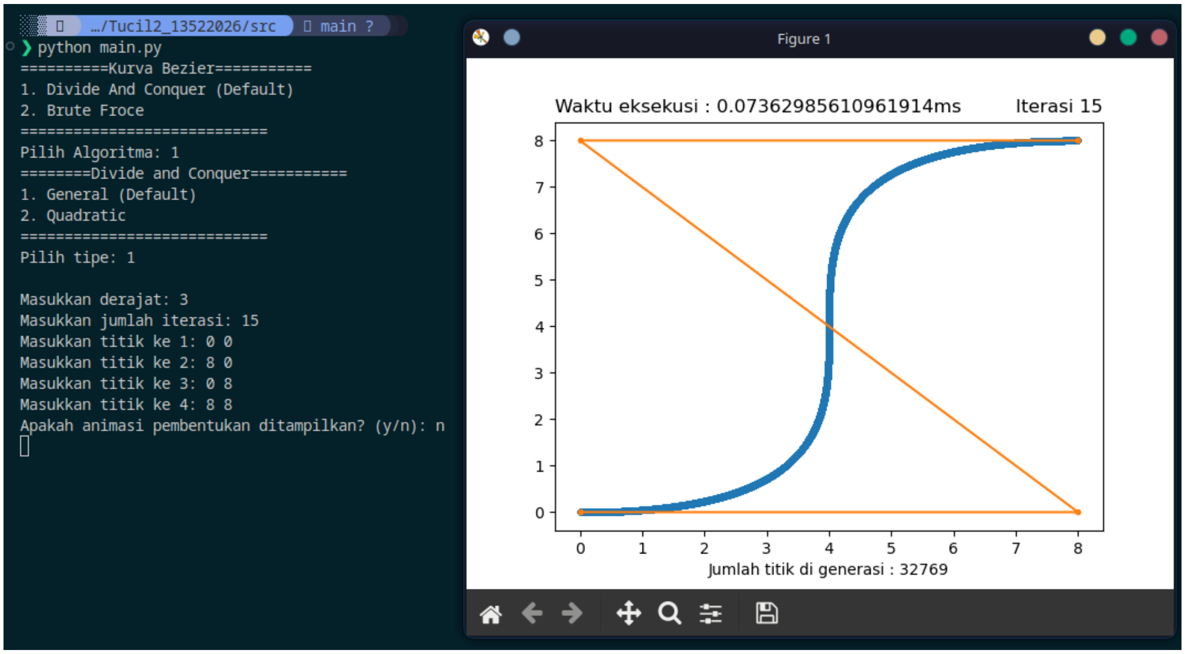
Gambar 4.6 Eksperimen Test Case 3 pada *Brute Force* Bezier Kuadratik

IF2211 Strategi Algoritma
Tugas Kecil 2

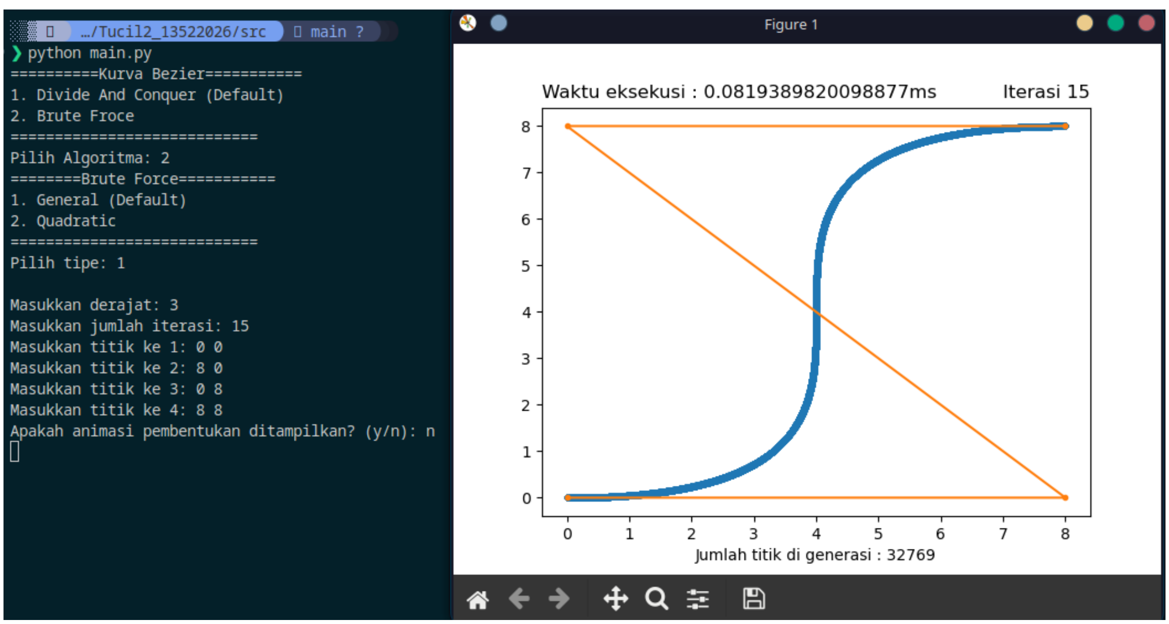
Test case 4 :

General dengan n = 3 dan iterasi = 15

Titik = [(0,0),(8,0),(0,8),(8,8)]



Gambar 4.7 Eksperimen Test Case 4 pada *Divide and Conquer* Bezier General (Kubik)

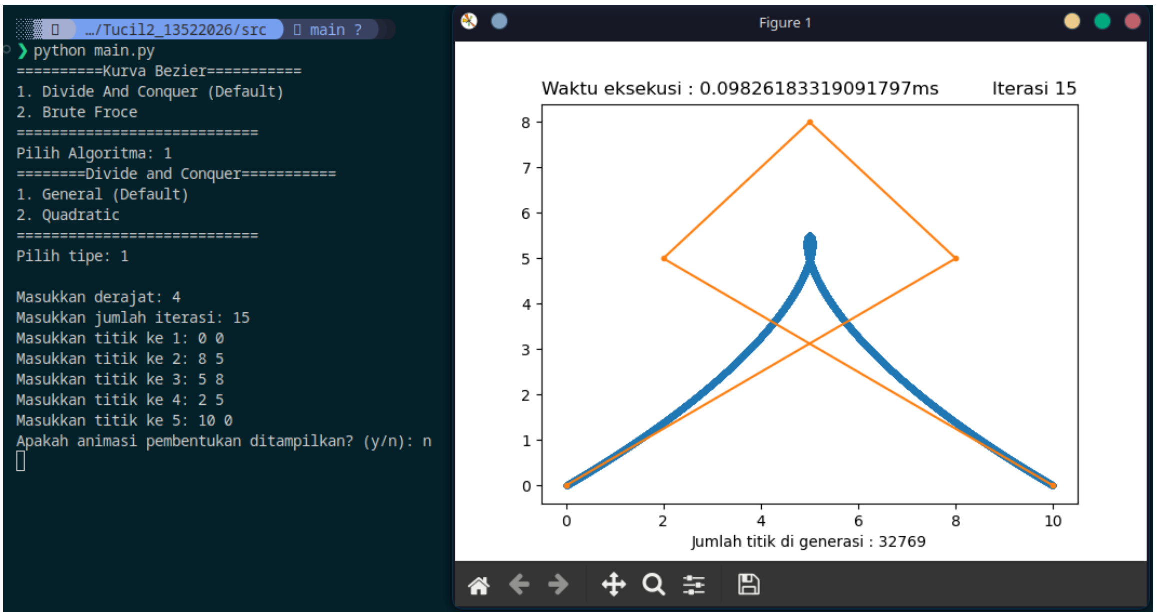


Gambar 4.8 Eksperimen Test Case 4 pada *Brute Force* Bezier General(Kubik)

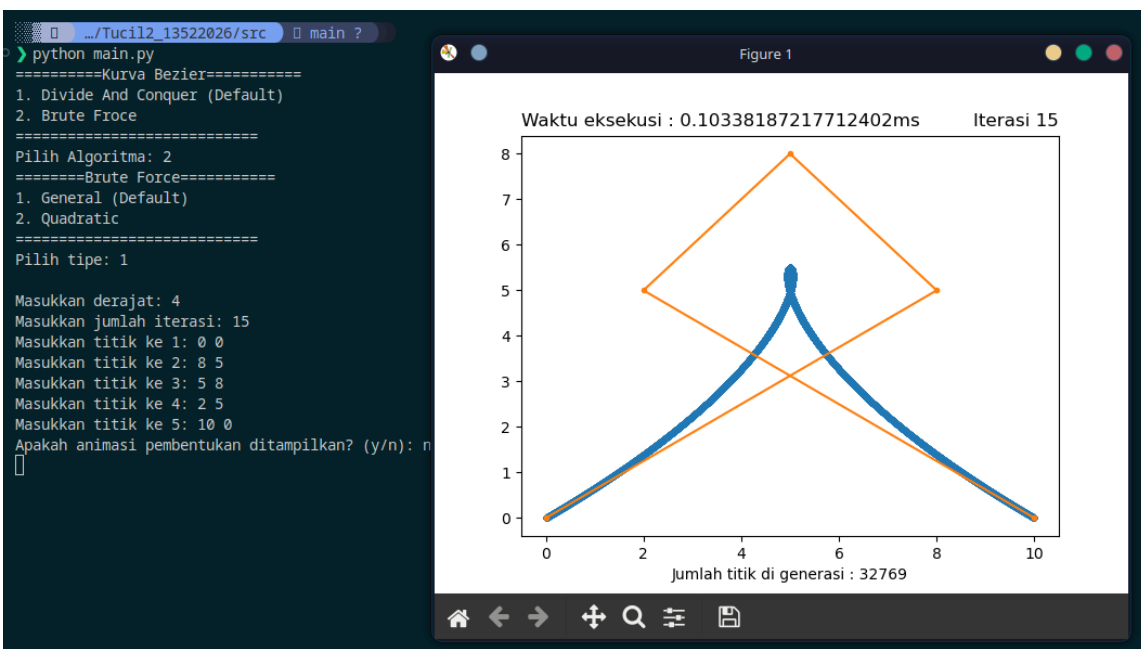
Test case 5 :

General dengan n = 4 dan iterasi = 15

Titik = [(0,0),(8,5),(5,8),(2,5),(10,0)]



Gambar 4.9 Eksperimen Test Case 5 pada *Divide and Conquer* Bezier General

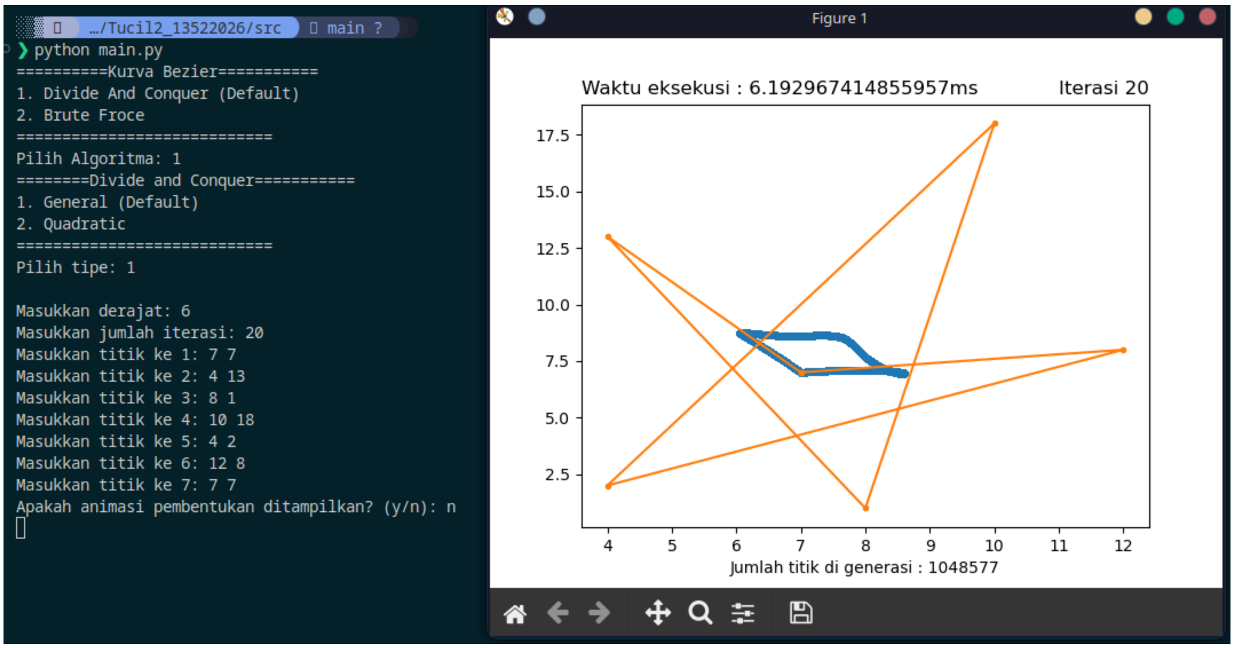


Gambar 4.10 Eksperimen Test Case 5 pada *Brute Force* Bezier General

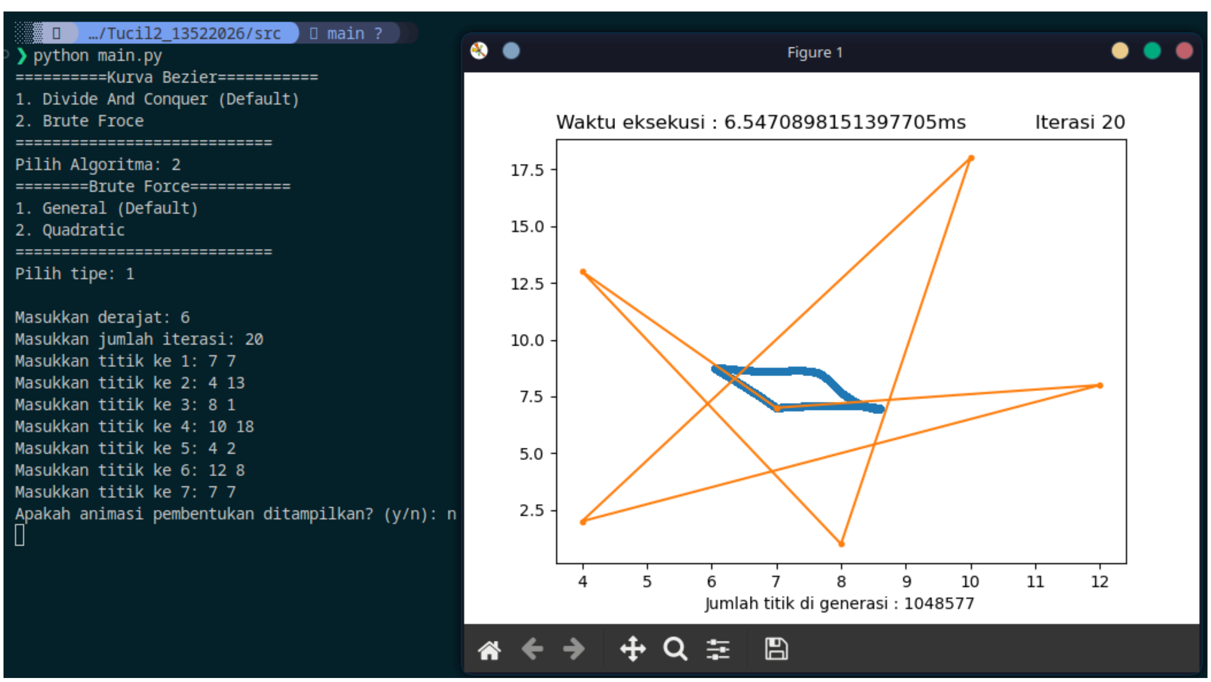
Test case 6 :

General dengan n = 6 dan iterasi = 20

Titik = [(7,7),(4,13),(8,1),(10,18),(4,2), (12, 8),(7,7)]



Gambar 4.11 Eksperimen Test Case 6 pada *Divide and Conquer* Bezier General



Gambar 4.12 Eksperimen Test Case 6 pada *Brute Force* Bezier General

IF2211 Strategi Algoritma
Tugas Kecil 2

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.		

Link Repository Github

https://github.com/RiciTrisnaP/Tucil2_13522026