

Laporan Tugas Kecil 2
IF2211 Strategi Algoritma
Pembentukan Kurva Bézier dengan Algoritma *Divide*
And Conquer



Disusun oleh:

Marvel Pangondian (13522075)

Berto Richardo Togatorop (13522118)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

Daftar Isi

Daftar Isi.....	2
Bab I	
Algoritma Divide and Conquer.....	3
1.1 Pembentukan Kurva Bézier.....	3
1.2 Alur Program.....	4
1.3 Analisis Implementasi Algoritma Program.....	4
1.3.1 Analisis Prosedur Algoritma Divide and Conquer Untuk 3 Titik Kontrol.....	5
1.3.2 Analisis Prosedur Algoritma Brute Force Untuk 3 Titik Kontrol.....	5
1.3.3 Analisis Prosedur Algoritma Divide and Conquer Untuk N Titik Kontrol.....	6
1.3.4 Analisis Prosedur Algoritma Brute Force.....	7
1.4 Analisis Kompleksitas Program.....	8
1.4.1 Analisis Kompleksitas Algoritma Divide and Conquer Untuk 3 Titik Kontrol.....	8
1.4.2 Analisis Kompleksitas Algoritma Brute Force Untuk 3 Titik Kontrol.....	8
1.4.3 Analisis Kompleksitas Algoritma Divide and Conquer Untuk N Titik Kontrol.....	9
1.4.4 Analisis Kompleksitas Algoritma Brute Force Untuk N Titik Kontrol.....	10
1.4.5. Analisis Perbandingan kompleksitas Algoritma Brute Force dan Divide and Conquer Untuk N Titik Kontrol.....	10
Bab II	
Source Code.....	11
2.1 File Program Utama.....	11
2.2 File Algoritma Divide and Conquer.....	12
2.3 File Algoritma Brute Force	
File algoritma Brute Force adalah brute_force_bezier.py. Isi file tersebut adalah sebagai berikut :.....	13
2.4 Kelas Kurva Bézier.....	14
2.5 File Input/Output.....	19
Bab III.....	24
TEST.....	24
3.1 Eksperimen dengan 3 Titik Kontrol dan D Iterasi.....	24
3.2 Eksperimen dengan N Titik Kontrol dan D Iterasi.....	32
Bab IV	
Kesimpulan dan Saran.....	42
4.1 Kesimpulan.....	42
4.2 Saran.....	42
Bab V	
LAMPIRAN.....	43
5.1 Link Repository.....	43
5.2 Tabel Checklist.....	43

Bab I

Algoritma *Divide and Conquer*

1.1 Pembentukan Kurva Bézier

Kurva Bézier adalah kurva halus yang memiliki banyak aplikasinya terutama dalam desain grafis, animasi, desain font komputer, manufaktur, dan lain - lain. Kurva Bézier dibentuk dari serangkaian titik kontrol yang menentukan bentuk dan arahnya, mulai dari titik kontrol pertama (P0) hingga titik kontrol terakhir (Pn), dengan 'n' menandakan order kurva tersebut. Titik kontrol pertama dan terakhir menandai ujung kurva, sedangkan titik-titik kontrol lainnya, yang berperan sebagai titik kontrol antara, biasanya tidak berada pada kurva itu sendiri.

Kurva Bézier yang paling sederhana adalah kurva Bézier linear ($n = 1$) dimana kurva yang akan terbentuk adalah garis linear yang menghubungkan titik kontrol P0 dengan P1. Untuk dua titik kontrol, terdapat titik Q0 di antara kedua titik tersebut yang dapat dicari dengan rumus berikut :

$$Q_0 = B(t) = (1 - t)P_0 + tP_1, 0 \leq t \leq 1$$

Kurva Bézier linear merupakan kurva yang terbentuk dari seluruh nilai hasil Q0 rentang variasi t dari 0 sampai 1. Kurva Bézier kuadrat adalah kurva yang terbentuk dari 3 titik kontrol (P0, P1, P2). Q1 adalah titik yang terletak di antara titik P1 dan P2, dan R0 adalah titik yang terletak antara titik Q0 dan Q1. Titik R0 dapat dicari dengan cara berikut :

$$Q_0 = B(t) = (1 - t)P_0 + tP_1, 0 \leq t \leq 1$$

$$Q_1 = B(t) = (1 - t)P_1 + tP_2, 0 \leq t \leq 1$$

$$R_0 = B(t) = (1 - t)Q_0 + tQ_1, 0 \leq t \leq 1$$

Dapat dilihat bahwa rumus untuk mencari R0 dapat diturunkan lagi menjadi :

$$R_0 = B(t) = (1 - t)[(1 - t)P_0 + tP_1] + t[(1 - t)P_1 + tP_2], 0 \leq t \leq 1$$

$$R_0 = B(t) = (1 - t)^2P_0 + (t - t^2)P_1 + (t - t^2)P_1 + t^2P_2, 0 \leq t \leq 1$$

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

Hasil akhir dari penurunan rumus R0 adalah sebagai berikut :

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

Sehingga, kurva Bézier kuadrat adalah kurva yang terbentuk dari seluruh nilai hasil R0 rentang variasi t dari 0 sampai 1 . Pada tugas ini, akan dibuat sebuah algoritma *Divide and Conquer* untuk membuat kurva Bézier kuadratik tanpa menggunakan rumus R0 yang sudah diturunkan diatas. Selain kurva Bézier kuadrat, akan dibahas generalisasi algoritma *Divide and Conquer* sehingga dapat membentuk kurva Bézier dengan N titik kontrol.

1.2 Alur Program

Program dimulai dengan meminta pengguna untuk memilih jenis algoritma yang ingin digunakan. Terdapat dua algoritma yang dapat digunakan pengguna untuk membuat kurva Bézier yakni (1) algoritma *Divide and Conquer* serta (2) algoritma *Brute Force*. Setelah memilih algoritma yang diinginkan, pengguna akan diminta untuk mengisi orde, titik kontrol, serta iterasi kurva Bézier yang akan dibuat. Berikutnya, untuk pilihan (1), pengguna dapat memilih untuk menampilkan hasil kurva Bézier dalam graf saja atau dalam bentuk animasi (bonus). Untuk pilihan (2), kurva Bézier akan ditunjukkan langsung dalam bentuk graf tanpa animasi. Pengguna kemudian memiliki opsi untuk menyimpan hasil kurva Bézier dalam bentuk .png pada folder test. Di akhiran, pengguna memiliki opsi untuk melanjutkan program (membuat kurva Bézier lain).

1.3 Analisis Implementasi Algoritma Program

Terdapat dua jenis algoritma yang digunakan untuk membuat kurva Bézier yakni algoritma *Divide and Conquer* serta algoritma *Brute Force*. Penulis sudah men-generalisasi algoritma sehingga program dapat menampilkan kurva bezier untuk orde n (n + 1 titik kontrol).

1.3.1 Analisis Prosedur Algoritma *Divide and Conquer* Untuk 3 Titik Kontrol

Pada algoritma ini, Karena jumlah titik kontrol konstan, maka hanya perlu memperhatikan iterasi (depth) algoritma saja. Langkah - langkah penyelesaian dengan *divide and conquer* adalah sebagai berikut :

1. Jika $\text{depth} = 0$, berarti pada tahap basis. Pada tahap ini, akan terjadi proses *Merge* sebuah array yang mengandung titik - titik yang diperlukan untuk membuat kurva, dengan titik awal dan akhir titik - titik kontrol saat itu.
2. Jika bukan di basis, maka akan dicari titik tengah antara titik kontrol P_0 dengan titik kontrol P_1 yang dinamakan Q_0 , titik kontrol P_1 dengan P_2 yang dinamakan Q_1 , serta titik Q_0 dan Q_1 yang dinamakan R_0 .
3. Setelah mencari masing titik tengah pada masing - masing titik, maka akan dipanggil cabang kanan dan cabang kiri fungsi secara rekursif. Untuk cabang kiri fungsi *divide and conquer* akan dipanggil dengan titik kontrol barunya yakni P_0 , Q_0 , dan R_0 serta iterasi pada cabang kiri adalah iterasi saat ini dikurang 1. Untuk Cabang kanan fungsi *divide and conquer* akan dipanggil dengan titik kontrol barunya yakni R_0 , Q_1 , P_1 serta iterasi pada cabang kanan adalah iterasi saat ini dikurang 1.

1.3.2 Analisis Prosedur Algoritma *Brute Force* Untuk 3 Titik Kontrol

Pada algoritma ini, Karena jumlah titik kontrol konstan, maka hanya perlu memperhatikan iterasi (depth) algoritma saja. Untuk k iterasi, jumlah titik yang dihasilkan untuk membuat kurva Bézier selalu $2^k + 1$, maka konsep *Brute Force* sangat sederhana yakni *loop* sebanyak $2^k + 1$ untuk menghasilkan $2^k + 1$ titik yang dibutuhkan. Langkah - langkah penyelesaian dengan *Brute Force* adalah sebagai berikut :

1. Pertama, dilakukannya *loop i* sampai 2^k dimana k adalah iterasi dan i menyatakan nomor loop, i dimulai dari 0 (total loop adalah $2^k + 1$).
2. Pada setiap *loop*, akan dicari t yaitu $\frac{i}{2^k}$. Nilai t akan digunakan untuk mencari letak 3 titik masing masing di antara titik kontrol P_0 dan P_1 (Q_0), P_1 dan P_2 (Q_1), serta Q_0 dengan Q_1 (R_0). Pencarian titik - titik tersebut dilakukan melalui linear interpolasi menggunakan t .

3. Hasil R0 kemudian dimasukkan kepada semua array, array tersebut nanti akan terisi semua titik yang diperlukan untuk membuat kurva Bézier.
4. *Loop* kemudian dilanjutkan sampai akhir (i adalah 2^k).

1.3.3 Analisis Prosedur Algoritma *Divide and Conquer* Untuk N Titik Kontrol

Pada algoritma ini, parameternya adalah *points*, *depth*, *curve_points*, dan *intermediates*. *Points* adalah kontrol point yang diterima. Parameter *depth* adalah jumlah iterasi. Parameter *curve_points* adalah titik kurva yang sudah didapatkan, yang pada pemanggilan awal adalah array kosong. Sementara itu, *intermediates* adalah titik kontrol turunan yang diperlukan untuk animasi pembentukan kurva. Algoritma ini nantinya akan menghasilkan titik-titik kurva dan titik-titik *intermediates*.

Langkah- langkah penyelesaian dengan *divide and conquer* adalah sebagai berikut :

1. Jika $depth = 0$, berarti di tahap basis. Di tahap inilah terjadi proses *MERGE* dengan cara mengembalikan titik kurva.
2. Jika bukan di basis, maka diambil titik kontrol awal dan akhir, serta dianalisis array kosong untuk menampung titik kontrol selanjutnya
3. Setelah itu, dilakukan iterasi untuk mencari parameter kontrol point untuk rekursi selanjutnya. Iterasi dilakukan hingga ditemukan satu titik kurva yang disetor dalam variabel *points*.
4. Di awal iterasi, dilakukan inisialisasi variable *temp* yang menyimpan titik-titik tengah dari control point.
5. Jika titik tengahnya sisa 1, maka hanya akan disimpan 1 nilai pada array final piece. Sementara itu, jika titik-titik tengahnya lebih dari 1, maka titik awalnya di-*append* pada awal *start_needed* dan nilai akhirnya di-*insert* ke awal array *last_needed*.
6. Setelah iterasi selesai, semua point dimasukkan ke array *all* dengan urutan *start_needed*, *final_piece*, dan *last_needed*.
7. Kemudian, dilakukan proses *DIVIDE* dengan cara memanggil fungsi *divide and conquer* untuk nilai di kiri dan di kanan. Point kontrol yang dipakai untuk bagian kiri adalah isi array *all* dari indeks 0 hingga index $\lceil \frac{len(all)}{2} \rceil$. Sementara

itu, nilai kontrol point untuk yang di kanan adalah array all dari indeks $\lceil \frac{\text{len}(\text{all})}{2} \rceil$ hingga ujung array all. Banyaknya $\lceil \frac{\text{len}(\text{all})}{2} \rceil$ pasti sama dengan jumlah kontrol poin di awal.

8. Kembalikan array `curve_points` yang berisi titik-titik kurva dan titik intermediates yang berisi titik-titik kontrol *intermediate* pembangun kurva

1.3.4 Analisis Prosedur Algoritma *Brute Force* Untuk N Titik Kontrol

Pada algoritma ini, parameternya adalah `points` dan `depth`. `Points` adalah kontrol point yang diterima. Parameter `depth` adalah jumlah iterasi. Algoritma ini nantinya akan menghasilkan titik-titik kurva sebanyak $2^k + 1$ dimana k adalah `depth` (iterasi).

Langkah-langkah penyelesaian dengan algoritma *brute force* adalah sebagai berikut :

1. Inisialisasi array `curve_point` yang akan menyimpan titik-titik kurva.
2. Inisialisasi nilai `i` dengan 0
3. Atur nilai `t` untuk setiap iterasi, yaitu $\frac{i}{2^k}$
4. Inisialisasi array `temporary` untuk menampung titik kontrol;
5. Jika array `temporary` hanya 1, maka nilai langsung di-*append* ke array baru.
Jika belum, maka dicari hasil interpolasi dari 2 titik yang bersebelahan pada array `temporary`. Misalkan array `temporary` berisi $P_0, P_1, P_2, P_3, \dots, P_n$. Maka akan dicari $Q_{01}, Q_{12}, \dots, Q_{n-1,n}$ dengan memakai rumus interpolasi linear B  zier

$$Q_{01} = (1 - t) * P_0 + t * P_1$$

$$Q_{12} = (1 - t) * P_1 + t * P_2$$

$$\dots$$

$$Q(n - 1), n = (1 - t) * P(n - 1) + t * P_n$$
 Titik-titik ini akan dimasukkan sebagai nilai baru array `temporary` dan akan diproses pada tahap ulang pada tahap 5.
6. Increment nilai `i` dan ulangi tahap 2 hingga 5
7. Kembalikan array yang berisi titik-titik kurva

1.4 Analisis Kompleksitas Program

1.4.1 Analisis Kompleksitas Algoritma *Divide and Conquer* Untuk 3 Titik Kontrol

Algoritma yang penulis terapkan merupakan fungsi rekursi dengan basis ketika jumlah iterasi = 0. Karena titik kontrol konstan, maka algoritma hanya akan fokus pada jumlah iterasi saja. Basis algoritma adalah ketika $k = 0$ (iterasi ke - 0), pada saat itu, hanya dilakukan satu operasi yakni mengembalikan titik kontrol awal dan titik kontrol akhir. Jika tidak basis, maka akan dilakukan 3 operasi (mencari Q_0 , Q_1 , R_1 sesuai penjelasan) dan secara rekursi memanggil dua cabang. Dengan demikian, dapat ditentukan $T(k)$ untuk algoritma pencarian kurva Bézier dengan *Divide and Conquer* sebagai berikut (analisis berikut hanya dalam bentuk teori, tidak sesuai dengan implementasi kode):

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

Persamaan rekursif ini dapat diselesaikan secara iteratif

$$T(k) = 2T(k-1) + 3$$

$$T(k) = 2(2T(k-2) + 3) + 3$$

$$T(k) = 2^2 T(k-2) + 2 \cdot 3 + 3$$

$$T(k) = 2^2 (2T(k-3) + 3) + 2 \cdot 3 + 3$$

$$T(k) = 2^3 T(k-3) + 2^2 \cdot 3 + 2 \cdot 3 + 3$$

$$T(k) = 2^k T(k-k) + (2^{k-1} + \dots + 2^2 + 2 + 1) \cdot 3$$

$$T(k) = 2^k T(0) + (2^k - 1) \cdot 3$$

$$T(k) = 2^k + (2^k - 1) \cdot 3$$

$$T(k) = (2^k - 1) \cdot 3 + 2^k$$

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

1.4.2 Analisis Kompleksitas Algoritma *Brute Force* Untuk 3 Titik Kontrol

Algoritma yang penulis terapkan merupakan algoritma dengan *loop* sebanyak $2^k + 1$, dimana setiap *loop* melakukan 3 operasi sesuai dengan penjelasan bagian 1.3.2. Dengan demikian, dapat ditentukan $T(k)$ untuk algoritma pencarian kurva

Bézier dengan *brute force* untuk 3 titik kontrol sebagai berikut (analisis berikut hanya dalam bentuk teori, tidak sesuai dengan implementasi kode):

$$T(k) = (2^k + 1) (3)$$

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

1.4.3 Analisis Kompleksitas Algoritma *Divide and Conquer* Untuk N Titik Kontrol

Algoritma yang penulis terapkan merupakan fungsi rekursi dengan basis ketika jumlah iterasi = 0. Misalkan n adalah jumlah titik kontrol dan k adalah jumlah iterasi. Untuk setiap pemanggilan rekursi, dilakukan iterasi bersarang agar menghasilkan 1 buah titik kontrol. Iterasi dilakukan sebanyak n kali dan di dalam iterasi tersebut dilakukan iterasi sebanyak $n, n - 1, n - 2 \dots 1$. Melalui informasi tersebut, dapat ditentukan $T(n, k)$ untuk algoritma pencarian kurva Bézier dengan *Divide and Conquer* sebagai berikut (analisis berikut **sesuai** dengan implementasi kode penulis) :

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

Persamaan rekursif ini dapat diselesaikan secara iteratif

$$T(n, k) = \frac{n(n-1)}{2} + 2T(n, k - 1)$$

$$T(n, k) = \frac{n(n-1)}{2} + 2T(n, k - 1)$$

$$T(n, k) = \frac{n(n-1)}{2} + 2\left(\frac{n(n-1)}{2} + 2T(n, k - 2)\right)$$

$$T(n, k) = \frac{n(n-1)}{2} + 2\frac{n(n-1)}{2} + 2^2T(n, k - 2)$$

$$T(n, k) = \frac{n(n-1)}{2} + 2\frac{n(n-1)}{2} + 2^2\frac{n(n-1)}{2} + 2^3T(n, k - 3)$$

$$T(n, k) = \frac{n(n-1)}{2} + 2\frac{n(n-1)}{2} + 2^2\frac{n(n-1)}{2} + \dots + 2^{k-1}\frac{n(n-1)}{2} + 2^kT(n, k - k)$$

$$T(n, k) = \frac{n(n-1)}{2} + 2\frac{n(n-1)}{2} + 2^2\frac{n(n-1)}{2} + \dots + 2^{k-1}\frac{n(n-1)}{2} + 2^kT(n, 0)$$

$$T(n, k) = \frac{n(n-1)}{2} + 2\frac{n(n-1)}{2} + 2^2\frac{n(n-1)}{2} + \dots + 2^{k-1}\frac{n(n-1)}{2} + 2^k$$

$$T(n, k) = (1 + 2 + 2^2 + \dots + 2^{k-1}) \frac{n(n-1)}{2} + 2^k$$

$$T(n, k) = (2^k - 1) \frac{n(n-1)}{2} + 2^k$$

$$T(n, k) = 2^k \frac{n^2 - n}{2} - \frac{n^2 - n}{2} + 2^k$$

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

1.4.4 Analisis Kompleksitas Algoritma *Brute Force* Untuk N Titik Kontrol

Misalkan n adalah jumlah titik kontrol dan k adalah jumlah iterasi. Pada algoritma *brute force*, dilakukan iterasi sebanyak $2^k + 1$ kali dan di dalam iterasi itu dilakukan iterasi bersarang untuk mencari titik-titik kurva. Iterasi dilakukan sebanyak n kali dan di dalam iterasi tersebut dilakukan iterasi sebanyak $n, n - 1, n - 2 \dots 1$. Dengan demikian, dapat ditentukan $T(n, k)$ untuk algoritma pencarian kurva Bézier dengan *brute force* sebagai berikut :

$$T(n, k) = (2^k + 1) \left(\frac{n(n-1)}{2} \right)$$

$$T(n, k) = 2^k \frac{n(n-1)}{2} + \frac{n(n-1)}{2}$$

$$T(n, k) = 2^k \frac{n^2 - n}{2} + \frac{n^2 - n}{2}$$

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

1.4.5. Analisis Perbandingan kompleksitas Algoritma *Brute Force* dan *Divide and Conquer* Untuk N Titik Kontrol

Melalui analisis kompleksitas di atas, dapat dilihat bahwa kompleksitas big-O untuk mencari kurva bezier baik menggunakan algoritma *divide and conquer* ataupun *brute force* sama-sama $O(2^k n^2)$. Namun, terdapat perbedaan kompleksitas $T(n)$ untuk kedua metode ini.

$$T(n, k)_{dnc} = 2^k \frac{n^2 - n}{2} - \frac{n^2 - n}{2} + 2^k$$

$$T(n, k)_{bruteforce} = 2^k \frac{n^2 - n}{2} + \frac{n^2 - n}{2}$$

Perlu diketahui bahwa $T(n, k)$ di sini adalah banyaknya iterasi yang dilakukan oleh tiap algoritma. Jadi, banyaknya operasi yang dilakukan mungkin berbeda. Kecepatan eksekusi kedua algoritma mungkin cukup berbeda pada skala yang kecil. Namun, pada skala yang besar, perbedaan waktu eksekusinya tidak terlalu jauh dan dapat diabaikan karena kompleksitas big-O nya sama.

Bab II

Source Code

2.1 File Program Utama

File program utama (main.py) yang akan dijalankan pertama, file ini cukup singkat. Berikut *source code* program utama :

1. Modul eksternal

```
from IO import *
```

2. Source Code

```
print("=====")
print("Bézier Curve Program")
print("By : ")
print("Marvel Pangondian (13522075)")
print("Berto Richardo Togatorop (13522118)")

end = False
while not end :
    method = method_input()
    bezier_curve = bezier_attribute_input(method)
    show_output(bezier_curve,method)
    save_fig(bezier_curve)

    cont = input("Do you wish to continue?(y/n) ")
    end = (cont != "y")
```

2.2 File Algoritma *Divide and Conquer*

File algoritma *Divide and Conquer* terletak adalah `dnc_bezier.py`. Isi file tersebut adalah sebagai berikut :

1. Modul eksternal

```
import numpy as np
from typing import List, Tuple
```

2. Fungsi `divide_and_conquer_bezier`

Fungsi untuk menghasilkan kurva Bézier menggunakan algoritma *Divide and Conquer*. Fungsi menerima `points` (titik kontrol), `depth` (iterasi kurva), `curve_points` (array hasil titik yang dibutuhkan untuk membuat kurva Bézier, array ini akan terbentuk sejalan rekursif fungsi), `intermediates` (array titik tengah yang dihasilkan sejalan rekursif fungsi). Fungsi ini akan mengembalikan `curve_points` dan `intermediates`.

```
# function to get the curve points and intermediates for the bezier curve
def divide_and_conquer_bezier(
    points: List[Tuple[float, float]],
    depth: int,
    curve_points: [],
    intermediates: []
) -> Tuple[List[Tuple[float, float]], List[Tuple[float, float]]]:

    # recursion basis
    if depth == 0:
        # merge processes
        curve_points.extend([points[0], points[-1]])
        return curve_points, intermediates

    start_needed = [points[0]]
    last_needed = [points[-1]]
    final_piece = []

    # iteration to find the curve points and the parameter points for the
    next recursion
    while (len(points) != 1):
        temp = []
        for i in range(len(points) - 1):
            temp.append( (points[i] + points[i+1])/2 )
```

```

        points = temp
        intermediates.append(temp)
        if (len(points) == 1):
            final_piece.append(points[0])
        else :
            start_needed.append(temp[0])
            last_needed.insert(0,temp[-1])

    # List all the points
    all = []
    all.extend(start_needed)
    all.extend(final_piece)
    all.extend(last_needed)

    # recursion that divide the points into left and right
    divide_and_conquer_bezier(np.array(all[0 : (len(all)//2) + 1]), depth - 1,
curve_points,intermediates)
    divide_and_conquer_bezier(np.array(all[(len(all) // 2) : ]), depth - 1,
curve_points,intermediates)
    return curve_points, intermediates

```

2.3 File Algoritma Brute Force

File algoritma *Brute Force* adalah `brute_force_bezier.py`. Isi file tersebut adalah sebagai berikut :

1. Impor modul eksternal

```

import numpy as np
from typing import List, Tuple

```

2. Fungsi interpolate

Fungsi menerima `point1`, `point2`, dan `t`, lalu mengembalikan sebuah point antara `point1` dan `point2` sesuai dengan `t`. Penamaan di main file nya masih salah, bukan float tapi tuple of floats seharusnya

```

# interpolation function
def interpolate(point1 : Tuple[float,float], point2 : Tuple[float,float], t :
float) -> np.ndarray :

```

```
return (1 - t) * np.array(point1) + t * np.array(point2)
```

3. Fungsi brute_force_bezier

Fungsi menerima points (list of point), dan depth (jumlah iterasi). Fungsi ini mengembalikan curve_points (list of point) yang merupakan titik - titik yang dibutuhkan untuk membuat kurva Bézier.

```
# function to get the curve points using brute force method
def brute_force_bezier(points : List[Tuple[float,float]], depth : int) ->
List[Tuple[float, float]] :
    curve_points = []

    # the points created is equal to (2 ^ iteration) + 1
    for i in range(((2 ** depth) + 1)) :
        # temp variable to store the control points
        temp_cp = points

        # adjust the t variable
        t = i / ((2 ** depth))

        # iterate until it get 1 point(the curve point)
        while (len(temp_cp) != 1):
            new_cp = []
            for i in range(len(temp_cp) - 1):
                new_cp.append(interpolate(temp_cp[i], temp_cp[i+1], t))
            temp_cp = new_cp

            if (len(temp_cp) == 1):
                curve_points.append(temp_cp[0])

    return curve_points
```

2.4 Kelas Kurva Bézier

File kelas kurva Bézier adalah bezier_curve.py. Kelas tersebut berguna untuk menampilkan kurva Bézier yang dihasilkan menggunakan algoritma *Divide and Conquer* dan algoritma *Brute Force*. Kelas ini juga digunakan untuk menampilkan animasi untuk

pembentukan kurva Bézier yang dibuat menggunakan algoritma *Divide and Conquer*. Berikut isi file tersebut :

1. Impor modul eksternal

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import time
from brute_force_bezier import brute_force_bezier
from dnc_bezier import divide_and_conquer_bezier
from math import ceil as ceil
```

2. Inisialisasi Kelas

Menginisialisasi atribut kelas

```
# initialize the attributes
def __init__(self, control_points, depth, method):
    self.control_points = control_points
    self.depth = depth
    self.method = method
    start_time = time.time()
    # create the curve based on the method chosen
    if method == 1 :
        self.curve_points, self.intermediates = divide_and_conquer_bezier(control_points, depth, [], [])
    else :
        self.curve_points = brute_force_bezier(control_points, depth)
    self.execution_time = (time.time() - start_time)*1000

    # Process the curve points into np array
    self.curve_points_array = np.array(self.curve_points)
    temp_start = self.curve_points_array[0] # avoid removing duplicate ends of control points
    temp_end = self.curve_points_array[-1]
    self.curve_points_array = self.curve_points_array[1:-1] # delete the first and last element
    _, unique_indices = np.unique(self.curve_points_array, axis=0, return_index=True)
    self.curve_points = self.curve_points_array[sorted(unique_indices)]
    # Reinsert the first last element
    if (temp_start is not None and temp_end is not None) :
```

```

self.curve_points =
np.insert(self.curve_points,0,temp_start,axis=0)
self.curve_points = np.append(self.curve_points,[temp_end],axis=0)
self.curve_points_array = np.array(self.curve_points)
self.fig, self.ax = None,None
self.texts = []
self.interval_points = 1

# Adjust for animation speed purpose
if self.depth > 5 :
    self.interval_points = ceil( ( 2**self.depth) + 1 ) / 33 )
    pass

```

3. Prosedur anggota kelas : init_plot

Prosedur untuk membentuk fig yang akan digunakan pada proses animasi

```

# initialize plot that will be used for the animation
def init_plot(self):
    self.fig, self.ax = plt.subplots(figsize=(10, 6))
    self.ax.set_xlabel('X')
    self.ax.set_ylabel('Y')
    self.ax.set_title('Animating Bézier Curve')
    self.ax.plot([point[0] for point in self.control_points], [point[1]
for point in self.control_points], marker='o', markersize=4, linestyle='--',
color='#8594e4', label='Control Points')
    for i, point in enumerate(self.control_points):
        formatted_point = f"C{i}: ({point[0]:.2f}, {point[1]:.2f})"
        plt.text(point[0], point[1], formatted_point, fontsize=9,
verticalalignment='bottom', horizontalalignment='right')

```

4. Prosedur anggota kelas : update_text_annotations

Prosedur untuk mengubah *text* pada graf dan animasi yang ditampilkan

```

# Update the plot text annotations
def update_text_annotations(self, points):
    # Remove old annotations
    for text in self.texts:
        text.remove()
    self.texts.clear()

    # Add new annotations for all points

```



```

        for point in points:
            self.texts.append(self.ax.text(point[0], point[1],
f'({point[0]:.2f}, {point[1]:.2f})', fontsize=6))

```

5. Prosedur anggota kelas : animate

Prosedur untuk menampilkan kurva Bézier *step by step* dengan cara menampilkan animasi proses pembentukan kurva tersebut.

```

# animate the curve (this is only for bezier curve)
def animate(self):
    self.init_plot()

    curve_plot, = self.ax.plot([], [], '#6643b5',
markersize=4,linestyle='-',label = 'Bezier Curve') # Bézier curve
    intermediate_plots, = self.ax.plot([], [], marker='o',
color='#6643b5', linestyle='None', markersize=3) # Intermediate points
    intermediate_points_accumulated = []

    def init():
        intermediate_plots.set_data([], [])
        curve_plot.set_data([], [])
        return intermediate_plots, curve_plot

    def update(frame):
        nonlocal intermediate_points_accumulated
        if frame == total_frames - 1:
            intermediate_points_accumulated = []
            intermediate_plots.set_data([point[0] for point in
self.curve_points if point not in self.control_points], [point[1] for point
in self.curve_points if point not in self.control_points])
            curve_plot.set_data([point[0] for point in self.curve_points],
[point[1] for point in self.curve_points])
            count = 0
            points_last_frame = []
            for points in self.curve_points:
                if points not in self.control_points:
                    count += 1
                    if (count % self.interval_points == 0):
                        points_last_frame.append(points)

            self.update_text_annotations(points_last_frame)
        elif frame < len(self.intermediates):
            points = np.array(self.intermediates[frame]).reshape(-1, 2)
            intermediate_points_accumulated.extend(points)

```

```

        xs = [point[0] for point in intermediate_points_accumulated]
        ys = [point[1] for point in intermediate_points_accumulated]
        intermediate_plots.set_data(xs, ys)
    else:
        curve_frame = frame - len(self.intermediates)
        curve_plot.set_data([point[0] for point in
self.curve_points_array[:curve_frame+1]], [point[1] for point in
self.curve_points_array[:curve_frame+1]])
        return intermediate_plots, curve_plot,

    total_frames = len(self.intermediates) + len(self.curve_points)
    interval = 1500
    if (self.depth > 5 and self.depth < 10):
        interval = 1000
    elif (self.depth >= 10):
        interval = 50
    animation = FuncAnimation(self.fig, update, frames=total_frames,
init_func=init, blit=False, interval=interval, repeat=False)
    plt.text(0.5, 0.01, f'Execution Time: {self.execution_time:.2f} ms',
fontsize=10, transform=plt.gcf().transFigure, horizontalalignment='center')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

```

6. Prosedur anggota kelas : showGraph

Prosedur untuk menampilkan kurva Bézier tanpa prosesnya.

```

# Procedure to show the graph
def showGraph(self):
    self.fig = plt.figure(figsize=(10, 6))
    plt.plot(*zip(*(self.curve_points)), label='Bézier Curve', marker='o',
markersize=4, linestyle='-', color='#6643b5')
    plt.plot(*zip(*(self.control_points)), marker='o', markersize=4,
linestyle='--', color='#8594e4', label='Control Points')
    plt.scatter(*zip(*(self.control_points)), color='#8594e4', zorder=5)
    for i, point in enumerate(self.control_points):
        formatted_point = f"C{i}: ({point[0]:.2f}, {point[1]:.2f})"
        plt.text(point[0], point[1], formatted_point, fontsize=9,
verticalalignment='bottom', horizontalalignment='right')
    count = 0
    for i, point in enumerate(self.curve_points):
        if (point not in self.control_points):
            count += 1

```

```

        if (count % self.interval_points == 0):
            formatted_point = f"({point[0]:.2f}, {point[1]:.2f})"
            plt.text(point[0], point[1], formatted_point, fontsize=6,
verticalalignment='top')

    all_points = np.concatenate([self.curve_points, self.control_points])
    x_coords = all_points[:, 0]
    y_coords = all_points[:, 1]
    x_min, x_max = x_coords.min(), x_coords.max()
    y_min, y_max = y_coords.min(), y_coords.max()

    # Adding some margin
    margin_x = (x_max - x_min) * 0.1
    margin_y = (y_max - y_min) * 0.1

    graph_title = "Bézier Curve via Divide and Conquer"
    if (self.method == 2):
        graph_title = "Bézier Curve via Brute Force"
    plt.xlim(x_min - margin_x, x_max + margin_x)
    plt.ylim(y_min - margin_y, y_max + margin_y)
    plt.text(0.5, 0.01, f'Execution Time: {self.execution_time:.2f} ms',
fontsize=10, transform=plt.gcf().transFigure, horizontalalignment='center')
    plt.legend()
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title(graph_title)
    plt.grid(True)
    plt.tight_layout()
    plt.show()

```

2.5 File Input/Output

File Input/Output adalah IO.py. File tersebut mengatur tampilan pada CLI serta mengatur fitur penyimpanan hasil kurva Bézier. Isi file tersebut adalah sebagai berikut :

1. Impor modul eksternal

```

import numpy as np
from bezier_curve import BezierCurve
import os

```

2. Fungsi method_input()

Fungsi untuk menerima masukan pengguna mengenai tipe jenis algoritma yang ingin digunakan.

```
# Function to accept user input for selecting a preferred algorithm.
# It ensures that the input is a valid integer representing a choice between
two algorithms.
def method_input() -> int:

    print("=====")
    valid = False
    choice = 0
    while (not valid):
        # Prompting the user to choose between two algorithms and handling
        invalid input.
        print("Choose your preferred algorithm !")
        print("1. Divide and Conquer")
        print("2. Brute Force")
        try:
            choice = int(input("Input : "))
            if (choice == 1 or choice == 2):
                valid = True
            else:
                print("Please choose a valid option !")
        except ValueError:
            print("Please input a valid integer !")
        print()
    # Return the user's choice of algorithm.
    return choice
```

3. Fungsi bezier_attribute_input

Fungsi yang menerima orde, titik kontrol, serta iterasi yang diinginkan pengguna. Fungsi ini mengembalikan kelas BezierCurve yang merupakan kelas untuk membuat dan menampilkan kurva Bézier

```
# Accept bezier attribute starting from order, the control points and then the
number of iteration
def bezier_attribute_input(choice : int) -> BezierCurve:
    valid = False

    print("=====")
    =====
```

```

while (not valid):
    try:
        # order input
        order = int(input("Bézier Curve order : "))
        if (order <= 0):
            print("Please input a valid order !")
            print()
            continue

        #control points
        control_points = [[0,0] for i in range(order + 1)]
        for i in range(order + 1):
            control_points[i][0] = float(input(f"X-axis Control point
number {i + 1}: "))
            control_points[i][1] = float(input(f"Y-axis Control point
number {i + 1}: "))
        control_points = np.array(control_points)

        # number of iteration
        depth = int(input("Input number of iterations: "))
        if (depth <0):
            print("Invalid depth input !")
            continue
        print()
        # create the bezier curve
        bezier = BezierCurve(control_points,depth,choice)
        return bezier

    except ValueError:
        print("Input is invalid !")

    except Exception as e:
        print(f"An unexpected error occurred: {e}")

print()

```

4. Prosedur show_output

Prosedur untuk menampilkan hasil kurva Bézier sesuai dengan pilihan pengguna

```

# Procedure to show the visualisation. It is either a graph or an animation
def show_output(result : BezierCurve , choice : int) -> None:
    if (choice == 1):

```

```

print("=====
=====")

valid = False
while (not valid):
    try:
        if (choice == 1):
            print("Please pick your preferred output")
            print("1. Graph, no animation")
            print("2. Graph with animation (bonus)")
            choice_output = int(input("Input: "))

            # Show graph
            if (choice_output == 1):
                result.showGraph()
                valid = True

            #
            elif (choice_output == 2):
                valid = True
                result.animate()

            else:
                print("Please enter the correct choice !")

        elif (choice == 2):
            result.showGraph()
            valid = True
    except ValueError:
        print("Please input the correct type !")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
print()

```

5. Prosedur save_fig

Prosedur untuk menyimpan hasil kurva Bézier dalam folder *test* jika pengguna menginginkan hal tersebut.

```

# Procedure to save the graph figure to the ../test folder.
def save_fig(result : BezierCurve):
    valid = False

```

```

print("=====
=====")
    while (not valid):
        save_option = input("Would you like to save the graph ? y(yes) / other
key (no): ")
        if (save_option == "y"):
            save_path = "../test/"
            file_name = input("file name (with .png): ")
            save_path_file = os.path.join(save_path, file_name)
            if (os.path.exists(save_path_file)):
                print("File already exists !")
            else:
                result.fig.savefig(save_path_file)
                valid = True
        else:
            valid = True
    print()

```

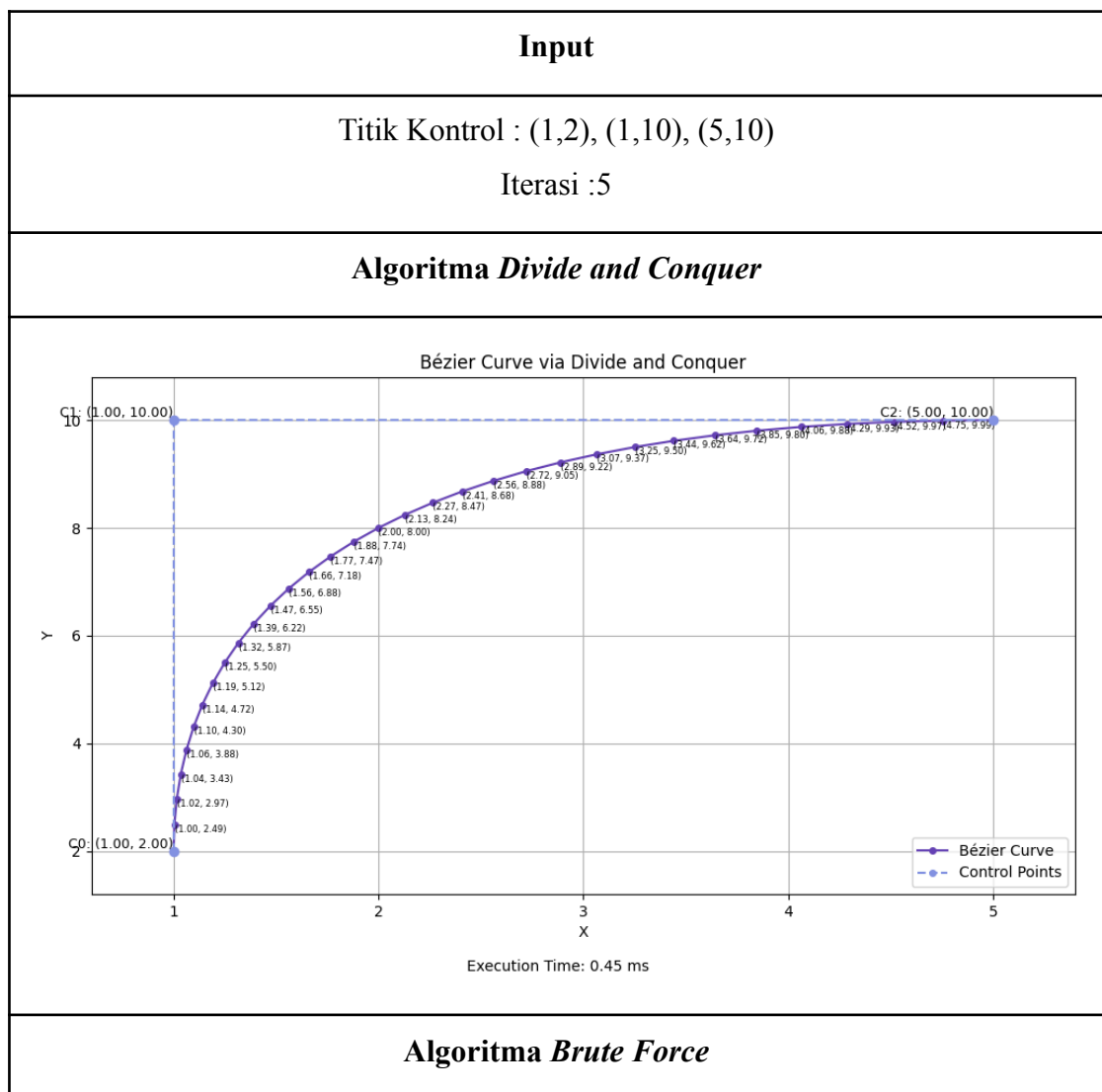
Bab III

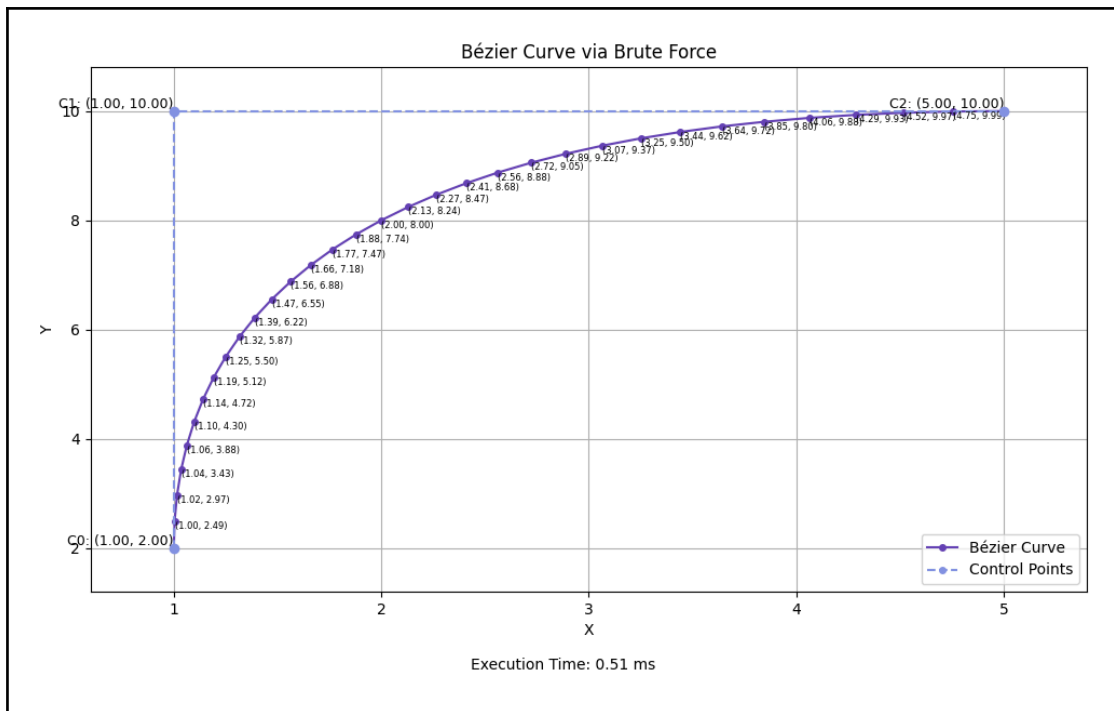
TEST

3.1 Eksperimen dengan 3 Titik Kontrol dan D Iterasi

Berikut adalah eksperimen yang dilakukan menggunakan 3 titik kontrol dan D iterasi. Untuk jumlah operasi, penulis melakukannya secara pribadi dan fitur tersebut tidak dimasukkan kedalam program utama, berikut adalah link file yang digunakan untuk menghitung jumlah operasi : [FileTest](#)

1.





Analisis

Algoritma *Divide and Conquer* sedikit lebih cepat dibandingkan dengan algoritma *Brute Force*. Untuk kasus ini, selisih waktunya sangat tipis sehingga dari sini saja tidak bisa disimpulkan algoritma mana yang lebih baik.

Divide and Conquer Execution Time : 0.45 ms

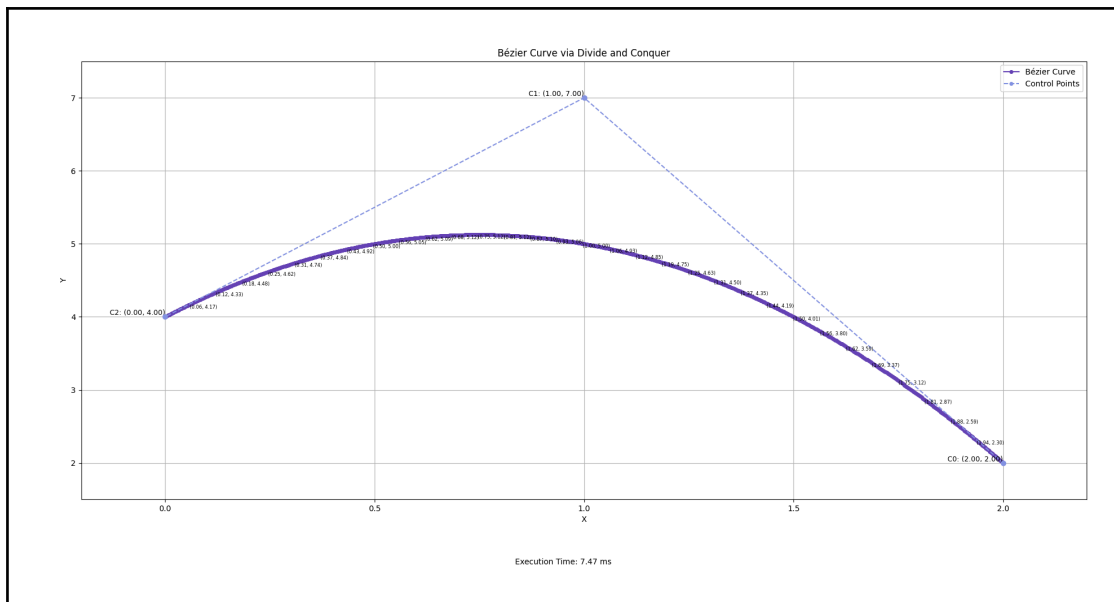
Divide and Conquer Number Of Operations : 652

Brute Force Execution Time : 0.51 ms

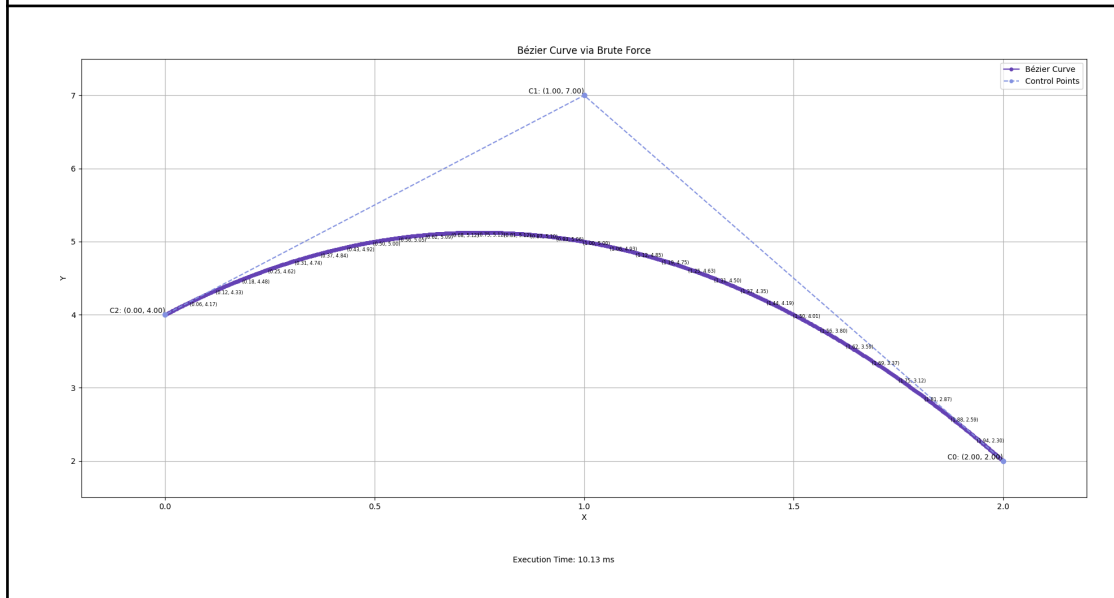
Brute Force Number of Operations : 693

2.

Input
Titik Kontrol : (2, 2), (1, 7), (0, 4) Iterasi : 10
Algoritma <i>Divide and Conquer</i>



Algoritma *Brute Force*



Analisis

Algoritma *Divide and Conquer* sedikit lebih cepat dibandingkan dengan algoritma *Brute Force*. Untuk kasus ini, selisih waktunya sangat tipis sehingga dari sini saja tidak bisa disimpulkan algoritma mana yang lebih baik.

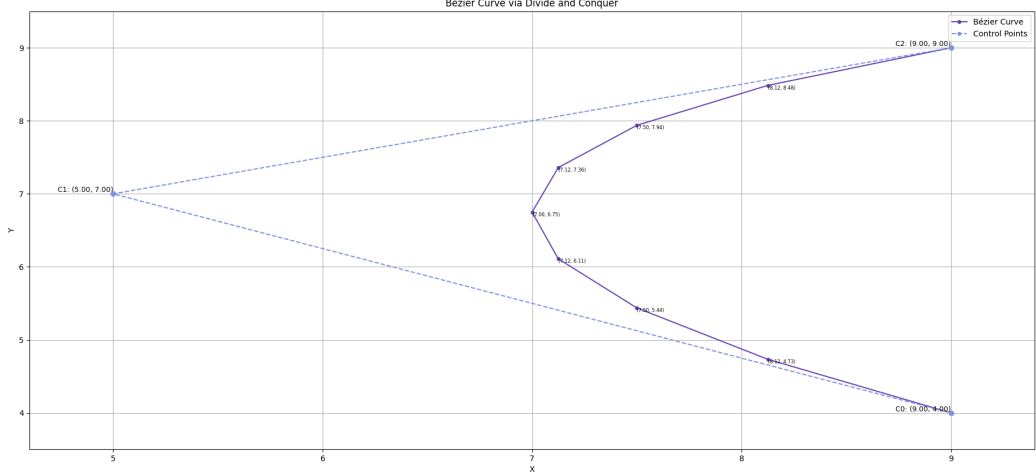
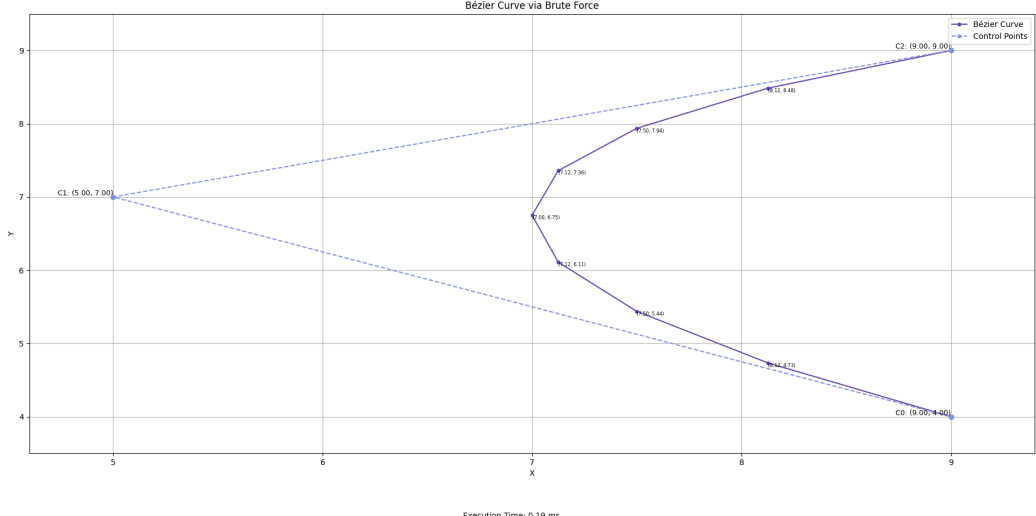
Divide and Conquer Execution Time : 7.47 ms

Divide and Conquer Number Of Operations : 21484

Brute Force Execution Time : 10.13 ms

Brute Force Number of Operations : 21525

3.

<p style="text-align: center;">Input</p>
<p style="text-align: center;">Titik Kontrol : (9, 4), (5, 7), (9, 9)</p> <p style="text-align: center;">Iterasi : 3</p>
<p style="text-align: center;">Algoritma <i>Divide and Conquer</i></p>
 <p style="text-align: center;">Execution Time: 0.14 ms</p>
<p style="text-align: center;">Algoritma <i>Brute Force</i></p>
 <p style="text-align: center;">Execution Time: 0.19 ms</p>
<p style="text-align: center;">Analisis</p>
<p style="text-align: center;">Algoritma <i>Divide and Conquer</i> sedikit lebih cepat dibandingkan dengan algoritma</p>

Brute Force. Untuk kasus ini, selisih waktunya sangat tipis sehingga dari sini saja tidak bisa disimpulkan algoritma mana yang lebih baik.

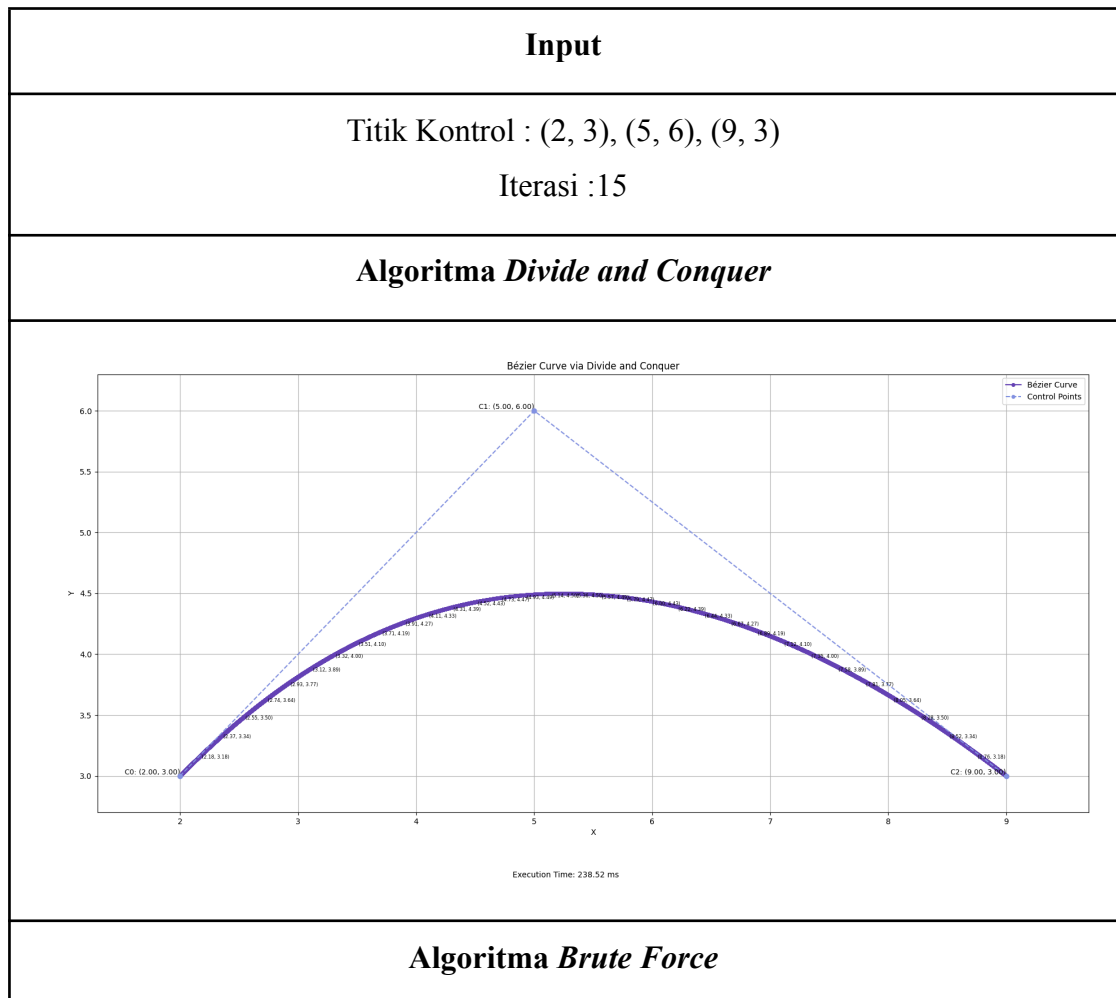
Divide and Conquer Execution Time : 0.14 ms

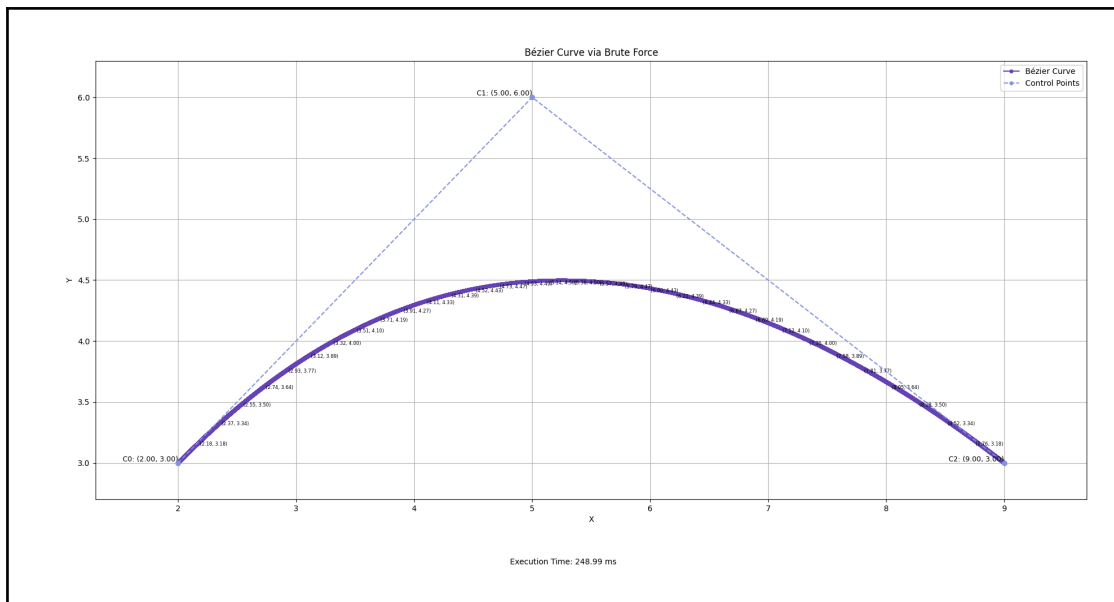
Divide and Conquer Number Of Operations : 148

Brute Force Execution Time : 0.19 ms

Brute Force Number of Operations : 189

4.





Analisis

Algoritma *Divide and Conquer* sedikit lebih cepat dibandingkan dengan algoritma *Brute Force*. Untuk kasus ini, selisih waktunya sangat tipis sehingga dari sini saja tidak bisa disimpulkan algoritma mana yang lebih baik.

Divide and Conquer Execution Time : 238.52 ms

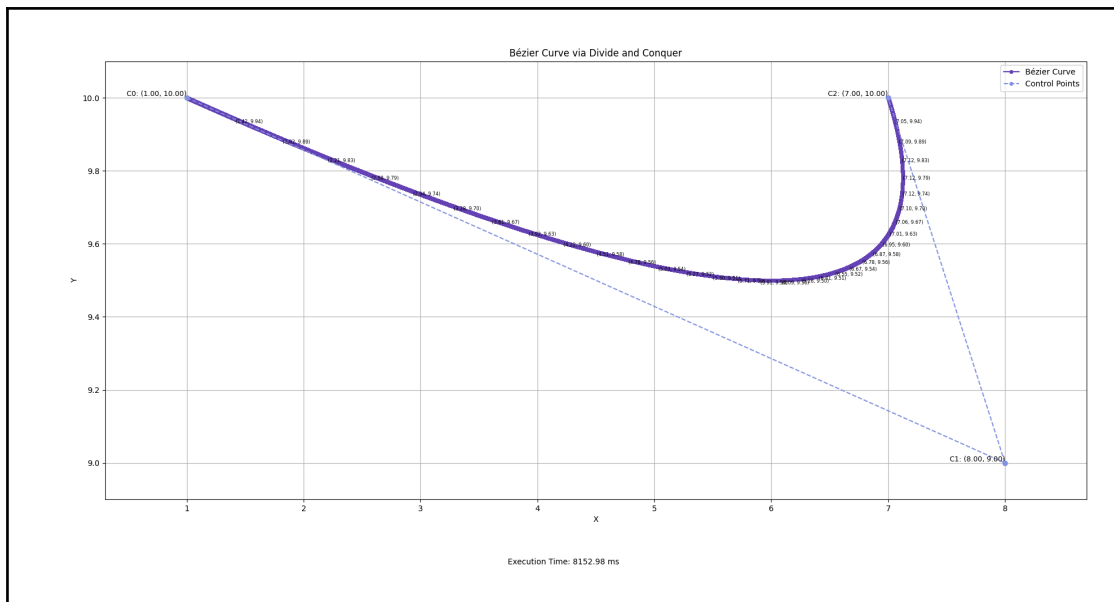
Divide and Conquer Number Of Operations : 688108

Brute Force Execution Time : 248.99 ms

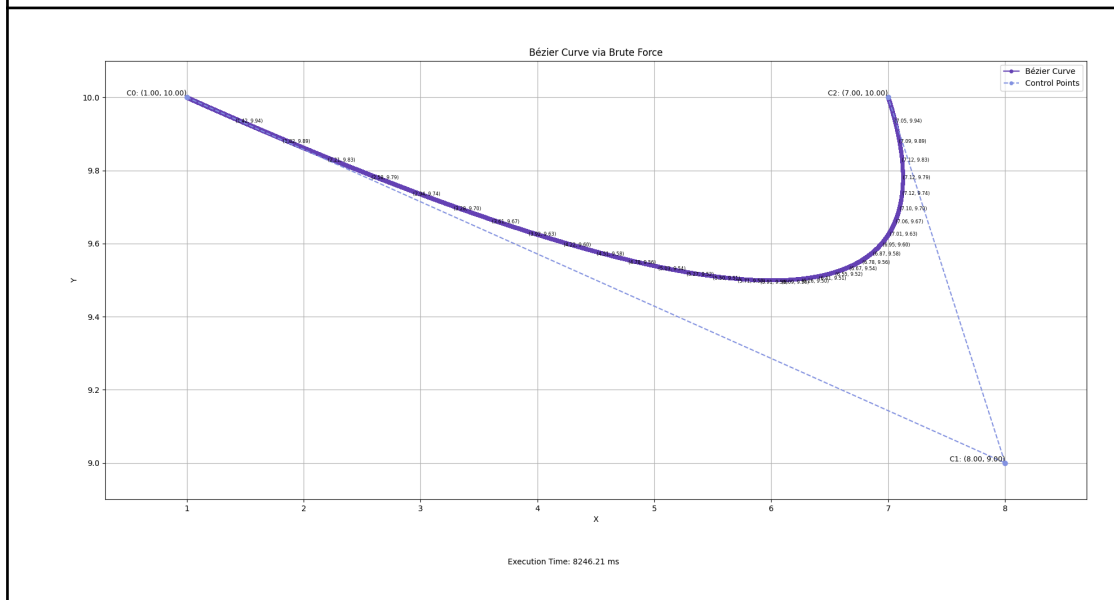
Brute Force Number of Operations : 688149

5.

Input
<p>Titik Kontrol : (1, 10), (8, 9), (7, 10)</p> <p>Iterasi : 20</p>
Algoritma <i>Divide and Conquer</i>



Algoritma *Brute Force*



Analisis

Algoritma *Divide and Conquer* sedikit lebih cepat dibandingkan dengan algoritma *Brute Force*. Untuk kasus ini, selisih waktunya sangat tipis sehingga dari sini saja tidak bisa disimpulkan algoritma mana yang lebih baik.

Divide and Conquer Execution Time : 8152.98 ms

Divide and Conquer Number Of Operations : 22020076

Brute Force Execution Time : 8246.21 ms

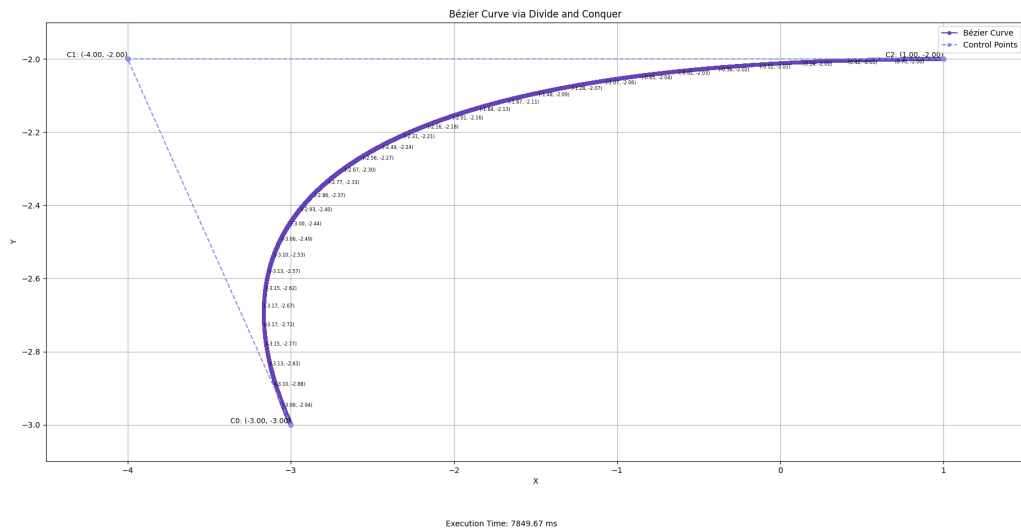
Brute Force Number of Operations : 22020117

Input

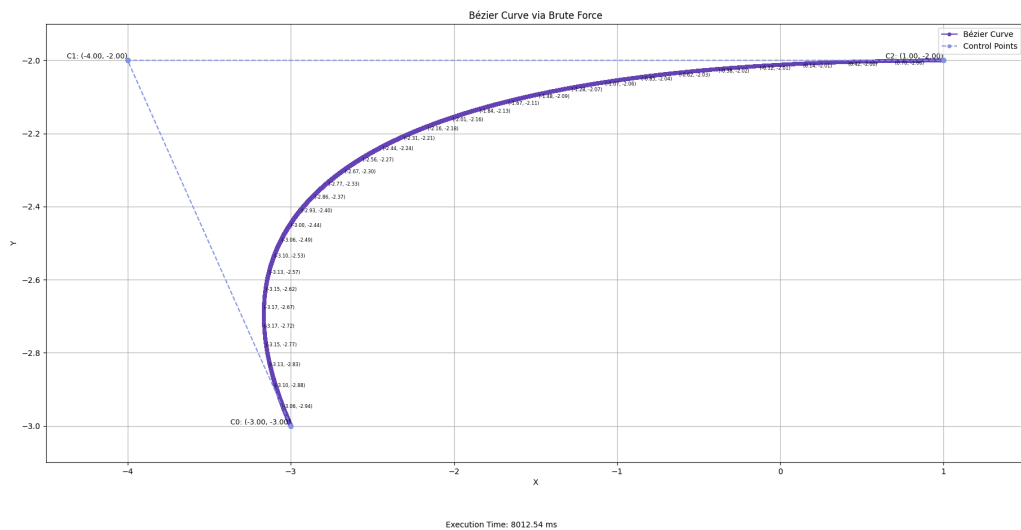
Titik Kontrol : $(-3,-3)$, $(-4,-2)$, $(1,-2)$

Iterasi : 20

Algoritma *Divide and Conquer*



Algoritma *Brute Force*



Analisis

Algoritma *Divide and Conquer* sedikit lebih cepat dibandingkan dengan algoritma *Brute Force*. Untuk kasus ini, selisih waktunya sangat tipis sehingga dari sini saja

tidak bisa disimpulkan algoritma mana yang lebih baik.

Divide and Conquer Execution Time : 7849.67 ms

Divide and Conquer Number Of Operations : 22020076

Brute Force Execution Time : 8012.54 ms

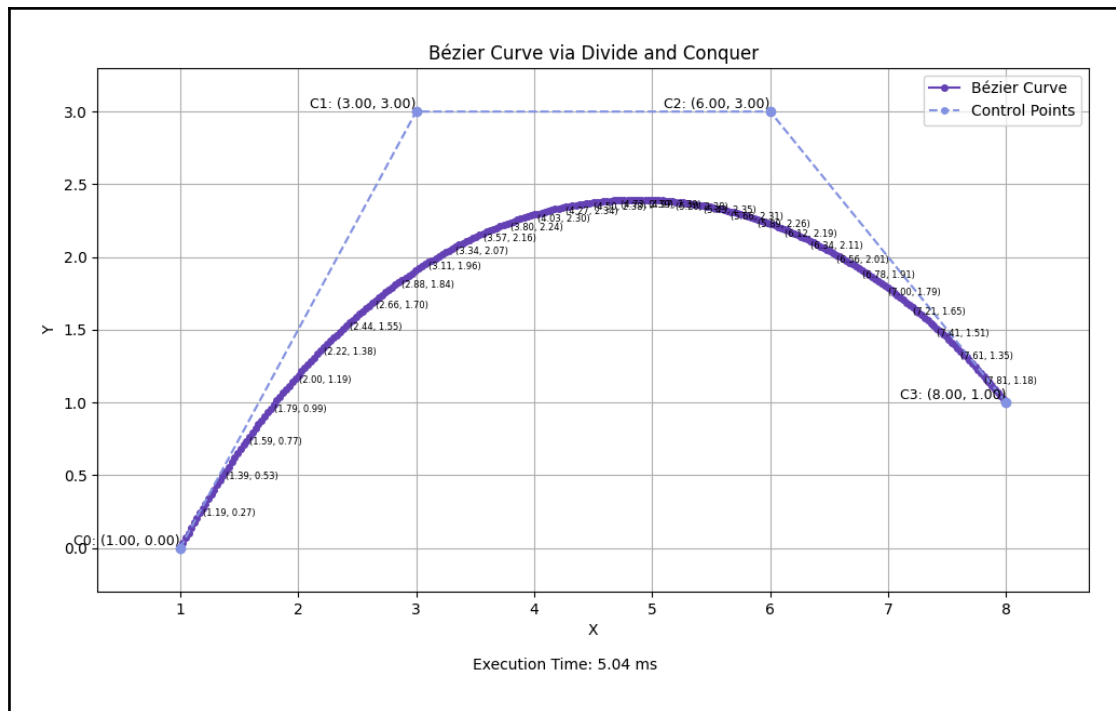
Brute Force Number of Operations : 22020117

3.2 Eksperimen dengan N Titik Kontrol dan D Iterasi

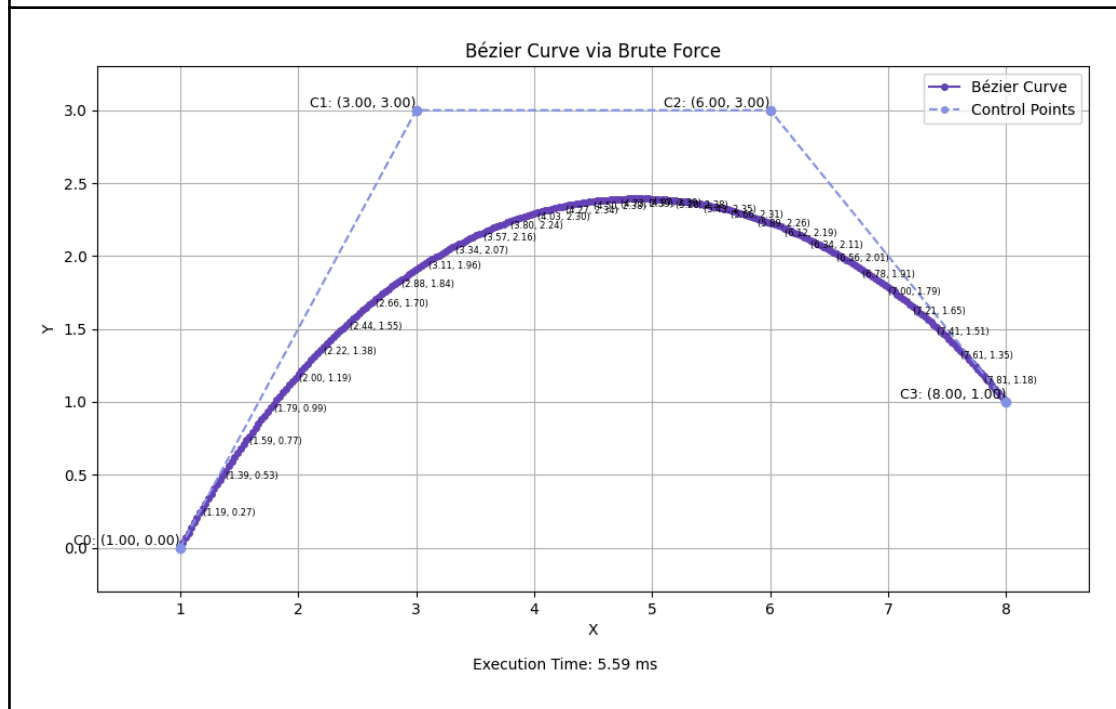
Berikut adalah eksperimen yang dilakukan menggunakan N titik kontrol dan D iterasi. Untuk jumlah operasi, penulis melakukannya secara pribadi dan fitur tersebut tidak dimasukkan kedalam program utama, berikut adalah link file yang digunakan untuk menghitung jumlah operasi : [FileTest](#)

1.

Input
Order : 3 Titik Kontrol : (1,0), (3,3), (6,3), (8,1) Iterasi : 8
<i>Algoritma Divide and Conquer</i>



Algoritma *Brute Force*



Analisis

Algoritma *Divide and Conquer* sedikit lebih cepat dibandingkan dengan algoritma *Brute Force*. Selisih waktu yang kecil ini dapat dikarenakan *environment* atau implementasi kode yang sedikit tidak efisien di salah satu algoritma.

Divide and Conquer Execution Time : 5.04 ms

Divide and Conquer Number Of Operations : 8161

Brute Force Execution Time : 5.59 ms

Brute Force Number of Operations : 10023

2.

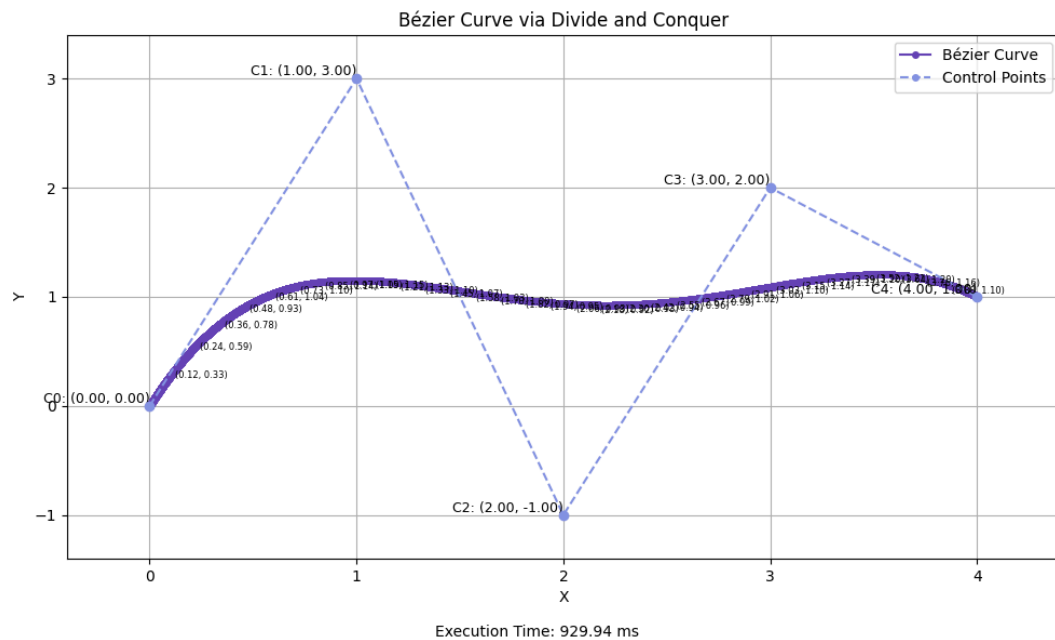
Input

Order : 4

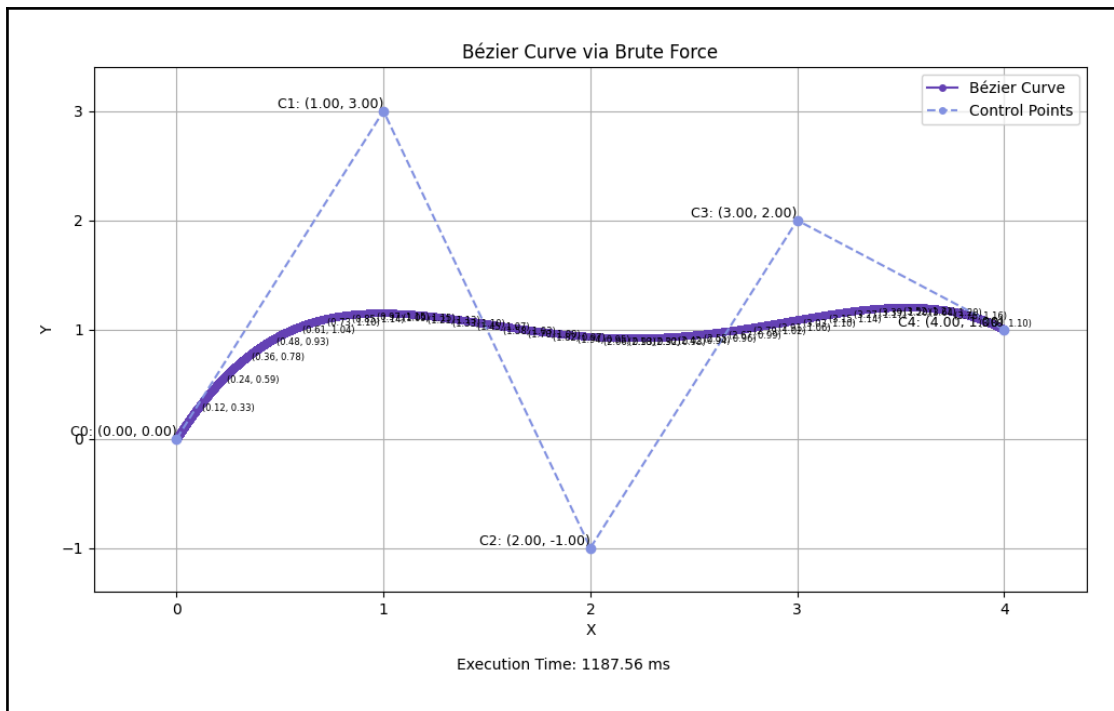
Titik Kontrol : (0,0), (1,3), (2, -1), (3,2), (4,1)

Iterasi : 15

Algoritma *Divide and Conquer*



Algoritma *Brute Force*



Analisis

Algoritma *Divide and Conquer* sedikit lebih cepat dibandingkan dengan algoritma *Brute Force*. Selisih waktu yang kecil ini dapat dikarenakan *environment* atau implementasi kode yang sedikit tidak efisien di salah satu algoritma.

Divide and Conquer Execution Time : 929.94 ms

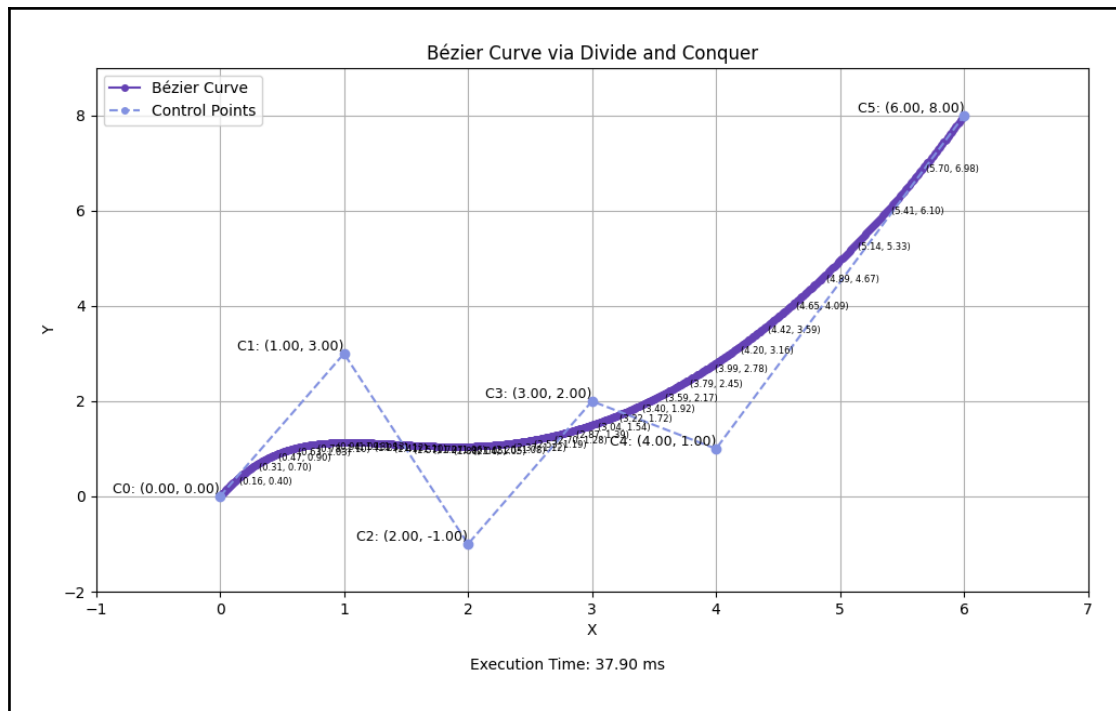
Divide and Conquer Number Of Operations : 1507283

Brute Force Execution Time : 1187.56 ms

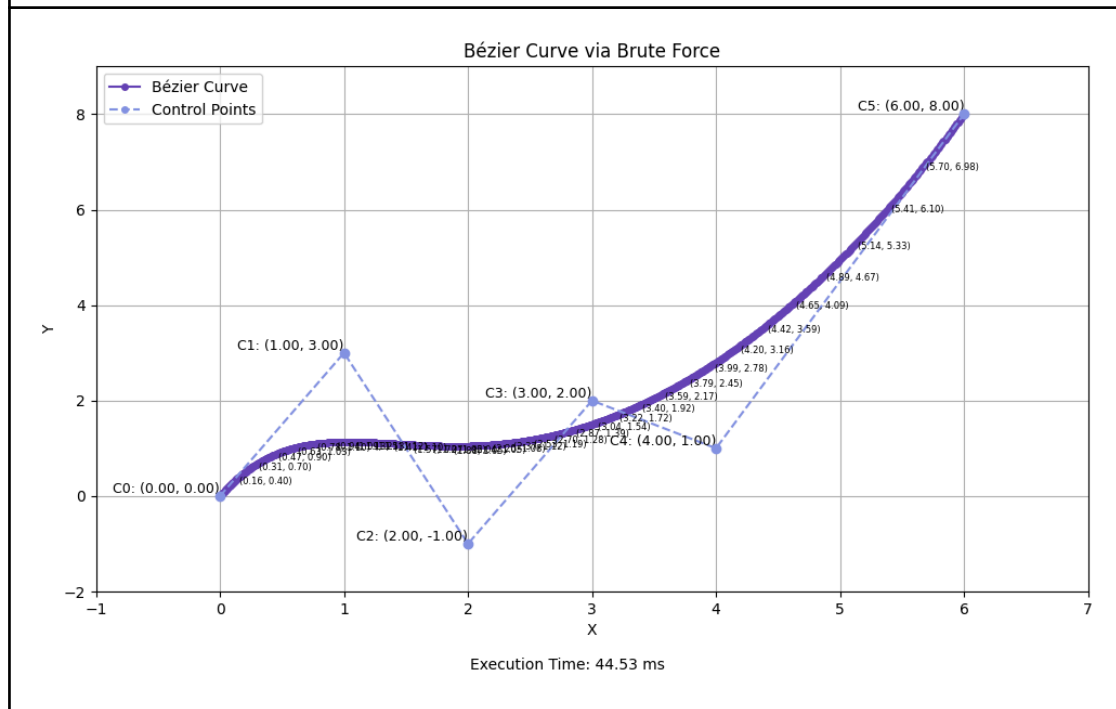
Brute Force Number of Operations : 2064447

3.

Input
Order : 5
Titik Kontrol : (0,0), (1,3), (2, -1), (3,2), (4,1), (6,8)
Iterasi : 10
Algoritma <i>Divide and Conquer</i>



Algoritma *Brute Force*



Analisis

Algoritma *Divide and Conquer* sedikit lebih cepat dibandingkan dengan algoritma *Brute Force*. Untuk kasus ini, selisih waktunya sangat tipis sehingga dari sini saja tidak bisa disimpulkan algoritma mana yang lebih baik.

Divide and Conquer Execution Time : 37.90 ms

Divide and Conquer Number Of Operations : 64450

Brute Force Execution Time : 44.53 ms

Brute Force Number of Operations : 95325

4.

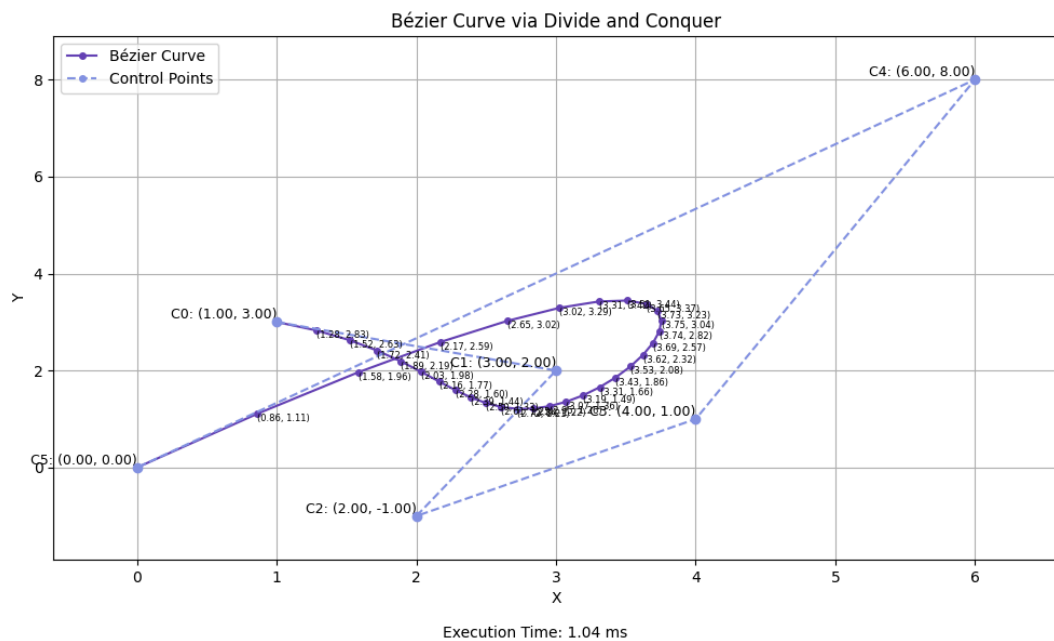
Input

Order : 5

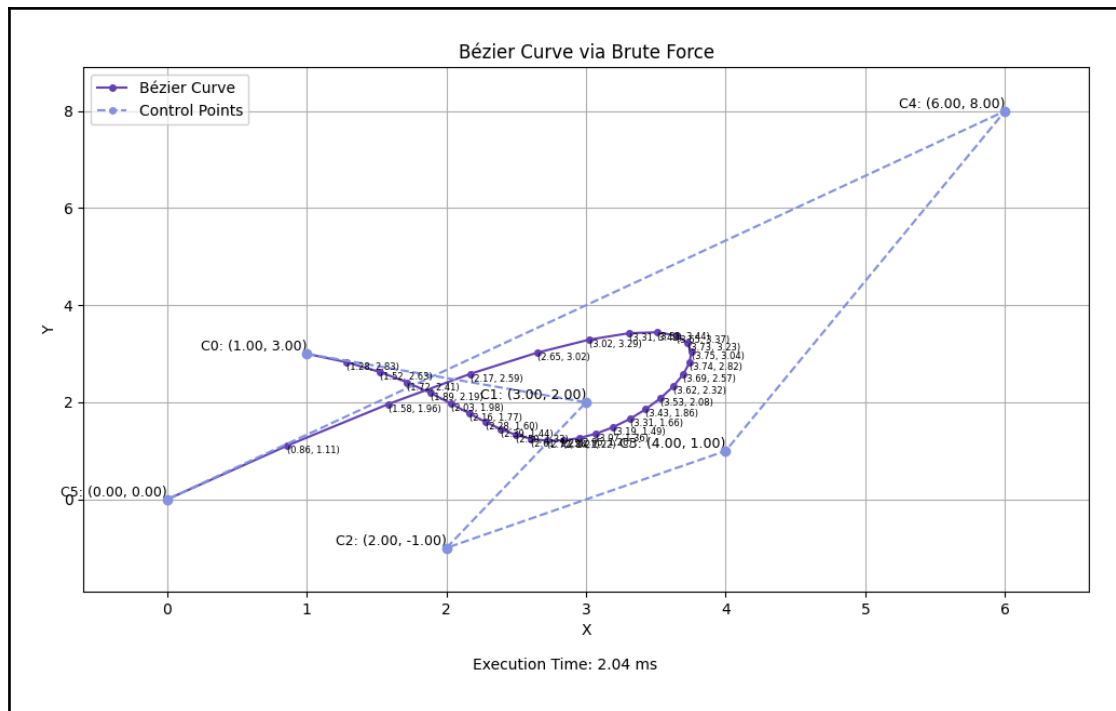
Titik Kontrol : (1,3), (3,2), (2,-1), (4,1), (6,8), (0,0)

Iterasi : 5

Algoritma *Divide and Conquer*



Algoritma *Brute Force*



Analisis

Algoritma *Divide and Conquer* sedikit lebih cepat dibandingkan dengan algoritma *Brute Force*. Untuk kasus ini, selisih waktunya sangat tipis sehingga dari sini saja tidak bisa disimpulkan algoritma mana yang lebih baik.

Divide and Conquer Execution Time : 1.04 ms

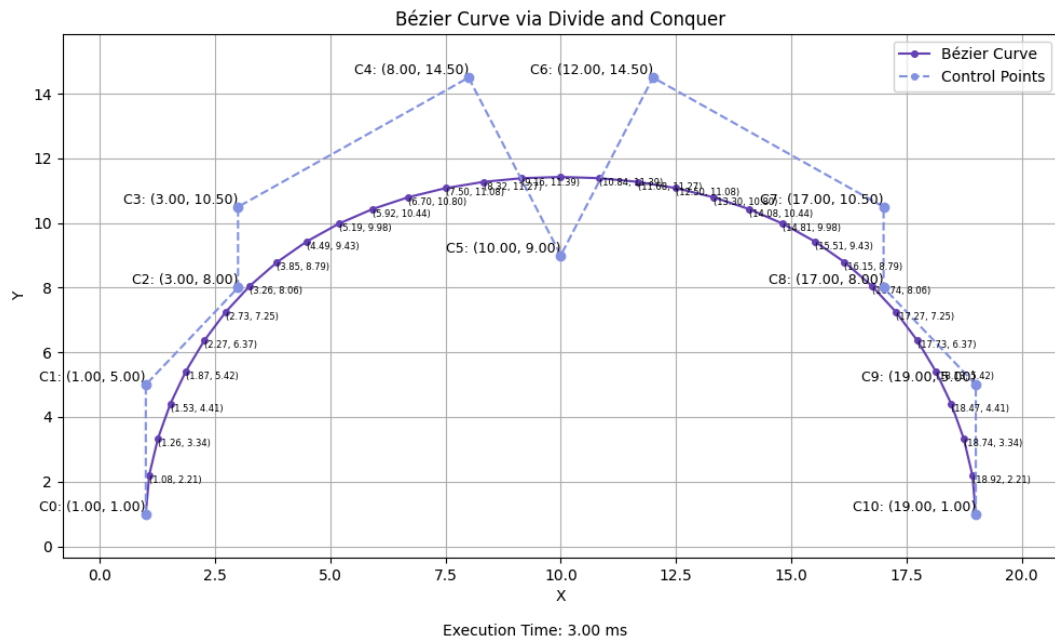
Divide and Conquer Number Of Operations : 1954

Brute Force Execution Time : 2.04 ms

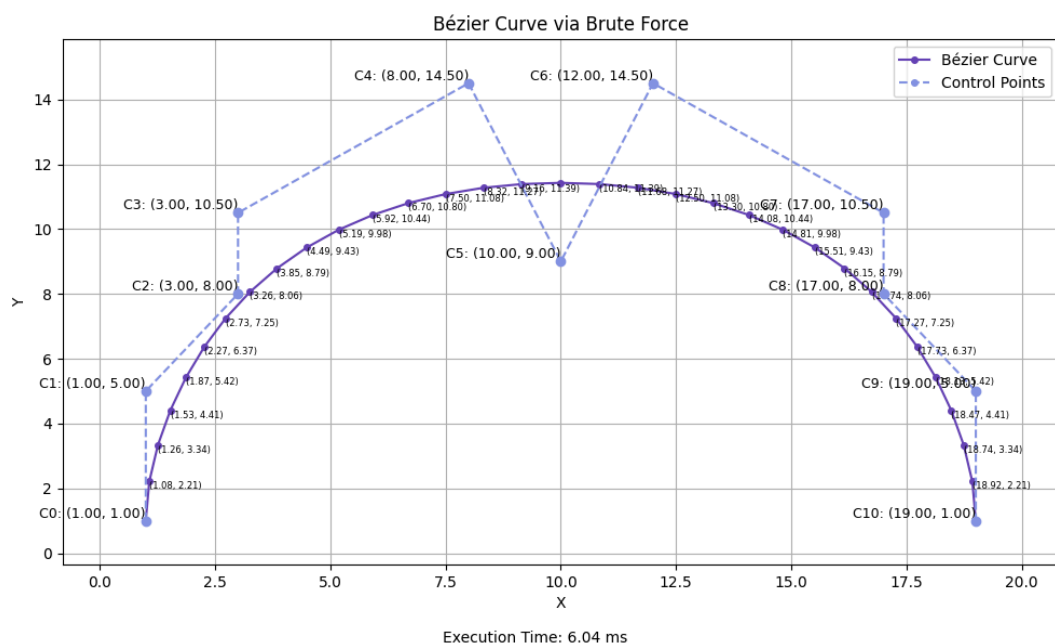
Brute Force Number of Operations : 3069

5.

Input
<p>Order : 10</p> <p>Titik Kontrol : (1,1), (1,5), (3,8), (3,10.5), (8, 14.5), (10, 9), (12, 14.5), (17, 10.5), (17, 8), (19, 5), (19, 1)</p> <p>Iterasi : 5</p>
Algoritma <i>Divide and Conquer</i>



Algoritma *Brute Force*



Analisis

Algoritma *Divide and Conquer* sedikit lebih cepat dibandingkan dengan algoritma *Brute Force*. Untuk kasus ini, selisih waktunya sangat tipis sehingga dari sini saja tidak bisa disimpulkan algoritma mana yang lebih baik.

Divide and Conquer Execution Time : 3.00 ms

Divide and Conquer Number Of Operations : 5984

Brute Force Execution Time : 6.04 ms

Brute Force Number of Operations : 10989

6.

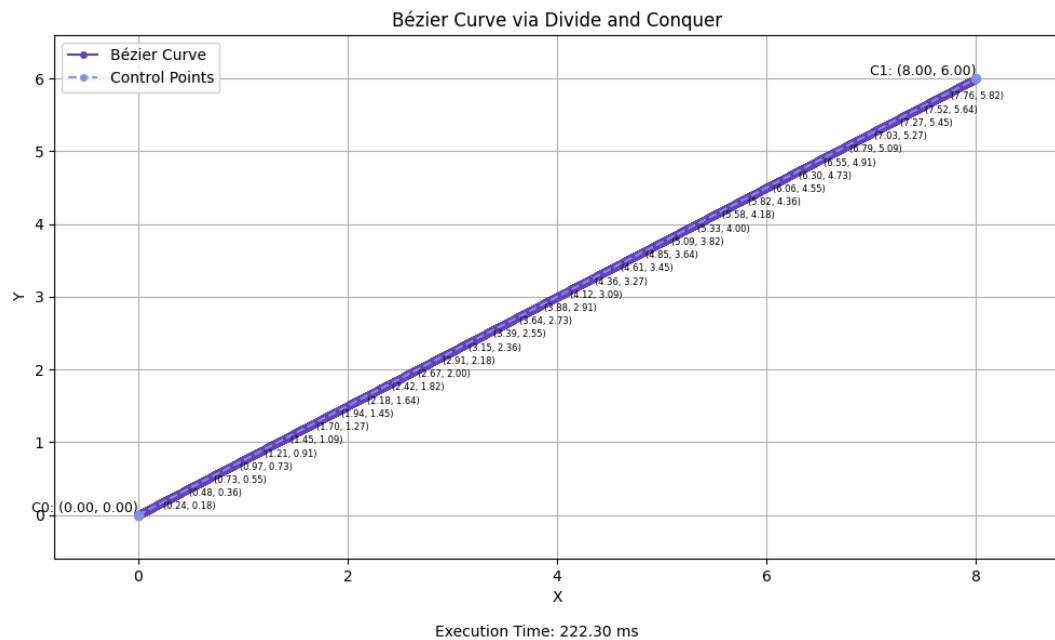
Input

Order : 1

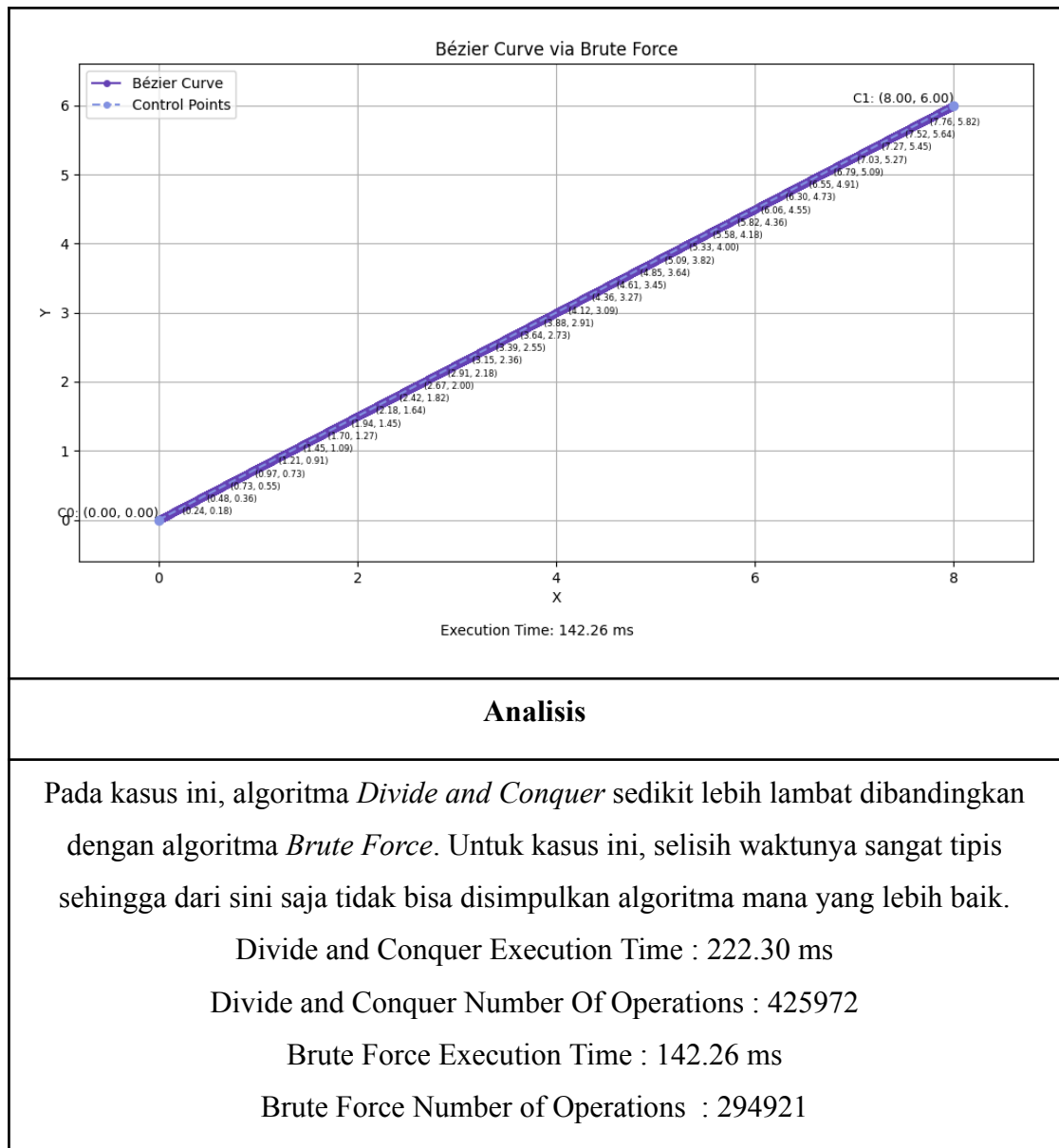
Titik Kontrol : (0,0), (8,6)

Iterasi : 15

Algoritma *Divide and Conquer*



Algoritma *Brute Force*



Bab IV

Kesimpulan dan Saran

4.1 Kesimpulan

Algoritma *divide and conquer* merupakan salah satu pendekatan dalam pembentukan kurva bezier jika diketahui titik-titik kontrol awal dan jumlah iterasi. Dengan algoritma ini, persoalan dipecah menjadi upa persoalan, yang dalam hal ini dibagi menjadi *left* dan *right* yang dalam tiap persoalan dicari titik tengahnya. Dengan algoritma ini, persoalan kurva bezier dapat diselesaikan dengan kompleksitas $O(2^k n^2)$.

Sebagai pembandingan, penulis juga mengimplementasikan algoritma dengan pendekatan *brute force*. Pendekatan ini adalah pendekatan yang lebih lempang, yaitu dengan cara mencari hasil interpolasi pada 2 buah titik kontrol yang terletak bersampingan pada array. Dengan algoritma ini, kompleksitas big-O nya juga $O(2^k n^2)$.

Dari analisis kompleksitas tersebut, dapat dilihat bahwa penggunaan *divide and conquer* tidak memberikan keuntungan kompleksitas yang signifikan. Namun, pada saat pengetesan untuk skala kecil, dapat dilihat bahwa pada kebanyakan kasus, algoritma *divide and conquer* lebih sering lebih cepat dieksekusi oleh program. Hal ini karena terdapat perbedaan pada $T(n, k)$ kedua pendekatan.

4.2 Saran

Tugas Kecil IF2211 Strategi Algoritma Semester II Tahun 2023/2024 memberikan pengalaman baru yang berharga bagi penulis. Berdasarkan pengalaman tersebut, berikut beberapa saran yang dapat diberikan kepada pembaca yang ingin menyelesaikan atau mengikuti tugas serupa:

1. Kerja sama tim yang efektif adalah kunci penting dalam menyelesaikan tugas ini. Penggunaan version control, seperti git, sangat disarankan.
2. Mengingat tugas ini melibatkan perhitungan matematis, diperlukan pemahaman dan ketelitian ekstra dalam pengembangan algoritma. Penulis juga dituntut memahami dan dapat menerapkan pendekatan *divide and conquer* dan *brute force* dalam menyelesaikan permasalahan

Bab V

LAMPIRAN

5.1 *Link Repository*

https://github.com/MarvelPangondian/Tucil2_13522075_13522118

5.2 *Tabel Checklist*

Status : *Completed*

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✓	
2. Program dapat melakukan visualisasi kurva Bézier.	✓	
3. Solusi yang diberikan program optimal.	✓	
4. Program dapat menghasilkan masukan secara acak.	✓	
5. Program dapat membuat kurva untuk n titik kontrol (Bonus) .	✓	
6. Program dapat melakukan visualisasi proses pembuatan kurva (Bonus) .	✓	