

#Tucil2LetsGo!

LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA
Membangun Kurva Bézier dengan Algoritma Titik Tengah
berbasis *Divide and Conquer*



Disusun Oleh:

13522069 Nabila Shikoofa Muida
13522104 Diana Tri Handayani

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

DAFTAR ISI.....	1
DAFTAR GAMBAR.....	3
DAFTAR TABEL.....	5
BAB I	
DESKRIPSI MASALAH.....	6
1.1. Deskripsi Persoalan.....	6
1.2. Spesifikasi Tugas.....	8
1.2.1. Input.....	9
1.2.2. Output.....	9
1.2.3. Bonus.....	9
BAB II	
LANDASAN TEORI.....	10
2.1. Kurva Bézier.....	10
2.2. Algoritma Titik Tengah.....	10
2.3. Algoritma Divide and Conquer.....	11
2.4. Algoritma Brute Force.....	13
BAB III	
IMPLEMENTASI PROGRAM.....	15
3.1. Kurva Bézier dengan Memanfaatkan Algoritma Divide and Conquer (Diana).....	15
3.2. Kurva Bézier dengan Memanfaatkan Algoritma Brute Force.....	17
3.3. Sistematika File.....	18
3.4. Struktur Data.....	19
3.5. Source Code Program.....	20
3.5.1. main.py.....	20
3.5.2. input.py.....	21
3.5.3. output.py.....	22
3.5.4. visualizer.py.....	23
3.5.5. quadraticBezierBF.py.....	25
3.5.6. quadraticBeziersDnC.py.....	26
3.5.7. unitTestMultipoint.py.....	27
3.5.8. multipointBeziersDnC.py.....	30
BAB IV	
PENGUJIAN DAN ANALISIS.....	31
4.1. Spesifikasi Komputer Uji Coba.....	31
4.2. Uji Coba Kurva Bézier Kuadratik.....	31

4.2.1. Pengujian I (2 Iterasi).....	32
4.2.2. Pengujian II (3 Iterasi).....	33
4.2.3. Pengujian III (5 Iterasi).....	35
4.2.4. Pengujian IV (7 Iterasi).....	36
4.2.5. Pengujian V (10 Iterasi).....	38
4.2.6. Pengujian VI (13 Iterasi).....	39
4.3. Uji Coba Kurva Bézier Berorde-N.....	41
4.3.1. Pengujian Berorde-4.....	41
4.3.2. Pengujian Berorde-5.....	42
4.3.3. Pengujian Berorde-6.....	43
4.3.4. Pengujian Berorde-9.....	44
4.3.5. Pengujian Berorde-15.....	45
4.3.6. Pengujian Berorde-20.....	46
4.4. Uji Coba Lainnya.....	47
4.5. Analisis Hasil Uji Coba.....	48
BAB V	
PENUTUP.....	52
5.1. Kesimpulan.....	52
5.2. Saran.....	52
DAFTAR PUSTAKA.....	53
LAMPIRAN.....	54
Lampiran 1 Link Repository.....	54
Lampiran 2 Tabel Checklist Poin.....	54

DAFTAR GAMBAR

Gambar 1.1. Kurva Bézier Kubik

Gambar 1.2. Pembentukan Kurva Bézier Kuadratik

Gambar 2.1. Ilustrasi Algoritma *Divide and Conquer*

Gambar 3.1. Ilustrasi Pembuatan Kurva Bézier dengan Algoritma *Divide and Conquer*

Gambar 3.2. *Source Code* “datatype.py”

Gambar 3.3. *Source Code* “main.py”

Gambar 3.4. *Source Code* “input.py”

Gambar 3.5. *Source Code* “output.py”

Gambar 3.6. *Source Code* “visualizer.py”

Gambar 3.7. *Source Code Brute Force di* “visualizer.py”

Gambar 3.8. *Source Code* “quadraticBezeirBF.py”

Gambar 3.9. *Source Code* “quadraticBezeirDnC.py”

Gambar 3.10. *Source Code* “unitTestMultipoint.py” (1)

Gambar 3.11. *Source Code* “unitTestMultipoint.py” (2)

Gambar 3.12. *Source Code* “unitTestMultipoint.py” (3)

Gambar 3.13. *Source Code* “multipointBezeirDnC.py”

Gambar 4.1. Pengujian I dengan Algoritma *Brute Force*

Gambar 4.2. Pengujian I dengan Algoritma *Divide and Conquer*

Gambar 4.3. Input dan Output Pengujian I

Gambar 4.4. Pengujian II dengan Algoritma *Brute Force*

Gambar 4.5. Pengujian II dengan Algoritma *Divide and Conquer*

Gambar 4.6. Input dan Output Pengujian II

Gambar 4.7. Pengujian III dengan Algoritma *Brute Force*

Gambar 4.8. Pengujian III dengan Algoritma *Divide and Conquer*

Gambar 4.9. Input dan Output Pengujian III

Gambar 4.10. Pengujian IV dengan Algoritma *Brute Force*

Gambar 4.11. Pengujian IV dengan Algoritma *Divide and Conquer*

Gambar 4.12. Input dan Output Pengujian IV

Gambar 4.13. Pengujian V dengan Algoritma *Brute Force*

Gambar 4.14. Pengujian V dengan Algoritma *Divide and Conquer*

Gambar 4.15. Input dan Output Pengujian V

Gambar 4.16. Pengujian VI dengan Algoritma *Brute Force*

Gambar 4.17. Pengujian VI dengan Algoritma *Divide and Conquer*

Gambar 4.18. Input dan Output Pengujian VI

Gambar 4.19. Input Pengujian Berorde-4

Gambar 4.20. Visualisasi Pengujian Berorde-4

Gambar 4.21. Input Pengujian Berorde-5

Gambar 4.22. Visualisasi Pengujian Berorde-5

Gambar 4.23. Input Pengujian Berorde-6

Gambar 4.24. Visualisasi Pengujian Berorde-6

Gambar 4.25. Input Pengujian Berorde-9

Gambar 4.26. Visualisasi Pengujian Berorde-9

Gambar 4.27. Input Pengujian Berorde-15

Gambar 4.28. Visualisasi Pengujian Berorde-15

Gambar 4.29. Input Pengujian Berorde-20

Gambar 4.30. Visualisasi Pengujian Berorde-20

Gambar 4.31. Grafik Waktu Eksekusi Algoritma dengan Iterasi 2-15

Gambar 4.32. Grafik Waktu Eksekusi Algoritma dengan Iterasi 2-25

DAFTAR TABEL

Tabel 4.1. Spesifikasi Komputer Uji Coba Nomor Ganjil

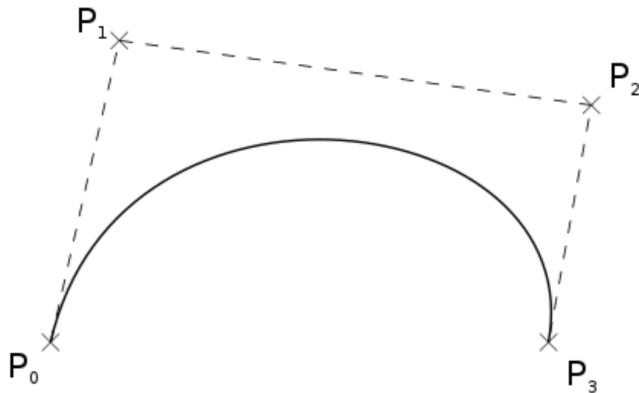
Tabel 4.2. Spesifikasi Komputer Uji Coba Nomor Genap

Tabel 4.3. Tabel Uji Coba Kurva Bézier Kuadratik dengan 2-25 Iterasi

BAB I

DESKRIPSI MASALAH

1.1. Deskripsi Persoalan



Gambar 1.1. Kurva Bézier Kubik

(Sumber: https://id.wikipedia.org/wiki/Kurva_B%C3%A9zier)

Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol, yang menentukan bentuk dan arah kurva. Cara membuatnya cukup mudah, yaitu dengan menentukan titik-titik kontrol dan menghubungkannya dengan kurva. Kurva Bézier memiliki banyak kegunaan dalam kehidupan nyata, seperti pen tool, animasi yang halus dan realistik, membuat desain produk yang kompleks dan presisi, dan membuat font yang indah dan unik. Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan.

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P_0 sampai P_n , dengan n disebut order ($n = 1$ untuk linier, $n = 2$ untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) umumnya tidak terletak pada kurva. Pada gambar 1 diatas, titik kontrol pertama adalah P_0 , sedangkan titik kontrol terakhir adalah P_3 . Titik kontrol P_1 dan P_2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P_0 dan P_1 yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan kurva Bézier linier. Misalkan terdapat sebuah titik Q_0 yang berada pada garis yang dibentuk oleh P_0 dan P_1 , maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

$$Q_0 = B(t) = (1 - t)P_0 + t P_1, \quad t \in [0, 1]$$

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh $B(t)$ dari P_0 ke P_1 . Misalnya ketika $t = 0.25$, maka $B(t)$ adalah seperempat jalan dari titik P_0 ke P_1 . Sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan $B(t)$ membentuk sebuah garis lurus dari P_0 ke P_1 .

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P_2 , dengan P_0 ke P_2 sebagai titik kontrol awal dan akhir, dan P_1 menjadi titik kontrol antara. Dengan menyatakan titik Q_1 terletak diantara garis yang menghubungkan P_1 dan P_2 , dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q_0 berada, maka dapat dinyatakan sebuah titik baru, R_0 yang berada diantara garis yang menghubungkan Q_0 dan Q_1 yang bergerak membentuk **kurva Bézier kuadratik** terhadap titik P_0 dan P_2 . Berikut adalah uraian persamaannya.

$$Q_0 = B(t) = (1 - t)P_0 + t P_1, \quad t \in [0, 1]$$

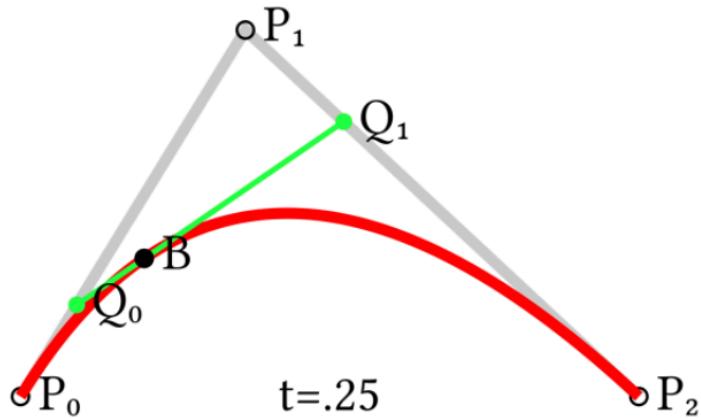
$$Q_1 = B(t) = (1 - t)P_1 + t P_2, \quad t \in [0, 1]$$

$$R_0 = B(t) = (1 - t)Q_0 + t Q_1, \quad t \in [0, 1]$$

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

$$R_0 = B(t) = (1 - t)^2 P_0 + 2(1 - t)t P_1 + t^2 P_2, \quad t \in [0, 1]$$

Berikut adalah ilustrasi dari kasus diatas.



Gambar 1.2. Pembentukan Kurva Bézier Kuadratik.

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

Proses ini dapat juga diaplikasikan untuk jumlah titik yang lebih dari tiga, misalnya empat titik akan menghasilkan **kurva Bézier kubik**, lima titik akan menghasilkan **kurva Bézier kuartik**, dan seterusnya. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya.

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

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

Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka Anda diminta untuk mengimplementasikan **pembuatan kurva Bézier** dengan algoritma titik tengah berbasis *divide and conquer*.

1.2. Spesifikasi Tugas

Program yang dibangun memanfaatkan algoritma titik tengah berbasis *divide and conquer* untuk membentuk sebuah **kurva Bézier kuadratik**. Program juga perlu diimplementasikan dalam algoritma brute force sebagai pembanding dengan solusi

sebelumnya. Program ditulis dalam salah satu bahasa di antara C++/Python/JavaScript/Go. Berikut adalah detail dari spesifikasi program yang harus dibuat.

1.2.1. *Input*

Pengguna perlu memasukkan:

- **Tiga buah pasangan titik.** Titik yang paling awal dimasukkan akan menjadi titik awal kurva, begitu juga dengan titik yang paling akhir.
- **Jumlah iterasi** yang ingin dilakukan.

1.2.2. *Output*

Program menampilkan:

- Hasil **kurva Bézier yang terbentuk** pada iterasi terkait.
- **Waktu eksekusi** program pembentukan kurva

1.2.3. Bonus

Berikut adalah spesifikasi bonus dari program yang dibangun.

- Melakukan generalisasi algoritma, ide, serta melakukan implementasinya sehingga program dapat membentuk kurva Bézier kubik, kuartik, dan selanjutnya dengan 4, 5, 6, hingga n titik kontrol.
- Memberikan visualisasi proses pembentukan kurva, tidak hanya hasil akhirnya saja.

BAB II

LANDASAN TEORI

2.1. Kurva Bézier

Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol, yang menentukan bentuk dan arah kurva. Kurva Bézier digunakan secara luas dalam grafika komputer untuk memodelkan kurva mulus. Karena kurva ini keseluruhan termasuk dalam convex hull dari titik pengendalinya, titik-titik ini dapat ditampilkan dalam gambar dan digunakan untuk memanipulasi kurva ini secara langsung. Rumus umum kurva Bezier dapat dinyatakan secara eksplisit sebagai berikut

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

Dimana $\binom{n}{i}$ adalah koefisien binomial.

2.2. Algoritma Titik Tengah

Algoritma titik tengah, atau midpoint algorithm, adalah algoritma yang digunakan untuk menemukan titik tengah segmen garis di dalam grid berbasis piksel. Algoritma ini digunakan dalam berbagai aplikasi, terutama dalam bidang grafika komputer untuk menggambar garis secara efisien.

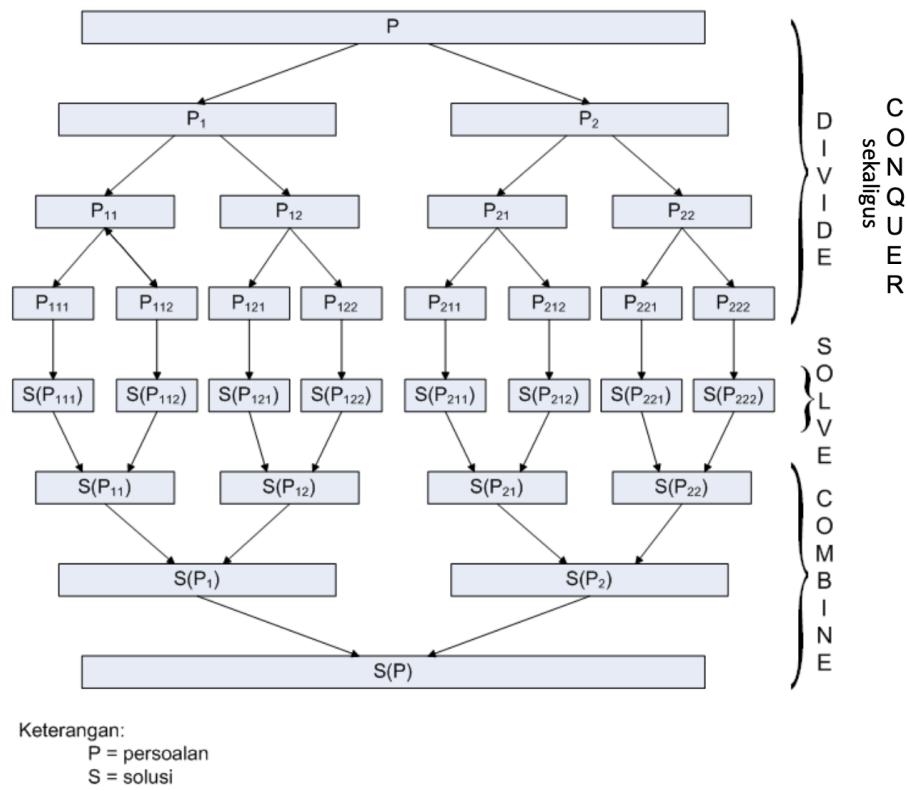
Prinsip dasar algoritma titik tengah adalah menggambar garis dengan menghitung titik-titik piksel yang terletak di dekat garis sejajar dengan sumbu x atau y. Untuk setiap langkah, algoritma memilih antara dua piksel yang berdekatan berdasarkan nilai titik tengah dari garis. Dengan demikian, algoritma ini memungkinkan kita untuk menggambar garis dengan menggunakan operasi perhitungan sederhana seperti penjumlahan dan perbandingan.

2.3. Algoritma Divide and Conquer

Algoritma *divide and conquer* adalah salah satu teknik dalam mendesain algoritma. Algoritma ini terbagi menjadi tiga langkah utama, yaitu sebagai berikut.

1. Membagi (*divide*) persoalan menjadi beberapa upa persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama),
2. Menyelesaikan (*conquer/solve*) masing-masing upa persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar).
3. Menggabungkan (*combine*) solusi masing-masing upa persoalan sehingga membentuk solusi persoalan semula.

Keseluruhan langkah di atas dapat diilustrasikan melalui gambar di bawah ini



Gambar 2.1. Ilustrasi Algoritma *Divide and Conquer*

(Sumber: <https://informatika.stei.itb.ac.id/>)

Berdasarkan metode pembagian dan penggabungan masalah yang digunakan, algoritma *divide and conquer* dapat dikelompokkan ke dalam:

1. *Easy-split/hard-join*: tahap divide sederhana, namun tahap combine rumit dan/atau memakan waktu.
2. *Hard-split/easy-join*: tahap divide rumit dan/atau memakan waktu, namun tahap combine sederhana.

Adapun skema umum algoritma *divide and conquer* antara lain sebagai berikut.

```

procedure DIVIDEandCONQUER(input P : problem, n : integer)
{ Menyelesaikan persoalan P dengan algoritma divide and conquer
  Masukan: masukan persoalan P berukuran n
  Luaran: solusi dari persoalan semula }

Deklarasi
  r : integer

Algoritma
  if n ≤ n0 then {ukuran persoalan P sudah cukup kecil}
    SOLVE persoalan P yang berukuran n ini
  else
    DIVIDE menjadi r upa-persoalan, P1, P2, ..., Pr, yang masing-masing
    berukuran n1, n2, ..., nr,
    for masing-masing P1, P2, ..., Pr, do
      DIVIDEandCONQUER(Pi, ni)
    endfor
    COMBINE solusi dari P1, P2, ..., Pr menjadi solusi persoalan semula
  endif
```

Berdasarkan skema di atas, algoritma *divide and conquer* memiliki kompleksitas waktu:

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \dots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

dengan $T(n)$ adalah kompleksitas waktu penyelesaian persoalan P yang berukuran n , $g(n)$ adalah kompleksitas waktu penyelesaian jika n sudah berukuran kecil, $T(n_1) + T(n_2) + \dots + T(n_r)$ adalah gabungan kompleksitas waktu dari setiap upa persoalan, serta $f(n)$ adalah kompleksitas waktu untuk menggabungkan solusi dari masing-masing upa persoalan menjadi solusi persoalan semula.

Jika pembagian selalu menghasilkan **dua** upa-persoalan yang berukuran sama, maka skema algoritma *divide and conquer* akan menjadi sebagai berikut.

```

procedure DIVIDEandCONQUER2(input P : problem, n : integer)
{ Menyelesaikan persoalan dengan algoritma divide and conquer
  Masukan: masukan yang berukuran n
  Luaran: solusi dari persoalan semula }

Deklarasi
  r : integer

Algoritma
  if n ≤ n0 then {ukuran persoalan P sudah cukup kecil }
    SOLVE persoalan P yang berukuran n ini
  else
    DIVIDE menjadi 2 upa-persoalan, P1 dan P2, masing-masing berukuran n/2
    DIVIDEandCONQUER(P1, n/2)
    DIVIDEandCONQUER(P2, n/2)
    COMBINE solusi dari P1 dan P2
  endif
```

2.4. Algoritma Brute Force

Algoritma Brute force merupakan sebuah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan yang didefinisikan. Biasanya algoritma *brute force* didasarkan pada pernyataan pada persoalan (*problem statement*) dan definisi/konsep yang dilibatkan. Algoritma *brute force* memecahkan persoalan dengan sangat sederhana, langsung, serta jelas caranya (*obvious way*).

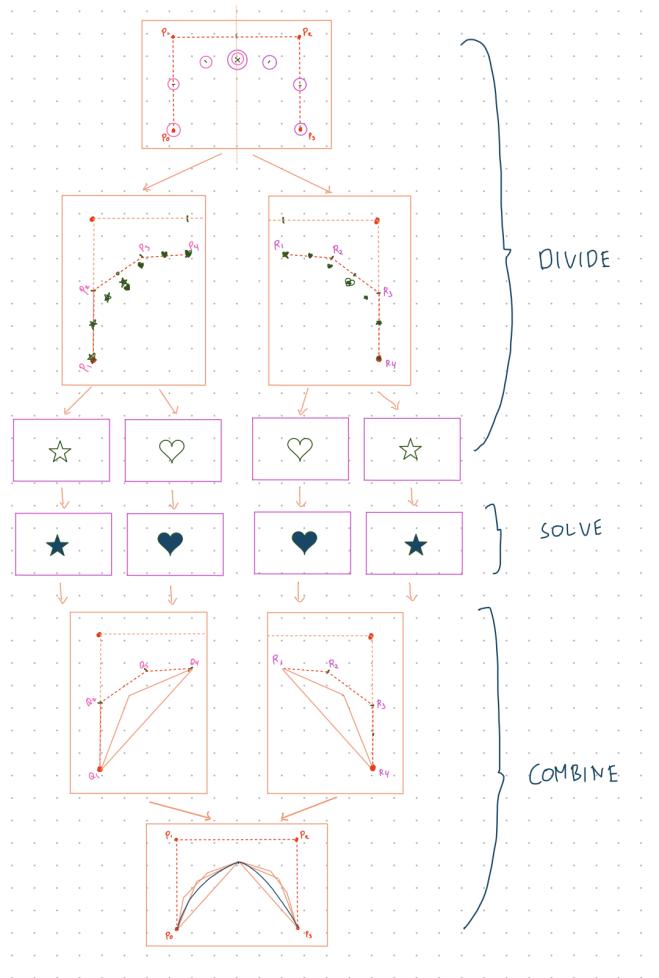
Pendekatan *brute force* seringkali melibatkan enumerasi semua kemungkinan solusi, sebelum akhirnya menghilangkan jawaban yang tidak memenuhi syarat dan mengambil solusi terbaik (apabila ada). Algoritma dengan pendekatan *brute force* dijamin akan menemukan sebuah solusi apabila solusi tersebut ada. Namun, algoritma dengan pendekatan *brute force* seringkali tidak mangkus atau tidak efektif, dengan $O(n)$ yang lebih buruk dari polinomial.

BAB III

IMPLEMENTASI PROGRAM

3.1. Kurva Bézier dengan Memanfaatkan Algoritma *Divide and Conquer* (Diana)

Salah satu cara untuk membentuk kurva Bézier adalah dengan menggunakan algoritma titik tengah atau berbasis algoritma *Divide and Conquer*. Pada algoritma ini, fungsi menerima dua masukan yaitu list yang berisi titik-titik kontrol dan jumlah iterasi yang akan dilakukan. Iterasi yang dimaksud adalah berapa banyak algoritma melakukan ‘divide’ atau pemecahan upa persoalan. Berikut adalah ilustrasi langkah-langkah yang dilakukan algoritma pada setiap iterasi:



Gambar 3.1. Ilustrasi Pembuatan Kurva Bézier dengan Algoritma *Divide and Conquer*

Langkah-Langkah Pembuatan **Kurva Bézier** dengan Algoritma *Divide and Conquer*:

1. Melakukan pengecekan terhadap nilai iterasi yang masuk. Jika iterasi sama dengan nol atau artinya sudah tidak perlu dilakukan iterasi lagi, maka akan langsung mengembalikan nilai (*output*) dua titik yaitu titik pertama dan terakhir. Hal tersebut dilakukan karena titik pertama dan terakhir akan selalu menjadi titik ujung kurva. Pada tahap ini telah dilakukan penyelesaian atau *solve/conquer*.
2. Jika nilai iterasi lebih dari nol atau masih akan dilakukan iterasi, maka akan dilakukan pencarian nilai titik tengah dari setiap dua titik kontrol bersebelahan, yang termuat dalam list secara berurutan. Setiap nilai titik tengah yang ditemukan, disimpan dalam sebuah list baru.
3. Dari list baru (list titik tengah) yang ada, dilakukan kembali pencarian nilai titik tengah dari list baru tersebut. Hal tersebut dilakukan secara terus-menerus hingga ditemukan satu titik tengah terakhir. Titik tengah terakhir inilah yang akan menjadi titik acuan untuk membagi kurva (sebagai ‘permasalahan’) menjadi dua bagian. Tahap ini merupakan tahap untuk mempersiapkan pembagian masalah atau *divide*.
4. Melakukan pembagian kurva menjadi dua bagian yaitu sebelah kiri dan kanan dari titik tengah terakhir. Lalu, setiap bagian melakukan pemanggilan fungsi diri sendiri atau biasa disebut rekursif. Tetapi pemanggilan kali ini memiliki masukan jumlah iterasi berkurang satu (iterasi awal - 1), karena iterasi telah berhasil dilakukan sekali. Sedangkan masukan list titik didapatkan dari selama pencarian titik tengah.
5. Hal yang perlu diperhatikan di sini ialah tidak semua titik tengah yang ditemukan menjadi titik kontrol untuk iterasi selanjutnya. Tetapi, hanya titik pertama dan terakhir yang ditemukan pada setiap pencarian titik tengah yang menjadi bagian list titik iterasi selanjutnya. Dengan titik pertama untuk list titik bagian kiri dan titik terakhir untuk list titik kontrol bagian kanan. Hal ini dilakukan karena titik tengah yang lain sudah tidak relevan atau tidak dapat digunakan kembali menjadi titik kontrol. Lalu, titik pertama dan terakhir pada list titik kontrol masukan awal juga disertakan dalam list titik kontrol bagian kiri (titik pertama) dan kanan (titik

terakhir). Sehingga setiap bagian baru, memiliki jumlah titik kontrol yang sama dengan titik kontrol awal.

6. Setelah seluruh proses berhasil dilakukan, algoritma akan menggabungkan setiap hasil (*output*) dari pemecahan kurva, yang dilakukan secara rekursif menjadi sebuah list. Sehingga, list tersebut berisi titik-titik yang mempresentasikan kurva Bézier dan akan divisualisasikan oleh proses atau fungsi selanjutnya. Pada tahap ini telah dilakukan tahap penggabungan hasil atau *combine*.

Langkah-langkah di atas adalah gambaran algoritma *Divide and Conquer* secara umum, dalam artian jumlah titik kontrol dan iterasi yang menjadi masukan tidak terbatas. Namun, apabila jumlah titik kontrol dipastikan tepat tiga buah titik sehingga menghasilkan kurva Bézier kuadratik, langkah-langkah yang dilakukan lebih sederhana. Masukan yang diperlukan bukan berupa list titik-titik, melainkan tiga buah titik, sehingga pemrosesan berjalan konstan atau tidak bergantung pada panjang list. Untuk tahap penyelesaian (*solve/conquer*) memiliki konsep yang sama, sehingga mengembalikan titik pertama dan terakhir dari tiga titik masukan. Sedangkan pada tahap pemecahan masalah (*divide*), penentuan titik tengah dilakukan tanpa perulangan, sehingga kompleksitas sedikit lebih sederhana. Dilanjutkan dengan pemanggilan secara rekursif setiap bagian kiri dan kanan hingga seluruh solusi dapat digabungkan (*combine*) menjadi satu seperti langkah secara umum.

Semakin banyak jumlah iterasi yang dilakukan, maka kurva Bézier yang dihasilkan akan semakin mulus. Tetapi, tentu saja hal tersebut akan berdampak pada kompleksitas algoritma dan waktu pemrosesan. Setiap iterasi membagi masalah menjadi dua bagian dan melakukan pemanggilan secara rekursif setiap bagiannya, sehingga kompleksitas waktunya adalah $O(2^n)$.

3.2. Kurva Bézier dengan Memanfaatkan Algoritma *Brute Force*

Pembentukan kurva Bézier juga dapat dilakukan dengan menggunakan algoritma *brute force*. Kurva Bezier didefinisikan oleh sebuah set titik kontrol, yang dapat bervariasi dalam jumlah. Dalam grafik vektor, kurva ini digunakan untuk memodelkan kurva mulus. Titik kontrol pertama dan terakhir selalu menjadi titik ujung dari kurva, namun titik kontrol

di antara keduanya tidak selalu berada pada kurva. Dalam contoh ini, kami menunjukkan bagaimana cara membentuk **kurva Bézier kuadratik** dengan algoritma *brute force*.

Langkah-Langkah Pembuatan **Kurva Bézier Kuadratik** dengan Algoritma *Brute Force* :

1. Menentukan 3 titik kontrol yang terdiri dari titik kontrol pertama (P_0), titik kontrol antara (P_1), dan titik kontrol terakhir (P_2).
2. Menentukan jumlah iterasi (i) yang diinginkan untuk menghasilkan kurva Bezier. Nilai iterasi ini akan mempengaruhi halusnya kurva yang dihasilkan.
3. Membuat titik pembentukan kurva pertama dengan $t_1 = 0$, sehingga posisi t_1 akan menggantikan titik kontrol pertama P_0 .
4. Lakukan pembentukan titik kurva selanjutnya dengan menambahkan $\frac{1}{2^i}$. Misalnya $t_2 = \frac{1}{2^i}$, $t_3 = \frac{2}{2^i}$, seterusnya hingga semua titik pada kurva Bezier dihasilkan ($t_k = 1$) menggunakan rumus

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

5. Menghubungkan setiap titik yang dibentuk untuk menghasilkan kurva bázier

Kompleksitas algoritma *brute force* untuk pembentukan kurva Bézier tergantung pada jumlah iterasi yang digunakan. Oleh karena itu, kompleksitas waktu yang diperlukan adalah $O(2^n)$.

3.3. Sistematika File

```
Tucil2_13522069_13522104
|-- bin
|   |-- main.exe
|-- doc
|   |-- Tucil2_13522069_13522104.pdf
|-- src
|   |-- input.py
```

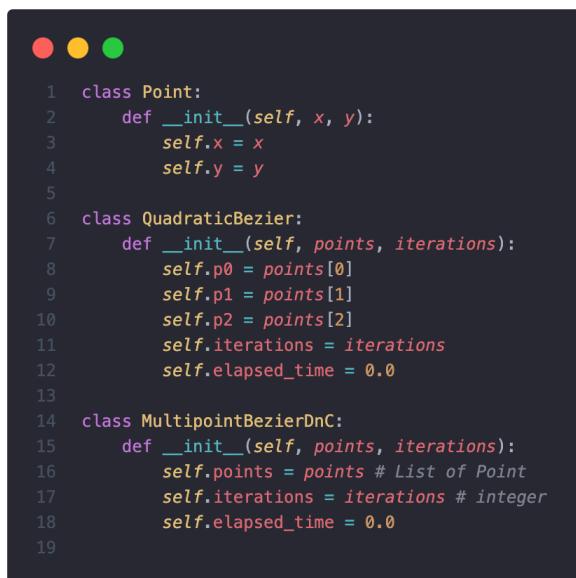
```

| |-- main.py
| |-- multipointBezierDnC.py
| |-- output.py
| |-- quadraticBezierBF.py
| |-- quadraticBezierDnC.py
| |-- unitTestMultipoint.py
| |-- visualizer.py
|-- test
| |-- matplotlib
| |-- terminal
-- README.md

```

3.4. Struktur Data

Struktur data yang dipakai pada implementasi kode adalah list dan struktur data buatan yaitu Point. List digunakan untuk memuat data himpunan Point. Sedangkan Point memiliki atribut x dan y yang mempresentasikan titik koordinat (x,y). Dengan adanya struktur data tersebut, data lebih terorganisir dan lebih mudah untuk diakses maupun diubah. Struktur data buatan Point ditulis dalam bahasa python. Kode terdapat pada direktori src dengan nama file datatype.py.



```

1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6  class QuadraticBezier:
7      def __init__(self, points, iterations):
8          self.p0 = points[0]
9          self.p1 = points[1]
10         self.p2 = points[2]
11         self.iterations = iterations
12         self.elapsed_time = 0.0
13
14 class MultipointBezierDnC:
15     def __init__(self, points, iterations):
16         self.points = points # List of Point
17         self.iterations = iterations # integer
18         self.elapsed_time = 0.0
19

```

Gambar 3.2. Source Code “datatype.py”

3.5. Source Code Program

3.5.1. main.py

```
● ● ●  
1 import input  
2 import output  
3 import visualizer  
4  
5 def main():  
6     output.displayMenu()  
7     choice = input.getChoice()  
8  
9     if choice == 1 or choice == 2:  
10         if choice == 1:  
11             curve_method = 'Brute Force'  
12         elif choice == 2:  
13             curve_method = 'Divide and Conquer'  
14         else:  
15             raise ValueError("Jenis kurva tidak valid.")  
16         visualizer.plot_curve(curve_method)  
17  
18     elif choice == 3: # Membandingkan 2 Algoritma  
19         points, iterations = input.getInputQuadratic()  
20         elapsed_time_bf = output.calculateElapsedTime(points, iterations, 'Brute Force')  
21         elapsed_time_dnc = output.calculateElapsedTime(points, iterations, 'Divide and Conquer')  
22         output.displayComparison(elapsed_time_bf, elapsed_time_dnc)  
23         output.displayTotalPoints(iterations)  
24  
25     elif choice == 4:  
26         curve_method = 'Divide and Conquer with Multipoint'  
27         visualizer.plot_curve(curve_method)  
28  
29     else:  
30         print("Pilihan tidak valid. Silakan pilih antara 1, 2, 3, atau 4.")  
31  
32 if __name__ == "__main__":  
33     main()  
34
```

Gambar 3.3. Source Code “main.py”

3.5.2. input.py



```

1  from datatypes import Point
2
3  # Mendapatkan titik koordinat dari pengguna
4  def getPoints(n):
5      points = []
6      print("\nMasukkan titik koordinat dipisahkan dengan koma ,")
7      for i in range(n):
8          while True:
9              point_input = input(f"Masukkan titik kontrol ke-{i+1} (P{i}): ")
10             coords = point_input.split(',')
11             if len(coords) != 2:
12                 print("Input tidak valid.")
13                 Mohon pastikan titik kontrol memiliki dua angka dan dipisahkan dengan satu koma (x,y)."
14                 continue
15             try:
16                 x, y = map(float, coords)
17                 point = Point(x,y)
18                 points.append(point)
19                 break
20             except ValueError:
21                 print("Input tidak valid. Mohon pastikan kedua angka adalah bilangan float atau integer.")
22     return points
23
24  # Mendapatkan jumlah iterasi dari pengguna
25  def getIteration():
26      while True:
27          iterations = int(input("\nMasukkan jumlah iterasi: "))
28          if iterations > 1:
29              return iterations
30          elif iterations == 1:
31              print("Iterasi 1 akan menghasilkan kurva linear."
32                  Mohon masukkan nilai iterasi yang lebih besar dari 1.")
33          else:
34              print("Jumlah iterasi harus lebih besar dari 1.
35                  Mohon masukkan nilai iterasi yang valid.")
36
37  # Mendapatkan pilihan menu dari pengguna
38  def getChoice():
39      while True:
40          choice = input("\nPilih menu (1/2/3/4): ")
41          try:
42              choice = int(choice)
43              if choice in [1, 2, 3, 4]:
44                  return choice
45              else:
46                  print("Pilihan tidak valid. Silakan pilih antara 1, 2, 3, atau 4.")
47          except ValueError:
48              print("Input tidak valid. Masukkan bilangan bulat antara 1 hingga 4.")
49
50  # Mendapatkan input titik dan iterasi untuk kurva quadratic Bézier
51  def getInputQuadratic():
52      points = getPoints(3)
53      iterations = getIteration()
54
55      return points, iterations
56
57  # Mendapatkan input titik dan iterasi untuk kurva multipoint Bézier
58  def getInputMultipoint():
59      n = int(input("Jumlah titik yang akan dimasukkan: "))
60      points = getPoints(n)
61      iterations = getIteration()
62
63      return points, iterations
64

```

Gambar 3.4. Source Code “input.py”

3.5.3. output.py



```

1 # Menampilkan menu pilihan kepada pengguna
2 def displayMenu():
3     print("\n===== MENU =====")
4     print("1. Quadratic Bézier Curve (Brute Force)")
5     print("2. Quadratic Bézier Curve (Divide and Conquer)")
6     print("3. Comparing Algorithm")
7     print("4. Multi Bézier Curve (Divide and Conquer)")
8     print("")
9
10 # Menampilkan perbandingan waktu antara algoritma Brute Force dan Divide and Conquer
11 def displayComparison(elapsed_time_bf, elapsed_time_dnc):
12     print("\n===== Membandingkan Algoritma =====")
13     print("Brute Force")
14     print("Elapsed time: {:.4e} seconds\n".format(elapsed_time_bf))
15
16     print("Divide and Conquer")
17     print("Elapsed time: {:.4e} seconds\n".format(elapsed_time_dnc))
18
19     print("===== Analisis =====")
20     if elapsed_time_bf < elapsed_time_dnc:
21         print("Pembentukan kurva Bézier lebih cepat dengan Algoritma Brute Force")
22         print(f"dengan selisih waktu eksekusi: {elapsed_time_dnc - elapsed_time_bf}")
23     elif elapsed_time_bf > elapsed_time_dnc:
24         print("Pembentukan kurva Bézier lebih cepat dengan Algoritma Divide and Conquer")
25         print(f"dengan selisih waktu eksekusi: {elapsed_time_dnc - elapsed_time_bf}")
26     else:
27         print("Waktu pembentukan kurva Bezier sama cepatnya dengan kedua metode.")
28
29 # Menampilkan jumlah total titik kontrol
30 def displayTotalPoints(iterations):
31     print(f"Total points: {2**iterations+1}\n")
32
33 # Menampilkan waktu yang diperlukan untuk pembentukan kurva
34 def displayElapsedTime(points, iterations, curve_method):
35     elapsed_time = calculateElapsedTime(points, iterations, curve_method)
36     print("Elapsed time: {:.4e} seconds\n".format(elapsed_time))
37
38 # Menghitung waktu yang diperlukan untuk pembentukan kurva
39 def calculateElapsedTime(points, iterations, curve_method):
40     import time
41     from quadraticBezierBF import quadraticBezierBF
42     from quadraticBezierDnC import quadraticBezierDnC
43
44     start_time = time.time()
45     if curve_method == 'Brute Force':
46         _ = quadraticBezierBF(points, iterations).quadratic_bezier()
47     elif curve_method == 'Divide and Conquer':
48         _ = quadraticBezierDnC(points, iterations).quadratic_bezier()
49     end_time = time.time()
50
51     elapsed_time = end_time - start_time
52     return elapsed_time

```

Gambar 3.5. Source Code “output.py”

3.5.4. visualizer.py



```

1 import matplotlib.pyplot as plt
2 import time
3 from quadraticBezierBF import quadraticBezierBF
4 from quadraticBezierDnC import quadraticBezierDnC
5 from multipointBezierDnC import multipointBezierDnC
6 from input import getInputQuadratic, getInputMultipoint
7
8 # Fungsi untuk menampilkan kurva Bézier kuadratik
9 def plot_curve(curve_method):
10
11     # Memilih metode pembentukan kurva Bézier berdasarkan pilihan pengguna
12     start_time = time.time()
13     if curve_method == 'Brute Force':
14         points, iterations = getInputQuadratic()
15         bezier_curve = quadraticBezierBF(points, iterations)
16         curve_points = bezier_curve.quadratic_bezier()
17     elif curve_method == 'Divide and Conquer':
18         points, iterations = getInputQuadratic()
19         bezier_curve = quadraticBezierDnC(points, iterations)
20         curve_points = bezier_curve.quadratic_bezier()
21     elif curve_method == 'Divide and Conquer with Multipoint':
22         points, iterations = getInputMultipoint()
23         bezier_curve = multipointBezierDnC(points, iterations)
24         curve_points = bezier_curve.multipoint_bezier()
25     else:
26         raise ValueError("Jenis kurva tidak valid.")
27
28     # Menghitung waktu yang diperlukan untuk pembentukan kurva
29     end_time = time.time()
30     elapsed_time = end_time - start_time
31
32     # Menampilkan titik awal input
33     x_points = [point.x for point in points]
34     y_points = [point.y for point in points]
35     plt.scatter(x_points, y_points, color='darkred', zorder=5)
36     plt.plot(x_points, y_points, color='red', linestyle='dashed')
37
38     # Menampilkan kurva Bézier dan titik kontrolnya
39     x_curve = [point.x for point in curve_points]
40     y_curve = [point.y for point in curve_points]
41     plt.plot(x_curve, y_curve, color='royalblue')
42     plt.scatter(x_curve, y_curve, color='blue', s=8, zorder=5)
43
44     plt.xlabel('X-axis')
45     plt.ylabel('Y-axis')
46
47     # Menampilkan judul plot dan waktu yang diperlukan untuk pembentukan kurva
48     if curve_method == 'Divide and Conquer with Multipoint':
49         plt.title(f'Bézier Curve with {bezier_curve.iterations} Iteration {curve_method}')
50     else:
51         plt.title(f'Quadratic Bézier Curve with {bezier_curve.iterations} Iteration {curve_method}')
52     plt.text(0.5, 0.05, f'Elapsed time: {elapsed_time:.4e} seconds', horizontalalignment='center', verticalalignment='center', transform=plt.gca().transAxes)
53
54     # Menampilkan grid
55     plt.grid(True)
56     plt.show()
57
58

```

Gambar 3.6. Source Code “visualizer.py”

```

12 # Memilih metode pembentukan kurva Bézier berdasarkan pilihan pengguna
13 if curve_method == 'Brute Force':
14     points, iterations = getInputQuadratic()
15     start_time = time.time()
16     bezier_curve = quadraticBezierBF(points, iterations)
17     curve_points = bezier_curve.quadratic_bezier()
18     end_time = time.time()
19     elapsed_time = end_time - start_time
20
21 # Menyiapkan plot untuk animasi
22 fig, ax = plt.subplots()
23 ax.set_xlabel('X-axis')
24 ax.set_ylabel('Y-axis')
25 ax.set_title(f'Quadratic Bézier Curve with {bezier_curve.iterations} Iteration {curve_method}')
26 plt.text(0.5, 0.05, f'Elapsed time: {:.4e} seconds'.format(elapsed_time),
27         horizontalalignment='center', verticalalignment='center', transform=plt.gca().transAxes)
28
29 # Inisialisasi plot untuk animasi
30 curve_lines = []
31 for i in range(len(curve_points) - 1):
32     curve_line, = ax.plot([], [], color='royalblue')
33     curve_lines.append(curve_line)
34
35 # Menampilkan titik awal input
36 x_points = [point.x for point in points]
37 y_points = [point.y for point in points]
38 ax.scatter(x_points, y_points, color='darkred', zorder=5)
39 ax.plot(x_points, y_points, color='red', linestyle='dashed')
40
41 # Fungsi inisialisasi untuk animasi
42 def init():
43     for line in curve_lines:
44         line.set_data([], [])
45     return curve_lines
46
47 # Fungsi update untuk animasi
48 def update(frame):
49     for i in range(len(curve_lines)):
50         if frame > i:
51             x_data = [curve_points[i].x, curve_points[i+1].x]
52             y_data = [curve_points[i].y, curve_points[i+1].y]
53             curve_lines[i].set_data(x_data, y_data)
54     return curve_lines
55
56 # Menampilkan kurva Bézier dan titik kontrolnya
57 x_curve = [point.x for point in curve_points]
58 y_curve = [point.y for point in curve_points]
59 plt.scatter(x_curve, y_curve, color='blue', s=8, zorder=5)
60
61 # Menampilkan animasi
62 ani = FuncAnimation(fig, update, frames=len(curve_points), init_func=init, blit=True, repeat=False)
63
64 # Menampilkan grid
65 ax.grid(True)
66 plt.show()
67

```

Gambar 3.7. Source Code Brute Force di “visualizer.py”

3.5.5. quadraticBezierBF.py

```
● ● ●  
1 import matplotlib.pyplot as plt  
2 import numpy as np  
3 from datatypes import Point  
4  
5 # Definisikan kelas quadraticBezierBF  
6 class quadraticBezierBF:  
7     # Inisialisasi atribut dari objek quadraticBezierBF  
8     def __init__(self, points, iterations):  
9         self.p0 = points[0]  
10        self.p1 = points[1]  
11        self.p2 = points[2]  
12        self.iterations = iterations  
13        self.elapsed_time = 0.0  
14  
15    # Fungsi untuk menghasilkan titik-titik pada kurva Bézier kuadratik menggunakan metode Brute Force  
16    def quadratic_bezier(self):  
17        points = []  
18        for t in np.linspace(0, 1, 2 ** self.iterations + 1):  
19            x = (1 - t) ** 2 * self.p0.x + 2 * (1 - t) * t * self.p1.x + t ** 2 * self.p2.x  
20            y = (1 - t) ** 2 * self.p0.y + 2 * (1 - t) * t * self.p1.y + t ** 2 * self.p2.y  
21            point = Point(x, y)  
22            points.append(point)  
23  
24        return points  
25
```

Gambar 3.8. Source Code “quadraticBezierBF.py”

3.5.6. quadraticBezierDnC.py

```

● ● ●

1 import matplotlib.pyplot as plt
2 from datatypes import Point
3
4 # Definisikan kelas quadraticBezierDnC
5 class quadraticBezierDnC:
6     # Inisialisasi atribut dari objek quadraticBezierDnC
7     def __init__(self, points, iterations):
8         self.p0 = points[0]
9         self.p1 = points[1]
10        self.p2 = points[2]
11        self.iterations = iterations
12        self.elapsed_time = 0.0
13
14    # Fungsi mencari titik tengah dari masukan 2 titik
15    def find_center_point(self, point1, point2):
16        x = (point1.x + point2.x)/2
17        y = (point1.y + point2.y)/2
18        center_point = Point(x,y)
19        return center_point # Point
20
21    # Fungsi rekursif untuk menghasilkan kurva Bézier kuadratik menggunakan metode Divide and Conquer
22    def generate_quadratic_bezier(self, P0, P1, P2, iterations):
23        # (Conquer/Solve) jika iterasi sudah mencapai 0, kembalikan titik awal dan titik akhir
24        if iterations == 0:
25            return [P0, P2]
26
27        # Mencari titik tengah dari ketiga titik masukan
28        Q0 = self.find_center_point(P0, P1)
29        Q1 = self.find_center_point(P1, P2)
30        R0 = self.find_center_point(Q0, Q1)
31
32        # (Divide and ) Rekursi untuk menemukan kurva di sisi kiri dan kanan
33        left_curve = self.generate_quadratic_bezier(P0, Q0, R0, iterations - 1)
34        right_curve = self.generate_quadratic_bezier(R0, Q1, P2, iterations - 1)
35
36        # (Combine) Gabungkan kurva dari kedua sisi dan hilangkan titik terakhir
37        # dari sisi kiri untuk menghindari duplikasi
38        return left_curve[:-1] + right_curve # List of Point
39
40    # Fungsi untuk menghasilkan titik-titik pada kurva Bézier kuadratik
41    # menggunakan metode Divide and Conquer
42    def quadratic_bezier(self):
43        curve_points = self.generate_quadratic_bezier(self.p0, self.p1, self.p2, self.iterations)
44        return curve_points # List of Point
45

```

Gambar 3.9. Source Code “quadraticBezierDnC.py”

3.5.7. unitTestMultipoint.py

```
● ● ●  
1 import matplotlib.pyplot as plt  
2 import time  
3 from datatypes import Point  
4  
5 # fungsi membuat list titik kurva bezier kuadratik  
6 def generate_quadratic_bezier(P0, P1, P2, iterations):  
7     if iterations == 0:  
8         return [P0, P2]  
9  
10    Q0_x = (P0.x + P1.x) / 2  
11    Q0_y = (P0.y + P1.y) / 2  
12    Q0 = Point(Q0_x, Q0_y)  
13    Q1_x = (P1.x + P2.x) / 2  
14    Q1_y = (P1.y + P2.y) / 2  
15    Q1 = Point(Q1_x, Q1_y)  
16    R0_x = (Q0.x + Q1.x) / 2  
17    R0_y = (Q0.y + Q1.y) / 2  
18    R0 = Point(R0_x, R0_y)  
19  
20    left_curve = generate_quadratic_bezier(P0, Q0, R0, iterations - 1)  
21    right_curve = generate_quadratic_bezier(R0, Q1, P2, iterations - 1)  
22  
23    return left_curve[:-1] + right_curve  
24  
25 # fungsi mencari titik tengah dari dua titik  
26 def find_center_point(point1, point2):  
27     x = (point1.x + point2.x)/2  
28     y = (point1.y + point2.y)/2  
29     center_point = Point(x,y)  
30     return center_point  
31  
32 # fungsi mengumpulkan titik tengah dari sebuah list titik terurut  
33 def list_center_point(listpoints):  
34     new_points = []  
35     n = len(listpoints)  
36  
37     for i in range (n-1):  
38         new_points.append(find_center_point(listpoints[i], listpoints[i+1]))  
39  
40     return new_points
```

Gambar 3.10. Source Code “unitTestMultipoint.py” (1)

```
● ● ●  
42 # fungsi membuat list titik kurva bezier secara general  
43 def bezier_multipoint(listpoints, iterations):  
44     if iterations == 0:  
45         return [listpoints[0], listpoints[len(listpoints)-1]]  
46  
47     n = len(listpoints)  
48  
49     left_points = []  
50     left_points.append(listpoints[0])  
51     right_points = []  
52     right_points.append(listpoints[n-1])  
53  
54     new_points = list_center_point(listpoints)  
55     left_points.append(new_points[0])  
56     if len(new_points) == 1:  
57         right_points.insert(0, new_points[0])  
58     else:  
59         right_points.insert(0, new_points[len(new_points)-1])  
60  
61     while len(new_points) > 1:  
62         print_list_points(new_points)  
63         new_points = list_center_point(new_points)  
64         left_points.append(new_points[0])  
65         if len(new_points) == 1:  
66             right_points.insert(0, new_points[0])  
67         else:  
68             right_points.insert(0, new_points[len(new_points)-1])  
69  
70     print("left points:" + str(iterations))  
71     print_list_points(left_points)  
72     print("right points:" + str(iterations))  
73     print_list_points(right_points)  
74  
75     left_curve = bezier_multipoint(left_points, iterations-1)  
76     right_curve = bezier_multipoint(right_points, iterations-1)  
77  
78     return left_curve[:-1] + right_curve
```

Gambar 3.11. Source Code “unitTestMultipoint.py” (2)

```
● ● ●  
80 # fungsi membuat visualisasi dalam plot  
81 def plot_curve(points, iterations):  
82     start_time = time.time()  
83     # curve = generate_quadratic_bezier(points[0], points[1], points[2], iterations)  
84     curve = bezier_multipoint(points, iterations)  
85     end_time = time.time()  
86     duration = (end_time-start_time)  
87     print(duration)  
88     # Generate x-coordinates for the curve  
89     x_curve = [point.x for point in curve]  
90     y_curve = [point.y for point in curve]  
91     # Plot the curve and points  
92     plt.plot(x_curve, y_curve, marker='o', color='blue')  
93     # Extract x and y coordinates from points  
94     x_points = [point.x for point in points]  
95     y_points = [point.y for point in points]  
96     # Plot the points  
97     plt.plot(x_points, y_points, marker='o', linestyle='dashed', color='red')  
98     plt.title(f'Quadratic Bézier Curve (Iterations: {iterations})')  
99     plt.xlabel('X')  
100    plt.ylabel('Y')  
101    plt.grid(True)  
102    plt.show()  
103  
104 # fungsi menerima input point  
105 def input_points():  
106     points = []  
107     for i in range(4):  
108         x = float(input(f"Masukkan koordinat x titik ke-{i+1}: "))  
109         y = float(input(f"Masukkan koordinat y titik ke-{i+1}: "))  
110         point = Point(x, y)  
111         points.append(point)  
112     return points  
113  
114 def print_list_points(points):  
115     for i, point in enumerate(points):  
116         print(f"Titik {i+1}: (x={point.x}, y={point.y})")  
117  
118 points = input_points()  
119 print("Titik yang dimasukkan:")  
120 for i, point in enumerate(points):  
121     print(f"Titik {i+1}: (x={point.x}, y={point.y})")  
122  
123 iterations = int(input(f"Masukkan iterasi: "))  
124  
125 plot_curve(points, iterations)  
126
```

Gambar 3.12. Source Code “unitTestMultipoint.py” (3)

3.5.8. multipointBezierDnC.py



```

1 import matplotlib.pyplot as plt
2 from datatypes import Point
3
4 # Definisikan kelas quadraticBezierDnC
5 class quadraticBezierDnC:
6     # Inisialisasi atribut dari objek quadraticBezierDnC
7     def __init__(self, points, iterations):
8         self.p0 = points[0]
9         self.p1 = points[1]
10        self.p2 = points[2]
11        self.iterations = iterations
12        self.elapsed_time = 0.0
13
14    # Fungsi mencari titik tengah dari masukan 2 titik
15    def find_center_point(self, point1, point2):
16        x = (point1.x + point2.x)/2
17        y = (point1.y + point2.y)/2
18        center_point = Point(x,y)
19        return center_point # Point
20
21    # Fungsi rekursif untuk menghasilkan kurva Bézier kuadratik
22    # menggunakan metode Divide and Conquer
23    def generate_quadratic_bezier(self, P0, P1, P2, iterations):
24        # (Conquer/Solve) jika iterasi sudah mencapai 0,
25        # kembalikan titik awal dan titik akhir
26        if iterations == 0:
27            return [P0, P2]
28
29        # Mencari titik tengah dari ketiga titik masukan
30        Q0 = self.find_center_point(P0, P1)
31        Q1 = self.find_center_point(P1, P2)
32        R0 = self.find_center_point(Q0, Q1)
33
34        plt.scatter([Q0.x, R0.x, Q1.x], [Q0.y, R0.y, Q1.y], color='orange', s=5.5, zorder=5)
35        plt.plot([Q0.x, R0.x, Q1.x], [Q0.y, R0.y, Q1.y], color='gold', linestyle='dashed')
36
37        # (Divide and ) Rekursi untuk menemukan kurva di sisi kiri dan kanan
38        left_curve = self.generate_quadratic_bezier(P0, Q0, R0, iterations - 1)
39        right_curve = self.generate_quadratic_bezier(R0, Q1, P2, iterations - 1)
40
41        # (Combine) Gabungkan kurva dari kedua sisi dan hilangkan titik terakhir
42        # dari sisi kiri untuk menghindari duplikasi
43        return left_curve[:-1] + right_curve # List of Point
44
45    # Fungsi untuk menghasilkan titik-titik pada kurva Bézier kuadratik
46    # menggunakan metode Divide and Conquer
47    def quadratic_bezier(self):
48        curve_points = self.generate_quadratic_bezier(self.p0, self.p1, self.p2, self.iterations)
49        return curve_points # List of Point
50

```

Gambar 3.13. Source Code “multipointBezierDnC.py”

BAB IV

PENGUJIAN DAN ANALISIS

4.1. Spesifikasi Komputer Uji Coba

Uji coba subbab 4.2 dan subbab 4.3 untuk nomor **ganjil** dilakukan pada komputer dengan spesifikasi sebagai berikut.

Tabel 4.1. Spesifikasi Komputer Uji Coba Nomor Ganjil

Nama	MacBook Air 2020
Prosesor	1,1 GHz Dual-Core Intel Core i3
RAM	8 GB 3733 MHz LPDDR4X
OS	macOS Sonoma Version 14.3.1

Sedangkan, uji coba subbab 4.2 dan subbab 4.3 untuk nomor **genap** dilakukan pada komputer dengan spesifikasi sebagai berikut. Begitu juga untuk subbab 4.4 secara keseluruhan.

Tabel 4.2. Spesifikasi Komputer Uji Coba Nomor Genap

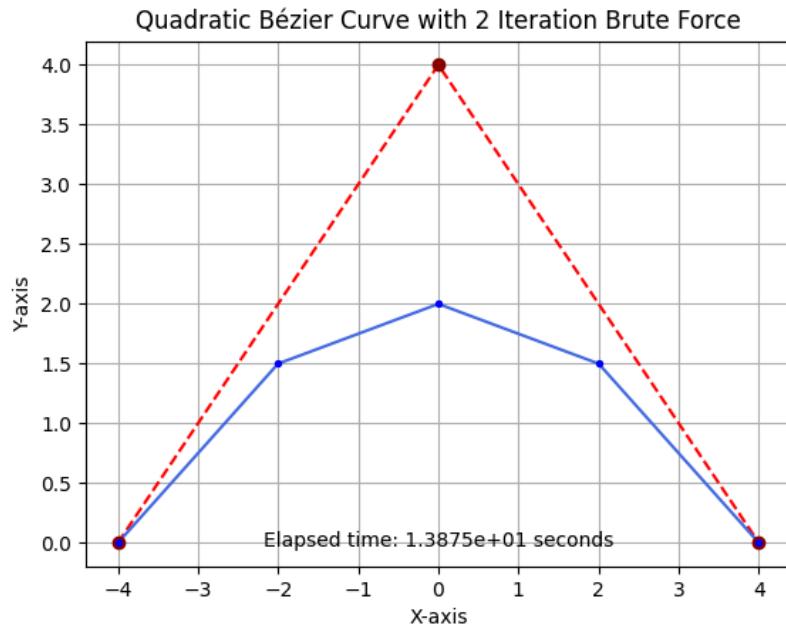
Nama	HP Pavilion Gaming Laptop 15
Prosesor	AMD Ryzen 5 5600H
RAM	16 GB
OS	Microsoft Windows 11 HSL

4.2. Uji Coba Kurva Bézier Kuadratik

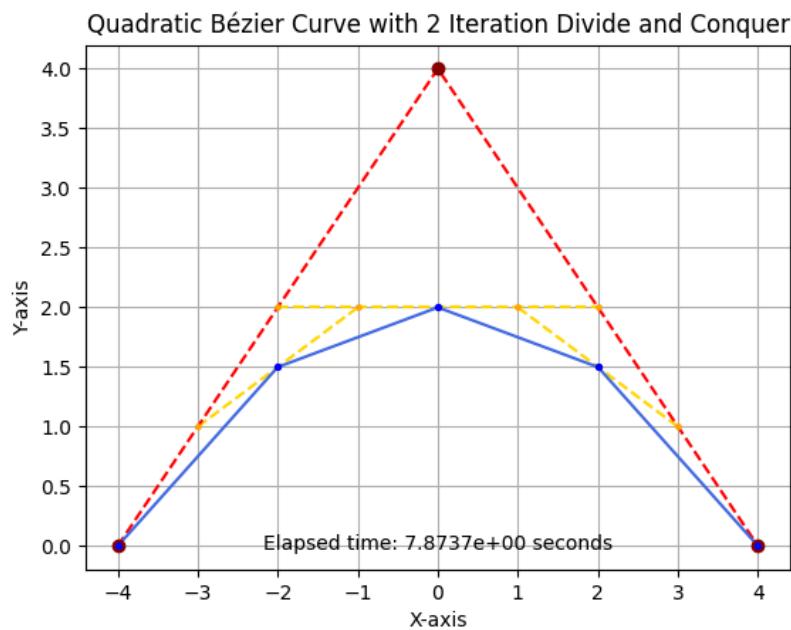
Setiap hasil visualisasi di bawah, grafik warna merah menggambarkan titik-titik masukan dari pengguna, grafik warna kuning menggambarkan proses penentuan titik tengah untuk membentuk kurva bezier, serta grafik warna biru menggambarkan hasil akhir kurva bezier.

4.2.1. Pengujian I (2 Iterasi)

Titik Koordinat: $P_0(-4, 0)$, $P_1(0, 4)$, $P_2(4, 0)$



Gambar 4.1. Pengujian I dengan Algoritma *Brute Force*



Gambar 4.2. Pengujian I dengan Algoritma *Divide and Conquer*

- Perbandingan Algoritma Brute Force dengan Algoritma Divide and Conquer

```
===== MENU =====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

Pilih menu (1/2/3/4): 3

Masukkan titik koordinat dipisahkan dengan koma (, )
Masukkan titik kontrol ke-1 (P0): -4,0
Masukkan titik kontrol ke-2 (P1): 0,4
Masukkan titik kontrol ke-3 (P2): 4,0

Masukkan jumlah iterasi: 2

===== Membandingkan Algoritma =====
Brute Force
Elapsed time: 3.9101e-04 seconds

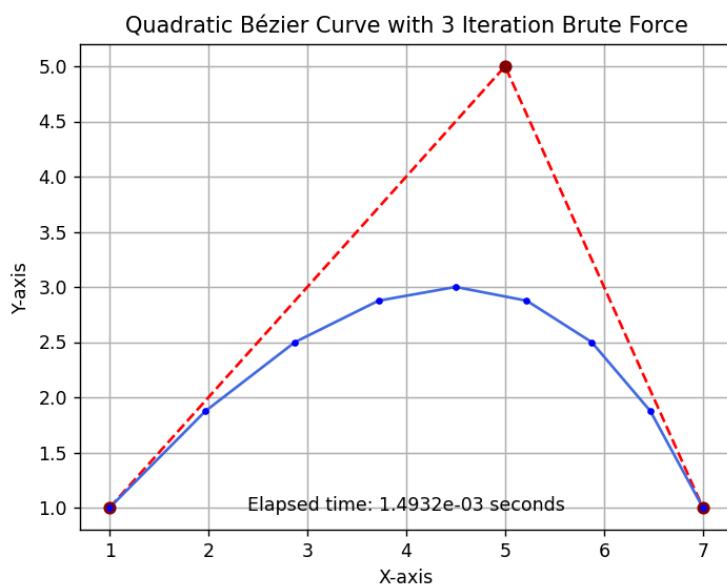
Divide and Conquer
Elapsed time: 2.6926e-01 seconds

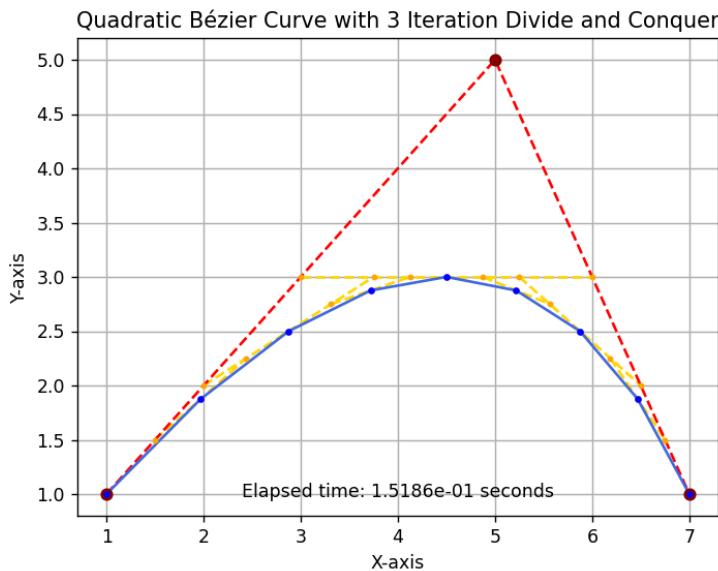
===== Analisis =====
Pembentukan kurva Bézier lebih cepat dengan Algoritma Brute Force.
Total points: 5
```

Gambar 4.3. Input dan Output Pengujian I

4.2.2. Pengujian II (3 Iterasi)

Titik Koordinat: $P_0(1, 1), P_1(5, 5), P_2(7, 1)$



Gambar 4.4. Pengujian II dengan Algoritma *Brute Force***Gambar 4.5.** Pengujian II dengan Algoritma *Divide and Conquer*

- Perbandingan Algoritma Brute Force dengan Algoritma Divide and Conquer

```
----- MENU -----
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

Pilih menu (1/2/3/4): 3

Masukkan titik koordinat dipisahkan dengan koma (,) 
Masukkan titik kontrol ke-1 (P0): 1,1
Masukkan titik kontrol ke-2 (P1): 5,5
Masukkan titik kontrol ke-3 (P2): 7,1

Masukkan jumlah iterasi: 3

===== Membandingkan Algoritma =====
Brute Force
Elapsed time: 0.0000e+00 seconds

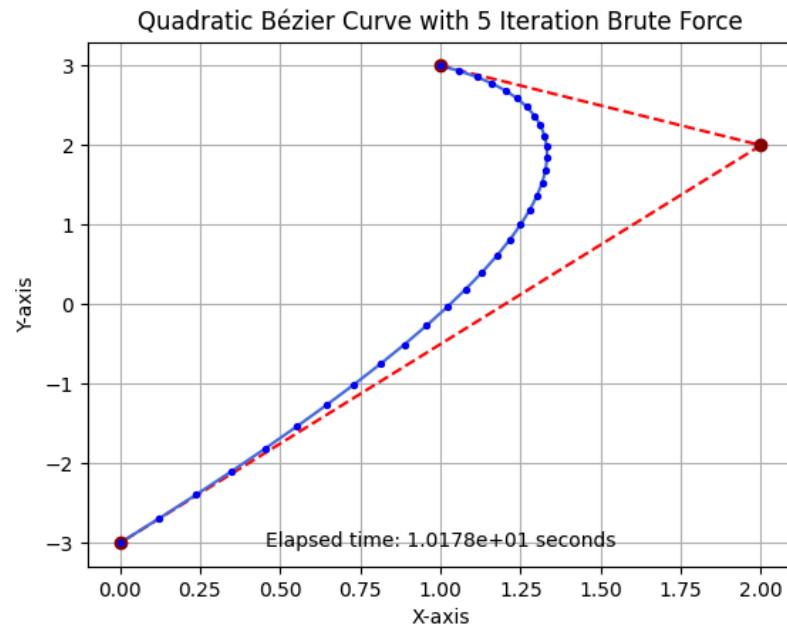
Divide and Conquer
Elapsed time: 0.0000e+00 seconds

===== Analisis =====
Waktu pembentukan kurva Bezier sama cepatnya dengan kedua metode.
Total points: 9
```

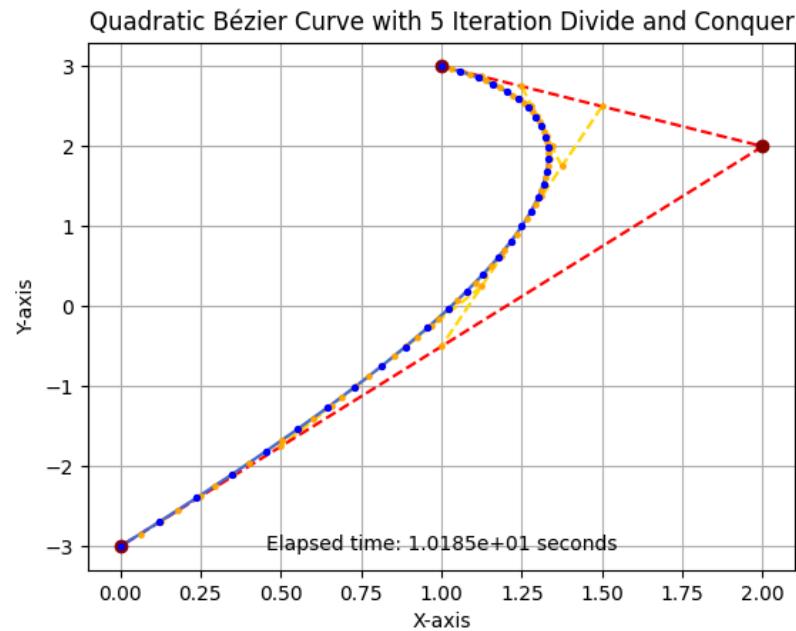
Gambar 4.6. Input dan Output Pengujian II

4.2.3. Pengujian III (5 Iterasi)

Titik Koordinat: $P_0(1, 3)$, $P_1(2, 2)$, $P_2(0, -3)$



Gambar 4.7. Pengujian III dengan Algoritma *Brute Force*



Gambar 4.8. Pengujian III dengan Algoritma *Divide and Conquer*

- Perbandingan Algoritma Brute Force dengan Algoritma Divide and Conquer

```
===== MENU =====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

Pilih menu (1/2/3/4): 3

Masukkan titik koordinat dipisahkan dengan koma (,)
Masukkan titik kontrol ke-1 (P0): 1,3
Masukkan titik kontrol ke-2 (P1): 2,2
Masukkan titik kontrol ke-3 (P2): 0,-3

Masukkan jumlah iterasi: 5

===== Membandingkan Algoritma =====
Brute Force
Elapsed time: 1.1530e-03 seconds

Divide and Conquer
Elapsed time: 4.1524e-01 seconds

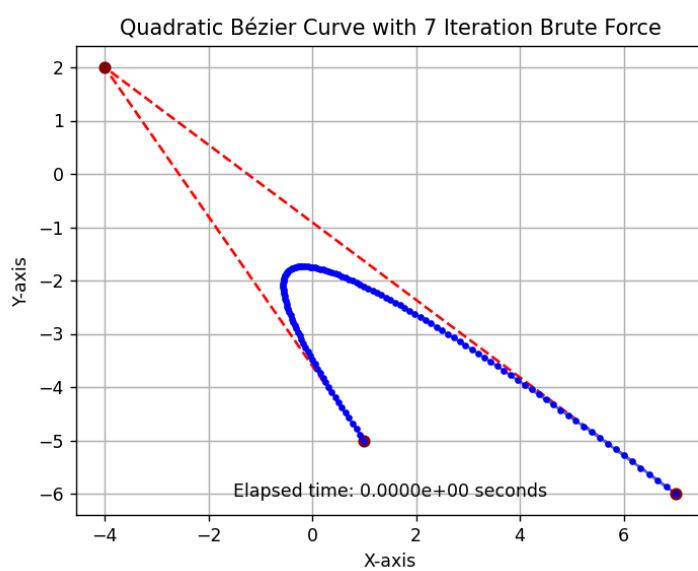
===== Analisis =====
Pembentukan kurva Bézier lebih cepat dengan Algoritma Brute Force.
Total points: 33
```

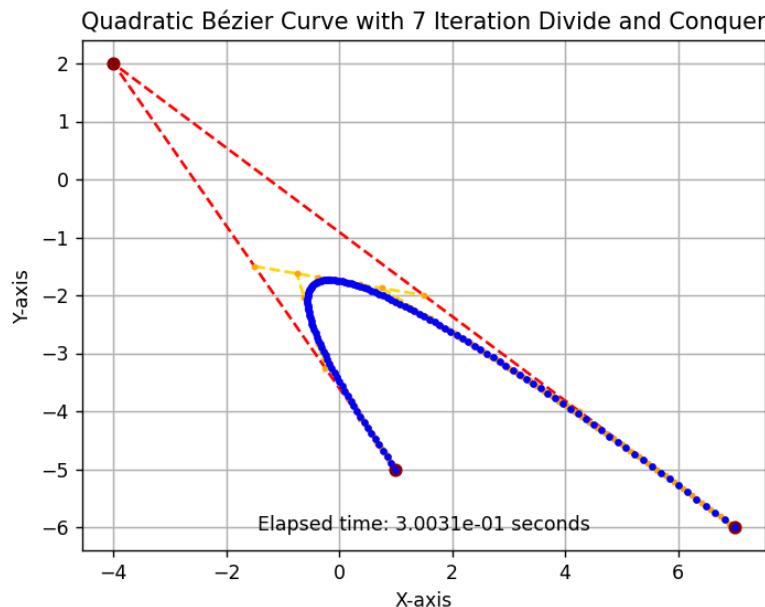
Gambar 4.9. Input dan Output Pengujian III

4.2.4. Pengujian IV (7 Iterasi)

- Metode Algoritma Brute Force

Titik Koordinat: $P_0(1,-5)$, $P_1(-4, 2)$, $P_2(7, -6)$



Gambar 4.10. Pengujian IV dengan Algoritma *Brute Force***Gambar 4.11.** Pengujian IV dengan Algoritma *Divide and Conquer*

- Perbandingan Algoritma Brute Force dengan Algoritma Divide and Conquer

```
=====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

Pilih menu (1/2/3/4): 3

Masukkan titik koordinat dipisahkan dengan koma (,) 
Masukkan titik kontrol ke-1 (P0): 1,-5
Masukkan titik kontrol ke-2 (P1): -4,2
Masukkan titik kontrol ke-3 (P2): 7,-6

Masukkan jumlah iterasi: 7

===== Membandingkan Algoritma =====
Brute Force
Elapsed time: 0.0000e+00 seconds

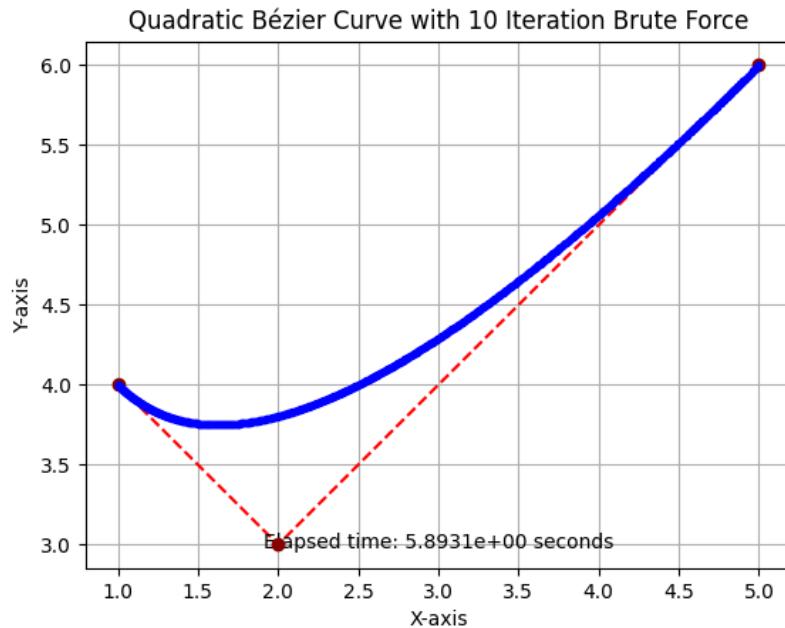
Divide and Conquer
Elapsed time: 5.0426e-04 seconds

===== Analisis =====
Pembentukan kurva Bézier lebih cepat dengan Algoritma Brute Force dengan selisih waktu eksekusi:
0.0005042552947998047
Total points: 129
```

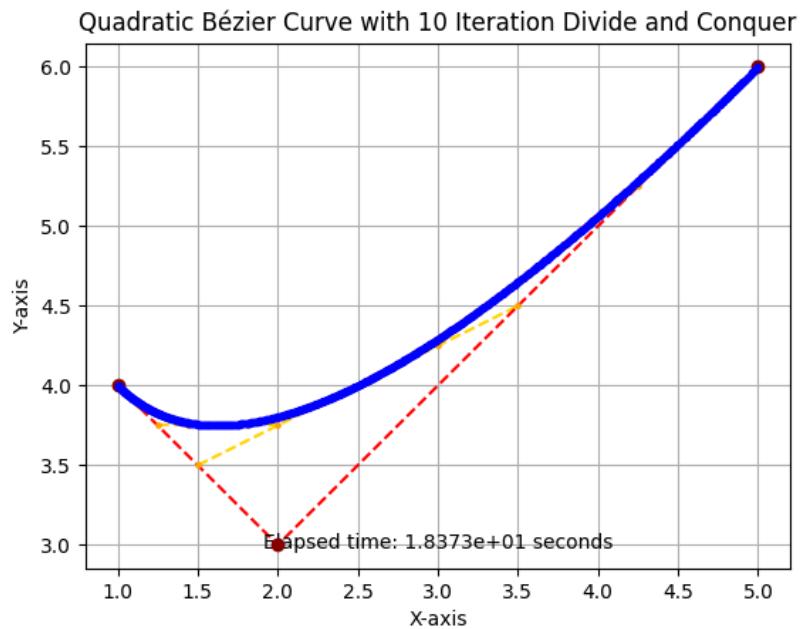
Gambar 4.12. Input dan Output Pengujian IV

4.2.5. Pengujian V (10 Iterasi)

Titik Koordinat: $P_0(1, 4)$, $P_1(2, 3)$, $P_2(5, 6)$



Gambar 4.13. Pengujian V dengan Algoritma *Brute Force*



Gambar 4.14. Pengujian V dengan Algoritma *Divide and Conquer*

- Perbandingan Algoritma Brute Force dengan Algoritma Divide and Conquer

```
===== MENU =====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

Pilih menu (1/2/3/4): 3

Masukkan titik koordinat dipisahkan dengan koma (,)
Masukkan titik kontrol ke-1 (P0): 1,4
Masukkan titik kontrol ke-2 (P1): 2,3
Masukkan titik kontrol ke-3 (P2): 5,6

Masukkan jumlah iterasi: 10

===== Membandingkan Algoritma =====
Brute Force
Elapsed time: 2.6979e-03 seconds

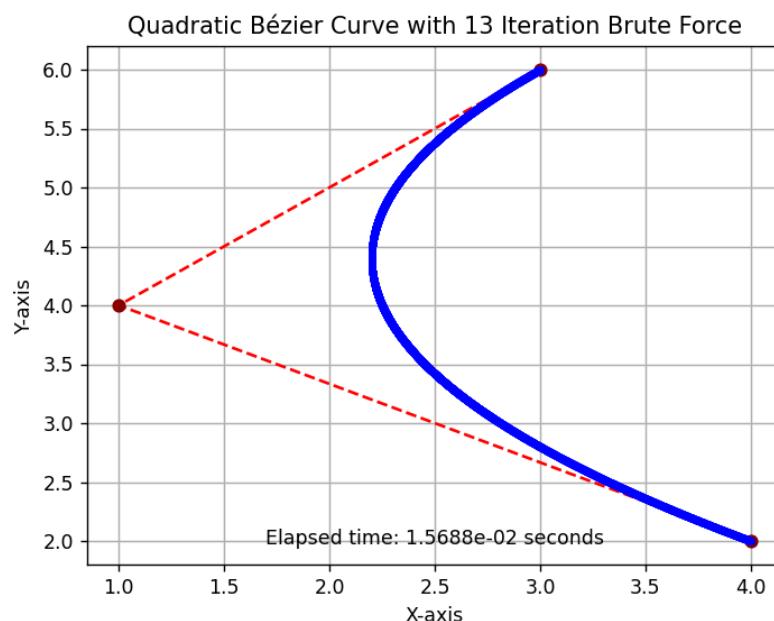
Divide and Conquer
Elapsed time: 5.1932e+00 seconds

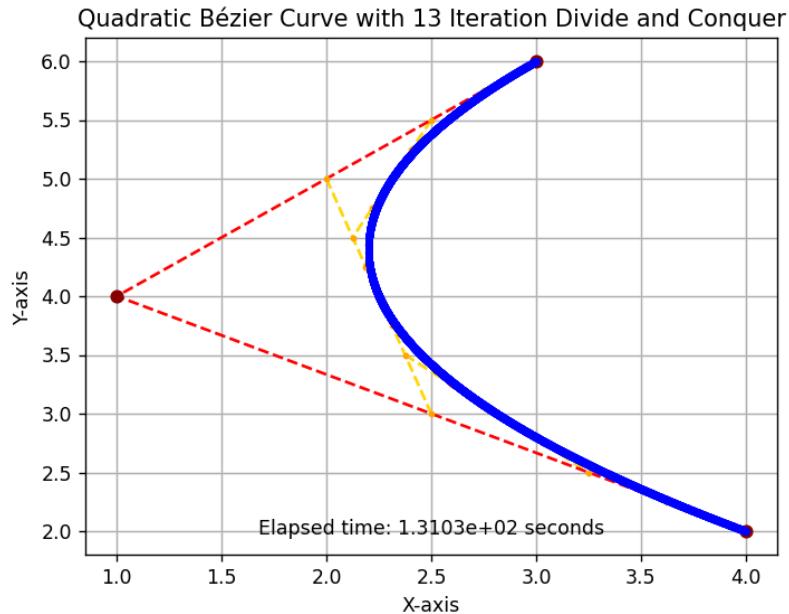
===== Analisis =====
Pembentukan kurva Bézier lebih cepat dengan Algoritma Brute Force.
Total points: 1025
```

Gambar 4.15. Input dan Output Pengujian V

4.2.6. Pengujian VI (13 Iterasi)

Titik Koordinat: $P_0(3, 6)$, $P_1(1, 4)$, $P_2(4, 2)$



Gambar 4.16. Pengujian VI dengan Algoritma *Brute Force***Gambar 4.17.** Pengujian VI dengan Algoritma *Divide and Conquer*

- Perbandingan Algoritma Brute Force dengan Algoritma Divide and Conquer

```

===== MENU =====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

Pilih menu (1/2/3/4): 3

Masukkan titik koordinat dipisahkan dengan koma (, )
Masukkan titik kontrol ke-1 (P0): 3,6
Masukkan titik kontrol ke-2 (P1): 1,4
Masukkan titik kontrol ke-3 (P2): 4,2

Masukkan jumlah iterasi: 13

===== Membandingkan Algoritma =====
Brute Force
Elapsed time: 1.3274e-02 seconds

Divide and Conquer
Elapsed time: 8.5039e-03 seconds

===== Analisis =====
Pembentukan kurva Bézier lebih cepat dengan Algoritma Divide and Conquer dengan selisih waktu eksekusi
0.0004770517349243164
Total points: 8193

```

Gambar 4.18. Input dan Output Pengujian VI

4.3. Uji Coba Kurva Bézier Berorde-N

4.3.1. Pengujian Berorde-4

```
===== MENU =====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

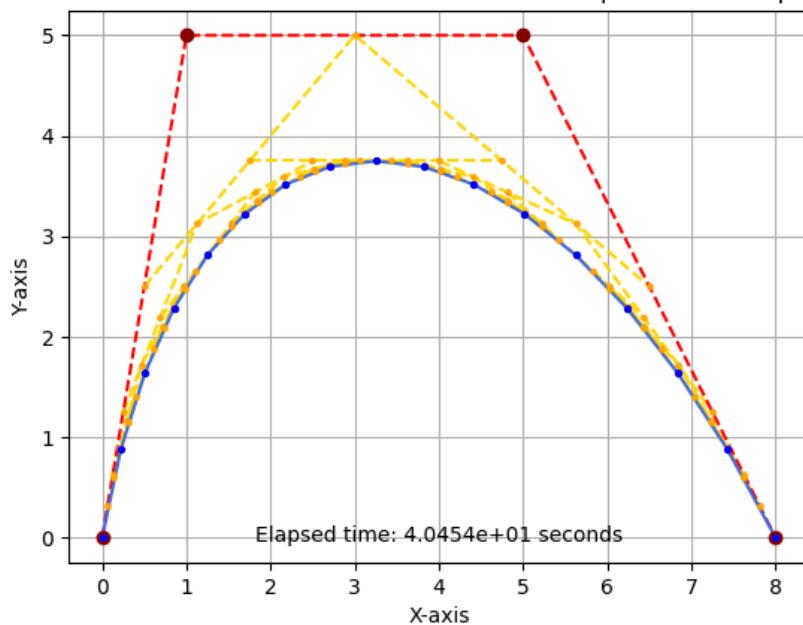
Pilih menu (1/2/3/4): 4
Jumlah titik yang akan dimasukkan: 4

Masukkan titik koordinat dipisahkan dengan koma (,)
Masukkan titik kontrol ke-1 (P0): 0,0
Masukkan titik kontrol ke-2 (P1): 1,5
Masukkan titik kontrol ke-3 (P2): 5,5
Masukkan titik kontrol ke-4 (P3): 8,0

Masukkan jumlah iterasi: 4
2024-03-17 17:56:29.394 Python[54826:3429018] WARNING
```

Gambar 4.19. Input Pengujian Berorde-4

Bézier Curve with 4 Iteration Divide and Conquer with Multipoint



Gambar 4.20. Visualisasi Pengujian Berorde-4

4.3.2. Pengujian Berorde-5

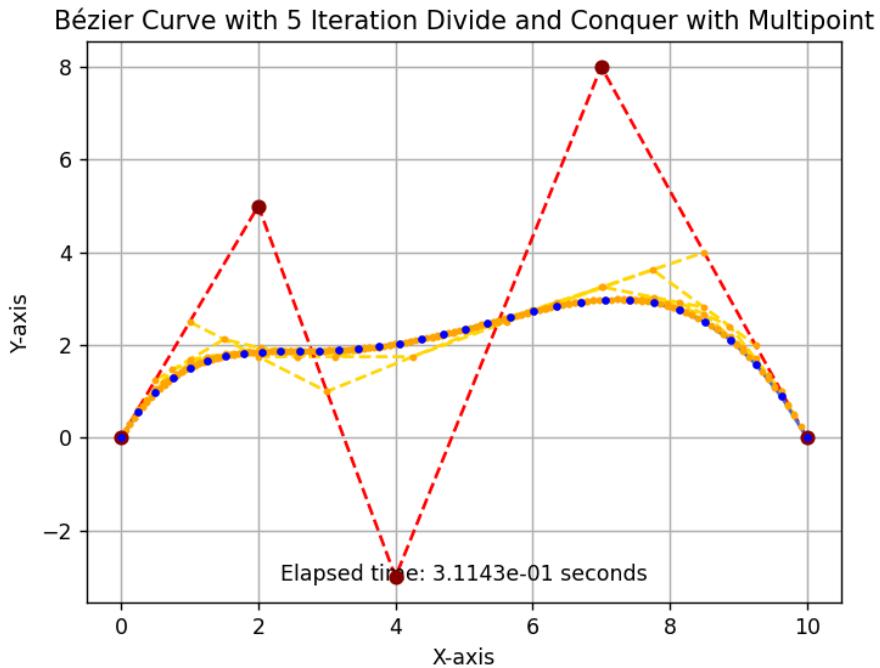
```
===== MENU =====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

Pilih menu (1/2/3/4): 4
Jumlah titik yang akan dimasukkan: 5

Masukkan titik koordinat dipisahkan dengan koma (,)
Masukkan titik kontrol ke-1 (P0): 0,0
Masukkan titik kontrol ke-2 (P1): 2,5
Masukkan titik kontrol ke-3 (P2): 4,-3
Masukkan titik kontrol ke-4 (P3): 7,8
Masukkan titik kontrol ke-5 (P4): 10,0

Masukkan jumlah iterasi: 5
```

Gambar 4.21. Input Pengujian Berorde-5



Gambar 4.22. Visualisasi Pengujian Berorde-5

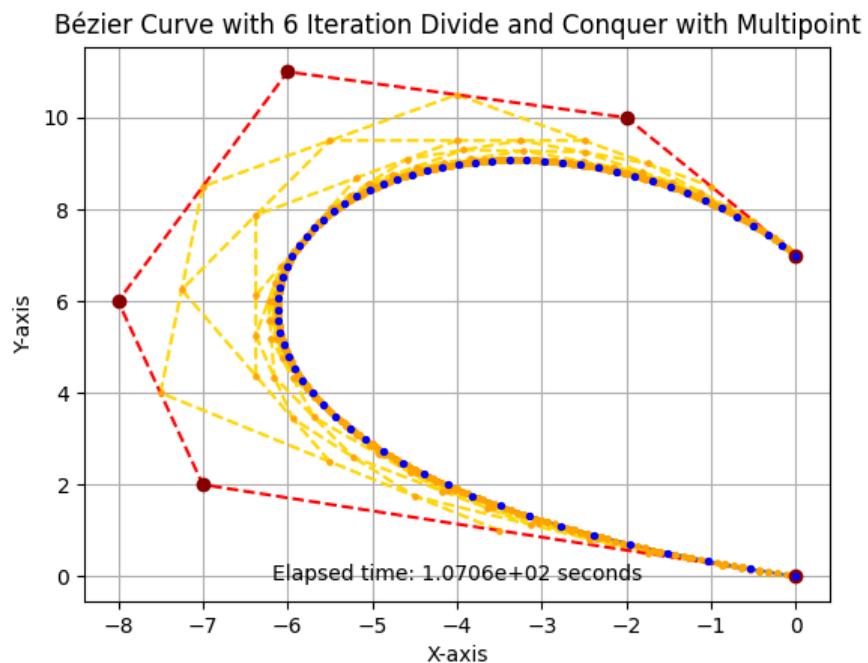
4.3.3. Pengujian Berorde-6

```
===== MENU =====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

Pilih menu (1/2/3/4): 4
Jumlah titik yang akan dimasukkan: 6

Masukkan titik koordinat dipisahkan dengan koma (, )
Masukkan titik kontrol ke-1 (P0): 0,0
Masukkan titik kontrol ke-2 (P1): -7,2
Masukkan titik kontrol ke-3 (P2): -8,6
Masukkan titik kontrol ke-4 (P3): -6,11
Masukkan titik kontrol ke-5 (P4): -2,10
Masukkan titik kontrol ke-6 (P5): 0,7

Masukkan jumlah iterasi: 6
2024-03-17 18:03:37.713 Python[57439:3443248] WARNING:
```

Gambar 4.23. Input Pengujian Berorde-6**Gambar 4.24.** Visualisasi Pengujian Berorde-6

4.3.4. Pengujian Berorde-9

```
===== MENU =====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

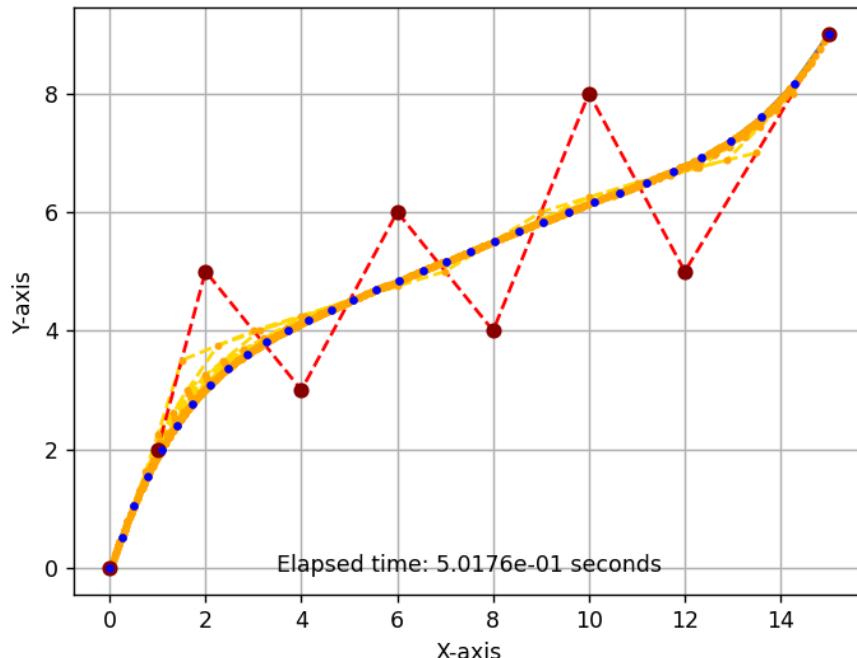
Pilih menu (1/2/3/4): 4
Jumlah titik yang akan dimasukkan: 9

Masukkan titik koordinat dipisahkan dengan koma (,)
Masukkan titik kontrol ke-1 (P0): 0,0
Masukkan titik kontrol ke-2 (P1): 1,2
Masukkan titik kontrol ke-3 (P2): 2,5
Masukkan titik kontrol ke-4 (P3): 4,3
Masukkan titik kontrol ke-5 (P4): 6,6
Masukkan titik kontrol ke-6 (P5): 8,4
Masukkan titik kontrol ke-7 (P6): 10,8
Masukkan titik kontrol ke-8 (P7): 12,5
Masukkan titik kontrol ke-9 (P8): 15,9

Masukkan jumlah iterasi: 5
```

Gambar 4.25. Input Pengujian Berorde-9

Bézier Curve with 5 Iteration Divide and Conquer with Multipoint



Gambar 4.26. Visualisasi Pengujian Berorde-9

4.3.5. Pengujian Berorde-15

```
===== MENU =====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

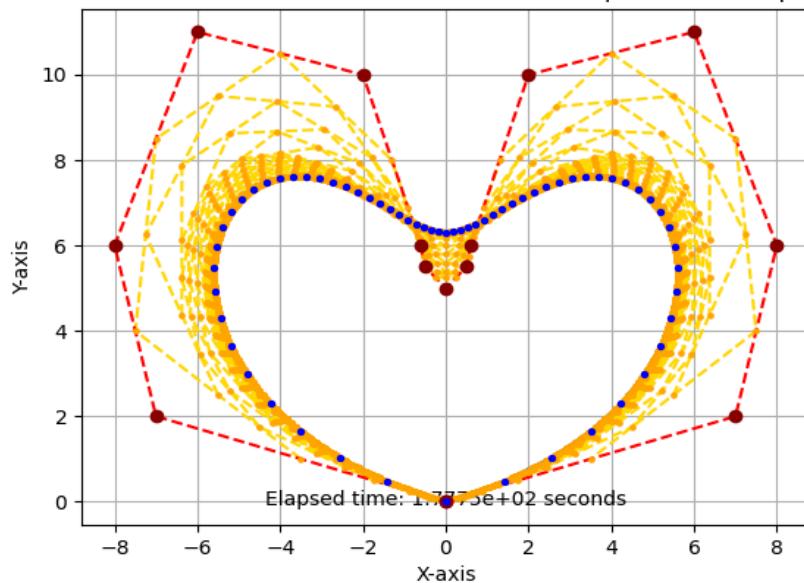
Pilih menu (1/2/3/4): 4
Jumlah titik yang akan dimasukkan: 15

Masukkan titik koordinat dipisahkan dengan koma (,)
Masukkan titik kontrol ke-1 (P0): 0,0
Masukkan titik kontrol ke-2 (P1): -7,2
Masukkan titik kontrol ke-3 (P2): -8,6
Masukkan titik kontrol ke-4 (P3): -6,11
Masukkan titik kontrol ke-5 (P4): -2,10
Masukkan titik kontrol ke-6 (P5): -0.6,6
Masukkan titik kontrol ke-7 (P6): -0.5,5.5
Masukkan titik kontrol ke-8 (P7): 0,5
Masukkan titik kontrol ke-9 (P8): 0.5,5.5
Masukkan titik kontrol ke-10 (P9): 0.6,6
Masukkan titik kontrol ke-11 (P10): 2,10
Masukkan titik kontrol ke-12 (P11): 6,11
Masukkan titik kontrol ke-13 (P12): 8,6
Masukkan titik kontrol ke-14 (P13): 7,2
Masukkan titik kontrol ke-15 (P14): 0,0

Masukkan jumlah iterasi: 6
2024-03-17 18:13:45.502 Python[61392:3464436] WARNING:
```

Gambar 4.27. Input Pengujian Berorde-15

Bézier Curve with 6 Iteration Divide and Conquer with Multipoint

**Gambar 4.28.** Visualisasi Pengujian Berorde-15

4.3.6. Pengujian Berorde-20

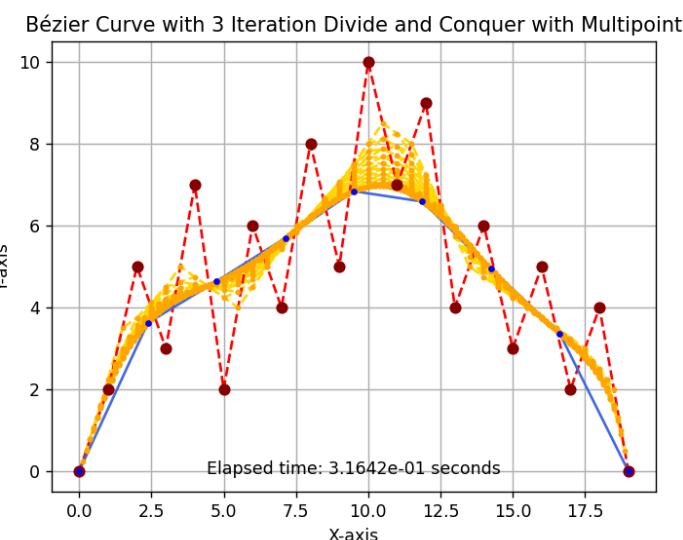
```
===== MENU =====
1. Quadratic Bézier Curve (Brute Force)
2. Quadratic Bézier Curve (Divide and Conquer)
3. Comparing Algorithm
4. Multi Bézier Curve (Divide and Conquer)

Pilih menu (1/2/3/4): 4
Jumlah titik yang akan dimasukkan: 20

Masukkan titik koordinat dipisahkan dengan koma (,)
Masukkan titik kontrol ke-1 (P0): 0,0
Masukkan titik kontrol ke-2 (P1): 1,2
Masukkan titik kontrol ke-3 (P2): 2,5
Masukkan titik kontrol ke-4 (P3): 3,3
Masukkan titik kontrol ke-5 (P4): 4,7
Masukkan titik kontrol ke-6 (P5): 5,2
Masukkan titik kontrol ke-7 (P6): 6,6
Masukkan titik kontrol ke-8 (P7): 7,4
Masukkan titik kontrol ke-9 (P8): 8,8
Masukkan titik kontrol ke-10 (P9): 9,5
Masukkan titik kontrol ke-11 (P10): 10,10
Masukkan titik kontrol ke-12 (P11): 11,7
Masukkan titik kontrol ke-13 (P12): 12,9
Masukkan titik kontrol ke-14 (P13): 13,4
Masukkan titik kontrol ke-15 (P14): 14,6
Masukkan titik kontrol ke-16 (P15): 15,3
Masukkan titik kontrol ke-17 (P16): 16,5
Masukkan titik kontrol ke-18 (P17): 17,2
Masukkan titik kontrol ke-19 (P18): 18,4
Masukkan titik kontrol ke-20 (P19): 19,0

Masukkan jumlah iterasi: 3
```

Gambar 4.29. Input Pengujian Berorde-20



Gambar 4.30. Visualisasi Pengujian Berorde-20

4.4. Uji Coba Lainnya

Uji coba kali ini dilakukan untuk membandingkan waktu eksekusi antara algoritma Brute Force dan algoritma Divide and Conquer. Pengujian ini dilakukan dalam perangkat yang sama dan visualisasi pada kedua algoritma sama-sama tidak dilakukan. Tetapi tetap menggunakan fungsi yang sama dengan uji coba pada sub-bab 4.2 sehingga dapat dipastikan kebenaran algoritmanya. Dengan demikian, variabel yang menjadi variabel bebas atau pembanding hanya pada perbedaan algoritmanya dan diusahakan tidak dipengaruhi variabel-variabel lain. Berikut hasilnya:

Tabel 4.3. Tabel Uji Coba Kurva Bézier Kuadratik dengan 2-25 Iterasi

No	Iterasi dan Titik Kontrol	Waktu eksekusi (detik)		Selisih (detik)
		Divide and Conquer	Brute Force	
1	Iterasi: 2 Titik kontrol: (2,5), (6,9), (9,3)	0.0000e+00	0.0000e+00	0
2	Iterasi: 3 Titik kontrol: (3,7), (5,4), (8,9)	0.0000e+00	0.0000e+00	0
3	Iterasi: 5 Titik kontrol: (1,-5), (-4,2), (7,-6)	0.0000e+00	0.0000e+00	0
4	Iterasi: 7 Titik kontrol: (0,3), (5,7), (9,1)	5.3477e-04	1.1044e-03	0.0005695819 854736328
5	Iterasi: 10 Titik kontrol: (-2, -6), (6, -4), (8, -8)	2.0001e-03	4.2186e-03	0.0022184848 78540039
6	Iterasi: 15 Titik kontrol: (-2,4), (6, -8), (9,3)	8.2878e-02	1.2174e-01	0.0388593673 7060547
7	Iterasi: 20 Titik kontrol: (1,5), (4,2), (7,6)	4.1717e+00	4.1679e+00	0.0037982463 83666992

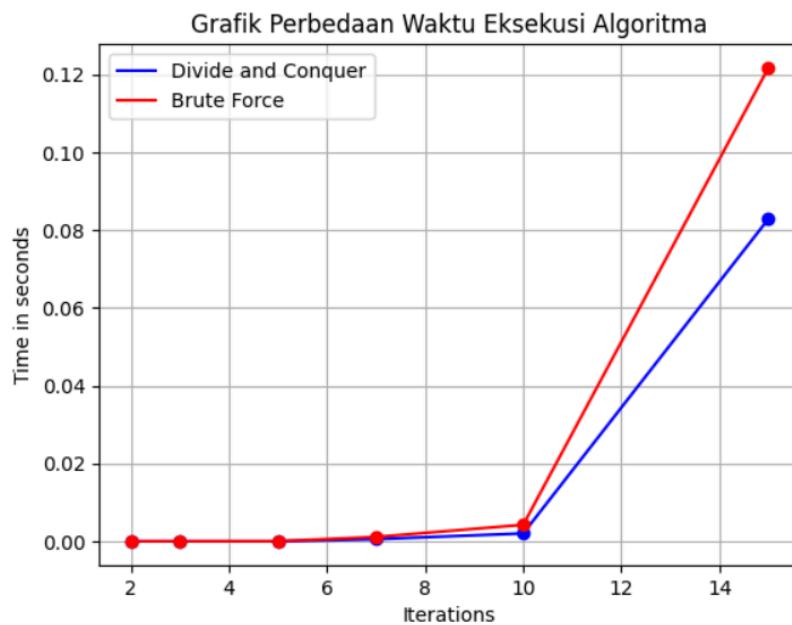
8	Iterasi: 25 Titik kontrol: (1,1), (5,5), (7,1)	1.7598e+02	1.4195e+02	34.035550832 74841
---	--	------------	------------	-----------------------

Dokumentasi hasil uji coba: [Dokumentasi Uji Coba Kurva Bézier](#)

4.5. Analisis Hasil Uji Coba

Berdasarkan hasil uji coba yang telah dibahas pada bagian sebelumnya, dapat dilihat bahwa baik algoritma *Divide and Conquer* maupun *Brute Force* menghasilkan solusi kurva Bézier yang sama. Jumlah titik yang dihasilkan pada kurva Bézier pun juga sama. Hal ini menunjukkan bahwa kedua metode algoritma benar-benar dapat membangun kurva Bézier. Dari uji coba juga terlihat bahwa semakin banyak iterasi yang dilakukan, semakin banyak pula waktu yang dibutuhkan untuk memproses. Terbukti bahwa jumlah iterasi memengaruhi kompleksitas algoritma keduanya.

Terdapat perbedaan waktu eksekusi pada iterasi yang variatif di uji coba lainnya. Perbandingan tersebut dapat terlihat pada grafik di bawah ini. Ketika iterasi sejumlah 2-15 kali berdasarkan uji coba pada 4.4:

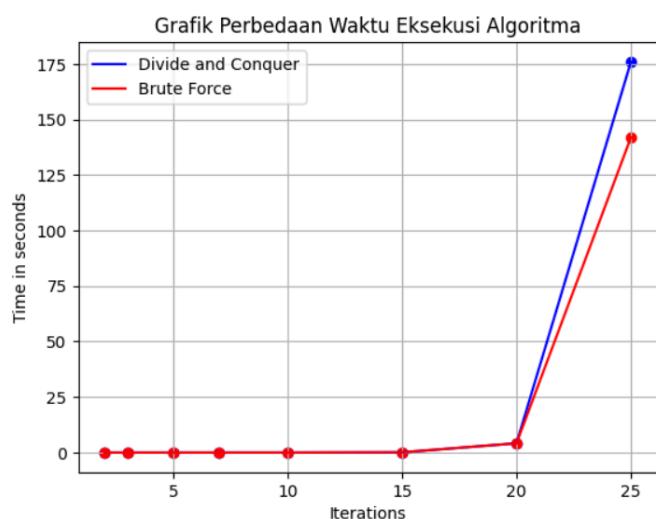


Gambar 4.31. Grafik Waktu Eksekusi Algoritma dengan Iterasi 2-15

Ketika jumlah iterasi yang dilakukan sedikit, berkisar 2-5 iterasi, waktu eksekusi kedua algoritma tidak menunjukkan perbedaan. Terlihat pada selisih waktu yang bernilai nol detik. Hal tersebut terjadi karena memang kompleksitas algoritma secara *Big O Notation* bernilai sama yaitu $O(2^n)$ dengan n jumlah iterasi yang juga telah dibahas pada bagian implementasi program.

Mulai terdapat perbedaan ketika jumlah iterasi bernilai 7-15, waktu eksekusi algoritma *Divide and Conquer* berjalan lebih cepat. Meskipun demikian, selisih waktu yang dihasilkan sangat sedikit, kurang dari 0.1 detik. Berdasarkan pengamatan implementasi program, hal tersebut dapat terjadi karena perbedaan jumlah operasi perhitungan setiap iterasi dari kedua algoritma. Pada algoritma *Brute Force* terdapat 18 operator aritmatika dasar (tambah, kurang, kali) dan 4 operator aritmatika lanjutan (perpangkatan) setiap pencarian titik. Sedangkan pada algoritma *Divide and Conquer* hanya membutuhkan 12 operator aritmatika dasar (tambah, bagi) untuk setiap pencarian titik. Ketika iterasi semakin banyak dan menyebabkan pencarian titik dilakukan semakin sering, perbedaan jumlah operasi ini berpengaruh. Namun, perlu diingat juga bahwa sebenarnya waktu yang dibutuhkan untuk perhitungan aritmatika tersebut tidaklah terlalu banyak. Oleh karena itu selisih perbedaan waktu juga sangat tipis.

Ketika iterasi dilakukan hingga 20 kali, uji coba 4.4 memberikan hasil grafik berikut:



Gambar 4.32. Grafik Waktu Eksekusi Algoritma dengan Iterasi 2-25

Pada jumlah iterasi 15-20 kali algoritma *Brute Force* berjalan lebih cepat. Bahkan, selisih waktu eksekusi pada 20 kali iterasi cukup besar hingga sekitar 34 detik. Berdasarkan pengamatan implementasi program, hal ini dapat terjadi karena adanya pemanggilan rekursif pada algoritma *Divide and Conquer*. Pemanggilan fungsi rekursif dapat menyebabkan adanya kerumitan dalam memanajemen tumpukan (*stack*) dan *overhead*. Jumlah dilakukannya rekursi yang tumbuh secara eksponensial terhadap jumlah iterasi ini, tentu akan sangat berdampak ketika nilai iterasi cukup tinggi, terlihat dari tingginya selisih waktu eksekusi yang terjadi. Sehingga, dalam hal ini algoritma *Brute Force* dapat berjalan lebih cepat karena perhitungan dilakukan secara langsung tanpa rekursif.

Dari pengamatan dan analisis tersebut, didapatkan kesamaan antara kedua algoritma, yaitu kompleksitas algoritma yang tumbuh secara eksponensial terhadap jumlah iterasi atau $O(2^n)$ dengan n merupakan jumlah iterasi. Tetapi hal tersebut tidak memperhatikan adanya faktor konstanta yang juga berpengaruh terhadap waktu eksekusi, terutama ketika jumlah iterasi yang dilakukan semakin banyak. Faktor konstanta berupa banyaknya jumlah operasi perhitungan, pemanggilan fungsi rekursif, serta manajemen tumpukan (*stack*) dan memori yang dialami kedua algoritma menyebabkan adanya selisih perbedaan waktu eksekusi. Algoritma *Divide and Conquer* yang mendapatkan faktor konstanta lebih tinggi membutuhkan waktu yang lebih lama ketika jumlah iterasi semakin tinggi.

Kode pembuatan grafik  Grafik Algoritma Bezier Curve.ipynb

Pada uji coba kurva Bézier berorde- n jumlah titik masukan dapat bervariasi. Berdasarkan hasil uji coba terlihat bahwa jumlah titik atau tingkat orde yang dimasukkan juga memengaruhi waktu eksekusi program. Contohnya seperti pada pengujian berorde 5 dan 9 memiliki iterasi yang sama yaitu sebanyak 5 kali. Namun, waktu untuk berorde 9 berjalan lebih lama. Begitu pula dengan pengujian orde 6 dan 15. Dari pengujian tersebut didapatkan kesimpulan bahwa semakin banyak titik yang dimasukkan atau semakin tinggi tingkatan orde, maka kompleksitas algoritma semakin tinggi dan waktu eksekusi semakin lama.

Jika ditinjau dari implementasi program masukan titik kontrol memengaruhi seberapa banyak perulangan mencari titik tengah untuk setiap iterasinya. Sehingga kompleksitas algoritma untuk *Brute Force* adalah $(m \cdot 2^n)$ dengan m adalah jumlah titik kontrol masukan.

Berbeda dengan algoritma *Divide and Conquer* yang nilai m -nya konstan yaitu $m = 3$, dapat dibilang angka konstanta tersebut cukup kecil. Sedangkan jika nilai m yang sangat besar tentu saja akan sangat berpengaruh pada kompleksitasnya. Oleh karena itu, baik secara uji coba maupun analisis kode implementasi menunjukkan bahwa kompleksitas algoritma dan waktu eksekusi dipengaruhi oleh jumlah titik kontrol dan jumlah iterasi dengan hubungan berbanding lurus.

BAB V

PENUTUP

5.1. Kesimpulan

Membangun Kurva Bézier dapat menggunakan algoritma *Brute Force* maupun *Divide and Conquer*. Dari hasil analisis dan implementasi menggunakan kedua algoritma tersebut, didapat bahwa pembuatan kurva bézier dengan algoritma *Divide and Conquer* memiliki performa *execution time* yang lebih cepat dibandingkan solusi dengan algoritma titik tengah berbasis *Brute Force* pada jumlah iterasi yang lebih kecil dari 15. Namun, untuk iterasi yang lebih besar *Brute Force* lebih unggul dibandingkan *Divide and Conquer*. Perbedaan performa dari kedua solusi tampak pada pengujian kami, dan perbedaan ini bersifat ekstrim ketika jumlah titik banyak. Sehingga, dapat disimpulkan bahwa baik secara uji coba maupun analisis kode implementasi menunjukkan bahwa kompleksitas algoritma dan waktu eksekusi dipengaruhi oleh jumlah titik kontrol dan jumlah iterasi dengan hubungan berbanding lurus.

5.2. Saran

- Stoplah ngasih tubes mulu, lagi puasa juga (Diana)
- Untuk memperjelas deskripsi dan penggunaan istilah dalam spesifikasi bonus, disarankan untuk menggunakan istilah "animasi" daripada "visualisasi proses".

DAFTAR PUSTAKA

- [1] Munir, R. (2020). *informatika.stei.itb.ac.id*, “Strategi Algoritma: Aplikasi Divide and Conquer”. Diakses Selasa, 12 Maret 2024. Dilansir dari Laman Rinaldi Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Aplikasi-Divide-and-Conquer-2020.pdf>
- [2] Bandara, R. *codeproject.com*, “Midpoint Algorithm (Divide and Conquer Method) for Drawing a Simple Bezier Curve”. Diakses Selasa, 12 Maret 2024. Dilansir dari Laman <https://www.codeproject.com/Articles/223159/Midpoint-Algorithm-Divide-and-Conquer-Method-for-D>
- [3] Munir, R. (2024). *informatika.stei.itb.ac.id*, “Strategi Algoritma: Algoritma Divide and Conquer Bagian 1”. Diakses Selasa, 12 Maret 2024. Dilansir dari Laman Rinaldi Munir: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf)
- [4] Munir, R. (2024). *informatika.stei.itb.ac.id*, “Strategi Algoritma: Algoritma Divide and Conquer Bagian 2”. Diakses Selasa, 12 Maret 2024. Dilansir dari Laman Rinaldi Munir: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf)
- [5] Munir, R. (2024). *informatika.stei.itb.ac.id*, “Strategi Algoritma: Algoritma Divide and Conquer Bagian 3”. Diakses Selasa, 12 Maret 2024. Dilansir dari Laman Rinaldi Munir: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf)
- [6] Munir, R. (2024). *informatika.stei.itb.ac.id*, “Strategi Algoritma: Algoritma Divide and Conquer Bagian 4”. Diakses Selasa, 12 Maret 2024. Dilansir dari Laman Rinaldi Munir: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-\(2024\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian4.pdf)

LAMPIRAN**Lampiran 1 Link Repository**

Repository untuk source code dan hasil kompilasi program ini dapat diakses melalui tautan berikut: http://github.com/dianatrihy/Tucil2_13522069_13522104

Lampiran 2 Tabel *Checklist Poin*

No.	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.	✓	