

**Tugas Kecil 2 IF2211**  
**Strategi Algoritma Semester II tahun 2023/2024**  
**Membangun Kurva Bézier dengan Algoritma Titik Tengah**  
**berbasis Divide and Conquer**



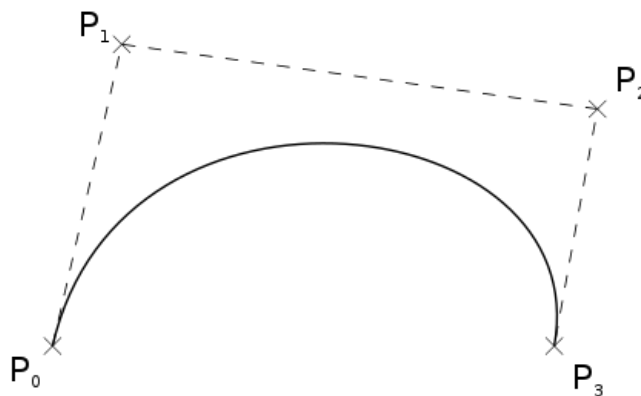
Disusun oleh:  
Bagas Sambega Rosyada 13522071  
Fedrianz Dharma 13522090  
**Program Studi Teknik Informatika**  
**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**2024**

## Daftar Isi

<b>Bab I.....</b>	<b>3</b>
<b>Bab II.....</b>	<b>7</b>
<b>Bab III.....</b>	<b>8</b>
<b>Bab IV.....</b>	<b>10</b>
A. Source Code.....	10
B. Test Case 3 Titik.....	17
C. Test Case n-Titik.....	28
D. Test Case CLI.....	35
<b>Bab V.....</b>	<b>40</b>
<b>Bab VI.....</b>	<b>41</b>
<b>Lampiran.....</b>	<b>43</b>

# Bab I

## Deskripsi Masalah



**Gambar 1.** Kurva Bézier Kubik

(Sumber: [https://id.wikipedia.org/wiki/Kurva\\_B%C3%A9zier](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  $P_0$  sampai  $P_n$ , 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  $P_0$ , sedangkan titik kontrol terakhir adalah  $P_3$ . Titik kontrol  $P_1$  dan  $P_2$  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  $P_0$  dan  $P_1$  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  $Q_0$  yang berada pada garis yang dibentuk oleh  $P_0$  dan  $P_1$ , maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

dengan  $t$  dalam fungsi kurva Bézier linier menggambarkan seberapa jauh  $B(t)$  dari  $P_0$  ke  $P_1$ . Misalnya ketika  $t = 0.25$ , maka  $B(t)$  adalah seperempat jalan dari titik  $P_0$  ke  $P_1$ .

sehingga seluruh rentang variasi nilai  $t$  dari 0 hingga 1 akan membuat persamaan  $B(t)$  membentuk sebuah garis lurus dari  $P_0$  ke  $P_1$ .

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja  $P_2$ , dengan  $P_0$  dan  $P_2$  sebagai titik kontrol awal dan akhir, dan  $P_1$  menjadi titik kontrol antara. Dengan menyatakan titik  $Q_1$  terletak diantara garis yang menghubungkan  $P_1$  dan  $P_2$ , dan membentuk kurva Bézier linier yang berbeda dengan kurva letak  $Q_0$  berada, maka dapat dinyatakan sebuah titik baru,  $R_0$  yang berada diantara garis yang menghubungkan  $Q_0$  dan  $Q_1$  yang bergerak membentuk **kurva Bézier kuadratik** terhadap titik  $P_0$  dan  $P_2$ . Berikut adalah uraian persamaannya.

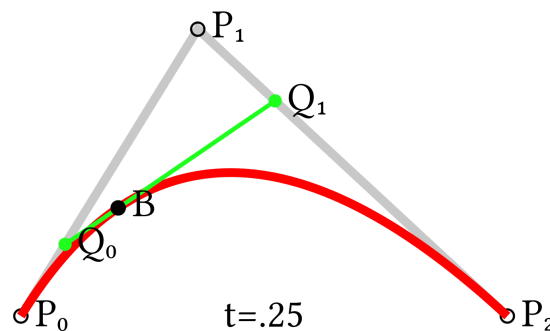
$$Q_0 = B(t) = (1 - t)P_0 + tP_1, t \in [0, 1]$$

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

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

dengan melakukan substitusi nilai  $Q_0$  dan  $Q_1$ , maka diperoleh persamaan sebagai berikut.

$$R_0 = B(t) = (1 - t)^2 P_0 + 2(1 - t)t P_1 + t^2 P_2, t \in [0, 1]$$



**Gambar 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.

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

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

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*.

### Ilustrasi kasus

Idenya cukup sederhana, relatif mirip dengan pembahasan sebelumnya, dan dilakukan secara iteratif. Misalkan terdapat tiga buah titik,  $P_0$ ,  $P_1$ , dan  $P_2$ , dengan titik  $P_1$  menjadi titik kontrol antara, maka:

- Buatlah sebuah titik baru  $Q_0$  yang berada di tengah garis yang menghubungkan  $P_0$  dan  $P_1$ , serta titik  $Q_1$  yang berada di tengah garis yang menghubungkan  $P_1$  dan  $P_2$ .
- Hubungkan  $Q_0$  dan  $Q_1$  sehingga terbentuk sebuah garis baru.
- Buatlah sebuah titik baru  $R_0$  yang berada di tengah  $Q_0$  dan  $Q_1$ .
- Buatlah sebuah garis yang menghubungkan  $P_0 - R_0 - P_2$ .

Melalui proses di atas, telah dilakukan 1 buah iterasi dan diperoleh sebuah “kurva” yang belum cukup mulus dengan aproksimasi 3 buah titik. Untuk membuat sebuah kurva yang lebih baik, perlu dilakukan iterasi lanjutan. Berikut adalah prosedurnya.

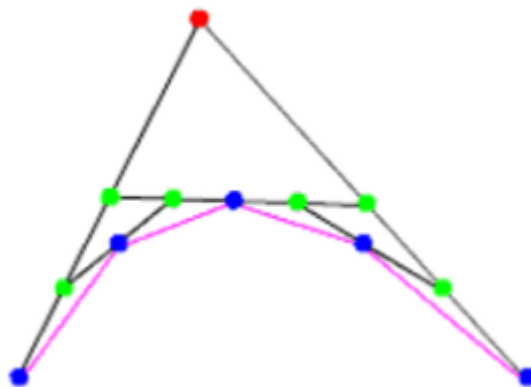
Buatlah beberapa titik baru, yaitu  $S_0$  yang berada di tengah  $P_0$  dan  $Q_0$ ,  $S_1$  yang berada di tengah  $Q_0$  dan  $R_0$ ,  $S_2$  yang berada di tengah  $R_0$  dan  $Q_1$ , dan  $S_3$  yang berada di tengah  $Q_1$  dan  $P_2$ .

Hubungkan  $S_0$  dengan  $S_1$  dan  $S_2$  dengan  $S_3$  sehingga terbentuk garis baru.

Buatlah dua buah titik baru, yaitu  $T_0$  yang berada di tengah  $S_0$  dan  $S_1$ , serta  $T_1$  yang berada di tengah  $S_2$  dan  $S_3$ .

Buatlah sebuah garis yang menghubungkan  $P_0 - T_0 - R_0 - T_1 - P_2$ .

Melalui iterasi kedua akan tampak semakin mendekati sebuah kurva, dengan aproksimasi 5 buah titik. Anda dapat membuat visualisasi atau gambaran secara mandiri terkait hal ini sehingga dapat diamati dan diterka dengan jelas bahwa semakin banyak iterasi yang dilakukan, maka akan membentuk sebuah kurva yang tidak lain adalah kurva Bézier.



**Gambar 3.** Hasil pembentukan Kurva Bézier Kuadratik dengan *divide and conquer* setelah iterasi ke-2.

Paragraf di atas dikutip dari:

<https://docs.google.com/document/d/161qTQR5PzjQUIsoLO00A0Rp1dvsahrXY2Dk-fSmJl2o/edit>

Pada Tupil 2 IF 2211 ini, kami akan membuat kurva bezier kuadratik dengan mencari titik-titik solusi menggunakan Algoritma Divide and Conquer. Kami juga akan membuat kurva

bezier kuadratik dengan menggunakan Algoritma Brute Force untuk dijadikan perbandingan terhadap Algoritma Divide and Conquer. Hal yang dibandingkan adalah banyaknya operasi (something) dan waktu eksekusi yang dibutuhkan pada kedua algoritma. Kami juga menggeneralisasikan program sehingga dapat membentuk kurva bezier dengan N titik kontrol. Selain itu, Kami juga akan membuat visualisasi hasil kurva bezier dari titik-titik solusi yang didapatkan.

## Bab II

### Analisis dan Implementasi dalam Algoritma Brute Force

Langkah-langkah pembuatan kurva bezier kuadratik dengan menggunakan Algoritma Brute Force adalah sebagai berikut:

#### Langkah 1

Langkah pertama yang dilakukan adalah catat lokasi titik  $P_0$ , titik  $P_1$ , dan titik  $P_2$ . Titik kontrol  $P_0$  dan titik kontrol  $P_2$  akan selalu menjadi titik ujung kurva, sedangkan titik  $P_1$  akan menjadi titik kontrol antara yang pada umumnya tidak akan ada pada kurva.

#### Langkah 2

Langkah berikutnya adalah menentukan jumlah titik solusi yang akan dihasilkan sesuai dengan jumlah iterasi yang diinginkan. Jumlah titik solusi yang dihasilkan adalah  $2^N + 1$  titik dengan  $N$  adalah jumlah iterasi.

#### Langkah 3

$$t_0 = \frac{1}{\text{jumlah titik solusi} - 1}$$

Gunakan rumus di atas untuk mendapatkan  $t_0$  atau jarak yang ditempuh setiap iterasinya.

$$R_0 = B(t) = (1 - t)^2 P_0 + 2(1 - t) t P_1 + t^2 P_2, t \in [0, 1]$$

Setelah mendapatkan  $t_0$ , lakukan substitusi  $t$  pada rumus di atas dari nilai 0 hingga 1 dengan lompatan sebesar  $t_0$ . Karena titik  $P_0$  dan titik  $P_2$  selalu menjadi titik solusi, kita dapat mengurangi pemakaian rumus dengan mensubstitusi  $t$  pada rumus mulai dari  $t_0$  hingga  $1 - t_0$ .

Kompleksitas waktu pada pembuatan kurva bezier kuadratik dengan menggunakan Algoritma Brute Force dihitung dari banyaknya penggunaan rumus

$$R_0 = B(t) = (1 - t)^2 P_0 + 2(1 - t) t P_1 + t^2 P_2, t \in [0, 1]$$

dan operasi pertambahan, pengurangan, perkalian, dan perpangkatan pada rumus untuk sumbu x dan sumbu y adalah:

$$f(n) = 8(2^n - 1)$$

$$T(n) = 2f(n) = 16(2^n - 1)$$

$$O(n) = 2^n$$

$f(n)$  adalah kompleksitas waktu untuk salah satu sumbu

$T(n)$  adalah kompleksitas waktu secara keseluruhan

## Bab III

### Analisis dan Implementasi dalam Algoritma Divide and Conquer

Nama *Divide and Conquer* berasal dari 2 kata, yaitu *Divide* dan *Conquer*. *Divide* artinya membagi suatu persoalan yang besar menjadi upa-persoalan yang memiliki kemiripan dengan persoalan semula, namun berukuran lebih kecil. *Conquer* artinya menyelesaikan masing-masing upa-persoalan (diselesaikan secara langsung atau secara rekursif jika masih berukuran besar). Setelah itu, solusi dari masing-masing upa-persoalan akan digabung untuk mendapatkan solusi untuk persoalan semula. Algoritma *Divide and Conquer* dapat digunakan untuk membentuk kurva bezier kuadrat, kubik, kuartik, dan seterusnya hingga  $N$  titik kontrol.

Langkah-langkah untuk menyelesaikan persoalan membentuk kurva bezier kuadrat dengan menggunakan Algoritma *Divide and Conquer* adalah sebagai berikut:

1. **Tahap Solve:** Cari titik tengah dari titik kontrol pertama  $P_0$  dengan titik kontrol antara  $P_1$  dan beri nama  $Q_1$ . Cari titik tengah dari titik kontrol antara  $P_1$  dengan titik kontrol terakhir  $P_2$  dan beri nama  $Q_2$ . Kemudian cari titik tengah dari  $Q_1$  dan  $Q_2$  dan beri nama  $R_0$ . Catat lokasi titik  $P_0$ ,  $P_2$ ,  $Q_1$ ,  $Q_2$ , dan  $R_0$ .
2. Masukkan lokasi titik  $P_0$  ke dalam larik solusi hanya pada iterasi yang pertama.
3. **Tahap Divide:** Partisi titik-titik yang telah didapatkan menjadi 2 bagian, yaitu kanan dan kiri. Masing-masing mendapatkan 3 titik dengan titik  $R_0$  ada di kedua bagian. Bagian kiri adalah titik  $P_0$ , titik  $Q_1$ , dan titik  $R_0$ . Bagian kanan adalah titik  $R_0$ , titik  $Q_1$ , dan titik  $P_2$ .
4. Lakukan kembali dari **tahap Solve** untuk yang bagian kiri jika  $N > 1$  dan dilakukan sebanyak  $N-1$  kali dengan  $N$  adalah jumlah iterasi. Pada bagian kiri, titik  $P_0$  sebagai titik  $P_0$ , titik  $Q_1$  sebagai titik  $P_1$ , dan titik  $R_0$  sebagai titik  $P_2$ .
5. Masukkan lokasi titik  $R_0$  ke dalam larik solusi.
6. Lakukan kembali dari **tahap Solve** untuk yang bagian kanan jika  $N > 1$  dan dilakukan sebanyak  $N-1$  kali dengan  $N$  adalah jumlah iterasi. Pada bagian kanan, titik  $R_0$  sebagai titik  $P_0$ , titik  $Q_2$  sebagai titik  $P_1$ , dan titik  $P_2$  sebagai titik  $P_2$ .
7. Masukkan lokasi titik  $P_2$  ke dalam larik solusi hanya pada iterasi yang pertama.

Pada persoalan ini, masukan titik-titik awal sudah merupakan sebuah upa-persoalan sehingga dapat langsung melakukan tahap *Solve*. Pada tahap *Solve*, operasi ini akan menghasilkan titik solusi yang dibutuhkan untuk membentuk kurva bezier. Selain menghasilkan solusi, tahap *Solve* juga membuat persoalan menjadi lebih besar. Setiap kali melakukan tahap *Solve*, jumlah titik akan menjadi  $2^N + 1$  dengan  $N$  adalah jumlah iterasi. Oleh karena itu, pada iterasi selanjutnya titik-titik yang ada tidak dapat langsung dilakukan tahap *Solve* dan harus melalui tahap *Divide* terlebih dahulu. Setelah dilakukan partisi menjadi bagian kanan dan bagian



kiri, masing-masing bagian dapat melakukan tahap *Solve*. Tahap 4 dan 6 akan melakukan pemanggilan Algoritma *Divide and Conquer* secara rekursif sesuai dengan jumlah iterasi yang diinginkan dikurangi satu.

Kompleksitas waktu untuk membuat kurva bezier kuadratik dengan Algoritma *Divide and Conquer* dihitung dari banyaknya jumlah perhitungan titik tengah, operasi penambahan dan pengurangan yang ada pada operasi titik tengah, dan konkatenasi solusi akhir adalah

$$g(p) = 4\left(\frac{p}{2}(p-1)\right), p \geq 3$$

$$f(n) = 2 + (2^n - 1) = 2^n + 1$$

$$T(p, n) = \left(\sum_{i=1}^n 2^{i-1}\right)g(p) + f(n) = (2^n - 1)g(p) + f(n), p = p_0, n \geq 0$$

$$T(p, n) = (2^n - 1)\left(4\left(\frac{p}{2}(p-1)\right)\right) + (2^n + 1), p = p_0, n \geq 0$$

Sehingga Big O notation-nya adalah:

$$O(2^n p^2), n \geq 0$$

*n* adalah jumlah iterasi

*p* adalah jumlah titik masukan

*p*<sub>0</sub> adalah jumlah titik kontrol yang diinginkan

*g(p)* adalah banyaknya operasi perhitungan titik tengah

*f(n)* adalah banyaknya operasi konkatenasi untuk solusi akhir

*T(n)* adalah kompleksitas waktu secara keseluruhan

Pada pembentukan kurva bezier kuadratik, titik kontrol akan berjumlah 3 titik sehingga kompleksitas waktunya adalah

$$T(n) = 4 \times 3 \times (2^n - 1) + 2^n + 1, n \geq 0$$

$$T(n) = 13 \cdot 2^n - 11, n \geq 0$$

Sehingga Big O notation-nya adalah:

$$O(2^n), n \geq 0$$

# Bab IV

## Source Code dan Test Case

Source code dapat diakses melalui tautan Github berikut: [Repository](#)

### A. Source Code

Secara garis besar, program terbagi menjadi tiga file utama yang dipanggil saat program dijalankan. File tersebut adalah *function.py* yang berisi algoritma *Divide and Conquer* dan algoritma *Brute Force* untuk menghasilkan titik-titik pada kurva Bezier, fungsi untuk menampilkan dan membuat animasi grafik kurva Bezier, dan fungsi pembantu untuk menyusun titik-titik hasil fungsi.

#### 1. Function.py

```
1 def Bezier3Point(point1, point2, point3, iterate, iterateMax):
2     solution = []
3     # titikBantu = []
4     if iterate == iterateMax:
5         pos1 = DnC(point1, point2)
6         pos2 = DnC(point2, point3)
7
8         if iterateMax == 1:
9             solution.append(point1)
10            # titikBantu.append([point1])
11            # titikBantu.append([pos1, pos2])
12
13            solution.append(DnC(pos1, pos2))
14
15            if iterateMax == 1:
16                solution.append(point3)
17                # titikBantu.append([point3])
18
19            return solution
20
21    elif iterate < iterateMax:
22        pos1 = DnC(point1, point2)
23        pos2 = DnC(point2, point3)
24
25        if iterate == 1:
26            solution.append(point1)
27
28            # titikBantu.append([pos1, pos2])
29
30            temp = Bezier3Point(point1, pos1, DnC(pos1, pos2), iterate + 1, iterateMax)
31            solution += temp
32            # titikBantu += temp[1]
33
34            solution.append(DnC(pos1, pos2))
35
36            temp = Bezier3Point(DnC(pos1, pos2), pos2, point3, iterate + 1, iterateMax)
37            solution += temp
38            # titikBantu += temp[1]
39
40            if iterate == 1:
41                solution.append(point3)
42
43            return solution
```

Fungsi di atas adalah fungsi untuk menghasilkan titik-titik solusi penyelesaian kurva Bezier dengan 3 titik input. Terdapat pula fungsi Bezier3PointHelper yang memiliki logika yang sama, hanya saja mengembalikan titikBantu untuk memvisualisasikan proses per iterasi pembentukan kurva Bezier. Keduanya menggunakan algoritma Divide and Conquer

```
1 def BezierNPoint(arr, iterate, iterateMax):
2     solution = []
3     titikBantu = []
4     if iterate == iterateMax:
5         tempArr = copy.deepcopy(arr) # buat menampung titik solusi sementara
6         temp = []
7         while len(temp) != 1:
8             temp = []
9             for i in range(len(tempArr)-1):
10                 temp.append(DnC(tempArr[i], tempArr[i+1]))
11             tempArr = temp
12             titikBantutemp = copy.deepcopy(temp)
13             titikBantu.append(titikBantutemp)
14         titikBantu = titikBantu[:-1]
15
16         if iterateMax == 1:
17             solution += [arr[0]]
18             titikBantu.append(arr[0])
19
20         solution += tempArr
21
22         if iterateMax == 1:
23             solution += [arr[-1]]
24             titikBantu.append(arr[-1])
25
26         return solution, titikBantu
27     elif iterate < iterateMax:
28
29         # Titik Awal
30         if iterate == 1:
31             solution += [arr[0]]
32
33         # Tahap Awal
34         arr2 = arr
35         arrKiri = []
36         arrKanan = []
37         while len(arrKiri) != len(arr):
38             temp = []
39             for i in range(len(arr2)-1):
40                 temp.append(DnC(arr2[i], arr2[i+1]))
41             arrKiri.append(arr2[0])
42             arrKanan.append(arr2[-1])
43             arr2 = temp
44             titikBantutemp = copy.deepcopy(temp)
45             titikBantu.append(titikBantutemp)
46         titikBantu = titikBantu[:-2]
47
48         # Bagian Kiri
49         tempCall = BezierNPoint(arrKiri, iterate+1, iterateMax)
50         solution += tempCall[0]
51         titikBantu += tempCall[1]
52
53         # Bagian Tengah
54         solution += [BezierNPoint(arr, 1, 1)[0][1]]
55
56         # Bagian Kanan
57         arrKanan = list(reversed(arrKanan))
58
59         tempCall = BezierNPoint(arrKanan, iterate+1, iterateMax)
60         solution += tempCall[0]
61         titikBantu += tempCall[1]
62
63         # Titik Akhir
64         if iterate == 1:
65             solution += [arr[-1]]
66         return solution, titikBantu
```

Fungsi di atas adalah fungsi yang mengembalikan titik-titik solusi penyelesaian kurva Bezier dengan n-titik input menggunakan algoritma Divide and Conquer.

```
1 def BezierBruteforce(p1, p2, p3, iterate):
2     npoint = 2**iterate + 1
3     temp = 1/ (npoint-1)
4     n = temp
5     solution = []
6     solution += [p1]
7     count = 0
8     while n <= (1-temp):
9         rx = ((1-n)**2)*p1[0] + 2*(1-n)*n*p2[0] + (n**2)*p3[0]
10        ry = ((1-n)**2)*p1[1] + 2*(1-n)*n*p2[1] + (n**2)*p3[1]
11        solution += [(rx,ry)]
12        n += temp
13        count+=1
14    solution += [p3]
15    return solution
```

Fungsi di atas adalah fungsi untuk menggenerasi kurva Bezier dengan 3 titik masukan menggunakan algoritma *bruteforce*.

```
1 def DnC(point1, point2):
2     x = (point1[0] + point2[0])/2
3     y = (point1[1] + point2[1])/2
4     return (x,y)
```

```
1 def parseArrayNPoint(titikBantu):
2     temp = []
3     for array in titikBantu:
4         for subarray in array:
5             temp.append(subarray)
6
7     new = []
8     res = []
9     for point in temp:
10        if len(new) != 2:
11            new.append(point)
12        else:
13            res.append(new)
14    return res
```

Fungsi di atas berturut-turut adalah fungsi antara untuk menghasilkan titik tengah di antara dua titik (DnC) dan fungsi untuk merapikan larik titik bantu yang digunakan untuk memvisualisasikan pembentukan animasi kurva Bezier (parseArrayNPoint).

```

1 def animatePlot(arrayOfPoints, arrayOfSol, arrayOfHelper):
2     plt.close()
3     fig, ax = plt.subplots()
4     linePoints, = ax.plot([], [], 'ro-')
5     lineSol, = ax.plot([], [], 'bo-', markersize=2 if len(arrayOfSol) > 30 else 3)
6     lineHelper, = ax.plot([], [], linestyle='dotted', color='green')
7
8     ax.set_xlim(min([point[0] for point in arrayOfPoints]) - 1, max([point[0] for point in arrayOfPoints]) + 1)
9     ax.set_ylim(min([point[1] for point in arrayOfPoints]) - 1, max([point[1] for point in arrayOfPoints]) + 1)
10    currentPointsX = []
11    currentPointsY = []
12    currentSolX = []
13    currentSolY = []
14    currentHelperX = []
15    currentHelperY = []
16    currentIteration = 0
17    def animate(i):
18        if i < len(arrayOfSol):
19            currentSolX.append(arrayOfSol[i][0])
20            currentSolY.append(arrayOfSol[i][1])
21            lineSol.set_data(currentSolX, currentSolY)
22        if i < len(arrayOfPoints):
23            currentPointsX.append(arrayOfPoints[i][0])
24            currentPointsY.append(arrayOfPoints[i][1])
25            linePoints.set_data(currentPointsX, currentPointsY)
26        if i < len(arrayOfHelper):
27            currentHelperX.append([point[0] for point in arrayOfHelper[i]])
28            currentHelperY.append([point[1] for point in arrayOfHelper[i]])
29            lineHelper.set_data(currentHelperX, currentHelperY)
30
31    if len(arrayOfHelper) < 100:
32        interval = 200
33    elif 100 <= len(arrayOfHelper) < 800:
34        interval = 50
35    else:
36        interval = 2
37    ani = FuncAnimation(fig, animate, frames=max(len(arrayOfSol), len(arrayOfHelper), len(arrayOfPoints)), interval=interval, repeat=False)
38    plt.grid()
39    plt.show()

```

Fungsi di atas adalah fungsi untuk melakukan visualisasi animasi pembentukan kurva Bezier secara bertahap menggunakan *library* Matplotlib Animation.

## 2. main.py

File ini adalah file yang menjadi file utama program. Ketika file di-run, akan dipanggil fungsi khusus *main* pada file sebagai gerbang utama program. Saat file main di-run lewat terminal, pengguna akan diberi pilihan untuk menggunakan CLI atau GUI. Input menggunakan GUI lebih mudah digunakan namun jika saat program dijalankan nilai iterasi yang diberikan sangat besar (misalnya 20), maka pengguna disarankan menggunakan CLI karena *library customtkinter* bisa mengalami *crash* karena proses yang dilakukan cukup besar.

```

1  if __name__ == "__main__":
2      while True:
3          print("BEZIER CURVE GENERATOR")
4          print('1. Masuk lewat GUI\n2. Masuk lewat CLI (jika iterasinya besar)\n3. Keluar')
5          try:
6              choice = int(input("Masukkan pilihan: "))
7          except ValueError:
8              print("Input harus berupa integer!\n")
9              continue
10
11         if choice == 1:
12             App = Gui()
13             print("Untuk mengakhiri program, tutup semua window GUI yang berjalan")
14             App.mainloop()
15             break
16
17         elif choice == 2:
18             print('1. Tiga Titik\n2. N Titik')
19             while True:
20                 try:
21                     choice = int(input("Masukkan pilihan: "))
22                 except ValueError:
23                     print("Input harus berupa integer!")
24                     continue
25                 if choice == 1 or choice == 2:
26                     break
27                 else:
28                     print('Pilihan tidak tersedia! Silahkan masukkan ulang (1/2)')
29                     continue
30
31             if choice == 1:
32                 point1, point2, point3 = threePointInput()
33                 while True:
34                     try:
35                         iterasi = int(input("Masukkan iterasi: "))
36                         if iterasi < 1:
37                             print("Iterasi minimal 1 kali!")
38                             continue
39                     except:
40                         break
41                     except ValueError:
42                         print("Input harus berupa integer!")
43                         continue
44                     startBrute = time.time()
45                     sol2 = function.BezierBruteforce(point1, point2, point3, iterasi)
46                     endBrute = time.time()
47                     print("Waktu eksekusi algoritma brute force: ", (endBrute - startBrute) * 1000)
48                     startMid = time.time()
49                     sol = function.Bezier3Point(point1, point2, point3, 1, iterasi)
50                     endMid = time.time()
51                     titikBantu = function.Bezier3PointHelper(point1, point2, point3, 1, iterasi)
52                     print("Waktu eksekusi algoritma titik tengah: ", (endMid - startMid) * 1000)
53                     titikBantu = function.parseArrayNPoint(titikBantu)
54                     print("Silahkan tutup plot untuk melanjutkan")
55                     function.animatePlot([point1, point2, point3], sol, titikBantu)
56                     # function.showPlot([point1, point2, point3], sol, titikBantu)
57
58             elif choice == 2:
59                 arr = nPointInput()
60                 while True:
61                     try:
62                         iterasi = int(input("Masukkan iterasi: "))
63                         if iterasi < 1:
64                             print("Iterasi minimal 1 kali!")
65                             continue
66                     except:
67                         break
68                     except ValueError:
69                         print("Input harus berupa integer!")
70                         continue
71                     startMid = time.time()
72                     temp = function.BezierNPoint(arr, 1, iterasi)
73                     sol = temp[0]
74                     titikBantu = temp[1]
75                     endMid = time.time()
76                     new_array = function.parseArrayNPoint(titikBantu)
77                     print("Waktu eksekusi algoritma titik tengah: ", (endMid - startMid) * 1000)
78                     print("Silahkan tutup plot untuk melanjutkan")
79                     function.animatePlot(arr, sol, new_array)
80                     # function.showPlot(arr, sol, new_array)
81                 else:
82                     print("\nPilihan tidak valid")
83
84             elif choice == 3:
85                 break
86             else:
87                 print("Pilihan tidak tersedia!\n")
88                 continue
89         print('\n')

```

Jika pengguna menggunakan GUI, program akan memanggil kelas GUI dan melakukan instansiasi untuk membuat jendela GUI baru.

```

1 def threePointInput():
2     arr = []
3     for i in range(3):
4         while True:
5             try:
6                 temp = (tuple(map(float, input(f'Masukkan koordinat titik {i+1} (x y): ').split()))
7                 if len(temp) != 2:
8                     print("Koordinat harus terdiri dari 2 nilai!")
9                 else:
10                    arr.append(temp)
11                    break
12            except ValueError:
13                print("Input harus berupa integer!")
14    return arr[0], arr[1], arr[2]

```

Fungsi di atas adalah fungsi yang dipanggil jika pengguna menggunakan CLI dan ingin melakukan pembentukan kurva Bezier menggunakan 3 titik. Fungsi akan mengembalikan titik-titik yang dimasukkan melalui terminal.

```

1 def nPointInput():
2     while True:
3         try:
4             n = int(input("Masukkan jumlah titik: "))
5             if n <= 2:
6                 print("Minimal 2 titik!")
7                 continue
8             else:
9                 break
10        except ValueError:
11            print("Input harus berupa integer!")
12    arr = []
13    for i in range(n):
14        while True:
15            try:
16                temp = (tuple(map(float, input(f'Masukkan koordinat titik {i+1} (x y): ').split()))
17                if len(temp) != 2:
18                    print("Koordinat harus terdiri dari 2 nilai!")
19                else:
20                    arr.append(temp)
21                    break
22            except ValueError:
23                print("Input harus berupa integer!")
24    return arr

```

Fungsi di atas adalah fungsi yang dipanggil jika pengguna menggunakan CLI dan ingin melakukan pembentukan kurva Bezier menggunakan  $n$  buah titik. Fungsi akan mengembalikan titik-titik yang dimasukkan melalui terminal.

### 3. GUI.py

File GUI berisi kelas Gui yang merupakan *child class* dari *library customtkinter*, sehingga pengguna perlu melakukan instalasi *customtkinter* dan juga *tkinter*.

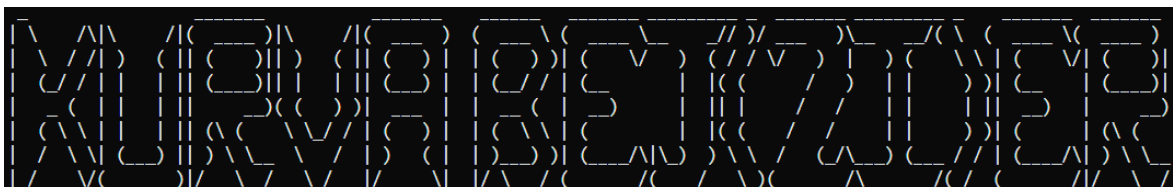
```

1 class Gui(ctl.CTk):
2     def __init__(self):
3         # initialization
4         super().__init__()
5         ctl.set_appearance_mode("light")
6         ctl.set_default_color_theme("blue")
7         self.title("Bezier Curve Generator with Divide and Conquer Algorithm")
8         self.columnconfigure(0, weight=1)
9         self.rowconfigure(0, weight=1)
10
11        # attribute
12        self.XPointInput = []
13        self.YPointInput = []
14        self.solutionResult = []
15        self.arrayOfInput = []
16        self.titikBantu = []
17        self.mainPage = ctl.CTkFrame(self)
18        self.pageThree = ctl.CTkFrame(self)
19        self.pageN = ctl.CTkFrame(self)
20        self.pagePlot = ctl.CTkFrame(self)
21        self.error = ""
22
23        # create frame
24        self.create_main_page()
25        self.create_page_three()
26        self.create_page_n()
27
28        # show page
29        self.show_page(self.mainPage)

```

Gambar di atas adalah atribut-atribut dan inisialisasi dari kelas Gui. Atribut title, columnconfigure, rowcondigure, dan window adalah atribut milik *customtkinter* yang nilainya diubah, dan atribut lainnya seperti *frames* adalah atribut yang dibuat untuk menjadi *layout* tampilan GUI. *Frame* halaman utama (mainPage) dibuat pada fungsi *create\_main\_page()*, dan fungsi untuk mengganti *frame* yang ditampilkan adalah fungsi *show\_page()*.

## Tampilan Awal Running



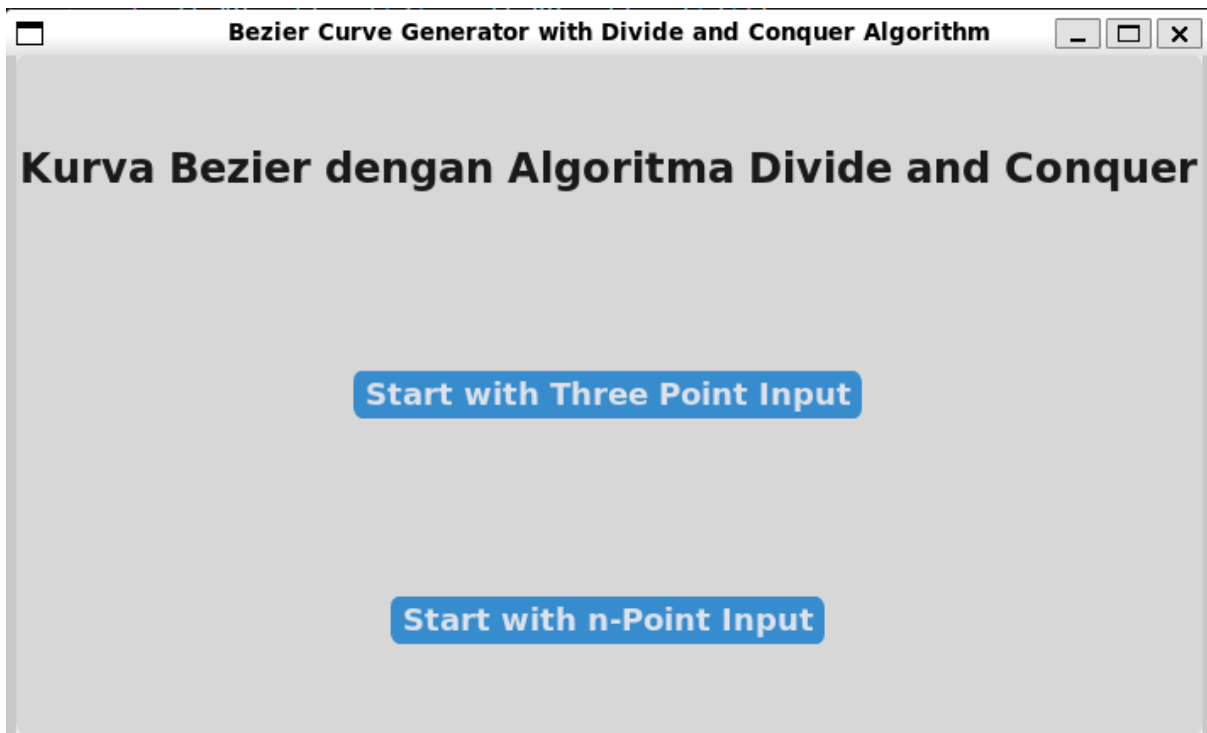
```

1. Masuk lewat GUI
2. Masuk lewat CLI (jika iterasinya besar (iterasi > 10 dianjurkan))
3. Keluar
Masukkan pilihan: █

```

## Tampilan Awal GUI





Tampilan CLI

```
Masukkan pilihan: 2
1. Tiga Titik
2. N Titik
Masukkan pilihan: |
```

## B. Test Case 3 Titik

### 1. Test Case 1 (3 Titik)

Input:

A screenshot of the application window showing the input form for 3 points. The title bar is "Bezier Curve Generator with Divide and Conquer Algorithm". The main title is "Kurva Bezier dengan 3 Titik". There is a "Main Menu" button. The input fields are: "Titik 1:" with value "1", "Titik 2:" with value "3", "Titik 3:" with value "5", and "Iterasi (positif):" with value "3". At the bottom, the execution times are displayed: "Waktu eksekusi (DnC algorithm): 0.027442998543847352 ms" and "Waktu eksekusi (Bruteforce algorithm): 0.03033800021512434 ms". Below that, it says "Untuk membuat grafik baru, tutup grafik yang masih terbuka".

Output:



## 2. Test Case 2 (3 Titik)

Input:

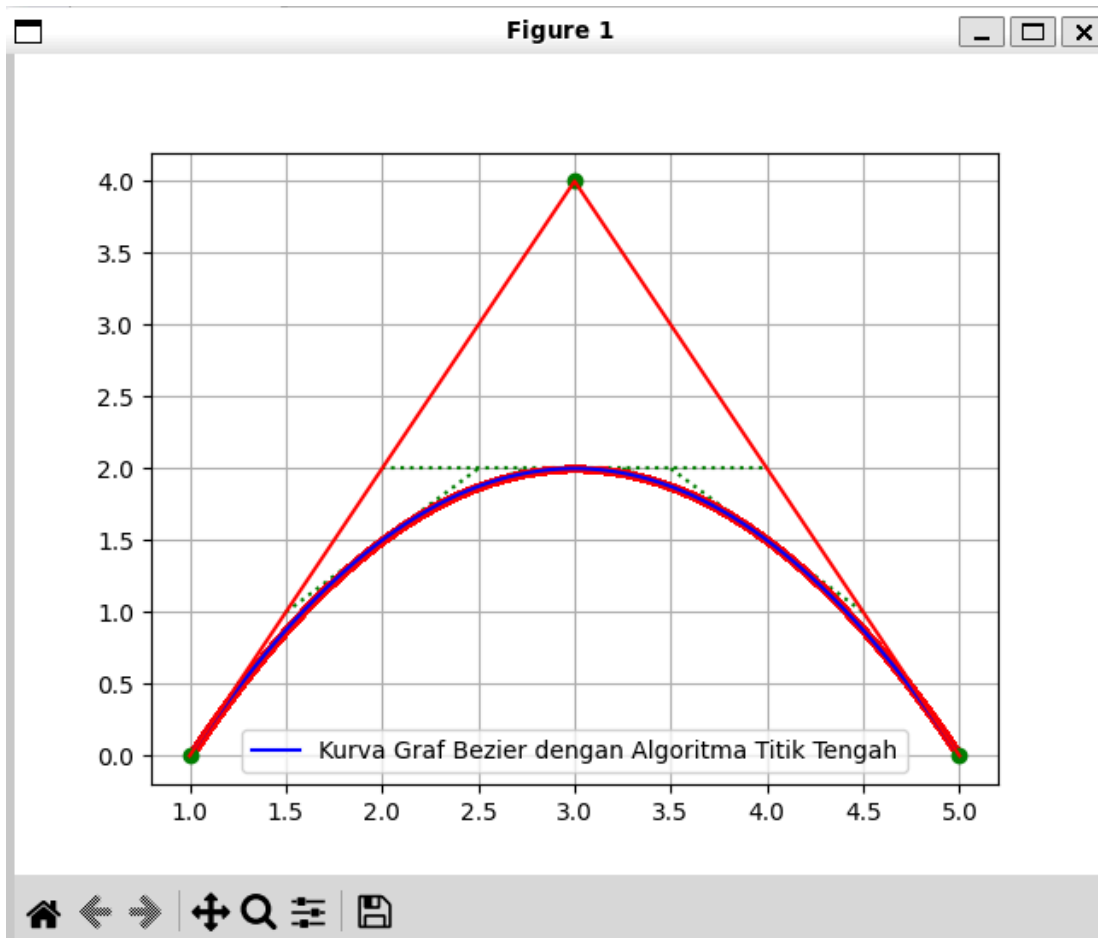
Bezier Curve Generator with Divide and Conquer Algorithm

**Kurva Bezier dengan 3 Titik** [Main Menu](#)

Titik 1:	<input type="text" value="1"/>	<input type="text" value="0"/>
Titik 2:	<input type="text" value="3"/>	<input type="text" value="4"/>
Titik 3:	<input type="text" value="5"/>	<input type="text" value="0"/>
Iterasi (positif):		<input type="text" value="15"/>

Waktu eksekusi (DnC algorithm): 27.27255799982231 ms  
Waktu eksekusi (Bruteforce algorithm): 33.431877000111854 ms  
Untuk membuat grafik baru, tutup grafik yang masih terbuka

Output:



### 3. Test Case 3 (3 Titik)

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

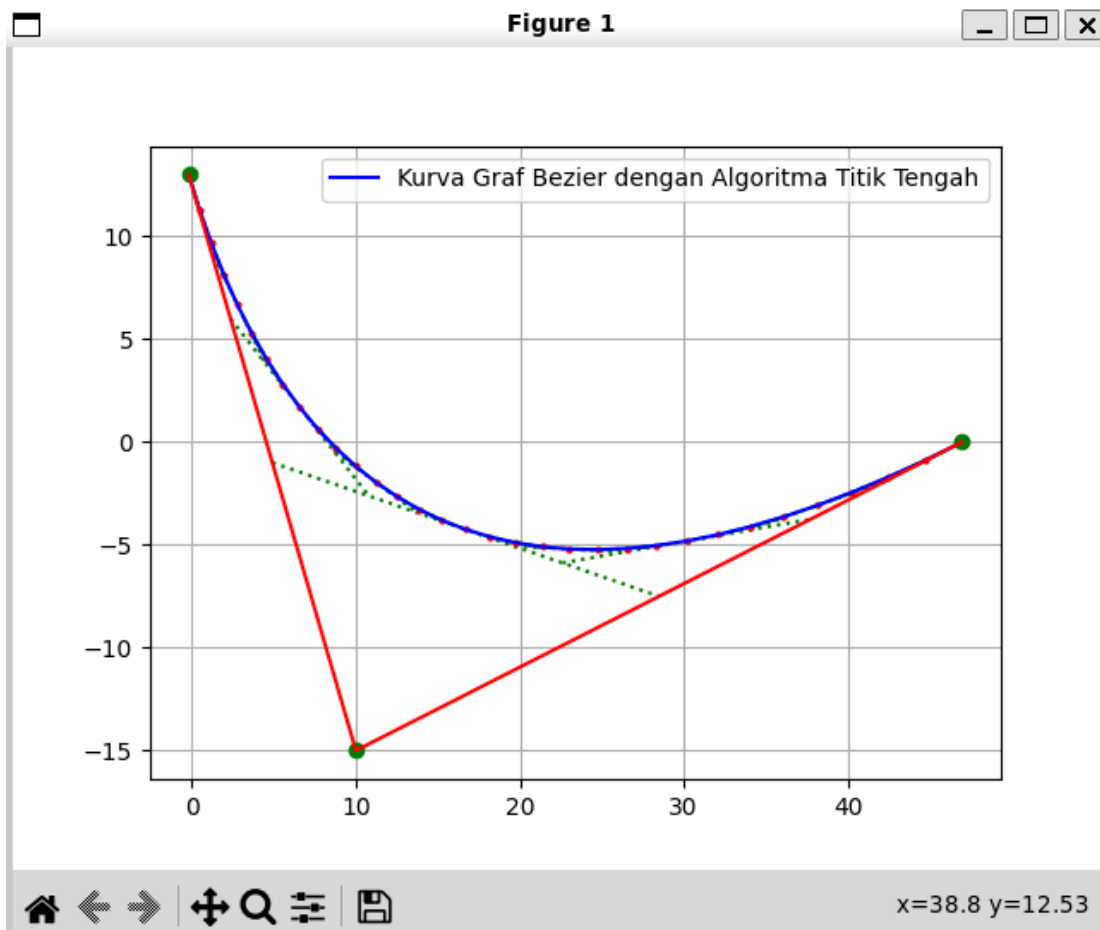
### Kurva Bezier dengan 3 Titik

Main Menu

Titik 1:	<input type="text" value="-0.2"/>	<input type="text" value="13"/>
Titik 2:	<input type="text" value="10"/>	<input type="text" value="-15"/>
Titik 3:	<input type="text" value="47"/>	<input type="text" value="0"/>
Iterasi (positif):		<input type="text" value="5"/>

Waktu eksekusi (DnC algorithm): 0.037601999792968854 ms  
Waktu eksekusi (Bruteforce algorithm): 0.05993400191073306 ms  
Untuk membuat grafik baru, tutup grafik yang masih terbuka

Output:



#### 4. Test Case 4

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

### Kurva Bezier dengan 3 Titik

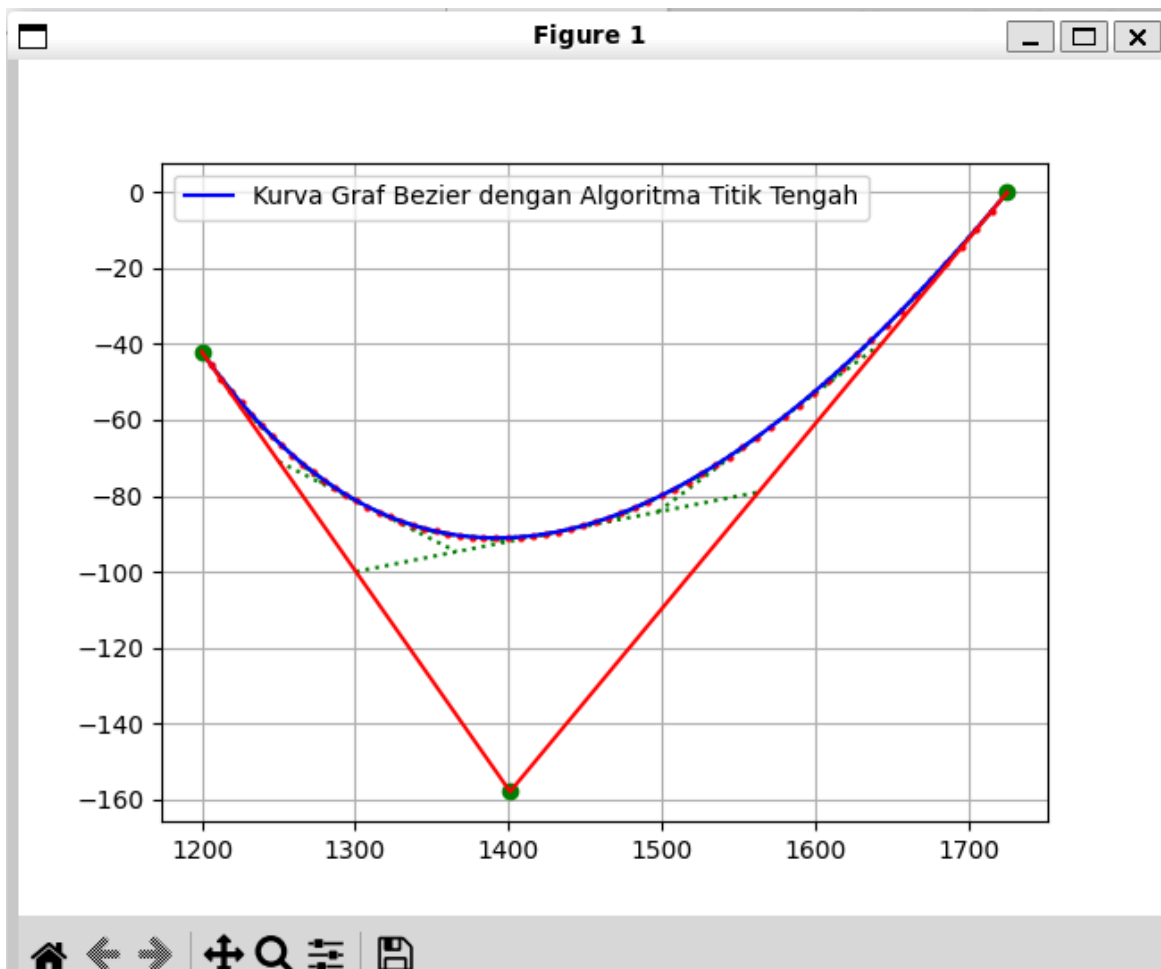
Main Menu

Titik 1:	<input type="text" value="1200"/>	<input type="text" value="-42"/>
Titik 2:	<input type="text" value="1401.5"/>	<input type="text" value="-158"/>
Titik 3:	<input type="text" value="1725"/>	<input type="text" value="0"/>

Iterasi (positif):

Waktu eksekusi (DnC algorithm): 0.07503300003008917 ms  
Waktu eksekusi (Bruteforce algorithm): 0.09024199971463531 ms  
Untuk membuat grafik baru, tutup grafik yang masih terbuka

Output:



## 5. Test Case 5

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

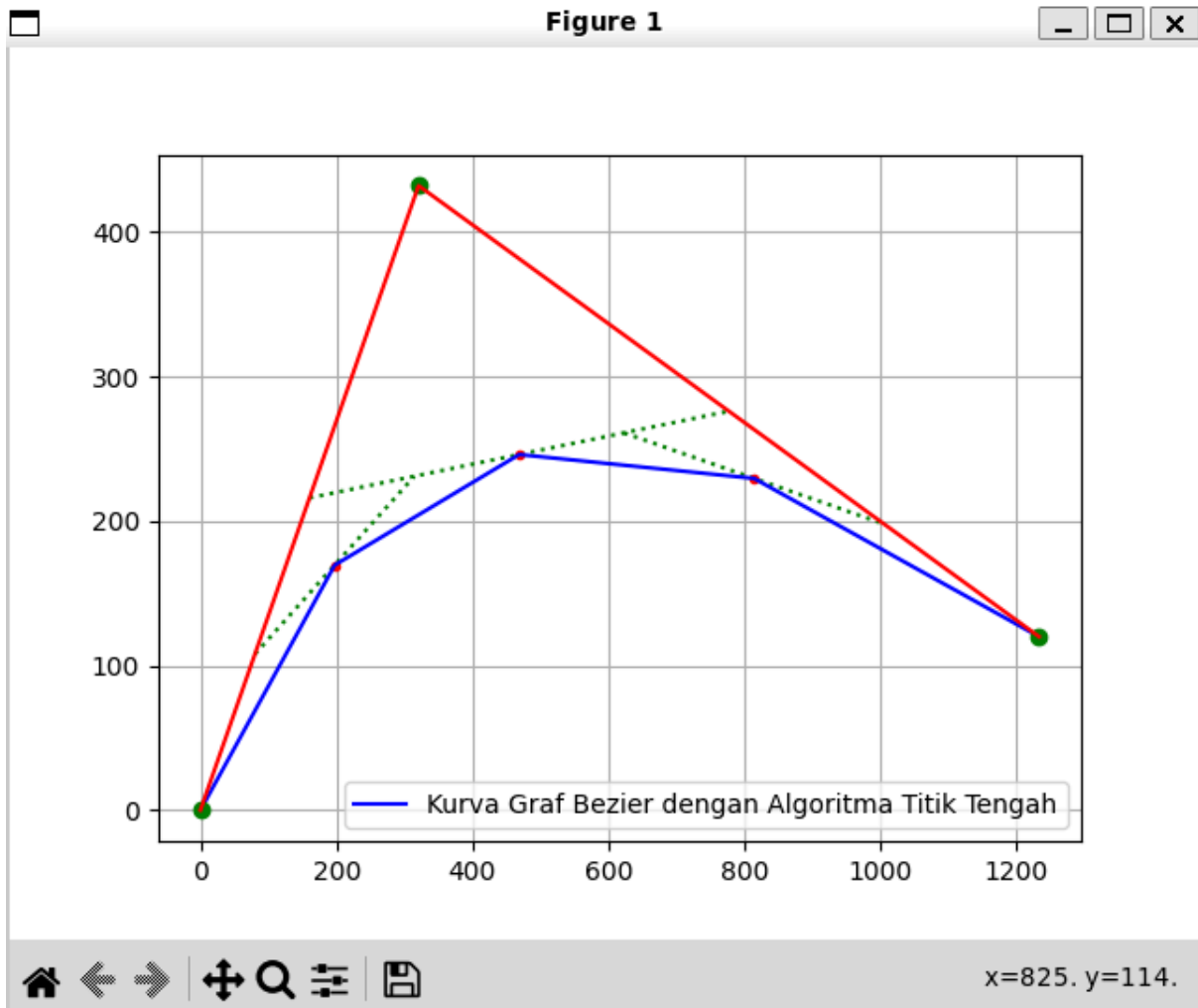
### Kurva Bezier dengan 3 Titik

Main Menu

Titik 1:	<input type="text" value="0"/>	<input type="text" value="0.000124"/>
Titik 2:	<input type="text" value="321"/>	<input type="text" value="432"/>
Titik 3:	<input type="text" value="1234"/>	<input type="text" value="120"/>
Iterasi (positif):		<input type="text" value="2"/>

Waktu eksekusi (DnC algorithm): 0.00862700107973069 ms  
Waktu eksekusi (Bruteforce algorithm): 0.028473998099798337 ms  
Untuk membuat grafik baru, tutup grafik yang masih terbuka

Output:



## 6. Test Case 6

Input:

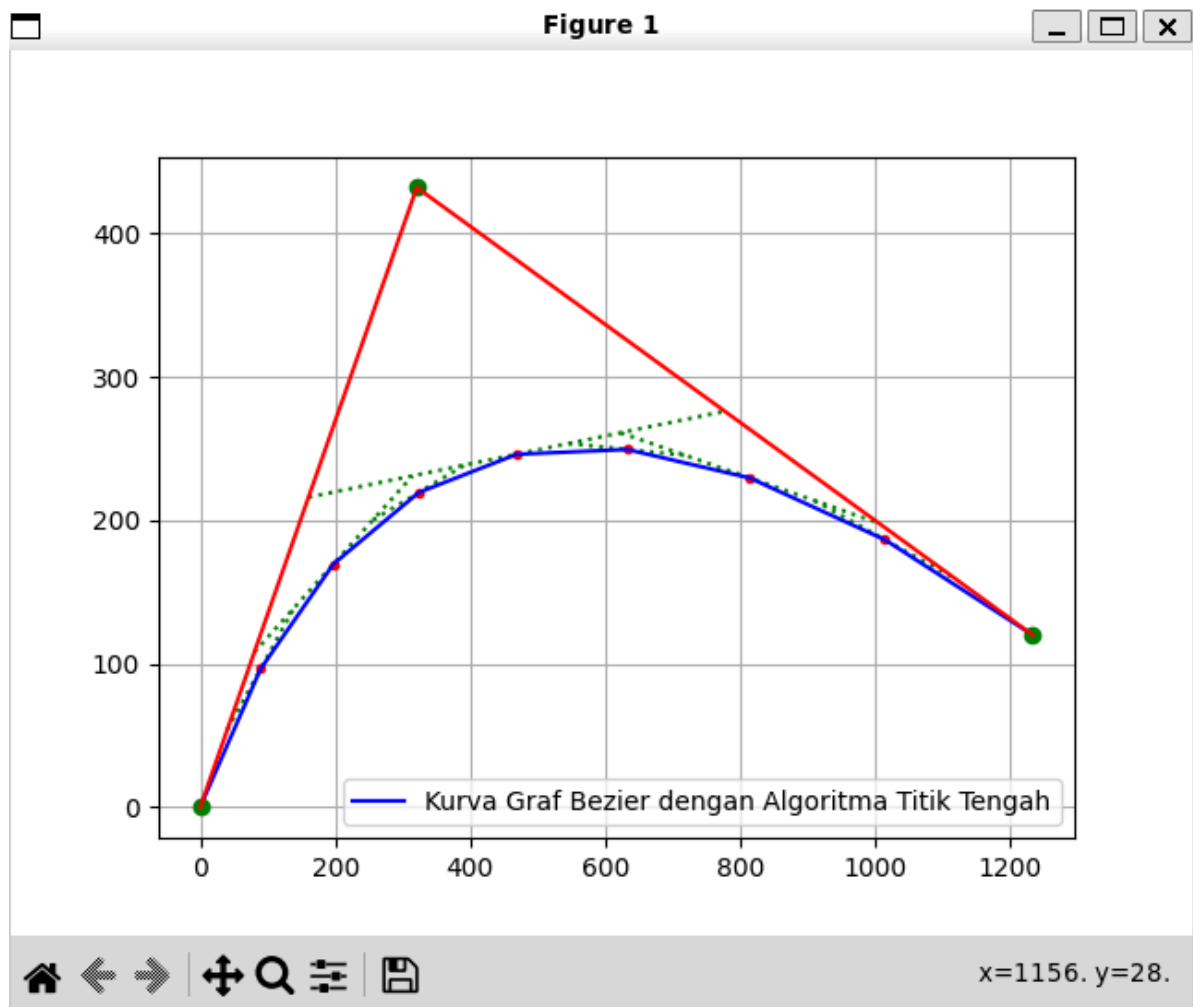
**Bezier Curve Generator with Divide and Conquer Algorithm**

**Kurva Bezier dengan 3 Titik** [Main Menu](#)

Titik 1:	<input type="text" value="0"/>	<input type="text" value="0.000124"/>
Titik 2:	<input type="text" value="321"/>	<input type="text" value="432"/>
Titik 3:	<input type="text" value="1234"/>	<input type="text" value="120"/>
Iterasi (positif):		<input type="text" value="3"/>

Waktu eksekusi (DnC algorithm): 0.014708002709085122 ms  
Waktu eksekusi (Bruteforce algorithm): 0.03120900146313943 ms  
Untuk membuat grafik baru, tutup grafik yang masih terbuka

Output:



## 7. Test Case 7

Input:

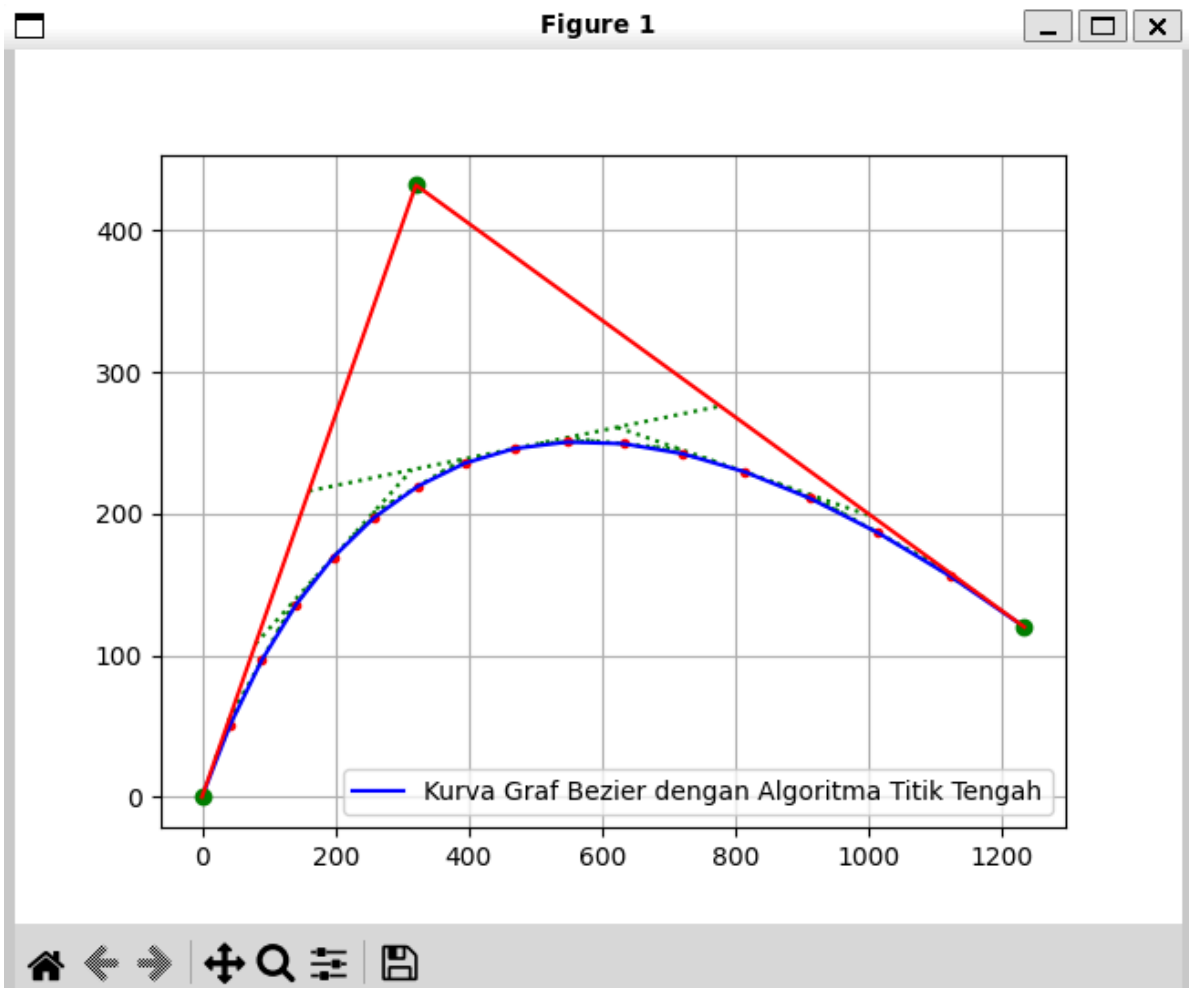
Bezier Curve Generator with Divide and Conquer Algorithm

**Kurva Bezier dengan 3 Titik** [Main Menu](#)

Titik 1:	<input type="text" value="0"/>	<input type="text" value="0.000124"/>
Titik 2:	<input type="text" value="321"/>	<input type="text" value="432"/>
Titik 3:	<input type="text" value="1234"/>	<input type="text" value="120"/>
Iterasi (positif):	<input type="text" value="4"/>	

Waktu eksekusi (DnC algorithm): 0.024726999981794506 ms  
 Waktu eksekusi (Bruteforce algorithm): 0.04370299939182587 ms  
 Untuk membuat grafik baru, tutup grafik yang masih terbuka

Output:



## 8. Test Case 8

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

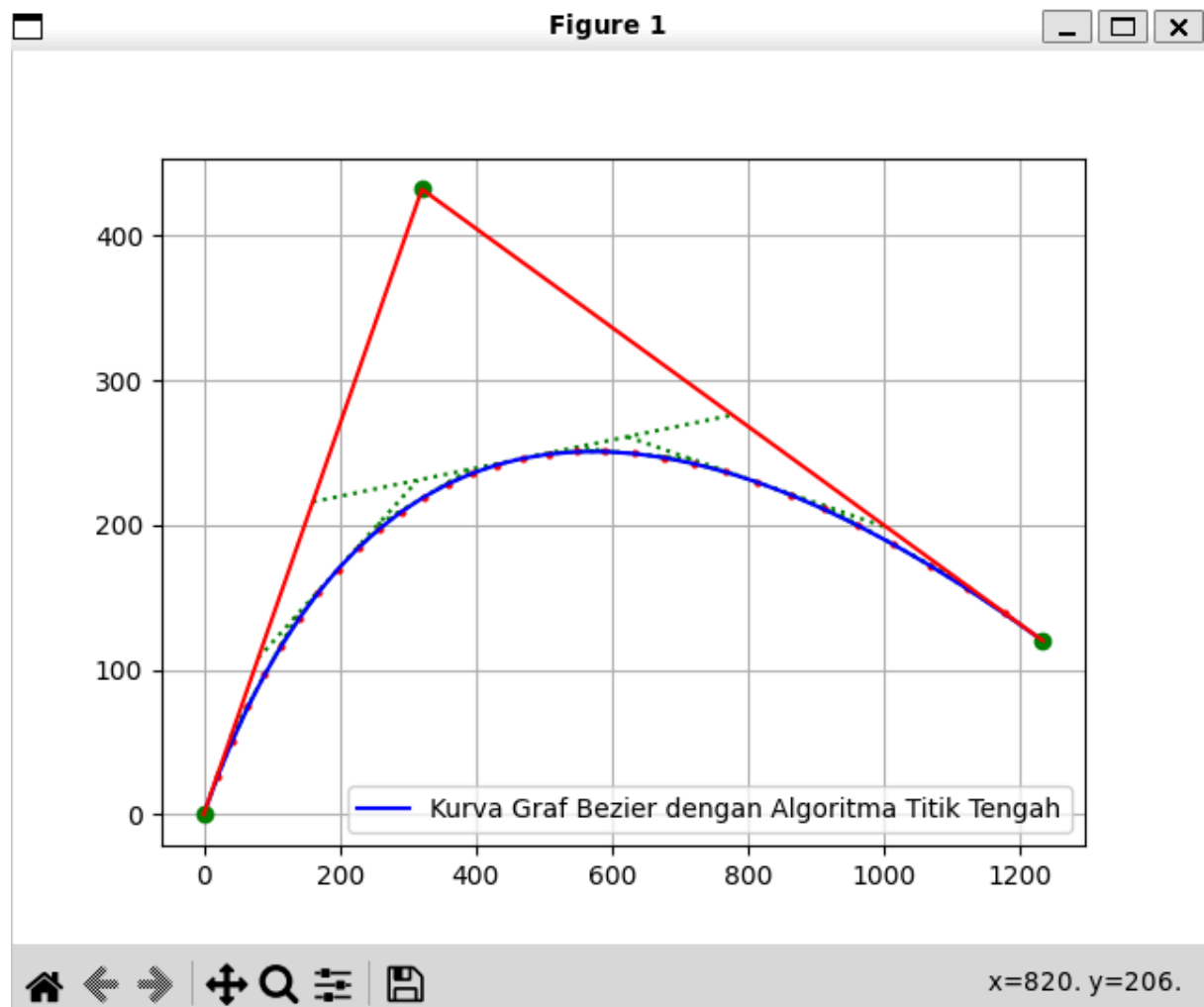
**Kurva Bezier dengan 3 Titik** [Main Menu](#)

Titik 1:	<input type="text" value="0"/>	<input type="text" value="0.000124"/>
Titik 2:	<input type="text" value="321"/>	<input type="text" value="432"/>
Titik 3:	<input type="text" value="1234"/>	<input type="text" value="120"/>
Iterasi (positif):		<input type="text" value="5"/>

Waktu eksekusi (DnC algorithm): 0.03671000013127923 ms  
 Waktu eksekusi (Bruteforce algorithm): 0.07612499757669866 ms  
 Untuk membuat grafik baru, tutup grafik yang masih terbuka



Output:



## 9. Test Case 9

Input:

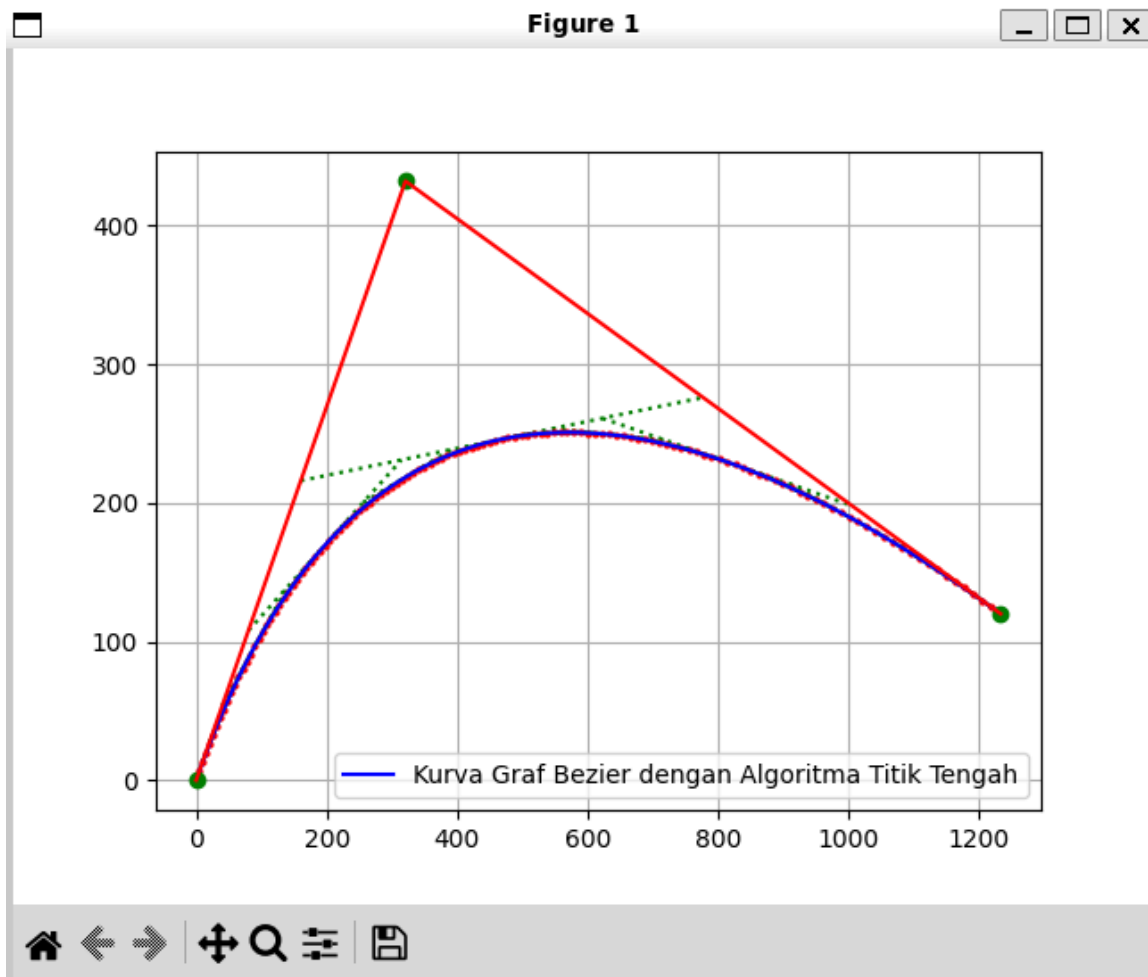
Bezier Curve Generator with Divide and Conquer Algorithm

**Kurva Bezier dengan 3 Titik** [Main Menu](#)

Titik 1:	<input type="text" value="0"/>	<input type="text" value="0.000124"/>
Titik 2:	<input type="text" value="321"/>	<input type="text" value="432"/>
Titik 3:	<input type="text" value="1234"/>	<input type="text" value="120"/>
Iterasi (positif):	<input type="text" value="7"/>	

Waktu eksekusi (DnC algorithm): 0.1180340004793834 ms  
Waktu eksekusi (Bruteforce algorithm): 0.1453869990655221 ms  
Untuk membuat grafik baru, tutup grafik yang masih terbuka

Output:



## 10. Test Case 10

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

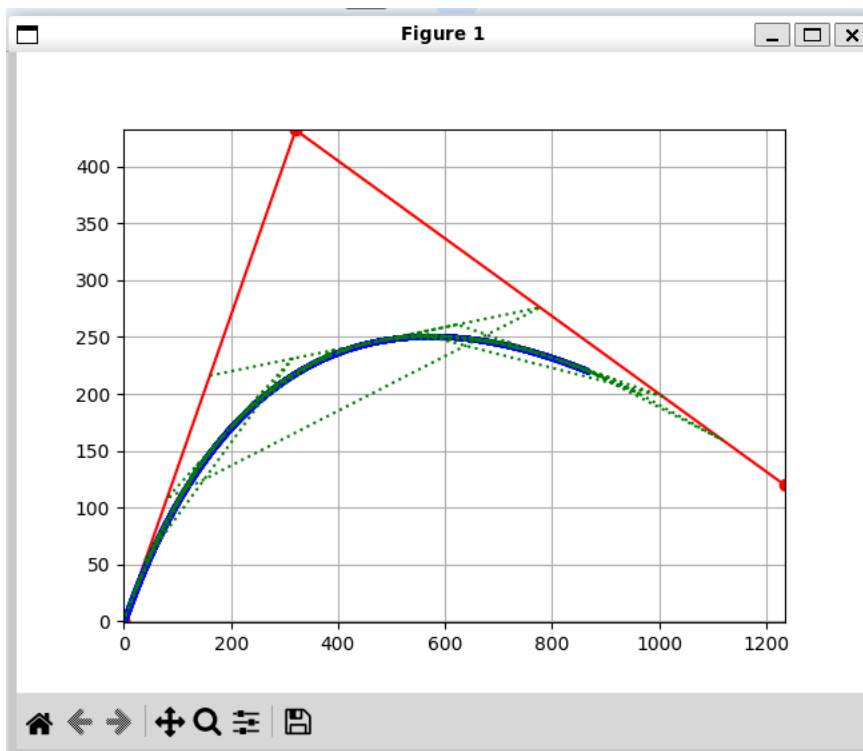
### Kurva Bezier dengan 3 Titik

Main Menu

Titik 1:	<input type="text" value="0"/>	<input type="text" value="0.000124"/>
Titik 2:	<input type="text" value="321"/>	<input type="text" value="432"/>
Titik 3:	<input type="text" value="1234"/>	<input type="text" value="120"/>
Iterasi (positif):		<input type="text" value="12"/>

Waktu eksekusi (DnC algorithm): 4.616324997186894 ms  
Waktu eksekusi (Bruteforce algorithm): 2.2449069983849768 ms  
Untuk membuat grafik baru, tutup grafik yang masih terbuka

Output:



## C. Test Case n-Titik

### 1. Test Case 1

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

Kurva Bezier dengan n Titik

Main Menu

Masukan jumlah titik:

12;12;6;4;5

-10;10;7;8;-5

Iterasi (positif):

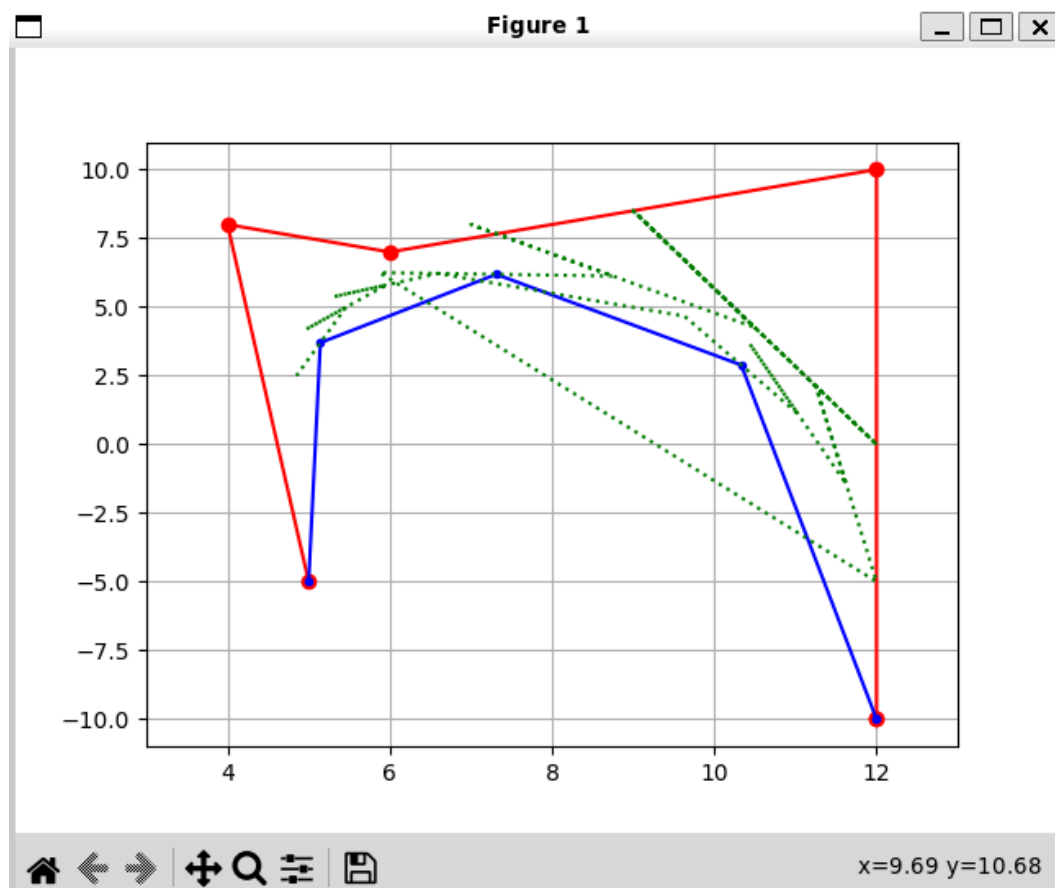
2

Pisahkan setiap titik dengan titik koma (;)  
Banyak titik X harus sama dengan titik Y

Submit

Waktu eksekusi: 0.13113021850585938 ms

Output:



## 2. Test Case 2

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

Kurva Bezier dengan n Titik

Main Menu

Masukan jumlah titik:

12;12;6;4;5

-10;10;7;8;-5

Iterasi (positif):

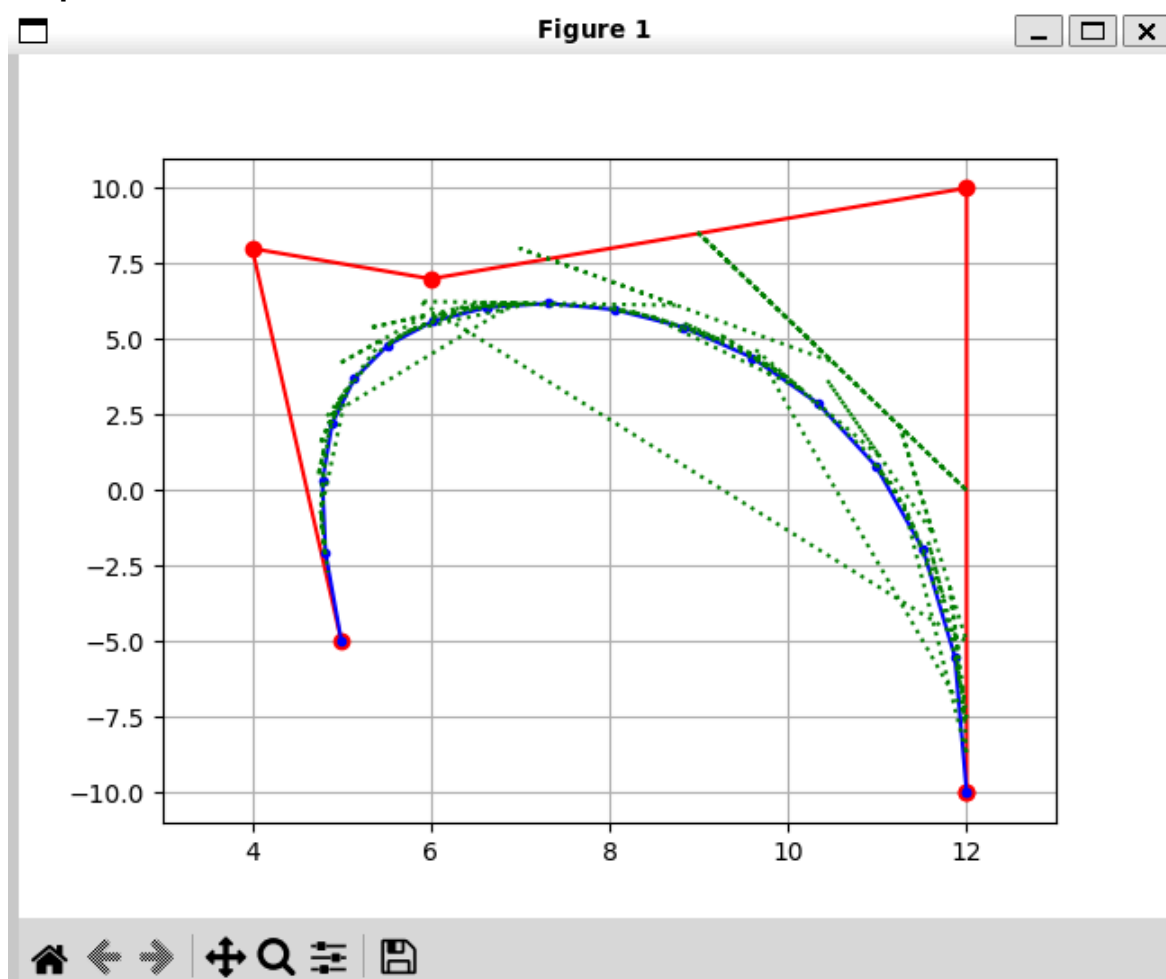
4

Pisahkan setiap titik dengan titik koma (;)  
Banyak titik X harus sama dengan titik Y

Submit

Waktu eksekusi: 0.5970001220703125 ms

Output:



### 3. Test Case 3

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

## Kurva Bezier dengan n Titik

Main Menu

Masukan jumlah titik:

12;12;6;4;5

-10;10;7;8;-5

Iterasi (positif):

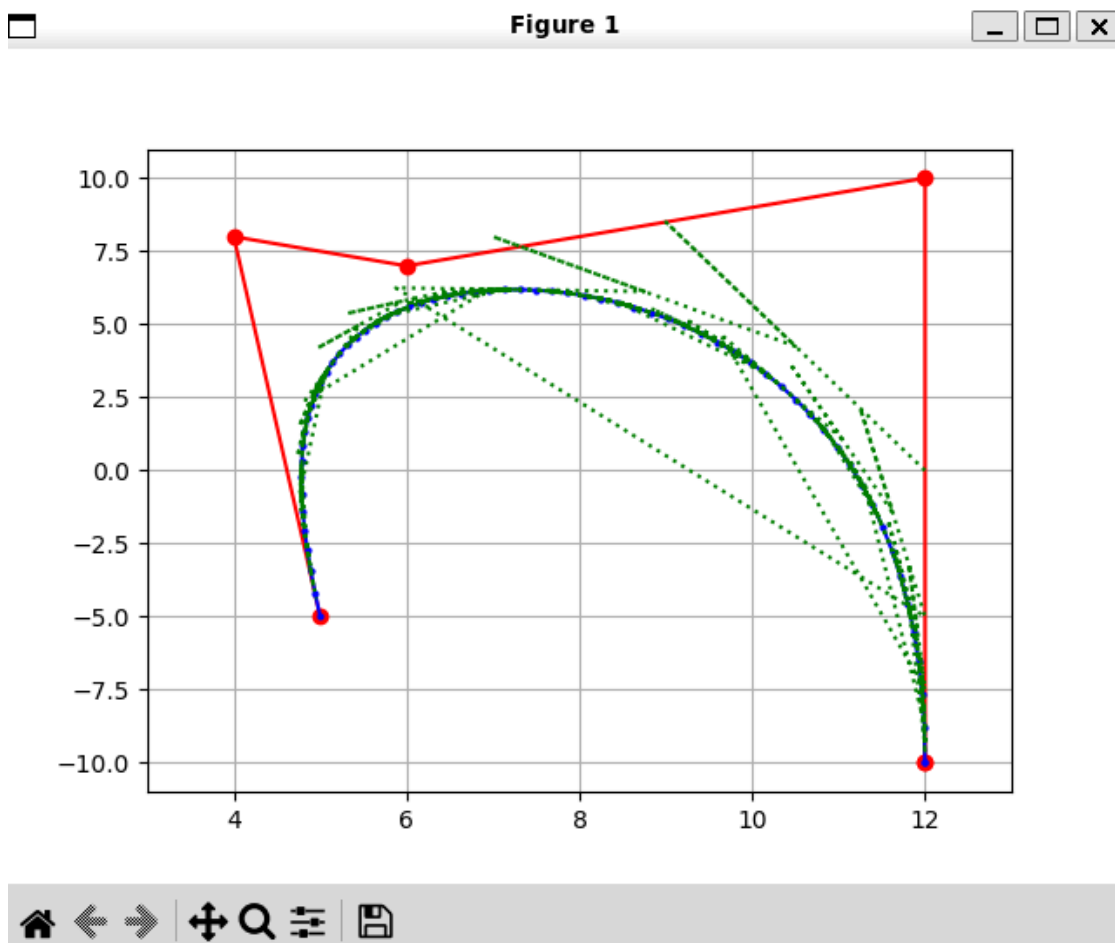
6

Pisahkan setiap titik dengan titik koma (;)  
Banyak titik X harus sama dengan titik Y

Submit

Waktu eksekusi: 2.4144649505615234 ms

Output:



#### 4. Test Case 4

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

Kurva Bezier dengan n Titik

Main Menu

Masukan jumlah titik:

-2;10;15;27;14;5

-1;4;-3;10;20;28

Iterasi (positif):

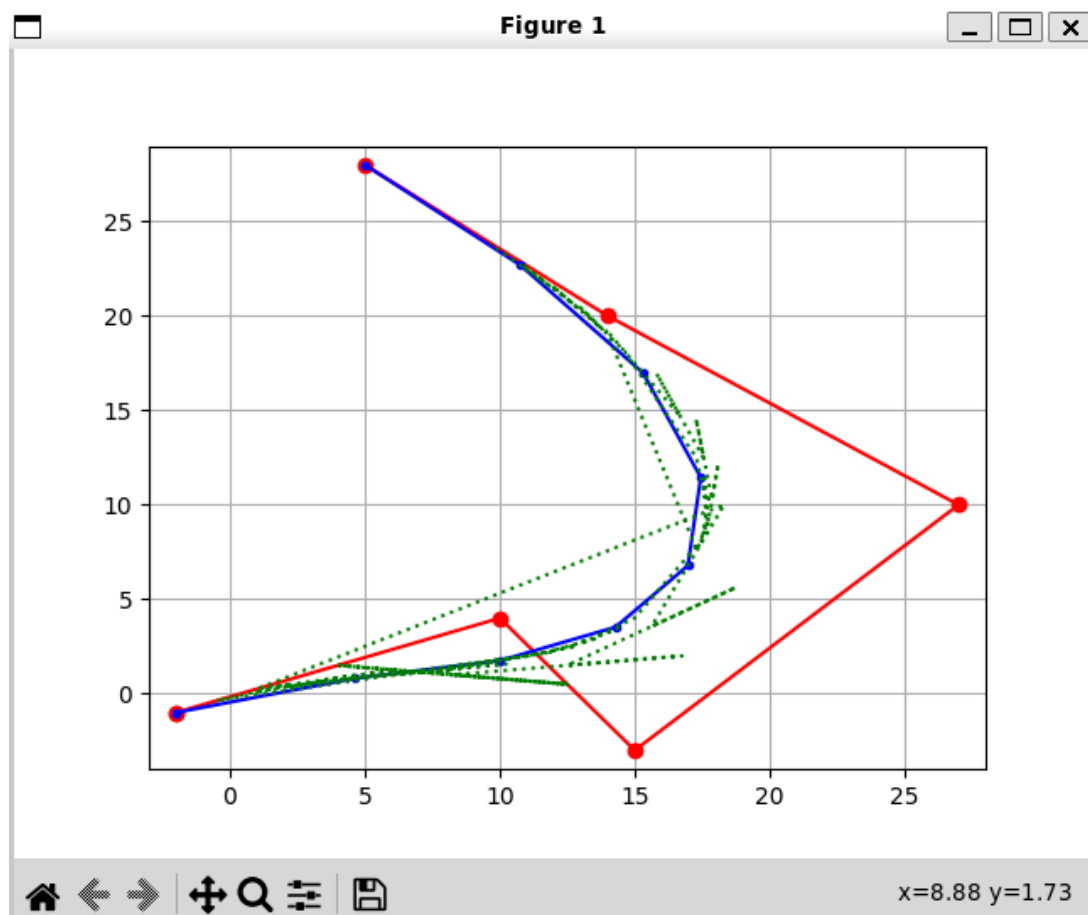
3

Pisahkan setiap titik dengan titik koma (;)  
Banyak titik X harus sama dengan titik Y

Submit

Waktu eksekusi: 0.37932395935058594 ms

Output:



## 5. Test Case 5

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

### Kurva Bezier dengan n Titik

Main Menu

Masukan jumlah titik:

-2;10;15;27;14;5

-1;4;-3;10;20;28

Iterasi (positif):

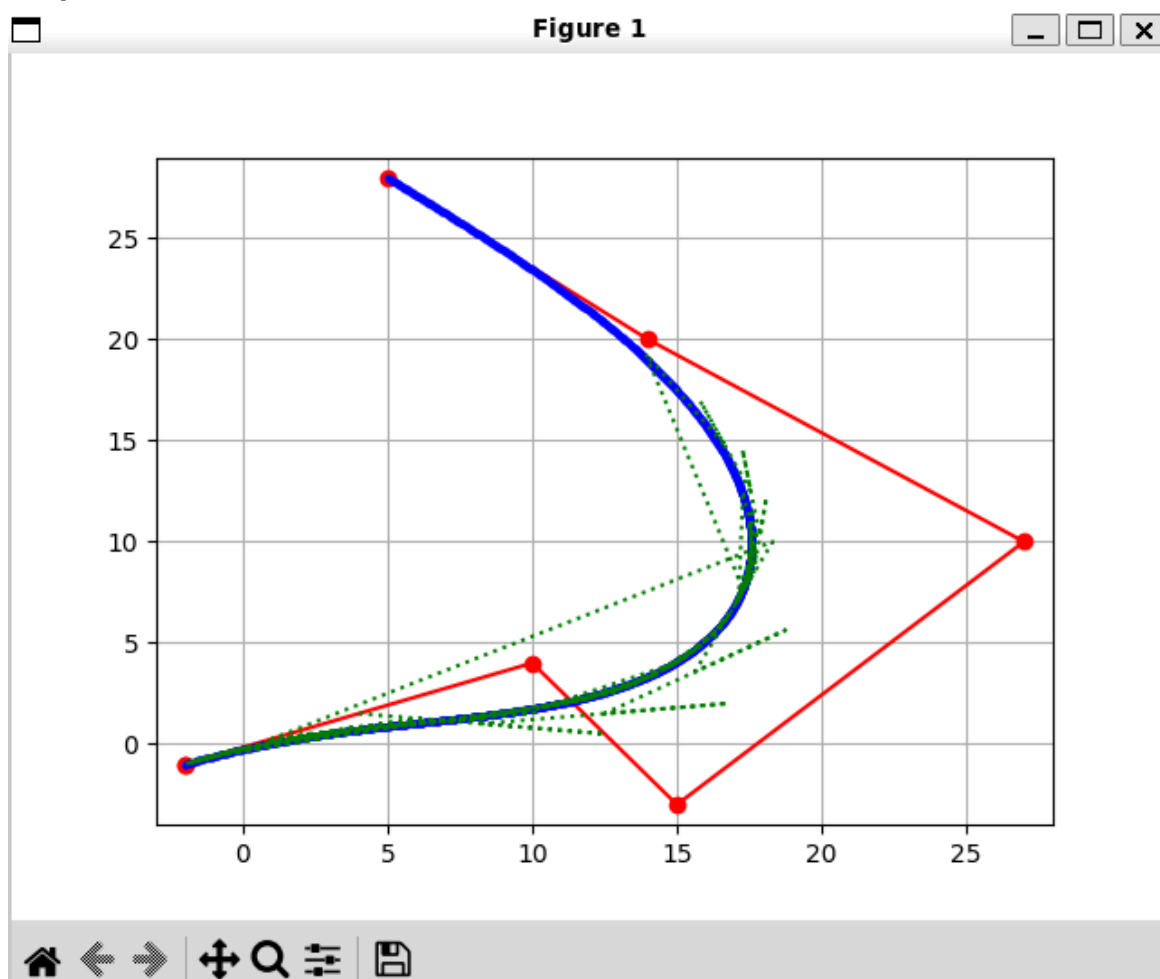
10

Pisahkan setiap titik dengan titik koma (;)  
Banyak titik X harus sama dengan titik Y

Submit

Waktu eksekusi: 51.34868621826172 ms

Output:





## 6. Test Case 6

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

### Kurva Bezier dengan n Titik

Main Menu

Masukan jumlah titik:

1; 3; 5

0; 4; 0

Iterasi (positif):

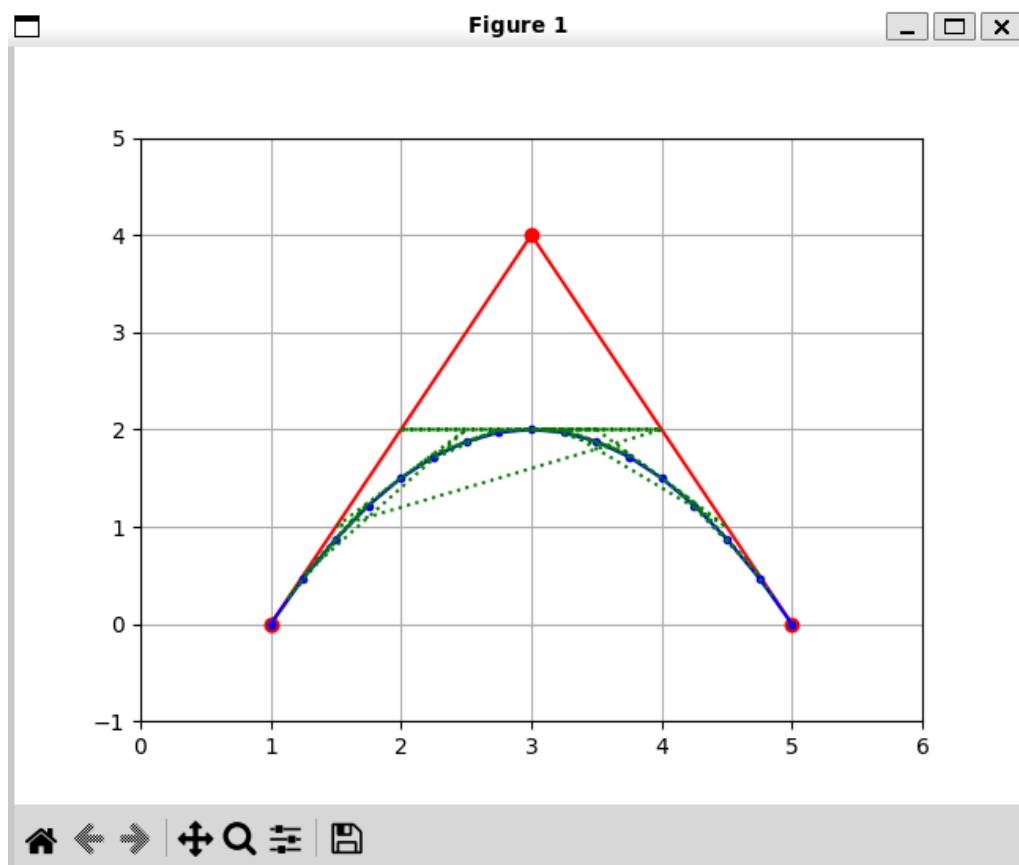
4

Pisahkan setiap titik dengan titik koma (;)  
Banyak titik X harus sama dengan titik Y

Submit

Waktu eksekusi: 0.26869773864746094 ms

Output:



## 7. Test Case 7

Input:

Bezier Curve Generator with Divide and Conquer Algorithm

Kurva Bezier dengan n Titik

Main Menu

Masukan jumlah titik:

0.2; -10.5; 3; 14

-7; 1.89; 5; -2.4

Iterasi (positif):

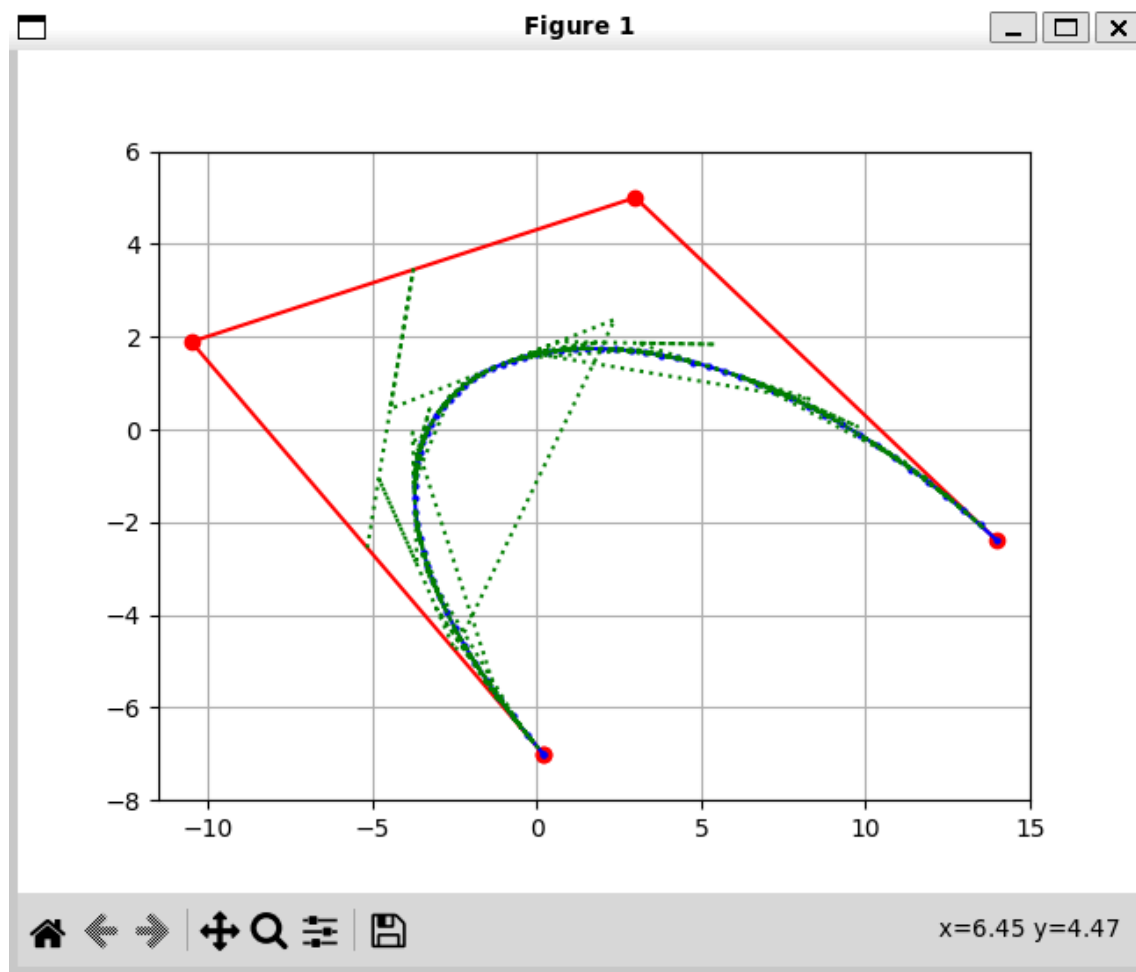
6

Pisahkan setiap titik dengan titik koma (;)  
Banyak titik X harus sama dengan titik Y

Submit

Waktu eksekusi: 1.6088485717773438 ms

Output:



## D. Test Case CLI

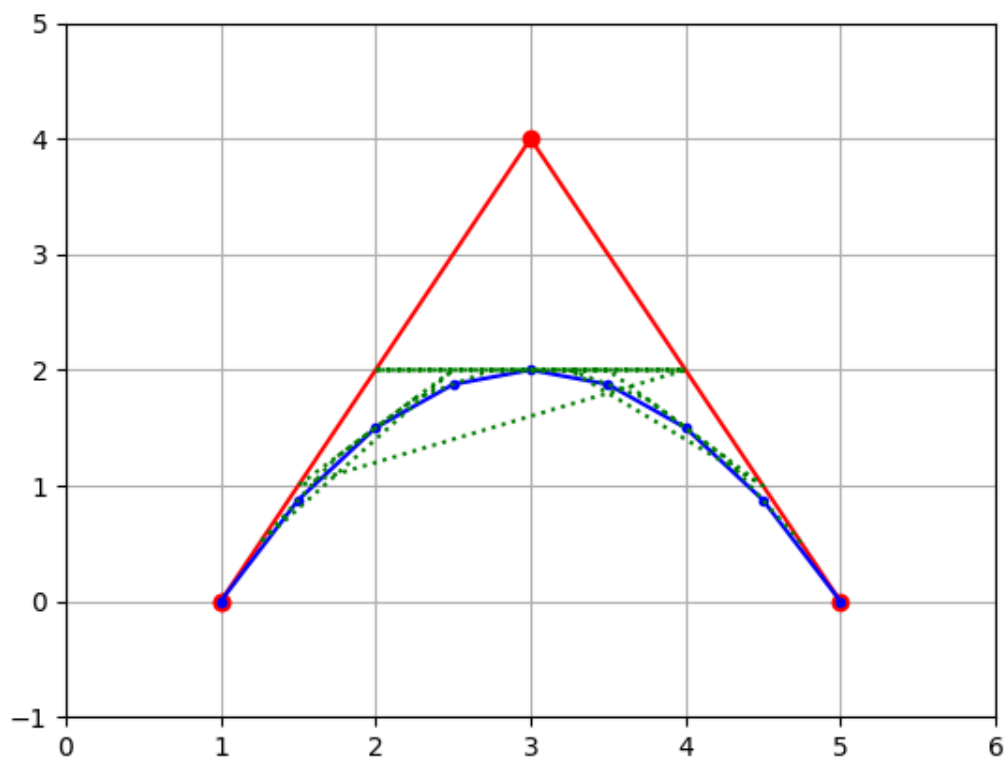
### 1. Test Case 1

Input:

```
2. Masuk lewat CLI (jika iterasinya besar)
3. Keluar
Masukkan pilihan: 2
1. Tiga Titik
2. N Titik
Masukkan pilihan: 1
Masukkan koordinat titik 1 (x y): 1 0
Masukkan koordinat titik 2 (x y): 3 4
Masukkan koordinat titik 3 (x y): 5 0
Masukkan iterasi: 3
```

Output:

```
Waktu eksekusi algoritma titik tengah: 0.05221366882324219
Waktu eksekusi algoritma brute force: 0.09179115295410156
Silahkan tutup plot untuk melanjutkan
```



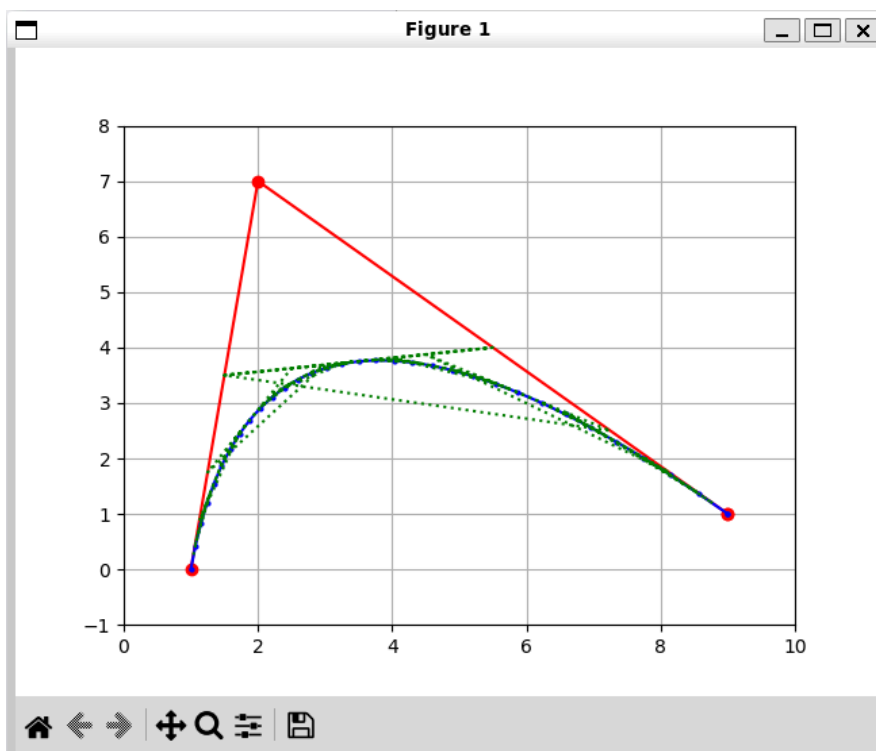
### 2. Test Case 2

Input:

```
2. Masuk lewat CLI (jika iterasinya besar)
3. Keluar
Masukkan pilihan: 2
1. Tiga Titik
2. N Titik
Masukkan pilihan: 1
Masukkan koordinat titik 1 (x y): 9 1
Masukkan koordinat titik 2 (x y): 2 7
Masukkan koordinat titik 3 (x y): 1 0
Masukkan iterasi: 5
```

Output:

```
Waktu eksekusi algoritma titik tengah: 0.05412101745605469
Waktu eksekusi algoritma brute force: 0.08511543273925781
Silahkan tutup plot untuk melanjutkan
```



### 3. Test Case 3

Input:

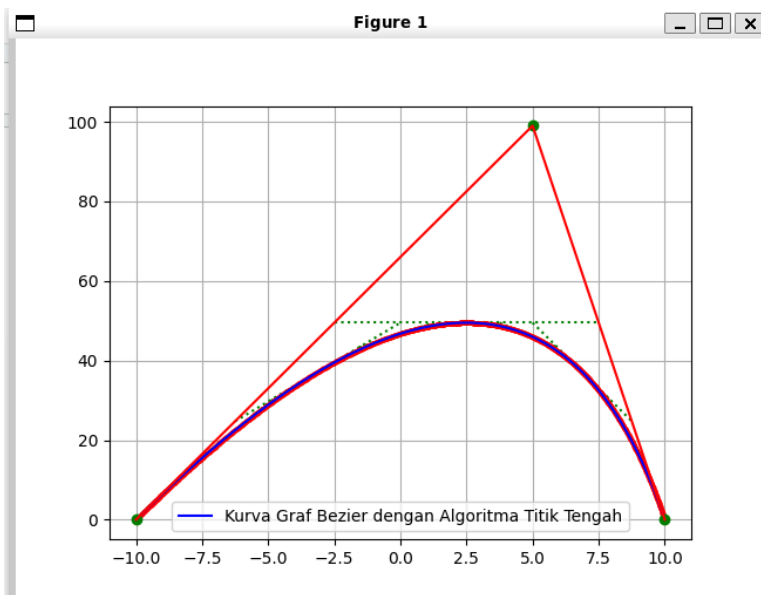
```
Masukkan pilihan: 2
1. Tiga Titik
2. N Titik
Masukkan pilihan: 1
Masukkan koordinat titik 1 (x y): -10 0
Masukkan koordinat titik 2 (x y): 5 99
Masukkan koordinat titik 3 (x y): 10 0
Masukkan iterasi: 10
```

```
Masukkan pilihan: 2
1. Tiga Titik
2. N Titik
Masukkan pilihan: 1
Masukkan koordinat titik 1 (x y): -10 0
Masukkan koordinat titik 2 (x y): 5 99
Masukkan koordinat titik 3 (x y): 10 0
Masukkan iterasi: 10
```

Output:

```
Waktu eksekusi algoritma titik tengah: 1.2104511260986328
Waktu eksekusi algoritma brute force: 1.0123252868652344
Silahkan tutup plot untuk melanjutkan
```

```
Masukkan iterasi: 10
Waktu eksekusi algoritma brute force: 0.6795039989810903
Waktu eksekusi algoritma titik tengah: 0.9101130017370451
Silahkan tutup plot untuk melanjutkan
```



#### 4. Tes Case 4

Input:

```
Masukkan pilihan: 2
1. Tiga Titik
2. N Titik
Masukkan pilihan: 1
Masukkan koordinat titik 1 (x y): 1 0
Masukkan koordinat titik 2 (x y): 3 4
Masukkan koordinat titik 3 (x y): 5 0
Masukkan iterasi: 20
```

Output:

```
Waktu eksekusi algoritma brute force: 561.9266033172607
Waktu eksekusi algoritma titik tengah: 922.8847026824951
Silahkan tutup plot untuk melanjutkan
```

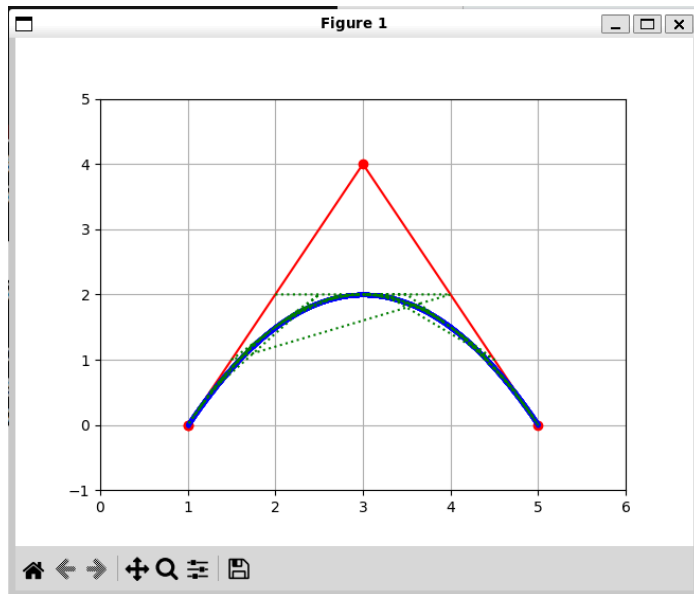
## 5. Test Case 5

Input:

```
Masukkan pilihan: 2
1. Tiga Titik
2. N Titik
Masukkan pilihan: 1
Masukkan koordinat titik 1 (x y): 1 0
Masukkan koordinat titik 2 (x y): 3 4
Masukkan koordinat titik 3 (x y): 5 0
Masukkan iterasi: 11
```

Output:

```
Masukkan koordinat titik 3 (x y): 5 0
Masukkan iterasi: 11
Waktu eksekusi algoritma brute force: 1.4893650004523806
Waktu eksekusi algoritma titik tengah: 1.8851290005841292
Silahkan tutup plot untuk melanjutkan
```



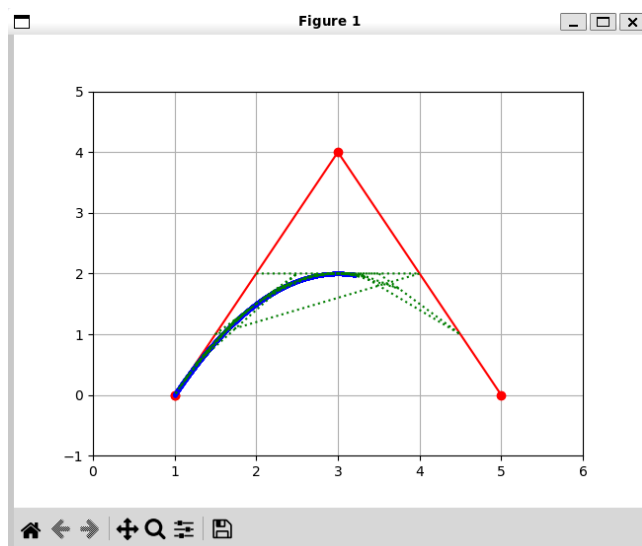
## 6. Test Case 6

Input:

```
Masukkan pilihan: 2
1. Tiga Titik
2. N Titik
Masukkan pilihan: 1
Masukkan koordinat titik 1 (x y): 1 0
Masukkan koordinat titik 2 (x y): 3 4
Masukkan koordinat titik 3 (x y): 5 0
Masukkan iterasi: 12
```

Output:

```
Masukkan koordinat titik 3 (x y): 5 0
Masukkan iterasi: 12
Waktu eksekusi algoritma brute force: 2.7479520031192806
Waktu eksekusi algoritma titik tengah: 3.238366000005044
Silahkan tutup plot untuk melanjutkan
```



## Bab V

### Analisis Perbandingan Solusi Brute Force dengan DnC

Berdasarkan Big O notation, secara garis besar kedua algoritma memiliki Big O notation yang sama yaitu  $O(2^n)$ , sehingga waktu eksekusinya tidak berbeda jauh. Hal yang menyebabkan kedua algoritma memiliki waktu eksekusi yang berbeda adalah  $T(n)$ nya yang memiliki perbedaan yang lebih spesifik lagi. Dalam hal kompleksitas waktu, algoritma *Brute Force* memiliki nilai yang lebih besar dibandingkan algoritma *divide and conquer*.

$T(n)$  pada algoritma *Brute Force* adalah:

$$T(n) = 16(2^n - 1),$$

Sementara pada algoritma *Divide and Conquer* adalah:

$$T(n) = 13 \cdot 2^n - 11$$

Berdasarkan perhitungan-perhitungan di atas, algoritma *Divide and Conquer* memiliki efisiensi yang lebih baik dan waktu eksekusi yang lebih cepat dibandingkan algoritma *Brute Force* untuk iterasi yang bernilai kecil. Namun, saat iterasi menjadi semakin besar, algoritma *Brute Force* akan memiliki waktu eksekusi yang lebih baik dibanding *Divide and Conquer*.

Hal yang menyebabkan perbedaan tersebut adalah karena pada algoritma *Divide and Conquer*, semakin banyak iterasi yang dimasukkan maka semakin banyak proses konkatenasi sub-larik yang dilakukan. Konkatenasi memiliki kompleksitas waktu dan ruang yang cukup besar, karena menggunakan dua sub-larik yang kemudian disalin dan digabungkan ke sebuah larik baru. Konkatenasi pada Python dengan menggunakan operator + memiliki kompleksitas waktu  $O(n)$  dan kompleksitas ruang  $O(n)$ . Hal ini menyebabkan proses yang dibutuhkan meningkat signifikan pada algoritma *Divide and Conquer*. Pada perhitungan  $T(n)$ , konkatenasi yang dihitung adalah konkatenasi solusi akhir, sedangkan operasi konkatenasi terjadi pada setiap pemanggilan rekursif fungsi.

Pada implementasi dengan Python, waktu eksekusi pada CLI dan GUI memiliki perbedaan. Waktu eksekusi pada GUI lebih cepat dibandingkan waktu eksekusi di CLI untuk kedua algoritma. Pada iterasi yang kecil (sekitar 10 ke bawah), hasil waktu eksekusi memiliki perbedaan yang tipis sehingga terkadang algoritma *Brute Force* yang lebih cepat. Selain itu, pada Python ada Pycache yang membuat hasil waktu eksekusinya menjadi lebih cepat. Hal ini membuat waktu eksekusi algoritma *Divide and Conquer* menjadi lebih cepat. Namun, pada iterasi yang besar (di atas 10) waktu eksekusi untuk algoritma *Brute Force* secara konsisten akan lebih cepat dibandingkan dengan algoritma *Divide and Conquer*.

Meskipun kedua algoritma memberikan waktu eksekusi dan kompleksitas waktu yang berbeda, namun kedua algoritma akan memberikan titik solusi dan bentuk kurva yang sama.



## Bab VI

### Implementasi Bonus

Algoritma *Divide and Conquer* dapat dikembangkan untuk dapat membuat kurva bezier kubik, kuartik, dan seterusnya hingga  $N$  titik kontrol. Langkah-langkah untuk menyelesaikan persoalan membentuk kurva bezier kuadratik dengan menggunakan Algoritma *Divide and Conquer* adalah sebagai berikut:

1. Masukkan titik kontrol masukan ke dalam suatu larik  $P$ .
2. **Tahap Solve:** Lakukan iterasi untuk mencari titik tengah pada suatu larik titik. Iterasi yang pertama adalah mencari titik tengah dari titik kontrol pertama  $P_0$  dengan titik kontrol antara  $P_1$ , titik kontrol antara  $P_1$  dengan titik kontrol antara  $P_2$ , dan seterusnya hingga titik kontrol antara  $P_{n-1}$  dengan titik kontrol terakhir  $P_n$ . Masukkan hasil titik tengah tersebut ke dalam suatu larik  $Q$  yang anggotanya  $Q_1$  hingga  $Q_{n-1}$ . Lakukan hal yang sama pada larik  $Q$  dan hasil larik berikutnya. Iterasi akan dilakukan hingga hasil larik berjumlah 1 titik tengah. Larik tersebut akan diberi nama larik  $R$ .
3. Masukkan elemen pertama dari larik  $P$  atau titik kontrol pertama ke dalam larik solusi hanya pada iterasi yang pertama.
4. **Tahap Divide:** Partisi titik-titik yang telah didapatkan menjadi 2 bagian, yaitu kanan dan kiri. Masukkan elemen pertama dari tiap larik yang terbentuk ke dalam larik Kiri dan masukkan elemen terakhir dari tiap larik yang terbentuk ke dalam larik Kanan.
5. Lakukan kembali dari **tahap Solve** untuk larik Kiri jika  $N > 1$  dan dilakukan sebanyak  $N - 1$  kali dengan  $N$  adalah jumlah iterasi.
6. Masukkan elemen dari larik  $R$  ke dalam larik solusi.
7. Lakukan kembali dari **tahap Solve** untuk larik Kanan jika  $N > 1$  dan dilakukan sebanyak  $N - 1$  kali dengan  $N$  adalah jumlah iterasi.
8. Masukkan elemen terakhir pada larik  $P$  ke dalam larik solusi hanya pada iterasi pertama.

Sama seperti pada persoalan pembentukan kurva bezier kuadratik, pada persoalan ini, masukan titik-titik awal sudah merupakan sebuah upa-persoalan sehingga dapat langsung melakukan tahap *Solve*. Pada tahap *Solve*, operasi ini akan menghasilkan titik solusi yang dibutuhkan untuk membentuk kurva bezier. Selain menghasilkan solusi, tahap *Solve* juga membuat persoalan menjadi lebih besar. Setiap kali melakukan tahap *Solve*, jumlah titik akan menjadi  $2^N + 1$  dengan  $N$  adalah jumlah iterasi. Oleh karena itu, pada iterasi selanjutnya titik-titik yang ada tidak dapat langsung dilakukan tahap *Solve* dan harus melalui tahap *Divide* terlebih dahulu. Setelah dilakukan partisi menjadi bagian kanan dan bagian kiri, masing-masing bagian dapat melakukan tahap *Solve*. Tahap 5 dan 7 akan melakukan pemanggilan Algoritma *Divide and Conquer* secara rekursif sesuai dengan jumlah iterasi yang diinginkan dikurangi satu.

Kompleksitas waktu untuk membuat kurva bezier n titik kontrol dengan Algoritma *Divide and Conquer* dihitung dari banyaknya jumlah perhitungan titik tengah dan operasi konkatenasi solusi adalah

$$g(p) = 4\left(\frac{p}{2}(p-1)\right), p \geq 3$$

$$f(n) = 2 + (2^n - 1) = 2^n + 1$$

$$T(p, n) = \left(\sum_{i=1}^n 2^{i-1}\right)g(p) + f(n) = (2^n - 1)g(p) + f(n), p = p_0, n \geq 0$$

$$T(p, n) = (2^n - 1)\left(4\left(\frac{p}{2}(p-1)\right)\right) + (2^n + 1), p = p_0, n \geq 0$$

Sehingga Big O notation-nya adalah:

$$O(p, n) = 2^n p^2, n \geq 0$$

*n* adalah jumlah iterasi

*p* adalah jumlah titik masukan

*p*<sub>0</sub> adalah jumlah titik kontrol yang diinginkan

*g(p)* adalah banyaknya operasi perhitungan titik tengah

*f(n)* adalah banyaknya operasi konkatenasi untuk solusi akhir

*T(n)* adalah kompleksitas waktu secara keseluruhan

# Lampiran

Link Github:

[https://github.com/FedrianzD/Tucil2\\_13522071\\_13522090](https://github.com/FedrianzD/Tucil2_13522071_13522090)

Referensi:

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

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf)

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