IF2211 STRATEGI ALGORITMA
TUGAS KECIL 2
MEMBANGUN KURVA BEZIER DENGAN ALGORITMA TITIK TENGAH BERBASIS DIVIDE AND CONQUER

Dipersiapkan oleh:

Ahmad Naufal Ramadan - 13522005

Imanuel Sebastian Girsang - 13522058

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2024

Daftar Isi

1. Deskripsi Tugas

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

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P0 sampai Pn , dengan n disebut order (n = 1 untuk linier, n = 2 untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) umumnya tidak terletak pada kurva. Pada gambar 1 diatas, titik kontrol pertama adalah P0, sedangkan titik kontrol terakhir adalah P3. Titik kontrol P1 dan P2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P0 dan P1 yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan kurva Bézier linier. Misalkan terdapat sebuah titik Q0 yang berada pada garis yang dibentuk oleh P0 dan P1, maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh B(t) dari P0 ke P1. Misalnya ketika t = 0.25, maka B(t) adalah seperempat jalan dari titik P0 ke P1. sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan B(t) membentuk sebuah garis lurus dari P0 ke P1.

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P2, dengan P0 dan P2 sebagai titik kontrol awal dan akhir, dan P1 menjadi titik kontrol antara. Dengan menyatakan titik Q1 terletak diantara garis yang menghubungkan P1 dan P2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q0 berada, maka dapat dinyatakan sebuah titik baru, R0 yang berada diantara garis yang menghubungkan Q0 dan Q1 yang bergerak membentuk kurva Bézier kuadratik terhadap titik P0 dan P2. 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 Q0 dan Q1, maka diperoleh persamaan sebagai berikut.

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

Proses ini dapat juga diaplikasikan untuk jumlah titik yang lebih dari tiga, misalnya empat titik akan menghasilkan kurva Bézier kubik, lima titik akan menghasilkan kurva Bézier kuartik, dan seterusnya. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya.

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

2. Analisis dan Implementasi dengan Algoritma Brute Force

Kurva Bézier dengan jumlah titik kontrol n didapatkan dengan mengiterasi nilai t pada interval 0 hingga 1 dengan menggunakan nilai t yang berubah dengan suatu nilai tertentu setiap waktunya. Untuk memperoleh formula umum dari Kurva Bézier dengan n titik kontrol, perhatikan dua contoh berikut:

$$S_0 = B(t) = (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + t^3P_3, \qquad t \in [0,1]$$

$$T_0 = B(t) = (1-t)^4P_0 + 4(1-t)^3tP_1 + 6(1-t)^2t^2P_2 + 4(1-t)t^3P_3 + t^4P_4, \qquad t \in [0,1]$$

Gambar 1. Formula membangun kurva Bézier untuk 4 titik kontrol dan 5 titik kontrol

Dari kedua contoh tersebut, dapat disimpulkan bahwa untuk mendapatkan suatu titik pada Kurva Bézier dengan nilai t tertentu, perlu dilakukan pengalian suatu pengali dengan setiap titik kontrol yang ada. Untuk mendapatkan pengali yang digunakan, dapat dilihat bahwa angka tersebut mengikuti ekspansi dari

$$((1 - t) + t)^{n-1}$$

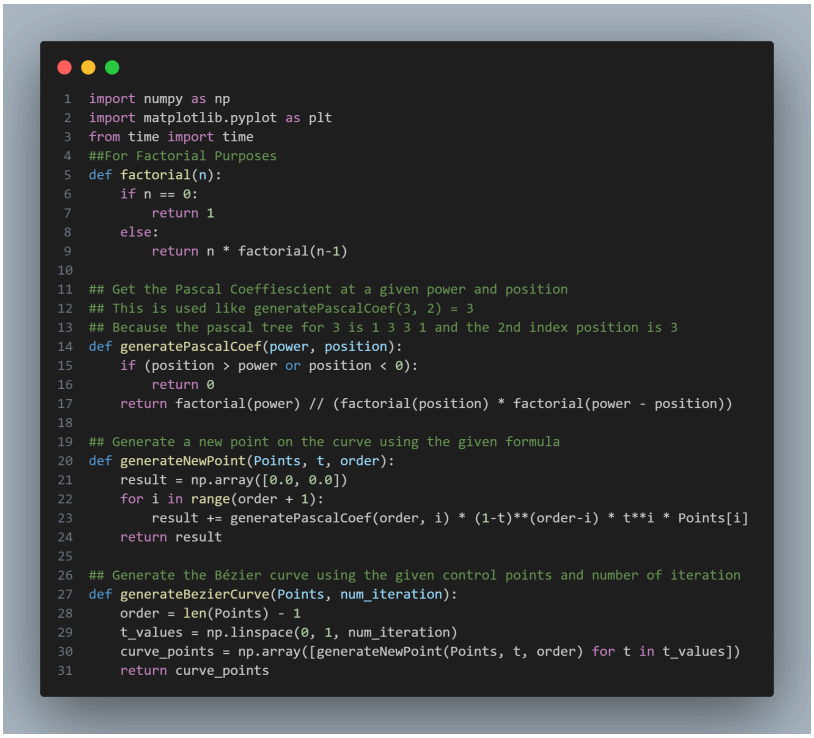
Dari situ, hasil ekspansi dikalikan dengan setiap titik kontrol mulai dari P_0 hingga P_{n-1} .

Algoritma brute force yang dimaksud disini adalah penggunaan secara naif formula yang diberikan untuk membangun sebuah kurva Bézier dengan nilai t yang berubah-rubah sesuai dengan interval perubahan nilai t yang ditentukan. Untuk memperoleh perbandingan yang adil antara algoritma brute force dengan *divide and conquer*, kita harus menentukan berapa nilai t yang harus dihasilkan untuk setiap variasi iterasi. Perhatikan bahwa pada algoritma *divide and conquer*, titik yang dihasilkan mengikuti pola berikut:

Tabel 1. Perbandingan titik yang dihasilkan dengan jumlah iterasi

Jumlah iterasi	Total titik yang dihasilkan
1	1
2	3
3	7
4	15
n	2^n - 1

Maka dari itu, untuk memperoleh perbandingan yang *fair* dengan jumlah iterasi yang sama, interval t yang digunakan adalah $\frac{1}{2^{banyak\ iterasi}}$. Misalkan pada jumlah iterasi 4, maka akan digunakan t yang berubah-rubah dengan interval $\frac{1}{16}$ dari 0 hingga 1 sehingga dihasilkan 15 total titik (terkecuali titik kontrol paling awal dan akhir).



```python
import numpy as np
import matplotlib.pyplot as plt
from time import time
##For Factorial Purposes
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

## Get the Pascal Coeffiescient at a given power and position
## This is used like generatePascalCoef(3, 2) = 3
## Because the pascal tree for 3 is 1 3 3 1 and the 2nd index position is 3
def generatePascalCoef(power, position):
    if (position > power or position < 0):
        return 0
    return factorial(power) // (factorial(position) * factorial(power - position))

## Generate a new point on the curve using the given formula
def generateNewPoint(Points, t, order):
    result = np.array([0.0, 0.0])
    for i in range(order + 1):
        result += generatePascalCoef(order, i) * (1-t)**(order-i) * t**i * Points[i]
    return result

## Generate the Bézier curve using the given control points and number of iteration
def generateBezierCurve(Points, num_iteration):
    order = len(Points) - 1
    t_values = np.linspace(0, 1, num_iteration)
    curve_points = np.array([generateNewPoint(Points, t, order) for t in t_values])
    return curve_points
```

Gambar 2. Implementasi pembuatan Kurva Bézier dalam python

Fungsi utama yang digunakan untuk membuat Kurva Bézier adalah *generateBezierCurve* yang menerima parameter input berupa Points (titik kontrol) serta num_iteration (jumlah iterasi). Dari situ dilakukan iterasi nilai t sesuai dengan penjelasan interval yang telah dilakukan sebelumnya dan didapatkan titik-titik yang merepresentasikan Kurva Bézier.

Misalkan jumlah iterasi adalah i dan jumlah titik kontrol adalah n. Untuk algoritma *bruteforce* dengan suatu titik yang sudah diketahui, misalkan di titik n = 3. Maka algoritma ini akan berjalan dengan dengan

$$T(i) = (2^i - 1)$$

Atau

$$O(2^i)$$

Sebab, dilakukan perkalian dengan “konstanta” yang selalu sama yaitu suku (1-t) t serta koefisien paskal di setiap sukunya.

Untuk kasus yang lebih general, yaitu ketika n sembarang, maka perlu ada beberapa asumsi yang dibuat. Dengan asumsi bahwa operasi perpangkatan dianggap memiliki kompleksitas algoritma *log n* serta dengan asumsi bahwa nantinya koefisien paskal sudah diketahui, perhatikan bahwa dari setiap suku pada formula yang ada, kita melakukan 2 kali operasi perpangkatan yaitu pada suku (1-t) serta t itu sendiri dengan

jumlah hasil pangkat adalah n. Lalu kita melakukan operasi penjumlahan sebanyak n kali. Dari sini, dapat disimpulkan bahwa setiap perhitungan nilai t memiliki kompleksitas waktu

$$T(n) = n\ (log\ n)$$

Karena program akan memanggil pencarian titik sebanyak 2^i -1 kali, maka kompleksitas algoritmanya akan menjadi

$$T(n,i) = (2^i - 1)(n\ (log\ n))$$

Sehingga notai Big O menjadi

$$O\ ((2^i)(n\ (log\ n)))$$

3. Analisis dan Implementasi dengan Algoritma Divide and Conquer

Implementasi algoritma divide and conquer untuk membentuk Kurva Bézier pada dasarnya terbatas pada 3 titik kontrol. Ketiga titik kontrol ini dapat dianggap sebagai titik awal, titik tengah, dan titik akhir. Proses awal untuk membentuk Kurva Bézier adalah dengan mencari seluruh titik tengah dari garis yang dibentuk pada titik yang bersebelahan. Selanjutnya, dari dua titik yang terbentuk dicari kembali titik tengah dari garis yang dibentuk pada kedua titik tersebut. Hasil dari langkah ini akan menghasilkan *midpoint* yang merupakan hasil akhir yang diinginkan untuk membentuk Kurva Bézier. Untuk iterasi selanjutnya, pembentukan Kurva Bézier dilanjutkan pada bagian "kiri" dan bagian "kanan" dari *midpoint* tersebut yang telah terbentuk dua buah upa persoalan baru. Langkah ini dilakukan seterusnya sebanyak i iterasi. Semakin banyak iterasi, semakin banyak *midpoint* yang dihasilkan dan Kurva Bézier yang dihasilkan akan terlihat semakin halus.

Jumlah *midpoint* yang dihasilkan dari implementasi dengan algoritma divide and conquer bergantung pada banyaknya iterasi yang dilakukan. Setiap penambahan iterasi, akan bertambah sebanyak 2^{i-1} *midpoint* dari iterasi sebelumnya dengan i adalah banyaknya iterasi. Hal ini berarti bahwa banyaknya *midpoint* yang dihasilkan untuk i iterasi adalah sebanyak $2^i - 1$ seperti pada tabel 1. Karena fungsi calculate_mid_point merupakan operasi aritmatika biasa, maka kompleksitas waktunya adalah O(1). Fungsi generate_bezier_curve merupakan *entry point* dari program, maka kompleksitas waktunya juga O(1). Fungsi fill_bezier_points akan memanggil dirinya sebanyak dua kali dengan banyaknya langkah yang bertambah hingga mencapai batas maksimal iterasi. Sesuai dengan hasil *midpoint* yang dihasilkan, maka kompleksitas fungsi ini adalah $O(2^i)$ dengan i adalah banyaknya iterasi. Maka kompleksitas waktu gabungan dari algoritma divide and conquer untuk 3 titik kontrol adalah $O(2^i)$ dengan i adalah banyaknya iterasi.

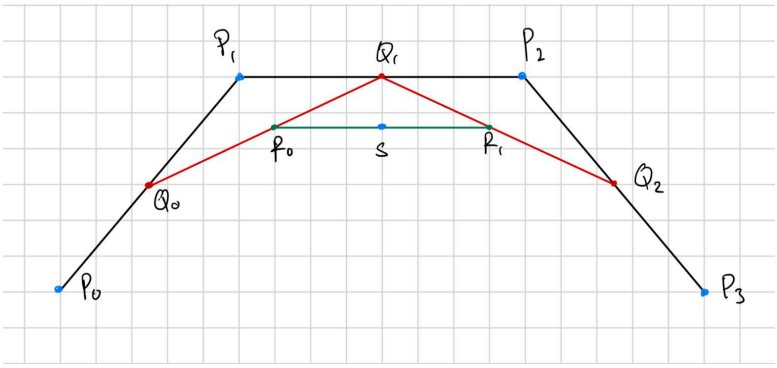
```python
## Divide and Conquer to generate Bezier curve
def fill_bezier_points(point1, point2, point3, current_step):
    global total_steps
    if current_step < total_steps:
        middle_point1 = calculate_mid_point(point1, point2)
        middle_point2 = calculate_mid_point(point2, point3)
        middle_point3 = calculate_mid_point(middle_point1, middle_point2)
        current_step += 1
        fill_bezier_points(point1, middle_point1, middle_point3, current_step)
        bezier_points.append(middle_point3)
        fill_bezier_points(middle_point3, middle_point2, point3, current_step)

## Calculate the middle point between two points
def calculate_mid_point(point1, point2):
    return ((point1[0] + point2[0]) / 2, (point1[1] + point2[1]) / 2)

## Generate the Bezier curve using the given control points
def generate_bezier_curve(point1, point2, point3):
    global bezier_points
    bezier_points = []
    bezier_points.append(point1)
    fill_bezier_points(point1, point2, point3, 0)
    bezier_points.append(point3)
```

Gambar 3. Implementasi pembuatan Kurva Bézier dengan algoritma Divide and Conquer

4. Analisis dan Implementasi Bonus

Untuk membuat Kurva Bézier dengan titik kontrol sembarang menggunakan algoritma *divide* and conquer, maka perlu dilakukan generalisasi dan pemecahan masalah secara umum. Titik tengah dapat dianggap sebagai sebuah *Root* dari *Binary Tree* yang dimana setiap titik tengah tersebut akan memiliki persis 2 anak lainnya atau tidak sama sekali. Algoritma *divide and conquer* dilakukan dengan memecah suatu titik tengah yang masih memiliki "utang" jumlah iterasi lebih dari 1 untuk kemudian diselesaikan secara masing-masing.

Langkah pembentukan Kurva Bézier dengan titik kontrol sembarang memiliki pendekatan yang sama seperti pembentukan Kurva Bézier dengan 3 titik. Pencarian *midpoint* untuk titik kontrol sembarang dilakukan dengan metode yang sama, hanya saja dilakukan lebih banyak untuk titik kontrol yang lebih banyak. Setelah ditemukan *midpoint* pertama, permasalahan dapat dijadikan menjadi dua upa persoalan. Kedua upa persoalan tersebut akan menghasilkan upa persoalan lagi yang kedalamannya sebanyak jumlah iterasinya. Persoalan ini dapat diselesaikan dengan relasi rekurens dengan basis ketika iterasinya 1 dan di dalam rekurensnya akan memanggil fungsi itu sendiri sebanyak dua kali, yaitu pada upa persoalan "kiri" dan upa persoalan "kanan". Ilustrasi untuk relasi rekurens tersebut dapat dilihat pada gambar di bawah ini.



Gambar 4. Ilustrasi persoalan Kurva Bézier untuk titik kontrol sembarang

Dua upa persoalan baru yang muncul setelah iterasi pertama adalah (P0, Q0, R0, S) dan (S, R1, Q2, P3) yang masing-masing memiliki 4 titik kontrol. Relasi rekurens ini dilakukan sebanyak jumlah iterasi.

Jumlah *midpoint* akhir yang dihasilkan pada metode ini sama seperti jumlah *midpoint* pada 3 titik kontrol, yaitu $2^i - 1$ dengan i adalah banyaknya iterasi, sehingga kompleksitas waktunya adalah O(2^i). Jumlah titik tengah perantara yang dihasilkan untuk setiap upa persoalan adalah $\frac{n(n-1)}{2}$ dengan n adalah jumlah titik kontrol, sehingga kompleksitas waktunya adalah O(n^2). Maka kompleksitas waktu gabungan dari algoritma divide and conquer untuk titik kontrol sembarang adalah O($2^i(n^2)$) dengan i adalah banyaknya iterasi.

5. Source Code Program

```python
import numpy as np
from bruteforce.bruteforce import plotBezierCurveBruteForce
from dnc.dnc_animate import plotBezierCurveDnC
from utils.utils import read_control_points_from_file
from time import time

def main():
    try:
        # Input control points from a file
        print("1. Metode Brute Force")
        print("2. Metode Divide and Conquer")
        method = input ("Masukkan metode yang ingin digunakan (1/2): ")
        while (method != "1" and method != "2"):
            print("Masukkan metode yang valid")
            method = input ("Masukkan metode yang ingin digunakan (1/2): ")

        filename = input("Masukkan nama file dengan konfigurasi terkait: ")
        Points = read_control_points_from_file(filename)
        if Points is None:
            return
        # Input number of samples
        if (method == "1"):
            print()
            print("Untuk metode brute force, jumlah nilai t akan disamakan dengan metode divide and conquer. \nMisal 3 iterasi akan menggunakan 2^3 - 1 = 7 variansi nilai t, diluar kecuali titik control awal")
            print()
        num_samples = int(input("Masukkan jumlah iterasi: "))


        if method == "1":
            num_samples = pow(2,num_samples) + 1
            plotBezierCurveBruteForce(Points, num_samples)
        else:
            displaymethod = input("Apakah anda ingin menampilkan animasi? (y/n): ")
            while (displaymethod != "y" and displaymethod != "n"):
                print("Masukkan y atau n")
                displaymethod = input("Apakah anda ingin menampilkan animasi? (y/n): ")
            if displaymethod == "y":
                interval = float(input("Masukkan interval waktu: "))
                plotBezierCurveDnC(Points, num_samples, interval, 1)
            else:
                plotBezierCurveDnC(Points, num_samples, 0, 2)
        # Plot the Bézier curve
    except ValueError as ve:
        print("Error:", ve)
        print("Please enter valid numeric values for coordinates and number of samples.")

if __name__ == "__main__":

    main()
```

Gambar 5. Ilustrasi program utama

Pada program ini terdapat beberapa validasi sederhana untuk meminta pengguna memasukkan input file txt dan selanjutnya memilih metode yang ingin digunakan serta jumlah iterasi yang ingin dilakukan. Terdapat pula pemilihan untuk menentukan apakah ingin dilakukan animasi terhadap hasil kurva atau tidak.

```python
from dataclasses import dataclass

@dataclass
class Point:
    x: float
    y: float

# Function to calculate mid point between two points
def calculate_mid_point(point1: Point, point2: Point) -> tuple[float, float]:
    return ((point1.x + point2.y) / 2, (point1.x + point2.y) / 2)


# Function to generate all mid points from the given control points
def search_midpoint(points: list[Point], result: list[Point]) -> None:
    if len(points) == 1:
        return
    else:
        new_points = []
        for i in range(len(points) - 1):
            new_points.append(Point((points[i].x + points[i + 1].x) / 2, (points[i].y + points[i + 1].y) / 2))
        result.extend(new_points)
        search_midpoint(new_points, result)


# Function to find the left side control points
def left_control_points(result: list[Point], control_points: list[Point], n: int) -> list[Point]:
    left = [control_points[0]]
    i = 0
    j = 0
    while i < (n - 1):
        left.append(result[j])
        j += n - i - 1
        i += 1
    return left

# Function to find the right side control points
def right_control_points(result: list[Point], control_points: list[Point], n: int) -> list[Point]:
    right = []
    i = 0
    j = len(result) - 1
    while i < (n - 1):
        right.append(result[j])
        j -= (i + 1)
        i += 1
    right.append(control_points[-1])
    return right


def create_bezier_points(control_points: list[Point], result_points: list[Point], mid_points: list[Point], iterations: int, n: int) -> None:
    if iterations == 1:
        result = []
        search_midpoint(control_points, result)
        result_points.append(result)
        mid_points.append(result[-1])
        return
    else:
        result = []
        search_midpoint(control_points, result)
        result_points.append(result)

        left = left_control_points(result, control_points, n)
        create_bezier_points(left, result_points, mid_points, iterations - 1, n)

        mid_points.append(result[-1])

        right = right_control_points(result, control_points, n)
        create_bezier_points(right, result_points, mid_points, iterations - 1, n)
```

Gambar 6. Kode Program untuk *Divide and Conquer* umum

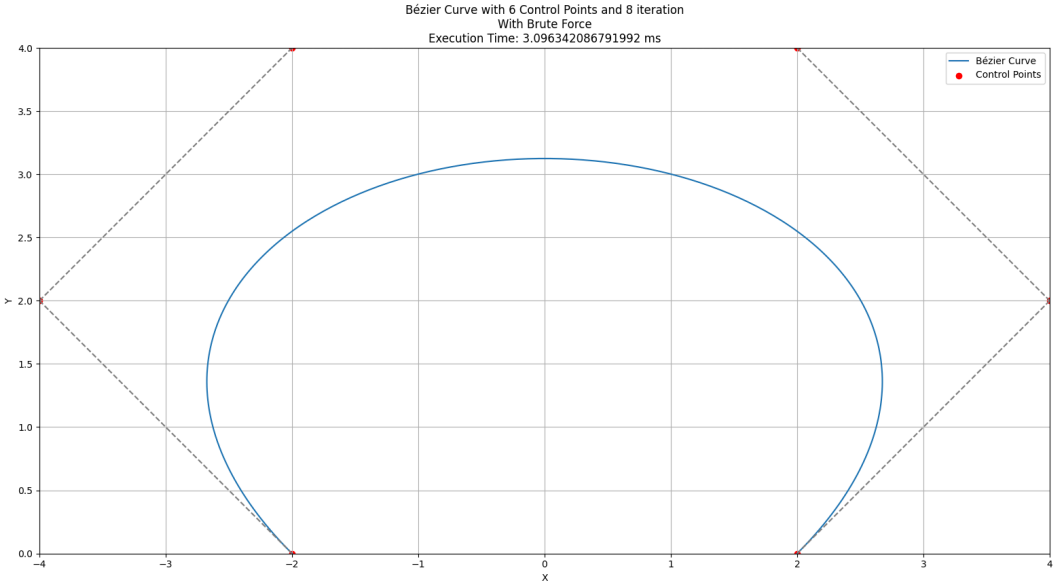
Pemanggilan utama program dilakukan di fungsi *create_bezier_points* yang menerima parameter input serta beberapa parameter output yang digunakan untuk melakukan visualisasi. Fungsi *right_control_points* digunakan untuk mendapatkan seluruh titik tengah di bagian kanan dari masalah yang sedang dipecah, sedangkat fungsi *left_control_points* digunakan untuk mendapatkan semua titik tengah di bagian kiri dari masalah yang sedang dipecahkan. Sedangkan, fungsi *search_midpoint* digunakan untuk mendapatkan seluruh *midpoint* yang dihasilkan baik

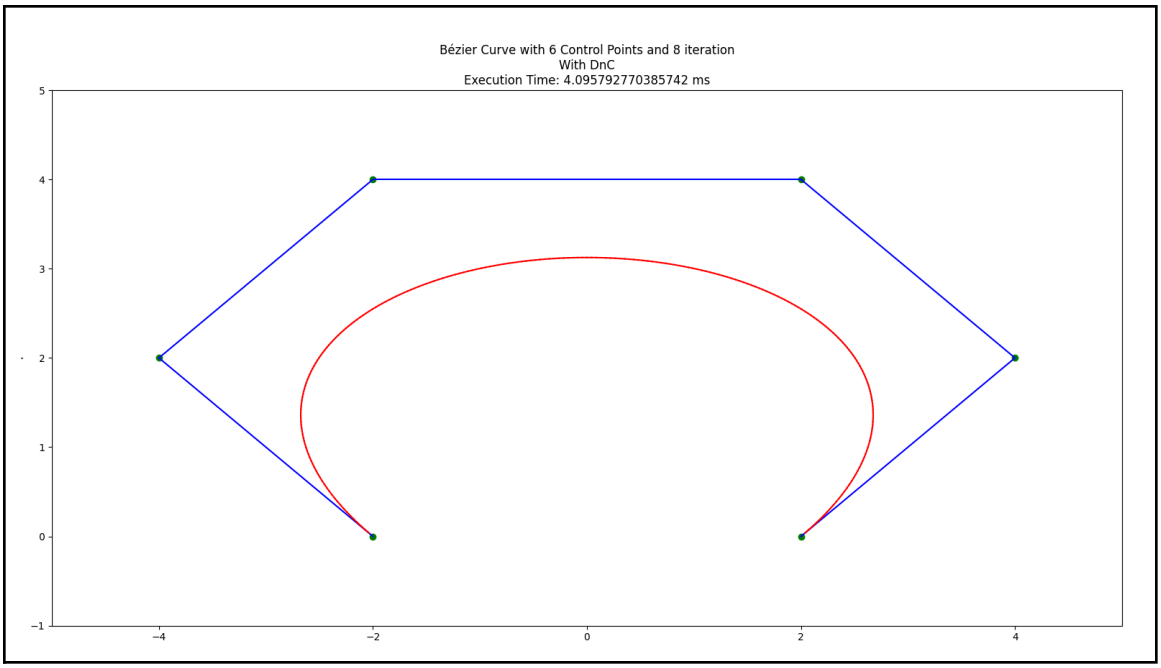
sebagai pembantu untuk mendapat *midpoint* yang nantinya akan dipakai di program, maupun *midpoint* utama (akan berada di indeks terakhir).

6. Uji Kasus

UJI KASUS 1

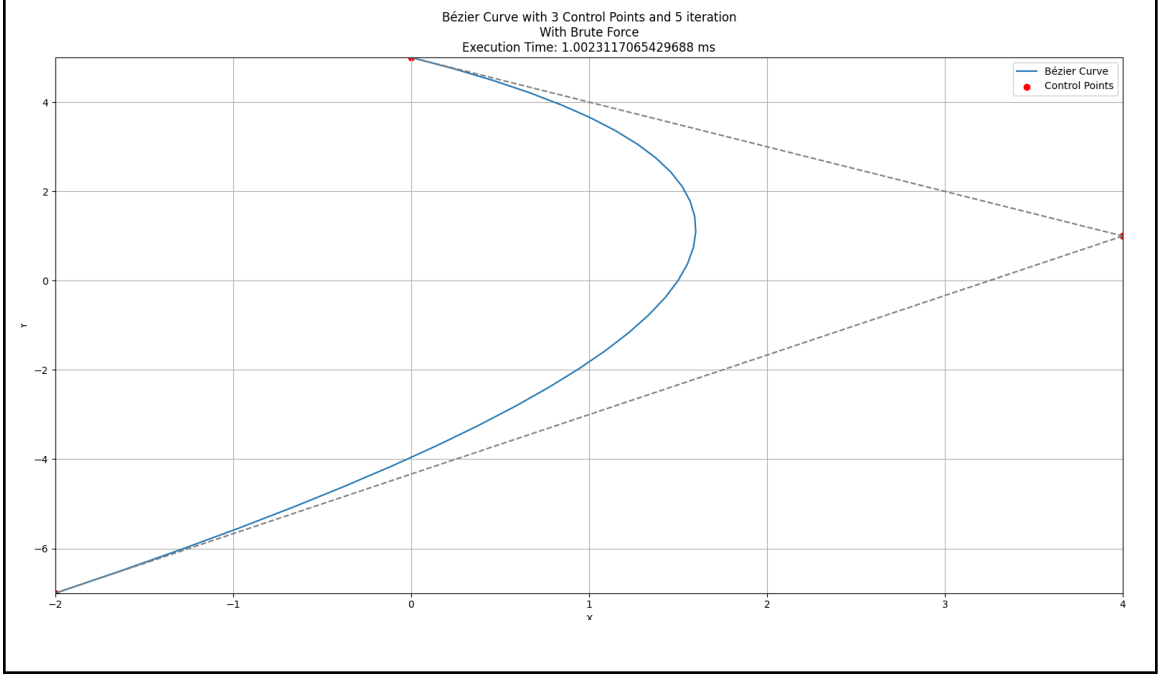
Titik Input	Jumlah Iterasi
6 -2 0 -4 2 -2 4 2 4 4 2 2 0	8

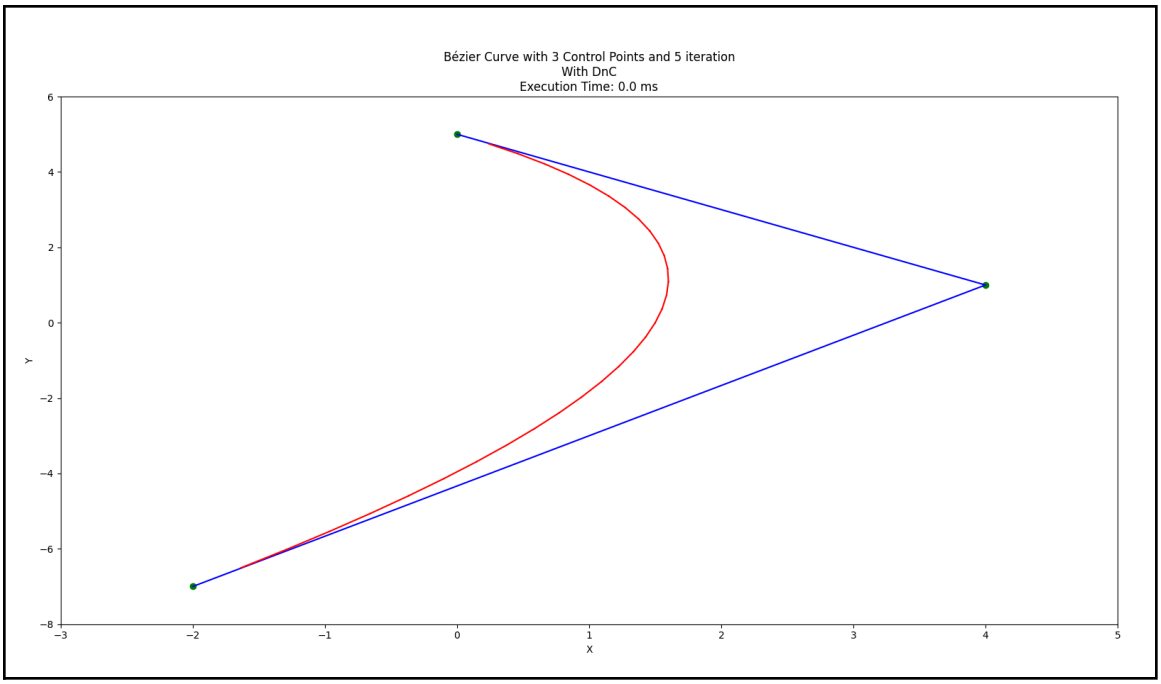




UJI KASUS 2

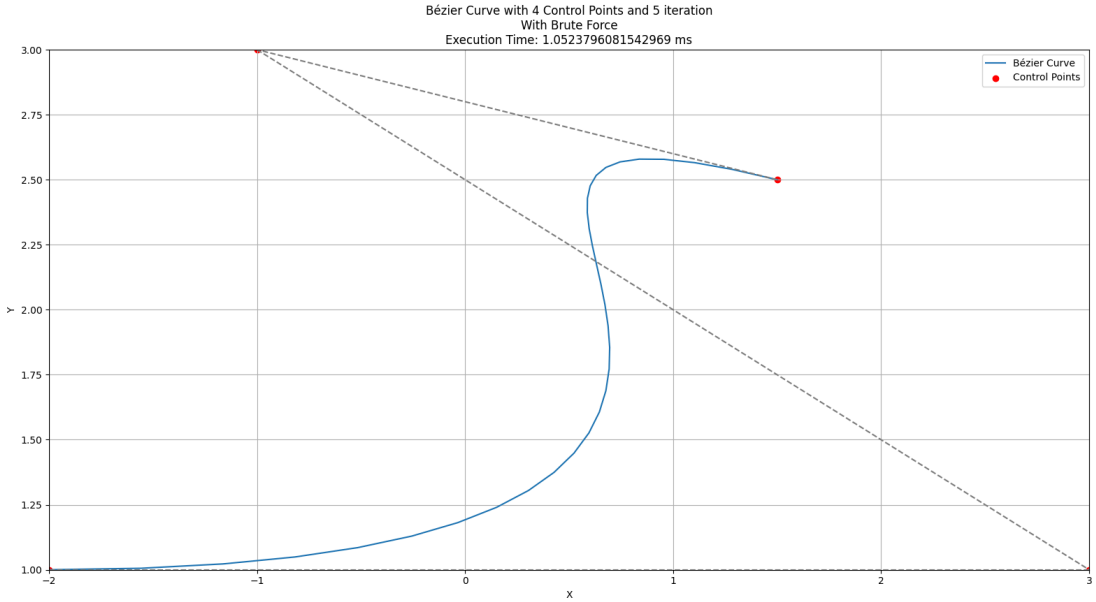
Titik Input	Jumlah Iterasi
3 0 5 4 1 -2 -7	5

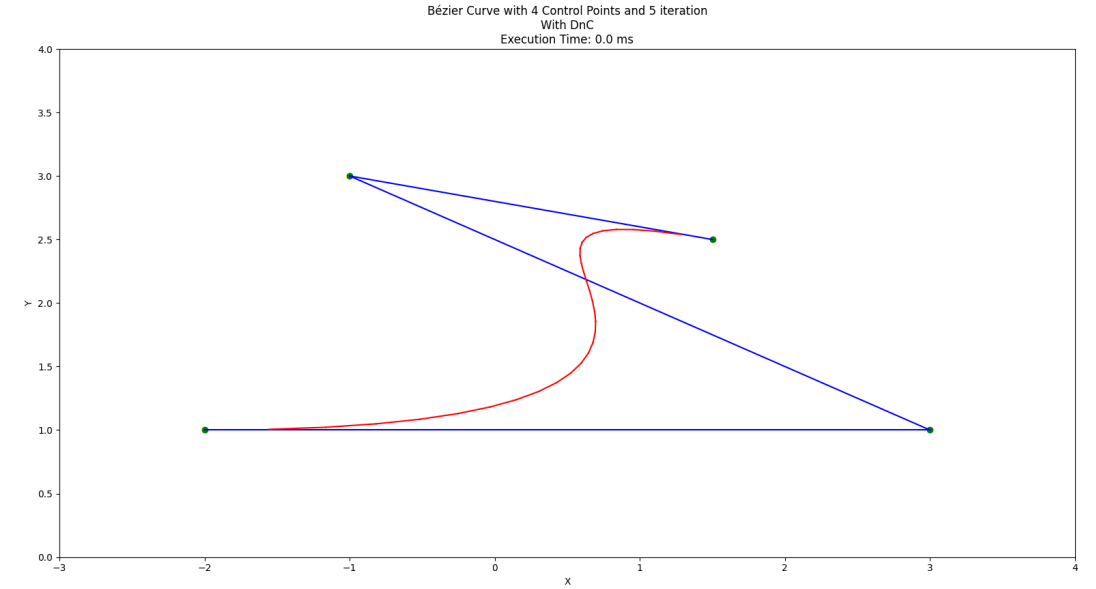




UJI KASUS 3

Titik Input	Jumlah Iterasi
4 1.5 2.5 -1 3 3 1 -2 1	5

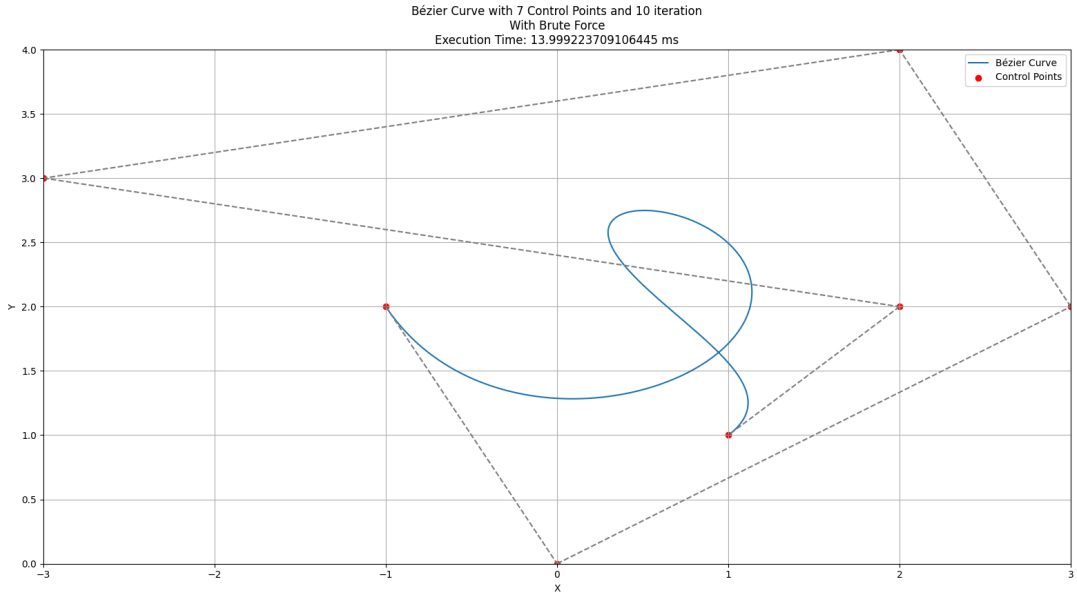


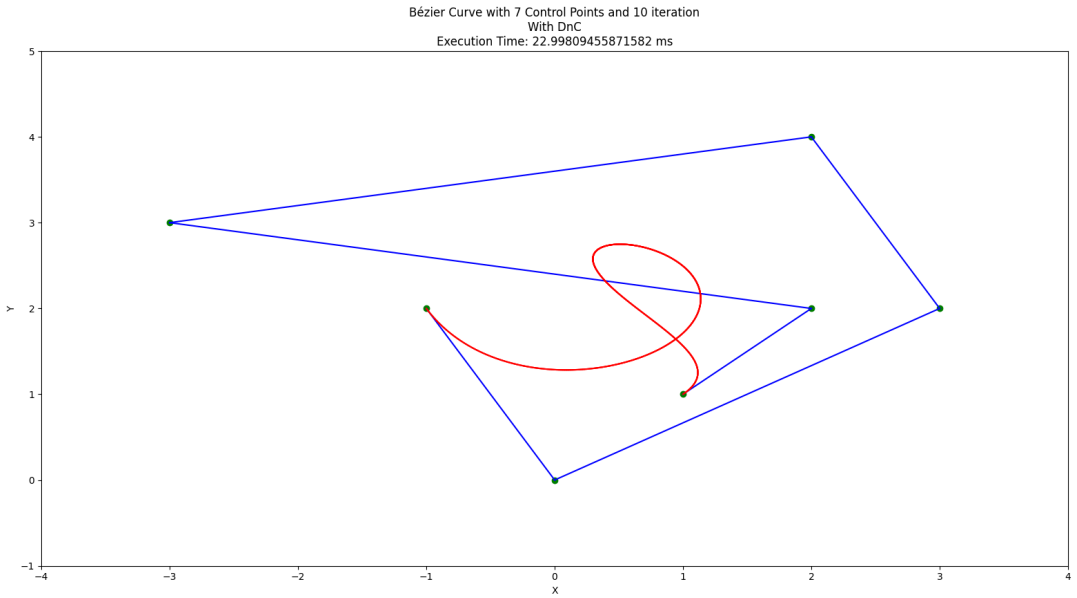


UJI KASUS 4

	Jumlah Iterasi
7 -1 2 0 0 3 2 2 4	10

-3 3 2 2 1 1	

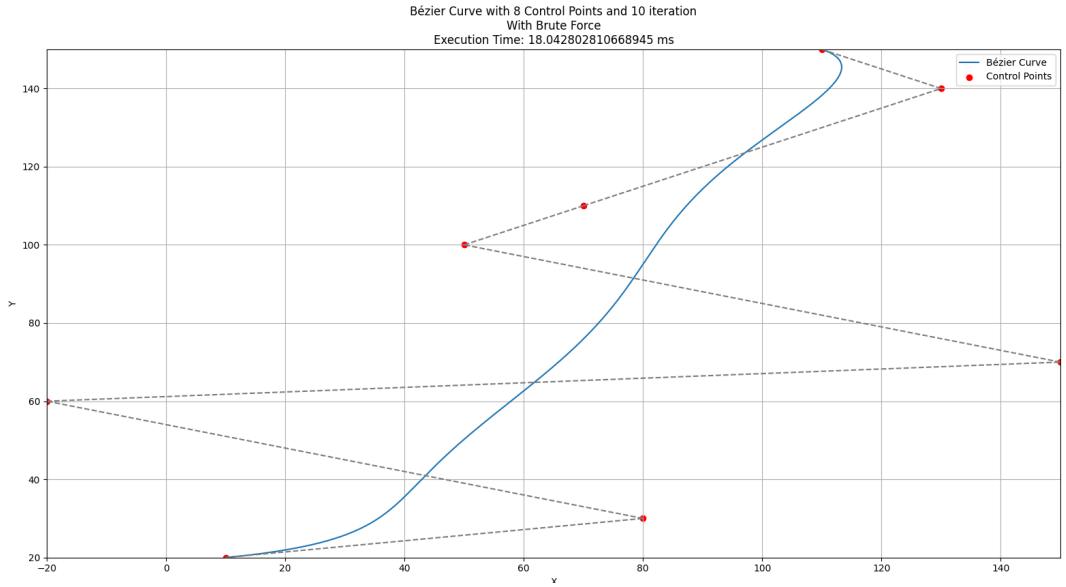


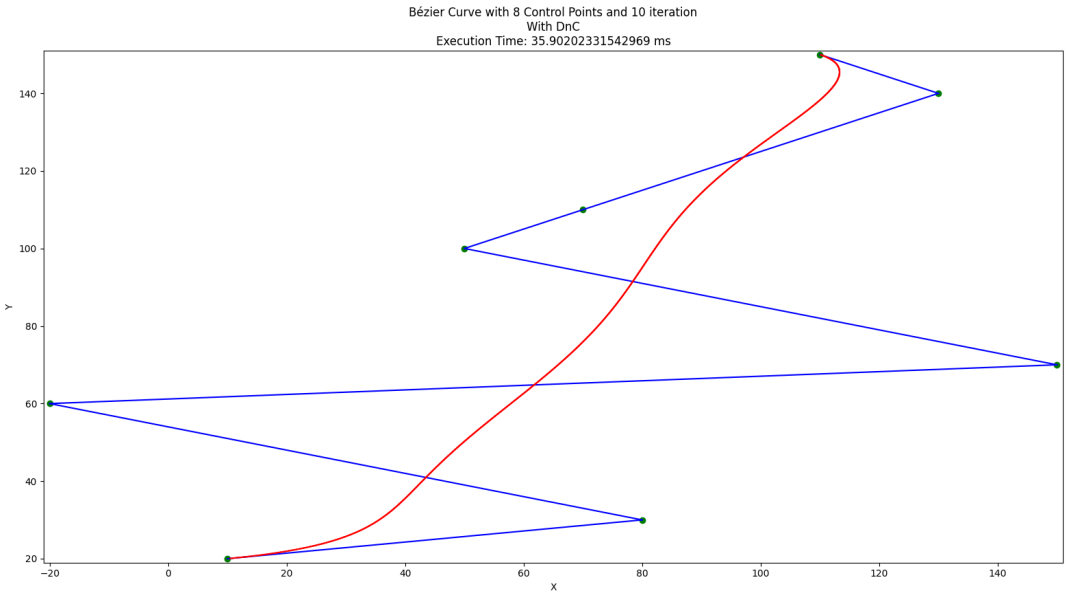


UJI KASUS 5

Titik Input	Jumlah Iterasi
8	10

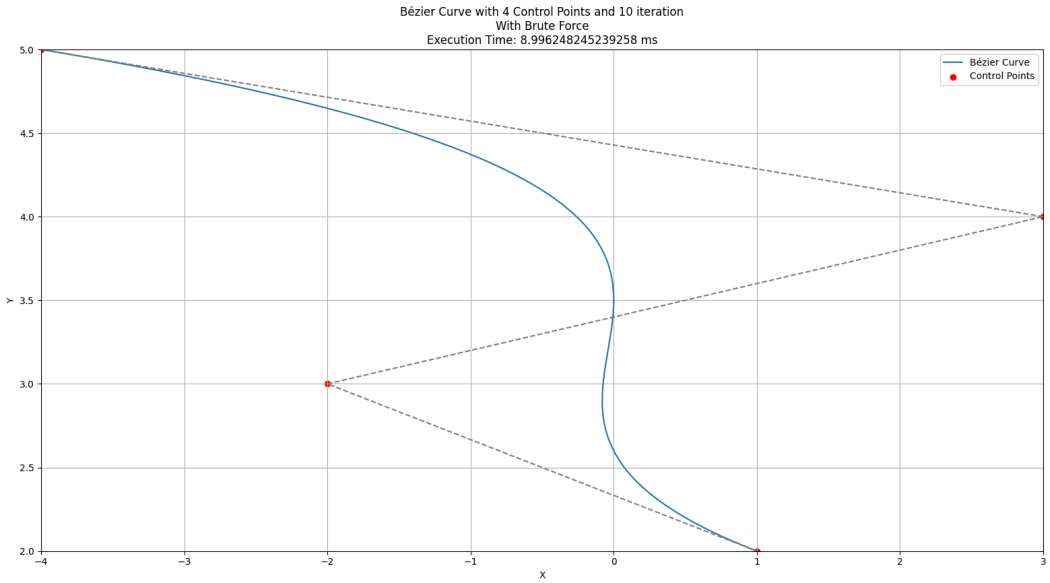
10 20
80 30
-20 60
150 70
50 100
70 110
130 140
110 150

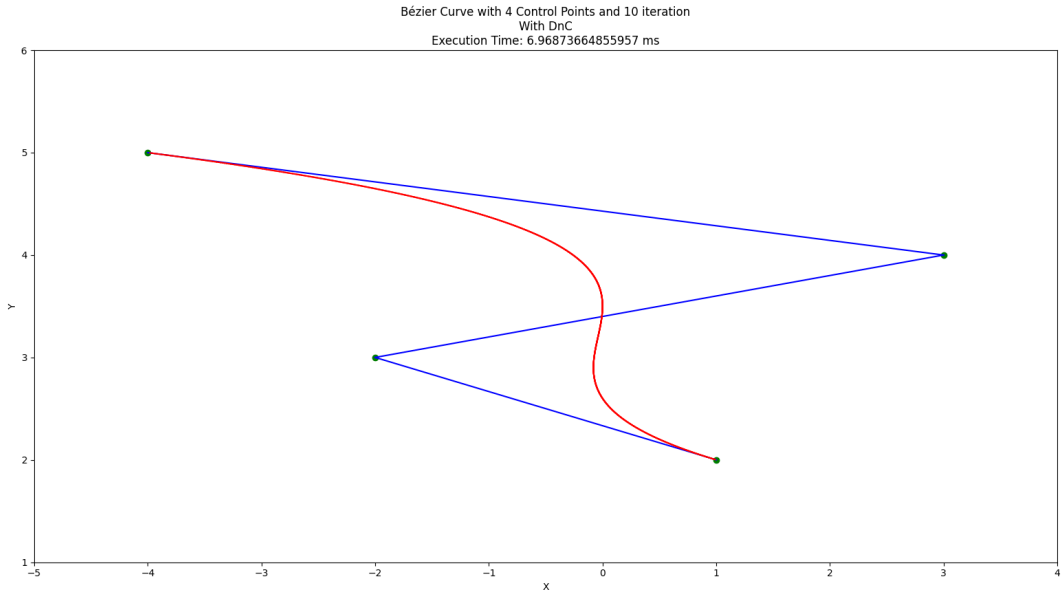




UJI KASUS 6

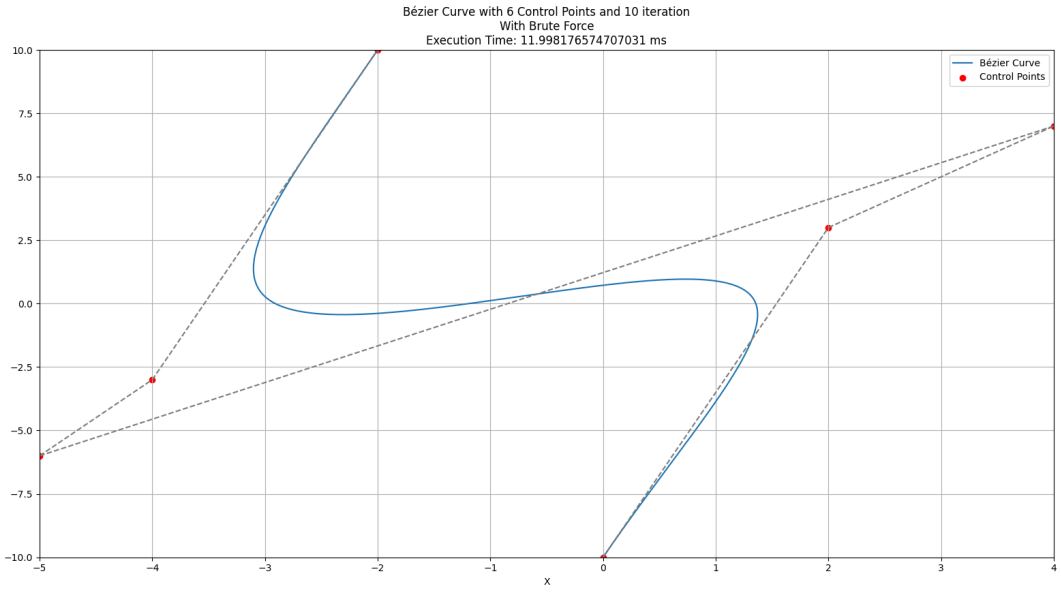
	Jumlah Iterasi
4 1 2 -2 3 3 4 -4 5	10

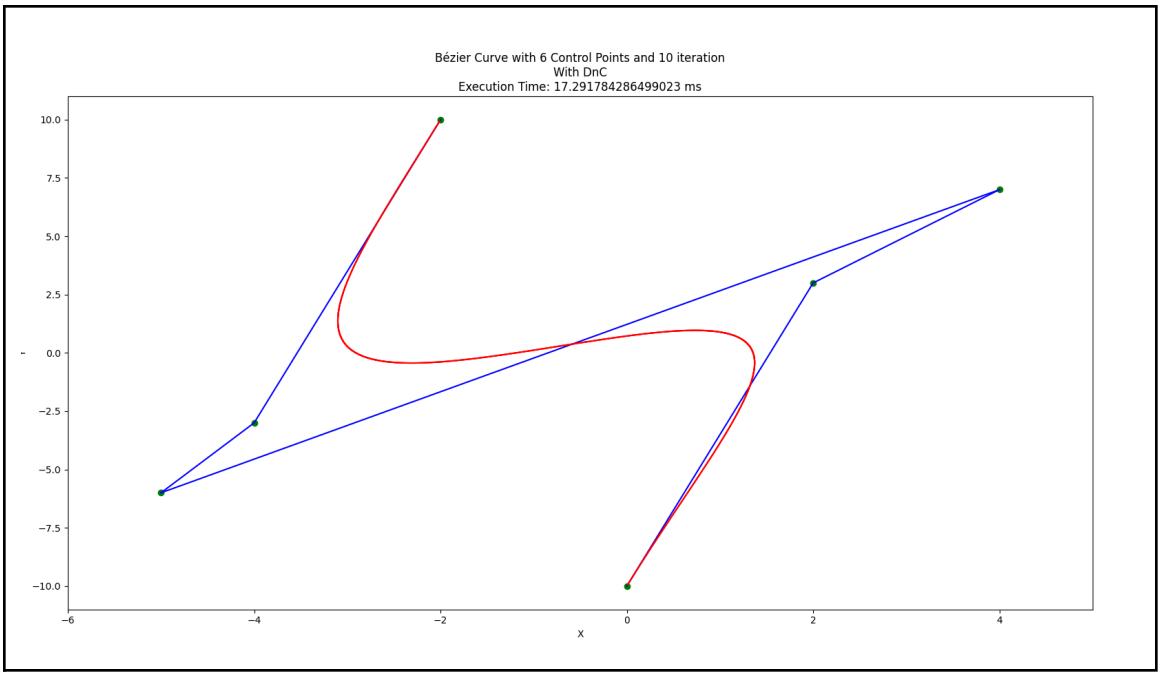




UJI KASUS 7

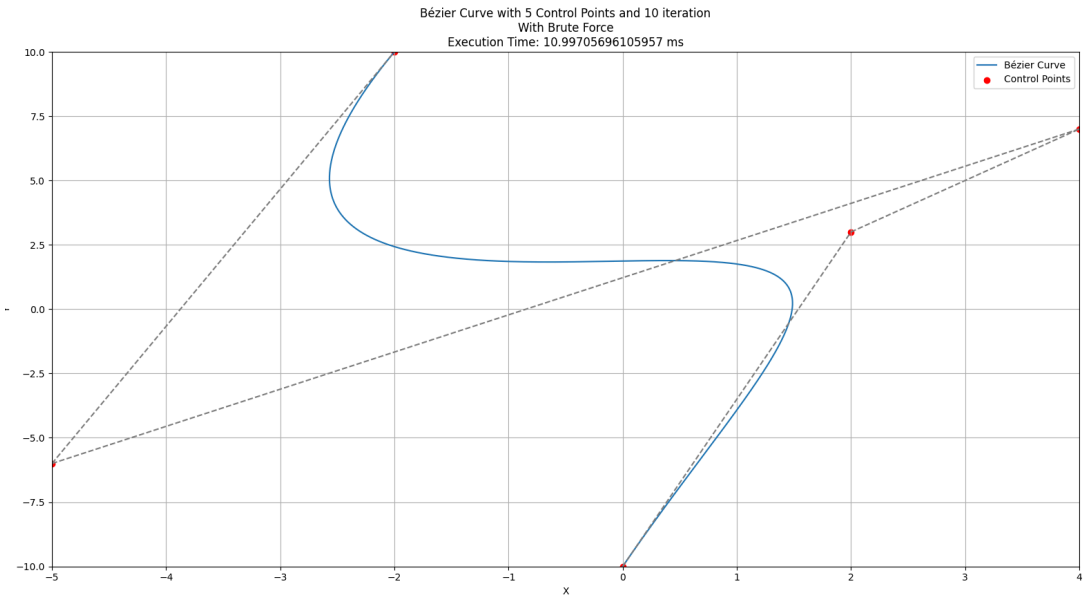
Titik Input	Jumlah Iterasi
6 0 -10 2 3 4 7 -5 -6 -4 -3 -2 10	10

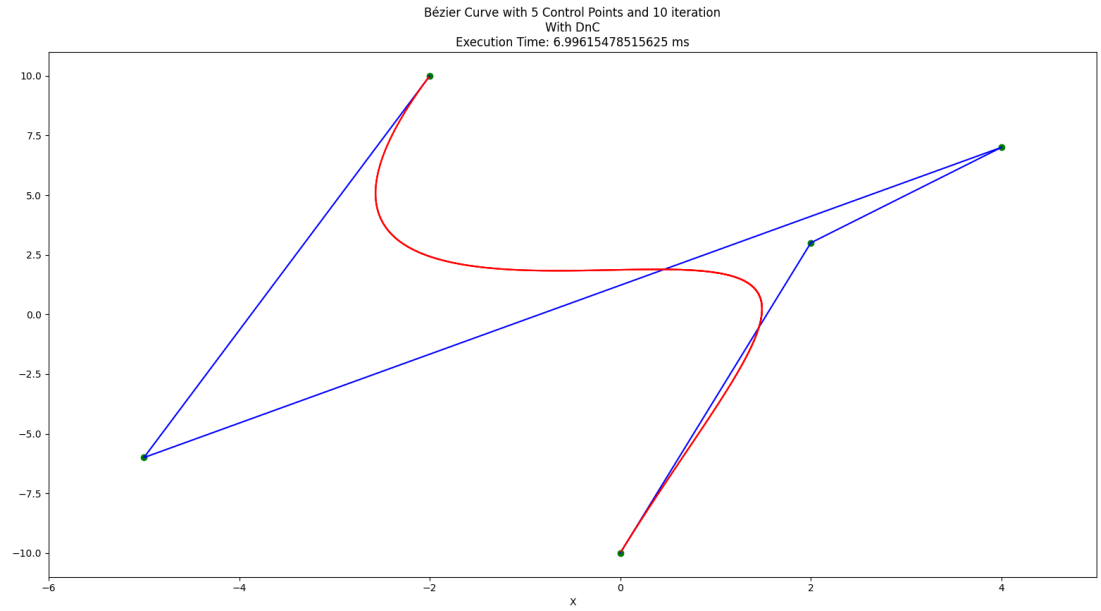




UJI KASUS 8

	Jumlah Iterasi
5 0 -10 2 3 4 7 -5 -6 -2 10	10

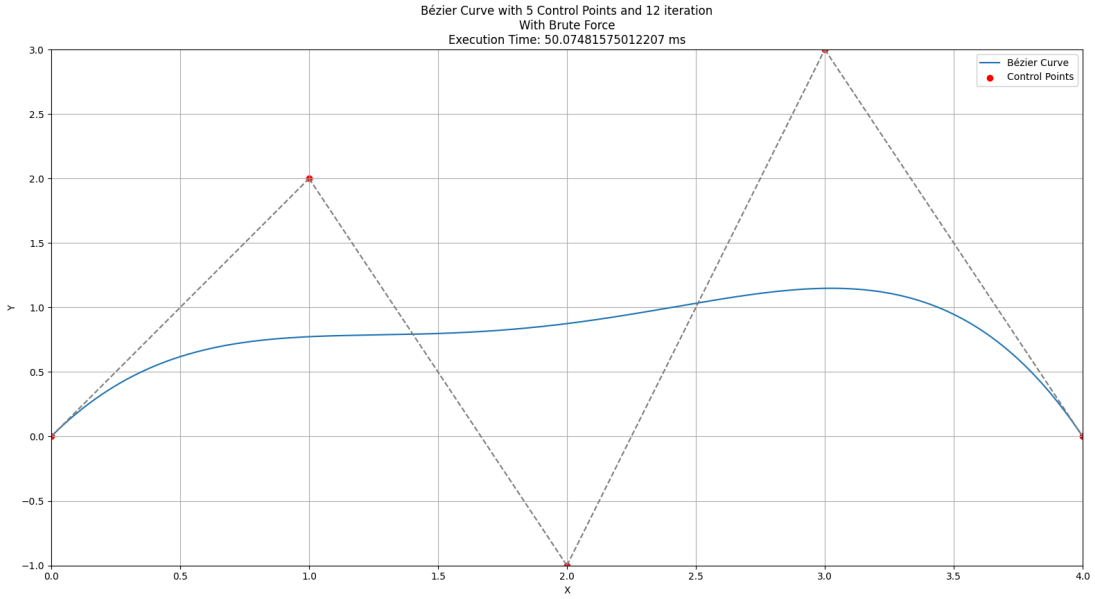


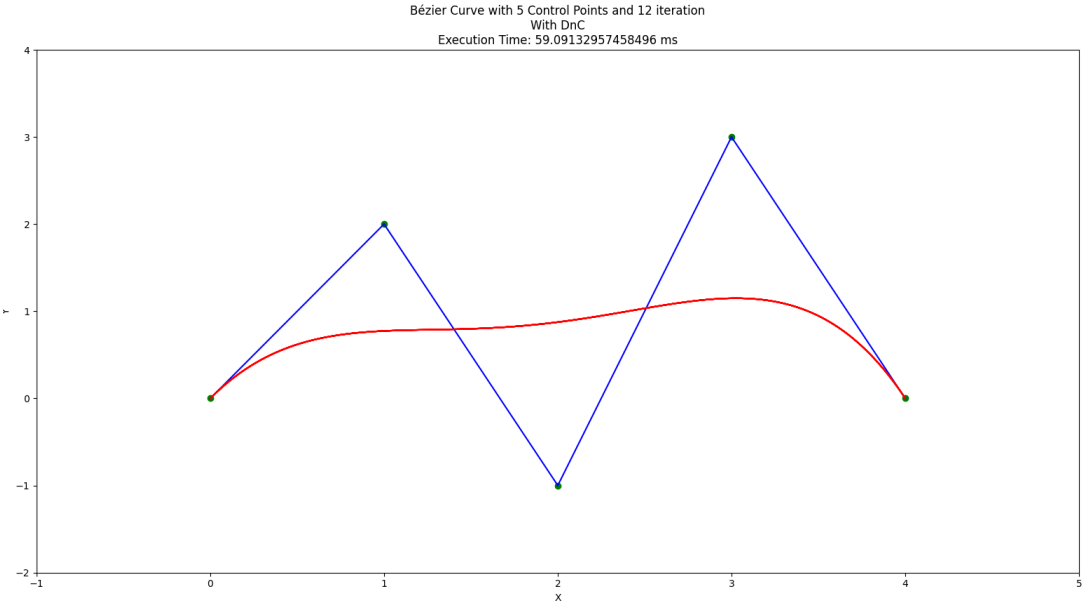


UJI KASUS 9

Titik Input	Jumlah Iterasi
5 0 0 1 2 2 -1 3 3	12

4 0	

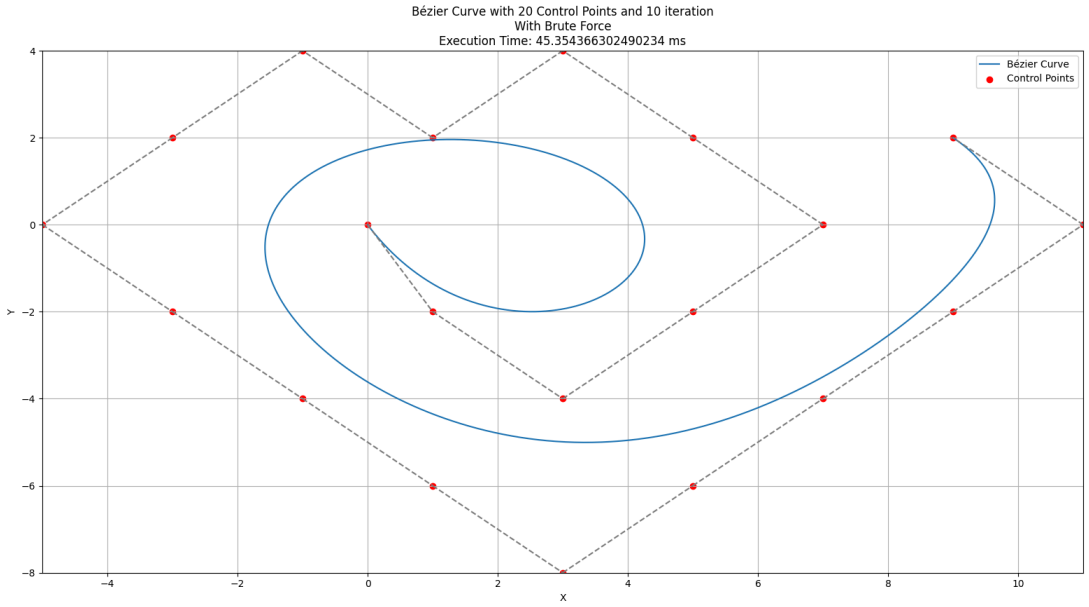


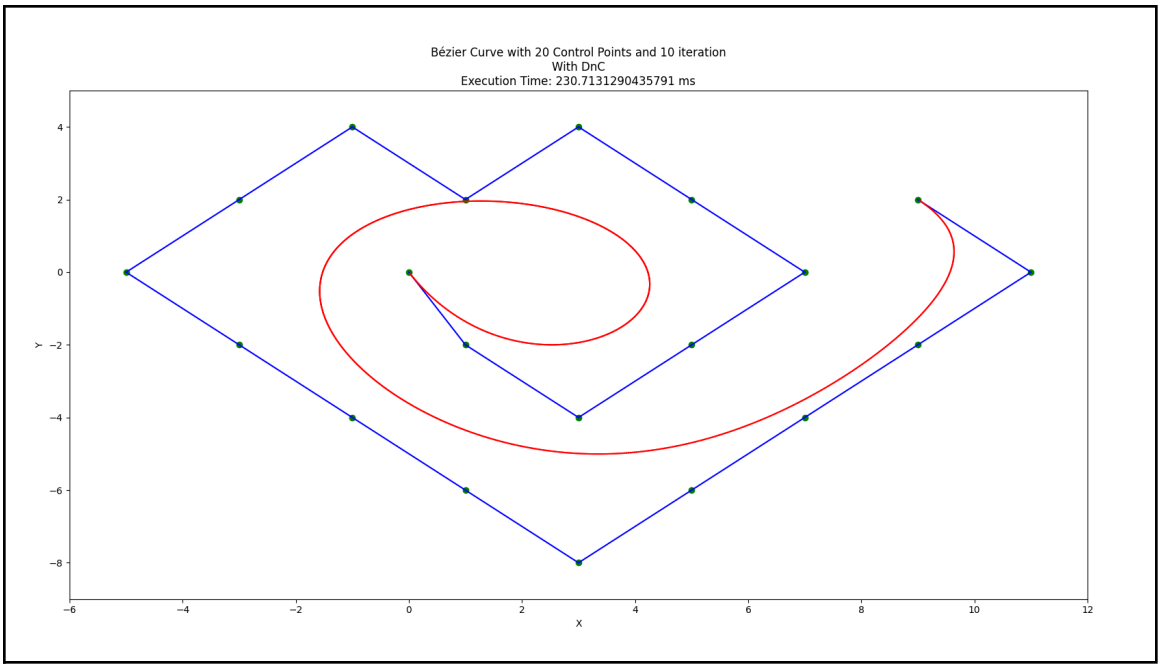


UJI KASUS 10

	Jumlah Iterasi
20 0 0 1 -2	10

3 -4
5 -2
7 0
5 2
3 4
1 2
-1 4
-3 2
-5 0
-3 -2
-1 -4
1 -6
3 -8
5 -6
7 -4
9 -2
11 0
9 2





7. Analisis Perbandingan Algoritma Brute Force dan Divide and Conquer

Tabel 2. Perbandingan waktu eksekusi algoritma brute force dan algoritma divide and conquer

Jumlah Iterasi	Jumlah Titik	Brute Force (ms)	DnC (ms)
8	6	3.09634	4.0957
5	3	0.0	0.0
5	4	1.005	0.0
10	7	13.996	22.99
10	8	18.042	35.902
10	4	8.996	6.968
10	6	11.998	17.291
10	5	10.997	6.996
12	5	50.074	59.091
10	20	45.354	230.713

Berdasarkan tabel di atas, dapat ditarik kesimpulan bahwa untuk jumlah titik kurang dari 6 dan jumlah iterasi kurang dari 10, algoritma divide and conquer yang kami implementasikan berjalan lebih cepat dibandingkan algoritma brute force. Sebaliknya, untuk iterasi lebih dari 6

atau iterasi lebih dari 10, algoritma brute force berjalan lebih cepat dibandingkan algoritma divide and conquer dalam kasus-kasus dimana iterasi serta jumlah titik yang digunakan besar. Ini menjadi masuk akal sebab kompleksitas dari algoritma *brute force* dengan mengabaikan proses pencarian koefisien paskal adalah

$$O\,((2^{i})(n\,(\log n)))$$

Dengan i adalah iterasi dan n adalah banyak titik. Sedangkan, untuk algoritma *Divide and Conquer* seperti yang telah dibahas sebelumnya, memiliki kompleksitas waktu

$$O(2^{i}(n^{2}))$$

Dapat dilihat pada semua kasus dimana n memiliki nilai yang besar, algoritma ini selalu kalah cepat dari *brute force*. Namun, pada kasus-kasus dimana n relatif kecil, algoritma ini masih dapat menang dari *brute force*. Analisis utama yang dapat menjadi rasionalisasi dari terjadinya anomali tersebut adalah kompleksitas dari algoritma brute force yang mengabaikan proses pencarian *array of pascal coeficient* dilakukan. Pencarian ini memang dilakukan diluar iterasi, sehingga selalu bernilai konstan, namun dalam nilai n yang kecil, hal ini dapat berpengaruh besar sebab pencariannya dilakukan sekali, namun menggunakan operasi faktorial di dalamnya.

Belum lagi, untuk nilai milisecond yang sangat kecil, faktor-faktor luar seperti kondisi laptop menjadi sangat berpengaruh untuk beberapa ms tersebut. Itulah mengapa untuk nilai n yang dibawah 5, algoritma *brute force* beberapa kali lebih lambat dibandingkan *divide and conaquer*.

Lampiran

Link Github: https://github.com/ImanuelSG/Tucil2_13522005_13522058

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.	✓	