

# LAPORAN TUGAS KECIL 1

## IF2211 STRATEGI ALGORITMA

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



Disusun oleh:

Marvin Scifo Y. Hutahaean      13522110

Naufal Adnan                        13522116

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

**2024**

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>1</b>
<b>BAB I.....</b>	<b>2</b>
1.1 Kurva Bézier.....	2
1.2 Algoritma Brute Force dalam Pembentukan Kurva Bézier.....	3
<b>BAB II.....</b>	<b>5</b>
2.1 Algoritma Divide and Conquer.....	5
2.2 Algoritma Divide and Conquer dalam Pembentukan Kurva Bézier.....	5
<b>BAB III.....</b>	<b>7</b>
3.1 Implementasi Algoritma Brute Force.....	7
3.2 Implementasi Algoritma Divide and Conquer.....	8
<b>BAB IV.....</b>	<b>9</b>
4.1 Test 1.....	10
4.2 Test 2.....	11
4.3 Test 3.....	14
4.4 Test 4.....	15
4.5 Test 5.....	17
4.6 Test 6.....	20
<b>BAB V.....</b>	<b>22</b>
4.1 Analisis Penggunaan Algoritma Brute Force dalam Pembentukan Kurva Bézier.....	22
4.2 Analisis Penggunaan Algoritma Divide and Conquer dalam Pembentukan Kurva Bézier.....	22
4.3 Perbandingan Penggunaan Algoritma Brute Force dan Algoritma Divide and Conquer.....	23
<b>BAB VI.....</b>	<b>24</b>
5.1 Implementasi Kurva Bézier n Buah Titik Kontrol.....	24
5.2 Implementasi GUI.....	24
<b>DAFTAR PUSTAKA.....</b>	<b>26</b>
<b>LAMPIRAN.....</b>	<b>26</b>
Link Repository.....	26
Check List.....	26

## BAB I

### Kurva Bézier dan Algoritma Brute Force

#### 1.1 Kurva Bézier

Kurva Bézier adalah kurva halus yang dibuat dengan menghubungkan beberapa titik kontrol yang menentukan bentuk dan arah kurva. 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.

Cara membuat Kurva Bézier pun cukup mudah. 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, yang disebut dengan kurva Bézier linier. Pada kurva yang terbentuk terdapat sebuah titik  $Q_0$  yang berada pada garis yang dibentuk oleh  $P_0$  dan  $P_1$ , yang posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

Jika 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 di antara 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.

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

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

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

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

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

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.

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

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

## 1.2 Algoritma Brute Force dalam Pembentukan Kurva Bézier

Persoalan pembentukan kurva Bézier secara lempang dapat diselesaikan dengan algoritma brute force. Algoritma brute force dalam pembentukan kurva Bézier dilakukan dengan tujuan untuk menemukan serangkaian titik pada kurva Bézier dengan mengevaluasi posisi titik-titik pada kurva untuk setiap nilai parameter  $t$  dalam rentang  $[0, 1]$ . Dalam pendekatan brute force, akan dicari titik-titik pada kurva Bézier dengan melakukan perhitungan langsung pada setiap iterasi untuk setiap nilai parameter  $t$  di dalam rentang yang ditentukan. Langkah-langkah dari algoritma brute force yang diimplementasikan adalah sebagai berikut.

1. Inisialisasi nilai parameter  $t$  dengan nilai 0, dan sebuah array hasil yang akan menjadi tempat untuk menampung titik-titik hasil iterasi penentuan titik-titik pada kurva Bézier.
2. Ketika nilai  $t$  masih berada dalam rentang  $[0, 1]$ , akan dicari nilai sebuah titik pada kurva Bézier. Pencarian dilakukan dengan langkah sebagai berikut.
  - a. Ambil dua titik kontrol berturut-turut.
  - b. Untuk setiap pasangan titik kontrol yang berdekatan, hitung koordinat x dan y yang baru dengan formula

$$\text{newX} = t(P_2.X - P_1.X) + P_1.X$$

$$\text{newY} = t(P_2.Y - P_1.Y) + P_1.Y$$

- c. Setelah semua titik yang berdekatan terhitung, untuk sejumlah  $n$  buah titik akan diperoleh  $n - 1$  buah titik.
- d. Lakukan proses a, b, dan c berulang-ulang secara urut hingga tersisa satu buah titik, yaitu titik pada kurva Bézier.

Pada proses ini intinya adalah mencari sebuah titik pada kurva Bézier dengan parameter  $t$  menurut formula berikut.

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

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

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

$R_0$  untuk 2 buah titik,  $S_0$  untuk 3 buah titik,  $T_0$  untuk 4 buah titik, dan seterusnya dengan pola yang sama.

3. Simpan titik tersebut pada array hasil setelah sebuah titik pada kurva Bézier untuk nilai parameter  $t$  tertentu diperoleh.

4. Selanjutnya nilai parameter  $t$  akan memperoleh *increment* sebesar  $0.5^i$ , dengan  $i$  adalah jumlah iterasi yang akan dilakukan untuk mencari titik pada kurva Bézier.
5. Kemudian proses akan terus berulang kembali dari langkah 2. Proses akan berhenti ketika nilai parameter  $t$  adalah 1, di mana pada kondisi ini titik pada kurva Bézier yang diperoleh adalah  $P_n$  atau titik akhir.

Pada akhir proses akan diperoleh sekumpulan titik-titik pada kurva Bézier yang selanjutnya jika dihubungkan secara berurutan akan tergambaran kurva Bézier yang terbentuk. Kehalusan dari kurva yang terbentuk bergantung pada banyaknya iterasi yang dilakukan. Semakin banyak iterasi, maka kurva Bézier yang terbentuk akan semakin halus.

## BAB II

### Algoritma Divide and Conquer pada Kurva Bézier

#### 2.1 Algoritma Divide and Conquer

Algoritma divide and conquer merupakan pendekatan yang digunakan dalam pemecahan masalah yang umumnya dengan langkah divide, conquer, kemudian combine. Divide artinya membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil. Idealnya setiap upa-persoalan berukuran hampir sama. Conquer artinya menyelesaikan (*solve*) setiap upa-persoalan. Penyelesaian persoalan bisa dilakukan secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar. Kemudian combine yang artinya menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

Algoritma ini cocok digunakan untuk menyelesaikan masalah yang dapat dibagi menjadi beberapa bagian kecil dengan karakteristik yang sama. Meskipun pada awalnya mungkin terlihat rumit, penggunaan algoritma divide and conquer memungkinkan penyelesaian masalah yang lebih efisien karena memecah masalah menjadi bagian-bagian yang lebih kecil dan lebih mudah dipecahkan. Algoritma divide and conquer memungkinkan pemecahan masalah yang lebih efektif dan efisien dalam berbagai konteks, seperti pemrosesan paralel, pengurutan, dan pencarian. Dengan demikian kompleksitas waktu untuk algoritma divide and conquer umumnya adalah sebagai berikut

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

dengan

- $T(n)$  : kompleksitas waktu penyelesaian persoalan P yang berukuran n
- $g(n)$  : kompleksitas waktu untuk SOLVE jika n sudah berukuran kecil
- $T(n_1) + T(n_2) + \dots + T(n_r)$  : kompleksitas waktu untuk memproses setiap upa-persoalan
- $f(n)$  : kompleksitas waktu untuk COMBINE solusi dari masing-masing upa-persoalan
- Tahap DIVIDE dapat dilakukan dalam  $O(1)$ , sehingga tidak dimasukkan ke dalam formula

#### 2.2 Algoritma Divide and Conquer dalam Pembentukan Kurva Bézier

Algoritma divide and conquer dapat digunakan dalam pembentukan kurva Bézier. Ide untuk menemukan titik-titik pada kurva Bézier dapat diperoleh dengan membagi persoalan ke dalam elemen algoritma divide and conquer sebagai berikut.

## 1. Divide

Persoalan dalam mencari titik yang membentuk kurva Bézier dapat dibagi menjadi upa-persoalan yang lebih kecil. Hal tersebut terjadi jika nilai iterasi yang dilakukan belum terpenuhi, artinya masih diinginkan untuk melakukan iterasi kembali dalam mencari titik pada kurva Bézier. Pembagian dilakukan dengan langkah sebagai berikut.

- a. Tentukan masing-masing daftar titik tengah dari setiap titik yang berurutan, titik ini akan menjadi titik kontrol baru.
- b. Bagi titik-titik tersebut menjadi dua segmen, segmen pertama akan menggunakan sebuah titik awal ( $P_0$ ) dan setengah titik-titik tengah awal, sedangkan segmen yang kedua akan menggunakan setengah titik-titik tengah akhir dan sebuah titik akhir ( $P_n$ ).
- c. Selesaikan secara rekursif upa-persoalan yang telah dibagi menjadi dua segmen tersebut.

## 2. Conquer

Pada setiap langkah rekursi, setiap upa-persoalan akan diselesaikan secara terpisah. Jika kondisi iterasi telah memenuhi jumlah yang diinginkan (maksimum), maka akan dilakukan langkah penyelesaian (*solve*). Pada kondisi ini, untuk setiap upa-persoalan akan dilakukan pencarian titik tengah menggunakan titik-titik kontrol terakhir. Kurva Bézier dari titik kontrol terakhir dan titik tengah terakhir ini kemudian akan dikonstruksikan nantinya.

## 3. Combine

Setelah menyelesaikan upa-persoalan dari langkah conquer, maka titik-titik pada kurva Bézier diperoleh. Titik-titik pada kurva Bézier yang diperoleh ini akan dikumpulkan dalam sebuah array hasil yang akan menjadi tempat untuk menampung titik-titik hasil iterasi penentuan titik-titik pada kurva Bézier. Titik-titik kontrol yang diperoleh dari rekursi akan digabungkan dengan titik-titik awal dan akhir pada kurva untuk membentuk kurva Bézier lengkap.

## BAB III

### Implementasi

#### 3.1 Implementasi Algoritma Brute Force

```

func findpoint(P []Point, t float64) []Point {
    var temp []Point = P
    numPoints := len(P)
    for numPoints > 1 {
        newP := make([]Point, numPoints-1)
        for i := 0; i < numPoints-1; i++ {
            newP[i].X = t*(temp[i+1].X-temp[i].X) + temp[i].X
            newP[i].Y = t*(temp[i+1].Y-temp[i].Y) + temp[i].Y
        }
        temp = newP
        numPoints--
    }
    return temp
}

func TraceCurve(input PointData_Numeric) []Point {
    var result []Point
    add := 0.5 * math.Pow(0.5, float64(input.IterationValue-1))
    t := 0.0

    for t <= 1 {
        fmt.Println(t)
        result = append(result, findpoint(input.Points, t)...)
        t = t + add
    }

    return result
}

```

Gambar 3.1.1 Algoritma Brute Force dalam Pembentukan Kurva Bézier

### 3.2 Implementasi Algoritma Divide and Conquer

```

func Find_Mid_Point(x Point, y Point) Point {
    var mid_point Point
    mid_point.X = (y.X + x.X) / 2
    mid_point.Y = (y.Y + x.Y) / 2
    return mid_point
}

func Construct_List_Mid_Point(x [][]Point) [][]Point {
    temp_point := make([][]Point, len(x)-1)
    for i := 0; i < len(x)-1; i++ {
        for j := 0; j < len(x)-i-1; j++ {
            if i == 0 {
                temp_point[i] = append(temp_point[i], Find_Mid_Point(x[j], x[j+1]))
            } else {
                temp_point[i] = append(temp_point[i], Find_Mid_Point(temp_point[i-1][j], temp_point[i-1][j+1]))
            }
        }
    }
    return temp_point
}

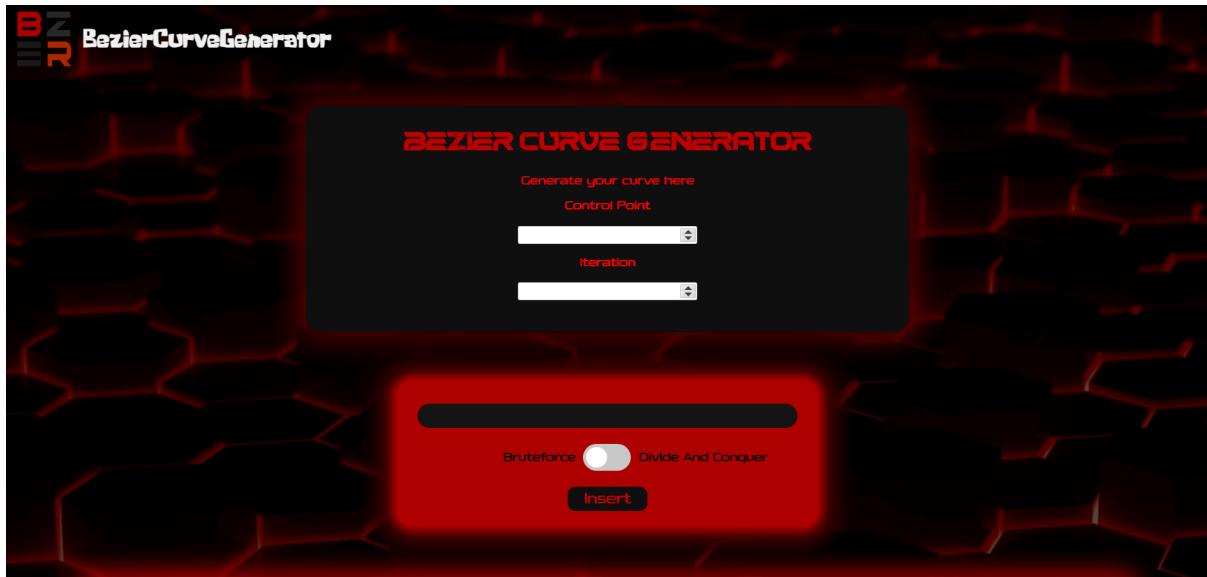
func Bezier_Line(x PointData_Numeric, iteration_counter int) {
    if iteration_counter == x.IterationValue {
        var constructed_point [][]Point = Construct_List_Mid_Point(x.Points)
        PointsGlobale = append(PointsGlobale, x.Points[0])
        PointsGlobale = append(PointsGlobale, constructed_point[len(constructed_point)-1][0])
    } else {
        var constructed_point [][]Point = Construct_List_Mid_Point(x.Points)
        var point_1, point_2 []Point
        point_1 = append(point_1, x.Points[0])
        for i := 0; i < x.ControlValue; i++ {
            point_1 = append(point_1, constructed_point[i][0])
        }
        for i := 0; i < x.ControlValue; i++ {
            point_2 = append(point_2, constructed_point[x.ControlValue-1-i][i])
        }
        point_2 = append(point_2, x.Points[len(x.Points)-1])
        var x_1 PointData_Numeric
        var x_2 PointData_Numeric
        x_1.ControlValue = x.ControlValue
        x_2.ControlValue = x.ControlValue
        x_1.IterationValue, x_2.IterationValue = x.IterationValue, x.IterationValue
        x_1.Points = point_1
        x_2.Points = point_2
        Bezier_Line(x_1, iteration_counter+1)
        Bezier_Line(x_2, iteration_counter+1)
    }
}

```

Gambar 3.2.1 Algoritma Divide and Conquer dalam Pembentukan Kurva Bézier

## BAB IV

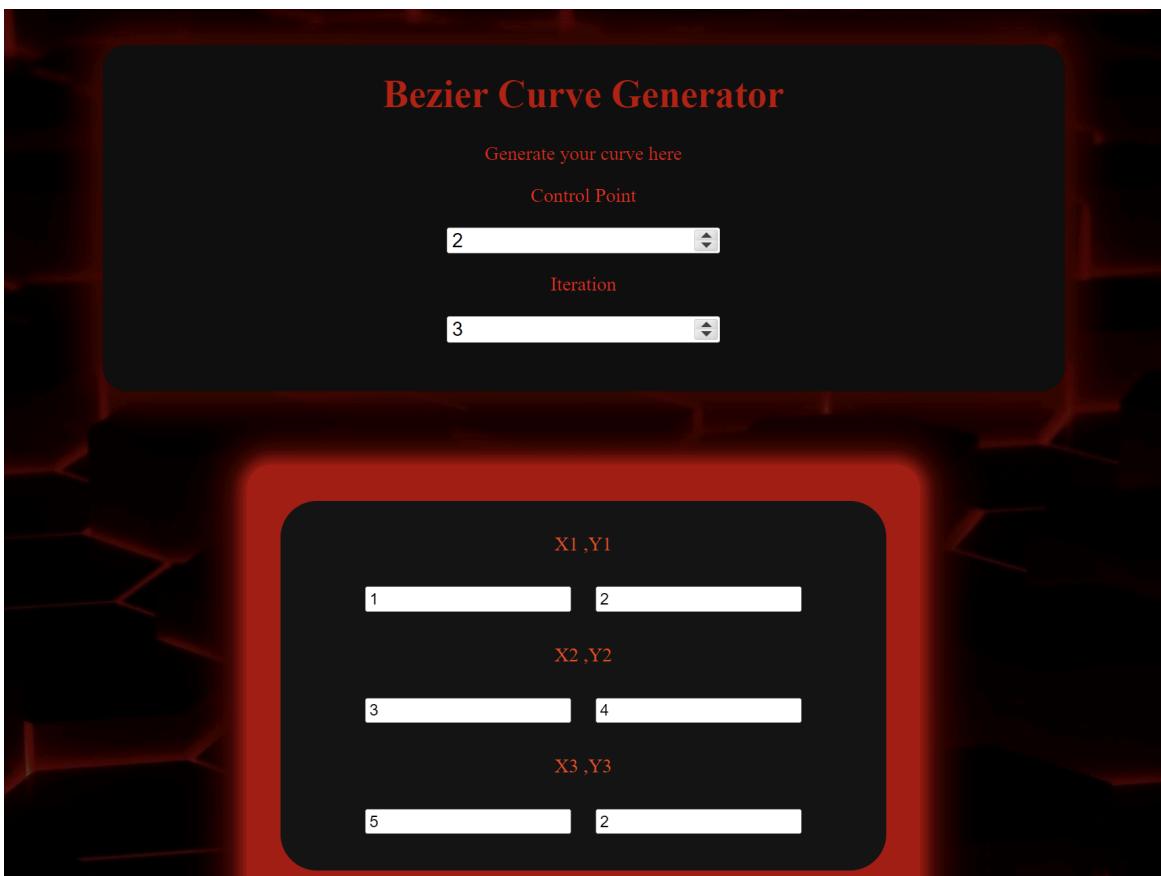
### Eksperimen

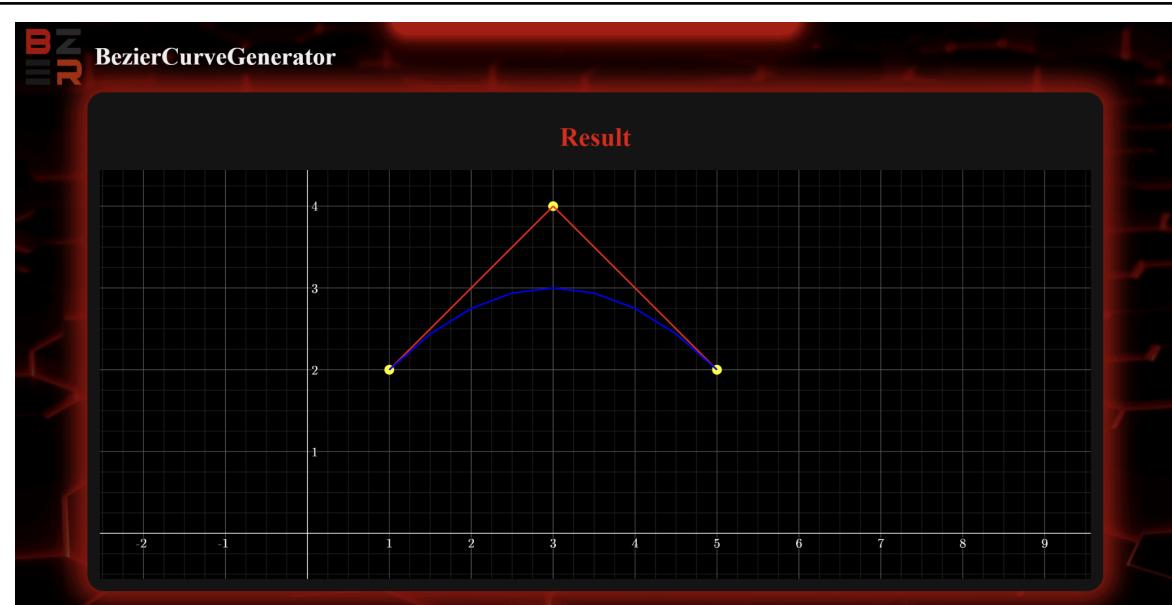


Gambar 4.1 Tampilan Utama GUI

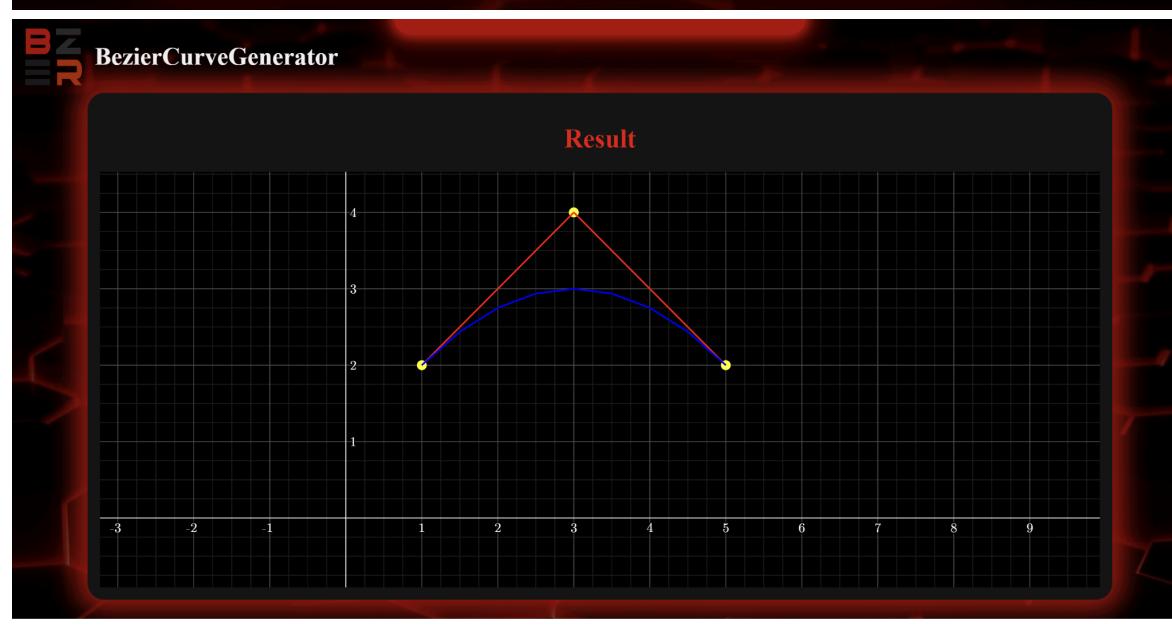
## 4.1 Test 1

Tabel 4.1 Test 1

Input

Output
<p>Output algoritma brute force:</p>  <p>Bruteforce <input checked="" type="checkbox"/> Divide And Conquer</p> <p>Execution Time: 2 ms.</p> <p>Insert</p>

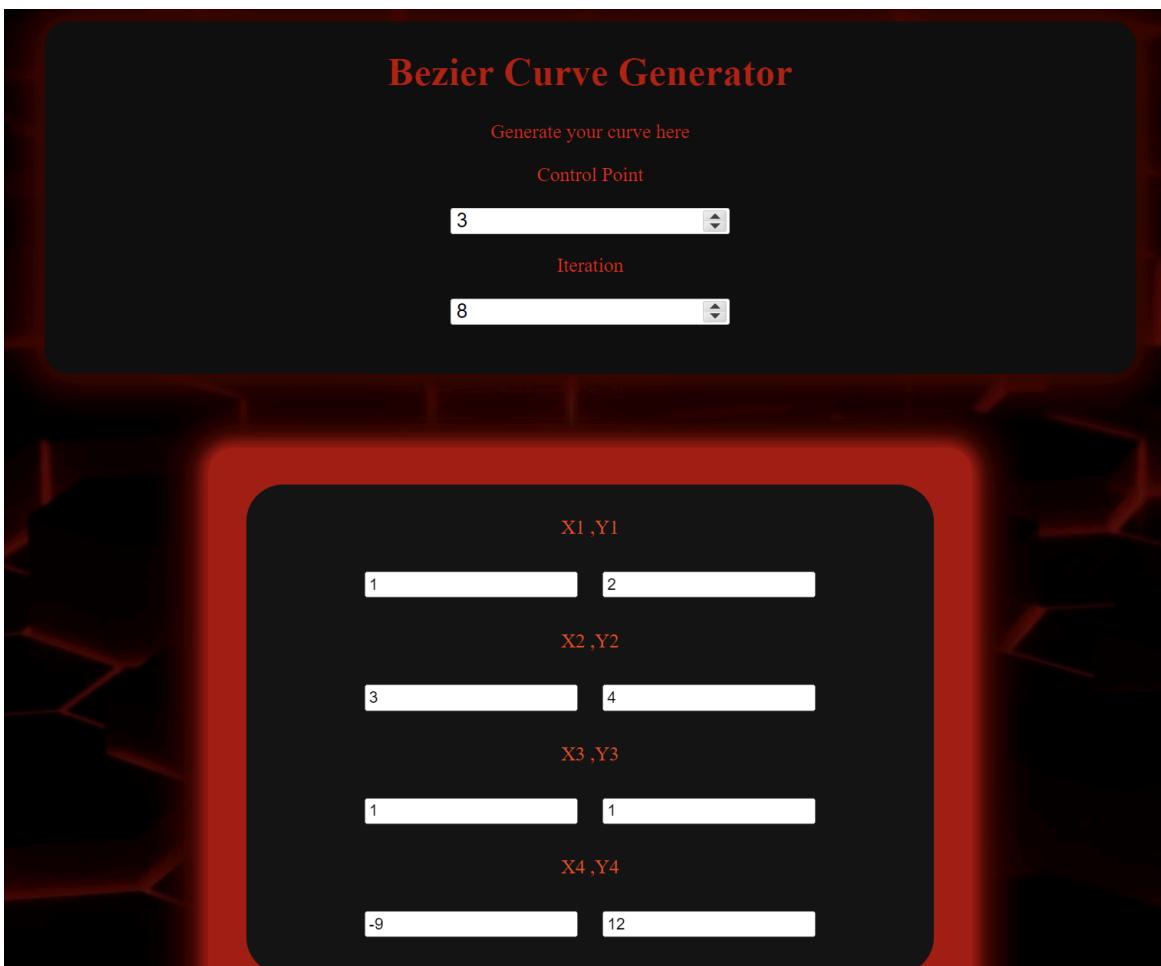
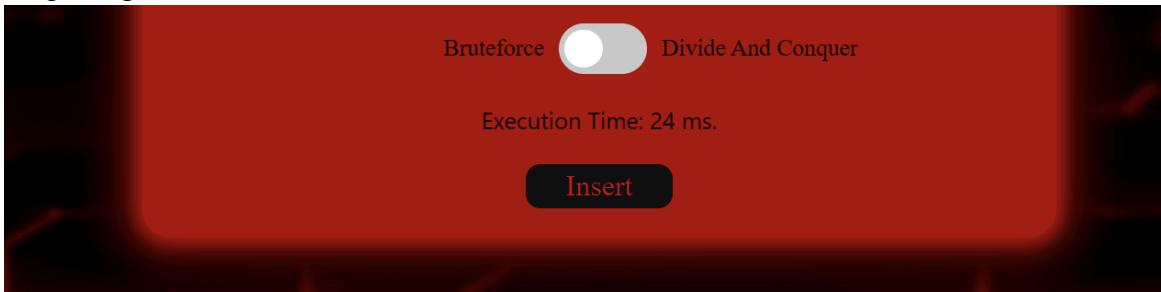


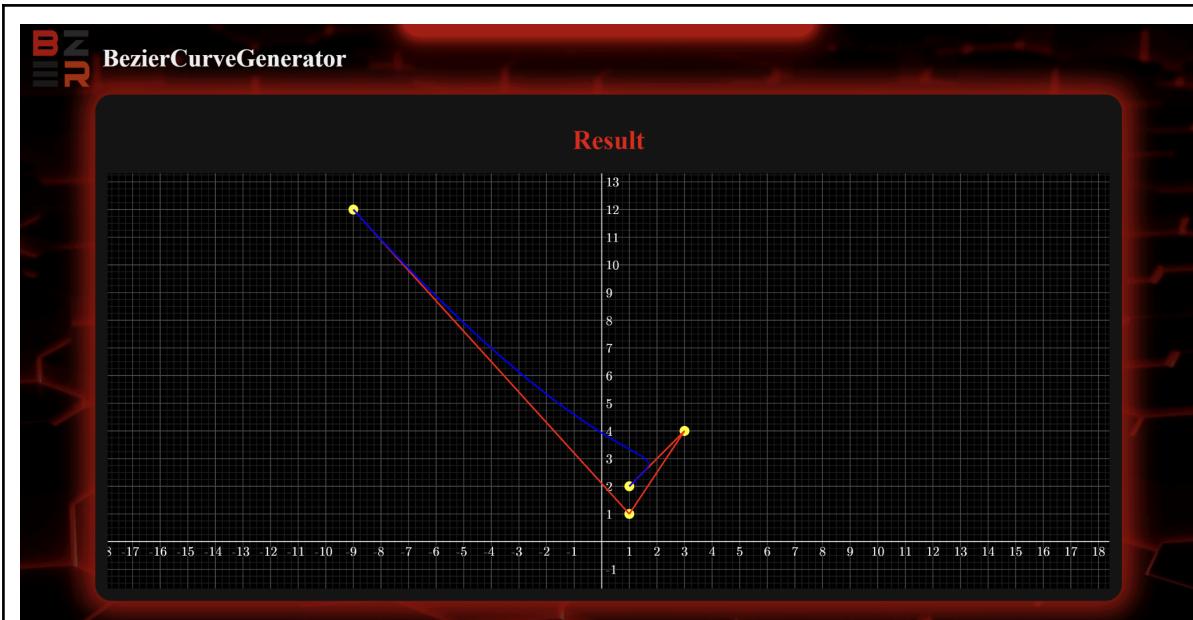
Output algoritma divide and conquer:



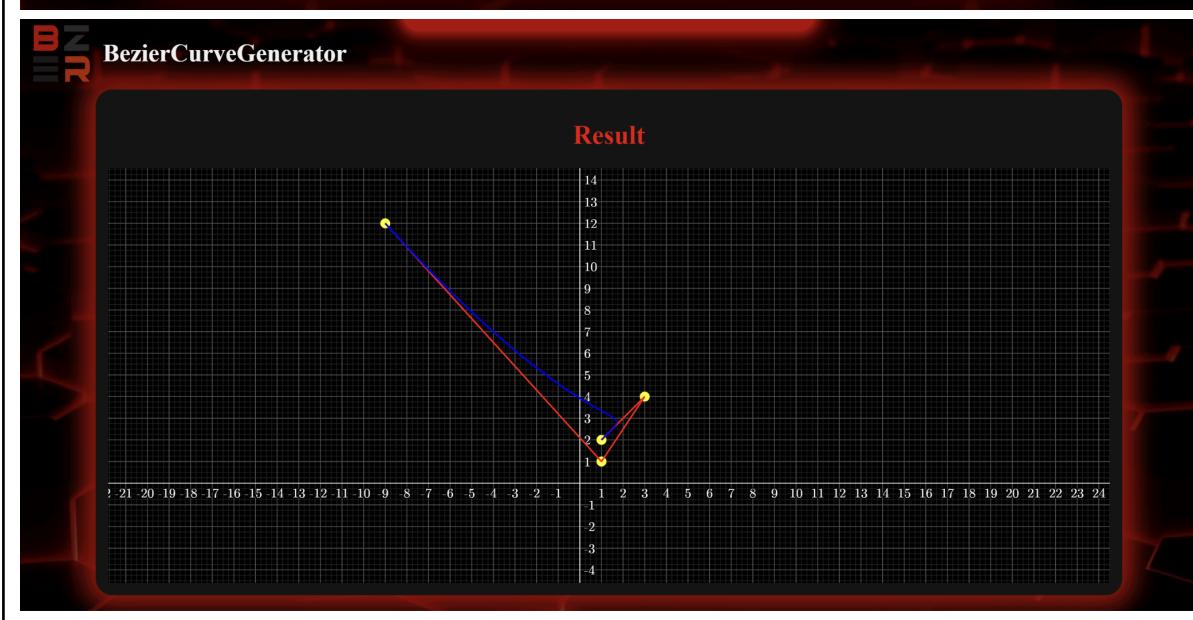
## 4.2 Test 2

Tabel 4.2 Test 2

Input

Output
<p>Output algoritma brute force:</p> 



Output algoritma divide and conquer:



## 4.3 Test 3

Tabel 4.3 Test 3

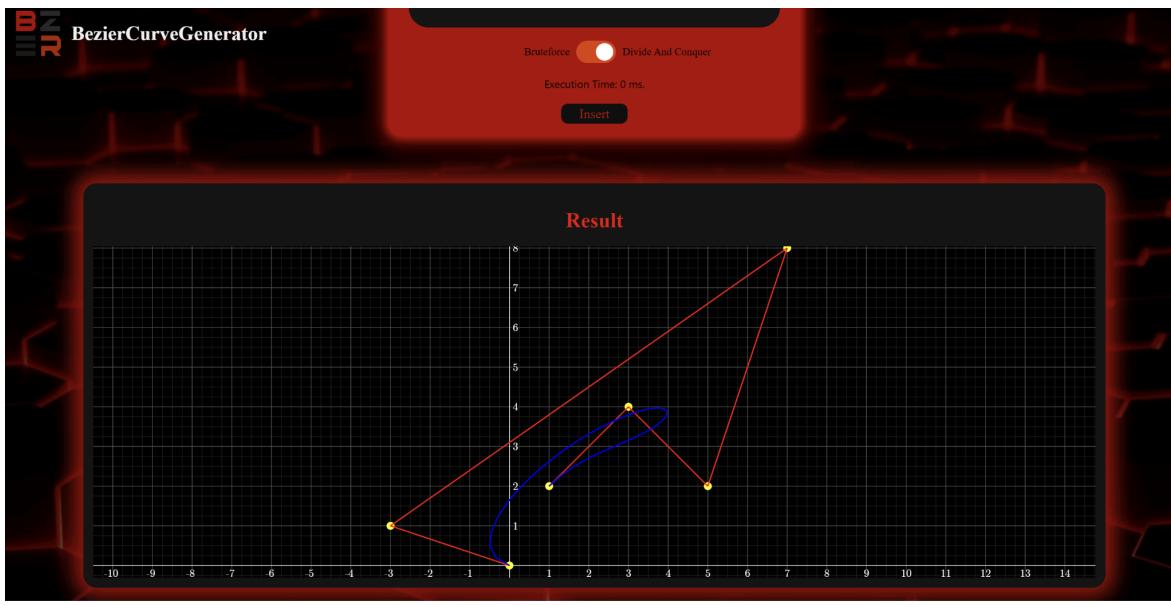
Input
<h2>Bezier Curve Generator</h2> <p>Generate your curve here</p> <p>Control Point</p> <p>5</p> <p>Iteration</p> <p>5</p> <p>X1 ,Y1</p> <p>1      2</p> <p>X2 ,Y2</p> <p>3      4</p> <p>X3 ,Y3</p> <p>5      2</p> <p>X4 ,Y4</p> <p>7      8</p> <p>X5 ,Y5</p> <p>-3      1</p> <p>X6 ,Y6</p> <p>0      0</p>

## Output

Output algoritma brute force:



Output algoritma divide and conquer:



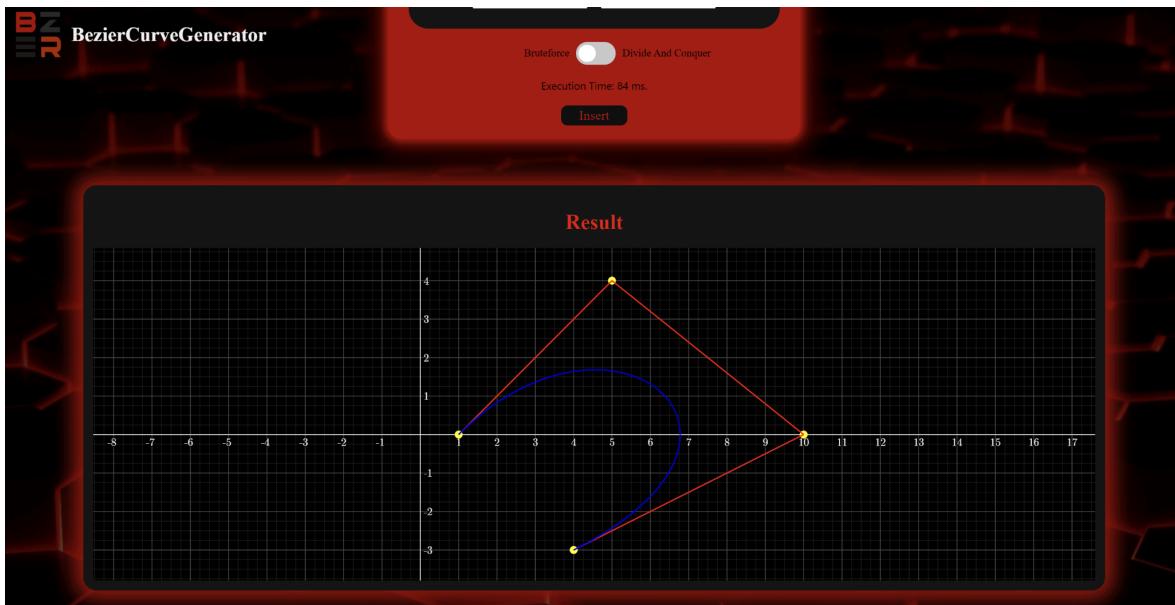
## 4.4 Test 4

Tabel 4.4 Test 4

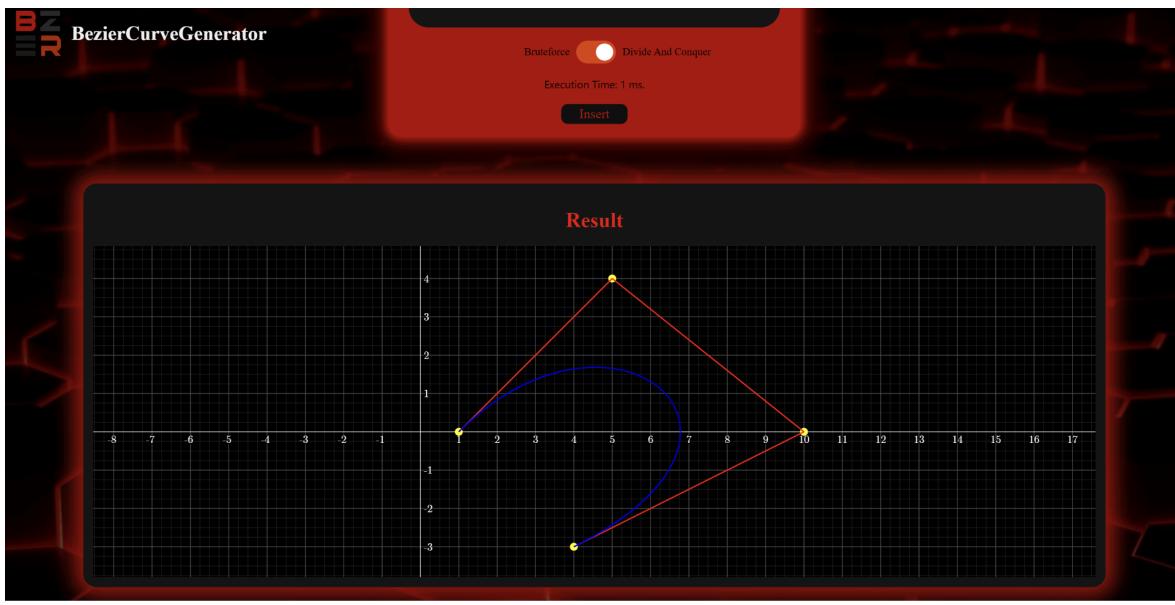
Input
<h1>Bezier Curve Generator</h1> <p>Generate your curve here</p> <p>Control Point</p> <p>3</p> <p>Iteration</p> <p>10</p> <p>X1 ,Y1</p> <p>1 0</p> <p>X2 ,Y2</p> <p>5 4</p> <p>X3 ,Y3</p> <p>10 0</p> <p>X4 ,Y4</p> <p>4 -3</p>

## Output

Output algoritma brute force:



Output algoritma divide and conquer:



4.5 Test 5

Tabel 4.5 Test 5

# Input

## Bezier Curve Generator

Generate your curve here

Control Point

10

Iteration

10

X1 ,Y1

1  0

X2 ,Y2

5  4

X3 ,Y3

10  0

X4 ,Y4

4  -3

X5 ,Y5

0  -10

X6 ,Y6

-3  3

X7 ,Y7

-5  0

X8 ,Y8

-3  -3

X9 ,Y9

0  -5

X10 ,Y10

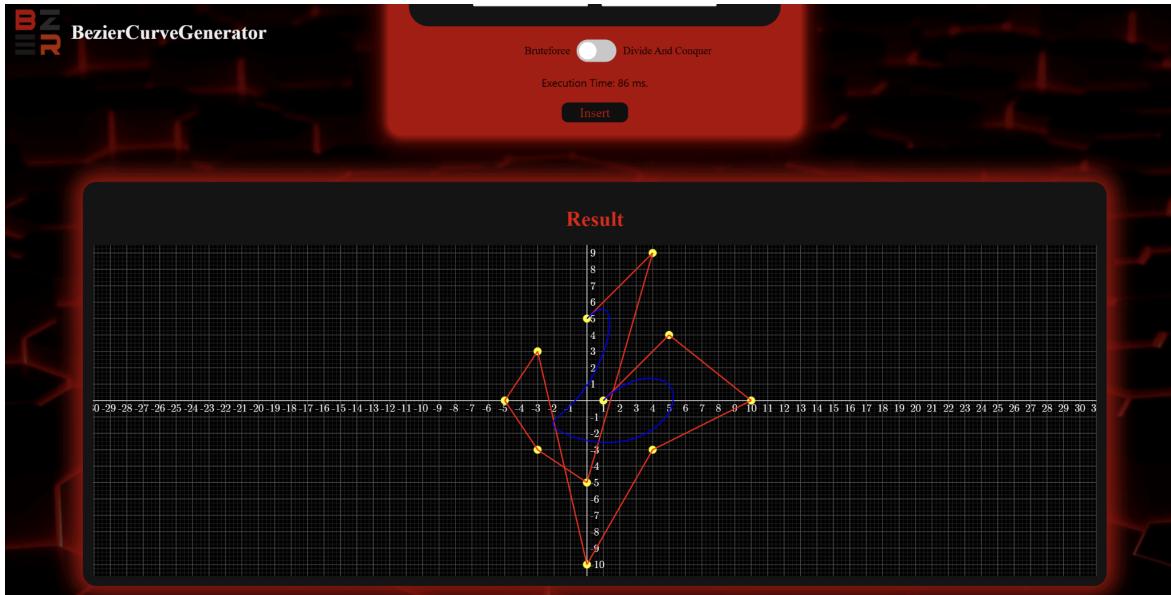
4  9

X11 ,Y11

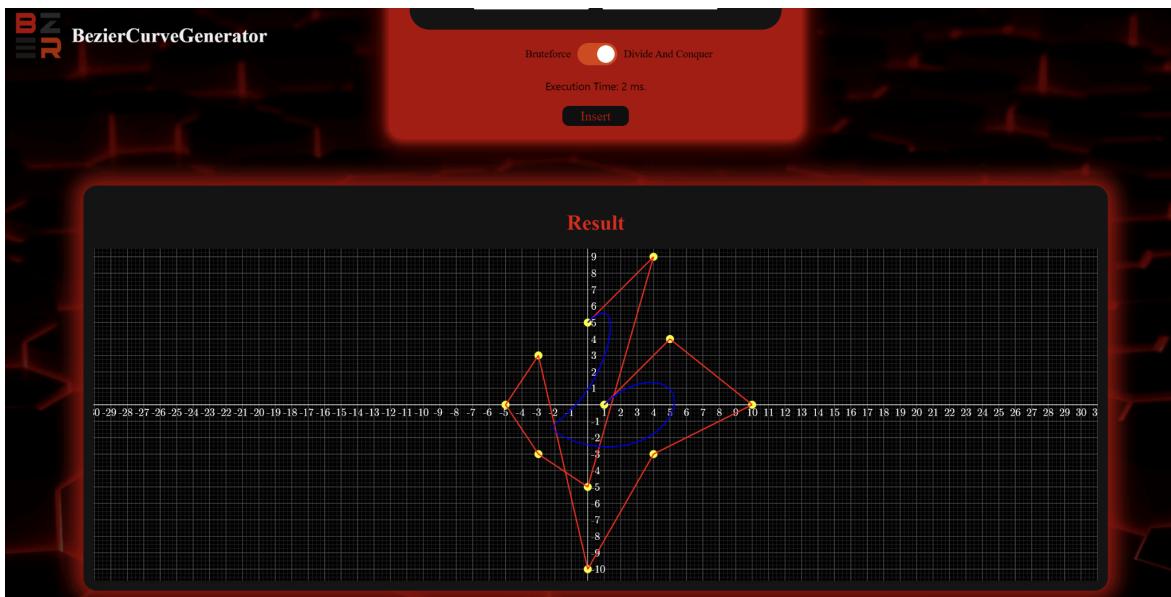
0  5

## Output

Output algoritma brute force:



Output algoritma divide and conquer:



## 4.6 Test 6

Tabel 4.6 Test 6

# Input

## Bezier Curve Generator

Generate your curve here

Control Point

8

Iteration

15

X1 ,Y1

-5  0

X2 ,Y2

0  4

X3 ,Y3

3  -3

X4 ,Y4

-10  -3

X5 ,Y5

3  1

X6 ,Y6

6  -6

X7 ,Y7

1  -6

X8 ,Y8

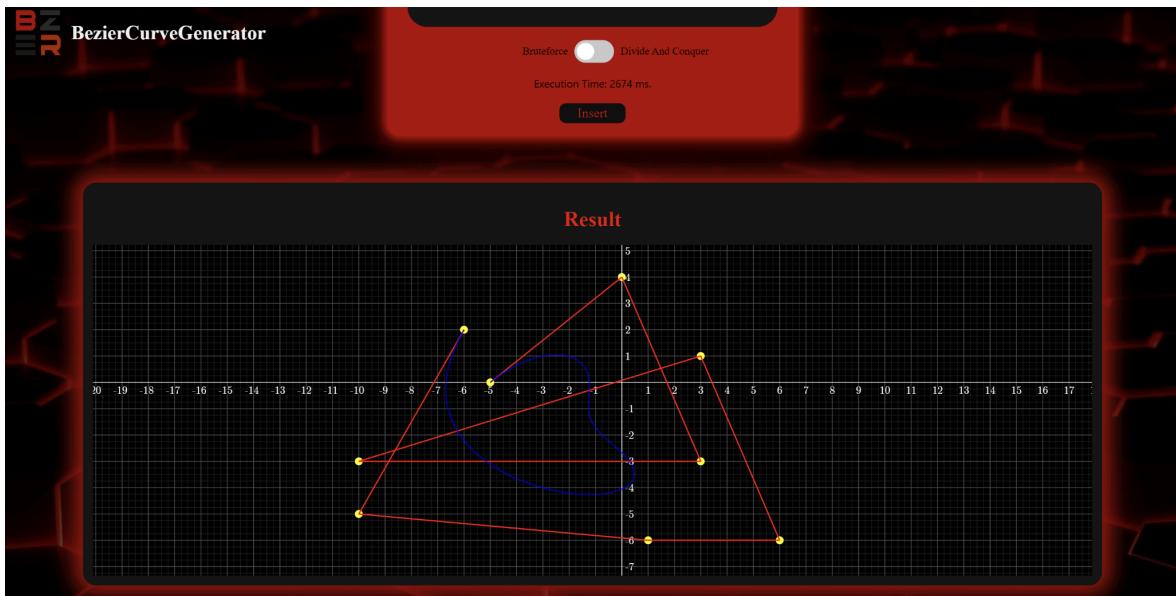
-10  -5

X9 ,Y9

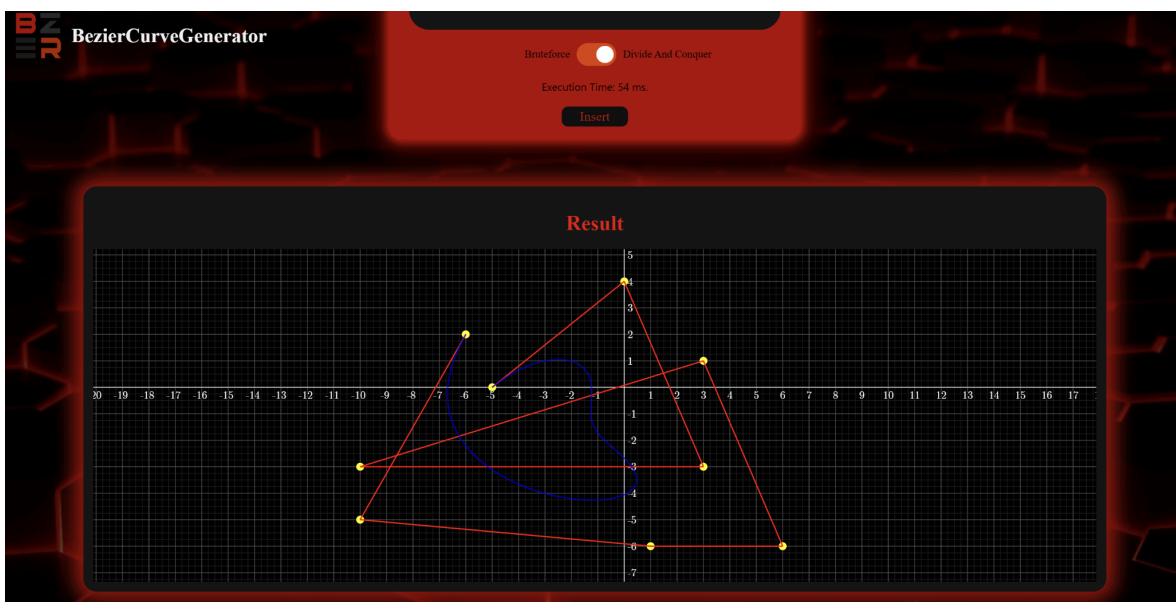
-6  2

## Output

Output algoritma brute force:



Output algoritma divide and conquer:



## BAB V

### Analisis

#### 4.1 Analisis Penggunaan Algoritma Brute Force dalam Pembentukan Kurva Bézier

Algoritma brute force ini digunakan untuk menghitung titik-titik pada kurva Bézier dengan melakukan perhitungan langsung pada setiap iterasi untuk setiap nilai parameter  $t$  dalam rentang yang ditentukan. Algoritma ini relatif sederhana dan mudah dipahami. Implementasi untuk algoritma ini pun cukup singkat dan mudah diimplementasikan. Akan tetapi, di sisi lain algoritma ini memiliki kompleksitas waktu yang tinggi karena perlu melakukan perhitungan untuk setiap nilai parameter  $t$  dalam rentang  $[0, 1]$ , serta melakukan perulangan di dalamnya. Algoritma ini cocok digunakan untuk kasus di mana jumlah titik kontrol kurang atau kecepatan eksekusi tidak menjadi faktor penentu.

Pada gambar 3.1.1 perulangan dalam fungsi *findpoint* memiliki dua buah kalang loop. Kompleksitas waktu dari perulangan dalam fungsi *findpoint* adalah  $O(n^2)$ , di mana  $n$  adalah jumlah titik kontrol. Sementara itu, dalam fungsi *TraceCurve* memiliki  $T(n) = (1/add) + 1$ . Hal ini membuat fungsi tersebut memiliki kompleksitas waktu sebesar  $O(1/add)$ . Dengan *add* adalah nilai penambahan pada nilai parameter  $t$  yang diperoleh berdasarkan jumlah iterasi, yaitu  $0.5^i$ , dengan  $i$  adalah jumlah iterasi sehingga kompleksitas untuk fungsi ini dapat dituliskan sebagai  $O(2^m)$ , dengan  $m$  adalah jumlah iterasi.

Total kompleksitas waktu dari kedua fungsi adalah  $O(n^2 \cdot 2^m)$ , di mana  $n$  adalah jumlah titik kontrol dan  $m$  adalah jumlah iterasi. Algoritma ini tidak mengoptimalkan penggunaan titik-titik kontrol yang sama di setiap iterasi, sehingga dapat memakan waktu yang cukup lama terutama untuk jumlah titik kontrol yang semakin besar. Performanya juga dapat menurun signifikan jika jumlah titik kontrol membesar atau iterasi yang diperlukan banyak.

#### 4.2 Analisis Penggunaan Algoritma Divide and Conquer dalam Pembentukan Kurva Bézier

Algoritma divide and conquer digunakan untuk mencari titik-titik pada kurva Bézier. Pada setiap iterasi, kurva Bezier dibagi menjadi dua bagian dengan menggunakan titik kontrol tengah. Setiap bagian kemudian diperlakukan sebagai kurva Bezier baru yang lebih sederhana. Pendekatan ini menciptakan struktur rekursif, di mana setiap langkah rekursif mengurangi kompleksitas masalah menjadi setengahnya.

Pada gambar 3.2.1 anggap pada perhitungan titik tengah antara dua buah titik pada fungsi *Find\_Mid\_Point* memiliki kompleksitas sebesar  $O(1)$ . Pada fungsi *Construct\_List\_Mid\_Point* memiliki kompleksitas waktu sebesar  $O(n^2)$ , yang merupakan hasil dari dua buah kalang loop bersarang yang di dalamnya terdapat pemanggilan fungsi *Find\_Mid\_Point* untuk menghitung titik tengah antara dua titik lalu melakukan *append* dengan array temporary di mana hal ini anggap kompleksitasnya sebesar  $O(1)$ .

Sementara itu, fungsi utama algoritma divide and conquer untuk mencari titik-titik pada kurva Bézier terdapat pada fungsi *Bezier\_Line*. Fungsi ini memiliki kondisi conquer

(*solve*) dengan melakukan pemanggilan fungsi `Construct_List_Mid_Point` yang kemudian dilanjutkan 2 buah proses komputasi untuk melakukan *append* terhadap hasil sebelumnya (proses conquer) sehingga memiliki  $T(n)$  untuk kondisi *solve* sebesar  $n^2 + 2$ . Selanjutnya pada kondisi divide, algoritma ini akan membagi titik menjadi dua buah segmen. Pada gambar 3.2.1 proses pembagian dilakukan melalui dua buah loop `for` terpisah sebesar  $n$ , dengan  $n$  adalah jumlah titik kontrol. Proses kemudian dilanjutkan dengan pemanggilan fungsi secara rekursif namun dengan masukan sebesar  $n/2 + 1$ . Hal tersebut membuat proses divide memiliki  $T(n)$  sebesar  $2T(n/2) + n$ . Karena setiap iterasi akan membagi masalah menjadi dua sub masalah, dan bahwa pada setiap level rekursi jumlah titik kontrol berkurang menjadi sekitar setengahnya, maka jumlah level rekursi memiliki kompleksitas waktu  $O(\log n)$ . Dengan demikian, kompleksitas waktu total algoritma divide and conquer ini adalah gabungan dari kompleksitas waktu pada kondisi conquer dan divide, yang dapat dihitung menggunakan metode *Master Theorem* dan diperoleh kompleksitasnya sebesar  $O(n^2 \log n)$ .

#### 4.3 Perbandingan Penggunaan Algoritma Brute Force dan Algoritma Divide and Conquer

Pencarian titik-titik pada kurva Bézier memiliki kompleksitas waktu yang berbeda antara algoritma brute force dengan algoritma divide and conquer. Pada algoritma brute force memiliki kompleksitas waktu sebesar  $O(n^2 \cdot 2^m)$ , di mana  $n$  adalah jumlah titik kontrol dan  $m$  adalah jumlah iterasi. Kompleksitasnya meningkat secara eksponensial dengan jumlah titik kontrol dan iterasi. Hal ini membuat algoritma brute force tidak efisien untuk jumlah titik kontrol yang besar. Sementara itu, pada algoritma divide and conquer memiliki kompleksitas waktu sebesar  $O(n^2 \log n)$ . Dibanding dengan algoritma brute force, algoritma ini memiliki kinerja yang lebih baik.

Algoritma brute force memiliki implementasi yang relatif sederhana dan mudah dipahami. Namun, algoritma brute force tidak mengoptimalkan penggunaan titik kontrol yang sama di setiap iterasi. Hal ini membuatnya kurang efisien, terutama untuk jumlah titik kontrol yang besar. Performanya dapat menurun secara signifikan ketika jumlah titik kontrol atau iterasi meningkat. Sementara algoritma divide and conquer secara efektif membagi masalah menjadi submasalah yang lebih kecil, mengurangi jumlah perhitungan yang diperlukan secara eksponensial dengan jumlah titik kontrol. Performa algoritma ini relatif lebih baik, terutama untuk jumlah titik kontrol yang besar. Struktur rekursifnya memungkinkan pengurangan jumlah perhitungan yang signifikan sehingga menghasilkan performa yang lebih baik dibandingkan dengan algoritma Brute Force.

Dengan demikian, algoritma brute force cocok digunakan untuk kasus di mana jumlah titik kontrol sedikit atau kecepatan eksekusi bukan faktor penentu. Namun, tidak efisien untuk jumlah titik kontrol yang besar karena kompleksitas waktu yang meningkat secara eksponensial. Sementara itu, algoritma divide and conquer lebih efisien untuk jumlah titik kontrol yang besar karena mengurangi jumlah perhitungan yang diperlukan secara eksponensial melalui pendekatan rekursifnya. Cocok digunakan untuk kasus di mana jumlah titik kontrol besar atau ketika performa yang baik diperlukan.

## BAB VI

### Implementasi Bonus

#### 5.1 Implementasi Kurva Bézier $n$ Buah Titik Kontrol

Pada implementasi algoritma divide and conquer yang diberikan pada gambar 3.2.1 telah meliputi pencarian titik-titik pada kurva Bézier untuk  $n$  buah titik kontrol. Pada dasarnya, persoalan dalam mencari titik yang membentuk kurva Bézier dapat dibagi menjadi upa-persoalan yang lebih kecil. Hal tersebut terjadi jika nilai iterasi yang dilakukan belum terpenuhi, artinya masih diinginkan untuk melakukan iterasi kembali dalam mencari titik pada kurva Bézier. Pembagian dilakukan dengan menentukan masing-masing daftar titik tengah dari setiap titik yang berurutan, titik ini akan menjadi titik kontrol baru. Kemudian membagi titik-titik tersebut menjadi dua segmen, segmen pertama akan menggunakan sebuah titik awal ( $P_0$ ) dan setengah titik-titik tengah awal, sedangkan segmen yang kedua akan menggunakan setengah titik-titik tengah akhir dan sebuah titik akhir ( $P_n$ ). Setelah itu, Selesaikan secara rekursif upa-persoalan yang telah dibagi menjadi dua segmen tersebut.

Pada setiap langkah rekursi, setiap upa-persoalan akan diselesaikan secara terpisah. Jika kondisi iterasi telah memenuhi jumlah yang diinginkan (maksimum), maka akan dilakukan langkah penyelesaian (*solve*). Pada kondisi ini, untuk setiap upa-persoalan akan dilakukan pencarian titik tengah menggunakan titik-titik kontrol terakhir. Kurva Bézier dari titik kontrol terakhir dan titik tengah terakhir ini kemudian akan dikonstruksikan nantinya.

Setelah menyelesaikan upa-persoalan dari langkah conquer, maka titik-titik pada kurva Bézier diperoleh. Titik-titik pada kurva Bézier yang diperoleh ini akan dikumpulkan dalam sebuah array hasil yang akan menjadi tempat untuk menampung titik-titik hasil iterasi penentuan titik-titik pada kurva Bézier. Titik-titik kontrol yang diperoleh dari rekursi akan digabungkan dengan titik-titik awal dan akhir pada kurva untuk membentuk kurva Bézier lengkap.

#### 5.2 Implementasi GUI

Implementasi GUI dilakukan dalam sebuah aplikasi web untuk menghasilkan kurva Bezier berbasis React, yang dapat diakses secara lokal pada <http://localhost:3000/>. Pada tampilan halaman utama, pengguna dapat mengatur jumlah titik kontrol, iterasi, dan titik-titik kontrol yang ingin digunakan untuk menghasilkan kurva Bezier. Pada tampilan halaman utama, pengguna dapat mengatur jumlah titik kontrol, iterasi, dan titik-titik kontrol yang ingin digunakan untuk menghasilkan kurva Bezier. Program ini terdiri dari beberapa komponen yang masing-masing memiliki fungsionalitasnya sendiri. Secara garis besar pada file home memiliki fungsionalitas sebagai berikut.

- State Management: Program menggunakan hook useState untuk mengelola beberapa state seperti controlValue, iterationValue, numberOffForms, points, uploadedPoints, controlPoints, timeTaken, dan isOn. Hal ini memungkinkan perubahan state dalam

komponen berdasarkan interaksi pengguna.

- Fungsi addPoint: Fungsi ini digunakan untuk menambahkan titik baru ke dalam array points berdasarkan indeks titik kontrol yang diberikan. Jika indeks tersebut melebihi panjang array saat ini, maka ukuran array akan diperbesar terlebih dahulu. Kemudian, titik baru akan ditambahkan ke dalam array pada indeks yang sesuai.
- Fungsi Home: Fungsi ini merupakan komponen utama yang menampilkan seluruh tampilan aplikasi. Di dalamnya terdapat beberapa komponen lain seperti ControlPointForm, IterationForm, PointsForm, OnOff, InsertForm, dan Mefs.
- Rendering Formulir: Pada bagian render, program menampilkan formulir untuk pengaturan jumlah titik kontrol dan iterasi, serta formulir untuk memasukkan titik-titik kontrol. Jumlah formulir untuk titik-titik kontrol akan disesuaikan dengan nilai dari state numberOfForms.
- Fungsi setUploadedPoints dan setTimeTaken: Fungsi-fungsi ini digunakan untuk mengatur state uploadedPoints dan timeTaken setelah proses pengiriman data ke backend selesai. Ini memungkinkan program untuk menampilkan hasil kurva Bezier yang dihasilkan dan waktu yang diperlukan untuk memprosesnya.

Program akan menerima permintaan HTTP POST untuk menyimpan data titik kontrol yang diberikan dalam format JSON. Saat menerima permintaan, server akan membaca data JSON dari permintaan, memecahnya, dan mengembalikan titik-titik yang diterima bersama dengan titik kontrol yang telah diolah ke dalam format JSON sebagai respons. Setiap permintaan diproses oleh fungsi handleInsert. Jika terjadi kesalahan dalam membaca atau memarsing data JSON, server akan memberikan respons yang sesuai. Pada bagian utama program main(), server akan mendengarkan permintaan masuk dan menanggapi permintaan tersebut dengan memanggil fungsi handleInsert. Program ini menggunakan *package* "app/util/lib" yang berisi fungsi-fungsi untuk mengolah titik-titik kontrol dengan fungsi *main*-nya adalah Back\_Main, yang merupakan fungsi utama untuk pemrosesan titik-titik kontrol sesuai algoritma yang digunakan, baik algoritma brute force atau pun divide and conquer.

Implementasi dari GUI ini akan memberikan kemudahan bagi *user* untuk menggunakan membuat kurva Bézier. Namun, di sisi lain perlu diketahui bahwa untuk iterasi dalam ukuran yang begitu besar akan membuat ukuran proses komputasi yang semakin besar pula. Dengan begitu, hal ini seringkali membuat program menjadi *not responding*.

## DAFTAR PUSTAKA

- [1] Munir, Rinaldi. 2024. Algoritma Divide and Conquer (Bagian 1).  
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf), diakses pada 16 Januari 2024.

## LAMPIRAN

Link Repository:

[https://github.com/scifo04/Tucil2\\_13522110\\_13522116](https://github.com/scifo04/Tucil2_13522110_13522116)

Check List:

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.		✓