LAPORAN TUGAS KECIL 2
IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Dipersiapkan oleh:

Diero Arga Purnama (13522056)

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung,

Jl. Ganesha 10, Bandung 40132

DAFTAR ISI

BAB I

DESKRIPSI MASALAH

Kurva beziér adalah kurva yang seringkali digunakan dalam bidang desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol yang menentukan bentuk dan arah dari kurva. Dalam konteks laporan ini, kurva beziér yang akan dibahas merupakan kurva beziér kuadratik, yaitu kurva beziér yang memiliki 3 titik kontrol. Algoritma yang akan digunakan untuk membuat kurva beziér dalam laporan ini adalah algoritma *divide and conquer* dan algoritma *brute force*.

BAB II

TEORI SINGKAT

2.1 Algoritma *Divide and Conquer*

Algoritma *divide and conquer* adalah algoritma yang membagi sebuah persoalan menjadi beberapa upapersoalan yang lebih kecil agar lebih mudah untuk diselesaikan.

2.2 Algoritma *Brute Force*

Algoritma *brute force* adalah pendekatan yang sederhana atau *straightforward* untuk memecahkan suatu persoalan.

2.3 Kurva Beziér

Kurva beziér adalah kurva yang seringkali digunakan dalam bidang desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol yang menentukan bentuk dan arah dari kurva. Dalam laporan ini, kurva beziér yang akan dibahas adalah kurva beziér kuadratik yang memiliki 3 tiga titik kontrol. Jika dalam suatu kurva beziér kuadratik terdapat titik-titik P_0, P_1, dan P_2 maka, rumus kurva beziér yang menghubungkan titik P_0 dengan P_2 adalah sebagai berikut,

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

BAB III

IMPLEMENTASI PROGRAM

3.1 Analisis dan Implementasi Algoritma *Brute Force*

Algoritma *brute force* membangun kurva beziér dengan mencari satu per satu titik-titik yang berada dalam kurva. Pencarian titik-titik dilakukan melalui rumus berikut,

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

Pertama-tama, algoritma akan mencari jumlah titik-titik yang akan dicari. Rumus yang digunakan untuk mencari jumlah titik-titik adalah sebagai berikut,

$$n(i) = \sum_{k=0}^{i-1} 2^k, \quad (2)$$

dengan i merupakan jumlah iterasi yang dimasukkan oleh pengguna.

Setelah didapatkan jumlah titik-titik yang akan dicari, algoritma akan membuat sebuah *list* yang berisi nilai-nilai *t* sebanyak *n* yang akan digunakan untuk membangun kurva beziér sesuai dengan rumus (1).

3.2 Analisis dan Implementasi Algoritma *Divide and Conquer*

Algoritma *divide and conquer* membangun kurva beziér dengan menggunakan metode *midpoint* atau titik tengah. Pertama-tama algoritma akan mencari tiga titik, yaitu:

1. Q_0 yaitu titik tengah dari titik kontrol P_0 dengan P_1.
2. Q_1 yaitu titik tengah dari titik kontrol P_1 dengan P_2.
3. R_0 yaitu titik tengah dari titik Q_0 dengan Q_1.

Setelah didapatkan tiga titik tersebut, algoritma akan mengulang secara rekursif sebanyak jumlah iterasi yang dimasukkan oleh pengguna untuk membangun dua kurva, yaitu kurva kiri dan kurva kanan. Setelah kurva kiri dan kurva kanan telah dibangun, kedua kurva akan digabungkan kembali menjadi sebuah kurva beziér yang lengkap.

3.3 Source Code

3.1.1 classes.py

```
class Point:
```

```
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

3.1.2 dnc.py (Algoritma *Divide and Conquer*)

```
from classes import Point
from typing import List

def titikTengah(p1: Point, p2: Point) -> Point:
    # Mengembalikan titik tengah dari dua titik
    newX = (p1.x + p2.x) / 2
    newY = (p1.y + p2.y) / 2
    return Point(newX, newY)

def kurvaBezierDNC(p0: Point, p1: Point, p2: Point, iterasi: int,
iterasiMax: int, listPoint: List[Point]):
    if iterasi < iterasiMax:
        iterasi += 1

        q0 = titikTengah(p0, p1)
        q1 = titikTengah(p1, p2)
        r0 = titikTengah(q0, q1)

        # Branch Kiri
        kurvaBezierDNC(p0, q0, r0, iterasi, iterasiMax, listPoint)
        # Append titik tengah
        listPoint.append(r0)
        # Branch Kanan
        kurvaBezierDNC(r0, q1, p2, iterasi, iterasiMax, listPoint)
    else:
        listPoint.append(p2)
```

3.1.3 bf.py (Algoritma *Brute Force*)

```
from classes import Point
from typing import List

def rumus(p0: Point, p1: Point, p2: Point, t: float) -> Point:
```

```
    newX = ((1 - t) ** 2) * p0.x + 2 * (1 - t) * t * p1.x + (t ** 2) * p2.x
    newY = ((1 - t) ** 2) * p0.y + 2 * (1 - t) * t * p1.y + (t ** 2) * p2.y
    return Point(newX, newY)

def generateN(i: int) -> int:
    if i == 0:
        return 0
    if i == 1:
        return 1
    else:
        return 2 ** (i - 1) + generateN(i - 1)

def generateT(iterasi: int) -> List[float]:
    if iterasi < 0:
        return []

    listofFloat = []
    selisih = 1 / ((generateN(iterasi)) + 1)

    i = selisih
    while i < 1:
        listofFloat.append(i)
        i += selisih
    return listofFloat

def kurvaBezierBF(p0: Point, p1: Point, p2: Point, iterasi: int, listPoint:
List[Point]):
    listT = generateT(iterasi)
    temp = [rumus(p0, p1, p2, t) for t in listT]
    listPoint.extend(temp)
    listPoint.append(p2)
```

3.1.4 main.py

```
from matplotlib import pyplot as plt
from typing import Tuple, List
from dnc import *
```

```
from bf import *
import time

def createKurva(p0: Point, p1: Point, p2: Point, iterasiMax: int, isBF:
bool) -> Tuple[List[Point], float]:
    waktuAwal = time.time()

    listPoint = [p0]
    if isBF:
        kurvaBezierBF(p0, p1, p2, iterasiMax, listPoint)
    else:
        kurvaBezierDNC(p0, p1, p2, 0, iterasiMax, listPoint)

    waktuAkhir = time.time()
    return listPoint, (waktuAkhir - waktuAwal)

def ambilInput():
    # Mengembalikan 3 control point, jumlah iterasi, dan metode yang
dimasukkan oleh user

    # Input control point
    pointList = []
    for i in range(3):
        print(f"Point {i + 1}")
        point = Point(float(input("x = ")), float(input("y = ")))
        pointList.append(point)

    # Input jumlah iterasi
    nIterasi = int(input("Jumlah Iterasi: "))
    while (nIterasi < 0):
        print("Jumlah iterasi harus lebih dari 0!")
        nIterasi = int(input("Jumlah Iterasi: "))

    # Input metode brute force atau divide and conquer
    metode = str(input("Metode Brute Force [B] / Divide and Conquer [D]: "))
    while metode != 'D' and metode != 'B':
        print("Masukan tidak valid, ", end="")
```

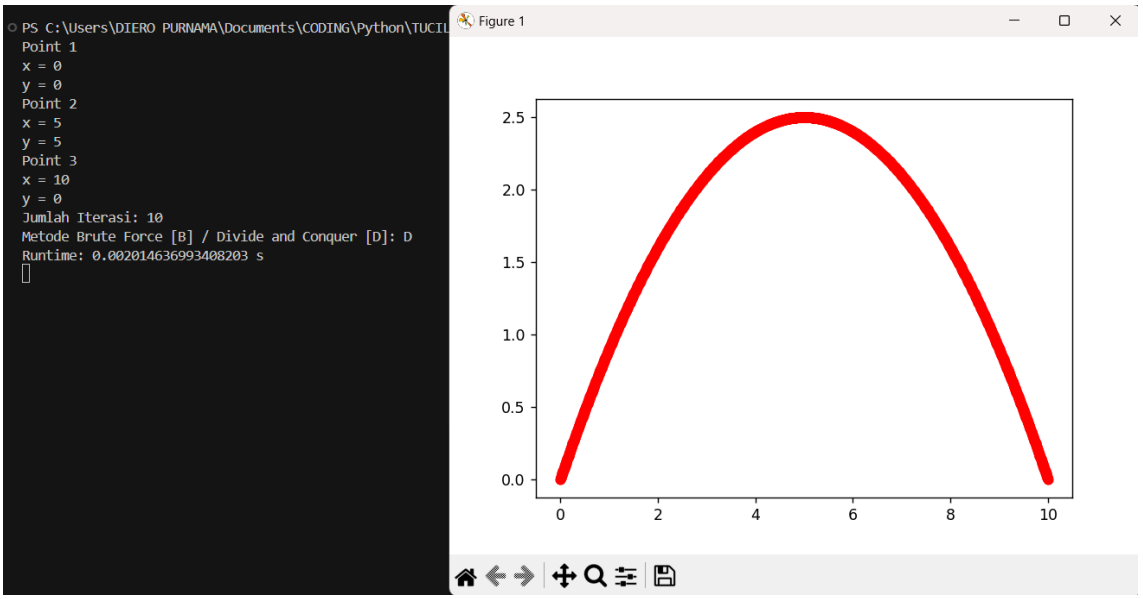
```
        metode = str(input("Metode Brute Force [B] / Divide and Conquer [D]:
"))
    if metode == 'D':
        isBF = False
    elif metode == 'B':
        isBF = True
    return pointList, nIterasi, isBF

def main():
    # Input
    inputUser, iterasiMax, isBF = ambilInput()

    # Proses
    listPoint, runtime = createKurva(inputUser[0], inputUser[1],
inputUser[2], iterasiMax, isBF)
    x = [p.x for p in listPoint]
    y = [p.y for p in listPoint]

    # Output
    print(f"Runtime: {runtime} s")
    plt.plot(x, y, "ro-")
    plt.show()

main()
```

BAB IV

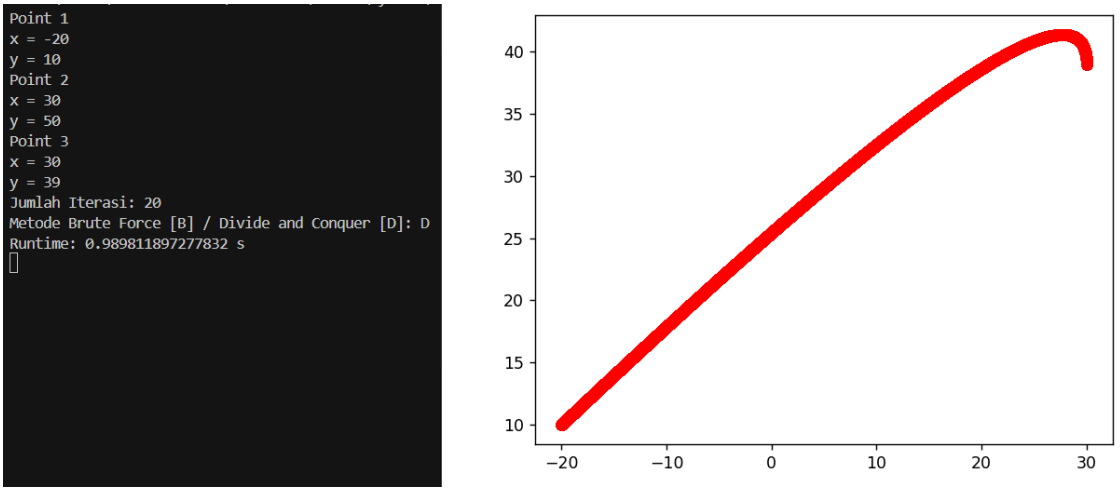
PENGUJIAN DAN ANALISIS

4.1 Hasil pengujian

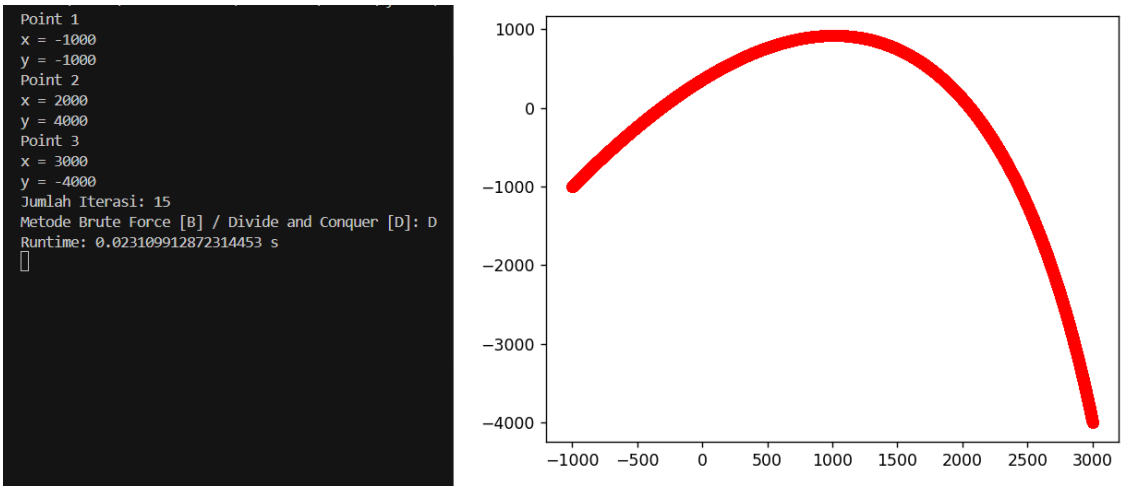
4.1.1 Algoritma *Divide and Conquer*



Gambar 4.1.1.1 Test Case 1 *Divide and Conquer*

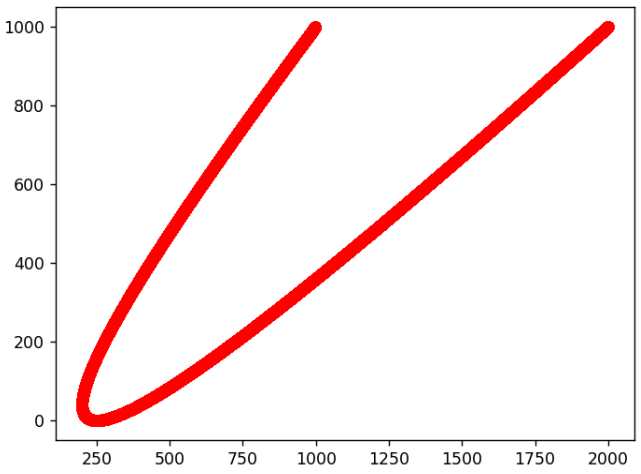


Gambar 4.1.1.2 Test Case 2 *Divide and Conquer*



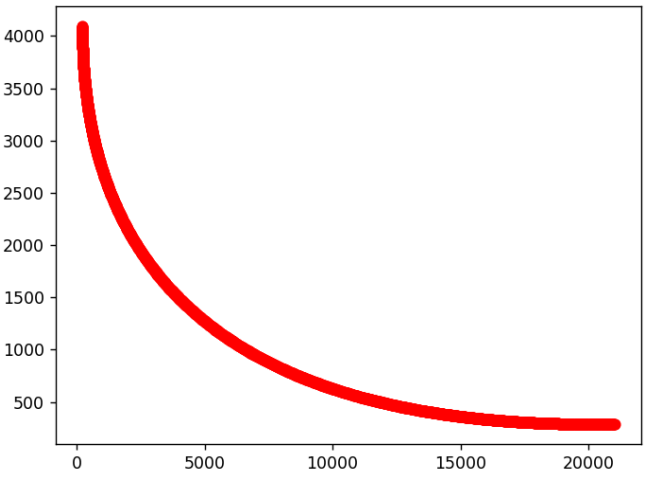
Gambar 4.1.1.3 Test Case 3 *Divide and Conquer*

```
Point 1
x = 999
y = 999
Point 2
x = -999
y = -999
Point 3
x = 2000
y = 999
Jumlah Iterasi: 20
Metode Brute Force [B] / Divide and Conquer [D]: D
Runtime: 1.0371768474578857 s
```



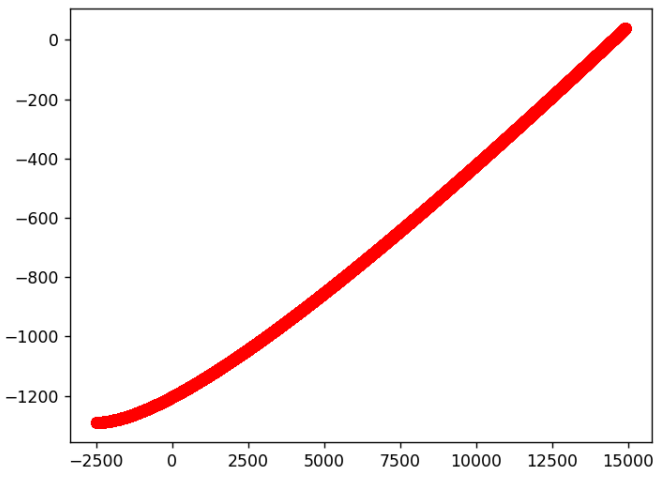
Gambar 4.1.1.4 Test Case 4 *Divide and Conquer*

```
Point 1
x = 231
y = 4095
Point 2
x = 134
y = 210
Point 3
x = 21014
y = 290
Jumlah Iterasi: 10
Metode Brute Force [B] / Divide and Conquer [D]: D
Runtime: 0.0019986629486083984 s
```



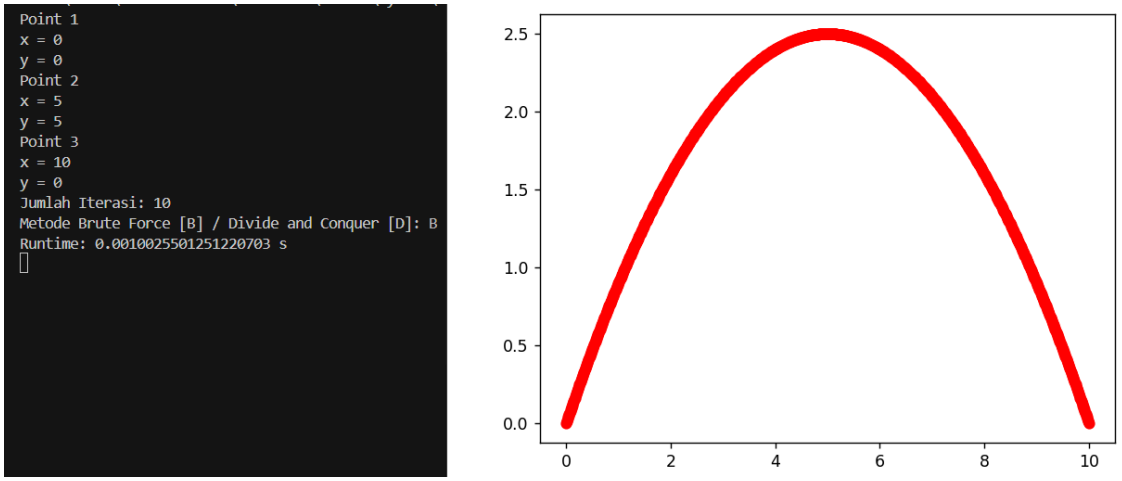
Gambar 4.1.1.5 Test Case 5 *Divide and Conquer*

```
Point 1
x = 14909
y = 39
Point 2
x = 1209
y = -1284
Point 3
x = -2490
y = -1290
Jumlah Iterasi: 15
Metode Brute Force [B] / Divide and Conquer [D]: D
Runtime: 0.025926828384399414 s
```

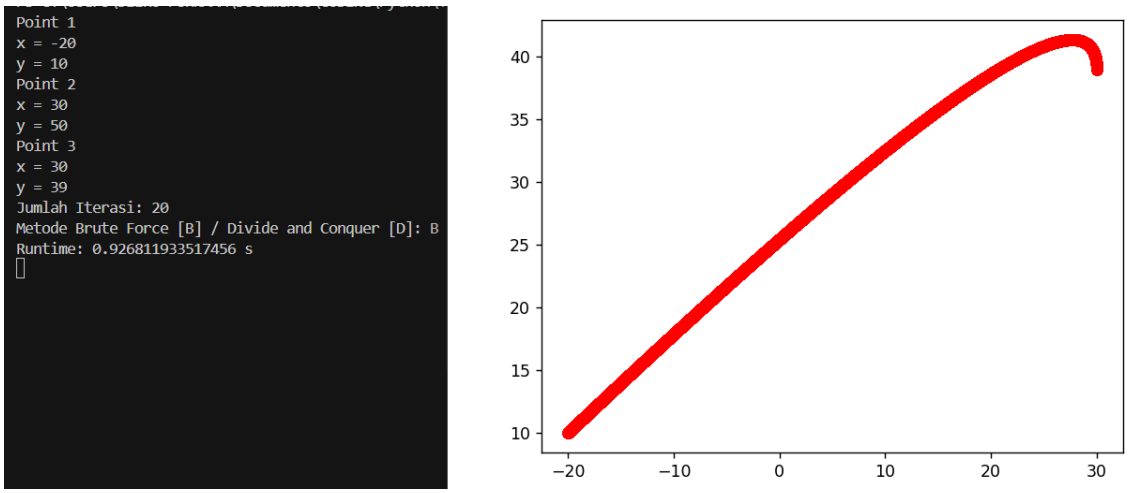


Gambar 4.1.1.6 Test Case 6 *Divide and Conquer*

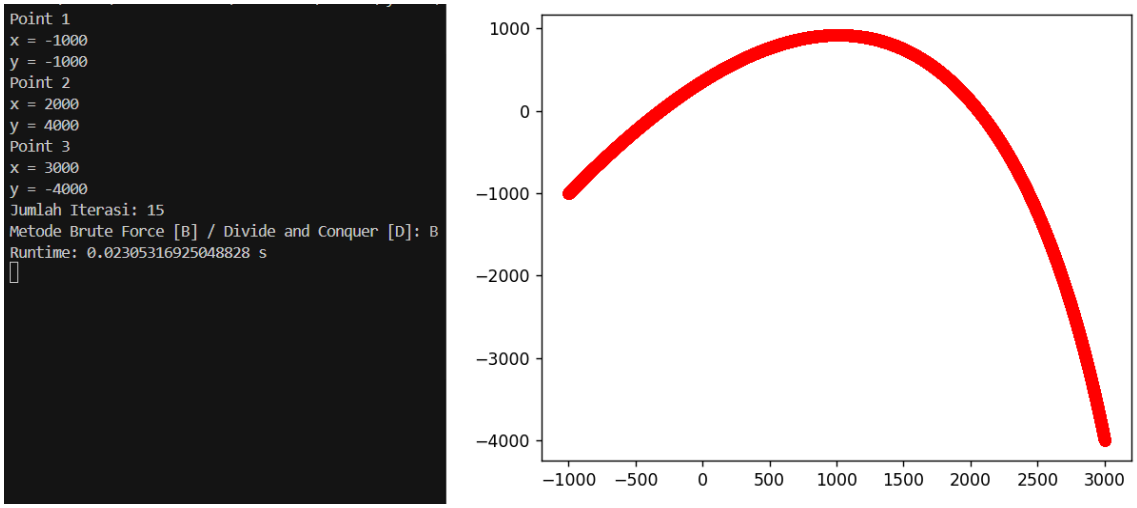
4.1.2 Algoritma *Brute Force*



Gambar 4.1.2.1 Test Case 1 *Brute Force*

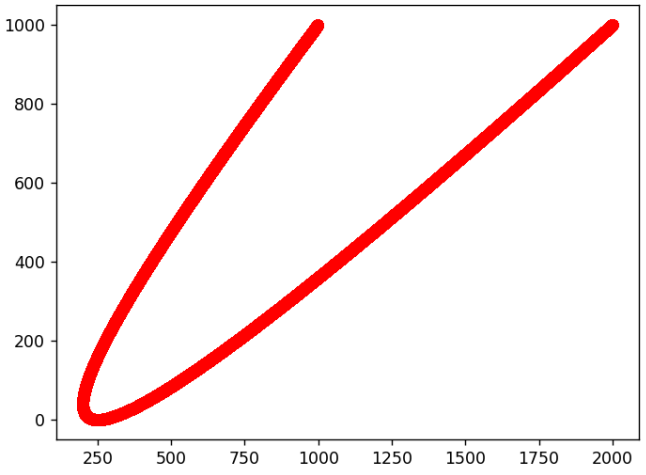


Gambar 4.1.2.2 Test Case 2 *Brute Force*



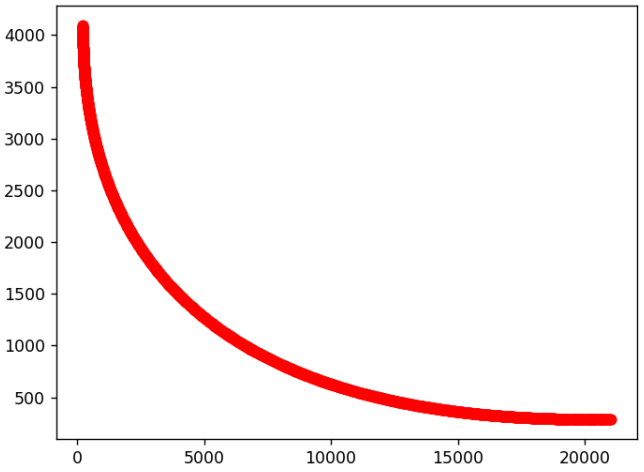
Gambar 4.1.2.3 Test Case 3 *Brute Force*

```
Point 1
x = 999
y = 999
Point 2
x = -999
y = -999
Point 3
x = 2000
y = 999
Jumlah Iterasi: 20
Metode Brute Force [B] / Divide and Conquer [D]: B
Runtime: 0.9686927795410156 s
```



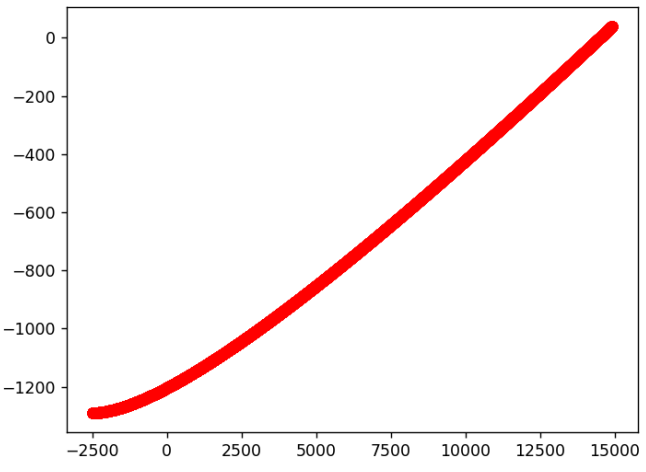
Gambar 4.1.2.4 Test Case 4 *Brute Force*

```
Point 1
x = 231
y = 4095
Point 2
x = 134
y = 210
Point 3
x = 21014
y = 290
Jumlah Iterasi: 10
Metode Brute Force [B] / Divide and Conquer [D]: B
Runtime: 0.0010547637939453125 s
```



Gambar 4.1.2.5 Test Case 5 *Brute Force*

```
x = 14909
y = 39
Point 2
x = 1209
y = -1284
Point 3
x = -2490
y = -1290
Jumlah Iterasi: 15
Metode Brute Force [B] / Divide and Conquer [D]: B
Runtime: 0.02165675163269043 s
```



Gambar 4.1.2.6 Test Case 6 *Brute Force*

4.2 Analisis Perbandingan Solusi

4.1.1 Analisis Kompleksitas Algoritma *Divide and Conquer*

Dalam algoritma *divide and conquer*, terdapat pemanggilan rekursif yang akan diulangi sebanyak *n* kali, dengan *n* merupakan jumlah iterasi yang dimasukkan oleh pengguna. Dalam setiap pemanggilan rekursi, fungsi akan dibagi menjadi dua cabang yang masing-masing akan menghasilkan dua cabang, dst. Hal ini menyebabkan kenaikan kompleksitas algoritma secara eksponensial, sehingga didapatkan kompleksitas algoritma *divide and conquer* sebesar $O(2^n)$.

4.1.2 Analisis Kompleksitas Algoritma *Brute Force*

Dalam algoritma *brute force*, pembangunan kurva beziér dilakukan dengan mencari titik satu per satu. Sehingga kompleksitas algoritma dari algoritma *brute force* dapat ditentukan dengan jumlah titik yang dicarinya tiap iterasi. Untuk menentukan jumlah titik yang dicari, algoritma akan memanggil fungsi generateN yang menggunakan rumus berikut,

$$n(i) = \sum_{k=0}^{i-1} 2^k.$$

Sehingga akan didapatkan kompleksitas algoritma *brute force* sebesar $O(2^n)$ dengan *n* merupakan jumlah iterasi yang dimasukkan oleh pengguna.

4.1.3 Analisis Perbandingan *Runtime* Hasil Pengujian

Berikut merupakan perbandingan *runtime* yang didapatkan dari pengujian kedua algoritma menggunakan 6 test case, *runtime* akan dibulatkan ke 4 angka belakang koma untuk memudahkan perbandingan.

No	Test Case	*Brute Force* (s)	*Divide and Conquer (s)*
1	Test Case 1	0.0010	0.0020
2	Test Case 2	0.9268	0.9898
3	Test Case 3	0.0231	0.0231
4	Test Case 4	0.9687	1.0372

5	Test Case 5	0.0010	0.0020
6	Test Case 6	0.022	0.0259

Gambar 4.1.3.1 Tabel *Runtime*

Berdasarkan tabel di atas, dapat dilihat bahwa di antara algoritma *brute force* dan algoritma *divide and conquer* memiliki *runtime* yang hampir sama dalam membangun kurva beziér kuadratik.

4.1.4 Perbandingan Kompleksitas Waktu

Berdasarkan hasil *runtime* yang didapatkan dari pengujian dan hasil analisis kompleksitas waktu algoritma, dapat dilihat bahwa algoritma *divide and conquer* dan algoritma *brute force* memiliki efisiensi dan kecepatan yang sama dalam membangun kurva beziér kuadratik.

BAB V

KESIMPULAN

Dalam tugas kecil ini, penulis mempelajari kurva beziér dan cara membangun kurva beziér menggunakan dua algoritma, yaitu algoritma *divide and conquer* dan algoritma *brute force*. Berdasarkan analisis dan pengujian yang telah dilakukan, telah disimpulkan bahwa algoritma *divide and conquer* dan algoritma *brute force* memiliki efisiensi dan kecepatan yang sama dalam membuat sebuah kurva beziér kuadratik.

LAMPIRAN

Link Repository: https://github.com/DieroA/Tucil2_13522056

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.		✓