

**Laporan Tugas Kecil 2**  
**IF2211 Strategi Algoritma**  
**Semester II Tahun 2023/2024**

**Membangun Kurva Bézier dengan Algoritma Titik Tengah**  
**berbasis *Divide and Conquer***



Disusun oleh:  
Bastian H. Suryapratama  
13522034  
K-02

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**BANDUNG**  
**2024**

## Daftar Isi

<b>Daftar Isi.....</b>	<b>2</b>
<b>I. Analisis dan Implementasi dalam Algoritma Brute Force.....</b>	<b>3</b>
<b>II. Analisis dan Implementasi dalam Algoritma Divide and Conquer.....</b>	<b>4</b>
<b>III. Source Code Program Implementasi Algoritma.....</b>	<b>6</b>
A. Brute Force 3 Titik Kontrol.....	6
B. Divide and Conquer 3 Titik Kontrol.....	7
C. Divide and Conquer n Titik Kontrol.....	8
<b>IV. Tangkapan Layar.....</b>	<b>11</b>
A. Brute Force 3 Titik Kontrol.....	11
B. Divide and Conquer 3 Titik Kontrol.....	13
C. Divide and Conquer n Titik Kontrol.....	15
D. Divide and Conquer Visualisasi Setiap Iterasi.....	17
<b>V. Hasil Analisis Perbandingan Algoritma Brute Force dengan Divide and Conquer.....</b>	<b>20</b>
<b>VI. Implementasi Bonus.....</b>	<b>22</b>
A. Divide and Conquer n Titik Kontrol.....	22
B. Divide and Conquer Visualisasi Setiap Iterasi.....	23
<b>Lampiran.....</b>	<b>25</b>

## I. Analisis dan Implementasi dalam *Algoritma Brute Force*

Untuk membentuk kurva Bézier kuadratik, kita membutuhkan 3 buah titik, misalkan  $P_0$ ,  $P_1$ , dan  $P_2$ , dengan  $P_0$  dan  $P_2$  sebagai titik kontrol awal dan akhir, dan  $P_1$  menjadi titik kontrol antara. Setelah itu, kita akan mencari titik  $Q_0$  serta  $Q_1$ . Titik  $Q_0$  terletak di antara garis yang menghubungkan  $P_0$  dan  $P_1$ , sedangkan titik  $Q_1$  terletak diantara garis yang menghubungkan  $P_1$  dan  $P_2$ . Kemudian, dari titik  $Q_0$  serta  $Q_1$ , kita dapat mencari titik  $R_0$  yang berada di antara garis yang menghubungkan  $Q_0$  dan  $Q_1$ . Pergerakan titik  $R_0$  inilah yang akan membentuk kurva Bézier kuadratik terhadap titik  $P_0$  dan  $P_2$ .

Berikut ini adalah uraian persamaannya:

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

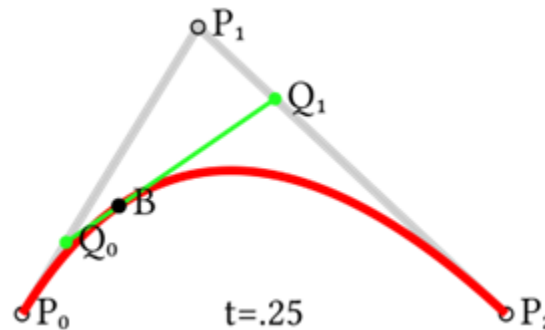
$$Q_1 = B(t) = (1 - t)P_1 + tP_2, t \in [0, 1]$$

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

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

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

Berikut ini adalah ilustrasi dari kasus di atas:



**Gambar 1.** Pembentukan Kurva Bézier Kuadratik.

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

Ada tak hingga nilai  $t$  di antara selang  $[0, 1]$ . Tidak mungkin kita akan melakukan kalkulasi kurva Bézier dengan sempurna yang perlu melibatkan semua nilai  $t$  tersebut. Solusinya, kita hanya akan menghitung dengan beberapa nilai  $t$  yang berjarak sama satu dengan yang lain. Misalnya, dengan 5 iterasi, berarti kita hanya akan melakukan kalkulasi dengan nilai  $t = 0, 0.25, 0.5, 0.75$ , dan  $1$ . Semakin banyak iterasi yang dilakukan, hasilnya akan semakin mulus sehingga semakin mendekati kurva Bézier yang asli.

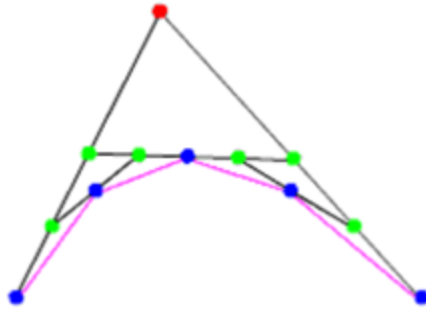
## II. Analisis dan Implementasi dalam *Algoritma Divide and Conquer*

Sama seperti kasus *Brute Force*, untuk membentuk kurva Bézier kuadrat, kita membutuhkan tiga buah titik, misalkan  $P_0$ ,  $P_1$ , dan  $P_2$ .  $P_0$  dan  $P_2$  menjadi titik kontrol awal dan akhir, sedangkan titik  $P_1$  menjadi titik kontrol antara. Berikut ini adalah langkah-langkah yang perlu dilakukan:

1. Buatlah sebuah titik baru  $Q_0$  yang berada di tengah garis yang menghubungkan  $P_0$  dan  $P_1$ , serta titik  $Q_1$  yang berada di tengah garis yang menghubungkan  $P_1$  dan  $P_2$ .
2. Hubungkan  $Q_0$  dan  $Q_1$  sehingga terbentuk sebuah garis baru.
3. Buatlah sebuah titik baru  $R_0$  yang berada di tengah  $Q_0$  dan  $Q_1$ .
4. Buatlah sebuah garis yang menghubungkan  $P_0 - R_0 - P_2$ .

Dengan proses di atas, telah dilakukan 1 buah iterasi dan diperoleh sebuah kurva yang belum cukup mulus dengan aproksimasi 3 buah titik. Untuk membuat sebuah kurva yang lebih baik, perlu dilakukan iterasi lanjutan dengan prosedur sebagai berikut:

5. Setelah mendapatkan garis  $P_0 - R_0 - P_2$ , kita akan membagi persoalan menjadi 2 bagian, yaitu  $P_0 - R_0$  dan  $R_0 - P_2$ .
6. Kita akan menyelesaikan persoalan  $P_0 - R_0$  terlebih dahulu.  $P_0$  akan dijadikan titik kontrol awal,  $Q_0$  berperan sebagai  $P_1$  yang menjadi titik kontrol antara, dan  $R_0$  berperan sebagai titik kontrol akhir.
7. Ulangi langkah 1 - 4. Setelah itu, ulangi langkah 5 - 6 jika diperlukan iterasi tambahan.
8. Setelah persoalan bagian  $P_0 - R_0$  terselesaikan, kita akan menyelesaikan persoalan  $R_0 - P_2$ .  $R_0$  kini berperan sebagai  $P_0$  yang menjadi titik kontrol awal,  $Q_1$  berperan sebagai  $P_1$  yang menjadi titik kontrol antara, dan  $P_2$  berperan sebagai titik kontrol akhir.
9. Ulangi langkah 1 - 4. Setelah itu, ulangi langkah 5 - 6 jika diperlukan iterasi tambahan.
10. Setelah menyelesaikan kedua bagian persoalan, kita akan menghubungkan titik-titik solusi dari kedua persoalan tersebut.
11. Hasil akhirnya, kita akan mendapatkan kurva Bézier kuadrat yang berawal dari titik  $P_0$  dan berakhir pada titik  $P_2$ .



**Gambar 2.** Hasil pembentukan Kurva Bézier Kuadratik dengan divide and conquer setelah iterasi ke-2.

### III. Source Code Program Implementasi Algoritma

Program ini ditulis menggunakan bahasa pemrograman Python. Program tersebut terdiri dari 3 file utama, yaitu BezierBF.py, BezierDnC.py, dan main.py. File BezierBF.py berkaitan dengan implementasi algoritma *brute force*, file BezierDnC.py berkaitan dengan implementasi algoritma *divide and conquer*, sedangkan main.py berkaitan dengan alur utama jalannya program yang akan memanggil fungsi-fungsi di dalam BezierBF.py dan BezierDnC.py.

#### A. Brute Force 3 Titik Kontrol

Algoritma *brute force* untuk 3 titik kontrol diimplementasikan pada fungsi `find_quad_bezier_bf()` yang ada di dalam file BezierBF.py. Berikut ini adalah *source code* fungsi tersebut:

```
# find points in quadratic bezier curve with brute force
def find_quad_bezier_bf(
    control_points: np.ndarray[np.ndarray[float]],
    num_of_iterations: int
) -> np.ndarray[np.ndarray[float]]:

    p0: np.ndarray[float] = control_points[0]
    p1: np.ndarray[float] = control_points[1]
    p2: np.ndarray[float] = control_points[2]

    # find points in bezier curve
    bezier_points: np.ndarray[np.ndarray[float]] = np.empty((0, 2),
dtype=np.float32)
    t: float
    for t in np.linspace(0, 1, num_of_iterations):
        bezier_point: np.ndarray[float] = (1 - t)**2 * p0 + 2 * (1 -
t) * t * p1 + t**2 * p2
        bezier_points = np.append(bezier_points,
np.array([bezier_point]), axis=0)

    return bezier_points
```

## B. Divide and Conquer 3 Titik Kontrol

Algoritma *divide and conquer* untuk 3 titik kontrol diimplementasikan pada fungsi `find_quad_bezier_dnc()` yang ada di dalam file `BezierDnC.py`. Berikut ini adalah *source code* fungsi tersebut:

```
# find points in quadratic bezier curve with divide and conquer
def find_quad_bezier_dnc(
    control_points: np.ndarray[np.ndarray[float]],
    i: int,
    num_of_iterations: int
) -> np.ndarray[np.ndarray[float]]:

    p0: np.ndarray[float] = control_points[0]
    p1: np.ndarray[float] = control_points[1]
    p2: np.ndarray[float] = control_points[2]
    q0: np.ndarray[float] = (p0 + p1) / 2
    q1: np.ndarray[float] = (p1 + p2) / 2
    r: np.ndarray[float] = (q0 + q1) / 2

    # base case when we reach end of iteration
    if i == num_of_iterations:
        return np.array((p0, r, p2))

    # find 1st half control points
    control_points1: np.ndarray[np.ndarray[float]] = np.array((p0,
q0, r))

    # find 2nd half control points
    control_points2: np.ndarray[np.ndarray[float]] = np.array((r,
q1, p2))

    # find bezier curve points from the 1st half and 2nd half
    bezier1: np.ndarray[np.ndarray[float]] =
find_quad_bezier_dnc(control_points1, i + 1, num_of_iterations)
    bezier2: np.ndarray[np.ndarray[float]] =
find_quad_bezier_dnc(control_points2, i + 1, num_of_iterations)

    # combine both result
    return np.concatenate((bezier1, bezier2))
```

## C. Divide and Conquer n Titik Kontrol

Algoritma *divide and conquer* untuk n titik kontrol diimplementasikan pada fungsi `find_bezier_dnc()` yang ada di dalam file `BezierDnC.py`. Fungsi tersebut memanggil fungsi `find_midpoints()` untuk mencari titik tengah yang diperlukan dalam suatu iterasi. Berikut ini adalah *source code* fungsi tersebut:

`find_bezier_dnc()`

```
# find points in general bezier curve with divide and conquer
def find_bezier_dnc(
    control_points: np.ndarray[np.ndarray[float]],
    i: int,
    num_of_iterations: int
) -> np.ndarray[np.ndarray[float]]:

    mid_points: np.ndarray[np.ndarray[float]] =
find_midpoints(control_points)

    # base case when we reach end of iteration
    if i == num_of_iterations:
        return np.array((control_points[0], mid_points[-1],
control_points[-1]))

    # find 1st half control points
    control_points1: np.ndarray[np.ndarray[float]] =
np.array([control_points[0]])
    j: int
    k: int = 1
    for j in range(control_points.shape[0] - 1, 1, -1):
        mid_point: np.ndarray[np.ndarray[float]] =
np.array([mid_points[k - 1]])
        control_points1 = np.append(control_points1, mid_point,
axis=0)
        k += j

    mid_point: np.ndarray[np.ndarray[float]] =
np.array([mid_points[k - 1]])
    control_points1 = np.append(control_points1, mid_point, axis=0)
```



```

# find 2nd half control points
control_points2: np.ndarray[np.ndarray[float]] =
np.array(mid_point)
    for j in range(1, control_points.shape[0] - 1):
        k -= j
        mid_point: np.ndarray[np.ndarray[float]] =
np.array([mid_points[k - 1]])
        control_points2 = np.append(control_points2, mid_point,
axis=0)

    control_points2 = np.append(control_points2,
np.array([control_points[-1]]), axis=0)

# find bezier curve points from the 1st half and 2nd half
bezier1: np.ndarray[np.ndarray[float]] =
findBezier_dnc(control_points1, i + 1, num_of_iterations)
    bezier2: np.ndarray[np.ndarray[float]] =
findBezier_dnc(control_points2, i + 1, num_of_iterations)

# combine both result
return np.concatenate((bezier1, bezier2))

```

### find\_midpoints()

```

# find midpoints for constructing general bezier curve
def find_midpoints(
    points: np.ndarray[np.ndarray[float]]
) -> np.ndarray[np.ndarray[np.ndarray[float]]]:

    mid_points: np.ndarray[np.ndarray[float]] = np.empty((0, 2),
dtype=np.float32)
    for i in range(points.shape[0] - 1):
        mid_point: np.ndarray[float] = np.array([(points[i] +
points[i+1]) / 2])
        mid_points = np.append(mid_points, mid_point, axis=0)

# base case when there is only 1 midpoint result
if mid_points.shape[0] == 1:
    return mid_points;

```

```
mid_points2: np.ndarray[np.ndarray[float]] =  
find_midpoints(mid_points)  
return np.concatenate((mid_points, mid_points2))
```

## IV. Tangkapan Layar

### A. Brute Force 3 Titik Kontrol

Input	Output
<pre>\$ python src/main.py Do you want to input from file or terminal? (file/terminal) file  Enter file name (must be in test folder): test1.txt  Which algorithm do you want to use? 1. divide and conquer 2. brute force enter the number or name: 2  Enter number of iteration: 3  Do you want to visualize the result? (yes/no): no  Execution time: 0.0 ms  █</pre>	<p>The plot shows a blue Bezier curve and an orange brute force approximation. The x-axis ranges from -10 to 0, and the y-axis ranges from 0 to 10. The control points are at (-10, 10), (-2, 8), and (0, 0). The orange curve is a piecewise linear approximation of the blue Bezier curve.</p>
<pre>bitkomputer@LaptopHP MINGW64 /e/Tucil2_13522034 (main) \$ python src/main.py Do you want to input from file or terminal? (file/terminal) file  Enter file name (must be in test folder): test1.txt  Which algorithm do you want to use? 1. divide and conquer 2. brute force enter the number or name: 2  Enter number of iteration: 9  Do you want to visualize the result? (yes/no): no  Execution time: 0.0 ms  █</pre>	<p>The plot shows a blue Bezier curve and an orange brute force approximation. The x-axis ranges from -10 to 0, and the y-axis ranges from 0 to 10. The control points are at (-10, 10), (-8, 9.5), (-6, 9), (-4, 8.5), (-2, 7.5), (-1, 6), (-0.5, 4), (-0.2, 2), and (0, 0). The orange curve is a piecewise linear approximation of the blue Bezier curve.</p>
<pre>bitkomputer@LaptopHP MINGW64 /e/Tucil2_13522034 (main) \$ python src/main.py Do you want to input from file or terminal? (file/terminal) file  Enter file name (must be in test folder): test2.txt  Which algorithm do you want to use? 1. divide and conquer 2. brute force enter the number or name: 2  Enter number of iteration: 33  Do you want to visualize the result? (yes/no): no  Execution time: 0.0 ms  █</pre>	<p>The plot shows a blue Bezier curve and an orange brute force approximation. The x-axis ranges from -60 to 20, and the y-axis ranges from -60 to 60. The control points are at (-60, 60), (-50, 50), (-40, 40), (-30, 30), (-20, 20), (-10, 10), (0, 0), (10, -10), (20, -20), (-40, -40), (-30, -30), (-20, -20), (-10, -10), (0, 0), (10, 10), (20, 20), (-40, -40), (-30, -30), (-20, -20), (-10, -10), (0, 0), (10, 10), (20, 20). The orange curve is a piecewise linear approximation of the blue Bezier curve.</p>

```

$ python src/main.py
Do you want to input from file or terminal? (file/terminal)
file

Enter file name (must be in test folder):
test2.txt

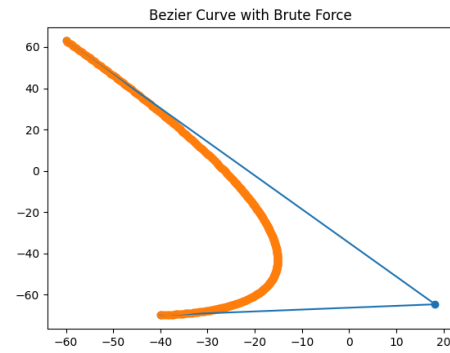
Which algorithm do you want to use?
1. divide and conquer
2. brute force
enter the number or name: 2

Enter number of iteration: 513

Do you want to visualize the result? (yes/no): no

Execution time: 15.154361724853516 ms

```



```

$ python src/main.py
Do you want to input from file or terminal? (file/terminal)
file

Enter file name (must be in test folder):
test3.txt

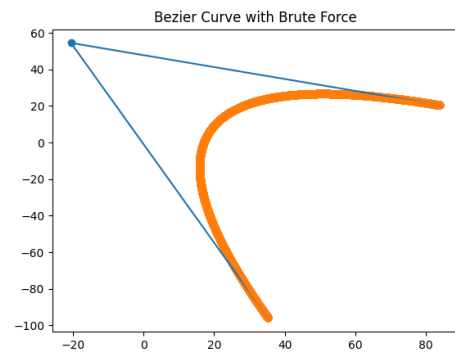
Which algorithm do you want to use?
1. divide and conquer
2. brute force
enter the number or name: 2

Enter number of iteration: 8193

Do you want to visualize the result? (yes/no): no

Execution time: 127.34794616699219 ms

```



```

$ python src/main.py
Do you want to input from file or terminal? (file/terminal)
file

Enter file name (must be in test folder):
test3.txt

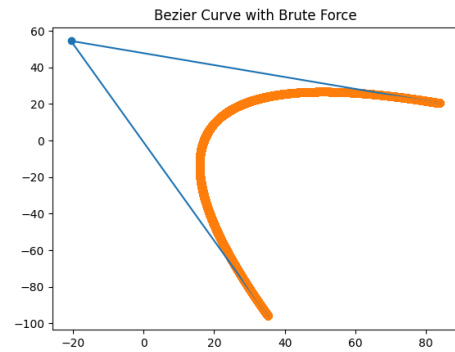
Which algorithm do you want to use?
1. divide and conquer
2. brute force
enter the number or name: 2

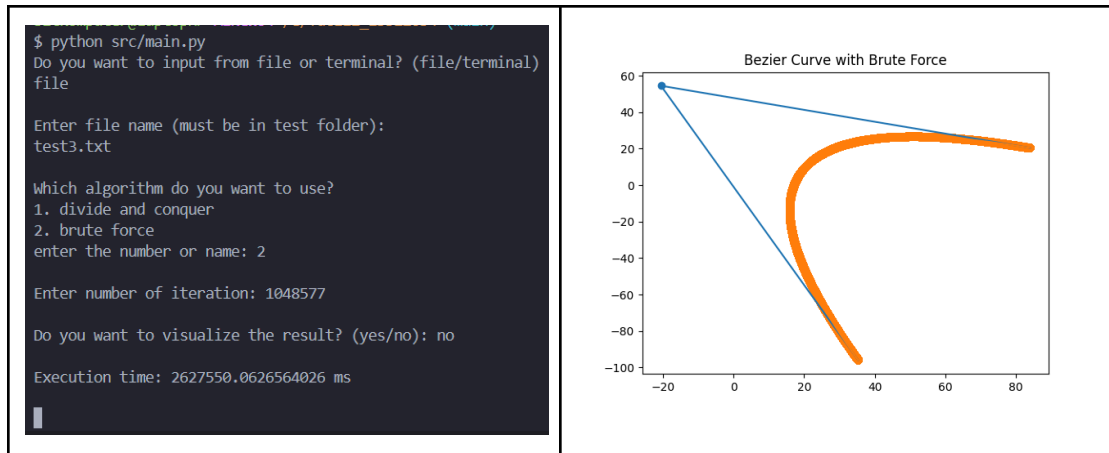
Enter number of iteration: 131073

Do you want to visualize the result? (yes/no): no

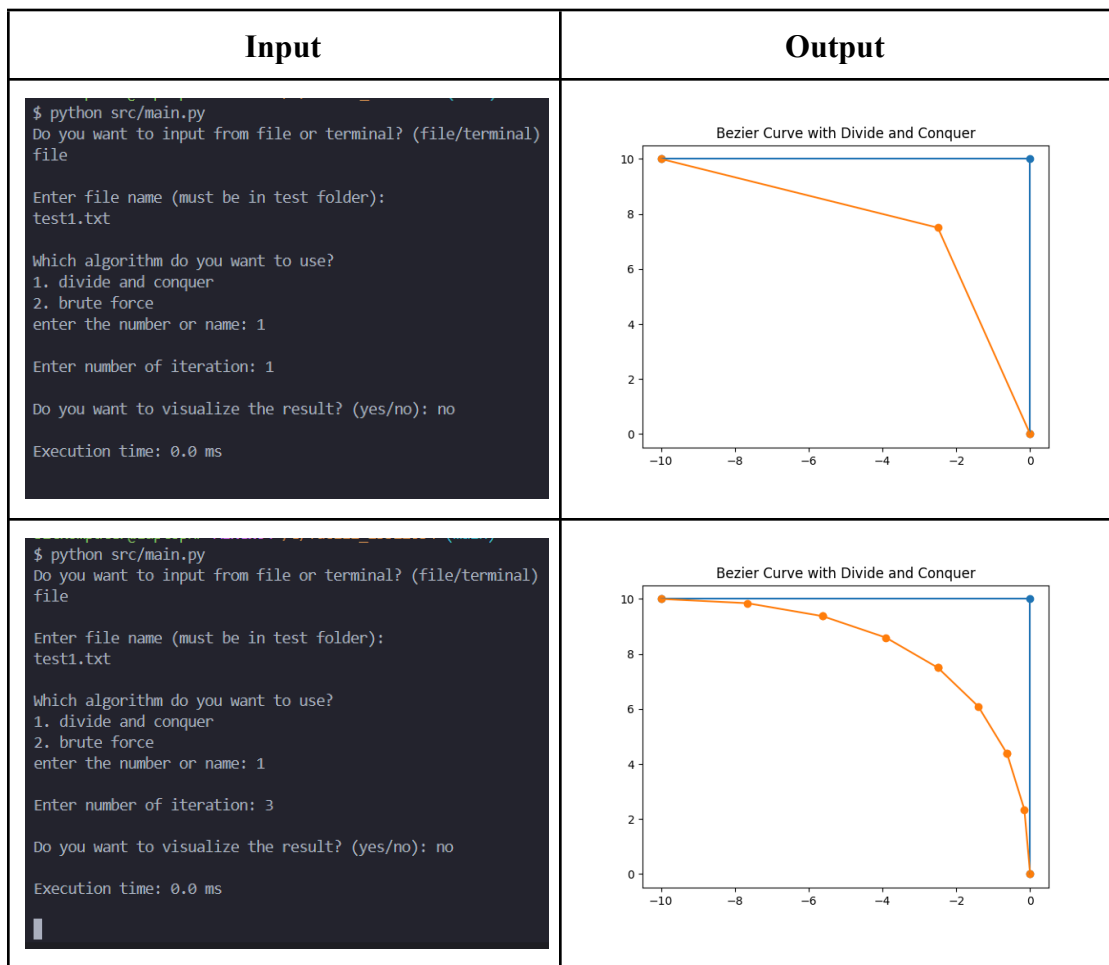
Execution time: 31513.044595718384 ms

```





## B. Divide and Conquer 3 Titik Kontrol



```
$ python src/main.py
Do you want to input from file or terminal? (file/terminal)
file

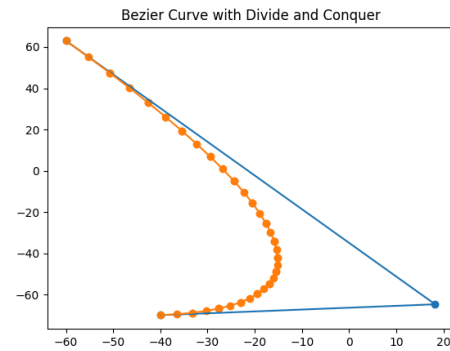
Enter file name (must be in test folder):
test2.txt

Which algorithm do you want to use?
1. divide and conquer
2. brute force
enter the number or name: 1

Enter number of iteration: 5

Do you want to visualize the result? (yes/no): no

Execution time: 0.0 ms
```



```
$ python src/main.py
Do you want to input from file or terminal? (file/terminal)
file

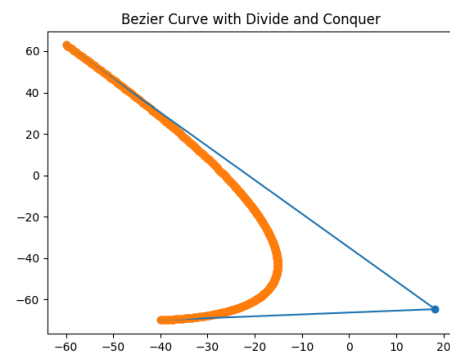
Enter file name (must be in test folder):
test2.txt

Which algorithm do you want to use?
1. divide and conquer
2. brute force
enter the number or name: 1

Enter number of iteration: 9

Do you want to visualize the result? (yes/no): no

Execution time: 15.833854675292969 ms
```



```
$ python src/main.py
Do you want to input from file or terminal? (file/terminal)
file

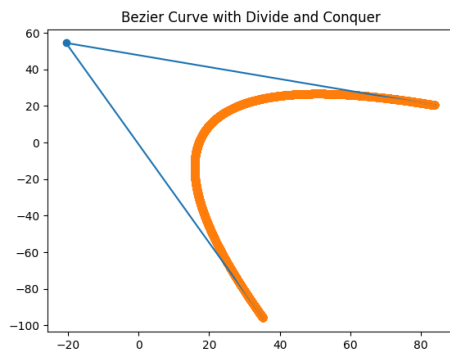
Enter file name (must be in test folder):
test3.txt

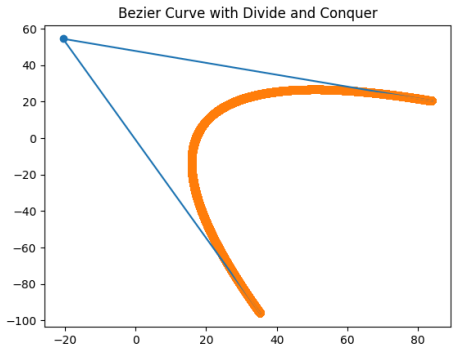
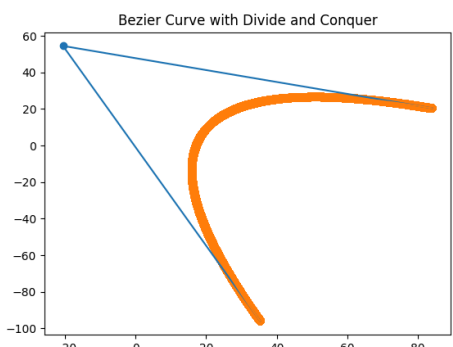
Which algorithm do you want to use?
1. divide and conquer
2. brute force
enter the number or name: 1

Enter number of iteration: 13

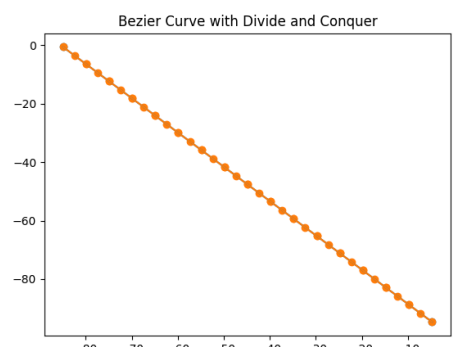
Do you want to visualize the result? (yes/no): no

Execution time: 126.65081024169922 ms
```



<pre> \$ python src/main.py Do you want to input from file or terminal? (file/terminal) file  Enter file name (must be in test folder): test3.txt  Which algorithm do you want to use? 1. divide and conquer 2. brute force enter the number or name: 1  Enter number of iteration: 17  Do you want to visualize the result? (yes/no): no  Execution time: 1768.5353755950928 ms </pre>	
<pre> \$ python src/main.py Do you want to input from file or terminal? (file/terminal) file  Enter file name (must be in test folder): test3.txt  Which algorithm do you want to use? 1. divide and conquer 2. brute force enter the number or name: 1  Enter number of iteration: 20  Do you want to visualize the result? (yes/no): no  Execution time: 14195.68920135498 ms </pre>	

### C. Divide and Conquer n Titik Kontrol

Input	Output
<pre> \$ python src/main.py Do you want to input from file or terminal? (file/terminal) file  Enter file name (must be in test folder): test4.txt  Enter number of iteration: 5  Do you want to visualize the result? (yes/no): no  Execution time: 0.0 ms </pre>	

```

$ python src/main.py
Do you want to input from file or terminal? (file/terminal)
file

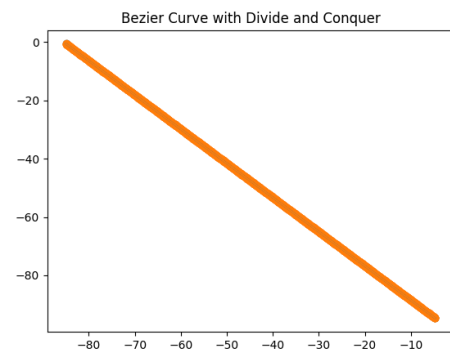
Enter file name (must be in test folder):
test4.txt

Enter number of iteration: 10

Do you want to visualize the result? (yes/no): no

Execution time: 22.44114875793457 ms

```



```

$ python src/main.py
Do you want to input from file or terminal? (file/terminal)
terminal

Enter number of points: 5

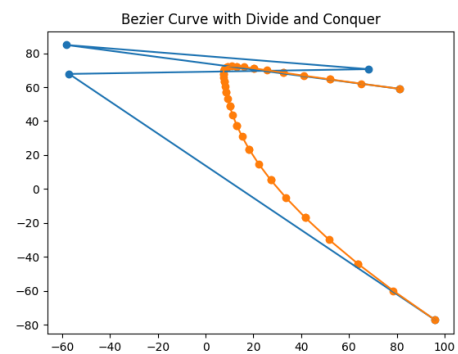
Enter 5 points
Point 1 (x, y): 80.97, 59.06
Point 2 (x, y): -58.39, 84.91
Point 3 (x, y): 68.19, 70.62
Point 4 (x, y): -57.06, 67.80
Point 5 (x, y): 96.01, -77.23

Enter number of iteration: 5

Do you want to visualize the result? (yes/no): no

Execution time: 0.0 ms

```



```

$ python src/main.py
Do you want to input from file or terminal? (file/terminal)
terminal

Enter number of points: 5

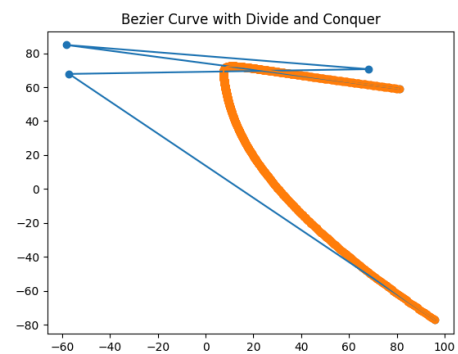
Enter 5 points
Point 1 (x, y): 80.97, 59.06
Point 2 (x, y): -58.39, 84.91
Point 3 (x, y): 68.19, 70.62
Point 4 (x, y): -57.06, 67.80
Point 5 (x, y): 96.01, -77.23

Enter number of iteration: 10

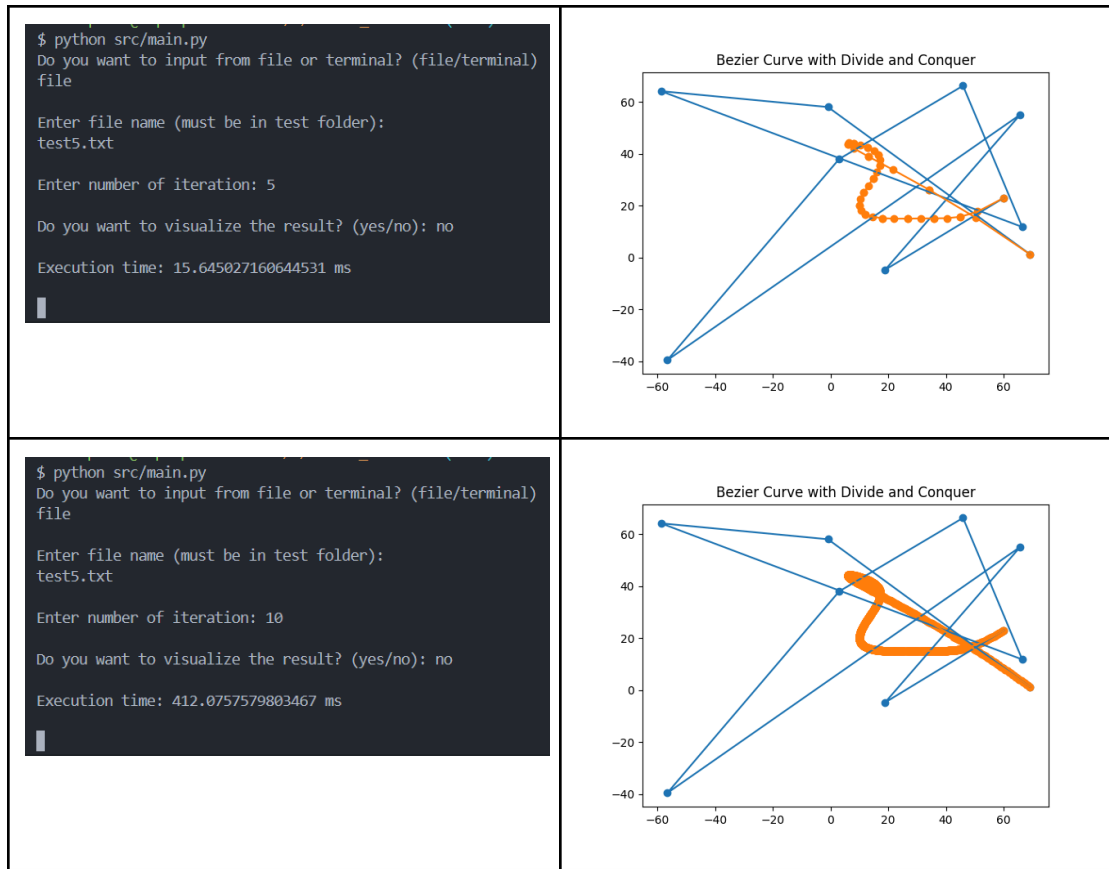
Do you want to visualize the result? (yes/no): no

Execution time: 129.1651725769043 ms

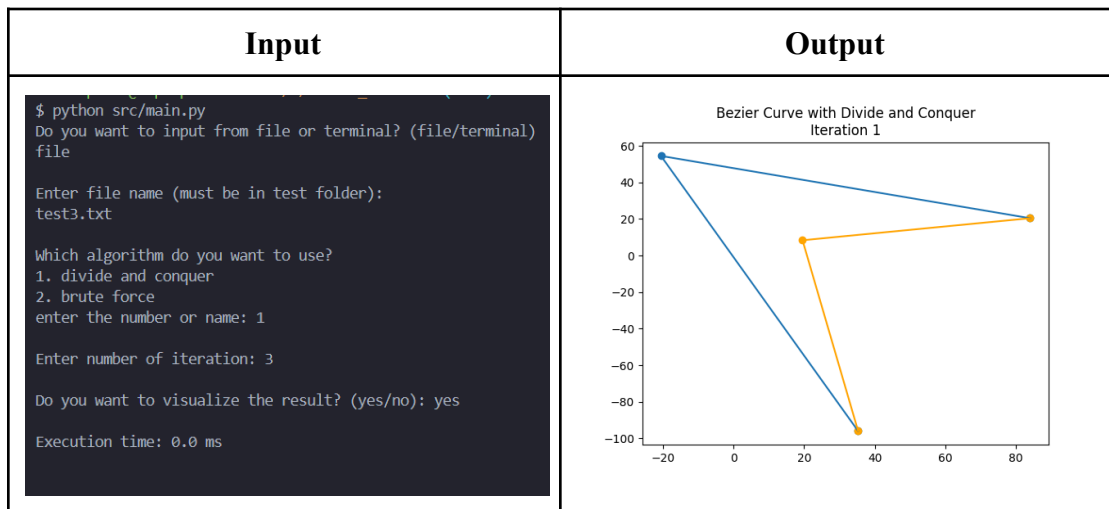
```

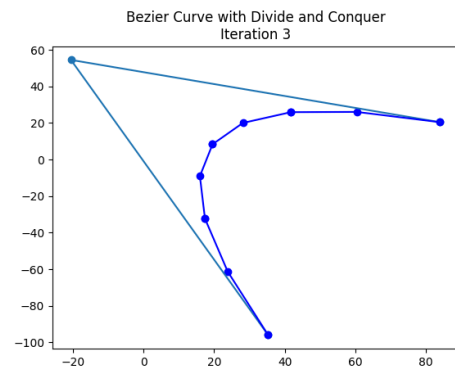
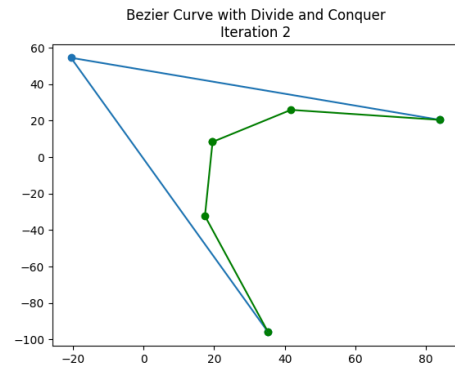






#### D. *Divide and Conquer* Visualisasi Setiap Iterasi





```
$ python src/main.py
Do you want to input from file or terminal? (file/terminal)
terminal

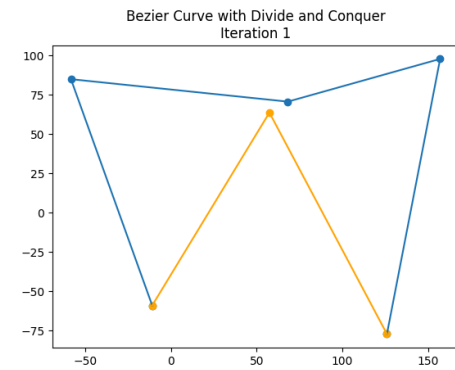
Enter number of points: 5

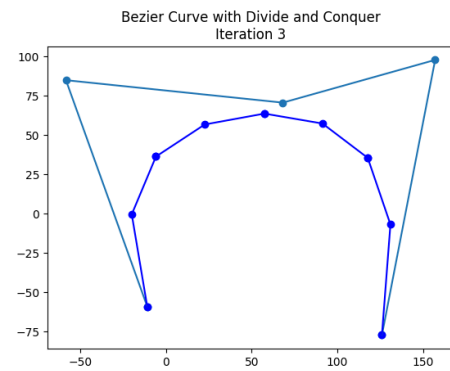
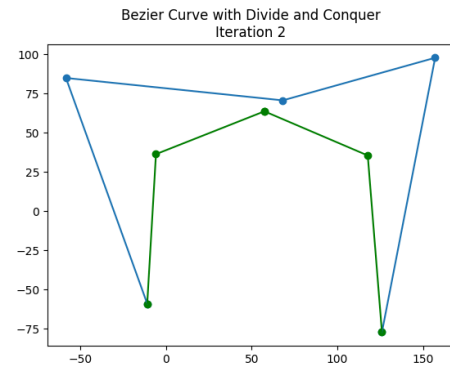
Enter 5 points
Point 1 (x, y): -10.97, -59.06
Point 2 (x, y): -58.39, 84.91
Point 3 (x, y): 68.19, 70.62
Point 4 (x, y): 157.06, 97.80
Point 5 (x, y): 126.01, -77.23

Enter number of iteration: 3

Do you want to visualize the result? (yes/no): yes

Execution time: 0.0 ms
```





## V. Hasil Analisis Perbandingan Algoritma *Brute Force* dengan *Divide and Conquer*

Perbandingan algoritma yang dituliskan pada bagian ini adalah algoritma untuk 3 buah titik kontrol karena algoritma *brute force* untuk  $n$  buah titik kontrol tidak diimplementasikan.

Pada algoritma *brute force*, terdapat 2 hal utama yang dilakukan, yaitu mencari koordinat titik dan memasukkan titik tersebut ke dalam kumpulan titik-titik solusi. Untuk  $n$  buah titik hasil yang diinginkan, kita perlu melakukan kedua operasi tersebut sebanyak  $n$  kali.

Dengan demikian, kita mendapatkan persamaan  $T(n)$  sebagai berikut:

$$T(n) = n + n = 2n$$

Dengan persamaan  $T(n)$  tersebut, kita mendapatkan persamaan  $O(n)$  sebagai berikut:

$$O(n) = n$$

Pada algoritma *divide and conquer*, terdapat 3 hal utama yang dilakukan, yaitu memecah persoalan menjadi 2 bagian, mencari titik-titik solusi dari setiap bagian persoalan. serta menggabungkan titik-titik solusi dari kedua persoalan. Untuk  $n$  buah titik hasil yang diinginkan, dari persoalan bagian pertama, kita akan mendapatkan  $n/2$  titik solusi. Dari persoalan bagian kedua, kita juga akan mendapatkan  $n/2$  titik solusi.

Dengan demikian, kita mendapatkan persamaan  $T(n)$  sebagai berikut:

$$T(n) = a, n = 2$$

$$T(n) = 2T\left(\frac{n}{2}\right) + c, n > 2$$

$c$  adalah waktu yang dibutuhkan untuk menggabungkan solusi

Dengan persamaan  $T(n)$  tersebut, kita mendapatkan persamaan  $O(n)$  dengan Teorema Master sebagai berikut:

$$O(n) = n$$

Dapat disimpulkan, kompleksitas algoritma untuk algoritma *brute force* dan *divide and conquer* adalah sama. Hal ini cukup konsisten dengan hasil percobaan yang ditunjukkan oleh percobaan ke-4 dan ke-5. Akan tetapi, pada percobaan ke-6 dan ke-7, waktu eksekusi yang ditampilkan cukup jauh perbedaannya. Hal tersebut dapat disebabkan oleh perbedaan banyaknya alokasi memori yang dilakukan saat memanggil fungsi `append()` dan `concatenate()`. Alokasi memori pada *brute force* jauh lebih sering dilakukan daripada alokasi memori pada *divide and conquer*. Untuk  $n$  buah titik yang dihasilkan, fungsi

`append()` pada *brute force* perlu dilakukan sebanyak  $n$  kali, sedangkan operasi `concatenate()` pada *divide and conquer* dilakukan sebanyak nilai yang jauh lebih kecil dari nilai  $n$  (terutama untuk nilai  $n$  yang sangat besar).

Selain itu, perbedaan waktu eksekusi juga dapat disebabkan oleh perbedaan efisiensi fungsi `append()` dan `concatenate()` yang disediakan oleh package NumPy.

## VI. Implementasi Bonus

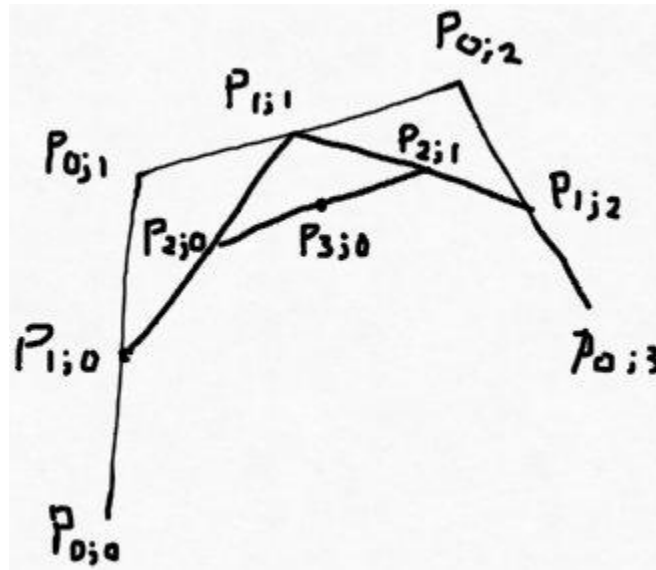
### A. Divide and Conquer n Titik Kontrol

Secara garis besar, algoritma *divide and conquer* untuk  $n$  titik kontrol mirip dengan algoritma *divide and conquer* untuk 3 titik kontrol. Perbedaannya terletak pada banyaknya titik tengah yang perlu dicari serta titik kontrol yang dipakai dalam pembagian persoalan.

Langkah-langkah untuk mencari titik tengah adalah sebagai berikut:

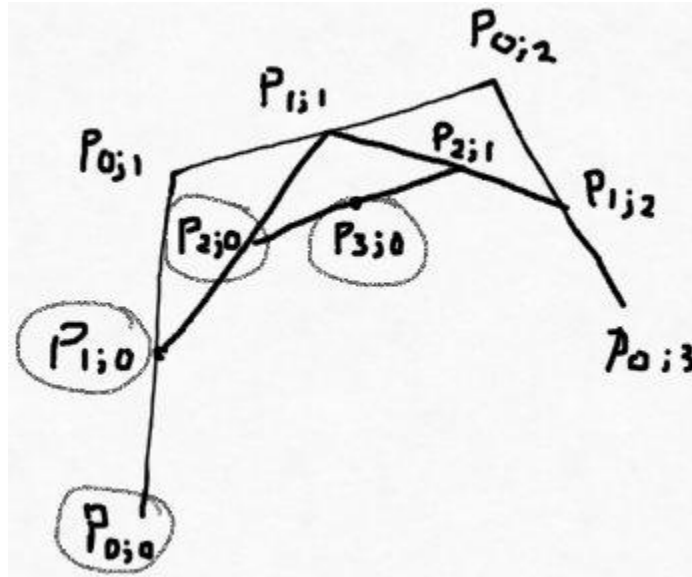
1. Untuk  $n$  titik kontrol, misalnya  $P_{0,0}, P_{0,1}, P_{0,2}, \dots, P_{0,n-1}$ , kita akan mencari titik tengah antara 2 buah titik yang berurutan, yaitu  $P_{0,0}$  dan  $P_{0,1}$ ,  $P_{0,1}$  dan  $P_{0,2}$ ,  $P_{0,2}$  dan  $P_{0,3}$ , dan seterusnya sampai  $P_{0,n-2}$  dan  $P_{0,n-1}$ . Misalkan hasilnya adalah  $P_{1,0}, P_{1,1}, P_{1,2}, \dots, P_{1,n-2}$ .
2. Dari sejumlah  $n-1$  titik tengah tersebut, kita akan kembali mencari titik tengah antara 2 buah titik yang berurutan seperti pada langkah sebelumnya. Misalkan hasilnya adalah  $P_{2,0}, P_{2,1}, P_{2,2}, \dots, P_{2,n-3}$ .
3. Dari sejumlah  $n-2$  titik tengah tersebut, kita akan kembali mencari titik tengah antara 2 buah titik yang berurutan seperti pada langkah nomor 1. Misalkan hasilnya adalah  $P_{3,0}, P_{3,1}, P_{3,2}, \dots, P_{3,n-4}$ .
4. Kita lakukan terus-menerus langkah sebelumnya. Semakin lama, banyaknya titik tengah yang perlu dicari akan semakin sedikit. Akhirnya, kita hanya akan mendapatkan 1 buah titik tengah. Kita sebut 1 buah titik tersebut sebagai  $P_{n-1,0}$ .

Berikut ini adalah ilustrasi untuk langkah 1 - 4 (dengan 4 titik kontrol):

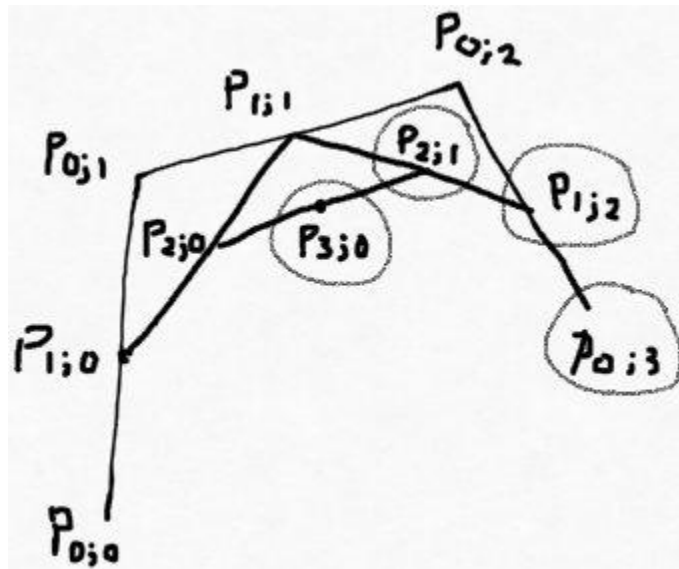


Sebagian titik-titik tengah tersebut akan dipakai dalam pembagian persoalan menjadi 2 bagian. Berikut ini adalah titik-titik yang akan dipakai sebagai titik kontrol:

5. Untuk menyelesaikan bagian persoalan pertama, titik-titik yang dijadikan titik kontrol adalah  $P_{0;0}$ ,  $P_{1;0}$ ,  $P_{2;0}$ , ...,  $P_{n-1;0}$ . Berikut ini adalah ilustrasi titik-titik yang dijadikan titik kontrol (dengan 4 titik kontrol):



6. Untuk menyelesaikan bagian persoalan kedua, titik-titik yang dijadikan titik kontrol adalah  $P_{n-1;0}$ ,  $P_{n-2;1}$ ,  $P_{n-3;2}$ , ...,  $P_{0;n-1}$ . Berikut ini adalah ilustrasi titik-titik yang dijadikan titik kontrol (dengan 4 titik kontrol):



7. Proses-proses lainnya akan mirip seperti algoritma *divide and conquer* untuk 3 titik kontrol.

## B. Divide and Conquer Visualisasi Setiap Iterasi

Visualisasi yang dilakukan adalah visualisasi hasil akhir dari setiap iterasi yang dilakukan. Untuk setiap iterasi, akan dilakukan *plotting* titik-titik yang dihasilkan dari masing-masing iterasi tersebut. Kemudian, akan ditampilkan juga garis yang menghubungkan titik-titik tersebut.



## Lampiran

Pranala repository GitHub: [https://github.com/bastianhs/Tucil2\\_13522034](https://github.com/bastianhs/Tucil2_13522034)

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✓	
2. Program dapat melakukan visualisasi kurva Bézier.	✓	
3. Solusi yang diberikan program optimal.	✓	
4. <b>[Bonus]</b> Program dapat membuat kurva untuk $n$ titik kontrol.	✓	
5. <b>[Bonus]</b> Program dapat melakukan visualisasi proses pembuatan kurva.	✓	