Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Disusun oleh :

Abdul Rafi Radityo Hutomo - 13522089

Daftar Isi

BAB I Deskripsi Permasalahan

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

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P0 sampai Pn , 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 P0, sedangkan titik kontrol terakhir adalah P3. Titik kontrol P1 dan P2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Secara matematis kurva bezier dapat didefinisikan secara rekursif dan secara eksplisit, secara rekursif kurva bezier didefinisikan sebagai berikut,

dengan $B_{P_0P_1\ldots P_N}$ adalah kurva bezier yang dibentuk dari kumpulan titik $P_0, P_1, P_2, \ldots P_n$ berlaku,

$$\mathbf{B}_{\mathbf{P}_0}(t) = \mathbf{P}_0,$$

$$\mathbf{B}(t) = \mathbf{B}_{\mathbf{P}_0\mathbf{P}_1\ldots\mathbf{P}_n}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0\mathbf{P}_1\ldots\mathbf{P}_{n-1}}(t) + t\mathbf{B}_{\mathbf{P}_1\mathbf{P}_2\ldots\mathbf{P}_n}(t)$$

Atau secara eksplisit kurva bezier dapat didefinisikan sebagai berikut :

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \\ &= (1-t)^n \mathbf{P}_0 + \binom{n}{1}(1-t)^{n-1} t\mathbf{P}_1 + \cdots + \binom{n}{n-1}(1-t)t^{n-1}\mathbf{P}_{n-1} + t^n\mathbf{P}_n, \qquad 0 \leqslant t \leqslant 1\end{aligned}$$

BAB II Algoritma Program

Dengan kedua definisi tersebut ada dua opsi algoritma yang dapat digunakan untuk menghasilkan kurva bezier sebanyak n titik.

A. Algoritma Brute Force

Algoritma brute force menghasilkan kurva bezier dengan menggunakan rumus eksplisit kurva bezier dan menghitungnya untuk beberapa nilai t dan mendapatkan titik-titik yang ada pada kurva bezier tersebut. Kemudian, kurva bezier digambar dengan menghubungkan titik-titik tersebut secara sekuensial dengan garis. Secara rinci algoritma brute force memiliki langkah-langkah sebagai berikut :

1. Titik-titik awal kurva bezier terdefinisi dan disimpan pada sebuah list 'Points' dan kurva bezier akan diaproksimasi dengan menghitung N titik.
2. Hitung derajat kurva n = jumlah point - 1
3. Inisialisasi list baru "Bezier_Points"
4. Lakukan traversal untuk nilai t [0..1] dengan step sebesar 1/(2^N)
5. Untuk setiap t pada 4 hitung titik yang didapat pada nilai t menggunakan rumus dengan cara :
   5.1. Inisialisasi Point baru dengan x = 0 dan y = 0
   5.2. Lakukan traversal nilai [0..n]
   5.3. Untuk setiap n pada 5.2, increment nilai x Point baru sebesar :

   $$\binom{n}{i} (1 - t)^{n-i} \cdot t^{i} \cdot (Points[i]).x$$

   dan nilai y Point baru sebesar :

   $$\binom{n}{i} (1 - t)^{n-i} \cdot t^{i} \cdot (Points[i]).y$$

   5.4. Append Point baru ke dalam list Bezier_Points
6. Hubungkan semua point pada list Bezier_Points secara sekuensial

B. Algoritma Divide and Conquer

Algoritma brute force menghasilkan kurva bezier dengan menggunakan pendekatan titik tengah untuk mendapatkan titik pada kurva bezier dengan nilai t = (t1 + t2)/2 dengan di awal

diketahui titik awal kurva (t = 0) dan titik akhir kurva (t = 1) sehingga dapat mengaproksimasi titik pada kurva bezier dengan akurasi yang semakin berdasarkan jumlah iterasi yang dilakukan, pendekatan divide and conquer pada metode ini dilakukan dengan mempartisi kurva bezier menjadi beberapa kurva bezier lainnya dan menghitung hanya satu titik untuk setiap partisi kurva bezier. Kemudian, kurva tersebut di-*merge* dengan satu sama lain sehingga didapat kurva bezier yang lebih akurat. Secara rinci algoritma dijalankan sebagai berikut

1. Titik-titik awal kurva bezier terdefinisi dalam list 'Points' dan kurva bezier akan diaproksimasi hingga i iterasi
2. current_iteration diinisialisasi dengan nilai 0
3. Derajat kurva, n diinisialisasi dengan jumlah point - 1
4. Kemudian, titik-titik pada iterasi berikutnya dihitung dengan divide and conquer yang terbagi menjadi dua kasus
    4.1. Base Case (current_iteration = 0)
        4.1.1. Untuk setiap pasangan point pada list Points secara terurut, dihitung midpoint dari pasangan point tersebut dan disimpan ke dalam list first_midpoints secara terurut juga
        4.1.2. Untuk setiap pasangan point pada list Point secara terurut, dihitung midpoint dari pasangan point tersebut dan disimpan ke dalam list second_midpoints
        4.1.3. Lakukan terus menerus hingga didapat list nth_midpoints berisi 1 point, dimana n adalah derajat dari kurva
        4.1.4. Dibuat list 'Points' baru berisi elemen pertama dari Points awal dan elemen pertama dari masing-masing list nth_midpoints secara terurut diikuti elemen terakhir dari masing-masing list (n-1)th_midpoints hingga first_midpoints dan elemen terakhir dari Points awal
    4.2. Kasus Divide and Conquer (current_iteration > 0)
        4.2.1. List Points dibagi menjadi dua yaitu first_half_points index [0:N/2] (inklusif) dan second_half_points index [N/2:N-1] (inklusif)
        4.2.2. Kedua paruh dari kurva bezier tersebut inisialisasi juga menjadi kurva bezier yang terbentuk dari titik-titik tersebut dengan derajat

sama dengan kurva asli dan current_iteration 1 lebih kecil daripada kurva asli

4.2.3. Kedua kurva bezier tersebut dihitung iterasi berikutnya dengan prosedur yang sama

4.2.4. Kedua kurva bezier yang telah dihitung iterasi berikutnya digabungkan kembali menjadi satu list of points dan menghasilkan kurva bezier dengan titik awal dan titik akhir yang sesuai dan current_iterationnya bertambah satu

5. Perhitungan iterasi berikutnya terus dilakukan hingga current_iteration = i
6. Kurva bezier dapat digambar dengan cara menarik garis untuk setiap pasangan point yang berindeks genap pada list of points secara berurutan

BAB III Implementasi dan Pengujian

A. Implementasi Algoritma

Kedua algoritma diimplementasikan dalam bahasa python dengan menggunakan kelas BezierCurve dengan source code inti algoritma sebagai berikut :

Algoritma Divide and Conquer :

```
def merge_curve(curve1 : "BezierCurve", curve2 : "BezierCurve") -> "BezierCurve" :
    if (curve1.current_iteration == curve2.current_iteration and curve1.degree == curve2.degree) :
        points = curve1.points + curve2.points[1:]

        result = BezierCurve(points, curve1.current_iteration + 1, curve1.degree)
        result.drawn_points = []
        for i in range(len(curve1.drawn_points)) :
            if (i == 0) :
                result.drawn_points.append(curve1.drawn_points[i] + curve2.drawn_points[i][1:])
            else :
                result.drawn_points.append(curve1.drawn_points[i] + curve2.drawn_points[i])

        return result

def go_next(self) :
    if (self.current_iteration == 0) :
        self.base_case()
    else :
        first_half_points = self.points[0:len(self.points)//2 + 1]
        second_half_points = self.points[len(self.points)//2:]

        first_half_curve = BezierCurve(first_half_points, self.current_iteration - 1, self.degree)
        second_half_curve = BezierCurve(second_half_points, self.current_iteration - 1, self.degree)

        first_half_curve.go_iterate(self.current_iteration)
        second_half_curve.go_iterate(self.current_iteration)

        result = BezierCurve.merge_curve(first_half_curve, second_half_curve)
        self.points = result.points
        self.current_iteration = result.current_iteration
        self.drawn_points = result.drawn_points
    self.memo.append(self.copy_curve())
```

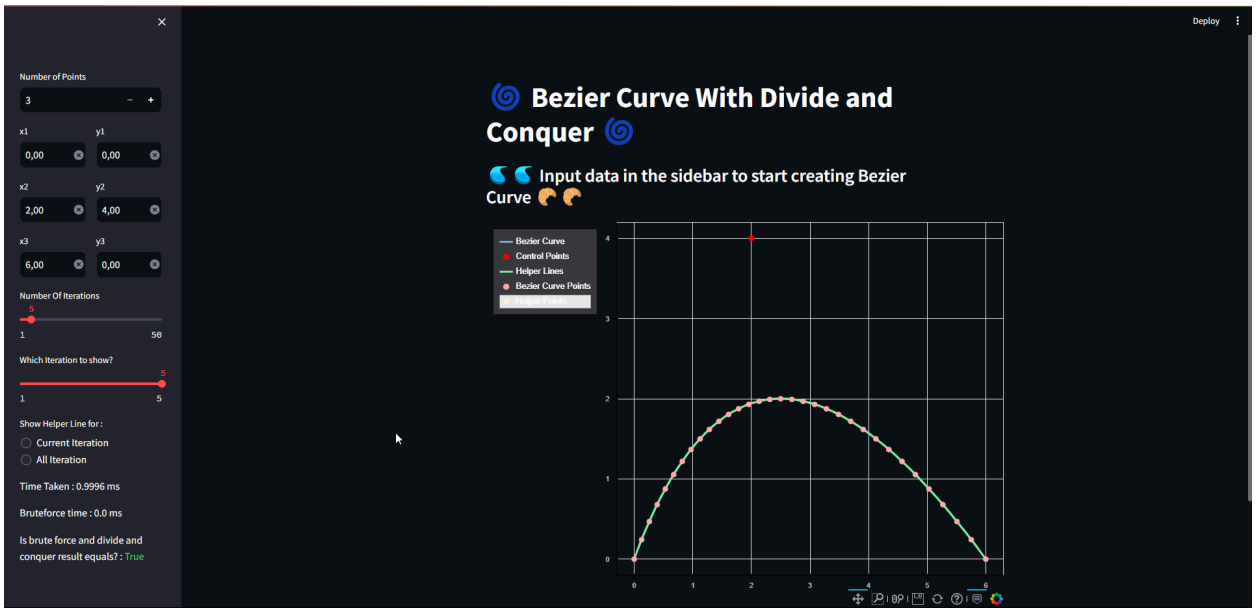
```
def base_case(self) :
    self.drawn_points = [self.points]
    self.memo.append(self.copy_curve())

    self.drawn_points = []
    current_points = self.points
    temp = [self.points[i] for i in range(len(self.points))]
    self.points = temp
    self.drawn_points.append([temp[i] for i in range(len(temp))])
    for i in range(self.degree) :
        new_points = []
        for j in range(len(current_points) - 1) :
            new_points.append(Point.midpoint(current_points[j], current_points[j + 1]))
        self.drawn_points.append(new_points)
        current_points = new_points
    self.points = BezierCurve.get_points_from_drawn_points(self.drawn_points)
    self.current_iteration = 1

def get_points_from_drawn_points(drawn_points) :
    points = []
    for i in range(len(drawn_points)) :
        points.append(drawn_points[i][0])
    for i in range(len(drawn_points) - 2, -1, -1) :
        points.append(drawn_points[i][-1])
    return points

def go_iterate(self, n : int) :

    start = time.time_ns()
    while (self.current_iteration != n) :
        self.go_next()
    self.time_taken = (time.time_ns() - start)/1000000
```

Algoritma Brute Force :

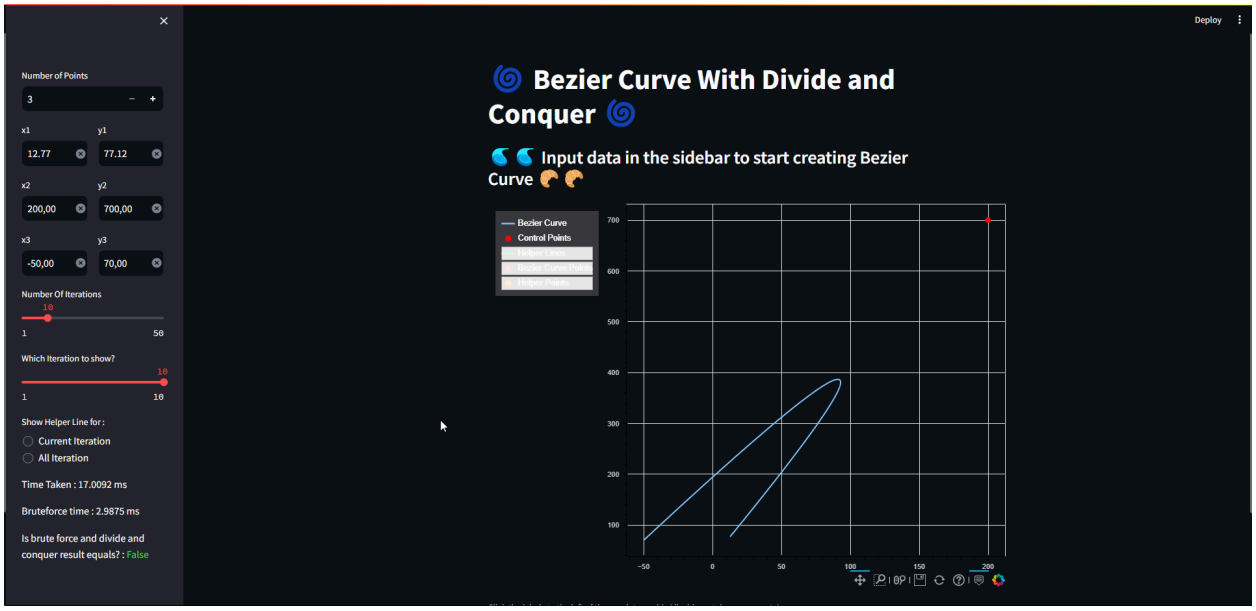
```python
def solve_by_bruteforce(self, iteration) :

    start_time = time.time_ns()
    incrementor = 1/(2**iteration)
    t = 0

    self.bruteforce_points = []
    n = len(self.original_points) - 1

    while (t <= 1) :
        new_x = 0
        new_y = 0
        for i in range(n + 1) :
            new_x += math.comb(n, i) * ((1-t)**(n - i)) * (t ** i) * self.original_points[i].x
            new_y += math.comb(n, i) * ((1-t)**(n - i)) * (t ** i) * self.original_points[i].y
        self.bruteforce_points.append(Point(new_x, new_y))
        t += incrementor

    self.bruteforce_time = (time.time_ns() - start_time)/1000000
    return self.bruteforce_points
```

B. Pengujian

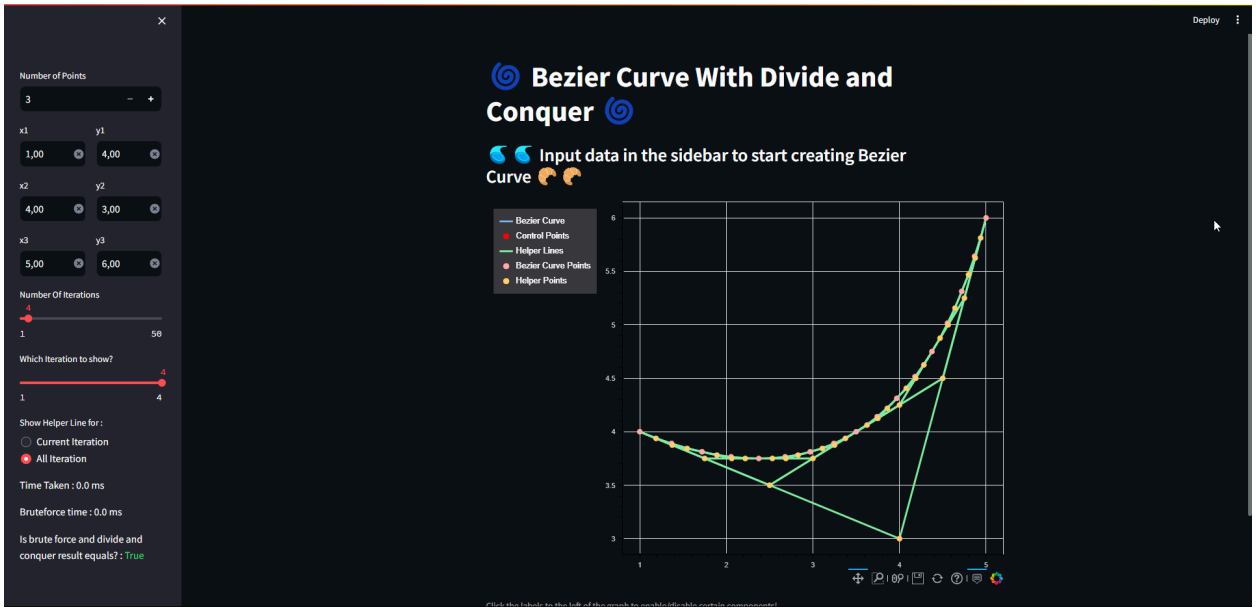
1. Kurva Bezier Kuadratik [(0, 0), (2, 4), (6, 0)], 5 iterasi



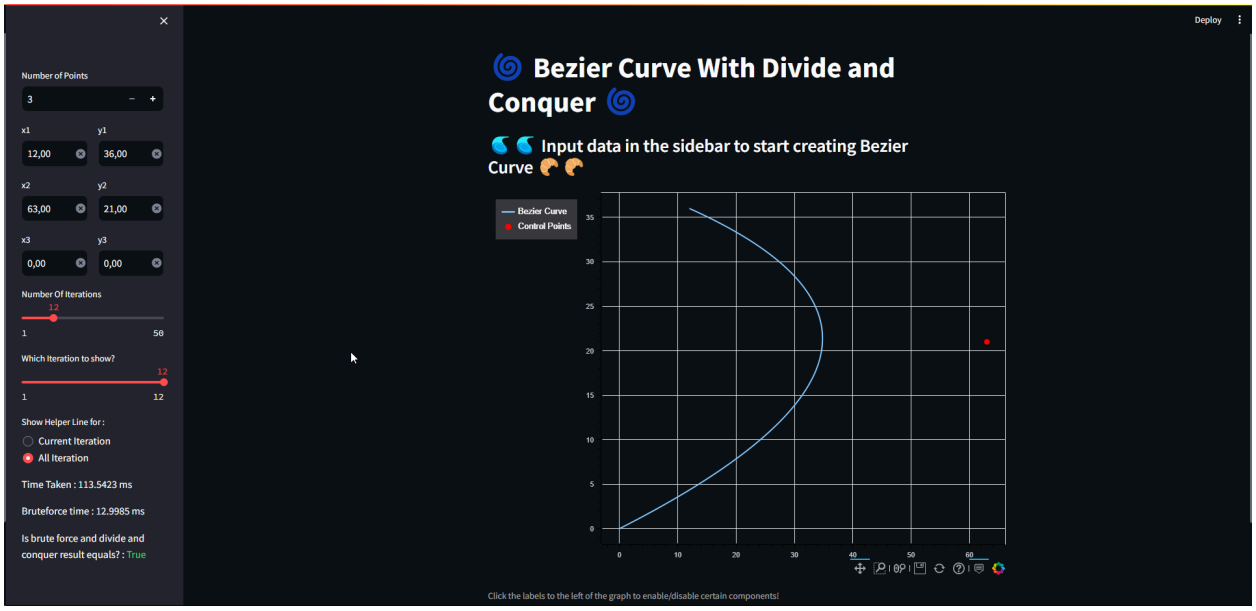
2. Kurva Bezier Kuadratik [(12.77, 77.12), (200, 700), (-50, 70)], 10 iterasi



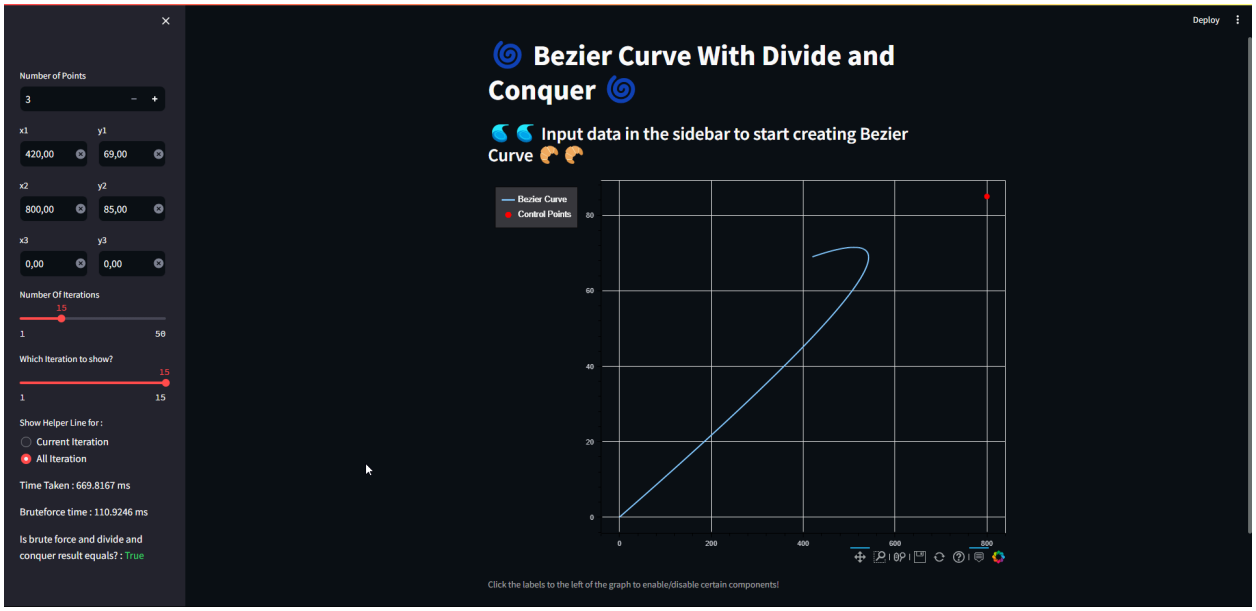
3. Kurva Bezier Kuadratik [(1, 4), (4, 3), (5, 6)], 4 iterasi with helper lines



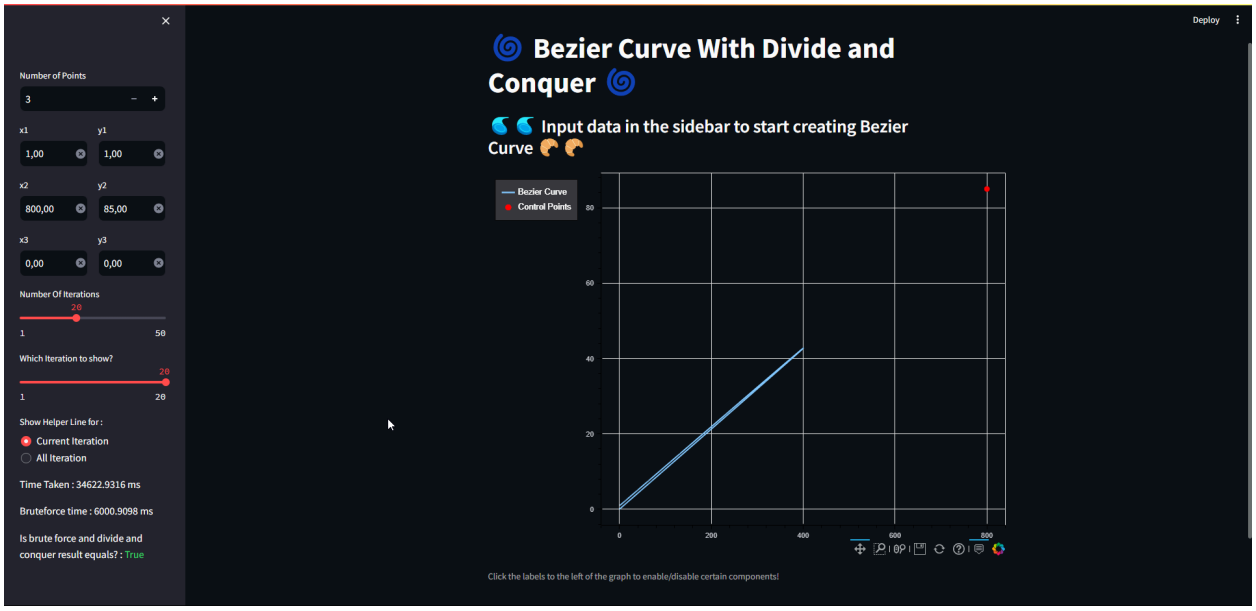
4. Kurva Bezier Kuadratik [(12, 36), (63, 21), (0, 0)], 12 iteration



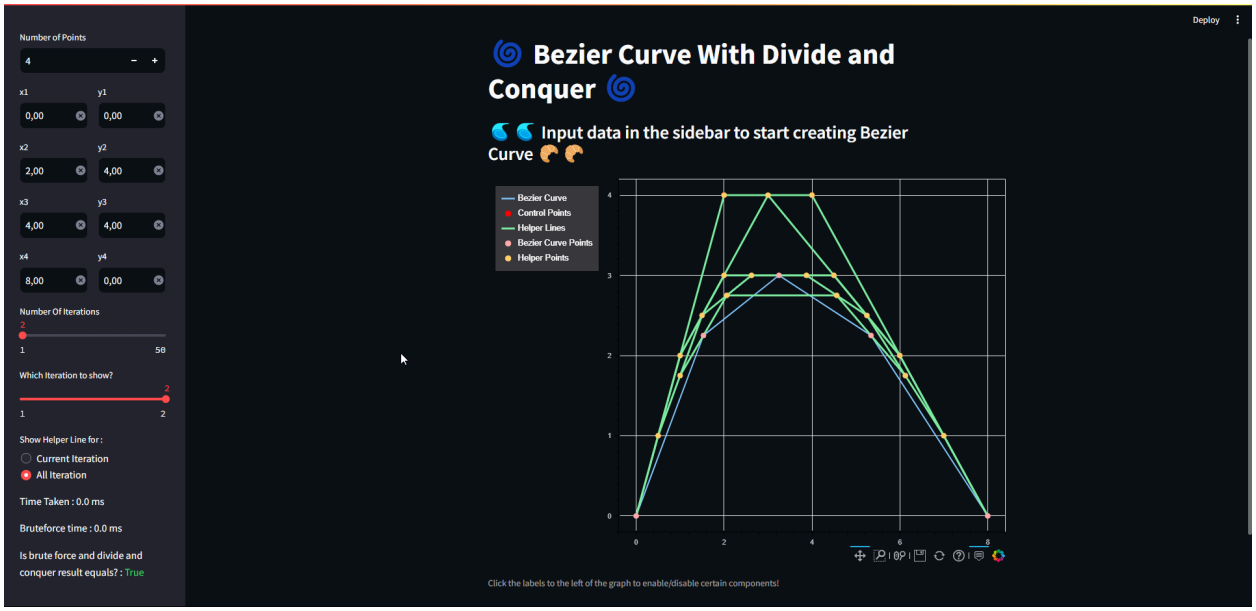
5. Kurva Bezier Kuadratik [(420, 69), (800, 85), (0, 0)], 15 iteration



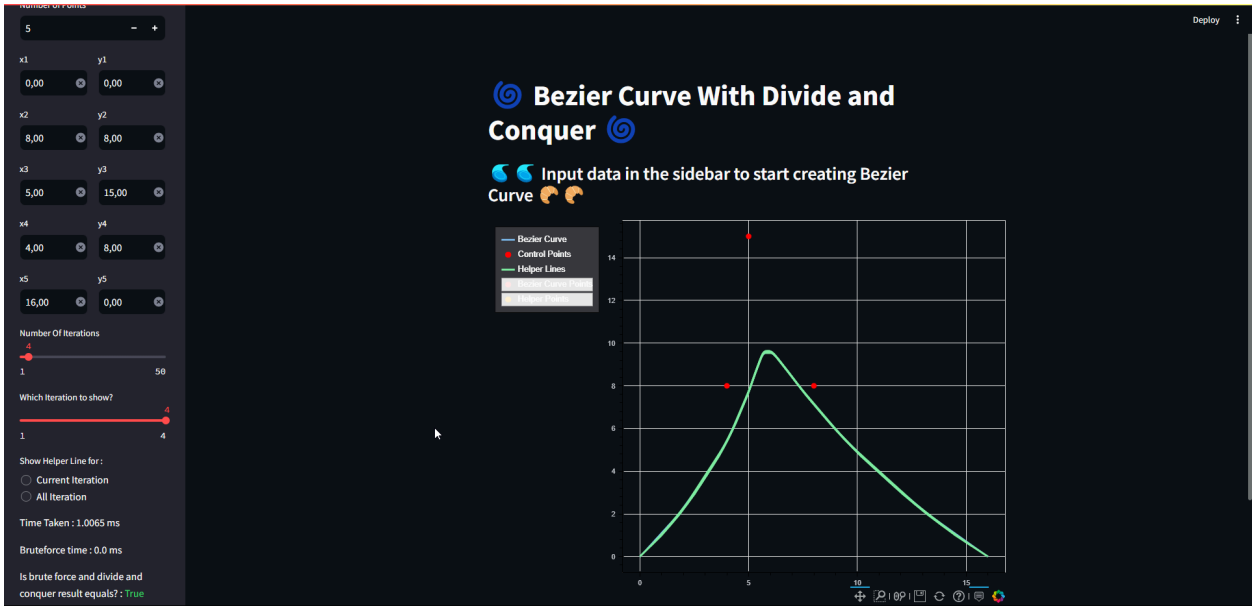
6. Kurva Bezier Kuadratik [(1, 1), (800, 85), (0, 0)], 20 iteration



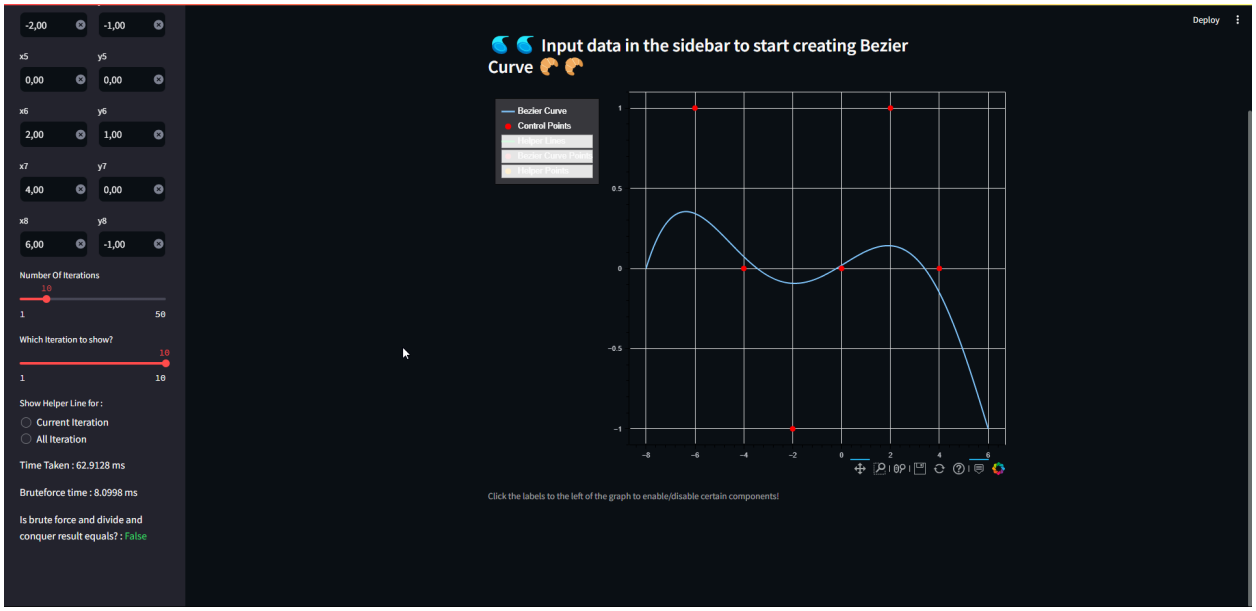
7. Kurva Bezier cubic [(0, 0), (2, 4), (4, 4) ,(8, 0)], iteration 2



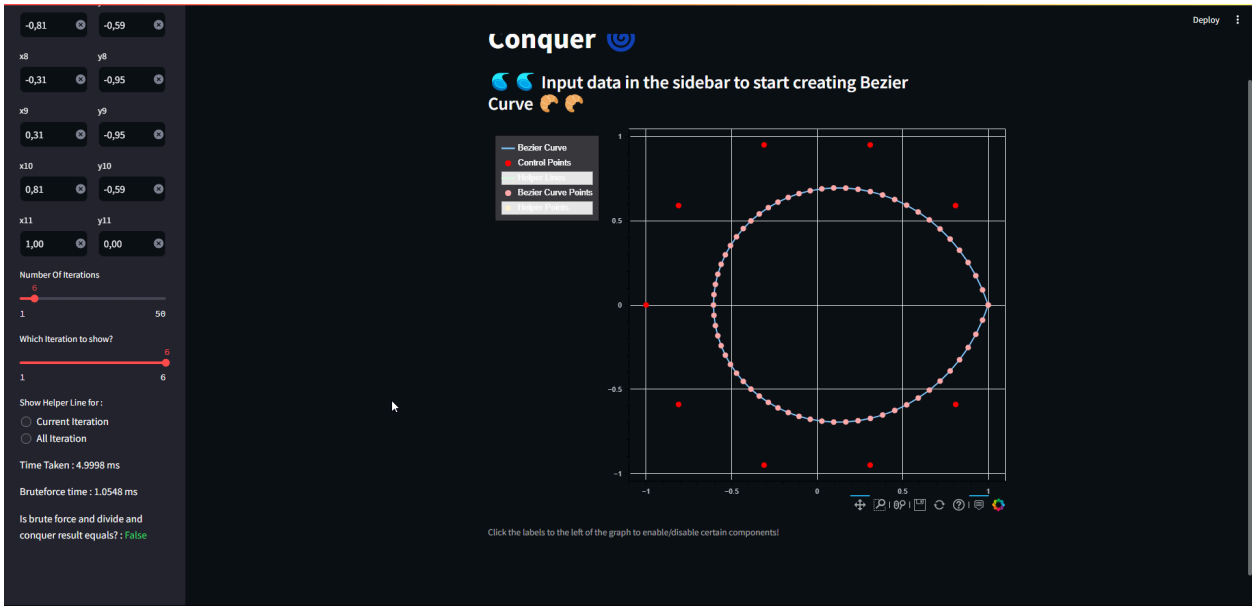
8. Kurva Bezier kuartik [(0, 0), (8, 8), (5, 15), (4, 8), (16, 0)], iteration 4



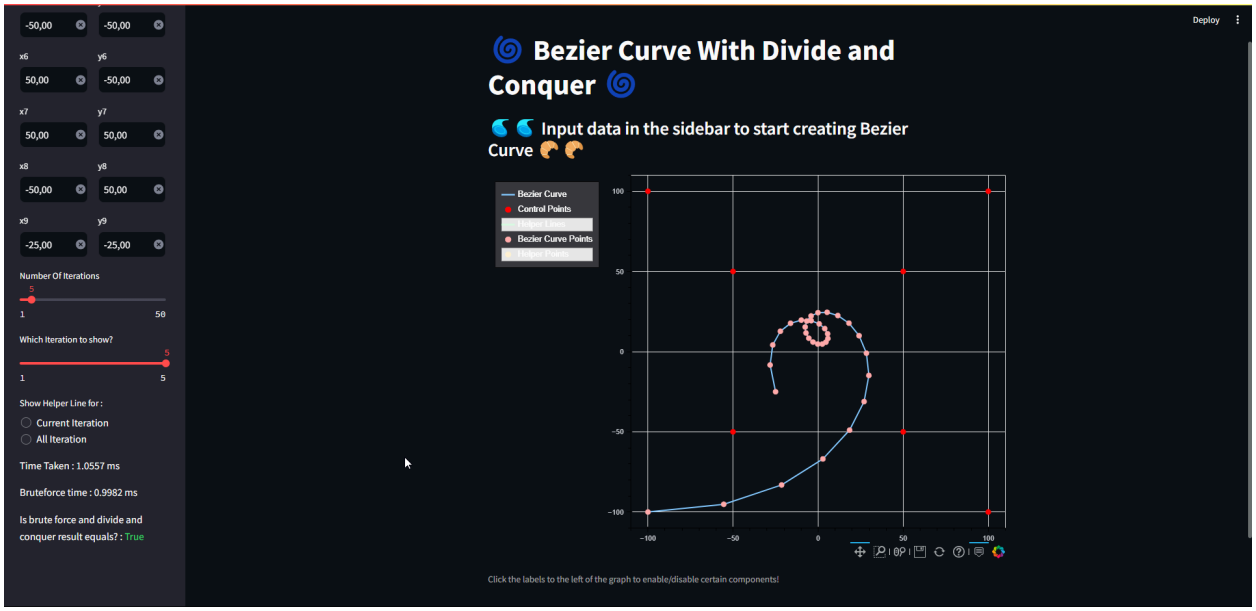
9. Kurva Bezier derajat 7 [(-8, 0), (-6, 1), (-4, 0), (-2, -1), (0, 0), (2, 1), (4, 0), (6, -1)], iteration 10



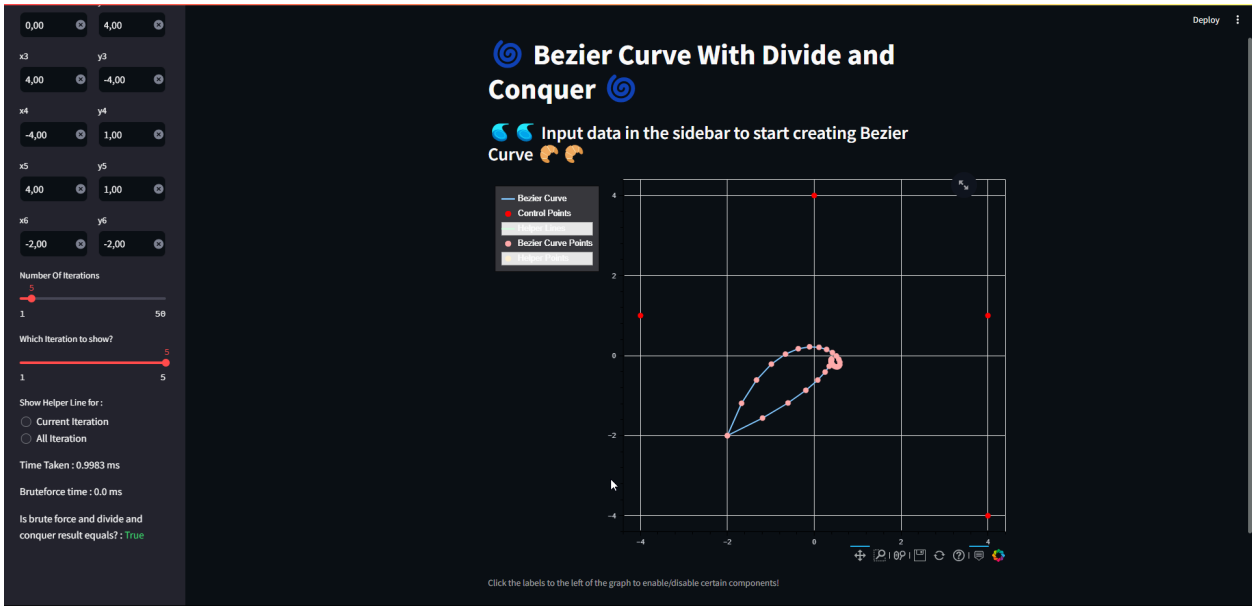
10. Kurva Bezier derajat 10 [(1, 0), (0.81, 0.59), (0.31, 0.95), (-0.31, 0.95), (-0.81, 0.59), (-1, 0), (-0.81, -0.59), (-0.31, -0.95), (0.31, -0.95), (0.81, -0.59), (1, 0)], iteration 6



11. Kurva Bezier derajat 8 [(-100, -100), (100, -100), (100, 100), (-100, 100), (-50, -50), (50, -50), (50, 50), (-50, 50), (-25, -25)], iteration 5



12. Kurva Bezier derajat 5 [(-2, -2), (0, 4), (4, -4), (-4, 1), (4, 1), (-2, -2)], iteration 5


0,00
4,00
x3
y3
4,00
-4,00
x4
y4
-4,00
1,00
x5
y5
4,00
1,00
x6
y6
-2,00
-2,00
Number Of Iterations
5
1
50
Which Iteration to show?
5
1
5
Show Helper Line for :
Current Iteration
All Iteration
Time Taken : 0.9983 ms
Bruteforce time : 0.0 ms
Is brute force and divide and conquer result equals? : True
Deploy
Bezier Curve With Divide and Conquer
Input data in the sidebar to start creating Bezier Curve
Bezier Curve
Control Points
Bezier Curve Points
Click the labels to the left of the graph to enable/disable certain components!

BAB IV Analisis Algoritma

Dalam melakukan analisis untuk kedua algoritma, akan dihitung jumlah operator dasar yang diperlukan untuk menghasilkan bezier curve, pada implementasi dnc operasi dasar yang dihitung hanyalah yang dibutuhkan dalam perhitungan kurva tanpa menghitung oprasi yang dibutuhkan untuk menggambar kurva

A. Analisis Algoritma Brute Force

```
def solve_by_bruteforce(self, iteration) :

    start_time = time.time_ns()
    incrementor = 1/(2**iteration)
    t = 0

    self.bruteforce_points = []
    n = len(self.original_points) - 1

    while (t <= 1) :
        new_x = 0
        new_y = 0
        for i in range(n + 1) :
            new_x += math.comb(n, i) * ((1-t)**(n - i)) * (t ** i) * self.original_points[i].x
            new_y += math.comb(n, i) * ((1-t)**(n - i)) * (t ** i) * self.original_points[i].y
        self.bruteforce_points.append(Point(new_x, new_y))
        t += incrementor

    self.bruteforce_time = (time.time_ns() - start_time)/1000000
    return self.bruteforce_points
```

looping sebanyak 2^i kali
2x assignment
looping n kali
Kompleksitan O(log(n - i))
Kompleksitas O(log i)
2x addition
kompleksitas O(N)

Berdasarkan analisis tersebut didapat T(n) = 2^i (2 + n(n + log(n + i) + log(i) + 2)) dengan mengasumsikan operasi dari fungsi bawaaan sebagai kompleksitasnya. Oleh karena itu, didapat kompleksitas dari algoritma ini dengan i adalah iterasi kurva bezier dan n adalah derajat kurva sebesar 2^i n^2 untuk menghasilkan 2^i points. Sehingga kompleksitas perhitungan setiap point adalah O(2^i n^2 / (2^i)) = O(n^2)

B. Analisis Algoritma Divide and Conquer

```
def base_case(self) :
    self.drawn_points = [self.points]
    self.memo.append(self.copy_curve())

    self.drawn_points = []
    current_points = self.points
    temp = [self.points[i] for i in range(len(self.points))]
    self.points = temp
    self.drawn_points.append([temp[i] for i in range(len(temp))])
    for i in range(self.degree) :  looping n kali
        new_points = []
        for j in range(len(current_points) - 1) : looping sebanyak length current_points = n + 1, n, n-1, n-2 ... 1
            new_points.append(Point.midpoint(current_points[j], current_points[j + 1]))  constant time
        self.drawn_points.append(new_points)
        current_points = new_points
    self.points = BezierCurve.get_points_from_drawn_points(self.drawn_points)
    self.current_iteration = 1
```

Fungsi base_case melakukan looping sebanyak n kali diikut dengan looping sebanyak current_points yang memiliki length [n+1..1] sehingga total operasi yang dilakukan adalah sum(1..n+1) atau (n+2)*(n+1)/2

```
def merge_curve(curve1 : "BezierCurve", curve2 : "BezierCurve") -> "BezierCurve" :
    if (curve1.current_iteration == curve2.current_iteration and curve1.degree == curve2.degree) :
        points = curve1.points + curve2.points[1:]

        result = BezierCurve(points, curve1.current_iteration + 1, curve1.degree)
        result.drawn_points = []
        for i in range(len(curve1.drawn_points)) :
            if (i == 0) :
                result.drawn_points.append(curve1.drawn_points[i] + curve2.drawn_points[i][1:])
            else :
                result.drawn_points.append(curve1.drawn_points[i] + curve2.drawn_points[i])

        return result
```

Fungsi merge_curve dapat dilakukan dalam kompleksitas O(jumlah points) dengan jumlah points sebanding 2^current_iteration

```
def go_next(self) :
    if (self.current_iteration == 0) :
        self.base_case() O(n^2)
    else :
        first_half_points = self.points[0:len(self.points)//2 + 1]
        second_half_points = self.points[len(self.points)//2:]

        first_half_curve = BezierCurve(first_half_points, self.current_iteration - 1, self.degree)
        second_half_curve = BezierCurve(second_half_points, self.current_iteration - 1, self.degree)

        first_half_curve.go_iterate(self.current_iteration)
        second_half_curve.go_iterate(self.current_iteration)

        result = BezierCurve.merge_curve(first_half_curve, second_half_curve)
        self.points = result.points
        self.current_iteration = result.current_iteration
        self.drawn_points = result.drawn_points
    self.memo.append(self.copy_curve())
```

membutuhkan log2(current_points) level hingga sampai ke base case dengan masing-masing level berisi 2^(level) instance

2^(current_iteration)

Fungsi go_next memiliki kompleksitas O(n^2) untuk current_iteration = 1. Untuk kasus i > 1 akan didapat jumlah instance bezier curve pada base case yang dibuat sebanyak 2^(level) = 2^(log2(current_points)) = current_points dan merge_curve dilakukan sebanyak 1 + 2 + 4 + 8 … + current_points = log(2^(log2(current_points) + 1)) = log(2*current_points) = O(log (current_points)) seihngga kompleksitas dari fungsi go_next secara keseluruhan adalah O(n^2 * O(current_points)) = O(n^2 current_points)

```
def go_iterate(self, n : int) :

    start = time.time_ns()
    while (self.current_iteration != n) :
        self.go_next()
    self.time_taken = (time.time_ns() - start)/1000000
```

current_iteration = [0..iteration]

Setiap iteration jumlah current_points akan bernilai current_iteration^i + 1 sehingga kompleksitas total dari program adalah sum(i = 0 -> iteration O(n^2 2^i)) = O(n^2 2^(i+1)) = **O(n^2 2^i))**

C. Perbandingan kedua algoritma

Berdasarkan hasil analisis kompleksitas algoritma didapat bahwa kedua program memiliki kompleksitas yang sama. Hal tersebut dapat divalidasi melalui hasil pengujian yang menunjukkan bahwa rasio waktu yang dibutuhkan kedua algoritma adalah konstan, yaitu sekitar 10 : 1 untuk algoritma divide and conquer terhadap brute force. Berdasarkan itu juga, dapat

diobservasi bahwa algoritma divide and conquer lebih lambat 10x lipat dibandingkan dengan algoritma brute force. Hal tersebut disebabkan pada algoritma divide and conquer dilakukan banyak perhitungan untuk menghitung midpoint dari titik-titik yang tidak menjadi bagian dari kurva. Selain itu, faktor yang lebih berpengaruh adalah karena pada algoritma divide and conquer dilakukan lebih banyak perhitungan yang digunakan untuk visualisasi.

BAB V Kesimpulan

Kesimpulan

Algoritma divide and conquer adalah algoritma yang dapat menyederhanakan suatu permasalahan dengan membaginya menjadi masalah-masalah yang lebih kecil yang kemudian digabungkan untuk menyelesaikan masalah secara keseluruhan. Namun, algoritma divide and conquer tidak menjamin bahwa algoritma yang dihasilkan akan lebih sangkil dibandingkan dengan algoritma brute force

BAB VI Lampiran

Link Repository Github : https://github.com/abdulrafirh/Tucil2_13522089.git

Poin	Ya	Tidak
1. Program berhasil dijalankan	V	
2. Program dapat melakukan visualisasi kurva bezier	V	
3. Solusi yang diberikan program optimal	V	
4. Program dapat membuat kurva untuk n titik kontrol	V	
5. Program dapat melakukan visualisasi pembuatan kurva	V	

.