LAPORAN TUGAS KECIL II

IF2211 STRATEGI ALGORITMA

SEMESTER II 2022-2023

PMembangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Disusun oleh:

Attara Majesta Ayub 13522139

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG 2024

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa saya ucapkan atas penuntasan Tugas Kecil II IF2211 Strategi Algoritma yang berjudul "Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer" ini.

Laporan ini merupakan dokumentasi dan penjelasan atas program dan algoritma yang telah saya implementasikan. Saya berusaha untuk merancang dan mengimplementasikan strategi *divide and conquer* juga *bruteforce* sedemikian rupa sehingga *bezier curve* dapat divisualisasikan beserta langkah-langkahnya.

Penyelesaian tugas besar ini tidak lepas dari bimbingan dan arahan yang diberikan oleh dosen pengampu mata kuliah Strategi Algoritma. Saya juga ingin mengucapkan terima kasih kepada asisten mata kuliah Strategi Algoritma yang telah memberikan informasi secara berkala melalui laman *Question and Answer*.

Saya menyadari bahwa tugas besar ini masih jauh dari sempurna. Oleh karena itu, saya sangat terbuka untuk menerima kritik dan saran yang konstruktif demi perbaikan di masa yang akan datang. Akhir kata, semoga tugas besar ini dapat bermanfaat bagi semua pihak yang berkepentingan.

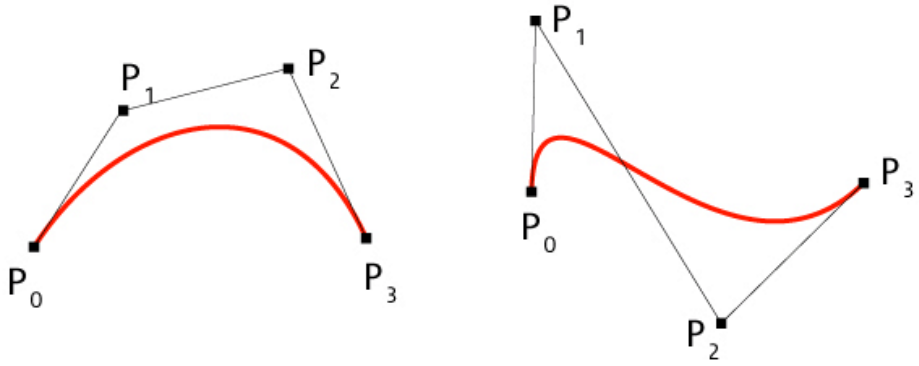
Sumedang, 19 Maret 2024,

13522139

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

1.1. Deskripsi Tugas



Bézier curve berasal dari nama Pierre Bézier, seorang insinyur yang menyebarluaskan konsep matematis tersebut sebagai kurva yang cocok untuk pekerjaan desain (1962), meskipun Pierre bukan orang pertama, atau satu-satunya, yang menginventaris jenis kurva ini. Kandidat lain yang diasumsikan sebagai penemu adalah de Casteljau. Namun, de Casteljau tidak mempublikasikan karyanya.

Kurva Bézier adalah bentuk dari fungsi "parametrik". Secara matematis, fungsi parametrik adalah istilah yang mewakili pemetaan dari *multiple inputs* menjadi satu *output*. Parameter yang digunakan dalam Kurva Bézier adalah t dengan $t \in [0,1]$, didefinisikan oleh sebuah set yang terdiri atas *control points*. Posisi-posisi *control points* terhadap satu sama lain akan membentuk bézier curve. Sehingga, kurva parametrik tidak mendefinisikan sebuah koordinat y dari koordinat x (*f(x)*) seperti fungsi umumnya. Tetapi, kurva menghubungkan nilai-nilai tersebut ke dalam variabel "kontrol". Kita dapat mengatur t dari negatif hingga positif tak hingga, dan koordinat (x,y) yang dihasilkan akan selalu berada pada lingkaran dengan radius 1 di sekitar titik pusat (0,0).

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P0 dan P1 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 Q0 yang berada pada garis yang dibentuk oleh P0 dan P1, maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh B(t) dari P0 ke P1. Misalnya ketika t = 0.25, maka B(t) adalah seperempat jalan dari titik P0 ke P1. sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan B(t) membentuk sebuah garis lurus dari P0 ke P1.

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P2, dengan P0 dan P2 sebagai titik kontrol awal dan akhir, dan P1 menjadi titik kontrol antara. Dengan menyatakan titik Q1 terletak diantara garis yang menghubungkan P1 dan P2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q0 berada, maka dapat dinyatakan sebuah titik baru, R0 yang berada diantara garis yang menghubungkan Q0 dan Q1 yang bergerak membentuk kurva Bézier kuadratik terhadap titik P0 dan P2. Berikut adalah uraian persamaannya.

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

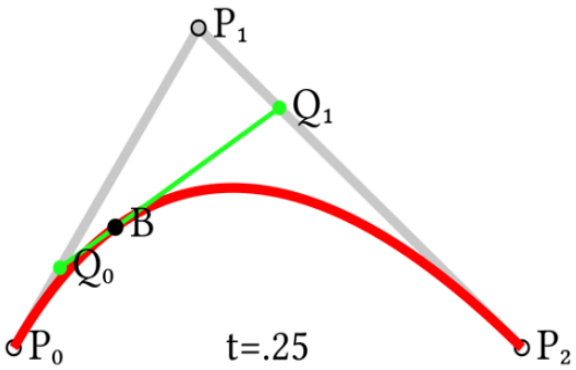
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

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

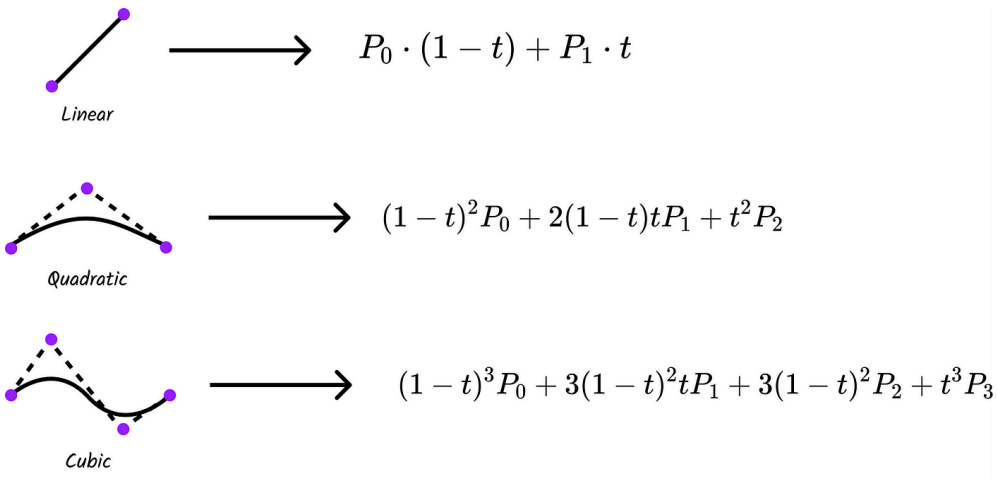
dengan melakukan substitusi nilai Q0 dan Q1, maka diperoleh persamaan sebagai berikut.

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

Berikut adalah ilustrasi dari kasus diatas



Secara singkat, posisi Q0 dapat langsung didefinisikan sesuai jumlah *control points* yang berbeda pula. Berikut adalah fungsi untuk kurva Bézier kuadratik dan kubik, dan seterusnya dengan rumus yang sama.



BAB II
LANDASAN TEORI

2.1 Algoritma Divide and Conquer

Algoritma Divide and Conquer merupakan strategi yang awalnya digunakan dalam konteks militer dengan nama "divide ut imperes," dan kemudian menjadi strategi fundamental dalam ilmu komputer dengan nama Divide and Conquer. Secara definitif, *divide* berarti pembagian persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil. C*onquer* berarti penyelesaian masing-masing upa-persoalan secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar. Terakhir, c*ombine*, penggabungan solusi masing-masing upa-persoalan sehingga membentuk solusi dari persoalan semula.

Algoritma Divide and Conquer umumnya didefinisikan dengan skema rekursif. Kompleksitas waktu algoritma Divide and Conquer dihitung berdasarkan kompleksitas waktu untuk memecahkan setiap upa-persoalan dan menggabungkan solusinya. Contoh persoalan yang dapat diselesaikan dengan algoritma Divide and Conquer meliputi pencarian minimum dan maksimum, menghitung perpangkatan, pengurutan (sorting), mencari sepasang titik terdekat, *convex hull*, perkalian matriks, perkalian bilangan bulat besar, perkalian dua buah polinom, *skyline problem*, dan lain-lain.

Aplikasi pada Permasalahan Kurva Bézier

2.2 Algoritma Brute-Force

Algoritma Brute-force merupakan pendekatan langsung dalam memecahkan masalah. Pendekatan ini sederhana, langsung, dan jelas. Semua kemungkinan solusi dari persoalan akan dikalkulasi dan diiterasi.

Kelebihan dari algoritma brute-force termasuk kemampuannya untuk memecahkan sebagian besar masalah, sederhana dan mudah dimengerti, serta menghasilkan algoritma standar untuk tugas-tugas komputasi penting. Namun, algoritma ini juga memiliki kelemahan, seperti jarang menghasilkan solusi yang efektif, lambat dalam eksekusi, dan kurang kreatif dibandingkan dengan teknik pemecahan masalah lainnya. *Time complexity* algoritma ini berbanding lurus dengan jumlah data yang harus dikalkulasi.

BAB III
APLIKASI ALGORITMA

5.1 Aplikasi

Divide and Conquer

Dalam implementasi algoritma, konsep *Divide and Conquer* diterapkan secara teknis dengan cara membagi segmen kurva menjadi dua bagian, sesuai dengan langkah *divide*. Ini dilakukan melalui fungsi `divide_conquer`, yang menerima input berupa titik kontrol (`control_points`), jumlah iterasi (`iterations`), serta daftar kosong untuk menyimpan titik-titik hasil kurva Bezier (`bezier`) dan titik-titik tengah (`midpoints`). Pada setiap iterasi, algoritma menghitung titik tengah dari dua titik kontrol yang diberikan dengan memanggil fungsi `calculate_midpoint`.

Tiga titik tengah akan dikalkulasi. Pertama, titik tengah dari titik kontrol pertama dan kedua. Lalu, titik tengah dari titik kontrol kedua dan ketiga. Terakhir, titik tengah dari hasil dua titik kontrol sebelumnya. Penyelesaian setiap segmen secara terpisah ini sejalan dengan konsep *conquer.* Selanjutnya, hasil kalkulasi terakhir akan menjadi titik yang sejalur dengan kurva bezier. Proses ini diulangi secara rekursif untuk setiap segmen yang dihasilkan. Setelah itu, algoritma menghasilkan daftar titik-titik tengah (nama variabel: midpoints) dan titik-titik kontrol tambahan yang membentuk kurva Bezier (nama variabel: bezier).

Array titik-titik tengah (midpoints) dan titik-titik kontrol tambahan yang membentuk kurva Bezier (bezier) akan divisualisasikan dengan library matplotlib dengan rincian langkahnya (pembuatan garis dan titik koordinat). Dengan demikian, algoritma ini mengaplikasikan konsep Divide and Conquer dalam membangun kurva Bezier dengan membagi masalah menjadi submasalah yang lebih kecil, menyelesaikan setiap submasalah secara terpisah, dan menggabungkan solusi-solusi tersebut untuk mendapatkan solusi akhir.

Brute-force

Dalam implementasi algoritma brute force, pendekatan yang digunakan adalah dengan mencoba titik-titik pada interval parameter secara berurutan untuk

setiap iterasi. Algoritma ini digunakan untuk menghitung titik-titik pada kurva Bezier dengan presisi tertentu sesuai dengan jumlah iterasi yang diberikan. Fungsi `brute_force` menerima input berupa titik kontrol (`control_points`) dan jumlah iterasi (`iterations`). Pada setiap iterasi, algoritma membagi interval parameter menjadi sejumlah bagian yang semakin kecil, dengan langkah (step) yang didefinisikan berdasarkan iterasi saat ini. Kemudian, algoritma mencoba titik-titik pada interval parameter yang telah dibagi tersebut untuk mengevaluasi posisi pada kurva Bezier.

Pada setiap titik yang dicoba, algoritma menggunakan formula umum untuk kurva Bezier, yaitu menggunakan persamaan polinomial Bezier yang didefinisikan oleh koefisien polinomial Bezier. Di sini, koefisien polinomial Bezier dihitung menggunakan rumus binomial dan parameter `t` yang berkisar antara 0 hingga 1. Algoritma ini melakukan perulangan untuk setiap iterasi dan setiap titik kontrol untuk menghitung posisi titik pada kurva Bezier. Titik-titik yang dihasilkan kemudian disimpan dalam daftar `points`.

Proses ini diulangi hingga mencapai jumlah iterasi yang diinginkan. Setelah itu, daftar titik-titik yang dihasilkan akan menjadi representasi dari kurva Bezier dengan presisi sesuai dengan jumlah iterasi yang ditentukan. Selanjutnya, hasil dari algoritma brute force dapat divisualisasikan dengan menggunakan library matplotlib, dengan mengplot titik-titik yang telah dihasilkan. Dengan demikian, algoritma ini menggunakan pendekatan brute force dengan mencoba banyak titik pada interval parameter untuk menghasilkan kurva Bezier dengan presisi yang diinginkan.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Divide and Conquer

4.1.1. Import

```
import numpy as np
```

4.1.2. Functions

def divide_conquer(control_points, iterations, bezier=[], midpoints=[], current_iteration=0)

```
def divide_conquer(control_points, iterations, bezier=[], midpoints=[], current_iteration=0):
      if iterations == 0:
      return control_points, bezier, midpoints
      else:
      midpoint_1 = calculate_midpoint(control_points[0], control_points[1])
      midpoint_2 = calculate_midpoint(control_points[1], control_points[2])
      midpoint_3 = calculate_midpoint(midpoint_1, midpoint_2)

      bezier.append(midpoint_3)
      midpoints.append([midpoint_1, midpoint_2, midpoint_3])

      # left branch
      divide_conquer([control_points[0], midpoint_1, midpoint_3], iterations - 1, bezier, midpoints, current_iteration + 1)
      # right branch
      divide_conquer([midpoint_3, midpoint_2, control_points[2]], iterations - 1, bezier, midpoints, current_iteration + 1)
```

```
    return bezier, midpoints
```

calculate_midpoints(point1: any, point2: any, point3: any)

```
def calculate_midpoint(point1, point2):
    return ((point1[0] + point2[0]) / 2,
(point1[1] + point2[1]) / 2)
```

4.2 Implementasi Algoritma Brute-Force

4.1.1. Import

```
import numpy as np
```

4.1.2. Functions

brute_force(control_points, iterations)

Procedure

```
def brute_force(control_points, iterations):
    points = []
    for i in range(1, iterations + 1):
    step = 1 / (2 ** i)
    for t in np.arange(0, 1 + step, step):
        x, y = 0, 0
        n = len(control_points)
        for j, point in
enumerate(control_points):
            c = np.math.comb(n - 1, j) * (1 -
t) ** (n - 1 - j) * t ** j
            x += c * point[0]
            y += c * point[1]
        if i == iterations:
            points.append((x, y))
    return points
```

calculate(control_points, iterations, algorithm)

```
def calculate(control_points, iterations,
algorithm):
    start_time = time.time()
    plt.figure(figsize=(7, 5))
    plt.title('Bézier Curve')
```

```
      plt.xlabel('X')
      plt.ylabel('Y')
      plt.grid(True)
      control_x = [point[0] for point in
control_points]
      control_y = [point[1] for point in
control_points]
      plt.plot(control_x, control_y,
color='#FC8EAC', linestyle='-', label='Control
Points')
      print("Start time:", start_time)

      if algorithm == 'bf':
      bezier = []
      bezier = brute_force(control_points,
iterations)
      end_time = time.time()
      print("End time:", end_time)

      for points in bezier:
            plt.pause(0.025)
            plt.plot(points[0], points[1], 'ko')

      bezier.extend([control_points[0],
control_points[-1]])
      sorted_bezier = sorted(bezier, key=lambda
point: point[0])

      plt.pause(1)
      plt.plot(*zip(*sorted_bezier), color='g',
label='Brute Force')
      execution_time = format((end_time -
start_time) * 1000, '.20f')
      print("Execution Time:", execution_time, "ms")
      elif algorithm == 'dac':
      bezier = []
      midpoints = []
      bezier, midpoints =
divide_conquer(control_points, iterations)
```

```
      end_time = time.time()
      print("End time:", end_time)

      for pair in midpoints:
            plt.pause(0.05)
            plt.plot(pair[0][0], pair[0][1], 'ko',
alpha=0.25, markersize=5)
            plt.pause(0.025)
            plt.plot([pair[0][0], pair[1][0]],
[pair[0][1], pair[1][1]], color='b', alpha=0.25)
            plt.plot(pair[1][0], pair[1][1], 'ko',
alpha=0.25, markersize=5)
            plt.pause(0.025)
            plt.plot(pair[2][0], pair[2][1], 'ko',
alpha=1, markersize=5)
            plt.text(pair[2][0] + 0.1, pair[2][1],
f'  ({pair[2][0]:.2f}, {pair[2][1]:.2f})',
fontsize=8, ha='left', va='center')
            plt.pause(0.025)

      bezier.extend([control_points[0],
control_points[-1]])
      sorted_bezier = sorted(bezier, key=lambda
point: point[0])

      plt.plot(*zip(*sorted_bezier), color='g',
label='Divide And Conquer')
      execution_time = format((end_time -
start_time) * 1000, '.10f')
      print("Execution Time:", execution_time, "ms")

      # Adjusting the scale of the plot based on
input data
      min_x = min(min(control_x), min([point[0] for
point in bezier]))
      max_x = max(max(control_x), max([point[0] for
point in bezier]))
      min_y = min(min(control_y), min([point[1] for
```

```
point in bezier]))
      max_y = max(max(control_y), max([point[1] for
point in bezier]))

      plt.xlim(min_x - 1, max_x + 1)
      plt.ylim(min_y - 1, max_y + 1)

      print("Close the plot to exit...")

      plt.legend()
      plt.grid(True)
      plt.show()
```

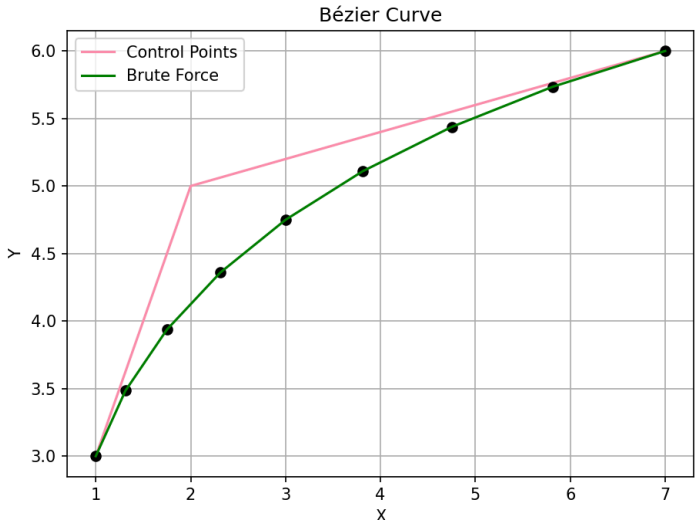
4. 3. Pengujian

1. Testcase 1

```
Enter the x and y coordinates of control point 1, separated by a space: 1 3
Enter the x and y coordinates of control point 2, separated by a space: 2 5
Enter the x and y coordinates of control point 3, separated by a space: 7 6
Enter the number of iterations: 3
```

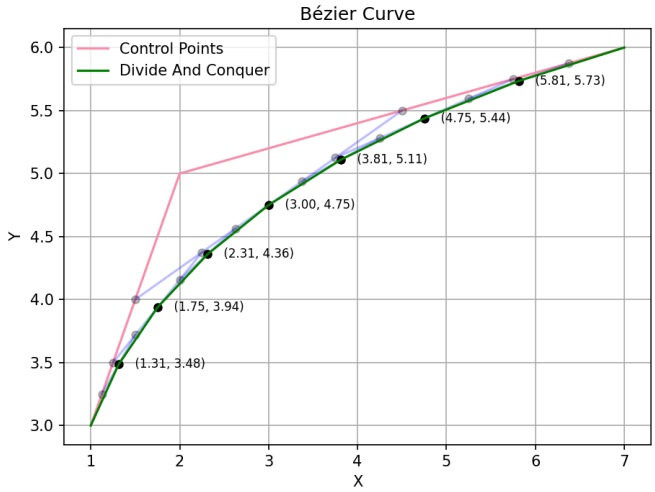
Brute force

Execution Time: 1.0001659393 ms



Divide and Conquer

Execution Time: 1.0752677917 ms

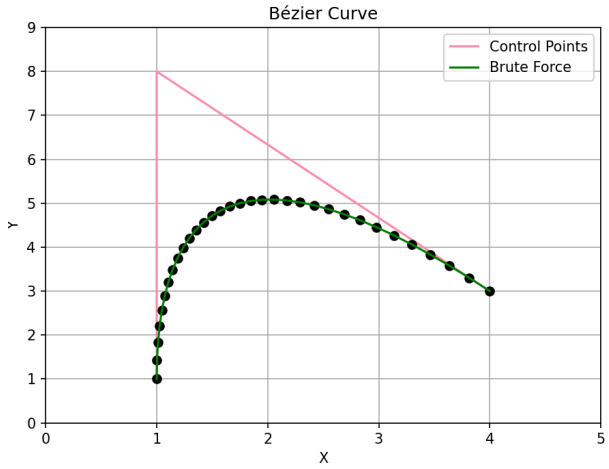


2. Testcase 2

```
Enter the x and y coordinates of control point 1, separated by a space: 1 1
Enter the x and y coordinates of control point 2, separated by a space: 1 8
Enter the x and y coordinates of control point 3, separated by a space: 4 3
Enter the number of iterations: 5
```

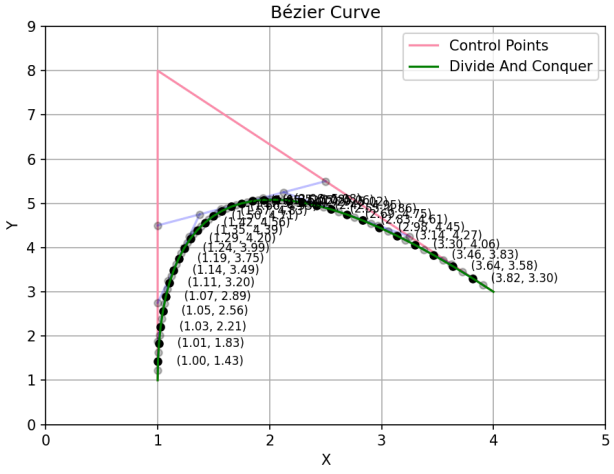
Brute force

Execution Time: 348.84977340698242187500 ms



Divide and Conquer

Execution Time: 316.0011768341 ms

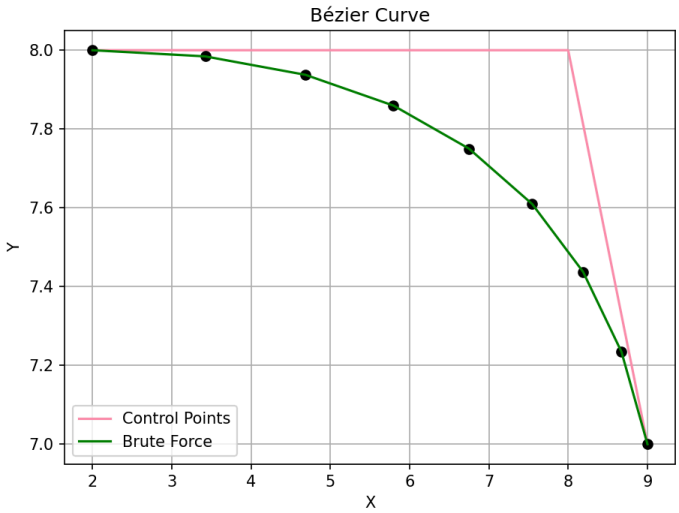


3. Testcase 3

```
Enter the x and y coordinates of control point 1, separated by a space: 1 7
Enter the x and y coordinates of control point 2, separated by a space: 8 6
Enter the x and y coordinates of control point 3, separated by a space: 9 1
Enter the number of iterations: 3
```

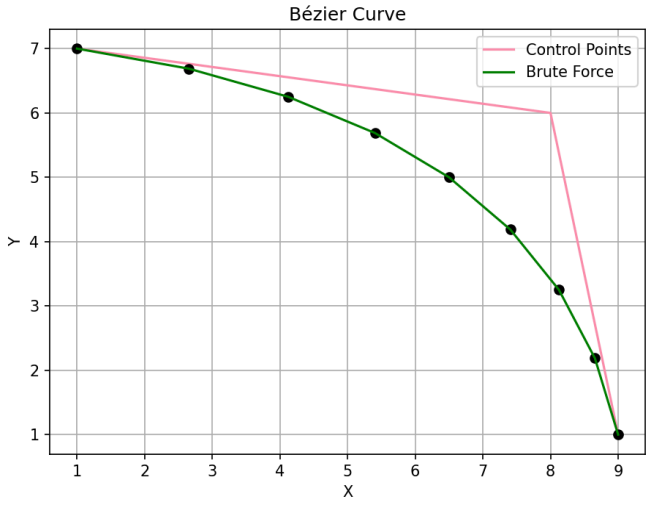
Brute force

Execution time: 0.000000000 ms



Divide and Conquer

Execution time: 0.000000000 ms

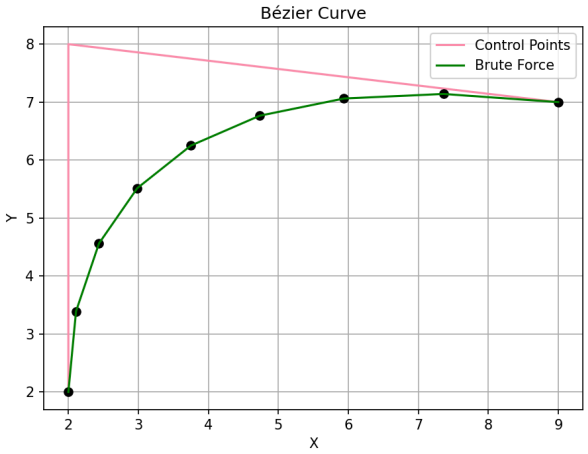


4. Testcase 4

```
Enter the x and y coordinates of control point 1, separated by a space: 2 2
Enter the x and y coordinates of control point 2, separated by a space: 2 8
Enter the x and y coordinates of control point 3, separated by a space: 9 7
Enter the number of iterations: 3
```

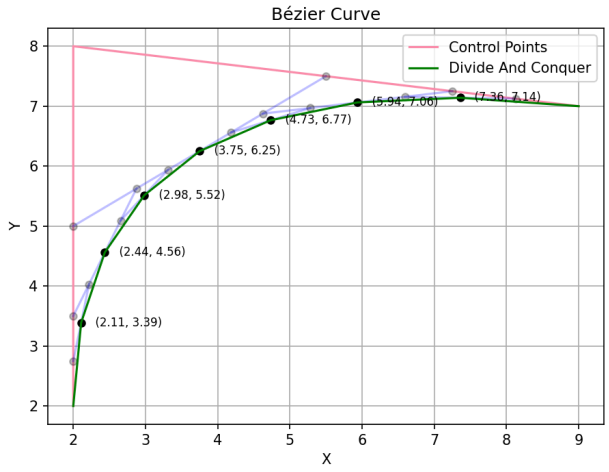
Brute force

Execution time: 0.000000000 ms



Divide and Conquer

Execution time: 0.000000000 ms

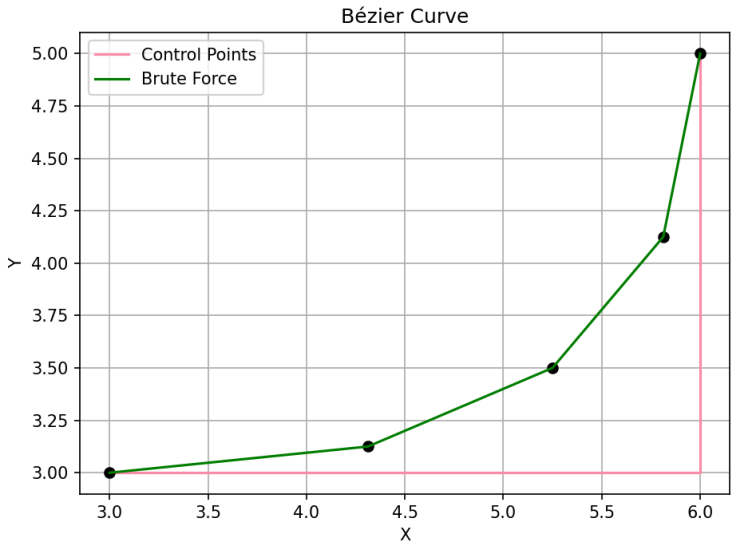


5. Testcase 5

```
Enter the x and y coordinates of control point 1, separated by a space: 6 5
Enter the x and y coordinates of control point 2, separated by a space: 6 3
Enter the x and y coordinates of control point 3, separated by a space: 3 3
Enter the number of iterations: 2
```

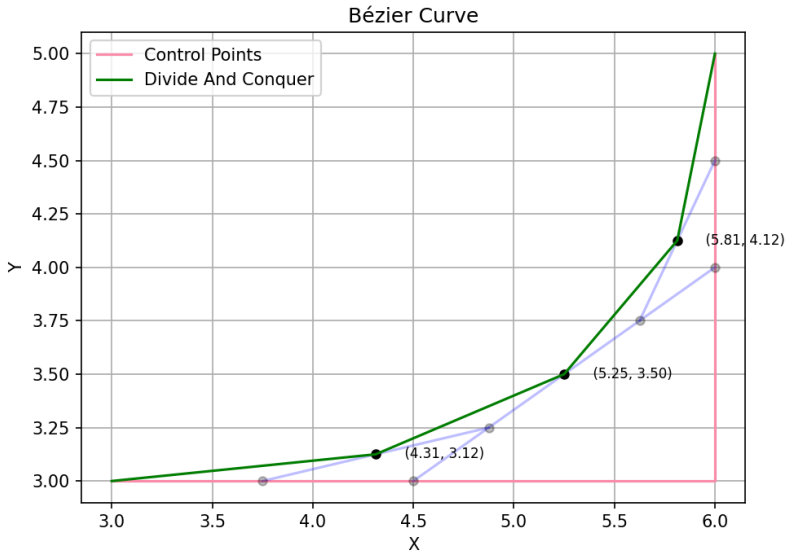
Brute force

Execution time: 0.000000000 ms



Divide and Conquer

Execution time: 0.000000000 ms

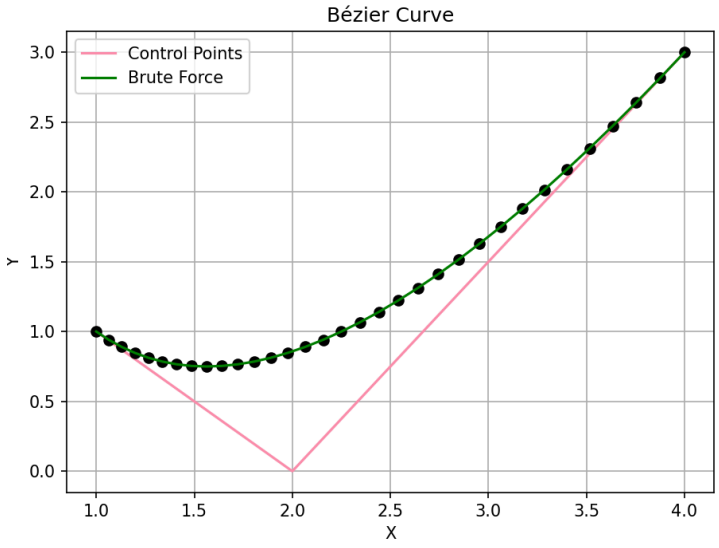


6. Testcase 6

```
Enter the x and y coordinates of control point 1, separated by a space: 4 3
Enter the x and y coordinates of control point 2, separated by a space: 2 0
Enter the x and y coordinates of control point 3, separated by a space: 1 1
Enter the number of iterations: 5
```

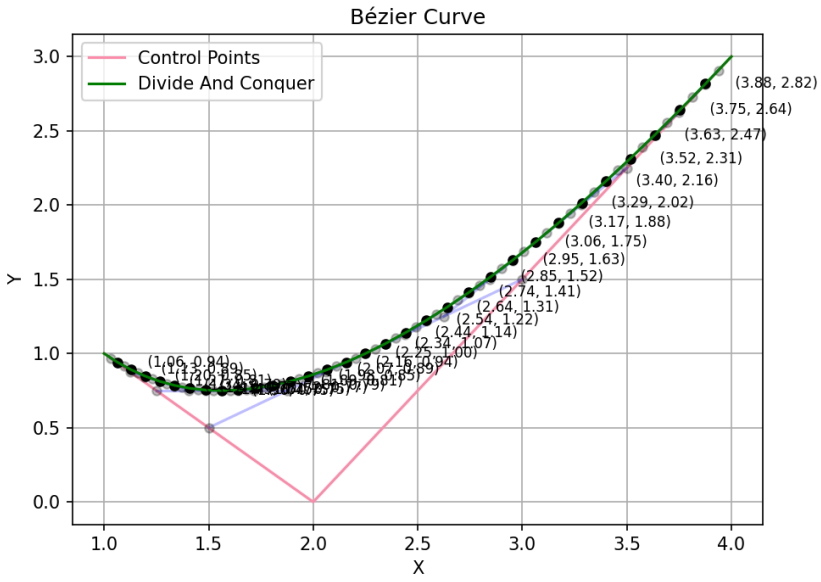
Brute force

Execution time: 0.000000000 ms



Divide and Conquer

Execution time: 0.000000000 ms


Bézier Curve
Control Points
Divide And Conquer
(3.88, 2.82)
(3.75, 2.64)
(3.63, 2.47)
(3.52, 2.31)
(3.40, 2.16)
(3.29, 2.02)
(3.17, 1.88)
(3.06, 1.75)
(2.95, 1.63)
(2.85, 1.52)
(2.74, 1.41)
(2.64, 1.31)
(2.54, 1.22)
(2.44, 1.14)
(2.34, 1.07)
(2.25, 1.00)
(2.16, 0.94)
(1.06, 0.94)
X
Y

BAB V
ANALISIS ALGORITMA

6.1 Analisis Efisiensi dan Efektivitas

Berikut adalah hasil analisis dari pengujian *testcase* dan analisis algoritma dalam *source code.*

5.1.1. Analisis Algoritma Divide and Conquer

Berikut adalah daftar tingkat efisiensi (diukur dengan kompleksitas waktu strategi menggunakan notasi Big O) masing-masing fungsi yang terlibat dalam kalkulasi *divide and conquer*:

1. **Divide_conquer**

   Pada fungsi ini, algoritma ini membagi kurva Bezier menjadi dua bagian dan mencari titik-titik yang berada di tengah-tengahnya secara rekursif . Ini merupakan pendekatan yang lebih efisien dibandingkan dengan brute force untuk menemukan titik-titik pada kurva Bezier. Kompleksitas waktu algoritma ini bergantung pada iterasi yang dilakukan dan jumlah titik kontrol. Tetapi dalam kasus terbaik, karena perhitungan hanya membagi dua titik kontrol pada setiap iterasi, kompleksitasnya bisa menjadi O(log(iterations)*n). Namun, dalam kasus terburuk, kompleksitasnya dapat menjadi O(iterations*n).

2. **calculate_midpoints**

   Pada fungsi ini, algoritma menghitung titik tengah antara dua titik kontrol dalam algoritma *divide and conquer*. Fungsi ini memiliki kompleksitas waktu konstan, yaitu O(1), karena tidak tergantung pada jumlah iterasi atau jumlah titik kontrol. Ini karena hanya melakukan operasi sederhana penambahan dan pembagian. Sehingga, tidak memperburuk kompleksitas algoritma secara keseluruhan.

5.1.2. Analisis Algoritma Brute-Force

Berikut adalah tingkat efisiensi (diukur dengan kompleksitas waktu strategi menggunakan notasi Big O) fungsi yang terlibat dalam kalkulasi *brute-force*:

1. **brute_force**

   Pada fungsi ini, algoritma menghitung titik-titik pada kurva Bezier dengan membagi interval parameter secara iteratif. Pendekatan ini sederhana tetapi memakan banyak waktu karena mencoba banyak titik pada interval parameter. Kompleksitas waktu algoritma ini adalah O(iterations*(n*2^iterations)), di mana iterations adalah jumlah iterasi yang diinginkan dan n adalah jumlah titik kontrol. Ini karena algoritma ini menggunakan nested loop, dengan jumlah iterasi pada iterasi luar dan pada setiap iterasi, melakukan perhitungan pada setiap titik kontrol. Jumlah iterasi dalam loop dalam (yang terkait dengan jumlah titik kontrol) berkembang eksponensial dengan setiap iterasi luar, karena pada setiap iterasi, interval parameter dibagi menjadi dua.

5.2 Perbandingan Algoritma

Dalam menentukan algoritma yang lebih baik, banyak kasus harus dipertimbangkan. Setelah *testing* yang dilakukan dengan berbagai titik kontrol *random,* didapatkan bahwa jika jumlah iterasi yang diberikan sedikit, maka algoritma *brute-force* akan mengeksekusi lebih cepat. Namum, ketika iterasi yang diberikan adalah jumlah yang cukup besar, maka kalkulasi *brute-force* menjadi ‘lambat’. Dalam hal ini, algoritma *divide and conquer* dapat menyaingi kecepatan perhitungan dari algoritma *brute-force.*

Sebelumnya telah didapatkan kompleksitas waktu untuk *divide and conquer* yakni O(iterations * n) atau O(log(iterations) * n) dan *brute-force* yakni O(iterations * (n * 2^iterations)). Secara matematis, kita dapat memeriksa kapan kompleksitas waktu dari algoritma *Divide and Conquer* lebih kecil dari kompleksitas waktu dari algoritma Brute Force. Berikut perhitungannya.

- Jika iterations = 1, maka:

Divide and Conquer: O(n)
Brute Force: O(n * 2^1) = O(2n)
Dalam iterasi ini, Kompleksitas waktu *Divide and Conquer* lebih kecil dari Brute Force.

- Jika iterations = 2, maka:
  Divide and Conquer: O(n)
  Brute Force: O(n * 2^2) = O(4n)
  Kompleksitas waktu Divide and Conquer masih lebih kecil daripada Brute Force.

- Namun, jika iterations cukup besar, misalnya iterations = 10, maka:
  Divide and Conquer: O(n)
  Brute Force: O(n * 2^10) = O(1024n)
  Di sini, kompleksitas waktu dari Brute Force jauh lebih besar daripada Divide and Conquer.

Kesimpulannya, dalam kasus-kasus di mana jumlah iterasi cukup kecil, Divide and Conquer akan lebih cepat daripada Brute Force. Namun, saat jumlah iterasi meningkat, Brute Force akan menjadi lebih lambat karena pertumbuhan eksponensial kompleksitasnya, memenangkan algoritma *divide and conquer* dalam hal waktu eksekusi.

BAB VI
PENUTUP

Kesimpulan

Kedua algoritma, yaitu Divide and Conquer serta Brute Force, merupakan pendekatan yang berbeda dalam menemukan kurva Bezier yang melewati beberapa titik kontrol. Meskipun keduanya memiliki kelebihan dan kelemahan masing-masing, analisis efisiensi menunjukkan bahwa algoritma Divide and Conquer cenderung lebih efisien dibandingkan Brute Force, terutama untuk jumlah iterasi yang cukup besar. Namun, efisiensi algoritma dapat sangat bervariasi tergantung pada kondisi masalah yang spesifik, seperti jumlah iterasi yang digunakan dan kompleksitas titik kontrol.

Saran

Lakukan eksplorasi algoritma lain selain algoritma *Divide and Conquer* dan *Brute Force*, seperti algoritma *dynamic programming* atau *backtracking* dapat memberikan wawasan baru dan solusi yang lebih efisien terhadap masalah kurva Bezier.

Refleksi terhadap Tugas Besar

Walaupun tidak semua bonus dikerjakan, dan mungkin masih terdapat kasus yang belum di-*handle,* tugas ini memberikan saya pengalaman berharga dalam memahami serta mengimplementasikan algoritma-algoritma yang berbeda dalam menyelesaikan masalah kurva Bezier. Manajemen waktu yang baik dan ketelitian menjadi kunci keberhasilan dalam menyelesaikan tugas ini. Saya juga menyadari pentingnya menghadapi kegagalan sebagai bagian dari proses pembelajaran, yang dapat menjadi langkah untuk memperbaiki dan memperbaiki solusi saya di masa depan.

BAB VII

LAMPIRAN

7.1 GitHub Repository

https://github.com/attaramajesta/Tucil2_13522139

7.2 Tabel Ketercapaian Program

Tabel 6.2.1. Tabel Ketercapaian Program

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.	☻	