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

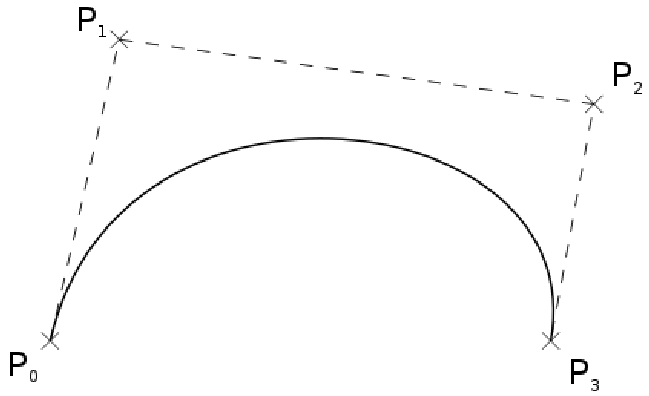
Institut Teknologi Bandung

2024

Daftar Isi

Bab I

Deskripsi Masalah



Gambar 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 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.

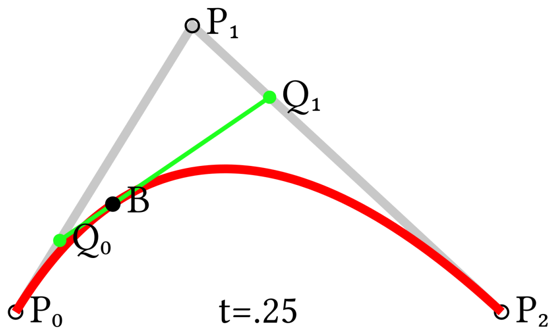
sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan $B(t)$ membentuk sebuah garis lurus dari P_0 ke P_1.

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.

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



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.

$$S_0 = B(t) = (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + t^3P_3, \qquad t \in [0,1]$$

$$T_0 = B(t) = (1-t)^4P_0 + 4(1-t)^3tP_1 + 6(1-t)^2t^2P_2 + 4(1-t)t^3P_3 + t^4P_4, \qquad t \in [0,1]$$

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***.

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:

a. 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 menghubunhkan P_1 dan P_2.
b. Hubungkan Q_0 dan Q_1 sehingga terbentuk sebuah garis baru.
c. Buatlah sebuah titik baru R_0 yang berada di tengah Q_0 dan Q_1.
d. 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.

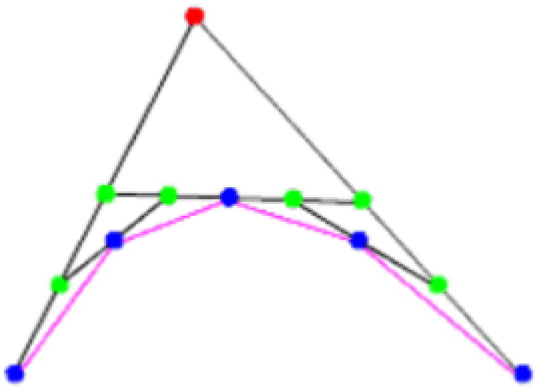
Buatlah beberapa titik baru, yaitu S_0 yang berada di tengah P_0 dan Q_0, S_1 yang berada di tengah Q_0 dan R_0, S_2 yang berada di tengah R_0 dan Q_1, dan S_3 yang berada di tengah Q_1 dan P_2.

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 Tucil 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

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

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

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

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

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

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 kuadratik, 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. **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 kiri, 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 pertambahan dan pengurangan yang ada pada operasi titik tengah, dan konkatenasi solusi akhir adalah

$$g(p) \;=\; 4(\tfrac{p}{2}(p - 1)),\; p \;\geq\; 3$$

$$f(n) \;= 2 \;+\; (2^{n} - 1) \;=\; 2^{n} \;+\; 1$$

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

$$T(p, n) \;=\; (2^{n} - 1)(4(\tfrac{p}{2}(p - 1))) \;+\; (2^{n} \;+\; 1),\; p \;=\; p_{0},\; n \;\geq\; 0$$

Sehingga Big O notation-nya adalah:

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

n *adalah jumlah iterasi*

p *adalah jumlah titik masukan*

p_{0} *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

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

$$T(n) \;=\; 13.\,2^{n} \;-\; 11,\; n \;\geq\; 0$$

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

```
def Bezier3Point(point1, point2, point3, iterate, iterateMax):
    solution = []
    # titikBantu = []
    if iterate == iterateMax:
        pos1 = DnC(point1, point2)
        pos2 = DnC(point2, point3)

        if iterateMax == 1:
            solution.append(point1)
        #     titikBantu.append([point1])
        # titikBantu.append([pos1, pos2])

        solution.append(DnC(pos1, pos2))

        if iterateMax == 1:
            solution.append(point3)
            # titikBantu.append([point3])

        return solution

    elif iterate < iterateMax:
        pos1 = DnC(point1, point2)
        pos2 = DnC(point2, point3)

        if iterate == 1:
            solution.append(point1)

        # titikBantu.append([pos1, pos2])

        temp = Bezier3Point(point1, pos1, DnC(pos1, pos2), iterate + 1, iterateMax)
        solution += temp
        # titikBantu += temp[1]

        solution.append(DnC(pos1, pos2))

        temp = Bezier3Point(DnC(pos1, pos2), pos2, point3, iterate + 1, iterateMax)
        solution += temp
        # titikBantu += temp[1]

        if iterate == 1:
            solution.append(point3)

        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

```python
def BezierNPoint(arr, iterate, iterateMax):
    solution = []
    titikBantu = []
    if iterate == iterateMax:
        tempArr = copy.deepcopy(arr) # buat menampung titik solusi sementara
        temp = []
        while len(temp) != 1:
            temp = []
            for i in range(len(tempArr)-1):
                temp.append(DnC(tempArr[i], tempArr[i+1]))
            tempArr = temp
            titikBantutemp = copy.deepcopy(temp)
            titikBantu.append(titikBantutemp)
        titikBantu = titikBantu[:-1]

        if iterateMax == 1:
            solution += [arr[0]]
            titikBantu.append(arr[0])

        solution += tempArr

        if iterateMax == 1:
            solution += [arr[-1]]
            titikBantu.append(arr[0])

        return solution, titikBantu
    elif iterate < iterateMax:

        # Titik Awal
        if iterate == 1:
            solution += [arr[0]]

        # Tahap Awal
        arr2 = arr
        arrKiri = []
        arrKanan = []
        while len(arrKiri) != len(arr):
            temp = []
            for i in range(len(arr2)-1):
                temp.append(DnC(arr2[i], arr2[i+1]))
            arrKiri.append(arr2[0])
            arrKanan.append(arr2[-1])
            arr2 = temp
            titikBantutemp = copy.deepcopy(temp)
            titikBantu.append(titikBantutemp)
        titikBantu = titikBantu[:-2]

        # Bagian Kiri
        tempCall = BezierNPoint(arrKiri, iterate+1, iterateMax)
        solution += tempCall[0]
        titikBantu += tempCall[1]

        # Bagian Tengah
        solution += [BezierNPoint(arr, 1, 1)[0][1]]

        # Bagian Kanan
        arrKanan = list(reversed(arrKanan))

        tempCall = BezierNPoint(arrKanan, iterate+1, iterateMax)
        solution += tempCall[0]
        titikBantu += tempCall[1]

        # Titik Akhir
        if iterate == 1:
            solution += [arr[-1]]
        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.

```python
def BezierBruteforce(p1, p2, p3, iterate):
    npoint = 2**iterate + 1
    temp = 1/ (npoint-1)
    n = temp
    solution = []
    solution += [p1]
    count = 0
    while n <= (1-temp):
        rx = ((1-n)**2)*p1[0] + 2*(1-n)*n*p2[0] + (n**2)*p3[0]
        ry = ((1-n)**2)*p1[1] + 2*(1-n)*n*p2[1] + (n**2)*p3[1]
        solution += [(rx,ry)]
        n += temp
        count+=1
    solution += [p3]
    return solution
```

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

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

```python
def parseArrayNPoint(titikBantu):
    temp = []
    for array in titikBantu:
        for subarray in array:
            temp.append(subarray)

    new = []
    res = []
    for point in temp:
        if len(new) != 2:
            new.append(point)
        else:
            res.append(new)
    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).

```python
def animatePlot(arrayOfPoints, arrayOfSol, arrayOfHelper):
    plt.close()
    fig, ax = plt.subplots()
    linePoints, = ax.plot([], [], 'ro-')
    lineSol, = ax.plot([], [], 'bo-', markersize=2 if len(arrayOfSol) > 30 else 3)
    lineHelper, = ax.plot([], [], linestyle='dotted', color='green')

    ax.set_xlim(min([point[0] for point in arrayOfPoints]) - 1, max([point[0] for point in arrayOfPoints]) + 1)
    ax.set_ylim(min([point[1] for point in arrayOfPoints]) - 1, max([point[1] for point in arrayOfPoints]) + 1)
    currentPointsX = []
    currentPointsY = []
    currentSolX = []
    currentSolY = []
    currentHelperX = []
    currentHelperY = []
    currentIteration = 0
    def animate(i):
        if i < len(arrayOfSol):
            currentSolX.append(arrayOfSol[i][0])
            currentSolY.append(arrayOfSol[i][1])
            lineSol.set_data(currentSolX, currentSolY)
        if i < len(arrayOfPoints):
            currentPointsX.append(arrayOfPoints[i][0])
            currentPointsY.append(arrayOfPoints[i][1])
            linePoints.set_data(currentPointsX, currentPointsY)
        if i < len(arrayOfHelper):
            currentHelperX.append([point[0] for point in arrayOfHelper[i]])
            currentHelperY.append([point[1] for point in arrayOfHelper[i]])
            lineHelper.set_data(currentHelperX, currentHelperY)

    if len(arrayOfHelper) < 100:
        interval = 200
    elif 100 <= len(arrayOfHelper) < 800:
        interval = 50
    else:
        interval = 2
    ani = FuncAnimation(fig, animate, frames=max(len(arrayOfSol), len(arrayOfHelper), len(arrayOfPoints)), interval=interval , repeat=False)
    plt.grid()
    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.

```python
if __name__ == "__main__":
    while True:
        print("BEZIER CURVE GENERATOR")
        print('1. Masuk lewat GUI\n2. Masuk lewat CLI (jika iterasinya besar)\n3. Keluar')
        try:
            choice = int(input("Masukkan pilihan: "))
        except ValueError:
            print("Input harus berupa integer!\n")
            continue

        if choice == 1:
            App = Gui()
            print("Untuk mengakhiri program, tutup semua window GUI yang berjalan")
            App.mainloop()
            break

        elif choice == 2:
            print('1. Tiga Titik\n2. N Titik')
            while True:
                try:
                    choice = int(input("Masukkan pilihan: "))
                except ValueError:
                    print("Input harus berupa integer!")
                    continue
                if choice == 1 or choice == 2:
                    break
                else:
                    print('Pilihan tidak tersedia! Silahkan masukkan ulang (1/2)')
                    continue

            if choice == 1:
                point1, point2, point3 = threePointInput()
                while True:
                    try:
                        iterasi = int(input("Masukkan iterasi: "))
                        if iterasi < 1:
                            print("Iterasi minimal 1 kali!")
                            continue
                        else:
                            break
                    except ValueError:
                        print("Input harus berupa integer!")
                        continue
                startBrute = time.time()
                sol2 = function.BezierBruteforce(point1, point2, point3, iterasi)
                endBrute = time.time()
                print("Waktu eksekusi algoritma brute force: ", (endBrute - startBrute) * 1000)
                startMid = time.time()
                sol = function.Bezier3Point(point1, point2, point3, 1, iterasi)
                endMid = time.time()
                titikBantu = function.Bezier3PointHelper(point1, point2, point3, 1, iterasi)
                print("Waktu eksekusi algoritma titik tengah: ", (endMid - startMid) * 1000)
                titikBantu = function.parseArrayNPoint(titikBantu)
                print("Silahkan tutup plot untuk melanjutkan")
                function.animatePlot([point1, point2, point3], sol, titikBantu)
                # function.showPlot([point1, point2, point3], sol, titikBantu)

            elif choice == 2:
                arr = nPointInput()
                while True:
                    try:
                        iterasi = int(input("Masukkan iterasi: "))
                        if iterasi < 1:
                            print("Iterasi minimal 1 kali!")
                            continue
                        else:
                            break
                    except ValueError:
                        print("Input harus berupa integer!")
                        continue
                startMid = time.time()
                temp = function.BezierNPoint(arr, 1, iterasi)
                sol = temp[0]
                titikBantu = temp[1]
                endMid = time.time()
                new_array = function.parseArrayNPoint(titikBantu)
                print("Waktu eksekusi algoritma titik tengah: ", (endMid - startMid) * 1000)
                print("Silahkan tutup plot untuk melanjutkan")
                function.animatePlot(arr, sol, new_array)
                # function.showPlot(arr, sol, new_array)
            else:
                print("\nPilihan tidak valid")

        elif choice == 3:
            break
        else:
            print("Pilihan tidak tersedia!\n")
            continue
        print('\n')
```

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

```python
def threePointInput():
    arr = []
    for i in range(3):
        while True:
            try:
                temp = (tuple(map(float, input(f"Masukkan koordinat titik {i+1} (x y): ").split())))
                if len(temp) != 2:
                    print("Koordinat harus terdiri dari 2 nilai!")
                else:
                    arr.append(temp)
                    break
            except ValueError:
                print("Input harus berupa integer!")
    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.

```python
def nPointInput():
    while True:
        try:
            n = int(input("Masukkan jumlah titik: "))
            if n <= 2:
                print("Minimal 2 titik!")
                continue
            else:
                break
        except ValueError:
            print("Input harus berupa integer!")
    arr = []
    for i in range(n):
        while True:
            try:
                temp = (tuple(map(float, input(f"Masukkan koordinat titik {i+1} (x y): ").split())))
                if len(temp) != 2:
                    print("Koordinat harus terdiri dari 2 nilai!")
                else:
                    arr.append(temp)
                    break
            except ValueError:
                print("Input harus berupa integer!")
    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.

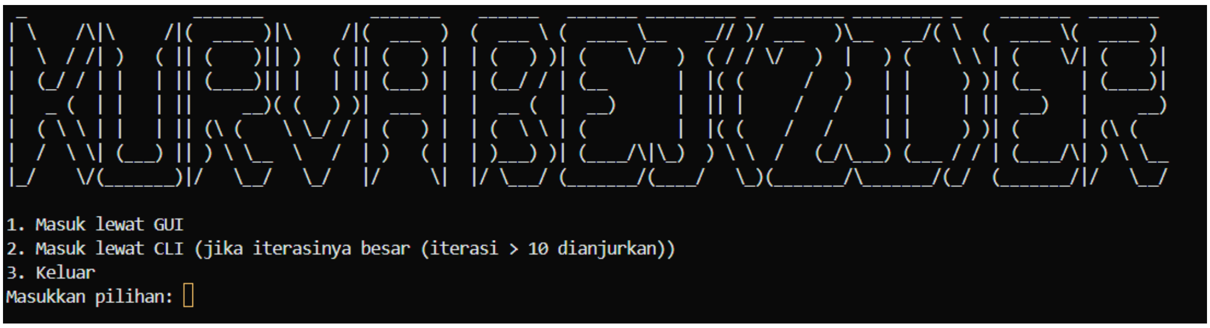
3. **GUI.py**

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

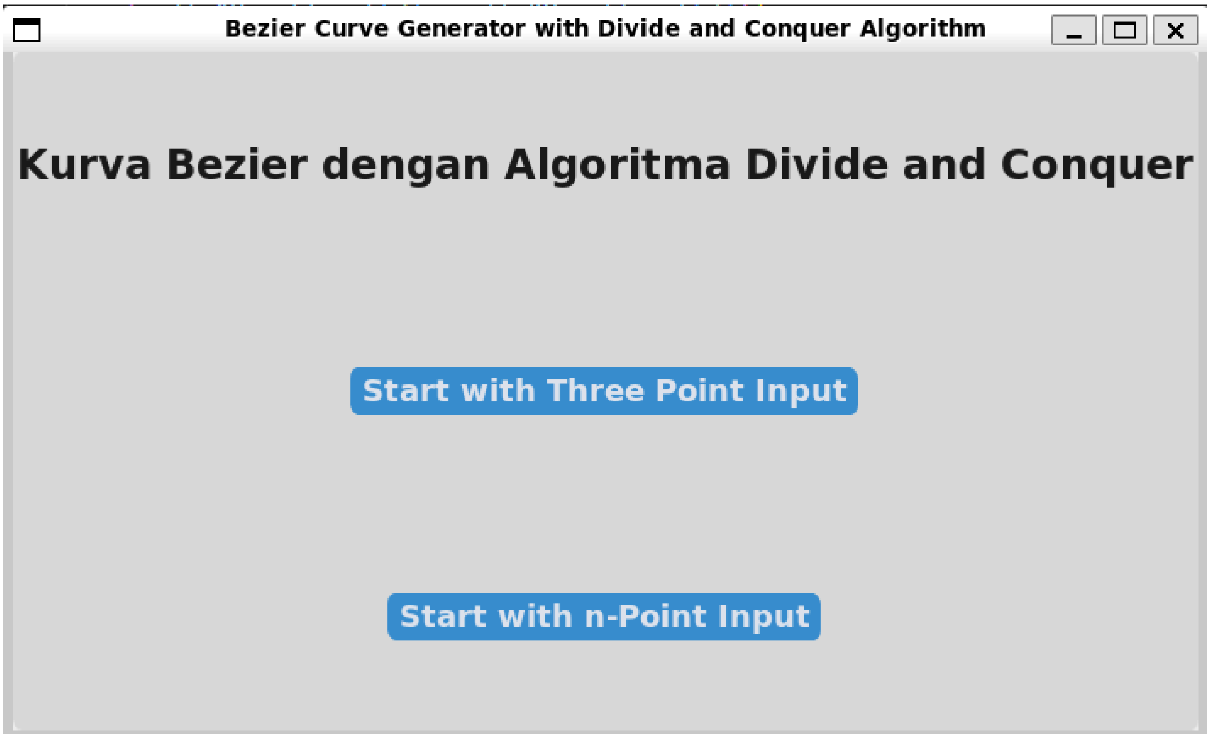
```python
class Gui(ctk.CTk):
    def __init__(self):
        # initialization
        super().__init__()
        ctk.set_appearance_mode("light")
        ctk.set_default_color_theme("blue")
        self.title("Bezier Curve Generator with Divide and Conquer Algorithm")
        self.columnconfigure(0, weight=1)
        self.rowconfigure(0, weight=1)

        # attribute
        self.XPointInput = []
        self.YPointInput = []
        self.solutionResult = []
        self.arrayOfInput = []
        self.titikBantu = []
        self.mainPage = ctk.CTkFrame(self)
        self.pageThree = ctk.CTkFrame(self)
        self.pageN = ctk.CTkFrame(self)
        self.pagePlot = ctk.CTkFrame(self)
        self.error = ""

        # create frame
        self.create_main_page()
        self.create_page_three()
        self.create_page_n()

        # show page
        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()*.

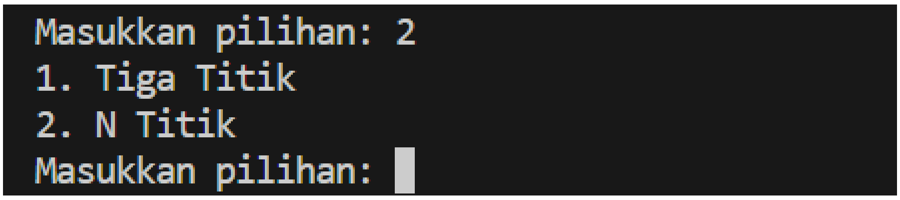
Tampilan Awal Running



Tampilan Awal GUI



Tampilan CLI

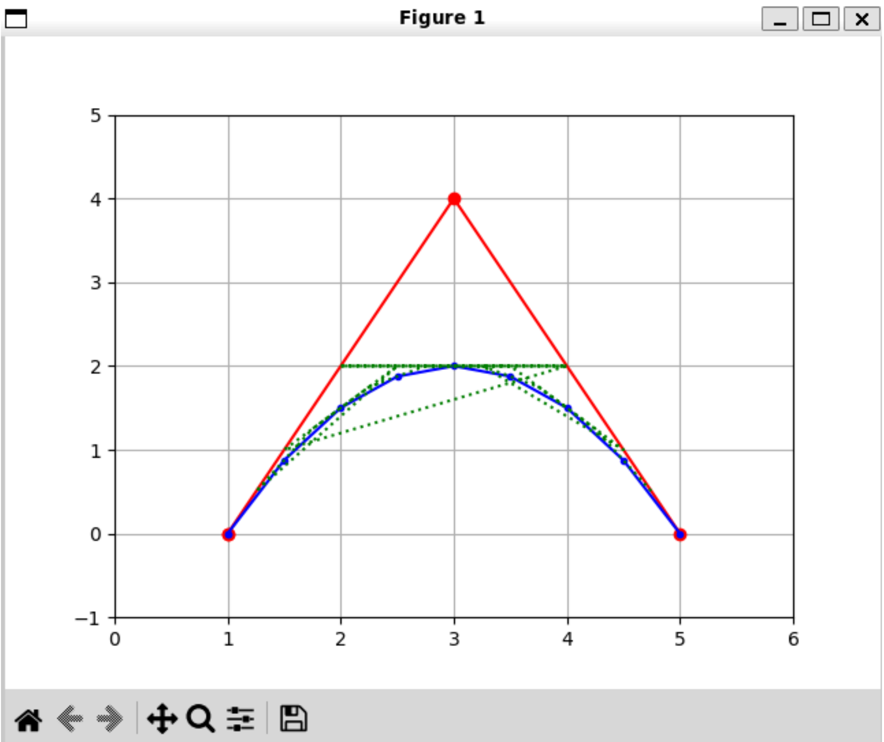


B. Test Case 3 Titik

1. Test Case 1 (3 Titik)

Input:

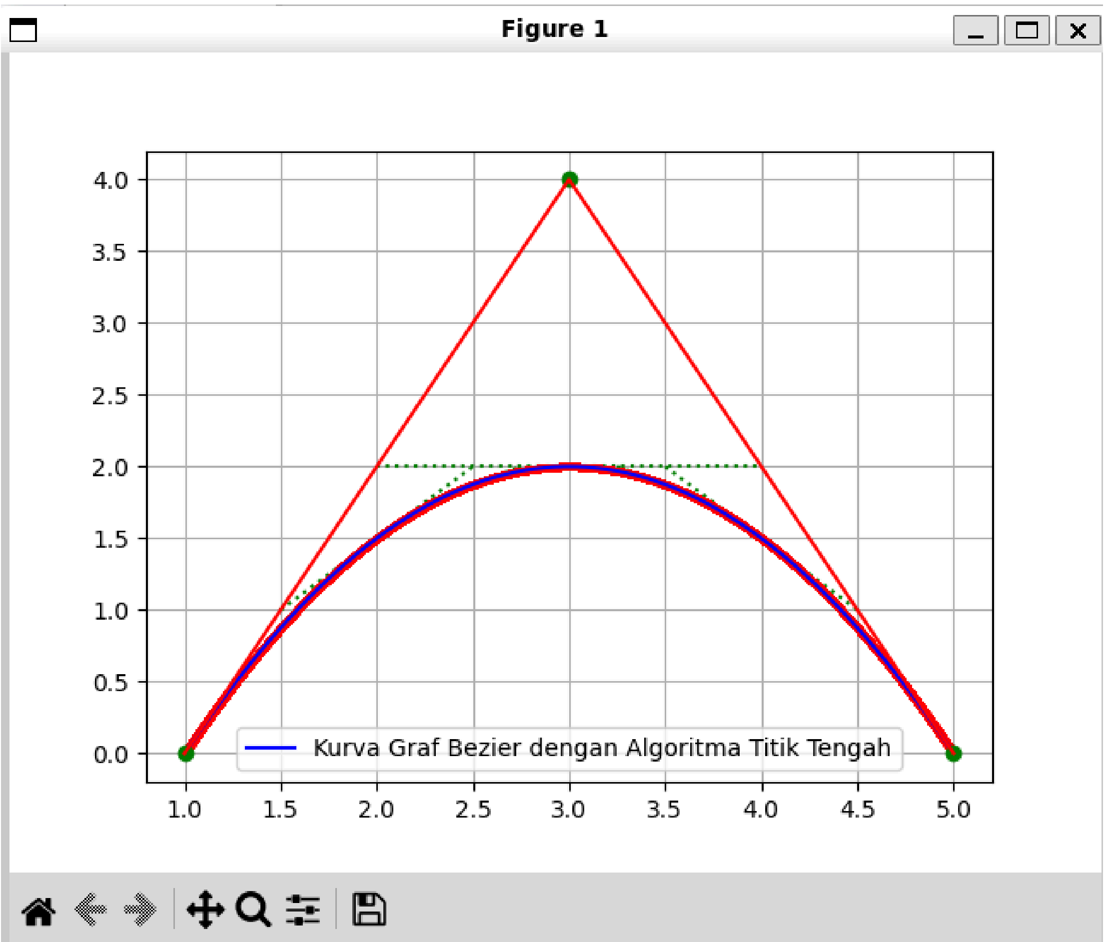
Output:



2. Test Case 2 (3 Titik)

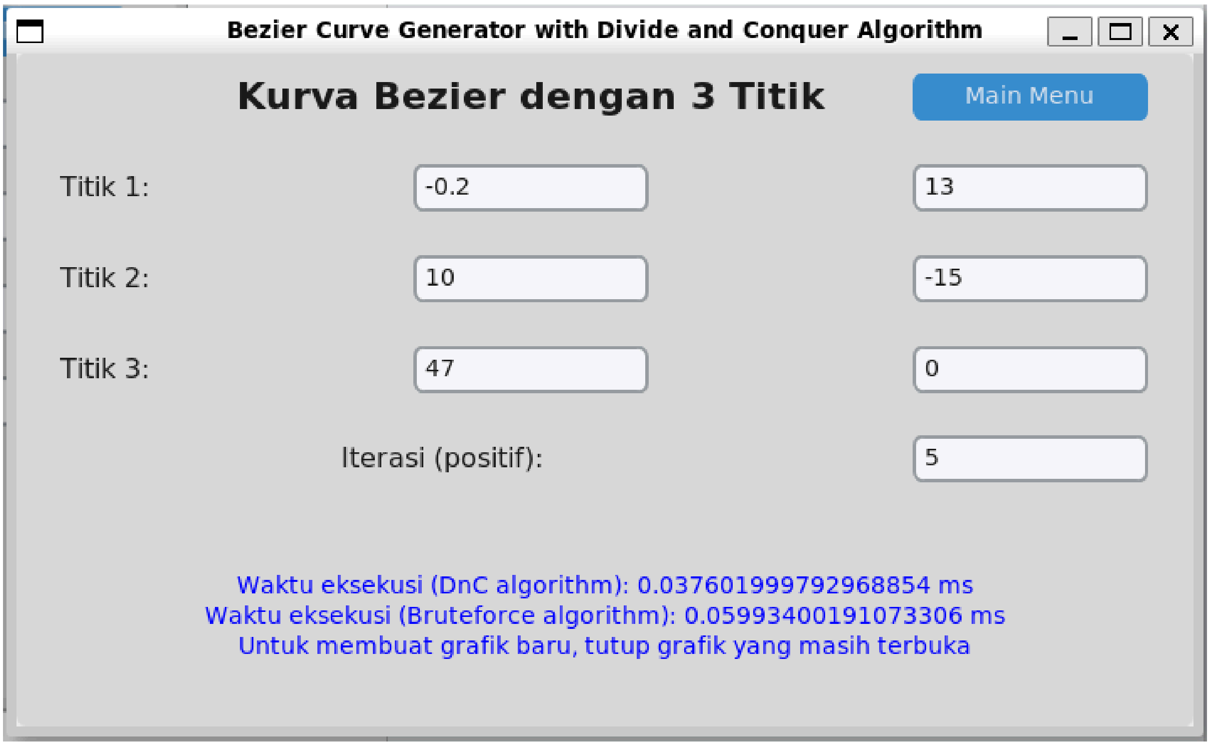
Input:

Output:

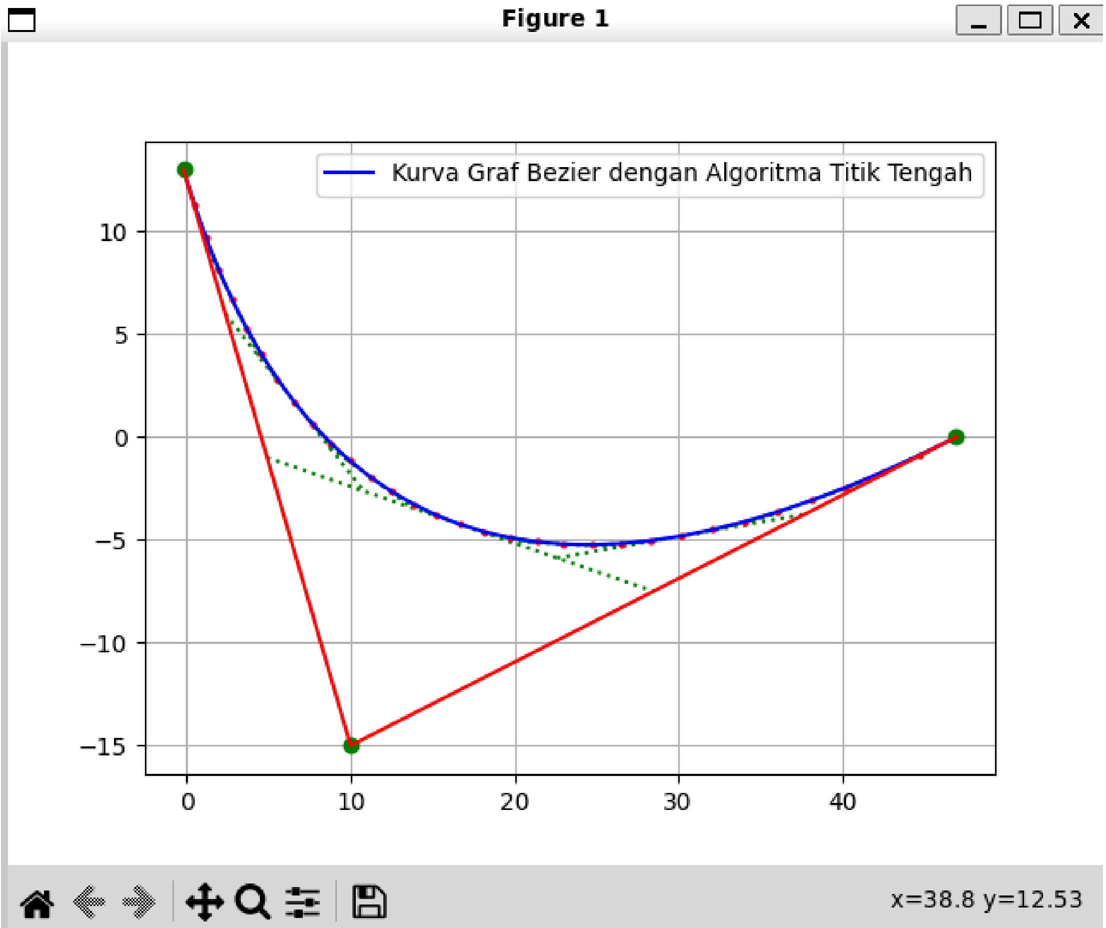


3. Test Case 3 (3 Titik)

Input:

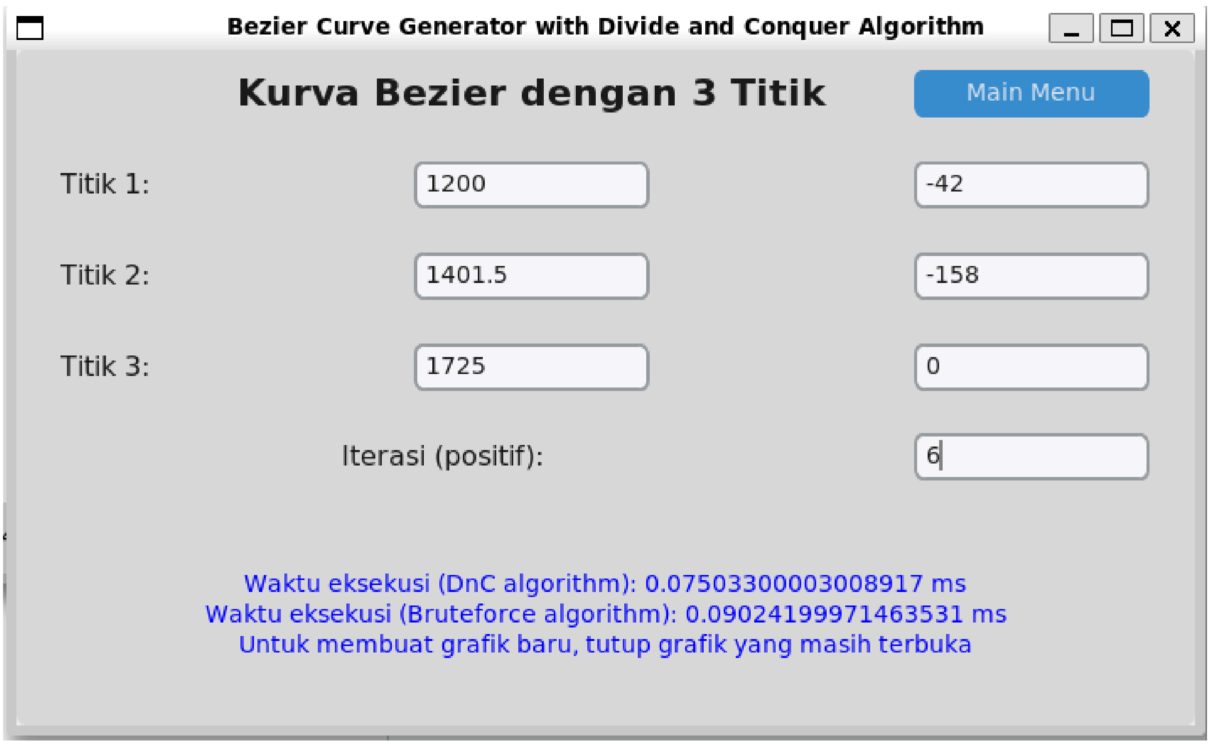


Output:

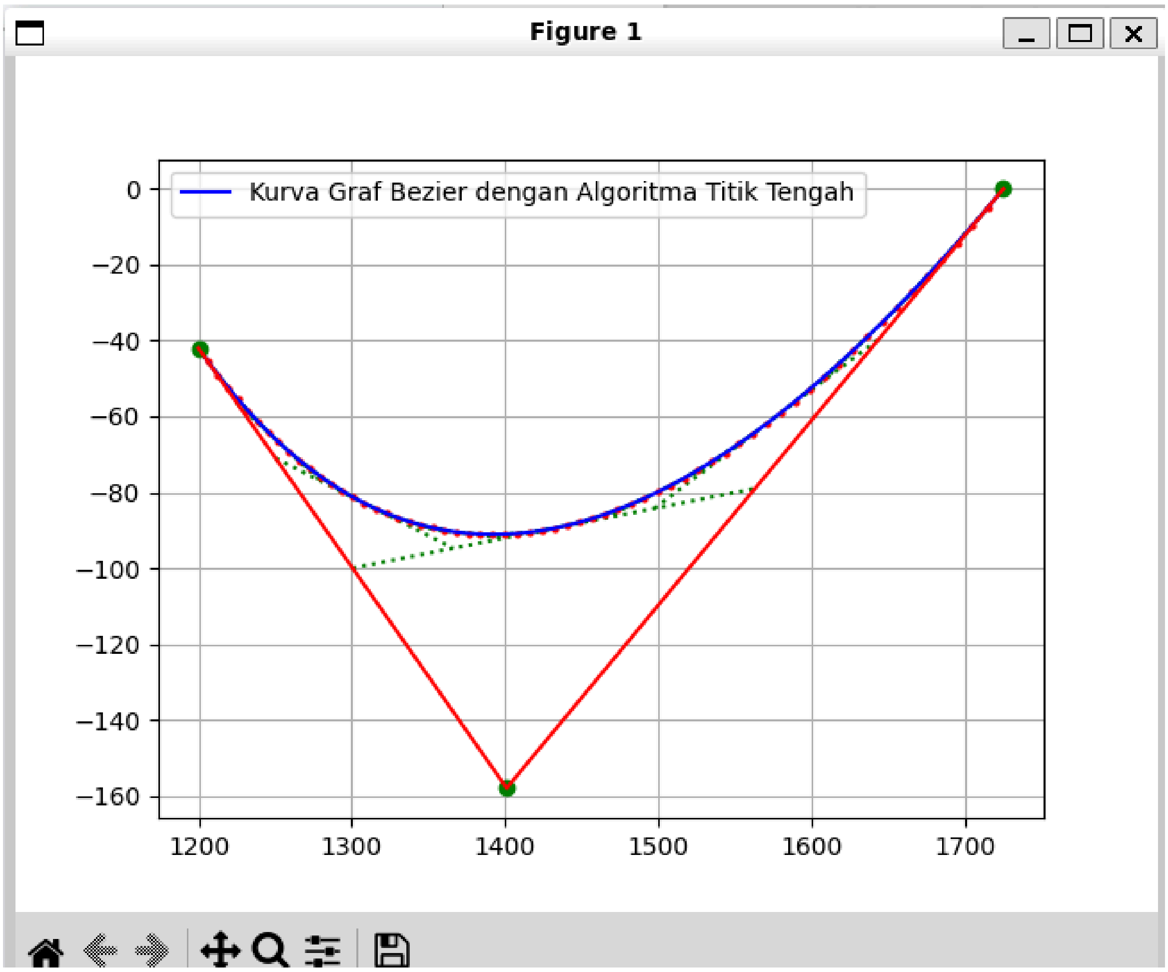


4. Test Case 4

Input:

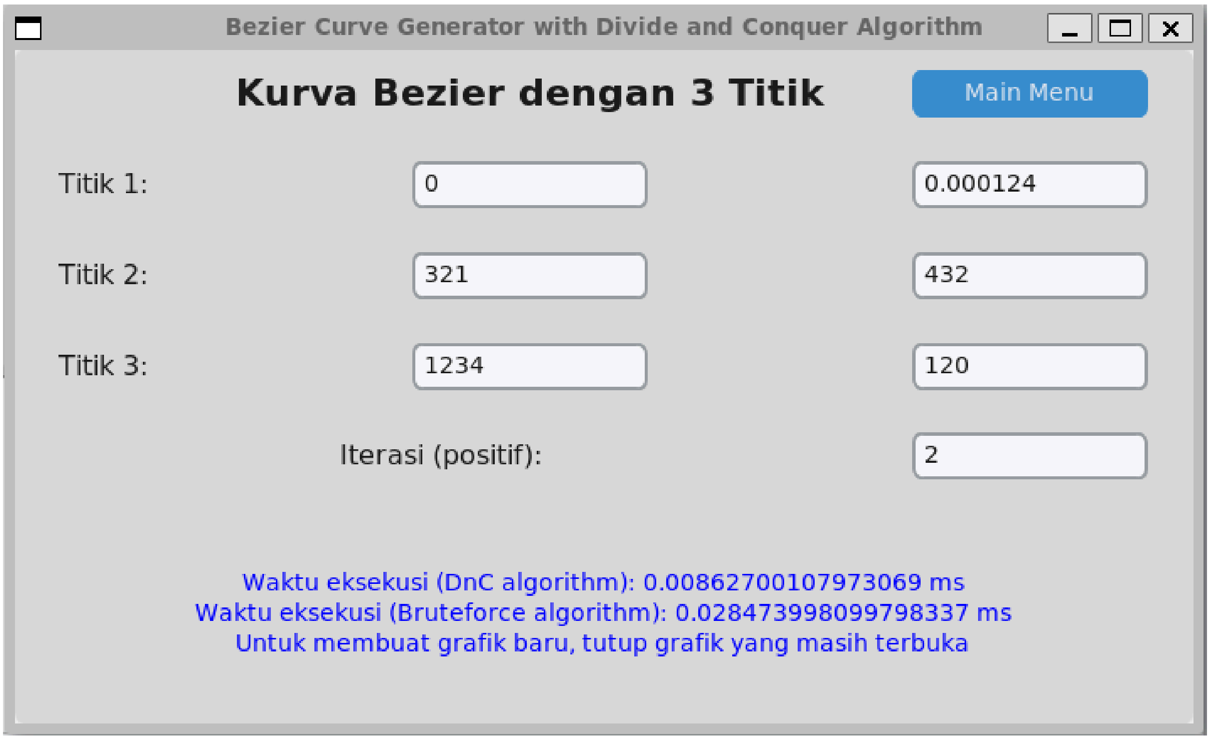


Output:

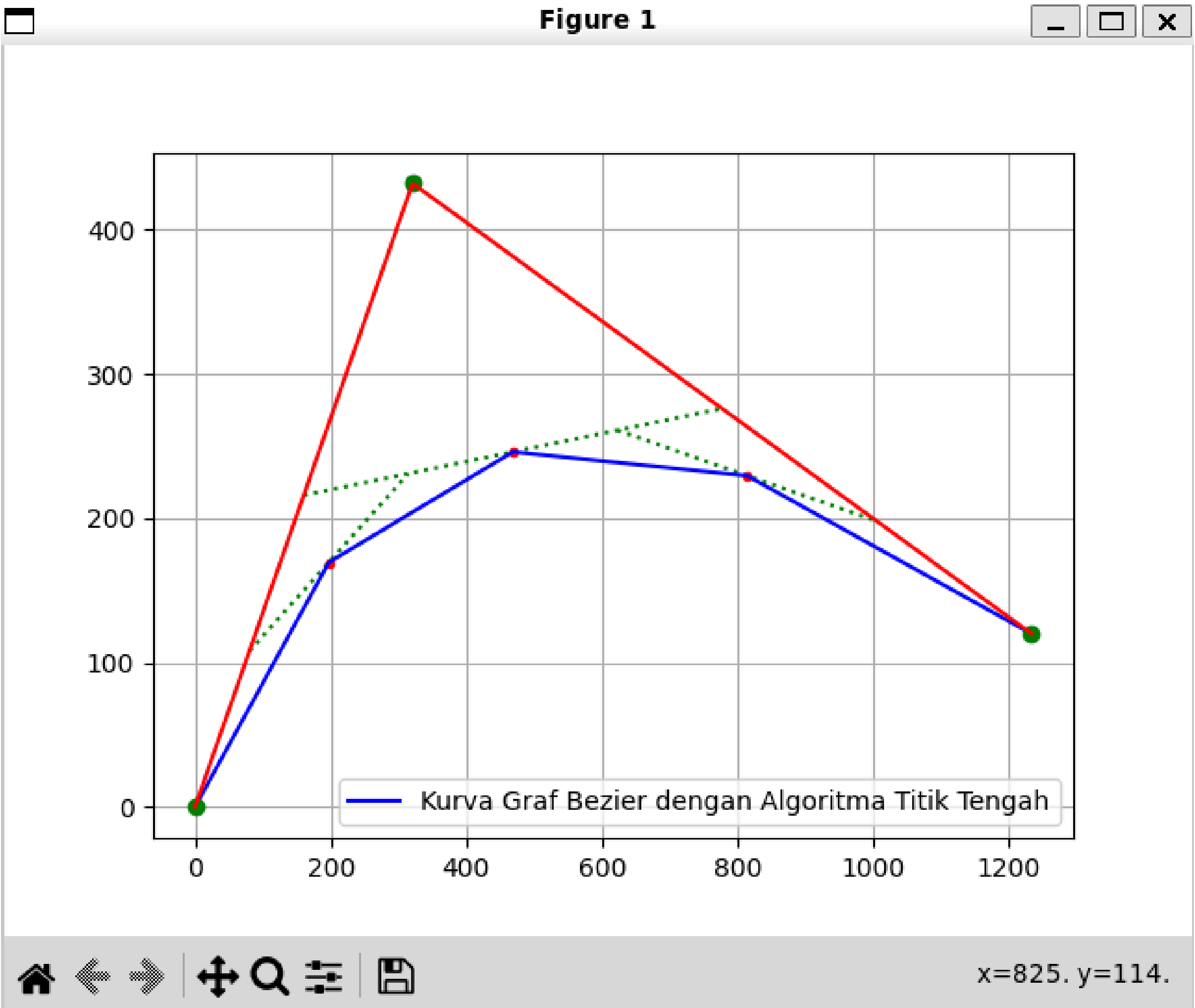


5. Test Case 5

Input:

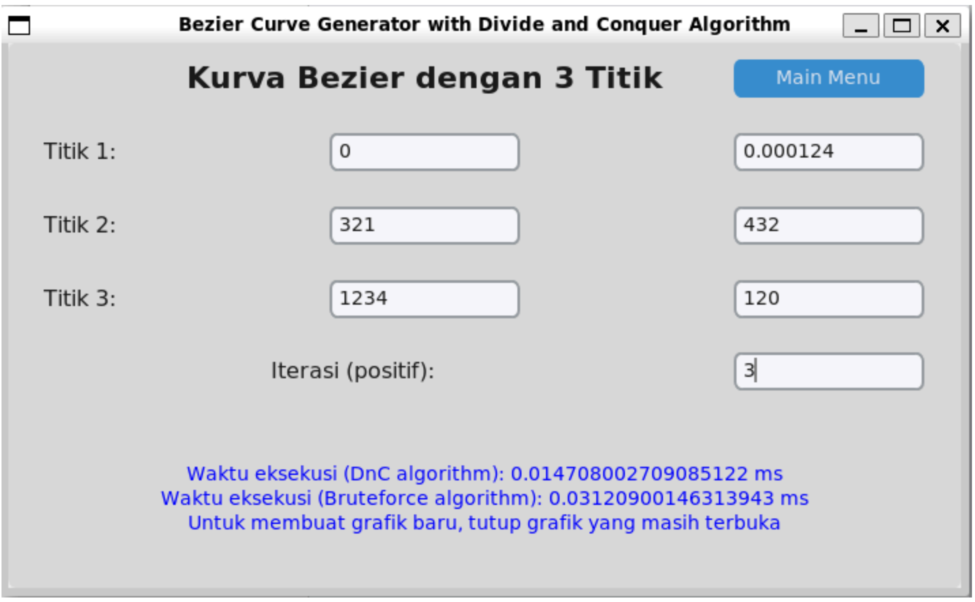


Output:

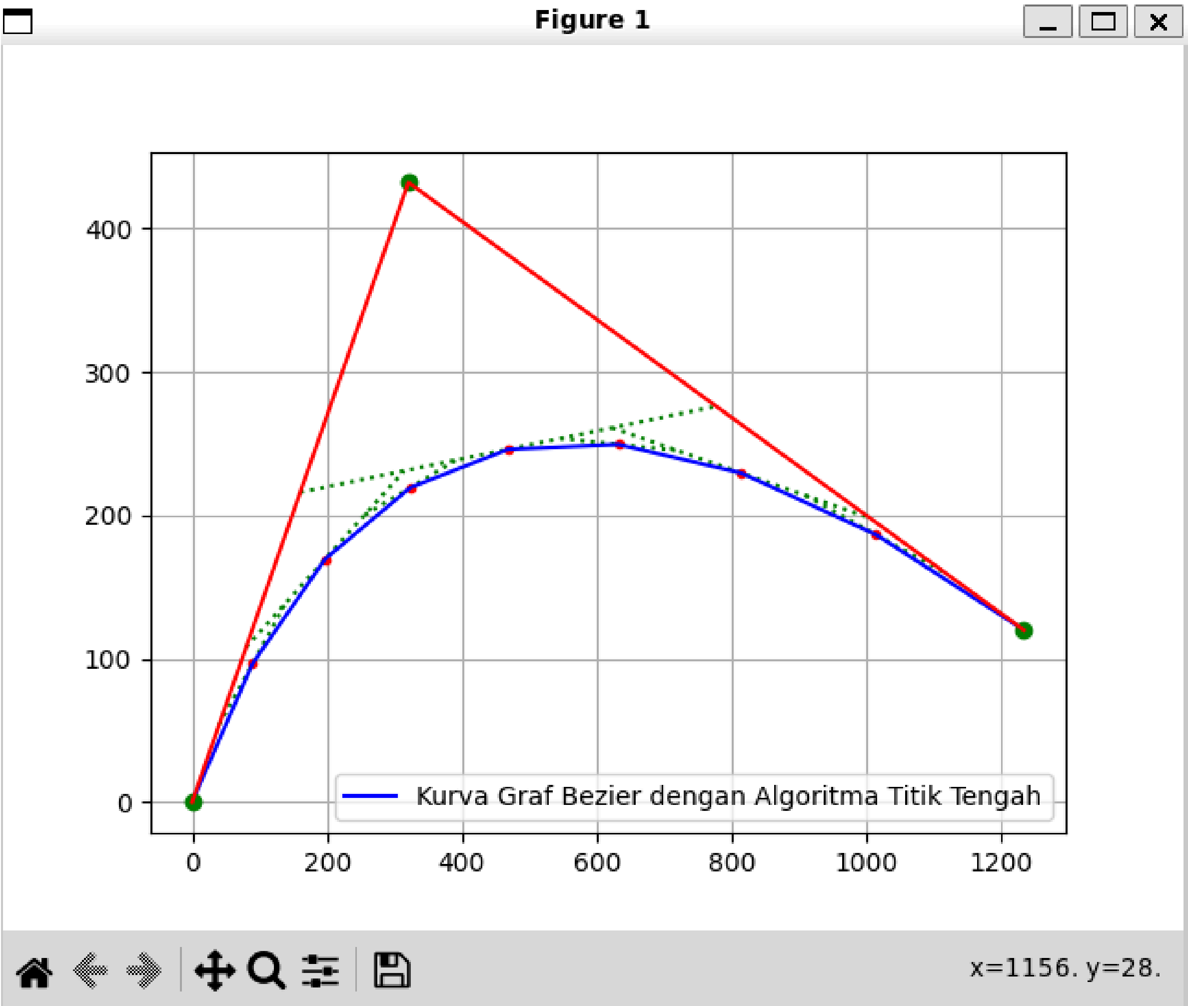


6. Test Case 6

Input:



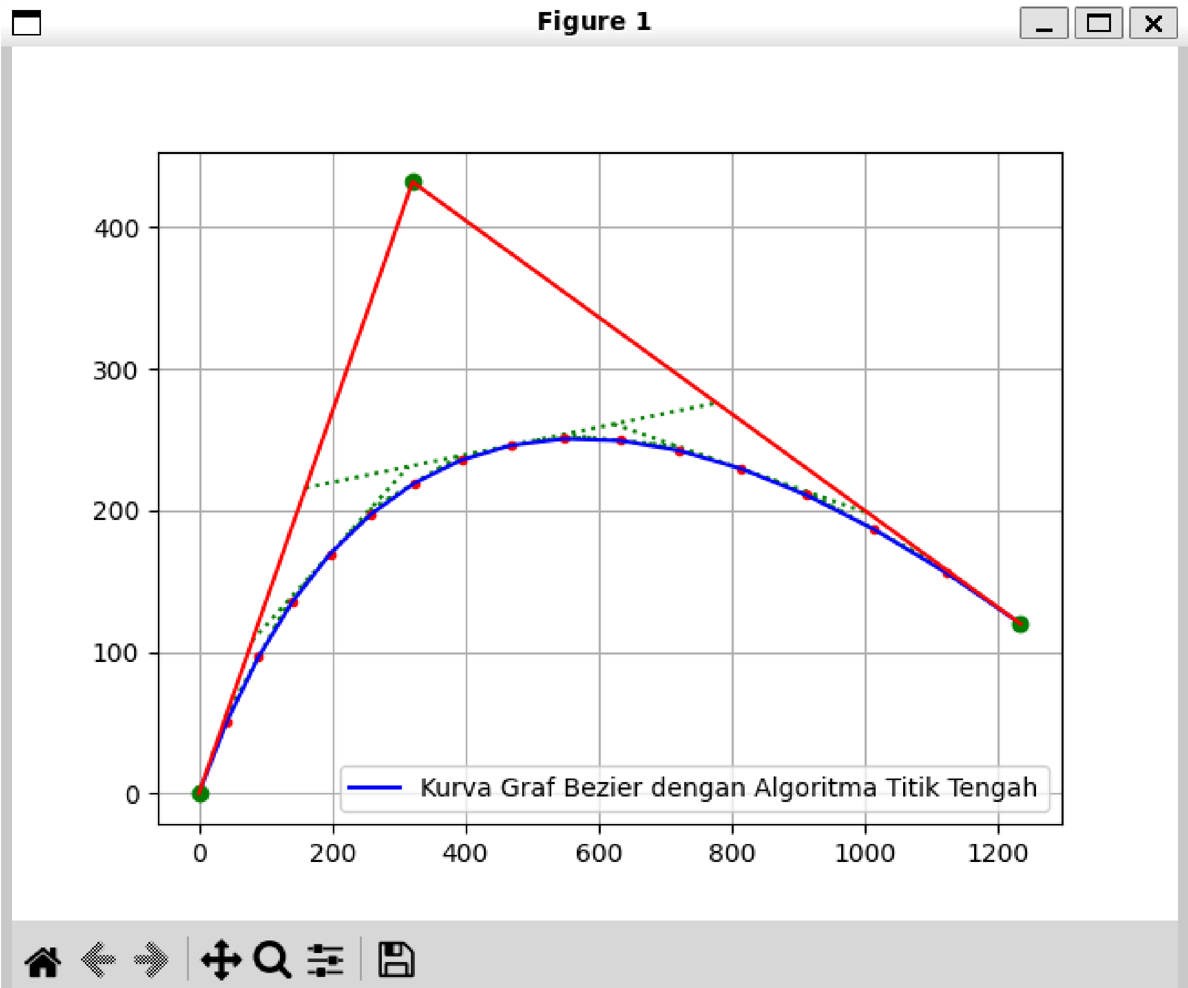
Output:



7. Test Case 7

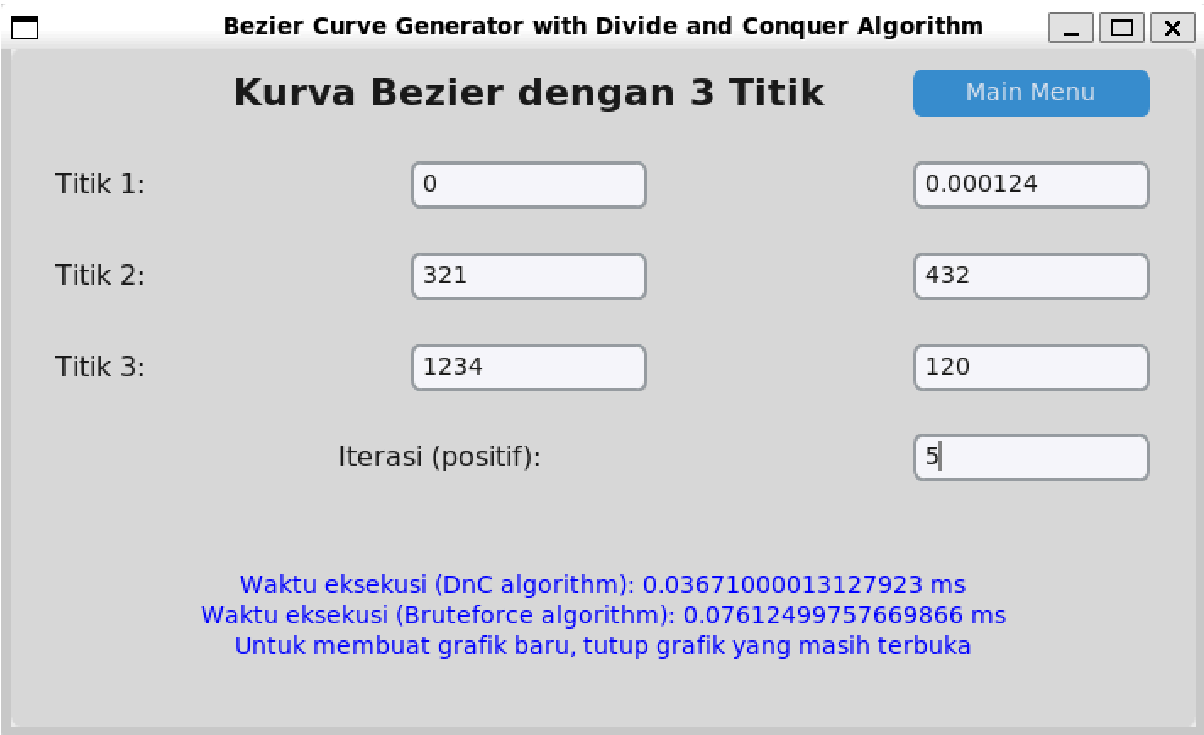
Input:

Output:

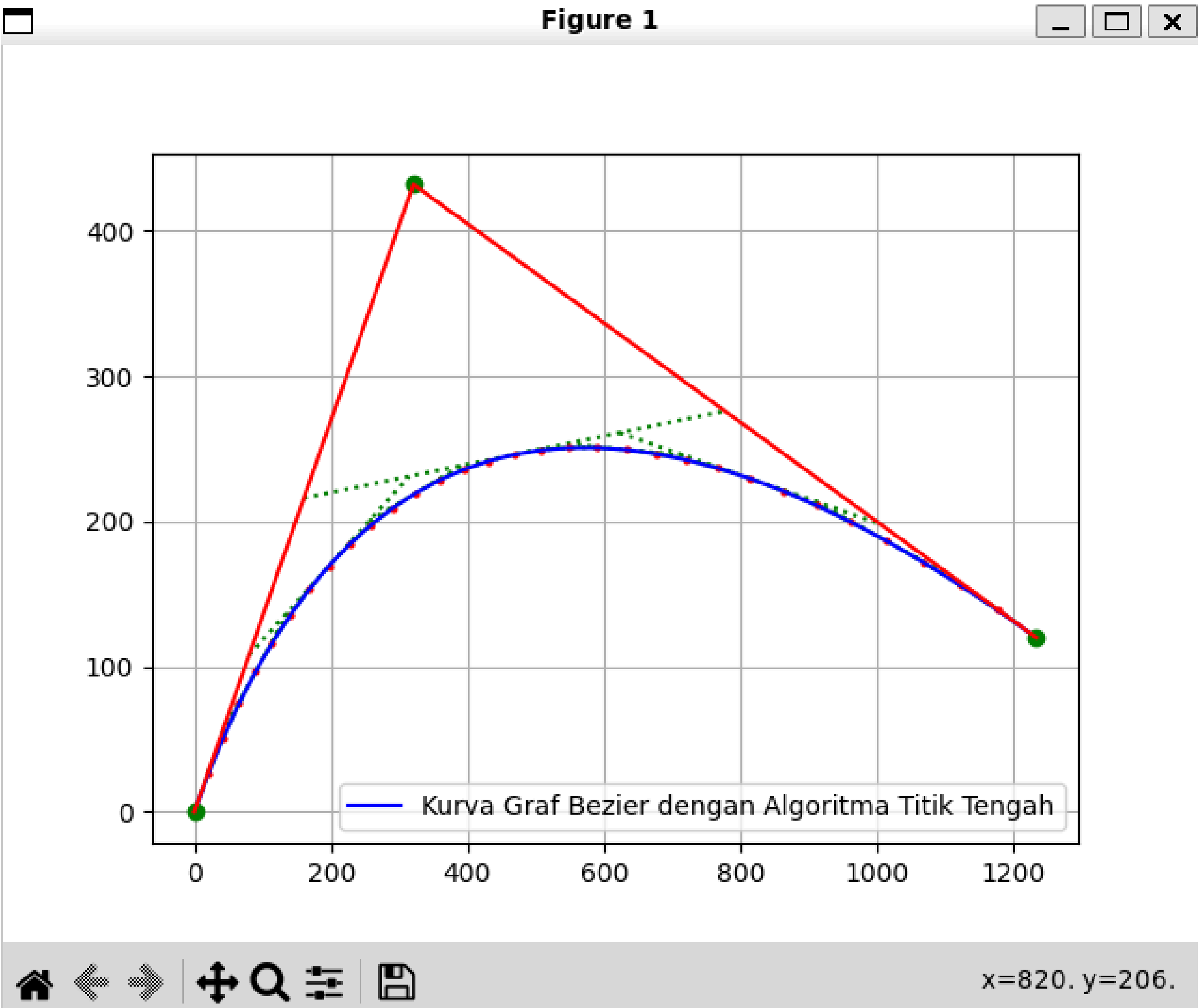


8. Test Case 8

Input:

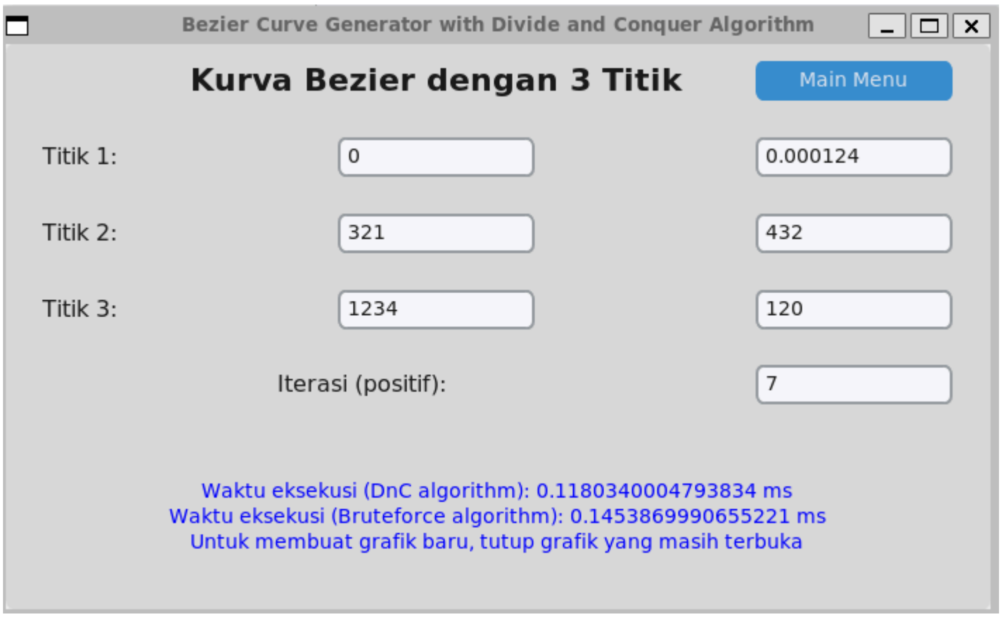


Output:

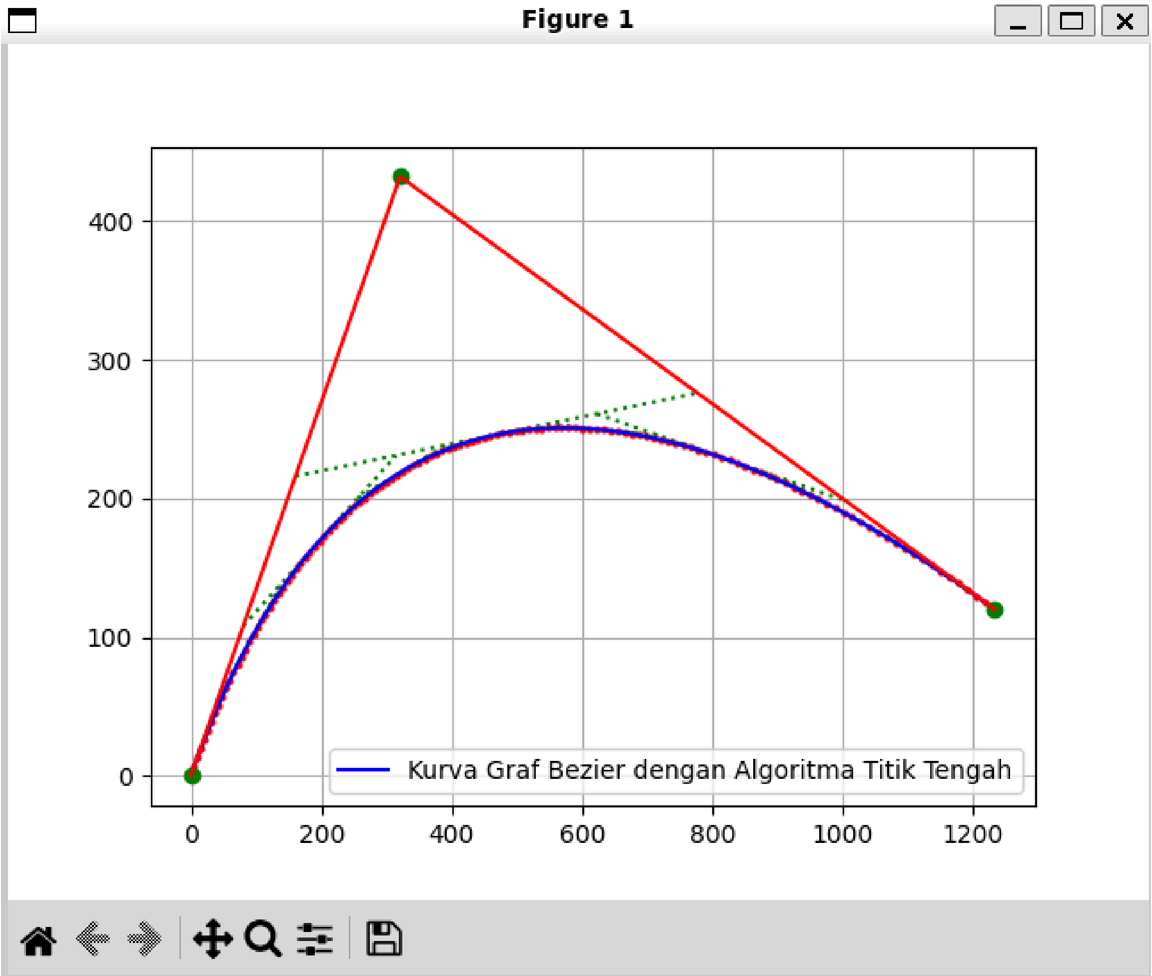


9. Test Case 9

Input:

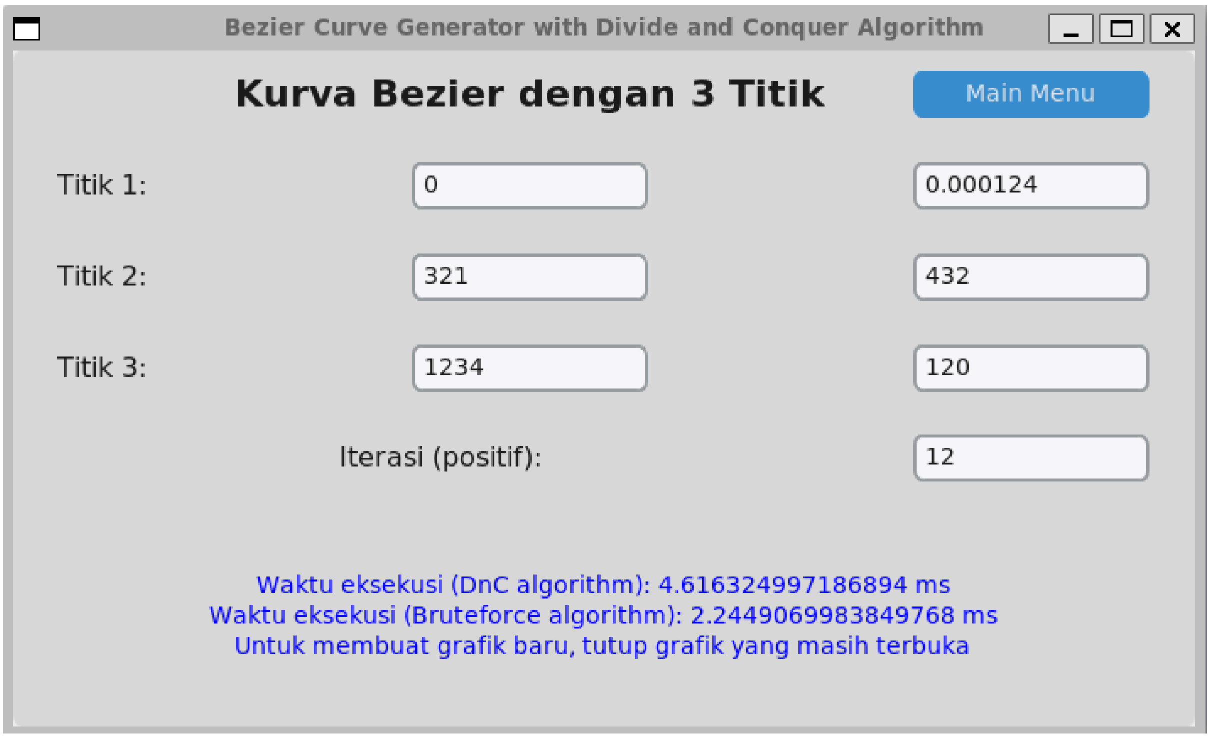


Output:



10. Test Case 10

Input:



Output:

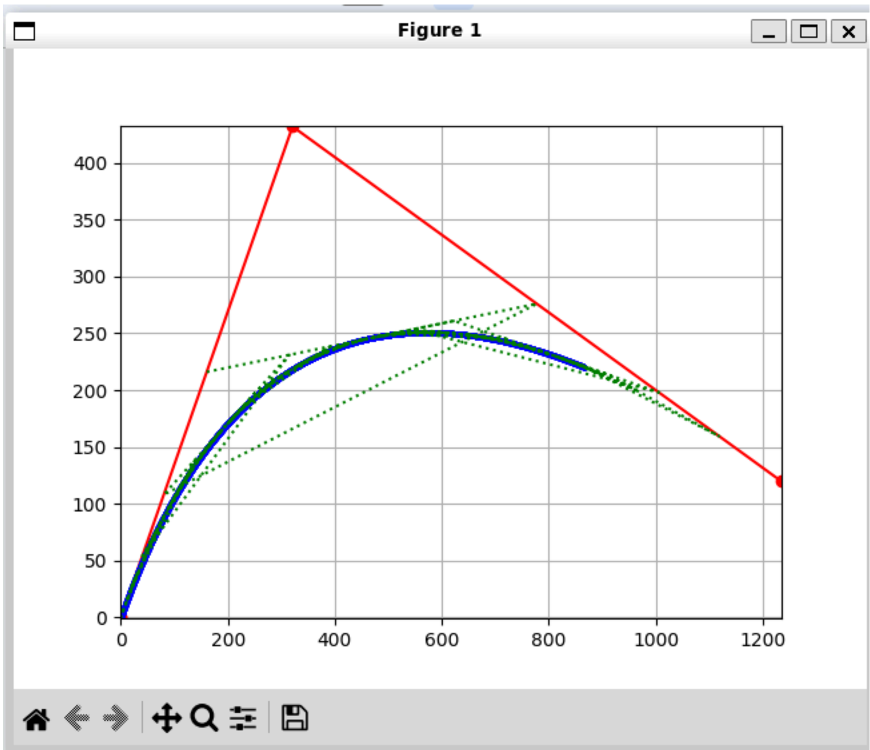


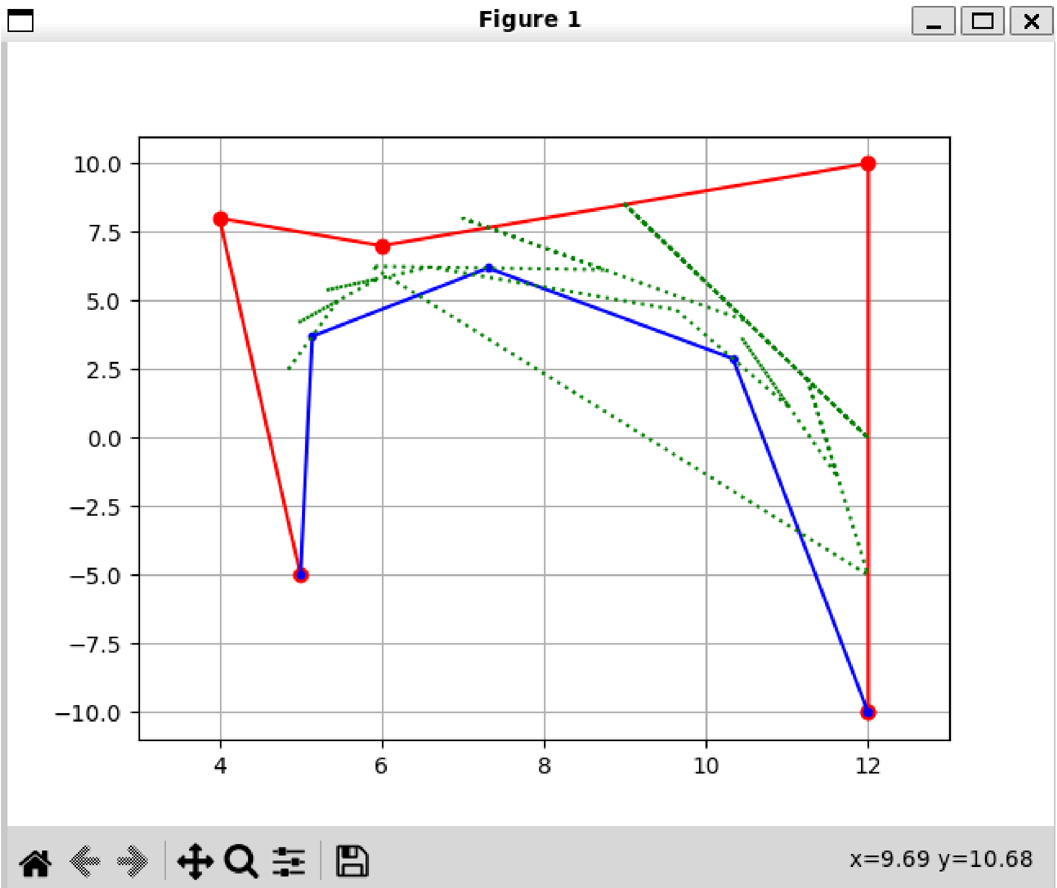
Figure 1
400
350
300
250
200
150
100
50
0
0
200
400
600
800
1000
1200

C. Test Case n-Titik

1. Test Case 1

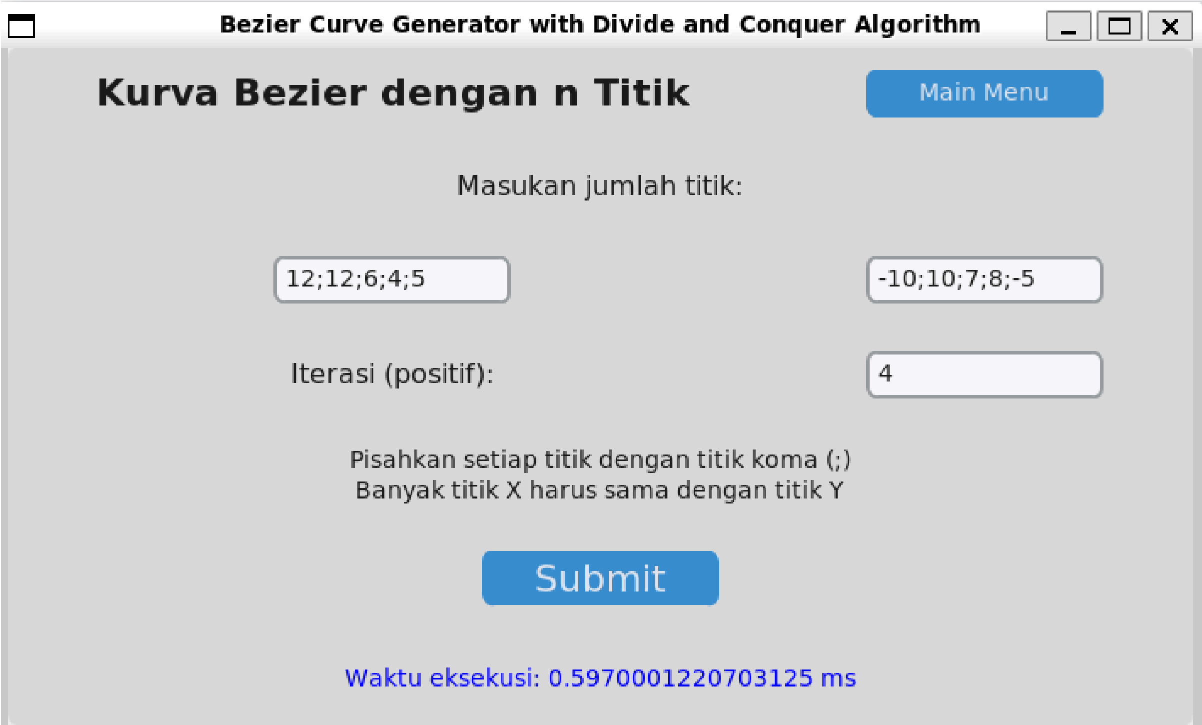
Input:

Output:

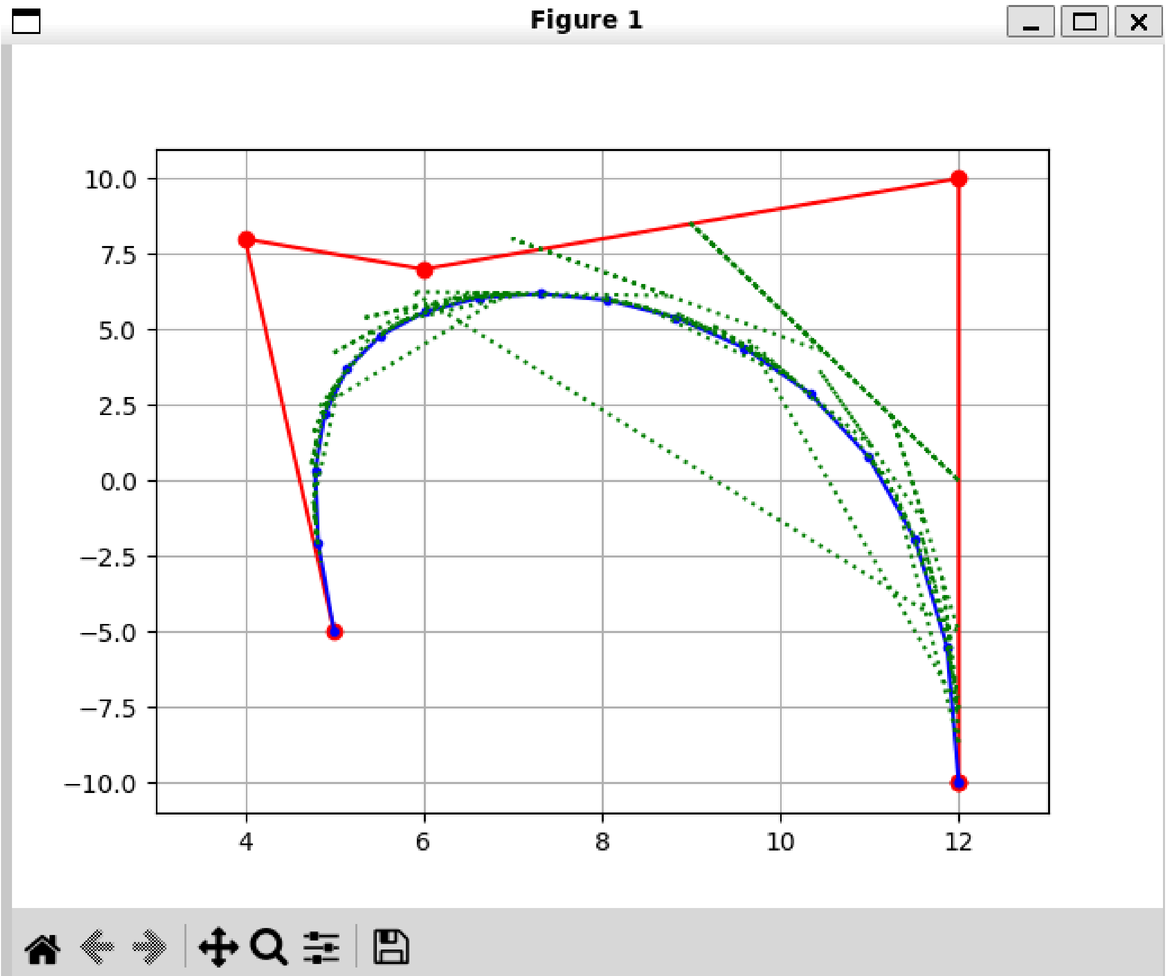


2. Test Case 2

Input:

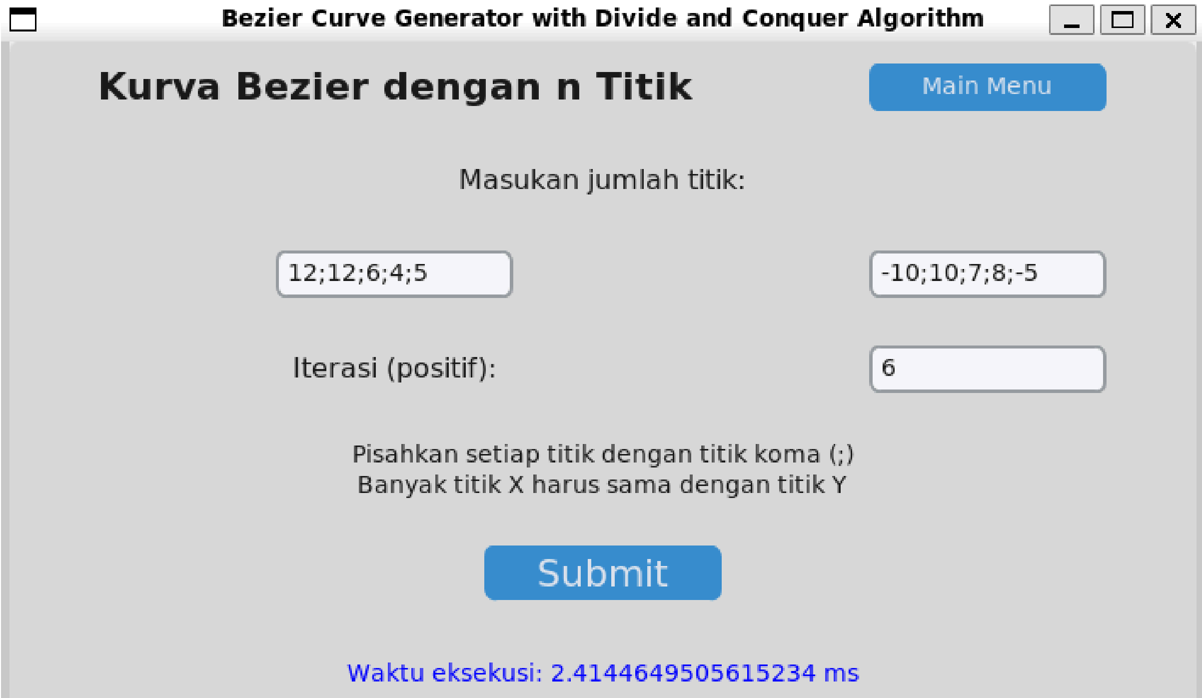


Output:

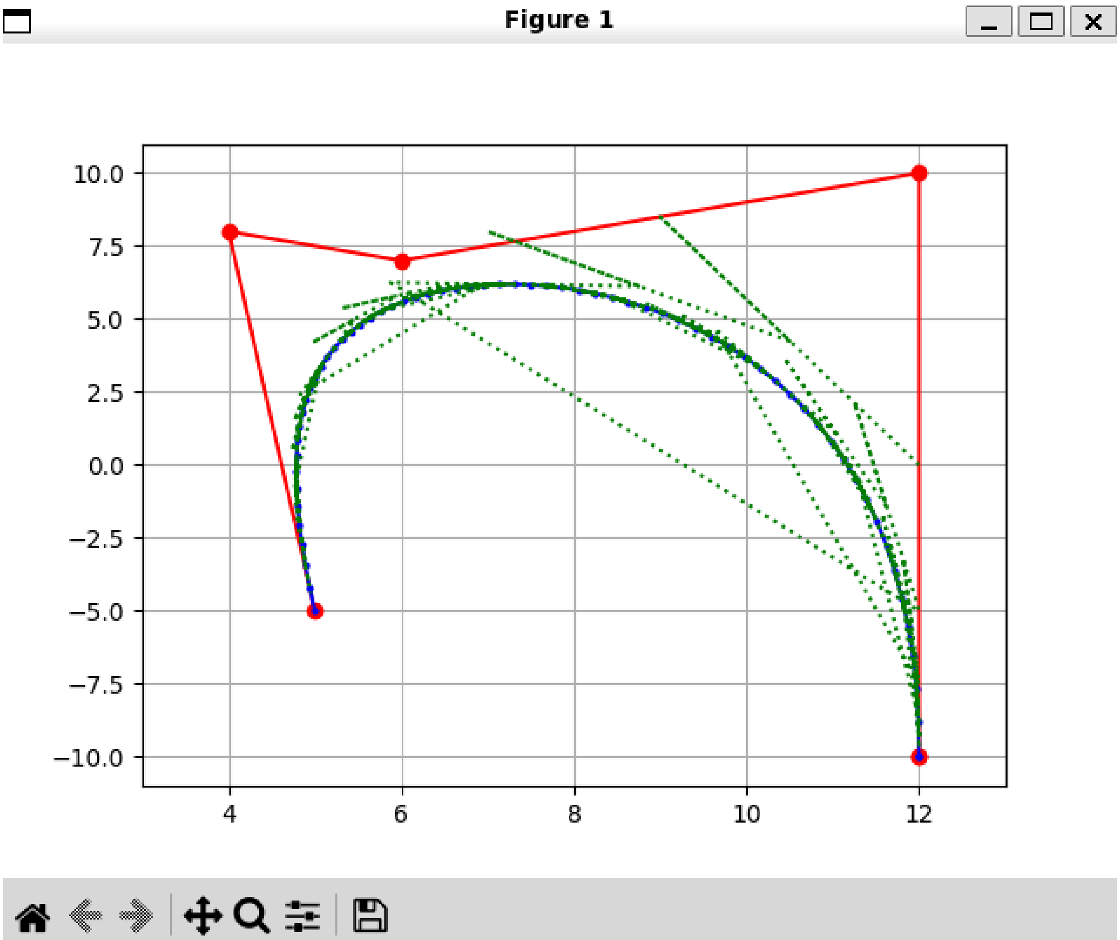


3. Test Case 3

Input:

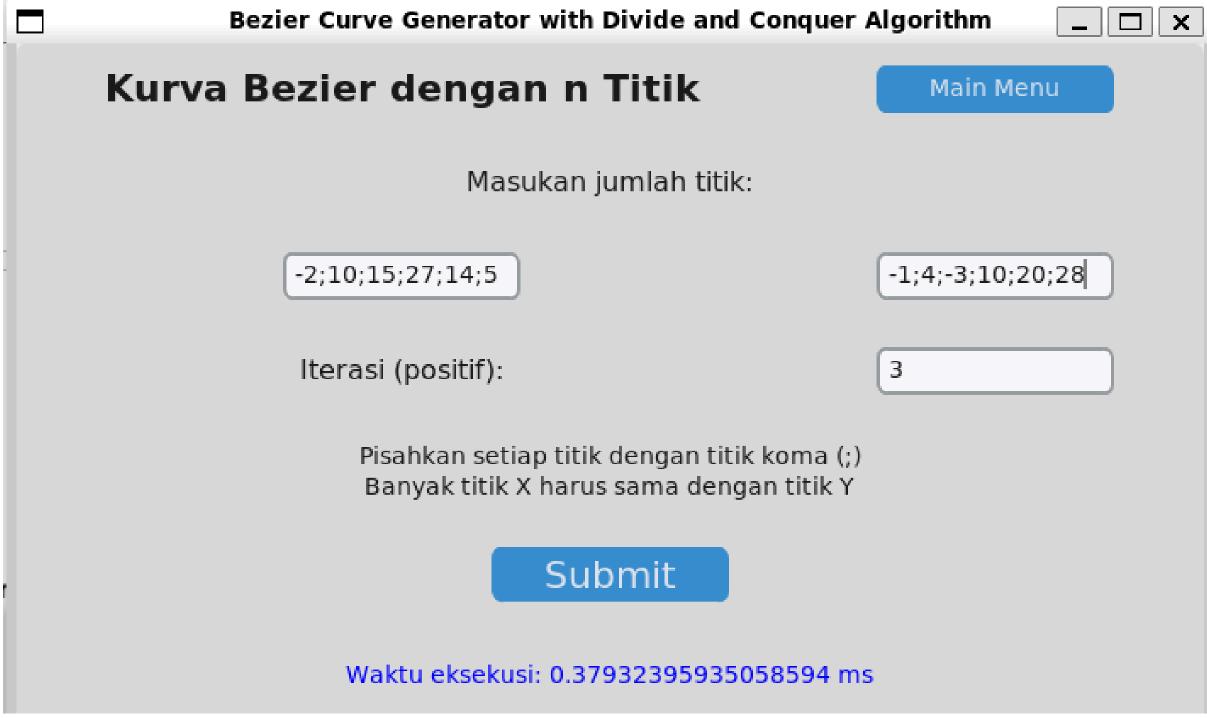


Output:

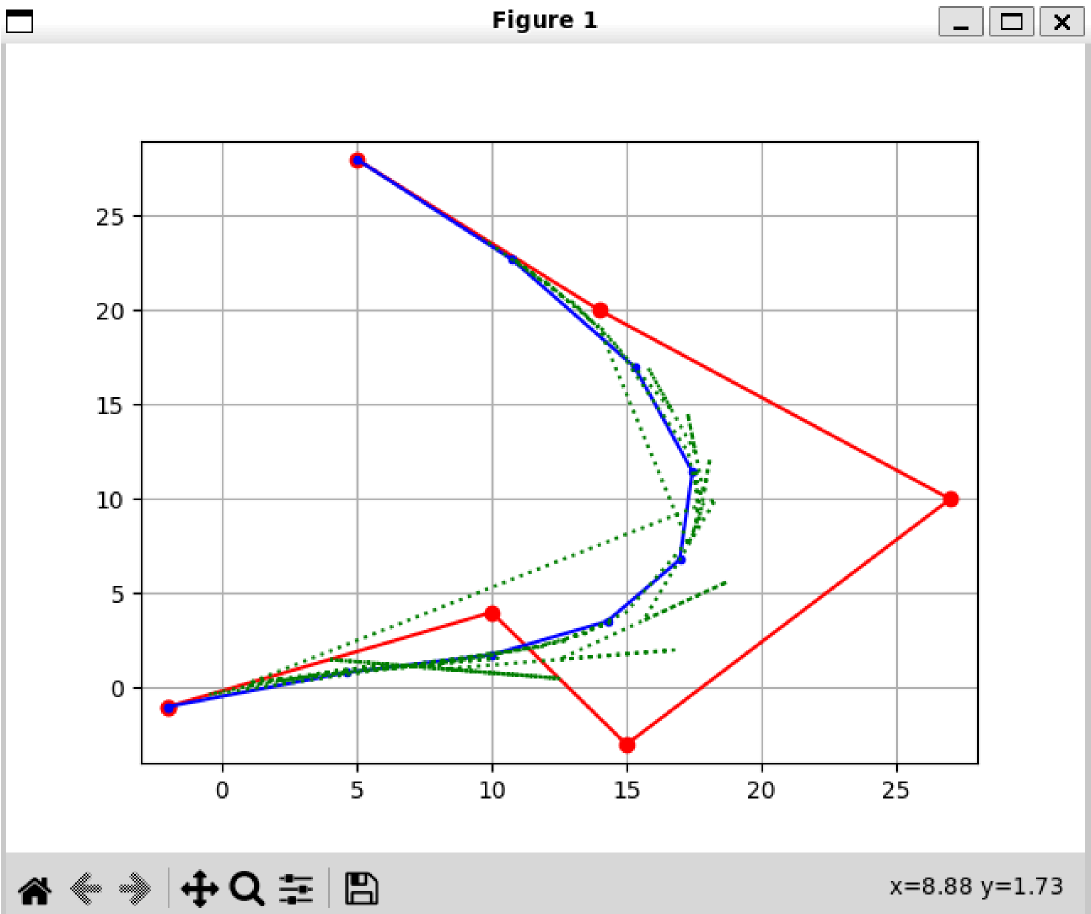


4. Test Case 4

Input:

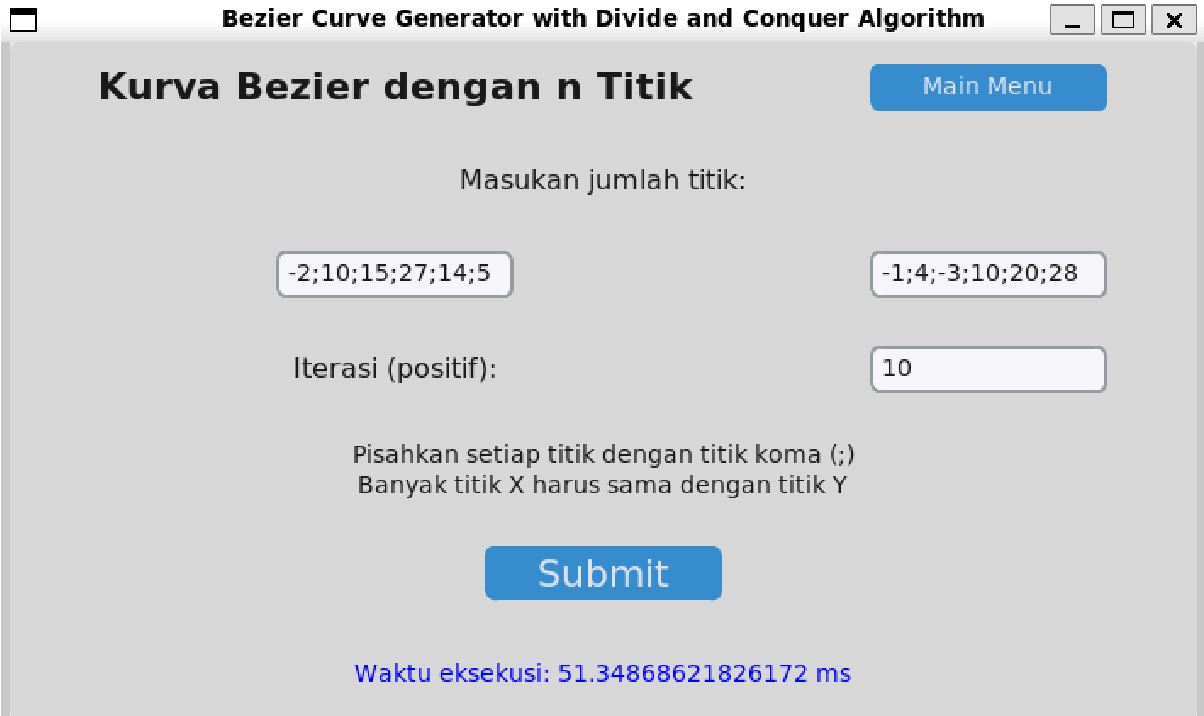


Output:

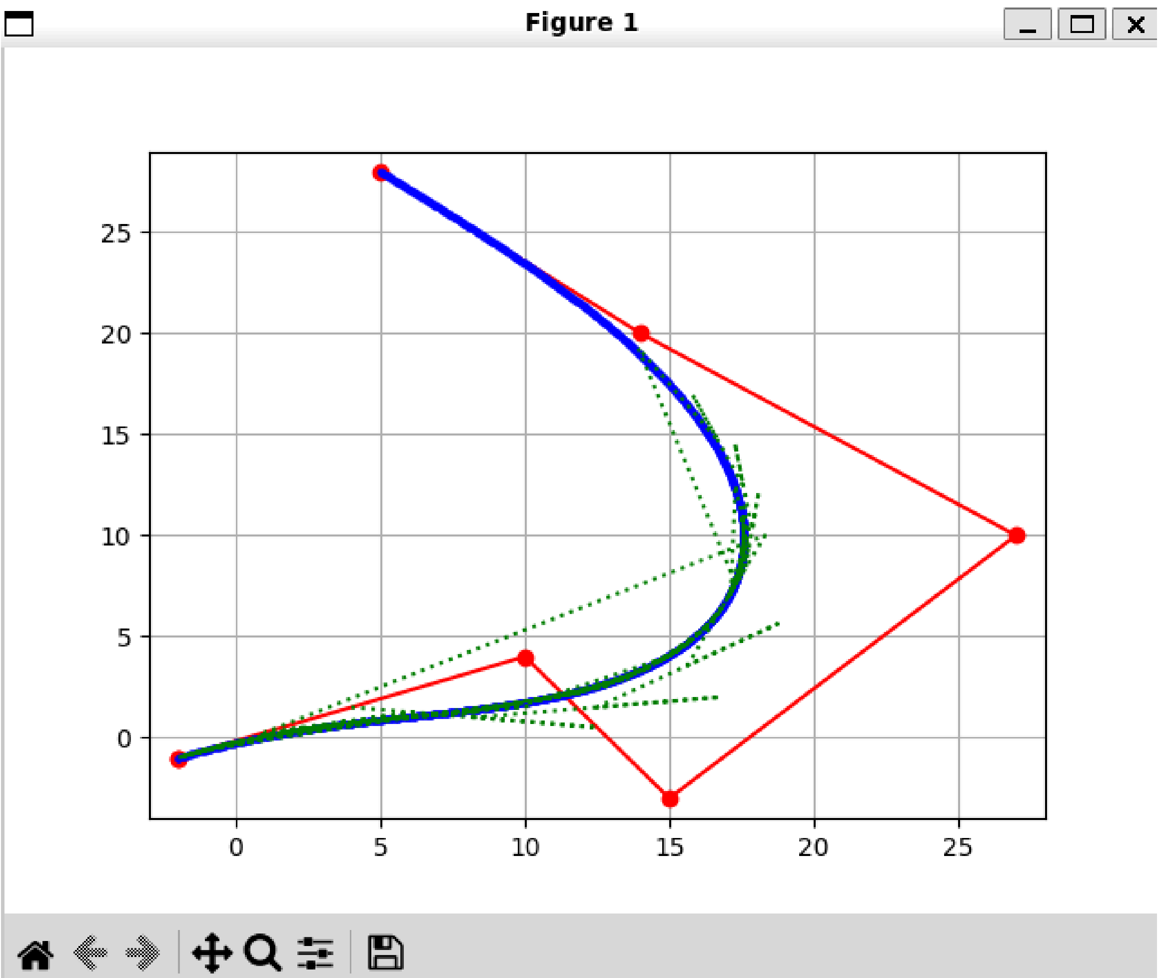


5. **Test Case 5**

Input:

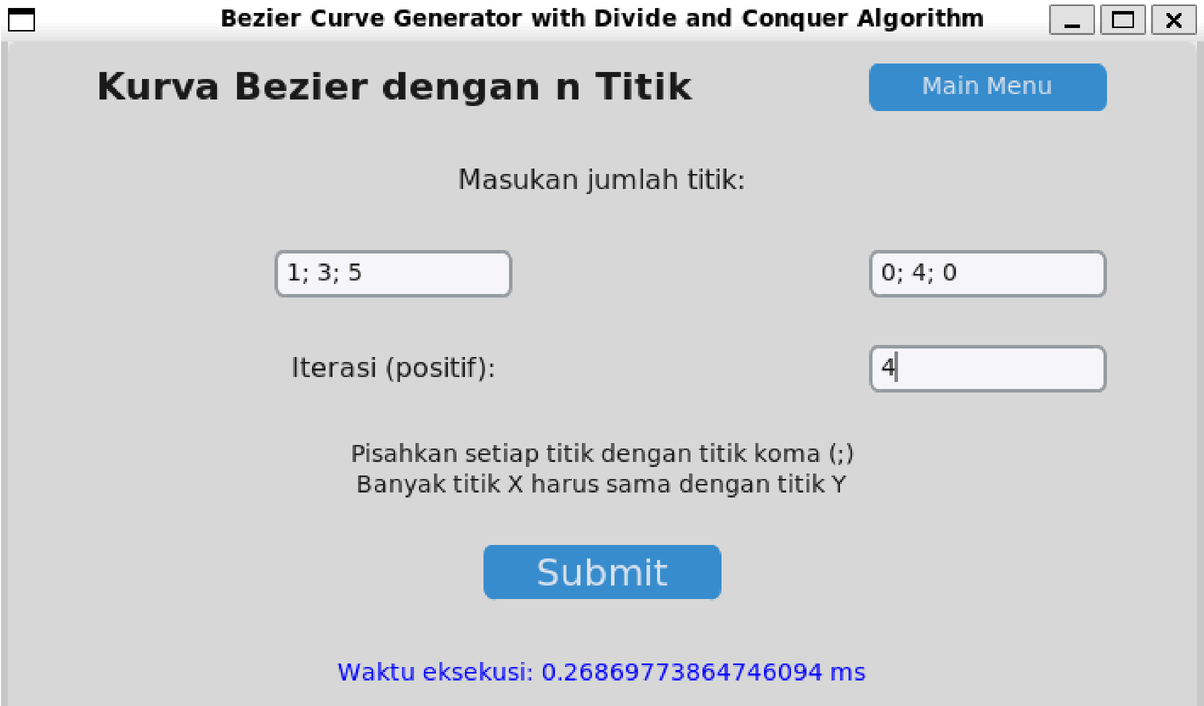


Output:

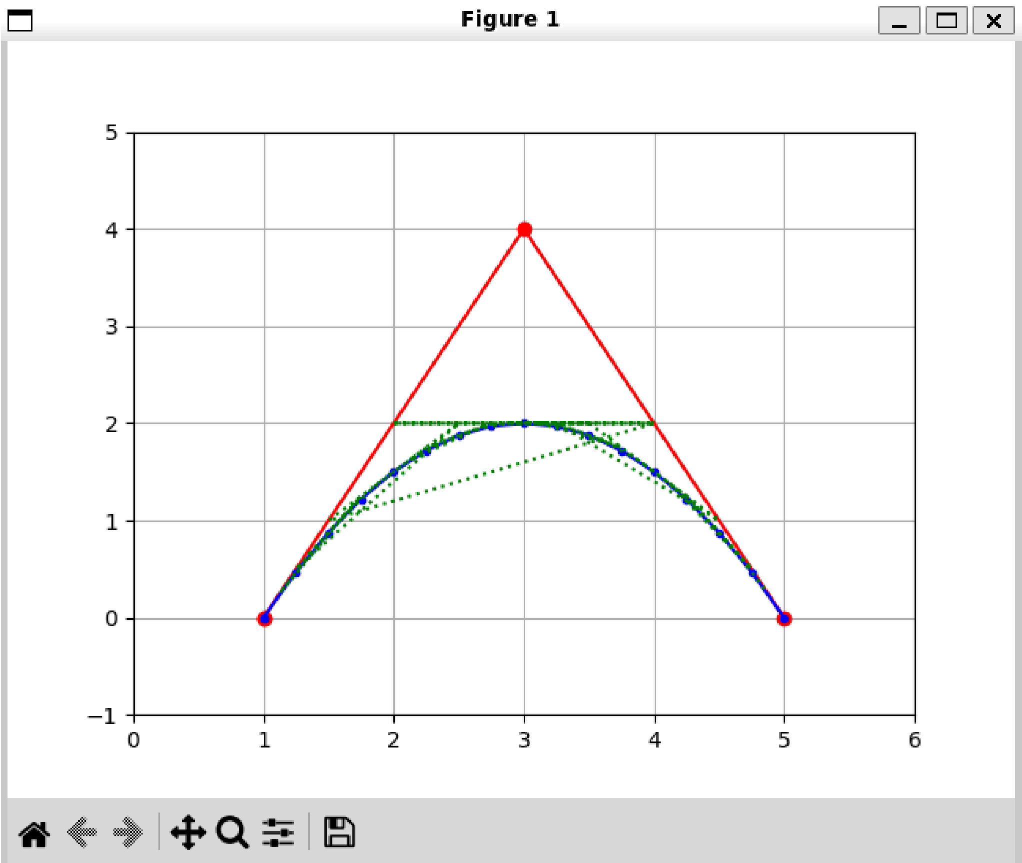


6. Test Case 6

Input:

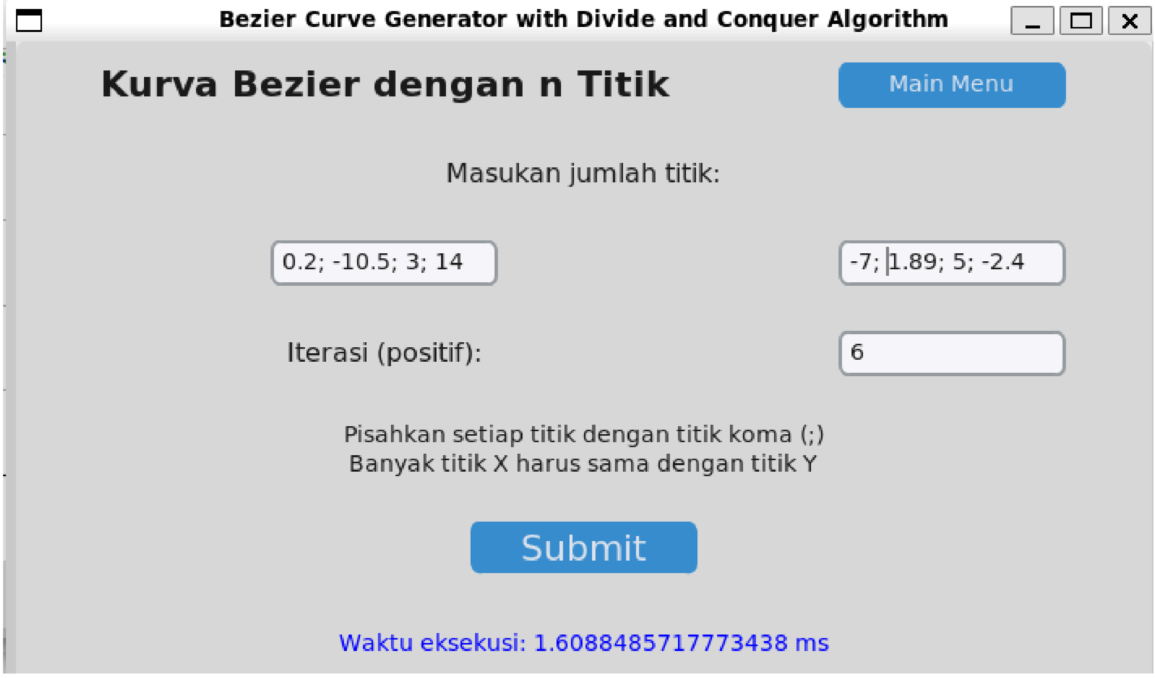


Output:

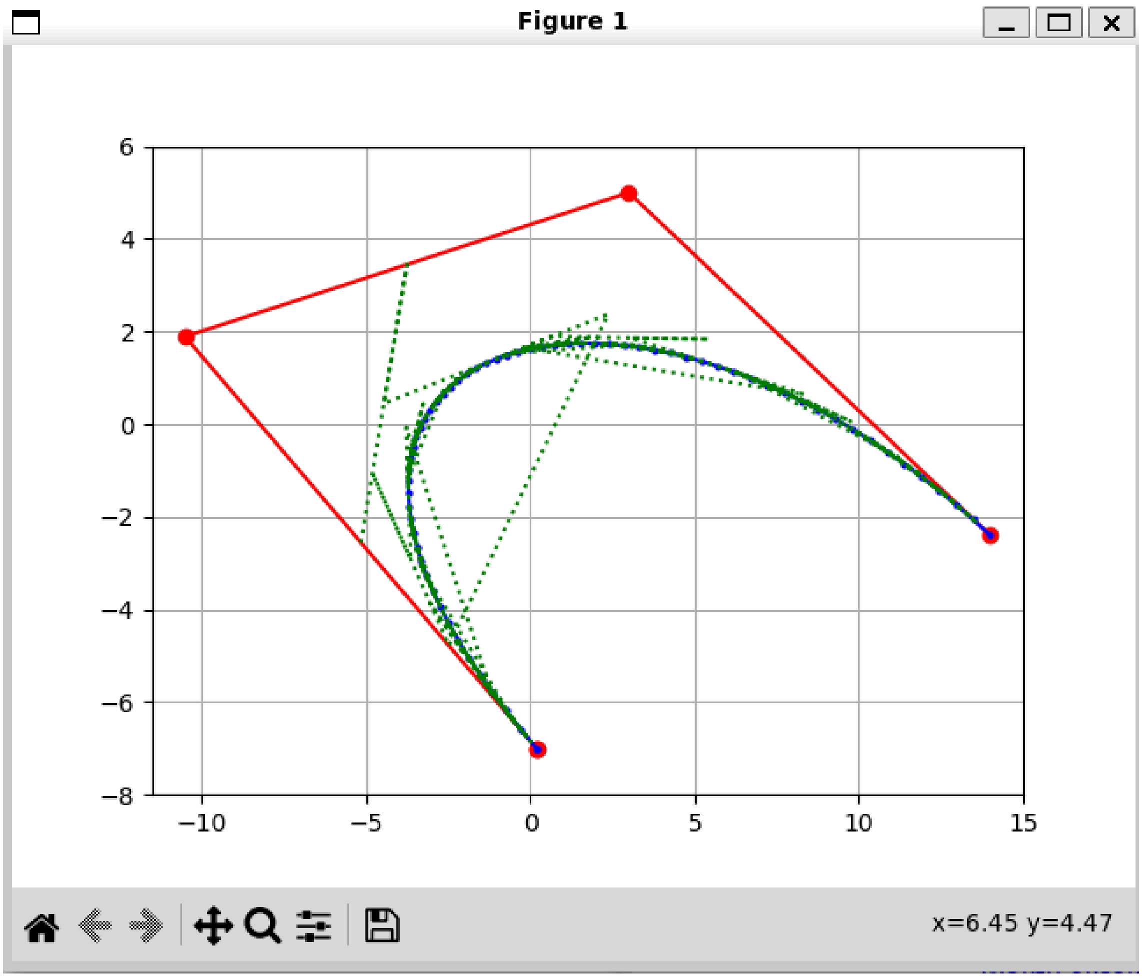


7. Test Case 7

Input:



Output:



D. Test Case CLI

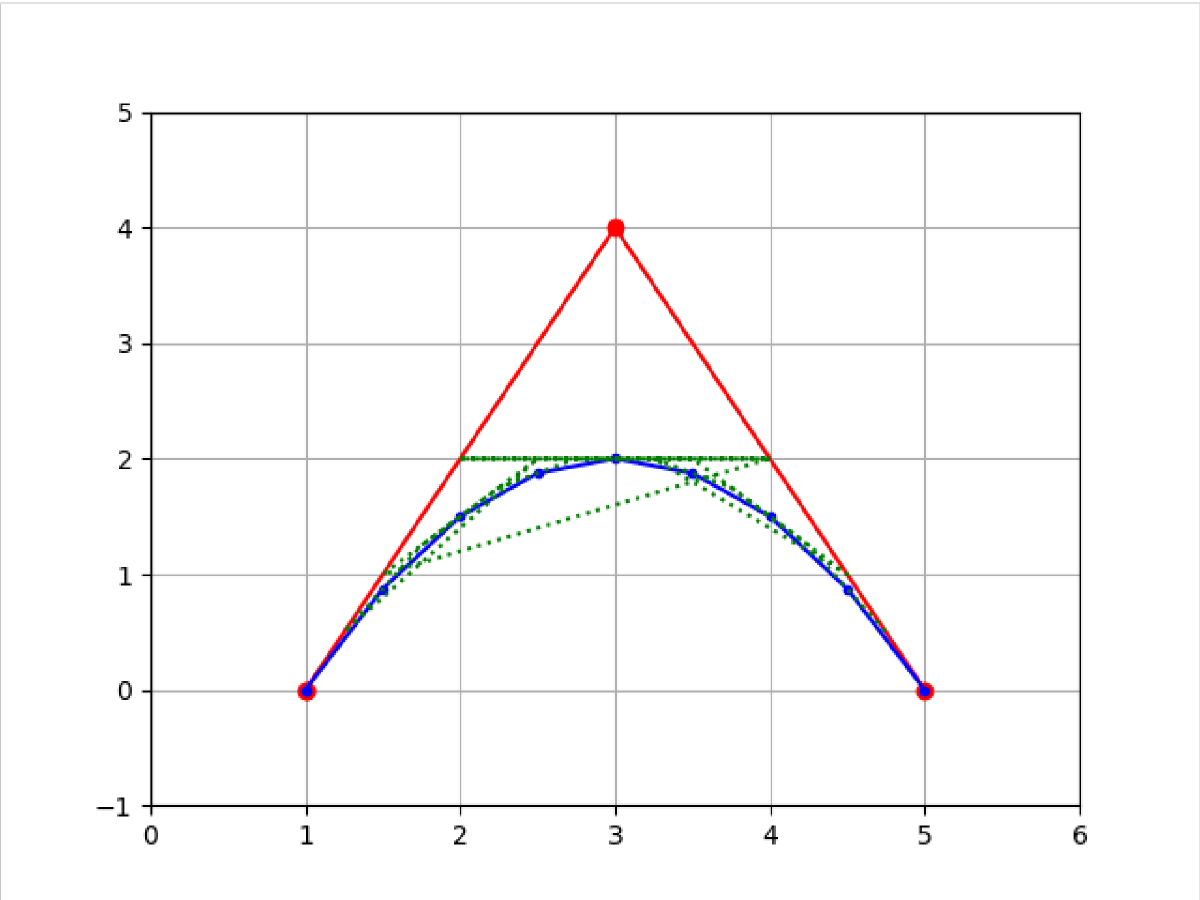
1. Test Case 1

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
```



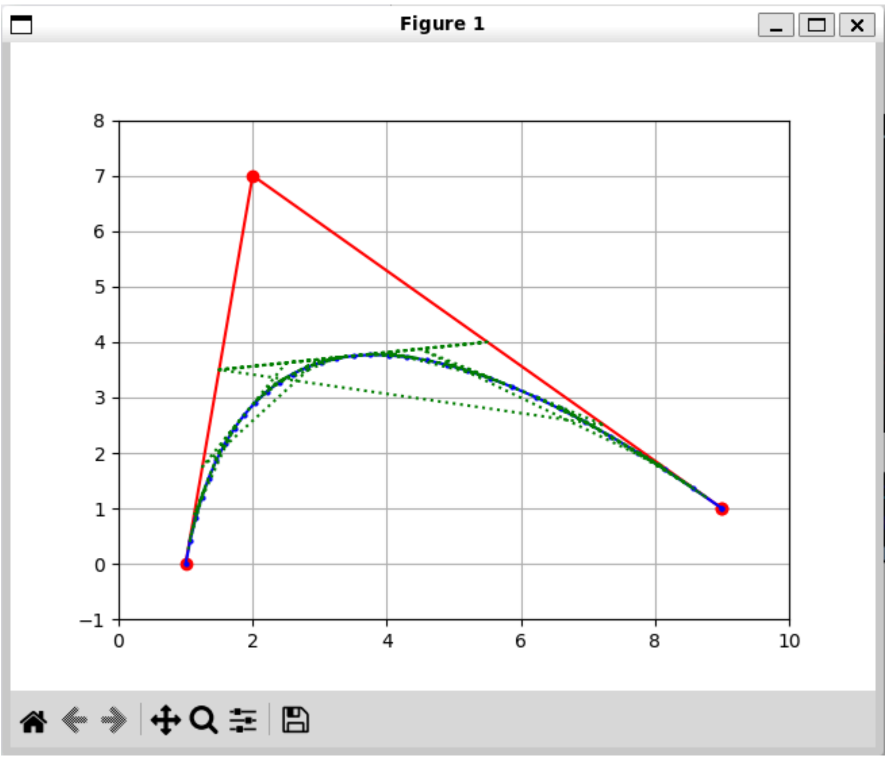
2. Test Case 2

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:

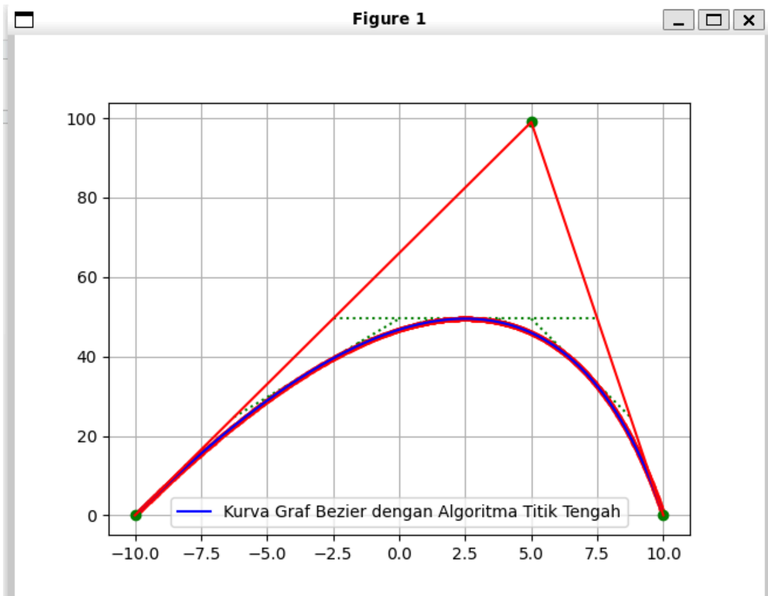
```
Masukkan pilihan: 2
1. Tiga Titik
2. N Titik
Masukkan pilihan: 1
Masukkan koordinat titik 1 (x y): -10 0
Masukkan koordinat titik 2 (x y): 5 99
Masukkan koordinat titik 3 (x y): 10 0
Masukkan iterasi: 10
```


```
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
```

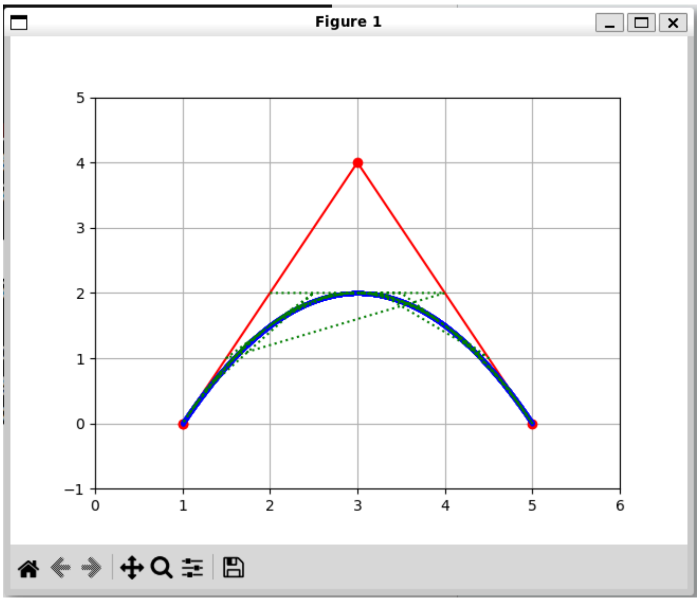
5. Test Case 5

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
```



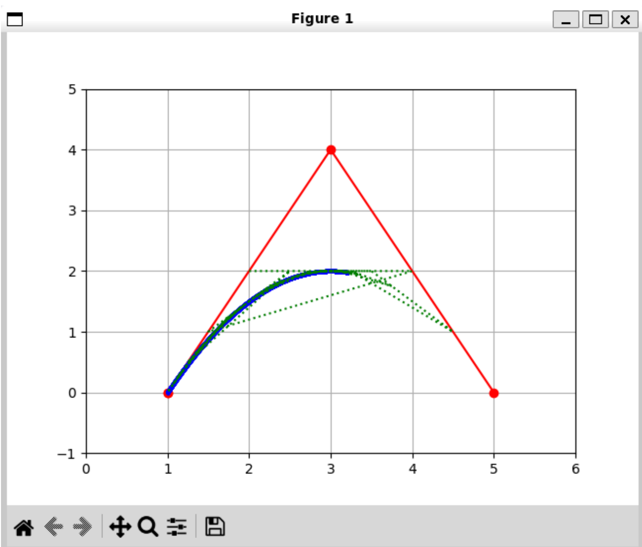
6. Test Case 6

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:  3.2383660000005044
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:

$$T(n) = 13.2^n - 11$$

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*.

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

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

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

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

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

Sehingga Big O notation-nya adalah:

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

$n\ adalah\ jumlah\ iterasi$

$p\ adalah\ jumlah\ titik\ masukan$

$p_0\ adalah\ jumlah\ titik\ kontrol\ yang\ diinginkan$

$g(p)\ adalah\ banyaknya\ operasi\ perhitungan\ titik\ tengah$

$f(n)\ adalah\ banyaknya\ operasi\ konkatenasi\ untuk\ solusi\ akhir$

$T(n)\ adalah\ kompleksitas\ waktu\ secara\ keseluruhan$

Lampiran

Link Github:
https://github.com/FedrianzD/Tucil2_13522071_13522090
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)-Bagian2.pdf

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.	✓	