

# **Laporan Tugas Kecil 2**

IF2211 Strategi Algoritma

## **Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer**



Disusun oleh :

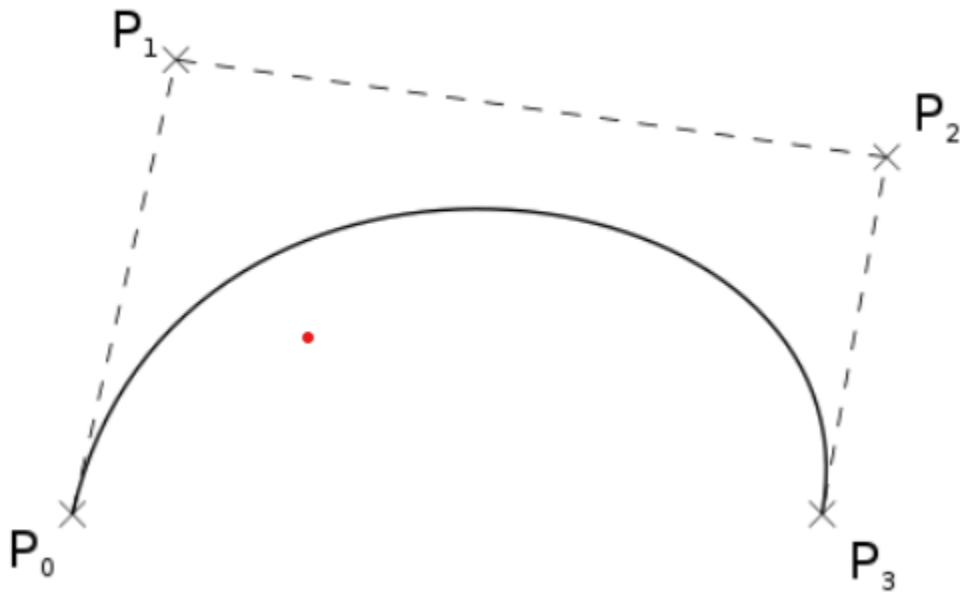
Muhammad Yusuf Rafi  
13522009  
K-01

**PROGRAM STUDI TEKNIK INFORMATIKA SEKOLAH TEKNIK  
ELEKTRO DAN INFORMATIKA INSTITUT TEKNOLOGI  
BANDUNG  
2023**

## BAB I DESKRIPSI MASALAH

### 1.1 Kurva Berzier

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.



**Gambar 1. Kurva Bézier Kubik**

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. Untuk kurva Bézier linier, terbentuk garis lurus antara dua titik kontrol. Namun, dengan adanya titik kontrol tambahan, seperti  $P_2$  di antara  $P_0$  dan  $P_1$  pada Gambar 1, kurva Bézier kuadratik terbentuk. Proses ini melibatkan interpolasi titik-titik kontrol untuk menciptakan kurva yang halus.

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

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

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

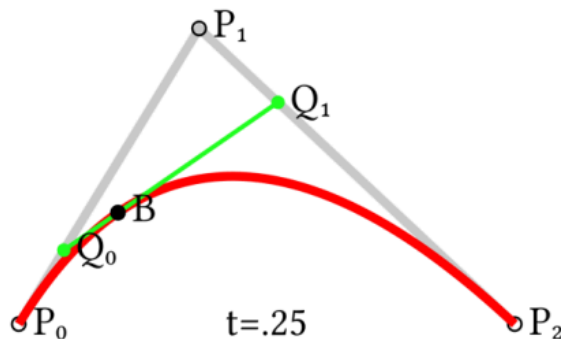
$$R_0 = B(t) = (1 - t)Q_0 + tQ_1, \quad t \in [0, 1]$$

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

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

**Gambar 2. Persamaan Kurva Bézier Kuadratik**

Berikut adalah ilustrasi dari kasus diatas.



**Gambar 3. Pembentukan Kurva Bézier Kuadratik.**

Tentu persamaan yang terbentuk akan sangat panjang dan 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*.

## **1.2 Spesifikasi Program**

Program sederhana dalam bahasa Python yang dapat membentuk sebuah kurva Bézier kuadratik dengan menggunakan algoritma titik tengah berbasis divide and conquer. Selain implementasi dalam algoritma divide and conquer, program juga diminta untuk diimplementasikan dalam algoritma brute force. Solusi ini ditujukan sebagai pembandingan dengan solusi sebelumnya.

### **1.2.1 *Input***

Program menerima masukan:

1. Tiga buah pasangan titik. Sebagai catatan, titik yang paling awal dimasukkan akan menjadi titik awal kurva, begitu juga dengan titik yang paling akhir.
2. Jumlah iterasi yang ingin dilakukan.

### **1.2.2 *Output***

Program menampilkan:

1. Hasil kurva Bézier yang terbentuk pada iterasi terkait.
2. Waktu eksekusi program pembentukan kurva.

### **1.2.3 Bonus**

Berikut adalah spesifikasi bonus dari program yang dibangun.

1. Visualisasi seluruh proses pembentukan kurva pada tiap iterasi.

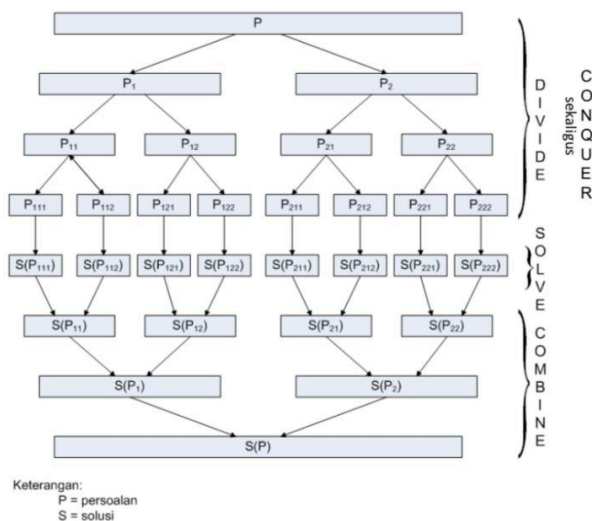
## BAB II LANDASAN TEORI

### 2.1 Algoritma *Divide and Conquer*

Algoritma *divide and conquer* adalah salah satu logika dalam mendesain algoritma dan terbagi menjadi tiga langkah utama, yaitu sebagai berikut.

1. Membagi (*divide*) persoalan menjadi beberapa sub-persoalan yang memiliki kemiripan dengan persoalan semula dengan ukuran lebih kecil.
2. Menyelesaikan (*conquer* atau *solve*) masing-masing sub-persoalan secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar
3. Menggabungkan solusi masing-masing sub-persoalan sehingga membentuk solusi persoalan semula

Ilustrasi algoritma tersebut melalui gambar di bawah ini.



**Gambar 3. Pembentukan Kurva Bézier Kuadratik.**

Skema Umum Algoritma Divide and Conquer:

```

procedure DIVIDEandCONQUER(input  $P$  : problem,  $n$  : integer)
{ Menyelesaikan persoalan  $P$  dengan algoritma divide and conquer
  Masukan: masukan persoalan  $P$  berukuran  $n$ 
  Luaran: solusi dari persoalan semula }
Deklarasi
   $r$  : integer

Algoritma
if  $n \leq n_0$  then {ukuran persoalan  $P$  sudah cukup kecil}
  SOLVE persoalan  $P$  yang berukuran  $n$  ini
else
  DIVIDE menjadi  $r$  upa-persoalan,  $P_1, P_2, \dots, P_r$ , yang masing-masing berukuran  $n_1, n_2, \dots, n_r$ 
  for masing-masing  $P_1, P_2, \dots, P_r$ , do
    DIVIDEandCONQUER( $P_i, n_i$ )
  endfor
  COMBINE solusi dari  $P_1, P_2, \dots, P_r$  menjadi solusi persoalan semula
endif

```

**Gambar 4. Pseudocode Divide and Conquer.**

## 2.2 Algoritma Titik Tengah Kurva Berzier

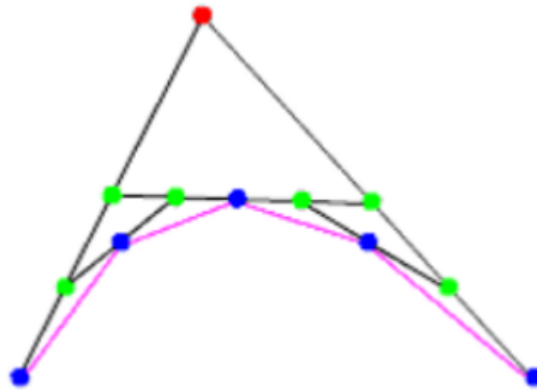
Pada kurva Berzier kuadratik, terdapat tiga buah titik,  $P_0$ ,  $P_1$ , dan  $P_2$ , dengan titik  $P_1$  menjadi titik kontrol antara, maka:

1. 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  $P_2$ .
2. Hubungkan  $Q_0$  dan  $Q_1$  sehingga terbentuk sebuah garis baru.
3. Buatlah sebuah titik baru  $R_0$  yang berada di tengah  $Q_0$  dan  $Q_1$ .
4. 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:

1. 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  $P_2$ .
2. Hubungkan  $S_0$  dengan  $S_1$  dan  $S_2$  dengan  $S_3$  sehingga terbentuk garis baru.
3. 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  $S_3$ .
4. 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. Dapat diterka dengan jelas bahwa semakin banyak iterasi yang dilakukan, maka akan membentuk sebuah kurva yang tidak lain adalah kurva Bézier.



**Gambar 5. Hasil pembentukan Kurva Bézier Kuadratik dengan divide and conquer setelah iterasi ke-2.**

## **BAB III**

### **IMPLEMENTASI**

#### **3.1 Implementasi Algoritma Divide and Conquer Pada Kurva Berzier**

Dengan pendekatan algoritma ini, penulis membagi penentuan titik-titik dalam pembentukan kurva menjadi 2 bagian, yakni bagian kiri dan kanan kurva. Lalu, hasil dari kedua bagian tersebut digabungkan menjadi solusi umum. Selain itu, algoritma ini dilakukan secara rekursif. Berikut langkah-langkah yang dilakukan:

1. Untuk Kasus iterasi = 0(SOLVE),
  - Kasus ini dijadikan sebagai basis rekursif.
  - Untuk input iterasi harus lebih besar dari 0.
2. Untuk kasus iterasi  $\geq 1$  (SOLVE),
  - Algoritma melakukan perhitungan titik tengah  $Q0$  dan  $Q1$ , serta titik tengah  $R0$  di antara  $Q0$  dan  $Q1$ .
  - DIVIDE:
    - A. Pada setiap iterasi selanjutnya, larik titik kontrol  $P0$ ,  $P1$ , dan  $P2$  dibagi menjadi dua bagian yang sama, yaitu  $P0 - Q0 - R0$ (bagian kiri) dan  $R0 - Q1 - P2$ (bagian kanan).
  - CONQUER:
    - A. Setiap segmen kurva Bézier kuadratik yang dihasilkan diperoleh dengan cara melakukan rekursi pada kedua segmen tersebut.
    - B. Rekursi akan dilakukan hingga mencapai basis rekursif, yakni iterasi = 0.
  - COMBINE:
    - A. Hasil dari kedua segmen digabungkan menjadi solusi umum.

### 3.2 Source Code Algoritma Divide and Conquer

Kode ditulis dalam bahasa Python. Kode terdapat pada direktori src dengan nama *file* DnC.py.

DnC.py

```
def QuadraticBezierCurve(P0, P1, P2, iterations):
    curve = [] # Inisialisasi larik untuk menyimpan titik-titik kurva

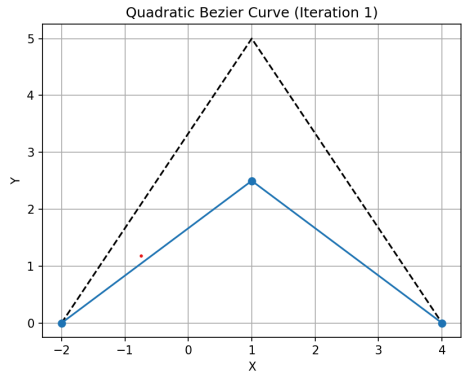
    # Kasus dasar: iterasi = 0, Basis rekursi
    if iterations == 0:
        curve.extend([P0, P2])
    else:
        # Hitung titik kontrol baru
        Midpoint1 = midpoint(P0,P1) # Titik tengah antara P0 dan P1
        Midpoint2 = midpoint(P1, P2) # Titik tengah antara P1 dan P2
        Midpoint_1_and_2 = midpoint(Midpoint1, Midpoint2) # Titik tengah antara Q0 dan Q1

        # Rekursi ke segmen kiri dan kanan(Divide and Conquer)
        # Segmen kiri: dari P0 ke R0
        left_curve = QuadraticBezierCurve(P0, Q0, R0, iterations - 1)
        # Segmen kanan: dari R0 ke P2
        right_curve = QuadraticBezierCurve(R0, Q1, P2, iterations - 1)

        # Gabungkan hasil dari kedua segmen(Combine)
        curve = left_curve[:-1] + right_curve # Hilangkan satu titik R0 yang berlebih

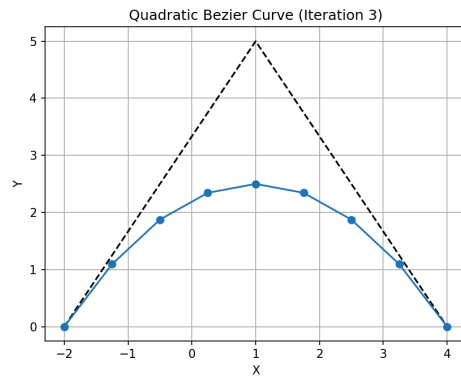
    return curve # Kembalikan larik titik-titik kurva
```

### 3.3 Input dan Output Program Divide and Conquer

Input	Output
Masukkan titik P0 (contoh: 2,3): -2,0 Masukkan titik P1 (contoh: 2,3): 1,5 Masukkan titik P2 (contoh: 2,3): 4,0 Masukkan Jumlah iterasi: 1	 <p>Execution time: 0.00000000 milliseconds</p>

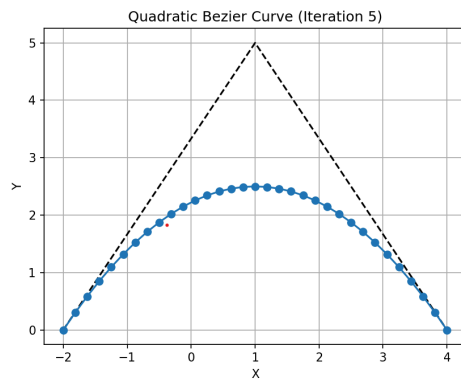


Masukkan titik P0 (contoh: 2,3): -2,0  
Masukkan titik P1 (contoh: 2,3): 1,5  
Masukkan titik P2 (contoh: 2,3): 4,0  
Masukkan Jumlah iterasi: 3



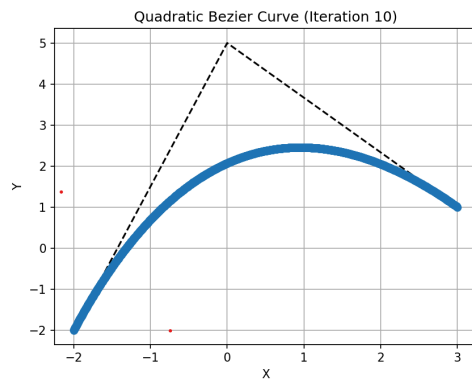
Execution time: 0.00000000 milliseconds

Masukkan titik P0 (contoh: 2,3): -2,0  
Masukkan titik P1 (contoh: 2,3): 1,5  
Masukkan titik P2 (contoh: 2,3): 4,0  
Masukkan Jumlah iterasi: 5



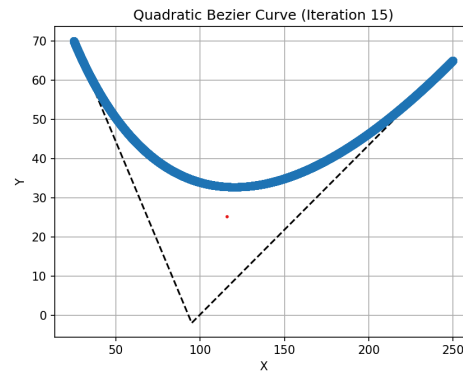
Execution time: 0.00000000 milliseconds

Masukkan titik P0 (contoh: 2,3): -2,-2  
Masukkan titik P1 (contoh: 2,3): 0,5  
Masukkan titik P2 (contoh: 2,3): 3,1  
Masukkan Jumlah iterasi: 10



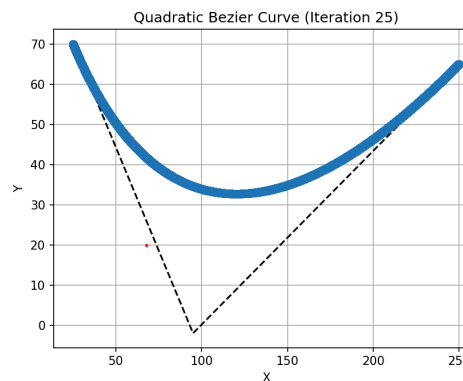
Execution time: 16.84488487  
milliseconds

Masukkan titik P0 (contoh: 2,3): 25,70  
 Masukkan titik P1 (contoh: 2,3): 95,-2  
 Masukkan titik P2 (contoh: 2,3): 250,65  
 Masukkan Jumlah iterasi: 15



Execution time: 61.65313721  
 milliseconds

Masukkan titik P0 (contoh: 2,3): 25,70  
 Masukkan titik P1 (contoh: 2,3): 95,-2  
 Masukkan titik P2 (contoh: 2,3): 250,65  
 Masukkan Jumlah iterasi: 25



Execution time: 134246.37389183  
 milliseconds

### 3.4 Implementasi Algoritma Brute Force Pada Kurva Berzier

Pada algoritma brute force, penulis melakukan operasi lebih lanjut pada variabel iterasi agar hasil titik pemetaan pada algoritma ini sesuai dengan algoritma divide and conquer. Perubahan yang dilakukan, yakni menyesuaikan nilai iterasi pada fungsi total\_dots().

BruteForce.py

```
def total_dots(n):
    if n == 1:
        return 3
    else:
        return 2 * total_dots(n - 1) - 1
```

Setelah menyesuaikan nilai iterasi, baru mulai memasuki langkah-langkah algortimanya sebagai berikut:

1. Melakukan iterasi sebanyak hasil total\_dots. Pada setiap iterasi, menghitung parameter t berdasarkan indeks iterasi dan jumlah total titik secara merata. Parameter t ini digunakan untuk menghitung titik-titik pada kurva Bézier.

2. Di dalam loop iterasi, setiap titik pada kurva Bézier dihitung menggunakan rumus kurva Bézier untuk titik-titik kuadrat (formula rumus terdapat pada bab 1). Rumus ini menggunakan parameter  $t$  yang dihitung sebelumnya pada langkah 1, dan titik kendali  $P_0$ ,  $P_1$ , dan  $P_2$  yang diberikan.
3. Setiap titik yang dihasilkan kemudian disimpan dalam variabel `curve` yang akan menjadi output dari fungsi. Titik-titik ini akan membentuk kurva Bézier ketika dihubungkan dalam urutan yang benar.

### 3.5 Source Code Algoritma Brute Force

Kode ditulis dalam bahasa Python. Kode terdapat pada direktori `src` dengan nama *file* `BruteForce.py`.

`BruteForce.py`

```
def QuadraticBezierCurveBruteForce(P0, P1, P2, iterations):
    # Inisialisasi daftar titik pada kurva
    curve = []

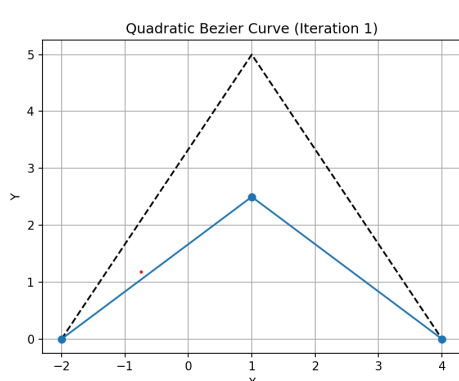
    # Hitung total titik yang akan dibuat
    iterations_new = total_dots(iterations)

    # Iterasi untuk menghasilkan titik pada kurva
    for i in range(iterations_new):
        # Hitung parameter t berdasarkan indeks iterasi
        t = i / (iterations_new - 1)

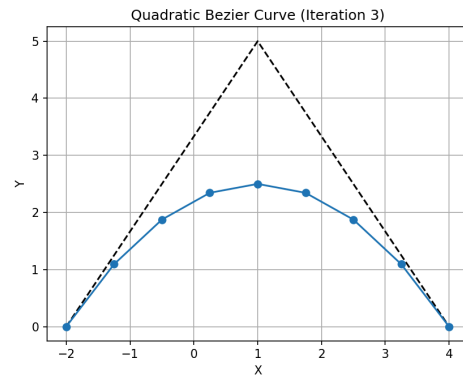
        # Hitung posisi titik pada kurva menggunakan rumus kurva Bézier
        curve.append((1 - t)**2 * P0 + 2 * (1 - t) * t * P1 + t**2 * P2)

    # Kembalikan daftar titik yang membentuk kurva Bézier
    return curve
```

### 3.6 Input dan Output Program Brute Force

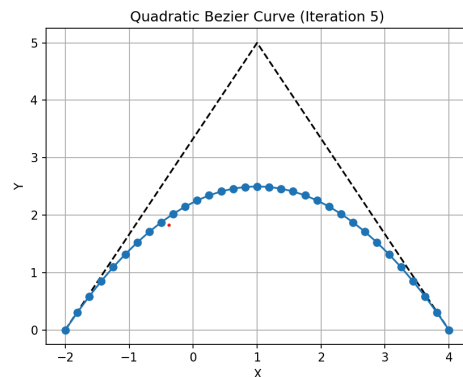
Input	Output
Masukkan titik P0 (contoh: 2,3): -2,0 Masukkan titik P1 (contoh: 2,3): 1,5 Masukkan titik P2 (contoh: 2,3): 4,0 Masukkan Jumlah iterasi: 1	 <p>Execution time: 0.00000000 milliseconds</p>

Masukkan titik P0 (contoh: 2,3): -2,0  
Masukkan titik P1 (contoh: 2,3): 1,5  
Masukkan titik P2 (contoh: 2,3): 4,0  
Masukkan Jumlah iterasi: 3



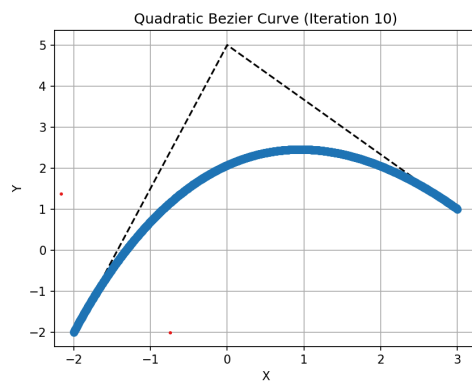
Execution time: 0.00000000 milliseconds

Masukkan titik P0 (contoh: 2,3): -2,0  
Masukkan titik P1 (contoh: 2,3): 1,5  
Masukkan titik P2 (contoh: 2,3): 4,0  
Masukkan Jumlah iterasi: 5



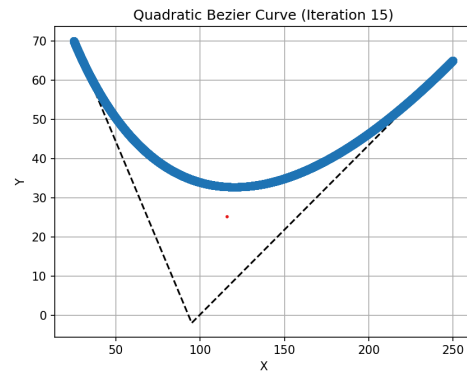
Execution time: 0.00000000 milliseconds

Masukkan titik P0 (contoh: 2,3): -2,-2  
Masukkan titik P1 (contoh: 2,3): 0,5  
Masukkan titik P2 (contoh: 2,3): 3,1  
Masukkan Jumlah iterasi: 10



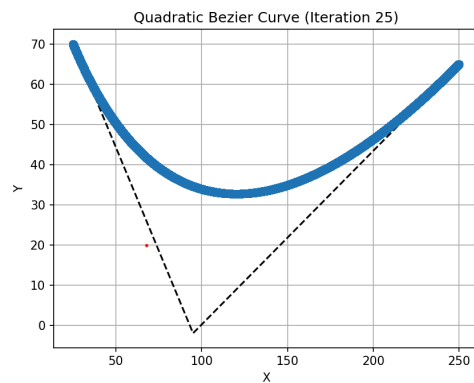
Execution time: 21.01898193  
milliseconds

Masukkan titik P0 (contoh: 2,3): 25,70  
Masukkan titik P1 (contoh: 2,3): 95,-2  
Masukkan titik P2 (contoh: 2,3): 250,65  
Masukkan Jumlah iterasi: 15



Execution time: 381.38818741  
milliseconds

Masukkan titik P0 (contoh: 2,3): 25,70  
Masukkan titik P1 (contoh: 2,3): 95,-2  
Masukkan titik P2 (contoh: 2,3): 250,65  
Masukkan Jumlah iterasi: 25



Execution time: 572740.85521698  
milliseconds

### 3.7 Kompleksitas Algoritma Divide and Conquer

Pada setiap tingkat rekursi, algoritma harus menggabungkan dua segmen kurva. Jumlah segmen yang harus digabungkan menjadi solusi akhir adalah eksponensial tergantung pada jumlah iterasi. Misalnya, untuk setiap level iterasi, jumlah segmen yang dihasilkan dua kali lipat dari level sebelumnya, sehingga jumlah total segmen pada setiap level adalah  $2^k$ , di mana  $k$  adalah nomor iterasi.

Kesimpulannya, jumlah total tingkat rekursi tergantung pada jumlah iterasi yang ditentukan dan jumlah total segmen yang harus digabungkan juga meningkat secara eksponensial. Sehingga, kompleksitas waktu keseluruhan,  $T(n) = O(2^k)$ , di mana  $k$  adalah jumlah iterasi.

### **3.8 Kompleksitas Algoritma Brute Force**

Dalam algoritma ini, terdapat fungsi `total_dots` memanggil dirinya sendiri rekursif hingga mencapai kasus dasar ketika  $n == 1$ . jumlah total panggilan rekursif adalah sekitar  $2^n - 1$  ketika  $n > 1$ . Berarti, Jumlah total iterasi dalam loop for adalah sekitar  $2^{\text{iterations}} - 1$ . Jadi, kompleksitas waktu algoritma ini adalah  $T(n) = O(2^n)$ , di mana  $n$  adalah jumlah iterasi.

### **3.9 Perbandingan Solusi Brute force dengan Divide and Conquer**

Dalam analisis kompleksitas waktu, baik pendekatan rekursif maupun brute force memiliki kompleksitas  $O(2^n)$ .

## Lampiran 1

*Checklist* penilaian :

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil dijalankan	✓	
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	✓	

## Lampiran 2

Link Repository github : [https://github.com/cupski/Tucil2\\_13522009](https://github.com/cupski/Tucil2_13522009)