

LAPORAN TUGAS KECIL 2

Semester II tahun 2023/2024

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis
Divide and Conquer



NIM 13522112

Nama : Dimas Bagoes Hendrianto

Kelas : K02

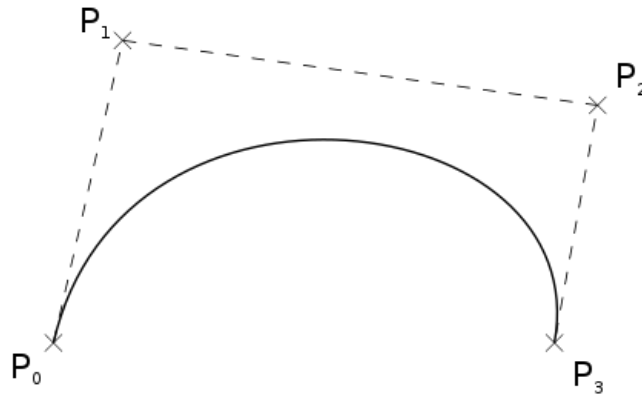
**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024**

DAFTAR ISI

DAFTAR ISI.....	1
BAB I Deskripsi Masalah	2
BAB II Teori Singkat.....	5
BAB III Implementasi.....	6
BAB IV Eksperimen	8
Lampiran	10

BAB I

Dekripsi Masalah



Gambar 1.1. Kurva Bézier Kubik

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

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

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

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

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

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

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

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

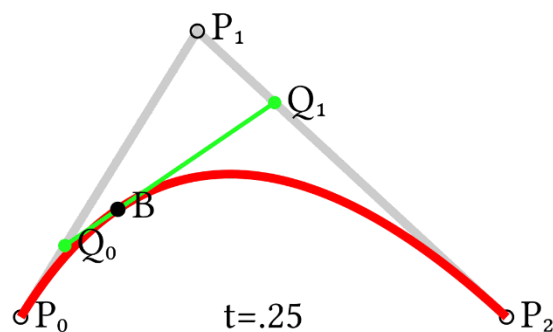
$$Q_1 = B(t) = (1 - t)P_1 + tP_2, \quad t \in [0, 1]$$

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

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

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

Berikut adalah ilustrasi dari kasus diatas.



Gambar 1.2. Pembentukan Kurva Bézier Kuadrat.

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

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

IF2211 Strategi Algoritma
Tugas Kecil 2

$$S_0 = B(t) = (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + t^3P_3, \quad t \in [0, 1]$$

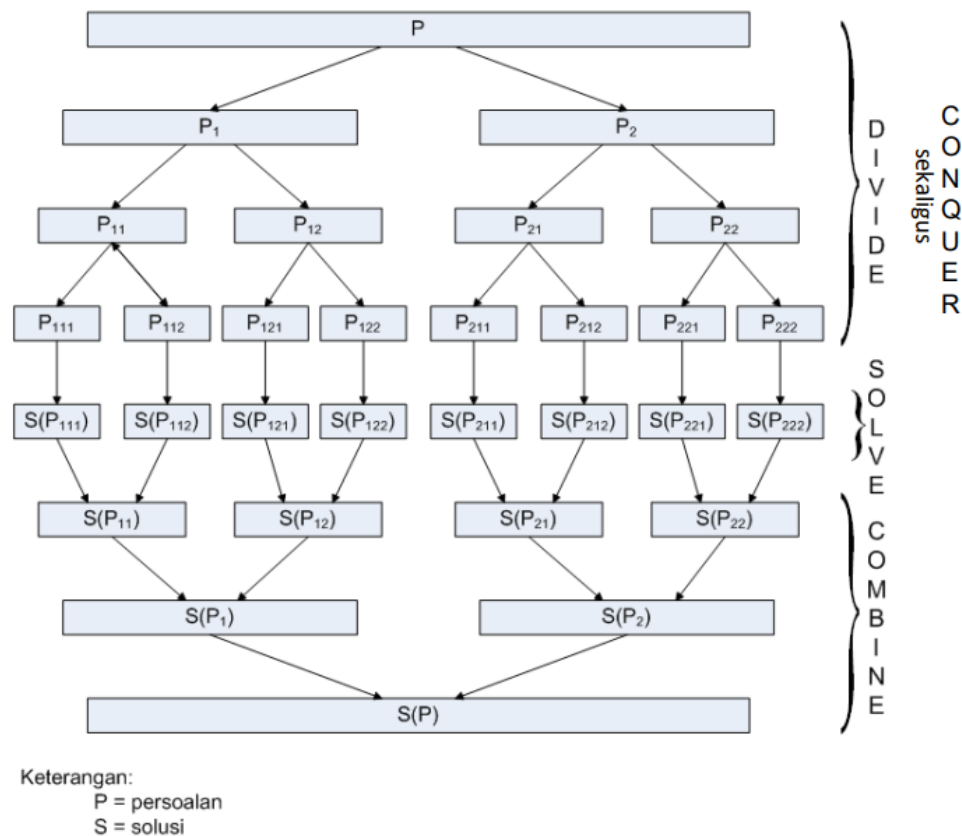
$$T_0 = B(t) = (1-t)^4P_0 + 4(1-t)^3tP_1 + 6(1-t)^2t^2P_2 + 4(1-t)t^3P_3 + t^4P_4, \quad t \in [0, 1]$$

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

BAB II

Teori Singkat

Divide didefinisikan menjadi membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama). Conquer didefinisikan menjadi menyelesaikan (solve) masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar). Combine didefinisikan menjadi mengabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.



Gambar 2.1. Diagram Proses Divide and Conquer

(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-%282021%29-Bagian1.pdf>)

Obyek persoalan yang dibagi : masukan (input) atau instances persoalan yang berukuran seperti tabel (larik), matriks, eksponen, polinom, dll, bergantung persoalannya. Tiap-tiap upa-persoalan memiliki karakteristik yang sama (*the same type*) dengan karakteristik persoalan semula namun berukuran lebih kecil sehingga metode Divide and Conquer lebih natural diungkapkan

BAB III

Implementasi

Algoritma Brute Force yang saya gunakan (spesifikasi wajib dan bonus):

```
def generateMid(x1, x2, x3, y1, y2, y3):
    xMidL, yMidL = midPoint(x1, x2, y1, y2)
    xMidR, yMidR = midPoint(x3, x2, y3, y2)
    xMidM, yMidM = midPoint(xMidL, xMidR, yMidL, yMidR)
    return xMidL, xMidM, xMidR, yMidL, yMidM, yMidR

def generateBezierBF(arrX, arrY):
    xRes = []
    yRes = []
    xRes.append(arrX[0])
    yRes.append(arrY[0])
    for i in range(len(arrX)-2):
        xMidL, xMidM, xMidR, yMidL, yMidM, yMidR = generateMid(arrX[i], arrX[i+1], arrX[i+2], arrY[i], arrY[i+1], arrY[i+2])
        if i == 0:
            xRes.append(xMidL)
            yRes.append(yMidL)
        xRes.append(xMidM)
        yRes.append(yMidM)
        xRes.append(xMidR)
        yRes.append(yMidR)
    xRes.append(arrX[len(arrX)-1])
    yRes.append(arrY[len(arrY)-1])
    return xRes, yRes
```

1. Membuat fungsi generateMid terlebih dahulu sebagai default “titik tengah” atau “titik antara” yang merupakan titik tengah di antara dua buah titik kontrol.
2. Pada fungsi generateBezierBF akan dilakukan pembuatan titik-titik antara yang titik kontrolnya disimpan dalam dua buah array yang pertama untuk absis dan yang kedua untuk ordinat.
3. Akan ditambahkan terlebih dahulu titik kontrol pertama basis sebelum dilakukan pengulangan untuk titik kontrol lain
4. Untuk setiap 3 titik kontrol yang berurutan di dalam array akan dihasilkan 3 titik antara yaitu antara tengah kiri, antara tengah tengah, dan antara tengah kanan. Antara tengah tengah dan antara tengah kanan yang kemudian akan ditambahkan ke array hasil karena tidak ingin adanya pengulangan titik-titik yang sama pada array hasil nanti. Pengecualian untuk pengulangan yang pertama akan ditambahkan antara tengah kiri akan ditambahkan ke dalam array hasil
5. Pada pengulangan yang terakhir akan ditambahkan titik kotrol terakhir karena tidak ingin kehilangan titik kontrol ini.

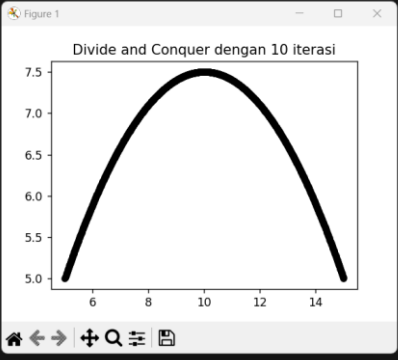
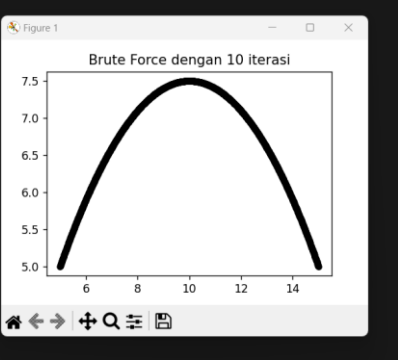
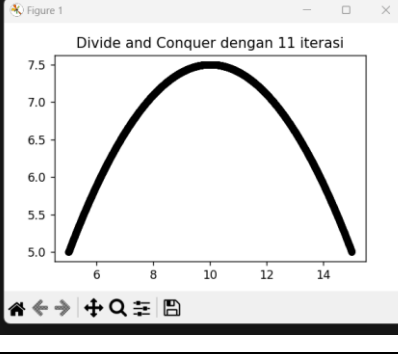
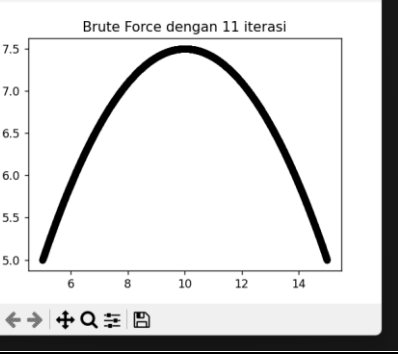
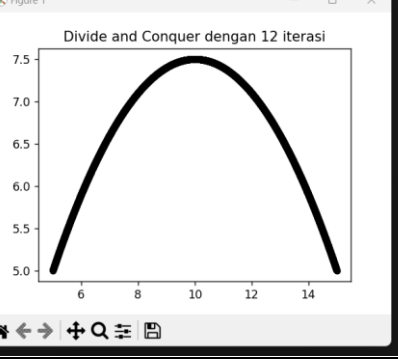
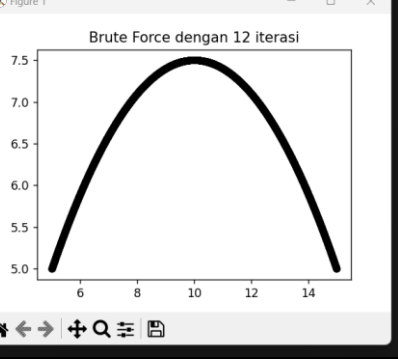
Algoritma Divide and Conquer yang saya gunakan (spesifikasi wajib) :

```
def midPoint(x1, x2, y1, y2):  
    xMid = (x1 + x2)/2  
    yMid = (y1 + y2)/2  
    return xMid, yMid  
  
def generateBezier(x1, x2, x3, y1, y2, y3, currIteration):  
    if (currIteration < iteration):  
        xMidLeft, yMidLeft = midPoint(x1, x2, y1, y2)  
        xMidRight, yMidRight = midPoint(x2, x3, y2, y3)  
        xMidMid, yMidMid = midPoint(xMidLeft, xMidRight, yMidLeft, yMidRight)  
        currIteration = currIteration + 1  
        generateBezier(x1, xMidLeft, xMidMid, y1, yMidLeft, yMidMid, currIteration)  
        x.append(xMidMid)  
        y.append(yMidMid)  
        generateBezier(xMidMid, xMidRight, x3, yMidMid, yMidRight, y3, currIteration)
```

1. Membuat fungsi midPoint terlebih dahulu sebagai default “titik tengah” atau “titik antara” yang merupakan titik tengah di antara dua buah titik kontrol
2. Pada fungsi generateBezier akan dilakukan pembuatan titik-titik antara yang titik kontrolnya ditambahkan ke dua buah array yang pertama untuk absis dan yang kedua untuk oordinat.
3. Dengan adanya iterasi yang menjadi salah satu parameter pada fungsi berguna untuk mengetahui kapan pengulangan akan berhenti.
4. Pada dasarnya akan dihasilkan 3 titik antara yaitu antara tengah kiri, antara tengah tengah, dan antara tengah kanan dari 3 titik kontrol input. Tidak lupa untuk menambahkan nilai pada parameter iterasi sehingga perulangan yang terjadi akan terhingga dan mendapat hasil yang sesuai. Kemudian akan dilakukan DIVIDE yang pertama adalah titik kontrol 1, antara tengah kiri, serta antara tengah tengah, DIVIDE yang kedua adalah antara tengah tengah, antara tengah kanan, serta titik kontrol 3. Titik kontrol 2 tidak lagi digunakan karena telah “tergantikan” oleh titik-titik antara tengah yang menjadi “titik kontrol” baru untuk pengulangan selanjutnya.
5. Pada proses COMBINE akan ditambahkan titik antara tengah tengah ke dalam array yang telah ada.
6. Hal ini akan berulang sesuai parameter iterasi yang dimasukkan oleh pengguna.

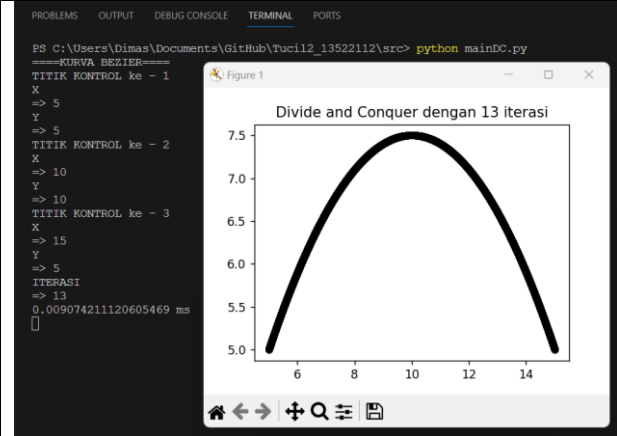
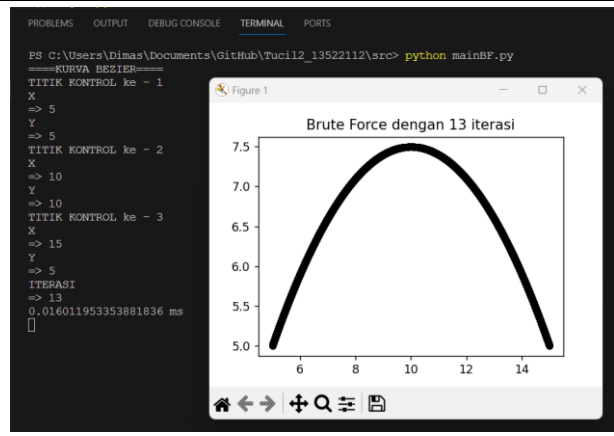
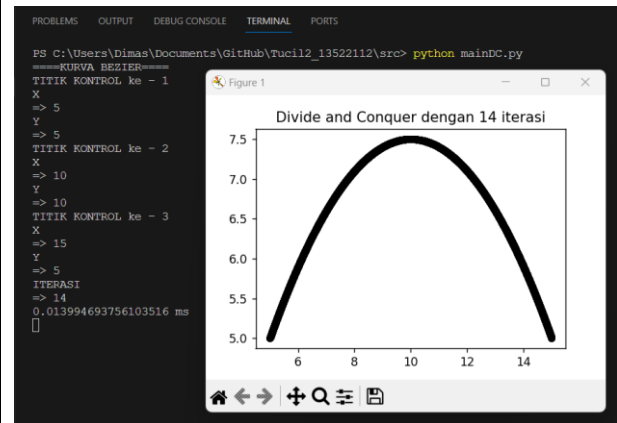
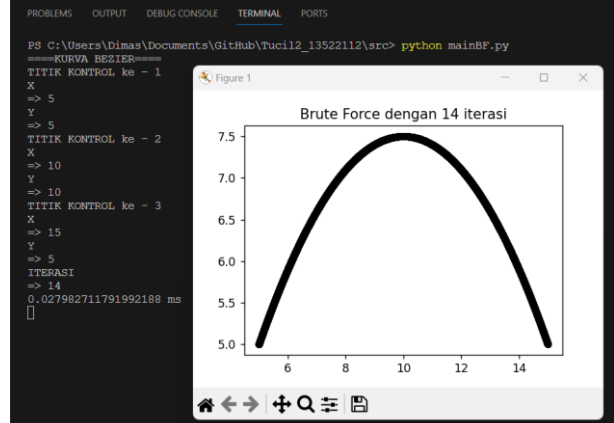
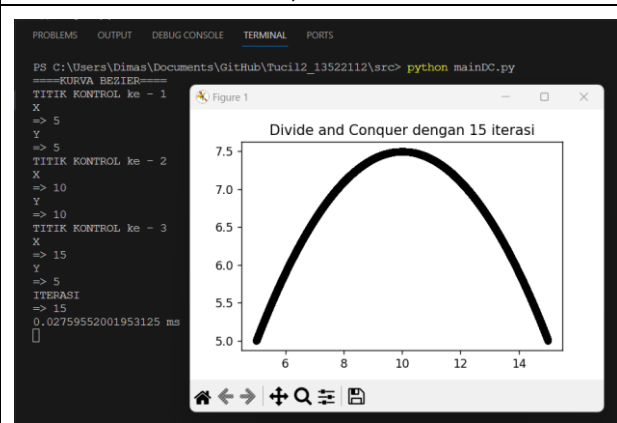
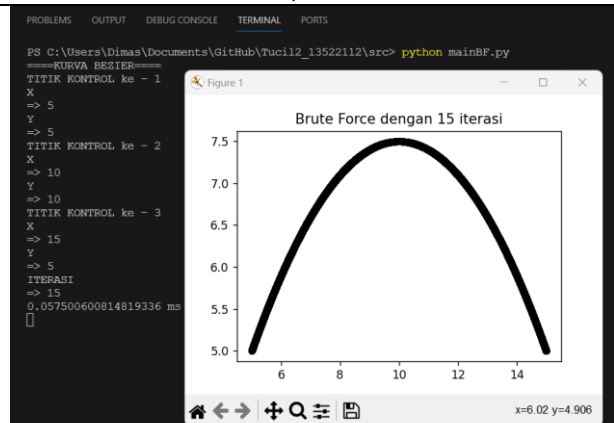
BAB IV

Eksperimen

Divide And Conquer	Bruteforce
<pre>PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainDC.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 10 0.0009045600891113281 ms []</pre> 	<pre>PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainBF.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 10 0.0019989013671875 ms []</pre> 
$T \approx 0,0009$ ms	$T \approx 0,002$ ms
<pre>PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainDC.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 11 0.0009908676147460938 ms []</pre> 	<pre>PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainBF.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 11 0.0040073394775390625 ms []</pre> 
$T \approx 0,001$ ms	$T \approx 0,004$ ms
<pre>PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainDC.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 12 0.002008676528930664 ms []</pre> 	<pre>PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainBF.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 12 0.007001638412475586 ms []</pre> 
$T \approx 0,002$ ms	$T \approx 0,0007$ ms

IF2211 Strategi Algoritma

Tugas Kecil 2

 <pre> PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainDC.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 13 0.009074211120605469 ms </pre>	 <pre> PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainBF.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 13 0.016011953353881836 ms </pre>
<p>$T \approx 0,009 \text{ ms}$</p>	<p>$T \approx 0,016 \text{ ms}$</p>
 <pre> PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainDC.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 14 0.013994693756103516 ms </pre>	 <pre> PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainBF.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 14 0.027982711791992188 ms </pre>
<p>$T \approx 0,014 \text{ ms}$</p>	<p>$T \approx 0,028 \text{ ms}$</p>
 <pre> PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainDC.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 15 0.02759552001953125 ms </pre>	 <pre> PS C:\Users\Dimas\Documents\GitHub\Tucil2_13522112\src> python mainBF.py ====KURVA BEZIER==== TITIK KONTROL ke - 1 X => 5 Y => 5 TITIK KONTROL ke - 2 X => 10 Y => 10 TITIK KONTROL ke - 3 X => 15 Y => 5 ITERASI => 15 0.057500600814819336 ms </pre>
<p>$T \approx 0,028 \text{ ms}$</p>	<p>$T \approx 0,056 \text{ ms}$</p>

Kesimpulan

Dari pengujian yang telah dilakukan bisa dilihat kalau waktu yang diperlukan untuk memperoleh kurva Bezier dengan algoritma Devide and Conquer dengan angka yang cukup konsisten yaitu 2 kai lebih cepat dua kali lipat apabila dibandingkan dengan algoritma Brute Force

Lampiran

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.	✓	

Link

https://github.com/dimasb1954/Tucil2_13522112.git