Tugas Kecil 2 IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Semester II Tahun 2023/2024

Disusun oleh

Sa’ad Abdul Hakim (13522092)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2024

Daftar Isi

A. Analisis dan Implementasi Algoritma Brute Force

Penggunaan algoritma *brute force* pada pembuatan kurva Bezier kuadratik melibatkan pengulangan nilai t dari 0 hingga 1 dengan interval tertentu untuk menghasilkan titik-titik pada kurva. Dalam konteks pembuatan kurva Bezier kuadratik, titik kontrol yang terlibat adalah titik awal (P0), titik kontrol (P1), dan titik akhir (P2). Selanjutnya, dengan menggunakan iterasi melalui parameter t dalam rentang 0 sampai 1, posisi titik pada kurva Bezier dihitung menggunakan persamaan Bezier kuadratik. Berikut adalah persamaan tersebut

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

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

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

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

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

Dengan menggunakan algoritma *brute force*, nilai t akan diubah secara bertahap dari 0 hingga 1 dengan interval yang telah ditentukan. Untuk setiap nilai t, posisi titik dihitung dan ditambahkan ke daftar titik yang terdefinisi pada kurva Bezier. Untuk memastikan perbandingan yang adil antara algoritma *brute force* dengan metode divide and conquer, kita perlu menyesuaikan jumlah iterasi dan interval t yang digunakan. Misalnya, jika kita ingin menggunakan jumlah iterasi yang sama dengan metode divide and conquer, maka interval t yang digunakan harus disesuaikan. Berdasarkan percobaan yang dilakukan, jumlah titik yang dihasilkan adalah $2^n + 1$ (termasuk titik awal dan akhir) dengan n adalah jumlah iterasi yang ingin dilakukan. Sebagai contoh, untuk jumlah iterasi 3, interval t akan diatur sedemikian rupa sehingga dihasilkan 9 total titik, yaitu 1 dibagi dengan 2^n adalah nilai interval t.
Berikut adalah implementasi algoritma *brute force* pada kurva Bezier kuadratik

```
def bruteforcebezier(pointlist, iteration):
    bezier = []
    temp = 1/(2**iteration)

    for i in range((2**iteration)+1):
        if i == 2**iteration:
            x = pointlist[2][0]
            y = pointlist[2][1]
            bezier.append((x, y))

        else:
            t = i*temp
            x = ((1-t)**2)*pointlist[0][0]+2*(1-t)*t*pointlist[1][0]+(t**2)*pointlist[2][0]
            y = ((1-t)**2)*pointlist[0][1]+2*(1-t)*t*pointlist[1][1]+(t**2)*pointlist[2][1]
            bezier.append((x, y))

    return bezier
```

Fungsi *bruteforcebezier* akan menerima parameter input berupa pointlist yaitu list dari titik-titik kontrol awal dan iteration yaitu jumlah iterasi yang diinginkan. Fungsi tersebut akan menghitung nilai interval berdasarkan penjelasan sebelumnya kemudian melakukan iterasi nilai t mulai dari 0 sampai 1 berdasarkan interval yang sudah dihitung. Titik-titik yang didapat kemudian dimasukan kedalam sebuah list yang nantinya akan menghasilkan kurva Bezier.

Berdasarkan gambar diatas, Kompleksitas waktu algoritma *brute force* ini bergantung pada jumlah iterasi, yang ditentukan oleh parameter iteration. Dengan demikian, kompleksitas waktu algoritma ini adalah $O(2^{iteration})$, karena jumlah iterasi adalah dua kali lipat dari iterasi sebelumnya. Oleh karena itu, algoritma ini memiliki kompleksitas waktu yang tinggi dan perlu dipertimbangkan penggunaan dari parameter iteration, karena ketika jumlah iterasi sangat besar, waktu yang dibutuhkan untuk memproses juga menjadi jauh lebih besar.

B. Analisis dan Implementasi Algoritma Divide and Conquer

Penggunaan algoritma *divide and conquer* untuk membentuk kurva Bezier kuadratik pada dasarnya mengikuti konsep pembagian masalah menjadi submasalah yang lebih kecil untuk menyelesaikannya secara efisien. Dalam konteks ini, algoritma membagi pembentukan kurva menjadi iterasi-iterasi yang menemukan *midpoint* (titik tengah) dari garis yang menghubungkan titik kontrol yang berdekatan. Proses ini dilakukan secara rekursif, di mana setiap langkah iterasi menghasilkan dua submasalah baru, yaitu pembentukan kurva pada bagian kiri dan kanan dari midpoint yang dihasilkan sebelumnya. Melalui iterasi yang berulang, jumlah midpoint yang dihasilkan akan meningkat, dan kurva Bézier yang terbentuk akan menjadi semakin halus. Jumlah midpoint yang dihasilkan dari algoritma ini bergantung pada jumlah iterasi yang dilakukan, dengan rumus $2^{iterasi}$ - 1 midpoint(tidak termasuk titik awal dan akhir). Hal ini menunjukkan bahwa semakin banyak iterasi yang dilakukan, semakin banyak titik tengah yang dihasilkan, dan akhirnya, semakin halus kurva Bezier yang terbentuk. Dengan demikian, algoritma *divide and conquer* memberikan kontrol yang lebih besar terhadap tingkat detail dan kehalusan dari kurva Bézier yang dihasilkan.

Berikut adalah implementasi algoritma *divide and conquer* pada kurva Bezier kuadratik

```
def titiktengah(point0, point1):
    x = (point0[0] + point1[0]) / 2
    y = (point0[1] + point1[1]) / 2

    return (x, y)

def computebezier(point0, point1, point2, bezierlist, currentIteration, iteration):
    if currentIteration <= iteration:
        titiktengah1 = titiktengah(point0, point1)
        titiktengah2 = titiktengah(point1, point2)
        titiktengah3 = titiktengah(titiktengah1, titiktengah2)
        currentIteration += 1
        computebezier(point0, titiktengah1, titiktengah3, bezierlist, currentIteration, iteration)
        bezierlist.append(titiktengah3)
        computebezier(titiktengah3, titiktengah2, point2, bezierlist, currentIteration, iteration)

def createbezier(pointlist, iteration):
    bezier = []
    bezier.append(pointlist[0])
    computebezier(pointlist[0], pointlist[1], pointlist[2], bezier, 1, iteration)
    bezier.append(pointlist[len(pointlist)-1])

    return bezier
```

Fungsi createbezier akan menerima parameter input berupa pointlist yaitu list dari titik-titik kontrol awal dan iteration yaitu jumlah iterasi yang diinginkan. Fungsi tersebut kemudian memasukkan titik awal ke dalam list bezier dan melakukan perhitungan titik midpoint pertama pada fungsi computebezier. Pada fungsi computebezier, akan dicari midpoint dari 3 titik kontrol dari parameter input point0, point1, dan point2. Kemudian akan dilakukan iterasi secara rekursif sesuai dengan jumlah iterasi yang diinginkan untuk menghasilkan dua submasalah baru, yaitu pembentukan kurva pada bagian kiri dan kanan dari midpoint yang dihasilkan seperti pada penjelasan diatas. Titik-titik midpoint yang didapat kemudian digabungkan pada fungsi createbezier dengan titik awal dan titik akhir dan menghasilkan kurva bezier.

Kompleksitas waktu algoritma ini sangat bergantung pada jumlah iterasi yang dilakukan. Pada setiap iterasi, proses pembagian kurva menjadi dua bagian memerlukan penghitungan titik tengah, yang memiliki kompleksitas waktu O(1). Namun, karena algoritma bersifat rekursif, jumlah panggilan rekursif akan meningkat secara eksponensial seiring dengan peningkatan iterasi. Oleh karena itu, kompleksitas waktu algoritma ini secara keseluruhan dapat dinyatakan sebagai $O(2^{iteration})$, di mana iteration adalah jumlah iterasi yang ditentukan oleh pengguna.

C. Source Code Program

bruteforcebezier.py

```
def bruteforcebezier(pointlist, iteration):
    bezier = []
    temp = 1/(2**iteration)

    for i in range((2**iteration)+1):
        if i == 2**iteration:
            x = pointlist[2][0]
            y = pointlist[2][1]
            bezier.append((x, y))

        else:
            t = i*temp
            x =
((1-t)**2)*pointlist[0][0]+2*(1-t)*t*pointlist[1][0]+(t**2)*pointlist[2][0
]
            y =
((1-t)**2)*pointlist[0][1]+2*(1-t)*t*pointlist[1][1]+(t**2)*pointlist[2][1
]
            bezier.append((x, y))

    return bezier
```

midpointbezier.py

```
import matplotlib.pyplot as plt

def createpoint():
    x = float(input("Enter x: "))
    y = float(input("Enter y: "))

    return (x, y)

def createallinitialpoint():
    pointlist = []

    for i in range(0,3):
        print("input point P"+str(i))
        pointlist.append(createpoint())

    return pointlist

def titiktengah(point0, point1):
    x = (point0[0] + point1[0]) / 2
    y = (point0[1] + point1[1]) / 2

    return (x, y)

def computebezier(point0, point1, point2, bezierlist, currentIteration,
iteration):
    if currentIteration <= iteration:
        titiktengah1 = titiktengah(point0, point1)
        titiktengah2 = titiktengah(point1, point2)
        titiktengah3 = titiktengah(titiktengah1, titiktengah2)
        currentIteration += 1
        computebezier(point0, titiktengah1, titiktengah3, bezierlist,
currentIteration, iteration)
        bezierlist.append(titiktengah3)
        computebezier(titiktengah3, titiktengah2, point2, bezierlist,
currentIteration, iteration)

def createbezier(pointlist, iteration):
```

```
    bezier = []
    bezier.append(pointlist[0])
    computebezier(pointlist[0], pointlist[1], pointlist[2], bezier, 1,
iteration)
    bezier.append(pointlist[len(pointlist)-1])

    return bezier

def plotter(pointlist, iterate):
    coorx = [point[0] for point in pointlist]
    coory = [point[1] for point in pointlist]
    title = "Iteration "+str(iterate)
    plt.plot(coorx, coory, marker='o', linestyle='-', label=title,
markersize=3)

def drawalliteration(pointlist, iteration):
    for i in range(1, iteration):
        beziercurve = createbezier(pointlist, i)
        plotter(beziercurve, i)
```

main.py

```
from midpointbezier import *
from bruteforcebezier import *
import time

# jumlahpoint = int(input("Masukkan jumlah point: "))
# while jumlahpoint < 3:
#     jumlahpoint = int(input("Masukkan jumlah point(minimal 3): "))

print("Silahkan pilih cara input:")
print("1. Input manual")
print("2. Input file")
masukkan = int(input("Pilihan: "))

if masukkan == 1:
    pointlist = createallinitialpoint()
```

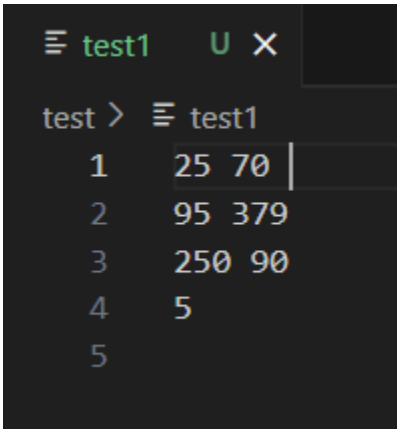
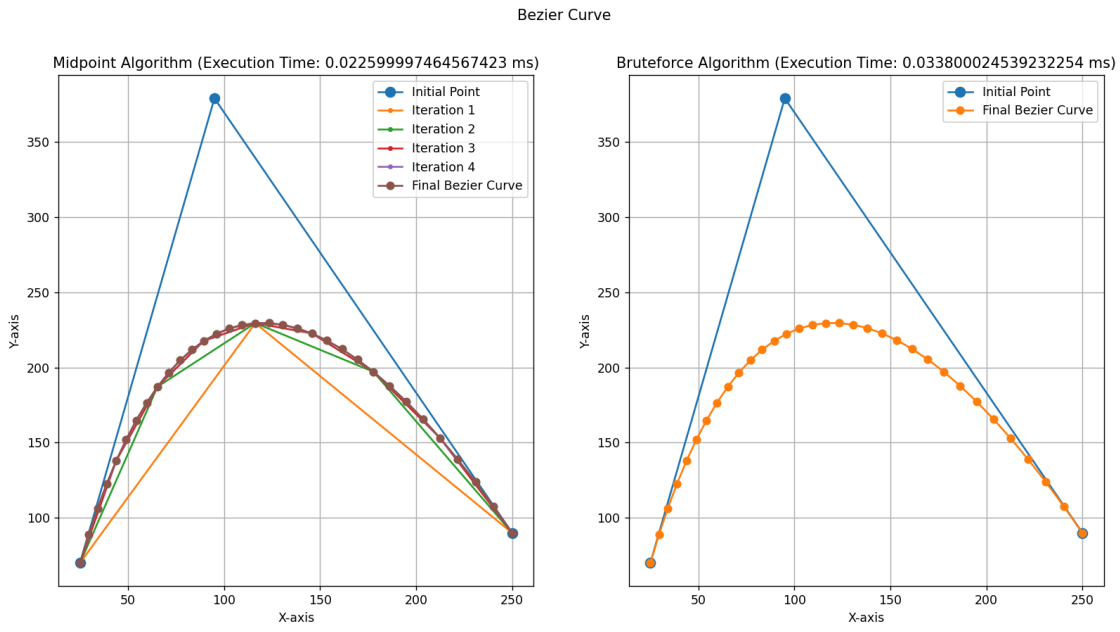
```
    iteration = int(input("Masukkan jumlah iterasi: "))
    while iteration < 1:
        print("Jumlah iterasi minimal 1")
        iteration = int(input("Masukkan jumlah iterasi(minimal 1): "))
elif masukkan == 2:
    filename = input("Masukkan nama file: ")
    filename = "../test/"+filename
    with open(filename, "r") as file:
        pointlist = []

        for i in range(3):
            line = file.readline()
            numbers = line.split()
            numbers_tuple = tuple(float(num) for num in numbers)
            pointlist.append(numbers_tuple)
        val = file.readline()
        iteration = int(val)
else:
    print("Input tidak valid")
    exit()

plt.subplot(1, 2, 1)

initcoorx = [point[0] for point in pointlist]
initcoory = [point[1] for point in pointlist]

plt.plot(initcoorx, initcoory, marker='o', linestyle='-', label='Initial
Point', markersize=8)

drawalliteration(pointlist, iteration)
startmidpoint = time.perf_counter()
midpointbeziercurve = createbezier(pointlist, iteration)
endmidpoint = time.perf_counter()
decimal_places = 4
# exectimemidpoint = f"{endmidpoint-startmidpoint:.{decimal_places}f}"
exectimemidpoint = (endmidpoint-startmidpoint)*1000

finalcoorx = [point[0] for point in midpointbeziercurve]
finalcoory = [point[1] for point in midpointbeziercurve]
```

```
plt.plot(finalcoorx, finalcoory, marker='o', linestyle='-', label='Final
Bezier Curve')

titlemidpoint = "Midpoint Algorithm (Execution Time:
"+str(exectimemidpoint)+" ms)"
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title(titlemidpoint)
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)

plt.plot(initcoorx, initcoory, marker='o', linestyle='-', label='Initial
Point', markersize=8)

startbruteforce = time.perf_counter()
bruteforcebeziercurve = bruteforcebezier(pointlist, iteration)
endbruteforce = time.perf_counter()
# exectimebruteforce =
f"{endbruteforce-startbruteforce:.{decimal_places}f}"
exectimebruteforce = (endbruteforce-startbruteforce)*1000

coorx = [point[0] for point in bruteforcebeziercurve]
coory = [point[1] for point in bruteforcebeziercurve]

plt.plot(coorx, coory, marker='o', linestyle='-', label='Final Bezier
Curve')

titlebruteforce = "Bruteforce Algorithm (Execution Time:
"+str(exectimebruteforce)+" ms)"
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title(titlebruteforce)
plt.legend()
plt.grid()

plt.suptitle('Bezier Curve')
```

```
plt.show()
```

D. Test Case

Test Case 1

Input

Output


Test Case 2

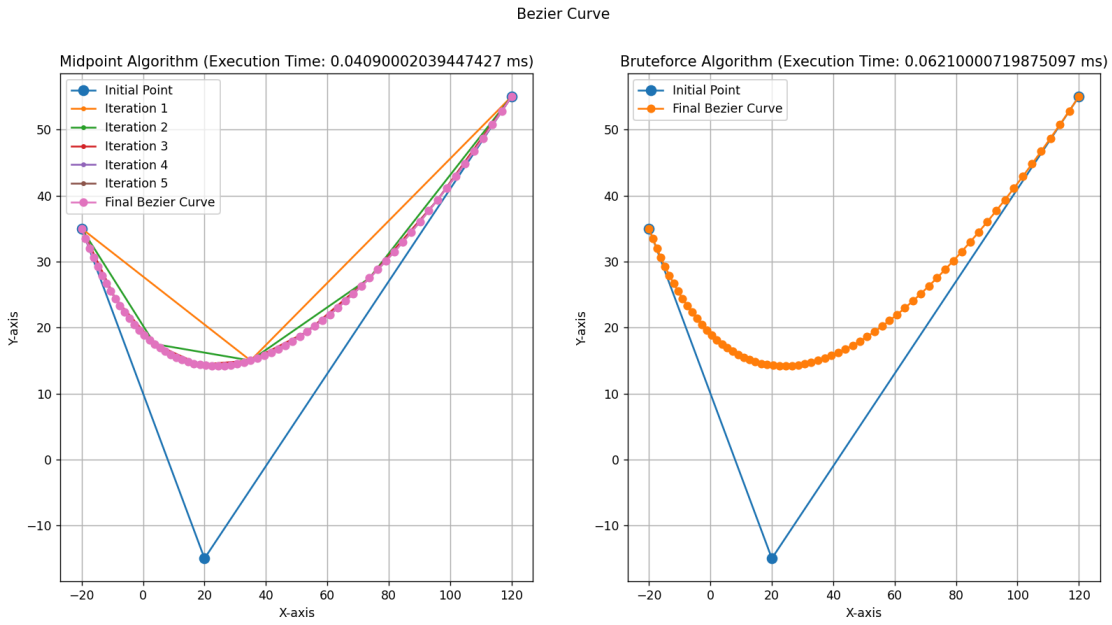
Input

test2 U

test > test2

```
1   -20 35
2   20 -15
3   120 55
4   6
```

Output



Test Case 3

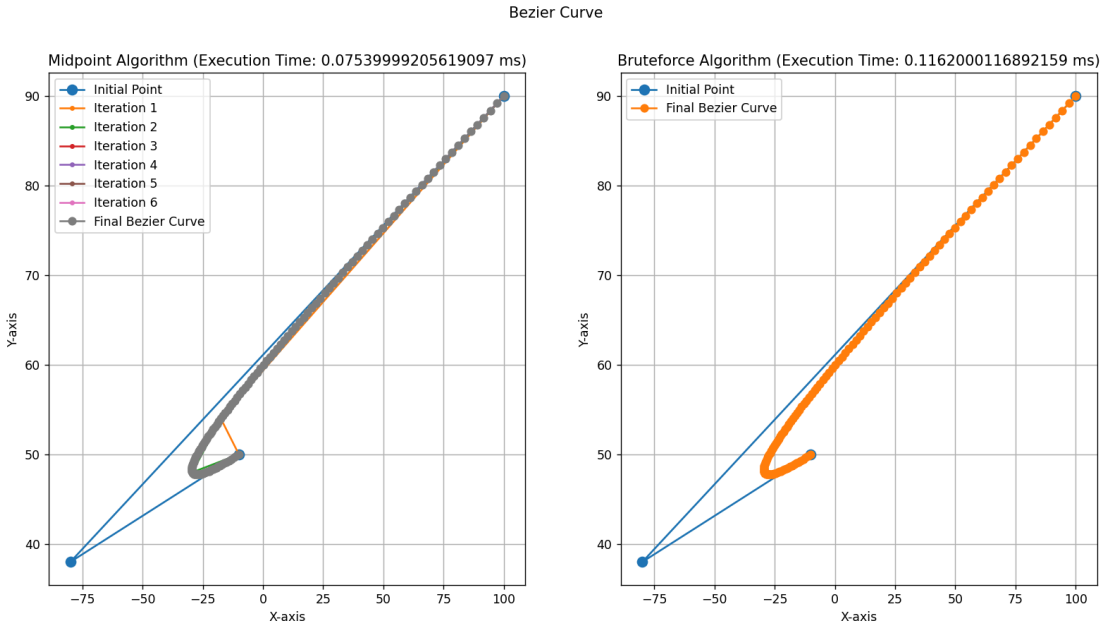
Input

test3 U

test > test3

```
1   -10 50
2   -80 38
3   100 90
4   7
```

Output

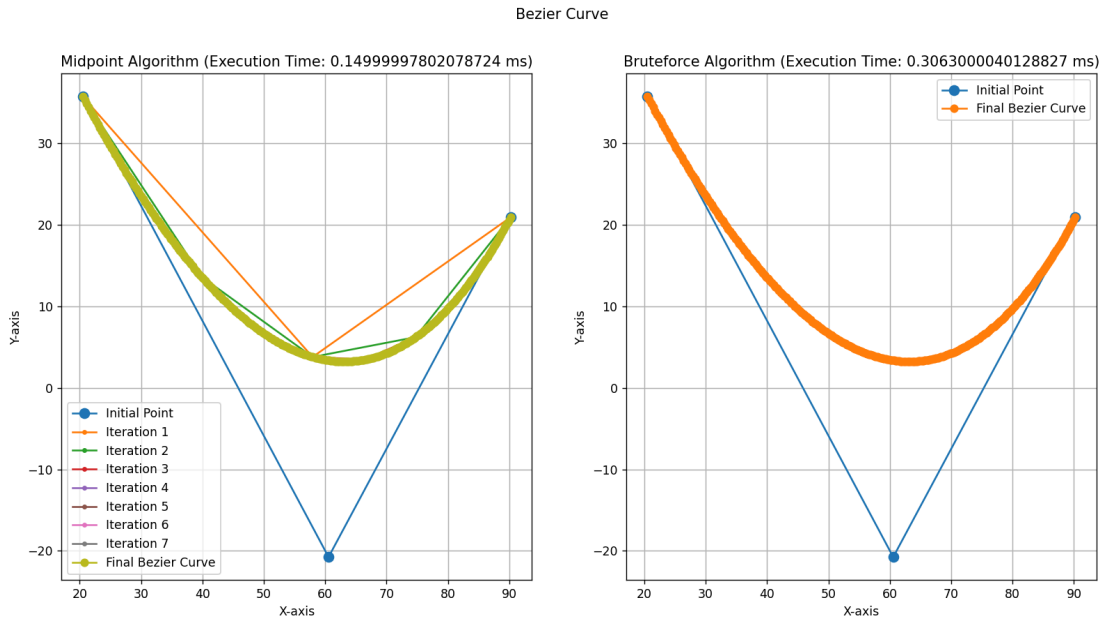


Test Case 4

Input

```
Silahkan pilih cara input:
1. Input manual
2. Input file
Pilihan: 1
input point P0
Enter x: 20.5
Enter y: 35.7
input point P1
Enter x: 60.55
Enter y: -20.75
input point P2
Enter x: 90.20
Enter y: 20.90
Masukkan jumlah iterasi: 8
```

Output

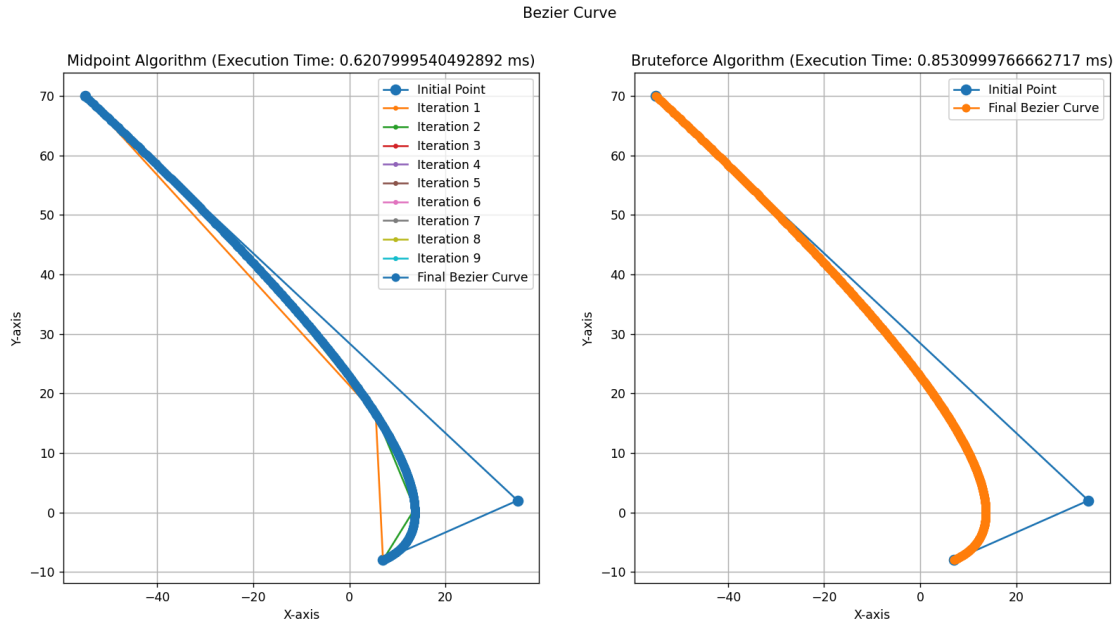


Test Case 5

Input

```
Silahkan pilih cara input:
1. Input manual
2. Input file
Pilihan: 1
input point P0
Enter x: -55
Enter y: 70
input point P1
Enter x: 35
Enter y: 2
input point P2
Enter x: 7
Enter y: -8
Masukkan jumlah iterasi: 10
```

Output

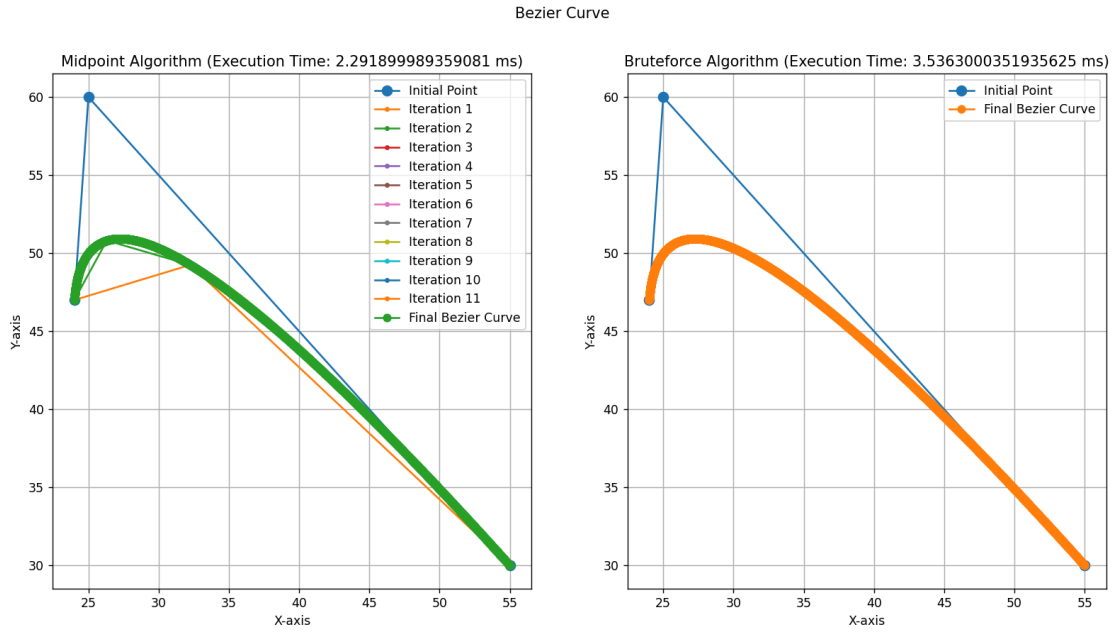


Test Case 6

Input

```
Silahkan pilih cara input:
1. Input manual
2. Input file
Pilihan: 1
input point P0
Enter x: 55
Enter y: 30
input point P1
Enter x: 25
Enter y: 60
input point P2
Enter x: 24
Enter y: 47
Masukkan jumlah iterasi: 12
```

Output



E. Analisis Perbandingan Algoritma Brute Force dan Divide and Conquer

Test Case	Jumlah Iterasi	Brute Force (execution time in ms)	Divide and Conquer (execution time in ms)

1	5	0.0338	0.0225
2	6	0.0621	0.0409
3	7	0.1162	0.0753
4	8	0.3063	0.1499
5	10	0.8530	0.6207
6	12	3.5363	2.2918

Berdasarkan tabel diatas, penggunaan algoritma *divide and conquer* dalam pembentukan kurva Bezier kuadratik terbukti selalu memiliki waktu eksekusi yang lebih cepat jika dibandingkan penggunaan algoritma *brute force* dalam pembentukan kurva Bezier kuadratik. Hal tersebut berarti penggunaan algoritma titik tengah berbasis *divide and conquer* dalam membuat kurva Bezier kuadratik lebih efisien jika dibandingkan dengan algoritma *brute force*. Selain itu, berdasarkan tabel, waktu eksekusi kedua algoritma akan semakin jauh ketika jumlah iterasi terus dinaikkan.

Lampiran

Link Repository Github : https://github.com/saadabha/Tucil2_13522092

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.	✓	