

LAPORAN TUGAS KECIL II
IF2211 STRATEGI ALGORITMA
MEMBANGUN KURVA BEZIER DENGAN ALGORITMA
TITIK TENGAH BERBASIS DIVIDE AND CONQUER



Disusun oleh:

Thea Josephine Halim 13522012

Debrina Veisha Rashika W 13522025

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024

DAFTAR ISI

DAFTAR ISI.....	1
DAFTAR GAMBAR.....	2
DAFTAR TABEL.....	3
A. Deskripsi Persoalan.....	4
B. Algoritma.....	5
1. Brute Force.....	5
2. Divide and Conquer.....	6
C. Source Code.....	8
1. Fungsi lain.....	8
2. Brute Force.....	10
3. Divide and Conquer.....	11
D. Contoh Output.....	15
1. Brute Force.....	15
2. Divide and Conquer.....	19
E. Analisis.....	23
1. Analisis Algoritma Brute Force.....	23
2. Analisis Algoritma Divide and Conquer.....	24
3. Kurva Bezier yang Optimal.....	25
4. Kemulusan Sebuah Kurva.....	26
5. Kompleksitas dan Waktu.....	27
F. Checklist.....	29
G. Link Akses.....	29

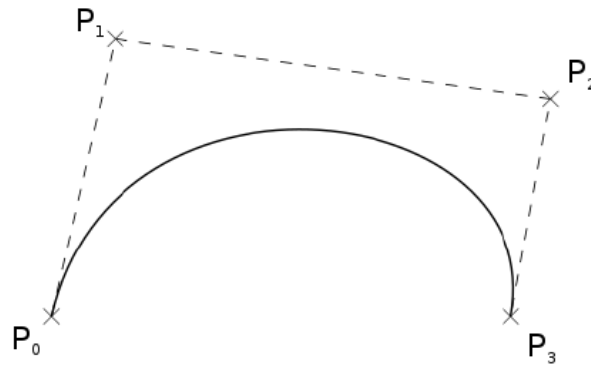
DAFTAR GAMBAR

Gambar 1.1 Kurva Bezier	4
Gambar 2.2.1 Bezier Kuadratik Iterasi Pertama	7
Gambar 2.2.2 Pemetaan Titik Bezier Kuadratik	8
Gambar 5.1. Grafik Perbandingan Waktu	28

DAFTAR TABEL

Tabel 3.1.1 Fungsi-fungsi	8
Tabel 3.2.1 Kode Brute Force	10
Tabel 3.3.1 Kode Divide and Conquer	11
Tabel 4.1 Uji Coba Brute Force	15
Tabel 4.2 Uji Coba Divide and Conquer	19
Tabel 4.3 Uji Coba Invalid Input	23
Tabel 5.1 Perbandingan Brute Force dan Divide and Conquer	27
Tabel 7.1. Checklist	29

A. Deskripsi Persoalan



Gambar 1.1 Kurva Bezier

Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Kurva Bézier dipublikasikan secara luas pada tahun 1962 oleh insinyur Pierre Bézier Perancis, yang menggunakannya untuk merancang badan mobil. 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.

Perlu ditambahkan juga bahwa kurva bezier terbentuk dari 3 titik kontrol atau lebih, sebab 2 titik kontrol hanya akan membentuk sebuah garis lurus, sehingga pada implementasi algoritma kami masukkan kasus 2 titik kontrol pada input invalid.

B. Algoritma

1. Brute Force

Brute Force pada pembentukan kurva bezier secara garis besar adalah mencari titik-titik yang dilewati kurva dengan menggunakan persamaan parametrik. Dimisalkan titik Q0 ada di garis yang terbentuk jika P0 dan P1 dihubungkan, dan titik P0 hingga Pn adalah titik kontrol masukan. Nilai t adalah seberapa jauh B(t) dari P0 ke P1, bernilai di antara 0 hingga 1 (inklusif). Nilai t didapatkan dari nilai antara 0 dan 1 dibagi bagian sama rata dengan range $(2^n)+1 + ((2^{n-1}) * (n-3))$ dengan nilai n adalah banyak iterasi (semakin banyak kurva semakin halus). Contoh dari nilai t dengan 4 iterasi dengan titik pembangun kurva berjumlah 6 adalah [0, 0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175, 0.2, 0.225, 0.25, 0.275, 0.3, 0.325, 0.35, 0.375, 0.4, 0.425, 0.45, 0.475, 0.5, 0.525, 0.55, 0.575, 0.6, 0.625, 0.65, 0.675, 0.7, 0.725, 0.75, 0.775, 0.8, 0.825, 0.85, 0.875, 0.9, 0.925, 0.95, 0.975, 1]. Titik R0 ada pada garis yang menghubungkan Q0 dan Q1, dan akan menjadi salah satu titik acuan yang dilewati kurva bezier.

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

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

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

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

Untuk kurva dengan jumlah titik kontrol yang bervariasi, kurva bezier yang terbentuk akan menggunakan persamaan parametrik yang hampir sama, tetapi dengan nilai koefisien yang berbeda. Apabila kita analisis, nilai koefisien ini mengikuti konsep pola segitiga Pascal dimulai pada baris ketiga untuk kurva bezier kuadratik (3 titik kontrol).

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1

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

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

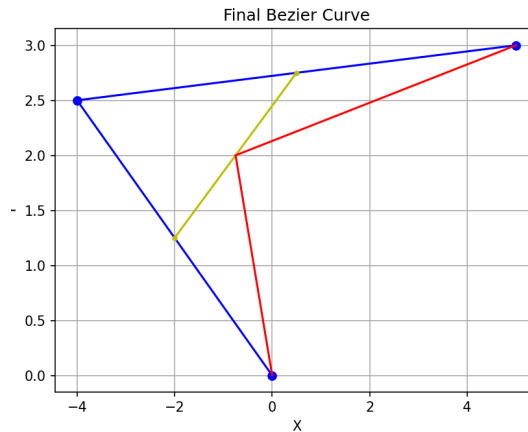
$$T_0 = B(t) = (1-t)^4P_0 + 4(1-t)^3tP_1 + 6(1-t)^2t^2P_2 + 4(1-t)t^3P_3 + t^4P_4, \quad t \in [0, 1]$$

Dengan menggunakan rumus parametrik yang sesuai untuk setiap derajat titik kontrol, dan nilai t yang bervariasi terbagi rata antara 0 dan 1, kita bisa melakukan iterasi sebanyak jumlah titik pembangun yang dikehendaki dan menemukan seluruh titik pembentuk kurva.

2. Divide and Conquer

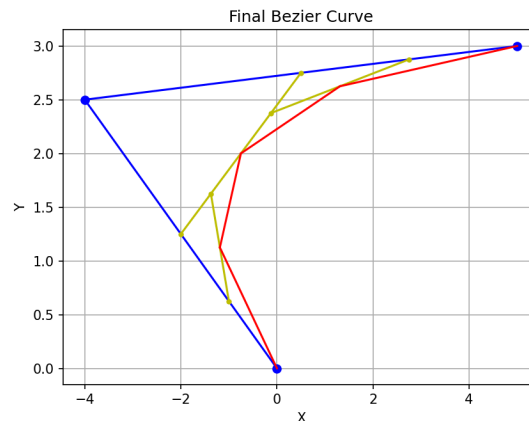
Algoritma ini pada dasarnya membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil dan menyelesaikan (solve) masing-masing upa-persoalan. Setelah itu, menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula. Algoritma ini diaplikasikan dalam membentuk kurva bezier sebagai berikut:

1. Dilakukan inisialisasi pertama untuk mencari titik-titik yang membentuk kurva pada iterasi pertama. Dimulai dari pencarian titik tengah dari garis penghubung antar titik kontrol (biru), yang kemudian akan digambarkan dengan warna kuning (Q_0 dan Q_1). Dari 2 titik kuning yang terbentuk akan saling dihubungkan dan dicari titik tengahnya (R_0), yang menjadi titik acuan dilewati kurva, berwarna merah.



Gambar 2.2.1 Bezier Kuadratik Iterasi Pertama

2. Bagi himpunan titik-titik yang membangun kurva menjadi dua bagian sama besar dan menyimpan masing-masing bagian di sebuah list (bag1 dan bag2).
3. Mencari titik tengah kembali dari masing-masing bagian secara rekursif untuk bag1 dan bag2 dengan membaginya kembali dalam *sublist-sublist* kecil. Basis yang digunakan adalah saat panjang list ≤ 5 . Pada kondisi tersebut pencarian titik tengah dilakukan secara brute-force.
4. Proses pengambilan titik tengah tiap sublist akan dilakukan dengan skenario berikut, mengambil titik tengah S0 dari tengah titik kontrol pertama P0 dan titik tengah garis bantu Q0, S1 di tengah Q0 dan R0, S2 di tengah R0 dan Q1, serta S3 di tengah Q1 dan P2. Cari titik tengah T0 dari garis penghubung S0 dan S1, serta T1 dari garis penghubung S2 dan S3.



Gambar 2.2.2 Pemetaan Titik Bezier Kuadratik

5. Setelah mendapat nilai titik tengah masing-masing sublist. Nilai titik tengah tersebut digabungkan menjadi satu. Selain itu, juga dilakukan pencatatan garis bantu untuk memudahkan melakukan pencarian titik tengah pada iterasi selanjutnya.
6. Dan selanjutnya titik-titik tengah yang baru ditemukan di gabungkan dengan keseluruhan titik yang telah dimiliki.
7. Menampilkan hasil kurva berdasarkan titik-titik yang ditemukan pada tiap iterasi.

C. Source Code

1. Fungsi lain

Tabel 3.1.1 Fungsi-fungsi

Kode	Keterangan
<pre>def takeinput(): while True: try: num_of_control_points = int(input("Jumlah titik kontrol: ")) if num_of_control_points < 3: print("Untuk Bezier Curve titik harus lebih dari 2\n") else: control_points = [] i=0 while i < num_of_control_points: x = float(input("Nilai x ke-" + str(i) + ": "))</pre>	<p>Fungsi untuk mengambil input berupa jumlah titik kontrol, titik-titik kontrol, dan jumlah iterasi.</p>

<pre>)) y = float(input("Nilai y ke-" + str(i) + ": ")))) control_points.append((x,y)) i+=1 num_of_iteration = int(input("Masukkan jumlah iterasi: ")) while num_of_iteration < 2: print("Minimal dilakukan 2 iterasi.\n") num_of_iteration = int(input("Masukkan jumlah iterasi: ")) return num_of_control_points,control_points,num_of_iteration except ValueError: print("Masukkan harus berupa angka. Silakan masukkan input kembali.\n") </pre>	
<pre> from dncfunc import Dncfunc as dn from BF import BF as bf while True: print("Welcome to Bezier Curve Generator!") print("You can choose to draw it with the methods below:") print("1. Divide and Conquer (Fast and efficient for large sets of points)") print("2. Brute Force (Simple and straightforward)") method = input("Choose a number: ") while method not in ['1', '2']: print("Invalid Number! Please choose 1 or 2.") method = input("Choose a number: ") method = int(method) if method == 1: print("You have chosen the Divide and Conquer method.") kurva = dn() kurva.solvednc() else: print("You have chosen the Brute Force method.") kurva = bf() kurva.solvebf() repeat = input("Do you want to draw another curve? (yes/no): ").lower() if repeat != "yes": print("Thank you for using Bezier Curve Generator. Goodbye!") break </pre>	<p>Fungsi main sebagai starting program.</p>

2. Brute Force

Tabel 3.2.1 Kode Brute Force

Kode	Keterangan
<pre>def formula(self, num_of_points): list = [1] for k in range(max(num_of_points,0)): list.append(list[k]*(num_of_points-k)/(k+1)) return list</pre>	<p>Fungsi untuk menentukan koefisien pada perumusan persamaan kurva Bezier. Contoh: Pada 3 titik: 1,2,1 Pada 4 titik: 1,3,3,1 dst.</p>
<pre>def inbetween(self, t,list_of_points): const = self.formula(len(list_of_points)-1) Rox = 0 Roy = 0 for i in range (len(const),0,-1): Rox += const[len(const)-i] * pow((1-t),i-1) * pow(t,(len(const)-i)) * list_of_points[len(const)-i][0] Roy += const[len(const)-i] * pow((1-t),i-1) * pow(t,(len(const)-i)) * list_of_points[len(const)-i][1] return ((Rox,Roy))</pre>	<p>Fungsi untuk melakukan perhitungan pada persamaan kurva bezier, menghasilkan tuple koordinat yang akan dilewati kurva.</p>
<pre>def solvebf(self): num_of_control_points,control_points,num_of_iteration = fn.takeinput() start = time.perf_counter() res = [] titik = 2*num_of_iteration+1 + (2**(num_of_iteration-1))*(num_of_control_points-3) t_val = np.linspace(0, 1, titik) for i in range (len(t_val)): res.append(self.inbetween(t_val[i],control_points)) points = res end = time.perf_counter() elapsed_time = end - start # Iterate through the points then plot the curve for i in range(1, len(points)): x_values = [points[j][0] for j in range(i+1)] y_values = [points[j][1] for j in range(i+1)] plt.grid(True) plt.xlabel('X') plt.ylabel('Y') plt.title('Final Bezier Curve') plt.plot([p[0] for p in control_points], [p[1] for p</pre>	<p>Program utama brute force untuk melakukan iterasi titik-titik dengan memanfaatkan kedua fungsi sebelumnya dan melakukan plotting graph menggunakan library matplotlib</p>

```

in control_points], 'ro-', label='Control Points')
    plt.plot(x_values, y_values, marker='o',
color='blue', markersize=2)
    plt.pause(0.05)

    print(str(elapsed_time) + " ms\n")
    # Plot final curve
    x_values = [point[0] for point in points]
    y_values = [point[1] for point in points]
    plt.text(0.9, -0.1, f"Elapsed Time: {elapsed_time:.2f}
ms", fontsize=10, ha='center', transform=plt.gca().transAxes)
    plt.plot(x_values, y_values, marker='o',
color='blue', markersize=2)
    plt.show()

```

3. Divide and Conquer

Tabel 3.3.1 Kode Divide and Conquer

Kode	Keterangan
<pre> def __init__(self): self.titik = [] # to save all the curve points (red line) self.garisbantu = [] # to save the helping points (yellow line) self.titikall = [] # menyimpan titik per iterasi self.garbanall = [] # menyimpan garis bantu per iterasi self.garban = 0 # jumlah titik garis bantu </pre>	Atribut dari kelas Dnffunc()
<pre> # Find the midpoint between two points def midpoint(self, p1, p2): return[(p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2] </pre>	Fungsi untuk menghitung titik tengah yang dibentuk di antara dua titik
<pre> # Function to delete odd indexed elements in a list # List starts from index 0 def deleteganjil(self, list): hapus = [] for i in range(1, len(list), 2): hapus.append(i) for i in sorted(hapus, reverse=True): del list[i] </pre>	Fungsi untuk melakukan penghapusan elemen list berindeks ganjil. List dimulai dari indeks 0.

<pre># Function to make the first iteration on bezier curve def firstinitiation(self, control_points): for i in range(len(control_points)-1): self.garisbantu.append(self.midpoint(control_points[i],control_points[i+1])) self.garban+=1 # menyimpan semua titik-titik yang dibutuhkan untuk membangun kurva self.titik.append(control_points[0]) self.titik.append(self.garisbantu[0]) for j in range(1,len(self.garisbantu)): self.titik.append(self.midpoint(self.garisbantu[j-1],self.garisbantu[j])) self.titik.append(self.garisbantu[j]) self.titik.append(control_points[len(control_points)-1]) titik = self.titik.copy() bantu = self.garisbantu.copy() self.titikall.append(titik) self.garbanall.append(bantu)</pre>	<p>Fungsi untuk menentukan titik-titik yang dilewati kurva pertama kali. Dimulai dengan melakukan looping pengisian list garis bantu (garban) dengan titik antara yang terbentuk dari jarak titik kontrol inputan. Lalu mengisi list titik dengan kombinasi titik kontrol dan garis bantu, yang nantinya akan diproses.</p>
<pre>def plot_bezier_curve(self, titik, garisbantu, num_of_control_points, initial_control_points): # mengcopy titik dan menghapus elemen ganjil untuk # menyisakan titik kurva saja titik2 = titik self.deleteganjil(titik2) plt.grid(True) plt.plot([p[0] for p in initial_control_points], [p[1] for p in initial_control_points], 'bo-', label='Control Points') # garban awal awal = num_of_control_points-2 for i in range(awal): plt.plot([garisbantu[i][0], garisbantu[i+1][0]], [garisbantu[i][1], garisbantu[i+1][1]], 'yo-', markersize=3) # garban setelah iterasi pertama for i in range(awal+1,len(garisbantu)-1,2): plt.plot([garisbantu[i][0], garisbantu[i+1][0]], [garisbantu[i][1], garisbantu[i+1][1]], 'yo-', markersize=3) # menggambar titik kurva for i in range(len(titik2)-1): x_values = [titik2[i][0], titik2[i+1][0]] y_values = [titik2[i][1], titik2[i+1][1]] plt.plot(x_values, y_values, 'r-') plt.pause(0.05)</pre>	<p>Melakukan plotting kurva berdasarkan hasil titik akhir inputan.</p>
<pre># mencari titik-titik kurva dengan rekursif membagi ke bagian-bagian kecil def mencairititik_rekursif(self, bag, mid): if len(bag) <= 5: return self.titikforce(bag,mid) bag1= bag[:len(bag)//2+2] bag2= bag[len(bag)//2:] # Rekursi untuk mencari titik-titik tengah di setiap sub-bagian</pre>	<p>Method untuk melakukan proses pencarian midpoint dari sub list titik (membagi 2 terus menerus) yang dimiliki secara rekursif.</p>

<pre>self.mencarititik_rekursif(bag1, mid) self.mencarititik_rekursif(bag2, mid)</pre>	
<pre># mencari titik-titik kurva dengan brute force jika panjang list <= 5 def titikforce(self,bag1,mid1): x = 0 temp1 = [] temp2 = [] # mencari titik bantu for i in range(len(bag1)-1): temp1.append(self.midpoint(bag1[x],bag1[x+1])) x+=1 # mencari titik-titik kurva for i in range(len(temp1)-1): mid1.append(self.midpoint(temp1[i],temp1[i+1])) # tambah titik bantu tiap bagian ke list garis bantu existing_garisbantu = set(map(tuple, self.garisbantu)) # Menambahkan elemen-elemen baru dari temp1 dan temp2 for point in temp1: if tuple(point) not in existing_garisbantu: self.garisbantu.append(point) existing_garisbantu.add(tuple(point)) self.garban = self.garban + 1 for point in temp2: if tuple(point) not in existing_garisbantu: self.garisbantu.append(point) existing_garisbantu.add(tuple(point)) self.garban = self.garban + 1</pre>	<p>Mencari titik tengah dari sub list yang dimiliki serta menambahkan garis bantu.</p>
<pre># method untuk melakukan insert titik kurva di akhir iterasi def inserttitik(self, mid1, mid2, length, garban2): j=1 if(len(mid1)<len(mid2)): j+=1 for i in range(len(mid1)): if(mid1[i] not in self.titik): self.titik.insert(j,mid1[i]) j+=2 j=1 for i in range(len(mid2)): if(mid2[i] not in self.titik): self.titik.insert(j+length,mid2[i]) j+=2 elif(self.titik.index(mid2[i])>=j+length): j+=1 j = garban2 for i in range(1,len(self.titik)+len(self.garisbantu)-j,2): if(garban2<len(self.garisbantu)): self.titik.insert(i,self.garisbantu[garban2]) garban2+=1</pre>	<p>Method untuk menggabung hasil titik tengah dari masing-masing bagian dan memasukkan titik tersebut ke list titik utama.</p>

```
def solvednc(self):
    # getting the inputs
    num_of_control_points, control_points, num_of_iteration =
fn.takeinput()
    start = time.perf_counter()

    initial_control_points = control_points.copy() # Make a
copy of the original control points
    self.firstinitiation(control_points) # Make the first
initiation (1st iteration)
    # Make Bezier curve for each iteration
    for i in range(num_of_iteration-1):
        # DEVIDE AND CONQUER !!
        # membagi kurva menjadi dua bagian
        length = len(self.titik)//2
        bag1= self.titik[:length+2]
        bag2= self.titik[length:]
        garban2 = self.garban
        # mencatat titik tengah di tiap bagian
        mid1 = []
        mid2 = []

        self.mencarititik_rekursif(bag1, mid1)
        self.mencarititik_rekursif(bag2, mid2)
        self.deletenganjil(self.titik) # menghapus elemen
ganjil dari list -> akan di rewrite dengan titik baru
        self.inserttitik(mid1,mid2,length,garban2)

        titik = self.titik.copy()
        bantu = self.garisbantu.copy()
        self.titikall.append(titik)
        self.garbanall.append(bantu)

    end = time.perf_counter()
    elapsed_time = end - start
    print("\n" + str(elapsed_time) + " ms\n")

    # Final Bezier curver
    for i in range(len(self.garbanall)):
        plt.gca().clear()

self.plot_bezier_curve(self.titikall[i],self.garbanall[i],num_of_c
ontrol_points,initial_control_points)
        plt.pause(0.5)

        plt.text(0.9, -0.1, f"Elapsed Time: {elapsed_time:.5f}
ms", fontsize=10, ha='center', transform=plt.gca().transAxes)
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.title('Final Bezier Curve')
        plt.grid(True)
        plt.show
```

Program utama divide and conquer untuk memproses hasil titik-titik awal dari *firstinitiation*, melakukan proses iterasi dengan perhitungan sisi kiri dan kanan dan melakukan plotting graph menggunakan library matplotlib

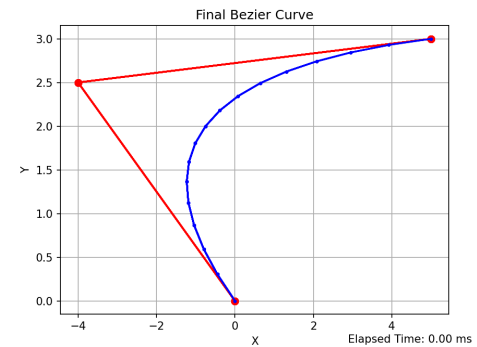
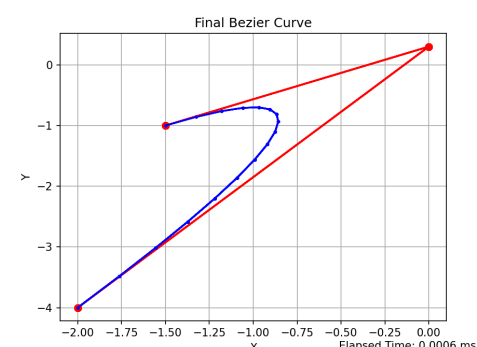
D. Contoh Output

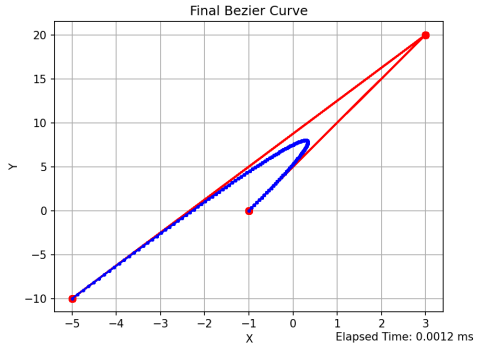
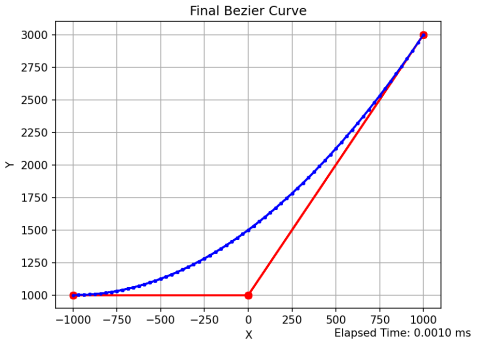
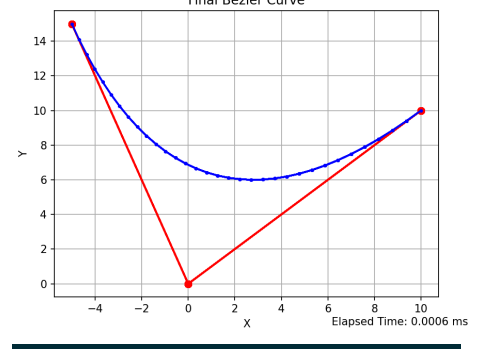
Catatan: Akan digunakan input test case yang sama (kecuali invalid input) untuk perbandingan

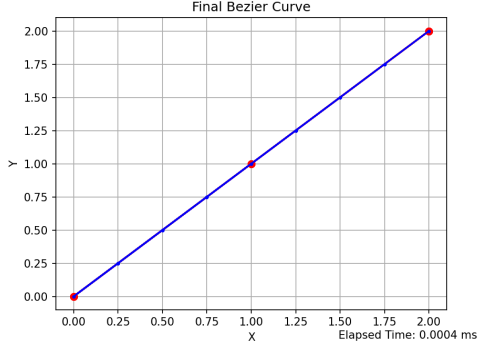
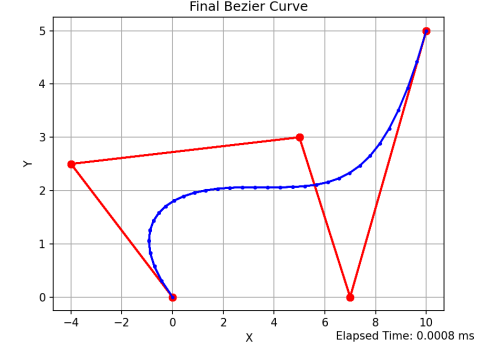
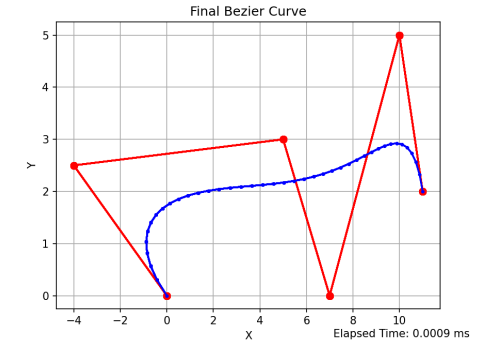
1. Brute Force

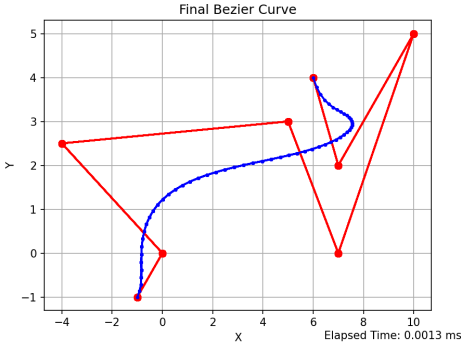
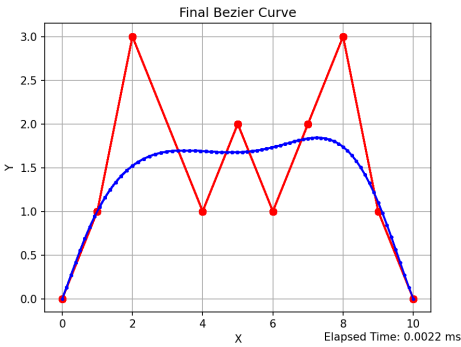
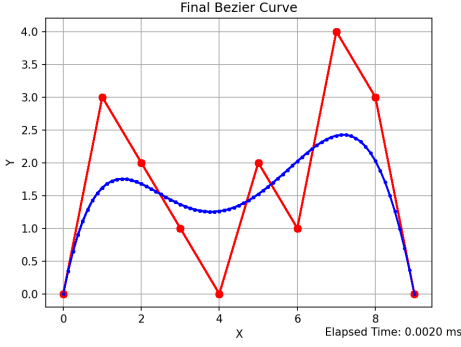
Kami melakukan uji coba dengan menggunakan titik kontrol dan banyak iterasi yang sama dengan algoritma *divide and conquer* sebagai perbandingan.

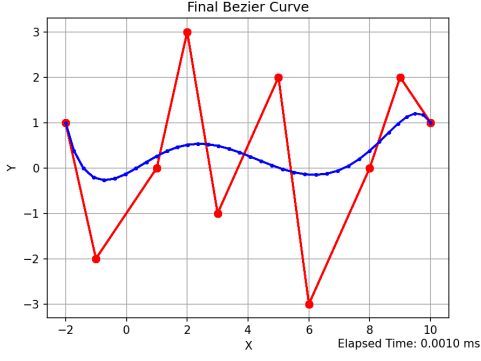
Tabel 4.1 Uji Coba Brute Force

Titik Kontrol = 3		
No	Input	Output
1.	Banyak titik kontrol = 3 [(0,0),(-4,2.5),(5,3)] Banyak iterasi = 4	 <p>0.00040929997339844704 ms</p>
2.	Banyak titik kontrol = 3 [(-2,-4),(0,0.3),(-1.5,-1)] Banyak iterasi = 4	 <p>0.0006256999913603067 ms</p>

3.	<p>Banyak titik kontrol = 3 $[(-1,0),(3,20),(-5,-10)]$ Banyak iterasi = 7</p>	 <p>0.0011890000896528363 ms</p>
4.	<p>Banyak titik kontrol = 3 $[(-1000,1000),(0,1000),(1000,3000)]$ Banyak iterasi = 6</p>	 <p>0.0009575000731274486 ms</p>
5.	<p>Banyak titik kontrol = 3 $[(10,10),(0,0),(-5,15)]$ Banyak iterasi = 5</p>	 <p>0.0005677000153809786 ms</p>

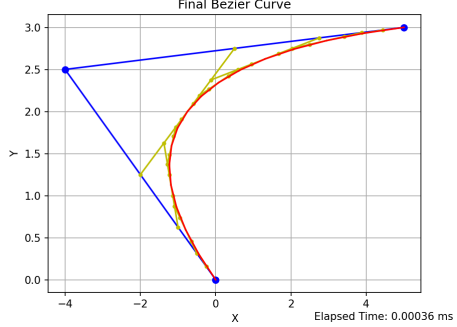
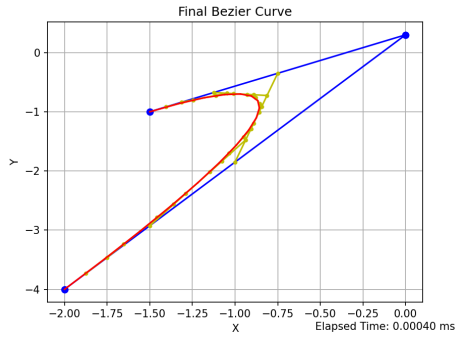
6.	<p>Banyak titik kontrol = 3 $[(0,0),(1,1),(2,2)]$ Banyak iterasi = 3</p>	 <p>0.00039469997864216566 ms</p>
Titik Kontrol >3		
7.	<p>Banyak titik kontrol = 5 $[(0,0),(-4,2.5),(5,3),$ $(7,0),(10,5)]$ Banyak iterasi = 4</p>	 <p>0.0008076999802142382 ms</p>
8.	<p>Banyak titik kontrol = 6 $[(0,0),(-4,2.5),(5,3),$ $(7,0),(10,5),(11,2)]$ Banyak iterasi = 4</p>	 <p>0.000924200052395463 ms</p>

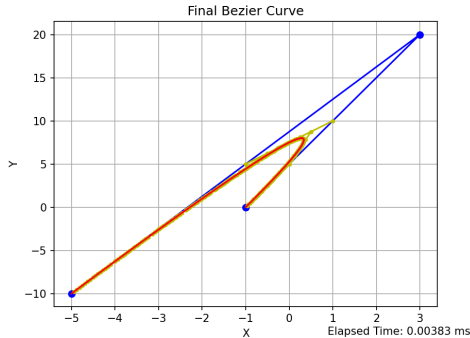
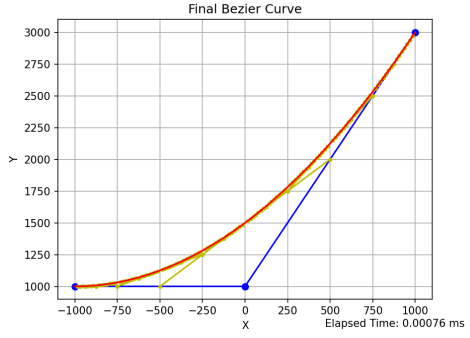
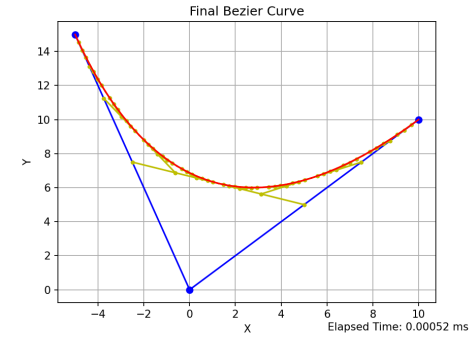
9.	<p>Banyak titik kontrol = 8 $[(-1,-1),(0,0),(-4,2.5),(5,3),$ $(7,0),(10,5),(7,2),(6,4)]$ Banyak iterasi = 4</p>	 <p>0.0012613000581040978 ms</p>
10.	<p>Banyak titik kontrol = 10 $[(0, 0), (1, 1), (2, 3), (4, 1),$ $(5, 2), (6, 1), (7, 2), (8, 3),$ $(9, 1), (10, 0)]$ Banyak iterasi = 4</p>	 <p>0.0021831999765709043 ms</p>
11	<p>Banyak titik kontrol = 10 $[(0, 0),(1, 3),(2, 2),(3,$ $1),(4,0),(5, 2),(6, 1),(7,$ $4),(8, 3),(9, 0)]$ Banyak iterasi = 4</p>	 <p>0.002046799985691905 ms</p>

12.	<p>Banyak titik kontrol = 10 $[(-2, 1), (-1, -2), (1, 0), (2, 3), (3, -1), (5, 2), (6, -3), (8, 0), (9, 2), (10, 1)]$ Banyak iterasi = 3</p>	 <p>Final Bezier Curve</p> <p>Elapsed Time: 0.0010 ms</p> <p>0.0009553999989293516 ms</p>
-----	---	---

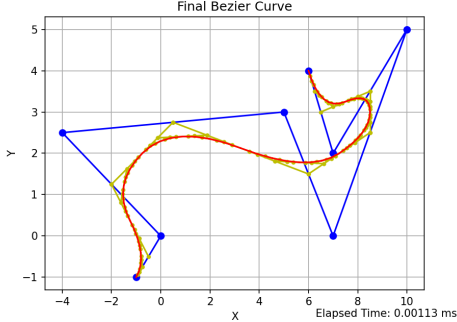
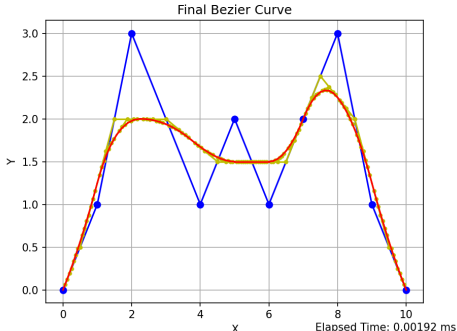
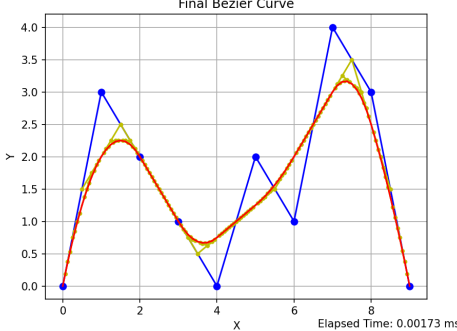
2. Divide and Conquer

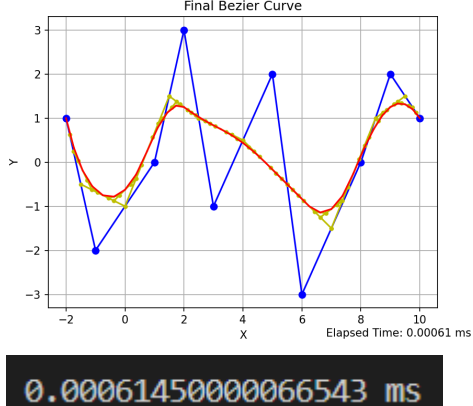
Tabel 4.2 Uji Coba Divide and Conquer

Titik Kontrol = 3		
No	Input	Output
1.	<p>Banyak titik kontrol = 3 $[(0,0),(-4,2.5),(5,3)]$ Banyak iterasi = 4</p>	 <p>Final Bezier Curve</p> <p>Elapsed Time: 0.00036 ms</p> <p>0.0003584999358281493 ms</p>
2.	<p>Banyak titik kontrol = 3 $[(-2,-4),(0,0.3),(-1.5,-1)]$ Banyak iterasi = 4</p>	 <p>Final Bezier Curve</p> <p>Elapsed Time: 0.00040 ms</p> <p>0.0003999000182375312 ms</p>

3.	<p>Banyak titik kontrol = 3 $[(-1,0),(3,20),(-5,-10)]$ Banyak iterasi = 7</p>	 <p>0.0038345999782904983 ms</p>
4.	<p>Banyak titik kontrol = 3 $[(-1000,1000),(0,1000),(1000,3000)]$ Banyak iterasi = 6</p>	 <p>0.0007552000461146235 ms</p>
5.	<p>Banyak titik kontrol = 3 $[(10,10),(0,0),(-5,15)]$ Banyak iterasi = 5</p>	 <p>0.0005164999747648835 ms</p>

6.	<p>Banyak titik kontrol = 3 $[(0,0),(1,1),(2,2)]$ Banyak iterasi = 3</p>	<p>Final Bezier Curve</p> <p>Elapsed Time: 0.00013 ms</p> <p>0.0001338999718427658 ms</p>
Titik Kontrol >3		
7.	<p>Banyak titik kontrol = 5 $[(0,0),(-4,2.5),(5,3), (7,0),(10,5)]$ Banyak iterasi = 4</p>	<p>Final Bezier Curve</p> <p>Elapsed Time: 0.00065 ms</p> <p>0.0006457000272348523 ms</p>
8.	<p>Banyak titik kontrol = 6 $[(0,0),(-4,2.5),(5,3), (7,0),(10,5),(11,2)]$ Banyak iterasi = 4</p>	<p>Final Bezier Curve</p> <p>Elapsed Time: 0.00082 ms</p> <p>0.0008186000632122159 ms</p>

9.	<p>Banyak titik kontrol = 8 $[(-1,-1),(0,0),(-4,2.5),(5,3),$ $(7,0),(10,5),(7,2),(6,4)]$ Banyak iterasi = 4</p>	 <p>Final Bezier Curve</p> <p>Elapsed Time: 0.00113 ms</p> <p>0.0011331000132486224 ms</p>
10.	<p>Banyak titik kontrol = 10 $[(0, 0), (1, 1), (2, 3), (4, 1),$ $(5, 2), (6, 1), (7, 2), (8, 3),$ $(9, 1), (10, 0)]$ Banyak iterasi = 4</p>	 <p>Final Bezier Curve</p> <p>Elapsed Time: 0.00192 ms</p> <p>0.0019221999682486057 ms</p>
11.	<p>Banyak titik kontrol = 10 $[(0, 0),(1, 3),(2, 2),(3,$ $1),(4,0),(5, 2),(6, 1),(7, 4),(8,$ $3),(9, 0)]$ Banyak iterasi = 4</p>	 <p>Final Bezier Curve</p> <p>Elapsed Time: 0.00173 ms</p> <p>0.0017256001010537148 ms</p>

12.	<p>Banyak titik kontrol = 10 $[(-2, 1), (-1, -2), (1, 0), (2, 3), (3, -1), (5, 2), (6, -3), (8, 0), (9, 2), (10, 1)]$ Banyak iterasi = 3</p>	
-----	---	--

3. Invalid Input

Tabel 4.3 Uji Coba Invalid Input

1.	Invalid input	<pre> Jumlah titik kontrol: -1 Untuk Bezier Curve titik harus lebih dari 2 Jumlah titik kontrol: 2 Untuk Bezier Curve titik harus lebih dari 2 Jumlah titik kontrol: 3 Nilai x ke-0: 2 Nilai y ke-0: 2 Nilai x ke-1: 1 Nilai y ke-1: 3 Nilai x ke-2: 1 Nilai y ke-2: 3 Terjadi duplikasi nilai, silakan masukan input titik kembali. Jumlah titik kontrol: s Masukkan harus berupa angka. Silakan masukan input kembali. Welcome to Bezier Curve Generator! You can choose to draw it with the methods below: 1. Divide and conquer (Fast and efficient for large sets of points) 2. Brute Force (Simple and straightforward) Choose a number: 3 Invalid Number! Please choose 1 or 2. </pre>
----	---------------	---

E. Analisis

1. Analisis Algoritma Brute Force

Algoritma Brute Force dimulai dengan pengambilan jumlah titik kontrol (*num_of_control_points*), koordinat titik kontrol (*control_points*), dan jumlah titik/iterasi (*num_of_iteration*). Penamaan jumlah titik sebagai *num_of_iteration* hanya untuk mempermudah perbandingan antara kedua algoritma, dan jumlah iterasi yang dilakukan memang bergantung pada jumlah titik acuan kurva yang dikehendaki. Akan disiapkan sebuah list kosong *res* untuk menampung hasil akhir titik yang didapat dari perhitungan. Dengan menggunakan fungsi numpy *linspace* akan dilakukan

pembagian dengan range 0 hingga 1 sebanyak $(2^n)+1 + ((2^{n-1}) * (n-3))$ lalu disimpan pada array *t_val*. Setelah itu akan dilakukan iterasi *for loop* sebanyak jumlah titik, dan dalam setiap iterasinya akan dilakukan penambahan elemen tuple koordinat titik kurva pada *res*. Perhitungan tuple koordinat ini menggunakan fungsi *inbetween*, yang mengambil nilai seluruh koefisien yang diberikan fungsi *formula*, lalu melakukan iterasi menghitung persamaan parametrik yang sesuai dengan derajatnya. Sisa algoritma akan menggambarkan kurva bezier yang terbentuk dari titik hasil tersebut dengan library matplotlib. Setiap penampilan titik ini akan diberikan waktu jeda 0.05 detik untuk mempermudah visualisasi. Maka dari itu, hasil perhitungan waktu algoritma nantinya akan dikurangi sebanyak waktu jeda yang dilakukan.

Berdasarkan analisis, kompleksitas algoritmanya adalah $T(n) = T(n) + T(n^2)$ untuk $n > 0$, sehingga didapat kompleksitasnya adalah $O(n^2)$.

2. Analisis Algoritma Divide and Conquer

Sama seperti brute force, algoritma divide and conquer dimulai dengan pengambilan ketiga nilai input, *num_of_points*, *control_points*, dan *num_of_iteration*. Selanjutnya, dilakukan inisialisasi pertama melalui fungsi *firstinitiation* dengan pencarian titik-titik tengah yang akan membentuk kurva pada iterasi pertama. Dengan cara mencari titik bantu dengan mencari tengah *control points* terlebih dahulu selanjutnya mencari kembali titik tengah dari titik bantu tersebut. Setelah mendapat titik dari iterasi pertama baru dilakukan *for loop* untuk mencari titik kurva tiap iterasinya. Pada tiap iterasinya list titik yang dimiliki akan dibagi menjadi sub list kecil secara rekursif. Tiap sub list ini akan dilakukan proses pencarian titik tengah yang membangun kurva melalui *method titikforce* yang mencari titik bantu serta tengah dengan melakukan *for loop* dari elemen ke 0 sampai $n-1$. Setelah menemukan titik tengah dari masing-masing bagian, titik tengah tersebut akan digabungkan menjadi satu dan ditambahkan kepada list titik yang dimiliki dengan *method inserttitik*. Selanjutnya akan ditampilkan proses

pembentukan kurva setiap iterasinya dengan waktu jeda 0.05 detik untuk mempermudah visualisasi. Maka dari itu, hasil perhitungan waktu algoritma nantinya akan dikurangi sebanyak waktu jeda yang dilakukan.

Berdasarkan analisis, kompleksitas algoritmanya adalah $T(n,m) = m * (T(n/2,m) + T(n/2,m)) + cn$ untuk $n > 5$ dan m adalah jumlah iterasi, Menurut teorema master didapat kompleksitas algoritmanya adalah $O(m * n \log(n))$.

3. Kurva Bezier yang Optimal

Untuk mempermudah penulisan analisis, selanjutnya divide and conquer disebut DNC dan brute force dengan BF. Kurva Bezier adalah kurva mulus dibentuk dari kumpulan (set) titik kontrol, dan umumnya pembentukannya menggunakan persamaan bezier polinomial yang sudah kita bahas di bagian pendahuluan, dan telah kita terapkan pada algoritma BF. Ada juga penggunaan teknik DNC seperti yang telah kita implementasikan di atas, membagi dua kurva terus menerus dan mencari titik tengah yang sesuai, sesuai dengan jumlah iterasi yang dikehendaki.

Kurva Bezier banyak digunakan dalam animasi, alat gambar grafis seperti pada Adobe Illustrator, tipografi dan lainnya, menjadikan penentuan keoptimalan sebuah kurva bezier ada pada kemulusan dan tingkat presisinya. Kemulusan ini penting untuk membuat rasa kedinamisan dari kurva yang ingin dibentuk. Tentunya, kurva bezier dengan tingkat derajat yang lebih tinggi lebih sulit untuk dipetakan sehingga akan dijadikan pengujian utama dalam perbandingan kedua algoritma ini. Pada kurva bezier yang optimal tidak akan ada tanjakan atau benjolan yang muncul tiba-tiba dengan sangat jelas. Selain itu, kompleksitas waktu yang dibutuhkan juga mempengaruhi keefisienan dari proses pembentukan kurva itu sendiri.

4. Kemulusan Sebuah Kurva

Dari hasil pengujian yang telah kita lakukan dengan berbagai macam jumlah titik kontrol, tidak terdapat banyak perbedaan signifikan dari algoritma DNC dan BF. Perhatikan pada pengujian derajat 3 pada tes uji nomor 2 dan 5, pengujian derajat terkecil yang telah kita lakukan. Memang ada sedikit perbedaan pada sudut penarikan kelengkungan, tetapi perbedaan ini sangat kecil sehingga tidak menjadi masalah, dan keduanya sama-sama terlihat mulus. Beralih ke derajat 5, 6, dan 8, kita mulai melihat perbedaan kelengkungan yang lebih drastis. Kurva yang dibentuk oleh BF memiliki derajat kelengkungan yang kecil daripada DNC. Kurva BF tidak memiliki banyak lekukan besar dan cenderung terlihat lebih lurus daripada kurva DNC yang lebih mengikuti pola gerak titik dan memiliki lekukan yang lebih luas, hal ini terlihat jelas pada tes uji 4. Lanjut dengan bezier derajat 10 pada tes uji 6 dan 7, derajat tertinggi yang telah kita analisis. Perbedaan kini jauh lebih tampak, dengan kurva BF tidak memiliki banyak lengkungan jelas dan kurva DNC memiliki lekukan tinggi yang tajam. Kira-kira kurva BF ini seperti bukit yang cenderung landai, sedangkan kurva DNC seperti pegunungan terjal.

Maka dari itu, dapat kita simpulkan bahwa kurva BF yang terbentuk memiliki lengkungan yang landai dan tidak tampak terlalu mengikuti pola pergerakan titik-titik kontrol sehingga terlihat lebih dinamis. Lain halnya dengan kurva bezier hasil algoritma DNC yang jauh lebih mengikuti pola pergerakan titik kontrol sehingga kurva yang terbentuk memiliki lengkungan-lengkungan jelas. Secara kemulusan kurva, algoritma BF menang jauh dari DNC, tetapi algoritma DNC memberikan hasil yang lebih presisi sesuai pola gerak titik, walaupun tingkat presisi ini sebenarnya juga subjektif. Akan tetapi sebenarnya kemulusan kurva BF ini disebabkan oleh kalkulasi satu per satu per titik kurva, sehingga waktu algoritma yang dibutuhkan lebih banyak. Berbeda dengan kurva DNC yang mengorbankan ketepatan titik untuk proses yang lebih cepat.

5. Kompleksitas dan Waktu

Sebelum membahas kompleksitas dan waktu algoritma, perlu kita ingat lagi bahwa pada DNC, jumlah iterasi yang semakin tinggi memberikan hasil kurva yang semakin mulus, tetapi membutuhkan lebih banyak waktu. Lain hal pada algoritma BF, jumlah iterasi di sini mewakili jumlah titik yang akan dilewati kurva, bertindak sebagai pembagi rata nilai t , sehingga jumlah titik yang banyak memberikan hasil kurva yang lebih mulus juga dan membutuhkan waktu proses lebih banyak.

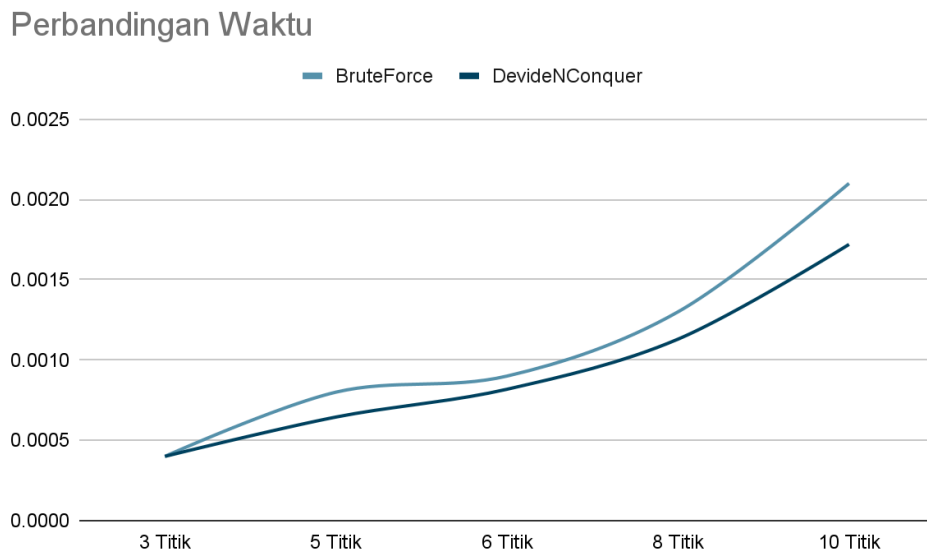
Dari hasil melakukan pengujian dengan tiga titik kontrol dengan titik yang berbeda dan jumlah iterasi yang berbeda-beda didapat hasil pengujian sebagai berikut:

Tabel 5.1 Perbandingan Brute Force dan Divide and Conquer

Banyak Iterasi	Brute Force	Divide and Conquer
3	0.0004 ms	0.00013 ms
4	0.0004 ms	0.00035 ms
4	0.0006 ms	0.00039 ms
5	0.00056 ms	0.00051 ms
6	0.0009 ms	0.00075 ms
7	0.0011 ms	0.00383 ms

Dari hasil pada tabel di atas dapat diketahui bahwa algoritma *divide and conquer* mayoritas berjalan lebih cepat dan efisien dibandingkan *brute force*. Hal ini terjadi karena pada algoritma *divide and conquer* dilakukan proses pada bagian-bagian sublist sehingga proses dapat dilakukan lebih cepat. Bisa dilihat pula kecepatan juga dipengaruhi oleh banyak iterasi yang dilakukan, semakin banyak iterasi yang dilakukan maka waktu yang dibutuhkan semakin lama. Hal ini sejalan dengan kompleksitas algoritma pada algoritma *divide and conquer* yakni $O(m \cdot n \log(n))$ dengan m adalah banyak iterasi yang dilakukan.

Pengujian juga dilakukan pada beberapa titik kontrol. Dilakukan menggunakan jumlah titik dan jumlah iterasi yang sama yaitu 4 sehingga pengujian waktu kompleksitas setara. Dari hasil pengujian, didapat grafik sebagai berikut:



Gambar 5.1. Grafik Perbandingan Waktu

Dari grafik, terlihat bahwa semakin banyak titik kontrol yang digunakan dalam metode *brute force*, waktu yang dibutuhkan juga meningkat secara signifikan. Penambahan waktu tersebut terjadi dengan cepat. Namun, saat menggunakan metode *divide and conquer*, waktu yang dibutuhkan cenderung stabil dan penambahan waktu antar penambahan titik kontrol cenderung yang kecil. Perbedaan ini sesuai dengan kompleksitas masing-masing metode. Metode *Brute Force* memiliki kompleksitas $O(n^2)$, sementara *Divide and Conquer* memiliki kompleksitas $O(n \log(n))$. Dengan demikian, dapat disimpulkan bahwa algoritma *Divide and Conquer* lebih efektif dan efisien, terutama ketika menangani kurva dengan banyak titik kontrol. Walaupun begitu, algoritma *Brute Force* dapat menghasilkan kurva yang lebih optimal dan halus sesuai dengan tujuan yang ingin dicapai sebuah kurva bezier, tetapi dengan konsekuensi waktu eksekusi yang lebih lama. Maka dari itu, algoritma terbaik bergantung pada kebutuhan kita,

apakah kita ingin mendapatkan hasil yang lebih mulus dengan *Brute Force*, atau hasil yang lebih cepat sesuai dengan algoritma *Divide and Conquer*.

F. Checklist

Tabel 7.1. Checklist

Poin	Ya	Tidak
Program berhasil dijalankan	v	
Program dapat melakukan visualisasi kurva Bezier	v	
Solusi yang diberikan program optimal	v	
[Bonus] Program dapat membuat kurva untuk n titik kontrol.	v	
[Bonus] Program dapat melakukan visualisasi proses pembuatan kurva.	v	

G. Link Akses

1. Repository:

https://github.com/pandaandsushi/Tucil2_13522012_13522025

2. References:

- [1] R. Munir, "Algoritma Divide and Conquer - Bagian 4," Institut Teknologi Bandung, 2024. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf)
- [2] R. Munir, "Aplikasi Divide and Conquer," Institut Teknologi Bandung, 2020. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Aplikasi-Divide-and-Conquer-2020.pdf>
- [3] "How to obtain the nth row of the Pascal Triangle," Stack Overflow. [Online]. Available:

<https://stackoverflow.com/questions/40067386/how-to-obtain-the-nth-row-of-the-pascal-triangle>

[4] H. Prautzsch, W. Boehm, and M. Paluszny, "Bezier Curves Explained," in Computer-Aided Geometric Design: A Totally Four-Dimensional Perspective, Springer, 2002.

[5] "Bezier Curves," [Online]. Available:

<https://www.youtube.com/watch?app=desktop&v=2HvH9cmHbG4>