LAPORAN TUGAS KECIL II

IF2211 STRATEGI ALGORITMA

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Disusun oleh:

Venantius Sean Ardi Nugroho	13522078
Raden Francisco Trianto Bratadiningrat	13522091

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2024

DAFTAR ISI

DAFTAR ISI **2**
DAFTAR GAMBAR **3**
DAFTAR TABEL **4**
BAB I DESKRIPSI MASALAH **5**
BAB II TEORI SINGKAT **7**
2.1 Algoritma Divide and Conquer 7
2.2 Pembuatan Bézier Curve 7
BAB III RANCANGAN DAN IMPLEMENTASI PROGRAM **9**
3.1 Rancangan Algoritma 9
3.2 Implementasi Program 10
3.3 Source Code Program 11
BAB IV TEST CASE **15**
4.1 Test case 1 15
4.2 Test case 2 16
4.3 Test case 3 17
4.4 Test case 4 18
4.5 Test case 5 19
4.6 Test case 6 20
4.7 Test case 7 21
4.8 Test case 8 22
4.8 Test case 9 23
4.10 Test case 10 24
BAB V PERBANDINGAN DAN ANALISIS SOLUSI **25**
BAB VI KESIMPULAN DAN SARAN **26**
5.1 Kesimpulan 26
5.2 Saran 26
LAMPIRAN **27**
Pranala Repository Github 27
Checklist Fitur 27
DAFTAR PUSTAKA **28**

DAFTAR GAMBAR

Gambar 1.1 Kurva Bézier Kubik

Gambar 1.2 Pembentukan Kurva Bézier Kuadratik

Gambar 2.2.1 Visualisasi contoh Recursive Subdivision pada Bezier Curve orde 3 serta definisi formalnya

Gambar 4.1.1 Input Test Case 1

Gambar 4.1.2 Hasil divide and conquer dan brute force Test Case 1

Gambar 4.2.1 Input Test Case 2

Gambar 4.2.2 Hasil divide and conquer dan brute force Test Case 2

Gambar 4.3.1 Input Test Case 3

Gambar 4.3.2 Hasil divide and conquer dan brute force Test Case 3

Gambar 4.4.1 Input Test Case 4

Gambar 4.4.2 Hasil divide and conquer dan brute force Test Case 4

Gambar 4.5.1 Input Test Case 5

Gambar 4.5.2 Hasil divide and conquer dan brute force Test Case 5

Gambar 4.6.1 Input Test Case 6

Gambar 4.6.2 Hasil divide and conquer dan brute force Test Case 6

Gambar 4.7.1 Input Test Case 7

Gambar 4.7.2 Hasil divide and conquer dan brute force Test Case 7

Gambar 4.8.1 Input Test Case 8

Gambar 4.8.2 Hasil divide and conquer Test Case 8

Gambar 4.9.1 Input Test Case 9

Gambar 4.9.2 Hasil divide and conquer Test Case 9

Gambar 4.10.1 Input Test Case 10

Gambar 4.10.2 Hasil divide and conquer Test Case 10

DAFTAR TABEL

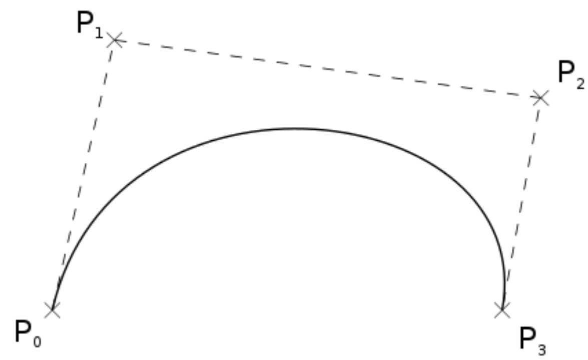
Tabel 3.2.1.1 Implementasi bfBezier.py

Tabel 3.2.2.1 Implementasi dcBezier.py

Tabel 3.2.1.1 Implementasi kelas Point

Tabel 3.2.3.2 Implementasi utils.py

BAB I
DESKRIPSI MASALAH



Gambar 1.1 Kurva Bézier Kubik
(Sumber: https://id.wikipedia.org/wiki/Kurva_B%C3%A9zier)

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

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

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

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

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

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P_2, dengan P_0 dan P_2 sebagai titik kontrol awal dan akhir, dan P_1 menjadi titik kontrol antara. Dengan menyatakan titik Q_1

terletak diantara garis yang menghubungkan P_1 dan P_2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q_0 berada, maka dapat dinyatakan sebuah titik baru, R_0 yang berada diantara garis yang menghubungkan Q_0 dan Q_1 yang bergerak membentuk kurva Bézier kuadratik terhadap titik P_0 dan P_2. Berikut adalah uraian persamaannya.

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

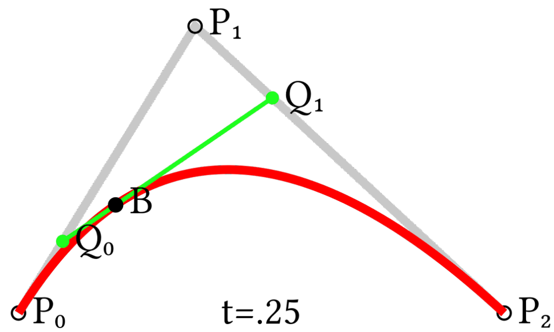
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

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

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

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

Berikut adalah ilustrasi dari kasus diatas.



Gambar 1.2 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.

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

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

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*.

BAB II
TEORI SINGKAT

2.1 Algoritma Divide and Conquer

Algoritma *Divide and Conquer* adalah salah satu strategi untuk memecahkan masalah dalam persoalan komputer. Ciri utama dari strategi ini adalah adanya pencacahan suatu permasalahan yang rumit menjadi masalah - masalah (upa masalah) yang lebih mudah. Algoritma ini mirip dengan algoritma *Decrease and Conquer*, bedanya *Decrease and Conquer* lebih mementingkan berkurangnya ukuran masalah daripada berkurangnya kompleksitas masalah. Contoh dari algoritma *divide and conquer* antara lain: quick sort, merge sort, pasangan titik terdekat, dan Algoritma Strassen.

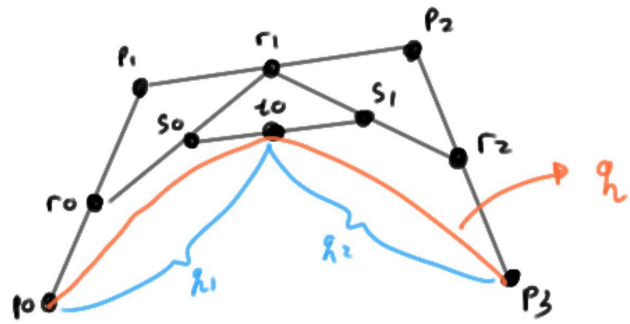
Terdapat 3 proses yang harus ada di dalam suatu strategi algoritma supaya bisa dikatakan bahwa algoritma tersebut tergolong *Divide and Conquer*, yaitu proses :

1. Divide: Pada proses ini masalah akan dibagi menjadi upa persoalan yang lebih kecil.
2. Conquer: Menyelesaikan upa persoalan hingga terselesaikan, menggunakan metode rekursi.
3. Combine: Menggabungkan hasil - hasil dari upa persoalan untuk mendapatkan hasil yang sesungguhnya.

2.2 Pembuatan Bézier Curve

Pada deskripsi masalah telah disajikan salah satu cara untuk membuat kurva bezier dengan cara *Brute Force*. Algoritma yang digunakan pada pembuatan kurva tersebut adalah algoritma de Casteljau, pada bahasan kali ini, penulis akan membahas konsep dan teori - teori yang digunakan dalam pembuatan kurva Bezier dengan algoritma *Divide and Conquer*.

Penulis menggunakan konsep *Recursive Subdivision* dalam penyelesaian masalah pembuatan kurva Bezier dengan algoritma *Divide and Conquer*. Penulis akan ambil contoh pada kurva Bezier berorde 3. *Recursive Subdivision* menyatakan bahwa untuk setiap kurva Bezier berorde 3 q (sebenarnya berlaku untuk orde berapapun, tapi di dalam konteks contoh ini 3), maka terdapat q1 yang merupakan kurva bezier berorde yang sama dengan titik kontrol p0,r0,s0 dan t0 serta terdapat q2 yang merupakan kurva bezier berorde yang sama dengan titik kontrol t0,s1,r2 dan p3 yang merupakan sisi kiri dan sisi kanan dari q. Untuk memperjelas penjelasan di atas, coba perhatikan ilustrasi di bawah.



$$q(u) = \begin{cases} q_1\left(\frac{u}{u_0}\right); & 0 \le u \le u_0 \\ q_1\left(\frac{u-u_0}{1-u_0}\right); & u_0 \le u \le 1 \end{cases}$$

Gambar 2.2.1 Visualisasi contoh *Recursive Subdivision*
pada Bezier Curve orde 3 serta definisi formalnya

Konsep ini relevan karena kita tetap bisa membagi upa - upa kurva ini menjadi upa - upa yang lebih kecil lagi. Dengan menyambungkan titik - titik tengah dari upa - upa tersebut yang dibuat dari iterasi algoritma yang cukup, maka kurva bezier yang cukup mulus akan tercipta.

BAB III
RANCANGAN DAN IMPLEMENTASI PROGRAM

3.1 Rancangan Algoritma

3.1.1 Algoritma *Brute Force*

Pada dasarnya penerapan ini hanya diciptakan sebagai pembanding dengan algoritma *divide and conquer* yang diciptakan, oleh sebab itu pendekatan ini menggunakan rumus yang sudah "jadi" untuk mencari titik yang digunakan. Penerapannya adalah sebagai berikut:

1. Cari banyak titik yang diperlukan pada jumlah iterasi
2. Gunakan banyak titik tersebut untuk mencari nilai t.
3. Gunakan nilai - nilai t pada rumus $B(t) = P1 + (1\text{-}t)^2 (p0\text{-}p1) + t^2 (p2 - p1)$ untuk mendapatkan titik - titik pada kurva bezier.

3.2.2 Algoritma *Divide and Conquer*

Berikut adalah langkah-per-langkah dari algoritma *Divide and Conquer* yang diterapkan:

1. Siapkanlah suatu fungsi untuk mengembalikan result serta memanggil dan menginisialisasikan fungsi utama dengan counter mulai dari 0.

2. Kita akan mencoba mencari mid point (didefinisikan sebagai titik tengah dari dua buah titik yang merupakan poin dari kurva bezier). Untuk mendapatkan mid point tersebut, panggilah suatu fungsi yang membuatnya dengan parameter titik - titik kontrol yang kami punya.

3. Di dalam fungsi pembuat mid point, titik - titik kontrol akan dicari titik - titik tengahnya secara rekursif hingga hanya ada dua titik tengah, pada saat itu kembalikanlah titik tengah dari dua titik tengah tersebut.

4. Perlu diingat bahwa fungsi ini perlu menyimpan titik - titik tengah yang berguna dalam membuat sisi kiri dan sisi kanan kurva bezier dan mengembalikannya ke fungsi utama.

5. Di sinilah tahap divide pada algoritma ini yaitu dengan membuat sisi kiri dan kanan kurva bezier menggunakan titik berguna yang telah kita temukan sebelumnya.

6. Inkrementasikan counter.

7. Tahap conquer terdapat pada pemanggilan fungsi utama pada sisi kiri dan sisi kanan kurva.

8. Tahap combine terdapat pada pemasukkan mid point pada suatu container yang akan dikembalikan sebagai result akhir.

3.2 Implementasi Program

3.2.1 bfBezier.py

Terdapat implementasi algoritma pembuatan kurva bezier dengan pendekatan brute force.

Tabel 3.2.1.1 Implementasi bfBezier.py

Nama Fungsi/Prosedur	Deskripsi
bfBezier	Implementasi dari fungsi pembuatan kurva bezier dengan pendekatan brute force, menggunakan algoritma de Casteljau. Fungsi ini hanya berfungsi untuk kurva bezier dengan 3 titik kontrol.

3.2.2 dcBezier.py

Terdapat implementasi algoritma pembuatan kurva bezier dengan pendekatan divide and conquer

Tabel 3.2.2.1 Implementasi dcBezier.py

Nama Fungsi/Prosedur	Deskripsi
dcBezier	Fungsi ini adalah fungsi yang akan mengembalikan hasil akhir pada user. Hasil dari dcBezier ini adalah suatu array of array of points dimana index ke - 0 adalah hasil point - point bezier pada iterasi pertama , index ke - 1 adalah hasil point - point bezier pada iterasi kedua, seterusnya sampai iterasi ke - n. Hal tersebut dilakukan untuk memudahkan visualisasi.
intermediaryBezier	Fungsi ini berfungsi dalam menghasilkan hasil kurva bezier pada iterasi tertentu. Selain itu intermediaryBezier berguna dalam menginisialisasi counter sebagai 0 dan container sebagai list kosong untuk dipakai di fungsi utama.
dcBuilder	dcBuilder merupakan fungsi utama dari algoritma ini, berguna dalam melakukan divide, conquer , dan combine pada algoritma ini.

makeMidpoint	Mengembalikan mid point dari suatu array of control points dan mengembalikan array kiri dan array kanan yang akan digunakan untuk iterasi selanjutnya.

3.2.3 utils.py

Fungsi - fungsi yang digunakan untuk membantu pembuatan program.

Point class definition:

Tabel 3.2.1.1 Implementasi kelas Point

Atribut	**Keterangan**
x	Posisi pada x.
y	Posisi pada y.
Method	**Keterangan**
__str__	Digunakan untuk mem - print Point
__eq__	Membandingkan dua buah Point. Mengembalikan true hanya jika x1 == x2 dan y1 == y2

Tabel 3.2.3.2 Implementasi utils.py

Nama Fungsi/Prosedur	**Deskripsi**
midPoint	Mengembalikan titik tengah dari dua buah Point
calculate_amount_of_point	Menghitung jumlah titik pada *brute force* kurva bezier
printList	Digunakan untuk mem - print tiap elemen dalam array of Point

3.3 Source Code Program

3.3.1 bfBezier.py

```
from utils import *

import time



# Build Bezier Curve using Brute Force algorithm

def bfBezier(start_point: Point, mid_point: Point, end_point: Point, 
desire_iteration: int) -> tuple[list[Point], int]:

    start_time = time.time()

    amount_of_point = calculate_amount_of_point(desire_iteration)

    tVals : list[int] = [i / (amount_of_point - 1) for i in range(amount_of_point)]

    result : list[Point] = []

    for t in tVals:

        newX = ((1 - t) ** 2 * start_point.x) + (2 * (1 - t) * t * mid_point.x) + 
(t**2 * end_point.x)

        newY = ((1 - t) ** 2 * start_point.y) + (2 * (1 - t) * t * mid_point.y) + 
(t**2 * end_point.y)

        newPoint = Point(newX, newY)

        result.append(newPoint)

    end_time = time.time()

    execution_time = end_time - start_time

    return [result, execution_time]
```

3.3.2 dcBezier.py

```
from utils import *

import time



# Build Bezier Curve using Divide and Conquer algorithm
```

```
def dcBezier(control_points: list[Point], desire_iteration: int, num_of_CP: int) ->
tuple[list[list[Point]], int]:

    result : list[list[Point]] = []

    for iteration in range(1, desire_iteration + 1):

        if iteration == desire_iteration:

            start_time = time.time()

        result.append(intermediary_Bezier(control_points, iteration, num_of_CP))

    end_time = time.time()

    execution_time = end_time - start_time

    return [result,execution_time]


def intermediary_Bezier(control_points: list[Point], desire_iteration: int,
num_of_CP: int) -> list[Point]:

    result : list[Point] = [control_points[0]]

    dcBuilder(control_points, result, 0, desire_iteration, num_of_CP)

    result.append(control_points[-1])

    return result


def dcBuilder(control_points: list[Point], container: list, counter: int,
desire_iteration: int, num_of_CP: int) -> None:

    if counter < desire_iteration:

        leftPoints= [control_points[0]]

        rightPoints = [control_points[-1]]

        midpoint = make_mid_point(control_points, num_of_CP,
leftPoints,rightPoints)

        counter += 1


        dcBuilder(leftPoints, container, counter, desire_iteration, num_of_CP)

        container.append(midpoint)

        dcBuilder(rightPoints, container, counter, desire_iteration, num_of_CP)
```

```
def make_mid_point(control_points: list[Point], num_of_CP: int, useful_midpoints_a: list[Point],useful_midpoints_b: list[Point]) -> Point:

    if num_of_CP == 2:

        real_midpoint = mid_point(control_points[0], control_points[1])

        useful_midpoints_a.append(real_midpoint)

        useful_midpoints_b.insert(0,real_midpoint)

        return real_midpoint


    else:

        points_between = []

        for i in range(num_of_CP - 1):

            points_between.append(mid_point(control_points[i], control_points[i + 1]))


        useful_midpoints_a.append(points_between[0])

        useful_midpoints_b.insert(0,points_between[-1])

        num_of_CP -= 1

        return make_mid_point(points_between, num_of_CP, useful_midpoints_a,useful_midpoints_b)
```

3.3.3 utils.py

```
# Global Import Definition

import tkinter as tk

from tkinter import Tk, messagebox, Entry


import matplotlib.pyplot as plt

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

from matplotlib.widgets import Slider


# Class Definition
```

```
class Point:

    def __init__(self, x: float, y: float):

        self.x = x

        self.y = y


    def __str__(self):

        return f"({self.x},{self.y})"


    def __eq__(self, otherPoint) -> bool:

        return (self.x == otherPoint.x) and (self.y == otherPoint.y)


# Global Function

def is_float(string: str) -> bool:

    try:

        float(string)

        return True

    except ValueError:

        return False


# Get the mid point between two point

def mid_point(p1: Point, p2: Point) -> Point:

    midP = Point(((p1.x + p2.x) / 2), ((p1.y + p2.y) / 2))

    return midP


# used to calculate how many points in bfBezier

def calculate_amount_of_point(n: int) -> int:

    if n == 1:

        return 3

    else:
```

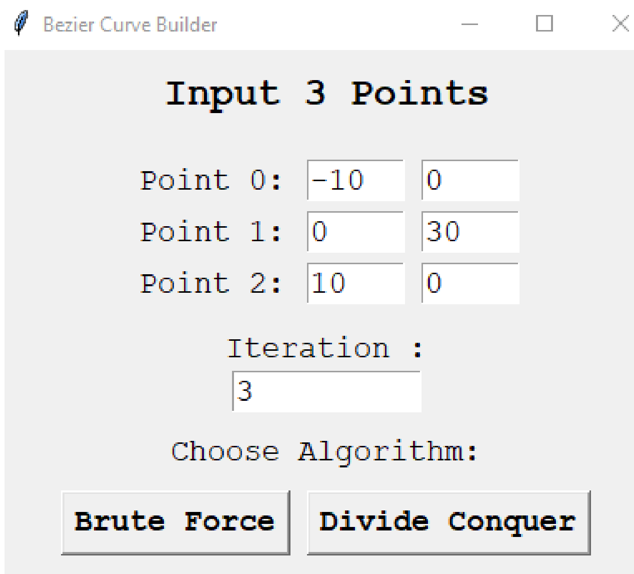
```python
        return (calculate_amount_of_point(n - 1) * 2) - 1


# Debugging Tool
def print_list(lis: list[Point]) -> None:
    for i in lis:
        print(i)
```

BAB IV
TEST CASE

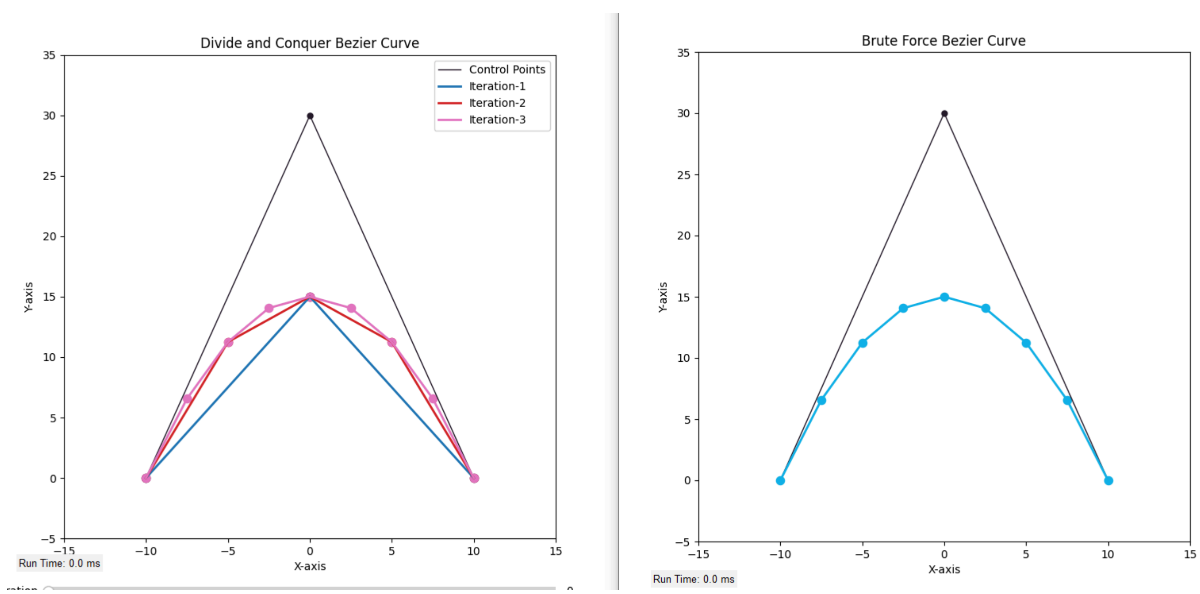
4.1 Test case 1 - 3 titik

Brute force runtime: 0.0 ms

Divide and Conquer runtime: 0.0 ms



Gambar 4.1.1 Input Test Case 1

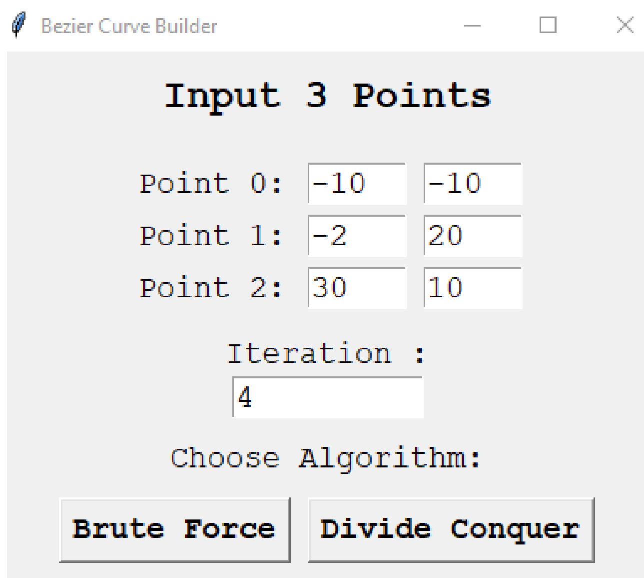


Gambar 4.1.2 Hasil divide and conquer dan *brute force* Test Case 1

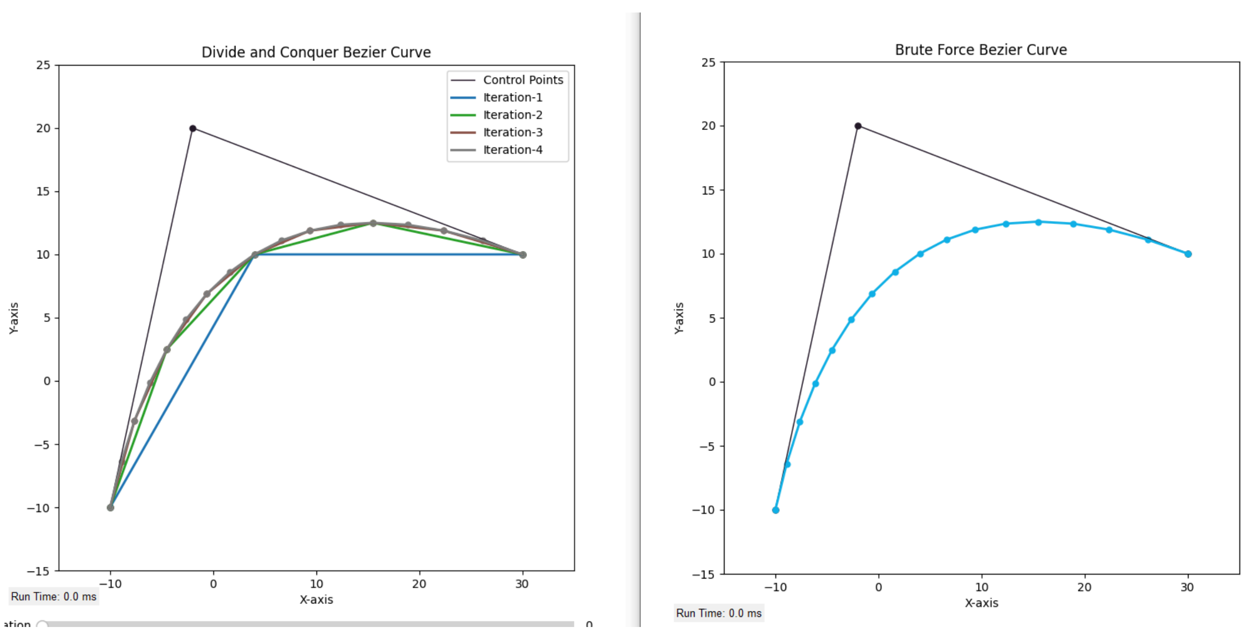
4.2 Test case 2 - 3 titik

Brute force runtime: 1.0 ms

Divide and Conquer runtime: 0.0 ms



Gambar 4.2.1 Input Test Case 2

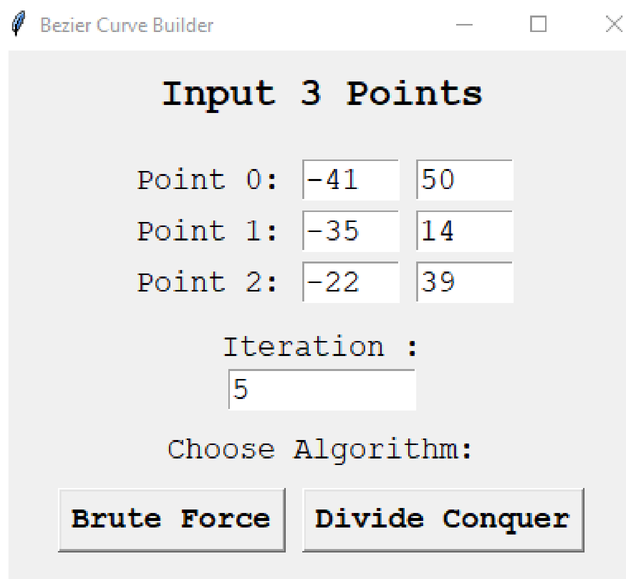


Gambar 4.2.2 Hasil divide and conquer dan brute force Test Case 2

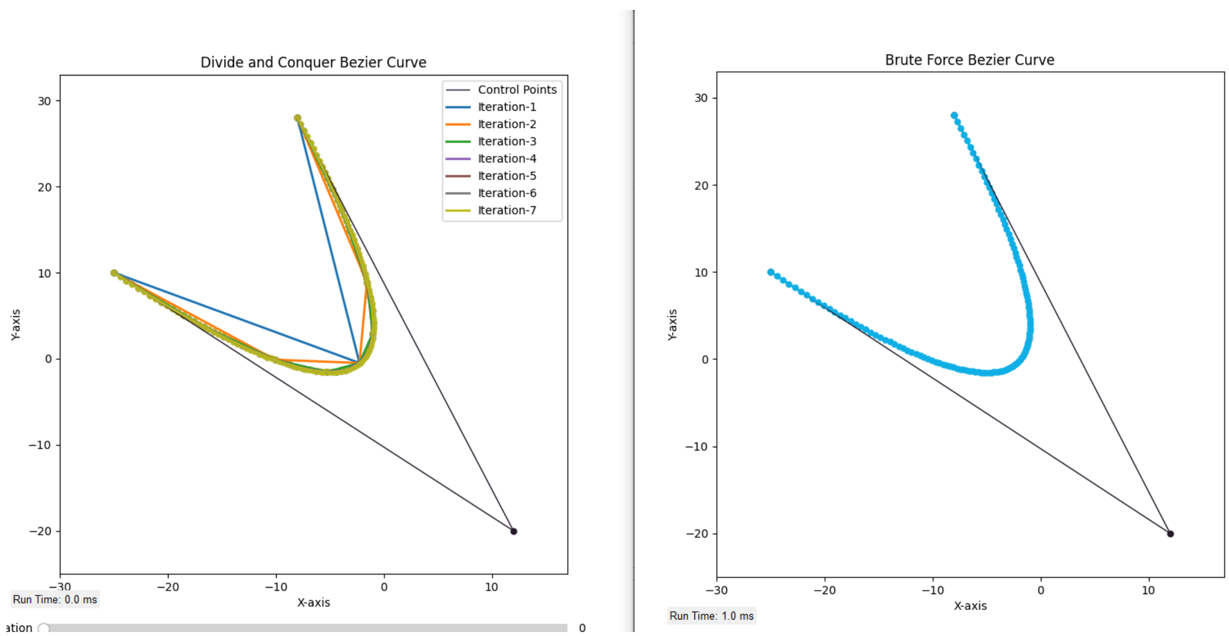
4.3 Test case 3 - 3 titik

Brute force runtime: 1.0 ms

Divide and Conquer runtime: 0.0 ms



Gambar 4.3.1 Input Test Case 3

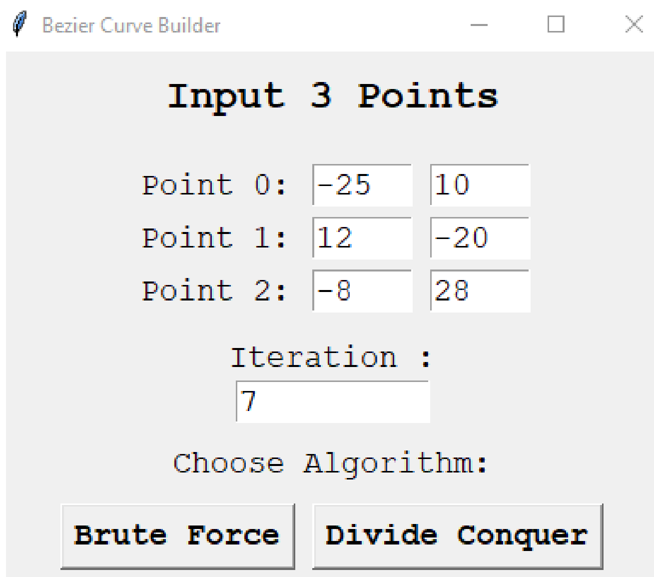


Gambar 4.3.2 Hasil divide and conquer dan brute force Test Case 3

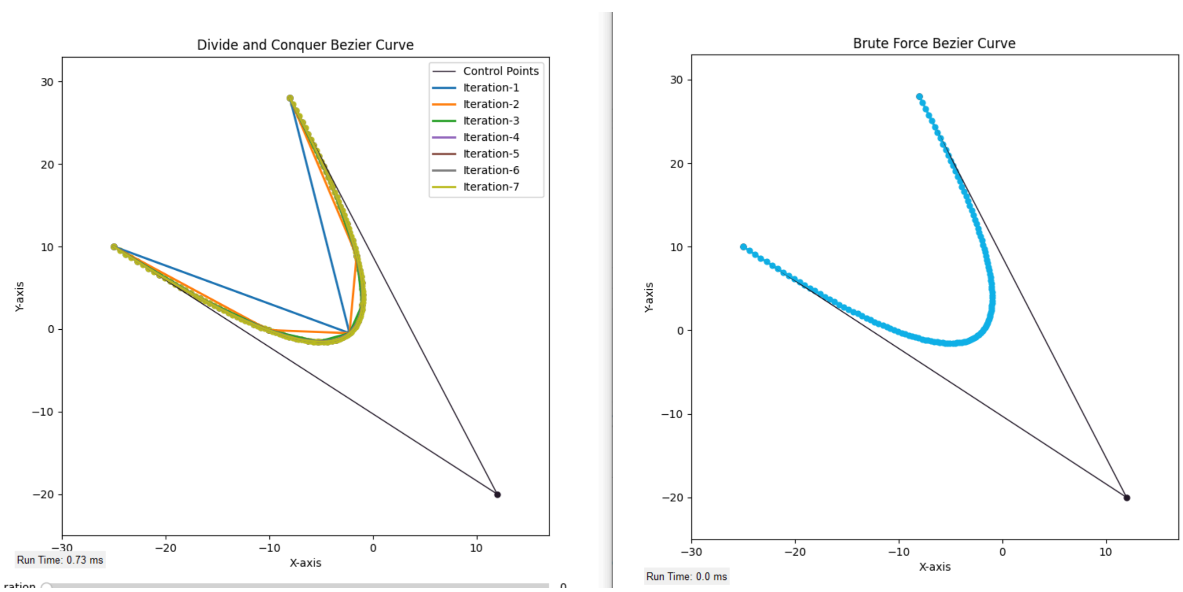
4.4 Test case 4 - 3 titik

Brute force runtime: 0.0 ms

Divide and Conquer runtime: 0.73 ms



Gambar 4.4.1 Input Test Case 4

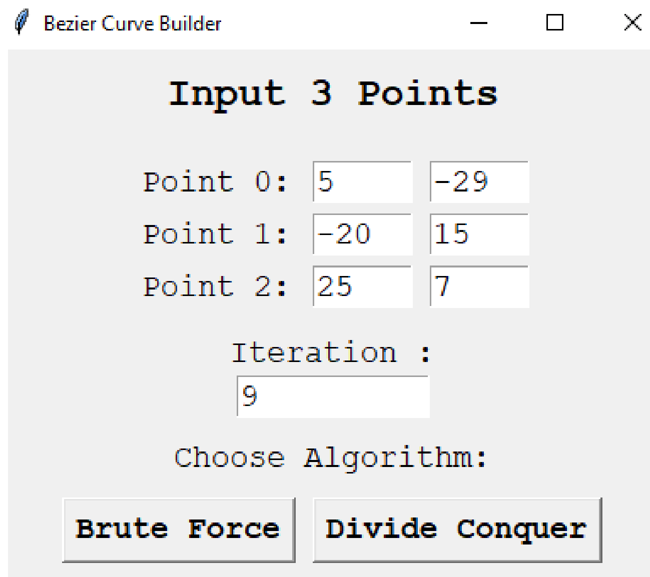


Gambar 4.4.2 Hasil divide and conquer dan brute force Test Case 4

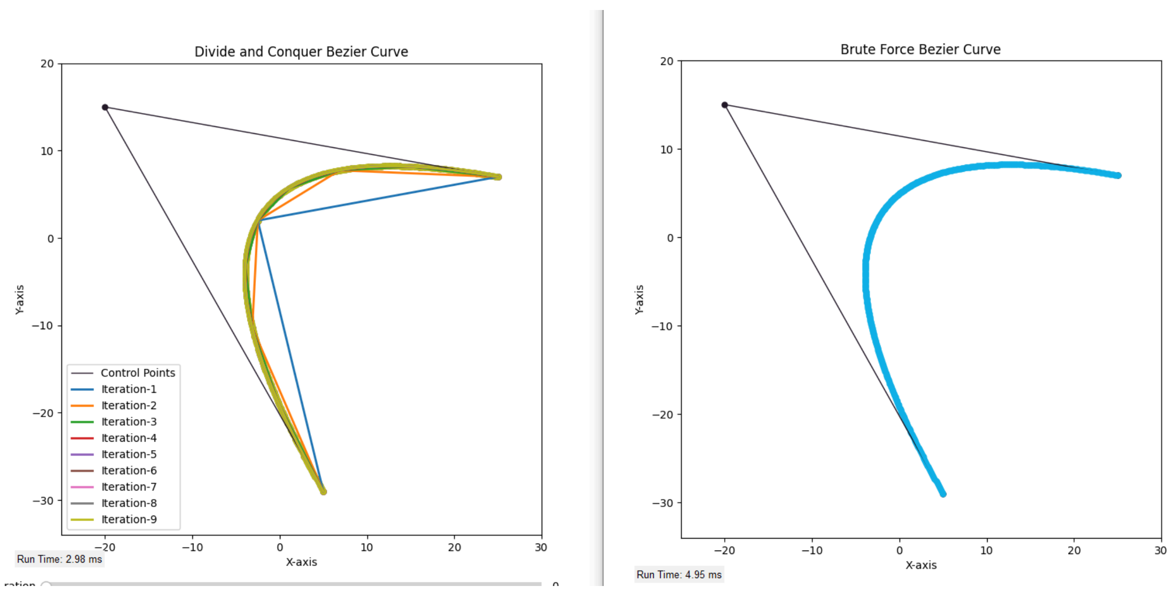
4.5 Test case 5 - 3 titik

Brute force runtime: 4.95 ms

Divide and Conquer runtime: 2.98 ms



Gambar 4.5.1 Input Test Case 5

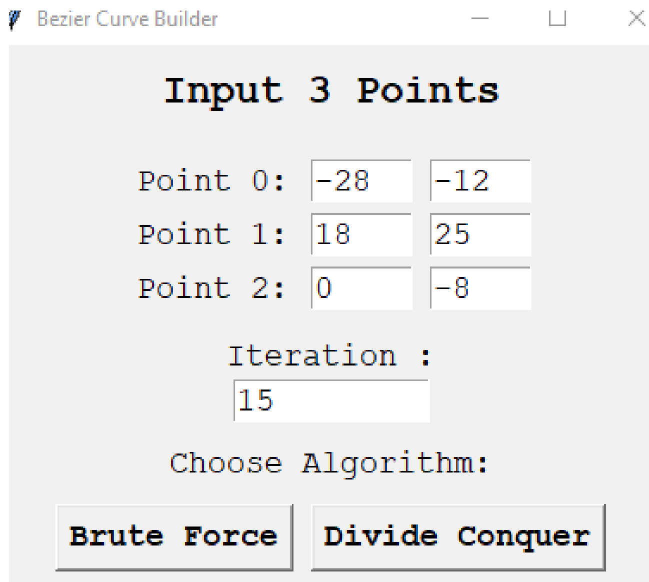


Gambar 4.5.2 Hasil divide and conquer dan brute force Test Case 5

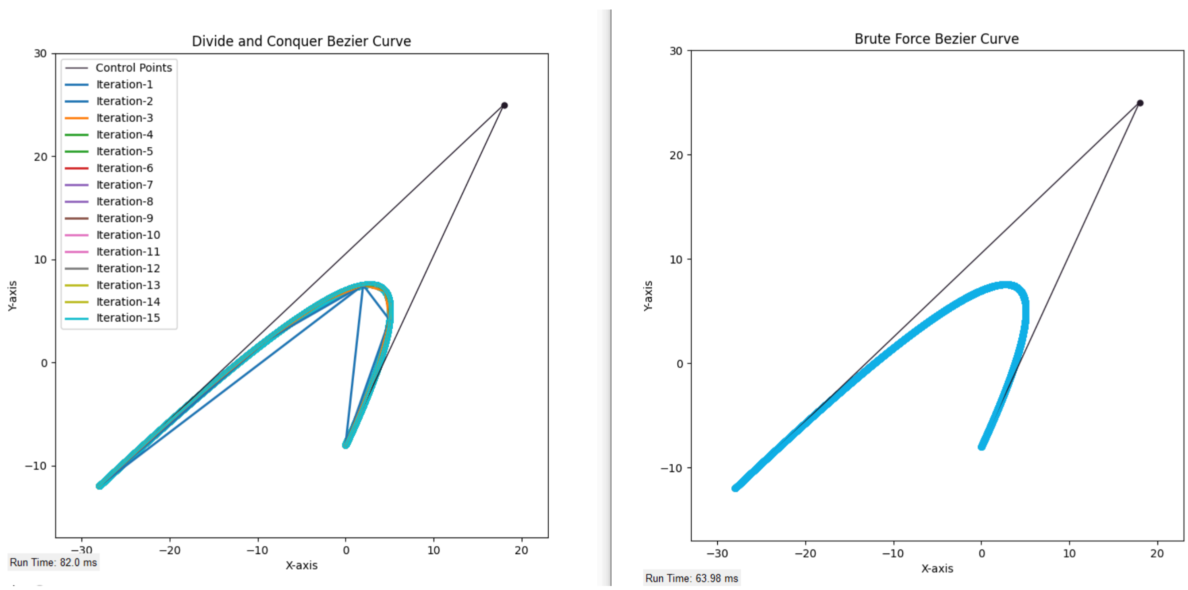
4.6 Test case 6 - 3 titik

Brute force runtime: 63.98 ms

Divide and Conquer runtime: 82 ms



Gambar 4.6.1 Input Test Case 6

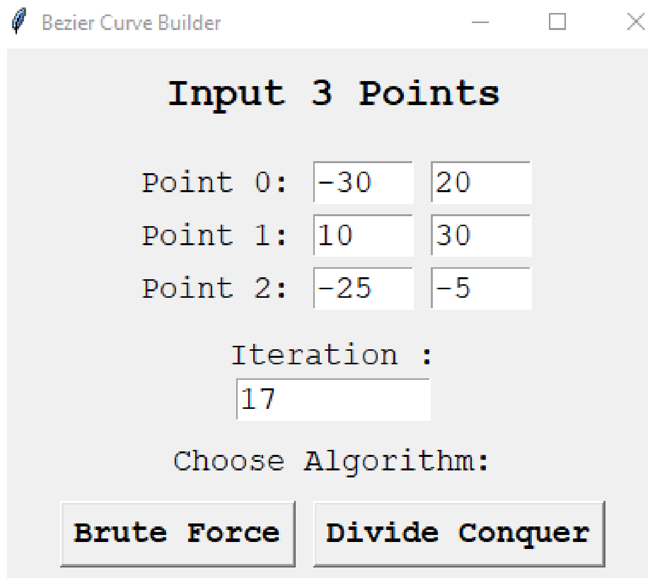


Gambar 4.6.2 Hasil divide and conquer dan brute force Test Case 6

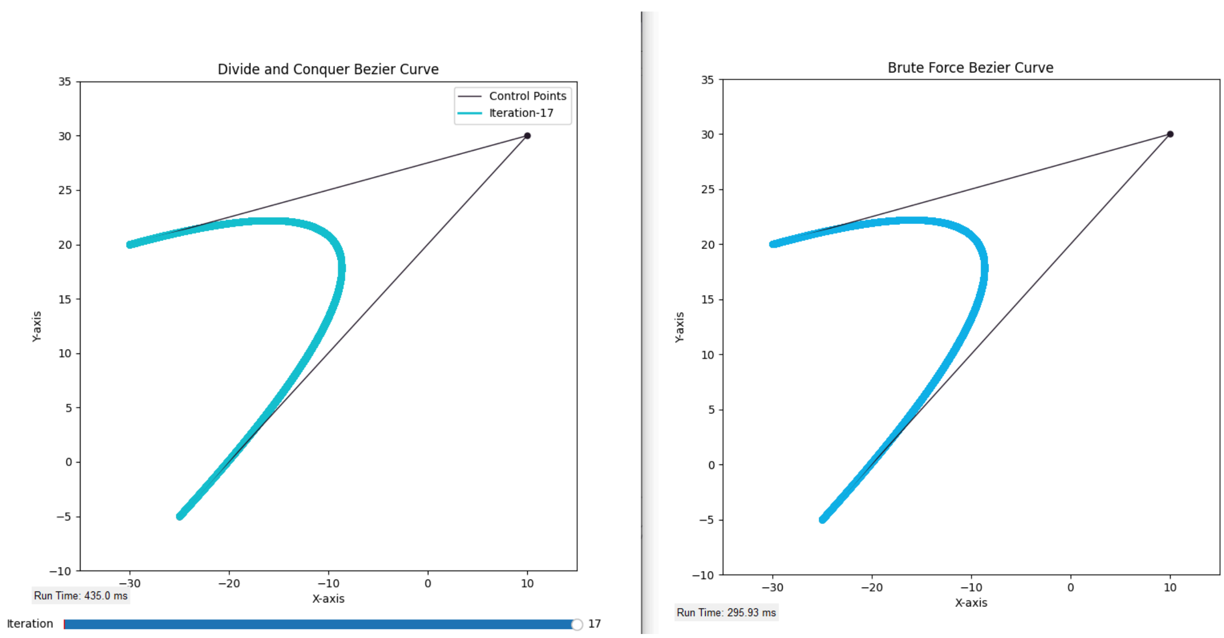
4.7 Test case 7 - 3 titik

Brute force runtime: 295.93 ms

Divide and Conquer runtime: 435.0 ms



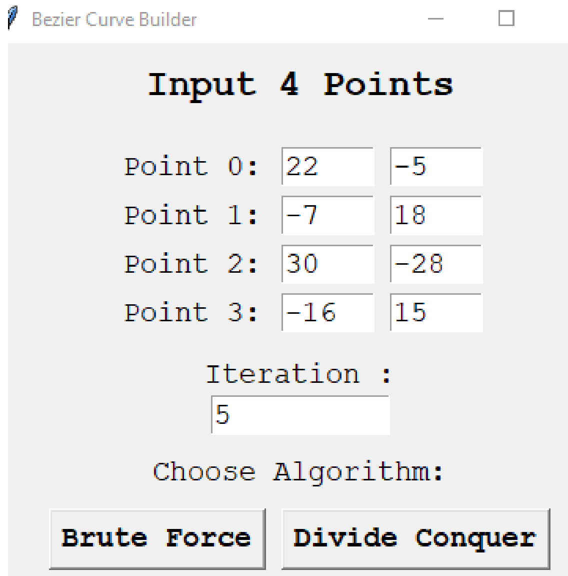
Gambar 4.7.1 Input Test Case 7



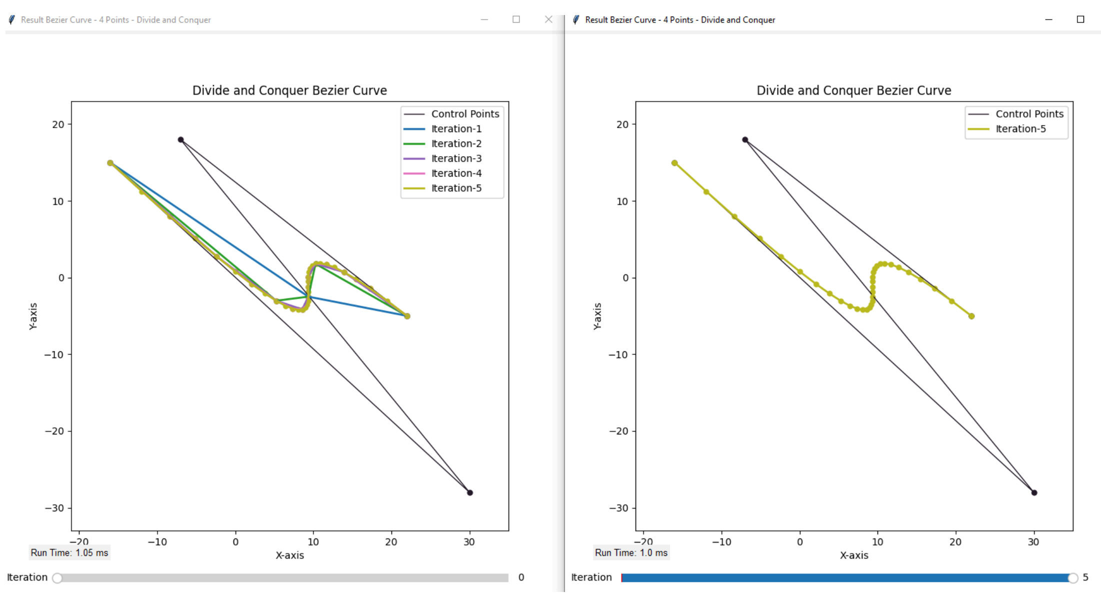
Gambar 4.7.2 Hasil divide and conquer dan brute force Test Case 7

4.8 Test case 8 - 4 titik

Divide and Conquer runtime avg: 1.25 ms



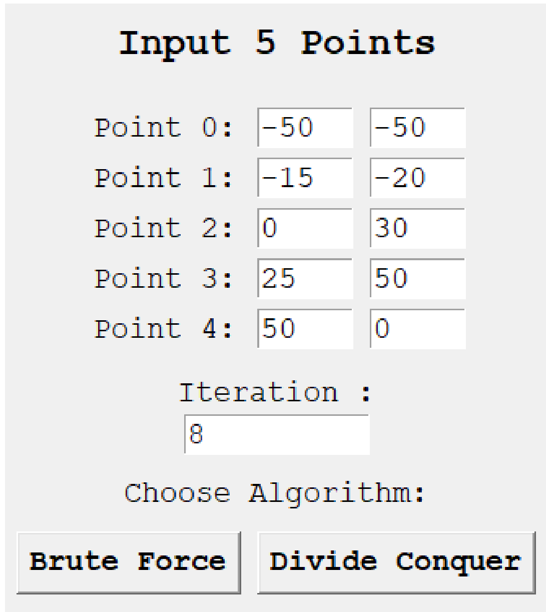
Gambar 4.8.1 Input Test Case 8



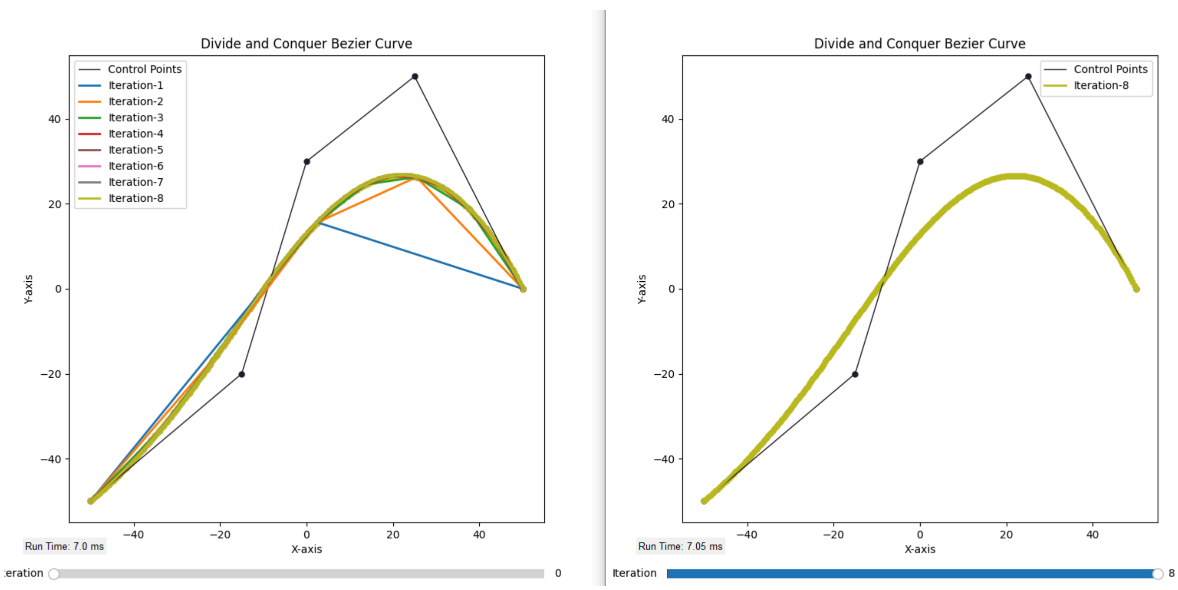
Gambar 4.8.2 Hasil *divide and conquer* Test Case 8

4.8 Test case 9 - 5 titik

Divide and Conquer runtime avg: 7.25 ms



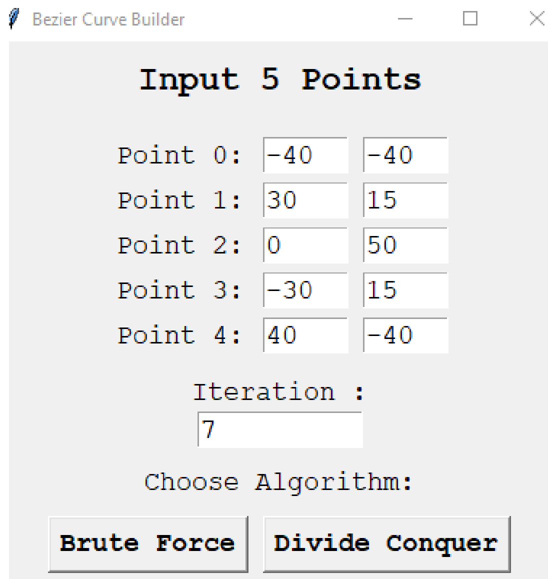
Gambar 4.9.1 Input Test Case 9



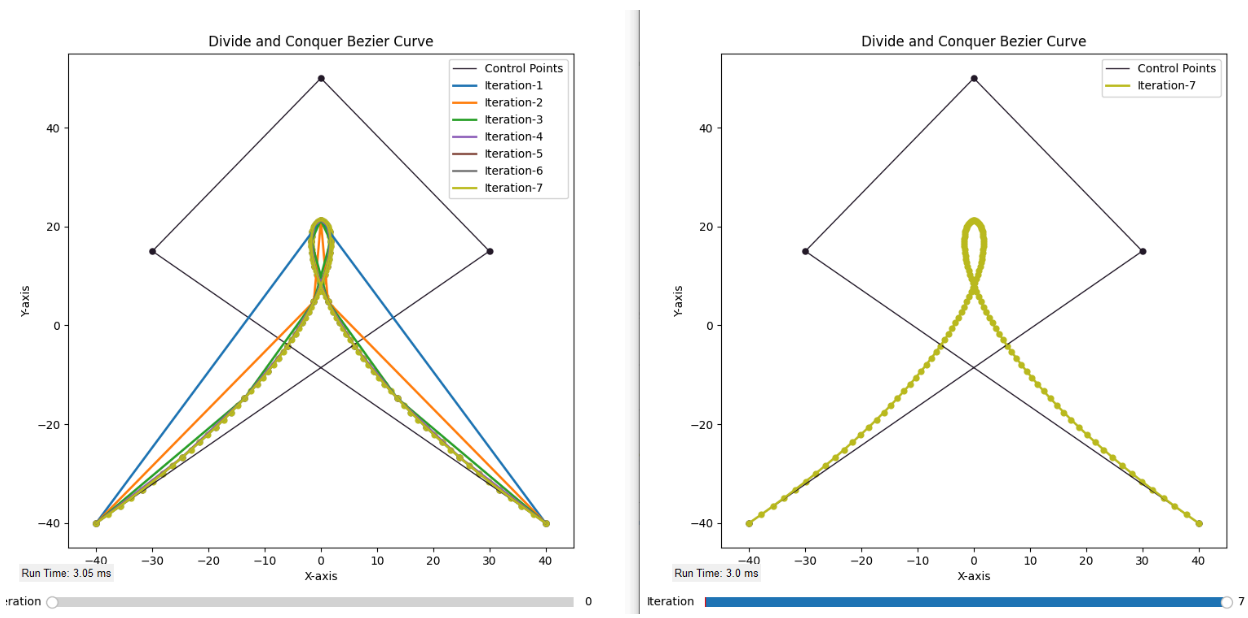
Gambar 4.9.2 Hasil *divide and conquer* Test Case 9

4.10 Test case 10 - 5 titik

Divide and Conquer runtime avg: 3.025 ms



Gambar 4.10.1 Input Test Case 10



Gambar 4.10.2 Hasil *divide and conquer* Test Case 10

BAB V
PERBANDINGAN DAN ANALISIS SOLUSI

Dengan melakukan analisis terhadap implementasi algoritma kami dapat melakukan perbandingan solusi hasil implementasi algoritma berbasis *Divide and Conquer* terhadap algoritma berbasis *Brute Force.*

Untuk algoritma berbasis *Brute Force*, kami menggunakan rangkain rumus yang dapat ditemukan pada deskripsi masalah. Dengan menggunakan rumus ini, kami mendapatkan 2 konsekuensi. Konsekuensi yang pertama dan yang baik adalah kompleksitas waktunya menjadi sangat cepat yaitu O(n). Hal tersebut didapat dengan karena pada tahap pencarian banyak dan tahap pembuatan titik pada kurva memiliki kompleksitas algoritma O(n). Konsekuensi yang buruknya adalah algoritma ini tidak bisa dipakai untuk n buah titik karena merupakan implementasi dari rumus yang diturunkan hanya untuk tiga titik.

Untuk algoritma berbasis *Divide and Conquer*, kami melakukan analisis terhadap kompleksitas waktu dalam komponen-komponen pembuatnya. Fungsi utama dalam membuat kurva bezier adalah fungsi intermediary_Bezier. Fungsi tersebut yang memanggil fungsi dcBuilder dua kali secara rekursif yang menunjukan kompleksitas waktu $O(2^m)$ dengan m adalah jumlah iterasi. Di dalam fungsi dcBuilder sendiri, terdapat pemanggilan fungsi make_mid_point yang memiliki kompleksitas algoritma O(n) dengan n adalah jumlah titik. Sehingga kami menemukan bahwa kompleksitas waktu algoritma *divide and conque*r adalah $O(2^m n)$ dengan m adalah jumlah iterasi dan n adalah banyak titik yang digunakan.

Untuk melakukan perbandingan antara kedua algoritma, kita terlebih dahulu perlu mendapatkan kompleksitas yang setara, yaitu kompleksitas saat terdapat 3 titik. Dikarenakan algoritma *brute force* hanya dapat menggunakan tiga titik, maka kita dapat membandingkannya dengan kompleksitas waktu algoritma *divide and conquer* pada tiga titik, yaitu $(3x2^m)$ atau $O(2^m)$. Terlihat bahwa algoritma *divide and conquer* jauh lebih lambat dibandingkan dengan algoritma *brute force*.

Hal tersebut terjadi karena penggunaan rumus yang sudah diturunkan untuk algoritma *brute force* sehingga algoritma *brute force* tersebut sudah sangat optimal dalam membangun kurva bezier. Algoritma *divide and conquer* yang optimal akan lebih cepat dibandingkan dengan algoritma *brute force*. Terlebihnya karena algoritma brute force yang sudah sangat optimal maka algoritma *divide and conquer* kami lebih lambat. Namun Hal tersebut tidak menunjukkan bahwa algoritma kami tidak optimal.

Dari alternatif algoritma *divide and conquer* yang kami eksplorasi dan setelah kami memilih yang terbaik. Algoritma kami tergolong optimal, jika kami bandingkan dengan algoritma berbasis *divide and conquer* yang lainnya, hanya saja jika dibandingkan dengan algoritma *brute force* yang didapatkan dari penurunan rumus, maka algoritma kami lebih lambat.

BAB VI
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dalam pengerjaan tugas ini, kami mendapat pelajaran dalam perancangan algoritma *divide and conquer* untuk menyelesaikan masalah kurva Bézier. Kami melakukan banyak eksplorasi alternatif solusi dan memilih solusi yang optimal. Namun solusi kami masih jauh lebih lambat jika dibandingkan dengan algoritma *brute force* yang sudah sangat optimal dikarenakan menggunakan rumus yang merupakan penurunan dari permasalahan kurva Bézier.

Dengan demikian, kami menyimpulkan bahwa melalui Tugas Kecil II IF2211 Strategi Algoritma ini, dapat dibuat sebuah algoritma titik tengah berbasis *Divide and Conquer* untuk membangun kurva Bézier dari n titik.

5.2 Saran

Tugas ini menjadi pengalaman yang memberikan pelajaran baru bagi kami. Berdasarkan pengalaman kami, berikut adalah saran untuk kelompok ini, diantaranya:

1. Membuat *timeline* pengerjaan tugas yang lebih teratur
2. Melakukan pembagian tugas yang jelas dan teratur
3. Memperbanyak diskusi dalam pengerjaan tugas
4. Mendalami pengetahuan mengenai implementasi algoritma *Divide and Conquer* dengan lebih baik

LAMPIRAN

Pranala Repository Github

https://github.com/NoHaitch/Tucil2_13522078_13522091

Checklist Fitur

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.	✓	

DAFTAR PUSTAKA

Rinaldi Munir. Algoritma Divide and Conquer (Bagian 1). Diakses pada 16 Maret 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf

Rinaldi Munir. Algoritma Divide and Conquer (Bagian 2). Diakses pada 16 Maret 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf

Rinaldi Munir. Algoritma Divide and Conquer (Bagian 3). Diakses pada 16 Maret 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf

Rinaldi Munir. Algoritma Divide and Conquer (Bagian 4). Diakses pada 16 Maret 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf

Geeksforgeeks. Introduction to Divide and Conquer Algorithm – Data Structure and Algorithm Tutorials. Diakses pada 16 Maret 2024 dari https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm-data-structure-and-algorithm-tutorials/

Mateus Melo. Understanding Bézier Curves. Diakses 17 Maret 2024 dari https://mmrndev.medium.com/understanding-b%C3%A9zier-curves-f6eaa0fa6c7d

3D Computer Graphics: Math Intro w/ OpenGL. 11 04 Recursive Subdivision for Degree Three Bezier Curves. Diakses 17 Maret 2024 dari https://www.youtube.com/watch?v=pQ2n60pUphc

codiecodemonkey. Computing Bezier curves using de Casteljau’s algorithm. Diakses 17 Maret 2024 dari https://www.youtube.com/watch?v=YATikPP2q70&t=1s