

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

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



Disusun oleh :

**BRYAN CORNELIUS LAUWRENCE**

13522033

**RAFFAEL BOYMIAN SIAHAAN**

13522046

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**

**2024**

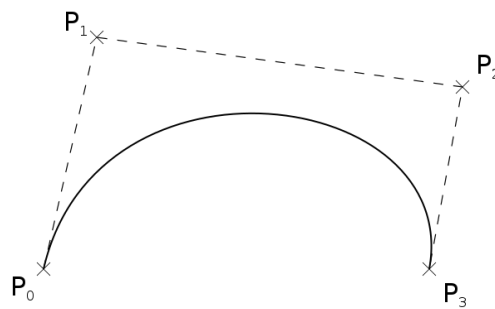
## DAFTAR ISI

<b>DESKRIPSI TUGAS.....</b>	<b>3</b>
<b>IMPLEMENTASI ALGORITMA.....</b>	<b>5</b>
2.1 Algoritma Brute Force.....	5
2.2 Algoritma Divide and Conquer.....	5
<b>ANALISIS KOMPLEKSITAS ALGORITMA.....</b>	<b>7</b>
3.1 Algoritma Brute Force.....	7
3.2 Algoritma Divide and Conquer.....	7
3.3 Perbandingan Algoritma.....	8
<b>SOURCE CODE PROGRAM.....</b>	<b>9</b>
4.1 Algoritma Brute Force.....	9
4.2 Algoritma Divide and Conquer.....	9
<b>PENGUJIAN PROGRAM DAN ANALISIS PERBANDINGAN.....</b>	<b>11</b>
5.1 Testing Program.....	11
5.2 Analisis Perbandingan.....	15
<b>KESIMPULAN DAN SARAN.....</b>	<b>18</b>
6.1 Kesimpulan.....	18
6.2 Saran.....	18
<b>LAMPIRAN.....</b>	<b>19</b>
<b>DAFTAR PUSTAKA.....</b>	<b>19</b>

## BAB I

### DESKRIPSI TUGAS

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. 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. Gambar 1.1 merupakan contoh kurva Bézier kubik dengan empat buah titik kontrol.  $P_0$  sebagai titik kontrol awal dan  $P_3$  sebagai titik kontrol akhir. Titik kontrol lainnya, yaitu  $P_1$  dan  $P_2$  sebagai titik kontrol antara yang tidak dilalui oleh kurva Bézier.



Gambar 1.1 Kurva Bézier Kubik

(Sumber: [https://en.wikipedia.org/wiki/B%C3%A9zier\\_curve](https://en.wikipedia.org/wiki/B%C3%A9zier_curve))

Misalkan terdapat kurva Bézier kuadratik yang dibentuk dari 3 titik, yaitu  $P_0$ ,  $P_1$ , dan  $P_2$ . Dengan menyatakan titik  $Q_1$  terletak diantara garis yang menghubungkan  $P_1$  dan  $P_2$ , dan membentuk kurva Bézier linier yang berbeda dengan kurva letak  $Q_0$  berada, maka dapat dinyatakan sebuah titik baru,  $R_0$  yang berada diantara garis yang menghubungkan  $Q_0$  dan  $Q_1$  yang bergerak membentuk kurva Bézier kuadratik terhadap titik  $P_0$  dan  $P_2$ . Persamaannya sebagai berikut:

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

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

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

dengan melakukan substitusi nilai  $Q_0$  dan  $Q_1$ , akan diperoleh persamaan berikut:

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

Tugas kecil ini bertujuan untuk mengimplementasikan program untuk membuat kurva Bézier dengan algoritma *brute force* dan algoritma *divide and conquer* agar kemudian memvisualisasikan kurva hasil. Program, utamanya, akan menerima tiga buah titik dan jumlah iterasi pembentukan kurva Bézier. Program akan dibuat dengan bahasa pemrograman Python 3.

## BAB II

### IMPLEMENTASI ALGORITMA

#### 2.1 Algoritma *Brute Force*

Algoritma *brute force* adalah pendekatan yang straightforward untuk memecahkan suatu persoalan. Algoritma ini didasarkan pada problem statement dan konsep yang dilibatkan. Algoritma ini akan memecahkan persoalan dengan langsung, jelas, dan sangat sederhana. Pada permasalahan pembuatan kurva Bézier, pencarian titik menggunakan rumus yang sudah disebutkan pada bab 1. Jumlah titik yang dicari mengikuti jumlah iterasi yang diberikan. Untuk  $n$  buah iterasi, akan dicari  $2^n + 1$  titik kurva Bézier.

Algoritma *brute force* yang dibuat memiliki langkah-langkah sebagai berikut:

1. Hitung jumlah titik yang perlu dicari berdasarkan jumlah iterasi ( $n$ ), yaitu  $2^n + 1$ .
2. Tetapkan nilai  $2^n$  sebagai penyebut untuk seluruh nilai  $t$ .
3. Lakukan perulangan satu per satu bilangan bulat dari 0 sampai  $2^n$
4. Untuk setiap perulangan, Gunakan bilangan sebagai pembilang untuk nilai  $t$ .
5. Setelah itu, lakukan juga perulangan sebanyak jumlah titik kontrol ( $m$ ), untuk mencari titik sesuai rumus berdasarkan setiap titik kontrol.
6. Simpan titik ke dalam sebuah larik (*array*).
7. Setelah perulangan selesai, gambarkan grafik berdasarkan titik yang sudah diperoleh.

#### 2.2 Algoritma *Divide and Conquer*

Algoritma *divide and conquer* adalah algoritma yang membagi suatu persoalan menjadi beberapa sub persoalan yang lebih kecil. Selanjutnya, setiap sub persoalan akan diselesaikan satu per satu. Terakhir, hasil dari sub persoalan akan digabungkan kembali untuk memperoleh hasil penyelesaian masalah semula. Pada permasalahan kurva Bézier, konsep *divide and conquer*-nya akan membagi  $n$  buah titik menjadi dua bagian, yaitu bagian kiri dan kanan dari titik tengah pada setiap iterasi, kemudian mencari setiap titik tengah dari pembagian tersebut sampai iterasi habis (bernilai 0).

Algoritma *divide and conquer* yang dibuat memiliki langkah-langkah sebagai berikut:

1. Untuk kasus iterasi = 0,  
SOLVE: gabungkan hasil akhir dengan titik kontrol awal dan titik kontrol akhir.
2. Untuk kasus iterasi  $> 0$ ,

DIVIDE: Cari titik tengah untuk setiap titik sampai tidak ada lagi titik yang bisa dicari titik tengahnya. bagi menjadi dua bagian dengan titik paling tengah sebagai titik kontrol baru pada setiap bagian.

CONQUER: hitung kembali untuk bagian kanan dan kiri dari titik paling tengah.

MERGE: gabungkan hasil bagian kanan dengan titik paling tengah, kemudian gabungkan lagi dengan hasil bagian kiri.

## BAB III

### ANALISIS KOMPLEKSITAS ALGORITMA

#### 3.1 Algoritma *Brute Force*

Pada algoritma *brute force*, ketika diberikan  $m$  buah titik kontrol untuk diproses sejumlah  $n$  iterasi. Pencarian titik kurva Bézier akan dilakukan sebanyak  $2^n + 1$  kali. Untuk setiap perulangan, akan dihitung lokasi titik berdasarkan setiap titik kontrol. Artinya pada setiap perulangan, dilakukan  $m$  buah proses. Oleh karena itu, kompleksitas algoritma *brute force* dapat dinyatakan sebagai berikut:

$$T(m, n) = m(2^n + 1)$$

Jika kurva Bézier merupakan kurva Bézier kubik yang terdiri dari 3 titik kontrol, persamaannya akan menjadi:

$$T(n) = 3(2^n + 1) = 3 + 3 \times 2^n$$

Notasi *Big-O* dari kompleksitas tersebut dapat dinyatakan sebagai berikut:

$$2^n + 3 \times 2^n \geq T(n), n \geq 2,$$

sehingga kompleksitas algoritma *brute force* dari pembentukan kurva Bézier kubik sejumlah  $n$  iterasi adalah  $O(2^n)$ .

#### 3.2 Algoritma *Divide and Conquer*

Saat iterasi algoritma *divide and conquer* bernilai nol, algoritma akan mengembalikan sebuah list kosong. Selain itu, akan dilakukan pencarian titik tengah. Jika masukkan berjumlah  $m$  titik kontrol, pencarian titik tengah akan dilakukan sebanyak  $(m^2 - m)/2$  kali. Untuk  $n$  iterasi, dengan  $n$  lebih dari nol, akan dicari kembali menggunakan algoritma yang sama titik tengah di kiri dan kanan dengan jumlah titik kontrol yang sama dan iterasi bernilai  $n - 1$ . Kompleksitasnya dapat dinyatakan sebagai berikut:

$$T(m, n) = a, n = 0$$

$$T(m, n) = ((m^2 - m)/2) + 2(T(m, n - 1)), n > 0$$

Pada kurva Bézier kubik, karena jumlah titik kontrol adalah tiga, maka pencarian titik dilakukan sebanyak 3 kali di setiap iterasinya. Kompleksitasnya sebagai berikut:

$$T(n) = a, n = 0$$

$$T(n) = 3 + 2T(n - 1), n > 0$$

dan penyelesaian dari kompleksitas untuk kurva Bézier kubik:

$$T(n) = 3 + 2T(n - 1)$$

$$T(n) = 3 + 2(3 + 2T(n - 2)) = 9 + 4T(n - 2)$$

$$T(n) = 3 + 2(3 + 2(3 + 2T(n - 3))) = 9 + 4(3 + 2T(n - 3)) = 21 + 8T(n - 3)$$

$$T(n) = 21 + 8(3 + 2T(n - 4)) = 45 + 16T(n - 4)$$

...

$$T(n) = 3 \times (2^n - 1) + 2^n = 4 \cdot 2^n - 3$$

Notasi *Big-O* dari kompleksitas tersebut dapat dinyatakan sebagai berikut:

$$4 \cdot 2^n \geq T(n), n \geq 0,$$

sehingga kompleksitas algoritma *divide and conquer* dari pembentukan kurva Bézier kubik sejumlah  $n$  iterasi adalah  $O(2^n)$ .

### 3.3 Perbandingan Algoritma

Berdasarkan analisis algoritma *brute force* dan algoritma *divide and conquer*, keduanya memiliki kompleksitas *Big-O* yang sama, yaitu  $O(2^n)$  dengan  $n$  adalah jumlah iterasi yang dilakukan. Jadi, algoritma *divide and conquer* tidak menghasilkan tingkat kompleksitas yang lebih optimal berdasarkan notasi *Big-O*. Hanya saja, pada perhitungan dengan algoritma *brute force* untuk kurva Bézier kubik, dilakukan tujuh belas kali perpangkatan dan enam belas kali perkalian pada setiap iterasinya. Sedangkan pada perhitungan dengan algoritma *divide and conquer* untuk kurva Bézier kubik, dilakukan enam kali penjumlahan dan enam kali pembagian. Karena perpangkatan dan perkalian sifatnya lebih mahal daripada penjumlahan dan pembagian, algoritma *divide and conquer* memiliki kompleksitas yang lebih efektif. Jadi, algoritma *divide and conquer* mampu menghasilkan algoritma yang lebih sangkil dan mangkus dibandingkan algoritma *brute force* pada pembentukan kurva Bézier kubik.



## BAB IV

### SOURCE CODE PROGRAM

#### 4.1 Algoritma *Brute Force*

Berikut merupakan implementasi algoritma *brute force* pada bahasa pemrograman Python:

```
def brute_force_bezier(points, iteration):
    pol_degree = len(points) - 1
    num_points = 2**iteration + 1

    bezier_points = []

    for i in range (num_points):
        t = i / (num_points - 1)
        x, y = 0, 0
        for j in range (len(points)):
            point = points[j]
            bin_coeff = comb(pol_degree, j)
            weight = bin_coeff * (t ** j) * ((1 - t) ** (pol_degree - j))
            x += point[0] * weight
            y += point[1] * weight
        bezier_points.append((x, y))
    return bezier_points
```

#### 4.2 Algoritma *Divide and Conquer*

Berikut merupakan implementasi algoritma *divide and conquer* pada bahasa pemrograman Python:

```
# Mencari titik tengah antara dua titik
def find_middle(p1, p2):
    return ((p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2)

# Mencari titik tengah dari seluruh titik
def find_all_middle(list_of_point):
    li = []
    for i in range (len(list_of_point) - 1):
        li.append(find_middle(list_of_point[i], list_of_point[i+1]))
    return li

# Mencari titik kontrol kurva bezier
def dnc_bezier_helper(list_of_point, iteration):
    if iteration == 0:
        return []
    else:
        # Divide and Conquer
        initial_lenght = len(list_of_point)
        temp = [list_of_point[0], list_of_point[len(list_of_point) - 1]]
        i = 1
        while len(list_of_point) != 1:
            list_of_point = find_all_middle(list_of_point)
            temp.insert(i, list_of_point[0])
```

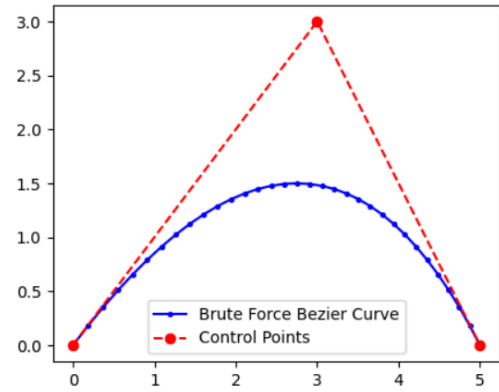
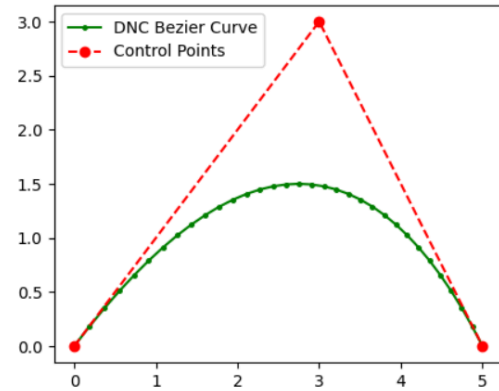
```
        temp.insert(i+1, list_of_poinst[len(list_of_poinst)-1])
        i += 1
    iteration -= 1
    # Merge
    return (dnc_bezier_helper(temp[:initial_lenght], iteration) +
            list_of_poinst + dnc_bezier_helper(temp[initial_lenght:],
            iteration))

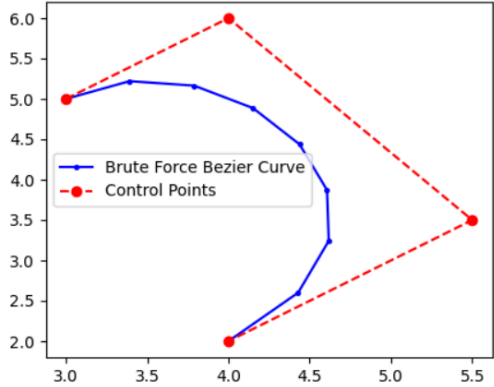
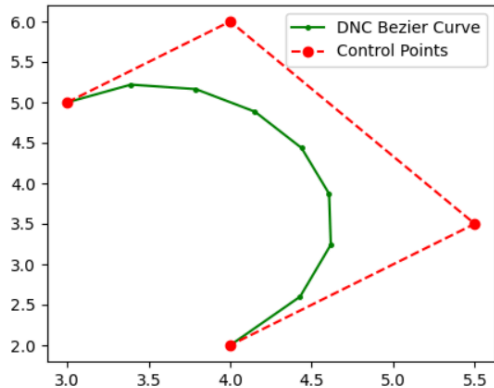
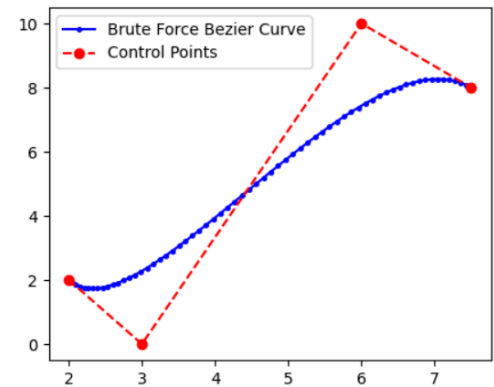
# Mengembalikan titik awal pada kurva bezier
def dnc_bezier(list_of_poinst, iteration):
    return [list_of_poinst[0]] + dnc_bezier_helper(list_of_poinst,
            iteration) + [list_of_poinst[len(list_of_poinst)-1]]
```

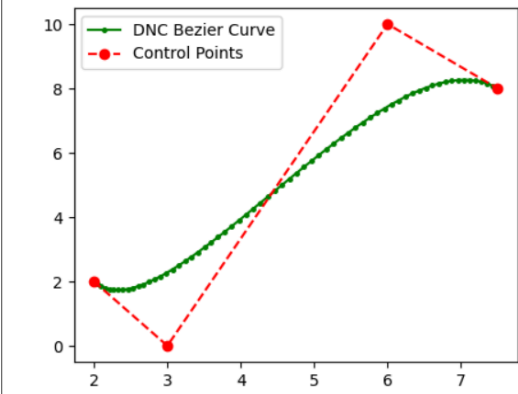
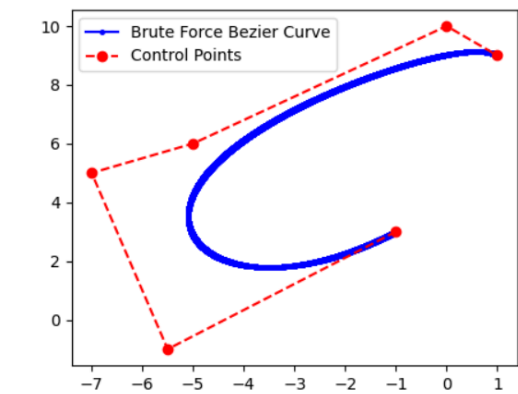
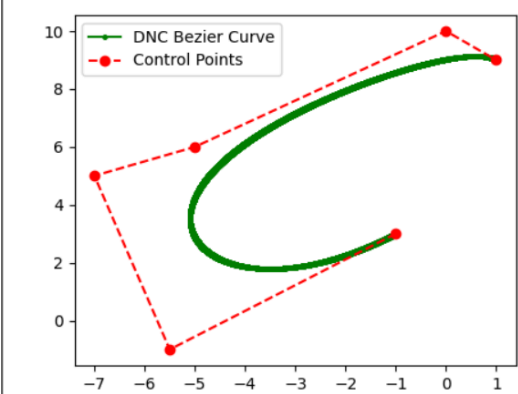
## BAB V

## PENGUJIAN PROGRAM DAN ANALISIS PERBANDINGAN

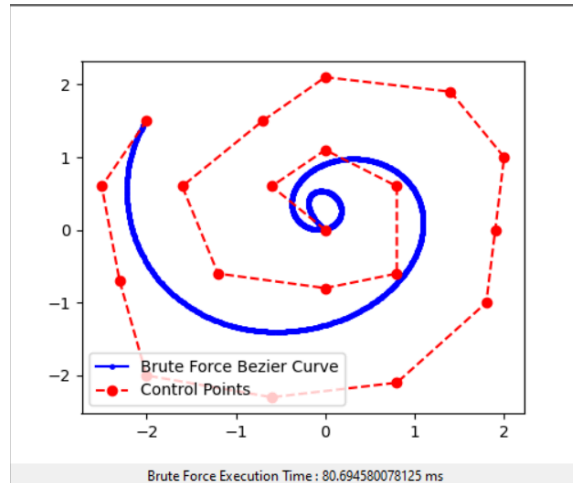
## 5.1 Testing Program

Test Case	Result
n_points = 3 iteration = 5 list_of_points = [(0.0, 0.0), (3.0, 3.0), (5.0, 0.0)]	<b>Brute Force</b>
	 <p>Brute Force Execution Time : 1.9990234375 ms</p>
	<b>Divide and Conquer</b>
	 <p>Divide &amp; Conquer Execution Time : 1.9990234375 ms</p>
n_points = 4 iteration = 3 list_of_points = [(3.0, 5.0), (4.0, 6.0), (5.5, 3.5), (4.0, 2.0)]	<b>Brute Force</b>

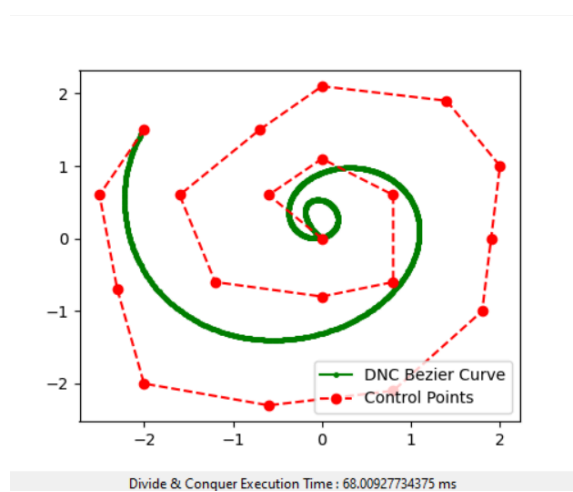
	 <p>Brute Force Execution Time : 1.00146484375 ms</p>
<p>n_points = 4 iteration = 6 list_of_points = [(2.0, 2.0), (3.0, 0.0), (6.0, 10.0), (7.5, 8.0)]</p>	<p><b><i>Divide and Conquer</i></b></p>
	 <p>Divide &amp; Conquer Execution Time : 1.00146484375 ms</p>
	<p><b><i>Brute Force</i></b></p>
	 <p>Brute Force Execution Time : 1.0009765625 ms</p>
	<p><b><i>Divide and Conquer</i></b></p>

	<div><p>A line graph showing a green solid line representing the DNC Bezier Curve and red dashed lines with dots representing the Control Points. The x-axis ranges from 2 to 7, and the y-axis ranges from 0 to 10. The curve starts at (2, 2), dips to (3, 0), rises to (6, 10), and ends at (7, 8).</p><p>Divide &amp; Conquer Execution Time : 1.0009765625 ms</p></div>
<div>n_points = 6 iteration = 15 list_of_points = [(-1.0, 3.0), (-5.5, -1.0), (-7.0, 5.0), (-5.0, 6.0), (0.0, 10.0), (1.0, 9.0)]</div>	<div><p><i><b>Brute Force</b></i></p><div><p>A line graph showing a blue solid line representing the Brute Force Bezier Curve and red dashed lines with dots representing the Control Points. The x-axis ranges from -7 to 1, and the y-axis ranges from 0 to 10. The curve starts at (-7, 5), goes to (-5, 6), then loops to (-1, 3), and ends at (1, 9).</p><p>Brute Force Execution Time : 416.450439453125 ms</p></div><p><i><b>Divide and Conquer</b></i></p><div><p>A line graph showing a green solid line representing the DNC Bezier Curve and red dashed lines with dots representing the Control Points. The x-axis ranges from -7 to 1, and the y-axis ranges from 0 to 10. The curve starts at (-7, 5), goes to (-5, 6), then loops to (-1, 3), and ends at (1, 9).</p><p>Divide &amp; Conquer Execution Time : 332.34521484375 ms</p></div></div>
<div>n_points = 20 iteration = 10 list_of_points = [(0.0, 0.0), (-0.6, 0.6), (0.0,</div>	<div><p><i><b>Brute Force</b></i></p></div>

1.1), (0.8, 0.6), (0.8, -0.6), (0.0, -0.8), (-1.2, -0.6), (-1.6, 0.6), (-0.7, 1.5), (0.0, 2.1), (1.4, 1.9), (2.0, 1.0), (1.9, 0.0), (1.8, -1.0), (0.8, -2.1), (-0.6, -2.3), (-2.0, -2.0), (-2.3, -0.7), (-2.5, 0.6), (-2.0, 1.5)]

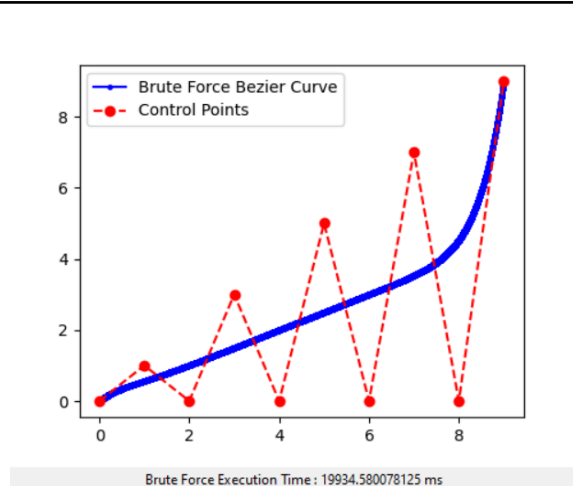


### ***Divide and Conquer***

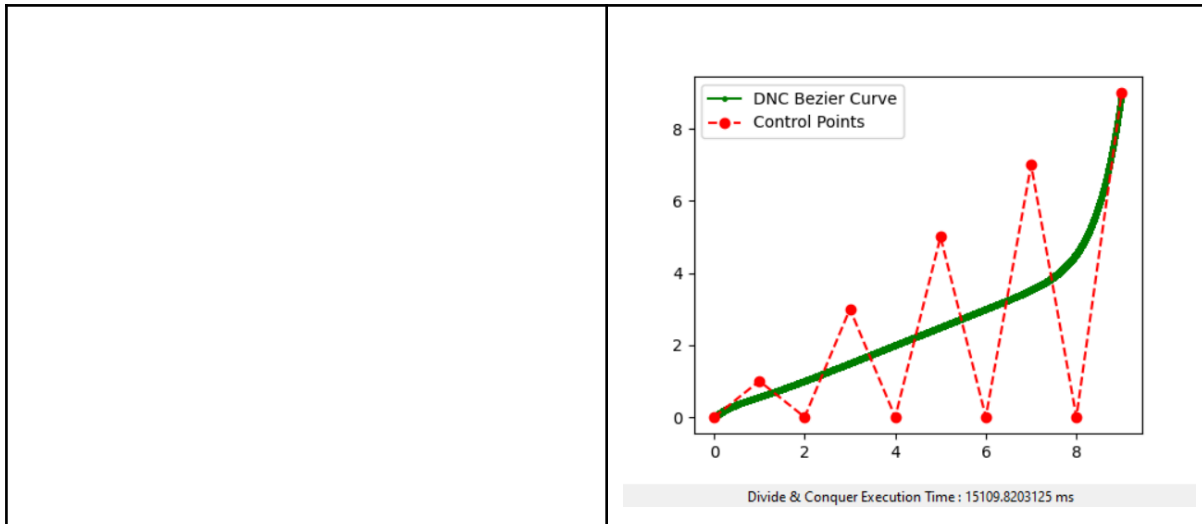


n\_points = 10  
iteration = 20  
list\_of\_points = [(0.0, 0.0), (1.0, 1.0), (2.0, 0.0), (3.0, 3.0), (4.0, 0.0), (5.0, 5.0), (6.0, 0.0), (7.0, 7.0), (8.0, 0.0), (9.0, 9.0)]

### ***Brute Force***



### ***Divide and Conquer***



## 5.2 Analisis Perbandingan

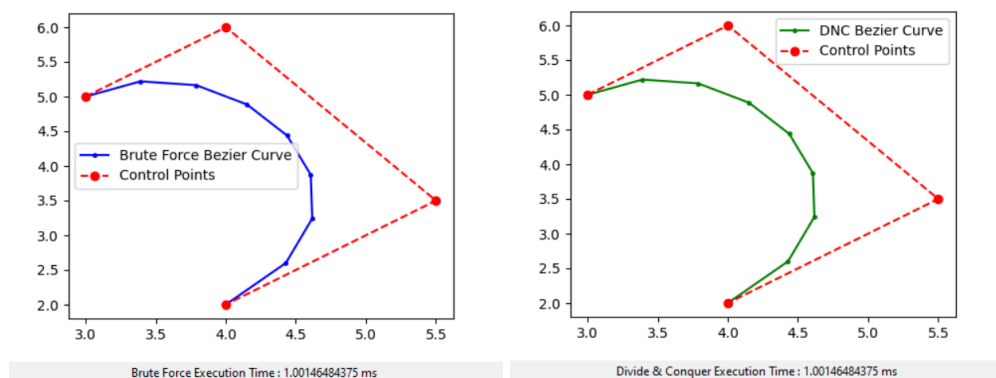
Berdasarkan pengujian, kedua algoritma mampu memberikan kurva Bézier yang sama untuk jumlah titik kontrol dan iterasi yang sama. Untuk menganalisis kompleksitas waktu dari kedua algoritma tersebut, kami menggunakan dua contoh test case sebagai perbandingan:

### 1. Test Case 1

Jumlah titik kontrol: 4

Jumlah iterasi: 3

Kumpulan titik kontrol:  $[(3.0, 5.0), (4.0, 6.0), (5.5, 3.5), (4.0, 2.0)]$



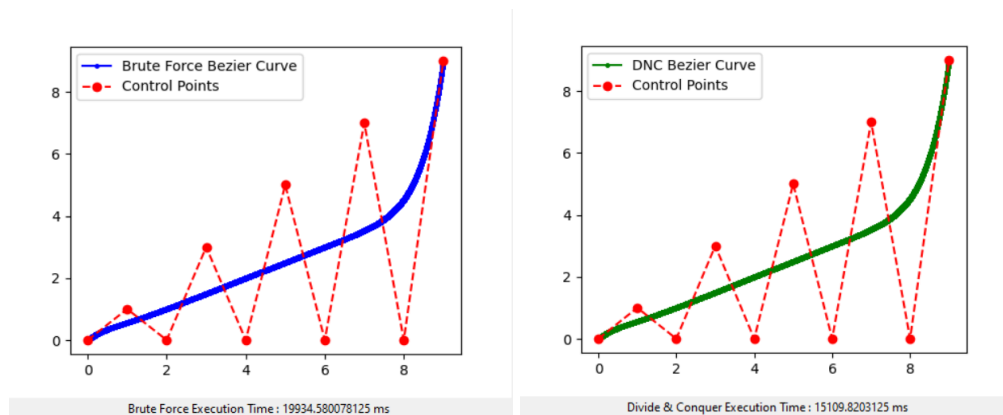
Berdasarkan kedua grafik tersebut, waktu eksekusi yang diberikan sama besar, yaitu sekitar 1,00 milisekon.

### 2. Test Case 2

Jumlah titik kontrol: 10

Jumlah iterasi: 20

Kumpulan titik kontrol: [(0.0, 0.0), (1.0, 1.0), (2.0, 0.0), (3.0, 3.0), (4.0, 0.0), (5.0, 5.0), (6.0, 0.0), (7.0, 7.0), (8.0, 0.0), (9.0, 9.0)]



Berdasarkan kedua grafik tersebut, waktu eksekusi yang diberikan berbeda. Algoritma *brute force* menghasilkan waktu eksekusi sebesar 19.934,58 milisekon, sedangkan algoritma *divide and conquer* menghasilkan waktu eksekusi sebesar 15.109,82 milisekon.

Berdasarkan analisis dari masing-masing *test case*, terdapat dua kesimpulan yang diperoleh, yaitu:

1. algoritma *brute force* dan algoritma *divide and conquer* dapat menghasilkan kompleksitas waktu yang sama ketika jumlah iterasi yang dilakukan tidak besar.
2. algoritma *brute force* dan algoritma *divide and conquer* menghasilkan kompleksitas waktu yang berbeda ketika jumlah iterasi yang dilakukan cukup besar, dimana algoritma *divide and conquer* memiliki kompleksitas waktu yang lebih efektif dan efisien dibandingkan algoritma *brute force*.

Kesimpulan di atas mendukung perbandingan algoritma berdasarkan analisis kompleksitas secara teoritis. Meskipun kedua algoritma memiliki kompleksitas waktu yang sama, yaitu  $O(2^n)$ , perbandingan tersebut memiliki perbedaan dalam jumlah operasi penjumlahan dan perkalian antara kedua algoritma.

Pada algoritma *brute force*, jumlah operasi penjumlahan dan perkalian jauh lebih besar dibandingkan dengan algoritma *divide and conquer*. Oleh karena itu, untuk kasus dengan jumlah iterasi ( $n$ ) yang besar, algoritma *divide and conquer* cenderung lebih efisien dalam pembentukan kurva *Bézier* kubik.



Dengan demikian, meskipun kedua algoritma memiliki kompleksitas yang sama secara teoritis, algoritma *divide and conquer* mampu menghasilkan kurva *Bézier* dengan kinerja yang lebih optimal dalam kasus-kasus dengan jumlah iterasi ( $n$ ) yang besar.

## BAB VI

### KESIMPULAN DAN SARAN

#### 6.1 Kesimpulan

Berdasarkan analisis dan pengujian yang telah dilakukan sebelumnya, kurva Bézier dapat dihasilkan dengan algoritma *brute force* dan *divide and conquer*. Algoritma *brute force* memanfaatkan rumus pencarian titik berdasarkan sejumlah titik kontrol. Algoritma *divide and conquer* menggambarkan kurva Bézier dengan mencari titik tengah dari sejumlah titik. Dengan pendekatan yang berbeda, keduanya mampu menghasilkan bentuk kurva Bézier yang sama untuk setiap iterasinya.

Meskipun hasil keduanya sama, kompleksitas kedua algoritma cukup berbeda. Keduanya memang memiliki kompleksitas sebesar  $O(2^n)$ , tetapi algoritma *brute force* melakukan perhitungan yang lebih mahal karena menggunakan perpangkatan dan perkalian, sedangkan algoritma *divide and conquer* lebih efektif karena hanya menggunakan penjumlahan dan pembagian. Proses *divide and conquer* pun melakukan lebih sedikit penjumlahan dan pembagian dibandingkan dengan perkalian dan perpangkatan oleh *brute force*.

Dari penjelasan tersebut, dapat disimpulkan bahwa algoritma *divide and conquer* pada pencarian kurva Bézier mampu menghasilkan proses yang lebih efektif dibandingkan algoritma *brute force*.

#### 6.2 Saran

Kami merasa program yang kami selesaikan masih jauh dari kata sempurna. Untuk kedepannya, masih banyak pengembangan yang dapat dilakukan untuk memaksimalkan kebermanfaatan program ini. Dengan memperhatikan fungsi, memanfaatkan kegunaan untuk hal lain, dan memperbaiki apa yang masih kurang, kami rasa akan semakin mengeluarkan potensinya.

## LAMPIRAN

Tautan Repository GitHub :

[Tucil2\\_13522033\\_13522046](https://github.com/Tucil2_13522033_13522046)

## DAFTAR PUSTAKA

- [1] R. Munir. (2020). Aplikasi Divide And Conquer 2020 [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Aplikasi-Divide-and-Conquer-2020.pdf> (accessed Mar. 12, 2024).
- [2] R. Bandara. "Midpoint Algorithm (Divide and Conquer Method) for Drawing a Simple Bezier Curve." [codeproject.com. https://www.codeproject.com/Articles/223159/Midpoint-Algorithm-Divide-and-Conquer-Method-for-D](https://www.codeproject.com/Articles/223159/Midpoint-Algorithm-Divide-and-Conquer-Method-for-D) (accessed Mar. 12, 2024)
- [3] Pomax, "A Primer on Bézier Curves." [pomax.github.io. https://pomax.github.io/bezierinfo/](https://pomax.github.io/bezierinfo/) (accessed Mar. 13, 2024)

## Progress Tracking

No	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	✓	