

**LAPORAN TUGAS KECIL II**  
**IF2211 STRATEGI ALGORITMA**



Laporan ini dibuat untuk memenuhi tugas

Mata Kuliah IF 2211 Strategi Algoritma

Disusun Oleh:

Rayendra Althaf Taraka Noor (13522107)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

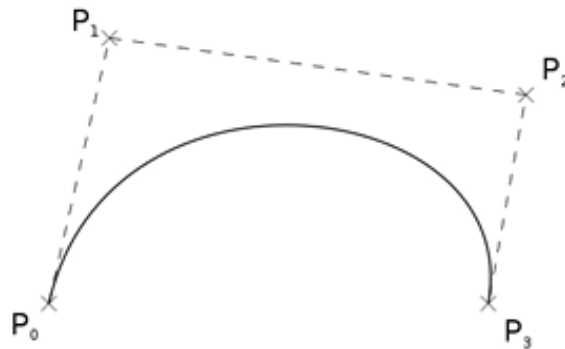
**INSTITUT TEKNOLOGI BANDUNG**

**SEMESTER II TAHUN 2023/2024**

## Daftar Isi

<b>Daftar Isi .....</b>	<b>2</b>
<b>BAB I DESKRIPSI TUGAS .....</b>	<b>3</b>
<b>BAB II ANALISIS PENYELESAIAN DAN IMPLEMENTASI .....</b>	<b>3</b>
2.1. Penyelesaian Brute Force .....	3
2.2. Penyelesaian Divide and Conquer .....	4
2.3. Kode Implementasi .....	5
<b>BAB III UJI KASUS .....</b>	<b>10</b>
<b>BAB IV ANALISIS PERBANDINGAN SOLUSI .....</b>	<b>13</b>
<b>LAMPIRAN .....</b>	<b>13</b>
<b>DAFTAR PUSTAKA .....</b>	<b>14</b>

## BAB I DESKRIPSI TUGAS



**Gambar 1.**  
Kurva Bézier Kubik

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.

Proses yang perlu dilakukan untuk mengkonstruksi kurva bezier bertambah banyak dan rumit seiring bertambahnya titik. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka kami diminta untuk mengimplementasikan pembuatan kurva Bézier dengan algoritma titik tengah berbasis divide and conquer.

## BAB II ANALISIS PENYELESAIAN DAN IMPLEMENTASI

### 2.1. Penyelesaian Brute Force

Kurva bezier dapat dibentuk dengan sebuah persamaan yang bergantung kepada banyaknya titik yang membentuknya. Persamaan ini diturunkan dari titik tengah pembentuknya dan dikarenakan penurunan tersebut persamaan ini memiliki pola yang menyerupai segitiga pascal. Sebagai contoh berikut beberapa contoh persamaan yang terbentuk:

- Persamaan untuk tiga titik

$$Q_0 = (1 - t)^2 P_0 + 2(1 - t)tP_1 + t^2 P_2$$

- Persamaan untuk empat titik

$$S_0 = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$$

- Persamaan untuk lima titik

$$T_0 = (1-t)^4 P_0 + 4(1-t)^3 t P_1 + 6(1-t)^2 t^2 P_2 + 4(1-t) t^3 P_3 + t^4 P_4$$

Sehingga untuk mengimplementasikannya secara *brute force* kita dapat menjalankan rumus tersebut sesuai dengan banyak titik pembentuknya dan menghitungnya untuk setiap titik. Untuk nilai  $t$  pada persamaan nilai tersebut akan divariasikan sesuai banyaknya iterasi yang diminta. Nilai  $t$  ini akan diubah-ubah dengan range 0 sampai 1 dan jarak per perubahannya dihitung dengan mengangkat 0.5 dengan banyaknya iterasi.

Dalam pengimplementasiannya, proses ini akan dijalankan dengan langkah-langkah berikut:

1. Menghitung jarak per perubahan dengan mengangkat 0.5 dengan jumlah iterasi
2. Membuat segitiga pascal dengan mengkonstruksinya hingga mencapai ke tingkat yang sesuai dengan jumlah titik
3. Menghitung persamaan dan mengulangnya untuk variasi  $t$  dari 0 sampai 1 dengan jarak setiap variasi  $t$  sebesar nilai yang telah dihitung pada langkah pertama

Untuk menganalisis kompleksitas waktu keseluruhan dari proses ini perlu dilakukan penghitungan kompleksitas waktu dari setiap langkahnya. Untuk memudahkan penyebutan, pada bagian selanjutnya banyak titik akan diwakili oleh  $n$  dan  $m$  mewakili banyaknya iterasi yang perlu dilakukan.

Pada langkah pertama, perlu dilakukan pemangkatan hingga  $m$  kali sehingga kompleksitas algoritmanya adalah

$$T_1(m) = O(m)$$

Selanjutnya untuk langkah yang kedua proses yang akan terjadi ialah, perancangan array berdasarkan array sebelumnya dengan penambahan. Jumlah elemen untuk setiap tingkatan arraynya akan sama dengan tingkatan array tersebut sehingga kompleksitas langkah ini ialah

$$T_2(n) = O(n^2)$$

Untuk langkah ketiga, proses tersebut akan mengulangi penyelesaian persamaan sebanyak 2 pangkat  $m$ . Dan dalam satu penyelesaian persamaan akan terdapat perpangkatan dengan besar pangkat maksimum  $n$  dikurangi 1 dan pemangkatan tersebut akan dilakukan sebanyak  $n$  kali (karena bukan bagian utama dari brute force pemangkatan pada bagian ini dilakukan dengan squaring sehingga kompleksitas pemangkatannya adalah  $O(\log n)$ ). Berdasarkan langkah proses yang barusan dijelaskan didapat kompleksitas waktunya adalah

$$T_3(n, m) = O(2^m n^2 \log n)$$

Terakhir, untuk menentukan kompleksitas waktu keseluruhan, kompleksitas waktu ketiga proses tersebut akan digabung menjadi satu sehingga didapat penyelesaian sebagai berikut

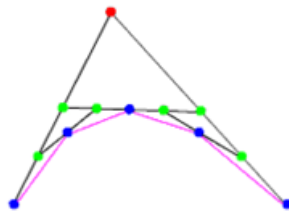
$$T_{(n,m)} = O(\max(m, n, 2^m n^2 \log n)) = O(2^m n^2 \log n)$$

Dan untuk kasus dasar (bukan bonus) dimana jumlah titik adalah 3 kompleksitas waktunya adalah

$$O(2^m)$$

## 2.2. Penyelesaian Divide and Conquer

Selain dengan menggunakan persamaan seperti pada subbab sebelumnya, pembentukan kurva bezier juga dapat dilakukan dengan metode titik tengah. Lebih jelasnya, dari semua kontrol poin akan diambil titik tengah lalu diambil titik tengahnya lagi hingga hanya tersisa 1 titik tengah dan titik tersebut dimasukkan ke dalam kumpulan titik akhir hasil pembentukan kurva. Hal tersebut adalah proses yang dilakukan dalam 1 iterasi. Jika iterasi berjumlah lebih dari 1 kurva bagian dalam dari proses pengambilan titik tengah akan dibagi 2 dan dilakukan pencarian titik tengah lagi dari 2 bagian tersebut. Proses ini berulang sebanyak jumlah iterasi yang diminta.



**Gambar 2.2.**

Hasil pembentukan Kurva Bézier Kuadratik dengan divide and conquer setelah iterasi ke-2.

Untuk memperjelas proses yang dilakukan dalam penggunaan paradigma Divide and Conquer, berikut langkah-langkah yang dilakukan dalam proses pembentukan kurva bezier dalam metode ini:

1. Hitung titik tengah dari titik tengah dan masukkan ke dalam list
2. Hitung titik tengah dari kumpulan titik tengah sebelumnya dan masukkan ke dalam list baru
3. Ulangi langkah ke-2 hingga hanya tersisa 1 titik tengah
4. Ambil kurva terdalam dan bagi menjadi 2 bagian dengan titik tengah yang didapat pada poin ke-3 dimasukkan ke dalam kedua bagian
5. Jika ini bukan iterasi terakhir, lakukan lagi proses (rekursif) ini secara terpisah dengan titik awal menjadi kumpulan titik bagian bagian pertama yang didapat dari pembagian pada langkah ke-4
6. Masukkan titik tengah dari bagian 3 ke dalam kumpulan titik kurva akhir
7. Jika ini bukan iterasi terakhir, lakukan lagi proses (rekursif) ini secara terpisah dengan titik awal menjadi kumpulan titik bagian kedua yang didapat dari pembagian pada langkah ke-4

Menghitung kompleksitas dari algoritma ini tidak serumit kompleksitas waktu pada metode sebelumnya. Jika diperhatikan, jumlah iterasi akan mempengaruhi jumlah fungsi ini yang terpanggil yaitu  $2^m - 1$ . dan kompleksitas 1 fungsi tanpa memanggil dirinya sendiri dapat dihitung dari banyaknya titik tengah yang perlu dibuat untuk mencapai titik tengah tunggal yaitu sebanyak  $n(n-1)/2$ . Dari penjelasan sebelumnya dapat ditentukan kompleksitas waktu algoritma pembentukan bezier curve dengan divide and conquer adalah

$$T(n) = O(2^m n^2)$$

### 2.3. Kode Implementasi

Pada tugas kali ini, kedua proses diimplementasikan dengan program dalam bahasa Python. Berikut kode lengkap kode dari program tersebut:

```
# File driver.py
from BezierCurve import BezierCurve
from util import Point
import time

# mengambil input dan validasi input
inp_nPoints = 0
while (inp_nPoints < 2):
    try:
        inp_nPoints = int(input("Masukkan jumlah titik (lebih besar atau sama dengan 2): "))
    except ValueError:
        print("Masukan bukan integer")

ps = []
```

```

for i in range(inp_nPoints):
    print(f"Titik ke-{i+1}")
    inp_x = "not defined"
    inp_y = "not defined"
    while not isinstance(inp_x, float) :
        try:
            inp_x = float(input("Masukkan Koordinat x: "))
        except ValueError:
            print("Masukan bukan float")
    while not isinstance(inp_y, float) :
        try:
            inp_y = float(input("Masukkan Koordinat y: "))
        except ValueError:
            print("Masukan bukan float")
    ps.append(Point(inp_x,inp_y))

inp_iteration = 0
while not(isinstance(inp_iteration, int) and inp_iteration>=1):
    try:
        inp_iteration = int(input("Masukkan jumlah iterasi (lebih besar atau sama dengan 1): "))
    except ValueError:
        print("Masukan bukan integer")

inp_type_method = ""
while (inp_type_method != "bf" and inp_type_method != "dnc"):
    print("Pilih tipe penyelesaian:\n1.Divide and conquer\n2.Bruteforce")
    inp_temp = input()
    if(inp_temp == "1"): inp_type_method = "dnc"
    if(inp_temp == "2"): inp_type_method = "bf"

start_time = time.time()
# membangun kurva bezier
bezier_curve = BezierCurve(ps,inp_iteration,inp_type_method)
bezier_curve.create_bezier()
# menampilkan waktu eksekusi
end_time = time.time()
execution_time_ms = (end_time - start_time) * 1000
# display hasil
bezier_curve.display_bezier(execution_time_ms)

```

```

# File BezierCurve.py
from util import Point,create_mid_point,display_2sets_dots_and_line
from collections import deque

class BezierCurve():
    def __init__(self,inp_points, inp_iteration,inp_method) -> None:

```

```

self.points = inp_points
self.nPoints = len(self.points)
self.iteration = inp_iteration
self.method = inp_method
self.bezier_points = []
self.power_of_2 = 1
self.pascal_triangle = []

# membuat bezier curve dengan divide and conquer
def dnc_populate_bezier_points(self, cur_points, iteration_count):
    if (iteration_count>0):
        left_points = [cur_points[0]]
        right_points = deque([cur_points[len(cur_points)-1]])

        # generate all mid points in 1 iteration
        n = self.nPoints
        while(n>1):
            new_points = []
            for i in range(n-1):
                new_point =
create_mid_point(cur_points[i],cur_points[i+1])
                new_points.append(new_point)
            if(i==0):
                left_points.append(new_point)
            if(i == n-2):
                right_points.appendleft(new_point)
            cur_points = new_points
            n-=1

        # rekursi bagian kiri dan kanan
        self.dnc_populate_bezier_points(left_points, iteration_count-1)
        self.tambah_point(left_points[len(left_points)-1])
        self.dnc_populate_bezier_points(right_points, iteration_count-1)

# membuat bezier curve dengan bruteforce
def bf_populate_bezier_points(self):
    dist_per_point = 0.5/self.power_of_2
    cur_dist = dist_per_point

    # mengisi setiap titik sesuai jumlah iterasi dan jumlah titik
    while(cur_dist<1):
        temp_x = 0
        temp_y = 0
        for i in range(self.nPoints):
            # print(f"(1-t):{(1-cur_dist)**(self.nPoints-
i)}\nt:{(cur_dist)**(i)}")
            temp_x +=
self.pascal_triangle[self.nPoints][i+1]*self.points[i].x*((1-
cur_dist)**(self.nPoints-i-1))*((cur_dist)**(i))

```

```

        tempy +=
self.pascal_triangle[self.nPoints][i+1]*self.points[i].y*((1-
cur_dist)**(self.nPoints-i-1))*((cur_dist)**(i))
        tempp = Point(tempx,tempy)
        self.tambah_point(tempp)
        cur_dist+=dist_per_point

# fungsi untuk menyiapkan nilai-nilai yang diperlukan dalam metode dnc
atau bf
def create_bezier(self):
    self.bezier_points = []
    self.bezier_points = []
    self.tambah_point(self.points[0])
    if(self.method == "dnc"):
        self.dnc_populate_bezier_points(self.points,self.iteration)
    else:
        for i in range(self.iteration-1):
            self.power_of_2=self.power_of_2*2
            self.pascal_triangle.append([0,0])
            self.pascal_triangle.append([0,1,0])
            # print(self.pascal_triangle)
            for i in range(2,self.nPoints+1):
                temp = []
                for j in range (i+2):
                    if(j == 0 or j == i +1):
                        temp.append(0)
                    else:
                        temp.append(self.pascal_triangle[i-1][j-
1]+self.pascal_triangle[i-1][j])
                        # print(self.pascal_triangle[i-1][j-
1]+self.pascal_triangle[i-1][j])
                self.pascal_triangle.append(temp)
                # print(self.pascal_triangle)
            self.bf_populate_bezier_points()
            self.tambah_point(self.points[len(self.points)-1])

def display_bezier(self,exec_time):
    display_2sets_dots_and_line(self.points,self.bezier_points,exec_time)

def tambah_point(self, point):
    self.bezier_points.append(Point(point.x,point.y))

```

---

```

# File util.py
from collections import namedtuple
import matplotlib.pyplot as plt

Point = namedtuple('Point', ['x', 'y'])

```



```

# mengembalikan titik tengah dari dua buah titik
def create_mid_point(point1, point2):
    createdPoint = Point((point1.x + point2.x) / 2, (point1.y + point2.y) /
2)
    return createdPoint

# menampilkan 2 set titik dengan setiap titik yang bersebalahan dihubungkan
dengan garis
def display_2sets_dots_and_line(points_1, points_2,exec_time):
    # variable for centering text
    max_x = "sapi"
    min_x = "sapi"
    max_y = "sapi"

    for i in range(len(points_1)-1) :
        plt.plot(points_1[i].x,points_1[i].y,'ro',markersize = 4, alpha=0.8)
        plt.plot([points_1[i].x, points_1[i+1].x],[points_1[i].y,
points_1[i+1].y],color = 'black',linewidth=1)
        if (max_x == "sapi" or max_x<points_1[i].x):
            max_x = points_1[i].x
        if (min_x == "sapi" or min_x>points_1[i].x):
            min_x = points_1[i].x
        if(max_y == "sapi" or max_y<points_1[i].y):
            max_y = points_1[i].y
    plt.plot(points_1[len(points_1)-1].x,points_1[len(points_1)-
1].y,'ro',markersize = 4, alpha=0.8)
    if (max_x == "sapi" or max_x<points_1[len(points_1)-1].x):
        max_x = points_1[len(points_1)-1].x
    if (min_x == "sapi" or min_x>points_1[len(points_1)-1].x):
        min_x = points_1[len(points_1)-1].x
    if(max_y == "sapi" or max_y<points_1[len(points_1)-1].y):
        max_y = points_1[len(points_1)-1].y

    for i in range(len(points_2)-1) :
        plt.plot(points_2[i].x,points_2[i].y,'bo',markersize = 4, alpha=0.8)
        plt.plot([points_2[i].x, points_2[i+1].x],[points_2[i].y,
points_2[i+1].y],color = 'blue',linewidth=1)
        if (max_x == "sapi" or max_x<points_2[i].x):
            max_x = points_2[i].x
        if (min_x == "sapi" or min_x>points_2[i].x):
            min_x = points_2[i].x
        if(max_y == "sapi" or max_y<points_2[i].y):
            max_y = points_2[i].y
    plt.plot(points_2[len(points_2)-1].x,points_2[len(points_2)-
1].y,'bo',markersize = 4, alpha=0.8)
    if (max_x == "sapi" or max_x<points_2[len(points_2)-1].x):
        max_x = points_2[len(points_2)-1].x
    if (min_x == "sapi" or min_x>points_2[len(points_2)-1].x):

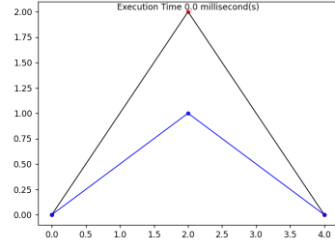
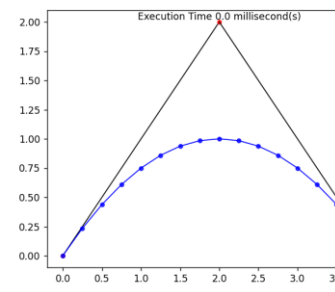
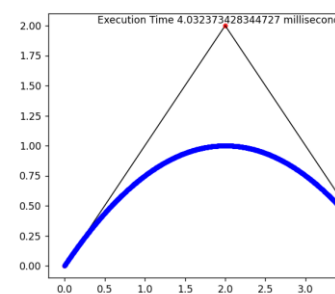
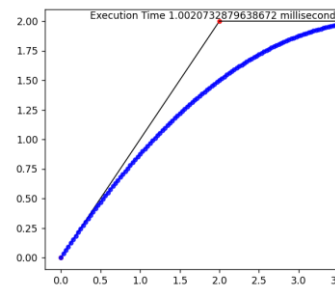
```

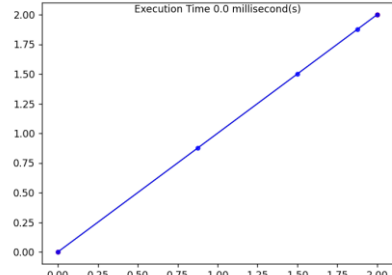
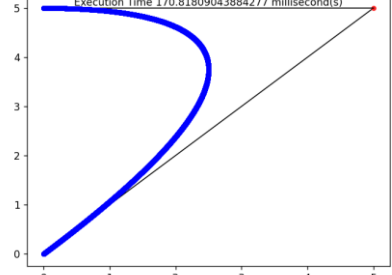
```

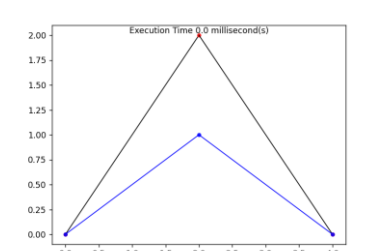
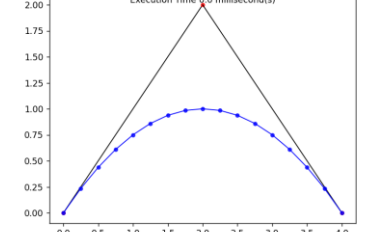
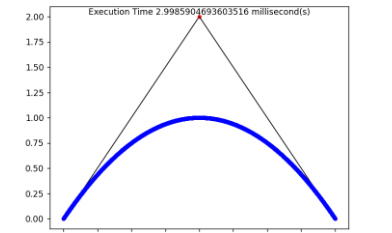
min_x = points_2[len(points_2)-1].x
if(max_y == "sapi" or max_y < points_2[len(points_2)-1].y):
    max_y = points_2[len(points_2)-1].y
mid_x = (max_x + min_x) / 2
plt.text(mid_x, max_y*1.01, f'Execution Time {exec_time}
millisecond(s)', ha='center')
plt.show()

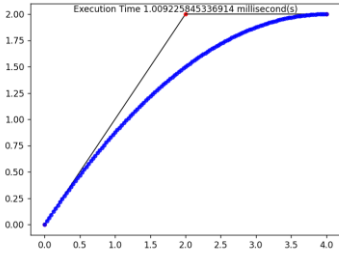
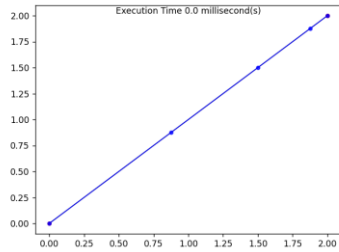
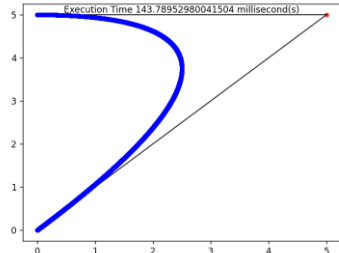
```

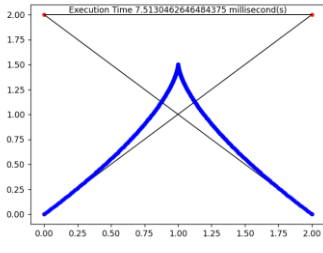
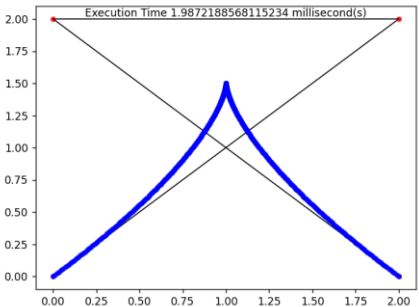
## BAB III UJI KASUS

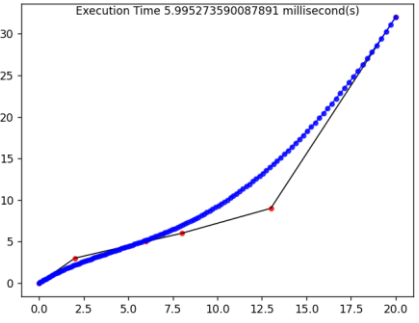
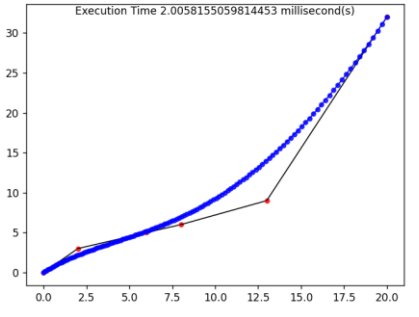
Penyelesaian Divide and Conquer		
Kasus 1	<pre> Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 4 Masukkan Koordinat y: 0 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 1 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 1 </pre>	
Kasus 2	<pre> Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 4 Masukkan Koordinat y: 0 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 4 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 1 </pre>	
Kasus 3	<pre> Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 4 Masukkan Koordinat y: 0 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 10 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 1 </pre>	
Kasus 4	<pre> Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 4 Masukkan Koordinat y: 0 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 7 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 1 </pre>	

Kasus 5	<pre> Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 2 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 1 </pre>	
Kasus 6	<pre> Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 5 Masukkan Koordinat y: 5 Titik ke-3 Masukkan Koordinat x: 0 Masukkan Koordinat y: 5 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 15 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 1 </pre>	

Penyelesaian Brute Force		
Kasus 1	<pre> Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 4 Masukkan Koordinat y: 0 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 1 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 2 </pre>	
Kasus 2	<pre> Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 4 Masukkan Koordinat y: 0 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 4 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 2 </pre>	
Kasus 3	<pre> Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 4 Masukkan Koordinat y: 0 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 10 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 2 </pre>	

Kasus 4	Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 4 Masukkan Koordinat y: 2 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 7 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 2	
Kasus 5	Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 2 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 2	
Kasus 6	Masukkan jumlah titik (lebih besar atau sama dengan 2): 3 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 5 Masukkan Koordinat y: 5 Titik ke-3 Masukkan Koordinat x: 0 Masukkan Koordinat y: 5 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 15 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 2	

Bonus		
Kasus 1 Divide and Conquer	Tucil2/repo/Tucil2_13522107/src/driver.py Masukkan jumlah titik (lebih besar atau sama dengan 2): 4 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 0 Masukkan Koordinat y: 2 Titik ke-4 Masukkan Koordinat x: 2 Masukkan Koordinat y: 0 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 9 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 1	
Kasus 1 Brute Force	Tucil2/repo/Tucil2_13522107/src/driver.py Masukkan jumlah titik (lebih besar atau sama dengan 2): 4 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 2 Titik ke-3 Masukkan Koordinat x: 0 Masukkan Koordinat y: 2 Titik ke-4 Masukkan Koordinat x: 2 Masukkan Koordinat y: 0 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 9 Pilih tipe penyelesaian: 1.Divide and conquer 2.Bruteforce 2	

<b>Kasus 2 Divide and Conquer</b>	Masukkan jumlah titik (lebih besar atau sama dengan 2): 6 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 3 Titik ke-3 Masukkan Koordinat x: 6 Masukkan Koordinat y: 5 Titik ke-4 Masukkan Koordinat x: 8 Masukkan Koordinat y: 6 Titik ke-5 Masukkan Koordinat x: 13 Masukkan Koordinat y: 9 Titik ke-6 Masukkan Koordinat x: 20 Masukkan Koordinat y: 32 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 7 Pilih tipe penyelesaian: 1. Divide and conquer 2. Brute force 1	
<b>Kasus 2 Brute Force</b>	Masukkan jumlah titik (lebih besar atau sama dengan 2): 6 Titik ke-1 Masukkan Koordinat x: 0 Masukkan Koordinat y: 0 Titik ke-2 Masukkan Koordinat x: 2 Masukkan Koordinat y: 3 Titik ke-3 Masukkan Koordinat x: 6 Masukkan Koordinat y: 5 Titik ke-4 Masukkan Koordinat x: 8 Masukkan Koordinat y: 6 Titik ke-5 Masukkan Koordinat x: 13 Masukkan Koordinat y: 9 Titik ke-6 Masukkan Koordinat x: 20 Masukkan Koordinat y: 32 Masukkan jumlah iterasi (lebih besar atau sama dengan 1): 7 Pilih tipe penyelesaian: 1. Divide and conquer 2. Brute force 2	

## BAB IV

### ANALISIS PERBANDINGAN SOLUSI

Berdasarkan hasil uji kasus, terlihat terdapat perbedaan waktu eksekusi dari penggunaan metode brute force dan divide and conquer. Metode brute force memiliki waktu eksekusi yang lebih cepat yang semakin terlihat pada kasus-kasus dimana jumlah iterasi dan/atau jumlah titik cukup besar. Hal ini sesuai dengan analisis kompleksitas waktu dimana metode brute force memiliki kompleksitas waktu

$$T(n) = O(2^m n^2 \log n)$$

dan metode divide and conquer memiliki kompleksitas waktu

$$T(n) = O(2^m n^2)$$

Hal ini juga menunjukkan bahwa metode divide and conquer tidak selalu lebih cepat dari metode brute force dalam semua kasus persoalan.

## LAMPIRAN

Link Repository : [https://github.com/RayNoor0/Tucil2\\_13522107](https://github.com/RayNoor0/Tucil2_13522107)

Poin	Ya	Tidak
1. Program berhasil dijalankan	✓	
2. Program dapat melakukan visualisasi kurva Bezier	✓	
3. Solusi yang diberikan program optimal	✓	
4. [Bonus] Program dapat membuat kurva untuk n titik kontrol.	✓	
5. [Bonus] Program dapat melakukan visualisasi proses pembuatan kurva.		✓

## DAFTAR PUSTAKA

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Tucil2-2024.pdf>

<https://www.codeproject.com/Articles/223159/Midpoint-Algorithm-Divide-and-Conquer-Method-for-D>