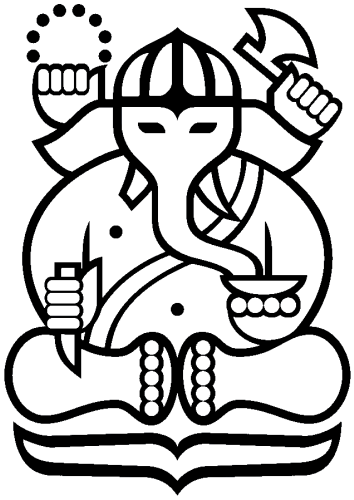
Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Semester II Tahun Akademik 2023/2024

Membangun Kurva Bezier dengan Algoritma Titik Tengah berbasis Divide and Conquer



Disusun oleh:

Muhammad Gilang Ramadhan

13520137

K01

Program Studi S1 Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

BAB I. Algoritma *Brute force* untuk membangun Kurva Bezier

Melalui definisi kurva bezier, untuk membuat suatu titik baru dari n buah garis poligon yang dapat menghampiri kurva bezier dengan segmen garis yang terhubung oleh n+1 titik, maka dapat kita nyatakan dalam persamaan matematika berikut.

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \\ &= (1-t)^n \mathbf{P}_0 + \binom{n}{1}(1-t)^{n-1} t\mathbf{P}_1 + \cdots + \binom{n}{n-1}(1-t)t^{n-1}\mathbf{P}_{n-1} + t^n \mathbf{P}_n, \qquad 0 \leqslant t \leqslant 1\end{aligned}$$

Yang mana hal tersebut tidak lain merupakan ekspresi persamaan binomial newton yang bisa kita hitung melalui program untuk suatu titik kontrol baru p_{n+1}, yang mana nilai t pada persamaan tersebut dalam kasus tugas ini adalah 0.5 atau 1/2.

Akibatnya, dalam hal ini yang perlu dilakukan adalah menghitung setiap nilai koefisien binomial dari persamaan tersebut. Karena, dalam pembangunan suatu kurva bezier, nilai koefisien binomial tersebut dapat dijadikan bobot relatif terhadap setiap titik kontrol pada titik-titik hasil yang diperoleh untuk mendapatkan hasil kurva bezier.

Dalam implementasinya terhadap nilai binomial tersebut, dapat dilakukan rekursi terhadap 2 parameter suatu fungsi kombinasi secara brute force juga, yaitu parameter n dan k pada suatu C(n,k) yang menyatakan besarnya bobot titik kontrol ke-k terhadap suatu titik lain pada kurva, Yang jika dihitung time complexitynya adalah $O(\frac{n^k}{k!})$.

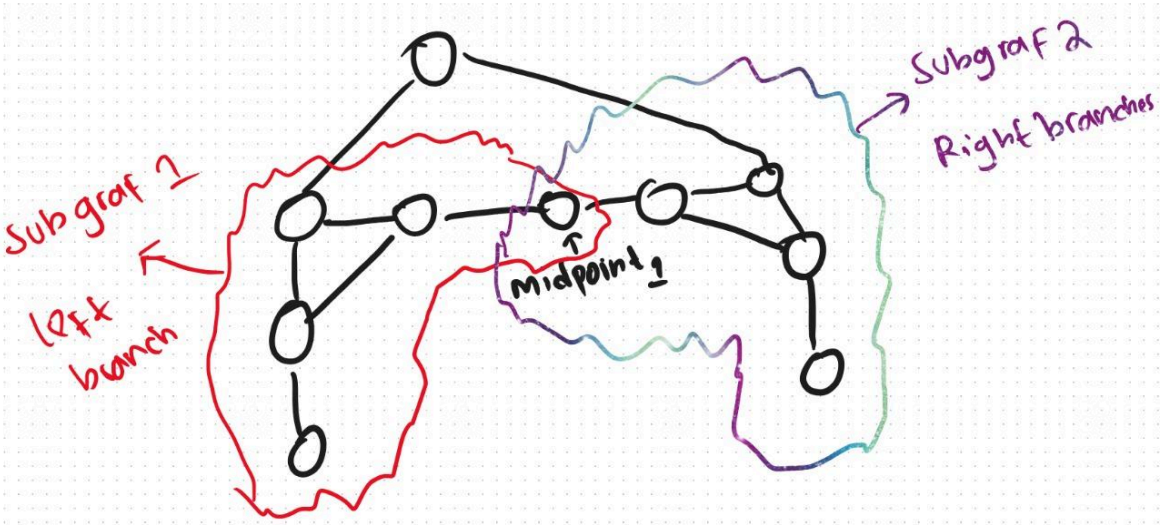
Setelah itu, pada setiap iterasi untuk setiap urutan titik dari titik ke-0 sampai dengan titik ke-n, kita dapat mengalikan nilainya seperti biasa dengan suatu value titik kontrol ke-i dan nila t=1/2. Kemudian simpan titik baru tersebut sebagai titik pada kurva bezier.

Ulangi langkah tersebut untuk titik-titik kontrol yang lain sampai sebanyak number of points dari kurva bezier yang akan dihasilkan tercapai. Yang mana hal tersebut sesuai dengan definisi dan logika dasar dalam menyelesaikan persamaan aljabar pada suatu kurva bezier. Total Complexity O(N*M* $\frac{n^k}{k!}$) = dimana untuk untuk n > k jelas kompleksitasnya menjadi exponential. Yang mana akan berpengaruh terhadap lambatnya performa pembangunan kurva bezier. Dalam kasus permasalahan ini, jumlah titik pada kurva bezier dan iterasi yang iniginkan dapat disesuaikan melalui T = 2^{ns} dengan ns ada banyaknya titik solusi pada kurva bezier.

BAB II. Algoritma *Divide and Conquer* untuk membangun Kurva Bezier (Beserta Bonus)

Adapun Ide utama dalam mengkonstruksi pembangunan kurva bezier dengan algoritma ini ialah dengan membagi dua bagian (branch) dari setiap array of titik kontrol yang ada pada saat mencari titik tengah dari masing-masing *n* buah titik.

Misalnya perhatikan ilustrasi graf berikut.



Gambar 2.1 Skema Divide Graf menjadi 2 Sub-graf
(Sumber Dokumentasi Penulis)

Caranya ialah dengan melakukan traversal secara mendalam dari graf level paling atas untuk setiap midpoint dari masing-masing subgraf. Yang mana untuk subgraf 1, untuk mendapatkan titik dari kurva bezier untuk iterasi ke-n ialah dengan mengambil titik midpoint pertama untuk setiap subgraf 1 (Left branch), jadi dari iterasi ke-0 sampai ke-n kita cukup ambil midpoint pertamanya dari kiri ke kanan untuk setiap left branch tersebut. Kemudian untuk right branch kebalikannya, kita ambil midpoint urutan terakhir yang terurut dari terkecil ke terbesar dari kiri ke kanan. Maka untuk right branch, titik midpoint yang diambil ialah dari kanan ke kiri untuk setiap iterasi ke-0 sampai ke-n.

Demikian iterasi tersebut selesai jika midpoint yang diperoleh hanya satu, akibatnya tidak ada lagi subgraf yang bisa diiterasi (karena hanya 1).

Untuk implementasinya bisa menggunakan berbagai macam cara, untuk dalam tugas ini penulis menggunakan cara rekursi dengan menyimpan left branch dan right branch dari setiap iterasi kemudian akan di-inisialisasi sebagai initial points untuk setiap iterasi tersebut. Yang mana left branch dan right branch tersebut akan di-update ketika diperoleh titik kontrol baru yang urutan memasukkan titik kontrol tersebut juga mengikuti urutan dari branch masing-masing. Misalnya, untuk left branch, maka titik akan disimpan dari kiri ke kanan untuk setiap update titik kontrol baru.

Untuk mengambil semua titik yang dilalui kurva, kita cukup ambil next control point yang merupakan titik kontrol baru pada suatu iterasi. Yang mana jika untuk membuat kurva dengan n iterasi, maka kita cukup ambil next control point tersebut pada rentang iterasi 0 sampai dengan n. Untuk next control point pada iterasi > n tidak perlu diambil karena sudah melewati batas iterasi yang diinginkan.

Adapun tahap **Conquer** yang dilakukan ialah terletak pada urutan pemasukkan elemen koordinat next control point pada array solusi kurva bezier yang berindeks current iteration sampai dengan jumlah iterasi, yaitu dapat dilihat melalui kode berikut.

```
# Append next control points on the bezier curve points
if(curr_iteration<=self.iteration):
    for i in range(curr_iteration, self.iteration+1):
        self.bezier_curve_points[i].append(next_control_point)
```

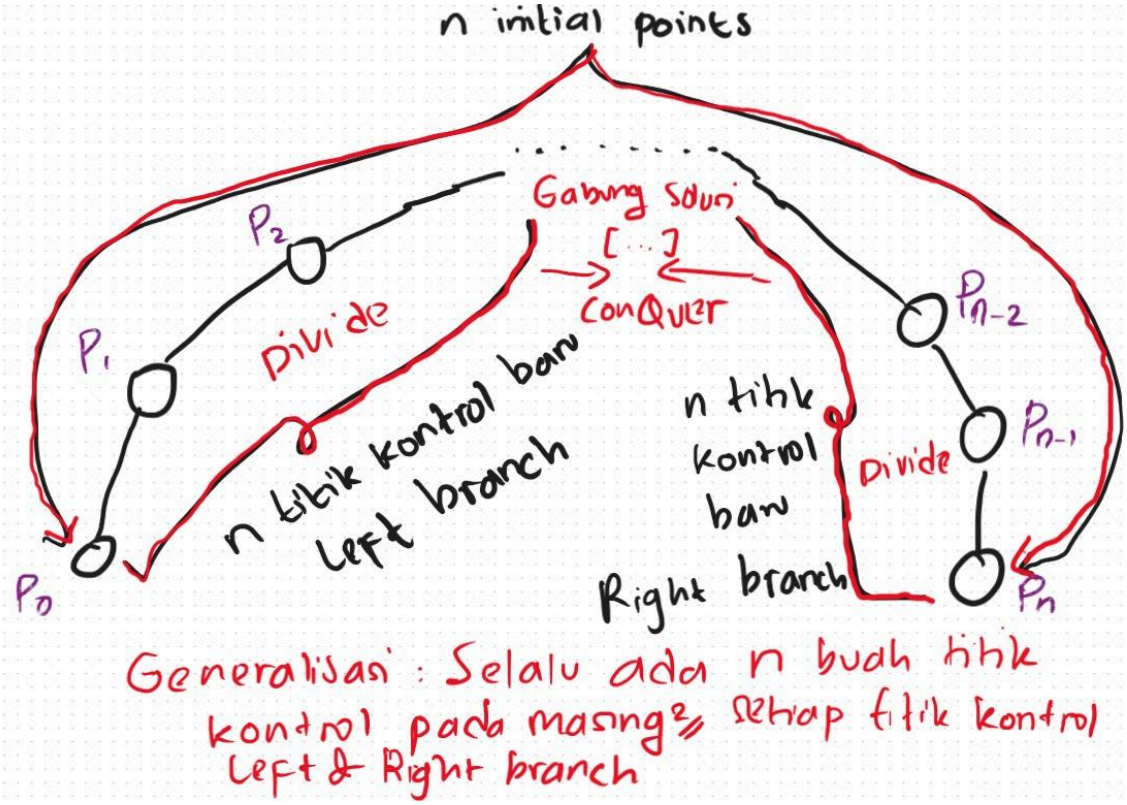
Core Bonus

Adapun penjelasan traversal left branch dan right branch yang dijelaskan sebelumnya tersebut juga bisa menangani kasus untuk untuk pembangunan kurva bezier dengan n initial points yang mana n yang digunakan bisa general. Untuk generalisasi tersebut, letaknya pada pengambilan kontrol point yang pada base case itu biasanya diimplementasikan dalam 3 initial points. Sedangkan yang diimplementasikan oleh penulis itu bisa mengambil n initial points yang mana untuk setiap traversal sub graf left branch dan right branch-nya akan berakibat pada pembagian jumlah titik kontrol per-traversal masing-masing sub grafnya ialah n buah titik. Untuk setiap titik kontrol yang dihasilkan baru pastilah ada n titik kontrol left branch dan n titik kontrol right

branch (include titik kontrol bersama jika ada). Untuk implementasi bonus prosess visualisasi yang disajikan, dilakukan juga cukup ambil initial points dari tiap-tiap rekursi pada setiap iterasi untuk disimpan pada array generated points sesuai indeks iterasinya saja. Kemudian pada saat visualisasi kurva barulah array generated points tersebut dipanggil setiap koordinatnya untuk divisualisasikan kurva garis yang dibentuk olehnya, yaitu dapat dilihat melalui potongan kode berikut.

```
# Append the initial points to the bezier generated points
if self.is_visualize == "y":
    for i in range(sz_initial_points):
        self.bezier_generated_points[curr_iteration].append(initial_points[i])
```

Dengan demikian, secara general untuk n initial point, skema tersebut dapat diilustrasikan melalui graf berikut.



Gambar 2.2 Skema Generalisasi untuk N points

(Sumber Dokumentasi Penulis)

BAB III. Source Code dalam Bahasa Python

```
import time
from dataclasses import dataclass
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import random
import numpy as np

@dataclass
class Point:
x: int
y: int
```

Gambar 3.1 Insialisasi Data Class Point

```
class CurveBezierSol():
def __init__(self):
# Initialize the input values
self.initial_points = []
print("Silahkan Pilih Algoritma yang ingin dicoba: ")
print("1. Brute Force")
print("2. Divide and Conquer")
self.algoritma = str(input("Pilih Algoritma [1,2]: "))
if(self.algoritma=="1"):
print("Menggunakan Algoritma Brute Force")
else:
print("Menggunakan Algoritma Divide and Conquer")
command = self.inputCommandHandling()
if(command=="2"):
file_content = self.fileInputHandling()

# Extract values from the input string
self.n = int(file_content[0]) # n is the first line
x_values = list(map(float, file_content[1].split()))
y_values = list(map(float, file_content[2].split()))
self.iteration = int(file_content[3])
for i in range(self.n):
self.initial_points.append(Point(x_values[i], y_values[i]))
# Print the extracted values
print("N:", self.n)
print("Points:", self.initial_points)
if(self.algoritma=="2"):
```

```
print("Iteration:", self.iteration)
else:
print("Jumlah Point:", self.iteration)
else:
self.manualInputHandling()
if(self.algoritma=="2"):
self.is_visualize = str(input("Apakah ingin melihat proses visualisasi? [y/n]: "))
self.bezier_generated_points = [[] for i in range(self.iteration+1)]
self.bezier_curve_points = [[] for i in range(self.iteration+1)]
```

Gambar 3.2 Insialisasi Inputan pada constructor di Main Class Solution

```
def inputCommandHandling(self):
# Get the input command from the user
print("Input Options: ")
print("1. Manual input")
print("2. File input")
command = str(input("Please choose format input file [1,2]: "))
while(command != "1" and command != "2"):
print("Unknown Input Command, please try again!")
print("Input Options: ")
print("1. Manual input")
print("2. File input")
command = str(input("Please choose format input file [1,2]: "))
return command

def fileInputHandling(self):
# Read the file input
with open('../test/input.txt', 'r') as file:
file_content = file.read().strip().split('\n')
return file_content

def manualInputHandling(self):
# Get the input from the user
if self.algoritma=="1":
self.n = 3
else:
self.n = int(input("Masukkan n: "))
temp_brute_or_dnc = (self.algoritma=="1" and "Masukkan jumlah point: ") or "Masukkan
jumlah iterasi: "
self.iteration = int(input(temp_brute_or_dnc))
for i in range(self.n):
```

```
x = float(input("Masukkan Px"+str(i)+": "))
y = float(input("Masukkan Py"+str(i)+": "))
self.initial_points.append(Point(x,y))
```

Gambar 3.3 Input CLI dan File Handling

```
def get_midpoint(self, a: Point, b: Point):
# Get the midpoint of two points
return Point((a.x+b.x)/2,(a.y+b.y)/2)
```

Gambar 3.4 Fungsi untuk menghitung jarak 2 buah titik

```
def binomial_coefficient(self, n, k):
# Get the binomial coefficient
if k == 0 or k == n:
return 1
return self.binomial_coefficient(n - 1, k - 1) + self.binomial_coefficient(n - 1, k)

def generate_bezier_points_brute_force(self, ctrl_points, num_points):
# Generate the bezier points using brute force
n = len(ctrl_points) - 1
t_values = [i / (num_points - 1) for i in range(num_points)]
curve_points = []

for t in t_values:
point = Point(0, 0)
for k in range(n + 1):
coefficient = self.binomial_coefficient(n, k) * (1 - t) ** (n - k) * t ** k
point = Point(point.x + coefficient * ctrl_points[k].x, point.y + coefficient *
ctrl_points[k].y)
curve_points.append(point)

return curve_points
```

Gambar 3.5 Method untuk meng-generate bezier curve secara brute force

```
def get_control_point(self, points: list[Point], curr_iteration, left_branches,
right_branches):
sz_points = len(points)
```

```
if sz_points==1: # If left and right branches are empty, then it is the base case with
only one point
return points[0], left_branches, right_branches

# Get all midpoints from current control points
next_control_points = []
for i in range(sz_points-1):
midpoint = self.get_midpoint(points[i], points[i+1])
next_control_points.append(midpoint)

# Divide the control points into left and right branches
left_branches.append(next_control_points[0])
right_branches.append(next_control_points[-1])

# Recursively call the function to get the next control points
return   self.get_control_point(next_control_points,   curr_iteration,   left_branches,
right_branches)

def generate_bezier_points_dnc(self, initial_points, curr_iteration):
# Base case if the current iteration is greater than the maximum iteration ()
if curr_iteration > self.iteration:
return

# Get the next control points and left and right branches
next_control_point,             left_branches,             right_branches             =
self.get_control_point(initial_points,       curr_iteration,       [initial_points[0]],
[initial_points[-1]])

# Reverse the right branches
right_branches = right_branches[::-1]

# Get the size of the initial points
sz_initial_points = len(initial_points)

# Append the initial points to the bezier generated points
for i in range(sz_initial_points):
self.bezier_generated_points[curr_iteration].append(initial_points[i])

# Append the next control points to the bezier generated points in the next iteration
curr_iteration += 1
# Recursively left branches to get the next control points on the left
self.generate_bezier_points_dnc(left_branches, curr_iteration)
```

```
# Append next control points on the bezier curve points
if(curr_iteration<=self.iteration):
for i in range(curr_iteration, self.iteration+1):
self.bezier_curve_points[i].append(next_control_point)
# Recursively right branches to get the next control points on the right
self.generate_bezier_points_dnc(right_branches, curr_iteration)
```

Gambar 3.6 Method untuk menggenerate kurva dan titik kontrol dengan DNC Algorithm

```
def get_bezier_solution(self):
if(self.algoritma=="1"):
self.bezier_curve_points[-1]                                                    =
self.generate_bezier_points_brute_force(self.initial_points, self.iteration)
else:
# Append the initial points to the bezier curve points
for i in range(self.iteration+1):
self.bezier_curve_points[i].append(self.initial_points[0])

# Generate the bezier points using divide and conquer
self.generate_bezier_points_dnc(self.initial_points, 0)

# Append the last points to the bezier curve points
for i in range(self.iteration+1):
self.bezier_curve_points[i].append(self.initial_points[-1])
```

Gambar 3.7 Method untuk mendapatkan solusi bezier curve dalam bentuk array of point

```
def visualize_all(self):
# Visualize the bezier generated points and bezier curve points
# sz_generated_points = len(self.bezier_generated_points)
visualize_points_x = []
visualize_points_y = []

if(self.algoritma=="1"):
self.is_visualize = "n"
temp_x = []
temp_y = []
for j in range(len(self.bezier_curve_points[-1])):
temp_x.append(self.bezier_curve_points[-1][j].x)
```

```
temp_y.append(self.bezier_curve_points[-1][j].y)
visualize_points_x.append(temp_x)
visualize_points_y.append(temp_y)
else:
if self.is_visualize == "y":
for i in range(self.iteration+1):
temp_x = []
temp_y = []
for j in range(len(self.bezier_generated_points[i])):
temp_x.append(self.bezier_generated_points[i][j].x)
temp_y.append(self.bezier_generated_points[i][j].y)
visualize_points_x.append(temp_x)
visualize_points_y.append(temp_y)
temp_x = []
temp_y = []
for j in range(len(self.bezier_curve_points[i])):
temp_x.append(self.bezier_curve_points[i][j].x)
temp_y.append(self.bezier_curve_points[i][j].y)
visualize_points_x.append(temp_x)
visualize_points_y.append(temp_y)
else:
temp_x = []
temp_y = []
for j in range(len(self.bezier_curve_points[-1])):
temp_x.append(self.bezier_curve_points[-1][j].x)
temp_y.append(self.bezier_curve_points[-1][j].y)
visualize_points_x.append(temp_x)
visualize_points_y.append(temp_y)
# Initialize the plot
fig, ax = plt.subplots()
length_visualize_points = len(visualize_points_x)
for i in range(length_visualize_points):
line, = ax.plot(visualize_points_x[i], visualize_points_y[i])
points, = ax.plot(visualize_points_x[i], visualize_points_y[i], 'o')

# Randomize line and point colors
line_color        =        ((i%2==1        or        self.is_visualize=='n')        and
"#{:06x}".format(random.randint(0, 0xFFFFFF), label="Point") or "black")
point_color = "#{:06x}".format(random.randint(0, 0xFFFFFF), label="Point")

# Set line and point colors
line.set_color(line_color)
```

```
points.set_color(point_color)
for j in range(len(visualize_points_x[i])):
line.set_data(visualize_points_x[i][:j+1], visualize_points_y[i][:j+1])
points.set_data(visualize_points_x[i][:j+1], visualize_points_y[i][:j+1])
plt.pause(0.4)
plt.draw() # Update the plot
# Adding labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Bezier Curve Visualization')
plt.ioff()

# Display the final plot
plt.show()
```

Gambar 3.8 Method untuk melakukan visualisasi terhadap Kurva maupun titik (Include Bonus)

```
def main():
test = CurveBezierSol()
start_time = time.time()
test.get_bezier_solution()
end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
test.visualize_all()

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

Gambar 3.9 Fungsi utama untuk menjalankan Program

BAB IV. Pengujian (Input/Output)

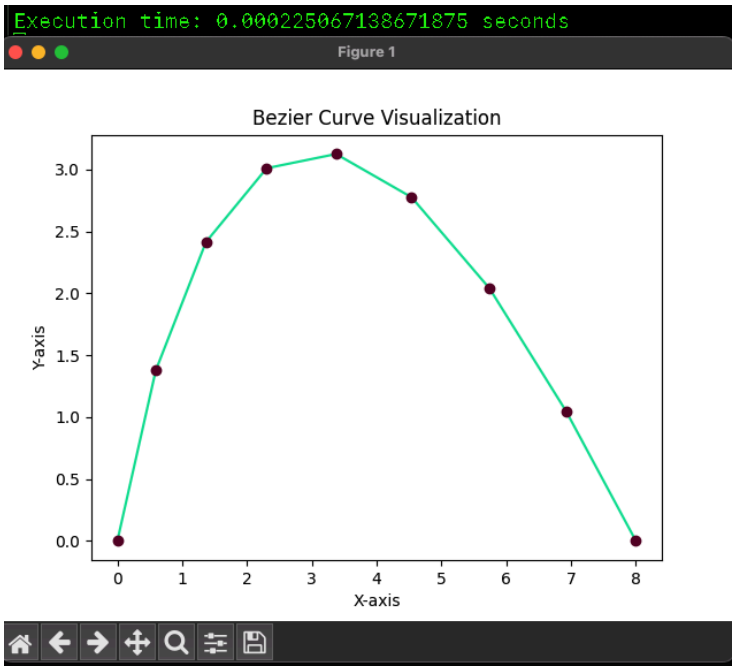
4.1. Test Case 1

Input Brute Force

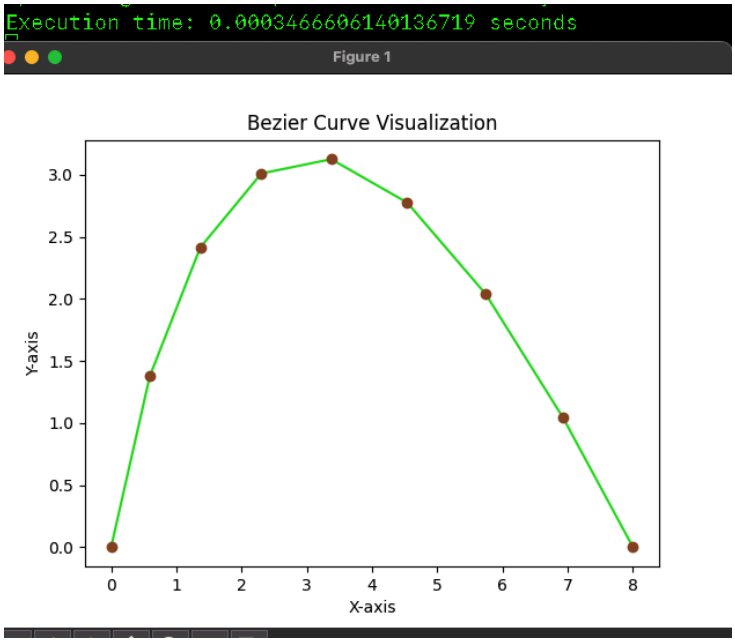
```
5
0.0 1.0 3.0 6.0 8.0
0.0 3.0 5.0 2.0 0.0
9
```

Input Divide and Conquer

```
5
0.0 1.0 3.0 6.0 8.0
0.0 3.0 5.0 2.0 0.0
3
```



Gambar 4.1 Output Test Case 1 Brute Force



Gambar 4.2 Output Test Case 1 Divide and Conquer

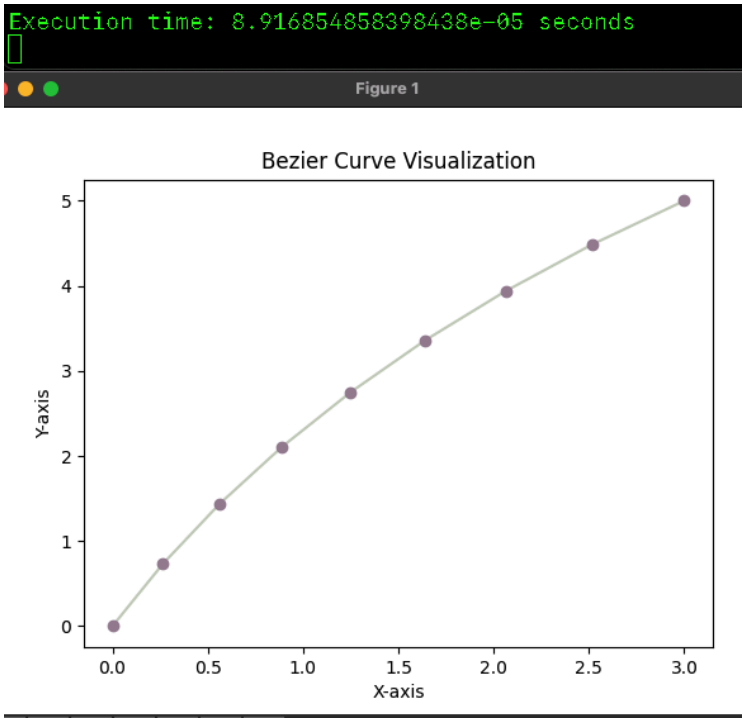
4.2. Test Case 2

Input Brute Force

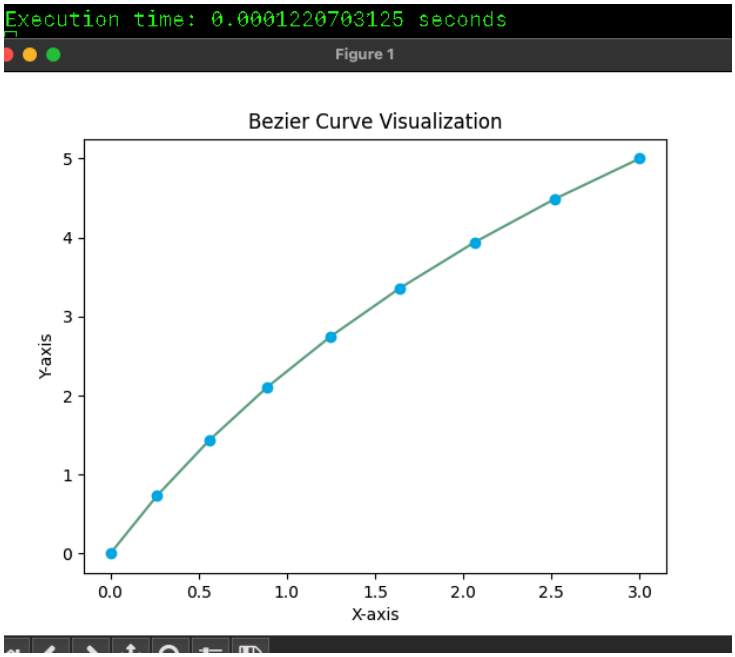
```
3
0.0 1.0 3.0
0.0 3.0 5.0
9
```

Input Divide and Conquer

```
3
0.0 1.0 3.0
0.0 3.0 5.0
3
```



Gambar 4.3 Output Test Case 2 Brute Force



Gambar 4.4 Output Test Case 2 Divide and Conquer

4.3. Test Case 3

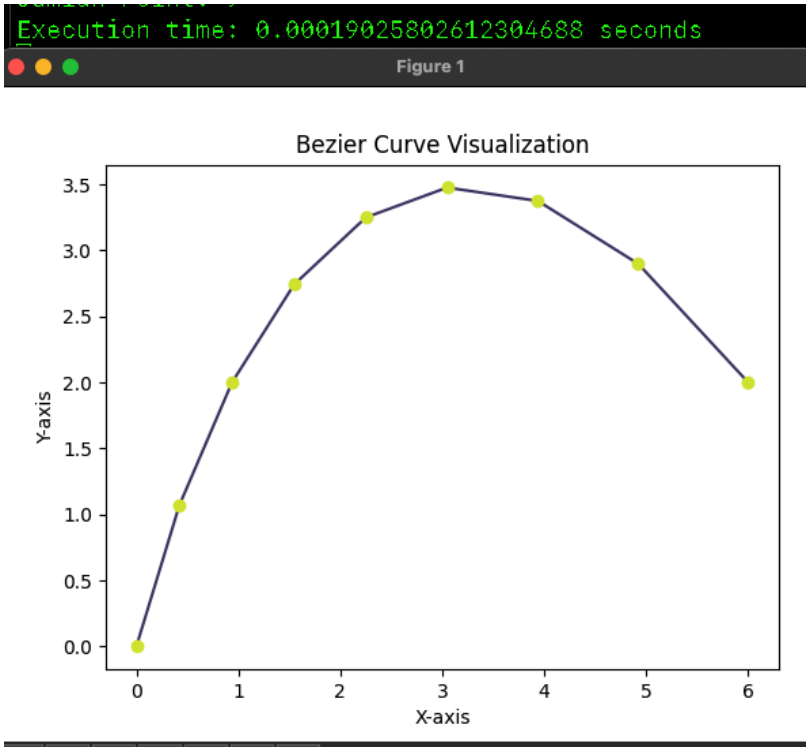
Input Brute Force

4

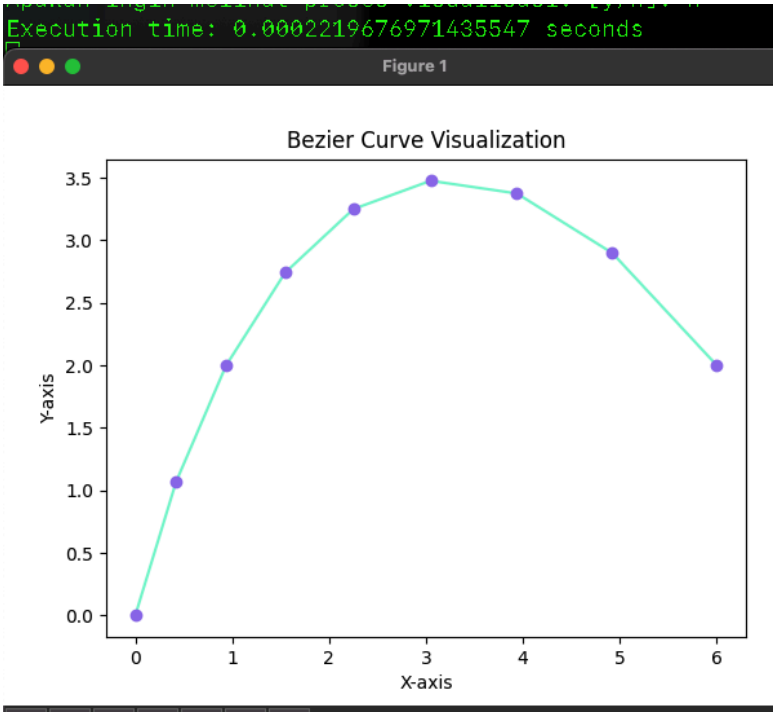
```
0.0 1.0 3.0 6.0
0.0 3.0 5.0 2.0
9
```

Input Divide and Conquer

```
4
0.0 1.0 3.0 6.0
0.0 3.0 5.0 2.0
3
```



Gambar 4.5 Output Test Case 3 Brute Force



Gambar 4.6 Output Test Case 3 Divide and Conquer

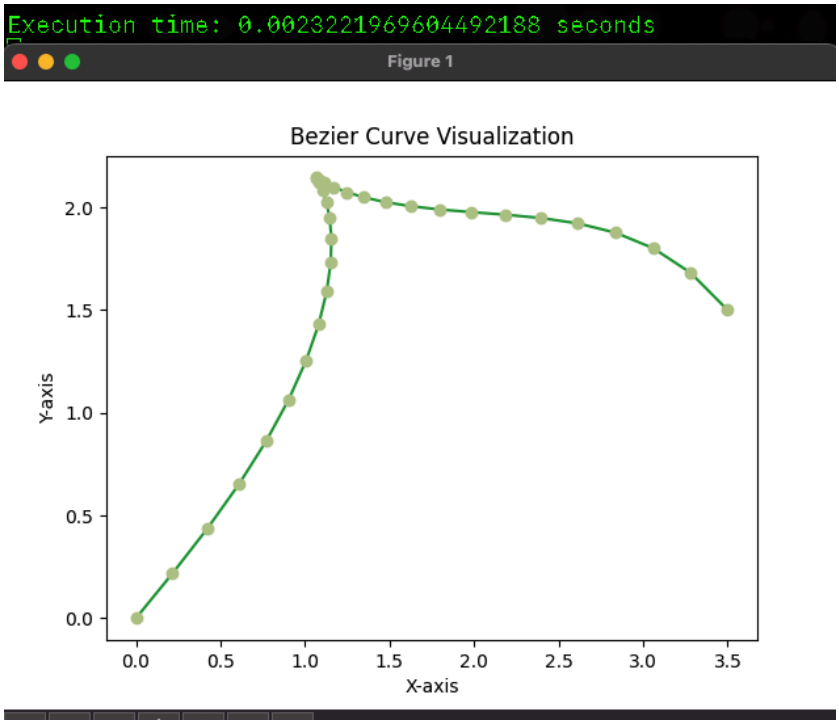
4.4. Test Case 4

Input Brute Force

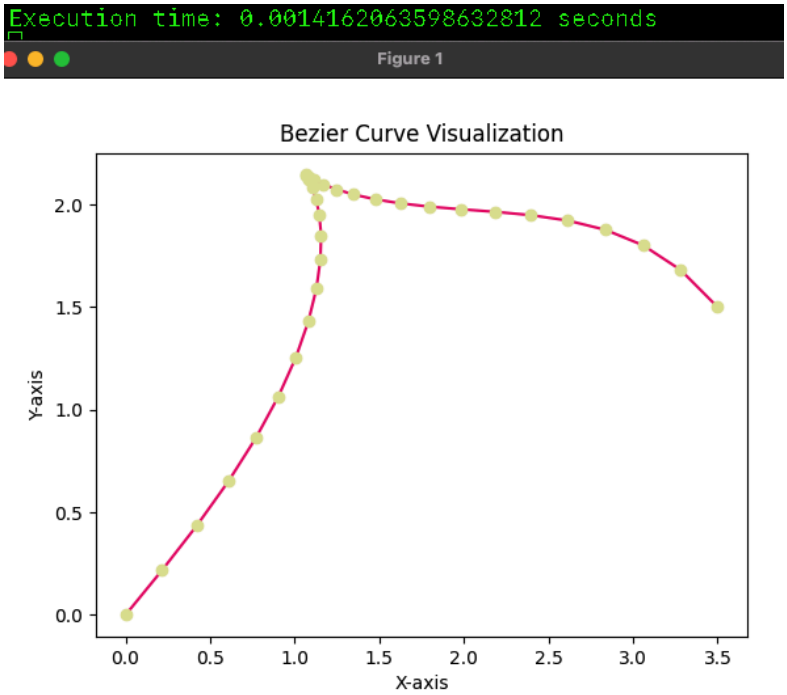
```
8
0.0 1.0 2.0 1.0 0.0 1.5 2.5 3.5
0.0 1.0 2.0 3.0 2.0 1.5 2.5 1.5
33
```

Input Divide and Conquer

```
8
0.0 1.0 2.0 1.0 0.0 1.5 2.5 3.5
0.0 1.0 2.0 3.0 2.0 1.5 2.5 1.5
5
```



Gambar 4.7 Output Test Case 4 Brute Force



Gambar 4.8 Output Test Case 4 Divide and Conquer

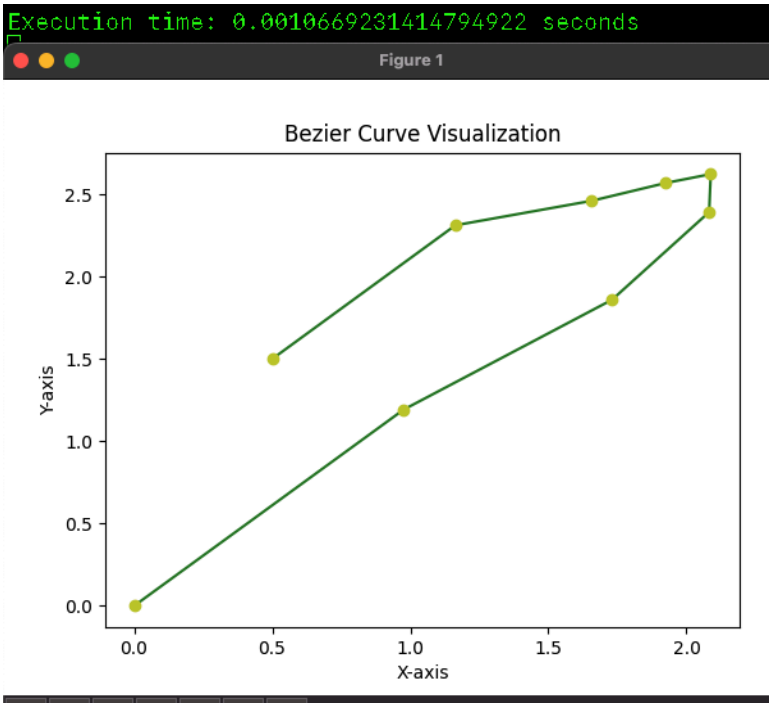
4.5. Test Case 5

Input Brute Force

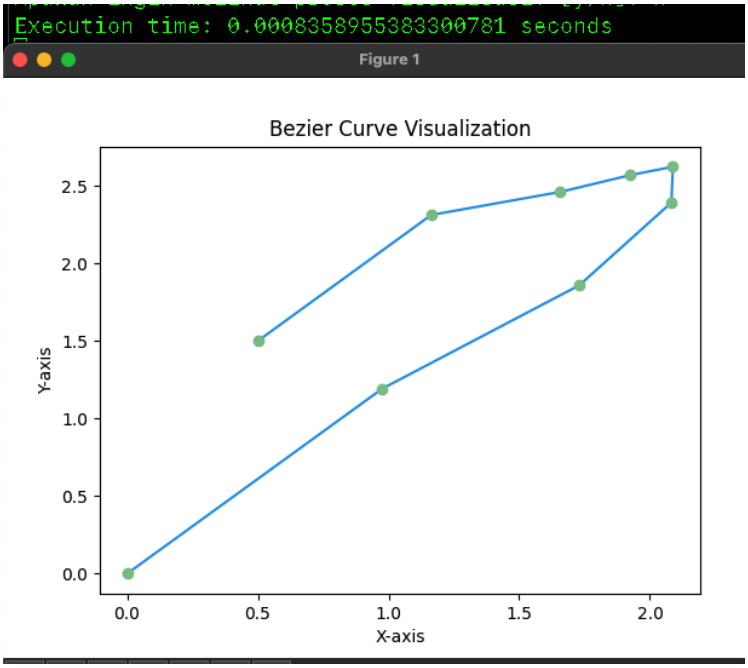
```
9
0.0 1.0 2.0 3.0 2.0 1.5 2.5 1.0 0.5
0.0 2.0 1.0 3.0 4.0 1.5 2.5 3.0 1.5
9
```

Input Divide and Conquer

```
9
0.0 1.0 2.0 3.0 2.0 1.5 2.5 1.0 0.5
0.0 2.0 1.0 3.0 4.0 1.5 2.5 3.0 1.5
3
```



Gambar 4.9 Output Test Case 5 Brute Force



Gambar 5.0 Output Test Case 5 Divide and Conquer

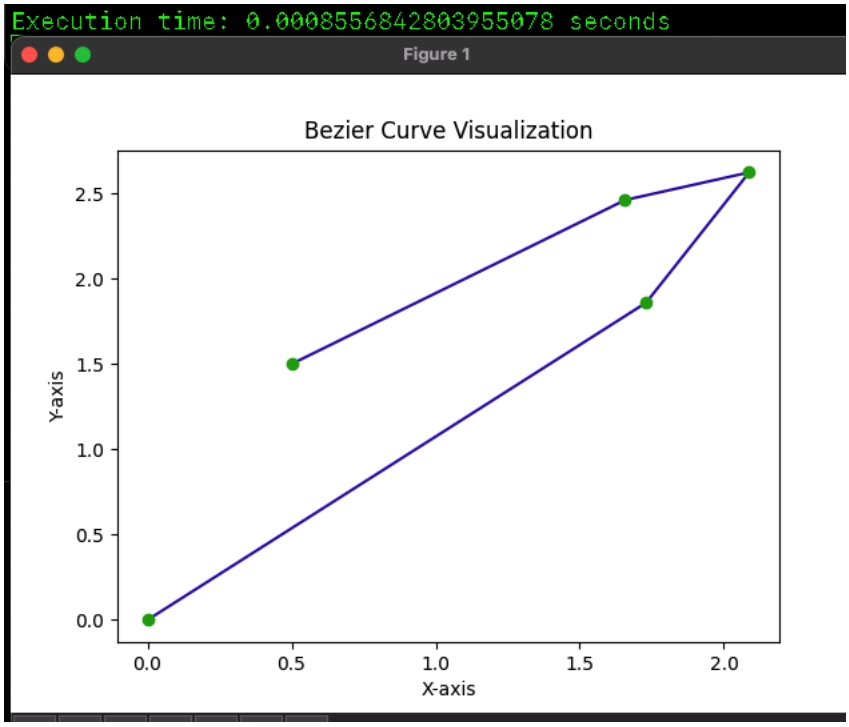
4.6. Test Case 6

Input Brute Force

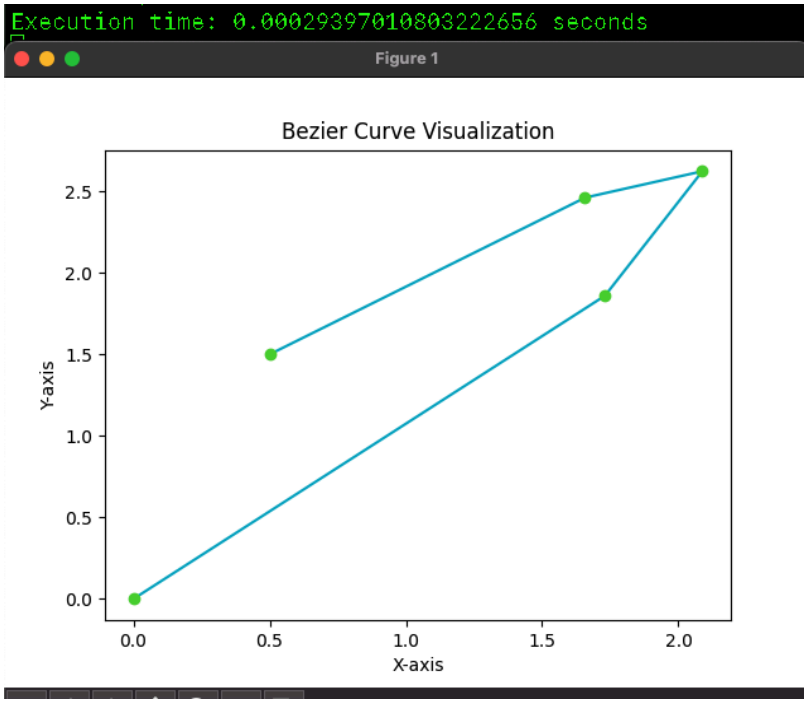
```
9
0.0 1.0 2.0 3.0 2.0 1.5 2.5 1.0 0.5
0.0 2.0 1.0 3.0 4.0 1.5 2.5 3.0 1.5
5
```

Input Divide and Conquer

```
9
0.0 1.0 2.0 3.0 2.0 1.5 2.5 1.0 0.5
0.0 2.0 1.0 3.0 4.0 1.5 2.5 3.0 1.5
2
```



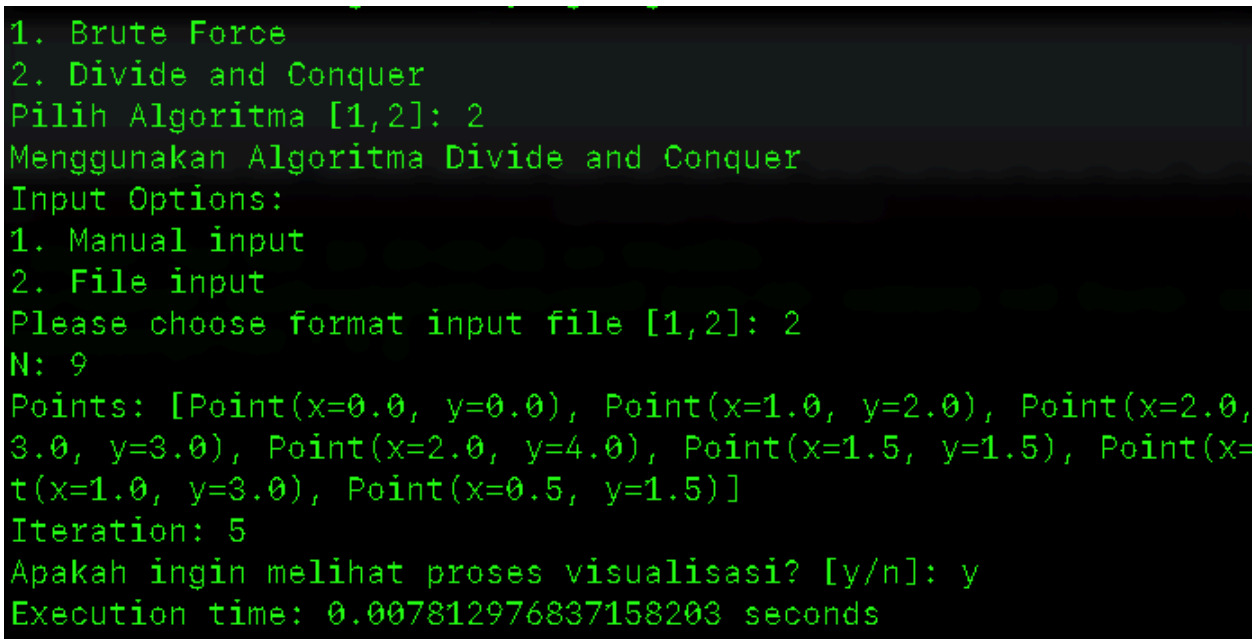
Gambar 5.1 Input Test Case 6 Brute Force



Gambar 5.2 Output Test Case 6 Divide and Conquer

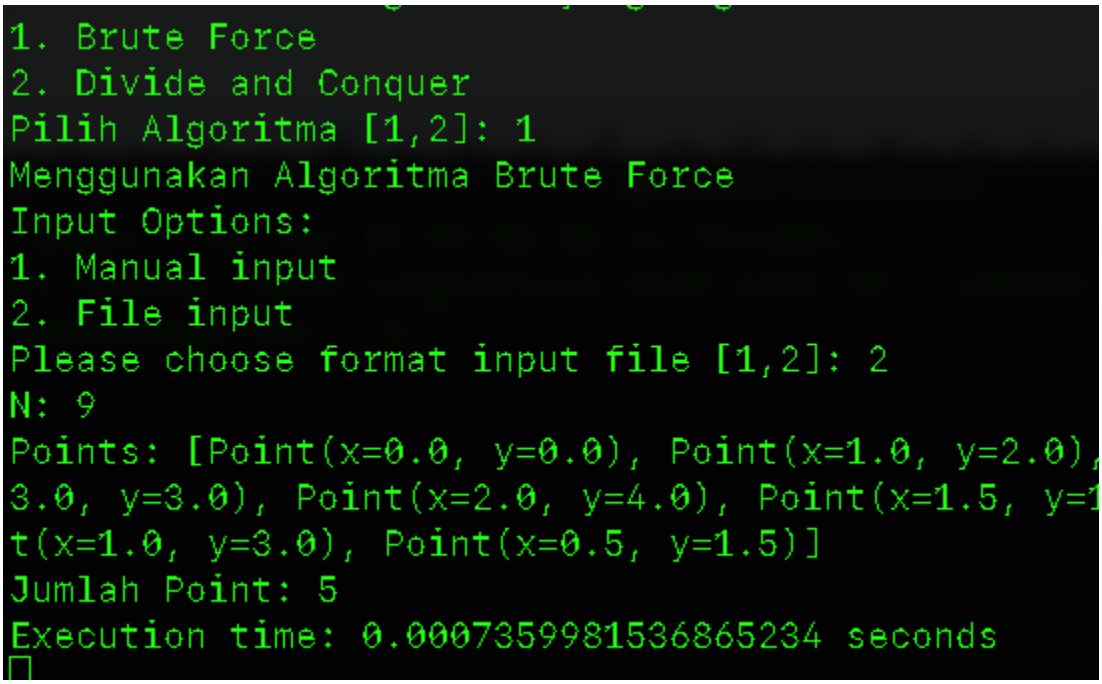
4.7 Tampilan Antar Muka CLI Program

1. Command CLI Brute Force



Gambar 30. Interaksi Input dan hasil output pada inputan model 1

2. Command CLI Divide and Conquer



Gambar 31. Interaksi input dan hasil output pada inputan model 2

BAB V Analisis Perbandingan Solusi Brute Force dan Divide and qonquer

Sebagaimana yang disampaikan sebelumnya untuk kompleksitas utama algoritma brute force yang diperoleh ialah O(N*M* $\frac{n^k}{k!}$) yang mana n > k dan n cukup besar maka kompleksitasnya menjadi eksponensial > 2^k dengan N menyatakan initial points, k menyatakan urutan kombinasi, M merupakan bobot koefisien. Hal tersebut juga didukung dengan percobaan yang dilakukan. Sedangkan kompleksitas divide and conquer algorithm pada pembangunan kurva bezier adalah $O(2^t * N)$ dimana t merupakan iterasi dan N jumlah initial points masing-masing sub-graf, karena pada Divide and Conquer dilakukan pembagian 2 left branch dan right branch. Serta untuk operasi penggabungan titik (Conquer) pada masing-masing traversal divide and conquer tersebut adalah sebanyak 2^banyaknya operasi = t, sehingga didapat banyaknya operasi penggabungan = Log(t) untuk setiap iterasi. Maka untuk kompleksitas penggabungan yang didapatkan ialah O(Log(N)*N) dengan N adalah banyaknya iterasi. Sehingga kompleksitas total yang dihasilkan ialah $O(2^t * N + Log(N) * N) = O(2^t * Log(N) * N)$. Dengan demikian, untuk N cukup kecil memang brute force lebih bagus, karena $\frac{n^k}{k!}$ masih mungkin < 2^N. Namun untuk N cukup besar brute force menjadi exponential lebih kompleks daripada bentuk 2^N, sehingga lebih baik menggunakan divide and conquer algorithm.

Lampiran

Lampiran 1

Checklist Penilaian:

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.	✓	

Lampiran 2

Link Repository Github: https://github.com/gilangr301102/tucil2_13520137