

**LAPORAN TUGAS KECIL II**  
**IF2211 STRATEGI ALGORITMA**  
**PATAN**



**Disusun oleh :**

Satriadhikara Panji Yudhistira (13522125)

Christian Justin Hendrawan (13522135)

**Program Studi Teknik Informatika**

**Institut Teknologi Bandung**

**2024**

## **Daftar Isi**

Daftar Isi.....	2
Daftar Gambar.....	3
BAB I.....	5
I. Tujuan.....	5
II. Spesifikasi.....	5
BAB II.....	8
2.1. Algoritma Divide and Conquer.....	8
2.2. Kurva Bézier.....	9
2.3. Algoritma De Casteljau.....	12
BAB III.....	14
3.1. Implementasi Algoritma Brute Force.....	14
3.2. Implementasi Algoritma Divide and Conquer.....	19
BAB IV.....	22
BAB V.....	34
BAB VI.....	40
6.1. Analisis Hasil Eksperimen.....	40
6.2. Perbandingan Kompleksitas Algoritma dengan Brute Force.....	41
6.3. Pembahasan Mengenai Perbedaan Hasil Algoritma.....	42
BAB VII.....	44
BAB VIII.....	47
BAB IX.....	48

## Daftar Gambar

Gambar 1.1 Kurva Bézier Kubik.....	5
Gambar 1.2 Kurva Bézier Kuadratik.....	7
Gambar 2.1 Kurva Bézier.....	9
Gambar 2.2 Kurva Bézier Linier.....	10
Gambar 2.3 Kurva Bézier Kuadratik.....	11
Gambar 2.4 Kurva Bézier Kubik.....	11
Gambar 2.5 Kurva Bézier dengan derajat tinggi.....	12
Gambar 3.1 Segitiga Pascal.....	15
Gambar 4.1 Layar input.....	22
Gambar 4.2 Layar hasil kurva algoritma Brute Force.....	23
Gambar 4.3 Layar hasil kurva algoritma Divide and Conquer.....	23
Gambar 4.4 Layar mengubah iterasi secara dinamis.....	24
Gambar 4.5 Layar input kasus 1.....	24
Gambar 4.6 Layar hasil kasus 1 dengan algoritma Divide and Conquer.....	25
Gambar 4.7 Layar hasil kasus 1 dengan algoritma Brute Force.....	25
Gambar 4.8 Layar input kasus 2.....	26
Gambar 4.9 Layar hasil kasus 2 dengan algoritma Brute Force.....	26
Gambar 4.10 Layar hasil kasus 2 dengan algoritma Divide and Conquer.....	27
Gambar 4.11 Layar input kasus 3.....	27
Gambar 4.12 Layar hasil kasus 3 dengan algoritma Brute Force.....	28
Gambar 4.13 Layar hasil kasus 3 dengan algoritma Divide and Conquer.....	28
Gambar 4.14 Layar input kasus 4.....	29
Gambar 4.15 Layar hasil kasus 4 dengan algoritma Brute Force.....	29
Gambar 4.16 Layar hasil kasus 4 dengan algoritma Divide and Conquer.....	30
Gambar 4.17 Layar input kasus 5.....	30
Gambar 4.18 Layar hasil kasus 5 dengan algoritma Brute Force.....	31

Gambar 4.19 Layar hasil kasus 5 dengan algoritma Divide and Conquer.....	31
Gambar 4.20 Layar input kasus 6.....	32
Gambar 4.21 Layar hasil kasus 5 dengan algoritma Brute Force.....	32
Gambar 4.22 Layar hasil kasus 5 dengan algoritma Divide and Conquer.....	33

# BAB I

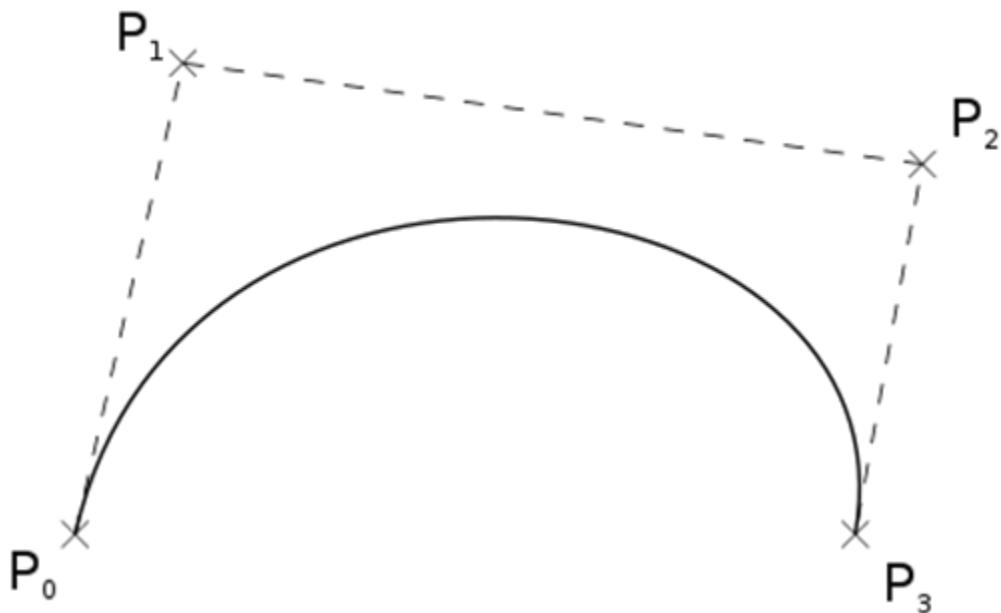
## Deskripsi Masalah

### I. Tujuan

Pada tugas kecil ini, kita diminta untuk :

- Membuat program sederhana dalam bahasa C++/Python/JavaScript/Go 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 pembanding dengan solusi sebelumnya.

### II. Spesifikasi



Gambar 1.1 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 realistik, 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.

Membahas lebih lanjut tentang pembentukan kurva Bézier, jika kita mempunyai dua titik kontrol,  $P_0$  dan  $P_1$ , kurva Bézier yang dihasilkan adalah sebuah garis lurus yang menghubungkan kedua titik tersebut. Kurva ini dikenal sebagai kurva Bézier linier. Jika ada titik  $Q_0$  yang letaknya di sepanjang garis antara  $P_0$  dan  $P_1$ , maka lokasi  $Q_0$  dapat dijelaskan menggunakan persamaan parameter berikut.

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

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  $P_1$ .

Dengan menambahkan sebuah titik baru, misalnya  $P_2$ , di antara dua titik yang sudah ada sebelumnya,  $P_0$  dan  $P_2$  bisa digunakan sebagai titik kontrol awal dan akhir, sementara  $P_1$  menjadi titik kontrol di antara keduanya. Dengan titik  $Q_1$  yang terletak di antara garis yang menghubungkan  $P_1$  dan  $P_2$ , kita dapat membentuk sebuah kurva Bezier linier baru dengan titik

$Q_0$  yang terletak di sepanjang kurva tersebut. Kemudian, kita dapat menentukan sebuah titik baru,  $R_0$ , yang berada di antara garis yang menghubungkan  $Q_0$  dan  $Q_1$ . Titik  $R_0$  ini akan bergerak membentuk sebuah kurva Bézier kuadratik terhadap titik  $P_0$  dan  $P_2$ . Persamaan matematika yang menggambarkan pergerakan  $R_0$  dalam kurva Bézier kuadratik tersebut dapat dinyatakan secara detail.

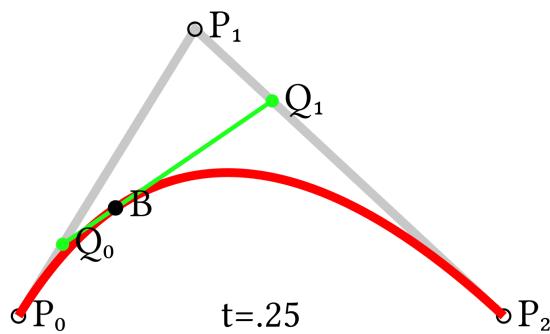
$$Q_0 = B(t) = (1 - t)P_0 + tP_1, \quad t \in [0, 1]$$

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

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

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

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



**Gambar 1.2** Kurva Bézier Kuadratik

Metode ini juga dapat diterapkan untuk lebih dari tiga titik kontrol. Sebagai contoh, empat titik kontrol akan menciptakan kurva Bézier kubik, lima titik kontrol akan menciptakan kurva Bézier kuartik, dan seterusnya. Berikut ini adalah formula untuk kurva Bézier kubik dan kuartik, menggunakan proses yang sama seperti yang dijelaskan sebelumnya.

Memang benar, semakin banyak titik yang terlibat, persamaan yang dihasilkan akan menjadi semakin panjang dan kompleks. Karena itu, untuk meningkatkan efisiensi dalam pembuatan kurva Bézier yang sangat berguna ini, Anda diminta untuk menerapkan pembuatan kurva dengan menggunakan algoritma titik tengah yang mengandalkan pendekatan *Divide and Conquer*.

## BAB II

### Teori Singkat

#### 2.1. Algoritma Divide and Conquer

Algoritma *Divide and Conquer* merupakan algoritma dengan pendekatan rekursif yang memanfaatkan pembagian persoalan menjadi upa persoalan yang lebih kecil, menyelesaiakannya secara rekursif dan menggabungkan solusinya untuk mendapatkan solusi pada masalah utama.

Berikut merupakan skema umum dalam 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 ≤ n0 then {ukuran persoalan P sudah cukup kecil}  
            SOLVE persoalan P yang berukuran n  
        else  
            DIVIDE menjadi r upa-persoalan, P1, P2, ..., Pr yang  
            masing masing berukuran n1, n2, ..., nr  
            for masing-masing P1, P2, ..., Pr , do
```

```

    DIVIDEandCONQUER ( $P_i$ ,  $n_i$ )
    endfor

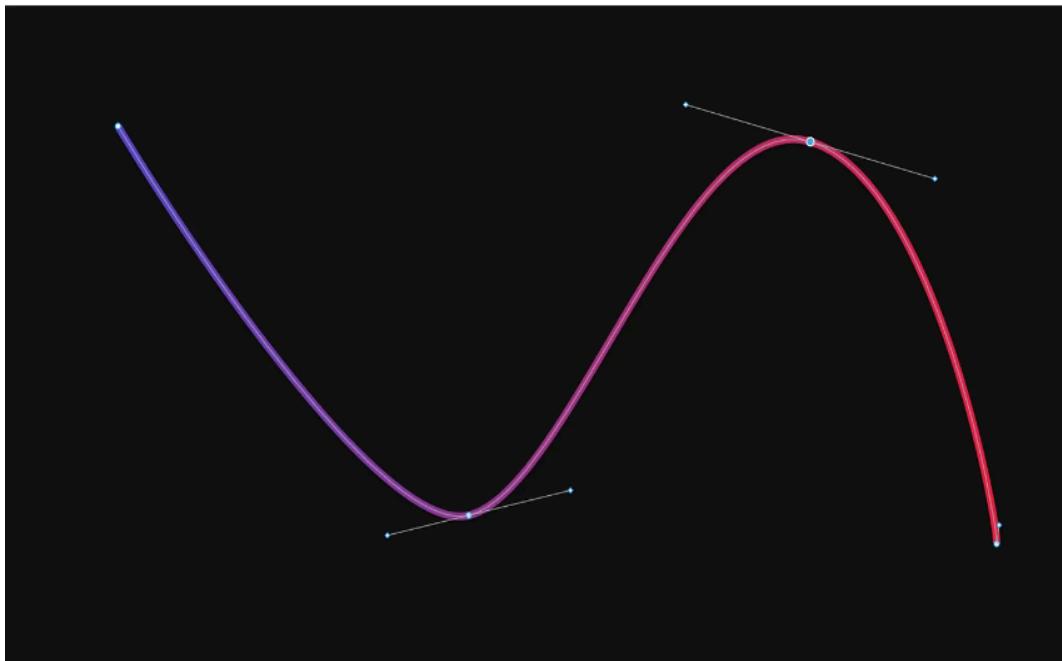
    COMBINE solusi dari  $P_1, P_2, \dots, P_r$  menjadi persoalan
semula

    endif

```

## 2.2. Kurva Bézier

Kurva Bézier atau lengkungan Bézier merupakan kurva berparameter yang sering digunakan dalam grafika komputer dan bidang yang berkaitan. Kurva Bezier didefinisikan oleh serangkaian titik kontrol (control points) yang menentukan jalur kurva. Kurva ini bisa berupa garis lurus, kurva melengkung, atau bentuk-bentuk yang lebih kompleks, tergantung pada jumlah dan posisi titik kontrolnya. Biasanya, kurva Bezier dibuat dalam dua atau tiga dimensi.



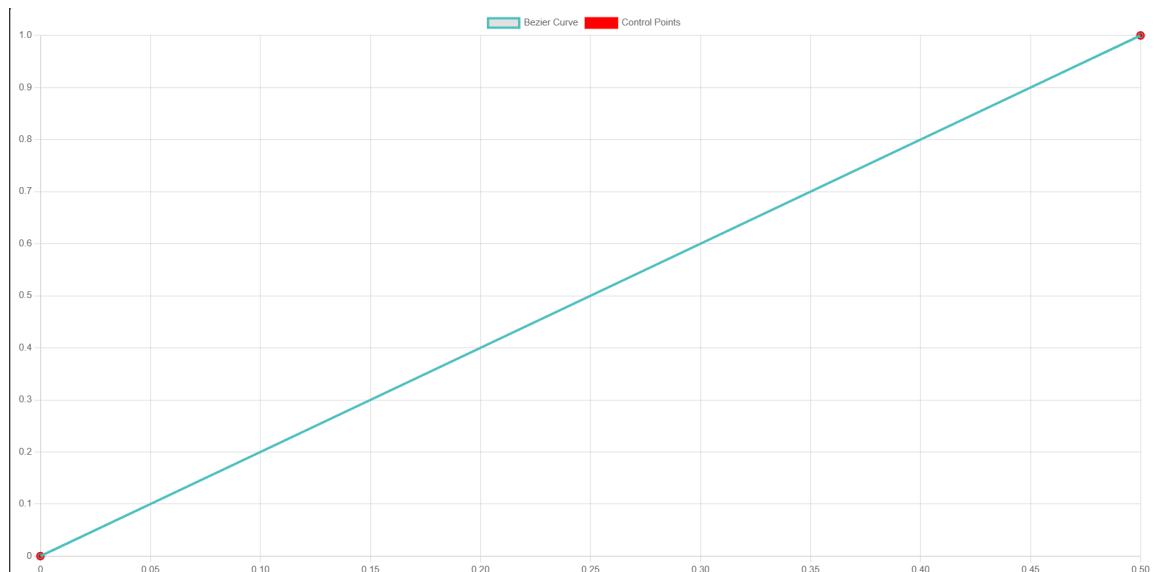
**Gambar 2.1** Kurva Bézier

Kurva Bézier adalah alat yang sangat serbaguna dan umum digunakan dalam berbagai konteks. Dalam desain grafis, kurva Bezier digunakan dalam perangkat lunak

desain grafis untuk membuat bentuk-bentuk kompleks seperti logo, karakter, atau ilustrasi. Dalam animasi komputer, digunakan untuk mengontrol gerakan objek, lintasan kamera, dan perubahan bentuk objek seiring waktu. Dalam desain industri, kurva Bezier diterapkan dalam perancangan produk fisik seperti mobil, pesawat terbang, dan produk konsumen lainnya. Dalam pemodelan 3D, kurva Bezier digunakan dalam pembuatan model 3D dan desain industri untuk menghasilkan bentuk dan lintasan yang akurat.

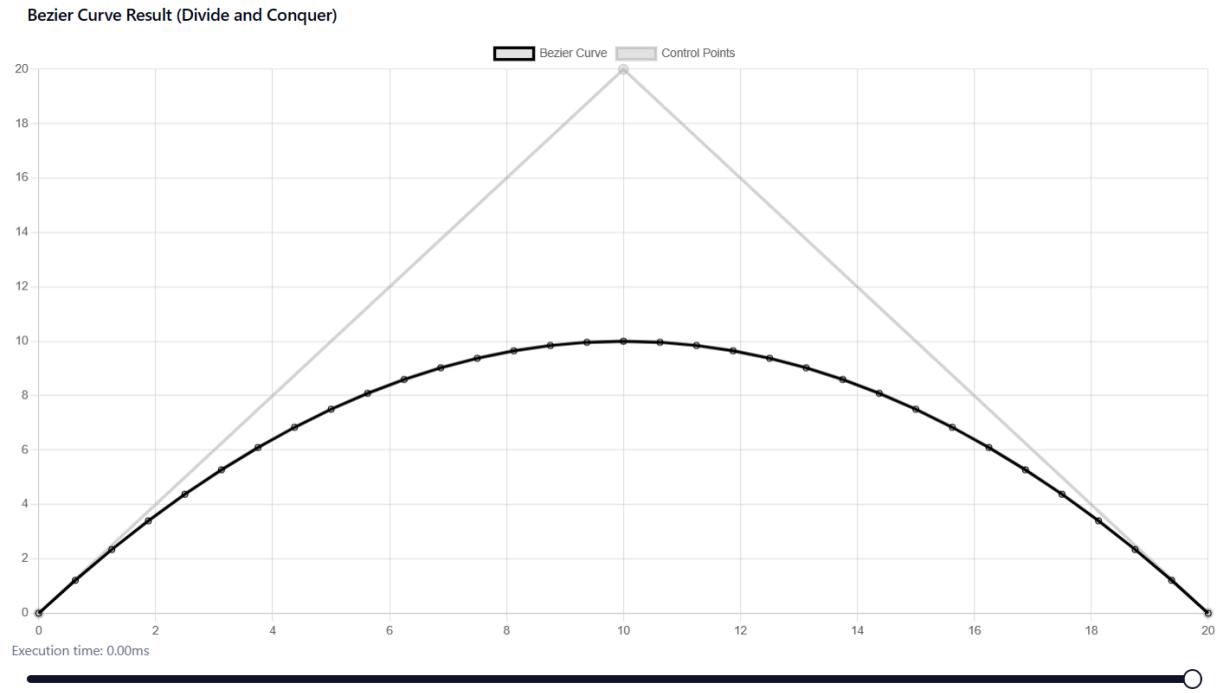
Terdapat beberapa jenis kurva Bezier yakni :

1. Kurva Bezier Linier (*Degree 1*): Didefinisikan oleh dua titik kontrol dan menghasilkan garis lurus.



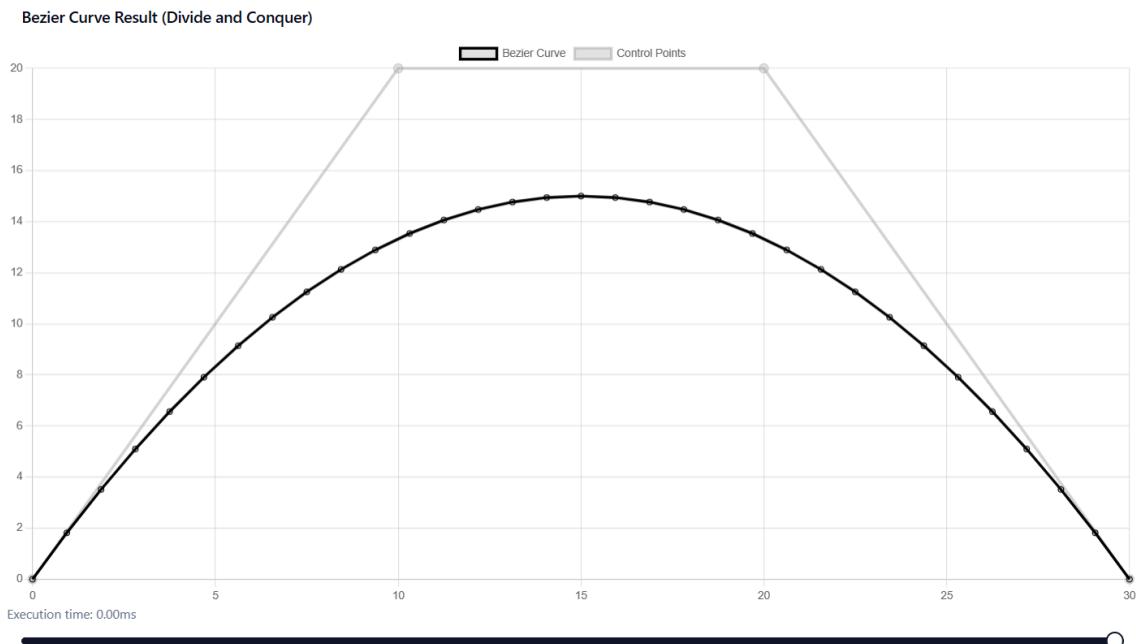
**Gambar 2.2** Kurva Bézier Linier

2. Kurva Bezier Kuadratik (*Degree 2*): Didefinisikan oleh tiga titik kontrol dan menghasilkan kurva melengkung satu tingkat.



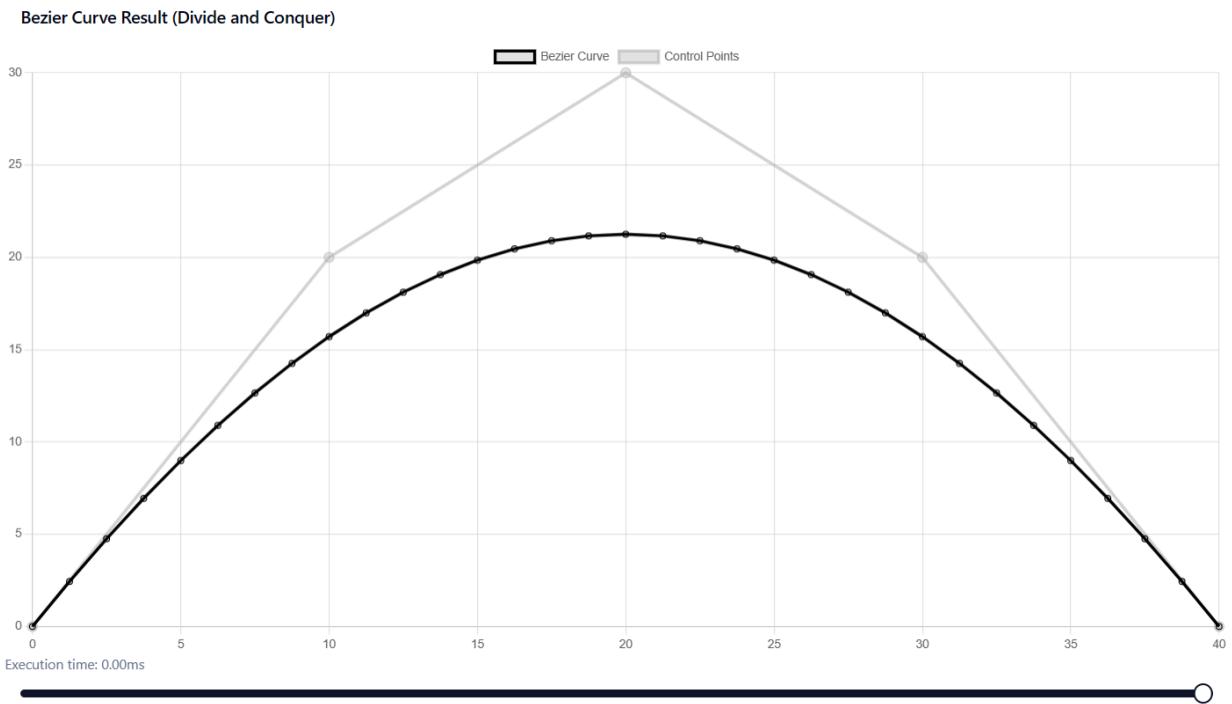
**Gambar 2.3** Kurva Bézier Kuadratik

3. Kurva Bezier Kubik (*Degree 3*): Didefinisikan oleh empat titik kontrol dan merupakan yang paling umum digunakan, menghasilkan kurva melengkung lebih kompleks.



**Gambar 2.4** Kurva Bézier Kubik

4. Kurva Bezier dengan Derajat Lebih Tinggi: Meskipun kurva Bezier kubik adalah yang paling umum, kurva dengan derajat yang lebih tinggi juga mungkin digunakan untuk menciptakan bentuk yang lebih kompleks.



**Gambar 2.5** Kurva Bézier dengan derajat tinggi

### 2.3. Algoritma De Casteljau

Algoritma Bezier De Casteljau adalah metode numerik yang digunakan untuk menghitung titik-titik pada kurva Bezier. Ini didasarkan pada konsep pemisahan titik kontrol kurva menjadi dua set titik kontrol yang lebih kecil, yang kemudian digunakan untuk menghasilkan titik-titik pada kurva tersebut.

Langkah-langkah algoritma Bezier De Casteljau adalah sebagai berikut:

1. Pemisahan Titik Kontrol: Pertama, kurva Bezier dengan titik kontrol awalnya dipisahkan menjadi dua set titik kontrol yang lebih kecil. Ini dilakukan dengan menghitung titik-titik "midpoint" antara setiap pasang titik kontrol berturut-turut.

2. Rekursif: Setelah pemisahan dilakukan, proses ini diulangi untuk setiap set titik kontrol yang baru dihasilkan. Proses pemisahan dan pengulangan ini dilakukan secara rekursif sampai tingkat kedalaman yang diinginkan tercapai.
3. Penggabungan: Akhirnya, titik-titik pada kurva Bezier dihitung dengan menggunakan titik kontrol yang dihasilkan dari pemisahan dan pengulangan sebelumnya. Ini dilakukan dengan cara interpolasi linear dari titik kontrol pada setiap tingkat kedalaman.

Hubungan antara algoritma Bezier De Casteljau dengan metode *Divide and Conquer* adalah bahwa algoritma ini mengadopsi pendekatan *Divide and Conquer* secara intrinsik. Dengan membagi kurva menjadi segmen-segmen yang lebih kecil dan menghitung titik-titik pada setiap segmen secara terpisah, algoritma ini memanfaatkan prinsip dasar dari strategi *Divide and Conquer*.

Dengan menggunakan pendekatan *Divide and Conquer*, algoritma Bezier De Casteljau dapat mencapai kompleksitas waktu yang lebih efisien daripada metode *Brute Force* untuk menghitung kurva Bezier, terutama ketika jumlah titik kontrol besar. Hal ini karena membagi masalah menjadi submasalah yang lebih kecil dapat mengurangi jumlah operasi yang diperlukan secara keseluruhan.

## BAB III

### Implementasi Algoritma Program

#### 3.1. Implementasi Algoritma Brute Force

Sesuai dengan namanya, *Brute Force*, maka dalam konteks pembuatan kurva Bezier adalah pendekatan yang sederhana namun memakan waktu yang banyak karena melibatkan perhitungan setiap titik pada kurva secara langsung dengan menguji sejumlah besar nilai parameter  $t$ . Karena algoritma ini mengharuskan untuk menghitung setiap titik pada kurva dengan menguji banyak nilai parameter  $t$ , kompleksitas waktu algoritma ini meningkat secara eksponensial dengan jumlah titik kontrol yang digunakan. Oleh karena itu, untuk kurva Bezier yang rumit atau memiliki banyak titik kontrol, pendekatan *Brute Force* mungkin tidak efisien secara praktis, dan pendekatan lain seperti algoritma *Divide and Conquer*.

Berikut adalah langkah-langkah implementasi algoritma *Brute Force* dalam pembuatan kurva Bezier yang diterapkan dalam program:

1. Definisikan Titik Kontrol: Langkah pertama adalah menentukan titik kontrol yang akan membentuk kurva Bezier. Biasanya, kurva Bezier didefinisikan oleh beberapa titik kontrol, yang dihubungkan oleh kurva. Titik kontrol terdefinisi menjadi  $P_0, P_1, \dots, P_r$ .
2. Tentukan Rentang Parameter  $t$ : Tentukan rentang nilai parameter  $t$  yang akan diuji. Secara umum, rentang ini adalah dari 0 hingga 1, karena  $t$  adalah parameter yang berkisar dari awal hingga akhir kurva Bezier.
3. Iterasi melalui Parameter  $t$  : Untuk setiap nilai parameter  $t$  dalam rentang yang ditentukan, lakukan langkah-langkah berikut:
  - a. Gunakan persamaan kurva Bezier untuk menghitung koordinat titik pada kurva untuk nilai  $t$  yang diberikan. Persamaan kurva bezier untuk setiap jenisnya :
    - Linier :

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

- Kuadratik :

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

- Kubik :

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

- Kuartik:

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

Dimana  $P_0, P_1, \dots, P_r$  adalah titik kontrol, dan  $t$  adalah parameter yang berada di rentang  $[0,1]$ . Ketika ditelusuri lebih lanjut, terdapat sebuah pola yang konsisten pada semua persamaan kurva Bézier. Koefisien binomial dalam persamaan kurva Bézier diperoleh dari segitiga Pascal. Hal ini berarti bahwa untuk setiap titik dalam kurva Bézier, koefisien binomial yang digunakan dalam persamaan tersebut dapat ditemukan dengan merujuk pada segitiga Pascal.

$$\begin{array}{ccccccc}
 & & & 1 & & & \\
 & & & 1 & 1 & & \\
 & & & 1 & 2 & 1 & \\
 & & & 1 & 3 & 3 & 1 \\
 & & & 1 & 4 & 6 & 4 & 1 \\
 & & & 1 & 5 & 10 & 10 & 5 & 1
 \end{array}$$

**Gambar 3.1** Segitiga Pascal

- Simpan koordinat titik yang dihitung untuk setiap nilai  $t$ . Ini akan membentuk serangkaian titik yang merupakan representasi dari kurva Bezier.

4. Gambar Kurva : Setelah semua titik terkumpul, maka tarik sebuah garis dari titik awal atau titik kontrol pertama menuju titik-titik yang sudah ditemukan dan berakhir pada titik terakhir yakni titik kontrol terakhir.

Berikut adalah implementasi *Brute Force* dalam source code kami:

Nama fungsi / prosedur	Deskripsi
generatePascalTriangle	Digunakan untuk menghasilkan segitiga Pascal hingga baris tertentu. Segitiga Pascal digunakan dalam perhitungan koefisien binomial untuk kurva Bezier. Fungsi ini akan memudahkan kita untuk mencari persamaan dari kurva Bezier N points terutama untuk kurva Bezier yang memiliki lebih dari 3 control point

Nama fungsi / prosedur	Deskripsi
generateBezierCurve3Points	Digunakan untuk menghasilkan titik-titik pada kurva Bezier jika jumlah titik kontrol adalah 3. Ini menggunakan formula Bezier khusus untuk 3 titik kontrol yakni formula Bezier Kuadratik. Fungsi ini menerima dua argumen: controlPoints, yang merupakan array dari titik-titik kontrol, dan iterationLevel. Fungsi ini mendefinisikan array kosong points untuk menyimpan titik-titik pada kurva. Fungsi ini kemudian melakukan iterasi dari 0 hingga iterationLevel (inklusif). Untuk setiap iterasi:

	<ul style="list-style-type: none"> <li>- Menghitung parameter <math>t</math> dengan membagi <math>i</math> (indeks iterasi saat ini) dengan <code>iterationLevel</code>. Parameter <math>t</math> ini berfungsi sebagai parameter interpolasi pada kurva Bezier, yang berkisar dari 0 hingga 1.</li> <li>- Menghitung koordinat <math>x</math> dan <math>y</math> dari titik pada kurva Bezier menggunakan formula Bezier</li> <li>- Menambahkan titik dengan koordinat <math>x</math> dan <math>y</math> ke array <code>points</code>.</li> </ul> <p>Setelah iterasi selesai, fungsi ini mengembalikan array <code>points</code> yang berisi titik-titik pada kurva Bezier.</p>
--	--

Nama fungsi / prosedur	Deskripsi
<code>generateBezierCurveNPoints</code>	Digunakan untuk menghasilkan titik-titik pada kurva Bezier jika jumlah titik kontrol lebih dari 3. Ini menggunakan formula Bezier umum yang melibatkan koefisien binomial (dihitung menggunakan segitiga Pascal). Fungsi ini menerima dua argumen: <code>controlPoints</code> , yang merupakan array dari titik-titik kontrol, dan <code>iterationLevel</code> . Fungsi ini mendefinisikan array kosong <code>points</code> untuk menyimpan titik-titik pada kurva. Fungsi ini juga menghitung $n$ sebagai jumlah titik kontrol dikurangi 1, dan menghasilkan

	<p>segitiga Pascal hingga <math>n + 1</math> baris menggunakan fungsi generatePascalTriangle. Fungsi ini kemudian melakukan iterasi dari 0 hingga iterationLevel (inklusif). Untuk setiap iterasi:</p> <ul style="list-style-type: none"> <li>- Membuat objek point dengan koordinat x dan y awalnya adalah 0.</li> <li>- Menghitung parameter t dengan membagi i (indeks iterasi saat ini) dengan iterationLevel. Parameter t ini berfungsi sebagai parameter interpolasi pada kurva Bezier, yang berkisar dari 0 hingga 1.</li> <li>- Melakukan iterasi kedua dari 0 hingga <math>n</math> (jumlah titik kontrol dikurangi 1). Untuk setiap iterasi ini: Menghitung b sebagai koefisien binomial dari segitiga Pascal dikalikan dengan <math>(1 - t)^{n - j}</math> dan <math>t^j</math>. Lalu menambahkan b dikalikan dengan koordinat x dan y dari titik kontrol ke koordinat x dan y dari point.</li> <li>- Menambahkan point ke array points.</li> </ul> <p>Setelah iterasi selesai, fungsi ini mengembalikan array points yang berisi titik-titik pada kurva Bezier.</p>
--	---

Perbedaan antara 3 titik kontrol dan lebih dari 3 titik kontrol adalah bagaimana titik-titik pada kurva dihitung. Jika ada 3 titik kontrol, kita dapat menggunakan formula Bezier khusus untuk 3 titik kontrol. Jika ada lebih dari 3 titik kontrol, kita harus

menggunakan formula Bezier umum yang melibatkan koefisien binomial. Koefisien binomial ini dihitung menggunakan segitiga Pascal.

Secara keseluruhan, meskipun algoritma *Brute Force* sederhana dan mudah dipahami, kelemahan utamanya adalah kinerja yang buruk terutama untuk kurva dengan jumlah titik kontrol yang besar. Ini membuatnya kurang sesuai untuk aplikasi praktis di mana efisiensi waktu eksekusi menjadi pertimbangan utama. Dalam kasus-kasus di mana kinerja menjadi faktor penting, pendekatan lain seperti metode *Divide and Conquer*.

### 3.2. Implementasi Algoritma Divide and Conquer

Nama algoritma *Divide and Conquer* berasal dari dua kata, yakni divide dan conquer. Divide berarti persoalan yang besar dibagi menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula, , tetapi ukurannya lebih kecil, sedangkan Conquer (solve) berarti upa-persoalan masing-masing diselesaikan (diselesaikan secara rekursif jika masih berukuran besar atau diselesaikan secara langsung jika sudah berukuran kecil). Dengan algoritma ini, nantinya solusi masing-masing upa-persoalan akan digabung untuk membentuk solusi persoalan semula.

*Divide and Conquer* dapat digunakan sebagai algoritma untuk menyelesaikan persoalan pembuatan kurva Bezier dengan 3 kontrol point sampai N kontrol point. Persoalan pembuatan kurva Bezier dengan 3 kontrol point dapat diselesaikan dengan cara berikut:

1. Divide: Proses pembagian masalah menjadi dua bagian yang lebih kecil terjadi dengan mencari titik tengah antara setiap pasangan titik kontrol. Ini dilakukan dengan menggunakan fungsi mid point untuk menghitung koordinat x dan y dari titik tengah antara dua titik kontrol.
2. Conquer: Setelah pembagian masalah, proses pemecahan terjadi di mana titik tengah pertama dihitung antara titik kontrol 1 dan 2, dan titik tengah kedua dihitung antara titik kontrol 2 dan 3. Kemudian, titik tengah ketiga dihitung dari titik tengah pertama dan kedua. Selanjutnya, fungsi akan memanggil dirinya

sendiri dua kali, masing-masing dengan setengah dari titik kontrol. Ini adalah bagian "divide" dari algoritma *Divide and Conquer*, yang merupakan proses pemecahan masalah secara rekursif sebanyak jumlah iterasi yang diinginkan.

3. Combine: Setelah mendapatkan titik tengah antara dua titik tengah, titik-titik tersebut dimasukkan ke dalam array yang menyimpan titik-titik kurva. Ini merupakan bagian "combine" dari algoritma, di mana titik-titik tersebut digunakan untuk membentuk kurva Bézier utuh.

Berikut adalah implementasi *Divide and Conquer* dengan 3 titik kontrol dalam source code kami:

Nama fungsi / prosedur	Deskripsi
divideConquerBezier3Points	fungsi utama yang menerima 3 titik kontrol dan jumlah iterasi sebagai argumen. Fungsi ini menginisialisasi array bezierPoints dan menambahkan titik kontrol pertama dan ketiga ke array tersebut. Kemudian, fungsi ini memanggil fungsi isiTitikCurve untuk mengisi titik-titik lainnya pada kurva dan mengembalikan array bezierPoints

Nama fungsi / prosedur	Deskripsi
isiTitikCurve	Ini adalah fungsi yang mengimplementasikan algoritma <i>Divide and Conquer</i> pada program kami. Fungsi rekursif yang menerima 3 titik kontrol dan iterasi saat ini dan jumlah iterasi sebagai argumen. Fungsi ini membagi masalah menjadi dua bagian yang lebih kecil

	dengan mencari titik tengah antara titik kontrol dan memanggil dirinya sendiri untuk setiap bagian. Fungsi ini juga menambahkan titik tengah antara dua titik tengah ke array bezierPoints. Proses ini diulangi hingga iterasi mencapai jumlah iterasi.
--	---

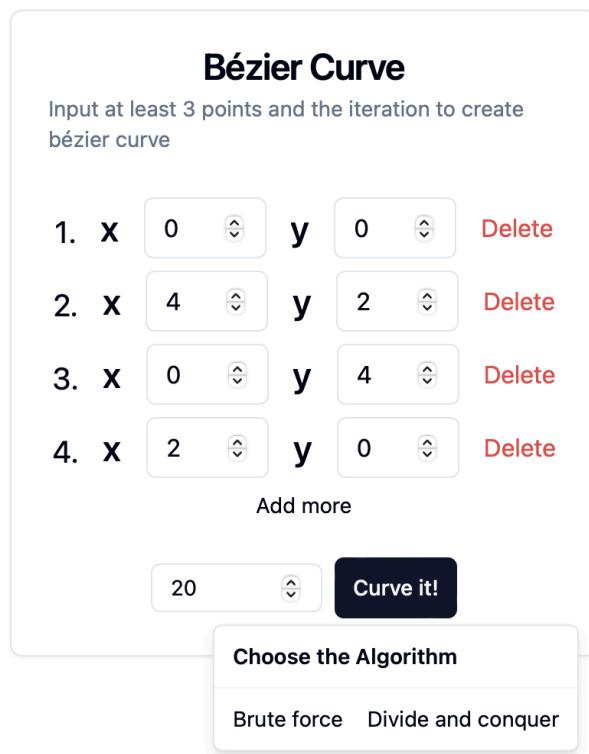
Nama fungsi / prosedur	Deskripsi
cariTitikTengah	fungsi helper yang menerima dua titik kontrol dan mengembalikan titik tengah antara dua titik tersebut (Midpoint)

Dengan menggunakan pendekatan *Divide and Conquer*, pembuatan kurva Bézier dapat dikelola dengan lebih efisien dan lebih mudah dipahami, karena memungkinkan kompleksitas pembuatan kurva yang rumit dipecah menjadi bagian-bagian yang lebih kecil dan lebih mudah dikelola. Untuk analisis dan implementasi algoritma *Divide and Conquer* untuk memecahkan permasalahan N points akan lebih dibahas pada bab VII.

## BAB IV

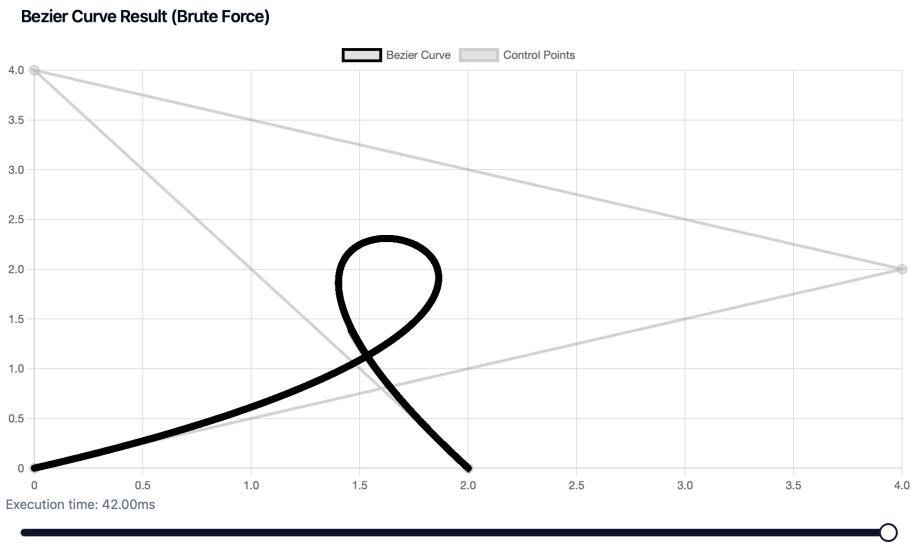
### Tangkapan Layar Input dan Output

Dalam tampilan layar program ini, ada sebuah kotak di tengah untuk melakukan input. Input dapat dilakukan dengan memasukkan minimal 3 poin (x,y) dan iterasi ( $> 1$ ). Input-an poin juga bisa bertambah seiring pengguna perlukan dan juga bisa dihapus. Pengguna hanya bisa memencet tombol “Curve it!” saat semua inputan valid, kemudian pengguna bisa memilih diantara dua algoritma yang tersedia seperti contoh berikut



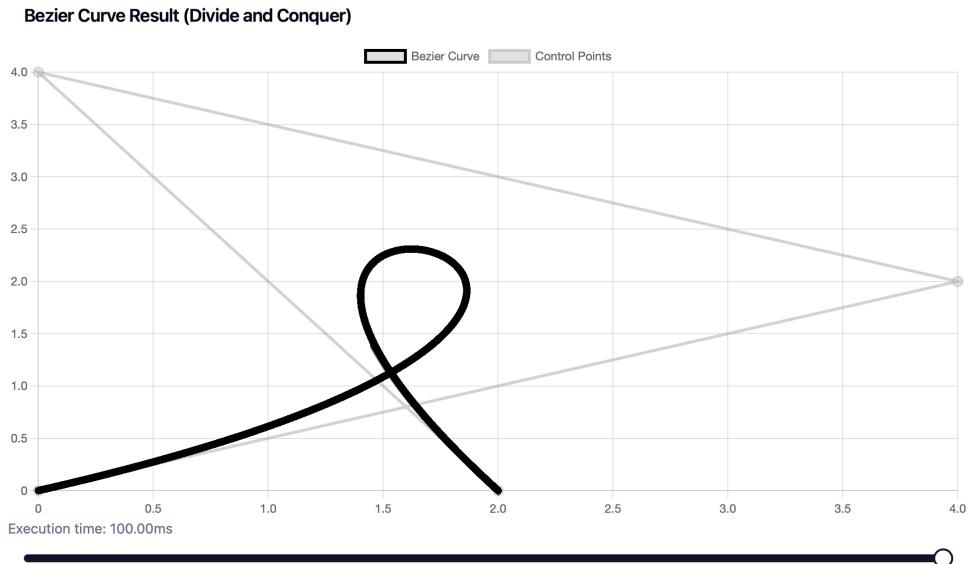
**Gambar 4.1** Layar input

Contoh di atas menunjukkan input 4 titik dengan iterasi 20. Sesuai dengan algoritma yang dipilih, program akan mengeluarkan output sesuai algoritmanya. Output berupa sebuah grafik kurva Bézier yang sudah dikalkulasi sesuai dengan titik kontrol yang diinput. Jika pengguna memilih algoritma “Brute Force” maka akan mengeluarkan seperti berikut:



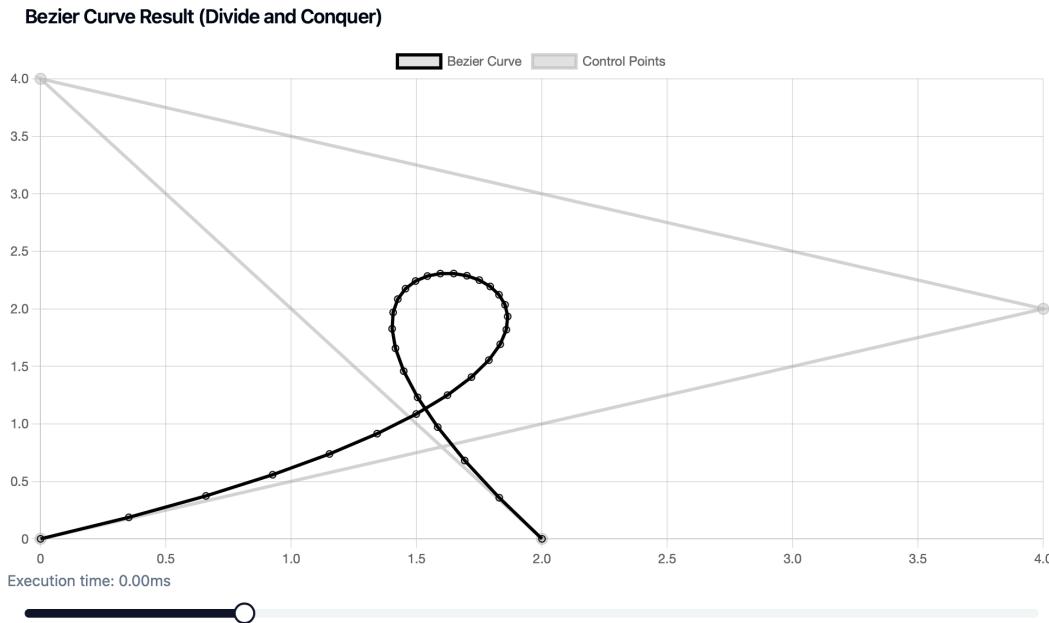
**Gambar 4.2** Layar hasil kurva algoritma Brute Force

Bisa dilihat kalau di bawah kiri ada “Execution time” yang menandakan berapa lama kalkulasi untuk menghasilkan hasil titik kurva tersebut dijalankan. Sebagai catatan, **waktu tersebut hanya memberikan info berapa lama kalkulasinya, tidak termasuk berapa lama web untuk me-render komponen-komponen nya**. Sama halnya dengan contoh di atas, kalau pengguna memilih algoritma “*Divide and Conquer*” maka akan menghasilkan output sebagai berikut:



**Gambar 4.3** Layar hasil kurva algoritma Divide and Conquer

Dalam output tersebut terdapat hasil kurva béziernya dan juga titik-titik (kontrol) yang di input oleh pengguna dan juga di paling bawah terdapat sebuah *slider* untuk mengubah iterasi dengan maksimal iterasi yang diinput dan minimal iterasi sama dengan 1.



**Gambar 4.4** Layar mengubah iterasi secara dinamis

### 1. Kasus 1

Dalam kasus ini, kita akan menginput 4 titik dengan iterasi 20

### Bézier Curve

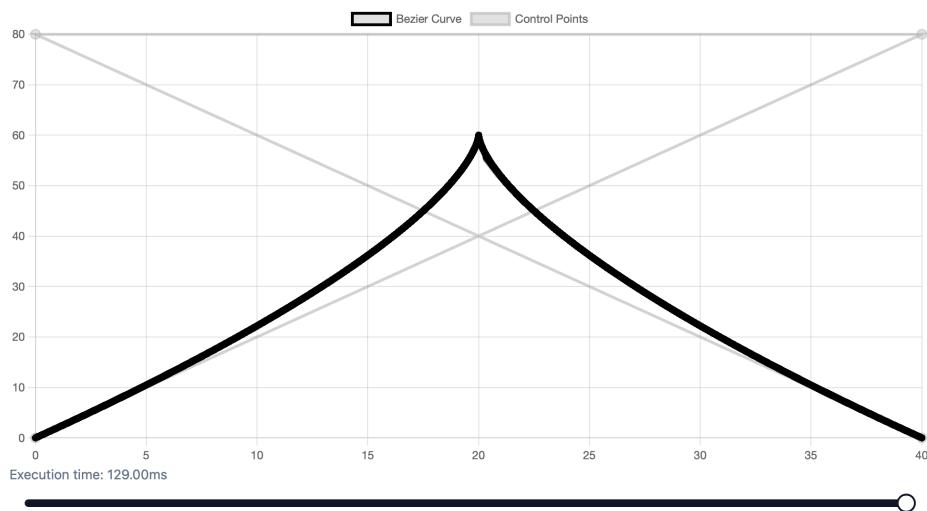
Input at least 3 points and the iteration to create  
bézier curve

1. x	<input type="text" value="0"/>	<input type="button" value="Delete"/>	y	<input type="text" value="0"/>	<input type="button" value="Delete"/>
2. x	<input type="text" value="40"/>	<input type="button" value="Delete"/>	y	<input type="text" value="80"/>	<input type="button" value="Delete"/>
3. x	<input type="text" value="0"/>	<input type="button" value="Delete"/>	y	<input type="text" value="80"/>	<input type="button" value="Delete"/>
4. x	<input type="text" value="40"/>	<input type="button" value="Delete"/>	y	<input type="text" value="0"/>	<input type="button" value="Delete"/>

[Add more](#)

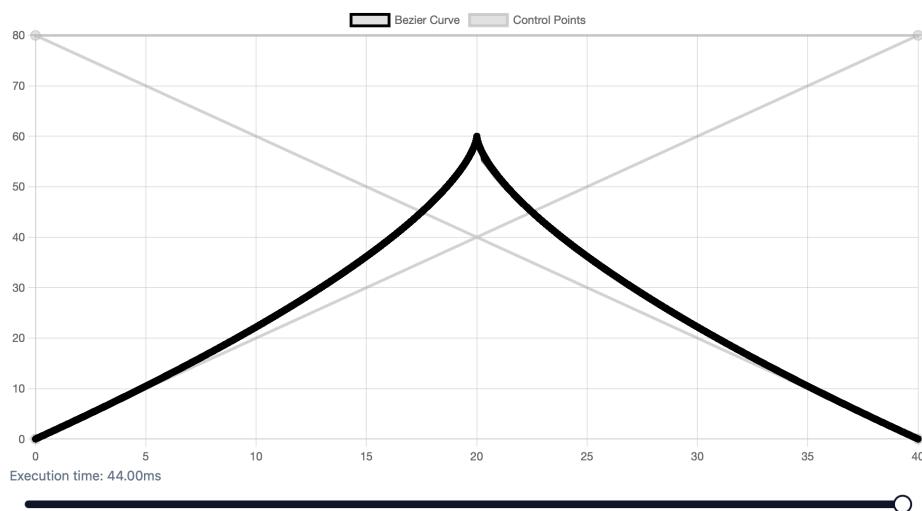
**Gambar 4.5** Layar input kasus 1

**Bezier Curve Result (Divide and Conquer)**



**Gambar 4.6** Layar hasil kasus 1 dengan algoritma Divide and Conquer

**Bezier Curve Result (Brute Force)**



**Gambar 4.7** Layar hasil kasus 1 dengan algoritma Brute Force

## 2. Kasus 2

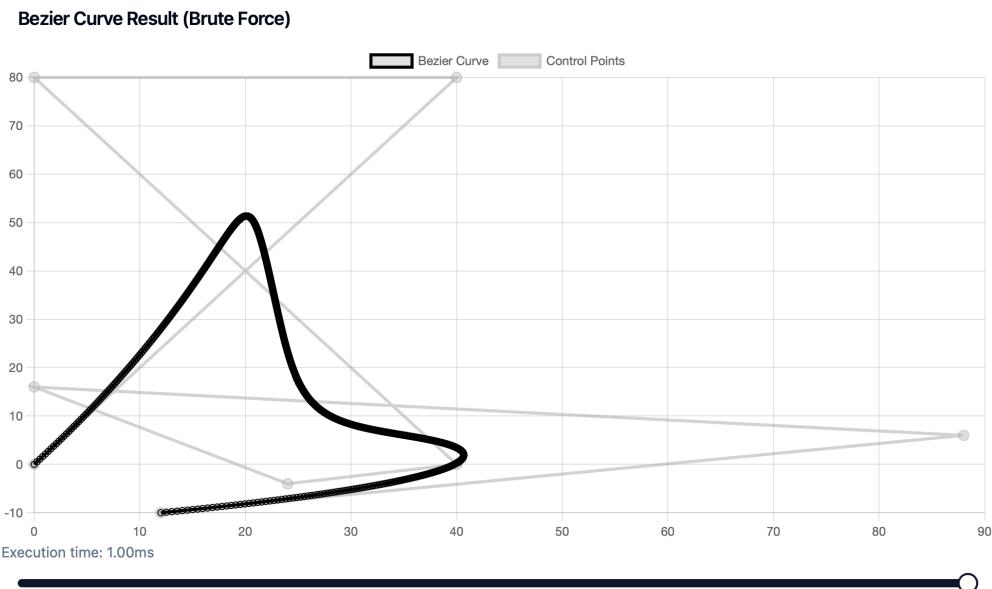
Dalam kasus ini, kita akan menginput sebanyak 8 titik dengan iterasi 10

**Bézier Curve**

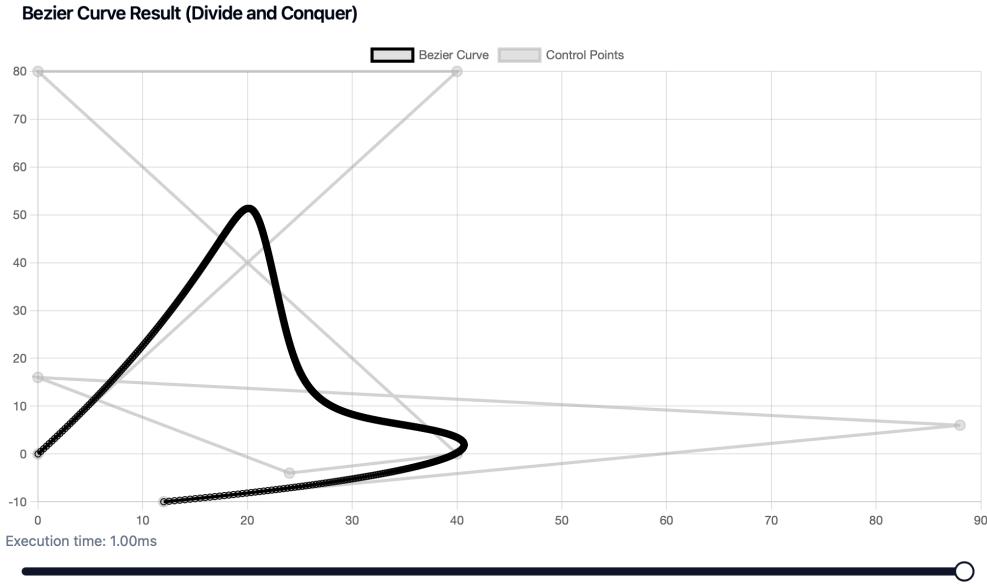
Input at least 3 points and the iteration to create  
bézier curve

1. x	0	y	0	Delete
2. x	40	y	80	Delete
3. x	0	y	80	Delete
4. x	40	y	0	Delete
5. x	24	y	-4	Delete
6. x	0	y	16	Delete
7. x	088	y	6	Delete
8. x	12	y	-10	Delete
10 <input style="background-color: black; color: white; border: none; padding: 2px 10px; border-radius: 5px;" type="button" value="Curve it!"/>				

**Gambar 4.8** Layar input kasus 2



**Gambar 4.9** Layar hasil kasus 2 dengan algoritma Brute Force



**Gambar 4.10** Layar hasil kasus 2 dengan algoritma Divide and Conquer

### 3. Kasus 3

Dalam kasus ini, kita akan menginput 4 titik

**Bézier Curve**

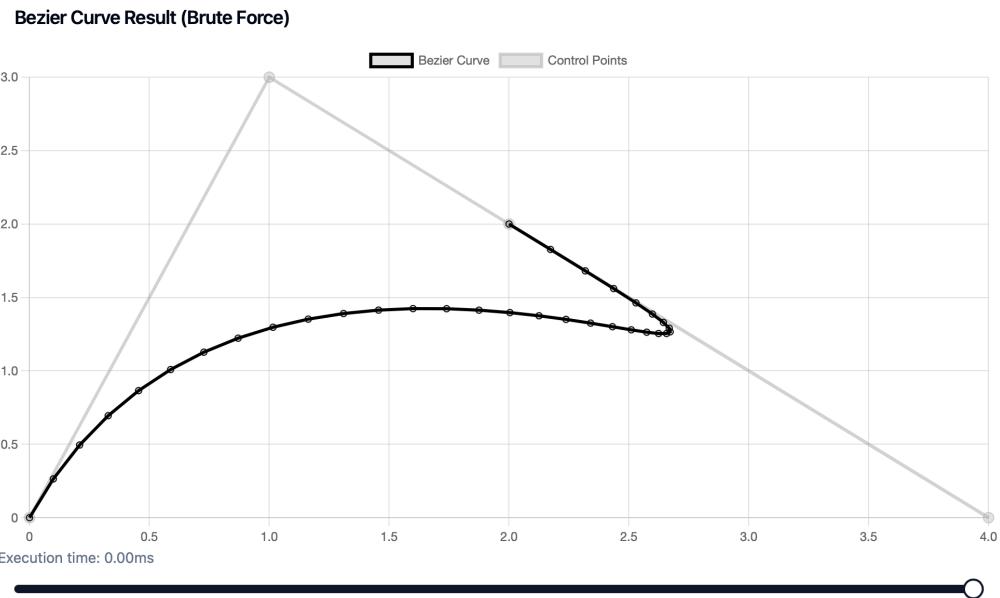
Input at least 3 points and the iteration to create  
bézier curve

1.	X	0	Y	0	Delete
2.	X	1	Y	3	Delete
3.	X	4	Y	0	Delete
4.	X	2	Y	2	Delete

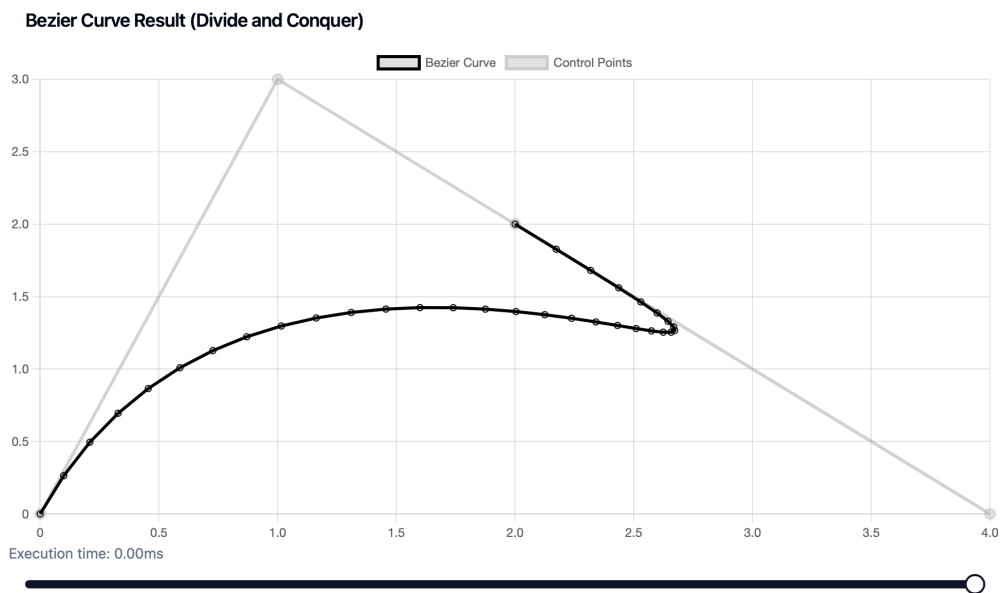
Add more

5

**Gambar 4.11** Layar input kasus 3



**Gambar 4.12** Layar hasil kasus 3 dengan algoritma Brute Force



**Gambar 4.13** Layar hasil kasus 3 dengan algoritma Divide and Conquer

#### 4. Kasus 4

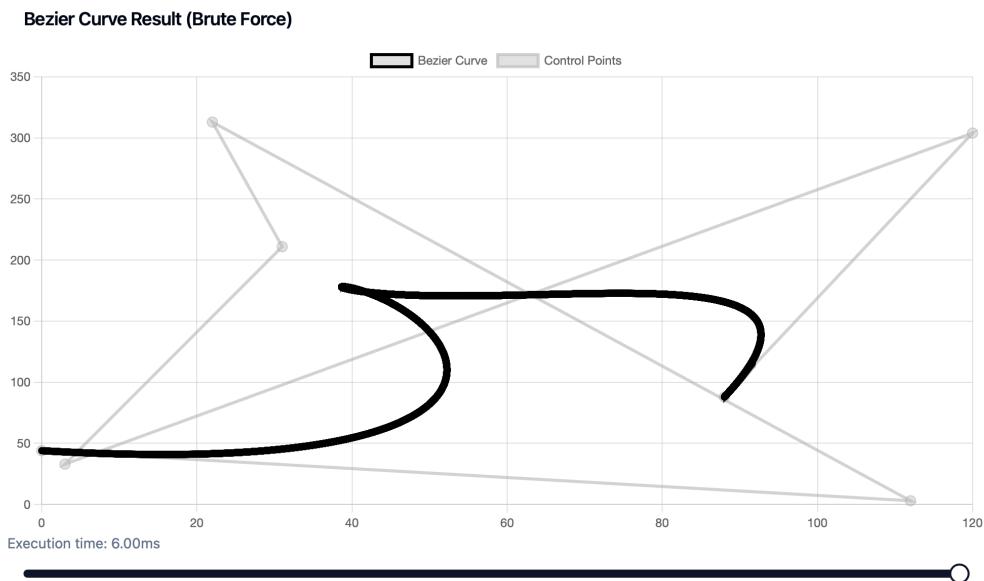
**Bézier Curve**

Input at least 3 points and the iteration to create  
bézier curve

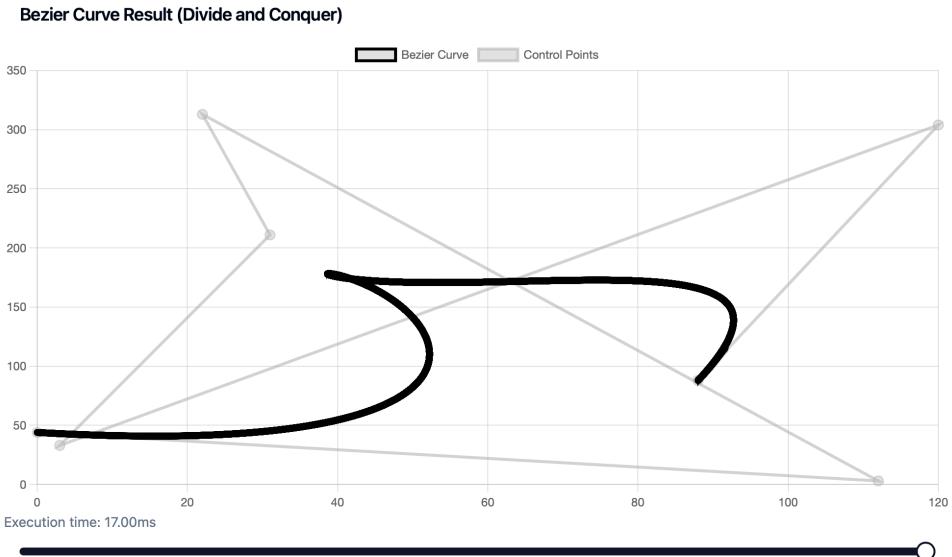
1. X	0	y	044	Delete
2. X	112	y	03	Delete
3. X	022	y	0313	Delete
4. X	031	y	211	Delete
5. X	03	y	33	Delete
6. X	120	y	304	Delete
7. X	088	y	88	Delete

Add more

**Gambar 4.14** Layar input kasus 4



**Gambar 4.15** Layar hasil kasus 4 dengan algoritma Brute Force



**Gambar 4.16** Layar hasil kasus 4 dengan algoritma Divide and Conquer

## 5. Kasus 5

### Bézier Curve

Input at least 3 points and the iteration to create  
bézier curve

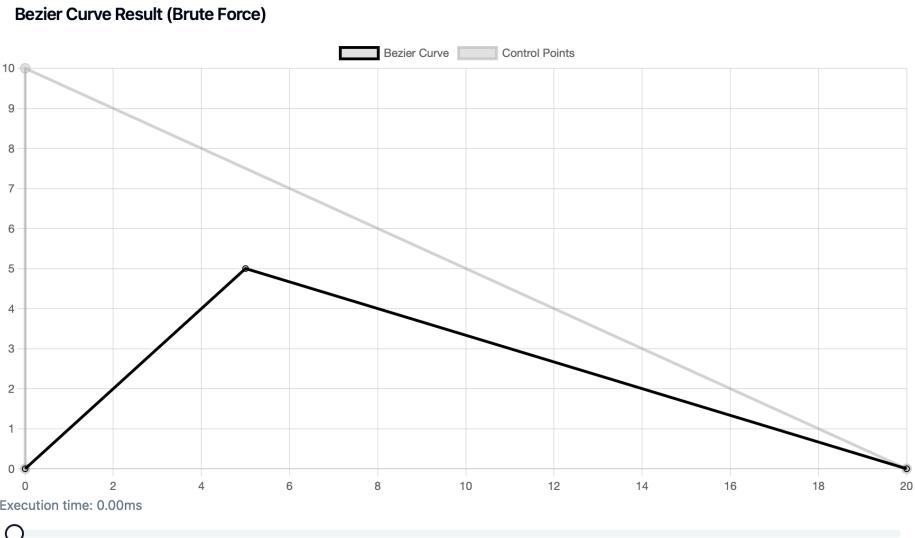
1. <b>x</b>	0	<input type="button" value="Delete"/>	<b>y</b>	0	<input type="button" value="Delete"/>	<input type="button" value="Delete"/>
-------------	---	---------------------------------------	----------	---	---------------------------------------	---------------------------------------

2. <b>x</b>	0	<input type="button" value="Delete"/>	<b>y</b>	10	<input type="button" value="Delete"/>	<input type="button" value="Delete"/>
-------------	---	---------------------------------------	----------	----	---------------------------------------	---------------------------------------

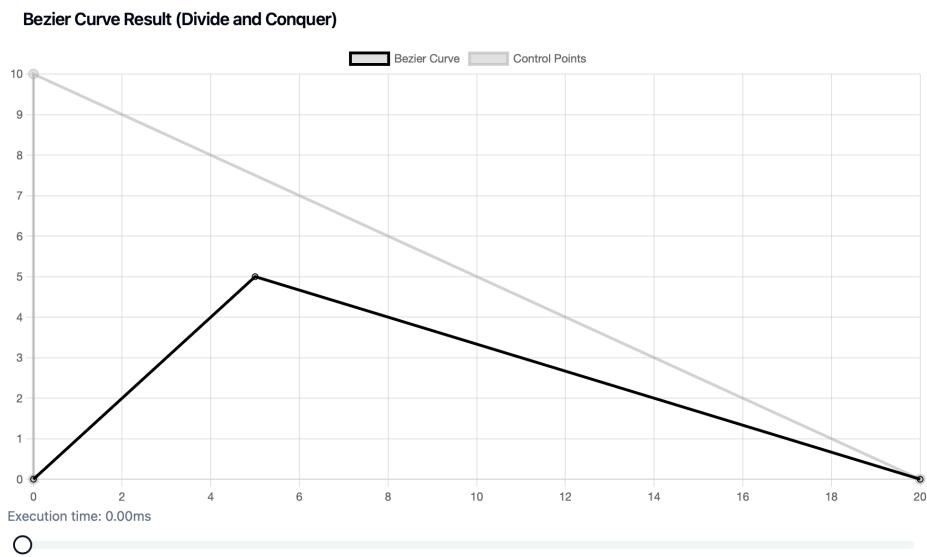
3. <b>x</b>	20	<input type="button" value="Delete"/>	<b>y</b>	0	<input type="button" value="Delete"/>	<input type="button" value="Delete"/>
-------------	----	---------------------------------------	----------	---	---------------------------------------	---------------------------------------

1	<input type="button" value="Curve it!"/>
---	--

**Gambar 4.17** Layar input kasus 5



**Gambar 4.18** Layar hasil kasus 5 dengan algoritma Brute Force



**Gambar 4.19** Layar hasil kasus 5 dengan algoritma Divide and Conquer

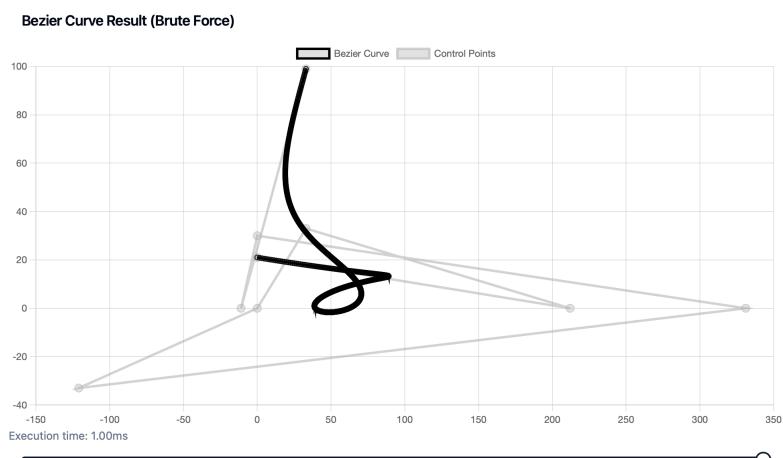
## 6. Kasus 6

## Bézier Curve

Input at least 3 points and the iteration to create  
bézier curve

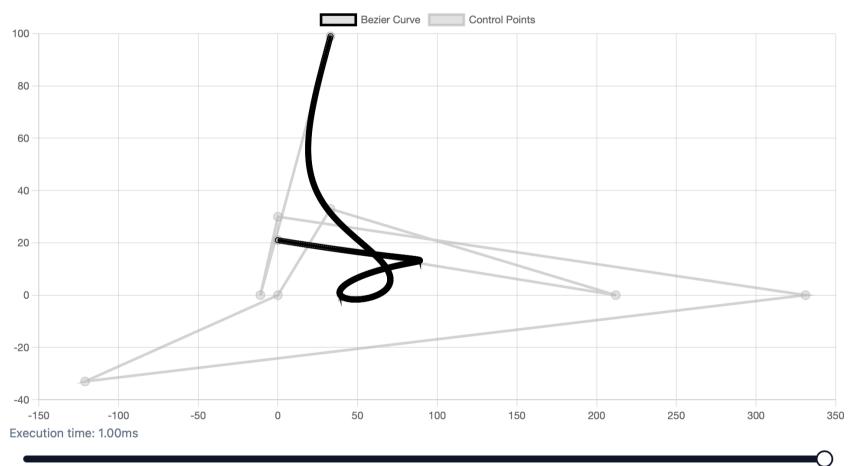
2.	x	212	↑ ↓	y	0	↑ ↓	Delete
3.	x	33	↑ ↓	y	33	↑ ↓	Delete
4.	x	0	↑ ↓	y	0	↑ ↓	Delete
5.	x	-121	↑ ↓	y	-33	↑ ↓	Delete
6.	x	331	↑ ↓	y	0	↑ ↓	Delete
7.	x	0	↑ ↓	y	30	↑ ↓	Delete
8.	x	-11	↑ ↓	y	0	↑ ↓	Delete
9.	x	33	↑ ↓	y	99	↑ ↓	Delete
		10	↑ ↓				<b>Curve it!</b>

**Gambar 4.20** Layar input kasus 6



**Gambar 4.21** Layar hasil kasus 6 dengan algoritma Brute Force

Bezier Curve Result (Divide and Conquer)



**Gambar 4.22** Layar hasil kasus 6 dengan algoritma Divide and Conquer

## BAB V

### Source Code

```
type Point = {
  x: number;
  y: number;
};

// Function for generate pascal triangle for Brute Force n
points

const generatePascalTriangle = (numRows: number) => {
  const triangle: number[][] = [];
  for (let i = 0; i < numRows; i++) {
    const row = [];
    for (let j = 0; j <= i; j++) {
      if (j === 0 || j === i) {
        row.push(1);
      } else {
        row.push(triangle[i - 1][j - 1] + triangle[i - 1][j]);
      }
    }
    triangle.push(row);
  }

  return triangle;
};

// Function for generate bezier curve for 3 points (using
// bezier quadratic formula)
const generateBezierCurve3Points = (
  controlPoints: Point[],
```

```
iterationLevel: number
) => {
  const points = [];
  const steps = Math.pow(2, iterationLevel) + 1;

  for (let i = 0; i <= steps - 1; i++) {
    const t = i / (steps - 1);
    const x =
      Math.pow(1 - t, 2) * controlPoints[0].x +
      2 * (1 - t) * t * controlPoints[1].x +
      Math.pow(t, 2) * controlPoints[2].x;
    const y =
      Math.pow(1 - t, 2) * controlPoints[0].y +
      2 * (1 - t) * t * controlPoints[1].y +
      Math.pow(t, 2) * controlPoints[2].y;
    points.push({ x, y });
  }

  return points;
};

// Function for generate bezier curve for n points (using
// pascal triangle)
const generateBezierCurveNPoints = (
  controlPoints: Point[],
  iterationLevel: number
) => {
  const points = [];
  const n = controlPoints.length - 1;
  const pascalTriangle = generatePascalTriangle(n + 1);
  const steps = Math.pow(2, iterationLevel) + 1;
```

```
for (let i = 0; i <= steps - 1; i++) {
    const point = { x: 0, y: 0 };
    const t = i / (steps - 1);
    for (let j = 0; j <= n; j++) {
        const b = pascalTriangle[n][j] * Math.pow(1 - t, n - j) *
Math.pow(t, j);
        point.x += b * controlPoints[j].x;
        point.y += b * controlPoints[j].y;
    }

    points.push(point);
}

return points;
};

// Fungsi untuk yang 3 titik doang
// Inggris-in chris biar sama semua
function divideConquerBezier3Points(
    control1: Point,
    control2: Point,
    control3: Point,
    banyakiterasi: number
) {
    const bezierPoints = [];
    bezierPoints.push(control1);
    isiTitikCurve(bezierPoints, control1, control2, control3, 0,
banyakiterasi);
    bezierPoints.push(control3);
    return bezierPoints;
}
```

```
}
```

```
function isiTitikCurve(
    bezierPoints: Point[],
    control1: Point,
    control2: Point,
    control3: Point,
    iterasi: number,
    banyakiterasi: number
) {
    if (iterasi < banyakiterasi) {
        const titikTengah1 = cariTitikTengah(control1, control2);
        const titikTengah2 = cariTitikTengah(control2, control3);
        const titikTengah3 = cariTitikTengah(titikTengah1,
titikTengah2);
        iterasi++;
        isiTitikCurve(
            bezierPoints,
            control1,
            titikTengah1,
            titikTengah3,
            iterasi,
            banyakiterasi
        );
        bezierPoints.push(titikTengah3);
        isiTitikCurve(
            bezierPoints,
            titikTengah3,
            titikTengah2,
            control3,
            iterasi,

```

```
        banyakIterasi

    );
}

}

function cariTitikTengah(controlPoint1: Point, controlPoint2: Point) {
    return {
        x: (controlPoint1.x + controlPoint2.x) / 2,
        y: (controlPoint1.y + controlPoint2.y) / 2,
    };
}

function divideAndConquerBezier(points: Point[], t: number) {
    if (points.length === 1) {
        return points[0];
    }

    const newPoints = [];
    for (let i = 0; i < points.length - 1; i++) {
        const x = (1 - t) * points[i].x + t * points[i + 1].x;
        const y = (1 - t) * points[i].y + t * points[i + 1].y;
        newPoints.push({ x, y });
    }

    return divideAndConquerBezier(newPoints, t);
}

const divideAndConquerBezierNPoints = (
    controlPoints: Point[],
    iteration: number
```

```
) => {

  const points = [];
  const stepSize = 1 / Math.pow(2, iteration);
  for (let t = 0; t <= 1; t += stepSize) {
    const point = divideAndConquerBezier(controlPoints, t);
    points.push({ x: point.x, y: point.y });
  }

  return points;
};
```

## **BAB VI**

### **Analisis Hasil**

#### **6.1. Analisis Hasil Eksperimen**

Pada sub bab ini akan dibahas mengenai analisis hasil dari penyelesaian menggunakan *Divide and Conquer* serta akan ada perbandingan antara algoritma *Divide and Conquer* dan algoritma *Brute Force* sebagai referensi. Parameter yang dijadikan perbandingan adalah kesamaan hasil penyelesaian serta waktu yang diperlukan untuk menemukan penyelesaian.

Untuk kasus 3 titik kontrol, berikut adalah hasil eksperimen,

Kasus Input Output	<i>Divide and Conquer</i> (ms)	<i>Brute Force</i> (ms)	Analisis
Kasus 1	129	44	Karena hanya terdapat 4 titik kontrol, algoritma <i>Divide and Conquer</i> yang lebih kompleks membuatnya menjadi lebih lambat ketimbang <i>Brute Force</i> .
Kasus 2	1	1	Dikarenakan jumlah iterasi yang cukup rendah, kedua algoritma memiliki kecepatan eksekusi program yang sama. Jumlah iterasi tentunya akan mempengaruhi kompleksitas program dan hal ini akan lebih dibahas pada sub bab 6.2

Kasus 3	0	0	Seperti halnya kasus 2, karena jumlah iterasi yang sangat kecil dan jumlah titik kontrol yang kecil juga, maka kedua algoritma bekerja dengan sangat cepat.
Kasus 4	17	6	Dikarenakan titik kontrol yang sedikit, seperti halnya kasus 1, algoritma <i>Brute Force</i> lebih diuntungkan dan memiliki performa yang lebih baik ketimbang <i>Divide and Conquer</i> .

## 6.2. Perbandingan Kompleksitas Algoritma dengan Brute Force

Pada sub bab ini akan dianalisis kompleksitas waktu dan ruang untuk algoritma *Divide and Conquer* serta algoritma *Brute Force* untuk melihat lebih lanjut alasan dari hasil-hasil yang terlihat.

### A. Algoritma *Brute Force* :

Kompleksitas waktu : Untuk 3 titik kontrol, kompleksitas waktu dari fungsi generateBezierCurve3Points adalah  $O(2^N)$  dengan N adalah banyak iterasi karena jumlah langkah atau steps dalam loop for ditentukan oleh  $\text{Math.pow}(2, \text{iterationLevel}) + 1$ . Loop for ini berjalan sebanyak steps kali, dan dalam setiap iterasi, ia melakukan sejumlah pekerjaan yang konstan: menghitung t, x, dan y, dan kemudian menambahkan titik baru ke array points. Setiap operasi ini adalah  $O(1)$ , atau konstan. Jadi, kompleksitas waktu dari fungsi ini didominasi oleh loop, membuatnya  $O(\text{steps})$ . Jika kita menganggap banyak iterasi sebagai ukuran input N, maka kompleksitas waktu adalah  $O(2^N)$ , karena steps

sebanding dengan 2 pangkat banyak iterasi. Sedangkan, untuk titik kontrol yang lebih dari 3 point, *Brute Force* memiliki kompleksitas waktu  $O(m * 2^N)$ , di mana m adalah jumlah titik kontrol. Ini karena fungsi ini memiliki loop luar yang berjalan steps kali (yang sebanding dengan  $2^N$ ), dan dalam setiap iterasi, ia melakukan sejumlah pekerjaan yang sebanding dengan m (jumlah titik kontrol)

B. Algoritma *Divide and Conquer* :

Kompleksitas waktu : Untuk 3 titik kontrol, kompleksitas waktu dari fungsi divideConquerBezier3Points dan isiTitikCurve adalah  $O(2^N)$  dengan N adalah banyak iterasi dikarenakan fungsi divideConquerBezier3Points memanggil fungsi isiTitikCurve yang merupakan fungsi rekursif. Fungsi isiTitikCurve membagi titik kontrol menjadi pasangan dan menghitung titik baru, dan kemudian memanggil dirinya sendiri dua kali dengan titik-titik baru ini. Jumlah total operasi yang dilakukan oleh fungsi ini adalah sebanding dengan  $2^N$  karena setiap panggilan rekursif menggandakan jumlah operasi. Oleh karena itu, kompleksitas waktu dari fungsi divideConquerBezier3Points adalah  $O(2^N)$ , karena untuk setiap langkah, ia memanggil fungsi isiTitikCurve yang memiliki kompleksitas waktu  $O(2^N)$ .

Untuk N titik kontrol, Kompleksitas waktu untuk fungsi divideAndConquerBezier dan divideAndConquerBezierNPoints ini adalah  $O(m * 2^N)$ , di mana m adalah jumlah titik kontrol. Hal ini didapat karena fungsi divideAndConquerBezier adalah fungsi rekursif yang membagi titik kontrol menjadi pasangan dan menghitung titik baru. Jumlah total operasi yang dilakukan oleh fungsi ini adalah sebanding dengan n, jumlah titik kontrol, karena setiap panggilan rekursif mengurangi jumlah titik kontrol sebanyak 1. Fungsi divideAndConquerBezierNPoints memiliki loop yang berjalan  $1 / \text{stepSize}$  kali, yang sebanding dengan  $2^N$  karena  $\text{stepSize} = 1 / \text{Math.pow}(2, \text{iteration})$ . Dalam setiap iterasi, ia memanggil fungsi divideAndConquerBezier yang memiliki kompleksitas waktu  $O(n)$ .

### 6.3. Pembahasan Mengenai Perbedaan Hasil Algoritma

Meskipun kedua algoritma memiliki kompleksitas yang sama, hasil algoritma berupa titik yang sama, kita dapat melihat dari hasil-hasil output bahwa *Brute Force* lebih cepat dalam menunjukkan kurva Bezier. Hal ini tidak dapat kita simpulkan secara langsung dikarenakan secara teori yang diajarkan seharusnya *Divide and Conquer* adalah algoritma yang lebih baik tetapi dalam kasus ini sebaliknya. Kalau kita lihat dari sisi lain, mungkin faktor perbedaan itu terdapat pada beberapa faktor eksternal. Dalam kasus ini pendekatan *Brute Force* menghitung titik kurva Bezier secara langsung menggunakan rumus matematika. Ini adalah penghitungan langsung yang dapat dilakukan dalam sekali jalan, sehingga relatif cepat. Di sisi lain, pendekatan membagi dan menaklukkan secara rekursif memecah masalah menjadi bagian-bagian yang lebih kecil dan menyelesaikan setiap bagian secara individual. Hal ini melibatkan lebih banyak pemanggilan fungsi dan logika yang lebih kompleks, yang dapat memperlambat eksekusi. Terlebih lagi *Brute Force* melalui iterasi yang diulang sedangkan *Divide and Conquer* menggunakan rekursif dimana kalau kita kutip dalam <https://stackoverflow.com/a/28033212>, kita bisa mengasumsi kalau perulangan iterasi lebih cepat dibanding rekursif.

Hasil algoritma *Brute Force* yang lebih cepat juga bisa terjadi karena titik kontrol yang terbilang cukup rendah dalam pengujian input output. Secara teori, algoritma *Divide and Conquer* akan berjalan lebih cepat dengan  $N$  titik kontrol yang mencapai jumlah sangat besar dikarenakan rumus matematika yang dipakai *Brute Force* akan menjadi sangat kompleks yang membuat komputasi program tidak mungkin tercapai.

## **BAB VII**

### **Implementasi Bonus**

Pada tucil 2 ini, terdapat bonus yang dapat diimplementasikan yakni diharuskan “Melakukan generalisasi algoritma, ide, serta melakukan implementasinya sehingga program dapat membentuk kurva Bézier kubik, kuartik, dan selanjutnya dengan 4, 5, 6, hingga n titik kontrol.” Untuk menangani permasalahan ini, diperlukan pemahaman mendalam tentang algoritma *Divide and Conquer*, terutama saat jumlah titik kontrol melebihi 3. Dengan kompleksitas yang meningkat seiring peningkatan jumlah titik kontrol, pendekatan yang digunakan adalah dengan membagi titik kontrol menjadi segmen-segmen yang lebih kecil, lalu memecahkan sub persoalan tersebut. Dengan demikian, kita dapat melakukan generalisasi untuk membentuk kurva Bézier dengan jumlah titik kontrol yang bervariasi. Oleh karena itu, kami mengimplementasikan algoritma De Casteljau sebagai pendekatan *Divide and Conquer*.

Algoritma De Casteljau dipilih karena memiliki keunggulan dalam menangani kurva Bézier dengan jumlah titik kontrol yang dinamis. Algoritma ini memecah kurva menjadi segmen-segmen yang lebih kecil dengan pendekatan linier, yang memungkinkan fleksibilitas dalam menangani berbagai jumlah titik kontrol. Selain itu, algoritma De Casteljau dapat diterapkan secara rekursif, sehingga cocok untuk pemecahan masalah secara *Divide and Conquer*. Dengan menggunakan pendekatan ini, program dapat dengan efisien membentuk kurva Bézier dengan jumlah titik kontrol yang berapapun, meningkatkan kemampuan untuk menggeneralisasi algoritma dan ide di dalam implementasi kurva Bézier.

Berikut langkah-langkah implementasi pada program bonus kami :

1. Fungsi `divideAndConquerBezier` menerima dua argumen: array `points` yang berisi titik-titik kontrol dan nilai `t` yang merupakan parameter kurva Bezier.
2. Jika array `points` hanya berisi satu titik, fungsi tersebut akan mengembalikan titik tersebut. Ini adalah kasus dasar untuk rekursi.

3. Jika ada lebih dari satu titik, fungsi tersebut akan menghitung titik-titik baru berdasarkan titik-titik kontrol saat ini. Titik baru dihitung dengan menginterpolasi antara setiap pasangan titik kontrol berturut-turut berdasarkan parameter  $t$
4. Fungsi `divideAndConquerBezier` kemudian memanggil dirinya sendiri secara rekursif dengan array titik baru dan parameter  $t$  yang sama, dan mengembalikan hasilnya. Ini adalah langkah divide dan conquer: masalah asli (menghitung titik pada kurva Bezier) dibagi menjadi submasalah yang lebih kecil (menghitung titik pada kurva Bezier yang lebih sederhana).
5. Fungsi `divideAndConquerBezierNPoints` menerima array `controlPoints` yang berisi titik-titik kontrol dan nilai `iteration` yang menentukan jumlah titik yang akan dihasilkan pada kurva Bezier.
6. Fungsi ini menghitung `stepSize`, yang merupakan jarak antara titik-titik pada kurva Bezier, dan kemudian menghasilkan titik-titik tersebut dengan memanggil `divideAndConquerBezier` untuk setiap nilai  $t$  dari 0 hingga 1 dengan langkah `stepSize`. `stepSize` itu sendiri memanfaatkan nilai iterasi dengan  $\text{stepSize} = 1 / \text{iteration}^2$
7. Akhirnya, fungsi `divideAndConquerBezierNPoints` mengembalikan array `points` yang berisi titik-titik pada kurva Bezier.

Berikut adalah implementasi *Divide and Conquer* dengan N titik kontrol dalam source code kami:

Nama fungsi / prosedur	Deskripsi
<code>divideAndConquerBezier</code>	Fungsi ini mengimplementasikan algoritma De Casteljau untuk menghitung titik pada kurva Bezier pada parameter $t$ yang diberikan.

Nama fungsi / prosedur	Deskripsi
<code>divideAndConquerBezierNPoints</code>	Melakukan iterasi untuk setiap parameter $t$ dari t sampai 1 dan memanggil <code>divideAndConquerBezier</code> untuk setiap iterasi

## **BAB VIII**

### **Daftar Pustaka dan Link Repository**

Berikut adalah daftar referensi yang dipakai dalam pengerajan tugas kecil ini.

1. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/stima23-24.htm>  
(Algoritma Divide and Conquer Bagian 1 - 4) (Diakses pada 14/03/2024)
2. <https://docs.google.com/document/d/161qTQR5PzjQUIsoLO00A0Rp1dvsahrXY2Dk-fSmJl2o/edit> (Spesifikasi Tucil 2 Strategi Algoritma) (Diakses pada 13/03/2024)
3. <https://javascript.info/bezier-curve> (Diakses pada 16/03/2024)
4. [Understanding Bézier Curves. A mathematical and intuitive approach | by Mateus Melo | Medium](https://medium.com/@mateusmelo/understanding-b%C3%A9zier-curves-a-mathematical-and-intuitive-approach-13522125_13522135) (Diakses pada 17/03/2024)

Link Repository GitHub : [https://github.com/satriadhikara/Tucil2\\_13522125\\_13522135](https://github.com/satriadhikara/Tucil2_13522125_13522135)

## **BAB IX**

### **Lampiran**

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		✓