

## **Laporan Tugas Kecil 2**

### **Pembuatan Kurva Bezier Menggunakan Algoritma *Divide and Conquer***



**Disusun oleh:**

Zaki Yudhistira Candra 13522031 (K01)

Vanson Kurnialim 13522049 (K01)

**Mata Kuliah IF 2211 - Strategi Algoritma**

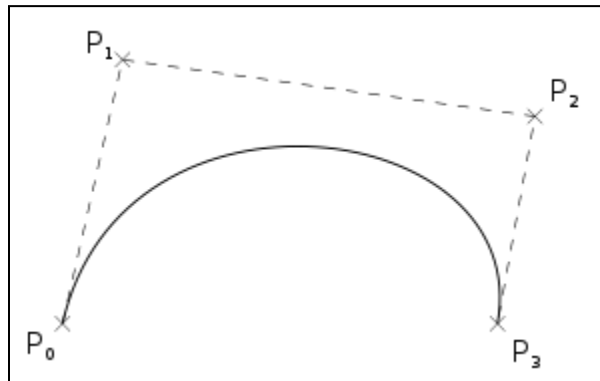
**Program Studi S1 Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

## A. Teori Singkat

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

Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan. Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol  $P_0$  sampai  $P_n$ , dengan  $n$  disebut order ( $n = 1$  untuk linier,  $n = 2$  untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) umumnya tidak terletak pada kurva. Pada gambar 1 diatas, titik kontrol pertama adalah  $P_0$ , sedangkan titik kontrol terakhir adalah  $P_3$ . Titik kontrol  $P_1$  dan  $P_2$  disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.



Gambar 1.1 Kurva Bezier Kubik

Diberikan dua titik  $P_0$  dan  $P_1$ , maka kurva Bézier linear adalah garis lurus dari kedua titik tersebut. Kurva ini diberikan dengan :

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1 - t)\mathbf{P}_0 + t\mathbf{P}_1, t \in [0, 1]$$

Untuk kurva Bézier dengan  $n$  titik kontrol direpresentasikan dengan formula berikut :

$$\mathbf{B}(t) = \sum_{i=0}^n \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i$$
$$= (1-t)^n \mathbf{P}_0 + \binom{n}{1} (1-t)^{n-1} t \mathbf{P}_1 + \cdots + \binom{n}{n-1} (1-t) t^{n-1} \mathbf{P}_{n-1} + t^n \mathbf{P}_n, \quad 0 \leq t \leq 1$$

## B. Deskripsi Program

Program kami diimplementasikan menggunakan bahasa python dan *library* Matplotlib sebagai visualisasi dari kurva bezier. Berikut merupakan beberapa fitur dari program kami:

1. CLI (*Command Line Interface*) based.
2. Pengguna dapat memasukkan sekumpulan titik yang akan menjadi *anchor point* dari kurva bezier.
3. Pengguna dapat memasukkan titik menggunakan *file* dengan ekstensi .txt dengan format sebagai berikut
4. Program dapat menghasilkan sebuah kurva bezier beserta kurva-kurva iterasi pembantu

```
n jumlah titik
X1 Y1
X2 Y2
X3 Y3
X4 Y4
Xn Yn
```

Gambar 2.1. Struktur *file* untuk *input* dari file.

Sebelum menjalankan program, pengguna harus menggunakan sistem operasi minimal windows 7 ataupun sistem operasi linux yang sudah terinstall python versi 3.9.x serta *library* python “Matplotlib”. Disarankan untuk menggunakan python dengan versi yang lebih baru demi performa yang lebih baik. Berikut adalah cara menjalankan program.

1. Buka terminal pada direktori program.

2. Untuk pengguna windows, silahkan ketik `"python src/Main.py"` pada terminal, atau pengguna dapat mengeksekusi file `.bat` yang terletak pada *folder* `bin`.
3. Anda dapat menggunakan file *custom* dengan menaruh file pada direktori `"test"` atau memasukkan titik secara manual.
4. Ikuti arahan program.

### C. Implementasi Program

Program menggunakan 2 pendekatan utama, yaitu pendekatan secara *brute force* dan *divide and conquer*.

#### 1. Pendekatan *Bruteforce*

Program apabila dilaksanakan secara *bruteforce*, akan menggunakan formula seperti yang dijelaskan pada bab pertama. Program akan terlebih dahulu membagi nilai  $t$  menjadi sebanyak  $2^n$  bagian,  $n$  merupakan *depth* atau jumlah iterasi pembuatan bezier curve.

Dengan menggunakan metode di atas, program akan memiliki kompleksitas waktu sebanyak  $O(m2^n)$  berhubung program harus memproses sebanyak  $m$  titik dan  $2^n$  iterasi.

#### 2. Pendekatan *Divide and Conquer*

Dengan pendekatan ini, program akan mencari *midpoint* atau titik tengah dari keseluruhan titik yang ada. Hasil titik tengah tersebut merupakan hasil dari iterasi pertama yang akan disimpan. Titik-titik yang tersisa setelah iterasi pertama akan dibagi dua menjadi kumpulan titik kiri dan juga kanan dengan titik tengah yang dihasilkan sebagai titik akhir pada kumpulan kiri dan sebagai titik awal pada kumpulan titik kanan. Akan dilakukan hal yang sama pada setiap kumpulan yang dihasilkan sampai iterasi ke- $n$ . Pada tiap iterasi hanya akan dihasilkan satu titik dan seluruh titik tersebut beserta dengan titik awal dan akhir akan menyusun kurva bezier yang final. Juga, iterasi penentuan titik acuan membutuhkan sedikitnya 3 titik *anchor*.

Berdasarkan implementasi algoritma *DIVIDE AND CONQUER* kami, ditemukan rumus kompleksitas sebagai berikut :

$T(n) = 2T(n - 1) + m$  dengan  $n$  adalah *depth* dan  $m$  adalah jumlah titik.  $m$  dapat bernilai konstan tergantung dari jumlah *depth* yang diberikan.  $m$  akan selalu bernilai konstan dan tidak berubah-ubah. Untuk  $T(n) = 2T(n - 1)$  akan berlangsung terus seperti sekuens berikut  $T(n) = 2T(n - 2) + 2(n - 1) + 2m$ , maka akan diperoleh kompleksitas algoritma senilai  $O(n^2/2) = O(n^2)$ .

Untuk penjelasan yang lebih jelas, berikut adalah proses divide and conquer dalam suatu iterasi secara prosedural :

1. Membuat suatu list kosong *anchor*
2. Mencari titik tengah *middle* dari kumpulan titik pada iterasi tersebut.
3. Mencatat beberapa *midpoint* yaitu titik tengah yang didapat pada saat pencarian *middle*.
4. *Middle* akan disimpan ke dalam list *results* dan *midpoints* akan disimpan ke dalam *anchor*.
5. Memanfaatkan *anchor*, akan dibentuk 2 list baru yaitu list kiri dan list kanan. List-list tersebut merupakan pembagian proses menjadi 2 bagian (kiri dan kanan) dalam algoritma divide and conquer.
6. Kumpulan list dalam list kiri dan list kanan dianggap sebagai sebuah garis yang utuh dan akan secara terpisah menjadi iterasi lanjutan yang diproses ulang.
7. Setelah semua proses iterasi selesai, list *result* akan menjadi hasil kurva bezier yang final. Untuk pengerjaan bonus, list *anchor* dimanfaatkan untuk menunjukkan proses pembentukan kurva pada setiap iterasinya.

#### D. Implementasi Bonus

Program kami mengimplementasikan kedua bonus spesifikasi yang diberikan, berikut penjelasan lebih lanjut :

1. Bonus n titik kontrol

Program kami dapat menerima n buah titik dan menyusun kurva bezier berdasarkan n buah titik yang diberikan. Baik *bruteforce* maupun *divide and conquer*, dapat mengimplementasikan fitur ini.

2. Bonus visualisasi proses pembuatan kurva

Program kami dapat menampilkan tahapan kurva mana saja yang diperlukan sebelum menghasilkan kurva bezier utama. Tampilan lebih jelasnya akan dilampirkan pada bab pengujian. Hal ini dilakukan dengan menyimpan tahapan-tahapan pembentukan kurva bezier kedalam suatu larik, dan mencetak larik pada akhir eksekusi ketika diperlukan.

## E. Source Code Program

```
class BezierCurve:
    def __init__(self, Iterate) -> None:
        # Attributes assignment
        self.points = [] # Anchor points
        self.nIteration = Iterate
        self.results = [] # Result points
        self.proses = []

    def add(self, point : Point):
        """
        Add a point into the anchor points array
        """
        self.points.append(point)

    def addSol(self, point : Point):
        """
        Add a point into the result array
        """
        self.results.append(point)

    def clearResults(self):
        """
        Delete result points
        """
        self.results = []
```

Gambar 5.1. Source Code Program Pembuatan Kurva Bezier

```
def createCurve(self, brute : bool):
    """
    Bezier curve generation algorithm
    """
    print(f"Depth : {self.nIteration}")
    if brute:
        # Using the brute force approach
        start = time.time()
        print(">> Using the BRUTEFORCE approach <<")
        increment = 1 / (2**self.nIteration)
        # n depth increment
        t = float(0)
        while(t <= 1):
            temp = Point.Point(0,0)
            n = len(self.points) - 1
            for i in range(len(self.points)):
                # Bezier sum formula
                temp += mt.comb(n,i)*((1-t)**(n-i))*(t**i)*self.points[i]
            t += increment
            self.addSol(temp)
            # bezier point addition to the result array
        end = time.time()
        print(">> BRUTEFORCE Runtime : ", end="")
        print((end - start) * 1000, "ms")
    else:
        # Using the divide and conquer approach
        start = time.time()
        print(">> Using the DIVIDE AND CONQUER approach <<")
        self.addSol(self.points[0])
        # Calling the recursive function
        self.createCurveDnc(self.nIteration, self.points)
        self.addSol(self.points[len(self.points)-1])
        end = time.time()
        print(">> DIVIDE AND CONQUER Runtime : ", end="")
        print((end - start) * 1000, "ms")
        print(">> Displaying Curve... <<")
        return end - start
```

Gambar 5.2. Source Code Program Pembuatan Kurva Bezier

```

def findMidpoint(self, list_of_points, anchor) :
    """
    To return a midpoint between 2 points
    """
    if (len(list_of_points) == 1) :
        return list_of_points[0]
    else :
        midpoints = []
        for i in range(len(list_of_points) - 1) :
            middle = (list_of_points[0+i] + list_of_points[1+i])/2
            midpoints.append(middle)
        if (len(midpoints) > 1) :
            anchor.append(midpoints[0])
            anchor.append(midpoints[-1])
        return self.findMidpoint(midpoints, anchor)

```

Gambar 5.3. *Source Code* Program Pembuatan Kurva Bezier

```

def printPoints(self):
    """
    To print each anchor point coordinate in the bezier curve
    """
    for x in self.points:
        x.printCoordinate()

def printResult(self):
    """
    To print each result point coordinate in the bezier curve
    """
    for x in self.results:
        x.printCoordinate()

def displayCurve(self, show):
    """
    Bezier curve render
    """
    plt.title("Bezier Curve")
    Display.plotDotLine(self.points)
    Display.plotLine(self.points)
    if (show) :
        Display.plotLineArray(self.proses)
    Display.plotLine(self.results)
    plt.show()

```

Gambar 5.4. *Source Code* Program Logging Kurva Bezier



```

def plotDot(x, y) :
    """
    plot a point
    """
    plt.plot(x, y, 'ro')

def plotDotLine(list_of_dot) :
    """
    plot multiple points from a list of point
    """
    for i in range(len(list_of_dot)) :
        plotDot(list_of_dot[i].x, list_of_dot[i].y)

def plotLine(list_of_dot) :
    """
    plot a line from a list of point
    """
    x = []
    y = []
    for i in range(len(list_of_dot)) :
        x.append(list_of_dot[i].x)
        y.append(list_of_dot[i].y)
    plt.plot(x, y)

def plotLineArray(list_of_line) :
    """
    plot multiple lines with its points from a list of line
    """
    for i in range(len(list_of_line)) :
        # plotDotLine(list_of_line[i])
        plotLine(list_of_line[i])

```

Gambar 5.5. *Source Code* Program Pencetakan Kurva Bezier

```

try :
    print("\n      || Welcome to the Bezier Curve Generator ||")
    print("|| Developed by : Zaki Yudhistira and Vanson Kurnialim ||\n")
    print("-----\n")
    # Welcome message

    flag = input("Do you want to use a file input (Y/N) ? : ").lower()

    if (flag == "y"):
        print(">> Using file input <<")
        points_temp = Rf.readFile()
        iterations = int(input("Input the number of iterations / depth :
"))
        if (iterations < 1) :
            # n iteration inputs / depth
            print("Invalid input, terminating program...")
            exit()
            # error handling, to be caught by exception
        main = Bc.BezierCurve(iterations)
        main.points = points_temp

    elif (flag == "n"):
        iterations = int(input("Input the number of iterations / depth :
"))
        if (iterations < 1) :
            # n iteration inputs / depth
            print("Invalid input, terminating program...")
            exit()
        main = Bc.BezierCurve(iterations)

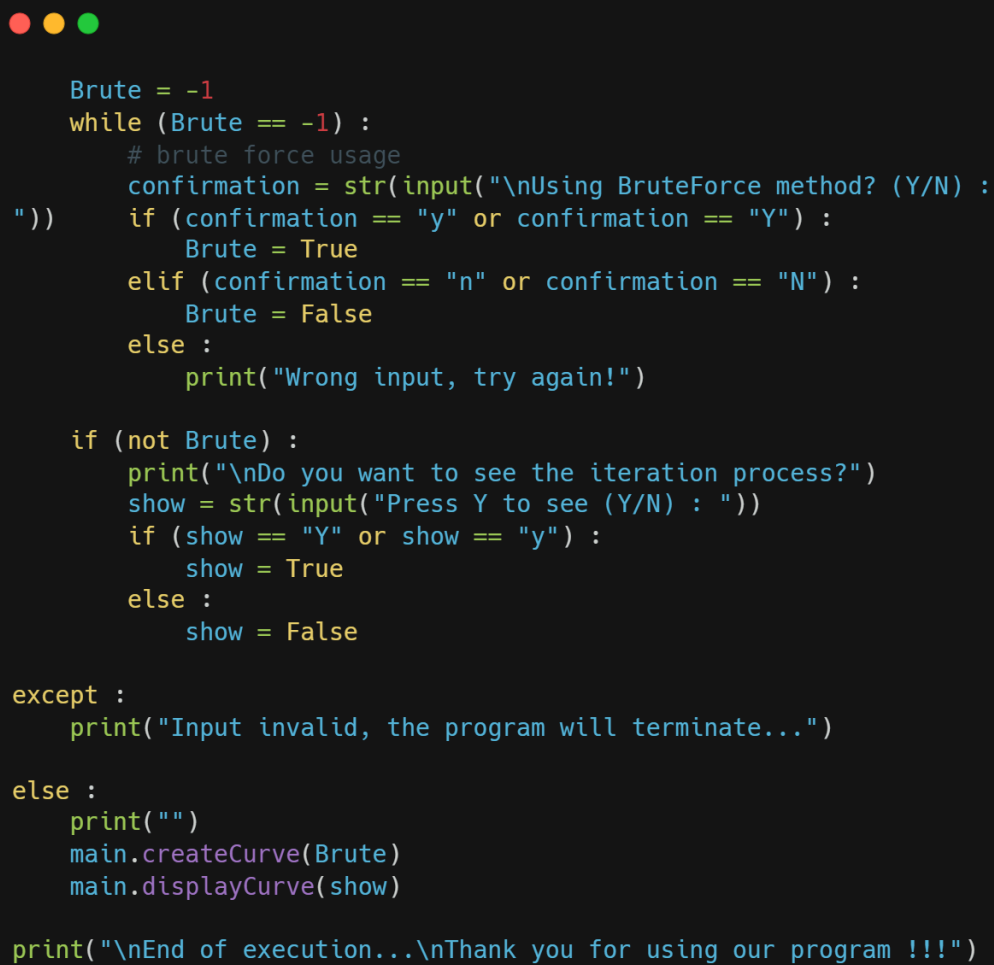
    n = int(input("Input the number of points : "))
    if (iterations < 1) :
        print("Invalid input, terminating program...")
        exit()

    for i in range(n) :
        # points input
        print(f"Point {i+1}")
        x = float(input("Enter x coordinate : "))
        y = float(input("Enter y coordinate : "))
        main.add(Point.Point(x,y))

    main.printPoints()

```

Gambar 5.6. *Source Code Utama Program*



```

Brute = -1
while (Brute == -1) :
    # brute force usage
    confirmation = str(input("\nUsing BruteForce method? (Y/N) :
"))
    if (confirmation == "y" or confirmation == "Y") :
        Brute = True
    elif (confirmation == "n" or confirmation == "N") :
        Brute = False
    else :
        print("Wrong input, try again!")

if (not Brute) :
    print("\nDo you want to see the iteration process?")
    show = str(input("Press Y to see (Y/N) : "))
    if (show == "Y" or show == "y") :
        show = True
    else :
        show = False

except :
    print("Input invalid, the program will terminate...")

else :
    print("")
    main.createCurve(Brute)
    main.displayCurve(show)

print("\nEnd of execution...\nThank you for using our program !!!")

```

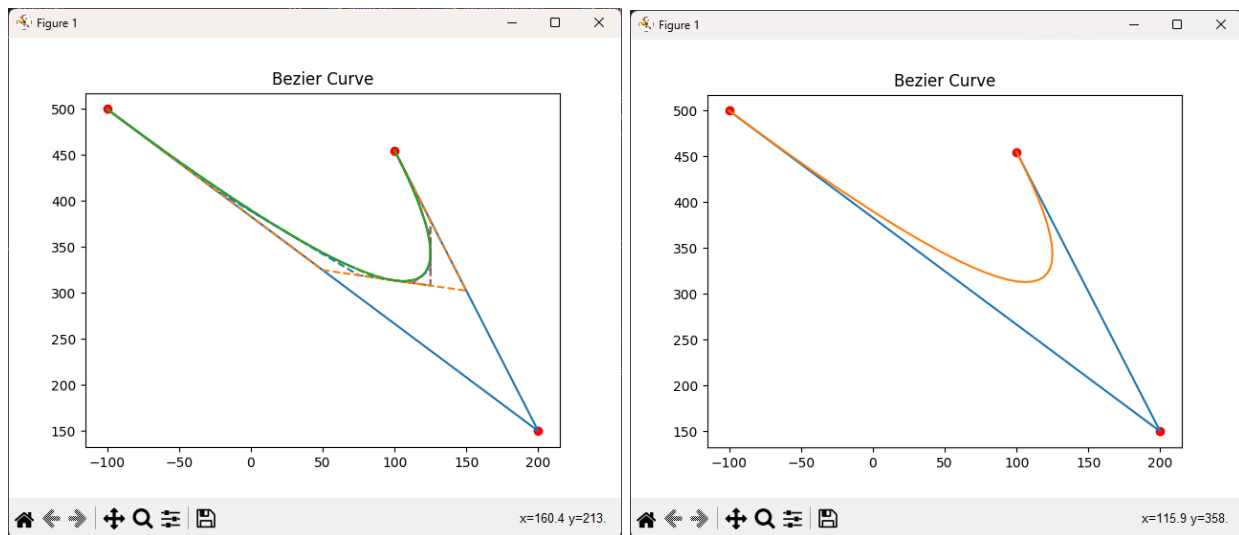
Gambar 5.7. *Source Code* Program Pembuatan Kurva Bezier

## F. Pengujian

Pengujian akan dilakukan dengan menginput beberapa titik dan menggunakan pendekatan *BRUTE FORCE* dan *DIVIDE AND CONQUER*, kemudian membandingkan *runtime* keduanya. Akan pengujian sebanyak 6 kali. *File* pengujian dapat dilihat pada *folder* “test” pada program.

### 1. Pengujian-1

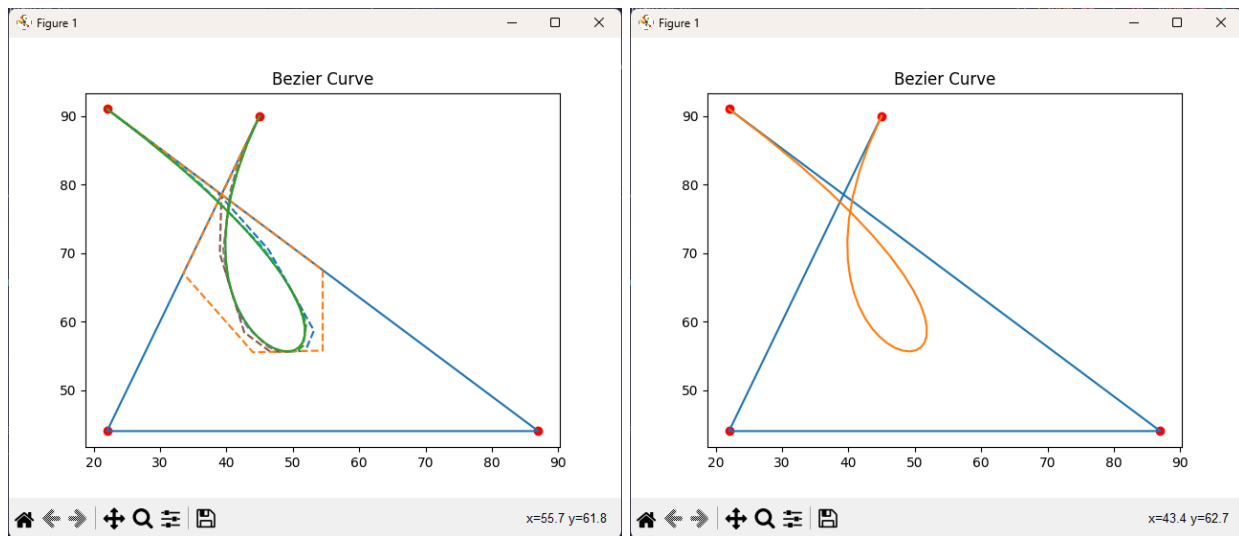
- a. Jumlah Titik : 3
- b. Jumlah Iterasi / *depth* : 5
- c. Koordinat Titik :
  - i. 100, 455
  - ii. 200, 150
  - iii. -100, 500
- d. Waktu eksekusi :
  - i. *Brute Force* : 0.0 ms
  - ii. *Divide and Conquer* : 1 ms



Gambar 6.1. Hasil Kurva Bezier Pengujian-1, Kiri *DIVIDE AND CONQUER* beserta tahapannya, kanan *BRUTE FORCE*

## 2. Pengujian-2

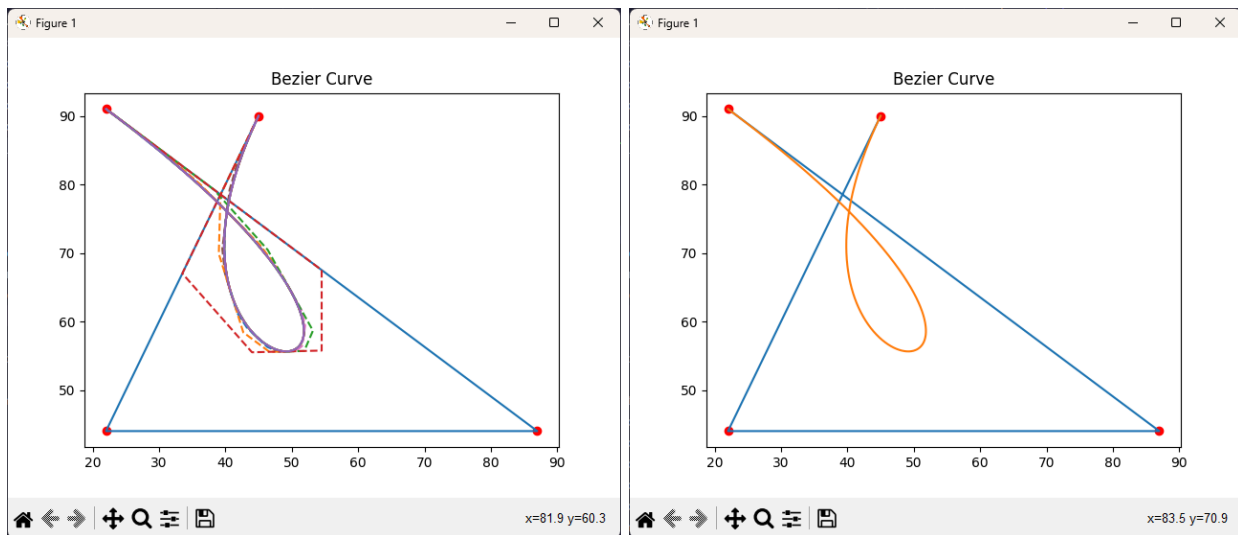
- a. Jumlah Titik : 4
- b. Jumlah Iterasi : 5
- c. Koordinat Titik :
  - i. 45, 90
  - ii. 22, 44
  - iii. 87, 44
  - iv. 22, 91
- d. Waktu Eksekusi:
  - i. *Brute Force* : 0.0 ms
  - ii. *Divide and Conquer* : 0.0 ms



Gambar 6.2. Hasil Kurva Bezier Pengujian-2, Kiri *DIVIDE AND CONQUER* beserta tahapannya, kanan *BRUTE FORCE*

### 3. Pengujian-3

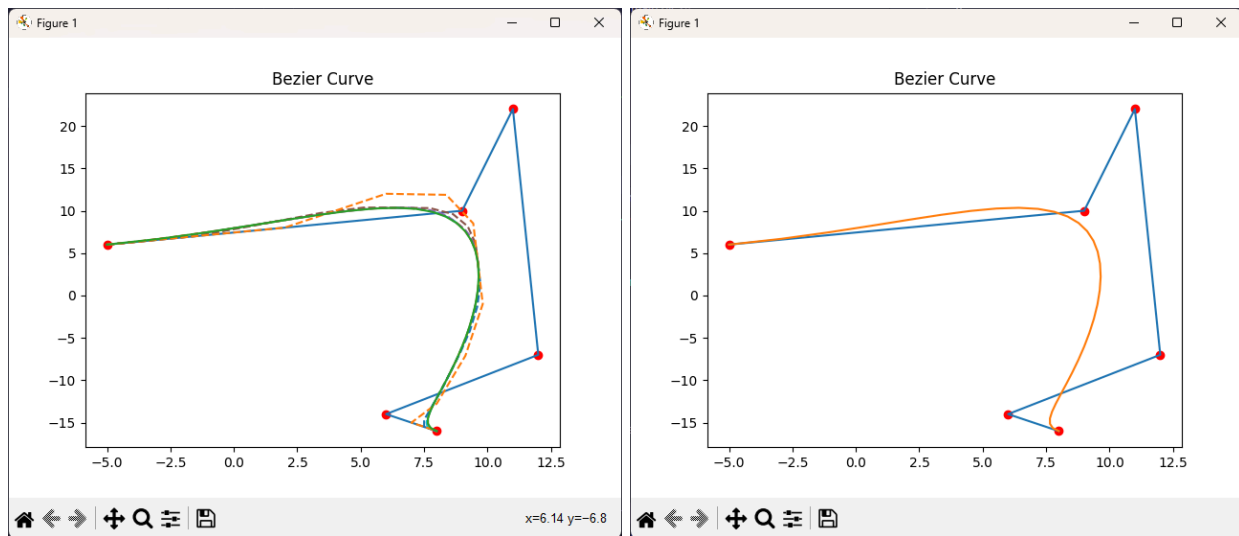
- a. Jumlah Titik : 4
- b. Jumlah Iterasi : 10
- c. Koordinat Titik :
  - i. 45, 90
  - ii. 22, 44
  - iii. 87, 44
  - iv. 22, 91
- d. Waktu Eksekusi:
  - i. *Brute Force* : 4.01 ms
  - ii. *Divide and Conquer* : 6.01 ms



Gambar 6.3. Hasil Kurva Bezier Pengujian-3, Kiri *DIVIDE AND CONQUER* beserta tahapannya, kanan *BRUTE FORCE*

#### 4. Pengujian-4

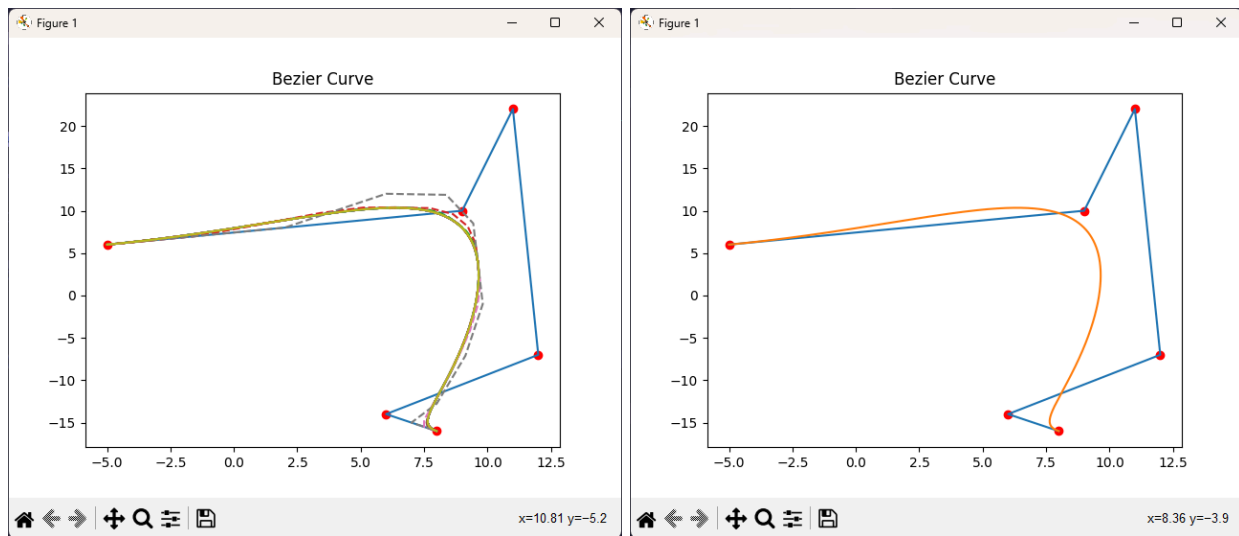
- a. Jumlah Titik : 6
- b. Jumlah Iterasi : 5
- c. Koordinat Titik :
  - i. -5, 6
  - ii. 9, 10
  - iii. 11, 22
  - iv. 12, -7
  - v. 6, -14
  - vi. 8, -16
- d. Waktu Eksekusi:
  - i. *Brute Force* : 1.3 ms
  - ii. *Divide and Conquer* : 0.0 ms



Gambar 6.4. Hasil Kurva Bezier Pengujian-4, Kiri *DIVIDE AND CONQUER* beserta tahapannya, kanan *BRUTE FORCE*

## 5. Pengujian-5

- a. Jumlah Titik : 6
- b. Jumlah Iterasi : 15
- c. Koordinat Titik :
  - i. -5, 6
  - ii. 9, 10
  - iii. 11, 22
  - iv. 12, -7
  - v. 6, -14
  - vi. 8, -16
- d. Waktu Eksekusi:
  - i. *Brute Force* : 216 ms
  - ii. *Divide and Conquer* : 535 ms



Gambar 6.5. Hasil Kurva Bezier Pengujian-5, Kiri *DIVIDE AND CONQUER* beserta tahapannya, kanan *BRUTE FORCE*



## 6. Pengujian-6

a. Jumlah Titik : 9

b. Jumlah Iterasi : 8

c. Koordinat Titik :

i. 1, -6

ii. 2, 10

iii. 3, 14

iv. 4, -2

v. 5, -7

vi. 6, 11

vii. 7, 11

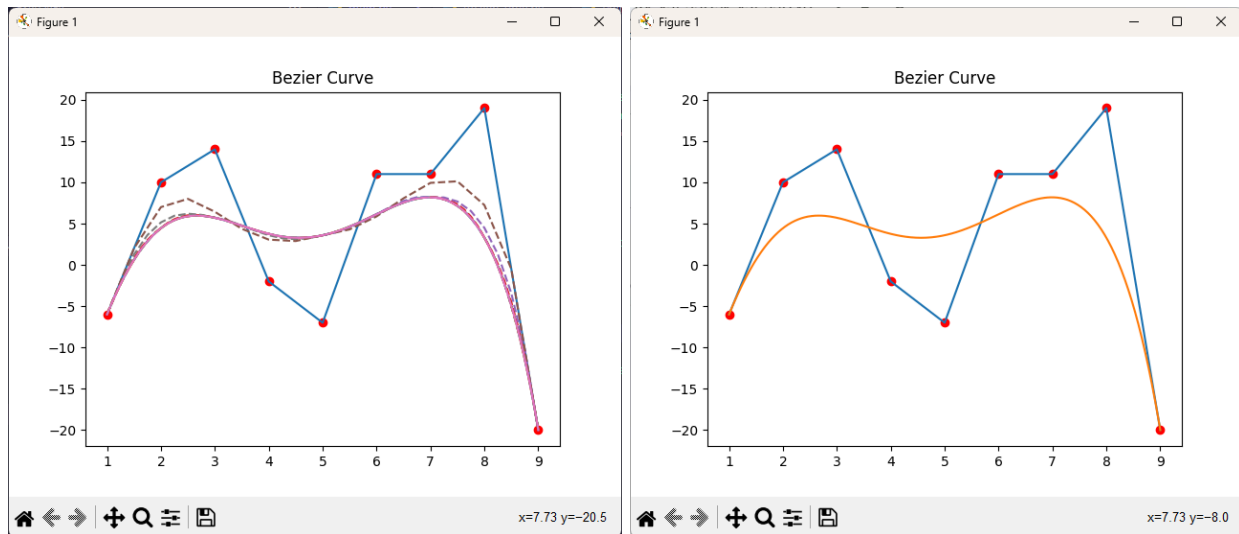
viii. 8, 19

ix. 9, -20

d. Waktu Eksekusi:

i. *Brute Force* : 3 ms

ii. *Divide and Conquer* : 7 ms



Gambar 6.6. Hasil Kurva Bezier Pengujian-6, Kiri *DIVIDE AND CONQUER* beserta tahapannya, kanan *BRUTE FORCE*

## G. Analisis Pendekatan

Dapat dilihat dari data-data pengujian bahwa pendekatan *BRUTE FORCE* memiliki kecepatan *runtime* yang lebih cepat daripada *DIVIDE AND CONQUER*, bahkan ketika iterasi / *depth* tinggi pun. Hal ini tentu di luar hipotesis kami yang menyatakan bahwa algoritma dengan pendekatan *DIVIDE AND CONQUER* seharusnya akan lebih sangkil daripada pendekatan *BRUTE FORCE*.

Menurut kami, faktor utama yang menyebabkan hal ini ialah, penyimpanan proses pembuatan kurva bezier yang merupakan fitur visualisasi tahapan pembentukan kurva. Penyimpanan kurva dapat meningkatkan penggunaan memori dan secara umum mengurangi kecepatan eksekusi berhubung program harus menangani prosedur tambahan yakni penyimpanan tahapan pada suatu larik.

## H. Kesimpulan

Kurva bezier dapat dibentuk menggunakan berbagai macam metode, pada kasus ini, kami berhasil mengimplementasikan metode pembuatan kurva bezier menggunakan pendekatan *BRUTE FORCE* dan *DIVIDE AND CONQUER*. Juga pada kasus ini, pembuatan kurva bezier dengan pendekatan *BRUTE FORCE* terbukti lebih cepat karena terdapat variabel yang tidak imbang, seperti yang telah dijelaskan pada bab sebelumnya.

Untuk kedepannya, salah satu peningkatan yang dapat kami lakukan adalah menjaga variabel yang sama antara kedua metode sehingga dapat dilakukan perbandingan yang adil.

## I. Lampiran

Tautan *repository github* : [https://github.com/ZakiYudhistira/Tucil2\\_13522031\\_13522049](https://github.com/ZakiYudhistira/Tucil2_13522031_13522049)

**Tabel Spesifikasi Program**

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