LAPORAN TUGAS KECIL 2 IF2211
STRATEGI ALGORITMA

MEMBANGUN KURVA BÉZIER DENGAN ALGORITMA TITIK TENGAH BERBASIS DIVIDE AND CONQUER

Disusun oleh:

Marzuli Suhada M - 13522070

Keanu Amadius Gonza Wrahatno - 13522082

PROGRAM STUDI TEKNIK INFORMATIKA

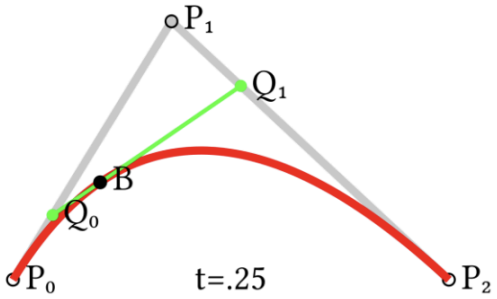
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023/2024

DAFTAR ISI

BAB I
DESKRIPSI TUGAS



Gambar 2. Pembentukan Kurva Bézier Kuadratik.

(Sumber: https://simonhalliday.com/2017/02/15/quadratic-bezier-curve-demo/)

Pada tugas ini, permasalahan utama yang akan dibahas adalah pembentukan kurva Kurva Bézier Kuadratik. Tujuan tugas ini adalah memvisualisasikan hasil kurva yang terbentuk dari titik-titik yang dicari dengan pendekatan **Algoritma Divide and Conquer**. Adapun dalam tugas ini juga akan dilakukan generalisasi algoritma tersebut sehingga dapat menyelesaikan permasalahan untuk pembentukan Kurva Bézier dari n titik . Pada tugas kecil ini, digunakan **algoritma brute force** sebagai tolak ukur yang akan dibandingkan secara kompleksitas dan validitas penyelesaian.

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. 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.

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

Tentu saja jika titik semakin banyak maka persamaan yang terbentuk akan sangat panjang dan akan semakin rumit. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka kami diminta untuk mengimplementasikan **pembuatan kurva Bézier** dengan algoritma titik tengah berbasis **divide and conquer**.

BAB II
LANDASAN TEORI

2.1 Algoritma Divide and Conquer

Algoritma Divide and Conquer merupakan salah satu algoritma dengan pendekatan rekursif yang memecah persoalan menjadi upapersoalan yang lebih kecil lalu menggabungkan solusinya kembali untuk mendapatkan solusi pada persoalan utama. Algoritma Divide and Conquer juga dapat didefinisikan dengan tiga tahapan yaitu *divide* (membagi persoalan), *conquer* (menyelesaikan upapersoalan), dan *combine* (menggabungkan solusi masing-masing upapersoalan).

Berikut merupakan skema umum dalam algoritma Divide and Conquer.

```
procedure DIVIDEandCONQUER(input P : problem, n : integer)
{ Menyelesaikan persoalan P dengan algoritma divide and conquer
  Masukan: masukan persoalan P berukuran n
  Luaran: solusi dari persoalan semula }

Deklarasi
   r : integer

Algoritma
   if n ≤ n0 then {ukuran persoalan P sudah cukup kecil }
      SOLVE persoalan P yang berukuran n ini
   else DIVIDE menjadi r upa-persoalan, P1 , P2 , …, Pr , yang masing-masing berukuran
n1 , n2 , …, nr
      for masing-masing P1 , P2 , …, Pr , do
          DIVIDEandCONQUER(Pi , ni )
      endfor
      COMBINE solusi dari P1 , P2 , …, Pr menjadi solusi persoalan semula
   endif
```

2.2 Pembangun Kurva Bézier

Pembentukan kurva Bézier menggunakan algoritma divide and conquer dilakukan dengan cara memecah persoalan yang kompleks menjadi upapersoalan yang lebih sederhana dan menggabungkan solusi masing-masing upapersoalan tersebut untuk membentuk solusi akhir. Berikut adalah langkah-langkah umum dari algoritmanya:

1. **Inisialisasi:** Anggaplah kita akan membentuk Kurva Bezier Kuadratik maka kita memiliki tiga titik kontrol yaitu titik awal P_0, titik kontrol P_1, dan titik akhir P_2.

2. **Iterasi awal:** Langkah ini melibatkan parameter iterasi yang menentukan seberapa halus atau kompleks kurva Bézier yang ingin dihasilkan. Misalnya, dengan satu parameter iterasi berarti kurva hanya dibagi menjadi dua segmen. Jumlah iterasi ini menentukan seberapa banyak pembagian yang akan dilakukan.

3. ***Divide*:** Kurva Bézier kuadratik dibagi menjadi dua segmen kurva Bézier yang lebih kecil. Pemisahan ini dilakukan dengan menentukan titik kontrol baru di tengah-tengah garis yang menghubungkan P_0 dan P_1, dan titik kontrol baru di tengah-tengah garis yang menghubungkan P_1 dan P_2.

4. **Rekursif:** Proses pembagian kurva diulangi secara rekursif sesuai dengan parameter iterasi yang telah ditentukan. Setiap segmen kurva Bézier yang dihasilkan kemudian dibagi lagi menjadi dua segmen baru, dan seterusnya, hingga mencapai jumlah iterasi yang diinginkan.

5. ***Combine***: Setelah mencapai titik di mana segmen-segmen yang sangat kecil atau cukup akurat telah dihasilkan sesuai dengan jumlah iterasi yang ditentukan, solusi submasalah ini digabungkan kembali secara rekursif. Ini dilakukan dengan menggabungkan titik akhir dari satu segmen dengan titik awal dari segmen berikutnya.

6. **Iterasi akhir:** Langkah ini diulangi sebanyak yang dibutuhkan sesuai dengan parameter iterasi yang telah ditentukan. Setelah semua segmen kurva Bézier telah digabungkan, kita akan memperoleh kurva Bézier kuadratik tunggal yang akhir.

Berikut merupakan skema umum dari algoritma pembentukan kurva bezier.

```
procedure Create_Bezier(input P : points, iterations : integer)
{ Menyelesaikan persoalan kurva bezier dengan algoritma divide and conquer
  Masukan: masukan persoalan berupa titik dan iterasi
  Luaran: titik-titik pembangun Kurva Bezier}

Deklarasi
   iterations : integer

Algoritma
   if iterations = 0 then
      return [P0, P2]
   else
      CONQUER dengan mencari titik tengah dari setiap control points
      Q0 = (P0 + P1) / 2
      Q1 = (P1 + P2) / 2
      R0 = (Q0 + Q1) / 2
```

```
    DIVIDE menjadi dua segmen yang lebih kecil
    leftCurve = Create_Bezier(P0, Q0, R0, iterations - 1)
    rightCurve = Create_Bezier(R0, Q1, P2, iterations - 1)

    COMBINE seluruh titik yang terbentuk dari segmen-segmen tersebut
    return concatenate(leftCurve, rightCurve)
  end if
end procedure
```

BAB III

IMPLEMENTASI

3.1 Implementasi dalam Algoritma Brute Force

Dalam menyelesaikan persoalan terkait pembentukan kurva bezier ini digunakan pendekatan dengan algoritma Brute Force. Algoritma ini memanfaatkan formula tertentu yang disebut sebagai polinomial Bernstein atau menggunakan segitiga Pascal untuk menghitung koefisien binomial yang diperlukan.

3.1.1 File BruteForce.py

File ini hanya terdiri dari satu fungsi saja yaitu `bezier_bf`. Fungsi ini digunakan khusus ketika jumlah titik kontrol yang dimasukkan berjumlah 3. Iterasi dilakukan pada nilai parameter t dari 0 hingga 1 dengan presisi yang diatur oleh variabel *iterations* yaitu sebesar $\frac{1}{2^{iterations}}$. Pada setiap iterasi, posisi kurva Bezier dihitung menggunakan rumus Bezier pada t. Setiap titik kurva yang dihitung lalu ditambahkan ke array `points_fix`. Output dari fungsi ini adalah mengembalikan daftar titik kurva bezier yang dihasilkan.

3.1.2 File BruteForceBonus.py

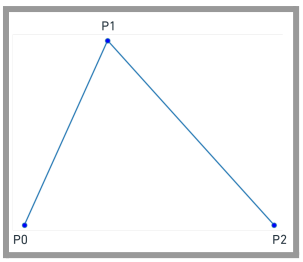
Konsep yang digunakan pada file ini kurang lebih sama seperti yang diterapkan pada file `BruteForce.py`. Hanya saja letak perbedaannya adalah, pada file ini, fungsi untuk mencari `points_fix` tidak langsung didefinisikan melainkan dibentuk terlebih dahulu dengan menggunakan formula segitigas pascal. Jadi, pertama-tama fungsi ini akan menghitung koefisien binomial terlebih dahulu menggunakan segitiga Pascal dengan memanggil fungsi `pascal_function()`. Kemudian, titik-titik kurva Bezier dihitung menggunakan koefisien binomial dan titik kontrol yang telah dimasukkan. Iterasi dilakukan pada nilai parameter t dari 0 hingga 1 dengan presisi yang diatur oleh variabel *iterations* yaitu sebesar $\frac{1}{2^{iterations}}$. Sama seperti sebelumnya setiap titik kurva yang dihitung ditambahkan ke array `points_fix`. Output dari fungsi ini adalah mengembalikan daftar titik kurva bezier yang dihasilkan.

3.2 Implementasi dalam Algoritma Divide and Conquer

Dalam menyelesaikan kurva bezier ini juga digunakan algoritma Divide and Conquer. Tentunya algoritma ini harus menerapkan bagian Divide, Conquer, dan Combine. Berikut ini merupakan penjelasan algoritmanya.

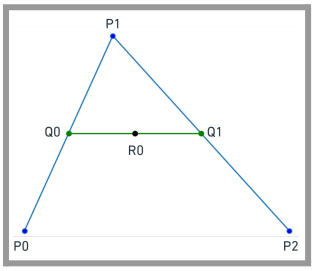
3.2.1 File DivideAndConquer.py

Langkah pertama yang dilakukan yaitu melakukan Conquer untuk mencari titik tengah dari 3 titik. Misal saja titik yang diberikan adalah titik P0, P1, dan P2.



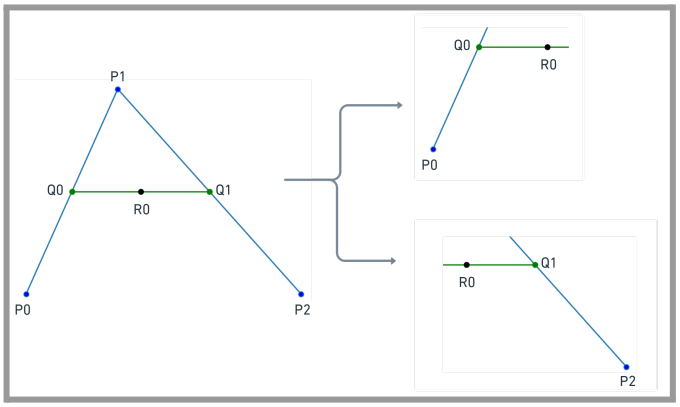
Gambar 3.2.1.1. Titik P0,P1,P2

Langkah Conquernya yaitu cari titik tengah dengan antara P0 dan P1 sebagai Q0 dan titik tengah P1 dan P2 sebagai Q1. Lalu dari Q0 dan Q1, dicari lagi titik tengahnya sebagai R0. R0 ini lah yang merupakan titik yang kita cari.



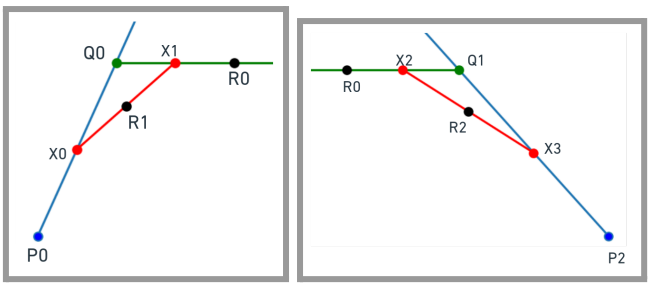
Gambar 3.2.1.2. Proses Conquer

Jika iterasi lebih dari 1, maka dilakukan langkah berikutnya yaitu Langkah Divide. Langkah ini membagi 2 titik di kanan dan kiri. Titik di kiri yaitu titik titik (P0,Q0,R0) dan titik di kanan yaitu titik titik (R0,Q1,P2).



Gambar 3.2.1.3. Proses Divide

Masing masing dari kedua bagian tersebut akan diproses lagi dengan langkah conquer menjadi seperti ini:



Gambar 3.2.1.4. Proses Conquer ke 2

Lalu langkah yang terakhir adalah combine dengan memasukan semua titik Ri kedalam array ditambah titik P0 dan titik P2 yang nantinya akan divisualisasikan dengan kakas matplotlib. Berikut ini merupakan potongan codenya:

```
1  #membentuk array kumpulan titik tengah sesuai iterasi
2  def newCoordinate(self, point1, point2, point3, iterationNow, iterations):
3      if iterationNow < iterations:
4          iterationNow += 1
5          titikTengah1 = self.titikTengah(point1, point2)
6          titikTengah2 = self.titikTengah(point2, point3)
7          titikTengah3 = self.titikTengah(titikTengah1, titikTengah2)
8          # mencari titik tengah dari 3 titik di kiri
9          self.newCoordinate(point1, titikTengah1, titikTengah3, iterationNow, iterations)
10         self.resultPointsX.append(titikTengah3[0])      #append titik tengah ke array
11         self.resultPointsY.append(titikTengah3[1])      #append titik tengah ke array
12         # mencari titik tengah dari 3 titik di kanan
13         self.newCoordinate(titikTengah3, titikTengah2, point3, iterationNow, iterations)
```

Gambar 3.2.1.5. Code

Conquer:

- def newCoordinate yang melakukan proses untuk mencari titikTengah3

Divide:

- pemanggilan newCoordinate untuk bagian kiri (line 9)
- pemanggilan newCoordinate untuk bagian kanan (line 13)

Combine:

- Memasukan hasil ke dalam array (line 10 dan 11)

3.2.2 File DivideAndConquerBonus.py

Langkah-langkah pada implementasi bonus tidak jauh beda. hanya melakukan generalisasi agar dapat digunakan N titik.

Proses pertama yaitu Conquer dengan melakukan rekursi untuk mencari titik tengah antara setiap pasangan titik hingga hanya satu titik tengah yang tersisa yaitu (R). Lalu langkah berikutnya yaitu Divide dengan membagi titik titik yang ada di sebelah kiri R dan di sebelah kanan R

Untuk setiap bagian, baik kiri dan kanan. Akan dilakukan lagi proses Conquer untuk mencari titik tengah (R) yang berikutnya sampai iterasi yang ditentukan. Langkah terakhir yaitu Combine dengan memasukan semua titik R ke dalam array.

Perlu diketahui bahwa implementasi bonus berupa generalisasi code memerlukan looping tambahan seperti untuk mencari titik tengah dari N titik. Lalu untuk membagi titik titik bagian kanan dan kiri yang tidak sesimpel pada file DivideAndConquer yang hanya ada 3 titik. Hal ini menyebabkan **kompleksitas waktu** pada file Bonus **bertambah**.

Kedua file diatas semuanya terpakai, file DivideAndConquer akan dipakai untuk titik berjumlah 3 dan file DivideAndConquerBonus akan digunakan untuk titik yang lebih dari 3.

3.3 Source Code

Struktur folder yang kami buat yaitu:

```
Tucil2_13522070_13522082
│
├── bin
├── doc
│   └── Tucil2_13522070_13522082.pdf
├── src
│   ├── assets
│   ├── lib
│   │   ├── BruteForce.py
│   │   ├── BruteforceBonus.py
│   │   ├── DivideAndConquer.py
│   │   ├── DivideAndConquerBonus.py
│   │   ├── InputHandler.py
│   │   └── Visualizer.py
│   ├── assets
│   ├── gui.py
│   └── main.py
├── test
└── README.md
```

3.3.1 BruteForce.py

BruteForce.py

```python
import numpy
import math

# Algoritma Brute Force untuk control point (n) = 3
def bezier_bf(points, iteration):
    precision = 1.0/2**iteration
    points_fix = [points[0]]
```

```
    t = 0
    while t < 1:
        t += precision
        x = (1 - t)**2 * points[0][0] + 2 * (1 - t) * t * points[1][0] + t**2 *
points[2][0]
        y = (1 - t)**2 * points[0][1] + 2 * (1 - t) * t * points[1][1] + t**2 *
points[2][1]
        points_fix.append((x, y))
    return points_fix
```

3.3.2 BruteForceBonus.py

BruteForceBonus.py

```
import numpy as np

# Algoritma Brute Force untuk control point (n) > 3
# Fungsi untuk membuat segitiga pascal
def pascal_triangle(n):
    triangle = [[0] * (n + 1) for _ in range(n + 1)]
    for line in range(n + 1):
        for i in range(line + 1):
            if i == 0 or i == line:
                triangle[line][i] = 1
            else:
                triangle[line][i] = triangle[line - 1][i - 1] + triangle[line - 1][i]
    return triangle

# Fungsi untuk membentuk formula sesuai dengan titik yang diinginkan
def pascal_function(n, t):
    triangle = pascal_triangle(n)
    result = []
    for i in range(n + 1):
        coeff = triangle[n][i] * ((1 - t) ** (n - i)) * (t ** i)
        result.append(coeff)
    return result

# Fungsi untuk membentuk titik-titik kurva bezier
def bezier_pascal(points, iteration):
    precision = 1.0/2**iteration
    points_fix = []
    t = 0
    n = len(points) - 1
    while t <= 1:
        pascal_coefficients = pascal_function(n, t)
        x = np.sum([coeff * point[0] for coeff, point in zip(pascal_coefficients,
points)])
        y = np.sum([coeff * point[1] for coeff, point in zip(pascal_coefficients,
points)])
```

```
        points_fix.append((x, y))
        t += precision
    return points_fix
```

3.3.3 DivideAndConquer.py

DivideAndConquer.py

```
import numpy as np

class DivideAndConquer:
    def __init__(self):
        self.resultPointsX = []
        self.resultPointsY = []

    #untuk mencari titik tengah antara 2 titik
    def titikTengah(self, point1, point2):
        return ((point1[0] + point2[0]) / 2, (point1[1] + point2[1]) / 2 )

    #membentuk array kumpulan titik tengah sesuai iterasi
    def newCoordinate(self, point1, point2, point3, iterationNow, iterations):
        if iterationNow < iterations:
            iterationNow += 1
            titikTengah1 = self.titikTengah(point1, point2)
            titikTengah2 = self.titikTengah(point2, point3)
            titikTengah3 = self.titikTengah(titikTengah1, titikTengah2)
            # mencari titik tengah dari 3 titik di kiri
            self.newCoordinate(point1, titikTengah1, titikTengah3, iterationNow,
iterations)
            self.resultPointsX.append(titikTengah3[0])  #append titik tengah ke array
            self.resultPointsY.append(titikTengah3[1])  #append titik tengah ke array
            # mencari titik tengah dari 3 titik di kanan
            self.newCoordinate(titikTengah3, titikTengah2, point3, iterationNow,
iterations)

    #membuat kurva bezier
    def create_bezier(self, points, iterations):
        self.resultPointsX.append(points[0][0])  #append titik awal
        self.resultPointsY.append(points[0][1])
        #cari titik titik tengahnya
        self.newCoordinate(points[0], points[1], points[2], 0, iterations)
        self.resultPointsX.append(points[2][0])
        self.resultPointsY.append(points[2][1])  #append titik akhir
```

3.3.4 DivideAndConquerBonus.py

DivideAndConquerBonus.py

```
import numpy as np

class DivideAndConquerBonus:
    def __init__(self):
        self.resultPoints = []
        self.new_points = []

    #mencari titik tengah antar 2 titik
    def titikTengah(self, point1, point2):
        x = (point1[0] + point2[0]) / 2
        y = (point1[1] + point2[1]) / 2
        return (x, y )

    #mencari titik tengah dari N titik
    def titikTengah_N(self, points):
        if len(points) == 2:
            return self.titikTengah(points[0], points[1])
        else:
            newPoints = []
            add_new_points = []
            for i in range (len(points) -1):
                temp = (self.titikTengah(points[i], points[i+1]))
                newPoints.append(temp)
                if i == 0 or i == (len(points) - 2):
                    add_new_points.append(temp)
            panjang = len(self.new_points)//2
            self.new_points.insert(panjang, add_new_points[1])
            self.new_points.insert(panjang, add_new_points[0])
            return self.titikTengah_N(newPoints)

    #membuat array berisi titik-titik tengah sesuai iterasi
    def newCoordinate_N(self, points, iterationNow, iterations):
        if iterationNow < iterations:
            iterationNow += 1

            #mencari titik tengah
            self.new_points.clear()
            titikTengah = self.titikTengah_N(points)
            split = np.array_split(self.new_points,2)

            #menambahkan titik di kiri dan di kanan titik tengah untuk diiterasi lagi
            left_points = split[0].tolist()
            right_points = split[1].tolist()
            left_points.insert(0, points[0])
            left_points.append(titikTengah)
            right_points.insert(0, titikTengah)
            right_points.append(points[-1])
```

```
            # mencari titik tengah dari N titik di kiri
            self.newCoordinate_N(left_points, iterationNow, iterations)
            self.resultPoints.append(titikTengah)    #append titik tengah ke array
result
            # mencari titik tengah dari N titik di kanan
            self.newCoordinate_N(right_points, iterationNow, iterations)

    #membentuk kurva bezier
    def create_bezier(self, points, iterations):
        self.resultPoints.append(points[0])    #append titik awal
        #cari titik titik tengah sesuai iterasi
        self.newCoordinate_N(points, 0, iterations)
self.resultPoints.append(points[-1])  #append titik akhir
```

3.3.5 InputHandler.py

InputHandler.py

```
# Fungsi ini untuk menerima masukan dari user dan
# Menghandle jika masukan tidak sesuai atau salah format

class textcolors:
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    LIGHTBLUE = '\033[94m'
    LIGHTGREEN = '\033[92m'
    BOLD = '\033[1m'
    ENDC = '\033[0m'
    CYAN = '\033[36m'

def get_points():
    while True:
        try:
            n = int(input(textcolors.BOLD + textcolors.CYAN + "Enter the number of
points: " + textcolors.ENDC))
            if n <= 2:
                print(textcolors.FAIL + "Bezier Curve Cannot be Constructed. Please
Re-enter Input!" + textcolors.ENDC)
                continue
            points = [tuple(map(float, input(textcolors.BOLD + textcolors.CYAN +
f"Enter the coordinates of point P{i} (separated by space): " +
textcolors.ENDC).split())) for i in range(n)]
            iterations = int(input(textcolors.BOLD + textcolors.CYAN + "Enter the
number of iterations: " + textcolors.ENDC))
            if iterations <= 0:
                print(textcolors.FAIL + "Number of iterations cannot be less than
one. Please Re-enter Input!" + textcolors.ENDC)
                continue
```

```
            return n, points, iterations
        except ValueError:
            print(textcolors.FAIL + "Input must be an integer. Please Re-enter
Input!" + textcolors.ENDC)
```

3.3.6 Visualizer.py

Visualizer.py

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

# Menggunakan kakas matplotlib untuk visualisasinya
def plot_curve(result, points, title):
    x = [p[0] for p in result]
    y = [p[1] for p in result]
    plt.title(f"Bézier Curve {title}")
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.grid(True)
    plt.plot(x, y, marker='.', markerfacecolor = "blue")
    xp = [p[0] for p in points]
    yp = [p[1] for p in points]
    plt.plot(xp,yp, marker = "o", markerfacecolor = 'red')
    plt.show()

# Menggunakan kakas matplotlib untuk visualisasi animasi titik per titik pada GUI
def animate_curve(results, points, title):
    fig, ax = plt.subplots()
    scat = ax.scatter([], [], s=5)
    line, = ax.plot([], [])
    xp = [p[0] for p in points]
    yp = [p[1] for p in points]
    plt.plot(xp,yp, marker = "o", markerfacecolor = 'red')
    # Mendapatkan ukuran yang sesuai untuk ukuran kurvanya
    all_x = [x[0] for x in results + points]
    all_y = [x[1] for x in results + points]
    rentangX = max(all_x) - min(all_x)
    rentangY = max(all_y) - min(all_y)
    min_x, max_x = (min(all_x) - (0.1*rentangX)), (max(all_x) + (0.1*rentangX))
    min_y, max_y = (min(all_y) - (0.1*rentangY)), (max(all_y) + (0.1*rentangY))
    ax.set(xlim=[min_x, max_x],
           ylim=[min_y, max_y],
           xlabel='X',
           ylabel='Y',
           title=f"Bézier Curve {title}")
    ax.legend()
```

```
    # Animasi update
    def update(frame):
        x = [p[0] for p in results[:frame]]
        y = [p[1] for p in results[:frame]]
        data = np.stack([x, y]).T
        scat.set_offsets(data)
        line.set_xdata(x)
        line.set_ydata(y)
        return scat, line

    # Membuat animasi
    ani = animation.FuncAnimation(fig, update, frames=len(results) + 1, interval=100)
    plt.show()
```

3.3.7 main.py

main.py

```
from lib.Visualizer import plot_curve
import lib.DivideAndConquer as dnc
import lib.DivideAndConquerBonus as dnc_bonus
import lib.BruteForce as bf
import lib.BruteForceBonus as bf_bonus
import lib.InputHandler as ih
import time
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

class textcolors:
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    LIGHTBLUE = '\033[94m'
    LIGHTGREEN = '\033[92m'
    BOLD = '\033[1m'
    ENDC = '\033[0m'
    CYAN = '\033[36m'

print(textcolors.LIGHTBLUE + """
____  ____  ____  ____  ____  ____      ___  __  __  ____  _  _  ____
(  _\\ ( ___)(_   )(_  _)( ___)(  _ \\    / __)(  )(  )(  _ \\( \\/ )( ___)
) _ < )__)  / /_  _)(_  )__)  )   /  ( (__  )(__)(  )    / \\  /  )__)
(____/(____)(____)(____)(____)(_)\\_)   \\___)(______)(_)\\_)  \\/  (____)
""" + textcolors.ENDC, end="")

print(textcolors.WARNING + """
  __                ___               __        ____          _____
 / /   __ __       / _ | ___ __ __   / /       / __/___      / ___/ ___   ___  ___ ___
_
```

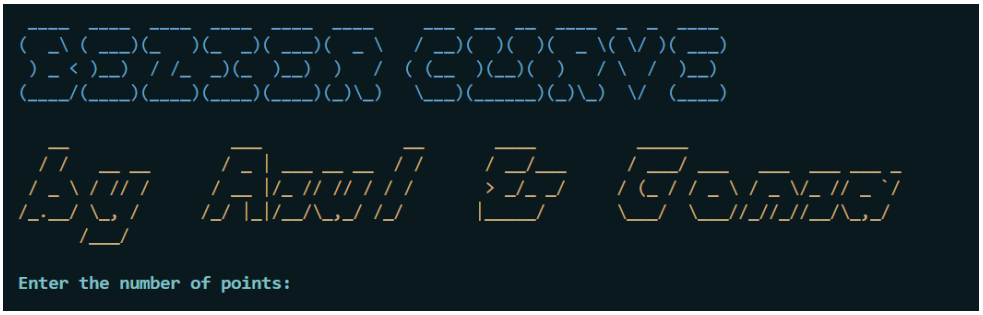
```
/ _ \\ / // /      / __ |/_ // // / / /       > _/_ _/     / (_ / / _ \\ / _ \\/_ //
_ `/
/_.__/ \\_, /      /_/ |_|/__/\\_,_/ /_/       |_____/       \\___/
\\___//_//_//__/\\_,_/
     /___/
""" + textcolors.ENDC)

# Mendapatkan input banyaknya titik dan masing-masing titiknya
# serta besaran iterasi yang diinginkan
n, points, iterations = ih.get_points()

# Algoritma untuk spek wajib
if (n == 3):
    # Waktu eksekusi menggunakan algoritma Divide and Conquer
    bezier_dnc = dnc.DivideAndConquer()
    start_time = time.time()
    bezier_dnc.create_bezier(points, iterations)
    end_time = time.time()
    print(textcolors.BOLD + textcolors.LIGHTGREEN + "Divide and Conquer Approach
Time:" + textcolors.ENDC, (end_time - start_time)*1000 , "ms")

    # Waktu eksekusi menggunakan algoritma Brute Force
    start_time = time.time()
    curve_points_bf = bf.bezier_bf(points, iterations)
    end_time = time.time()
    print(textcolors.BOLD + textcolors.LIGHTGREEN + "Brute Force Approach Time:" +
textcolors.ENDC, (end_time - start_time)*1000, "ms")

    curve_points_dnc = []
    for i in range(len(bezier_dnc.resultPointsX)):
        temp = (bezier_dnc.resultPointsX[i], bezier_dnc.resultPointsY[i])
        curve_points_dnc.append(temp)

    # Mengecek bahwa kurva yang terbentuk akan sama antara kedua algoritma
    if len(curve_points_dnc) == len(curve_points_bf):
        print(textcolors.BOLD + textcolors.LIGHTGREEN + "Both DnC and Brute Force
algorithms yield the same Bezier point:" + textcolors.ENDC, len(curve_points_dnc),
"points")
    else:
        print(textcolors.FAIL + "Oops, there is a difference between the two
algorithms." + textcolors.ENDC)

    # Visualisasi menggunakan matplotlib
    plot_curve(curve_points_dnc, points, "Divide and Conquer")
    plot_curve(curve_points_bf, points, "Brute Force")

# Algoritma untuk spek bonus
else:
    # Waktu eksekusi menggunakan algoritma Divide and Conquer
    bezier_dnc = dnc_bonus.DivideAndConquerBonus()
```

```
    start_time = time.time()
    bezier_dnc.create_bezier(points, iterations)
    end_time = time.time()
    print(textcolors.BOLD + textcolors.LIGHTGREEN + "Divide and Conquer Approach
Time:" + textcolors.ENDC, (end_time - start_time)*1000 , "ms")

    # Waktu eksekusi menggunakan algoritma Brute Force
    start_time = time.time()
    curve_points_bf = bf_bonus.bezier_pascal(points, iterations)
    end_time = time.time()
    print(textcolors.BOLD + textcolors.LIGHTGREEN + "Brute Force Approach Time:" +
textcolors.ENDC, (end_time - start_time)*1000 , "ms")

    # Mengecek bahwa kurva yang terbentuk akan sama antara kedua algoritma
    if len(bezier_dnc.resultPoints) == len(curve_points_bf):
        print(textcolors.BOLD + textcolors.LIGHTGREEN + "Both DnC and Brute Force
algorithms yield the same Bezier point:" + textcolors.ENDC,
len(bezier_dnc.resultPoints), "points")
    else:
        print(textcolors.FAIL + "Oops, there is a difference between the two
algorithms." + textcolors.ENDC)

    # Visualisasi menggunakan matplotlib
    plot_curve(bezier_dnc.resultPoints, points, "Divide and Conquer")
    plot_curve(curve_points_bf, points, "Brute Force")
```

BAB IV

HASIL UJI DAN ANALISIS

Setelah dibuat algoritma penyelesaian menggunakan Divide and Conquer, hasilnya akan dibandingkan dengan algoritma brute force sebagai referensi. Parameter yang dijadikan perbandingan adalah kesamaan titik-titik yang divisualisasikan, banyak titik yang terbentuk, serta waktu yang diperlukan untuk membentuk kurva bezier.



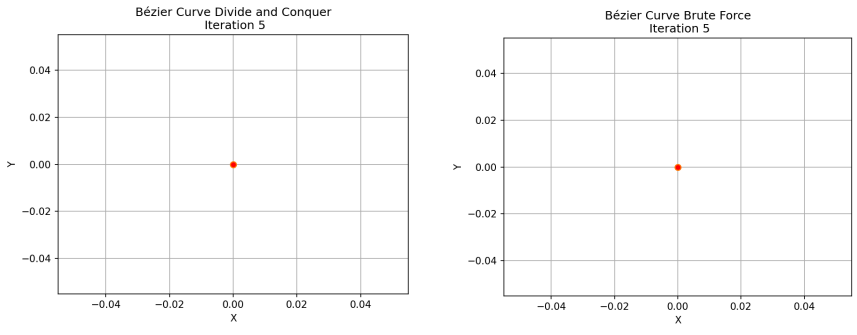
Gambar 4. Tampilan awal CLI

4.1 Hasil Uji Program Wajib

Hasil uji program menggunakan CLI karena menggunakan algoritma yang khusus untuk 3 titik (belum di generalisasi)

4.1.1 Uji Program 1

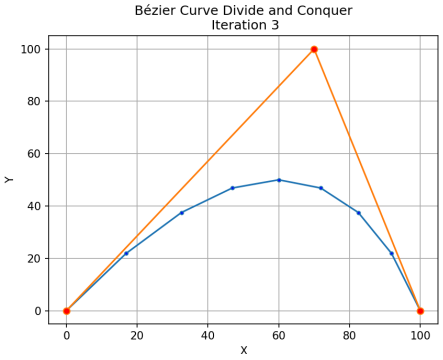
```
Enter the number of points: 3
Enter the coordinates of point P0 (separated by space): 0 0
Enter the coordinates of point P1 (separated by space): 0 0
Enter the coordinates of point P2 (separated by space): 0 0
Enter the number of iterations: 5
Divide and Conquer Approach Time: 0.0 ms
Brute Force Approach Time: 0.0 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 33 points
```

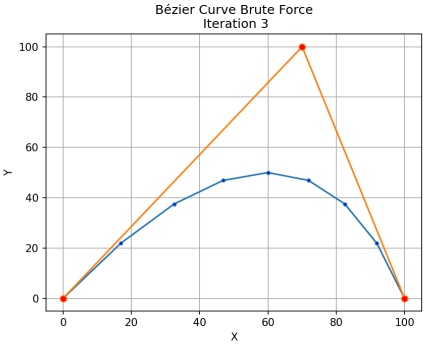


Time DnC = 0.0 ms
Time BF = 0.0 ms

4.1.2 Uji Program 2

```
Enter the number of points: 3
Enter the coordinates of point P0 (separated by space): 0 0
Enter the coordinates of point P1 (separated by space): 70 100
Enter the coordinates of point P2 (separated by space): 100 0
Enter the number of iterations: 3
Divide and Conquer Approach Time: 0.0 ms
Brute Force Approach Time: 0.0 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 9 points
```

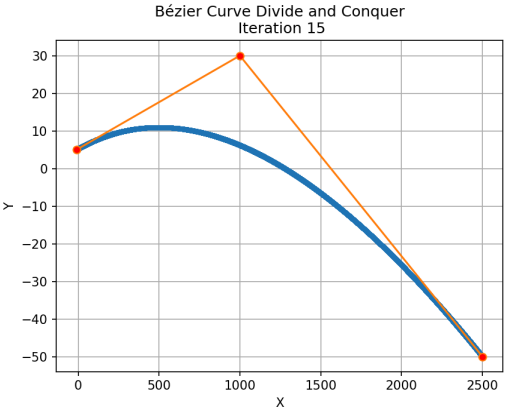


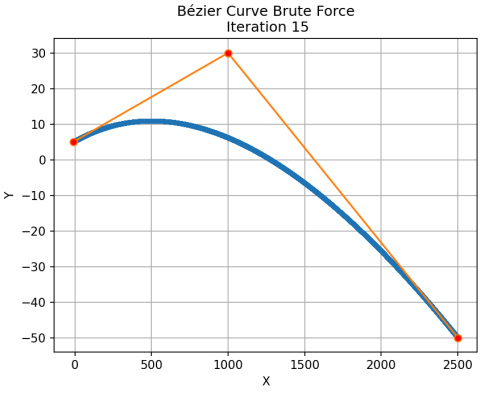


Time DnC = 0.0 ms
Time BF = 0.0 ms

4.1.3 Uji Program 3

```
Enter the number of points: 3
Enter the coordinates of point P0 (separated by space): -10 5
Enter the coordinates of point P1 (separated by space): 1000 30
Enter the coordinates of point P2 (separated by space): 2500 -50
Enter the number of iterations: 15
Divide and Conquer Approach Time: 26.604890823364258 ms
Brute Force Approach Time: 25.000333786010742 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 32769 points
```

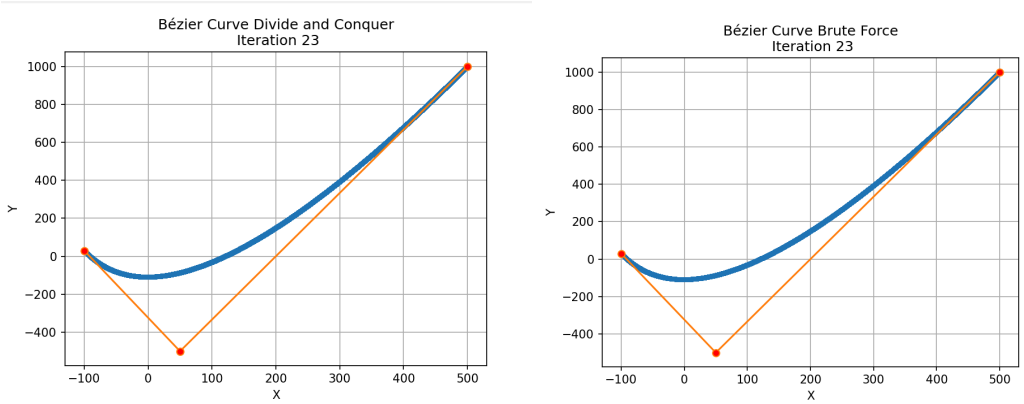




Time DnC = 26,604 ms
Time BF = 25,00 ms

4.1.4 Uji Program 4

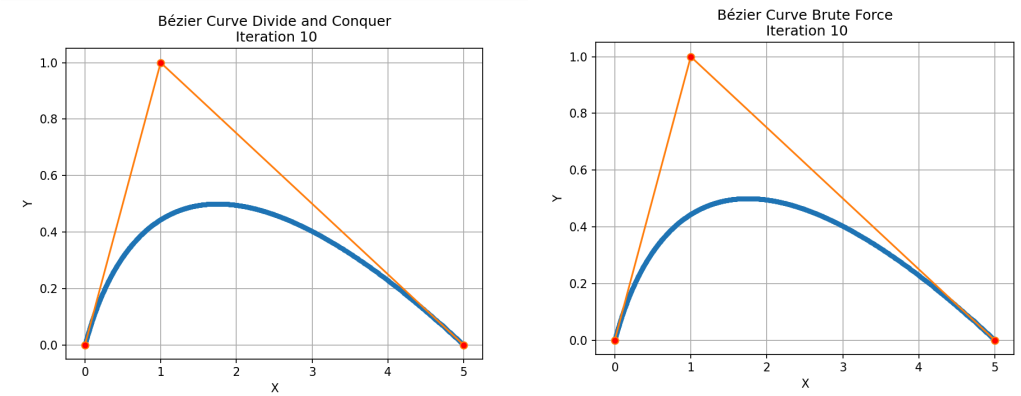
```
Enter the number of points: 3
Enter the coordinates of point P0 (separated by space): -100 30
Enter the coordinates of point P1 (separated by space): 50 -500
Enter the coordinates of point P2 (separated by space): 500 1000
Enter the number of iterations: 23
Divide and Conquer Approach Time: 6108.22606086731 ms
Brute Force Approach Time: 7117.451429367065 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 8388609 points
```



Time DnC = 6108,2 ms
Time BF = 7117,4 ms

4.1.5 Uji Program 5

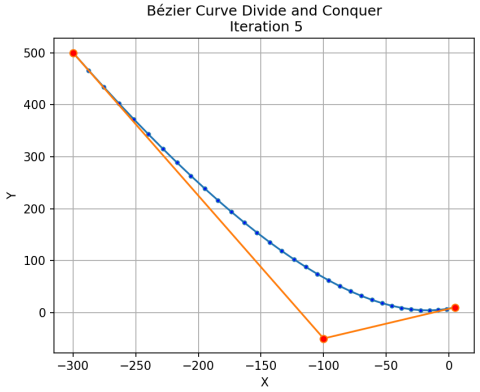
```
Enter the number of points: 3
Enter the coordinates of point P0 (separated by space): 0 0
Enter the coordinates of point P1 (separated by space): 1 1
Enter the coordinates of point P2 (separated by space): 5 0
Enter the number of iterations: 10
Divide and Conquer Approach Time: 0.9992122650146484 ms
Brute Force Approach Time: 1.0004043579101562 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 1025 points
```

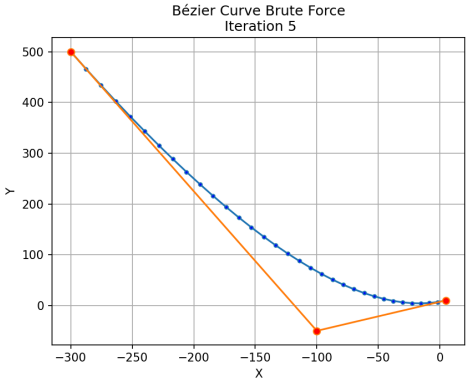


Time DnC = 0.999ms
Time BF = 1.0004 ms

4.1.6 Uji Program 6

```
Enter the number of points: -3
Bezier Curve Cannot be Constructed. Please Re-enter Input!
Enter the number of points: 2
Bezier Curve Cannot be Constructed. Please Re-enter Input!
Enter the number of points: 3
Enter the coordinates of point P0 (separated by space): a b
Input must be an integer. Please Re-enter Input!
Enter the number of points: 3
Enter the coordinates of point P0 (separated by space): 5 10
Enter the coordinates of point P1 (separated by space): -100 -50
Enter the coordinates of point P2 (separated by space): -300 500
Enter the number of iterations: 5
Divide and Conquer Approach Time: 0.0 ms
Brute Force Approach Time: 0.0 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 33 points
```

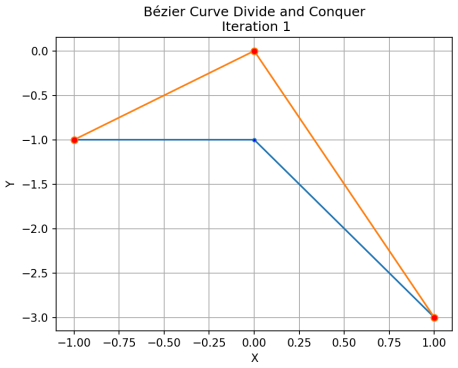


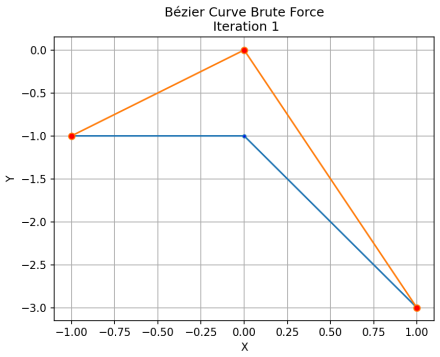


Time DnC = 0,0 ms
Time BF = 0,0 ms

4.1.7 Uji Program 7

```
Enter the number of points: 3
Enter the coordinates of point P0 (separated by space): -1 -1
Enter the coordinates of point P1 (separated by space): 0 0
Enter the coordinates of point P2 (separated by space): 1 -3
Enter the number of iterations: 1
Divide and Conquer Approach Time: 0.0 ms
Brute Force Approach Time: 0.0 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 3 points
```

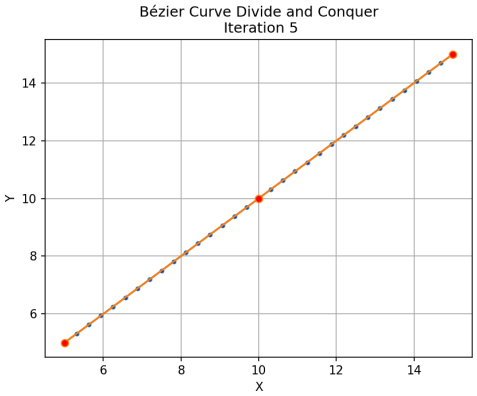


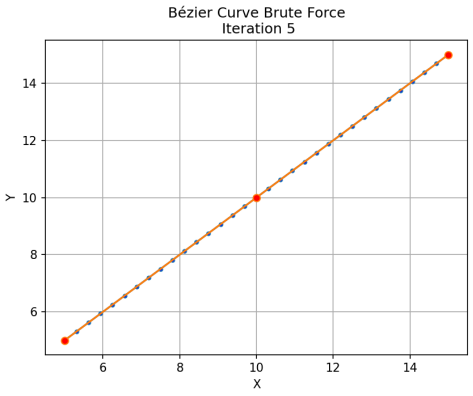


Time DnC = 0,0 ms
Time BF = 0,0 ms

4.1.8 Uji Program 8

```
Enter the number of points: 3
Enter the coordinates of point P0 (separated by space): 5 5
Enter the coordinates of point P1 (separated by space): 10 10
Enter the coordinates of point P2 (separated by space): 15 15
Enter the number of iterations: 5
Divide and Conquer Approach Time: 0.0 ms
Brute Force Approach Time: 0.9968280792236328 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 33 points
```

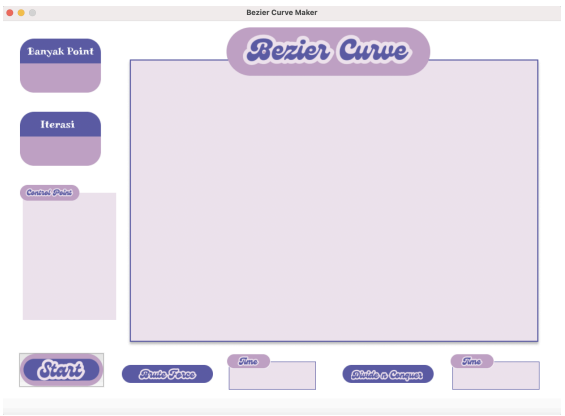




Time DnC = 0,0 ms
Time BF = 0,996 ms

4.2 Hasil Uji Program Bonus

Hasil Uji pada bagian bonus dilakukan dengan CLI maupun dengan menampilkan GUI yang dibuat dengan tkInter. Algoritma yang digunakan yaitu algoritma yang sudah digeneralisasi (bisa dipakai untuk N>3).



Gambar 4.2.1 Tampilan awal GUI

Untuk bagian bonus ini, GUI divisualisasikan dengan menampilkan beberapa kolom input seperti banyak point, iterasi yang diinginkan dan juga *control point*. Control point dimasukkan dengan format x, y lalu untuk pemisah antara point nya ditandai dengan enter. Jadi jika ada 3 point maka masukkan pada *Control Point*-nya adalah

x_1, y_1

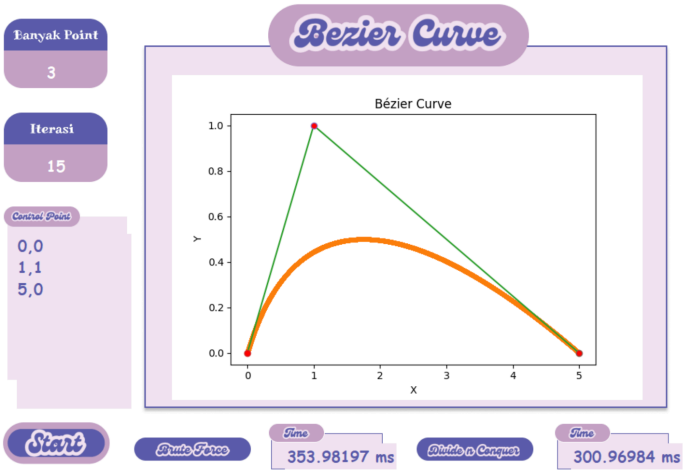
x_2, y_2

x_3, y_3

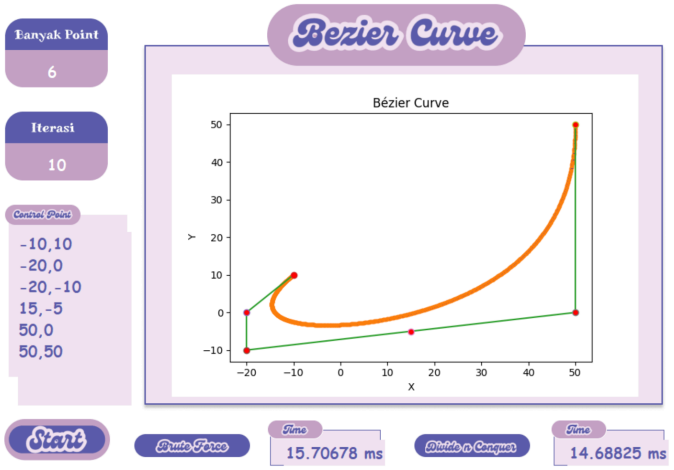
Selain itu, pada GUI yang kami buat juga tidak menghandle input error atau ketidaksesuaian antara jumlah point dan *control points* yang dimasukkan. Ketika semua parameter sudah diinput maka untuk meng-*generate* kurvanya, user dapat langsung mengklik tombol start di kiri bagian bawah. Setelah di klik, maka kurva akan terbentuk di tempat yang telah disediakan dan di bagian bawahnya juga sudah terdapat informasi mengenai waktu eksekusi baik dengan pendekatan algoritma Brute Force maupun dengan algoritma Divide and Conquer. Pembentukan kurva yang diaplikasikan di GUI ini hanya dengan pendekatan algoritma Divide and Conquer saja. Terakhir, user dapat melihat langsung kurva yang terbentuk saat melakukan pergantian besaran iterasi pada kolom yang tersedia tanpa harus mereset dari awal terlebih dahulu.

Namun, hal yang menjadi perhatian adalah, untuk visualisasi animasi pergerakan kurvanya hanya berlaku untuk iterasi kurang dari 10. Hal ini dikarenakan untuk iterasi yang lebih dari sama dengan 10, animasi akan berjalan dengan sangat lambat dan akan sulit untuk disesuaikan per iterasinya. Terbukti bahwa jika input iterasinya sebesar 11, maka akan ada $2^{11} + 1$ titik kurva bezier yang terbentuk atau sekitar 2049 titik. Sehingga untuk iterasi yang lebih dari 11 maka visualisasi kurva diatur untuk langsung menunjukkan hasil akhirnya saja tanpa pergerakan animasi. Kekurangan lainnya untuk GUI ini adalah dibagian waktu eksekusi. Kedua algoritma digabung ke dalam satu tombol yang artinya ketika tombol start di klik kedua algoritma akan berjalan secara bergantian dengan urutan algoritma brute force terlebih dahulu lalu algoritma divide and conquer. Hal ini menyebabkan saat GUI pertama kali dijalankan akan ada keterlambatan di salah satu algoritma yang menyebabkan kurang akurat perhitungan waktunya sehingga solusi yang dapat dilakukan adalah mengklik sekali lagi tombol startnya dan setelah itu selisih waktu antara keduanya akan kembali normal dan menjadi lebih akurat.

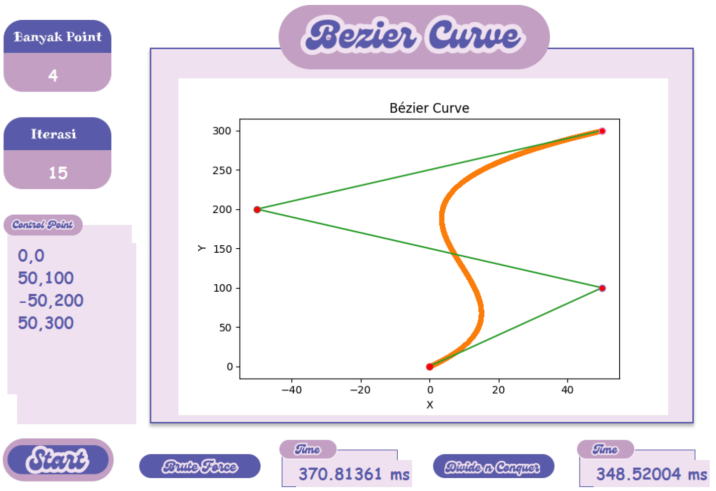
4.2.1 Uji Program 1



4.2.2 Uji Program 2

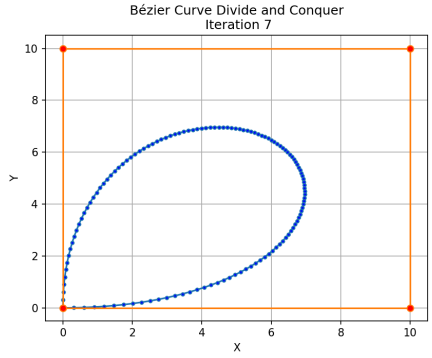


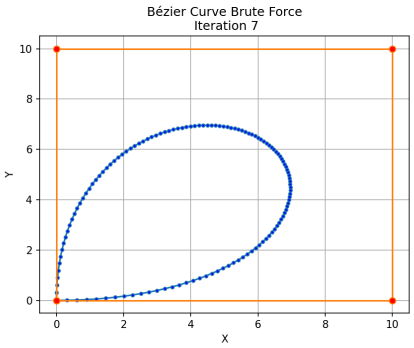
4.2.3 Uji Program 3



4.2.4 Uji Program 4

```
Enter the number of points: 5
Enter the coordinates of point P0 (separated by space): 0 0
Enter the coordinates of point P1 (separated by space): 0 10
Enter the coordinates of point P2 (separated by space): 10 10
Enter the coordinates of point P3 (separated by space): 10 0
Enter the coordinates of point P4 (separated by space): 0 0
Enter the number of iterations: 7
Divide and Conquer Approach Time: 2.000093460083008 ms
Brute Force Approach Time: 2.0177364349365234 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 129 points
```

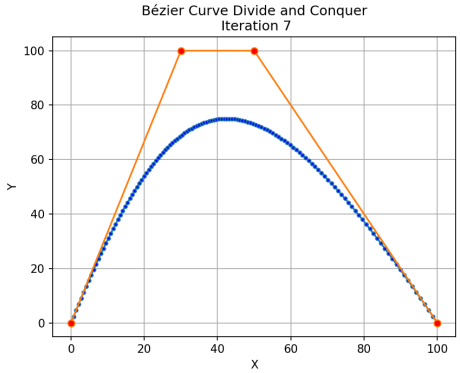


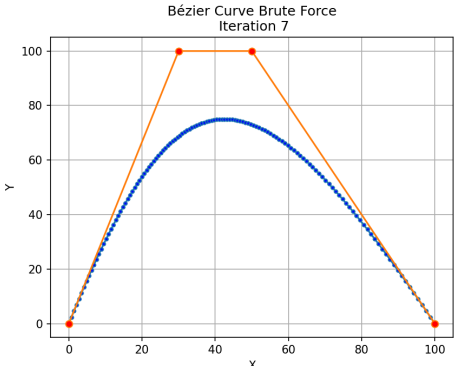


Time DnC = 2,00 ms
Time BF = 2.01 ms

4.2.5 Uji Program 5

```
Enter the number of points: 4
Enter the coordinates of point P0 (separated by space): 0 0
Enter the coordinates of point P1 (separated by space): 30 100
Enter the coordinates of point P2 (separated by space): 50 100
Enter the coordinates of point P3 (separated by space): 100 0
Enter the number of iterations: 7
Divide and Conquer Approach Time: 1.058340072631836 ms
Brute Force Approach Time: 2.0160675048828125 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 129 points
```

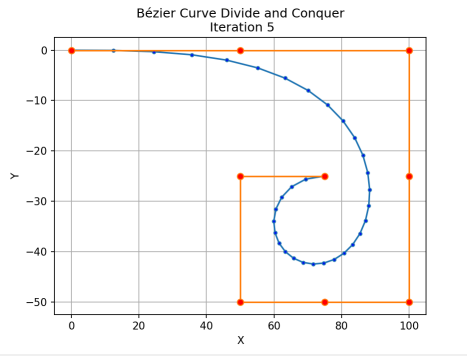


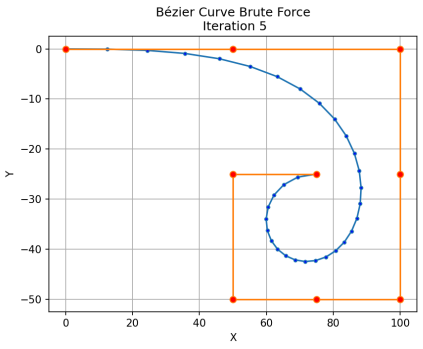


Time DnC = 1.05 ms
Time BF = 2,01 ms

4.2.6 Uji Program 6

```
Enter the number of points: 9
Enter the coordinates of point P0 (separated by space): 0 0
Enter the coordinates of point P1 (separated by space): 50 0
Enter the coordinates of point P2 (separated by space): 100 0
Enter the coordinates of point P3 (separated by space): 100 -25
Enter the coordinates of point P4 (separated by space): 100 -50
Enter the coordinates of point P5 (separated by space): 75 -50
Enter the coordinates of point P6 (separated by space): 50 -50
Enter the coordinates of point P7 (separated by space): 50 -25
Enter the coordinates of point P8 (separated by space): 75 -25
Enter the number of iterations: 5
Divide and Conquer Approach Time: 1.0099411010742188 ms
Brute Force Approach Time: 1.2390613555908203 ms
Both DnC and Brute Force algorithms yield the same Bezier point: 33 points
```





Time DnC = 1,009 ms
Time BF = 1,239 ms

4.3 Analisis Algoritma Brute Force

Algoritma Brute Force yang diimplementasikan untuk spek wajib maupun spek bonus sebenarnya tidak jauh berbeda. Letak perbedaannya hanyalah untuk spek wajib, formula yang digunakan untuk menghitung titik kurva sudah didefinisikan sehingga besaran nilai *t* atau presisinya dapat langsung digunakan sepanjang iterasi. Sedangkan pada spek bonus, formula untuk menghitung kurva harus dibentuk terlebih dahulu menyesuaikan dengan jumlah *control point* yang dimasukkan oleh user.

Selanjutnya, kita akan menganalisis kompleksitas waktu algoritma wajib. Algoritma ini hanya ada 1 fungsi yaitu bezier_bf. Fungsi ini tidak menggunakan pendekatan rekrusif namun iteratif berdasarkan banyaknya iterasi. Semakin kecil nilai iteration, semakin tinggi presisinya, dan semakin banyak iterasi yang diperlukan, sehingga kompleksitas waktu akan meningkat. Jadi kompleksitas waktu pada algoritma ini adalah O($2^{iterations}$)

Selanjutnya, kita akan menganalisis kompleksitas waktu algoritma bonus dari masing-masing fungsi yang dihasilkan dengan pendekatan algoritma Brute Force.

1. **pascal_triangle(n)**
   Fungsi ini digunakan untuk menghasilkan segitiga Pascal dengan tinggi n. Sehingga kompleksitas waktunya adalah O(n^2), karena terdapat dua loop bersarang, masing-masing dengan iterasi sebanyak n kali.
2. **pascal_function(n, t)**
   Fungsi ini menghasilkan koefisien Pascal untuk suatu nilai t. Oleh karenanya kompleksitas waktu fungsi ini adalah sebesar O(n), karena hanya terdiri dari satu loop dengan iterasi sebanyak n kali.
3. **bezier_pascal($points$, $iteration$)**
   Fungsi ini menghasilkan titik-titik pada kurva Bezier dengan menggunakan koefisien Pascal. Iterasi dalam while loop dihitung berdasarkan presisi yang diinginkan. Jumlah iterasi dalam while loop adalah tergantung pada variabel *iteration*, yang berarti semakin besar nilai *iterations*, semakin banyak iterasi yang dilakukan. Jumlah iterasi pada while loop: $2^{iterations}$. Di dalam while loop, ada pemanggilan fungsi pascal_function(n, t) yang memiliki kompleksitas waktu O(n), di mana n adalah jumlah titik kontrol. Jumlah total pemanggilan pascal_function dalam while loop

adalah $2^{iterations}$ kali. Jadi, kompleksitas waktu total untuk fungsi bezier_pascal adalah O($n \cdot 2^{iterations}$), di mana n adalah jumlah titik kontrol.

Fungsi pascal_triangle dan pascal_function memiliki kompleksitas waktu yang cukup efisien. Namun, keefisienan dari bezier_pascal sangat dipengaruhi oleh nilai *iterations*. Semakin besar nilai *iterations*, semakin banyak iterasi yang diperlukan, yang dapat membuat waktu eksekusi menjadi lebih lambat.

Fungsi ini adalah pendekatan Brute Force untuk menghitung titik pembentuk kurva Bezier. Meskipun metode ini sederhana dan mudah dimengerti, namun dapat menjadi tidak efisien ketika jumlah titik kontrol atau tingkat presisi yang diinginkan menjadi besar. Metode ini efektif untuk keperluan implementasi sederhana, tetapi tidak cocok untuk aplikasi yang membutuhkan performa tinggi, terutama ketika jumlah titik kontrol besar atau presisi yang tinggi diperlukan.

4.4 Analisis Algoritma Divide and Conquer

Algoritma Divide and Conquer dapat dilihat memiliki perbedaan waktu yang cukup signifikan antara spek wajib dan spek bonus. Perbedaan terlihat jelas saat kedua algoritma sama sama menggunakan 3 titik. Dapat dilihat pada Uji (4.1.5) dan uji (4.2.1). Dimana Algoritma wajib memerlukan waktu 0,99 ms dan algoritma bonus yang sudah di generalisasi memerlukan waktu 300,96 ms

Hal ini dikarenakan pada implementasi spek bonus, diperlukan adanya iterasi iterasi tambahan agar dapat menghandle titik general (N > 3). Contoh iterasi tambahan yaitu pada algoritma bonus, perlu disimpan titik titik yang ada di bagian kiri dan yang ada di bagian kanan pada array. Padahal pada algoritma wajib, tidak perlu.

Selanjutnya, kita akan menganalisis kompleksitas waktu Algoritma wajib dari masing-masing fungsi yang dihasilkan dengan pendekatan algoritma Divide and Conquer.

1. **titikTengah**
   fungsi ini memiliki kompleksitas O(1) karena bersifat konstan
2. **newCoordinate**
   fungsi ini memiliki kompleksitas O($2^{iterations}$) karena fungsi ini rekrusif dan memanggil dirinya sendiri 2 kali pada setiap iterasi. Sehingga Jumlah panggilan rekursif secara eksponensial meningkat dengan jumlah iterasi.
3. **createBezier**
   fungsi ini hanya memanggil newCoordinate dan tidak melakukan iterasi sehingga kompleksitasnya O($2^{iterations}$)

Jadi, kompleksitas waktu untuk algoritma wajib adalah O($2^{iterations}$).

Selanjutnya, kita akan menganalisis kompleksitas waktu Algoritma bonus dari masing-masing fungsi yang dihasilkan dengan pendekatan algoritma Divide and Conquer.

1. **titikTengah**
   fungsi ini memiliki kompleksitas O(1) karena bersifat konstan

2. **titikTengah_N**
   Fungsi ini melakukan rekursi untuk mencari titik tengah antara setiap pasangan titik hingga hanya satu titik tengah yang tersisa. Sehingga kompleksitasnya O(n) dengan n adalah jumlah titik.
3. **newCoordinate_N**
   fungsi ini melakukan rekrusi dengan memanggil dirinya sendiri sebanyak 2 kali untuk tiap iterasi sehingga kompleksitasnya O($2^{iterations}$). Pada setiap iterasi juga, fungsi ini memanggil fungsi titikTengah_N sehingga kompleksitasnya menjadi O(n.$2^{iterations}$)
4. **create_bezier**
   fungsi ini hanya memanggil newCoordinat_Ne dan tidak melakukan iterasi sehingga kompleksitasnya O(n.$2^{iterations}$)

Jadi, kompleksitas waktu untuk algoritma wajib adalah O(n. $2^{iterations}$) dengan n adalah jumlah titik.

Fungsi newCoordinate dan newCoordinate_N memiliki kompleksitas waktu yang sangat dipengaruhi oleh nilai *iterations*. Semakin besar nilai *iterations*, semakin banyak iterasi yang diperlukan, yang dapat membuat waktu eksekusi menjadi lebih lambat.

Fungsi ini adalah pendekatan Divide and Conquer untuk menghitung titik pembentuk kurva Bezier. Dalam kasus kurva 3 titik, lebih efektif untuk menggunakan algoritma wajib yang belum di generalisasi.

4.5 Perbandingan Antara Algoritma Brute Force dan Divide and Conquer

Algoritma Brute Force dan Algoritma Divide and Conquer memiliki hasil akhir kurva yang sama. Menandakan bahwa kedua algoritma sudah benar. Algoritma Divide and Conquer memiliki waktu execution yang cenderung lebih cepat dari brute force walaupun kadang masih kalah juga dari brute force.

Algoritma Brute Force dan Algoritma Divide and Conquer memiliki kompleksitas waktu yang sama karena waktu yang diperlukan baik untuk proses cepat ataupun lama memiliki selisih yang sedikit dan linear.

Kompleksitas algoritma wajib pada algoritma Brute Force dan algoritma Divide and Conquer sama sama O($2^{iterations}$). Sedangkan untuk algoritma bonus, keduanya baik algoritma Brute Force dan Divide and Conquer memiliki kompleksitas yang sama yaitu O(n. $2^{iterations}$) dengan n merupakan banyaknya titik.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Algoritma Brute Force digunakan untuk pembanding dan mengecek apakah kurva yang dibuat dengan Divide and Conquer sudah sama atau belum. Selain itu Algoritma Brute Force juga digunakan sebagai pembanding waktu / kompleksitas waktu.

Kompleksitas waktu pada algoritma bonus dengan menggeneralisasi titik untuk N titik lebih besar dari kompleksitas waktu algoritma yang hanya khusus untuk 3 titik saja. DImana kompleksitas algoritma general yaitu O($n.\ 2^{iterations}$) dengan n adalah jumlah titik dan kompleksitas waktu algoritma khusus untuk 3 titik saja yaitu O($2^{iterations}$).

Dalam kasus algoritma yang efisien baik di Brute Force maupun Divide and Conquer. Divide and Conquer belum bisa menghasilkan kompleksitas waktu yang lebih cepat dari brute force

5.2 Saran

Perlu explore lagi agar dapat menganimasikan kurva lebih baik lagi.

DAFTAR PUSTAKA

Munir, R. (2022). Algoritma Divide and Conquer (1). Retrieved from Homepage Rinaldi Munir: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf

Munir, R. (2022). Algoritma Divide and Conquer (2). Retrieved from Homepage Rinaldi Munir: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf

Munir, R. (2022). Algoritma Divide and Conquer (3). Retrieved from Homepage Rinaldi Munir: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf

Munir, R. (2022). Algoritma Divide and Conquer (4). Retrieved from Homepage Rinaldi Munir: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf

LAMPIRAN

Pranala

Kode program dapat diakses pada link berikut

https://github.com/keanugonza/Tucil2_13522070_13522082.git

How to Use

- Untuk menjalankan program melalui CLI,

  python run main.py

  Catatan: memasukan titik koordinat dengan format tanpa spasi seperti ini

  3 5

  2 3

- Untuk menjalankan program menggunakan GUI,

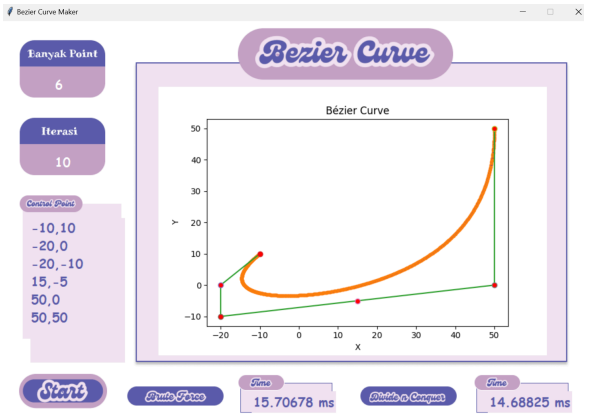
  python run gui.py

  Catatan: memasukan titik koordinat dengan koma seperti ini

  3,5

  2,3

  Contoh:



Tabel Poin

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