

# **LAPORAN TUGAS KECIL STRATEGI ALGORITMA**

**IF 2211**



**Disusun oleh:**

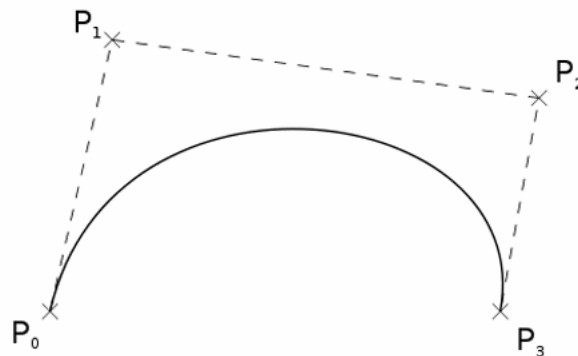
Raden Rafly Hanggaraksa Budiarto -13522014

Rayhan Fadhlán Azka – 13522095

## **I. DAFTAR ISI**

II. Deskripsi Masalah .....	3
III. Implementasi algoritma Brute Force .....	6
IV. Implementasi Algoritma Divide & Conquer .....	7
V. Source Code.....	9
VI. Hasil Pengujian.....	13
VII. Analisis kasus .....	18
VIII. Implementasi Bonus .....	20
IX. Lampiran.....	22

## II. DESKRIPSI MASALAH



*Gambar 1 Kurva Bezier*

Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol, yang menentukan bentuk dan arah kurva. Cara membuatnya cukup mudah, yaitu dengan menentukan titik-titik kontrol dan menghubungkannya dengan kurva. Kurva Bézier memiliki banyak kegunaan dalam kehidupan nyata, seperti pen tool, animasi yang halus dan realistis, membuat desain produk yang kompleks dan presisi, dan membuat font yang indah dan unik. Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan.

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol  $P_0$  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.

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik  $P_0$  dan  $P_1$  yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan kurva Bézier linier. Misalkan terdapat sebuah titik  $Q_0$  yang berada pada garis yang dibentuk oleh  $P_0$  dan  $P_1$ , maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

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  $P_1$ .

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja  $P_2$ , dengan  $P_0$  dan  $P_2$  sebagai titik kontrol awal dan akhir, dan  $P_1$  menjadi titik kontrol antara. Dengan menyatakan titik  $Q_1$  terletak diantara garis yang menghubungkan  $P_1$  dan  $P_2$ , dan membentuk kurva Bézier linier yang berbeda dengan kurva letak  $Q_0$  berada, maka dapat dinyatakan sebuah titik baru,  $R_0$  yang berada diantara garis yang menghubungkan  $Q_0$  dan  $Q_1$  yang bergerak membentuk kurva Bézier kuadratik terhadap titik  $P_0$  dan  $P_2$ . Berikut adalah uraian persamaannya.

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

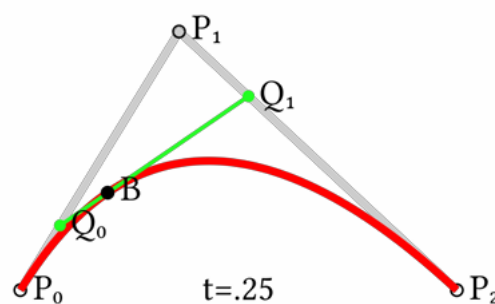
$$Q_1 = B(t) = (1 - t)P_1 + tP_2 \quad t \in [0,1]$$

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

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

$$R_0 = B(t) = (1 - t)^2P_0 + (1 - t)P_1 + t^2P_2$$

Berikut adalah ilustrasi dari kasus diatas.



*Gambar 2 Interpolasi*

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.

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

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

Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka Anda diminta untuk mengimplementasikan pembuatan kurva Bézier dengan algoritma titik tengah berbasis divide and conquer.

### III. IMPLEMENTASI ALGORITMA BRUTE FORCE

Bezier curve adalah kumpulan dari titik-titik *control points*  $P_0$  sampai  $P_n$ , dengan  $n$  adalah derajat dari kurva ( $n = 1$  untuk linear,  $n = 2$  untuk kuadratik,  $n = 3$  untuk kubik, dst). Dalam membuat bezier curve secara brute force, kami mengimplementasikannya dengan menggunakan formula

$$\begin{aligned} B(t) &= \sum_{i=0}^n \binom{n}{i} (1-t)^{n-i} t^i P_i \\ &= (1-t)^n P_0 + \binom{n}{1} (1-t)^{n-1} t P_1 + \dots + \binom{n}{n-1} (1-t) t^{n-1} P_{n-1} + t^n P_n, \quad 0 \leq t \leq 1 \end{aligned}$$

where  $\binom{n}{i}$  are the [binomial coefficients](#).

dimana  $B(t)$  menghasilkan sebuah titik pada sebuah kurva bezier pada saat  $t$ . dengan formula ini, kami bisa membuat Bezier curve dengan derajat lebih dari 2. sebagai contoh, saat derajat = 2, maka akan dibentuk kuadratik bezier curve, maka dengan melakukan substitusi terhadap  $n$ , kita bisa mendapatkan

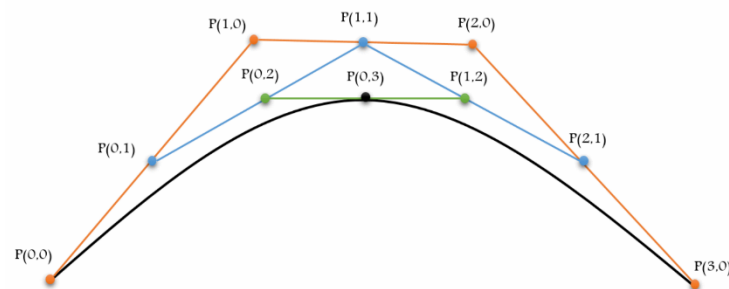
$$B(t) = (1-t)^2 P_0 + 2(1-t)t P_1 + t^2 P_2, \quad 0 \leq t \leq 1.$$

Implementasi algoritma brute force ini pada program kami (ditulis dalam bahasa pemrograman python) dilakukan dengan langkah-langkah sebagai berikut.

1. Diberikan beberapa *control points* dari  $P_0$  sampai  $P_n$ , dan jumlah iterasi yang ingin dilakukan. Dalam implementasi algoritma brute force ini, sebenarnya jumlah iterasi tidak secara langsung berpengaruh, tetapi jumlah iterasi digunakan agar jumlah titik yang dihasilkan nanti akan sama dengan jumlah titik yang dihasilkan jika menggunakan algoritma divide and conquer, contoh : jika pada algoritma divide and conquer dengan 4 iterasi menghasilkan 17 titik, maka dengan algoritma brute force dengan 4 iterasi juga akan menghasilkan 17 titik.
2. Hitung jumlah titik yang akan dihasilkan, misalkan jumlah iterasi =  $i$ , maka jumlah titik =  $2^i + 1$ .
3. Cari jumlah titik yang berada di antara control points  $P_0$  dan  $P_n$ , yaitu sebanyak jumlah titik - 2, dan cari beda  $t$  untuk masing masing titik dengan beda =  $1/(\text{jumlah\_titik} + 1)$ .
4. Untuk setiap titik diantara  $P_0$  dan  $P_n$ , kalkulasi posisinya dengan menggunakan formula pada gambar 3.1, dengan  $t = i * \text{beda}$ , dengan  $i$  adalah posisi titik ke berapa  $P_0, \dots, P_i, P_{i+1}, \dots, P_n$ .

#### IV. IMPLEMENTASI ALGORITMA DIVIDE & CONQUER

Bezier curve dapat juga diimplementasikan dengan menggunakan algoritma divide and conquer, yaitu dengan cara membagi menjadi dua bagian dan mencari titik tengahnya. Ide utama dari algoritma ini adalah membagi bezier curve menjadi sangat banyak bagian, sehingga masing-masing bagian tersebut akan terlihat seperti sebuah titik karena jaraknya satu sama lain yang sangat berdekatan, setelah itu masing-masing bagian kecil tersebut akan dianalisis dan kembali digabungkan diakhir



Gambar 3 Kurva Bezier dengan empat titik kontrol

Pada gambar diatas, bezier curve memiliki 4 control points pada awalnya, yaitu  $P(0,0)$ ,  $P(1,0)$ ,  $P(2,0)$ , dan  $P(3,0)$ , dengan algoritma divide and conquer, pembentukan bezier curve membagi bezier curve tersebut menjadi 2 bagian, yaitu  $P(0,0)$ ,  $P(0,1)$ ,  $P(0,2)$ ,  $P(0,3)$  dan  $P(0,3)$ ,  $P(1,2)$ ,  $P(2,1)$ ,  $P(3,0)$ , lalu kedua bagian tersebut akan dipisahkan prosesnya, dan pada akhirnya akan digabungkan kembali. Implementasi dari algoritma divide and conquer pada bezier curve memiliki cara kerja sebagai berikut :

1. Untuk  $n$  control points, akan dicari titik paling tengah pada bezier curve, pada contoh diatas, titik tengah tersebut adalah  $P(0,3)$ .
2. Untuk mencari titik paling tengah tersebut, kita dapat lakukan dengan mencari titik tengah dari control\_points  $i$  dan control points  $i+1$ . Sebut saja kumpulan titik tengah ini dengan  $mid_1$ , pada contoh diatas,  $mid_1$  akan berisi  $P(0,1)$ ,  $P(1,1)$ ,  $P(2,1)$ .
3. Selanjutnya, cari titik tengah diantara kumpulan  $mid_1$  tadi. Kumpulan titik tengah tersebut kita sebut saja  $mid_2$ , pada contoh diatas  $mid_2$  akan berisi  $P(0,2)$  dan  $P(1,2)$ .
4. Lakukan step 2 dan 3 sampai diperoleh  $mid_n$  yang hanya berisi satu titik dalam contoh diatas yaitu  $P(0,3)$ .
5. Lakukan kembali step 1 sampai 5 sebanyak 2 kali secara rekursif , dengan menggunakan control points 1 yaitu titik awal ( $P(0,0)$ ) ditambah elemen pertama  $mid_1$  sampai  $mid_n$  . Dan

control points 2 yaitu elemen terakhir  $mid_n$  sampai  $mid_1$  ditambah titik akhir  $(P(3,0))$ .

Lakukan sebanyak jumlah iterasi yang diinginkan

6. Gabungkan masing-masing  $mid_n$  pada tiap iterasi rekursif.
7. Dalam konteks divide and conquer, step 1 sampai 5 adalah divide, yaitu memecahkan persoalan tersebut menjadi persoalan yang lebih kecil (dalam hal ini, semakin kecil berarti jarak antar titik semakin dikit, membuat bezier curve semakin akurat) dan step 6 adalah conquer, yaitu menggabungkan hasil-hasil pemrosesan yang telah di divide sebelumnya.



## V. SOURCE CODE

Program ini dibuat dengan bahasa pemrograman Python dan JS sebagai GUI website. Penggunaan python dikarenakan bahasa ini mempunyai library yang lengkap dan mudah diintegrasikan dengan website.

1. Library yang digunakan pada pemrosesan algoritma bezier curve dan inisialisasi kelas

```
1. from typing import List
2. import math
3. import time
4.
5. class Point:
6.     def __init__(self, x: float, y: float):
7.         self.x = x
8.         self.y = y
9.
10. class BezierCurve:
11.     def __init__(self, control_points: List[Point], iteration : int):
12.         self.control_points = control_points
13.         self.iteration = iteration
14.         self.result_points_dnc = []
15.         self.iter_res = []
16.         self.result_points_brutal = []
17.         self.dnc_execution_time = 0
18.         self.brutal_execution_time = 0
19.
20.
```

2. Kalkulasi bezier curve secara brute force

```
1. def createBezierBrutal(self):
2.
3.
4.     start_time = time.time()
5.     self.result_points_brutal.append(self.control_points[0])
6.     jml_titik = 2**self.iteration + 1
7.     beda = 1/(jml_titik-1)
8.     count_ctrl = len(self.control_points)
9.     for i in range(jml_titik - 2):
10.         t = (i+1) * beda
11.         x = 0
12.         y = 0
13.         for j in range(count_ctrl):
```

```

14.             x += (math.comb(count_ctrl-1,j))*((1-t)**(count_ctrl-j-
15.             1))*(t**j)*self.control_points[j].x
16.             y += (math.comb(count_ctrl-1,j))*((1-t)**(count_ctrl-j-
17.             1))*(t**j)*self.control_points[j].y
18.             point = Point(x,y)
19.             self.result_points_brutal.append(point)
20.
21.
22.             self.result_points_brutal.append(self.control_points[-1])
23.
24.             end_time = time.time()
25.             execution_time = (end_time - start_time) * 1000
26.             self.brutal_execution_time = int(execution_time)
27.

```

### 3. Kalkulasi bezier curve secara divide and conquer

```

1.     def create_bezier_dnc(self):
2.
3.         start_time = time.time()
4.
5.         self.result_points_dnc.append(self.control_points[0])
6.         self.calculate_new_point(self.control_points,0)
7.         self.result_points_dnc.append(self.control_points[-1])
8.
9.         end_time = time.time()
10.        execution_time = (end_time - start_time) * 1000
11.        self.dnc_execution_time = int(execution_time)
12.
13.
14.    def calculate_new_point(self,ctrl_points : List[Point],current_iteration : int):
15.        if current_iteration < self.iteration:
16.            temp = ctrl_points
17.            res = []
18.            while len(temp) > 0:
19.                res.append(temp)
20.                temp2 = []
21.                for i in range(len(temp)-1):
22.                    mid = self.get_mid_point(temp[i],temp[i+1])
23.                    temp2.append(mid)
24.                temp = temp2
25.

```

```

26.         param1 = []
27.         param2 = []
28.         for i in range(len(res)):
29.             param1.append(res[i][0])
30.             param2.append(res[len(res)-1-i][-1])
31.
32.         current_iteration += 1
33.         self.calculate_new_point(param1,current_iteration)
34.         self.result_points_dnc.append(res[-1][0])
35.         self.calculate_new_point(param2,current_iteration)
36.

```

#### 4. Pembuatan array untuk titik-titik di setiap iterasi

```

1. def calculate_iter_res(self):
2.     for i in range(self.iteration):
3.         temp = []
4.         to_append = 2*(i+1) -1
5.         step = len(self.result_points_dnc) / (to_append+1)
6.         for j in range(0,len(self.result_points_dnc),math.floor(step)):
7.             temp.append(self.result_points_dnc[j])
8.         self.iter_res.append(temp)
9.

```

#### 5. Fungsi getter dan print

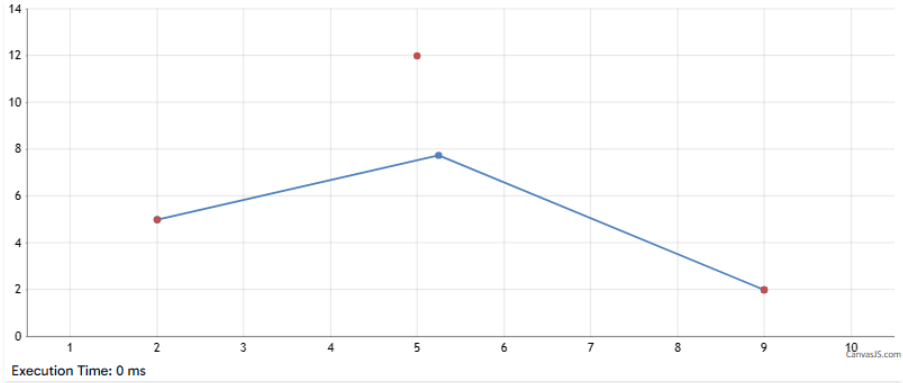
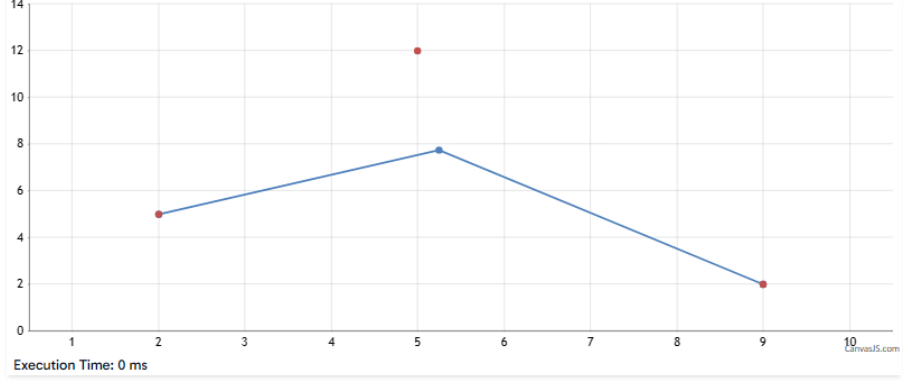
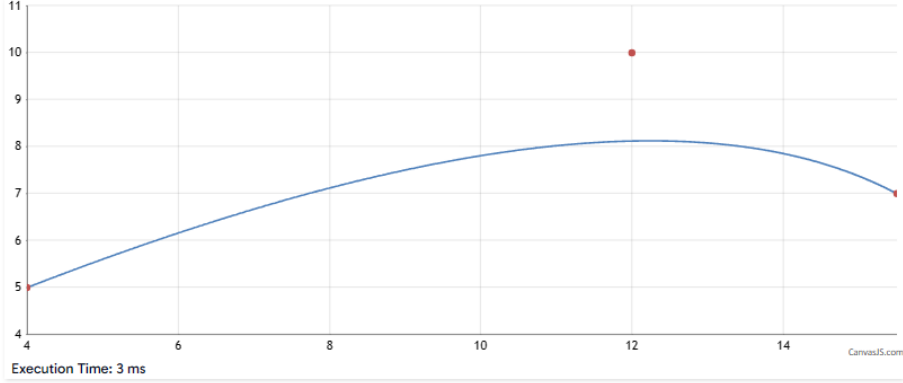
```

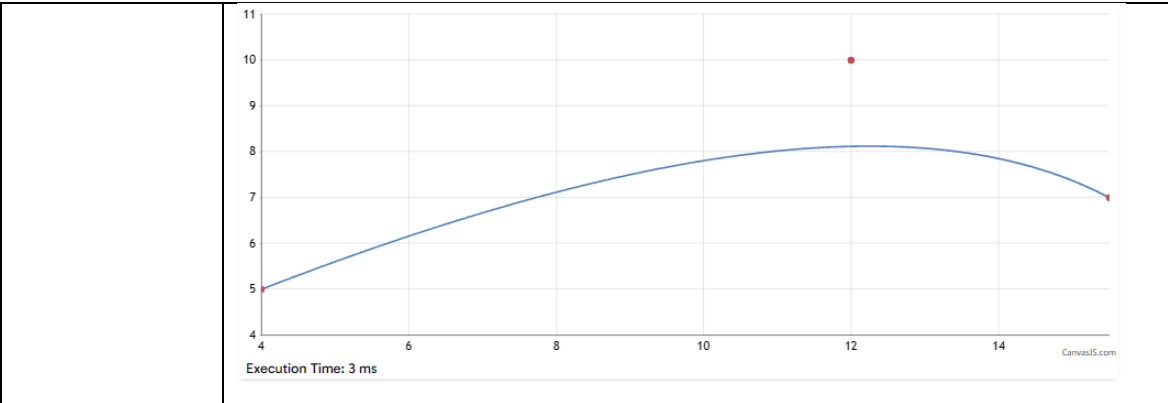
1. def get_mid_point(self, point1 : Point , point2 : Point) -> Point :
2.     return Point((point1.x + point2.x) / 2, (point1.y + point2.y) / 2)
3.
4. def print_points(self, list : List[Point]):
5.     for point in list:
6.         print(point.x,point.y)
7.
8. def print_dnc_points(self):
9.     self.print_points(self.result_points_dnc)
10.
11. def print_brutal_points(self):
12.     self.print_points(self.result_points_brutal)
13.
14. def print_iter_res(self):
15.     for i in range(len(self.iter_res)):
16.         print("Iterasi ke-",i+1)
17.         self.print_points(self.iter_res[i])
18.
19. def get_result_points_dnc(self):

```

```
20.         return self.result_points_dnc
21.
22.     def get_result_points_brutal(self):
23.         return self.result_points_brutal
24.
25.     def get_control_points(self):
26.         return self.control_points
27.
28.     def get_iter_res(self):
29.         return self.iter_res
30.
31.     def get_jumlah_control_points(self):
32.         return len(self.control_points)
33.
34.     def get_time_dnc(self):
35.         return self.dnc_execution_time
36.
37.     def get_time_brutal(self):
38.         return self.brutal_execution_time
39.
```

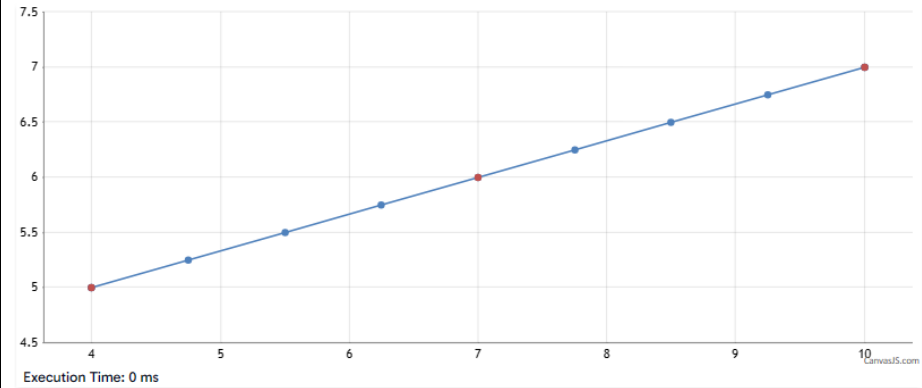
## VI. HASIL PENGUJIAN

Data	Hasil
Titik: 3 Iterasi: 1 (2,5), (5,12), (9,2)	Divide & Conquer: 
	Brute Force: 
Titik: 3 Iterasi: 10 (4,5), (12,10), (15.5,7)	Divide & Conquer: 
	Brute Force:

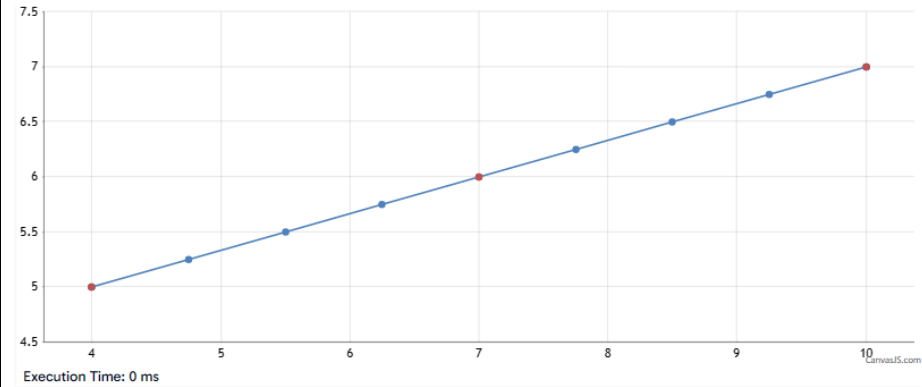


Titik: 3  
Iterasi: 3  
(4,5),(7,6),  
(10,7)

Divide & Conquer:

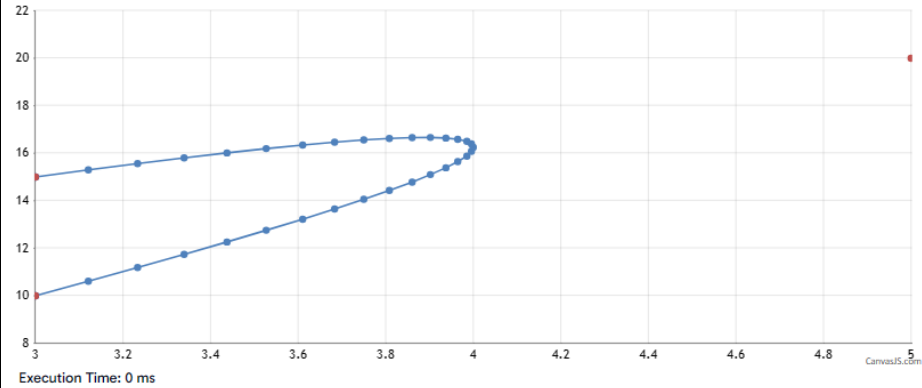


Brute Force:

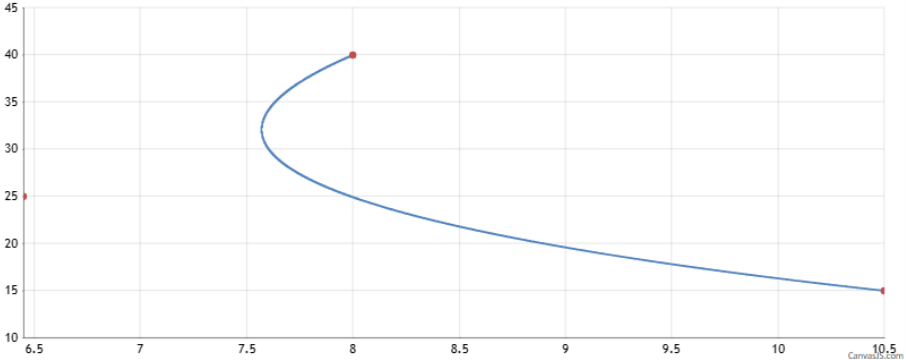
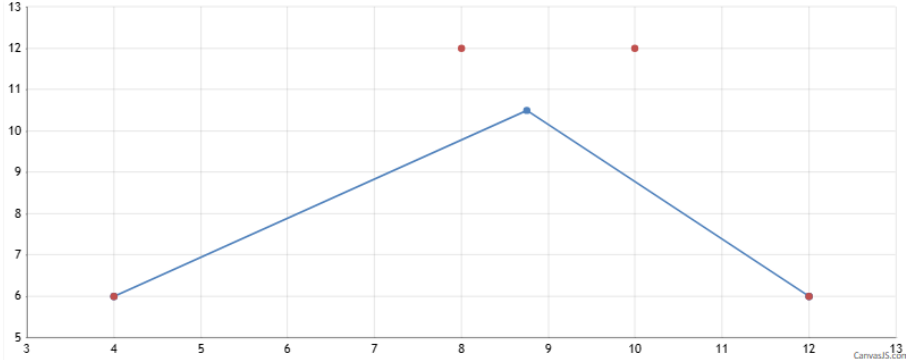
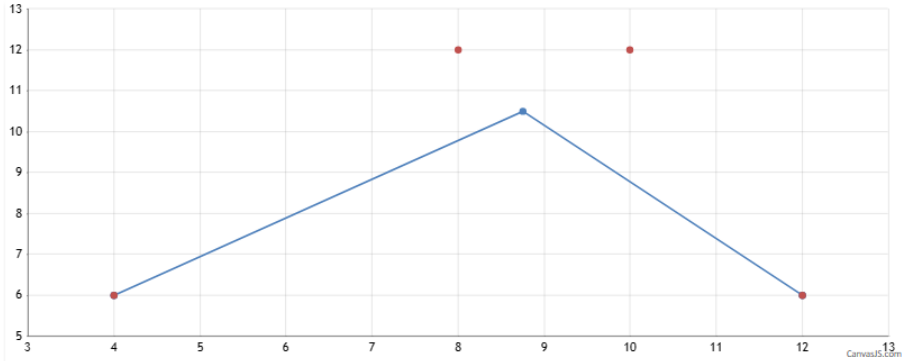
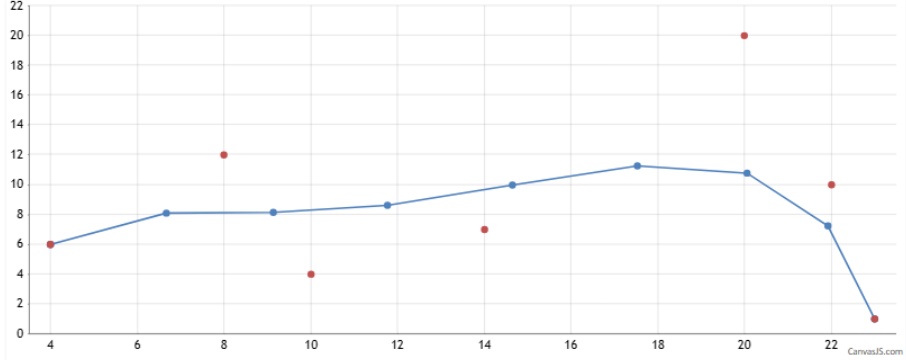


Titik: 3  
Iterasi: 5  
(3,10),(5,20),  
(3,15)

Divide & Conquer:

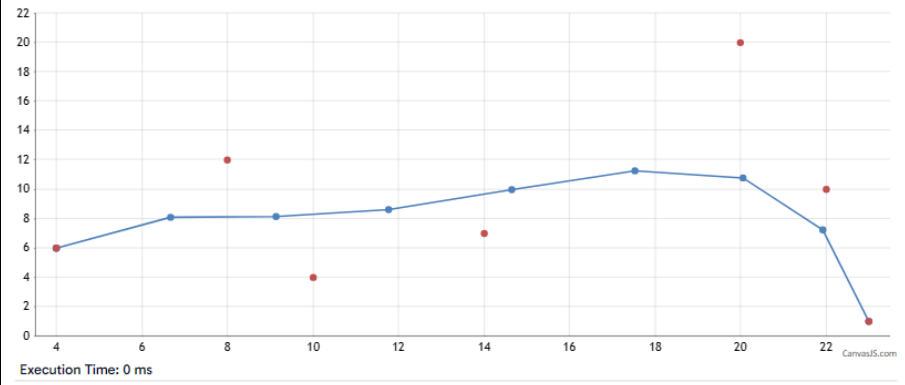


	<p>Brute Force:</p> <p>Execution Time: 0 ms</p>
<p>Titik: 3 Iterasi: 20 (7.23,10),(1,3), (-5.45,15)</p>	<p>Divide &amp; Conquer:</p> <p>Execution Time: 3373 ms</p>
	<p>Brute Force:</p> <p>Execution Time: 3188 ms</p>
<p>Titik: 3 Iterasi: 15 (8,40), (6.45,25), (10.5,15)</p>	<p>Divide &amp; Conquer:</p> <p>Execution Time: 99 ms</p>

	<p>Brute Force:</p>  <p>Execution Time: 88 ms</p>
<p>Titik: 4 Iterasi: 1 (4,6),(8,12), (10,12),(12,6)</p>	<p>Divide &amp; Conquer:</p>  <p>Execution Time: 0 ms</p>
	<p>Brute Force:</p>  <p>Execution Time: 0 ms</p>
<p>Titik: 7 Iterasi: 3 (4,6),(8,12), (10,4),(14,7), (20,20),(22,10), (23,1)</p>	<p>Divide &amp; Conquer:</p>  <p>Execution Time: 0 ms</p>



### Brute Force:



## VII. ANALISIS KASUS

Pendekatan brute force maupun divide & conquer sama – sama menghasilkan kurva bezier yang serupa. Setiap uji coba yang dilakukan selalu memperoleh bentuk kurva yang sama serta dengan titik – titik yang sama. Semakin besar iterasi yang dilakukan oleh algoritma, grafik yang dihasilkan akan lebih “melengkung” atau “halus” sehingga menciptakan beban yang besar kepada perhitungan. Pada iterasi yang kecil, seperti iterasi dengan nilai satu sehingga lima, perbedaan eksekusi waktu yang dihasilkan kedua pendekatan tidaklah signifikan. Namun, semakin meningkat nilai iterasi serta jumlah titik yang dihitung, waktu yang ditempuh oleh kedua pendekatan mengalami peningkatan yang signifikan juga.

Dalam implementasinya (source code 5.2), pendekatan bruteforce mencari posisi titik sebanyak  $2^n - 1$  dengan n adalah jumlah iterasi, lalu dalam proses mencari satu titik dilakukan looping sebanyak c kali, dengan c adalah jumlah titik kontrol, di setiap looping tersebut dilakukan operasi kombinasi dan perpangkatan terhadap c yang dimana kombinasi terhadap c dan perpangkatan terhadap c memiliki kompleksitas c. Sehingga total kompleksitas algoritma brute force adalah

$$T(n) = (2^n - 1).c^2 = O(2^n . c^2)$$

Pada pendekatan algoritma divide and conquer (source code 5.3), program akan melakukan pemanggilan fungsi rekursi calculate\_new\_points sebanyak  $2^n$  kali dengan n adalah jumlah iterasi, di setiap pemanggilan fungsi rekursi dilakukan while loop sebanyak c kali dengan c adalah jumlah titik kontrol, lalu didalam while loop tersebut dilakukan for loop sebanyak temp, dengan temp nilainya sebanyak iterasi while loop saat ini. Setelah itu akan dilakukan for loop sebanyak c untuk melakukan append sebagai parameter rekursi selanjutnya. Maka akan didapat kompleksitas algoritma divide and conquer adalah

$$T(n) = (2^n) \cdot \frac{c(c+1)}{2} + c = O(2^n . c^2)$$

Dari hasil kedua perhitungan ini, dapat disimpulkan bahwa algoritma brute force dan divide and conquer memiliki kompleksitas Big O yang sama, yaitu  $O(2^n . c^2)$ . kompleksitas ini membuat waktu pemrosesan pembuatan bezier curve meningkat secara eksponensial sesuai dengan jumlah iterasinya. Hal ini dibuktikan dengan peningkatan yang sangat jauh ketika dilakukan pemrosesan dengan jumlah iterasi 15 dan 20, dengan jumlah iterasi 15 membutuhkan waktu 99ms dan jumlah iterasi 20 membutuhkan waktu 3373ms.

Perbedaan waktu eksekusi algoritma brute force dan divide and conquer yang tidak terlalu signifikan bisa jadi disebabkan karena beberapa hal, yaitu pada algoritma divide and conquer, kami menggunakan rekursi sedangkan pada brute force menggunakan for loop, penggunaan rekursi terbukti memakan waktu lebih lama daripada for loop pada bahasa python pada kompleksitas yang

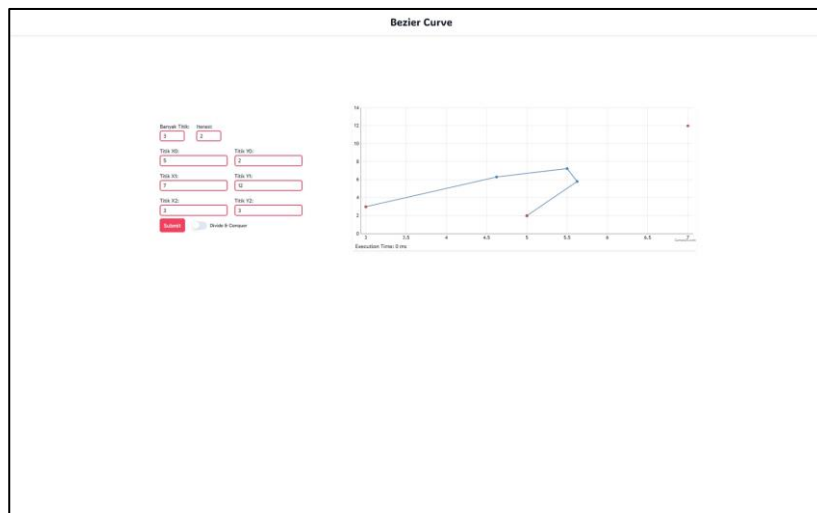
sama. Selain itu, terdapat faktor lain yaitu penggunaan fungsi bawaan python saat melakukan operasi kombinasi dan perpangkatan pada brute force yang menurut sumber di internet memerlukan kompleksitas sebesar  $n$ , namun bisa saja operasi bawaan tersebut telah dioptimasi oleh python.

## VIII. IMPLEMENTASI BONUS

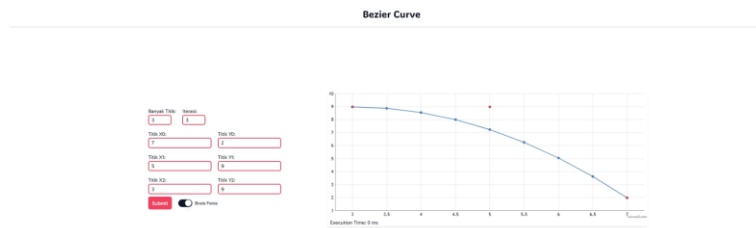
Program implementasi bezier curve ini melakukan proses inputnya dalam bentuk GUI berbasis web. Input yang disediakan adalah banyak poin, iterasi, dan nilai poin-poinnya. Hasil dari perhitungan algoritma, baik dengan brute force maupun divide & conquer, akan ditampilkan didalam interface website.



Gambar 4 Tampilan antarmuka



Gambar 5 Tampilan Divide & Conquer



*Gambar 6 Tampilan Brute Force*

Stack yang dipakai dalam pembuatan dari GUI berbasis website ini adalah menggunakan Framework Next JS dengan Flask API Python. Stack ini dipilih dikarenakan mempermudah kami untuk memadukan Python dengan JavaScript sehingga segala perhitungan dapat dilakukan oleh Python dan penampilan dapat dilakukann oleh JavaScript, HTML, dan CSS.

Program ini juga mengimplementasikan pembuatan kurva untuk n titik kontrol yang penjelasannya sudah terdapat pada bab 3.

## IX. LAMPIRAN

Link menuju repositori program : [https://github.com/raflyhangga/Tucil2\\_13522014\\_13522095](https://github.com/raflyhangga/Tucil2_13522014_13522095)

Poin	Ya	Tidak
Program berhasil dijalankan.	X	
Program dapat melakukan visualisasi kurva Bézier.	X	
Solusi yang diberikan program optimal.	X	
<b>[Bonus]</b> Program dapat membuat kurva untuk n titik kontrol.	X	
<b>[Bonus]</b> Program dapat melakukan visualisasi proses pembuatan kurva.		X