LAPORAN TUGAS KECIL 2

IF2211 - STRATEGI ALGORITMA

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

Dosen:

Ir. Rila Mandala, M.Eng., Ph.D.

Monterico Adrian, S.T., M.T.

K-03:

Muhammad Fatihul Irhab (13522143)

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2023/2024

BAB I
LATAR BELAKANG MASALAH

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 P_0 dan P_1 yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan kurva Bézier linier. Misalkan terdapat sebuah titik Q_0 yang berada pada garis yang dibentuk oleh P_0 dan P_1, maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh $B(t)$ dari P_0 ke P_1. Misalnya ketika t = 0.25, maka $B(t)$ adalah seperempat jalan dari titik P_0 ke P_1. sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan $B(t)$ membentuk sebuah garis lurus dari P_0 ke P_1.

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P_2, dengan P_0 dan P_2 sebagai titik kontrol awal dan akhir, dan P_1 menjadi titik kontrol antara. Dengan menyatakan titik Q_1 terletak diantara garis yang menghubungkan P_1 dan P_2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q_0 berada, maka dapat dinyatakan sebuah titik baru, R_0 yang berada diantara garis yang menghubungkan Q_0 dan Q_1 yang bergerak membentuk kurva Bézier kuadratik terhadap titik P_0 dan P_2. Berikut adalah uraian persamaannya.

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

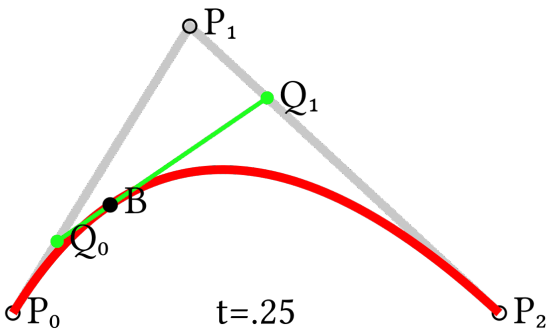
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

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

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

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

Berikut adalah ilustrasi dari kasus diatas.



BAB II
DESKRIPSI MASALAH DAN ALGORITMA

1.1 Algoritma Divide and Conquer

Algoritma divide and conquer adalah pendekatan yang digunakan dalam memecahkan masalah dengan dua elemen utama, yaitu Divide dan Conquer. Pendekatan ini membagi masalah besar menjadi beberapa sub-masalah yang serupa namun lebih kecil. Idealnya, sub-masalah ini memiliki ukuran yang sama. Setelah membagi masalah menjadi sub-masalah, langkah selanjutnya adalah menyelesaikan masing-masing sub-masalah (conquer) secara langsung atau dengan pendekatan rekursif jika sub-masalahnya masih besar. Terakhir, solusi dari masing-masing sub-masalah digabungkan untuk membentuk solusi akhir dari masalah utama. Karena setiap sub-masalah memiliki sifat yang sama, algoritma divide and conquer sering dinyatakan secara rekursif dan diimplementasikan menggunakan fungsi rekursif.

1.2 Pencarian Titik Tengah dari Beberapa Titik

Persoalan pencarian titik tengah dari beberapa titik adalah sebuah masalah analisis data dan geometrik dalam komputasi ketika mencari titik tengah dari beberapa titik yang ada. Secara sederhana, persoalan ini dapat diartikan sebagai proses mencari titik tengah dari sekumpulan titik yang diberikan.

Persoalan pencarian titik tengah dari beberapa titik dapat dirumuskan dalam sebuah bidang, bisa dalam bidang datar, bidang ruang, ataupun dalam ruang vektor R^n. Contoh dari persoalan ini adalah n buah titik pada ruang 2D, dimana setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x,y), dari sana, kita akan mencari titik tengah dari sekumpulan titik. Kita dapat mengetahui titik tengah dari dua buah titik P1 = (x1,y1) dan P2 = (x2,y2) dengan menggunakan rumus P = (P1 + P2)/2. Dilakukan terus menerus hingga kita mendapatkan titik tengah dari sekumpulan titik.

Persoalan ini dapat diterapkan dalam berbagai aplikasi, seperti dalam pemodelan data, optimasi dan pemrograman linier, geometri dan grafika komputer, pemodelan dan analisis fisika, dan klasifikasi pengelompokan data. Contoh penggunaannya adalah dalam geometri, kita dapat menggambar *bezier curve* dari beberapa titik kontrol dengan menggunakan algoritma *divide and conquer,* algoritma ini membutuhkan pencarian titik tengah dari beberapa titik untuk menghasilkan solusi.

1.3 Menggabungkan Beberapa Titik untuk Membentuk Kurva

Persoalan menggabungkan beberapa titik untuk membentuk kurva adalah sebuah persoalan dalam bidang matematika dan pemrograman yang melibatkan pengaturan titik-titik data dalam suatu pola atau kurva yang memvisualisasikan hubungan dalam data tersebut. Pada dasarnya, persoalan ini mencakup langkah-langkah untuk menghubungkan titik-titik data yang diberikan, baik dalam ruang 2D maupun 3D, sehingga membentuk pola atau garis lengkung yang merepresentasikan pola atau hubungan di antara titik-titik

tersebut. Dalam konteks analisis data, penggabungan titik-titik ini dapat membantu dalam pemahaman tren, pola, atau distribusi data yang ada, sementara dalam konteks pemrograman grafis, penggabungan titik-titik ini penting untuk menggambar kurva atau bentuk geometris lainnya yang sesuai dengan data yang dimiliki. Dengan demikian, menggabungkan beberapa titik untuk membentuk kurva merupakan salah satu langkah kunci dalam analisis dan visualisasi data yang membutuhkan pemahaman matematis dan penggunaan algoritma yang tepat.

1.4 Algoritma Penyelesaian Membangun Kurva Bezier dengan Titik Tengah Berbasis *Divide and Conquer*

Dalam menyelesaikan permasalahan membangun kurva Bezier secara algoritmik, digunakan pendekatan divide and conquer. Berikut adalah langkah-langkah penyelesaian permasalahan dalam algoritma:

1. Program akan meminta pengguna untuk memasukkan satu buah integer n (jumlah titik *control*). Kemudian program akan meminta n buah integer dengan dipisahkan spasi ke dalam arrayX (titik-titik koordinat x sebanyak n buah). Kemudian program akan meminta n buah integer dengan dipisahkan spasi ke dalam arrayY (titik-titik koordinat Y sebanyak n buah). Kemudian program akan meminta satu buah integer t (jumlah iterasi). Kemudian program akan meminta *button* mana yang ingin dipilih, *button* terdiri dari dua yaitu *divide and conquer* (*bezier curve* akan dibangun dengan *divide and conquer)* dan (*bezier curve* akan dibangun dengan *brute force)*.
2. Program akan mengecek seluruh masukkan dari pengguna, jika terdapat masukkan yang kurang sesuai maka program akan menampilkan pesan error. Jika masukkan sudah tepat maka program akan memilih *function* yang akan digunakan, sesuai dengan pilihan pengguna.
3. Jika pilihan pengguna merupakan *divide and conquer* maka program akan menjalankan *function getPointToDrawDivideAndConquer()*.
4. *Function getPointToDrawDivideAndConquer()* akan mengecek apakah parameter t (jumlah iterasi) sama dengan satu atau tidak. Jika t sama dengan satu, maka program akan memanggil *function getMiddlePoint(),* kemudian *return* dari *getMiddlePoint()* akan digabungkan dengan koordinat awal dan koordinat akhir, kemudian hasil ini menjadi *return* dari *function.* Jika t tidak sama dengan satu, maka program akan memanggil *function getMiddlePoint()*, kemudian *return* dari *getMiddlePoint()* akan digabungkan dengan koordinat awal, kemudian memanggil *function getPointToDrawDivideAndConquer()* dengan parameter t (jumlah iterasi) berkurang dan disimpan dalam array koordinat kiri, kemudian *return* dari *getMiddlePoint()* akan digabungkan dengan koordinat akhir, kemudian memanggil *function getPointToDrawDivideAndConquer()* dengan parameter t (jumlah iterasi) berkurang dan disimpan dalam array koordinat kanan, kemudian array koordinat kiri dan kanan digabung menjadi satu untuk *return* dari *function.*

5. *Function getMiddlePoint(),* akan terus memeriksa sekumpulan titik yang diberikan hingga menemukan titik tengahnya. Proses ini melibatkan perhitungan rata-rata koordinat x dan y dari seluruh titik untuk menentukan titik tengah akhir, yang kemudian dapat digunakan dalam pembuatan kurva Bezier atau analisis data lainnya.
6. Hasil array koordinat yang dihasilkan oleh *function getPointToDrawDivideAndConquer()* akan diplot menjadi kurva Bezier. Proses ini melibatkan penggunaan koordinat titik-titik yang ditemukan oleh algoritma divide and conquer untuk membentuk titik-titik kontrol dari kurva Bezier. Kurva Bezier yang dihasilkan dari proses ini akan mencerminkan pola atau bentuk yang diinginkan berdasarkan koordinat titik-titik kontrol yang telah ditemukan sebelumnya.

BAB III

IMPLEMENTASI ALGORITMA

Dalam pembuatan program ini, penulis menggunakan bahasa pemrograman Python. Struktur dari program ini terbagi menjadi 3 file, yaitu ***bezierCurve.py***, ***bruteForce.py***, dan ***dnc.py***.

3.1 File bezierCurve.py

File ini merupakan *driver* utama dari program ini, sehingga program ini hanya berisi menu utama untuk menampilkan GUI, meminta masukkan pengguna, dan deklarasi variabel yang digunakan.

Variable Name	***Description***
int inputN	*Variable input* banyak titik kontrol dari GUI
array inputX	*Variable input* koordinat titik x
array inputY	*Variable input* koordinat titik y
int inputT	*Variable input* banyak iterasi
int jumlahTitik	*Variable* banyak titik kontrol
int jumlahIterasi	*Variable* banyak iterasi
array arrayTitikX	*Variable* koordinat titik x
array arrayTitikY	*Variable* koordinat titik y
array arrayTitikBezierX	*Variable* koordinat titik bezier x
array arrayTitikBezierY	*Variable* koordinat titik bezier y
float executionTime	*Variable* waktu eksekusi
int choice	*Variable* pilihan algoritma (*divide and conquer* atau *brute force*)

Methods	***Description***
void button(choice)	Menerima *choice* sebagai pilihan algoritma yang akan digunakan, menghitung waktu eksekusi program, memanggil algoritma untuk membentuk kurva bezier (*getPointToDrawDivideAndConquer()*

	atau *getPointToDrawBruteForce()*), dan memanggil *function drawCurve()*.
void drawCurve(arrayTitikX, arrayTitikY, arrayTitikBezierX, arrayTitikBezierY, choiche)	Menerima arrayTitikX, arrayTitikY, sebagai array koordinat titik kontrol, arrayTitikBezierX, arrayTitikBezierY, sebagai array koordinat titik kurva bezier, choice sebagai pilihan algoritma. Kemudian fungsi ini akan menggambar koordinat titik kontrol dan kurva bezier.

3.2 File dnc.py

File ini berisi fungsi-fungsi untuk mencari solusi *bezier curve* dengan menggunakan algoritma *divide and conquer*.

Methods	***Description***
array getMiddlePoint(arrayTitikX,arrayTitikY,jumlahIterasi)	Menerima arrayTitikX, arrayTitikY, sebagai array koordinat titik kontrol, jumlahIterasi sebagai banyak iterasi yang akan dilakukan. *Function* ini akan melakukan perulangan untuk mendapatkan sekumpulan titik tengah, kemudian akan dilakukan rekursif dengan memanggil *function getMiddlePoint()* kembali, proses ini akan terjadi terus menerus hingga didapatkan satu titik tengah sebagai *return* dari *function* ini.
array getPointToDrawDivideAndConquer(arrayTitikX,arrayTitikY,jumlahIterasi)	Menerima arrayTitikX, arrayTitikY, sebagai array koordinat titik kontrol, jumlahIterasi sebagai banyak iterasi yang akan dilakukan. *Function* ini akan memanggil *function getMiddlePoint()*, kemudian hasil dari pemanggilan tersebut akan dibagi menjadi dua bagian (*divide*) yaitu kiri dan kanan, lalu masing-masing bagian tersebut (kiri dan kanan) akan melakukan rekursif dengan memanggil *function getPointToDrawDivideAndConquer()* kembali tetapi dengan parameter jumlahIterasi berkurang satu, proses ini akan dilakukan terus menerus hingga jumlahIterasi satu. Hasil kembalian dari masing-masing bagian (kiri dan kanan) akan disatukan (*conquer)*, kemudian hasil ini lah yang akan menjadi *return* dari *function* ini.

3.3 File bruteForce.py

File ini berisi fungsi-fungsi untuk mencari solusi *bezier curve* dengan menggunakan algoritma *brute force*.

Methods	*Description*
array getRumus(pangkat)	Menerima pangkat sebagai parameter rumus koefisien yang akan dihasilkan, fungsi ini akan mengembalikan array dari koefisien yang didapat dari rumus (x+y)^pangkat.
array getPointToDrawBruteForce(array TitikX,arrayTitikY,jumlahIterasi)	Menerima arrayTitikX, arrayTitikY, sebagai array koordinat titik kontrol, jumlahIterasi sebagai banyak iterasi yang akan dilakukan. *Function* ini akan memanggil *function getRumus()* untuk mendapatkan array koefisien. Fungsi ini akan melakukan loop sebanyak *range* t (sesuai iterasi) dan setiap loop nya akan melakukan loop lagi sebanyak jumlah koefisien yang didapat, kemudian dilakukan perhitungan titik dengan menggunakan rumus yang tertera pada bab 1. *Function* ini akan mengeluarkan *return* titik koordinat *bezier curve*.

3.4 Library

Beberapa pustaka atau library yang digunakan dalam program ini meliputi:

- math
- time
- numpy
- tkinter
- tkinter.messagebox
- matplotlib.figure
- matplotlib.backends.backend_tkagg

3.5 Analisis Algoritma

Algoritma *brute force dan divide and conquer* adalah dua pendekatan umum dalam menyelesaikan masalah algoritma. Dalam konteks implementasi kurva bezier, keduanya dapat digunakan untuk menghasilkan kurva yang serupa, namun dengan perbedaan dalam cara komputasinya. Kita dapat menganalisis kedua algoritma dengan membandingkan kompleksitas algoritma tersebut.

Kompleksitas waktu algoritma *brute force* dapat dianggap sebagai O(n^2), dimana n adalah jumlah iterasi yang akan dilakukan. Hal ini karena, pada setiap iterasi, algoritma perlu mengevaluasi polinomial bezier pada titik-titik tertentu untuk membangun kurva.

Kompleksitas waktu algoritma *divide and conquer* untuk implementasi kurva bezier ini dapat dianggap sebagai O(n^2), dimana n adalah jumlah iterasi yang akan dilakukan. Hal ini disebabkan karena kurva Bézier dipartisi menjadi sub-kurva yang semakin kecil secara eksponensial dengan setiap iterasi.

Kesimpulan, jika dilihat dari waktu kompleksitas algoritma keduanya sama, tetapi pada proses pengujiannya langsung, algoritma *brute force* lebih cepat daripada algoritma *divide and conquer*, hal ini disebabkan pada algoritma *divide and conquer* terjadi rekursif *double* (untuk kiri dan kanan). Oleh karena itu, untuk pembentukan kurva bezier lebih efektif menggunakan algoritma *brute force*

BAB IV
SOURCE CODE

4.1 Repository Program

Repository program dapat diakses melalui tautan GitHub berikut :

https://github.com/muhhul/Tucil2_13522143

4.2 Source Code Program

4.2.1 bezierCurve.py

bezierCurve.py > drawCurve

```python
import tkinter as tk
import tkinter.messagebox as messagebox
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import time
import dnc
import bruteForce

def drawCurve(arrayTitikX, arrayTitikY, arrayTitikBezierX, arrayTitikBezierY, choiche):
    global canvas, widget
    if widget is not None:
        widget.destroy()

    fig = Figure(figsize=(5, 5), dpi=100)
    ax = fig.add_subplot(111)
    if choiche == 0:
        arrayTitikBezierX = list(arrayTitikBezierX)
        arrayTitikBezierX.insert(0,arrayTitikX[0])
        arrayTitikBezierX.append(arrayTitikX[len(arrayTitikX)-1])
        arrayTitikBezierX = tuple(arrayTitikBezierX)
        arrayTitikBezierY = list(arrayTitikBezierY)
        arrayTitikBezierY.insert(0,arrayTitikY[0])
        arrayTitikBezierY.append(arrayTitikY[len(arrayTitikY)-1])
        arrayTitikBezierY = tuple(arrayTitikBezierY)

    ax.plot(arrayTitikX, arrayTitikY, marker='o', label="Titik kontrol")
    ax.plot(arrayTitikBezierX, arrayTitikBezierY, marker='o', label="Bezier curve")
    ax.legend()

    if choiche == 0:
        ax.set_title("Divide and Conquer")
    else:
        ax.set_title("Brute Force")

    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.draw()
    widget = canvas.get_tk_widget()
    widget.pack()

def button(choice):
    jumlahTitik = int(inputN.get())
    jumlahIterasi = int(inputT.get())
    arrayTitikX = [float(i) for i in inputX.get().split()]
    arrayTitikY = [float(i) for i in inputY.get().split()]
```

```
    if len(arrayTitikX) != jumlahTitik or len(arrayTitikY) != jumlahTitik:
        messagebox.showinfo("Warning","Maaf jumlah titik dan jumlah x atau y tidak sama.")
        return
    else:
        startTime = time.time()
        if choice == 0:
            arrayTitikBezierXY = dnc.getPointToDrawDivideAndConquer(arrayTitikX, arrayTitikY, jumlahIterasi)
        else:
            arrayTitikBezierXY = bruteForce.getPointToDrawBruteForce(arrayTitikX, arrayTitikY, jumlahIterasi)
        endTime = time.time()
        executionTime = endTime - startTime

        executionTimeLabel.config(text=f"Waktu eksekusi: {executionTime} detik")
        arrayTitikBezierX, arrayTitikBezierY = zip(*arrayTitikBezierXY)
        drawCurve(arrayTitikX, arrayTitikY, arrayTitikBezierX, arrayTitikBezierY, choice)

canvas = None
widget = None

root = tk.Tk()

textInputN = tk.Label(root, text="Masukkan jumlah titik: ")
textInputN.pack()
inputN = tk.Entry(root)
inputN.pack()

textInputX = tk.Label(root, text="Masukkan nilai x (pisahkan dengan spasi): ")
textInputX.pack()
inputX = tk.Entry(root)
inputX.pack()
```

```
textInputY = tk.Label(root, text="Masukkan nilai y (pisahkan dengan spasi): ")
textInputY.pack()
inputY = tk.Entry(root)
inputY.pack()

textInputT = tk.Label(root, text="Masukkan jumlah iterasi yang ada ingin kan: ")
textInputT.pack()
inputT = tk.Entry(root)
inputT.pack()

buttonFrame = tk.Frame(root)
buttonFrame.pack()
divideAndConquerButton = tk.Button(buttonFrame, text="Divide and Conquer", command=lambda: button(0))
divideAndConquerButton.grid(row=0, column=0)
bruteForceButton = tk.Button(buttonFrame, text="Brute Force", command=lambda: button(1))
bruteForceButton.grid(row=0, column=1)

executionTimeLabel = tk.Label(root)
executionTimeLabel.pack()

root.mainloop()
```

4.2.2 dnc.py

dnc.py > getPointToDrawDivideAndConquer

```python
def getMiddlePoint(arrayTitikX,arrayTitikY,jumlahIterasi):
    arrayMiddleX = []
    arrayMiddleY = []
    x = []
    y = []

    if len(arrayTitikX)==2:
        tempX = (arrayTitikX[0]+arrayTitikX[1])/2
        tempY = (arrayTitikY[0]+arrayTitikY[1])/2
        arrayMiddleX.append(tempX)
        arrayMiddleY.append(tempY)

        return zip(arrayMiddleX,arrayMiddleY)
    elif len(arrayTitikX)>2:
        for i in range(len(arrayTitikX)-1):
            tempX = (arrayTitikX[i]+arrayTitikX[i+1])/2
            tempY = (arrayTitikY[i]+arrayTitikY[i+1])/2
            arrayMiddleX.append(tempX)
            arrayMiddleY.append(tempY)
            if i == 0 or i == len(arrayTitikX)-2:
                x.append(tempX)
                y.append(tempY)
        arrayMiddleX, arrayMiddleY = zip(*getMiddlePoint(arrayMiddleX,arrayMiddleY,jumlahIterasi))
        x = x[:1] + list(arrayMiddleX) + x[1:]
        y = y[:1] + list(arrayMiddleY) + y[1:]

        return zip(x,y)

def getPointToDrawDivideAndConquer(arrayTitikX,arrayTitikY,jumlahIterasi):
    x = []
    y = []
    if jumlahIterasi > 1:
        arrayMiddleX, arrayMiddleY = zip(*getMiddlePoint(arrayTitikX,arrayTitikY,jumlahIterasi))
        jumlahIterasi = jumlahIterasi - 1

        arrayMiddleX = list(arrayMiddleX)
        arrayMiddleY = list(arrayMiddleY)
        arrayLeftX = []
        arrayLeftY = []
        newSizeArray = int((len(arrayMiddleX)+1)/2)
        for i in range(newSizeArray):
            arrayLeftX.append(arrayMiddleX[i])
            arrayLeftY.append(arrayMiddleY[i])

        arrayRightX = []
        arrayRightY = []
        for i in range(newSizeArray):
            arrayRightX.append(arrayMiddleX[i+int((len(x)/2))+int(len(arrayMiddleX)/2)])
            arrayRightY.append(arrayMiddleY[i+int((len(x)/2))+int(len(arrayMiddleX)/2)])

        arrayLeftX.insert(0,arrayTitikX[0])
        arrayLeftY.insert(0,arrayTitikY[0])
        arrayRightX.append(arrayTitikX[len(arrayTitikX)-1])
        arrayRightY.append(arrayTitikY[len(arrayTitikY)-1])
        arrayLeftX,arrayLeftY = zip(*getPointToDrawDivideAndConquer(arrayLeftX,arrayLeftY,jumlahIterasi))
        arrayRightX,arrayRightY = zip(*getPointToDrawDivideAndConquer(arrayRightX,arrayRightY,jumlahIterasi))

        x.extend(arrayLeftX)
        y.extend(arrayLeftY)
        x.append(arrayMiddleX[int(len(arrayMiddleX)/2)])
        y.append(arrayMiddleY[int(len(arrayMiddleY)/2)])
        x.extend(arrayRightX)
        y.extend(arrayRightY)
        return zip(x,y)
```

```
65
66      elif jumlahIterasi == 1 :
67          arrayMiddleX, arrayMiddleY = zip(*getMiddlePoint(arrayTitikX,arrayTitikY,jumlahIterasi))
68          x.append(arrayMiddleX[int(len(arrayMiddleX)/2)])
69          y.append(arrayMiddleY[int(len(arrayMiddleY)/2)])
70          return zip(x,y)
71
72      else:
73          arrayBezierX = []
74          arrayBezierY = []
75          arrayBezierX.append(arrayTitikX[0])
76          arrayBezierY.append(arrayTitikY[0])
77
78          arrayBezierX.append(arrayTitikX[len(arrayTitikX)-1])
79          arrayBezierY.append(arrayTitikY[len(arrayTitikY)-1])
80
81          return zip(arrayBezierX,arrayBezierY)
82
```

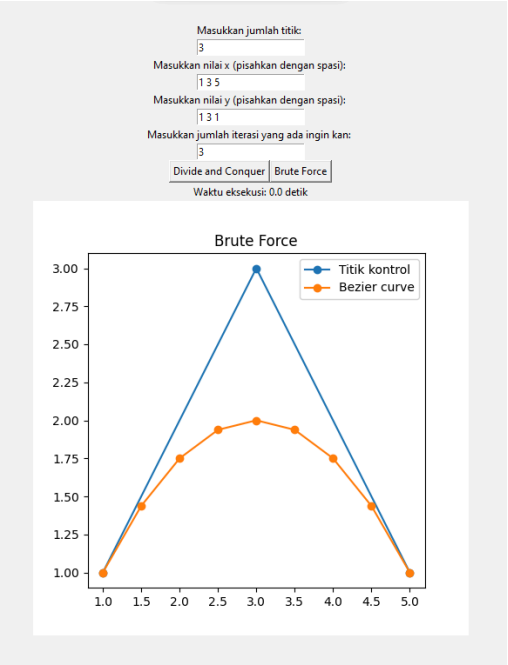
4.2.3 bruteForce.py

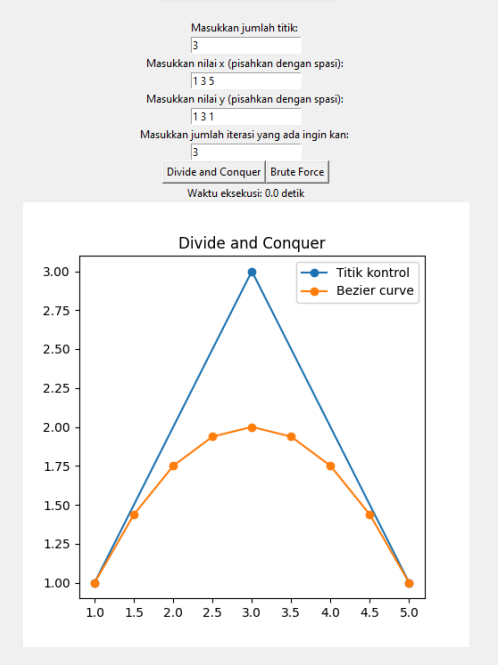
```
1   import numpy as np
2   import math
3
4   def getRumus(pangkat):
5       arrayKoefisien = []
6       for i in range(pangkat + 1):
7           koeefisien = math.comb(pangkat, i)
8           arrayKoefisien.append(koeefisien)
9
10      return arrayKoefisien
11
12  def getPointToDrawBruteForce(arrayTitikX,arrayTitikY,jumlahIterasi):
13      koefisien = getRumus(len(arrayTitikX)-1)
14      arrayTitikBezierX = []
15      arrayTitikBezierY = []
16      rangeValueT = np.linspace(0, 1, num=1+pow(2,jumlahIterasi))
17      for valueT in rangeValueT:
18          tempx = 0
19          tempy = 0
20          for j in range(len(arrayTitikX)):
21              tempx = tempx + pow((1-valueT),len(koefisien)-1-j)*pow(valueT,j)*arrayTitikX[j]*koefisien[j]
22              tempy = tempy + pow((1-valueT),len(koefisien)-1-j)*pow(valueT,j)*arrayTitikY[j]*koefisien[j]
23          arrayTitikBezierX.append(tempx)
24          arrayTitikBezierY.append(tempy)
25      return zip(arrayTitikBezierX,arrayTitikBezierY)
26
```

BAB V

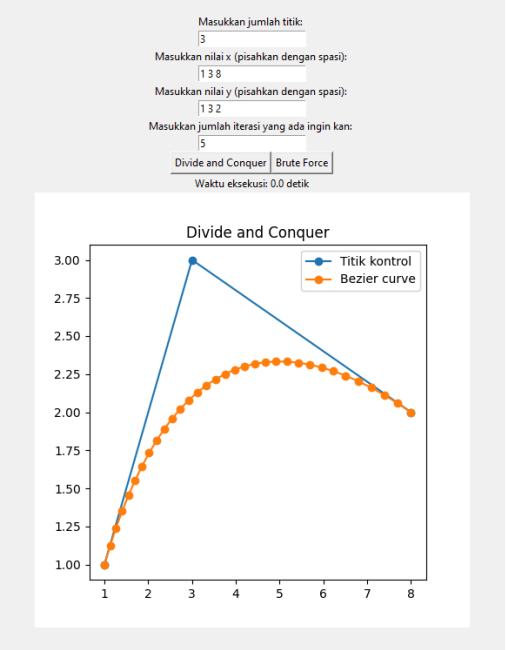
TESTING PROGRAM

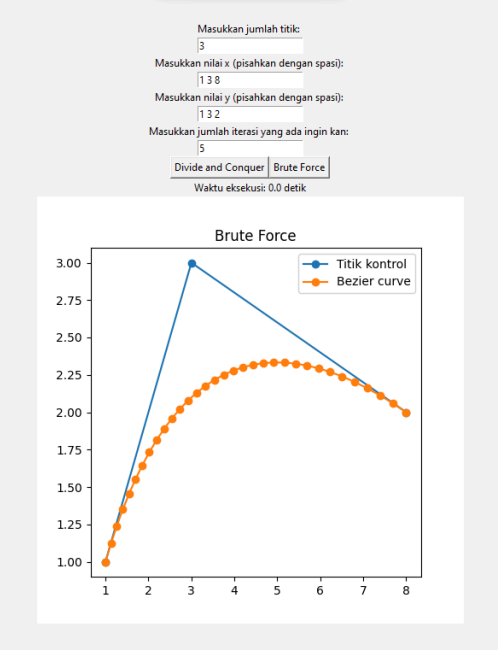
1. Percobaan 1



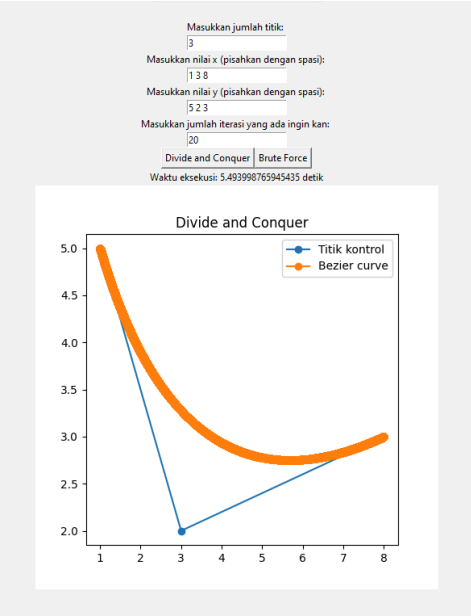


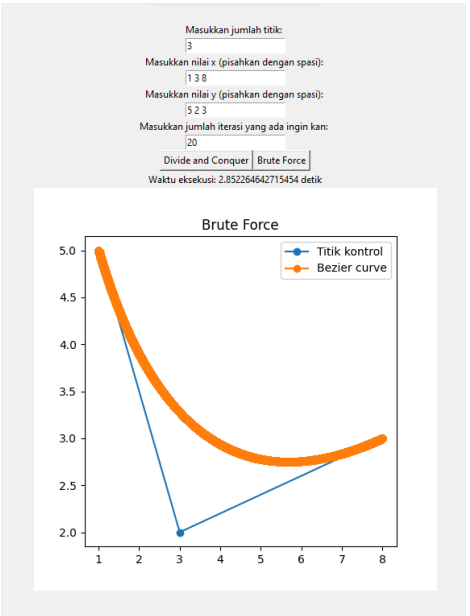
2. Percobaan 2



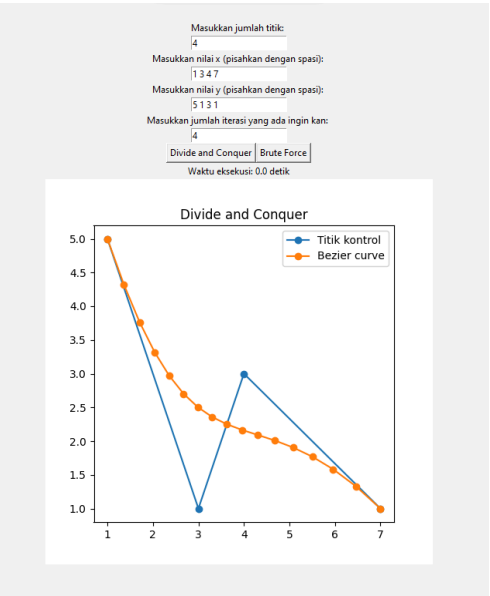


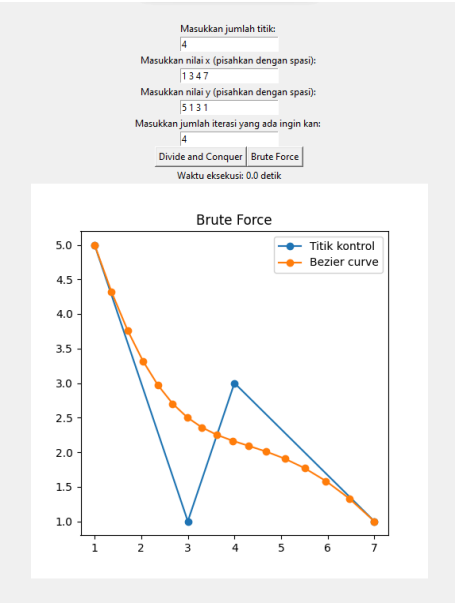
3. Percobaan 3



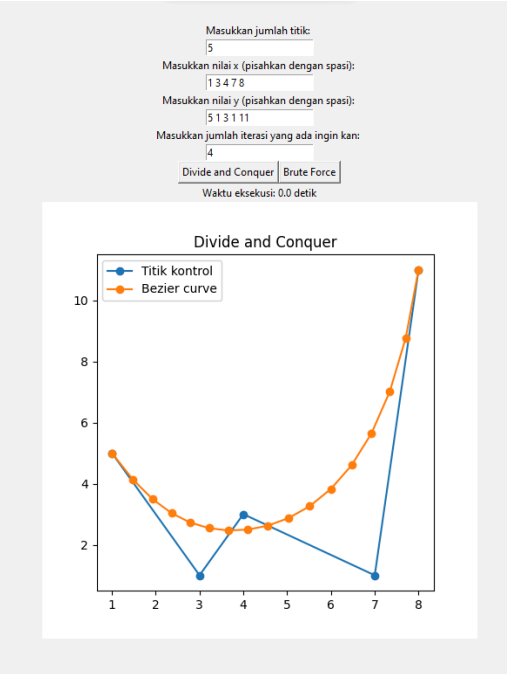


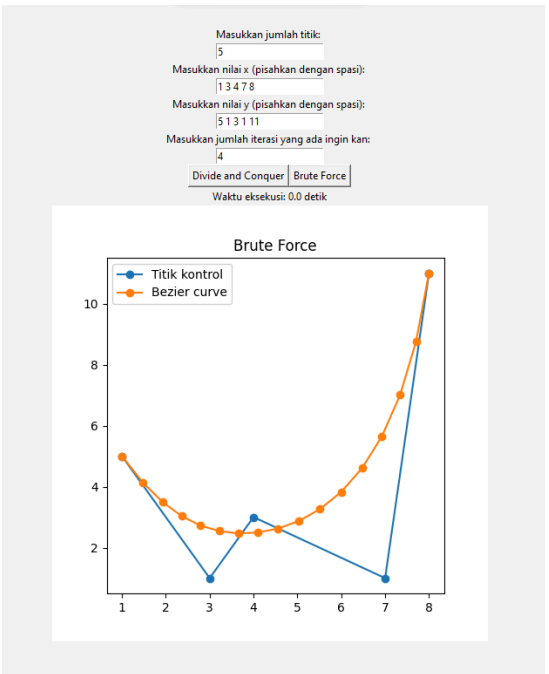
4. Percobaan 4



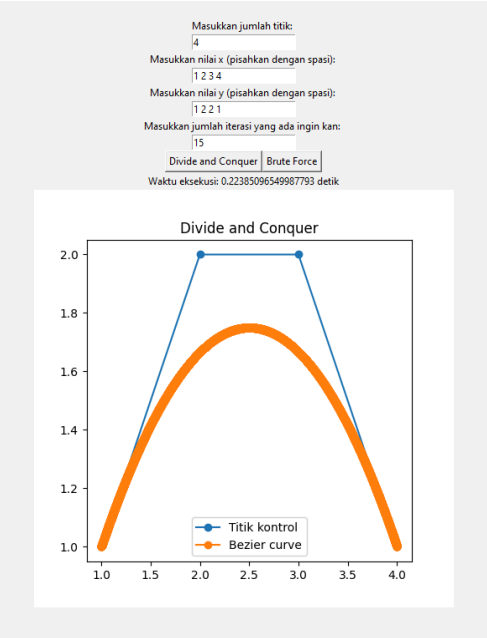


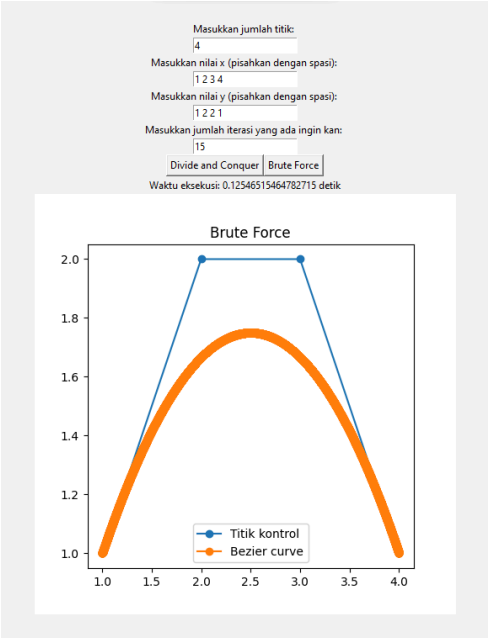
5. Percobaan 5





6. Percobaan 6





BAB VI

LAMPIRAN

1.

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