LAPORAN TUGAS KECIL 2

IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2023/2024

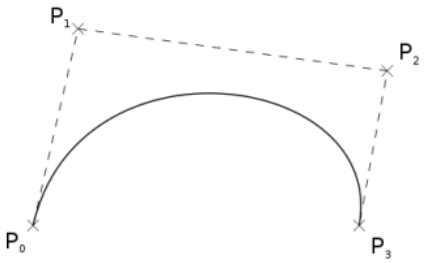
MEMBANGUN KURVA BÉZIER DENGAN ALGORITMA TITIK TENGAH BERBASIS DIVIDE AND CONQUER

DIBUAT OLEH:

Andi Marihot Sitorus

13522138

1. Teori Dasar



Gambar 1. Kurva Bézier

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. Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P0 dan P1 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 Q0 yang berada pada garis yang dibentuk oleh P0 dan P1, maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

Semakin banyak titik kontrol yang digunakan maka persamaan yang digunakan juga akan semakin rumit. Seperti pada persamaan parametrik untuk n=2 berikut.

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

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

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

Sehingga jika Q_0 dan Q_1 disubstitusi ke R_0 didapatkan:

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

2. Algoritma Brute Force untuk 3 titik

Untuk membuat kurva bezier dengan algoritma brute force, kita terlebih dahulu menentukan persamaan yang digunakan untuk mencari titik pada kurva bezier dari titik kontrol yang diberikan. Melihat persamaan yang semakin kompleks tersebut, kita dapat menemukan bahwa koefisien dari variabel pada persamaan tersebut mengikuti segitiga pascal berikut.

$$
\begin{array}{ccccccc}
 & & & 1 & & & \\
 & & 1 & & 1 & & \\
 & 1 & & 2 & & 1 & \\
1 & & 3 & & 3 & & 1 \\
1 & 4 & & 6 & & 4 & 1 \\
1 & 5 & 10 & & 10 & 5 & 1
\end{array}
$$

Gambar 2. Segitiga Pascal

Dalam implementasinya di kode python, fungsi untuk koefisien akan menjadi berikut.

```python
def koefisien(n):
    base = [1,1]
    if n==1:
        return base
    else :
        new = []
        for i in range(1, n):
            new = [1 for i in range(len(base)+1)]
            for j in range (1,len(new)-1):
                new[j] = base[j-1]+base[j]
            base = new
        return new
```

Fungsi akan mengembalikan koefisien sesuai dengan jumlah titik kontrol -1 sebagai n.

Oleh karena itu kita juga bisa menggunakan persamaan binomial berikut.

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}.$$

dimana, x sebagai t dan y sebagai 1-t.

Lalu, agar tepat digunakan sebagai pembanding terhadap algoritma titik tengah berbasis divide and conquer, nilai t yang digunakan akan berkisar dari 0 hingga 1 sebanyak 2^iterasi+1.

Sehingga dapat diimplementasikan sebagai kode berikut.

```python
n = len(control_points)
curve_points = []
koef = koefisien(n-1)
for j in range(2**iterations + 1):
    x=0
    y=0
    t = j / (2**iterations)
    for i in range (n):
        x += koef[i] * (1-t)**(n-1-i) * t**i * control_points[i][0]
        y += koef[i] * (1-t)**(n-1-i) * t**i * control_points[i][1]
    curve_points.append([x, y])
```

3. Algoritma Titik Tengah Berbasis Divine and Conquer untuk 3 titik

Kurva bezier dapat juga dibuat dengan mencari titik tengah dari titik-titik yang diberikan, lalu mencari lagi titik tengah dari titik-titik yang ditemukan hingga akhirnya mendapat 1 titik tengah paling akhir untuk 1 kali iterasi.

Lebih jelasnya Misalkan terdapat tiga buah titik, P0, P1, dan P2, dengan titik P1 menjadi titik kontrol antara, maka:
a. Buatlah sebuah titik baru mid1 yang berada di tengah garis yang menghubungkan P0 dan P1, serta titik mid2 yang berada di tengah garis yang menghubunhkan P1 dan P2.
b. Hubungkan Q0 dan Q1 sehingga terbentuk sebuah garis baru.
c. Buatlah sebuah titik baru mid12 yang berada di tengah mid1 dan mid2.
d. Buatlah sebuah garis yang menghubungkan P0 – mid12 - P2. Melalui proses di atas, telah dilakukan 1 buah iterasi dan diperoleh sebuah “kurva” yang belum cukup mulus dengan aproksimasi 3 buah titik.

Untuk implementasi dari algoritma divine and conquer, jika iterasi yang digunakan lebih dari 1 kali, maka kita akan menggunakan ketiga titik yang baru saja kita dapatkan sebagai titik kontrolnya sehingga ada 5 buah titik, P0, mid1, mid12, mid2, P2. Lalu kelima titik tersebut akan dibagi 2 menjadi left dan right (**divide**) yang masing masing terdiri atas 3 titik. Yaitu, left = P0, mid1, mid12 dan right = mid12, mid2, P2. Kemudian langkah sebelumnya akan diterapkan untuk menyelesaikan (**conquer**) bagian left dan right secara terpisah. Lalu digabungkan kembali setelah prosesnya selesai. Sehingga akan didapatkan 2^iterasi +1 buah titik pada kurva bezier.
Berikut adalah implementasi dari algoritma tersebut dalam python

```python
def bezier(control_points, iterations):
    if iterations == 0:
        return np.array([control_points[0], control_points[2]])

    mid1 = midpoint(control_points[0] ,control_points[1])
    mid2 = midpoint(control_points[1] ,control_points[2])

    mid12 = midpoint(mid1, mid2)

    left = [control_points[0], mid1, mid12,]
    right = [mid12, mid2, control_points[2]]
    leftmid = bezier(left, iterations - 1)
    rightmid = bezier(right, iterations - 1)

    return np.concatenate([leftmid[:-1], rightmid])
```

4. Source Code

4.1. Algoritma Brute Force Untuk 3 Titik

```
import numpy as np
import matplotlib.pyplot as plt
import time
def plot_curve(points, control_points, figure_name):
    plt.figure(figure_name)
    plt.plot(points[:, 0], points[:, 1], 'b-')
    plt.plot(points[:, 0], points[:, 1], 'ro')
    plt.plot(control_points[:,0], control_points[:,1], 'go-')
    plt.title(figure_name)
    plt.show()

def koefisien(n):
    base = [1,1]
    if n==1:
        return base
    else :
        new = []
        for i in range(1, n):
            new = [1 for k in range(len(base)+1)]
            for j in range (1,len(new)-1):
                new[j] = base[j-1]+base[j]
            base = new
        return new

def curve_point(control_points, iterations):
    n = len(control_points)
    curve_points = []
    koef = koefisien(n-1)
    for j in range(2**iterations + 1):
        x=0
        y=0
        t = j / (2**iterations)
        for i in range (n):
            x += koef[i] * (1-t)**(n-1-i) * t**i * control_points[i][0]
            y += koef[i] * (1-t)**(n-1-i) * t**i * control_points[i][1]
        curve_points.append([x, y])
    return np.array(curve_points)


control_points = []
for i in range (3):
    print(f"\nKordinat titik kontrol ke {i+1}")
    x = int(input("Masukkan absis (x): "))
    y = int(input("Masukkan ordinat (y): "))
```

```
    control_points.append([x,y])

iterations = int(input("\nMasukkan jumlah iterasi: "))

start = time.time()
points = curve_point(control_points, iterations)
end = time.time()
runtime = end - start
print("Runtime", runtime, "seconds")

cp = np.array(control_points)
plot_curve(points,cp,"Brute Force Bezier Curve")
```

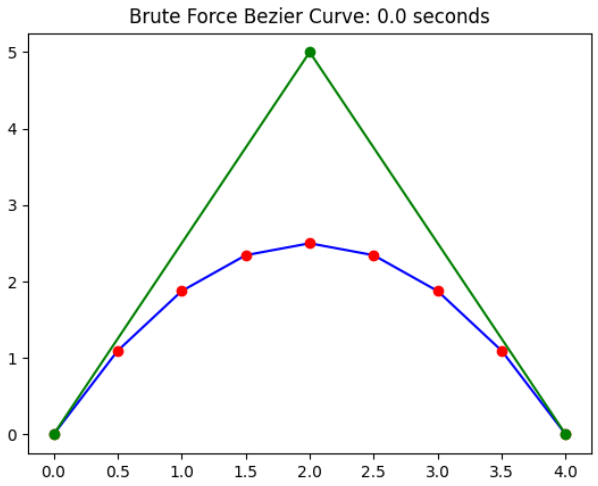
4.2. Algoritma Titik Tengah Berbasis Divine and Conquer Untuk 3 Titik

```
import numpy as np
import matplotlib.pyplot as plt
import time

def plot_curve(points, control_points, figure_name):
    plt.figure(figure_name)
    plt.plot(points[:, 0], points[:, 1], 'b-')
    plt.plot(points[:, 0], points[:, 1], 'ro')
    plt.plot(control_points[:,0], control_points[:,1], 'go-')
    plt.title(figure_name)
    plt.show()

def midpoint(point1, point2):
    return [(point1[0] + point2[0]) / 2 , (point1[1] + point2[1]) / 2]

def bezier(control_points, iterations):
    if iterations == 0:
        return np.array([control_points[0], control_points[2]])

    mid1 = midpoint(control_points[0] ,control_points[1])
    mid2 = midpoint(control_points[1] ,control_points[2])

    mid12 = midpoint(mid1, mid2)

    left = [control_points[0], mid1, mid12,]
    right = [mid12, mid2, control_points[2]]
    leftmid = bezier(left, iterations - 1)
    rightmid = bezier(right, iterations - 1)

    return np.concatenate([leftmid[:-1], rightmid])

control_points = []
for i in range (3):
```

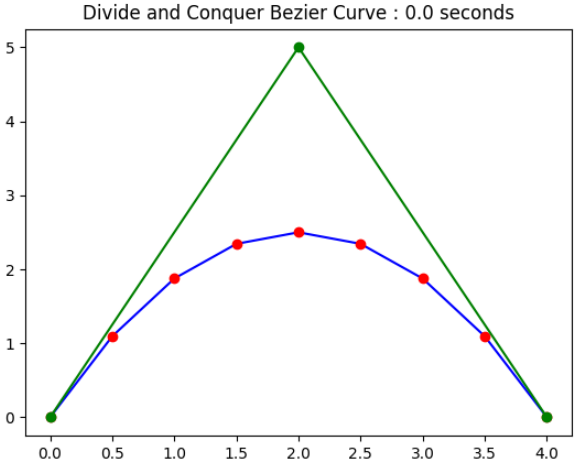
```
    print(f"\nKordinat titik kontrol ke {i+1}")
    x = int(input("Masukkan absis (x): "))
    y = int(input("Masukkan ordinat (y): "))
    control_points.append([x,y])

iterations = int(input("Masukkan jumlah iterasi: "))

cp = np.array(control_points)
start = time.time()
points = bezier(control_points, iterations)
end = time.time()
runtime = end - start
print("Runtime", runtime, "seconds")

# print(len(points))

plot_curve(points, cp,"Divide and Conquer Bezier Curve")
```

5. Uji Coba

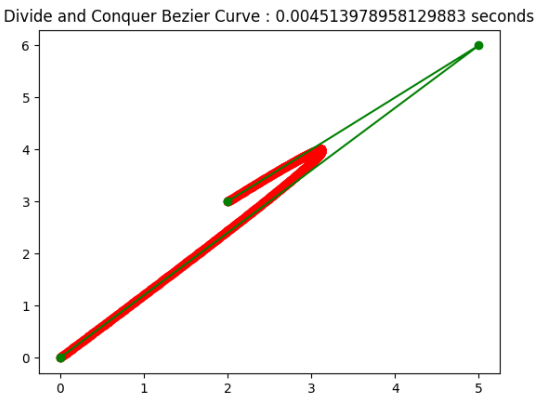
5.1. Testcase 1



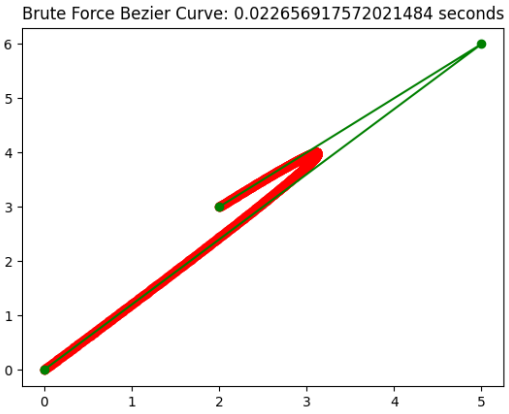

```
Kordinat titik kontrol ke 1
Masukkan absis (x): 0
Masukkan ordinat (y): 0

Kordinat titik kontrol ke 2
Masukkan absis (x): 2
Masukkan ordinat (y): 5

Kordinat titik kontrol ke 3
Masukkan absis (x): 4
Masukkan ordinat (y): 0

Masukkan jumlah iterasi: 3
```



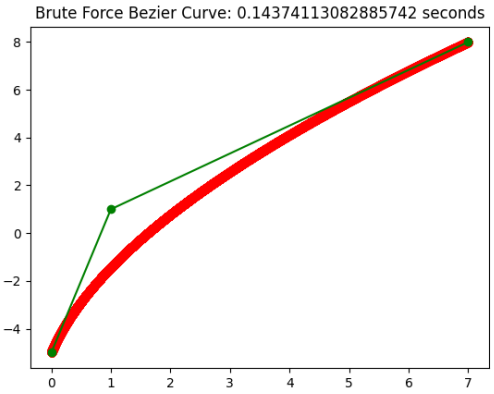
5.2. TestCase 2



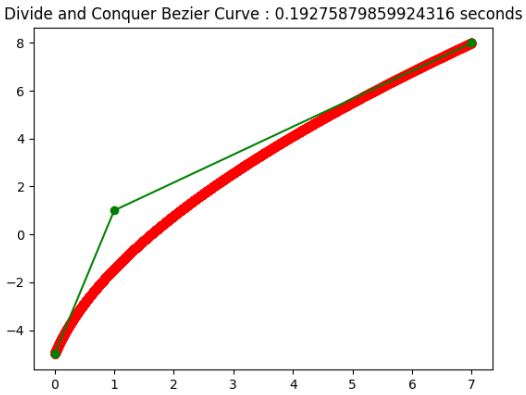

```
Kordinat titik kontrol ke 1
Masukkan absis (x): 0
Masukkan ordinat (y): 0

Kordinat titik kontrol ke 2
Masukkan absis (x): 5
Masukkan ordinat (y): 6

Kordinat titik kontrol ke 3
Masukkan absis (x): 2
Masukkan ordinat (y): 3
Masukkan jumlah iterasi: 10
```



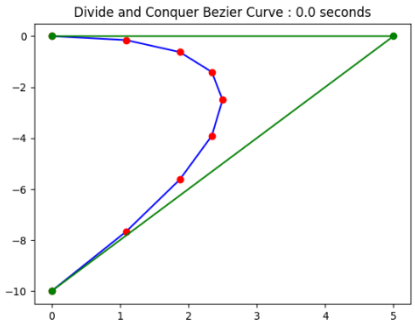
5.3. Testcase 3



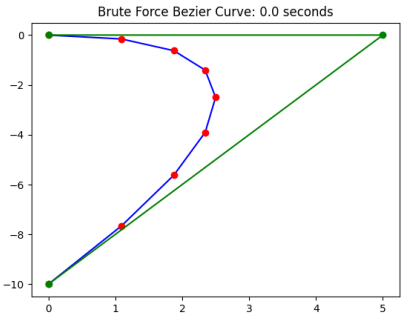

```
Kordinat titik kontrol ke 1
Masukkan absis (x): 7
Masukkan ordinat (y): 8

Kordinat titik kontrol ke 2
Masukkan absis (x): 1
Masukkan ordinat (y): 1

Kordinat titik kontrol ke 3
Masukkan absis (x): 0
Masukkan ordinat (y): -5

Masukkan jumlah iterasi: 15
```



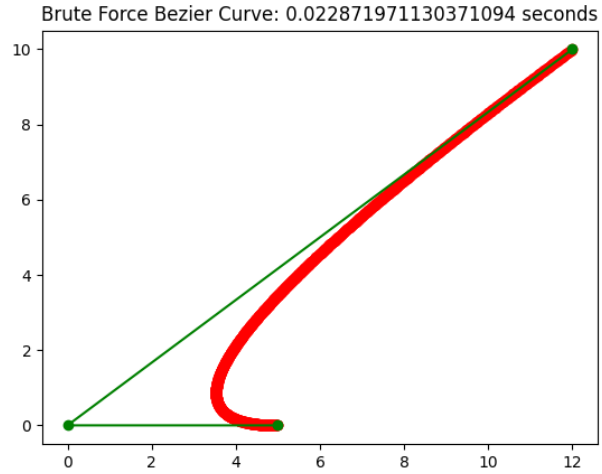
5.4. Testcase 4



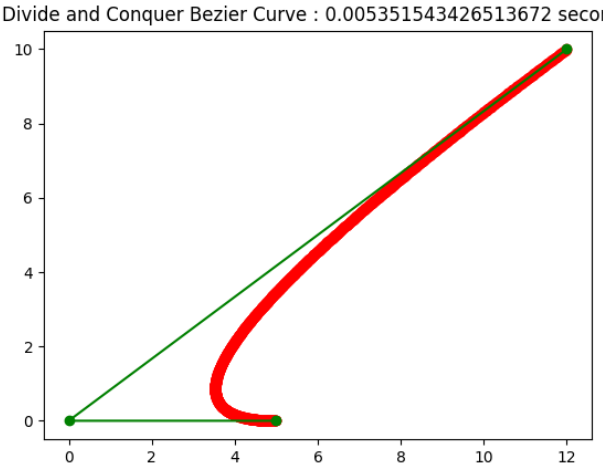

```
Kordinat titik kontrol ke 1
Masukkan absis (x): 0
Masukkan ordinat (y): 0

Kordinat titik kontrol ke 2
Masukkan absis (x): 5
Masukkan ordinat (y): 0

Kordinat titik kontrol ke 3
Masukkan absis (x): 0
Masukkan ordinat (y): -10
Masukkan jumlah iterasi: 3
```



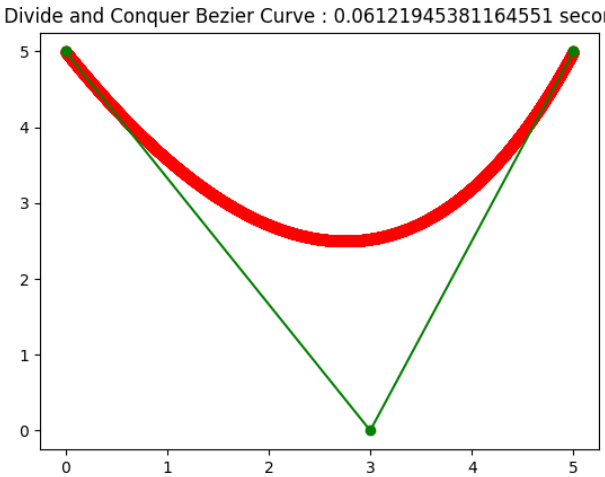
5.5. Testcase 5



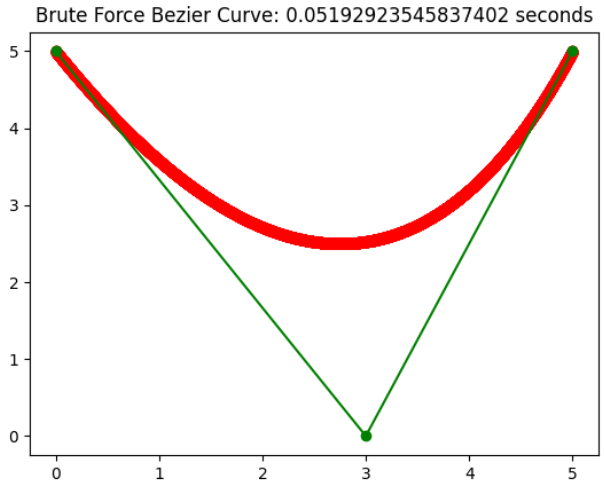

```
Kordinat titik kontrol ke 1
Masukkan absis (x): 12
Masukkan ordinat (y): 10

Kordinat titik kontrol ke 2
Masukkan absis (x): 0
Masukkan ordinat (y): 0

Kordinat titik kontrol ke 3
Masukkan absis (x): 5
Masukkan ordinat (y): 0

Masukkan jumlah iterasi: 10
```



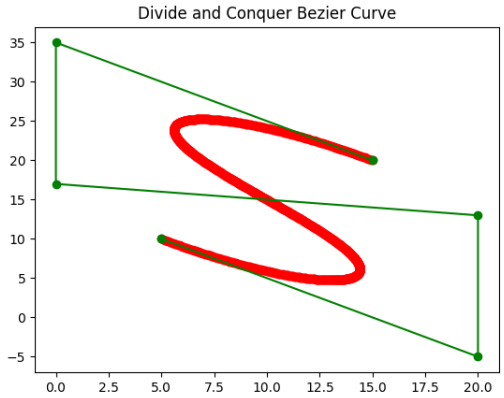
5.6. Testcase 6




```
Kordinat titik kontrol ke 1
Masukkan absis (x): 0
Masukkan ordinat (y): 5

Kordinat titik kontrol ke 2
Masukkan absis (x): 3
Masukkan ordinat (y): 0

Kordinat titik kontrol ke 3
Masukkan absis (x): 5
Masukkan ordinat (y): 5
Masukkan jumlah iterasi: 13
```

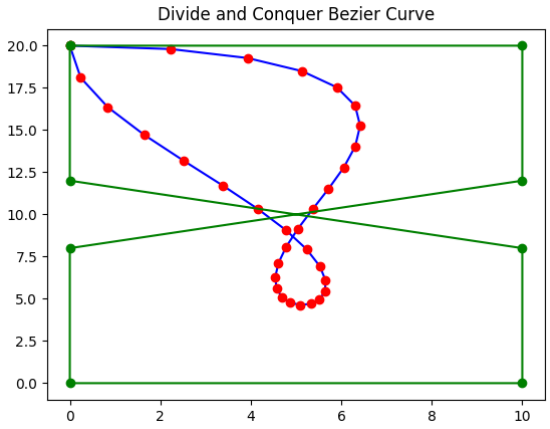


5.7. Testcase 7 (n titik kontrol)




```
Masukkan jumlah titik kontrol: 6

Kordinat titik kontrol ke 1
Masukkan absis (x): 15
Masukkan ordinat (y): 20

Kordinat titik kontrol ke 2
Masukkan absis (x): 0
Masukkan ordinat (y): 35

Kordinat titik kontrol ke 3
Masukkan absis (x): 0
Masukkan ordinat (y): 17

Kordinat titik kontrol ke 4
Masukkan absis (x): 20
Masukkan ordinat (y): 13

Kordinat titik kontrol ke 5
Masukkan absis (x): 20
Masukkan ordinat (y): -5

Kordinat titik kontrol ke 6
Masukkan absis (x): 5
Masukkan ordinat (y): 10

Masukkan jumlah iterasi: 10
Runtime 0.032416582107543945 seconds
```

5.8. Testcase 8 (n titik kontrol)




```
Masukkan jumlah titik kontrol: 9

Kordinat titik kontrol ke 1
Masukkan absis (x): 0
Masukkan ordinat (y): 20

Kordinat titik kontrol ke 2
Masukkan absis (x): 10
Masukkan ordinat (y): 20

Kordinat titik kontrol ke 3
Masukkan absis (x): 10
Masukkan ordinat (y): 12

Kordinat titik kontrol ke 4
Masukkan absis (x): 0
Masukkan ordinat (y): 8

Kordinat titik kontrol ke 5
Masukkan absis (x): 0
Masukkan ordinat (y): 0

Kordinat titik kontrol ke 6
Masukkan absis (x): 10
Masukkan ordinat (y): 0

Kordinat titik kontrol ke 7
Masukkan absis (x): 10
Masukkan ordinat (y): 8

Kordinat titik kontrol ke 8
Masukkan absis (x): 0
Masukkan ordinat (y): 12

Kordinat titik kontrol ke 9
Masukkan absis (x): 0
Masukkan ordinat (y): 20

Masukkan jumlah iterasi: 5
Runtime 0.020621299743652344 seconds
```

6. Analisis

Kurva bezier yang ditunjukkan oleh kedua algoritma tidak memiliki perbedaan yang signifikan. Hal yang dapat diamati adalah perbedaan waktu dari kedua algoritma. Hal ini dapat disebabkan oleh perbedaan kompleksitas algoritma keduanya. Algoritma bruteforce memiliki fungsi koefisien dengan kompeleksitas $O(n^2)$. Kemudian hasil dari fungsi tersebut mengalami loop sebanyak 2^iteration sehingga kompleksitas dari algoritma brute force adalah $O(n^2$ 2^iteration) dimana n adalah jumlah titik kontrol dan iteration adalah jumlah iterasi yang digunakan. Karena nilai n adalah 3, maka kompleksitasnya menjadi O(9.2^iteration). Sedangkan pada algoritma divide and conquer, fungsi melakukan rekursi dimana setiap rekursi akan menghasilkan 2 proses lebih banyak dan jumlah rekursinya bergantung dari jumlah iterasi. Sehingga kompleksitas algoritma divide and conquer adalah O(2^iteration).

Hanya dengan melihat kompleksitas keduanya, kita dapat menyimpulkan bahwa algoritma divide and conquer akan memiliki waktu yang lebih cepat. Namun, hasil tes menunjukkan bahwa mulai dari 13 iterasi, brute force menjadi lebih cepat. Hal itu disebabkan oleh penggunaan rekursi pada algoritma divide and conquer. Jumlah rekursi yang dilakukan meningkat dengan eksponensial sehingga pada iterasi tinggi akan membutuhkan alokasi memori yang cukup besar untuk melakukannya dan memakan lebih banyak waktu. Seballiknya pada algoritma brute force, dilakukan pemrosesan langsung sehingga tidak ada peningkatan penggunaan memori yang signifikan.

7. Implementasi Bonus

7.1. Algoritma Titik Tengah Berbasis Divide and Conquer Untuk n Buah Titik Kontrol

Sama seperti algoritma untuk 3 buah titik, kita akan mencari titik tengah dari titik-titik yang diberikan. Lalu, mencari titik tengah dari titik-titik yang ditemukan, hingga menemukan 1 titik tengah paling akhir. Untuk mengimplementasikannya, kita memerlukan sebuah fungsi yang akan mencari titik tengah dari titik-titik yang diberikan. Misalkan sebuah fungsi mid_algorithm dan ada 5 titik, A, B, C, D, E. Fungsi akan mencari titik tengah mereka yaitu AB, BC, CD ,DE lalu mengembalikan titik A, mid_algorithm(AB, BC, CD, DE), E. Fungsi akan melakukan rekursi hingga hanya terdapat 1 titik tengah. Berikut adalah implementasinya dalam kode python.

```python
def midalgorithm(control_points):
    if (len(control_points)==1):
        return [control_points[0]]

    mid=[]
    for i in range (len(control_points)-1) :
        mid.append(midpoint(control_points[i], control_points[i+1]))

    return [control_points[0]] + midalgorithm(mid) + [control_points[-1]]
```

Setelah itu, hasil dari fungsi tersebut akan dibagi menjadi left dan right sama seperti sebelumnya dan dilakukan rekursi hingga iterasinya habis lalu disatukan. Berikut implementasinya dalam kode python

```python
def bezier(control_points, iterations):
    if iterations == 0:
        return np.array([control_points[0], control_points[-1]])

    mid = midalgorithm(control_points)
    m = len(mid)
    if m % 2 == 0:
        left = [mid[i] for i in range(m//2)]
        right = [mid[i] for i in range(m//2, m)]
    else:
        left = [mid[i] for i in range(m//2 + 1)]
        right = [mid[i] for i in range(m//2, m)]

    leftmid = bezier(left, iterations - 1)
    rightmid = bezier(right, iterations - 1)

    return np.concatenate([leftmid[:-1], rightmid])
```

8. **Pranala**
   https://github.com/MelonSeed9/Tucil2_13522138