Laporan Tugas Kecil 2
IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Disusun oleh:

1. Maximilian Sulistiyo 13522061
2. Muhamad Rafli Rasyiidin 13522088

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

BAB 1
Deskripsi Masalah

Kurva Bézier merupakan kurva yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Dalam pembuatannya kurva ini memiliki N titik kontrol dimana setiap titik kontrol akan dihubungkan dengan sebuah kurva. Oleh karena itu suatu kurva bézier didefinisikan dengan set titik kontrol dari P0 sampai Pn dengan n biasa disebut dengan order, n = 1 untuk linier, n = 2 untuk kuadratik, n = 3 untuk kubik dan seterusnya. P0 dan Pn menjadi titik pada ujung kurva sementara titik lainnya tidak terletak dalam kurva.

Menggunakan dua titik kontrol P0 dan P1 kita dapat membuat kurva bézier linier. Dibuat sebuah titik Q0 yang merupakan titik diantara P0 dan P1 maka posisi Q0 dapat dinyatakan dengan persamaan parametrik

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

t menyatakan seberapa jauh B(t) dari P0 ke P1, misal t = 0.5 maka B(t) tepat ditengah antara P0 dan P1. Maka seluruh variasi nilai t dari 0 hingga 1 akan membuat persamaan B(t) yang membentuk garis lurus antara P0 dan P1.

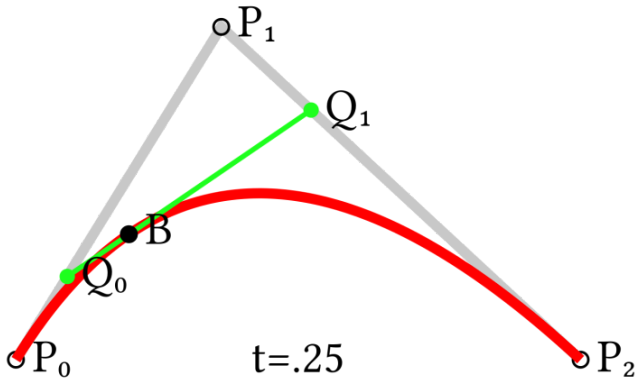
Kemudian ditambahkan sebuah titik kontrol baru P2 dimana P0 dan P2 sekarang menjadi awal dan akhir dari kurva dan P1 menjadi titik kontrol diantara keduanya. Dibuat sebuah titik Q1 yang merupakan titik di antara P1 dan P2 yang akan membuat kurva bézier yang berbeda dengan kurva linier letak Q0. Kemudian dibuat titik baru R0 yang berada di antara Q0 dan Q1. Seiring bergeraknya titik ini kita dapat membuat sebuah kurva bézier kuadratik terhadap titik P0 dan P2, berikut adalah penguraian 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 pensubtitusian nilai dari Q0 dan Q1 maka akan didapatkan persamaan

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



Gambar 1. Contoh ilustrasi Bézier Kuadratik pada t = 0.25

Sumber : https://simonhalliday.com/2017/02/15/quadratic-bezier-curve-demo/

Langkah-langkah tersebut dapat dilakukan kembali untuk membentuk kurva bézier dengan lebih dari tiga titik kontrol seperti kurva bézier kubik dengan empat titik kontrol dan kurva bézier kuartik dengan lima titik kontrol, berikut adalah persamaan bézier kubik dan kuartik dengan langkah yang sama

$$S_0 = B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + (1-t) t^2 P_2 + t^3 P_3 \qquad t \in [0,1]$$

$$T_0 = B(t) = (1-t)^4 P_0 + 4(1-t)^3 t P_1 + 6(1-t)^2 t^2 P_2 + 4(1-t) t^3 P_3 + t^4 P_4 \qquad t \in [0,1]$$

Dapat dilihat semakin banyak titik kontrol yang digunakan maka persamaan pun semakin panjang dan rumit. Oleh karena itu kami ditugaskan untuk mengefektifkan pembentukan kurva Bézier dengan algoritma titik tengah berbasis divide and conquer.

BAB 2
Landasan Teori

2.1 Penjelasan Brute Force

Algoritma brute force dapat dideskripsikan sebagai algoritma yang menyelesaikan suatu masalah secara straight forward / lempang. Hal ini karena algoritma brute force memecahkan suatu masalah dengan cara mencoba semua solusi yang mungkin terjadi. Biasanya algoritma ini pun didasarkan pada pernyataan pada persoalan (problem statement) dan juga definisi/konsep yang dilibatkan.

Algoritma brute force memiliki beberapa ciri khas yaitu sederhana, jelas, dan langsung dimana algoritma yang diimplementasikan tidak memerlukan pemecahan masalah yang canggih. Namun karena hal tersebut maka algoritma menjadi sangat tidak efisien karena memerlukan banyak enumerasi untuk mencari semua solusi yang mungkin terjadi. Oleh karena itu juga dijaminkan bahwa jika terdapat sebuah solusi maka algoritma ini akan menemukannya.

2.2 Penjelasan Divide and Conquer

Algoritma divide and conquer dapat dipecah menjadi tiga langkah pengerjaan. Pertama adalah divide dimana kita memecah permasalahan menjadi beberapa sub-persoalan yang memiliki kemiripan yang sama dengan permasalahan sebelumnya namun memiliki ukuran lebih kecil, idealnya setiap sub-persoalan berukuran hampir sama. Kedua ialah conquer dimana masing-masing sub-persoalan diselesaikan, secara langsung jika ukuran sudah kecil atau secara rekursif jika ukuran masih besar. Ketiga dan terakhir adalah combine dimana semua solusi sub-persoalan digabungkan untuk membentuk solusi persoalan awal.

Biasanya objek persoalan yang dimasukkan adalah array, matriks, eksponen, polinom. Dengan ini setiap sub-persoalan memiliki karakteristik yang sama namun dengan ukuran yang lebih kecil. Oleh karena itu juga penyelesaian dengan divide and conquer biasanya diungkapkan dengan skema rekursif.

BAB 3
Analisis dan Implementasi Masalah

3.1 Analisis dan Implementasi dengan Algoritma Brute Force

Dalam pembentukan kurva bezier menggunakan *brute force*, kita dapat melakukan langkah-langkah berikut:

1. Tentukan jumlah titik beserta titik koordinatnya dan banyak iterasi yang akan dilakukan
2. Masukkan ketiga hal tersebut ke dalam program
3. Program akan mencari titik yang membentuk kurva bezier menggunakan persamaan $B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i$, dengan melakukan iterasi nilai t pada rentang [0,1] dan *increment* sebesar $\frac{1}{iterasi}$
4. Setelah program selesai mencari titik-titik kurva bezier, program akan menampilkan grafik yang terbentuk pada layar

Pada algoritma *brute force*, iterasi yang dilakukan agak berbeda dengan *divide and conquer*. Misalkan kita menggunakan *divide and conquer* dengan iterasi sebanyak X, maka program akan menghasilkan N titik pembentuk kurva bezier (titik akhir yang dipakai untuk membuat grafik). Jika kita ingin menggunakan *brute force*, kita tidak bisa menggunakan iterasi sebanyak X untuk menghasilkan jumlah titik yang sama (N titik). Melainkan, kita harus menggunakan iterasi sebanyak 2^X agar jumlah titik yang dihasilkan kurang lebih sama dengan *divide and conquer*. Pada program yang kami buat, iterasi yang dilakukan telah disesuaikan sehingga ketika pengguna memasukkan jumlah iterasi sebanyak X pada *divide and conquer* dan *brute force*, program akan menghasilkan jumlah titik yang sama.

Dalam algoritma *brute force*, terjadi 2 iterasi dengan 1 iterasi berada di dalam iterasi lainnya. Iterasi pertama memiliki rentang [0,1] dengan *increment* sebesar *1/m* dimana *m* adalah jumlah iterasi yang akan dilakukan untuk membentuk kurva bezier. Untuk menyamakan jumlah titik yang didapatkan dengan algoritma *divide and conquer*, maka *m* harus disesuaikan sehingga $m = 2^n$, dengan *n* adalah jumlah iterasi pada algoritma *divide and conquer*. Iterasi kedua (berada di dalam iterasi pertama) memiliki rentang [1,k] dengan k adalah jumlah titik kontrol. Dari informasi tersebut, kita dapat menentukan T(n) dan O(n) untuk algoritma *brute force*:

$$T(n) = k \times 2^n = O(2^n)$$

Berikut merupakan *source code* dari algoritma *brute force*:

algorithm.py

```
30  def bezierBF(list, nTitik, iterasi, points):
31      inc = 1/iterasi
32      N = nTitik-1
33      for t in np.arange(inc,1,inc):
34          x = 0
35          y = 0
36          list_index = 0
37          nPangkat_p = N
38          nPangkat_t = 0
39          p = 1-t
40          for j in range (0,nTitik):
41              fact = (factorial(N)//(factorial(j)*factorial(N-j)))
42              x += fact*(p**nPangkat_p)*(t**nPangkat_t)*(list[list_index][0])
43              y += fact*(p**nPangkat_p)*(t**nPangkat_t)*(list[list_index][1])
44              list_index += 1
45              nPangkat_p -= 1
46              nPangkat_t += 1
47          points.append((x,y))
```

3.2 Analisis dan Implementasi dengan Algoritma Divide and Conquer

Dalam penyelesaian menggunakan divide and conquer kita melakukan langkah langkah berikut:

1. Pertama program menerima tiga titik kontrol kurva bézier dan juga banyak iterasi yang diinginkan
2. Program menginisialisasikan array kosong bezierPoints yang akan menggambarkan kurva bézier
3. Program memanggil fungsi rekursif bezierDnC yang pada kasus basis yaitu pada saat iterasi = 0 akan membuat sebuah bezier linier dengan cara memasukkan titik ujung dari kontrol point, dalam kasus ini karena hanya tiga titik kontrol, misal P0, P1, dan P2 maka program akan memasukkan P0 dan P2 ke dalam bezierPoints
4. Jika bukan kasus basis maka program akan menghitung M0 yang merupakan titik tengah antara P0 dan P1 dan juga menghitung M1 yang merupakan titik tengah antara P1 dan P2.

Kemudian dihitung juga Q0 yang merupakan titik tengah dari M0 dan M1, kemudian program akan memanggil fungsi bezierDnC dengan bezierPoints dan iterasi dikurang satu dengan titik kontrol P0, M0 dan Q0 untuk menyelesaikan kurva pada sisi kiri dan dengan titik kontrol Q0, M1, dan P2 untuk menyelesaikan kurva pada sisi kanan.

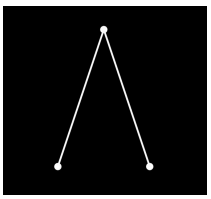
5. Setelah perhitungan titik selesai maka bezierDnC mengeluarkan bezierPoints yang sudah merepresentasikan kurva bézier sesuai dengan iterasi yang diinginkan, program pun akan menampilkan kurva dengan plot.

Melihat algoritma merupakan fungsi rekursi maka kita dapat menyatakan bahwa:

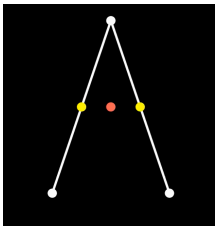
1. Pada kasus basis yaitu iterasi = 0 maka titik ujung P0 dan P2 akan dimasukkan ke dalam array bezier Points
2. Pada kasus iterasi > 0 maka algoritma akan menghitung M0 yang merupakan titik tengah antara P0 dan P1 dan juga M1 yang merupakan titik tengah antara P1 dan P2 kemudian menghitung titik tengah antara M0 dan M1 untuk mendapatkan Q0.
    a. Divide : Setelah menghitung M0, M1, dan Q0 pun algoritma membagi titik kontrol menjadi dua yaitu P0, M0 dan Q0 dan juga Q0, M1, dan P2
    b. Conquer : Algoritma menyelesaikan semua sub persoalan menggunakan dua set titik kontrol tersebut untuk mencari kurva pada sisi kanan dan juga sisi kiri hingga mencapai kasus basis dimana titik P0 dan P2 dianggap sebagai titik dari kurva bézier
    c. Combine : Titik-titik bezierPoints yang dihasilkan dari hasil tiap sub-persoalan pun digabungkan menjadi satu untuk mendapatkan kurva bézier secara keseluruhan.

Untuk mempermudah ilustrasi dari langkah program maka berikut adalah contoh eksekusi algoritma dengan tiga titik dan iterasi sebanyak dua:

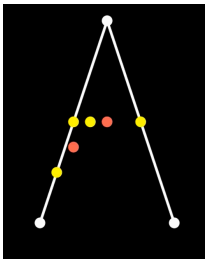
1. Pembentukan 3 titik kontrol awal P0 P1 dan P2



2. Pencarian titik M0 dan M1 (berwarna kuning) sekaligus Q0 (berwarna merah)

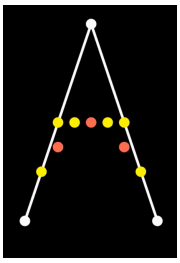


3. Membagi set titik kontrol menjadi P0, M0 dan Q0 dan juga Q0, M1, dan P2
4. Menggunakan P0, M0 dan Q0 kemudian melakukan langkah yang sama dengan sebelumnya untuk menghitung M0, M1 (keduanya berwarna kuning) dan Q0 (berwarna merah) yang baru, kemudian membagi menjadi dua set titik kontrol seperti sebelumnya

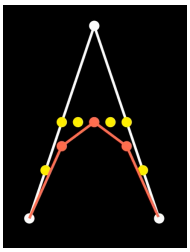


5. Karena iterasi sudah dua kali maka setiap ujung dari titik kontrol dimasukkan ke dalam array bezierPoints

6. Melakukan hal yang sama dari langkah 4 dan 5 untuk titik kontrol Q0, M1, dan P2



7. Proses pun selesai dan bezierPoints berisi titik-titik dari kurva bézier kuadratik



Untuk melihat animasi dari ilustrasi tersebut maka kunjungi pranala :
https://drive.google.com/file/d/1T4SwtmjWEPDpVz8BfkUzA-Qm7nW-9YJ2/view?usp=sharing

Metode animasi tersebut menggunakan library python bernama Manim. Metode tersebut tidak dijadikan metode animasi pada bonus dikarenakan pada iterasi yang banyak maka animasi akan sangat lama dan juga menghasilkan titik-titik yang akan membuat visualisasi membingungkan

Setelah semua penjelasan diatas kami dapat menyatakan kompleksitas waktu dari program ini dimana:

- Pada kasus basis hanya memasukkan titik P0 dan P2 ke dalam array bezierPoints maka memiliki kompleksitas O(*1)* dimana O(*1)* merepresentasikan waktu konstan untuk memasukkan kedua titik tersebut ke dalam array bezierPoints
- Pada kasus iterasi > 0 maka algoritma menghitung 3 titik tengah dan juga membagi menjadi dua set titik kontrol dan memanggil fungsi rekursif kembali dengan titik kontrol baru dan iterasi dikurang satu

Dengan itu dapat disimpulkan bahwa T(n) adalah :

$$T(n) = \begin{cases} 1 & , n = 0 \\ 2T(n-1) + 3 & , n > 1 \end{cases}$$

Melihat bahwa b =1 disini, maka kita tidak dapat menggunakan teorema master sehingga harus dilakukan penyelesaian fungsi rekursi dimana

$$\begin{aligned} T(n) &= 2T(n-1) + 3 \\ &= 2(2T(n-2) + 3) + 3 = 2^2T(n-2) + 2.3 \\ &= 2(2^2T(n-3) + 2k) + k = 2^3T(n-3) + 2^23 \\ &\dots \\ &= 2^nT(0) + 3(2^n - 1) = 2^n + 3(2^n - 1) = O(2^n) \end{aligned}$$

Berikut merupakan *source code* dari algoritma *brute force*:

algorithm.py

```python
def bezierTigaDnC(P0, P1, P2, iterasi, points):
    if iterasi == 0:
        points.append(P0)
        points.append(P2)
    else:
        M0 = (P0 + P1) / 2
        M1 = (P1 + P2) / 2
        Q = (M0 + M1) / 2
        bezierDnC(P0, M0, Q, iterasi - 1, points)
        bezierDnC(Q, M1, P2, iterasi - 1, points)
```

3.3 Analisis dan Implementasi dengan Algoritma Divide and Conquer untuk N Titik Kontrol

Untuk generalisasi titik kontrol N >= 3 titik dapat dilakukan analisis dari penyelesaian menggunakan tiga titik. Dilihat bahwa jika titik kontrol dimasukkan ke dalam suatu array maka:

$$[P_0, P_1, P_2]$$

$$[M_0, M_1]$$

$$[Q_0]$$

Disini M0 merupakan titik tengah dari P0 dan P1 dan M1 merupakan titik tengah dari P1 dan P2. Kemudian Q0 merupakan titik tengah dari M0 dan M1. Untuk pembagian dua set titik kontrol terlihat bahwa titik kontrol set baru mengambil dari ujung dari tiap iterasi dimana iterasi kiri mengambil P0, M0, dan Q0 dan iterasi kanan mengambil Q0, M1, dan P2. Dengan ini kita mencoba membawa algoritma ini ke empat titik sehingga

$$[P_0, P_1, P_2, P_3]$$

$$[M_0, M_1, M_2]$$

$$[Q_0, Q_1]$$

$$[R_0]$$

Terlihat bahwa sama dengan sebelumnya dimana M0 merupakan titik tengah dari P0 dan P1, M1 merupakan titik tengah dari P1 dan P2, dan terdapat tambahan titik M2 yang merupakan titik tengah dari P2 dan P3. Setelah itu sama seperti sebelumnya bahwa Q0 merupakan titik tengah dari M0 dan M1 dan titik baru Q1 yang merupakan pertengahan M1 dan M2. Terakhir terdapat titkk baru R0 yang merupakan pertengahan dari Q0 dan Q1. Terlihat juga set titik baru merupakan titik P0, M0, Q0, R0 untuk sisi kiri dan R0, Q1, M2, P3. Sama seperti sebelumnya dimana titik kontrol baru merupakan titik ujung dari setiap iterasi pembagian ini. Dengan dua contoh tersebut kita dapat men-generalisir untuk mendapatkan titik kontrol pada N>=3:

$$[P_0,\ P_1,\ P_2,\ P_3,\ \dots\ P_n]$$

$$[M_0,\ M_1,\ M_2,\ \dots\ M_n]$$

$$[Q_0,\ Q_1,\ \dots\ Q_n]$$

$$[R_0,\ \dots,\ R_n]$$

$$\dots$$

$$[Z_0]$$

Pertama di inisialisasi array leftControlPoints dan rightControlPoints, kemudian dimasukkan P0 pada leftControlPoints yang dimasukkan dari kanan dan dimasukkan Pn pada rightControlPoints yang dimasukkan dari kiri array. Proses pemasukkan titik baru melakukan hal yang sama dimana masuknya titik baru pada leftControlPoints dari kanan array dan untuk righControlPoints dari sisi kiri array, hal ini untuk menjaga order dari masuknya titik dan juga agar titik kontrol tepat terbagi dua pada Z0.

Kemudian Mn merupakan hasil tengah dari P0 dan P1 atau dapat dikatakan

$$M_n = \frac{P_n + P_{n+1}}{2}$$

Dengan itu dipastikan bahwa iterasi berikutnya akan menghasilkan n-1 titik. Pada iterasi selanjutnya juga dimasukkan titik ujung ke dalam leftControlPoints dan rightControlPoints dengan cara yang sudah disebut sebelumnya. Qn, Rn, Sn dan seterusnya melakukan yang sama hingga panjang array berukuran satu. Dengan ini dipastikan bahwa leftControlPoints dan rightControlPoints berukuran N titik kontrol (jika N = 3 maka pada pemanggilan rekursif selanjutnya leftControlPoints dan rightControlPoints berukuran 3 titik, sama untuk N=3,4,5, ….).

Setelah mendapatkan titik kontrol maka perlakuan rekursi sama dengan sebelumnya pada tiga titik kontrol. Dengan semua itu kami dapat menulis ulang cara pengerjaan algoritma dengan N>= 3 titik:

1. Pertama program menerima N>=3 kontrol kurva bézier dan juga banyak iterasi yang diinginkan

2. Program menginisialisasikan array kosong bezierPoints yang akan menggambarkan kurva bézier
3. Program memanggil fungsi rekursif bezierDnC yang pada kasus basis yaitu pada saat iterasi = 0 akan membuat sebuah bezier linier dengan cara memasukkan titik ujung dari kontrol point, dalam kasus ini pada P0, P1, … Pn maka yang dimasukkan ke dalam bezierPoints adaa P0 dan Pn dimana N adalah banyaknya titik kontrol.
4. Jika bukan kasus basis maka program akan menginisialisasi leftControlPoints dan rightControlPoints. Kemudian algoritma akan masuk ke while loop hingga panjang controlPoints masih lebih besar dari satu. Disini ujung kiri controlPoints dimasukkan ke leftControlPoints dan ujung kanan controlPoints dimasukkan ke rightControlPoints. Setelah itu dinisiliasisasikan newControlPoints dan dicari titik-titiknya dengan cara menghitung titik tengah untuk semua Pn dan Pn+1. Proses ini dilakukan hingga panjang controlPoints sama dengan satu. Titik saat panjang sama dengan satu pun dimasukkan ke dalam leftControlPoints dan rightControlPoints. Kemudian dipanggil kembali fungsi rekursif bezierDnC untuk sisi kiri dengan leftControlPoints dan sisi kanan dengan rightControlPoints dengan input bezierPoints dan juga iterasi dikurang 1.
5. Setelah perhitungan titik selesai maka bezierDnC mengeluarkan bezierPoints yang sudah merepresentasikan kurva bézier sesuai dengan iterasi yang diinginkan, program pun akan menampilkan kurva dengan plot.

Dengan itu dapat dilihat bahwa:

1. Pada kasus basis yaitu iterasi = 0 maka titik ujung P0 dan Pn akan dimasukkan ke dalam array bezierPoints
2. Pada kasus iterasi > 0 maka algoritma akan :
    a. Divide : Membagi dua titik kontrol dengan cara memasukkan ujung titik kontrol P0 ke dalam leftControlPoints dan Pn ke dalam rightControlPoints. Kemudian melakukan inisialisasi newControlPoints dan memasukkannya dengan titik tengah dari Pn dan Pn+1 dan sama seperti sebelumnya memasukkan ujung titik kontol ke dalam left dan right control points, iterasi ini dilakukan hingga ukuran controlPoints sama dengan satu dan memasukkan titik terakhir tersebut ke dalam left dan right control points.
    b. Conquer : Algoritma menyelesaikan semua sub persoalan menggunakan dua set titik kontrol tersebut untuk mencari kurva pada sisi kanan dan juga sisi kiri hingga mencapai kasus basis dimana titik P0 dan Pn dianggap sebagai titik dari kurva bézier

c. Combine : Titik-titik bezierPoints yang dihasilkan dari hasil tiap sub-persoalan pun digabungkan menjadi satu untuk mendapatkan kurva bézier secara keseluruhan.

Setelah semua penjelasan diatas kami dapat menyatakan kompleksitas waktu dari algoritma N=> titik ini dimana:

- Pada kasus basis hanya memasukkan titik P0 dan Pn ke dalam array bezierPoints maka memiliki kompleksitas *O(1)* dimana *O(1)* merepresentasikan waktu konstan untuk memasukkan kedua titik tersebut ke dalam array bezierPoints
- Pada kasus iterasi > 0 maka algoritma menghitung k titik tengah dan juga membagi menjadi dua set titik kontrol

Dengan itu dapat disimpulkan bahwa T(n) adalah :

$$T(n) = \begin{cases} 1 & , n = 0 \\ 2T(n-1) + k & , n > 1 \end{cases}$$

Melihat bahwa b =1 disini, maka kita tidak dapat menggunakan teorema master sehingga harus dilakukan penyelesaian fungsi rekursi dimana

$$\begin{aligned} T(n) &= 2T(n-1) + k \\ &= 2(2T(n-2) + k) + k = 2^2T(n-2) + 2k \\ &= 2(2^2T(n-3) + 2k) + k = 2^3T(n-3) + 2^2k \\ &\dots \\ &= 2^nT(0) + k(2^n - 1) = 2^n + k(2^n - 1) = O(2^n) \end{aligned}$$

algorithm.py

```
7   def bezierDnC(controlPoints, bezierPoints, iteration):
8       if iteration == 0:
9           bezierPoints.append(controlPoints[0])
10          bezierPoints.append(controlPoints[-1])
11          return
12      else:
13          leftControlPoints = []
14          rightControlPoints = []
15          while len(controlPoints) > 1:
16              leftControlPoints.append(controlPoints[0])
17              rightControlPoints.insert(0, controlPoints[-1])
18
19              newControlPoints = []
20              for i in range(len(controlPoints) - 1):
21                  newControlPoints.append(midPoint(controlPoints[i], controlPoints[i+1]))
22              controlPoints = newControlPoints
23
24          leftControlPoints.append(controlPoints[0])
25          rightControlPoints.insert(0, controlPoints[0])
26
27          bezierDnC(leftControlPoints, bezierPoints, iteration-1)
28          bezierDnC(rightControlPoints, bezierPoints, iteration-1)
```

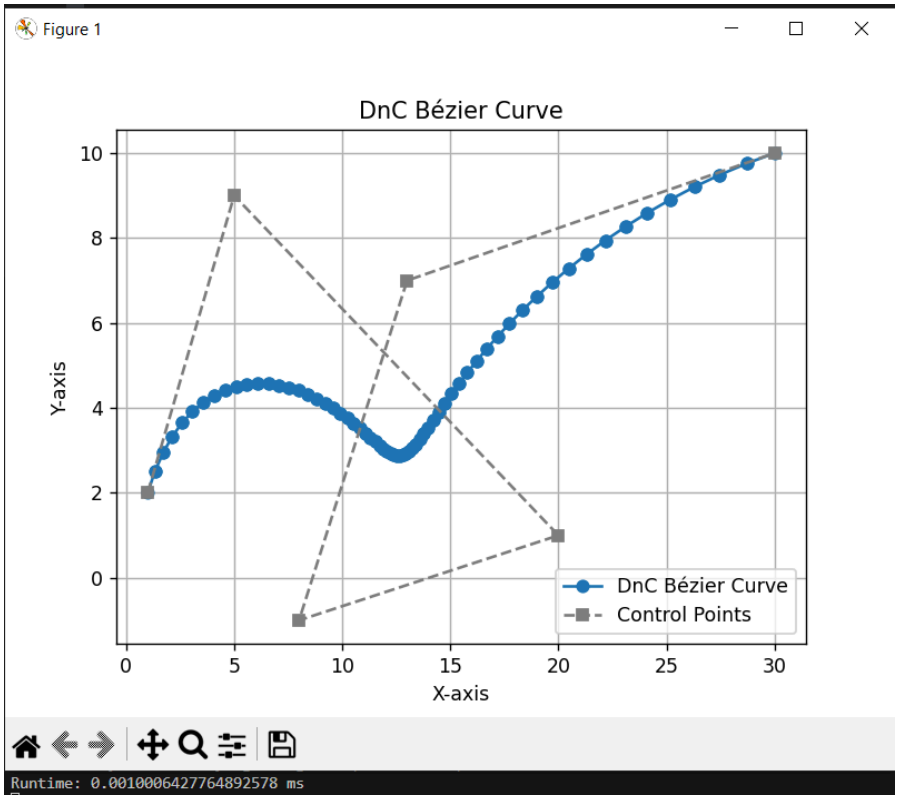
BAB 4
Pengujian

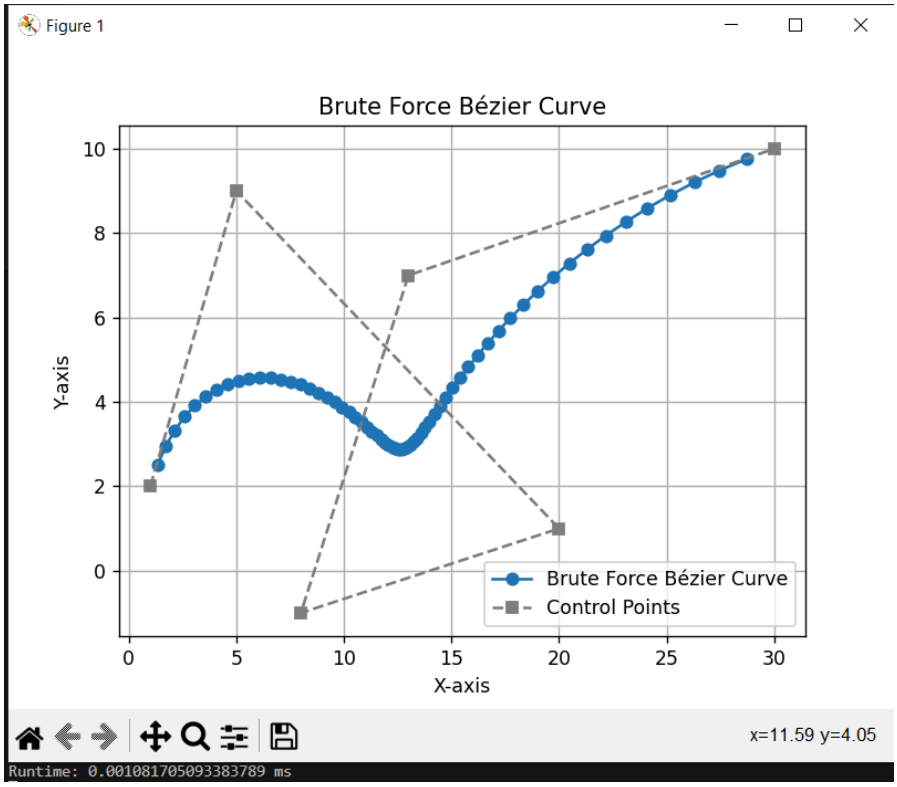
4.1 Pengujian dan Perbandingan Algoritma Divide and Conquer dan Brute Force Untuk Titik > 3

1. Test Case 1

   Pengujian ini menggunakan file tc1.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 6 titik kontrol dengan iterasi sebanyak 6 kali.

Titik Kontrol	(1,2), (5,9), (20,1), (8,1), (13,7), (30,10)
Iterasi	6

Divide and Conquer	

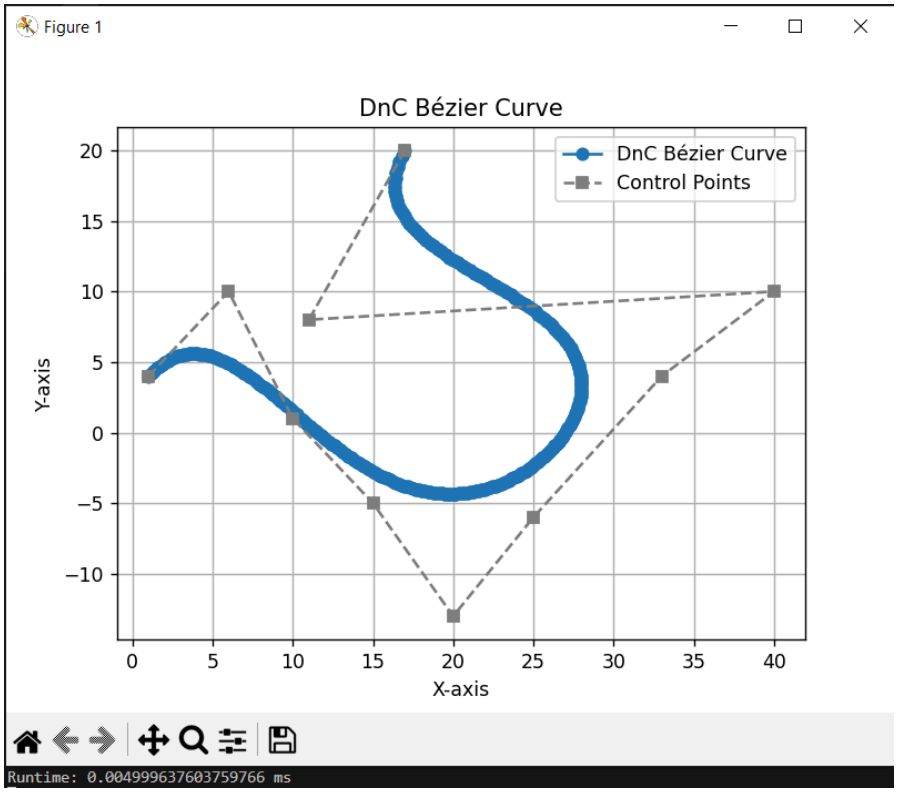
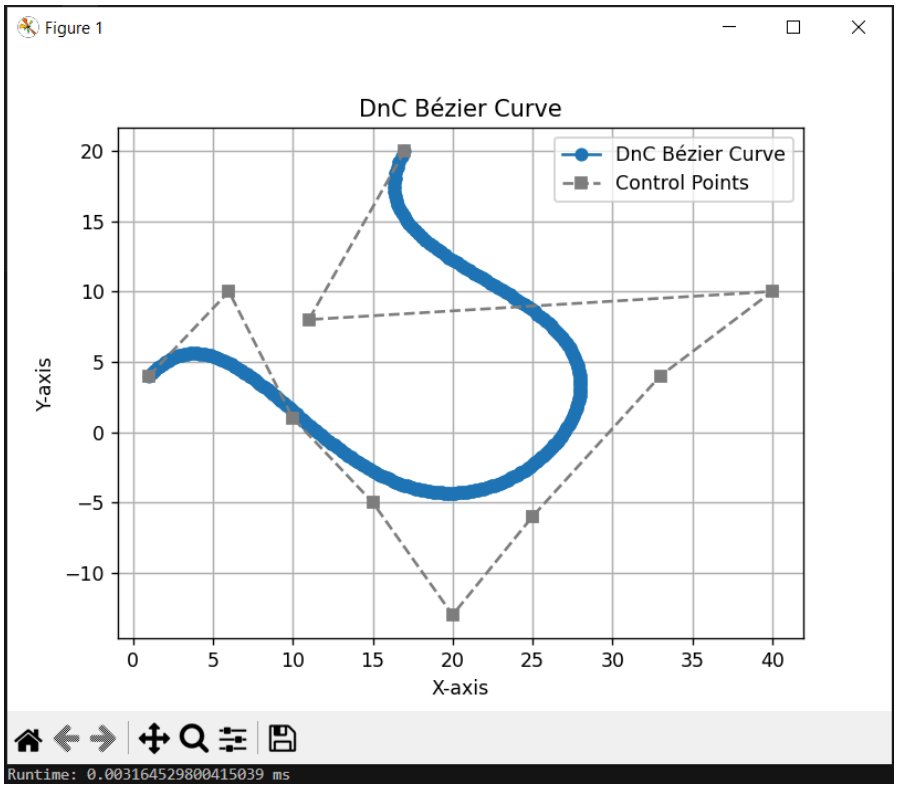
Brute Force	

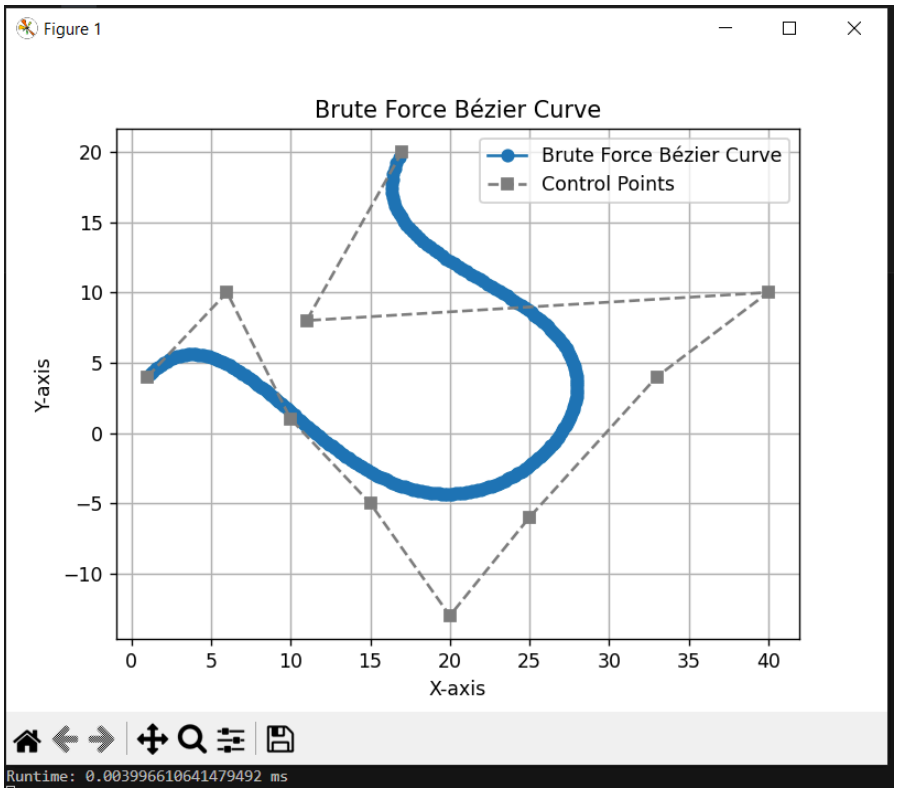
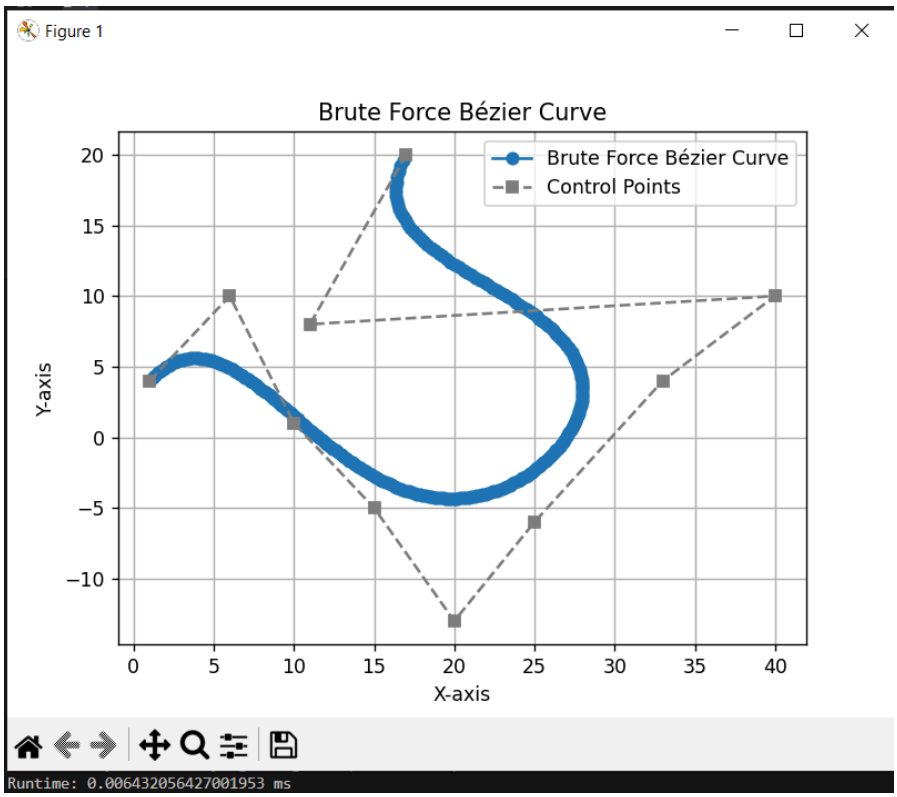
Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 0.0010006427764892578 ms, sedangkan *brute force* sebesar 0.001081705093383789 ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *divide and conquer* lebih cepat daripada *brute force*.

2. Test Case 2

   Pengujian ini menggunakan file tc2.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 10 titik kontrol dengan iterasi sebanyak 8 kali.

Titik Kontrol	(1,4), (6,10), (10,1), (15,-5), (20,-13), (25,-6), (33,4), (40,10), (11,8), (17,20)
Iterasi	8

Divide and Conquer	 

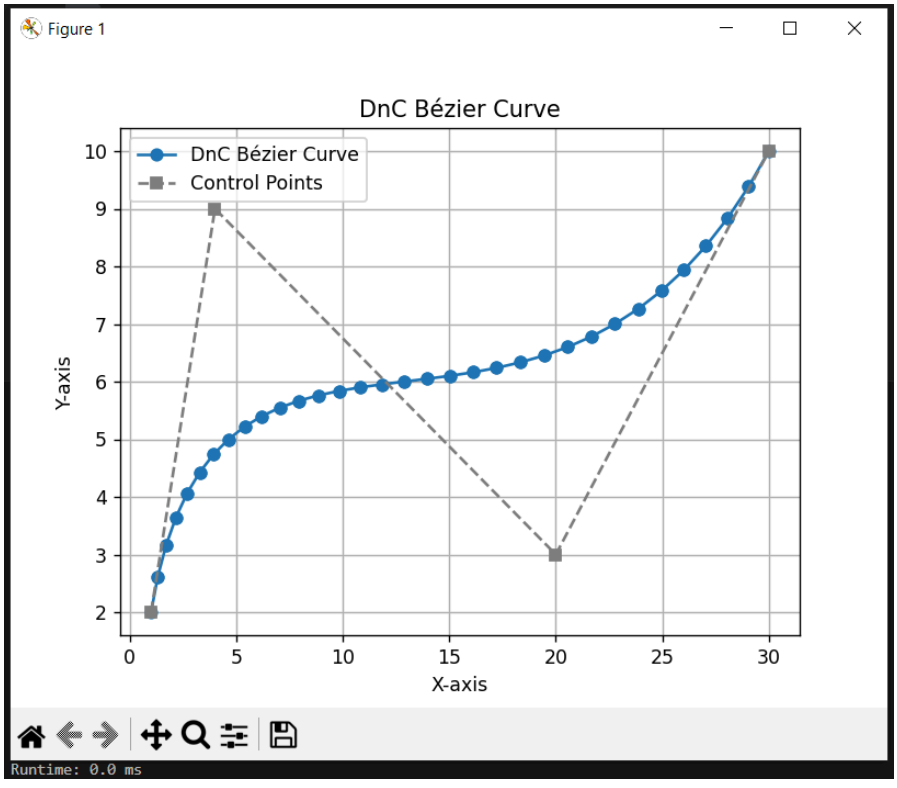
Brute Force	 

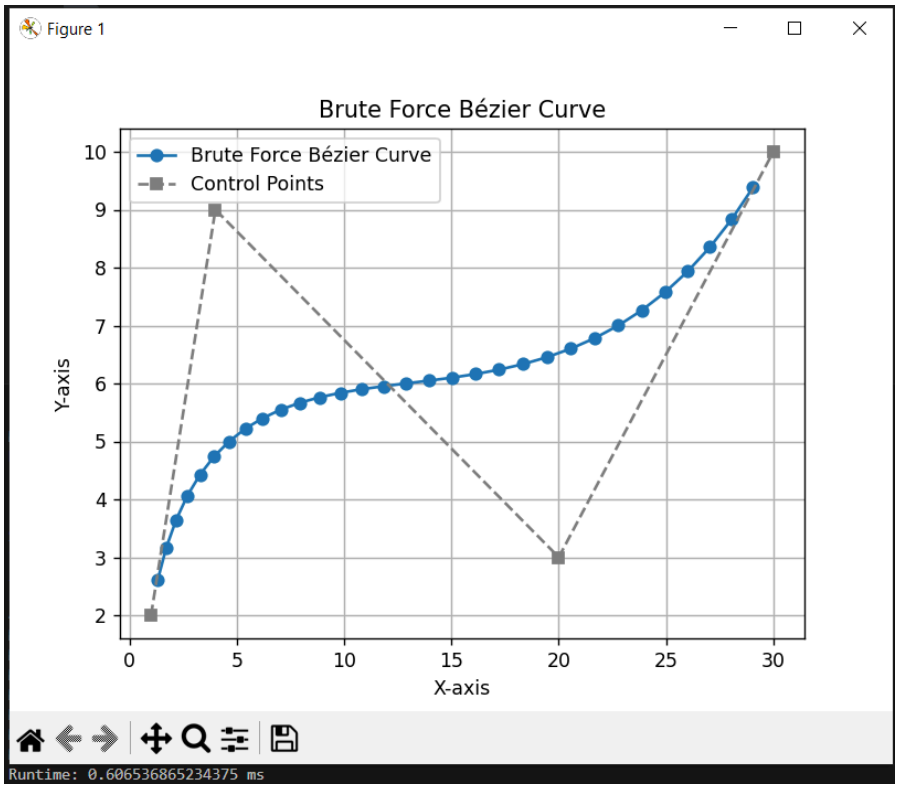
Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Pada percobaan pertama *test case 2* (gambar pertama *divide and conquer* dengan gambar pertama *brute force*), kami mendapatkan hasil algoritma *brute force* lebih cepat daripada *divide and conquer* dengan waktu eksekusi secara berturut-turut 0.003996610641479492 ms dan 0.004999637603759766 ms. Namun, pada percobaan kedua *test case 2* (gambar kedua *divide and conquer* dengan gambar kedua *brute force*), kami mendapatkan hasil algoritma *divide and conquer* lebih cepat daripada *brute force* dengan waktu eksekusi secara berturut-turut 0.003164529800415039 ms dan 0.006432056427001953 ms. Dari hasil tersebut dapat dilihat bahwa algoritma *brute force* bisa lebih cepat dari *divide and conquer* dalam suatu kondisi tertentu.

3. Test Case 3

   Pengujian ini menggunakan file tc3.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 4 titik kontrol dengan iterasi sebanyak 5 kali.

Titik Kontrol	(1,2), (4,9), (20,3), (30,10)
Iterasi	5

Divide and Conquer	

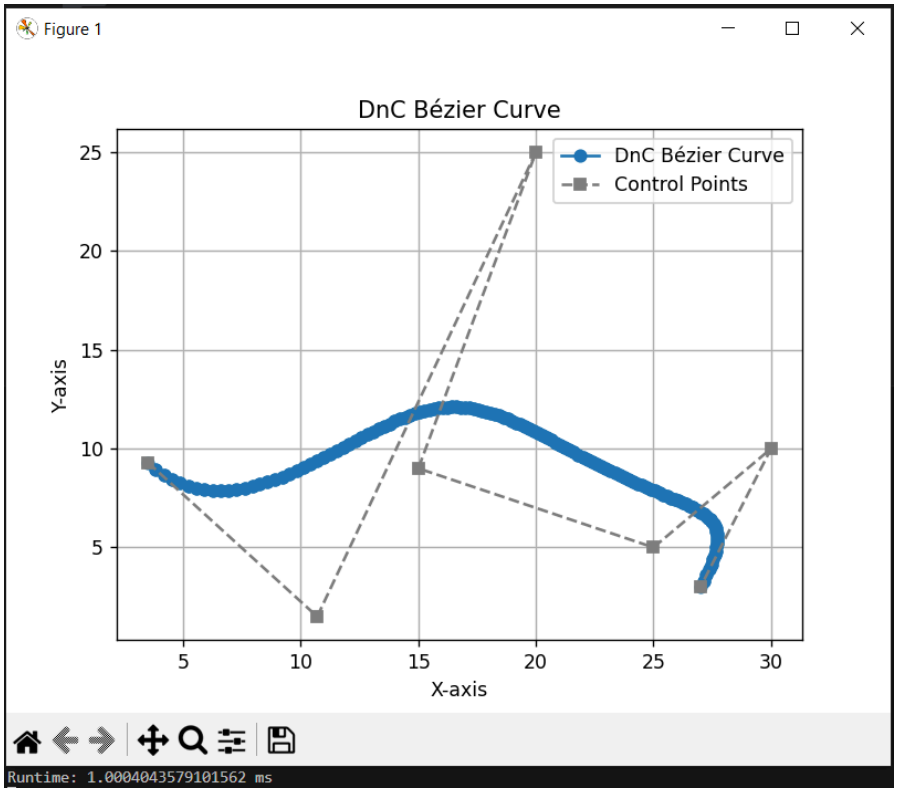
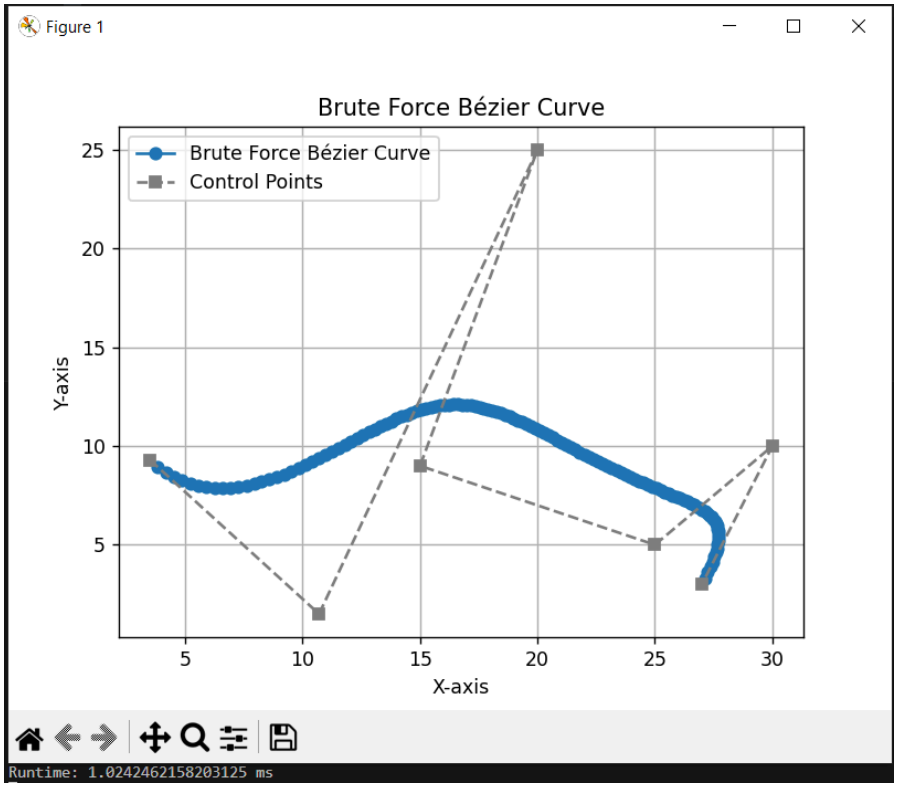
Brute Force	

Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 0.0 ms (0.0 ms karena program berjalan sangat cepat), sedangkan *brute force* sebesar 0.606536865234375 ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *divide and conquer* lebih cepat daripada *brute force*.

4. Test Case 4

   Pengujian ini menggunakan file tc4.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 7 titik kontrol dengan iterasi sebanyak 7 kali.

Titik Kontrol	(3.5,9.3), (10.7,1.5), (20, 25), (15,9), (25,5), (30,10), (27,3)
Iterasi	7

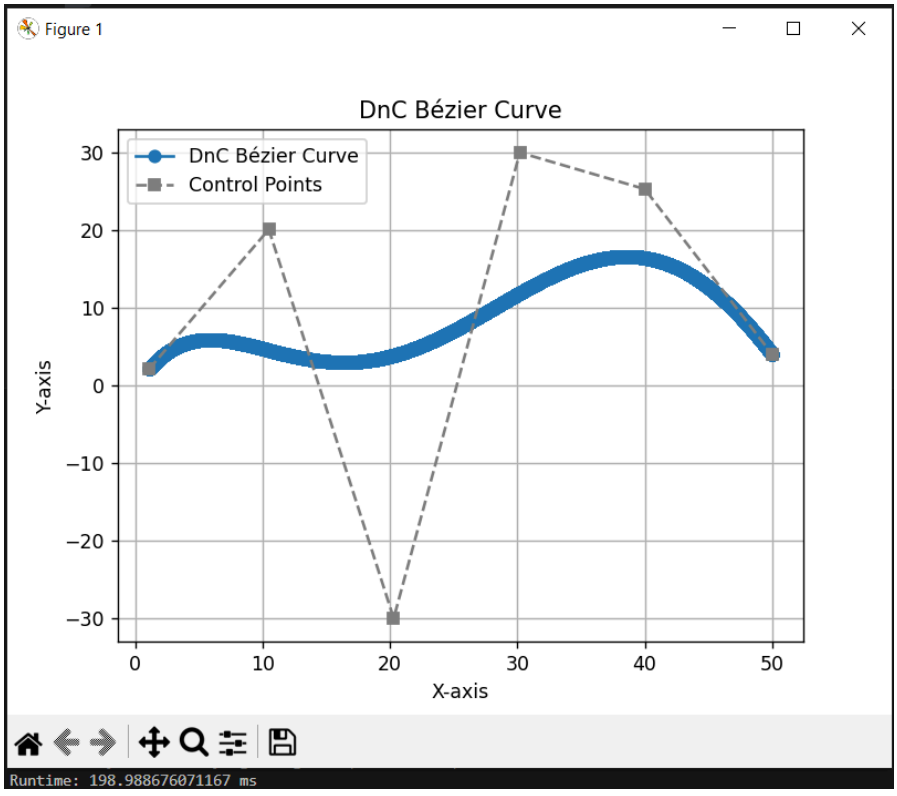
Divide and Conquer	
Brute Force	

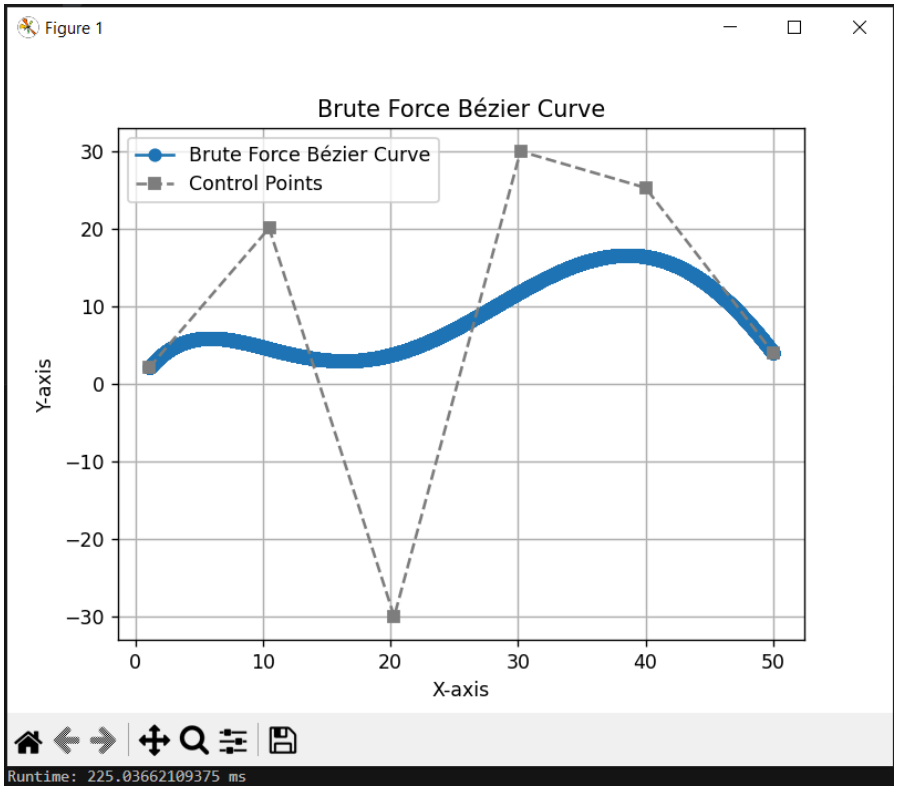
Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 1.0004043579101562 ms, sedangkan *brute force* sebesar Runtime: 1.0242462158203125 ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *divide and conquer* lebih cepat daripada *brute force*.

5. Test Case 5

   Pengujian ini menggunakan file tc5.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 6 titik kontrol dengan iterasi sebanyak 15 kali.

Titik Kontrol	(1.1, 2.1), (10.5, 20.1), (20.3, -30), (30.2, 30), (40, 25.3), (50, 4)
Iterasi	15

Divide and Conquer	

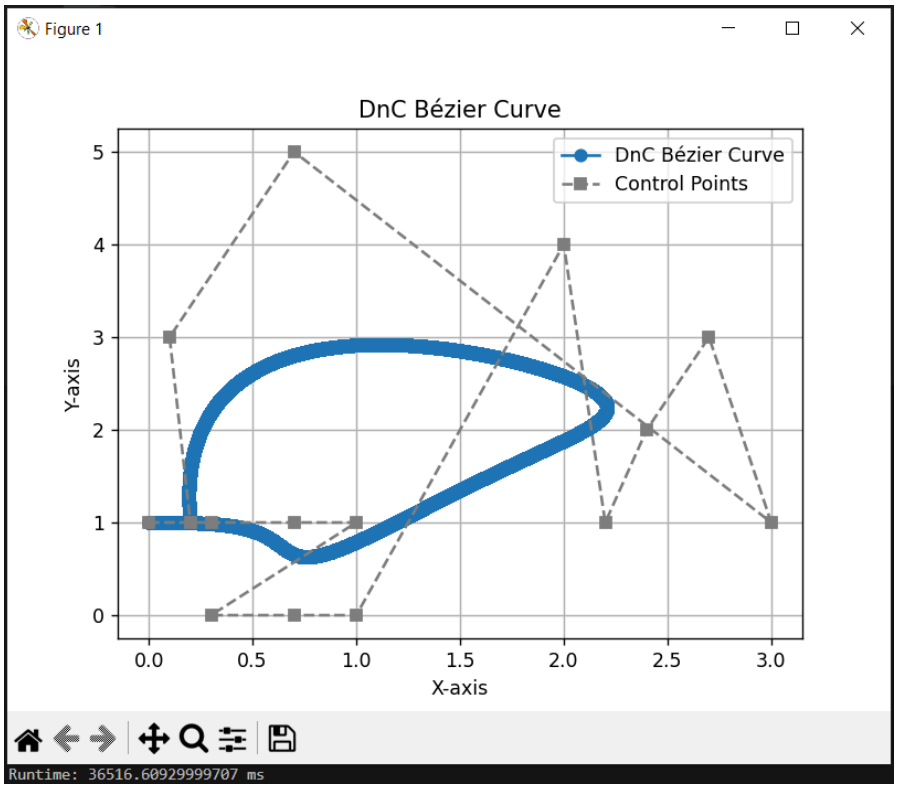
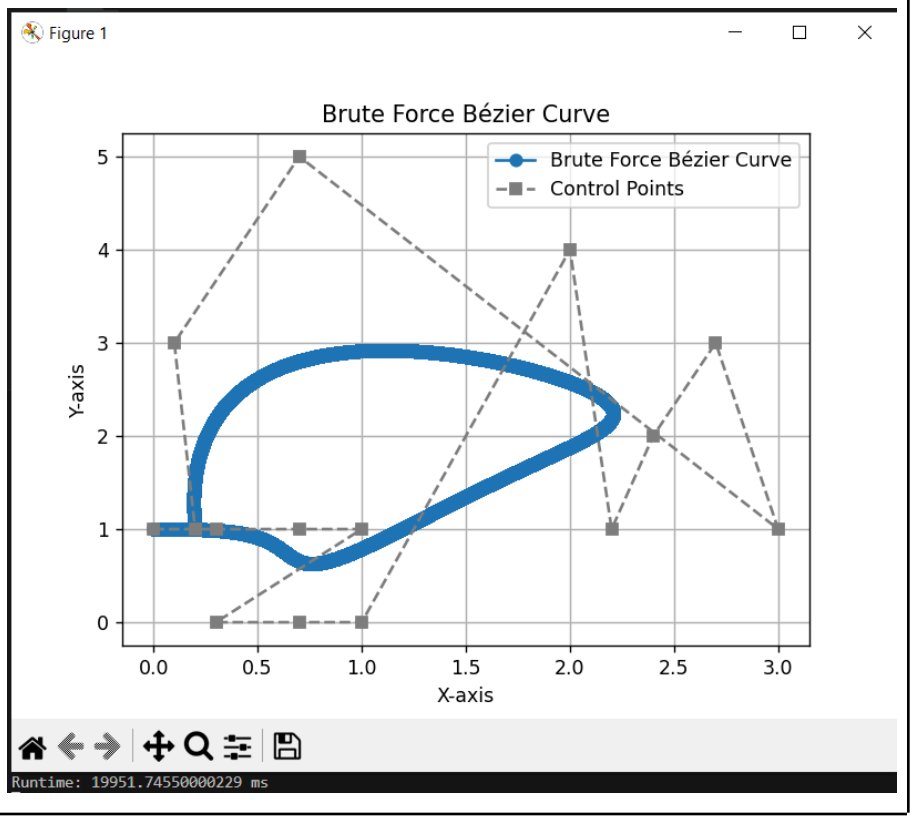
Brute Force	

Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 198.988676071167 ms, sedangkan *brute force* sebesar 225.03662109375 ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *divide and conquer* lebih cepat daripada *brute force*.

6. Test Case 6

   Pengujian ini menggunakan file tc6.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 15 titik kontrol dengan iterasi sebanyak 20 kali.

Titik Kontrol	(0, 1), (0.3, 1), (0.7, 1), (1, 1), (0.3, 0), (0.7, 0), (1, 0), (2, 4), (2.2, 1), (2.4, 2), (2.7, 3), (3, 1), (0.7, 5), (0.1, 3), (0.2, 1)
Iterasi	20

Divide and Conquer	
Brute Force	

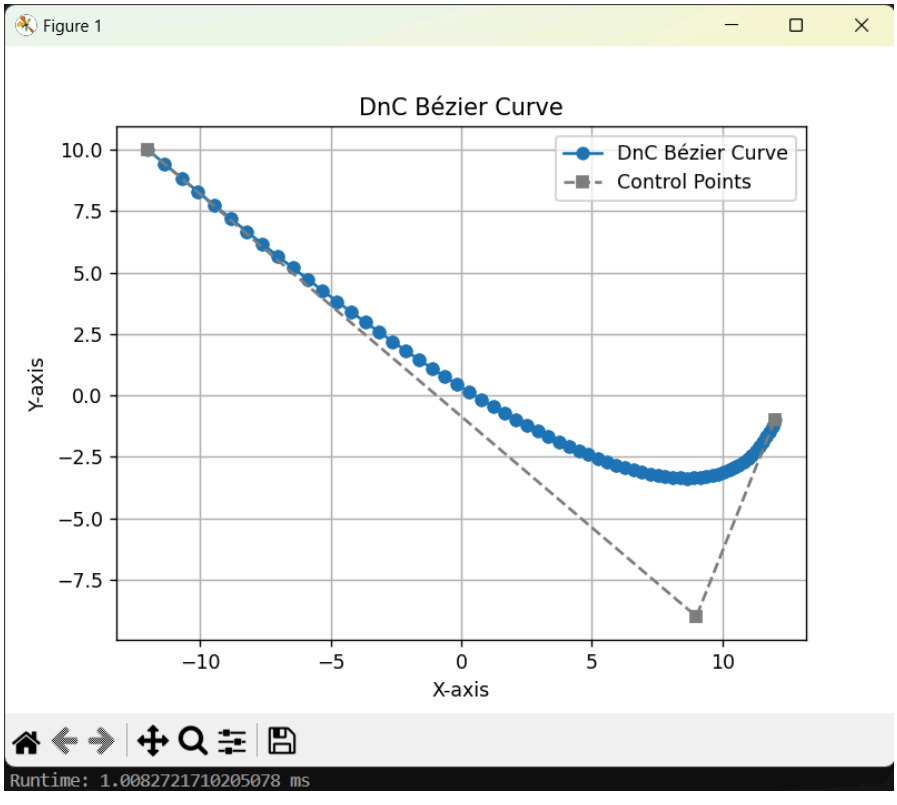
Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 36516.60929999707 ms, sedangkan *brute force* sebesar 19951.74550000229 ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *brute force* lebih cepat daripada *divide and conquer*.

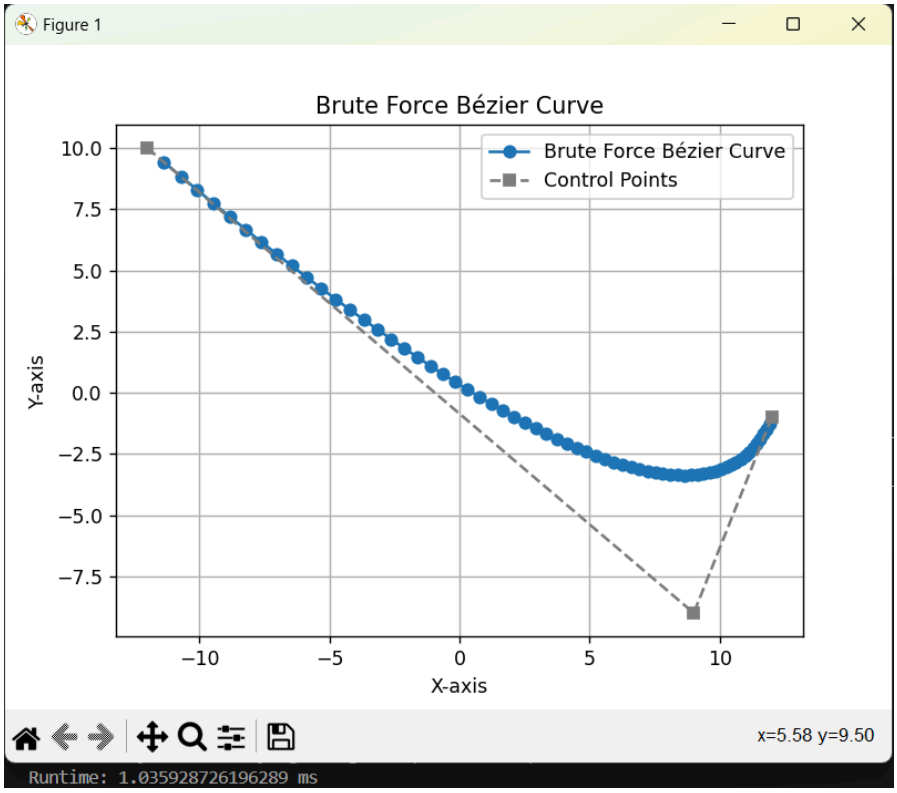
4.2 Pengujian dan Perbandingan Algoritma Divide and Conquer dan Brute Force Untuk Titik = 3

1. Test Case 7

   Pengujian ini menggunakan file tc7.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 3 titik kontrol dengan iterasi sebanyak 6 kali.

Titik Kontrol	(-12, 10), (9, -9), (12, -1)
Iterasi	6

Divide and Conquer	

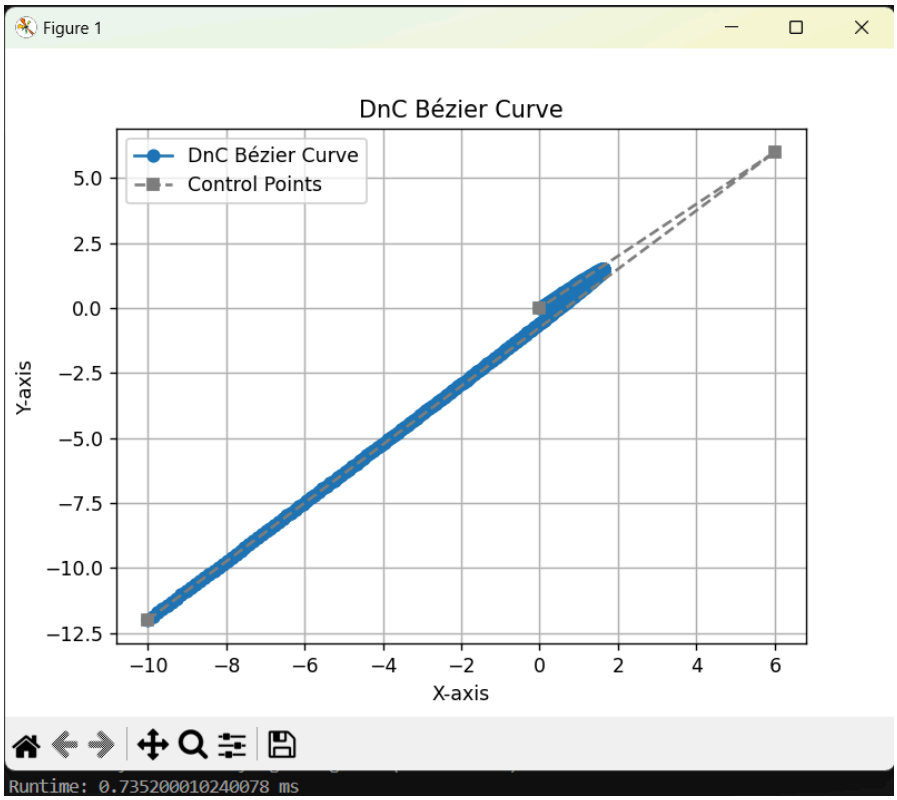
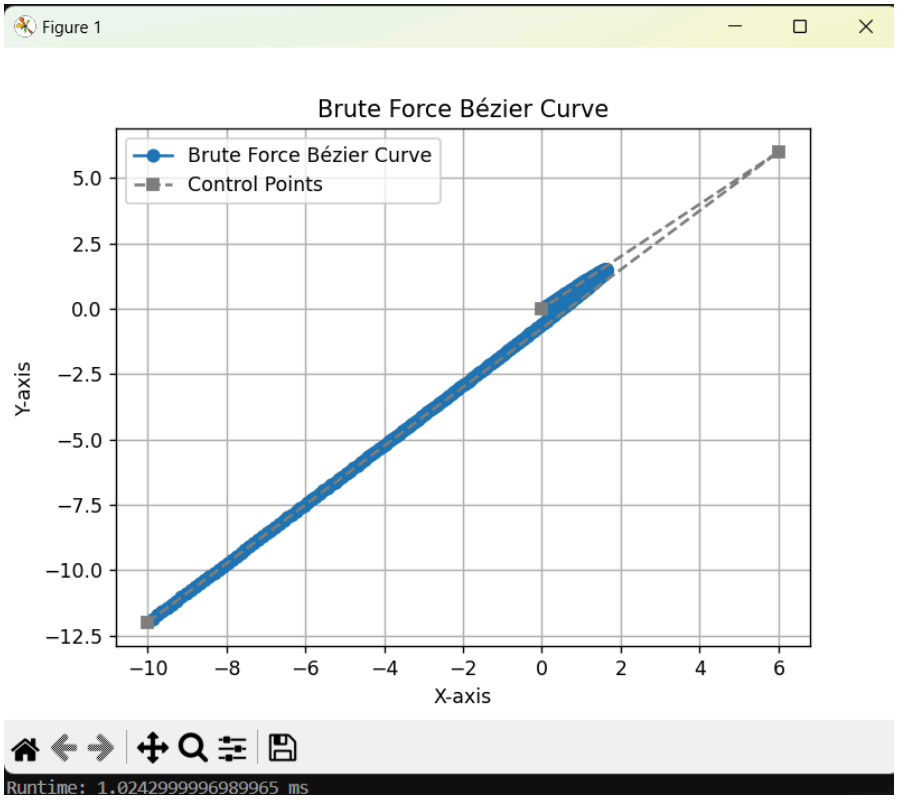
Brute Force	

Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 1.0082721710205078 ms, sedangkan *brute force* sebesar 1.035928726196289 ms ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *divide and conquer* lebih cepat daripada *brute force*..

2. Test Case 8

   Pengujian ini menggunakan file tc8.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 3 titik kontrol dengan iterasi sebanyak 8 kali.

Titik Kontrol	(-10, 12), (6, 6), (0, 0)
Iterasi	8

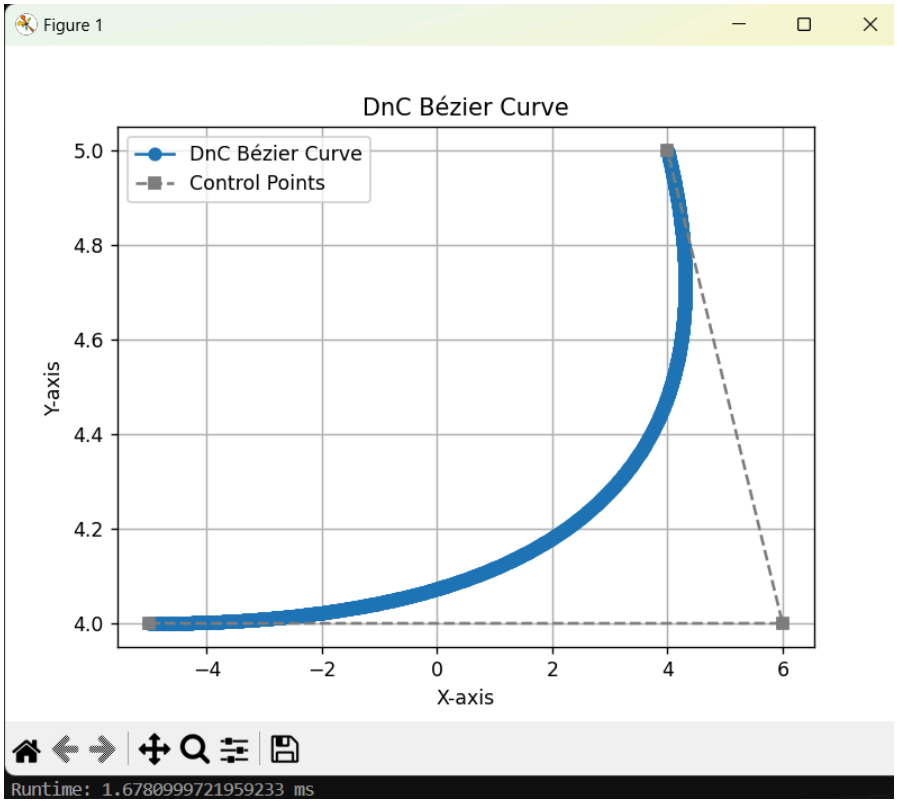
Divide and Conquer	
Brute Force	

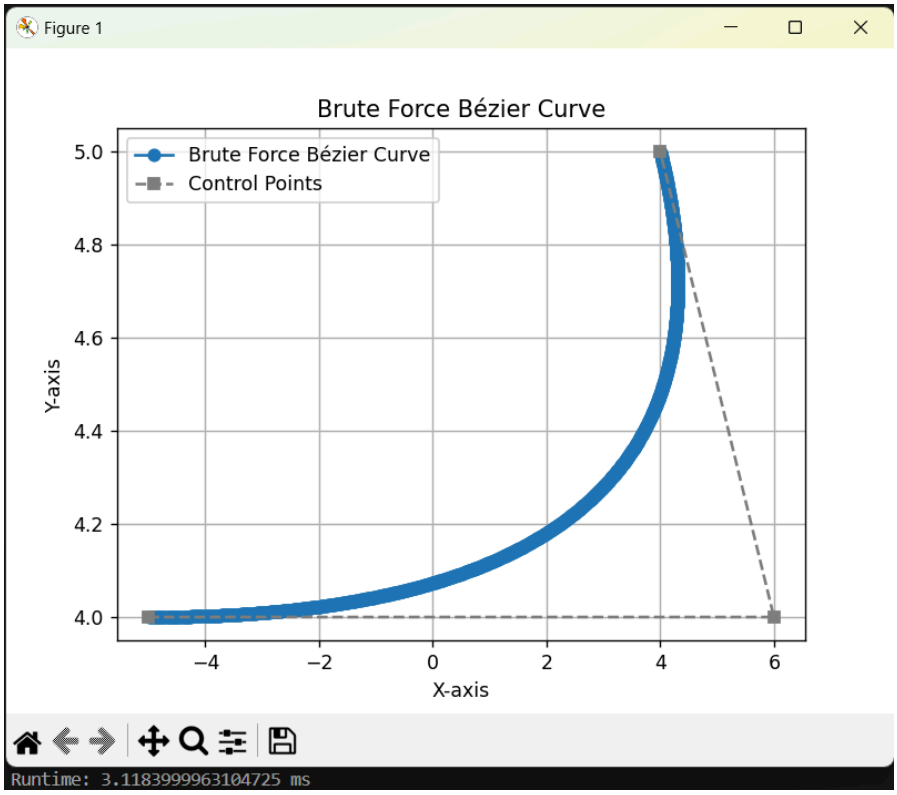
Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 0.735200010240078 ms, sedangkan *brute force* sebesar 1.0242999996989965 ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *divide and conquer* lebih cepat daripada *brute force*.

3. Test Case 9

   Pengujian ini menggunakan file tc9.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 3 titik kontrol dengan iterasi sebanyak 10 kali.

Titik Kontrol	(-5, 4), (6, 4), (4, 5)
Iterasi	10

Divide and Conquer	

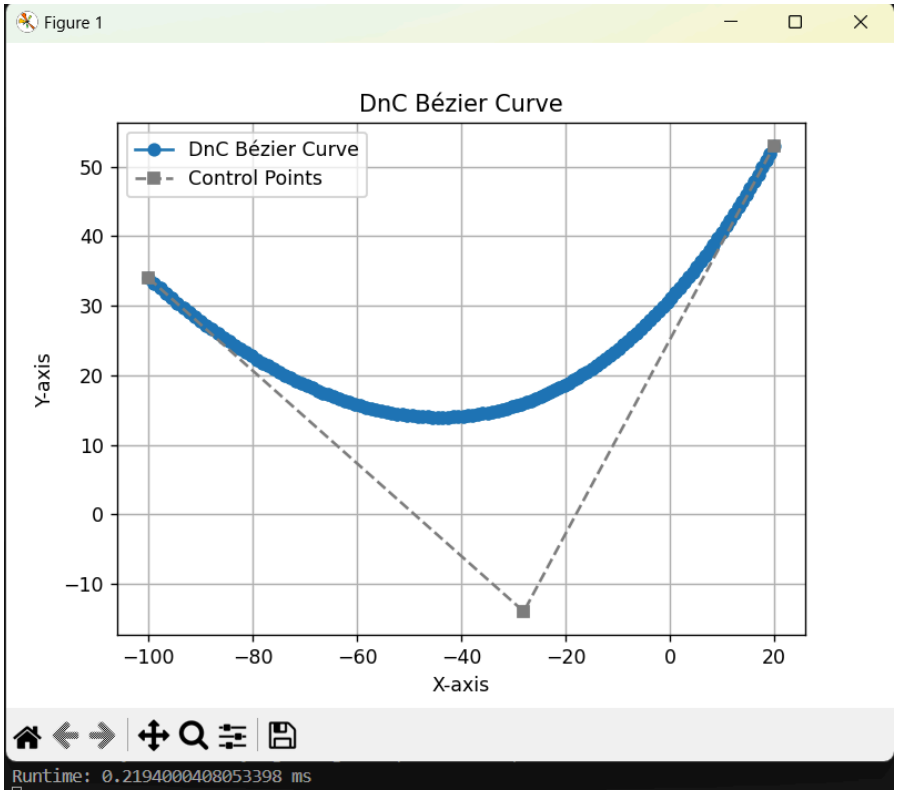
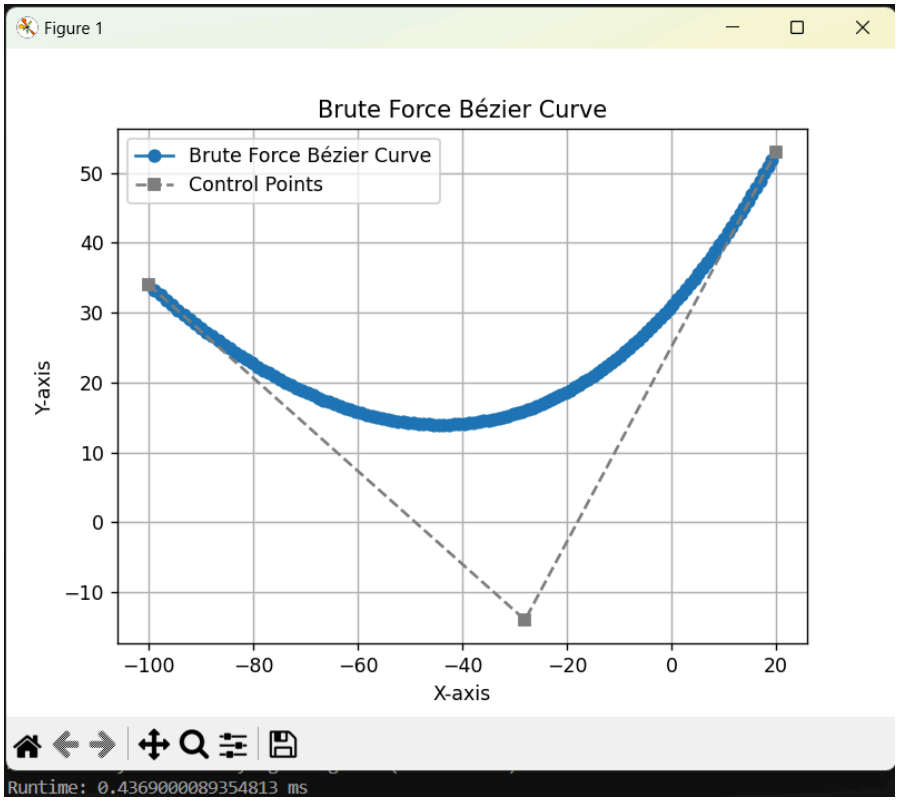
Brute Force	

Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 1.6780999721959233 ms, sedangkan *brute force* sebesar 3.1183999963104725 ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *divide and conquer* lebih cepat daripada *brute force*.

4. Test Case 10

   Pengujian ini menggunakan file tc10.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 6 titik kontrol dengan iterasi sebanyak 7 kali.

Titik Kontrol	(-100, 34), (-28, -14), (20, 53)
Iterasi	7

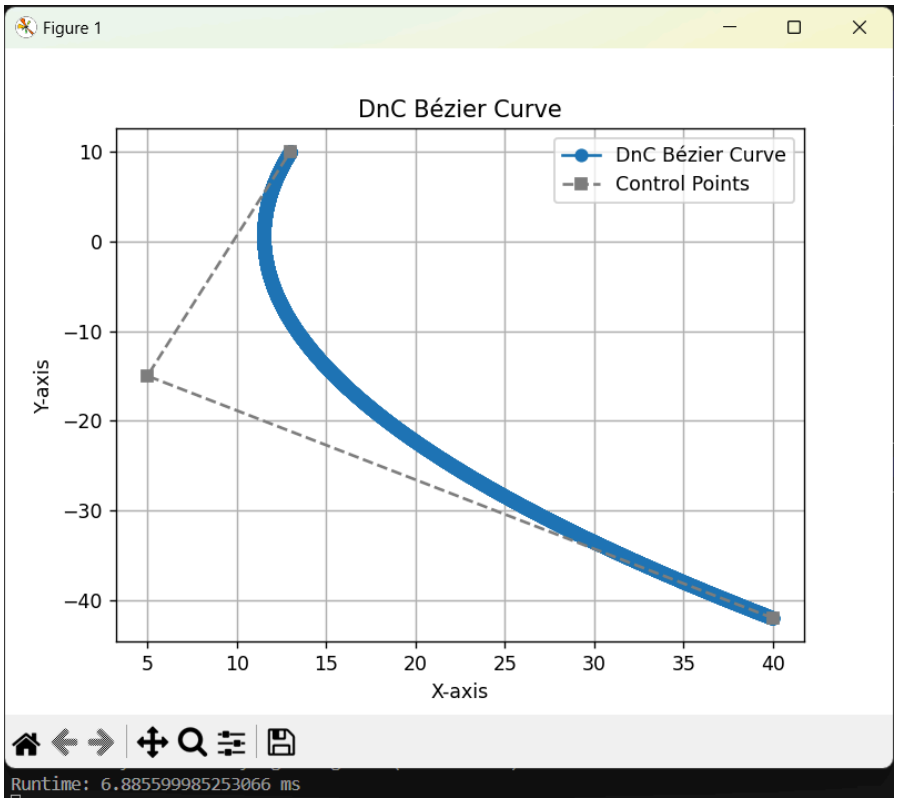
Divide and Conquer	
Brute Force	

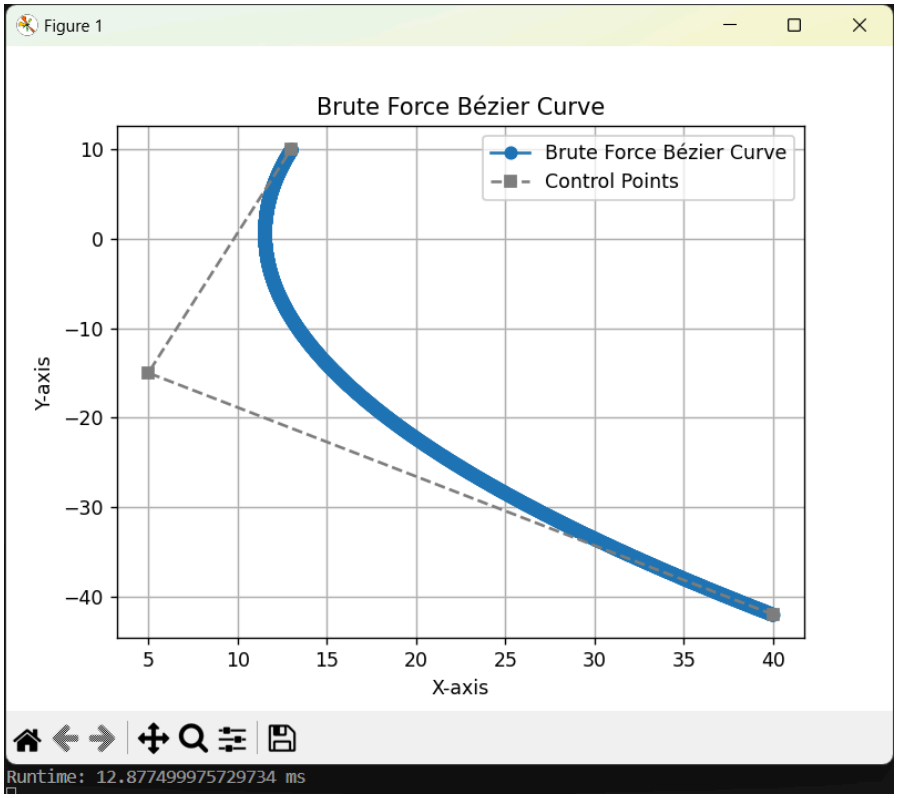
Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 0.2194000408053398 ms, sedangkan *brute force* sebesar 0.4369000089354813 ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *divide and conquer* lebih cepat daripada *brute force*.

5. Test Case 11

   Pengujian ini menggunakan file tc11.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 3 titik kontrol dengan iterasi sebanyak 12 kali.

Titik Kontrol	(40, -42), (5, -15), (13, 10)
Iterasi	12

Divide and Conquer	

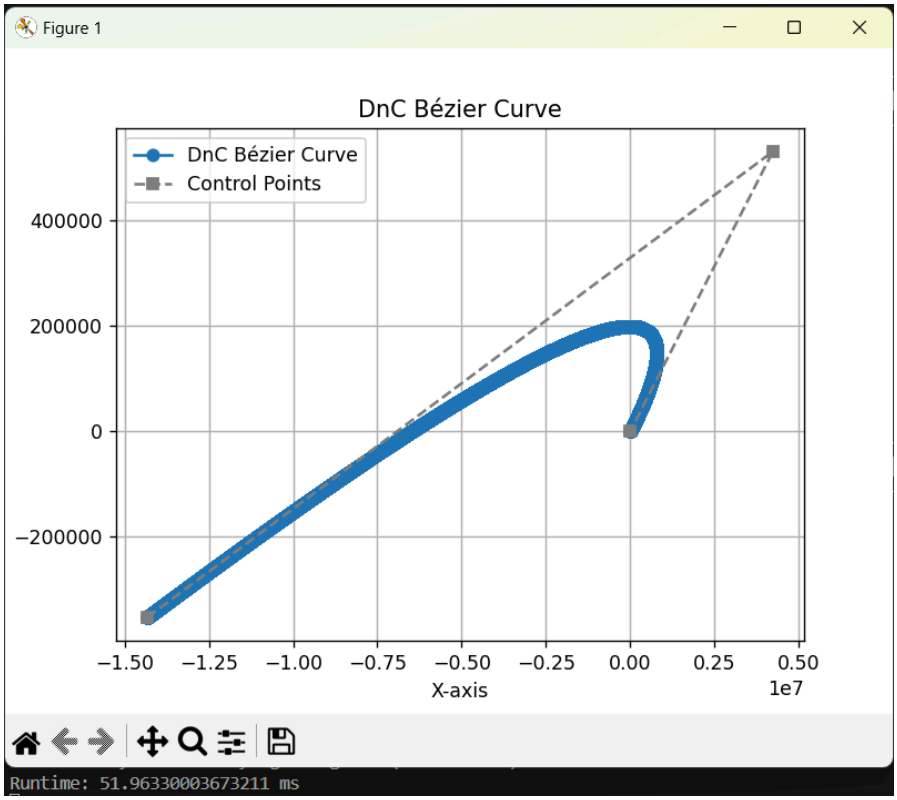
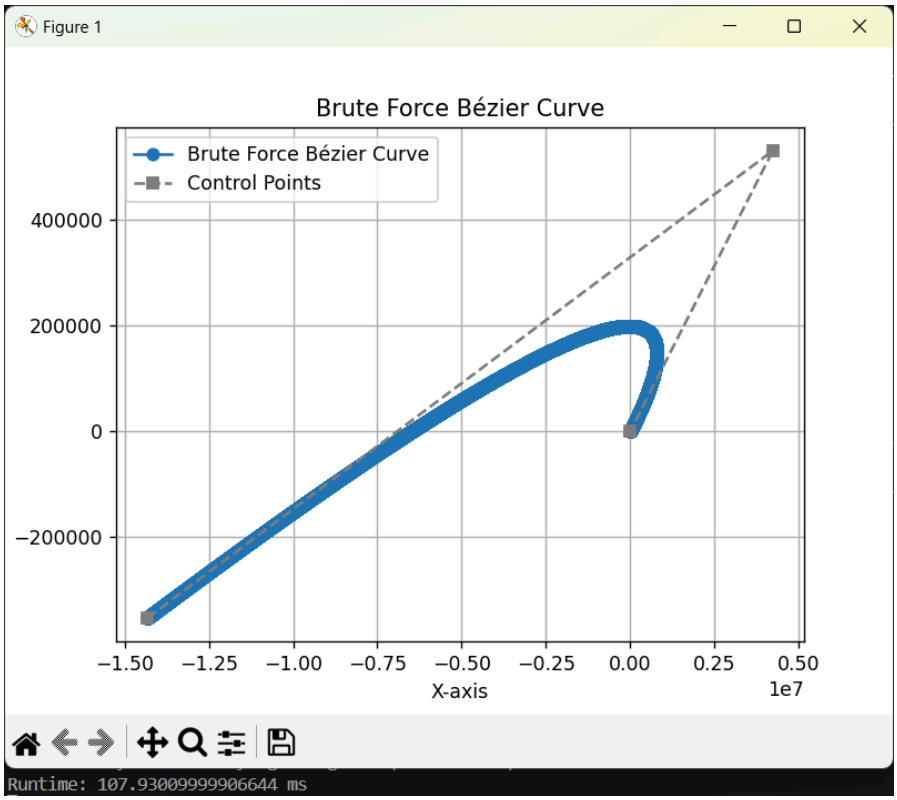
Brute Force	

Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 6.885599985253066 ms, sedangkan *brute force* sebesar 12.877499975729734 ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *divide and conquer* lebih cepat daripada *brute force*.

6. Test Case 6

   Pengujian ini menggunakan file tc12.txt yang berada pada folder test. Dalam pengujian ini, kami menggunakan 3 titik kontrol dengan iterasi sebanyak 15 kali.

Titik Kontrol	(-14324235, -353234), (4231412, 531421), (0, 1)
Iterasi	15

Divide and Conquer	
Brute Force	

Kurva yang didapat antara algoritma *divide and conquer* sama seperti *brute force*. Namun, waktu eksekusi algoritma berbeda. Algoritma *divide and conquer* memiliki waktu eksekusi sebesar 51.96330003673211 ms, sedangkan *brute force* sebesar 107.93009999906644 ms. Dari hasil tersebut, dapat dilihat bahwa algoritma *divide and conquer* lebih cepat daripada *brute force*.

BAB 5
Analisis Perbandingan Solusi Brute Force dan Divide and Conquer

5.1 Analisis Perbandingan untuk Titik Kontrol N = 3

Berdasarkan hasil pengujian tc7 hingga tc12 terlihat bahwa untuk titik kontrol tiga maka algoritma divide and conquer dapat menyelesaikan masalah dengan lebih cepat. Hal ini dapat dijelaskan ketika kita melihat kompleksitas algoritma untuk masing-masing cara seperti yang sudah di perlihatkan di dalam Bab3

- Kompleksitas Pada Brute Force

$$T(n) = 2^n . k = O(2^n)$$

- Kompleksitas Pada Divide and Conquer titik kontrol = 3

$$T(n) = 2^n + 3(2^n - 1) = O(2^n)$$

Terlihat bahwa untuk notasi big O, keduanya memiliki kompleksitas yang sama. Namun jika kita melihat cara pengimplementasian kedua algoritma terlihat bahwa algoritma brute force memerlukan perkalian dan juga perpangkatan sementara dalam divide and conquer hanya memerlukan operasi bagi, dikarenakan jumlah k yang rendah maka algoritma divide and conquer pun unggul dalam hal ini.

5.2 Analisis Perbandingan untuk Titik Kontrol N > 3

Berdasarkan hasil pengujian tc1-tc6 terlihat bahwa dalam beberapa kasus algoritma divide and conquer lebih efisien, tetapi dalam beberapa kasus pun algoritma brute force adalah pemenangnya. Hal tersebut dapat dilihat pada kasus Test Case 5. Pada *test case* tersebut, titik kontrol berjumlah 6 dengan iterasi sebesar 15, menghasilkan algoritma DnC sebagai algoritma yang lebih cepat. Namun, pada Test Case 6, dengan jumlah titik kontrol 15 dan iterasi 20, maka algoritma DnC kalah cepat dibandingkan brute force. Hal ini dapat disebabkan oleh kondisi laptop pada saat algoritma dijalankan. Kondisi laptop dapat mempengaruhi waktu eksekusi dari algoritma. Hal ini telah dicoba beberapa kali dengan menggunakan titik kontrol dan iterasi yang sama secara terus menerus. Didapatkan hasil bahwa baik algoritma *divide and conquer* maupun *brute force* dapat menjadi algoritma yang lebih cepat dalam satu waktu. Selain itu, setelah dilakukan percobaan berkali-kali (di luar *test case*), kami mendapatkan bahwa semakin banyak titik kontrol (sampai batas tertentu), algoritma *divide and conquer* akan menjadi lebih lambat daripada *brute force* dan semakin sedikit titik kontrol, algoritma *divide and conquer* akan lebih cepat daripada *brute force*. Tentu hal tersebut juga tetap dipengaruhi oleh jumlah iterasi yang dilakukan.

- Kompleksitas Pada Brute Force

$$T(n) = 2^n . k = O(2^n)$$

- Kompleksitas Pada Divide and Conquer titik kontrol > 3

$$T(n) = 2^n + k(2^n - 1) = O(2^n)$$

T(n) dari *divide and conquer* sekarang sedikit berbeda dimana sekarang k = 3 diubah menjadi k bebas yang merupakan banyak titik kontrol. Jika k semakin besar maka perkalian dalam $k(2^n - 1)$ menjadi semakin besar, hal ini akan menyebabkan *divide and conquer* memerlukan lebih banyak langkah dibandingkan dengan *brute force*.

BAB 6
Kesimpulan

Dalam menyelesaikan suatu permasalahan, terdapat berbagai cara untuk menyelesaikannya. Pada tugas kecil ini, permasalahan tersebut diselesaikan menggunakan dua algoritma, yaitu algoritma *brute force* dan algoritma *divide and conquer*. Kedua algoritma tersebut dapat digunakan untuk membentuk kurva bezier dengan titik kontrol sebanyak N. Namun, meskipun hasil yang didapatkan sama, waktu eksekusi kedua algoritma tersebut berbeda. Oleh karena itu, pemilihan algoritma dalam penyelesaian masalah sangat penting untuk mendapatkan metode paling baik.

Pada tugas kecil ini, kami melakukan pengujian kepada kedua algoritma tersebut. Pengujian tersebut dilakukan dengan membandingkan waktu eksekusi setiap algoritma dengan jumlah titik dan iterasi yang sama. Hasil yang kami dapatkan adalah algoritma *divide and conquer* lebih cepat daripada algoritma *brute force*. Meskipun begitu, algoritma *brute force* juga terkadang menjadi algoritma yang lebih cepat daripada *divide and conquer*. Hal ini dipengaruhi oleh jumlah titik kontrol yang digunakan dan juga jumlah iterasi yang digunakan. Selain itu, kondisi perangkat yang digunakan juga dapat mempengaruhi kecepatan kedua algoritma tersebut.

Daftar Pustaka

1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf
2. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf
3. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf
4. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf

Lampiran

Poin	Ya	Tidak
1. Program berhasil dijalankan	☑	☐
2. Program dapat melakukan 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	☑	☐

Link Repository : https://github.com/MRafliRasyiidin/Tucil2_13522061_13522088