IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Laporan Tugas Kecil

Disusun untuk memenuhi tugas besar mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2023/2024

Oleh

Shabrina Maharani 13522134

Muhammad Davis Adhipramana 13522157

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024

DAFTAR ISI

BAB 1 DESKRIPSI MASALAH

1.1 Kurva Beziér

Kurva Bézier adalah jenis kurva yang digunakan untuk membuat desain grafis, animasi, dan produk manufaktur. Cara membuatnya cukup sederhana, yaitu dengan menghubungkan beberapa titik kontrol untuk menentukan bentuk dan arah kurva. Kurva Bézier memiliki banyak aplikasi praktis dalam kehidupan sehari-hari, seperti dalam pembuatan *pen tool*, animasi yang halus dan realistis, desain produk yang kompleks dan akurat, serta pembuatan font yang indah dan unik.

Keunggulan utama dari kurva Bézier adalah fleksibilitasnya dalam mengubah dan memanipulasi desain dengan presisi, sesuai dengan kebutuhan pengguna. Kurva Bézier didefinisikan oleh satu set titik kontrol, dengan titik awal dan akhir menjadi ujung dari kurva. Titik-titik kontrol di antara titik kontrol pertama dan terakhir dapat mengatur bentuk dan arah kurva, tetapi tidak selalu terletak pada kurva yang dihasilkan.

Untuk membentuk kurva Bézier, misalnya, diberikan dua titik kontrol P0 dan P1, kurva yang dihasilkan akan menjadi garis lurus antara kedua titik tersebut, yang disebut kurva Bézier linier. Jika terdapat titik lain yang berada pada garis yang dibentuk oleh kedua titik kontrol tersebut, posisinya dapat dihitung menggunakan persamaan parametrik tertentu. Ini memberikan fleksibilitas dalam mengontrol bentuk dan posisi kurva dengan memanipulasi titik kontrolnya. Persamaan parametrik tersebut dapat dilihat sebagai berikut.

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

Dalam fungsi kurva Bézier linier, nilai parameter t menentukan seberapa jauh titik B(t) dari P0 ke P1. Contohnya jika t = 0.5 maka B(t) adalah setengah jalan dari titik P0 ke P1, sehingga setiap nilai t dari 0 hingga 1 akan menghasilkan garis lurus dari P0 ke P1.

Jika ditambahkan titik baru, P2, sehingga P0 dan P2 menjadi titik awal dan akhir, dan P1 menjadi titik kontrol di antara P0 dan P2. Dengan menempatkan titik Q1 pada garis yang menghubungkan P1 dan P2, dan membentuk kurva Bézier linier yang berbeda dengan kurva yang melewati Q0, titik baru dan Q1, membentuk kurva Bézier kuadratik terhadap P0 dan P2. Berikut adalah persamaannya.

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$
$$R_0 = B(t) = (1-t)Q_0 + tQ_1, \qquad t \in [0,1]$$

Setelah itu akan dimasukkan substitusi nilai Q0 dan Q1, maka akan diperoleh persamaan sebagai berikut.

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

Proses ini juga berlaku untuk jumlah titik yang lebih dari tiga, di mana empat titik akan menghasilkan kurva Bézier kubik, lima titik akan menghasilkan kurva Bézier kuartik, dan seterusnya. Persamaan untuk kurva Bézier kubik dan kuartik dapat dibentuk dengan menggunakan prosedur yang sama seperti sebelumnya.

Namun, persamaan yang dihasilkan menjadi sangat panjang dan semakin rumit seiring bertambahnya jumlah titik. Oleh karena itu, untuk efisiensi pembuatan kurva Bézier yang bermanfaat ini, Anda diminta untuk mengimplementasikan pembuatan kurva Bézier menggunakan algoritma titik tengah berbasis divide and conquer.

BAB 2 LANDASAN TEORI

2.1 Algoritma Brute Force

Algoritma *Brute Force* adalah sebuah metode algoritma dengan pendekatan yang *straightforward* (lempang) untuk menyelesaikan suatu persoalan. Algoritma *brute force* biasanya didasarkan pada definisi atau konsep yang dilibatkan dan pernyataan pada persoalan (*problem statement*). Algoritma *brute force* memiliki ciri-ciri untuk memecahkan persoalan dengan sangat sederhana, langsung, langkah penyelesaiannya yang jelas (obviuos way) dan dilakukan dengan prinsip *just do it!* atau *just solve it!*.

Pada umumnya, algoritma *brute force* bukan merupakan algoritma yang cerdas dan mangkus. Hal ini dikarenakan algoritma *brute force* membutuhkan waktu yang cukup lama dan komputasi dengan volume yang besar dalam menyelesaiakan persoalan. Kata *"force"* dalam *brute force* memiliki makna konotasi yaitu tenaga dibandingkan otak. Terkadang, algoritma *brute force* ini disebut juga dengan *naive algorithm* (algoritma naif).

Algoritma *brute force* lebih sesuai digunakan untuk persoalan dengan ukuran masukannya kecil dengan pertimbangan algoritma *brute force* cocok dipakai untuk persoalan yang sederhana dan implementasinya mudah. Algoritma *brute force* juga marak digunakan untuk menjadi basis pembanding dengan algoritma lain yang lebih cerdas dan mangkus. Namun, walaupun algoritma *brute force* bukan merupakan metode menyelesaikan permasalahan dengan mangkus, semua persoalan hampir dapat diselesaikan dengan algoritma *brute force*. Kecil kemungkinannya untuk menemukan persoalan yang tidak dapat diselesaikan dengan metode *brute force*. Bahkan, terdapat persoalan yang hanya mampu diselesaikan dengan algoritma *brute force*.

Algoritma *brute force* memiliki aplikabilitas yang luas dalam menyelesaikan berbagai masalah, karena sederhana dan mudah dipahami. Pendekatan ini menghasilkan

solusi yang layak untuk berbagai masalah penting, termasuk pencarian, pengurutan, pencocokan string, dan perkalian matriks. Selain itu, algoritma brute force seringkali menjadi standar dalam tugas-tugas komputasi seperti penjumlahan atau perkalian sejumlah besar bilangan, serta menemukan elemen minimum atau maksimum dalam sebuah daftar.

Meskipun memiliki kelebihan dalam aplikabilitas dan kemudahan pemahaman, algoritma brute force memiliki beberapa kelemahan yang perlu dipertimbangkan. Pertama, algoritma ini jarang menghasilkan solusi yang optimal atau efisien. Kedua, kinerjanya umumnya lambat untuk masukan berukuran besar, membuatnya tidak praktis dalam situasi di mana efisiensi waktu menjadi faktor penting. Terakhir, pendekatan brute force cenderung kurang konstruktif atau kreatif dalam memecahkan masalah, karena hanya bergantung pada enumerasi seluruh kemungkinan solusi tanpa mempertimbangkan strategi yang lebih canggih atau pintar.

1.2 Algoritma Divide dan Conquer

Divide and conquer (DC) adalah salah satu teknik algoritma yang paling penting dan dapat digunakan untuk memecahkan berbagai masalah komputasi. Algoritma Divide and Conquer adalah metode untuk menyelesaikan masalah kompleks dengan memecahkannya menjadi submasalah yang lebih kecil, menyelesaikan setiap submasalah secara independen, dan kemudian menggabungkan hasilnya untuk mendapatkan solusi keseluruhan. Pendekatan ini terdiri dari tiga langkah utama, yaitu:

1. Divide: Membagi masalah menjadi beberapa submasalah yang memiliki kemiripan persoalan dengan masalah semula, tetapi lebih kecil dalam ukuran (idealnya setiap submasalah berukuran hampir sama).
2. Conquer: Menyelesaikan setiap submasalah secara terpisah, langsung jika ukuran sudah cukup kecil, atau dengan pendekatan rekursif jika masih cukup besar.

3. Combine: Menggabungkan solusi dari setiap submasalah untuk membentuk solusi akhir dari masalah asal.

Objek dari persoalan yang dibagi terdiri dari masukan (*input*) atau *instances* persoalan yang berukuran n seperti :

- Matriks
- Eksponen
- Polinom
- *Array*
- dan sebagainya, tergantung persoalan

Setiap submasalah memiliki sifat yang sama dengan masalah asal, namun memiliki ukuran yang lebih kecil. Oleh karena itu, pendekatan Divide and Conquer lebih mudah dijelaskan menggunakan skema rekursif.

Divide-and-Conquer memiliki beberapa keuntungan. Pertama, divide-and-conquer dapat menghasilkan solusi yang cukup efisien untuk sebuah masalah. Namun, untuk menjadi efisien, kita perlu memastikan bahwa langkah pembagian dan penggabungan yang dilakukan itu efisien, dan bahwa mereka tidak membuat terlalu banyak sub-instansi. Kedua, pekerjaan dan jangka waktu untuk algoritma divide-and-conquer dapat diekspresikan sebagai bentuk persamaan matematika yang disebut sebagai rekurensi. Seringkali, rekurensi tersebut dapat dipecahkan tanpa terlalu banyak kesulitan, yang membuat analisis kerja dan waktu eksekusi dari banyak algoritma divide-and-conquer menjadi lebih sederhana dan jelas. Keuntungan terakhir, *divide-and-conquer* adalah teknik algoritma yang secara alami paralel. Kebanyakan dari kita seringkali dapat menyelesaikan sub-instansi secara paralel. Hal ini dapat menghasilkan jumlah paralelisme yang signifikan karena pada setiap tingkat dapat menciptakan lebih banyak instansi untuk diselesaikan secara paralel. Bahkan jika kita

hanya membagi instansi kita menjadi dua sub-instansi, masing-masing dari sub-instansi tersebut akan menghasilkan dua lagi sub-instansi, dan ini terus berulang

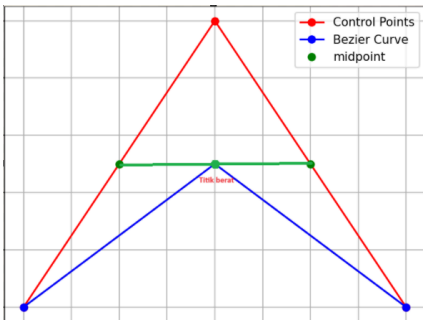
BAB 3 Analisis dan Implementasi

3.1 Analisis dan Implementasi dalam Algoritma Divide dan Conquer

3.1.1 Algoritma Divide dan Conquer untuk Tiga Titik Kontrol

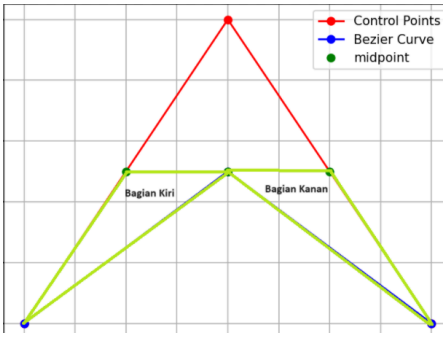
Setelah memahami algoritma *divide-and-conquer* untuk tiga titik kontrol terdapat langkah-langkah yang dapat dilakukan untuk mendapatkan hasil yang sesuai dengan konsep kurva bézier. Misalnya, terdapat 3 titik kontrol P0,P1, dan P2. Persoalan tersebut dapat diselesaikan dengan langkah - langkah pada uraian berikut.

1. Terdapat **basis** dalam kasus ini, yaitu, saat iterasi sama dengan 1 (iterasi pertama kali dilakukan) dan titik kontrol tentunya memiliki jumlah sama dengan tiga, algoritma dapat langsung menghitung titik berat dari segitiga yang terbentuk antara P0, P1, dan P2. Titik berat tersebut dapat dihitung dengan cara mencari Q0 dan Q1. Q0 merupakan titik tengah (*midpoint*) dari P0 dan P1 dan Q1 merupakan titik tengah (*midpoint*) dari P1 dan P2. Jika dilihat dari konsepnya, ini sama saja seperti melakukan rekursif dengan menggunakan konsep kurva bézier linier yaitu mencari titik tengah dari dua titik kontrol.

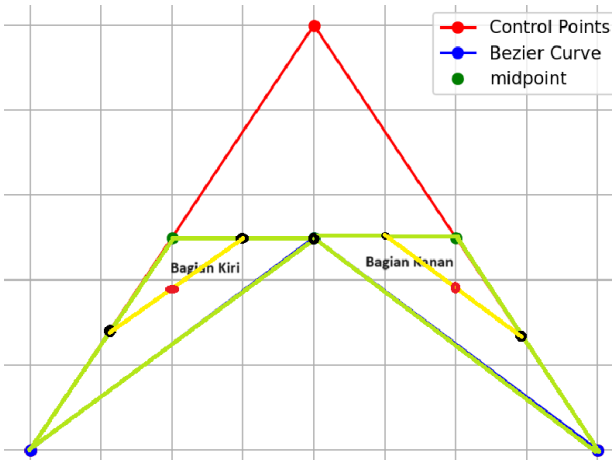


Dari langkah ini, kita akan mendapatkan 1 titik kurva yang baru. Jika iterasi hanya ingin dilakukan 1 kali maka, titik tersebut akan disambungkan dengan titik kontrol pertama dan titik kontrol terakhir. Iterasi pertama akan selalu menghasilkan bentuk yang menyerupai segitiga.

2. Jika iterasi ingin dilakukan lebih dari 1 kali, misalnya 2, maka sesuai dengan konsep *divide-and-conquer* terutama langkah pertama yaitu ***divide***, membagi persoalan menjadi beberapa upa-persoalan, algoritma akan membagi kurva menjadi dua bagian, kiri dan kanan.



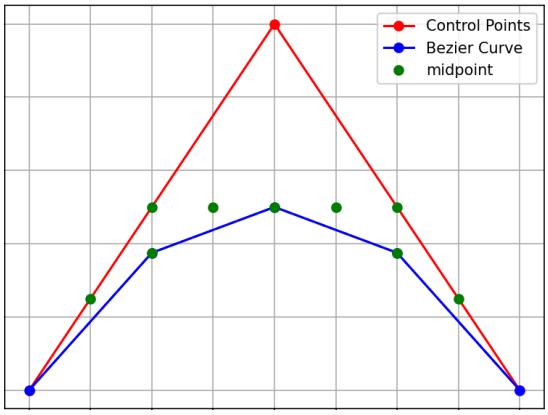
3. Kemudian, untuk setiap upa-persoalan akan diselesaikan masing-masing sesuai dengan konsep ***conquer***, penyelesaian upa-persoalan ini akan sama penyelesaiannya dengan penyelesaian persoalan yang pertama kali diselesaikan (iterasi pertama kali). Hal ini menunjukkan bahwa pada tahap ini bisa dilakukan rekursi kasus basis yaitu mencari titik berat dari segitiga yang terbentuk pada masing-masing bagian (bagian kiri dan bagian kanan).



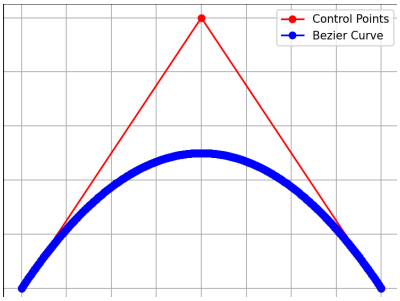
4. Dari langkah ketiga, akan didapatkan dua titik kurva baru yang dihasilkan oleh bagian kiri dan bagian kanan. Jika iterasi hanya ingin dilakukan sampai 2 kali, maka algoritma akan membutuhkan tempat penyimpanan (variabel baru atau atribut) untuk menjadi tempat sementara untuk mengurutkan titik kurva yang dihasilkan agar membentuk kurva yang sesuai. Titik yang akan pertama kali dimasukkan adalah titik yang dihasilkan oleh penyelesaian persoalan bagian kiri. Kemudian, algoritma akan memasukkan hasil titik dari iterasi pertama ke dalam tempat penyimpanan tersebut. Terakhir, titik dari penyelesaian persoalan bagian kanan yang akan dimasukkan. Dengan begitu, kurva yang belum sempurna bentuknya akan mulai terlihat menuju kurva yang semakin sempurna. Langkah ini menerapkan bagian terakhir dari algoritma yaitu ***combine***.

```
curve_points = [hasil bagian kiri, iterasi pertama, hasil bagian kanan]
```

Dari langkah ini kita akan mendapatkan bentuk kurva sebagai berikut.



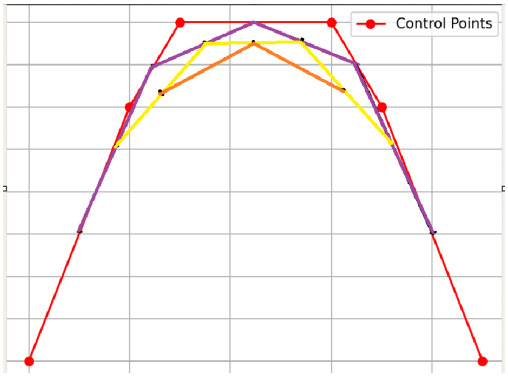
5. Apabila ingin dilakukan iterasi lanjutan, maka algoritma akan mengulangi langkah kedua sampai keempat mulai dari *divide, conquer,* dan *combine*. Dari titik berat yang dihasilkan bagian kiri akan dilakukan pembagian lagi (*divide*) dan akan melakukan rekursi dengan langkah rekursi yang berisi langkah kedua sampai keempat. Hal tersebut juga berlaku untuk bagian kanan. Semakin banyak iterasinya, kurva yang terbentuk akan semakin sempurna.



3.1.2 Algoritma Divide dan Conquer untuk n Titik Kontrol dengan n lebih dari 3

Penyelesaian kurva bezier untuk n titik kontrol dengan n lebih dari 3 juga dapat diselesaikan dengan konsep *divide and conquer*. Jika dianalisis lebih lanjut, setiap pertambahan n titik kontrol akan membentuk pola yang rekursif terhadap

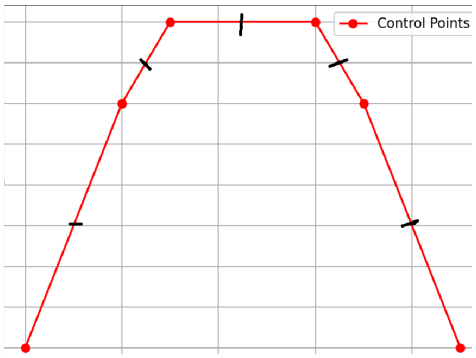
titik-titik tengah dari masing-masing titik kontrol. Ilustrasi tersebut dapat dilihat pada gambar berikut.



Dari gambar tersebut, dapat terlihat bahwa jika kita memiliki n titik kontrol, algoritma akan menjalankan algoritma n-1 titik kontrol dengan memperbarui titik kontrol nya menjadi titik tengah dari tiap titik secara rekursif sampai tersisa tiga titik. Setelah tersisa tiga titik, maka kita dapat kembali pada kasus basis dimana dari tiga kontrol poin tersebut dapat dicari titik beratnya.

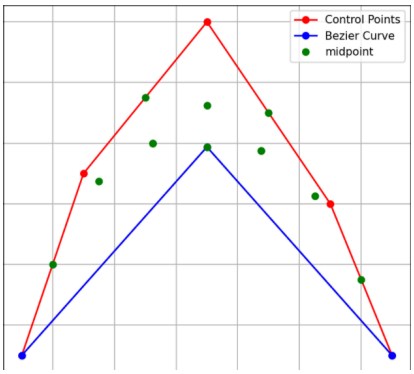
Penyelesaian persoalan tersebut dapat dilihat lebih lanjut pada uraian berikut.

1. Langkah pertama yang harus dilakukan dalam menyelesaikan persoalan kurva dengan n titik kontrol adalah mencari titik tengah dari dua titik kontrol yang tersambung. Misalnya, kita memiliki 6 titik kontrol, maka kita harus mencari titik tengah dari titik kontrol pertama dengan titik kontrol kedua, titik tengah dari titik kontrol kedua dengan titik kontrol ketiga sampai seterusnya. Kumpulan titik tengah tersebut akan menjadi sebuah titik kontrol baru yang akan dilakukan pengecekan terlebih dahulu.

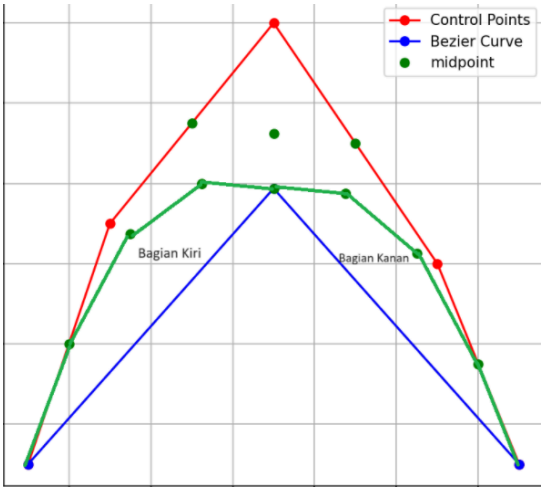


Dari gambar tersebut terdapat 5 titik kontrol baru yang akan dilakukan pengecekan.

2. Pengecekan dilakukan untuk melihat apakah titik kontrol baru yang telah dihasilkan sudah mencapai kasus **basis** dimana hanya terdapat 3 titik kontrol baru. Jika ternyata banyak titik kontrol baru belum mencapai kasus basis, program akan melakukan rekursif dengan cara mengulang kembali langkah pertama sampai banyak titik kontrol baru adalah 3.
3. Jika titik kontrol baru sudah mencapai kasus basis maka bisa dilakukan perhitungan untuk mencari titik berat dari 3 titik kontrol baru. Setelah itu, hasil perhitungan 3 titik kontrol tersebut akan dimasukkan ke dalam sebuah atribut (dalam program ini adalah self.firs_iterate) yang akan menampung hasil iterasi di kasus basis dari berbagai rekursi yang dilakukan oleh program.
4. Kemudian, apabila iterasi yang dilakukan hanya ingin sebanyak satu kali, maka program akan mengembalikan hasil dari self.first_iterate tersebut. Setelah itu, kumpulan iterasi tersebut akan dimasukkan ke dalam kumpulan titik kurva. Kemudian satu titik kurva tersebut akan disambungkan dengan titik kontrol pertama (titik kontrol dari input) dan titik kontrol terakhir. Hasil iterasi pertama akan selalu membentuk sebuah segitiga.



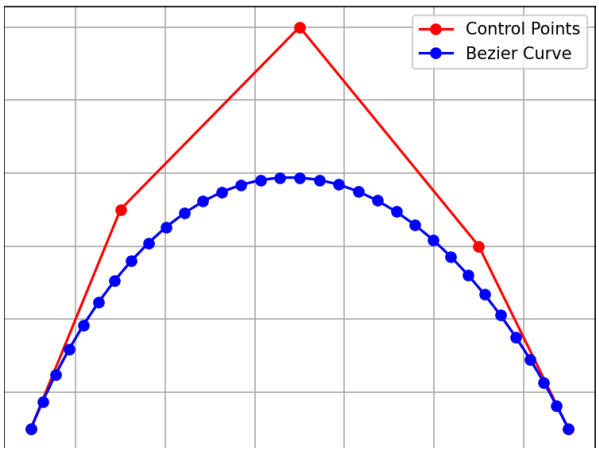
5. Jika iterasi ingin dilakukan lebih dari 1 kali, konsep ***divide*** berlaku kembali pada proses ini. Program akan membagi menjadi bagian kiri dan bagian kanan.



Gambar 3.9 Konsep Divide pada 5 Titik Kontrol

6. Setelah itu akan dilakukan tahap ***conquer*** , dimana setiap bagian akan dicari hasil dari iterasi pertama nya untuk disimpan ke dalam self.first_iterate. Pencarian hasil dari iterasi pertama itu dilakukan secara rekursif dengan mengulangi langkah 1 sampai dengan langkah 4. Rekursi akan terus berulang sesuai iterasi yang diinginkan.

7. Tahap terakhir adalah ***combine.*** Pada tahap ini, program tidak langsung mengisi hasil titik kurva (self.curve_points) dengan kumpulan hasil titik iterasi. Terdapat beberapa pengecekan terhadap pengaruh titik kontrol dengan titik hasil iterasi. Namun, pada dasarnya, program akan membagi hasil titik iterasi menjadi hasil operasi kiri dan operasi kanan, serta titik iterasi awal (self.curve_points[0]), jika tidak termasuk kasus khusus atau nilai x akan selalu berkembang, maka yang akan dimasukkan terlebih dahulu adalah titik-titik hasil operasi kiri, lalu titik iterasi pertama, dan yang paling terakhir adalah titik hasil operasi kanan. Setelah itu, titik kurva pertama akan disambungkan dengan titik kontrol pertama dan titik kurva terakhir akan disambungkan dengan titik kontrol terakhir. Dengan begitu, akan terbentuk sebuah kurva dari hasil langkah-langkah ini.



Gambar 3.9 Konsep Divide pada 5 Titik Kontrol

3.2 Analisis dan Implementasi dalam Algoritma Brute Force

3.1.2 Algoritma Brute Force untuk Tiga Titik Kontrol

Terdapat beberapa langkah yang dilakukan dalam algoritma pembuatan kurva Bézier menggunakan pendekatan brute force ini. Berikut adalah langkah-langkahnya:

1. Langkah awal dalam algoritma adalah melakukan inisialisasi dan persiapan awal. Ini termasuk membersihkan daftar titik kurva yang mungkin sudah ada sebelumnya dan menetapkan titik kontrol yang akan digunakan untuk membentuk kurva Bézier.
2. Algoritma kemudian menghitung jumlah titik yang akan digunakan untuk kurva Bézier berdasarkan nilai iterasi yang diberikan. Jumlah titik ini sangat memengaruhi kehalusan kurva Bézier yang akan dihasilkan, dengan nilai iterasi yang lebih tinggi menghasilkan kurva yang lebih halus.
3. Setelah menentukan jumlah titik, algoritma membuat basis nilai t awal menggunakan fungsi linspace dari NumPy. Nilai-nilai ini nantinya akan digunakan untuk menghitung posisi titik kurva pada setiap iterasi. Perhitungan nilai t dipengaruhi oleh iterasi yang akan dilakukan. Misalnya, iterasi sama dengan 1 maka iterasi akan membagi titik menjadi 3, t = 0, 0.5, 1. Hal ini terjadi karena t menggambarkan seberapa jauh B(t) dari P0 ke P1. Jumlah titik akan meningkat secara eksponensial yaitu 2 pangkat n + 1. Oleh karena itu, t akan sangat berpengaruh pada perhitungan titik kurva.
4. Algoritma melakukan iterasi untuk setiap nilai t dalam basis nilai t. Selama iterasi, titik-titik kurva dihitung menggunakan formula kurva Bézier (menggunakan rumus yang disediakan), yang melibatkan

interpolasi linear antara setiap pasangan titik kontrol. Ini memungkinkan algoritma untuk menghasilkan titik-titik kurva yang halus.

5. Tiap titik kurva baru yang dihasilkan selama iterasi disimpan dalam daftar titik kurva. Penting untuk memastikan bahwa tidak ada duplikat titik yang disimpan, sehingga setiap titik kurva unik.
6. Setelah menghitung seluruh titik kurva, algoritma menyusunnya untuk membentuk kurva Bézier. Proses penyusunan dilakukan dengan menghubungkan titik-titik kurva secara berurutan sesuai dengan nilai t yang dihasilkan selama iterasi.

BAB 4 SOURCE CODE PROGRAM IMPLEMENTASI

4.1 Spesifikasi Teknis Program

4.1.1 Fungsi dan Prosedur

Fungsi / Prosedur	Tujuan
Class BezierCurve dari bruteforce.py	
`def __init__(self)`	Fungsi inisialisasi untuk membuat objek BezierCurve.
`def make_bezier(self, *control_points, iterations)`	Fungsi ini bertujuan untuk membuat kurva Bézier berdasarkan titik kontrol yang diberikan serta jumlah iterasi yang ditentukan. Pertama-tama, kurva sebelumnya dibersihkan dan variabel kontrol diinisialisasi. Algoritma kemudian memeriksa apakah jumlah titik kontrol sesuai dengan yang dibutuhkan untuk kurva Bézier. Selanjutnya, jumlah titik yang akan digunakan dihitung berdasarkan

	nilai iterasi. Basis nilai t awal dibuat menggunakan linspace dari NumPy untuk perhitungan titik kurva. Selama iterasi, titik-titik kurva dihitung menggunakan formula kurva Bézier untuk interpolasi linear antara setiap pasangan titik kontrol. Setiap titik kurva baru yang dihasilkan selama iterasi disimpan dalam daftar titik kurva, memastikan bahwa tidak ada duplikat. Terakhir, titik terakhir dari kurva dimasukkan ke dalam daftar untuk memastikan kurva Bézier terbentuk secara lengkap.
`def plot_curve(self)`	Fungsi ini memiliki tujuan untuk memvisualisasikan kurva Bézier yang telah dibuat bersama dengan titik kontrolnya. Pertama-tama, titik kontrol ditampilkan sebagai titik-titik berwarna merah pada plot. Selanjutnya, kurva Bézier yang dihasilkan dari titik kontrol tersebut ditampilkan sebagai garis putus-putus dengan warna abu-abu.

	Seluruh titik kurva juga ditampilkan sebagai titik berwarna biru. Penjelasan ini membantu pengguna untuk dengan cepat memahami struktur dan karakteristik dari kurva Bézier yang telah dibuat, serta memfasilitasi analisis visual terhadap kurva tersebut.
Class BezierCurve dari beziercurve.py (metode divide and conquer)	
`def __init__(self)`	Fungsi inisialisasi untuk membuat objek BezierCurve dengan metode divide and conquer.
`def calculate_bezier_three_point(self, control_points,iterations)`	Fungsi ini bertujuan untuk menghitung titik-titik kurva Bézier berdasarkan tiga titik kontrol dan jumlah iterasi yang diberikan. Jika jumlah titik kontrol adalah tiga dan iterasi sama dengan satu, maka fungsi ini menghitung titik tengah dari ketiga titik kontrol tersebut. Namun, jika iterasi lebih dari satu,

	fungsi akan membagi kurva menjadi dua bagian, masing-masing berisi dua titik kontrol baru yang dihasilkan dari titik tengah kurva sebelumnya. Proses ini dilakukan secara rekursif hingga iterasi mencapai satu. Seluruh titik kurva yang dihasilkan dari setiap iterasi digabungkan menjadi satu daftar dan dikembalikan sebagai hasil fungsi. Dengan demikian, fungsi ini memfasilitasi pembentukan kurva Bézier yang lebih kompleks dengan membagi persoalan menjadi sub-persoalan yang lebih kecil dan menggabungkan hasilnya.
`def make_bezier_three_point(self, *control_points,iterations)`	Fungsi ini bertujuan untuk membuat kurva Bézier berdasarkan tiga titik kontrol dan jumlah iterasi yang ditentukan. Pada iterasi pertama, fungsi menghitung titik tengah dari tiga titik kontrol yang diberikan. Jika jumlah iterasi lebih dari satu, fungsi memanggil fungsi

	rekursif untuk menghasilkan semua titik kurva Bézier dari titik kontrol yang diberikan. Selanjutnya, titik kurva tersebut dibagi menjadi dua bagian, di mana setengah bagian pertama dimulai dari titik tengah dan setengah bagian kedua dimulai dari titik tengah ke titik terakhir. Titik kurva Bézier yang dihasilkan kemudian digabungkan dan dikembalikan sebagai hasil fungsi. Dengan demikian, fungsi ini memungkinkan pembentukan kurva Bézier yang lebih kompleks dengan membagi persoalan menjadi sub-persoalan yang lebih kecil dan menggabungkan hasilnya.
`def make_bezier(self, *control_points,iterations)`	Fungsi ini bertujuan untuk membuat kurva Bézier berdasarkan titik kontrol yang diberikan, dengan jumlah titik kontrol yang dapat bervariasi. Jika jumlah titik kontrol sama dengan tiga, maka fungsi akan menggunakan metode make_bezier_three_point untuk

	menghasilkan kurva Bézier. Namun, jika jumlah titik kontrol lebih dari tiga, fungsi akan menggunakan pendekatan yang lebih kompleks. Fungsi akan melakukan sejumlah langkah untuk menentukan urutan yang tepat dari titik kurva yang dihasilkan, berdasarkan pola hubungan antara titik kontrol yang diberikan. Selanjutnya, fungsi akan mengurutkan dan menggabungkan titik kurva yang dihasilkan sesuai dengan pola yang telah ditentukan. Dengan demikian, fungsi ini memungkinkan pembuatan kurva Bézier yang lebih fleksibel dan adaptif terhadap berbagai pola titik kontrol yang diberikan.
`def make_bezier_n_point(self, *control_points,iterations)`	Fungsi ini bertujuan untuk menghasilkan kurva Bézier dengan jumlah titik kontrol yang lebih dari tiga, yang memungkinkan fleksibilitas dalam menangani pola titik kontrol yang kompleks. Pada

	awalnya, fungsi ini menghitung titik tengah antara setiap pasangan titik kontrol yang diberikan, dan kemudian menggunakan titik tengah tersebut sebagai kontrol baru untuk menghasilkan kurva Bézier dengan menggunakan fungsi `make_bezier_three_point`. Jika jumlah kontrol baru yang dihasilkan masih lebih dari tiga, fungsi akan melakukan rekursi dengan mengulang proses ini hingga jumlah kontrol menjadi tiga. Selanjutnya, fungsi akan membagi kurva menjadi dua bagian dan melakukan rekursi untuk setiap bagian, sehingga menghasilkan kurva Bézier yang lebih kompleks dan dapat menyesuaikan dengan pola yang diberikan.
`def midpoint(self, p1,p2)`	Fungsi untuk mencari titik tengah dari dua titik.

`def midpoint_of_three(self, a,b,c)`	Fungsi untuk mencari titik berat dari tiga titik.
`def plot_curve(self)`	Fungsi ini memiliki tujuan untuk memvisualisasikan kurva Bézier yang telah dibuat bersama dengan titik kontrolnya. Pertama-tama, titik kontrol ditampilkan sebagai titik-titik berwarna merah pada plot. Selanjutnya, kurva Bézier yang dihasilkan dari titik kontrol tersebut ditampilkan sebagai garis putus-putus dengan warna abu-abu. Seluruh titik kurva juga ditampilkan sebagai titik berwarna biru. Penjelasan ini membantu pengguna untuk dengan cepat memahami struktur dan karakteristik dari kurva Bézier yang telah dibuat, serta memfasilitasi analisis visual terhadap kurva tersebut.
Class App(tk.ctk)	

Class ini menginherit sebuah tk.ctk yang merupakan suatu class yang dimiliki oleh library customtkinter. Class ini merupakan class yang digunakan untuk menginisiasi seluruh widget tkinter pada project ini. Class ini akan melakukan main loop untuk me-render widgetnya. Selain itu, class ini yang nantinya akan memanggil frame lainnya yang merupakan suatu class bernama InputFrame dan OutputFrame. Seluruh Frame ini akan di pack pada App dan lalu app akan di inisiasi pada main dalam file gui.py. Yang nantinya akan digunakan pada main.py dalam menjalankan window tkinternya. Terdapat beberapa function yang ada di class ini yaitu setupFrame untuk menyimpan InputFrame dan OutputFrame sebagai object dalam App. Selain itu, terdapat function setupOutputCanvas yang merupakan salah satu function penting dalam memulai rangkaian animasi pada canvas dalam outputframe
Class InputFrame(tk.CTkFrame)
Class ini menginherit suatu class bernama ctkframe yang merupakan class yang berfungsi sebagai container untuk widget yang lain yang ada dalam frame tersebut. Input frame ini bertujuan dalam menyimpan suatu class bernama InputContainer dan menyimpan suatu canvas yang merupakan scrollable canvas agar user dapat menggeser layar widget keatas dan bawah. Pada InputFrame ini, terdapat beberapa function yang digunakan seperti on_mousewheel yang merupakan suatu event function yang berguna saat user memutar mousewheel pada container ini dimana container akan terscroll ke atas bawah sesuai input dari mousewheelnya, setupScrollableContainer yang melakukan setup berupa memasukan InputContainer dalam suatu scrollable canvas yang berguna agar

user dapat melakukan drag pada layar input sehingga widget input frame dapat bergeser ke atas bawah.
Class InputContainer(tk.CtkFrame)
Sama seperti InputFrame, class ini juga menginherit class tk.CtkFrame. InputContainer ini berfungsi menyimpan segala informasi terkait button, entry, dan label untuk InputFrame. Terdapat beberapa function yang seluruhnya merupakan inisiasi dari widget yang ada pada input container ini seperti setupIterationField yang melakukan setup widget entry yang meminta input iteration pada user, setupPointsField yang merupakan list widget entry yang meminta user berupa control pointsnya, setupAnimationSpeedEntry yang merupakan entry widget juga yang meminta input user berupa kecepatan animation untuk output nantinya, dan terakhir merupakan setupButton yang berisikan button bruteforce dan divideandconquer. Pada setupButton ini, terdapat beberapa hal penting yang perlu diperhatikan yaitu dimana terdapat function bernama dataSave yang melakukan saving segala input data pada database statik, dan melakukan callback function pada suatu function di class App yang nantinya akan mengirimkan sinyal pada canvas dalam OutputFrame untuk menginisialisasi output animation.
Class InputFieldsDots(tk.CtkFrame)
Class ini merupakan class yang dipanggil pada Inputcontainer yang mana class ini hanya berfungsi dalam mempersingkat dan meningkatkan modularitas pada

setupPointsField dalam class InputContainer. Class ini berisikan function dan data input control points oleh user.
Class OutputFrame(tk.CtkFrame)
Class ini merupakan suatu class yang berfungsi sebagai Container untuk segala output yaitu sebuah canvas untuk animation dan time execution yang menampilkan data waktu yang dibutuhkan suatu algoritma dalam menyelesaikan logikanya. Terdapat beberapa function yang digunakan pada class ini diantaranya setupOutputTitle, setupTimeExecution, setupOutputCanvas yang merupakan part yang sekaligus bertanggung jawab dalam pembuatan animasi dan kurva bezier, setupTime yang berbeda dengan setupTimeExecution dimana pada setupTimeExecution merupakan function yang berfungsi mengatur data Time pada widget Time execution sedangkan setupTime yang menginisialisasi widget time pada frame.
GlobalData.py
GlobalData.py merupakan sebuah file yang berisikan static method dan attribute yang digunakan dalam transfer data antara inputFrame dengan Canvas pada Outputframe. Terdapat beberapa attribut yang terdapat didalamnya seperti control_points yang merupakan container seluruh control points yang akan dimanipulasi nantinya, iteraitons yang menyimpan attribute jumlah iterasi, selected_points yang menyimpan jumlah dari control_points yang akan digunakan, hal ini karena pada implementasi ini, elemen yang di input pada

entry tidak akan terhapus sehingga entry yang digunakan harus dispecify jumlahnya. Apps_height dan apps_width yang menyimpan data tinggi dan panjang canvas pada saat itu, animation speed, is_button_clicked yang menjadi trigger function button di click, data algorithm yang menyimpan jenis algoritma yang di click berdasarkan buttonnya, max_x, min_x, max_y, min_y, origin_points yang menyimpan data control_points sebelum dilakukan manipulasi, dan is_animated yang berfungsi sebagai event prevent handler yang mencegah canvas mengalami override terus menerus saat button di click. Terdapat beberapa function yang ada dalam file ini seperti beberapa getter setter untuk atribut seperti control points, iterations, min max, dll. Selain itu terdapat beberapa fungsi penting yang menangani konversi control points pada canvas, seperti handle_negatives_data yang menghandle data negatif karena canvas tkinter tidak mau menerima input negative oleh karena itu perlu sebuah kalkulasi yang mengganti seluruh control points agar dapat masuk pada area canvas, terdapat pulahandle_points_flipped yang membalikkan seluruh data y pada control points karena titik pusat (0,0) pada canvas tkinter berada pada kiri atas sehingga dan titik puncaknya berada pada kanan bawah sehingga diperlukan algoritma untuk menerapkan logika tersebut pada control points. Terakhir ada function penting yaitu handle_points_scaling dimana function ini menormalisasikan seluruh data yang ada yang kemudian di scale berdasarkan ukuran canvas sehingga seluruh point akan selalu berada dalam canvas walaupun ukurannya desimal maupun jutaan.

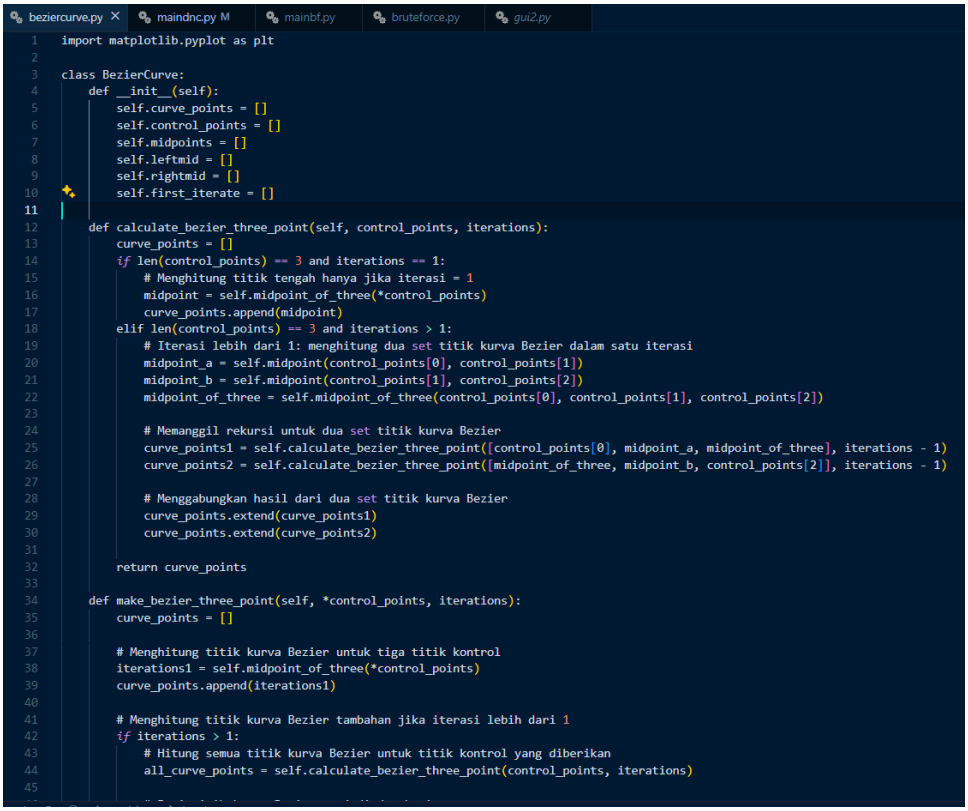
Class BezierCanvasAnimation(tk.CTkCanvas)

Class ini adalah sebuah base class yang berisikan metode animation pada canvas. Terdapat beberapa function yang perlu diperhatikan yaitu draw_points yang merupakan fungsi yang menggambarkan seluruh titip control dan garis yang menghubungkan titik titik tersebut. Selain itu, juga terdapat interpolate_line yang berfungsi membuat interpolasi garis untuk garis bantu dalam pembuatan bezier curve. Terakhir terdapat fungsi animation yang merupakan fungsi yang bertugas menjalankan animasi kurva dari titik kontrol awal hingga titik kontrol akhir.

4.1.2 Source Code Program

4.1.2.1 Algoritma Divide dan Conquer

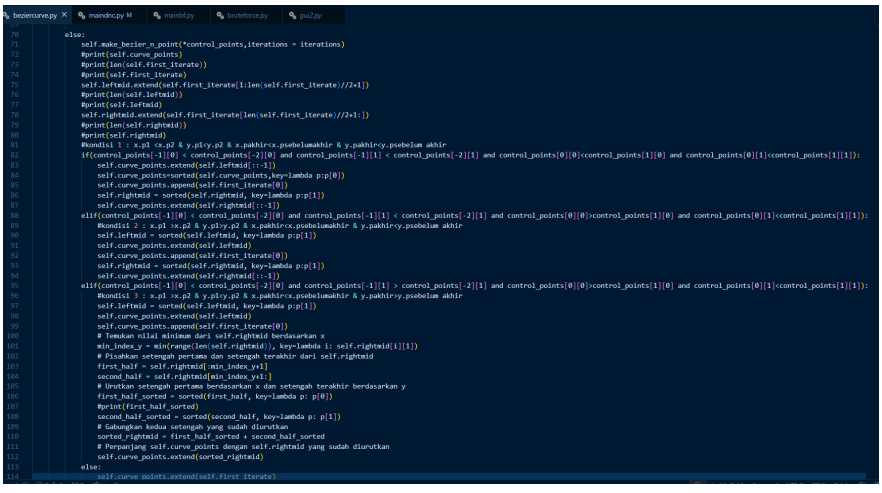
1. File beziercurve.py

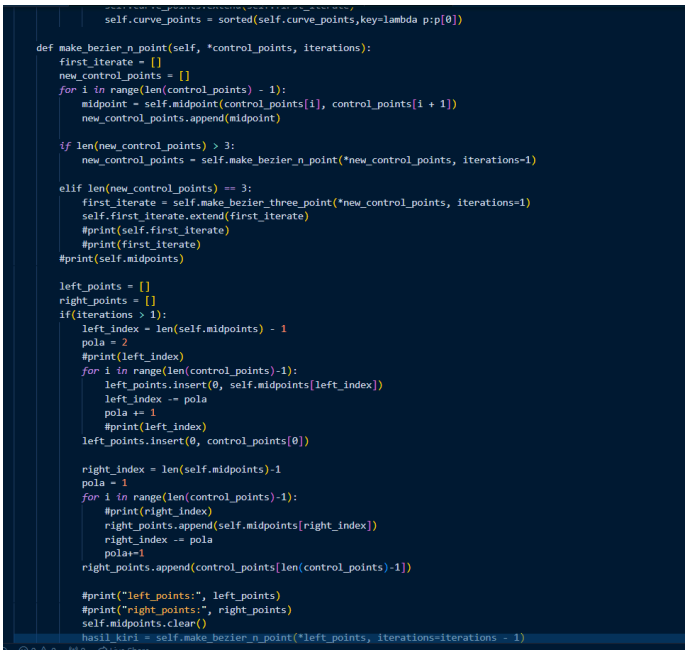


```python
import matplotlib.pyplot as plt

class BezierCurve:
    def __init__(self):
        self.curve_points = []
        self.control_points = []
        self.midpoints = []
        self.leftmid = []
        self.rightmid = []
        self.first_iterate = []

    def calculate_bezier_three_point(self, control_points, iterations):
        curve_points = []
        if len(control_points) == 3 and iterations == 1:
            # Menghitung titik tengah hanya jika iterasi = 1
            midpoint = self.midpoint_of_three(*control_points)
            curve_points.append(midpoint)
        elif len(control_points) == 3 and iterations > 1:
            # Iterasi lebih dari 1: menghitung dua set titik kurva Bezier dalam satu iterasi
            midpoint_a = self.midpoint(control_points[0], control_points[1])
            midpoint_b = self.midpoint(control_points[1], control_points[2])
            midpoint_of_three = self.midpoint_of_three(control_points[0], control_points[1], control_points[2])

            # Memanggil rekursi untuk dua set titik kurva Bezier
            curve_points1 = self.calculate_bezier_three_point([control_points[0], midpoint_a, midpoint_of_three], iterations - 1)
            curve_points2 = self.calculate_bezier_three_point([midpoint_of_three, midpoint_b, control_points[2]], iterations - 1)

            # Menggabungkan hasil dari dua set titik kurva Bezier
            curve_points.extend(curve_points1)
            curve_points.extend(curve_points2)

        return curve_points

    def make_bezier_three_point(self, *control_points, iterations):
        curve_points = []

        # Menghitung titik kurva Bezier untuk tiga titik kontrol
        iterations1 = self.midpoint_of_three(*control_points)
        curve_points.append(iterations1)

        # Menghitung titik kurva Bezier tambahan jika iterasi lebih dari 1
        if iterations > 1:
            # Hitung semua titik kurva Bezier untuk titik kontrol yang diberikan
            all_curve_points = self.calculate_bezier_three_point(control_points, iterations)
```

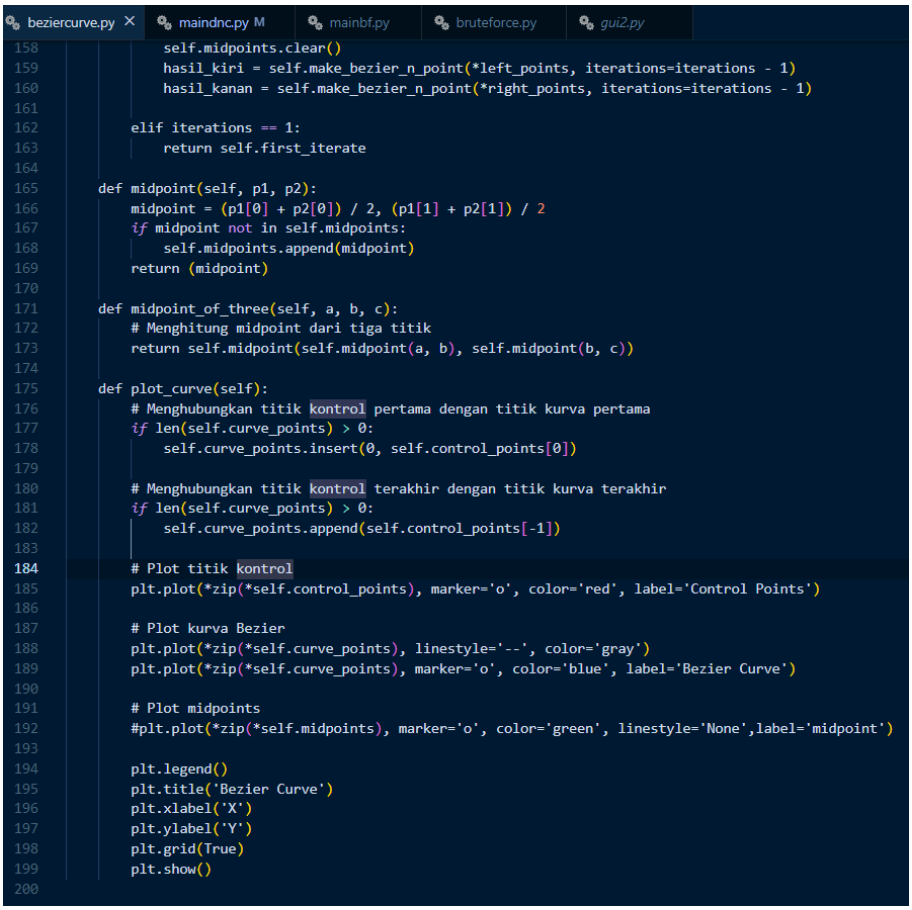
beziercurve.py | maindnc.py M | mainbf.py | bruteforce.py | gui2.py

```python
    def make_bezier_three_point(self, *control_points, iterations):
        curve_points = []

        # Menghitung titik kurva Bezier untuk tiga titik kontrol
        iterations1 = self.midpoint_of_three(*control_points)
        curve_points.append(iterations1)

        # Menghitung titik kurva Bezier tambahan jika iterasi lebih dari 1
        if iterations > 1:
            # Hitung semua titik kurva Bezier untuk titik kontrol yang diberikan
            all_curve_points = self.calculate_bezier_three_point(control_points, iterations)

            # Bagi titik kurva Bezier menjadi dua bagian
            left = all_curve_points[:len(all_curve_points)//2]
            for point in left[::-1]:
                curve_points.insert(0, point)
            right = all_curve_points[len(all_curve_points)//2:]

            curve_points.extend(right)

        return curve_points


    def make_bezier(self, *control_points, iterations):
        #print(len(control_points))
        self.control_points = list(control_points)
        # Jumlah titik kontrol harus minimal 3
        if len(control_points) < 3:
            print("Error: Minimum 3 control points required.")
            return

        # Jika jumlah titik kontrol sama dengan 3, gunakan make_bezier_three_point
        elif len(control_points) == 3:
            self.curve_points = self.make_bezier_three_point(*control_points, iterations=iterations)
            return

        else:
            self.make_bezier_n_point(*control_points,iterations = iterations)
            #print(self.curve_points)
            #print(len(self.first_iterate))
            #print(self.first_iterate)
            self.leftmid.extend(self.first_iterate[1:len(self.first_iterate)//2+1])
            #print(len(self.leftmid))
            #print(self.leftmid)
            self.rightmid.extend(self.first_iterate[len(self.first_iterate)//2+1:])
```





```
            self.curve_points = sorted(self.curve_points,key=lambda p:p[0])

    def make_bezier_n_point(self, *control_points, iterations):
        first_iterate = []
        new_control_points = []
        for i in range(len(control_points) - 1):
            midpoint = self.midpoint(control_points[i], control_points[i + 1])
            new_control_points.append(midpoint)

        if len(new_control_points) > 3:
            new_control_points = self.make_bezier_n_point(*new_control_points, iterations=1)

        elif len(new_control_points) == 3:
            first_iterate = self.make_bezier_three_point(*new_control_points, iterations=1)
            self.first_iterate.extend(first_iterate)
            #print(self.first_iterate)
            #print(first_iterate)
        #print(self.midpoints)

        left_points = []
        right_points = []
        if(iterations > 1):
            left_index = len(self.midpoints) - 1
            pola = 2
            #print(left_index)
            for i in range(len(control_points)-1):
                left_points.insert(0, self.midpoints[left_index])
                left_index -= pola
                pola += 1
                #print(left_index)
            left_points.insert(0, control_points[0])

            right_index = len(self.midpoints)-1
            pola = 1
            for i in range(len(control_points)-1):
                #print(right_index)
                right_points.append(self.midpoints[right_index])
                right_index -= pola
                pola+=1
            right_points.append(control_points[len(control_points)-1])

            #print("left_points:", left_points)
            #print("right_points:", right_points)
            self.midpoints.clear()
            hasil_kiri = self.make_bezier_n_point(*left_points, iterations=iterations - 1)
```



beziercurve.py | maindnc.py M | mainbf.py | bruteforce.py | gui2.py

```
158                 self.midpoints.clear()
159                 hasil_kiri = self.make_bezier_n_point(*left_points, iterations=iterations - 1)
160                 hasil_kanan = self.make_bezier_n_point(*right_points, iterations=iterations - 1)
161
162             elif iterations == 1:
163                 return self.first_iterate
164
165         def midpoint(self, p1, p2):
166             midpoint = (p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2
167             if midpoint not in self.midpoints:
168                 self.midpoints.append(midpoint)
169             return (midpoint)
170
171         def midpoint_of_three(self, a, b, c):
172             # Menghitung midpoint dari tiga titik
173             return self.midpoint(self.midpoint(a, b), self.midpoint(b, c))
174
175         def plot_curve(self):
176             # Menghubungkan titik kontrol pertama dengan titik kurva pertama
177             if len(self.curve_points) > 0:
178                 self.curve_points.insert(0, self.control_points[0])
179
180             # Menghubungkan titik kontrol terakhir dengan titik kurva terakhir
181             if len(self.curve_points) > 0:
182                 self.curve_points.append(self.control_points[-1])
183
184             # Plot titik kontrol
185             plt.plot(*zip(*self.control_points), marker='o', color='red', label='Control Points')
186
187             # Plot kurva Bezier
188             plt.plot(*zip(*self.curve_points), linestyle='--', color='gray')
189             plt.plot(*zip(*self.curve_points), marker='o', color='blue', label='Bezier Curve')
190
191             # Plot midpoints
192             #plt.plot(*zip(*self.midpoints), marker='o', color='green', linestyle='None',label='midpoint')
193
194             plt.legend()
195             plt.title('Bezier Curve')
196             plt.xlabel('X')
197             plt.ylabel('Y')
198             plt.grid(True)
199             plt.show()
200
```

2. File maindnc.py

4.1.2.2 Algoritma Brute Force

1. File bruteforce.py

```python
import numpy as np
import matplotlib.pyplot as plt
class BezierCurve:
    def __init__(self):
        self.curve_points = []
        self.control_points = []

    def make_bezier(self, *control_points, iterations):
        self.curve_points.clear()
        self.control_points = list(control_points)

        # Jumlah titik kontrol harus sama dengan tiga
        if len(control_points) != 3:
            print("Error: 3 control points required")
            return

        p0, p1, p2 = control_points

        # Menghitung jumlah titik berdasarkan iterasi
        N = 2**iterations + 1

        # Membuat basis nilai t awal
        t_base = np.linspace(0, 1, num=N)

        # Inisialisasi array nilai t
        t_values = np.zeros(N)

        # Memasukkan basis nilai t awal
        for i in range(len(t_base)):
            t_values[i] = t_base[i]
        for i in range(len(self.control_points) - 2):
            for t in t_values:
                #print(t)
                qx0 = (1 - t) * p0[0] + t * p1[0]
                qy0 = (1 - t) * p0[1] + t * p1[1]
                qx1 = (1 - t) * p1[0] + t * p2[0]
                qy1 = (1 - t) * p1[1] + t * p2[1]
                x_new = (1 - t) * qx0 + t * qx1
                y_new = (1 - t) * qy0 + t * qy1
                new_points = (x_new, y_new)
                if new_points not in self.curve_points:
                    self.curve_points.append(new_points)

        # Menambahkan titik terakhir
        if self.control_points[-1] not in self.curve_points:
```

```
        # Memasukkan basis nilai t awal
        for i in range(len(t_base)):
            t_values[i] = t_base[i]
        for i in range(len(self.control_points) - 2):
            for t in t_values:
                #print(t)
                qx0 = (1 - t) * p0[0] + t * p1[0]
                qy0 = (1 - t) * p0[1] + t * p1[1]
                qx1 = (1 - t) * p1[0] + t * p2[0]
                qy1 = (1 - t) * p1[1] + t * p2[1]
                x_new = (1 - t) * qx0 + t * qx1
                y_new = (1 - t) * qy0 + t * qy1
                new_points = (x_new, y_new)
                if new_points not in self.curve_points:
                    self.curve_points.append(new_points)

        # Menambahkan titik terakhir
        if self.control_points[-1] not in self.curve_points:
            self.curve_points.append(self.control_points[-1])

    def plot_curve(self):
        # Plot titik kontrol
        plt.plot(*zip(*self.control_points), marker='o', color='red', label='Control Points')

        # Plot kurva Bezier
        plt.plot(*zip(*self.curve_points), linestyle='--', color='gray')
        plt.plot(*zip(*self.curve_points), marker='o', color='blue', label='Bezier Curve')

        plt.legend()
        plt.title('Quadratic Bezier Curve')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.grid(True)
        plt.show()
```

2. File mainbf.py

```
import matplotlib.pyplot as plt
import numpy as np
import time

from bruteforce import BezierCurve

bezier = BezierCurve()

# Masukkan banyaknya titik kontrol
while True:
    n = int(input("Masukkan banyak titik kontrol: "))
    if n != 3:
        print("Error: Minimum 3 control points required.")
    else:
        break

# Masukkan titik kontrol sesuai dengan jumlah yang dimasukkan sebelumnya
control_points = []
for i in range(n):
    x = float(input(f"Masukkan koordinat x titik kontrol ke-{i+1}: "))
    y = float(input(f"Masukkan koordinat y titik kontrol ke-{i+1}: "))
    control_points.append((x, y))

while True:
    iterations = int(input("Masukkan jumlah iterasi: "))
    if iterations < 1:
        print("Error: Minimum 3 control points required.")
    else:
        break

start_time = time.time()
bezier.make_bezier(*control_points, iterations=iterations)
end_time = time.time()

print("Waktu eksekusi:", end_time - start_time, "detik")
bezier.plot_curve()
```

4.1.3 Library

1. Import OS

Library os digunakan untuk berinteraksi dengan sistem operasi. Ini membantu program untuk melakukan berbagai operasi terkait sistem file, seperti membuat, mengubah, dan menghapus direktori atau file, mengelola variabel lingkungan, dan menavigasi struktur direktori. Dengan os, program dapat lebih fleksibel dan dapat beradaptasi dengan berbagai lingkungan sistem operasi

2. Import time

   Library time digunakan untuk bekerja dengan waktu dan penundaan dalam program Python. *Library* ini menyediakan fungsi-fungsi untuk mendapatkan waktu saat ini, mengonversi waktu antara format yang berbeda, serta mengukur durasi eksekusi suatu kode.

3. Import matplotlib.pyplot

   Matplotlib.pyplot adalah salah satu modul dari library Matplotlib yang digunakan untuk membuat visualisasi data dalam bentuk grafik atau plot di lingkungan Python. Modul ini menyediakan beragam fungsi yang memungkinkan pengguna untuk membuat berbagai jenis plot, seperti garis, scatter plot, histogram, dan sebagainya. Dalam konteks program ini, Matplotlib.pyplot digunakan untuk membuat visualisasi kurva Bézier yang dihasilkan oleh fungsi-fungsi pembuat kurva Bézier. Dengan menggunakan Matplotlib.pyplot, kita dapat dengan mudah membuat plot yang menampilkan titik kontrol dan kurva Bézier yang dihasilkan, sehingga memudahkan dalam pemahaman dan analisis visual atas kurva yang dibuat.

4. Import customtkinter

customtkinter adalah modul kustom yang mungkin telah dibuat untuk menyediakan fungsi tambahan atau penyesuaian tertentu untuk framework GUI Python, Tkinter. Modul ini dapat digunakan untuk memperluas kemampuan atau menyesuaikan antarmuka pengguna yang dibangun dengan Tkinter sesuai kebutuhan pengembangan perangkat lunak.

5. Import PIL

   PIL pada project ini hanya digunakan untuk membuka gambar dimana gambar tersebut didapatkan dari path yang didefinisikan oleh OS.

6. Import tkinter

   tkinter adalah library standar Python yang digunakan untuk membuat antarmuka pengguna grafis (GUI). Tkinter menyediakan berbagai widget GUI, seperti tombol, label, input teks, dan lainnya, serta metode untuk mengatur tata letak dan menangani interaksi pengguna. Dengan Tkinter, pengembang dapat membuat aplikasi desktop yang interaktif dan mudah digunakan oleh pengguna.

7. Import typing

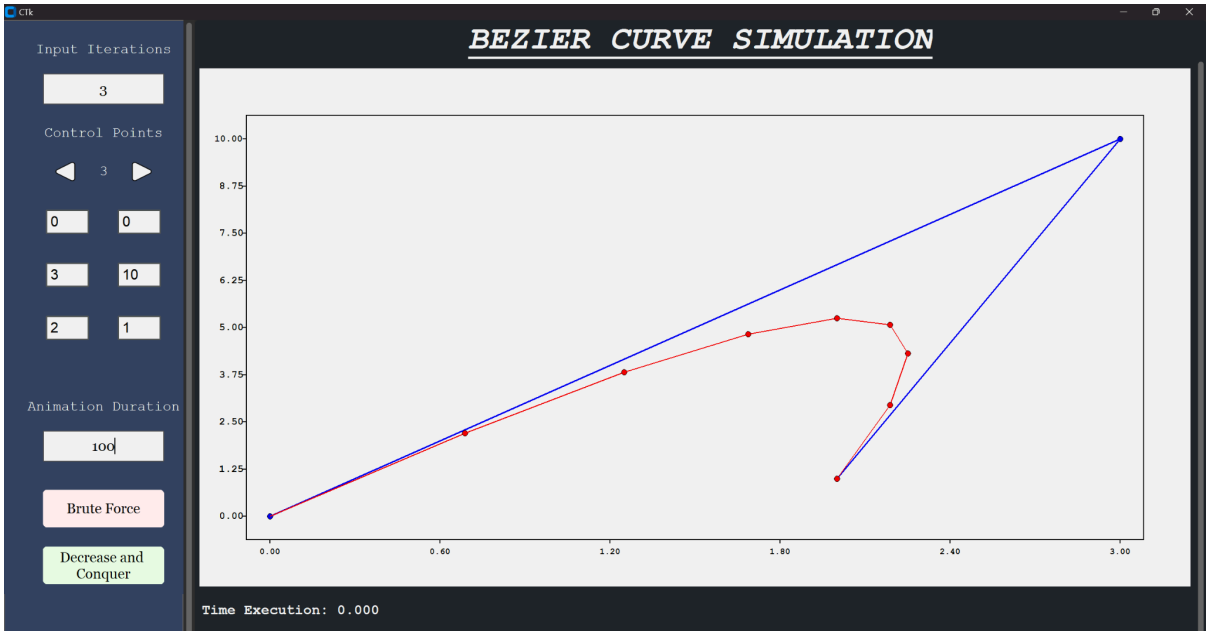
   typing adalah modul Python yang menyediakan alat untuk melakukan type hinting, yaitu menentukan tipe data yang diharapkan untuk parameter dan return value dari fungsi. Type hinting membantu dalam dokumentasi kode, pemahaman kode oleh pengembang lain, dan memvalidasi tipe data secara statis menggunakan alat analisis statis seperti Mypy. Ini membantu dalam

menangani kesalahan tipe data sebelum runtime dan meningkatkan kejelasan dan keamanan kode Python.

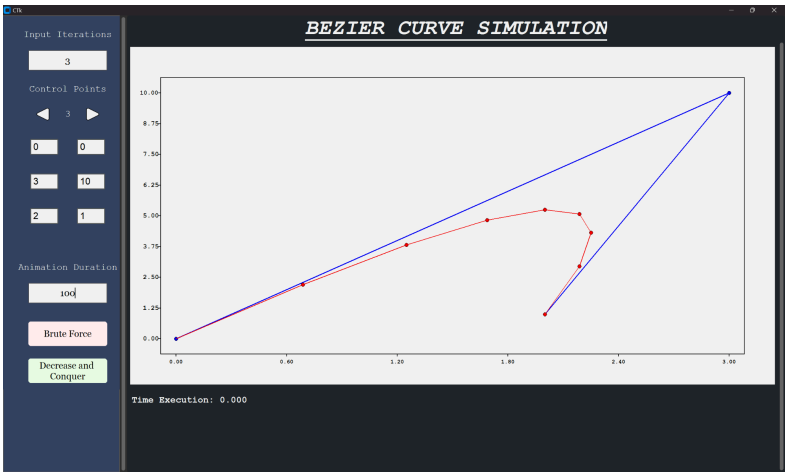
BAB 5 ANALISIS DAN PENGUJIAN

5.1 Kasus 1

Bruteforce

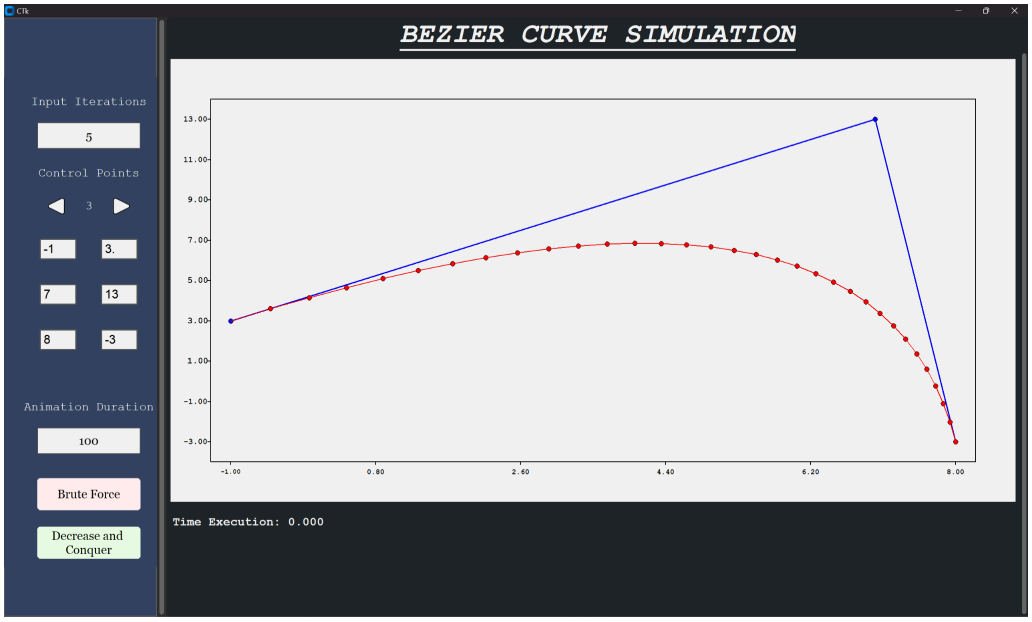


Divide and Conquer

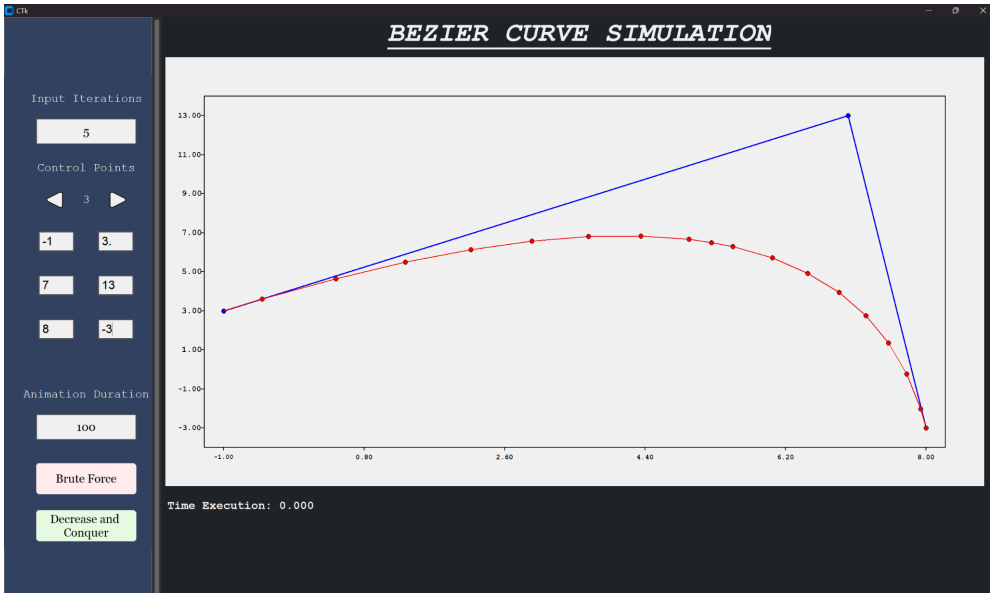


5.2 Kasus 2

Bruteforce

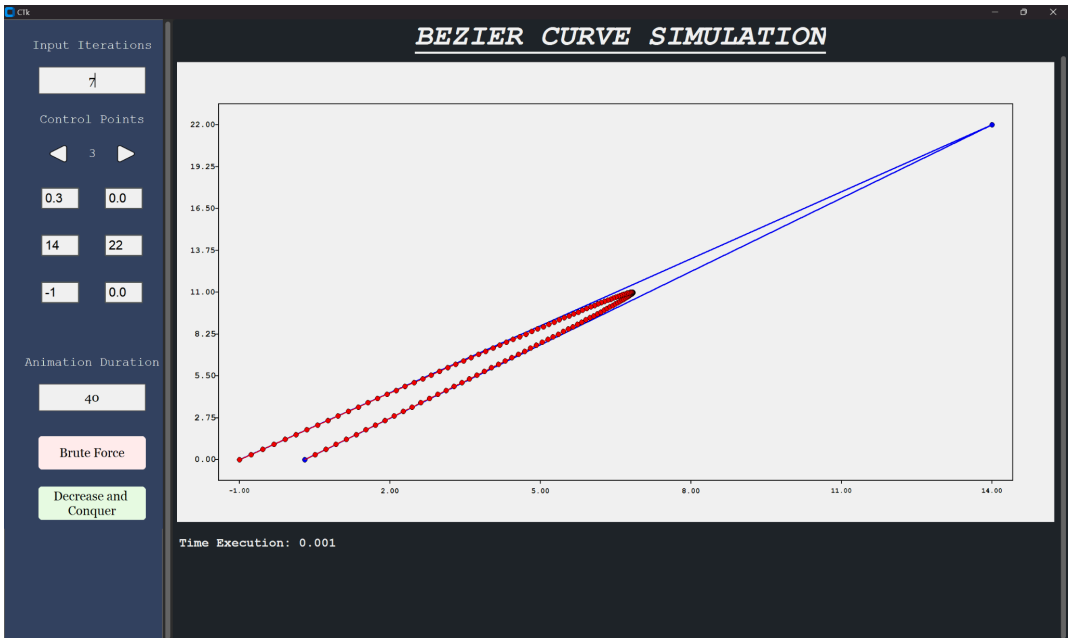


Divide and Conquer

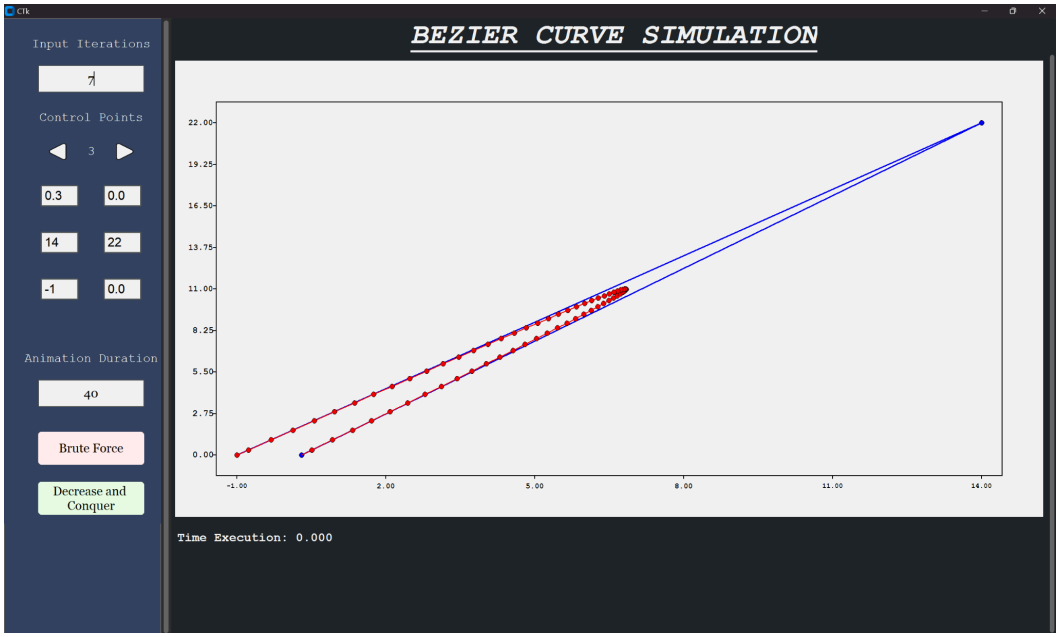


5.3 Kasus 3

Bruteforce

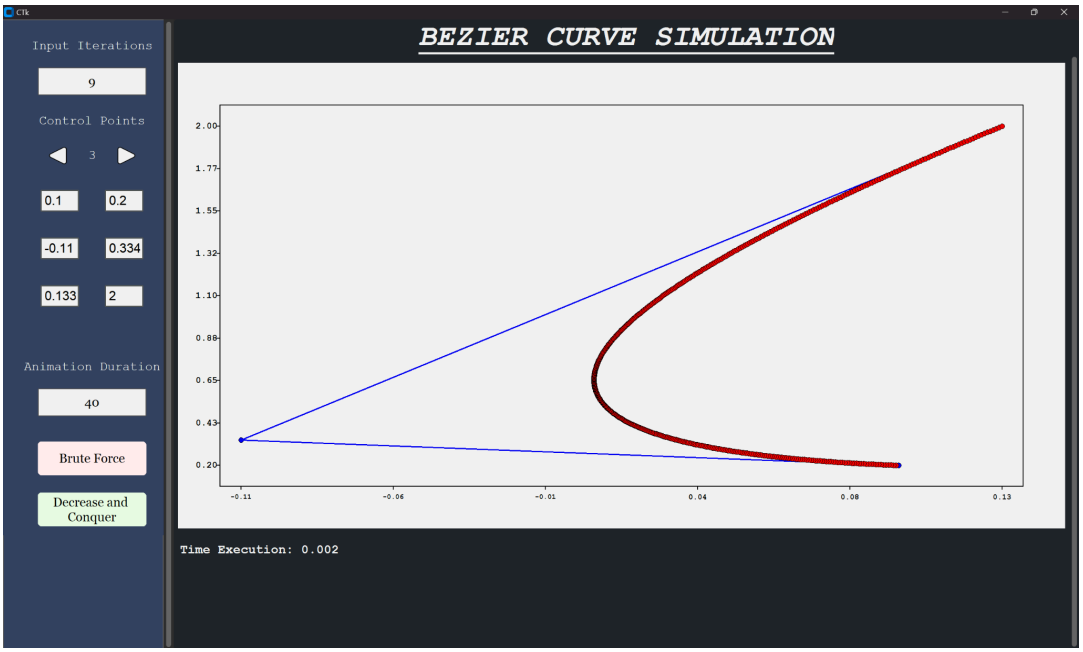


Divide and Conquer

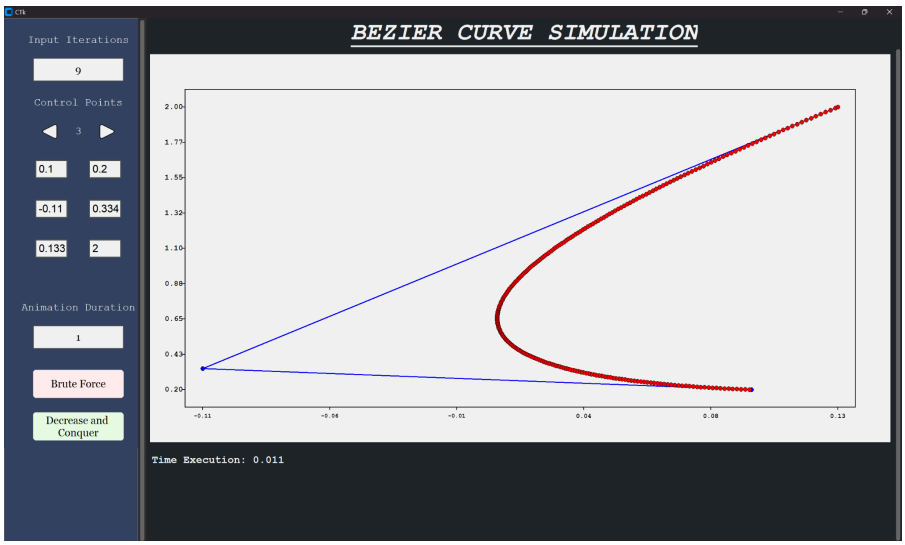


5.4 Kasus 4

Bruteforce

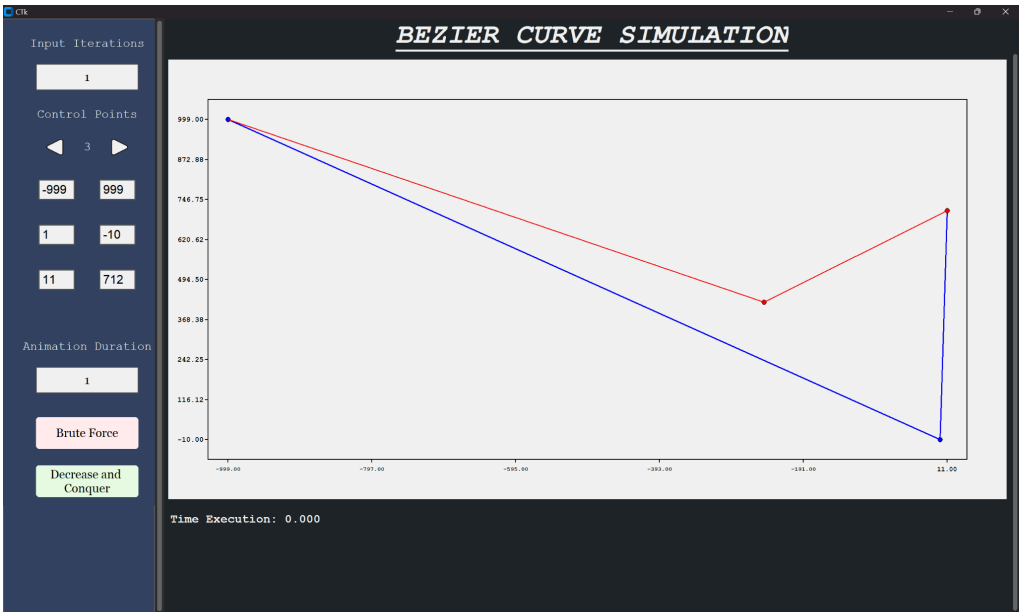


Divide and Conquer

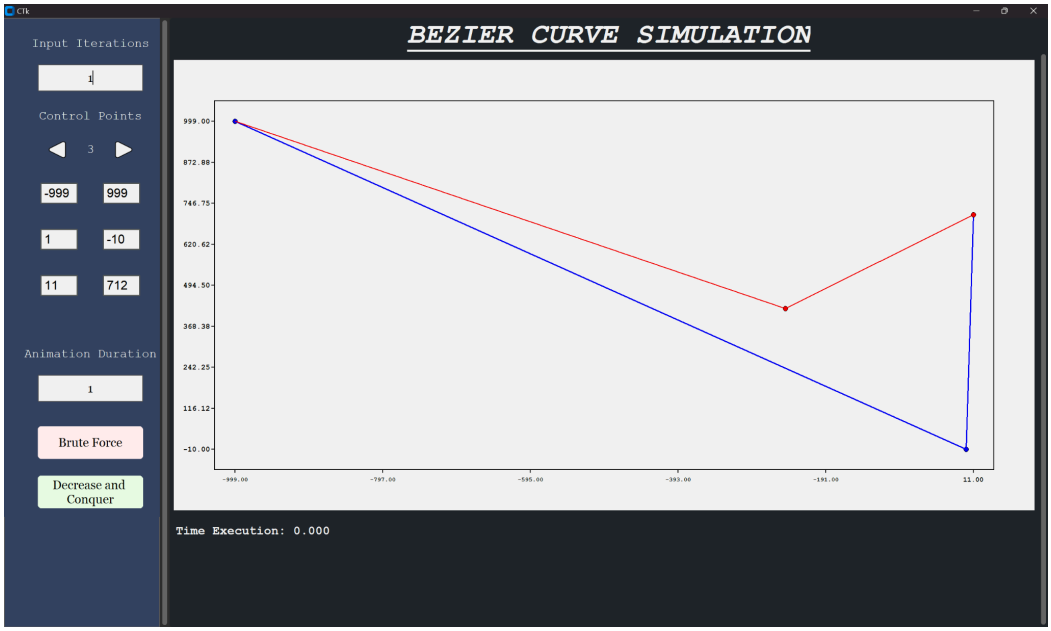


5.1 Kasus 5

Bruteforce

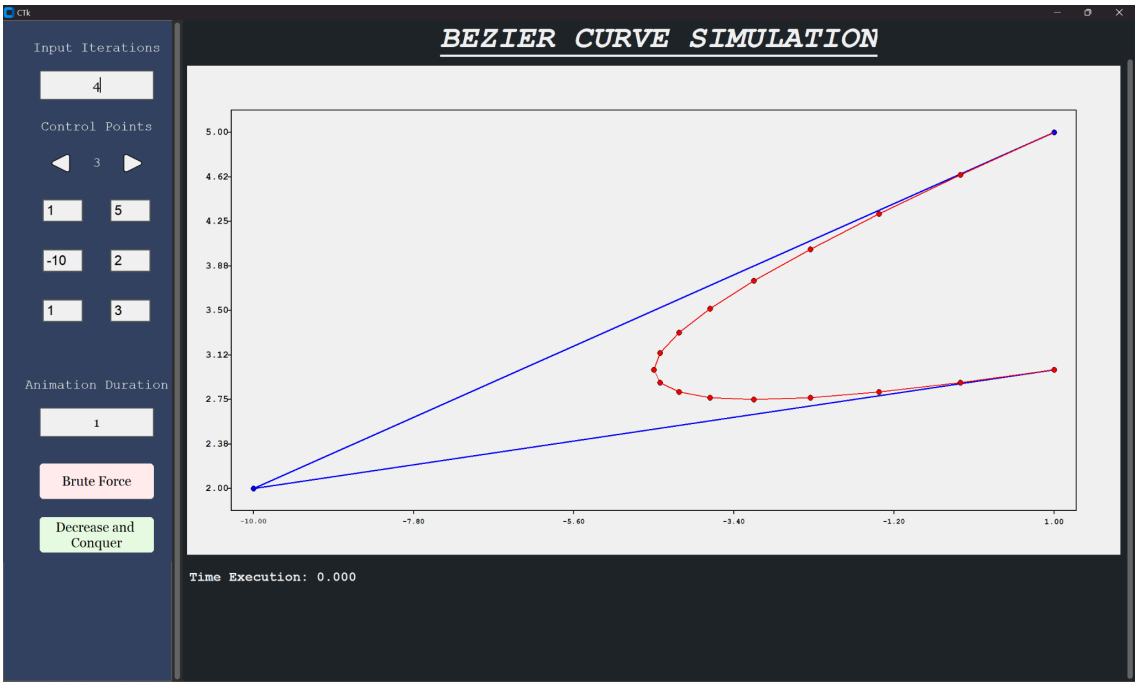


Divide dan Conquer

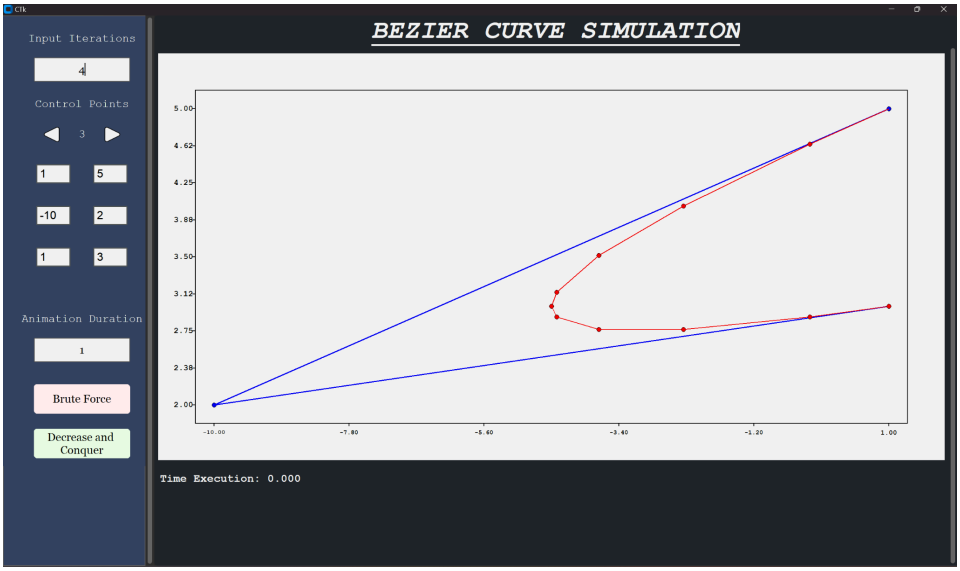


5.6 Kasus 6

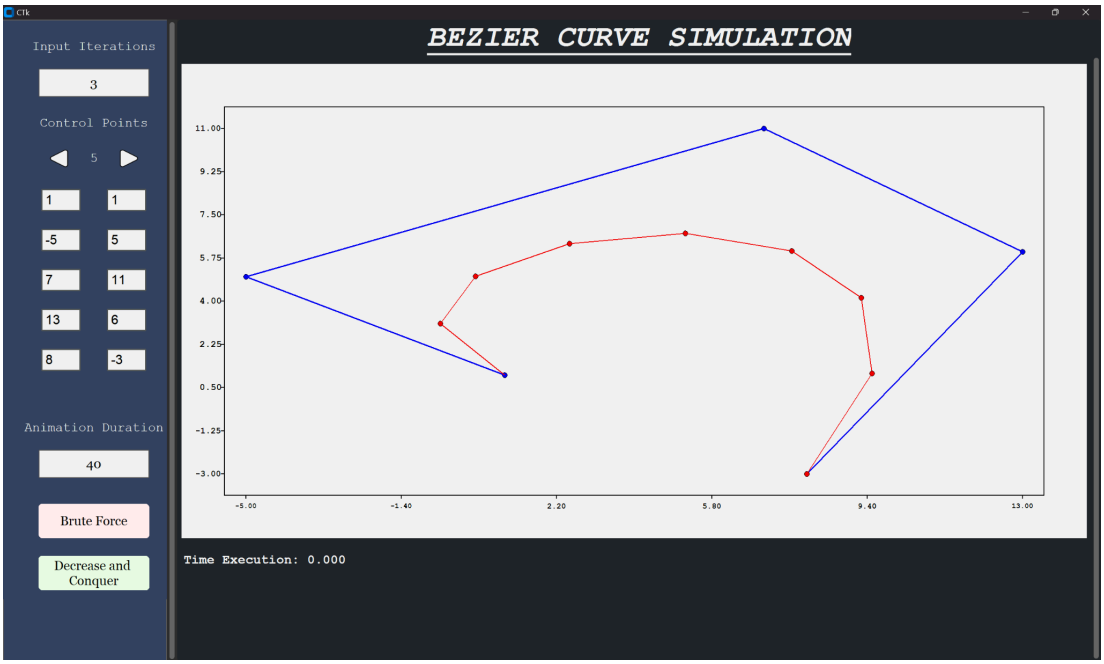
Bruteforce



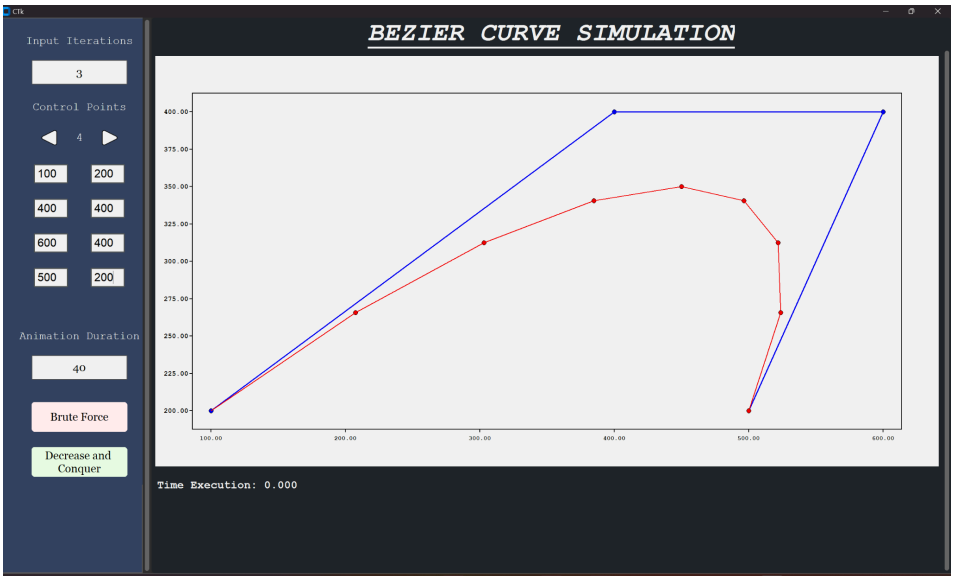
Divide and Conquer



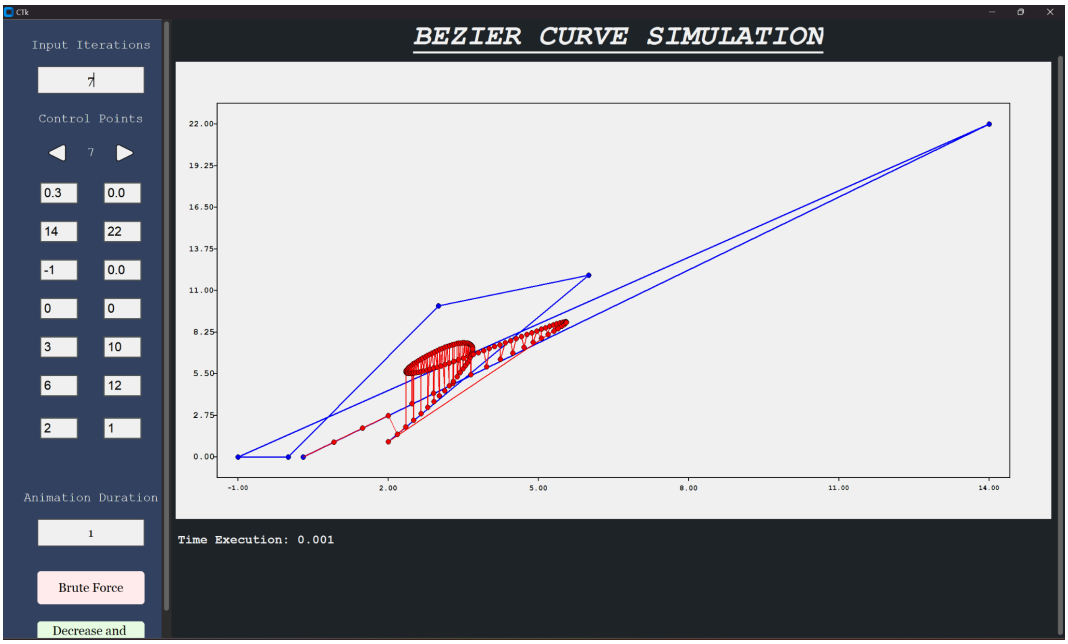
5.6 Kasus Bonus 1



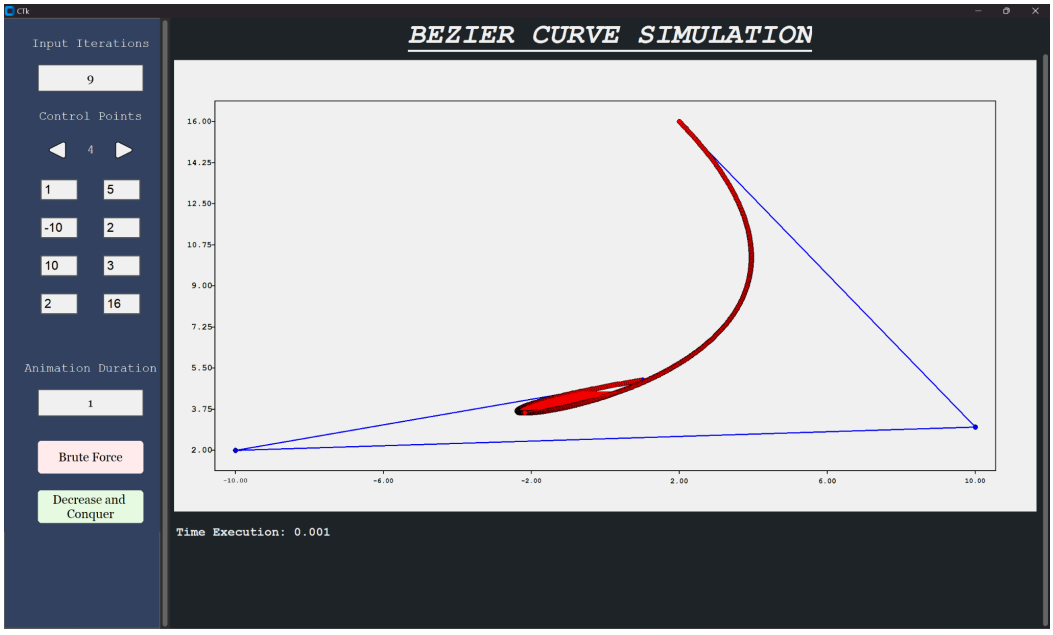
5.7 Kasus Bonus 2



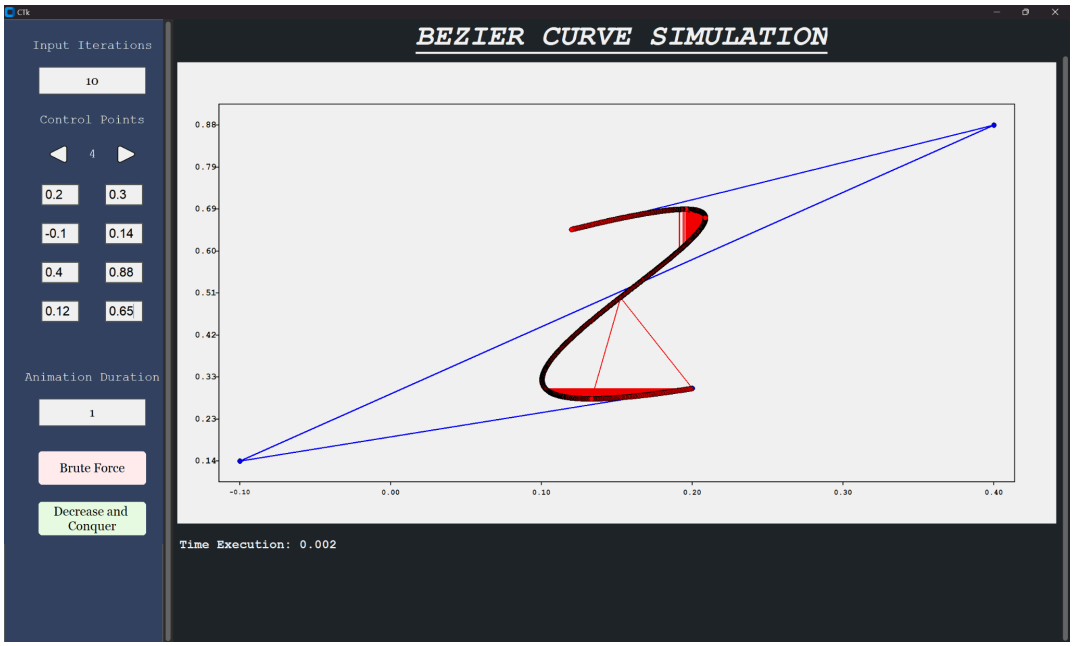
5.8 Kasus Bonus 3



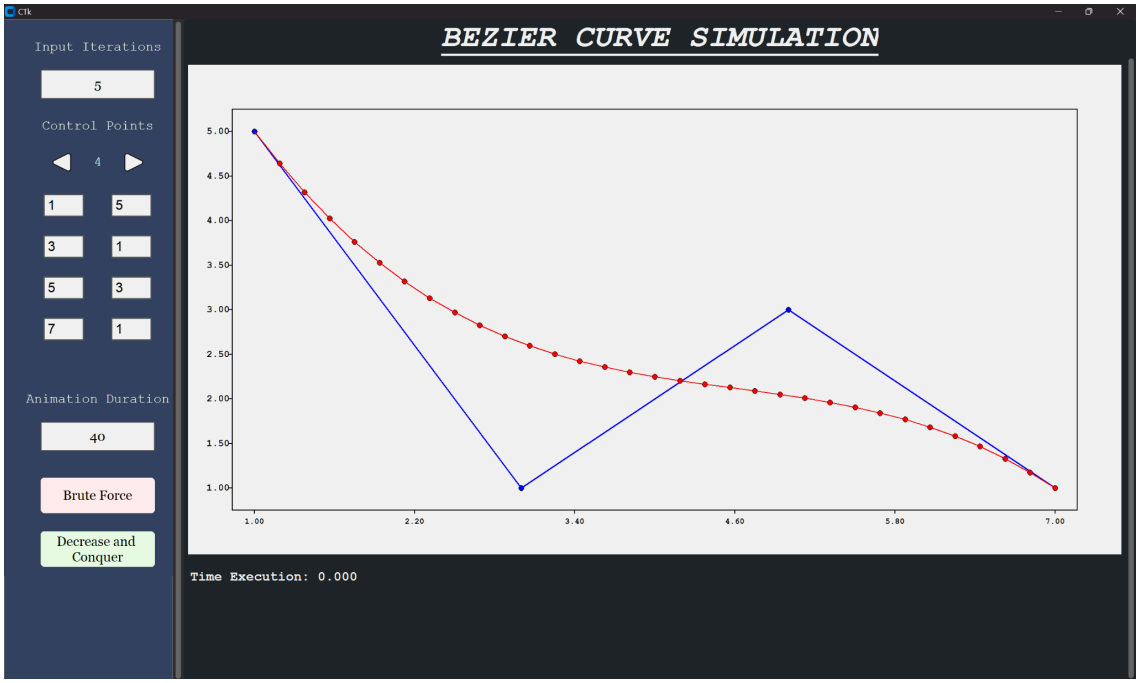
5.9 Kasus Bonus 4



5.10 Kasus Bonus 5



5.11 Kasus Bonus 6



5.12 Kompleksitas Waktu Brute Force dengan Divide and Conquer

Perbedaan mendasar antara algoritma brute force dan divide and conquer terletak pada pendekatan dalam menyelesaikan masalah. Dalam algoritma brute force, setiap kemungkinan solusi dieksplorasi secara langsung tanpa mempertimbangkan optimasi atau pola yang mungkin ada dalam masalah tersebut. Dalam kasus pembuatan kurva Bezier dengan pendekatan brute force, titik-titik kurva dihitung dengan membagi rentang parameter t menjadi sejumlah besar interval dan menghitung nilai kurva pada setiap titik interval. Ini menghasilkan kompleksitas waktu yang lebih tinggi karena semua kemungkinan titik harus dievaluasi, terutama karena jumlah iterasi meningkat. Dalam kasus brute force, jika setiap iterasi membagi rentang parameter t menjadi dua bagian, maka jumlah evaluasi total akan menjadi 2 pangkat n + 1, di mana n adalah jumlah iterasi yang diberikan. Oleh karena itu, kompleksitas waktu untuk algoritma brute force dapat diekspresikan sebagai O(2^n), di mana n adalah jumlah iterasi.

Di sisi lain, dalam algoritma divide and conquer, masalah dibagi menjadi submasalah yang lebih kecil yang kemudian diselesaikan secara terpisah. Dalam pembuatan kurva Bezier dengan pendekatan divide and conquer, titik-titik kurva dihitung dengan membagi kontrol poin menjadi subset yang lebih kecil dan menghitung kurva pada setiap subset. Kemudian, hasilnya digabungkan untuk menghasilkan kurva Bezier akhir. Meskipun algoritma ini menggunakan rekursi, namun kompleksitasnya dapat lebih rendah karena memanfaatkan struktur masalah untuk mengurangi jumlah perhitungan yang diperlukan. Pendekatan ini memiliki kompleksitas waktu O(log n), di mana n adalah jumlah titik kontrol. Kompleksitas ini meningkat secara logaritmik, membuatnya jauh lebih efisien daripada brute force ketika jumlah titik kontrol besar.

Kompleksitas waktu algoritma brute force dalam kasus ini akan lebih tinggi karena harus mempertimbangkan setiap kemungkinan kombinasi titik kontrol, sementara algoritma divide and conquer membagi masalah menjadi subset yang lebih kecil, mengurangi jumlah perhitungan yang diperlukan secara signifikan.

Namun, ada kasus tertentu di mana brute force dapat lebih cepat dari divide and conquer. Misalnya, jika jumlah titik kontrol sangat sedikit sehingga overhead dari rekursi dan penggabungan hasil menjadi lebih besar daripada keuntungan yang diperoleh dari membagi masalah menjadi subset yang lebih kecil, maka brute force dapat menjadi pilihan yang lebih efisien. Selain itu, jika struktur persoalan tidak mendukung pembagian yang efisien, algoritma divide and conquer mungkin tidak memberikan keuntungan yang diharapkan.

5.13 Analisis Kompleksitas Waktu 3 Titik Kontrol dengan n Titik Kontrol

Dalam kode yang diberikan untuk menghitung kurva Bezier menggunakan algoritma divide and conquer, kompleksitas waktu berbeda tergantung pada jumlah titik kontrol yang digunakan.

Dalam kasus 3 titik kontrol, fungsi `calculate_bezier_three_point` mendominasi perhitungan. Fungsi ini secara rekursif membagi kurva menjadi dua segmen yang lebih kecil dengan titik tengah sebagai titik kontrol baru. Kedalaman rekursi dalam kasus ini hanya 1 (hanya satu pembagian), sehingga perhitungan di setiap level rekursi melibatkan operasi konstan seperti mencari titik tengah. Oleh karena itu, kompleksitas waktu untuk 3 titik kontrol dapat dianggap konstan O(1) atau dianggap logaritmik O(log n) karena kedalaman rekursi yang dangkal.

Sementara itu, dalam kasus dengan n titik kontrol (n > 3), fungsi `make_bezier_n_point` menangani kasus umum. Fungsi ini secara rekursif memanggil dirinya sendiri untuk membagi kumpulan n titik kontrol menjadi segmen yang lebih kecil hingga didapatkan 3 titik kontrol. Kedalaman rekursi dalam kasus ini bergantung pada jumlah titik kontrol (n) dan mengikuti aturan pembagian rekursif. Umumnya, kedalaman rekursi akan berorde logaritmik terhadap n (log n). Perhitungan di setiap level rekursi melibatkan pembagian kumpulan titik kontrol dan panggilan fungsi rekursif. Oleh karena itu, kompleksitas waktu untuk n titik kontrol adalah logaritmik O(log n).

Dengan demikian, kompleksitas waktu untuk 3 titik kontrol jauh lebih efisien dibandingkan dengan kasus umum n titik kontrol. Peningkatan jumlah titik kontrol (n) menyebabkan peningkatan logaritmik pada kedalaman rekursi dan keseluruhan kompleksitas waktu. Meskipun kode ini memiliki overhead tambahan di luar perhitungan inti, efisiensi divide and conquer tetap terlihat terutama untuk kasus dengan banyak titik kontrol.

BAB 6 KESIMPULAN DAN SARAN

6.1 Kesimpulan

1. Dalam pemahaman algoritma divide-and-conquer untuk menghitung kurva Bézier dengan tiga titik kontrol, langkah-langkah spesifik telah ditemukan untuk mendapatkan hasil sesuai dengan konsep tersebut. Pertama, terdapat langkah-langkah dasar ketika iterasi pertama dilakukan dan hanya ada tiga titik kontrol. Selanjutnya, untuk iterasi lanjutan, konsep divide-and-conquer diaplikasikan dengan membagi masalah menjadi dua bagian, menyelesaikan masing-masing bagian secara terpisah, dan menggabungkan hasilnya. Proses ini dilakukan rekursif hingga diperoleh hasil yang diinginkan.

2. Dalam kasus umum dengan lebih dari tiga titik kontrol, algoritma divide-and-conquer juga dapat diterapkan dengan membagi masalah menjadi submasalah yang lebih kecil. Setiap submasalah kemudian diselesaikan menggunakan konsep yang sama, dengan mempertimbangkan kasus basis ketika hanya ada tiga titik kontrol. Hal ini menghasilkan kurva Bézier dengan kompleksitas waktu yang lebih efisien dibandingkan dengan pendekatan brute force.

6.2 Saran

1. Untuk meningkatkan efisiensi algoritma, diperlukan mempertimbangkan untuk mengoptimalkan rekursi dengan memperhitungkan teknik seperti memoisasi atau eliminasi ekstra rekursi jika memungkinkan.
2. Memperhatikan penanganan kasus khusus, seperti ketika hanya ada tiga titik kontrol, untuk menghindari overhead yang tidak perlu dalam pemrosesan.
3. Melakukan uji kinerja dan *benchmarking* untuk membandingkan kinerja algoritma divide-and-conquer dengan pendekatan brute force dalam berbagai skenario kasus uji. Ini akan membantu dalam mengevaluasi keefektifan algoritma.

BAB 7 LAMPIRAN

7.1 Github

https://github.com/Loxenary/Tucil2_13522134_13522157

7.2 Tabel Pemeriksaan

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.	✓	

DAFTAR PUSTAKA

Munir, Rinaldi. (2024). Algoritma Brute Force (Bagian 1). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf (diakses pada 15 Maret 2024).

Munir, Rinaldi. (2024). Algoritma Divide and Conquer (Bagian 1). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf (diakses pada 15 Maret 2024)

Munir, Rinaldi. (2024). Algoritma Divide and Conquer (Bagian 2). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf (diakses pada 15 Maret 2024)

Munir, Rinaldi. (2024). Algoritma Divide and Conquer (Bagian 3). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian3.pdf (diakses pada 15 Maret 2024)

https://www.summbit.com/blog/bezier-curve-guide/#quadratic-bezier-curve-point-code. (diakses pada 14 Maret 2024)

Bandara, Ravimal. 2014. Midpoint Algorithm (Divide and Conquer Method) for Drawing a Simple Bezier Curve. https://www.codeproject.com/script/Articles/ViewDownloads.aspx?aid=223159 (diakses pada 14 Maret 2024)

https://www.clear.rice.edu/comp360/lectures/old/SubdivisionTextNew.pdf .(diakses pada 14 Maret 2024)

https://javascript.info/bezier-curve. (diakses pada 14 Maret 2024)