

**Laporan Tugas Kecil 1 IF2211 Strategi Algoritma
Semester II Tahun Akademik 2023/2024**

Pembuatan *Bezier Curve* Dengan Algoritma Divide And Conquer



Disusun Oleh:

Ibrahim Ihsan Rasyid 13522018

Edbert Eddyson Gunawan 13522039

K-01

**Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024**

Daftar Isi

Daftar Isi.....	2
1. Deskripsi Tugas.....	4
2. Algoritma.....	6
2.1 Pendekatan Divide and Conquer.....	6
2.2 Pendekatan Bruteforce.....	6
2.3 Analisis Algoritma Divide and Conquer.....	7
2.4 Analisis Algoritma Bruteforce.....	7
2.5 Bonus - Generalisasi Bezier Curve n Titik.....	7
2.6 Bonus - Animasi.....	8
3. Source Code.....	9
3.1 main.py.....	10
3.2 Point.py.....	11
3.3 InputClass.py.....	12
3.4 MainAlgorithm.py.....	15
3.5 Draw.py.....	18
4. How To Use.....	22
5. Test Case.....	22
5.1 Test Case 1.....	22
Input.....	22
Result.....	23
5.2 Test Case 2.....	24
Input.....	24
Result.....	24
5.3 Test Case 3.....	25
Input.....	25
Result.....	25
5.4 Test Case 4.....	26
Input.....	26
Result.....	26
5.5 Test Case 5.....	27
Input.....	27
Result.....	27
5.6 Test Case 6.....	28
Input.....	28
Result.....	28
5.7 Test Case 7.....	29
Input.....	29
Result.....	29
5.8 Test Case 8.....	30
Input.....	30

Result.....	30
5.9 Test Case 9.....	31
Input.....	31
Result.....	31
5.10 Test Case 10.....	32
Input.....	32
Result.....	32
Lampiran.....	33
GitHub.....	33
Poin-Poin Penting.....	33

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 realistik, membuat desain produk yang kompleks dan presisi, dan membuat font yang indah dan unik. Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan.

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

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.

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

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.

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

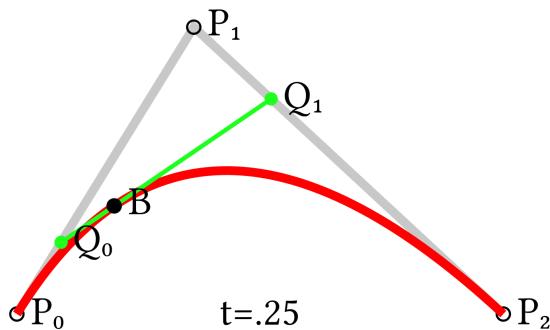
$$Q_1 = B(t) = (1 - t)P_1 + tP_2, \quad t \in [0, 1]$$

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

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

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

Berikut adalah ilustrasi dari kasus diatas.



Gambar 2. Pembentukan Kurva Bézier Kuadratik.

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

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.

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

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

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. Algoritma

Pada laporan ini, algoritma yang digunakan oleh penulis adalah *Divide and Conquer*.

2.1 Pendekatan *Divide and Conquer*

Spesifikasi wajib yang diberikan adalah untuk membentuk *Bezier Curve* kuadratik. Maka dari itu, masukan berupa senarai P yang terdiri dari 3 buah titik P_0, P_1, P_2 , dengan P_1 merupakan titik kontrol antara dan P_0, P_2 merupakan titik kontrol awal dan akhir, dan i merupakan jumlah iterasi yang diinginkan. Dengan asumsi masukan memenuhi *constraint* $i > 0$ dikarenakan jika iterasi bernilai 0 maka *Bezier Curve* kuadratik akan berupa garis lurus yang menghubungkan P_0, P_2 . Dengan memenuhi definisi permasalahan tersebut maka secara umum alur jalannya Algoritma *Divide and Conquer* untuk menyelesaikan permasalahan membentuk *Bezier Curve* kuadratik adalah sebagai berikut:

- a. Basis: saat iterasi = 1. Buat sebuah titik Q_0 dan Q_1 yang merupakan titik tengah dari P_0, P_1 dan P_1, P_2 . Lalu buat sebuah titik R_0 yang merupakan titik tengah dari Q_0, Q_1 . Gambarkan garis $P_0 - R_0 - P_2$.
- b. Rekurens: saat iterasi > 1 . Buat sebuah titik Q_0 dan Q_1 yang merupakan titik tengah dari P_0, P_1 dan P_1, P_2 . Lalu buat sebuah titik R_0 yang merupakan titik tengah dari Q_0, Q_1 . Lalu pecah permasalahan menjadi 2 upa-permasalahan. Dengan upa-permasalahan kiri melakukan proses (b) kembali dengan masukan sebuah senarai Q yang berisi P_0, Q_0, R_0 dan $i-1$ dan upa-permasalahan kanan melakukan proses (b) kembali dengan masukan sebuah senarai R yang berisi R_0, Q_1, P_2 dan $i-1$.

2.2 Pendekatan Bruteforce

Secara umum alur jalannya Algoritma *Brute Force* untuk menyelesaikan masalah membentuk *Bezier Curve* kuadratik adalah sebagai berikut:

- a. Tentukan *step* dengan cara membagi 1 dengan 2^n jumlah titik.
- b. Inisiasi nilai $t=0$. Selanjutnya masukkan nilai t, P_0, P_1, P_2 pada persamaan garis

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

- c. Simpan hasil dari persamaan sebagai titik-titik yang valid.

2.3 Analisis Algoritma *Divide and Conquer*

Solusi yang diimplementasikan dalam algoritma ini adalah dengan membagi pembentukan menjadi 2 daerah yakni kiri dan kanan. Untuk mencari titik tengah dalam 1 iterasi, maka kita akan melakukan 3 kali perhitungan titik tengah, sehingga kompleksitas waktunya adalah $O(1)$. Untuk memecah permasalahan maka kita melakukan rekursivitas dengan $2T(\text{iterasi} - 1)$. Untuk menggabungkan seluruh titik diperlukan 2 operasi pada setiap iterasi karena akan akan melakukan konkatenasi kompleksitas $O(n)$, n merupakan banyak titik, dengan panjang senarai sebesar 2^n dengan urutan

hasil pemecahan kiri – titik tengah – hasil pemecahan kanan
sehingga kompleksitas *combine* $O(2^{\text{iterasi}})$. Maka, didapatkan kompleksitas algoritma untuk membentuk *Bezier Curve* Kuadratik:

$$T(\text{iterasi}) = 2T(\text{iterasi} - 1) + 3 + 2^{\text{iterasi}} = O(2^{\text{iterasi}})$$

2.4 Analisis Algoritma *Bruteforce*

Penyelesaian masalah pembentukan *Bezier Curve* kuadratik dengan algoritma *Brute Force* adalah dengan cara menggunakan rumus berikut dengan t merupakan step yang digunakan.

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

Karena algoritma *brute force* digunakan sebagai pembanding bagi algoritma *Divide and Conquer*, untuk iterasi i akan dibentuk $2^i + 1$ titik. Kompleksitas algoritma untuk menghitung titik saat t adalah $O(1)$. Dikarenakan algoritma akan membuat $2^i + 1$ titik untuk i iterasi, maka kompleksitas akhir algoritma adalah $T(i) = 2^i + 1 = O(2^i)$

2.5 Bonus - Generalisasi *Bezier Curve* n Titik

Poin utama dari penyelesaian permasalahan ini dalam algoritma *Divide and Conquer* adalah serupa seperti yang telah dijabarkan sebelumnya. Namun, terdapat sedikit penyesuaian dalam menggambar *Bezier Curve* untuk n titik. Perbedaan utama adalah penentuan titik kontrol atau R_0 (pada *Bezier Curve* kuadratik).

Pada *Bezier Curve* dengan n titik kontrol, pada iterasi pertama maka kita harus mencari titik tengah dari setiap segmen garis (Q_i), lalu mencari titik tengah (R_i) dari segmen garis yang menghubungkan Q_i , Q_{i+1} . lalu dilanjutkan dengan mencari titik-titik tengah selanjutnya S_i

dan seterusnya hingga dalam 1 iterasi, hanya boleh terdapat 1 titik kontrol final.

Setelah didapatkan, maka pembagian upa-permasalahan tetap menjadi 2 yakni kiri dan kanan. Namun titik-titik yang digunakan untuk upa-permasalahan kiri adalah $P_0, Q_0, R_0 \dots$ dan upa-permasalahan kanan adalah ..., $R_{n-1}, Q_{n-1}, P_{n-1}$. Kompleksitas dari algoritma yang sudah di generalisasi ini menjadi

$$T(\text{iterasi}) = 2T(\text{iterasi} - 1) + c_0 n^2 + c_1 2^{\text{iterasi}} = O(2^{\text{iterasi}} * n^2)$$

Dengan $2T(\text{iterasi} - 1)$ merupakan pemecahan masalah menjadi 2 upa-permasalahan. Banyaknya proses untuk mencari titik tengah dalam 1 iterasi pada n titik adalah $(n - 1) + (n - 2) + \dots + 1 = n^2/2 = O(n^2)$.

2.6 Bonus - Animasi

Implementasi bonus pembuatan animasi pada tugas kecil ini menggunakan *library* yang ada dalam bahasa pemrograman *Python*. *Library* tersebut adalah **matplotlib**. Hal ini karena matplotlib memiliki fungsi bawaan untuk menangani titik koordinat pada bidang kartesius. Berikut langkah-langkah implementasi:

- a. Menerima data balikan hasil dari fungsi algoritma. Data tersebut berupa senarai berisi titik-titik koordinat yang akan di visualisasikan.
- a. Membuat sebuah *figure*
- b. Menggambarkan plot-plot dengan fungsi *handle_draw_line()*

3. Source Code

Penulis membagi file menjadi 4 file yakni main.py, Point.py, InputClass.py, MainAlgorithm.py, Draw.py. Struktur folder sebagai berikut

```
/Tucil2_13522018_13522039
.
├── LICENSE
├── README.md
├── bin
├── doc
└── src
    ├── Draw.py
    ├── InputClass.py
    ├── MainAlgorithm.py
    ├── Point.py
    └── main.py
└── test
```

3.1 main.py

```
● ● ●

from InputClass import InputClass
from MainAlgorithm import MainAlgorithm
from Draw import Draw
from Point import Point
from colorama import Fore, Style, Back
import time

class App:
    def __init__(self) -> None:
        """
        Inisiasi attribut kelas App
        """
        self.input_handle = InputClass()
        """
        Inisiasi untuk handling input
        """
        self.algorithm = MainAlgorithm()
        """
        Inisiasi untuk handling algoritma DnC
        """
        self.draw = Draw()
        """
        Inisiasi untuk handling draw dengan matplotlib
        """

    def main(self):
        """
        Fungsi utama aplikasi App. Berisi pemanggilan input, algoritma, dan penggambaran
        """
        self.input_handle.main()
        self.algorithm.init_variables_runtime(self.input_handle.point_list, self.input_handle.iterate)

        print(Fore.YELLOW)
        print("===== Divide and Conquer")
        print(Style.RESET_ALL)
        start_time = time.perf_counter() * 1000 # menghitung start time algoritma dnc
        self.algorithm.main()
        end_time = time.perf_counter() * 1000 # menghitung end time algoritma dnc

        print(Fore.GREEN)
        print(f"Elapsed Time: {end_time - start_time} ms")
        print(Style.RESET_ALL)
        print(self.algorithm.draw_list[-1])
        print(len(self.algorithm.draw_list[-1]))

        print(Fore.YELLOW)
        print("===== Bruteforce")
        print(Style.RESET_ALL)
        start_time = time.perf_counter() * 1000 # menghitung start time algoritma brute force
        brute_force_result = self.algorithm.brute_force(self.input_handle.point_list,
        self.input_handle.iterate)
        end_time = time.perf_counter() * 1000 # menghitung end time algoritma brute force

        print(Fore.GREEN)
        print(f"Elapsed Time: {end_time - start_time} ms")
        print(Style.RESET_ALL)
        print(brute_force_result)
        print(len(brute_force_result))

        # Inisiasi array untuk menggambar
        self.draw.init_variables_runtime(self.algorithm.draw_list)

        # fungsi menggambar Bezier Curve
        self.draw.main()

if __name__ == "__main__":
    app = App()
    app.main()
```

3.2 Point.py

```
● ● ●

from dataclasses import dataclass
import typing

@dataclass
class Point:
    """
    Kelas point. Memiliki atribut x dan y
    - `x: float`
    - `y: float`
    """
    x: float
    y: float

    def __add__(self, p2):
        """
        overload operator +
        Add 2 points together
        """
        return Point(self.x + p2.x, self.y + p2.y)

    def __mul__(self, constant):
        """
        overload operator *. comutative
        Multiply a point with a constant
        """
        return Point(self.x * constant, self.y * constant)

    def __rmul__(self, constant):
        """
        overload operator *. comutative
        Multiply a point with a constant
        """
        return Point(self.x * constant, self.y * constant)
```

3.3 InputClass.py

```
● ● ●

from typing import List, Optional, Union
from Point import Point
from colorama import Fore, Style, Back

class InputClass:
    def __init__(self) -> None:
        """
        Inisialisasi dari kelas InputClass sebagai handling input
        """
        self.n_titik: int = 0
        """
        Banyak titik input
        """
        self.point_list: list[Point] = []
        """
        Kumpulan Titik Koordinat
        struktur `list[Point]`
        """
        self.iterate: int = 0
        """
        Banyak iterasi sesuai masukan
        """
        self.magic_number: int = 20040406
        """
        Angka untuk stop looping
        """
        self.function_name: dict[str, function] = {
            "n": self.handle_input_n,
            "points": self.handle_input_points,
            "iterate": self.handle_input_iterate
        }

        self.error_code: dict[int, str] = {
            1: "Format Input Salah!. Mohon Masukkan Angka Saja",
            2: "Banyak Titik Minimal Tidak Boleh kurang dari 3.",
            3: "Program Terminated!",
            4: "Banyak input tidak sesuai banyak titik (n)",
            5: "Banyak Iterasi Minimal Tidak Boleh kurang dari 0.",
        }

    def main(self) -> None:
        """
        Fungsi utama menjalankan handle input n, points, iterate
        """
        self.handle_inputs("n")
        self.handle_inputs("points")
        self.handle_inputs("iterate")

    def handle_inputs(self, func_name: str) -> None:
        """
        Fungsi Template untuk menghandle masukan sesuai argumen func_name
        Args:
            `func_name: str` . Function name
        Rets:
            `int` . Kode Error
        """
        ret_val = None
        while(self.n_titik != self.magic_number and # break loop mechanism
              ret_val != 0): # input succeed

            ret_val = self.function_name[func_name]()

            # print error message
            if(ret_val != 0):
                self.error_message(ret_val)

        ret_val = -1
        self.end_program_mechanism()
```



```
● ● ●

def error_message(self, code: int) -> None:
    """
    Prosedur untuk menampilkan pesan error.

    Args:
        `code: int` . error code
    """
    print(Fore.RED)
    message_len = len(self.error_code[code]) + 2
    print("+" + "-" * message_len + "+")
    print("| " + self.error_code[code] + " |")
    print("+" + "-" * message_len + "+")
    print(Style.RESET_ALL)

def end_program_mechanism(self) -> None:
    """
    Prosedur menghentikan eksekusi program
    """
    if(self.n_titik == self.magic_number):
        self.error_message(3)
        exit(0)

if __name__ == "__main__":
    InputClass().main()
```

3.4 MainAlgorithm.py

```
● ● ●

from typing import List, Optional, Union
from Point import Point
from collections import deque

class MainAlgorithm:
    def __init__(self) -> None:
        """
        Inisiasi seluruh atribut yang digunakan oleh MainAlgorithm
        """
        self.max_iter: int = 0
        """
        Maksimum iterasi yang dilakukan sesuai input
        """
        self.point_list: list[Point] = []
        """
        List berisi point yang akan diolah.

        struktur `list[Point]`
        """
        self.draw_list: list[list[Point]] = []
        """
        List titik yang disimpan untuk digambar oleh visualizer
        """
        self.t: float = 0.5
        """
        t merupakan persen jarak antara titik awal dengan titik akhir
        `0 <= t <= 1`
        """

    def init_variables_runtime(self, points: list[Point], iter: int) -> None:
        """
        Inisiasi variable satu kali saja saat runtime. Point list dan max iter
        """
        self.point_list = points
        self.max_iter = iter

        self.draw_list.append(self.point_list) # draw the problem

    def main(self) -> None:
        """
        Fungsi utama untuk menjalankan algoritma Divide And Conquer
        """
        list_bezier_points = self.div_n_con(self.point_list, self.max_iter)

        # pemasukkan hasil titik kedalam list untuk digambar
        self.draw_list.append([self.point_list[0] + list_bezier_points + [self.point_list[-1]]])

    def get_post(self, point_a: Point, point_b: Point) -> Point:
        """
        Fungsi untuk mendapatkan titik t% dari point_a dan point_b.

        Args:
            `point_a: Point`. Koordinat Awal
            `point_b: point`. Koordinat Akhir
        Rets:
            `q_post: Point`. Titik tengah jika t = 0.5
        """
        q_post = (1-self.t) * point_a + self.t * point_b
        return q_post
```

```
● ● ●

def div_n_con(self, list_points: list[Point], iter: int) -> list[Point]:
    """
        Algoritma Utama Divide dan Conquer

    Args:
        `list_points: list[Point]`. Kumpulan titik koordinat untuk dicari nilai tengahnya
        `iter: int`. Mencatat iterasi
    Rets:
        `final_ans: list[Point]`. Kumpulan titik r (titik kontrol) untuk dikembalikan.
    """

    """ ===== CONQUER ===== """
    if iter <= 0:
        return []

    copy_list = list_points
    first = []
    last = []
    while(len(copy_list) != 1):
        temp = []
        for i in range(len(copy_list)-1):
            temp += [self.get_post(copy_list[i], copy_list[i+1])]

        self.draw_list.append(temp)
        first.append(temp[0])
        last.append(temp[-1])
        copy_list = temp

    last.reverse()

    """ ===== DIVIDE ===== """
    # go left
    left_branch = self.div_n_con([list_points[0], *first], iter - 1)

    # go right
    right_branch = self.div_n_con([*list(last), list_points[-1]], iter - 1)

    """ ===== COMBINE ===== """
    final_ans = left_branch + [copy_list[0]] + right_branch

    return final_ans
```



```
def brute_force(self, list_points : list[Point], iter : int) -> list[Point]:
    """
        Algoritma utama bruteforce dengan rumus untuk n sembarang

    Args:
        `list_points: list[Point]`. kumpulan titik untuk diolah
        `iter: int`. iterasi
    Rets:
        `solution: list[Point]`. Titik-titik bezier curve
    """

    # divide t by n * n
    t = 0
    step = 1 / (2 ** iter)

    n_titik = len(list_points)
    solution = []

    # main algorithm
    while(t < 1):
        new_point = Point(0, 0)
        for i in range(n_titik, 0, -1):
            konstanta = self.get_pascal_triangle(n_titik-1, n_titik-i)
            temp = konstanta * ((1-t)**(i-1)) * (t**(n_titik-i)) * list_points[n_titik-i]
            new_point = new_point + temp
        t += step
        solution.append(new_point)

    solution.append(list_points[-1]) # insert titik kontrol akhir
    return solution

def get_pascal_triangle(self, n: int, r: int) -> int:
    """
        Fungsi untuk mendapatkan konstanta dalam persamaan garis sesuai dengan segitiga pascal - optimized.

    C(n, r).

    Args:
        `n: int`. nilai n
        `r: int`. nilai r
    Rets:
        `res: int`. hasil dari konstanta segitiga pascal yang didapatkan.
    """

    res = 1
    for i in range(r):
        res = res * (n-i)
        res = res // (i+1)
    return res
```

3.5 Draw.py



```
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from Point import Point
import math
import numpy as np

class Draw:
    def __init__(self) -> None:
        """
        Inisialisasi variable atribut draw
        """
        self.layer_list: list[list[Point]] = []
        """
        List berisi iterasi dari setiap point untuk penggambaran
        struktur `list[list[Point]]`
        """
        self.function_name: dict[str, function] = {
            "dot": self.handle_draw_dot,
            "line": self.handle_draw_line_animate,
            "no-line": self.handle_draw_line
        }
        """
        Daftar nama fungsi yang available.

        Dapat di Extend sesuai kebutuhan
        """
        self.fig, self.axes = plt.subplots()
        """
        figure dan axes dari matplotlib
        """
        self.frame_tot: int = 0
        """
        total frame animasi
        """
        self.animation_data: list = []
        """
        Senarai untuk menyimpan data animasi garis
        """
        self.animation_dot_data: list = []
        """
        Senarai untuk menyimpan data animasi titik
        """
        self.layer_list: list[list[Point]] = []
        """
        Senarai berisi lapisan dari animasi
        """

    def init_variables_runtime(self, points: list[list[Point]]) -> None:
        """
        Inisiasi variable pertama kali saat run time
        """
        self.layer_list: list[list[Point]] = points

        x_points = []
        y_points = []

        for layer in self.layer_list:
            for points in layer:
                x_points.append(points.x)
                y_points.append(points.y)
        max_width = max(x_points) + 2
        min_width = min(x_points) - 2
        max_height = max(y_points) + 2
        min_height = min(y_points) - 2

        self.axes.set_xlim([min_width, max_width])
        self.axes.set_ylim([min_height, max_height])
```



```
def main(self) -> None:
    """
    Fungsi utama untuk menjalankan penggambaran
    """
    self.draw("dot")
    self.draw("line") # dengan animasi
    # self.draw("no-line") # tanpa animasi

    # animate
    self.anim_list[0] = animation.FuncAnimation(self.fig, self.update, interval=1,
frames=self.frame_tot, repeat=False)
    self.anim_list[1] = animation.FuncAnimation(self.fig, self.update_dot, interval=1,
frames=self.frame_tot, repeat=False)

    plt.show()

def handle_draw_dot(self):
    for i, layer in enumerate(self.layer_list):
        x_dot_points = [p.x for p in layer]
        y_dot_points = [p.y for p in layer]

        if i == 0:
            temp, = self.axes.plot([], [], 'bo', alpha=0.5)
        elif i == len(self.layer_list)-1:
            temp, = self.axes.plot([], [], 'go', alpha=0.5)
        else:
            temp, = self.axes.plot([], [], 'co', alpha=0.5)

        self.animation_dot_data.append([x_dot_points, y_dot_points, temp, 1])

def draw(self, func_name: str) -> None:
    """
    Fungsi Handle untuk penggambaran
    """
    self.function_name[func_name]()

def handle_draw_line(self) -> None:
    """
    Fungsi untuk menggambar garis tanpa animasi
    """
    self.anim_list = [None for _ in range(len(self.layer_list))]
    # print(self.layer_list)
    for i, layer in enumerate(self.layer_list):
        self.x_points = [p.x for p in layer]
        self.y_points = [p.y for p in layer]

        if i == 0: # gambar soal
            self.axes.plot(self.x_points, self.y_points, '-bo')
        elif i == len(self.layer_list)-1: # gambar bezier curve akhir
            self.axes.plot(self.x_points, self.y_points, '-go')
        else: # gambar titik koordinat antara (proses pembentukan bezier curve)
            self.axes.plot(self.x_points, self.y_points, '--co', alpha=0.5)
```

```
● ● ●
```

```
def handle_draw_line_animate(self) -> None:
    """
    Fungsi untuk menggambar Garis dengan titik koordinatnya. Dengan animasi
    """
    self.anim_list = [None for _ in range(len(self.layer_list))]
    # print(self.layer_list)
    for i, layer in enumerate(self.layer_list):
        self.x_points = [p.x for p in layer]
        self.y_points = [p.y for p in layer]

        if i == 0: # gambar soal
            temp, = self.axes.plot([], [], '-b')
        elif i == len(self.layer_list)-1: # gambar bezier curve akhir
            temp, = self.axes.plot([], [], '-g')
        else: # gambar titik koordinat antara (proses pembentukan bezier curve)
            temp, = self.axes.plot([], [], '--c', alpha=0.5)

        # if i == 0:
        x_data = np.array([])
        y_data = np.array([])
        for j in range(len(self.x_points)-1):
            x_data = np.append(x_data, np.linspace(self.x_points[j], self.x_points[j+1], 10))
            y_data = np.append(y_data, np.linspace(self.y_points[j], self.y_points[j+1], 10))
        x_data = np.append(x_data, self.x_points[-1])
        y_data = np.append(y_data, self.y_points[-1])

        # if i==0:
        self.frame_tot += len(x_data)
        self.animation_data.append([x_data, y_data, temp])

    def update(self, frame) -> None:
        """
        fungsi untuk memperbaharui axes plot garis sesuai dengan frame

        Args:
            `frame: int`. frame saat ini
        """
        # make boundary for interval
        boundary_list = [0]
        for i in range(len(self.animation_data)):
            boundary_list.append(boundary_list[i] + len(self.animation_data[i][0]))

        # look for which interval
        for i in range(len(boundary_list)):
            if (frame < boundary_list[i]):
                self.animation_data[i-1][2].set_data(self.animation_data[i-1][0][:frame - boundary_list[i-1]], self.animation_data[i-1][1][:frame - boundary_list[i-1]])
                return self.animation_data[i-1][2]

    def update_dot(self, frame) -> None:
        """
        fungsi untuk memperbaharui axes plot titik sesuai dengan frame

        Args:
            `frame: int`. frame saat ini
        """
        boundary_list = [0]
        for i in range(len(self.animation_data)):
            boundary_list.append(boundary_list[i] + len(self.animation_data[i][0]))

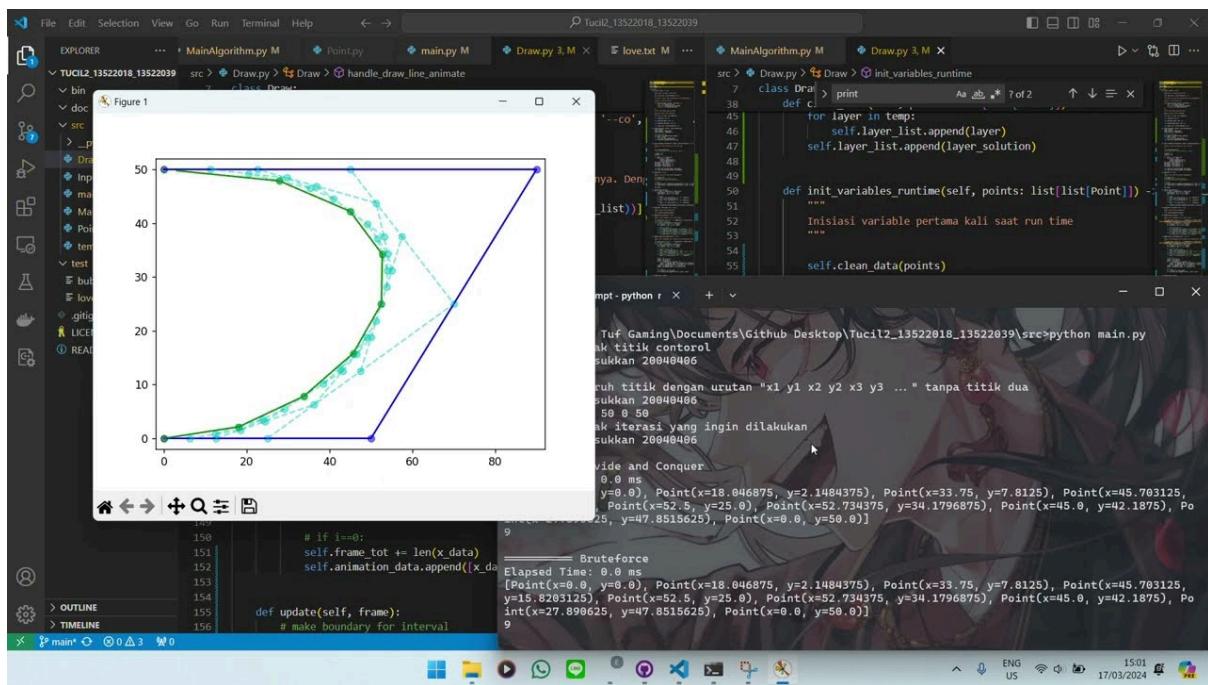
        for i in range(len(boundary_list)):
            if(frame < boundary_list[i]):
                end = self.animation_dot_data[i-1][3]
                self.animation_dot_data[i-1][2].set_data(self.animation_dot_data[i-1][0][:end],
                self.animation_dot_data[i-1][1][:end])
                self.animation_dot_data[i-1][3] = end + 1

        return self.animation_dot_data[i-1][2],
```

4. How To Use

Berikut merupakan langkah-langkah untuk menjalankan program.

1. Jalankan program dengan perintah python main.py
2. Masukkan input sesuai dengan prompt lalu tekan enter.
3. Animasi akan muncul.



5. Test Case

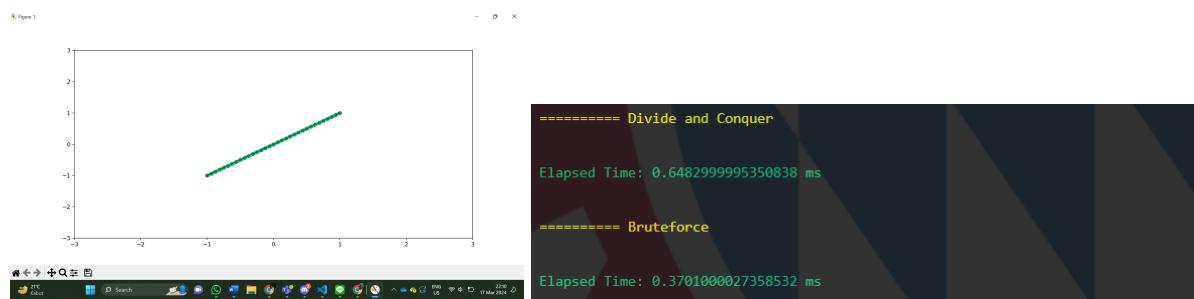
5.1 Test Case 1

Input

Kurva Bezier Linier

```
Masukkan banyak titik kontrol
Untuk stop masukkan 20040406
> 3
Masukkan seluruh titik dengan urutan "x1 y1 x2 y2 x3 y3 ..." tanpa titik dua
Untuk stop masukkan 20040406
> -1 -1 0 0 1 1
Masukkan banyak iterasi yang ingin dilakukan
Untuk stop masukkan 20040406
> 5
```

Result



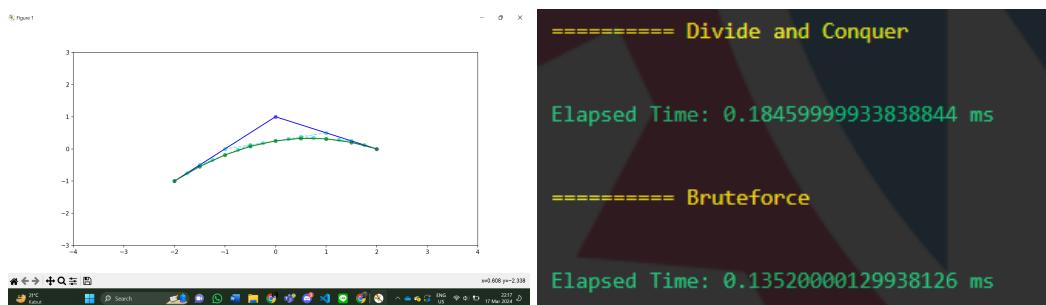
5.2 Test Case 2

Input

Kurva Bezier Biasa

```
Masukkan banyak titik kontrol  
Untuk stop masukkan 20040406  
> 3  
Masukkan seluruh titik dengan urutan "x1 y1 x2 y2 x3 y3 ..." tanpa titik dua  
Untuk stop masukkan 20040406  
> -2 -1 0 1 2 0  
Masukkan banyak iterasi yang ingin dilakukan  
Untuk stop masukkan 20040406  
> 3
```

Result



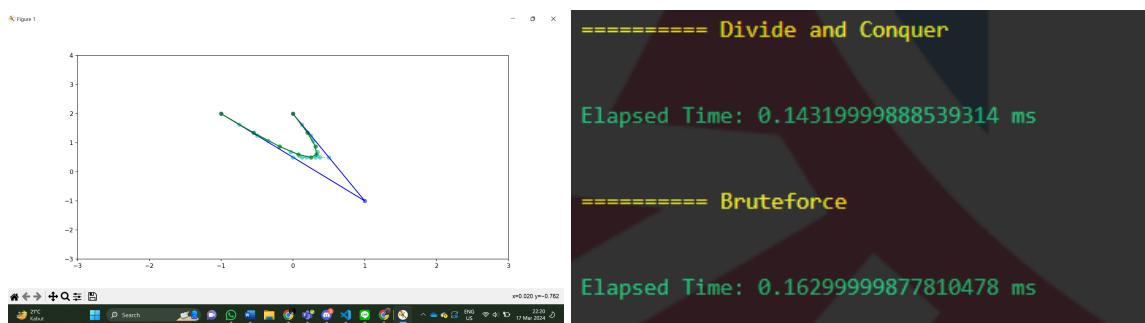
5.3 Test Case 3

Input

Kurva Bezier Biasa

```
Masukkan banyak titik control  
Untuk stop masukkan 20040406  
> 3  
Masukkan seluruh titik dengan urutan "x1 y1 x2 y2 x3 y3 ..." tanpa titik dua  
Untuk stop masukkan 20040406  
> -1 2 1 -1 0 2  
Masukkan banyak iterasi yang ingin dilakukan  
Untuk stop masukkan 20040406  
> 3
```

Result



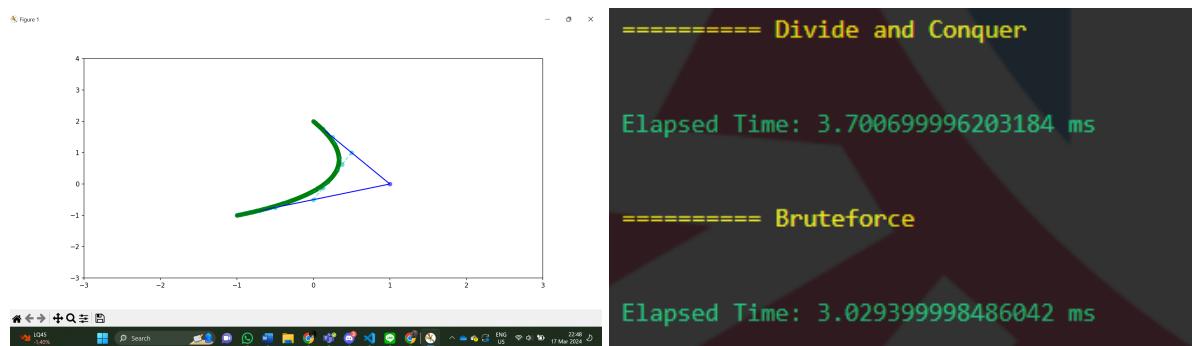
5.4 Test Case 4

Input

Kurva Bezier dengan iterasi besar

```
Masukkan banyak titik control  
Untuk stop masukkan 20040406  
> 3  
Masukkan seluruh titik dengan urutan "x1 y1 x2 y2 x3 y3 ..." tanpa titik dua  
Untuk stop masukkan 20040406  
> -1 -1 1 0 0 2  
Masukkan banyak iterasi yang ingin dilakukan  
Untuk stop masukkan 20040406  
> 8
```

Result



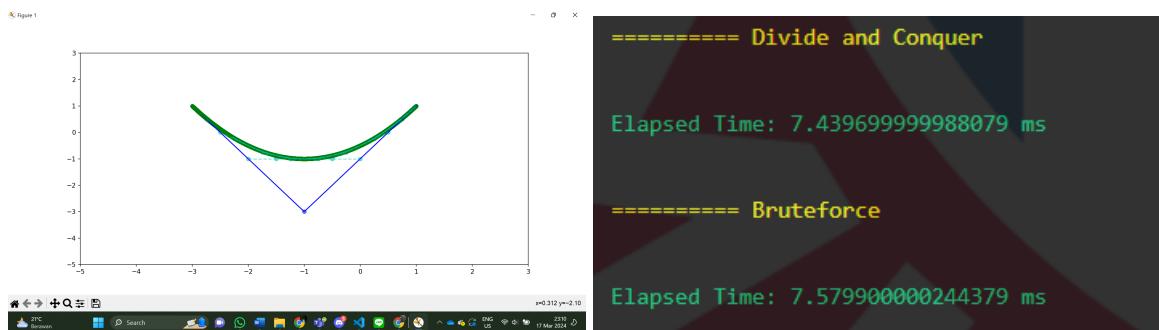
5.5 Test Case 5

Input

Kurva Bezier dengan iterasi besar

```
Masukkan banyak titik control  
Untuk stop masukkan 20040406  
> 3  
Masukkan seluruh titik dengan urutan "x1 y1 x2 y2 x3 y3 ..." tanpa titik dua  
Untuk stop masukkan 20040406  
> -3 1 -1 -3 1 1  
Masukkan banyak iterasi yang ingin dilakukan  
Untuk stop masukkan 20040406  
> 10
```

Result



5.6 Test Case 6

Input

Handle input banyak titik salah (bukan tipe integer)

```
Masukkan banyak titik control
Untuk stop masukkan 20040406
> tes
```

Result

```
+-----+
| Format Input Salah!. Mohon Masukkan Angka Saja |
+-----+
```

5.7 Test Case 7

Input

Handle input banyak titik salah (tidak sesuai batas minimal banyak titik)

```
Masukkan banyak titik control  
Untuk stop masukkan 20040406  
> 2
```

Result

```
+-----+  
| Banyak Titik Minimal Tidak Boleh kurang dari 3. |  
+-----+
```

5.8 Test Case 8

Input

Handle input salah (banyak titik kontrol tidak sesuai input sebelumnya)

```
Masukkan seluruh titik dengan urutan "x1 y1 x2 y2 x3 y3 ..." tanpa titik dua  
Untuk stop masukkan 20040406  
> 2 0 1 -2
```

Result

```
+-----+  
| Banyak input tidak sesuai banyak titik (n) |  
+-----+
```

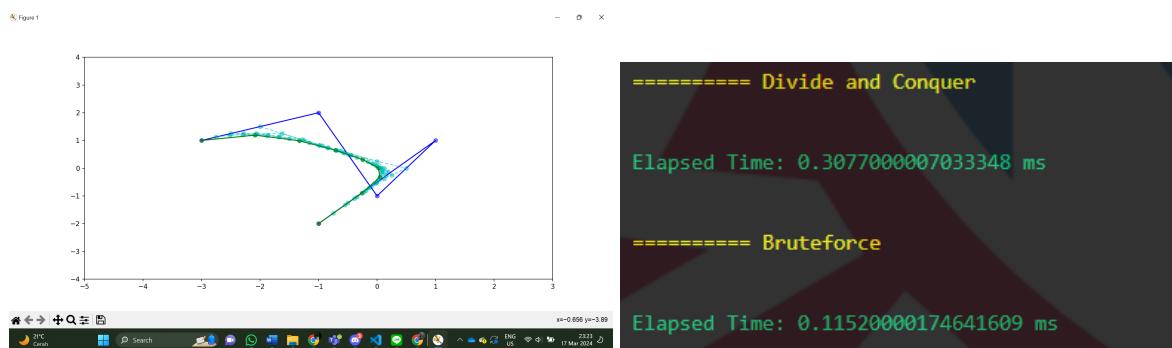
5.9 Test Case 9

Input

Bonus : Kurva Bezier dengan titik kontrol > 3

```
Masukkan banyak titik control  
Untuk stop masukkan 20040406  
> 5  
Masukkan seluruh titik dengan urutan "x1 y1 x2 y2 x3 y3 ..." tanpa titik dua  
Untuk stop masukkan 20040406  
> -3 1 -1 2 0 -1 1 1 -1 -2  
Masukkan banyak iterasi yang ingin dilakukan  
Untuk stop masukkan 20040406  
> 3
```

Result



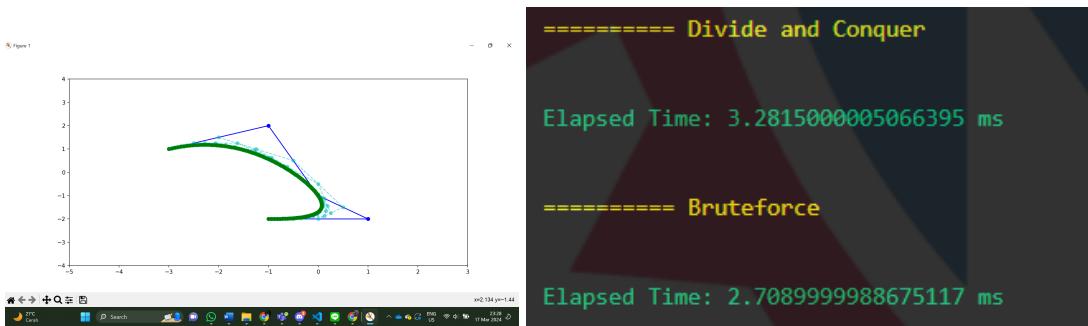
5.10 Test Case 10

Input

Bonus : Kurva Bezier dengan titik kontrol > 3 dan iterasi besar

```
Masukkan banyak titik control  
Untuk stop masukkan 20040406  
> 5  
Masukkan seluruh titik dengan urutan "x1 y1 x2 y2 x3 y3 ..." tanpa titik dua  
Untuk stop masukkan 20040406  
> -3 1 -1 2 0 -1 1 -2 -1 -2  
Masukkan banyak iterasi yang ingin dilakukan  
Untuk stop masukkan 20040406  
> 7
```

Result



Lampiran

GitHub

Kode dapat diakses pada repository GitHub

https://github.com/WazeAzure/Tucil2_13522018_13522039

Repository akan di public pada Hari Senin 18 Maret, dan paling telat pada Selasa 19 Maret pagi.

Jika tidak dapat diakses, mohon kontak penulis melalui LINE yenyenhu atau ibrahimrasyid__ atau melalui Ms Teams.

Poin-Poin Penting

No	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	