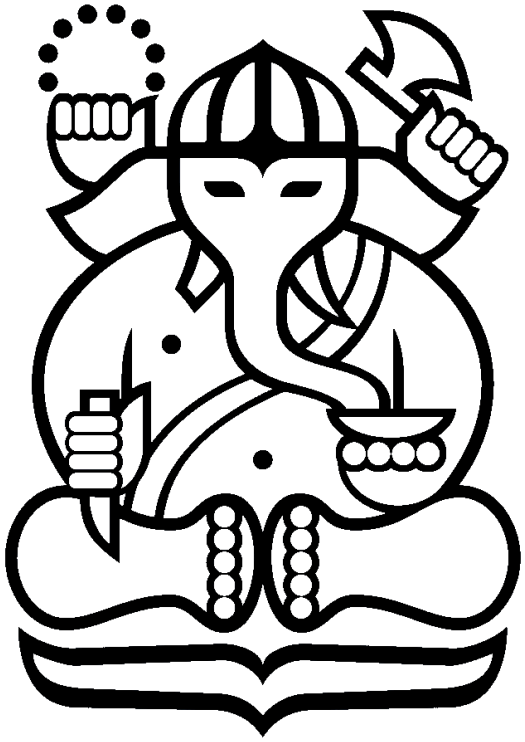
# LAPORAN TUGAS BESAR I
# IF2211 STRATEGI ALGORITMA

## Membangun Kurva Bézier
## dengan Algoritma Titik Tengah
## berbasis *Divide and Conquer*




**Disusun Oleh:**

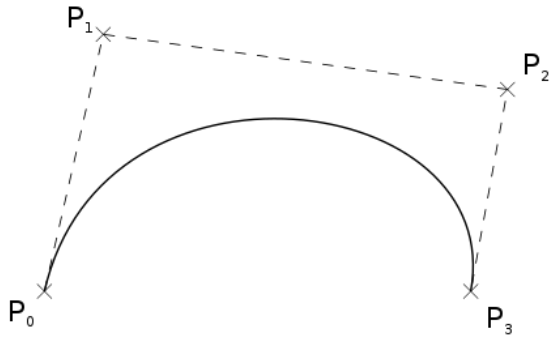
Zachary Samuel Tobing / 13522016

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


**2024**

# Bab I
# Latar Belakang

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.




**Gambar 1.** Kurva Bézier Kubik
(Sumber: https://id.wikipedia.org/wiki/Kurva_B%C3%A9zier)

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.

Kurva Bezier kuadratik adalah salah satu jenis kurva Bezier yang memiliki orde dua, yang berarti bahwa kurva ini ditentukan oleh tiga titik kontrol. Dalam kurva Bezier kuadratik, dua titik kontrol menentukan ujung-ujung kurva, sedangkan satu titik kontrol menentukan "panggung tengah" di mana kurva "menarik" atau "ditarik" menuju titik tersebut.

Algoritma titik tengah (*midpoint algorithm*) untuk membentuk sebuah kurva Bezier adalah salah satu metode numerik yang digunakan untuk menghasilkan titik-titik yang berada pada kurva Bezier dengan menggunakan pendekatan titik-tengah atau *midpoint*. Kurva Bezier adalah kurva parametrik yang digunakan dalam grafika komputer untuk merepresentasikan bentuk-bentuk yang halus seperti kurva atau permukaan.

Pada tugas ini, diminta membuat algoritma membuat sebuah **kurva Bézier kuadratik** dengan menggunakan algoritma titik tengah berbasis *divide and conquer*.

# Bab II
# Brute Force

## 2.1 Brute Force

Algoritma *brute force* adalah pendekatan yang sederhana dan langsung dalam menyelesaikan masalah dengan mencoba semua kemungkinan solusi secara berurutan dan memilih solusi terbaik yang ditemukan. Istilah "brute force" sendiri dapat diartikan sebagai kekuatan kasar atau kekerasan kasar, yang menggambarkan pendekatan ini yang mencoba semua kemungkinan tanpa menggunakan pengetahuan tambahan atau strategi yang lebih canggih.

Karakteristik umum dari algoritma brute force termasuk:

1. Pencarian Exhaustif: Algoritma ini mencoba semua kemungkinan solusi yang mungkin dengan memeriksa setiap opsi secara berurutan.
2. Kesederhanaan: Pendekatan ini sering kali mudah dipahami dan diimplementasikan karena tidak memerlukan pengetahuan khusus atau strategi yang kompleks.
3. Kinerja Tidak Efisien: Meskipun mudah dipahami, algoritma brute force cenderung memiliki kinerja yang buruk, terutama untuk masalah-masalah dengan ukuran masukan yang besar, karena mencoba semua kemungkinan solusi dapat menjadi tidak praktis.
4. Kesimpulan yang Akurat: Karena mencoba semua kemungkinan, algoritma brute force dapat menjamin menemukan solusi optimal, jika ada, untuk masalah yang diberikan.

Meskipun algoritma *brute force* sering kali tidak efisien untuk masalah-masalah besar, namun kadang-kadang merupakan pilihan yang baik untuk masalah-masalah dengan ukuran masukan yang kecil atau ketika tidak ada solusi yang lebih efisien yang tersedia. Selain itu, algoritma *brute force* juga dapat digunakan sebagai langkah awal dalam pengembangan solusi yang lebih efisien, seperti dalam tahap pencarian solusi untuk memverifikasi kebenaran atau untuk menguji koreksi implementasi algoritma yang lebih canggih.

## 2.2 Algoritma Brute Force yang Digunakan

### 2.2.1 Spek

Algoritma *brute force* digunakan untuk mencari kurva sehalus mungkin dengan cara sebanyak mungkin iterasi. Titik-titik pada kurva Bezier kuadratik dapat dicari dengan rumus berikut:

$$\mathbf{B}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2],\ 0 \le t \le 1,$$

**Gambar 2.2.1.1** Rumus Bezier Kuadratik

Dengan B(t) berupa nilai fungsi pada parameter t antara 0 dan 1, $P_i$ berupa titik kontrol yang dapat disederhanakan menjadi:

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2,\ 0 \leq t \leq 1.$$

**Gambar 2.2.1.2** Rumus Sederhana Bezier Kuadratik

Dengan memanggil rumus ini pada *interval* t yang sangat kecil, nilai yang diperoleh akan semakin banyak dan jika digabungkan dalam sebuah ilustrasi, akan terbentuk kurva yang sangat halus sehingga hampir menjadi sebuah garis mulus. Akan tetapi, karena perlu banyaknya pemanggilan rumus ini, waktu yang dibutuhkan untuk menyelesaikan kurva ini dan mendapatkan hasil yang akurat akan sangat lama.

Ketentuan algoritma *brute force* dipenuhi dengan pencarian exhaustif setiap titik dengan rumus, kesederhanaan dengan menggunakan rumus saja, kinerja tidak efisien karena harus mencari banyak sekali titik untuk membentuk kurva, tetapi memberikan kesimpulan yang akurat dengan terpetakan semua nilai dengan baik.

### 2.2.2 Alternatif : Array

Dalam membentuk kurva Bezier kuadratik, akan selalu ada 3 titik kontrol. Dalam algoritma brute force yang digunakan, sifat ini digunakan sangat spesifik untuk 3 titik kontrol. Fungsi tambahan yang dipakai juga hanya midpoint yang menghitung titik tengah antara 2 titik. Program yang digunakan juga masih bersifat iteratif sebanyak yang diinginkan melewati pemanggilan fungsi pada program utama. Perlu dicatat juga bahwa akan menggunakan istilah "penggabungan ke-..." untuk memudahkan penjelasan algoritma *brute force* yang merujuk pada penggabungan titik sesuai urutannya.

Langkah algoritmanya sebagai berikut:

1. Dua *array* dari tipe *point* didefinisikan yaitu *points* dan *display*. Points akan digunakan untuk menentukan dan menghitung titik kontrol dan titik tengah dari setiap iterasi dan *display* digunakan untuk hanya menampilkan titik yang dihitung dan menjadi bagian dari kurva bezier akhir.
2. Untuk iterasi pertama, *array* masih kosong karena belum ada titik yang dimasukkan pada points. Oleh karena itu, array akan ditambahkan 6 titik karena dimasukkan 3 titik kontrol awal (t0,t1,t2) ditambahkan dengan titik tengah t0 dan t1 (t01), titik tengah t1 dan t2 (t12), dan titik tengah keduanya (t0112), sementara titik tengah keduanya (t0112) merupakan titik pertama dari kurva bezier yang akan dibentuk sehingga akan ditambahkan pada *display* juga.
3. Untuk iterasi kedua, sudah ada 6 titik pada *points*. Untuk mencari titik tengah baru, digunakan informasi dari 3 titik sebelumnya yang dibentuk (t01,t12,t0112) karena akan menjadi titik patokan baru. Untuk iterasi kedua, jumlah titik baru yang dibentuk sebanyak

2 karena akan dipisah menjadi bagian kiri dan bagian kanan. Kedua titik dihitung dengan cara serupa dengan iterasi sebelumnya karena konsepnya sama tetapi bentuk dan titiknya saja yang berbeda. Titik bagian kiri akan dihitung dengan titik tengah t0 dan t01 (t001), titik tengah t01 dan t0112 (t010112), dan titik tengah keduanya (t001010112), sementara titik bagian kanan akan dihitung dengan titik tengah t2 dan t12 (t212), titik tengah t12 dan t0112 (t120112), dan titik tengah keduanya (t212120112). Titik t0 dan t2 merupakan titik kontrol awal yang selalu disimpan sebagai parameter fungsi sehingga akan terus ada dan hanya tinggal dipanggil, sementara t01, t12, dan t0112 diperoleh dari *points* hasil iterasi sebelumnya yang semuanya berupa 3 titik terakhir pada *points* sehingga tinggal dipanggil. Semua titik baru yang baru dibuat pada iterasi kedua ini kemudian dimasukkan pada *points* dan kedua titik hasil gabungan pada setiap bagian (kiri dan kanan) dimasukkan pada *display* sehingga pada akhir akan ada 6 titik baru pada *points* yang akan digunakan pada iterasi selanjutnya.

4. Iterasi ketiga dan selanjutnya sudah sama semua karena sudah harus membuat 2 titik baru sehingga metodenya sendiri serupa dengan iterasi kedua, perbedaannya ada pada titik yang diberikan pada iterasi sebelumnya. Pada iterasi kedua, jumlah titik yang diberikan dari iterasi sebelumnya (iterasi pertama) hanya 3 titik kontrol karena baru 1 sisi, berbeda dengan iterasi ketiga dan seterusnya yang sudah menerima 6 titik kontrol baru (2 sisi, kiri dan kanan) dan representasi nilai yang digunakan juga berbeda. Pada iterasi pertama, untuk membentuk titik tengah pertama, diperlukan titik tengah "gabungan kedua" (berdasarkan urutan penggabungan yang sudah dijelaskan) (t12) dalam perhitungan. Akan tetapi, sejak sudah dibagi menjadi 2 bagian, titik dari "gabungan kedua" sudah tidak dipakai lagi, sehingga algoritma hanya mengambil "gabungan pertama" dan "gabungan ketiga". Oleh karena itu, nilai yang diambil hanya elemen indeks pertama, ketiga, keempat, dan keenam terakhir dari *points* meskipun pada akhirnya akan dimasukkan juga semua titik yang dihasilkan iterasi saat ini seperti iterasi kedua.

Ketentuan algoritma *brute force* dipenuhi dengan pencarian exhaustif setiap titik dengan operasi yang berulang, kesederhanaan dengan menggunakan operasi *midpoint* saja, kinerja tidak efisien karena harus mencari banyak sekali titik untuk membentuk kurva, tetapi memberikan kesimpulan yang akurat dengan terpetakan semua nilai dengan baik.

# Bab III
# Algoritma Divide and Conquer

## 3.1 Divide and Conquer

Algoritma *divide and conquer* (bagi dan kuasai) adalah pendekatan algoritmik yang terdiri dari tiga langkah utama: "bagi", "taklukkan", dan "gabung". Pendekatan ini bertujuan untuk memecahkan masalah yang kompleks menjadi submasalah yang lebih kecil, menyelesaikan masing-masing sub masalah secara terpisah, dan kemudian menggabungkan solusi-solusi submasalah tersebut untuk mendapatkan solusi akhir dari masalah asli.

Langkah-langkah umum dalam algoritma *divide and conquer* adalah sebagai berikut:

1. Bagi (*Divide*): Langkah pertama adalah membagi masalah asli menjadi dua atau lebih submasalah yang lebih kecil, biasanya dengan ukuran yang sama atau hampir sama. Pembagian ini dilakukan secara rekursif hingga masalah terpecah menjadi ukuran yang cukup kecil untuk diselesaikan secara langsung (basis kasus).
2. Taklukkan (*Conquer*): Langkah kedua adalah menyelesaikan masing-masing sub masalah secara terpisah. Biasanya, sub masalah diselesaikan dengan menggunakan pendekatan rekursif, terutama jika submasalah tersebut dapat dibagi lagi.
3. Gabung (*Combine*): Langkah terakhir adalah menggabungkan solusi-solusi dari sub masalah yang telah diselesaikan menjadi solusi akhir dari masalah asli.

Pendekatan *divide and conquer* sering digunakan untuk menyelesaikan masalah-masalah dalam algoritma dan pemrograman yang kompleks, karena dapat meningkatkan efisiensi dan mempermudah pemahaman serta implementasi algoritma.

## 3.2 Algoritma Divide and Conquer yang Digunakan

Dalam pembuatan algoritma divide and conquer, digunakan pendekatan yang sangat berbeda dengan brute force. Jika pada brute force menggunakan titik yang diperoleh sebelumnya langsung pada beberapa nilai akhir di *points* dan menjalankan prosedur yang serupa terus-menerus, algoritma *divide and conquer* menggunakan dan menganggap sebagai sifat awal kurva Bezier kuadratik itu sendiri (3 titik kontrol) sehingga setiap operasi akan sama dengan menghitung titik kontrol pada iterasi pertama, diperoleh dengan memodifikasi parameter t0, t1, dan t2 yang awalnya bersifat sebagai 3 titik kontrol awal. Jumlah iterasi juga dicatat melalui parameter (indeks) dengan nilai n sebagai jumlah iterasi yang diinginkan dipertahankan dalam parameter sebagai pembanding dengan indeks.

Pada penjelasan langkah algoritma juga menggunakan beberapa tanda kutip seperti “kiri”, “tengah”, dan “kanan” untuk mengilustrasikan seperti gambar awal 3 titik kontrol yang disambungkan, sebelum ada iterasi atau perhitungan titik tengah apapun.

Langkah-langkah algoritmanya sebagai berikut:

1. *Base case* berupa indeks = n-1 karena setiap blok operasi akan menghitung titik tengah akhir sehingga hanya diperlukan pemanggilan rekursi sebanyak n-1. *Base case* hanya akan menghasilkan titik tengah akhir seperti iterasi pertama.
2. Proses perhitungan titik tengah akhir berlaku sama pada basis maupun rekursi, diawali dengan proses yang sama untuk iterasi pertama kali, yaitu menghitung titik tengah t0 dan t1 (t01), titik tengah t1 dan t2 (t12), kemudian menghitung titik tengah keduanya (t0112).
3. Perbedaan *base case* dan tahap rekursi ada pada pemanggilan fungsi rekursi saja. Pada tahap rekursi, setelah perhitungan titik tengah akhir berhasil diperoleh, dipanggil fungsi rekursif itu lagi, dibagi menjadi 2 bagian, yaitu kiri dan kanan karena selain iterasi pertama (*base case*), kurva harus membentuk 2 titik baru pada setiap sisinya. Algoritma ini berfokus pada memanipulasi ketiga titik kontrol sebagai parameter menjadi 3 "titik kontrol" baru pada bagian berikutnya. Artinya, pada bagian kiri t0 tetap t0 karena berupa sisi "kiri" seperti t0 di awal yang tidak pernah berubah, t01 menjadi parameter untuk t1 karena berupa sisi "tengah" seperti t1 di awal, dan t0112 menjadi parameter untuk t2 t0112 menjadi parameter t0 karena berupa sisi "kanan" seperti t2 di awal, dan pada bagian kanan t0112 menjadi parameter t0 karena berupa sisi "kiri" seperti t0 di awal, t12 sebagai parameter t1 karena berupa sisi "tengah" seperti t1, dan t2 tetap menjadi parameter t2 karena bagian "kanan" kurva tidak akan pernah berubah.
4. Semua titik yang dihasilkan dari perhitungan dimasukkan pada *points*, sementara titik tengah akhir dari setiap blok akan dimasukkan pada *display* sebagai bagian dari kurva bezier akhir.
5. Setiap pemanggilan rekursi akan menambahkan *increment* parameter indeks hingga mencapai n-1.

Bagi (*Divide*) terjadi ketika kurva awal dibagi menjadi bagian kiri dan kanan berdasarkan bagiannya dengan mengubah ketiga nilai parameter seperti bentuk awal dengan 3 titik kontrol mulai iterasi kedua dan akan terus dibagi dua hingga fungsi rekursi mencapai basis, yaitu ke dalam rekurens sebesar n-1 dengan n sebagai jumlah iterasi. Langkah ini membagi masalah menjadi 2 upa persoalan setiap langkahnya sehingga ukuran permasalahan menjadi setengahnya dan kompleksitasnya adalah O(log n).

Taklukkan (*Conquer*) terjadi setiap blok kode, baik dalam rekurens maupun basis dengan menghitung nilai tengah sesuai bentuk 3 titik kontrol awal sebelum ada operasi apapun. Titik tengah dari setiap bagian adalah solusi untuk setiap upa persoalan. Langkah ini menggunakan jumlah operasi yang bersifat konstan, terlepas dari jumlah data, sehingga kompleksitasnya O(1).

Gabung (*Combine*) terjadi secara implisit karena semua hasil titik tengah yang ditaklukkan pada setiap upa persoalan ditaruh dalam *display* yang mengandung setiap titik yang menjadi bagian kurva Bezier kuadratik di akhir. Hasil *return* dari rekurens sebenarnya tidak relevan karena tidak akan digunakan pada kurva Bezier akhir, tetapi digunakan untuk mempertahankan nilai yang melewati fungsi rekursifnya. Langkah penggabungan yaitu *appending* ke dalam *array* sehingga akan konstan dan kompleksitasnya O(1).

# Bab IV
# Source Code

## 4.1 Brute Force

```
# rumus spek (utama)

def spekqp(t0, t1, t2, i):
    # semacam persentase koordinat
    q0 = Point((1-i)*t0.x + i*t1.x, (1-i)*t0.y + i*t1.y)
    q1 = Point((1-i)*t1.x + i*t2.x, (1-i)*t1.y + i*t2.y)
    final = Point((1-i)*q0.x + i*q1.x, (1-i)*q0.y + i*q1.y)
    return final
```

**Gambar 4.1.1** Fungsi Brute Force

```
def spekqc(t0, t1, t2, i):
    itr = 1/i
    temparray = []
    for i in np.arange(0.0, 1.001, itr):
        temp = spekqp(t0,t1,t2,i)
        temparray.append(temp)
    return temparray
```

**Gambar 4.1.4** Pemanggilan Fungsi untuk Jumlah Iterasi

## 4.2 Divide and Conquer

```python
# points, display declare dari luar fungsi kosong dan akan diisi
def greedyqc(t0,t1,t2,i,n,points,display):
    if (i == n-1):
        newt1kiri = midpoint(t0,t1)
        points.append(newt1kiri)
        newt1kanan = midpoint(t1,t2)
        points.append(newt1kanan)
        newcomb = midpoint(newt1kiri,newt1kanan)
        points.append(newcomb)
        display.append(newcomb)
        return newcomb
    else: # sisanya hanya proses
        # hitung dan masukan array
        newt1kiri = midpoint(t0,t1)
        points.append(newt1kiri)
        newt1kanan = midpoint(t1,t2)
        points.append(newt1kanan)
        newcomb = midpoint(newt1kiri,newt1kanan)
        points.append(newcomb)
        display.append(newcomb)

        # rekursi
        left_result = greedyqc(t0, newt1kiri, newcomb, i + 1, n, points, display)
        right_result = greedyqc(newcomb, newt1kanan, t2, i + 1, n, points, display)
        return midpoint(left_result, right_result)
```

**Gambar 4.2.1** Algoritma Divide and Conquer

## 4.3 Main

```python
import bruteforce
import dividenconquer
import dividenconquerumumbonus
import matplotlib.pyplot as plt
import time

# inisialisasi kelas point
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def displaypoint(t0):
    print(t0.x,t0.y)

def displayarraypoint(arr):
    for i in range (len(arr)):
        displaypoint(arr[i])
    print()

while True:
    try:
        # Minta input
        jtitik = int(input("Masukkan banyak titik: "))

        # Cek brute force atau divide and conquer (tidak ada brute force untuk bonus)
        if jtitik != 3 and jtitik > 0:
            pilihan = 2
            break
        elif jtitik == 3:
            break
        else:
            print("Masukkan bilangan positif\n")
    except ValueError:
        # bukan integer
        print("Masukkan angka yang benar\n")
```

**Gambar 4.3.1** Main Part 1

```python
# fungsi validasi titik
def get_point_input(indeks):
    while True:
        try:
            titikmasukan = input("Titik ke-"+ str(indeks) + ": ")

            # Pisah
            xstr, ystr = titikmasukan.split(',')

            # Ubah jadi float
            x = float(xstr)
            y = float(ystr)

            # Mengeluarkan titiknya
            return Point(x, y)
        except ValueError:
            # Coba sampai valid
            print("Masukkan lagi sesuai format <x,y>\n")

# minta masukan titik
arraytitik = []
print("Masukkan titik sesuai format <x,y>")
for i in range (jtitik):
    temp = get_point_input(i+1)
    arraytitik.append(temp)
print()

# minta jumlah iterasi
while True:
    try:
        # Minta input
        jiterasi = int(input("Masukkan jumlah iterasi: "))

        # Cek brute force atau divide and conquer (tidak ada brute force untuk bonus)
        if jiterasi <= 0:
            print("Masukkan angka positif\n")
        else:
            break
    except ValueError:
        # bukan integer
        print("Masukkan angka yang benar\n")
```

**Gambar 4.3.2** Main Part 2

```python
# sesuai masukan
if (jtitik == 3):
    while True:
        try:
            # Ask for input
            print("Pilih algoritma penyelesaian:")
            print("1. Brute Force")
            print("2. Divide and Conquer")
            pilihan = int(input("Masukkan pilihan: "))

            # Cek brute force atau divide and conquer
            if pilihan < 1 or pilihan > 2: # salah input
                print("Masukkan angka yang benar (1 atau 2)\n")
            else: # bisa kedua cara
                break
        except ValueError:
            # bukan integer
            print("Masukkan angka yang benar (1 atau 2)\n")

# inisialisasi array untuk diisi
points = []
display = []

# operasi
if (jtitik == 3 and pilihan == 1): # brute force
    startbrute = time.time()
    display = bruteforce.spekqc(arraytitik[0],arraytitik[1],arraytitik[2],jiterasi)
    endbrute = time.time()
    executiontime = endbrute-startbrute
elif (jtitik == 3 and pilihan == 2): # divide and conquer
    startdnc = time.time()
    dividenconquer.dividenconquerqc(arraytitik[0],arraytitik[1],arraytitik[2],0,jiterasi,points,display)
    enddnc = time.time()
    executiontime = enddnc-startdnc
else: # divide and conquer bonus
    startdncbonus = time.time()
    dividenconquerumumbonus.dividenconquerqcgen(arraytitik,0,jiterasi,points,display)
    enddncbonus = time.time()
    executiontime = enddncbonus-startdncbonus
```

**Gambar 4.3.3** Main Part 3

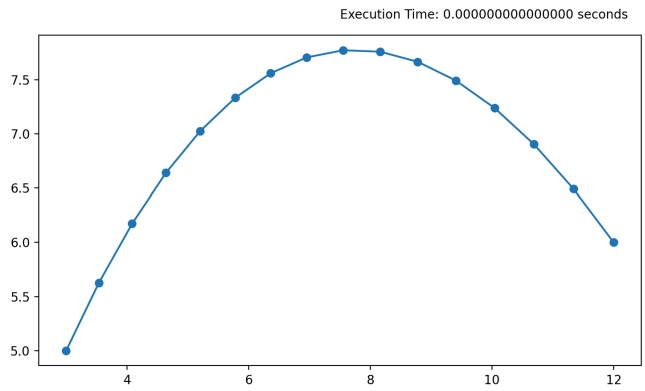
```
# Pisah point
xdisplay = [point.x for point in display]
ydisplay = [point.y for point in display]

sorted_indices = sorted(range(len(xdisplay)), key=lambda i: xdisplay[i])
x_values_sorted = [xdisplay[i] for i in sorted_indices]
y_values_sorted = [ydisplay[i] for i in sorted_indices]

displayarraypoint(display)
print(len(display))

# Plot the points
plt.plot(x_values_sorted, y_values_sorted, marker='o', linestyle='-')

plt.text(0.5, 1.05, f'Execution Time: {executiontime:.15f} seconds', transform=plt.gca().transAxes, fontsize=10)

# Display
plt.show()
```

**Gambar 4.3.4** Main Part 4

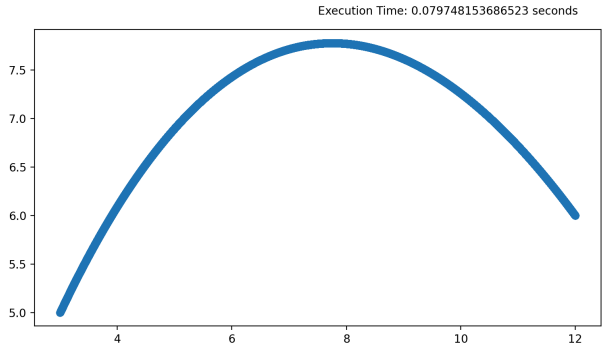
# Bab V
# Test Case

## 5.1 Test Case 1 (Random)

Control Point 1: (3, 5)
Control Point 2: (7, 10)
Control Point 3: (12, 6)
Iterasi : 15



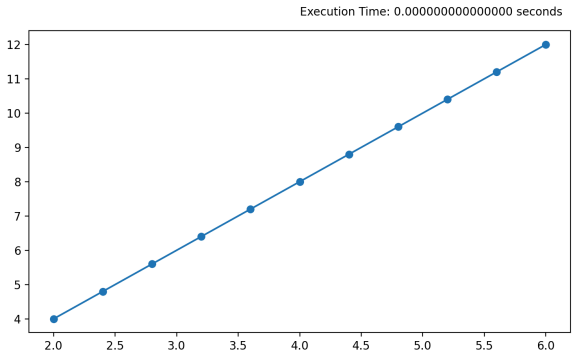

**Gambar 5.1.1** Hasil Test Case 1 Brute Force



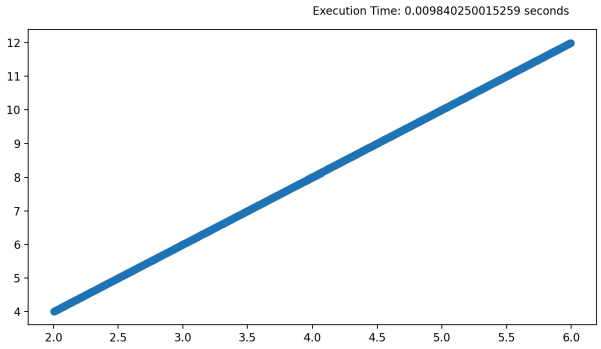

**Gambar 5.1.2** Hasil Test Case 1 Divide and Conquer

## 5.2 Test Case 2 (Linier)

Control Point 1: (2, 4)
Control Point 2: (4, 8)
Control Point 3: (6, 12)
Iterasi : 10



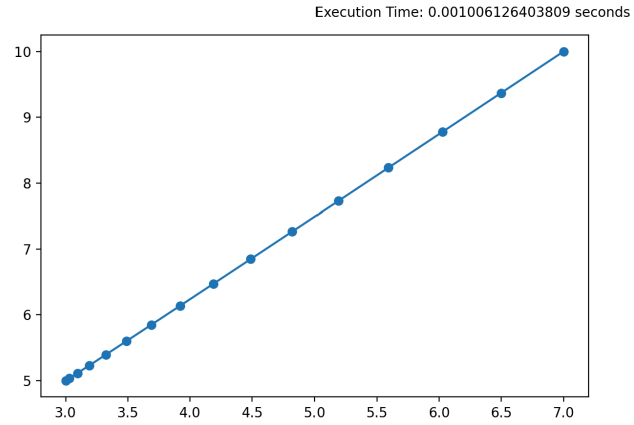

**Gambar 5.2.1** Hasil Test Case 2 Brute Force



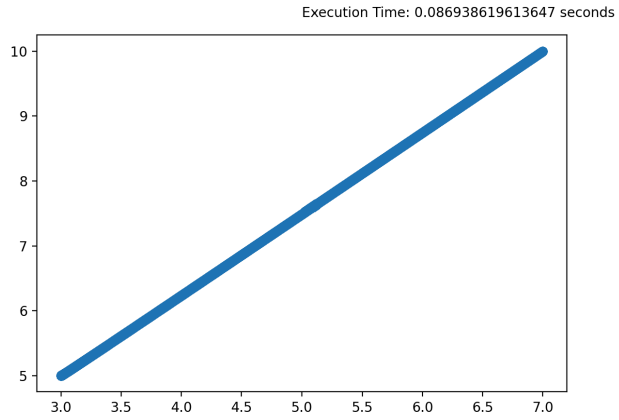

**Gambar 5.2.2** Hasil Test Case 2 Divide and Conquer

## 5.3 Test Case 3 (Collapsed)

Control Point 1: (3, 5)
Control Point 2: (3.1, 5.1)
Control Point 3: (7, 10)
Iterasi : 15



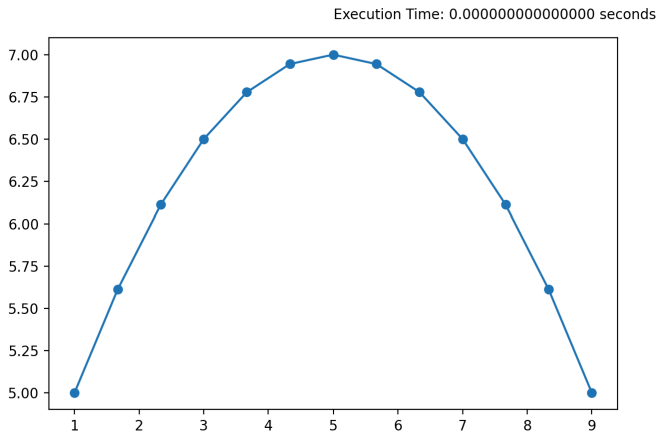

**Gambar 5.3.1** Hasil Test Case 3 Brute Force



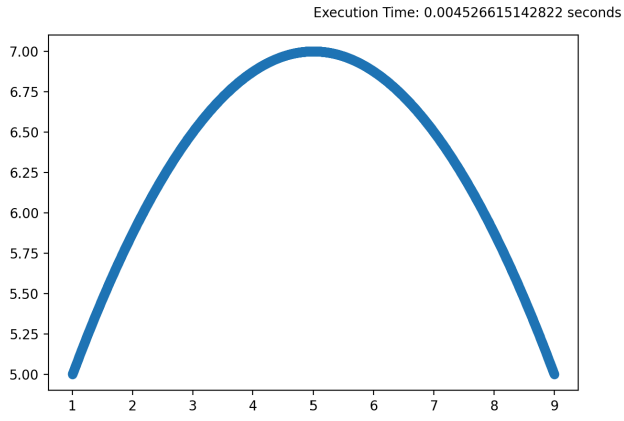

**Gambar 5.3.2** Hasil Test Case 3 Divide and Conquer

## 5.4 Test Case 4 (Simetris)

Control Point 1: (1, 5)
Control Point 2: (5, 9)
Control Point 3: (9, 5)
Iterasi : 12



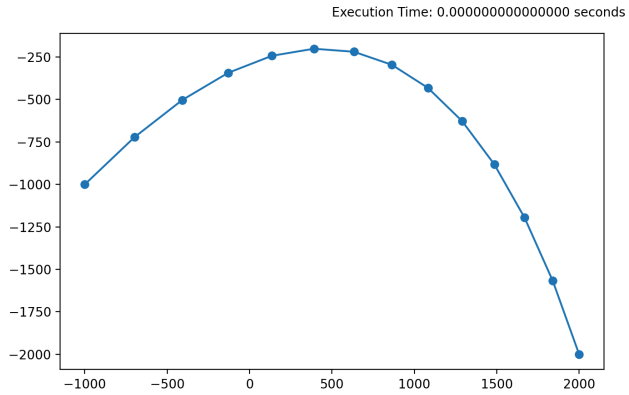

**Gambar 5.4.1** Hasil Test Case 4 Brute Force



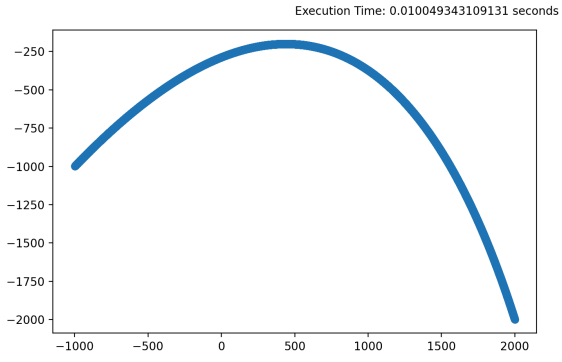

**Gambar 5.4.2** Hasil Test Case 4 Divide and Conquer

## 5.5 Test Case 5 (Ekstrim)

Control Point 1: (-1000, -1000)
Control Point 2: (1000, 1000)
Control Point 3: (2000, -2000)
Iterasi : 13



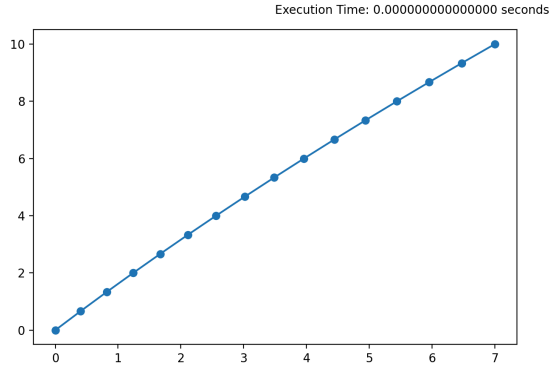

**Gambar 5.5.1** Hasil Test Case 5 Brute Force



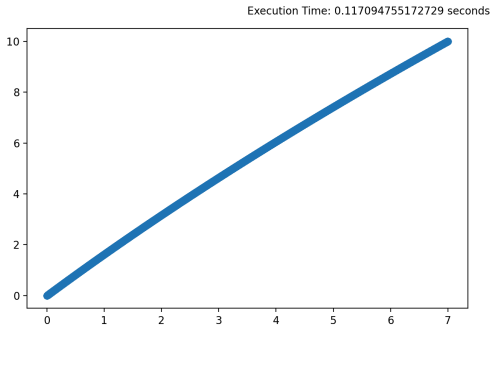

**Gambar 5.5.2** Hasil Test Case 5 Divide and Conquer

## 5.6 Test Case 6 (Origin)

Control Point 1: (0,0)
Control Point 2: (3, 5)
Control Point 3: (7, 10)
Iterasi : 15



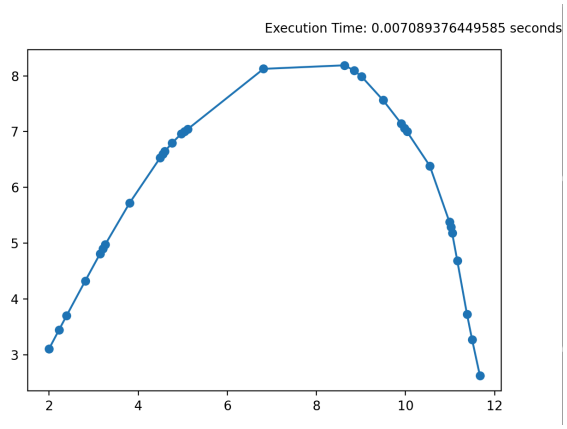

**Gambar 5.6.1** Hasil Test Case 6 Brute Force




**Gambar 5.6.2** Hasil Test Case 6 Divide and Conquer

## 5.7 Test Case 7 (Bonus #1)

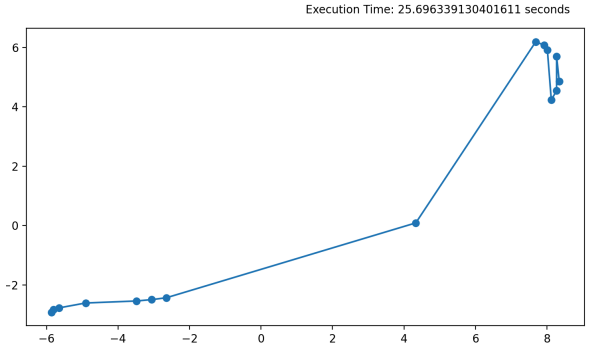
Control Point 1: (0, 0)
Control Point 2: (3, 5)
Control Point 3: (7, 10)
Control Point 4: (10, 15)
Control Point 5: (15, -10)
Iterasi : 5




**Gambar 5.7.1** Hasil Test Case 7 Bonus

## 5.8 Test Case 8 (Bonus #2)

Control Point 1: (0, 0)
Control Point 2: (3, 5)
Control Point 3: (7, 10)
Control Point 4: (10, 15)
Control Point 5: (15, -10)
Control Point 6: (-1,1)
Control Point 7: (-2,-4)
Control Point 8: (-10,-5)
Control Point 9: (-15,5)
Control Point 10: (-2,-15)
Iterasi : 4




**Gambar 5.8.1** Hasil Test Case 8 Brute Force

# Bab VI
# Analisis Perbandingan

Pada setiap *test case*, dapat dilihat dengan mudah bahwa algoritma *brute force* jauh lebih cepat daripada algoritma *divide and conquer*. Ini dikarenakan beberapa hal yaitu:

- Lebih banyak nilai yang dihasilkan *divide and conquer* : setiap rekurens membuat 2 titik baru sementara setiap iterasi *brute force* hanya menghasilkan 1 titik sehingga operasi yang dibutuhkan untuk menyelesaikan algoritma *divide and conquer* jauh lebih banyak daripada iterasi *brute force*.

- Jumlah operasi untuk setiap blok iterasi maupun blok rekurens berjumlah sama sehingga yang akan berpengaruh adalah seberapa banyak blok pada iterasi dan rekursif akan dipanggil oleh program.

- Kompleksitas lebih tinggi pada algoritma *divide and conquer* yang berupa log n sementara kompleksitas *brute force* adalah konstan (1) sehingga algoritma *divide and conquer* akan lebih lama berjalannya daripada *brute force*.

# Bab VII
# Bonus

Pada bonus, dilakukan pendekatan yang serupa dengan algoritma *divide and conquer* pada versi kurva Bezier kuadratik. Perbedaannya ada pada parameter yang dipegang, yaitu jika ada titik sejumlah n, tidak dapat dibuat setiap titik menjadi parameter sehingga diperlukan sebuah *array*. Perbedaannya sepertinya sederhana, tetapi implementasinya menjadi sangat berbeda dari biasanya.

Langkah-langkah algoritmanya adalah:

1. Basis serupa dengan implementasi *divide and conquer* orde 2 yaitu ketika indeks mencapai n-1 karena setiap blok rekurens akan mencari titik tengah berikutnya.
2. Kemudian setiap elemen dari 0 hingga len(array)-2 diubah menjadi *midpoint* dari dirinya dan elemen pada indeks setelahnya. Ini merepresentasikan titik tengah dari kedua titik yang berdekatan. Jika proses ini terus dilakukan dengan menghapus elemen terakhir pada setiap iterasi hingga jumlah elemennya menjadi 1, itu adalah titik tengah dari semua titik pada *array* mula-mula yang akan dikembalikan oleh basis.
3. Pada rekurens, ada perbedaan karena perlunya pembagian persoalan menjadi bagian kiri dan kanan, serupa dengan algoritma *divide and conquer* sebelumnya. Semua elemen sebelah kiri dari panjang array dibagi 2 pada setiap iterasi ditaruh pada *arraykiri* dan semua elemen sebelah kanan dari panjang *array* dibagi 2 pada setiap iterasi ditaruh pada *arraykanan*.
4. Kedua *array* yang sudah dibentuk ini yang kemudian digunakan pada rekurens selanjutnya hingga mencapai basis dengan pencatatan indeks mencapai n-1.

Dibuat juga bonus untuk membuat animasi pembuatan titik-titiknya yang akan muncul setelah menutup gambar hasil yang terlebih dahulu ditampilkan.

# Bab VIII
# Pranala Repository

https://github.com/ZachS17/Tucil2_13522016

# Bab IX
# 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. | ✓ | |