

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

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



**Anggota**

**13522019 Wilson Yusda**

**13522055 Benardo**

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

# DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>DAFTAR GAMBAR.....</b>	<b>4</b>
<b>BAB 1 DESKRIPSI TUGAS.....</b>	<b>5</b>
<b>BAB II LANDASAN TEORI.....</b>	<b>6</b>
Algoritma Divide And Conquer.....	6
Algoritma Brute Force.....	6
Kurva Bezier.....	7
<b>BAB III IMPLEMENTASI PROGRAM.....</b>	<b>10</b>
Analisis dan Implementasi Algoritma Brute Force.....	10
Analisis dan Implementasi Algoritma Divide And Conquer.....	11
<b>BAB IV SOURCE CODE.....</b>	<b>16</b>
Algoritma Brute Force.....	16
Algoritma Divide And Conquer.....	18
Algoritma GUI dan Animasi.....	20
<b>BAB V TESTING PROGRAM.....</b>	<b>28</b>
Test Case 1.....	28
Test Case 2.....	28
Test Case 3.....	29
Test Case 4.....	29
Test Case 5.....	30
Test Case 6.....	30
Test Case 7.....	31
Test Case 8.....	31
<b>BAB VI ANALISIS PERBANDINGAN.....</b>	<b>33</b>
Analisis Kompleksitas Algoritma Brute Force.....	33
Analisis Kompleksitas Algoritma Divide and Conquer.....	33
Analisis Perbandingan Kompleksitas Algoritma antara Algoritma Divide and Conquer dengan Algoritma Bruteforce.....	34
<b>BAB VII IMPLEMENTASI BONUS.....</b>	<b>36</b>
Bonus generalisasi algoritma sehingga program dapat membentuk kurva Bezier kubik, kuartik dan selanjutnya dengan 4,5,6, hingga n titik kontrol.....	36
Bonus Visualisasi Proses Pembentukan Kurva.....	36
<b>BAB VIII KESIMPULAN DAN SARAN.....</b>	<b>39</b>
Kesimpulan.....	39
Saran.....	39

<b>BAB IX LAMPIRAN.....</b>	<b>40</b>
Link Repository.....	40
Program Checkpoint.....	40

## DAFTAR GAMBAR

Gambar 2.1 Ilustrasi Kasus Kurva Bezier.....	8
Gambar 3.1. Proses Divide And Conquer.....	11
Gambar 3.2 Hasil pencarian titik tengah pada iterasi pertama.....	12
Gambar 3.3.. Hasil divide terhadap titik tengah yang dihasilkan pada iterasi sebelumnya.	13
Gambar 3.4. Conquer pada upa-persoalan yang dihasilkan dari divide sebelumnya.....	14
Gambar 3.5. Hasil combine dari solusi pada upa-persoalan kiri dan kana (iterasi kedua)...	14
Gambar 3.6. Ilustrasi divide dan conquer untuk menghasilkan titik titik pada kurva biezer dengan iterasi berjumlah dua.....	15
Gambar 4.1. Source Code Fungsi Binomial.....	16
Gambar 4.2. Source Code Fungsi Pembentukan Kurva Bezier (Brute Force).....	17
Gambar 4.3. Source Code Fungsi Pembentukan Kurva Bezier (Divide And Conquer)....	18
Gambar 4.4. Source Code Fungsi Find Bezier Points (Divide And Conquer).....	19
Gambar 4.5. Source Code mid_point (Divide And Conquer).....	20
Gambar 4.6. Source Code Inisialisasi Library Interface (GUI).....	20
Gambar 4.7. Source Code Fungsi Pembentukan Fitur Komponen (GUI).....	21
Gambar 4.8. Source Code Onclick / Fungsi Utama GUI.....	22
Gambar 5.1. Test Case 1 dengan 3 titik kontrol.....	28
Gambar 5.2. Test case 2 dengan 3 titik kontrol.....	29
Gambar 5.3. Test Case 3 dengan 4 titik kontrol.....	29
Gambar 5.4. Test Case 4 dengan 4 titik kontrol.....	30
Gambar 5.5. Test Case 5 dengan 4 titik kontrol.....	30
Gambar 5.6. Test Case 6 dengan 5 titik kontrol.....	31
Gambar 5.7. Test Case 7 dengan 6 titik kontrol.....	31
Gambar 5.8. Test Case 8 dengan 7 titik kontrol.....	32
Gambar 8.1 Tampilan input error.....	38

## **BAB 1**

### **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. 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.

Tugas Kecil 2 pada mata kuliah Strategi Algoritma mengajak mahasiswa untuk mengembangkan program sederhana dalam bahasa C++, Python, JavaScript, atau Go. Program ini bertujuan untuk membentuk sebuah kurva Bézier kuadratik dengan memanfaatkan algoritma titik tengah berbasis divide and conquer. Sebagai tantangan tambahan, mahasiswa juga diminta untuk mengimplementasikan solusi dengan algoritma brute force sebagai pembanding terhadap solusi divide and conquer.

Adapun input yang diperlukan untuk program ini meliputi n pasangan titik yang akan menentukan bentuk kurva Bézier. Titik pertama dan terakhir yang dimasukkan akan menjadi titik awal dan akhir kurva. Selain itu, jumlah iterasi yang ingin dilakukan juga perlu ditentukan sebagai input.

Output yang diharapkan dari program ini adalah kurva Bézier yang terbentuk pada iterasi terkait dan waktu eksekusi program dalam pembentukan kurva tersebut. Mahasiswa diberi kebebasan dalam memilih alat visualisasi yang digunakan untuk menampilkan hasil kurva Bézier.

## BAB II

# LANDASAN TEORI

### **Algoritma Divide And Conquer**

Divide and Conquer (DnC) adalah sebuah pendekatan algoritmik yang penting dalam pemrograman dan ilmu komputer. Pendekatan ini melibatkan tiga langkah utama: membagi masalah menjadi sub-masalah yang lebih kecil (divide), menyelesaikan masing-masing sub-masalah secara rekursif (conquer), dan menggabungkan solusi dari sub-masalah untuk mendapatkan solusi masalah asli (combine):

- Divide: membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama)
- Conquer: menyelesaikan (solve) masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar).
- Combine: menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

Pendekatan Divide and Conquer merupakan teknik algoritmik yang sangat berguna, terutama untuk masalah yang dapat dibagi menjadi sub-masalah yang lebih kecil. Dengan memecah masalah menjadi bagian yang lebih kecil, DnC sering kali dapat menemukan solusi yang lebih efisien daripada pendekatan brute force. Namun, penting untuk mempertimbangkan overhead rekursi dan kebutuhan memori ketika mengimplementasikan algoritma DnC.

### **Algoritma Brute Force**

Algoritma brute force, juga dikenal sebagai pendekatan "kekuatan kasar" atau "pencarian lengkap", adalah metode pemecahan masalah komputasi yang sangat dasar dan langsung. Algoritma brute force mengandalkan kekuatan komputasi dan cakupan yang luas untuk mengeksplorasi setiap kemungkinan solusi dalam ruang pencarian. Algoritma ini tidak memilih-milih dan tidak menggunakan prasangka atau pengetahuan sebelumnya untuk mengoptimalkan pencarian.

1. Eksplorasi Semua Kemungkinan

Algoritma brute force akan menelusuri semua kemungkinan solusi yang ada tanpa menghilangkan satu pun, hingga menemukan solusi yang benar atau terbaik.

## 2. Penerapan Sederhana

Algoritma ini biasanya mudah diimplementasikan dan mudah dipahami karena hanya memerlukan iterasi atas semua kemungkinan solusi tanpa membutuhkan strategi khusus atau heuristik.

## 3. Kekurangan

Kinerja algoritma brute force sering kali kurang efisien dibandingkan dengan algoritma yang lebih cerdas, terutama untuk masalah dengan ruang pencarian yang besar.

## 4. Kompleksitas Waktu

Kompleksitas waktu untuk algoritma brute force biasanya eksponensial terhadap ukuran masalah, yang berarti ruang pencarian yang semakin besar akan memperlambat pemrosesan algoritma brute force.

## 5. Optimalitas

Algoritma ini menjamin penemuan solusi (jika ada) karena mencakup semua kemungkinan solusi, asalkan waktu dan sumber daya komputasi yang diberikan mencukupi.

## **Kurva Bezier**

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.

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

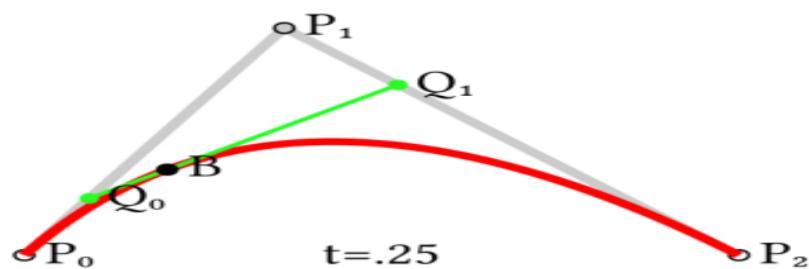
$$Q_1 = B(t) = (1 - t)P_1 + tP_2, \quad t \in [0, 1]$$

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

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

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

Berikut adalah ilustrasi dari kasus diatas.



**Gambar 2.1 Ilustrasi Kasus Kurva Bezier**

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 kuatrik, dan

seterusnya. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya

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

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

## **BAB III**

### **IMPLEMENTASI PROGRAM**

#### **Analisis dan Implementasi Algoritma Brute Force**

Pada pengerjaan tugas kecil ini, algoritma brute force digunakan sebagai pembanding efisiensi dengan algoritma *Divide And Conquer*. Secara umum, algoritma *Brute Force* menerapkan formula yang telah disediakan diatas dan diiterasi sesuai kebutuhan. Beberapa komponen penting dalam pembentukan algoritma *Brute Force* yakni berupa:

##### **1. Binomial**

Implementasi binomial diperlukan guna mencari koefisien yang tepat untuk formula yang digunakan, dimana untuk setiap *control points*, maka akan memiliki formula yang unik pula.

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

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

Gambar diatas memperlihatkan perbedaan atas perbedaan *control points*. Ketika jumlah control points bernilai 4, maka koefisien untuk P0,P1,P2, dan P3 berturut turut adalah 1,3,3,1. Sedangkan jika *control points* bernilai 5, maka koefisien pada P0, P1, P2, P3, dan P4 berturut-turut adalah 1,4,6,4,1. Urutan ini memenuhi syarat serta hasil dari kombinasi dimana

$$Pk\ Coefficient = \frac{n!}{k!(n-k)!}$$

Dengan n sebagai jumlah *control points* -1 dan k sebagai urutan titik yang ada dengan k = 0,1,2,..(n-1).

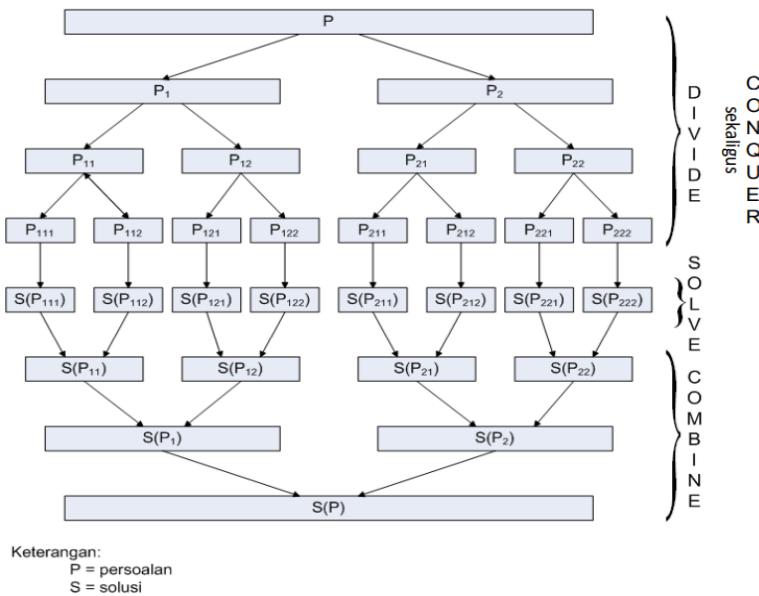
##### **2. Pembentukan Bezier**

Pembentukan titik Bezier secara *Brute Force* membutuhkan seluruh *control points* dan iterasi yang diinginkan. Pembentukan titik Bezier in mengimplementasikan pendekatan brute force untuk menghitung titik-titik pada kurva Bézier. Ini dilakukan dengan cara langsung menghitung nilai-nilai titik kurva menggunakan persamaan kurva Bézier untuk sejumlah nilai parameter t yang berbeda. Pendekatan ini secara eksplisit mengevaluasi

persamaan kurva untuk setiap nilai  $t$  yang dibutuhkan dengan mempertimbangkan iterasi yang dibutuhkan, daripada menggunakan metode yang lebih efisien atau cerdas untuk menghasilkan titik-titik kurva. Jumlah perhitungan terhadap iterasi yang ada yakni sebanyak 2 dipangkatkan dengan jumlah iterasi ditambah 1 guna memastikan titik awal dan titik akhir terhitung dalam titik Bezier.

### Analisis dan Implementasi Algoritma Divide And Conquer

Algoritma Divide and Conquer dulunya adalah strategi militer yang dikenal dengan nama divide ut imperies. Divide and Conquer terbagi menjadi dua kata yaitu divide dan conquer. Divide artinya membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-upa persoalan berukuran hampir sama). Conquer artinya menyelesaikan masalah masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar). Setelah melakukan divide and conquer , barulah diterapkan combine yaitu menggabungkan solusi masing-masing upa persoalan sehingga membentuk solusi persoalan semula.

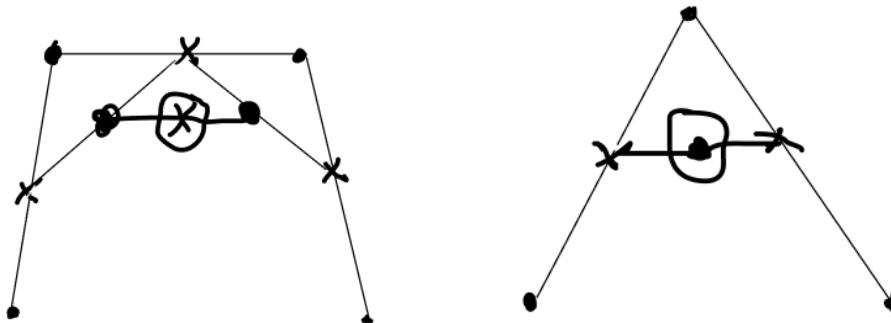


Gambar 3.1. Proses Divide And Conquer

Dengan menggunakan konsep seperti ini , maka konsep ini dapat digunakan untuk menentukan titik - titik yang dapat menyusun kurva bezier dengan menggunakan titik-titik kontrol yang telah ditentukan. Kurva Bezier yang dapat dibentuk dapat terdiri dari n titik kontrol dengan menerapkan algoritma divide and conquer. Adapun tahap-tahap prosedur menentukan titik-titik kurva Bezier pada persoalan sebagai berikut.

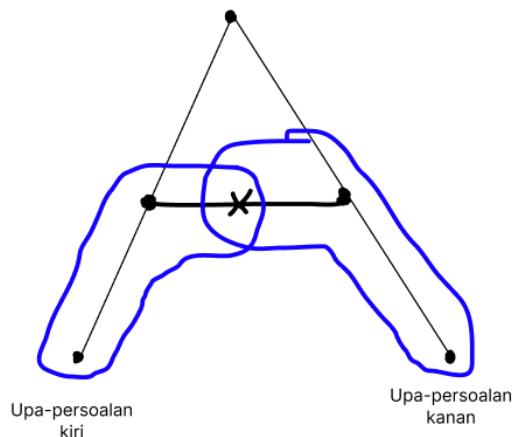
1. **Conquer** : Pertama, akan dilakukan *conquer* (solve) terlebih dahulu terhadap titik-titik kontrol yang dimasukkan dengan mencari titik tengah dari titik -titik kontrol yang bersebelahan. Kemudian , dari titik tengah tersebut akan dicari titik tengah baru dari garis yang menghubungkan titik tengah lama. Proses ini akan berlangsung sampai memperoleh satu titik tengah . Setelah memperoleh titik tengah dari titik tengah ini, barulah bisa dibilang iterasi pertama telah selesai. Untuk ilustrasi yang lebih lengkap dapat melihat gambar dibawah ini.

Sebagai tambahan analisis, pada tahap 'conquer' dari algoritma divide and conquer untuk pembuatan kurva Bézier, dilakukan proses iteratif yang mengulang sebanyak dua kali, yang secara umum dapat dianggap sebagai penerapan pendekatan brute-force. Ini dilakukan untuk menentukan titik tengah antara titik tengah yang tidak diketahui, berdasarkan jumlah titik kontrol yang disediakan. Dalam hal ini, algoritma menggunakan perulangan yang berulang dua kali: perulangan pertama untuk menemukan titik tengah dari titik-titik kontrol, dan perulangan kedua untuk menemukan titik tengah dari titik-titik tengah tersebut, yang akan menjadi titik pembagi kurva menjadi dua upa-persoalan. Jumlah langkah yang diperlukan hanya untuk tahap 'conquer' adalah  $\left( \sum_{1}^{n-1} (n) \right)$  dengan n adalah jumlah titik kontrol. Oleh karena itu, kompleksitas waktu dari tahap ini adalah  $O(n^2)$ .



**Gambar 3.2 Hasil pencarian titik tengah pada iterasi pertama**

2. Divide : Setelah mendapatkan titik tengah pada iterasi pertama (yang dilingkar pada gambar 3.2.2 ), kemudian akan dilakukan pembagian upa-persoalan menjadi lebih kecil yaitu upa-persoalan kanan dan upa-persoalan kiri. Hasil pembagian dapat dilihat pada gambar 3.2.3 (ditandai dengan lingkaran yang membagi titik tengah hasil iterasi sebelumnya untuk dicari titik tengah dari upa-persoalan ini). Setelah dibagi, akan dilakukan rekursif sebanyak dua kali dengan menjadikan upa-persoalan kanan dan upa-persoalan kiri masing-masing menjadi parameter pada pemanggilan fungsi rekursif tersebut.



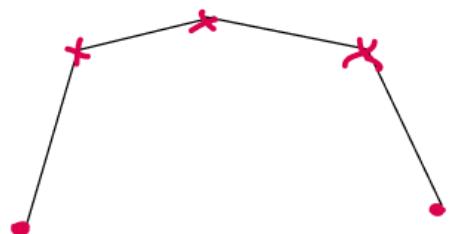
**Gambar 3.3.. Hasil divide terhadap titik tengah yang dihasilkan pada iterasi sebelumnya**

3. **Conquer upa-persoalan** : Setelah melakukan divide, hasil divide atau upa-persoalan yang diperoleh akan dilakukan solve atau conquer lagi untuk mencari titik tengah dari titik-titik tengah. Proses ini sama seperti conquer pada langkah pertama , namun pada conquer pada langkah ketiga ini bergantung kepada jumlah iterasi yang ditentukan oleh pengguna. Jika jumlah iterasi yang telah dilakukan sudah sama dengan jumlah iterasi yang ditentukan oleh pengguna, maka akan proses conquer ini tidak akan dilakukan dan akan dilakukan ke tahap selanjutnya yaitu combine. Ilustrasi conquer upa-persoalan ini adalah sebagai berikut.



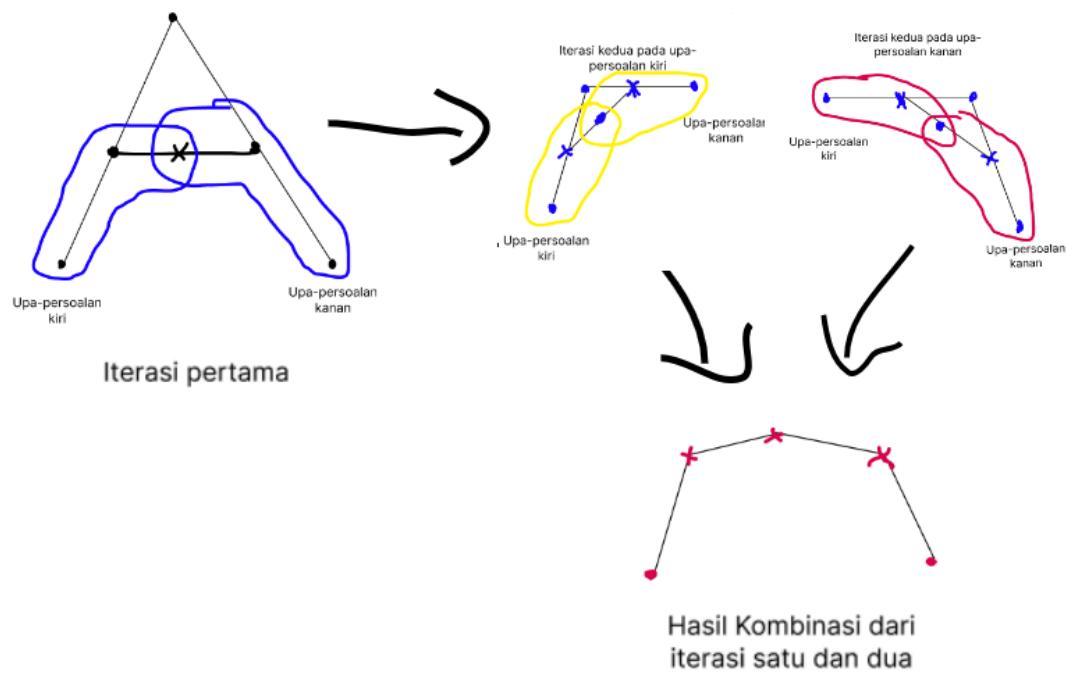
**Gambar 3.4. Conquer pada upa-persoalan yang dihasilkan dari divide sebelumnya**

4. **Combine** : Apabila jumlah iterasi yang telah dilakukan sudah sama dengan jumlah iterasi yang ditentukan oleh pengguna, maka proses divide dan conquer akan dihentikan. Solusi -solusi dari titik tengah yang telah diperoleh dari iterasi pertama sama terakhir akan di combine atau digabung dalam suatu variabel, yang akan dikembalikan oleh fungsi sebagai titik-titik yang akan digunakan untuk membuat kurva bizer. Hasil dari combine ini akan menyimpan titik-titik berurut dari awal sampai akhir iterasi. Ilustrasi terhadap combine dapat dilihat pada gambar berikut.



**Gambar 3.5. Hasil combine dari solusi pada upa-persoalan kiri dan kana (iterasi kedua)**

Untuk ilustrasi yang lebih jelas dapat dilihat gambar sebagai berikut.



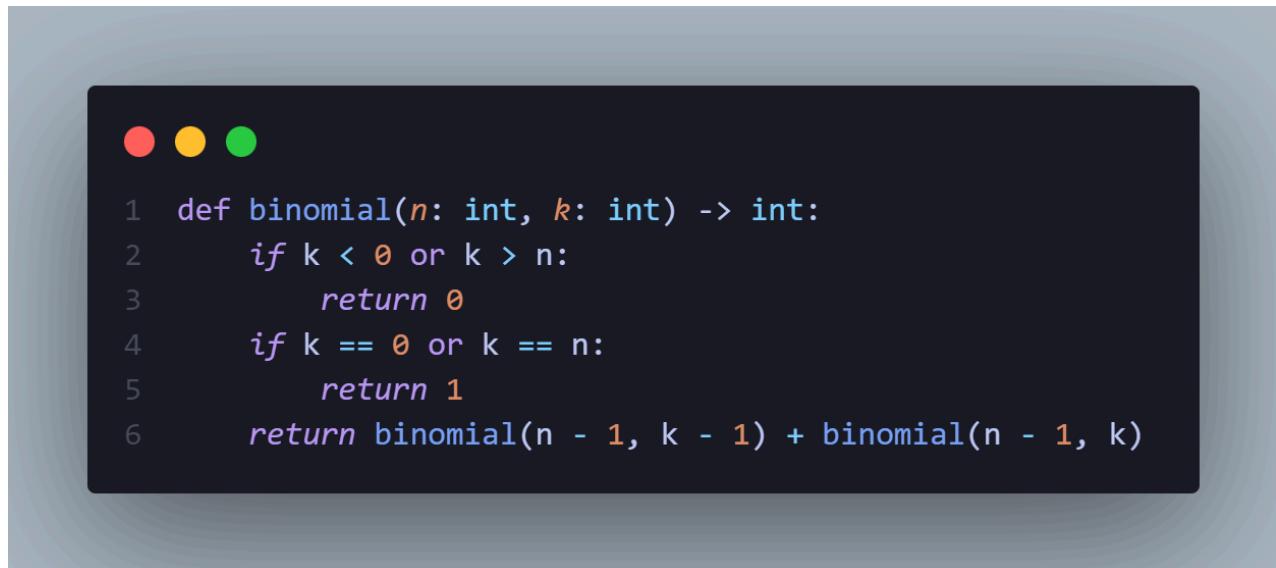
**Gambar 3.6. Ilustrasi** divide and conquer untuk menghasilkan titik titik pada kurva biezer dengan iterasi berjumlah dua.

Jumlah titik yang dihasilkan dari proses divide and conquer ini bergantung kepada jumlah iterasi yang dipilih. Semakin tinggi jumlah iterasi, maka titik yang dihasilkan akan semakin banyak, sehingga kurva bezier yang dihasilkan akan lebih halus bila dibandingkan kurva bezier dengan iterasi yang lebih rendah.

## BAB IV

### SOURCE CODE

#### Algoritma Brute Force



```
● ● ●

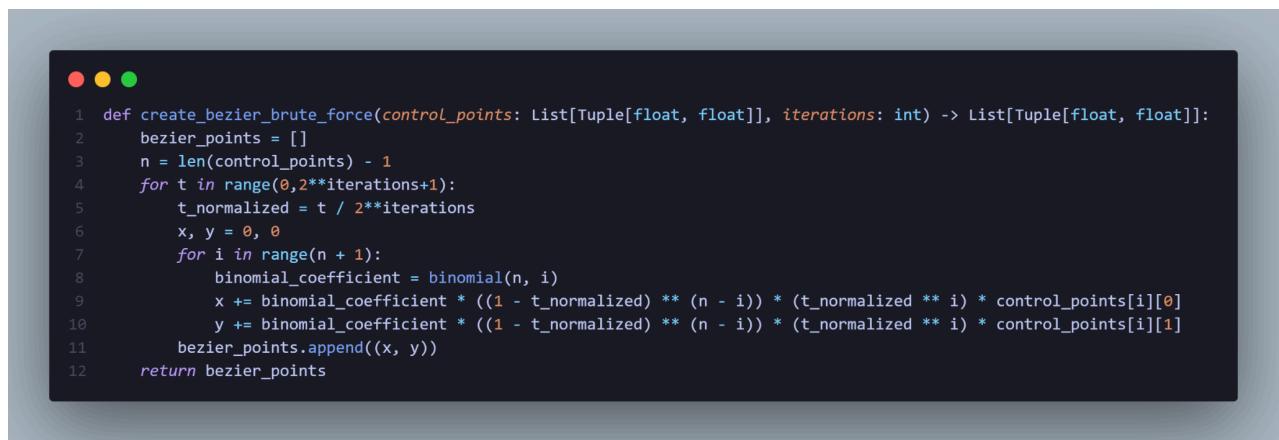
1 def binomial(n: int, k: int) -> int:
2     if k < 0 or k > n:
3         return 0
4     if k == 0 or k == n:
5         return 1
6     return binomial(n - 1, k - 1) + binomial(n - 1, k)
```

**Gambar 4.1. Source Code Fungsi Binomial**

Fungsi binomial dalam kode ini digunakan untuk menghitung koefisien binomial, yang merupakan angka yang muncul sebagai koefisien dalam persamaan pembentukan  $B(t)$  pada kurva Bezier. Fungsi ini mengimplementasikan rumus rekursif untuk koefisien binomial:

$$C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$$

Kondisi unik yang langsung dipertimbangkan dalam pengembalian nilai yakni ketika inputan bersifat  $C(n,0)$  ataupun  $C(n,n)$  (ketika  $k = n$ ) dimana nilai hasil dari kombinasi tersebut adalah 1. Kemudian juga diperiksa jika memang inputan invalid dimana nilai  $k < 0$  atau nilai  $k > n$  maka akan memberikan hasil 0.



```
1 def create_bezier_brute_force(control_points: List[Tuple[float, float]], iterations: int) -> List[Tuple[float, float]]:
2     bezier_points = []
3     n = len(control_points) - 1
4     for t in range(0, 2**iterations+1):
5         t_normalized = t / 2**iterations
6         x, y = 0, 0
7         for i in range(n + 1):
8             binomial_coefficient = binomial(n, i)
9             x += binomial_coefficient * ((1 - t_normalized) ** (n - i)) * (t_normalized ** i) * control_points[i][0]
10            y += binomial_coefficient * ((1 - t_normalized) ** (n - i)) * (t_normalized ** i) * control_points[i][1]
11        bezier_points.append((x, y))
12    return bezier_points
```

**Gambar 4.2. Source Code Fungsi Pembentukan Kurva Bezier (Brute Force)**

Fungsi *create\_bezier\_brute\_force* menerima *control points* dan jumlah iterasi dan mengembalikan titik bezier sesuai inputan pengguna

Cara kerja fungsi:

1. Inisialisasi

Inisialisasi list kosong tempat menampung titik bezier yang nantinya akan digunakan dalam proses visualisasi dan juga nilai n dimana untuk n sama dengan jumlah control points dikurangi 1. (Pk dengan k dimulai dari 0 dalam rumus binomial atau segitiga pascal).

2. Iterasi

Fungsi melakukan iterasi sebanyak  $2^{\text{iterations}} + 1$  kali. Untuk setiap iterasi, nilai t dinormalisasi (disebut *t\_normalized*) agar berada dalam rentang  $[0, 1]$ . Nilai t ini merepresentasikan posisi titik pada kurva Bézier. Tujuan dari permulaan dari 0 serta penambahan 1 di akhir diperlukan agar titik awal dan titik akhir masih berada dalam hasil titik Bezier.

3. Perhitungan Titik Kurva

Untuk setiap iterasi, akan dilakukan kembali iterasi kedua guna mencari nilai k untuk setiap urutan titik pada n. Setiap iterasi kedua akan memanggil fungsi binomial untuk mendapat koefisien yang sesuai dengan nilai k yang dihitung. Formula pencarian titik x dan y sesuai dengan formula yang sudah ditulis diatas yakni:

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

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

Dengan koefisien yang berbeda beda sesuai dengan nilai n yang digunakan.

#### 4. Penambahan pada array

Setelah dihitung  $B(t)$  hasil iterasi kedua, akan disimpan dalam sebuah array dan akan dilanjutkan lagi oleh iterasi pertama. Setelah semua iterasi selesai, maka akan mengembalikan array hasil titik Bezier yang akan digunakan.

### Algoritma Divide And Conquer



```
1 def create_bezier(control_points: List[Tuple[float, float]], iterations: int) -> Tuple[List[List[Tuple[float, float]]], List[List[Tuple[float, float]]]]:
2     if len(control_points) < 2:
3         raise ValueError("At least two control points are required")
4     bezier_points = [control_points[0]]
5     iteration_points = [[control_points[0],0]]
6     find_bezier_points(bezier_points, control_points, 0, iterations, iteration_points)
7     bezier_points.append(control_points[-1])
8     iteration_points.append([control_points[-1],0])
9     return bezier_points, iteration_points
```

Gambar 4.3. Source Code Fungsi Pembentukan Kurva Bezier (Divide And Conquer)

Fungsi *create\_bezier* ini untuk mencari titik kurva bezeir dengan titik kontrol sebanyak n (termasuk titik awal dan akhir). Fungsi ini diawali dengan menvalidasi jumlah dari *control\_points*, apakah lebih besar dari 2 atau tidak. Jika tidak , akan melempar error , namun jika iya akan dilanjutkan proses pencarian titik bezier. Selanjutnya akan dilakukan dengan menginisiasi suatu array kosong dan array of array kosong, lalu menambahkan titik awal ke array tersebut. Setelah inisialisasi , akan dilakukan pemanggilan fungsi *find\_bezier\_points* untuk mendapatkan titik -titik pembentuk kurva bezier. Setelah mendapatkan titik-titik pembentuk kurva bezier , akan dilanjutkan dengan penambahan titik kontrol terakhir pada array *bezier\_points* dan *iteration\_points* dan kemudian akan mengembalikan kedua array ini.



```

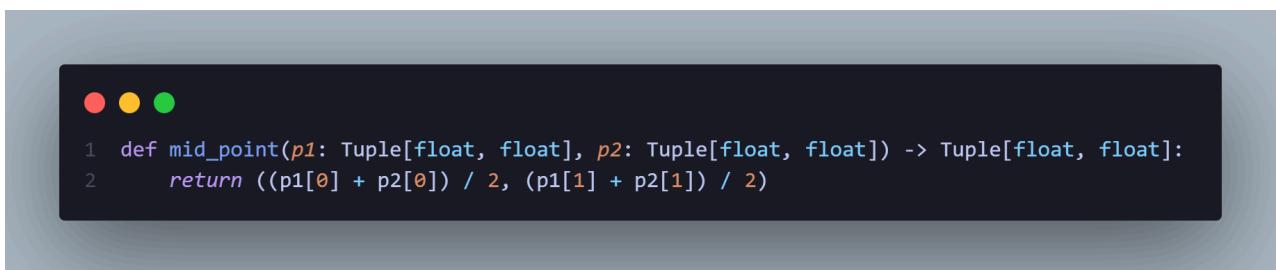
1 def find_bezier_points(bezier_points: List[Tuple[float, float]], control_points: List[Tuple[float, float]], current_iteration: int, iterations: int, iteration_points: List[List[Tuple[float, float]]]):
2     if current_iteration < iterations:
3         next_level_points = [control_points]
4         new_control_points = control_points
5         for _ in range(len(control_points) - 1):
6             temp = []
7             for i in range(len(new_control_points) - 1):
8                 temp.append(mid_point(new_control_points[i], new_control_points[i + 1]))
9             next_level_points.append(temp)
10            new_control_points = temp
11
12    left_half = [point[0] for point in next_level_points]
13    right_half = [point[-1] for point in reversed(next_level_points)]
14
15    find_bezier_points(bezier_points, left_half, current_iteration + 1, iterations, iteration_points)
16    iteration_points.append([next_level_points[-1][0], current_iteration + 1])
17    bezier_points.append(next_level_points[-1][0])
18    find_bezier_points(bezier_points, right_half, current_iteration + 1, iterations, iteration_points)

```

**Gambar 4.4. Source Code Fungsi Find Bezier Points (Divide And Conquer)**

Fungsi *find\_bezier\_points* ini untuk mencari titik Bezier untuk membentuk kurva Bezier sesuai dengan iterasi yang ditentukan. Pada awalnya dimulai dengan memeriksa apakah iterasi saat ini kurang dari jumlah iterasi yang ditentukan. Jika iya , maka akan dilakukan inisialisasi suatu array *next\_level\_points* dengan *control\_points*. Lalu untuk setiap pasangan titik kontrol berdekatan akan dilakukan perhitungan titik tengah , lalu hasil perhitungannya akan disimpan dalam suatu array temp. Temp ini akan dijadikan sebagai *new\_control\_points* untuk langkah berikutnya. Proses ini diulang sampai mendapatkan hanya tersisa satu titik tengah, yang merupakan titik pada kurva Bezier untuk iterasi tersebut.

Selanjutnya , akan dilakukan pemisahan dari titik-titik dari yang dihasilkan menjadi dua bagian, yaitu bagian kiri dan bagian kanan berdasarkan titik pertama dan titik terakhir pada setiap level array. Bagian kiri dan kanan ini kemudian digunakan sebagai input untuk pemanggilan rekursif fungsi *find\_bezier\_points* sendiri, dengan meningkatkan nilai iterasi saat ini (*current\_iteration*) dengan satu. Selama proses rekursi, fungsi juga memperbarui *iteration\_points*, yang mengumpulkan titik-titik pada kurva Bezier untuk setiap iterasi. Ini memungkinkan penggambaran progresi pembentukan kurva Bezier dari iterasi ke iterasi. *bezier\_points* diisi dengan titik-titik kurva Bezier akhir yang dihasilkan setelah mencapai jumlah iterasi yang ditentukan.



```

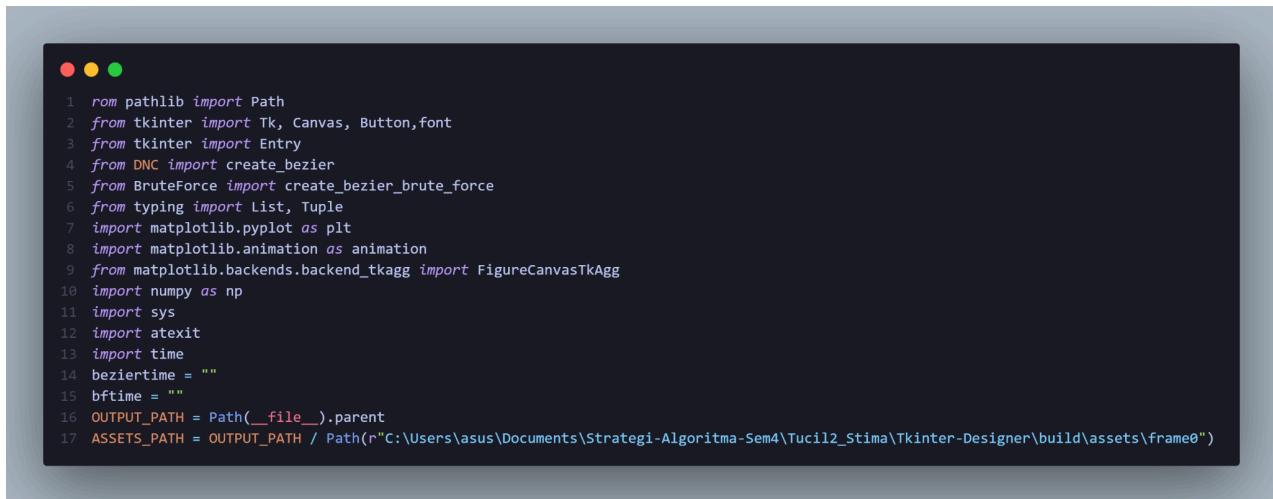
1 def mid_point(p1: Tuple[float, float], p2: Tuple[float, float]) -> Tuple[float, float]:
2     return ((p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2)

```

**Gambar 4.5. Source Code mid\_point (Divide And Conquer)**

Fungsi *mid\_point* ini digunakan untuk mencari titik tengah dari 2 titik.

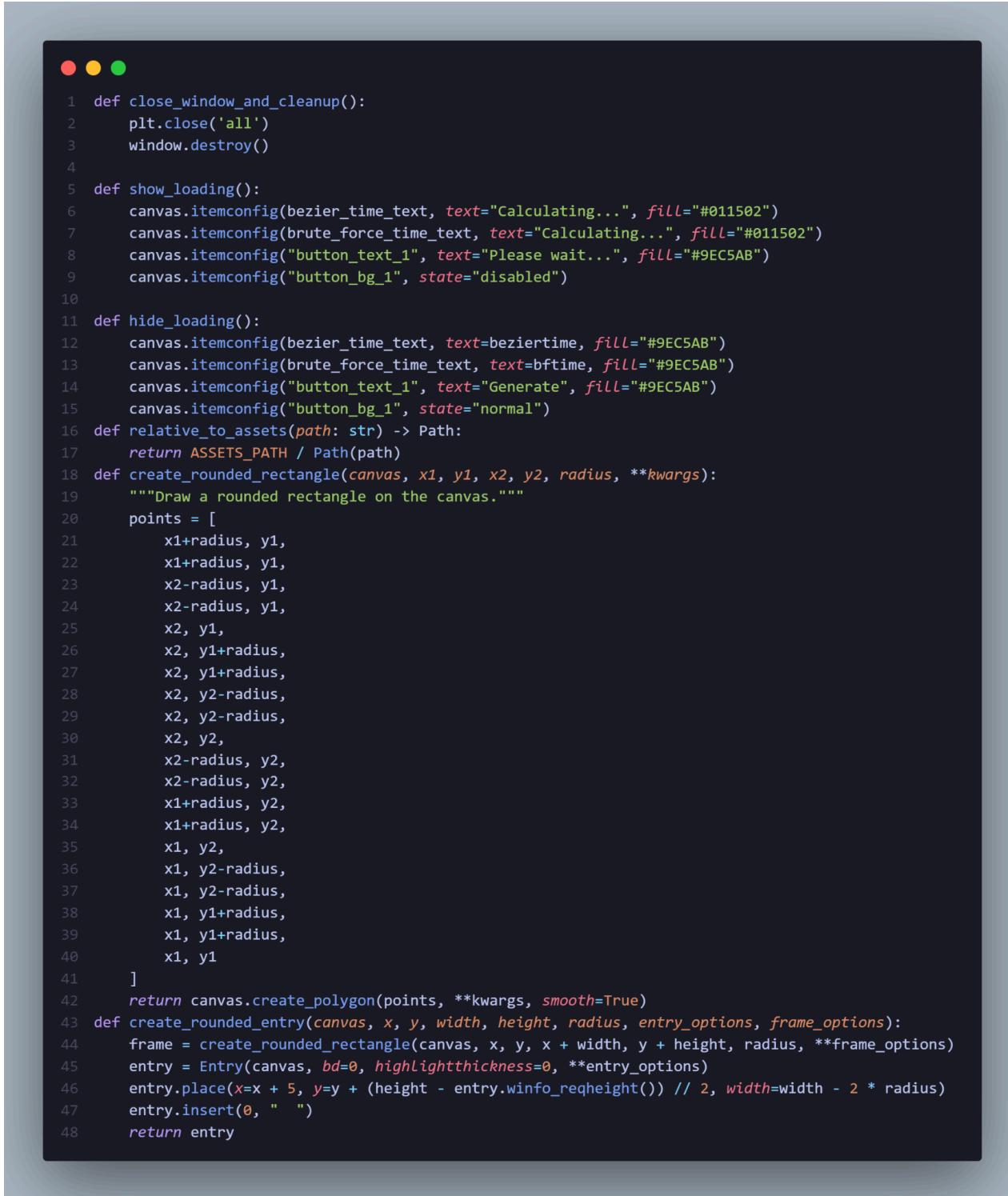
## Algoritma GUI dan Animasi



```
1  from pathlib import Path
2  from tkinter import Tk, Canvas, Button, font
3  from tkinter import Entry
4  from DNC import create_bezier
5  from BruteForce import create_bezier_brute_force
6  from typing import List, Tuple
7  import matplotlib.pyplot as plt
8  import matplotlib.animation as animation
9  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
10 import numpy as np
11 import sys
12 import atexit
13 import time
14 beziertime = ""
15 bftime = ""
16 OUTPUT_PATH = Path(__file__).parent
17 ASSETS_PATH = OUTPUT_PATH / Path(r'C:\Users\asus\Documents\Strategi-Algoritma-Sem4\Tucil2_Stima\Tkinter-Designer\build\assets\frame0")
```

**Gambar 4.6. Source Code Inisialisasi Library Interface (GUI)**

Fungsi dimulai dengan menginisialisasi berbagai library serta pengambilan fungsi dari file lain. Penggeraan tugas kecil ini menggunakan Tkinter Designer dengan format yang diperlukan yakni yaitu Output dan Asset Path.



The screenshot shows a dark-themed code editor window with a light gray background. At the top left are three circular window control buttons (red, yellow, green). The code itself is written in Python and defines several functions for managing windows and creating graphical user interface (GUI) components using the Tkinter library.

```
1 def close_window_and_cleanup():
2     plt.close('all')
3     window.destroy()
4
5 def show_loading():
6     canvas.itemconfig(bezier_time_text, text="Calculating...", fill="#011502")
7     canvas.itemconfig(brute_force_time_text, text="Calculating...", fill="#011502")
8     canvas.itemconfig("button_text_1", text="Please wait...", fill="#9EC5AB")
9     canvas.itemconfig("button_bg_1", state="disabled")
10
11 def hide_loading():
12     canvas.itemconfig(bezier_time_text, text=beziertime, fill="#9EC5AB")
13     canvas.itemconfig(brute_force_time_text, text=bftime, fill="#9EC5AB")
14     canvas.itemconfig("button_text_1", text="Generate", fill="#9EC5AB")
15     canvas.itemconfig("button_bg_1", state="normal")
16 def relative_to_assets(path: str) -> Path:
17     return ASSETS_PATH / Path(path)
18 def create_rounded_rectangle(canvas, x1, y1, x2, y2, radius, **kwargs):
19     """Draw a rounded rectangle on the canvas."""
20     points = [
21         x1+radius, y1,
22         x1+radius, y1,
23         x2-radius, y1,
24         x2-radius, y1,
25         x2, y1,
26         x2, y1+radius,
27         x2, y1+radius,
28         x2, y2-radius,
29         x2, y2-radius,
30         x2, y2,
31         x2-radius, y2,
32         x2-radius, y2,
33         x1+radius, y2,
34         x1+radius, y2,
35         x1, y2,
36         x1, y2-radius,
37         x1, y2-radius,
38         x1, y1+radius,
39         x1, y1+radius,
40         x1, y1
41     ]
42     return canvas.create_polygon(points, **kwargs, smooth=True)
43 def create_rounded_entry(canvas, x, y, width, height, radius, entry_options, frame_options):
44     frame = create_rounded_rectangle(canvas, x, y, x + width, y + height, radius, **frame_options)
45     entry = Entry(canvas, bd=0, highlightthickness=0, **entry_options)
46     entry.place(x=x + 5, y=y + (height - entry.winfo_reqheight()) // 2, width=width - 2 * radius)
47     entry.insert(0, " ")
48     return entry
```

Gambar 4.7. Source Code Fungsi Pembentukan Fitur Komponen (GUI)

Kode program diatas berfungsi sebagai styling dari komponen yang sudah ada, yakni berupa penambahan *rounded border*, penghancuran layar ketika selesai, fungsi bawaan Tkinter Designer, serta penambahan bagian input (*entry*) dan penambahan tampilan loading sebelum fungsi selesai.

```

1  def button_click():
2      global bezierTime, bfTime, fig, fig2, canvas1, canvas2, ani, ani2
3
4      if 'fig' in globals():
5          plt.close(fig)
6      if 'fig2' in globals():
7          plt.close(fig2)
8
9      show_loading()
10     canvas.update_idletasks()
11
12     try:
13         control_points_str = point_number_entry.get().strip()
14         ctrl_points = [tuple(map(int, point_str[0].strip('(').split(',')[:-1])) for point in control_points_str.split('),(')]
15         iterations = int(number_of_iterations_entry.get().strip())
16         if len(ctrl_points) < 2:
17             raise ValueError("Number of points must be at least 2")
18         if iterations < 1:
19             raise ValueError("Number of iterations must be at least 1")
20         start = time.time()
21         end = time.time()
22         bezierTime = f"Time: {bezierTime+1000:.3f} ms"
23         start = time.time()
24         bezierTime = f"Time: {bezierTime+1000:.3f} ms"
25         start = time.time()
26         bruteForceBezierPoints = create_bezier_brute_force(ctrl_points, iterations)
27         end = time.time()
28         bfTime = f"Time: {(bfTime*1000:.3f)} ms"
29         all_points = ctrl_points + bruteForceBezierPoints
30         all_x = [p[0] for p in all_points]
31         all_y = [p[1] for p in all_points]
32         x_buffer, y_buffer = 0.1, 0.1
33         fig, ax = plt.subplots()
34         iterationPoints = [[ctrl_points[i] for i in range(iterations)]]
35         for i in range(1, iterations + 1):
36             for j in range(len(points)):
37                 if(points[j][1] <= i):
38                     iterationPoints[i-1].append(points[j][0])
39         ax.set_xlim(min(all_x) - x_buffer, max(all_x) + x_buffer)
40         ax.set_ylim(min(all_y) - y_buffer, max(all_y) + y_buffer)
41         ax.plot([p[0] for p in ctrl_points], [p[1] for p in ctrl_points], 'ro-', label='Control Points', markersize=8)
42         bezier_line, = ax.plot([], [], 'g-', label='Bezier Curve')
43         bezier_points_line, = ax.plot([], [], 'bo', label='Bezier Points', markersize=4)
44         ax.legend()
45
46         def init():
47             bezier_line.set_data([], [])
48             bezier_points_line.set_data([], [])
49             return bezier_line, bezier_points_line
50
51         def animate(i):
52             if i < len(iterationPoints):
53                 x_vals, y_vals = zip(*iterationPoints[i])
54                 bezier_line.set_data(x_vals, y_vals)
55                 bezier_points_line.set_data(x_vals, y_vals)
56             return bezier_line, bezier_points_line
57
58         ani = animation.FuncAnimation(fig, animate, frames=len(iterationPoints), init_func=init, blit=True, repeat=True, interval=250)
59
60         fig2, ax2 = plt.subplots()
61         ax2.plot([p[0] for p in ctrl_points], [p[1] for p in ctrl_points], 'ro-', label='Control Points')
62         for iterations > 1:
63             ax2.plot([p[0] for p in bruteForceBezierPoints], [p[1] for p in bruteForceBezierPoints], 'g-', label='Brute Force Bezier Curve')
64         ax2.legend()
65
66         bruteForceBezier_line, = ax2.plot([], [], 'g-', label='Brute Force Bezier Curve')
67         def animateBruteForce():
68             x_vals, y_vals = zip(*bruteForceBezierPoints[:i + 1])
69             bruteForceBezier_line.set_data(x_vals, y_vals)
70             return bruteForceBezier_line,
71
72         ani2 = animation.FuncAnimation(fig2, animateBruteForce, frames=len(bruteForceBezierPoints), blit=True, interval=50)
73
74         canvas1 = FigureCanvasTkAgg(fig, master=window)
75         plot1_widget = canvas1.get_tk_widget()
76         plot1_widget.place(x=432, y=180, width=507, height=459)
77         canvas1.draw()
78         canvas2 = FigureCanvasTkAgg(fig2, master=window)
79         plot2_widget = canvas2.get_tk_widget()
80         plot2_widget.place(x=960, y=180, width=507, height=459)
81         canvas2.draw()
82         canvas.itemconfig(bezier_time_text, text=bezierTime, fill="#E6C9AB")
83         canvas.itemconfig(brute_force_time_text, text=bfTime, fill="#E6C9AB")
84
85     except Exception as e:
86         print("Error:", e)
87         bezierTime = "Input Format Error"
88         bfTime = "Input Format Error"
89         canvas.itemconfig(bezier_time_text, text="Input Format Error", fill="#FF3131")
90         canvas.itemconfig(brute_force_time_text, text="Input Format Error", fill="#FF3131")
91         canvas.update_idletasks()
92
93         if 'ani' in globals():
94             ani.event_source.stop()
95         if 'ani2' in globals():
96             ani2.event_source.stop()
97
98         if 'fig' in globals():
99             fig.clear()
100        if 'fig2' in globals():
101            canvas1.get_tk_widget().destroy()
102        if 'canvas2' in globals():
103            canvas2.get_tk_widget().destroy()
104
105    finally:
106        hide_loading()
107        canvas.update_idletasks()

```

**Gambar 4.8. Source Code Onclick / Fungsi Utama GUI**

Fungsi `button_click` dirancang untuk dijalankan ketika tombol di antarmuka pengguna berbasis Tkinter diklik. Fungsi ini memulai dengan mengambil input dari pengguna melalui entri teks untuk titik kontrol dan jumlah iterasi. Titik kontrol diubah menjadi daftar tupel yang berisi koordinat x

dan  $y$ , sementara jumlah iterasi diubah menjadi bilangan bulat. Setelah input diperoleh, fungsi ini memvalidasi input untuk memastikan bahwa jumlah titik kontrol minimal dua dan jumlah iterasi minimal satu. Jika validasi gagal, fungsi akan menampilkan pesan kesalahan yang nantinya akan di-*catch* dalam *error handling* yang telah disediakan.

Setelah validasi berhasil, fungsi ini melanjutkan dengan menghitung titik-titik pada kurva Bézier menggunakan dua metode: algoritma *Divide And Conquer* yang biasa dan algoritma *Brute Force*. Waktu yang diperlukan untuk setiap perhitungan diukur dan disimpan. Fungsi ini kemudian menggunakan Matplotlib untuk membuat plot dari titik-titik kontrol dan kurva Bézier yang dihasilkan. Plot ini ditampilkan dalam antarmuka pengguna Tkinter dengan bantuan library FigureCanvasTkAgg.

Selain itu, fungsi ini menciptakan animasi yang menunjukkan pembentukan kurva Bézier secara bertahap untuk kedua metode perhitungan, dimana tampilan tiap iterasi disediakan pada visualisasi algoritma *Divide And Conquer* dan visualisasi proses pembentukan kurva dari titik awal ke akhir pada algoritma *Brute Force*. Setelah perhitungan dan plot selesai, fungsi ini memperbarui antarmuka pengguna dengan plot baru dan waktu yang diperlukan untuk setiap metode perhitungan. Jika terjadi kesalahan selama proses, seperti format input yang salah, fungsi ini akan menampilkan pesan kesalahan di antarmuka pengguna dan menghentikan animasi yang mungkin sedang berjalan.

Fungsi ini juga mencatat waktu yang dibutuhkan bagi sebuah pendekatan pembentukan kurva Bezier berjalan. Format waktu yang dihasilkan yakni dalam milisekon dengan pembulatan sebanyak 3 angka dibelakang koma. Jika hasil yang dihasilkan berupa 0.000 ms , maka dipastikan waktu yang dibutuhkan sangatlah kecil.



```
1 window = Tk()
2 window.geometry("1500x1000")
3 window.configure(bg="#32746D")
4 canvas = Canvas(
5     window,
6     bg="#32746D",
7     height=750,
8     width=1550,
9     bd=0,
10    highlightthickness=0,
11    relief="ridge"
12 )
13 canvas.place(x=0, y=0)
14 font.families()
15 bezier_time_text = canvas.create_text(
16     431 + 253.5,
17     160,
18     anchor="center",
19     text=bezierTime,
20     fill="#9EC5AB",
21     font=("Batang Che", 24, "bold")
22 )
23
24 brute_force_time_text = canvas.create_text(
25     960 + 253.5,
26     160,
27     anchor="center",
28     text=bfTime,
29     fill="#9EC5AB",
30     font=("Batang Che", 24, "bold")
31 )
32 #Plot
33 create_rounded_rectangle(
34     canvas,
35     431.0,
36     180.0,
37     938.0,
38     639.0,
39     radius=100,
40     fill="#104F55",
41     outline=""
42 )
43 create_rounded_rectangle(
44     canvas,
45     960.0,
46     180.0,
47     1467.0,
48     639.0,
49     radius=100,
50     fill="#104F55",
51     outline=""
52 )
53 entry_options = {
54     'fg': '#011502',
55     'bg': '#9EC5AB',
56     'font': ("Batang Che", 18, "bold")
57 }
58
59 frame_options = {
60     'fill': '#9EC5AB',
61     'outline': ''
62 }
63 point_number_entry = create_rounded_entry(
64     canvas,
65     x=21.0, y=174.0, width=300, height=50, radius=40,
66     entry_options=entry_options,
67     frame_options=frame_options
68 )
69 number_of_iterations_entry = create_rounded_entry(
70     canvas,
71     x=21.0, y=334.0, width=300, height=50, radius=40,
72     entry_options=entry_options,
73     frame_options=frame_options
74 )
```

Gambar 4.9. Source Code Komponen GUI (1)

```
1  canvas.create_rectangle(
2      0,
3      0,
4      1553.5,
5      97.5,
6      fill="#011502",
7      outline=""
8  )
9
10 canvas.create_text(
11     21.0 + 25,
12     0.0 + 25,
13     anchor="nw",
14     text="BEZIER",
15     fill="#9EC5AB",
16     font=("Batang Che", 36, "bold")
17 )
18
19 canvas.create_text(
20     21.0,
21     114.0 + 15,
22     anchor="nw",
23     text="Control Points:",
24     fill="#9EC5AB",
25     font=("Batang Che", 24, "bold")
26 )
27
28 canvas.create_text(
29     530,
30     644.0 + 15,
31     anchor="nw",
32     text="Divide and Conquer",
33     fill="#9EC5AB",
34     font=("Batang Che", 24, "bold")
35 )
36
37 canvas.create_text(
38     1130,
39     644.0 + 15,
40     anchor="nw",
41     text="Brute Force",
42     fill="#9EC5AB",
43     font=("Batang Che", 24, "bold")
44 )
45
46 canvas.create_text(
47     21.0,
48     284.0,
49     anchor="nw",
50     text="Number of Iterations:",
51     fill="#9EC5AB",
52     font=("Batang Che", 24, "bold")
53 )
```

**Gambar 4.10.** Source Code Komponen GUI (2)

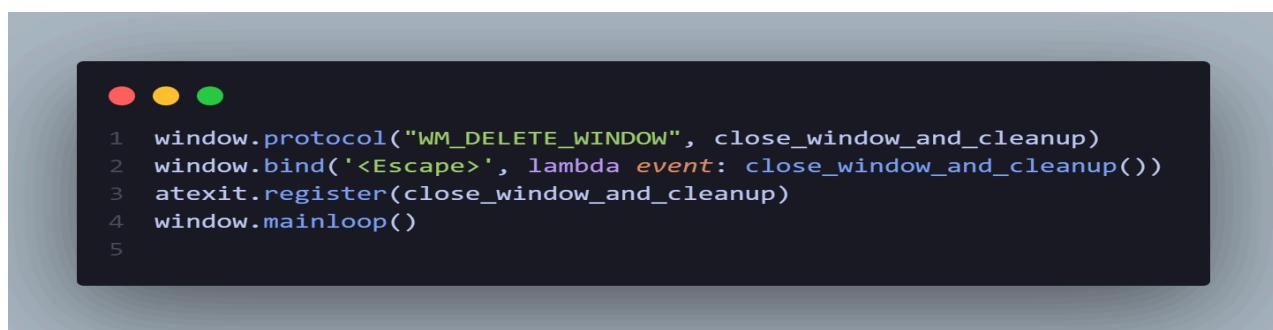
Kedua tampilan kode diatas hanya berupa pembentukan komponen seperti background, entry, serta text dengan mengandalkan fungsi pembangun diatas guna untuk memberikan visualisasi yang menarik.



```
 1 button_width = 200
 2 button_height = 70
 3 button_x = 21.0
 4 button_y = 454.0
 5 button_radius = 20 # Adjust the radius for roundness
 6 create_rounded_rectangle(
 7     canvas,
 8     button_x, button_y,
 9     button_x + button_width, button_y + button_height,
10     button_radius,
11     fill="#011502",
12     outline="",
13     tags="button_bg_1"
14 )
15 canvas.create_text(
16     button_x + button_width / 2, button_y + button_height / 2,
17     text="Generate",
18     fill="#9ECSAB",
19     font=("Batang Che", 24, "bold"),
20     tags="button_text_1"
21 )
22 canvas.tag_bind("button_bg_1", "<Enter>", lambda event: button_hover(event, "button_bg_1", "button_text_1"))
23 canvas.tag_bind("button_bg_1", "<Leave>", lambda event: button_leave(event, "button_bg_1", "button_text_1"))
24 canvas.tag_bind("button_text_1", "<Enter>", lambda event: button_hover(event, "button_bg_1", "button_text_1"))
25 canvas.tag_bind("button_text_1", "<Leave>", lambda event: button_leave(event, "button_bg_1", "button_text_1"))
26 canvas.tag_bind("button_bg_1", "<Button-1>", lambda event: button_click())
27 canvas.tag_bind("button_text_1", "<Button-1>", lambda event: button_click())
```

**Gambar 4.11.** Source Code Styling Button (GUI)

Tampilan diatas merupakan berbagai komponen pembentuk button dimana dalam pembentukan button diperhatikan fungsi apa yang dipanggil sewaktu button ditekan serta bagaimana tampilan yang menarik ketika sedang *hover*, *leave*, maupun *enter*.



```
 1 window.protocol("WM_DELETE_WINDOW", close_window_and_cleanup)
 2 window.bind('<Escape>', lambda event: close_window_and_cleanup())
 3 atexit.register(close_window_and_cleanup)
 4 window.mainloop()
 5
```

**Gambar 4.9.** Source Code End-Feature (GUI)

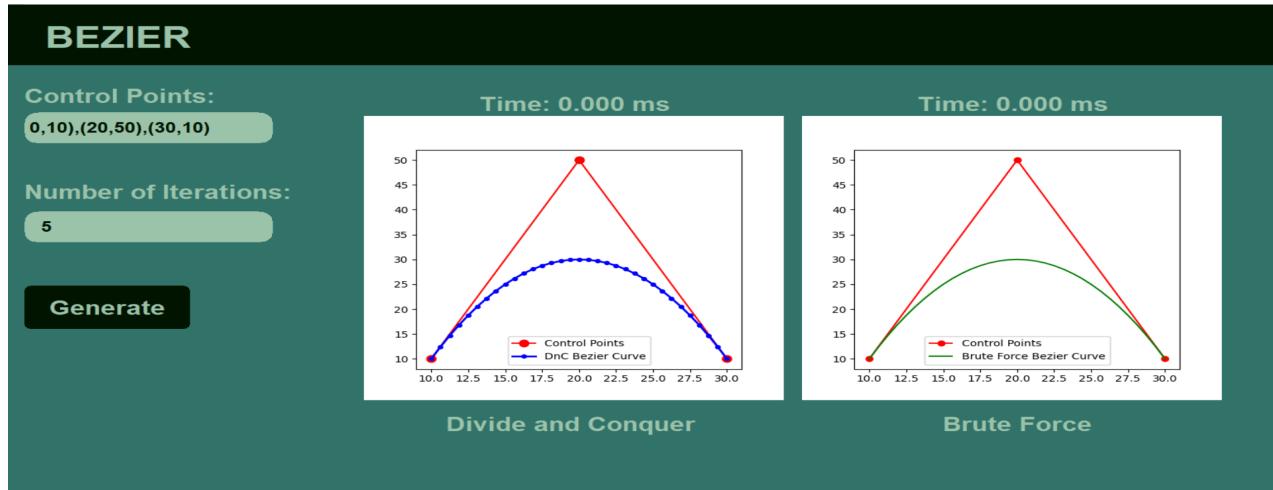
Tampilan diatas merupakan kode dalam tampilan GUI yang dibuat guna memaksimalkan pengalaman pengguna. Fitur yang disediakan berupa penghapusan Tkinter dan Matplotlib setelah menekan Esc maupun tombol X pada windows serta memastikan tampilan layar terus berjalan dengan perintah *mainloop*.

# BAB V

## TESTING PROGRAM

### Test Case 1

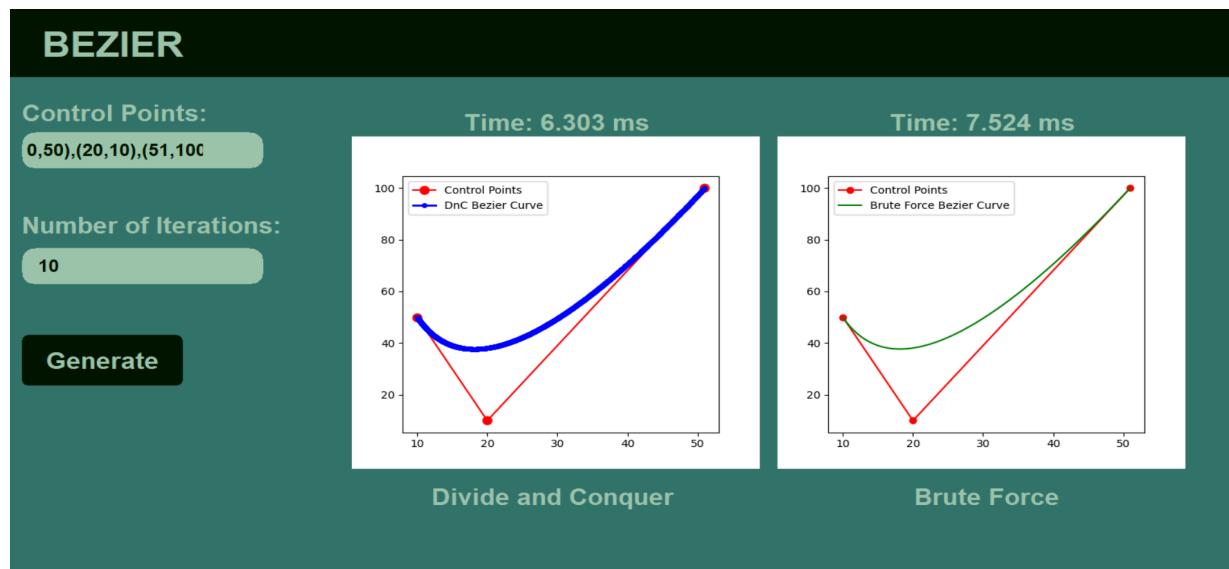
Control Points = (10,10),(20,50),(30,10)



Gambar 5.1. Test Case 1 dengan 3 titik kontrol

### Test Case 2

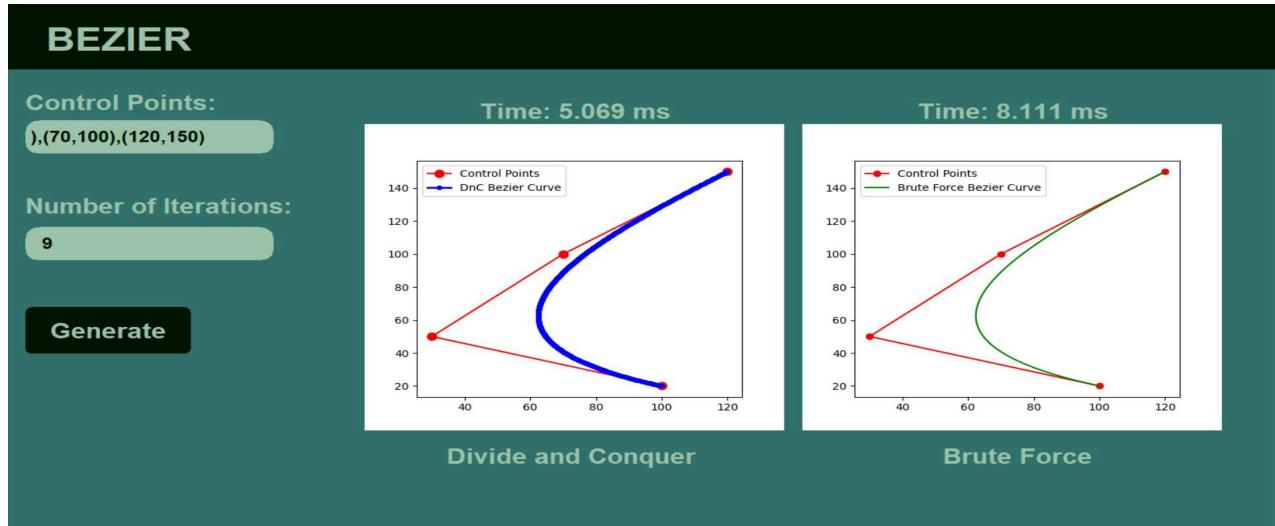
Control Points: (10,50),(20,10),(51,100)



**Gambar 5.2.** Test case 2 dengan 3 titik kontrol

### Test Case 3

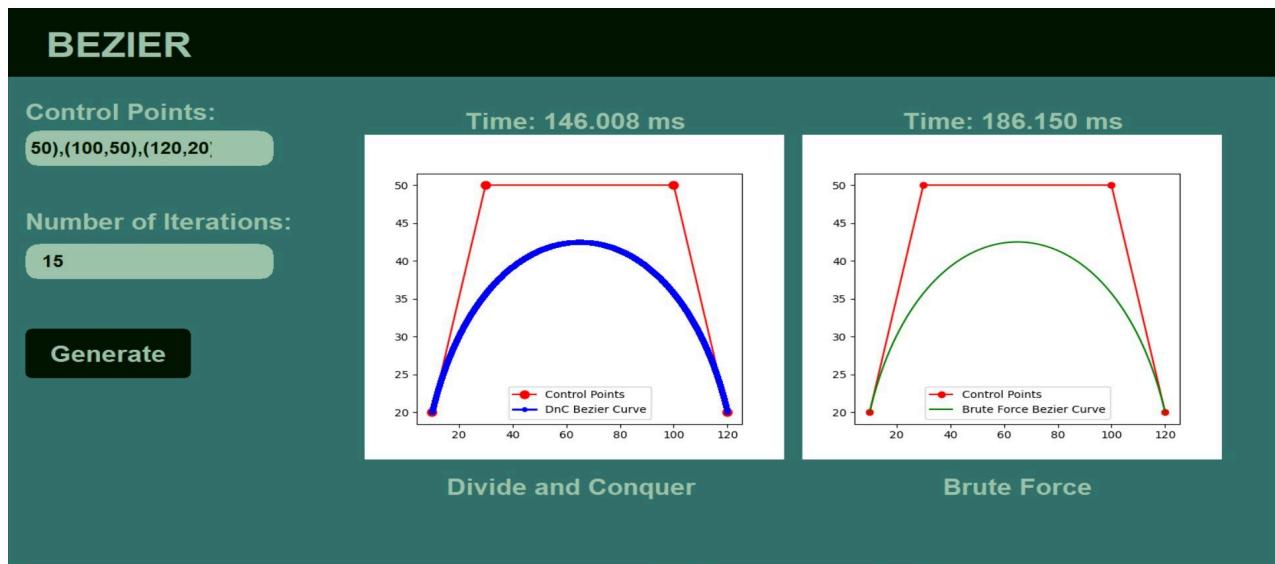
Control Points: (100,20),(30,50),(70,100),(120,150)



**Gambar 5.3.** Test Case 3 dengan 4 titik kontrol

### Test Case 4

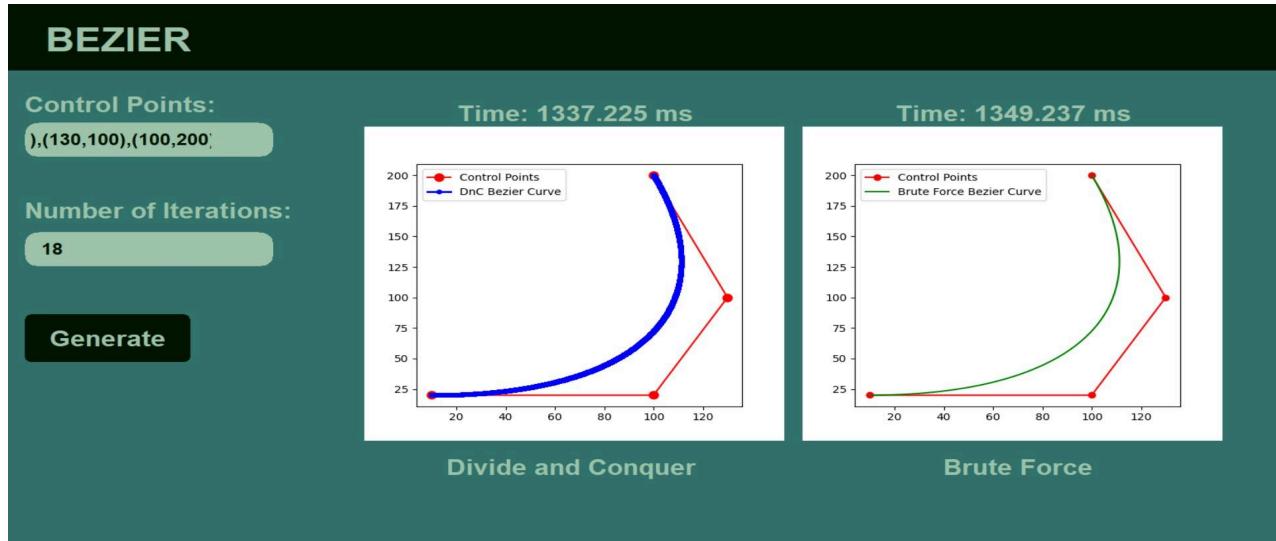
Control Points: (10,20),(30,50),(100,50),(120,20)



**Gambar 5.4.** Test Case 4 dengan 4 titik kontrol

### Test Case 5

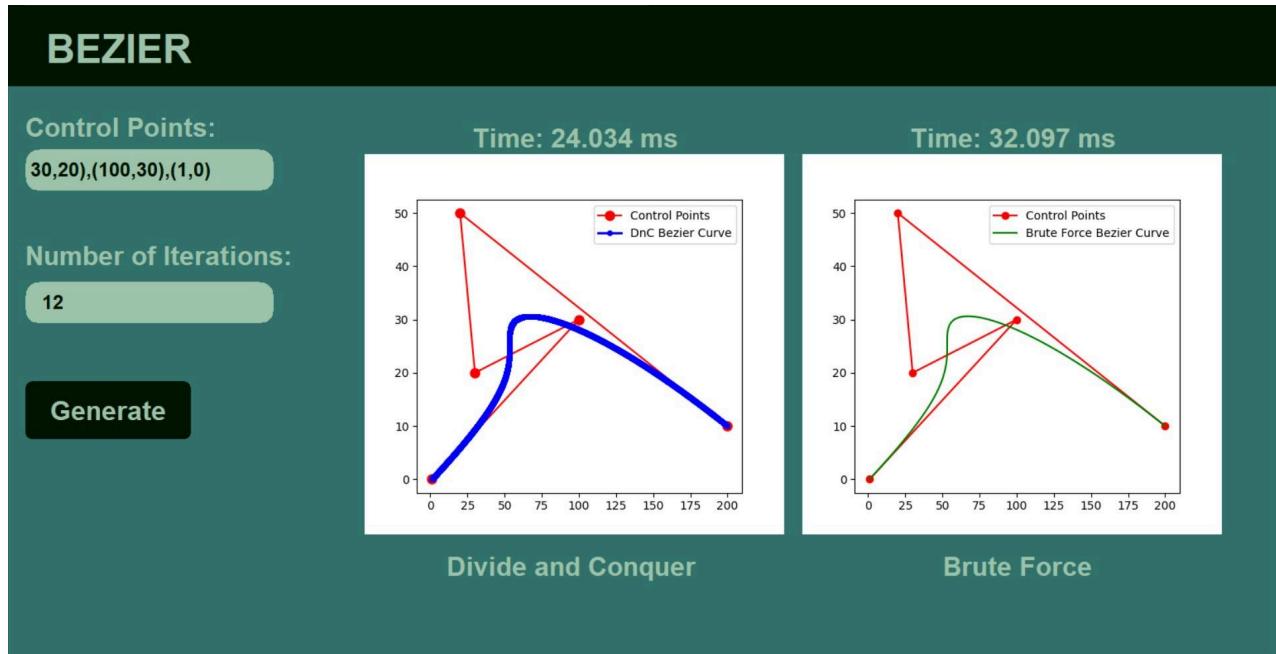
Control Points: (10,20),(100,20),(130,100),(100,200)



**Gambar 5.5.** Test Case 5 dengan 4 titik kontrol

### Test Case 6

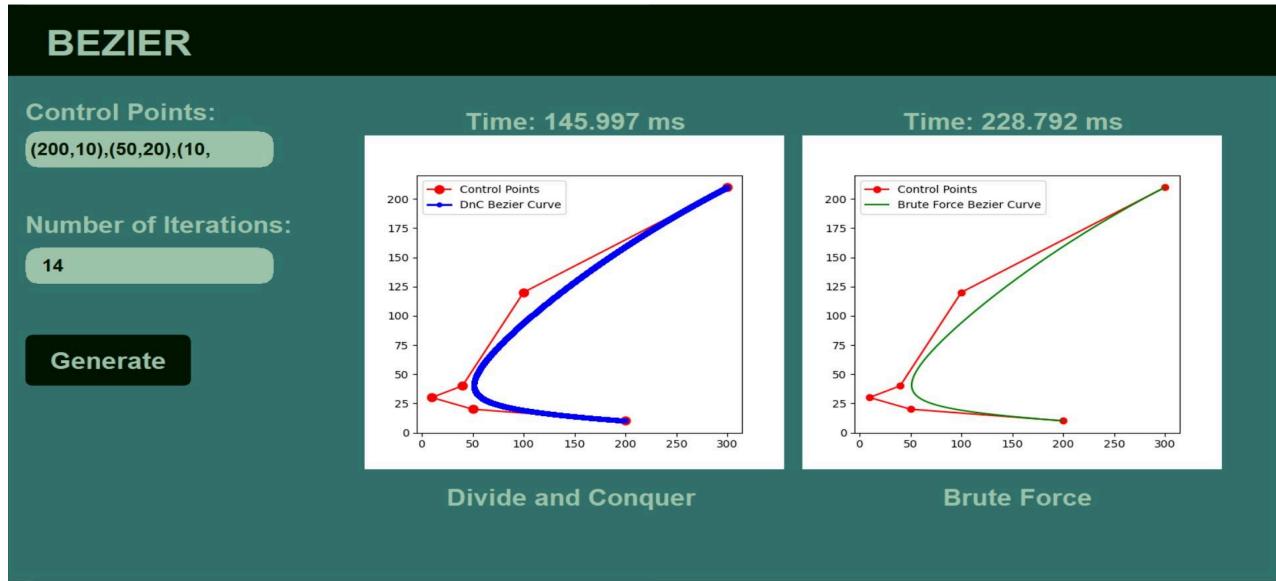
Control Points: (200,10),(20,50),(30,20),(100,30),(1,0)



**Gambar 5.6.** Test Case 6 dengan 5 titik kontrol

### Test Case 7

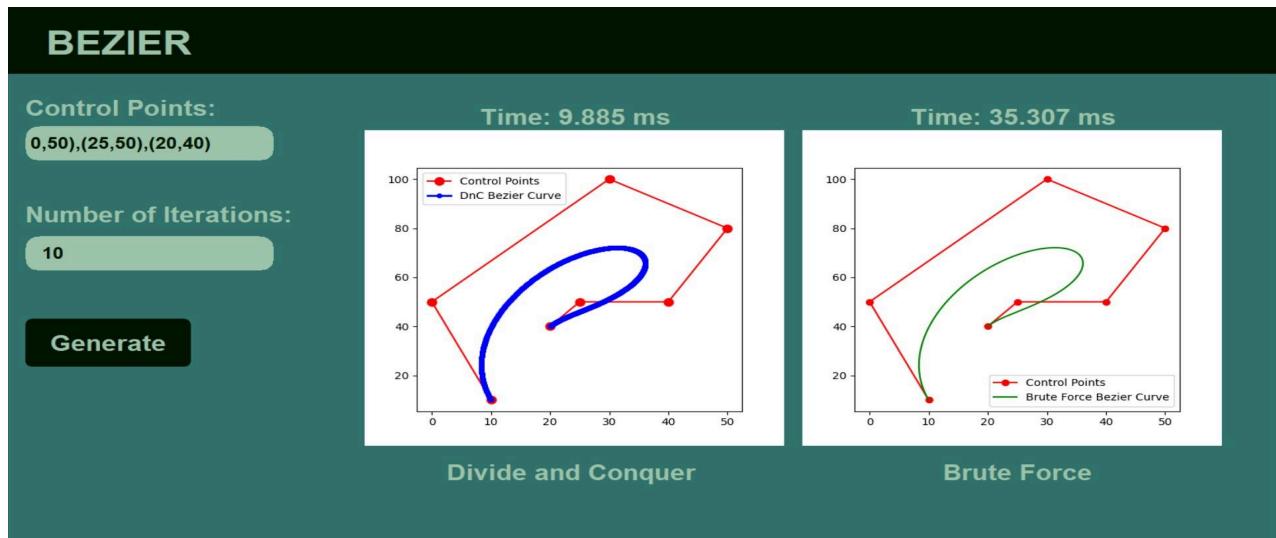
Control Points: (200,10),(50,20),(10,30),(40,40),(100,120),(300,210)



**Gambar 5.7.** Test Case 7 dengan 6 titik kontrol

### Test Case 8

Control Points: (10,10),(0,50),(30,100),(50,80),(40,50),(25,50),(20,40)



**Gambar 5.8.** Test Case 8 dengan 7 titik kontrol

Dari 8 test case diatas, terlihat bahwa seringkali algoritma *Divide And Conquer* lebih unggul dalam membentuk titik Bezier. Namun pada titik kontrol yang sedikit (Pada Test Case 1), perbedaan antara keduanya sangat minim dimana keduanya memiliki durasi kurang dari 0,001 ms dimana pada tugas ekcil ini dilakukan pembulatan sampai 3 angka dibelakang koma sehingga untuk nilai yang terlalu kecil akan dilampirkan sebagai 0 ms. Secara keseluruhan, dari test case ini, terlihat bahwa algoritma *Divide And Conquer* lebih unggul dalam membentuk titik Bezier.

## BAB VI

# ANALISIS PERBANDINGAN

### **Analisis Kompleksitas Algoritma Brute Force**

Poin utama dalam membahas perbandingan antara 2 algoritma yakni berupa komparasi kompleksitas dari program yang dibuat. Fungsi pembentukan kurva Bezier secara *Brute Force* bergantung kepada 2 variabel, yakni *control points* serta iterasi.

$$T(n, i) = n^2 \cdot (2^i + 1)$$

Dimana n merupakan banyak *control points* dan i merupakan iterasi.  $n^2$  dihasilkan dari perulangan terhadap n untuk mencari  $B(t)$  (seperti yang dijelaskan pada bab III) serta rekursi pada fungsi binomial untuk mencari koefisien terhadap  $P_k$ .  $2^i$  dihasilkan pada rekursi pertama guna untuk memenuhi tujuan dari pembentukan kurva Bezier sesuai dengan iterasi yang diinginkan pengguna dan penambahan 1 diakhir diperlukan untuk memastikan titik akhir juga terhitung dalam pengembalian nilai Bezier seperti yang dijelaskan pada bab V. Untuk test case titik kontrol percobaan konstan, seperti dalam tugas kecil ini yakni berupa n=3, dan memperhitungkan pertumbuhan cepatnya  $2^i$ , maka dalam notasi *bigO* dapat disederhanakan menjadi  $O(2^n)$ .

### **Analisis Kompleksitas Algoritma Divide and Conquer**

Algoritma divide and conquer ini juga bergantung kepada dua faktor yaitu banyaknya iterasi (i) dan banyak titik kontrol (n). Untuk mendapatkan suatu titik tengah akhir sampai berjumlah satu pada setiap iterasi dibutuhkan sebanyak  $\frac{n(n-1)}{2}$  langkah dengan n adalah jumlah titik kontrol. Setelah mendapatkan titik tengah akhir pada iterasi tersebut, akan dilakukan pembagian menjadi dua bagian terhadap titik - titik tengah yang dihasilkan (ini akan membutuhkan sebanyak  $2n$  langkah), lalu akan dilakukan pemanggilan rekursif untuk kedua upa persoalan tersebut. Proses ini membutuhkan sebanyak  $2^i$  langkah dengan  $i$  adalah jumlah iterasi. Dengan analisa di atas, maka dapat dihitung T(n) adalah sebagai berikut.

$$T(i, n) = (\frac{n(n-1)}{2} + 2n) \cdot 2^i$$

$$\begin{aligned}
 &= (\frac{1}{2}n^2 - \frac{1}{2}n + 2n) \cdot 2^i \\
 &= (\frac{1}{2}n^2 + \frac{3}{2}n) \cdot 2^i
 \end{aligned}$$

Dimana n adalah titik kontrol dan i adalah jumlah iterasi. Kompleksitas algoritma ini sangat bergantung kepada jumlah i dan n. Jika jumlah iterasi  $i$  sangat besar , maka kompleksitas algoritma akan bertambah secara eksponensial. Dari hasil  $T(n)$  di atas, jika n titik sebanyak 3 , maka kompleksitasnya dapat menjadi  $O(2^n)$  , karena jika n = 3 , tidak dibutuhkan perulangan untuk mencari titik tengah dengan langkah  $(\frac{1}{2}n^2 + \frac{3}{2}n)$  .

### **Analisis Perbandingan Kompleksitas Algoritma antara Algoritma Divide and Conquer dengan Algoritma Bruteforce**

Jika kita melakukan perbandingan pada algoritma *divide and conquer* dengan algoritma *Bruteforce*, dapat dilihat terdapat perbedaan banyak langkah antara kedua algoritma ini. Berikut kita uji pada kasus n = 3 dan i = 3.

Untuk algoritma bruteforce,

$$T(n = 3, i = 3) = (2^3 + 1) \cdot 3^2 = 81 \text{ langkah}$$

Untuk algoritma divide and conquer,

$$T(n = 3, i = 3) = (\frac{1}{2}3^2 + \frac{3}{2}3) \cdot 2^3 = 48 \text{ langkah}$$

Dari hasil perhitungan  $T(n,i)$  pada kedua algoritma , dapat dilihat bahwa secara teori bahwa langkah yang dibutuhkan algoritma *brute force* jauh lebih banyak daripada langkah pada algoritma divide and conquer. Hasil perhitungan di atas sejalan dengan hasil uji coba test case pada Bab 5. Dari perhitungan dapat dilihat perbedaannya cukup besar mencapai 1/2 kali lipat, namun dalam pengujian test case di bab 5 ,dapat dilihat bahwa pada beberapa test case perbedaan waktu antara algoritma *brute force* dan algoritma divide and conquer tidak terlalu besar. Namun secara keseluruhan dapat dilihat dalam uji coba test case pada Bab 5 bahwa cenderung lebih cepat

algoritma divide and conquer. Perbedaannya mulai terlihat lebih jauh apabila jumlah iterasi ( i ) dan jumlah titik kontrol ( n ) semakin besar.

Namun, dari hasil pengujian test case , terkadang terdapat beberapa kondisi dimana waktu eksekusi dari algoritma *brute force* lebih cepat dibandingkan dengan algoritma *divide and conquer*. Hal ini mungkin karena terkadang untuk penerapan algoritma *divide and conquer* memerlukan akses memori yang lebih kompleks , sehingga dalam praktiknya terkadang lebih lambat dari algoritma *brute force* atau bisa juga karena terkadang untuk n dan i tertentu , dimana membuat algoritma *brute force* lebih efisien.

Dari hasil analisis perbandingan di atas , didapatkan bahwa untuk membentuk suatu kurva Bezier dengan jumlah titik n kontrol dan jumlah iterasi yang rendah , algoritma *brute force* dan algoritma divide and conquer dapat diterapkan karena kedua algoritma tersebut tidak begitu berbeda dalam segi waktu eksekusi pada n dan i kecil, tetapi tetap lebih cepat algoritma divide and conquer. Namun jika untuk mencari titik kurva bezier dengan n dan i yang besar , maka lebih fleksibel menggunakan algoritma divide and conquer , karena kompleksitas algoritma yang lebih baik daripada algoritma *brute force*.

Namun masih perlu diingat bahwa kemampuan perangkat dalam mengakses memori serta pembuatan kode *Divide And Conquer* dalam tugas kecil ini memiliki fitur yang lebih kompleks sehingga tidak heran jika dalam skala kecil, atau titik kontrol serta iterasi yang lebih kecil algoritma *Brute Force* dapat dipertimbangkan. Jika ingin menggeneralisasi, lebih cocok jika algoritma *Divide And Conquer* dianggap algoritma yang lebih efektif dalam pembuatan kurva Bezier.

Dengan demikian, dapat disimpulkan bahwa diantara kedua algoritma, algoritma divide and conquer lebih baik daripada algoritma *brute force*, terutama jika skala algoritma mulai terbilang besar. Hal tersebut karena kompleksitas dan fleksibilitas dari algoritma divide and conquer lebih baik , baik untuk jumlah titik kontrol (n) dan iterasi (i) yang kecil terutama yang lebih besar.

## **BAB VII**

### **IMPLEMENTASI BONUS**

**Bonus generalisasi algoritma sehingga program dapat membentuk kurva Bezier kubik, kuartik dan selanjutnya dengan 4,5,6, hingga n titik kontrol.**

Pada program ini kami menambahkan fitur bonus untuk menggeneralisasi kurva bezier dengan titik kontrol. Konsep algoritma dari generalisasi dari kurva bezier dengan N titik kontrol sebenarnya sama seperti konsep algoritma divide and conquer pada kurva bezier dengan 1 titik kontrol. Perbedaannya hanya terdapat pada jumlah pengulangan untuk mencari titik tengah sampai mendapatkan satu titik tengah pada setiap iterasi. Semakin banyak titik kontrol , maka titik tengah dari pasangan titik bersebelahan akan semakin banyak. Sehingga pengulangan untuk mencari titik tengah akhir pada setiap iterasi akan semakin banyak dengan jumlah  $N - 1$ , dengan  $N$  merupakan jumlah titik kontrol. Setelah memperoleh list titik tengah , kemudian akan dilakukan divide menjadi dua bagian upa-persoalan kiri dan upa-persoalan kanan. Setelah itu , akan dilakukan pemanggilan rekursif dengan menjadikan upa-persoalan kiri dan upa-persoalan kanan menjadi titik kontrol yang baru pada iterasi selanjutnya.

Setelah sampai pada iterasi terakhir, maka titik tengah akhir hasil iterasi pertama sampai terakhir akan dimasukkan kedalam sebuah list yang akan dikembalikan oleh fungsi. Penemuan cara ini bisa didapatkan melalui pola yang terlihat saat pembuatan algoritma divide and conquer untuk 1 titik kontrol , dimana proses pencarian titik tengah akhir mengikuti pola  $1n-1(n)$ . Dengan menerapkan pola tersebut, dibuatlah sebuah fungsi gabungan yang dapat menerima masukan N titik kontrol , dan mengembalikan titik tengah akhir sesuai dengan titik kontrol dan jumlah iterasi yang dipilih.

#### **Bonus Visualisasi Proses Pembentukan Kurva**

Pada program ini , kami menerapkan visualisasi kurva dengan menggunakan tkinter dan matplotlib sebagai library GUI standar python untuk membuat aplikasi desktop ini. Dalam GUI aplikasi desktop kami, diawali dengan meminta input titik kontrol dan jumlah iterasi yang dipilih. User dapat mengisi titik kontrol dan jumlah iterasi pada label input yang telah disediakan oleh aplikasi desktop. Setelah mengisi titik kontrol dan jumlah iterasi, pengguna dapat mengklik tombol

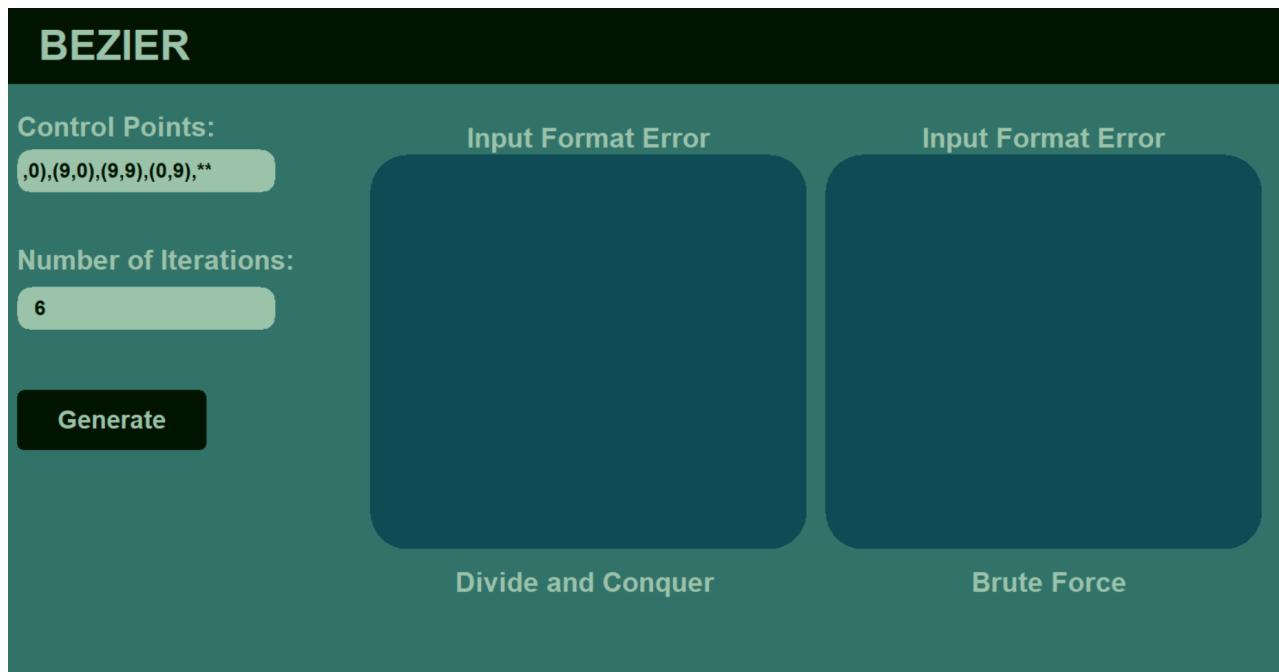
GENERATE untuk menampilkan kurva hasil dari masukan titik kontrol dan jumlah iterasi yang dipilih. Proses animasi penampilan kurva bezier secara *Divide and Conquer* akan dimulai dengan menunjukan kurva bezier dengan iterasi 1 sampai dengan iterasi yang dipilih. Proses animasi ini akan berulang terus sampai user memasukan data titik kontrol dan jumlah iterasi yang baru , lalu mengklik tombol GENERATE. Kurva Bezier yang dibentuk akan digambarkan dalam suatu bidang koordinat , lalu titik kurva bezier akan ditandai dengan dengan bulatan tebal berwarna biru , dan kurva bezier akan ditandai dengan garis berwarna biru. Sedangkan untuk titik kontrol akan ditandai dengan bulatan tebal berwarna merah.

Dalam aplikasi program, diberikan tampilan perbandingan dengan pengerajan menggunakan algoritma *Brute Force* dimana algoritma *Brute Force* hanya menampilkan pembentukan kurva dari titik awal sampai dengan titik akhir.

Perlu diperhatikan bahwa terdapat format inputan yang telah disesuaikan dan harus diikuti untuk membentuk visualisasi kurva Bezier. Penginputan titik kontrol dan iterasi harus memenuhi format berikut:

- Input titik kontrol dalam format  $(x_1,y_1),(x_2,y_2),(x_3,y_3),\dots,(x_n,y_n)$  tanpa spasi baik didepan, ditengah, maupun dibelakang inputan.
- Panjang dari control points harus minimal berjumlah 3
- Jumlah iterasi yang dimasukkan harus lebih dari 0 serta tidak memiliki elemen spasi pada ruang inputan tersebut
- Inputan harus selalu berupa integer.

Jika format diatas tidak dipenuhi, maka tampilan kurva tidak akan muncul, melainkan akan memberikan pesan error “Input Format Error”.



**Gambar 8.1** Tampilan input error

Batasan lain pada animasi pembentukan kurva Bezier dengan algoritma *brute force*, yaitu jika iterasi lebih dari 8, proses animasi pembentukan grafik untuk algoritma *brute force* tidak akan ditampilkan , dan akan langsung menampilkan hasil akhir grafik. Hal ini disebabkan karena adanya waktu iterasi yang terlalu lama sehingga tergolong tidak efisien, terutama dalam pembahasan tugas kecil ini dimana algoritma *brute force* hanya berfungsi sebagai pembanding efisiensi. Selain itu, terdapat pula batasan kendala tampilan ketika jumlah iterasi yang dimasukkan terlalu besar sehingga tampilan layar tidak akan keluar atau tampilan sebelumnya akan terhenti sebelum tampilan baru keluar yang diakibatkan oleh iterasi yang terlalu besar sehingga program membutuhkan waktu yang lebih lama untuk mengembalikan titik Bezier.

## **BAB VIII**

### **KESIMPULAN DAN SARAN**

#### **Kesimpulan**

Dalam proses pembuatan kurva Bézier dengan menggunakan n titik kontrol, algoritma Divide and Conquer terbukti menjadi pilihan yang lebih efektif dan efisien. Berdasarkan analisis dan implementasi yang telah dilakukan, algoritma Divide and Conquer secara konsisten menunjukkan performa yang lebih baik dibandingkan dengan pendekatan bruteforce, baik untuk kasus dengan jumlah titik kontrol yang banyak maupun sedikit. Meskipun demikian, perlu diakui bahwa ketika jumlah titik kontrol dan iterasi tidak besar, perbedaan kecepatan antara kedua algoritma menjadi tidak terlalu mencolok. Ini mengindikasikan bahwa untuk kasus dengan skala lebih kecil, penggunaan algoritma brute force masih dapat dipertimbangkan. Pengerjaan tugas kecil ini juga memberi wawasan mengenai efektivitas suatu program dimana faktor-faktor luar mulai dari penambahan dalam *array*, sampai dengan faktor pribadi seperti perbedaan perangkat juga mempengaruhi waktu efektif suatu program. Namun, secara umum dan dari perspektif yang lebih luas, algoritma Divide and Conquer lebih unggul dan lebih direkomendasikan dalam menyelesaikan persoalan yang lebih luas.

#### **Saran**

Dalam pembuatan tugas kecil ini, sebaiknya ditanyakan spesifikasi secara jelas dan lengkap agar pada saat pembuatan tugas kecil dapat segera dilakukan tanpa adanya hambatan akibat keambiguan tugas kecil yang sedang dikerjakan. Perlu juga difokuskan kepada pengertian masalah terlebih dahulu sebelum langsung melangkah ke pembuatan program agar pengerjaan tugas kecil dapat dilakukan secara lebih efektif.

## **BAB IX**

### **LAMPIRAN**

#### **Link Repository**

[https://github.com/Benardo07/Tucil2\\_13522019\\_13522055](https://github.com/Benardo07/Tucil2_13522019_13522055)

#### **Program Checkpoint**

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✓	
2. Program dapat melakukan visualisasi kurva Bézier.	✓	
3. Solusi yang diberikan program optimal.	✓	
4. [Bonus] Program dapat membuat kurva untuk n titik kontrol.	✓	
5. [Bonus] Program dapat melakukan visualisasi proses pembuatan kurva	✓	