LAPORAN TUGAS KECIL 2
Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Disusun oleh:
1. 13522079 - Emery Fathan Zwageri
2. 13522152 - Muhammad Roihan

IF2211 - Strategi Algoritma

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

DAFTAR ISI

BAB 1
DESKRIPSI MASALAH

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 P0 sampai Pn , 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 P0, sedangkan titik kontrol terakhir adalah P3. Titik kontrol P1 dan P2 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 P0 dan P1 yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan kurva Bézier linier. Misalkan terdapat sebuah titik Q0 yang berada pada garis yang dibentuk oleh P0 dan P1, maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

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

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh B(t) dari P0 ke P1. Misalnya ketika t = 0.25, maka B(t) adalah seperempat jalan dari titik P0 ke P1. sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan B(t) membentuk sebuah garis lurus dari P0 ke P1. Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P2, dengan P0 dan P2 sebagai titik kontrol awal dan akhir, dan P1 menjadi titik kontrol antara. Dengan menyatakan titik Q1 terletak diantara garis yang menghubungkan P1 dan P2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q0 berada, maka dapat dinyatakan sebuah titik baru, R0 yang berada diantara garis yang menghubungkan Q0 dan Q1 yang bergerak membentuk kurva Bézier kuadratik terhadap titik P0 dan P2. Berikut adalah uraian persamaannya.

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

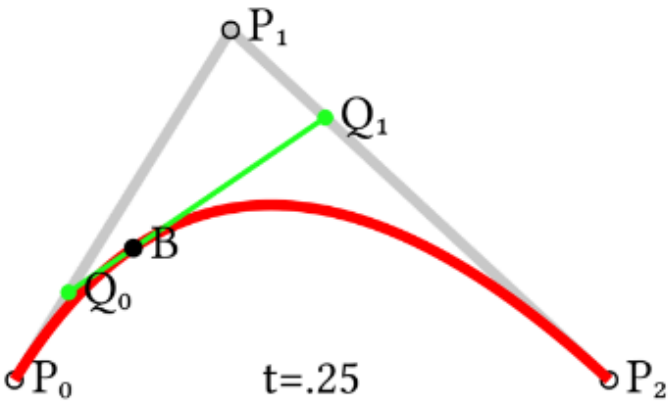
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

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

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

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

Berikut adalah ilustrasi dari kasus diatas.



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*.**

Untuk Tugas Kecil ini, implementasi kami menggunakan bahasa Python.

BAB 2
TEORI DASAR

2.1. Algoritma *Brute Force*

Algoritma Bruteforce dilakukan dengan mengiterasi setiap titik sesuai jumlah iterasi yang diinginkan. titik dicari dengan menggunakan persamaan parametrik di bab sebelumnya.

2.2. Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* adalah algoritma pemecahan masalah yang menggunakan strategi membagi sebuah permasalahan besar menjadi bagian-bagian permasalahan yang lebih kecil secara rekursif. Permasalahan yang lebih kecil tersebut kemudian dicari solusinya kemudian digabungkan dengan solusi dari bagian permasalahan kecil lainnya sehingga menjadi sebuah solusi akhir.

berikut merupakan ilustrasi kasus dengan algoritma divide and conquer

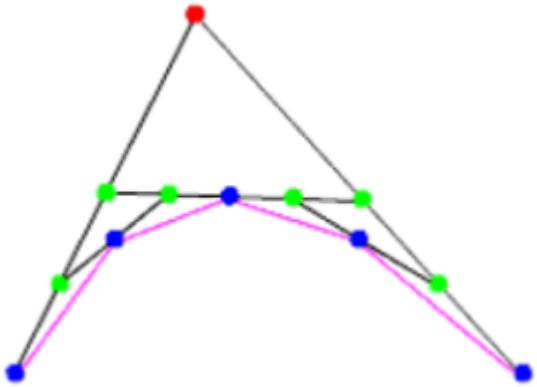
idenya cukup sederhana, relatif mirip dengan pembahasan sebelumnya, dan dilakukan secara iteratif. Misalkan terdapat tiga buah titik, P0, P1, dan P2, dengan titik P1 menjadi titik kontrol antara, maka:

a. Buatlah sebuah titik baru Q0 yang berada di tengah garis yang menghubungkan P0 dan P1, serta titik Q1 yang berada di tengah garis yang menghubungkan P1 dan P2.
b. Hubungkan Q0 dan Q1 sehingga terbentuk sebuah garis baru.
c. Buatlah sebuah titik baru R0 yang berada di tengah Q0 dan Q1.
d. Buatlah sebuah garis yang menghubungkan P0 - R0 - P2.

Melalui proses di atas, telah dilakukan 1 buah iterasi dan diperoleh sebuah “kurva” yang belum cukup mulus dengan aproksimasi 3 buah titik. Untuk membuat sebuah kurva yang lebih baik, perlu dilakukan iterasi lanjutan. Berikut adalah prosedurnya.

a. Buatlah beberapa titik baru, yaitu S0 yang berada di tengah P0 dan Q0, S1 yang berada di tengah Q0 dan R0, S2 yang berada di tengah R0 dan Q1, dan S3 yang berada di tengah Q1 dan P2.
b. Hubungkan S0 dengan S1 dan S2 dengan S3 sehingga terbentuk garis baru.
c. Buatlah dua buah titik baru, yaitu T0 yang berada di tengah S0 dan S1, serta T1 yang berada di tengah S2 dan S3.
d. Buatlah sebuah garis yang menghubungkan P0 - T0 - R0 - T1 - P2.

Melalui iterasi kedua akan tampak semakin mendekati sebuah kurva, dengan aproksimasi 5 buah titik. Anda dapat membuat visualisasi atau gambaran secara mandiri terkait hal ini sehingga dapat diamati dan diterka dengan jelas bahwa semakin banyak iterasi yang dilakukan, maka akan membentuk sebuah kurva yang tidak lain adalah kurva Bézier.



BAB 3
ANALISIS DAN IMPLEMENTASI PROGRAM

3.1. bruteforce.py

Analisis dan implementasi bruteforce sebagai berikut:

1. kami menggunakan rumus

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

2. jadi strategi dari implementasi bruteforce pada bezier curve adalah dengan mengiterasi setiap titik secara langsung menggunakan rumus yang ada jadi iterasi dilakukan sebanyak iterasi titik yang mau digunakan sebagai penggambaran kurva misal kita pengen 3 titik pada kurva maka masukan iterasi adalah 1,misal 5 titik maka 2 kali iterasi

3. setelah selesai mengiterasi semua titik yang mungkin maka kumpulan titik yang akan dibuat untuk menggambar curva sudah siap. lalu diplotting.

kompleksitas algoritma bruteforce pada kurva bezier adalah o(2^n) dimana n adalah jumlah iterasi

```python
import time
import numpy as np
from plotting import *

curve_points = []
control_points = []
new_points= []
iteration = 0


def bruteforce(p0,p1,p2,n):
    global curve_points,new_points
```

```
    if n>1:
        for i in range(n):
            x = (1 - i/(n-1))**2 * p0[0] + 2 * (1 - i/(n-1)) * i/(n-1) * p1[0] +
(i/(n-1))**2 * p2[0]
            y = (1 - i/(n-1))**2 * p0[1] + 2 * (1 - i/(n-1)) * i/(n-1) * p1[1] +
(i/(n-1))**2 * p2[1]
            new_points.append([x,y])
            curve_points.append([x,y])
    else:
        new_points.extend([p0,p1,p2])
        curve_points.extend([p0,p1,p2])

def main():
    global control_points,iteration
    # input titik kontrol
    print("Masukkan control point: ")
    p0 = tuple(map(float, input().split(" ")))
    p1 = tuple(map(float, input().split(" ")))
    p2 = tuple(map(float, input().split(" ")))

    # input jumlah iterasi
    iteration = int(input("Masukkan jumlah iterasi : "))

    temp = iteration
    for i in range( temp):
        if i==0 :
            iteration = 3
        else:
            iteration =iteration*2-1
    control_points.append(p0)
    control_points.append(p1)
    control_points.append(p2)

    start_time = time.time()
    bruteforce(p0,p1,p2,iteration)
    end_time = time.time()
    execution_time = end_time - start_time

    for i in new_points:
        print(i)
    print("Bezier Curve Points:")
    for point in curve_points:
```

```
        print(point)

    print("\nExecution Time:", execution_time, "seconds")
    plot_curve(control_points, new_points, curve_points)


if __name__ == "__main__":
    main()
```

3.2. divnqon.py

Ide dari algoritma divide and conquer adalah membagi problem utama menjadi sub-sub problem lalu menggabungkan solusi masing masing.
pada kurva bezier strategi divide and conquer dilakukan dengan cara membuat titik tengah di setiap 2 titik yang bertetangga lalu membuat lagi titik tengah di titik tersebut lalu titik titik tersebut dihubungkan dan yang paling kiri dihubungkan dengan titik awal dan yang paling kanan dengan titik akhir.
Implementasinya adalah sebagai berikut: (dilakukan secara rekursi) awalnya fungsi divnqon membawa titik awal titik kendali dan titik akhir lalu setelah titik tengah1, mid point,titik tengah2 dimasukan kedalam array of mid_point maka dipanggil fungsi divnqon yang memanggil titik awal utama sebagai titik awal titik tengah1 sebagai titik kendali lalu titik midpoint sebagai titik akhir. lalu midpoint dimasukan kedalam array curva.lalu dipanggil lagi untuk bagian kanan yaitu titik midpoint sebagai awal titik tengah2 sebagai titik kendali titik akhir sebagai titik akhir. dilakukan sampai iterasi kali.
Algoritma ini memiliki kompleksitas O(2^n)

```
import time

from plotting import *

curve_points = []
control_points = []
mid_points = []
iteration = 0


# Fungsi untuk mendapatkan titik kurva
def solve(p0, p1, p2):
    global curve_points
    curve_points.append(p0)
    divide_conquer(p0, p1, p2, 0)
    curve_points.append(p2)
```

```
# Fungsi divide and conquer
def divide_conquer(p0, p1, p2, iterationNow):
    global curve_points,mid_points
    if iterationNow < iteration:
        mid1 = [(p0[0] + p1[0]) / 2, (p0[1] + p1[1]) / 2]
        mid2 = [(p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2]
        mid = [(mid1[0] + mid2[0]) / 2, (mid1[1] + mid2[1]) / 2]
        mid_points.append(mid1)
        mid_points.append(mid2)
        mid_points.append(mid)

        iterationNow += 1
        # kiri
        divide_conquer(p0, mid1, mid, iterationNow)
        curve_points.append(mid)
        # kanan
        divide_conquer(mid, mid2, p2, iterationNow)

def main():
    global control_points,iteration
    # input titik kontrol
    print("Masukkan control point: ")
    p0 = tuple(map(float, input().split(" ")))
    p1 = tuple(map(float, input().split(" ")))
    p2 = tuple(map(float, input().split(" ")))

    # input jumlah iterasi
    iteration = int(input("Masukkan jumlah iterasi : "))
    control_points.append(p0)
    control_points.append(p1)
    control_points.append(p2)

    start_time = time.time()
    solve(p0, p1, p2)
    end_time = time.time()
    execution_time = end_time - start_time
    for i in mid_points:
        print(i)
    print("Bezier Curve Points:")
    for point in curve_points:
        print(point)
```

```
    print("\nExecution Time:", execution_time, "seconds")
    plot_curve(control_points, mid_points, curve_points)


if __name__ == "__main__":
    main()
```

3.3. dncGeneral.py(bonus)

Program ini adalah implementasi algoritma rekursif untuk menghasilkan kurva Bezier untuk n kontrol point dengan menggunakan pendekatan divide and conquer. Pada dasarnya program ini mirip dengan program divnqon.py. Program ini terinspirasi dari Algoritma De Casteljau dengan mengevaluasi kurva Bezier dan membaginya menjadi segmen-segmen yang lebih kecil.Berikut adalah penjelasan implementasi program.

Pertama, program menerima titik-titik kontrol sebagai input bersama dengan iterasi saat ini. Selanjutnya, jika iterasi saat ini kurang dari jumlah iterasi yang ditentukan, fungsi akan membagi kurva menjadi dua bagian. Proses ini dilakukan dengan membagi setiap segmen kurva menjadi dua segmen baru dengan menemukan titik tengah baru melalui interpolasi linier antara titik kontrol. Hasilnya adalah pembagian kurva menjadi segmen-segmen yang semakin halus. Iterasi dilanjutkan secara rekursif pada setiap segmen kurva yang dihasilkan hingga mencapai batas iterasi yang ditentukan.

```
import time
from plotting import *

curve_points = []
control_points = []
mid_points = []
iteration = 0
x_points = []
y_points = []


def solve(control_points):
    global curve_points
    curve_points.append(control_points[0])
    bezier_curve(control_points, 0)
    curve_points.append(control_points[len(control_points) - 1])
```

```
def bezier_curve(control_points, iterationNow):
    global curve_points
    if iterationNow < iteration:
        right = control_points.copy()
        left = control_points.copy()
        left = left[::-1]
        n = len(control_points) -1
        while(n > 0):
            for i in range (n):
                right[i] = [(right[i][0] + right[i+1][0]) / 2, (right[i][1] +
right[i+1][1]) / 2]
                left[i] = [(left[i][0] + left[i+1][0]) / 2, (left[i][1] +
left[i+1][1]) / 2]
                mid_points.append(right[i])
            n-=1

        iterationNow+=1
        left = left[::-1]
        bezier_curve(left, iterationNow)
        curve_points.append(right[0])
        bezier_curve(right, iterationNow)


def main():
    global iteration,n
    # Input titik kontrol
    print("Masukkan jumlah titik kontrol:")
    n = int(input())
    points = []
    for i in range(n):
        point = tuple(map(float, input(f"Masukkan titik ke-{i+1}: ").split(" ")))
        points.append(point)
        control_points.append(point)

    iteration = int(input("Masukkan jumlah iterasi : "))

    start_time = time.time()
    solve(points)
    end_time = time.time()
    execution_time = end_time - start_time
```

```
    print("Bezier Curve Points:")
    for point in curve_points:
        print(point)

    print("\nExecution Time:", execution_time, "seconds")
    plot_curve(control_points, mid_points, curve_points)


if __name__ == "__main__":
    main()
```

implementasi bonus dilakukan mirip dengan yang bukan bonus. yaitu pendekatan dilakukan secara rekursif. namun juga dengan iteratif digabung.

3.4. plotting.py

mungkin tidak perlu banyak penjelasan difile ini karena bukan bagian dari implementasi algoritma bruteforce maupun dnc(hanya untuk visualize saja).

plotting dilakukan secara bertahap.

```
import matplotlib.pyplot as plt

def plot_curve(control_points, mid_points, curve_points):
    x_points = []
    y_points = []
    for step_points in control_points:
        x_points.append(step_points[0])
        y_points.append(step_points[1])

        plt.title("Bezier Curve (Step by Step)")
        plt.xlabel("X-axis")
        plt.ylabel("Y-axis")
        plt.grid(True)
        plt.scatter(x_points, y_points, color="red")
        plt.pause(0.15)
        plt.plot(x_points, y_points, "-o", color="black",
markerfacecolor="red")
```

```
            plt.pause(0.15)
        plot_mid_point(mid_points)
        plot_curve_point(curve_points)
        plt.show()


def plot_mid_point(mid_points):
        x_points = []
        y_points = []
        for step_points in mid_points:
            x_points.append(step_points[0])
            y_points.append(step_points[1])

            plt.title("Quadratic Bezier Curve (Step by Step)")
            plt.xlabel("X-axis")
            plt.ylabel("Y-axis")
            plt.grid(True)
            plt.scatter(x_points, y_points, color="grey")
            plt.pause(0.15)
            plt.plot(x_points, y_points, "-o", color="silver",
markerfacecolor="grey")
            plt.pause(0.15)


def plot_curve_point(curve_points):
        x_points = []
        y_points = []
        for step_points in curve_points:
            x_points.append(step_points[0])
            y_points.append(step_points[1])

            plt.title("Quadratic Bezier Curve (Step by Step)")
            plt.xlabel("X-axis")
            plt.ylabel("Y-axis")
            plt.grid(True)
            plt.scatter(x_points, y_points, color="midnightblue")
            plt.pause(0.15)
            plt.plot(x_points, y_points, "-o", color="navy",
markerfacecolor="midnightblue")
            plt.pause(0.15)
```

BAB 4 EKSPERIMEN

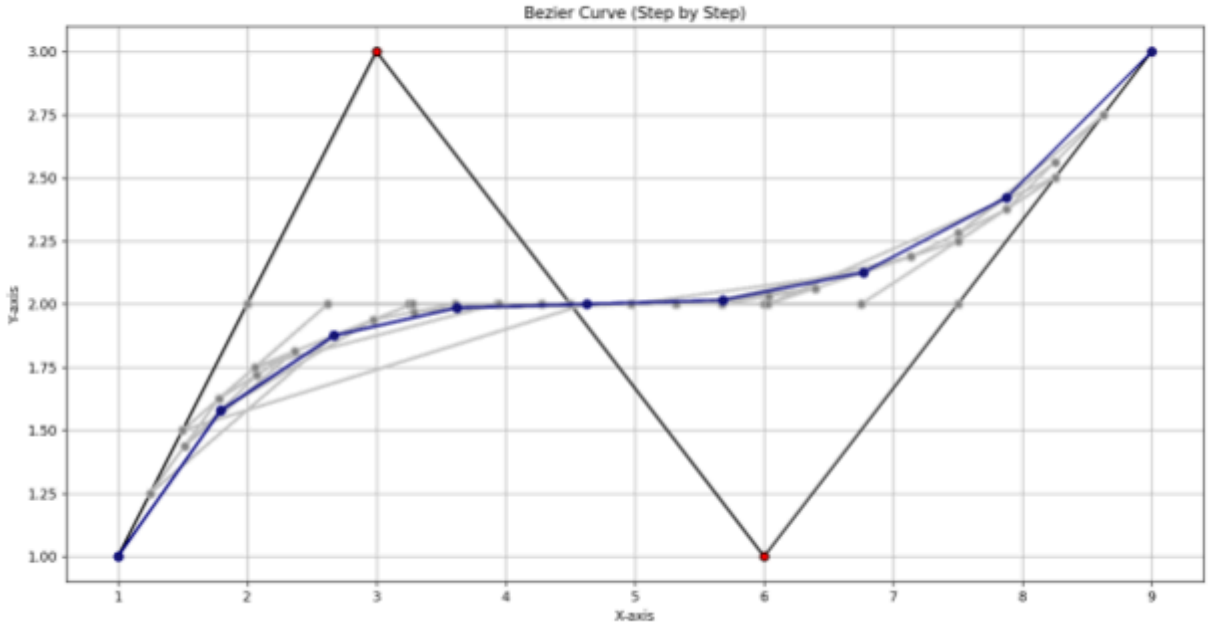
4.1. Test Case 1

src > tes1.txt

```
4
1 1
3 3
6 1
9 3
3
```

```
Masukkan jumlah titik kontrol:
4
Masukkan titik ke-1: 1 1
Masukkan titik ke-2: 3 3
Masukkan titik ke-3: 6 1
Masukkan titik ke-4: 9 3
Masukkan jumlah iterasi : 3
Bezier Curve Points:
(1.0, 1.0)
[1.794921875, 1.578125]
[2.671875, 1.875]
[3.619140625, 1.984375]
[4.625, 2.0]
[5.677734375, 2.015625]
[6.765625, 2.125]
[7.876953125, 2.421875]
(9.0, 3.0)

Execution Time: 0.0 seconds
```



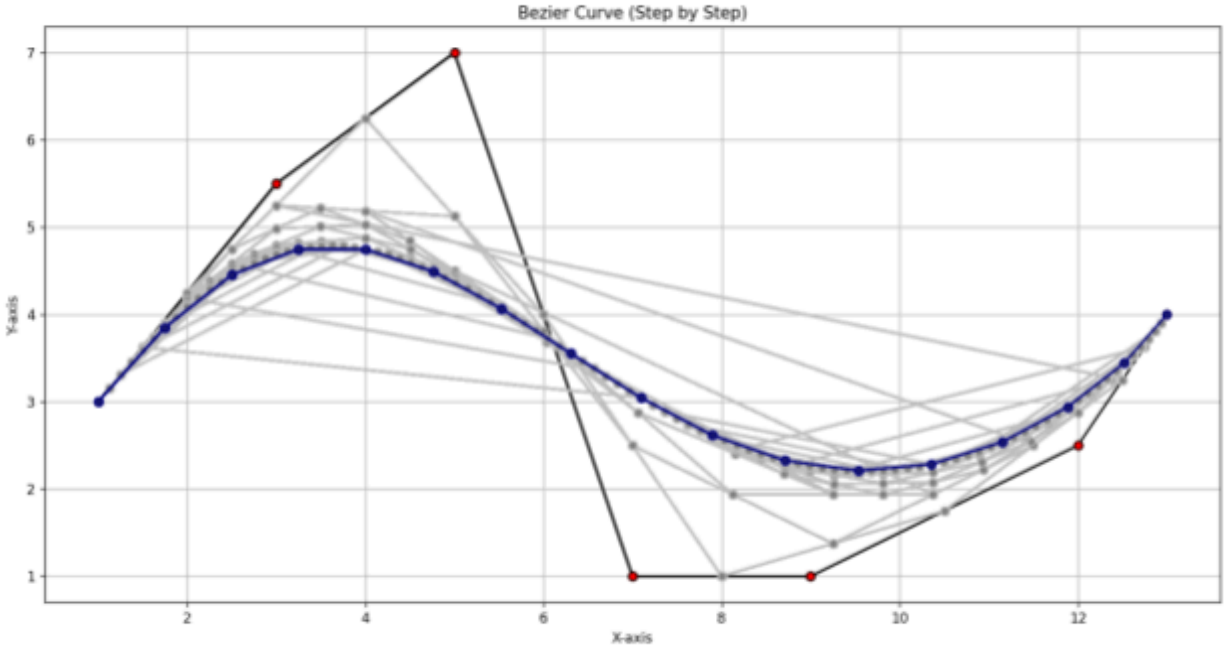
4.2. Test Case 2

src > tes2.txt

```
7
1 3
3 5.5
5 7
7 1
9 1
12 2.5
13 4
4
```

```
Masukkan jumlah titik kontrol:
7
Masukkan titik ke-1: 1 3
Masukkan titik ke-2: 3 5.5
Masukkan titik ke-3: 5 7
Masukkan titik ke-4: 7 1
Masukkan titik ke-5: 9 1
Masukkan titik ke-6: 12 2.5
Masukkan titik ke-7: 13 4
Masukkan jumlah iterasi : 4
Bezier Curve Points:
(1.0, 3.0)
[1.7500053644180298, 3.851531684398651]
[2.5001602172851562, 4.453227996826172]
[3.2511297464370728, 4.748737514019012 5]
[4.00439453125, 4.744873046875]
[4.762293457984924, 4.49184912443161]
[5.527809143066406, 4.066051483154297]
[6.3040958642959595, 3.5553385615348816]
[7.09375, 3.046875]
[7.897822976112366, 2.617497146129608]
[8.714576721191406, 2.326610565185547]
[9.537981629371643, 2.211619555950165]
[10.35595703125, 2.285888671875]
[11.148354172706604, 2.5392362475395203]
[11.884681701660156, 2.940959930419922]
[12.521573662757874, 3.445394217967987]
(13.0, 4.0)

Execution Time: 0.0010037422180175781 seconds
```



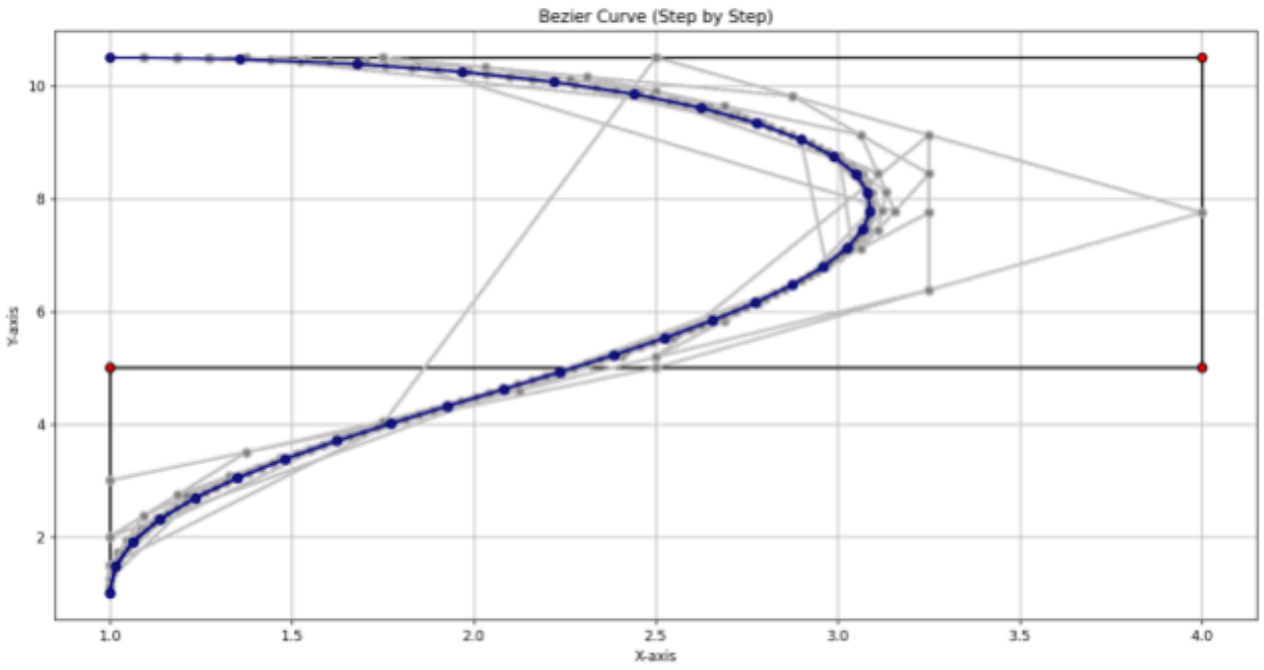
4.3. Test Case 3

src > tes3.txt

```
5
1 1
1 5
4 5
4 10.5
1 10.5
5
```

```
Masukkan jumlah titik kontrol:
5
Masukkan titik ke-1: 1 1
Masukkan titik ke-2: 1 5
Masukkan titik ke-3: 4 5
Masukkan titik ke-4: 4 10.5
Masukkan titik ke-5: 1 10.5
Masukkan jumlah iterasi : 5
Bezier Curve Points:
(1.0, 1.0)
[1.0168514251708984, 1.4777026176452637]
[1.064544677734375, 1.9152145385742188]
[1.138891220092734, 2.3187899589538574]
[1.23583984375, 2.694213867187 5]
[1.3514766693115234, 3.046802043914795]
[1.482025146484375, 3.3814010620117188]
[1.6238460540771484, 3.702388286590576]
[1.7734375, 4.013671875]
[1.9274349212646484, 4.318690776824951]
[2.082611083984375, 4.620414733886719]
[2.2358760833740234, 4.92134428024292]
[2.38427734375, 5.2235107421875]
[2.5249996185302734, 5.528476238250732]
[2.655364990234375, 5.837333679199219]
[2.7728328704833984, 6.150706768035889]
[2.875, 6.46875]
[2.9596004486083984, 6.791148662567139]
[3.024505615234375, 7.117118835449219]
[3.067724227905273 4, 7.445407390594482]
[3.08740234375, 7.774291992187 5]
[3.0818233489990234, 8.10158109664917]
[3.049407958984375, 8.424613952636719]
[2.988714218139648 4, 8.748260601043701]
[2.8984375, 9.04492187 5]
[2.7774105072021484, 9.334529399871826]
[2.624603271484375, 9.604545593261719]
[2.439123153686523 4, 9.849963665008545]
[2.22021484375, 10.0653076171875]
[1.967260360717773 4, 10.244632244110107]
[1.679779052734375, 10.381523132324219]
[1.3574275970458984, 10.469096660614014]
(1.0, 10.5)

Execution Time: 0.0 seconds
```



4.4. Test Case 4

test > test4.txt

```
3
1.5 1.5
3.5 3.5
5.5 1.5
7
```

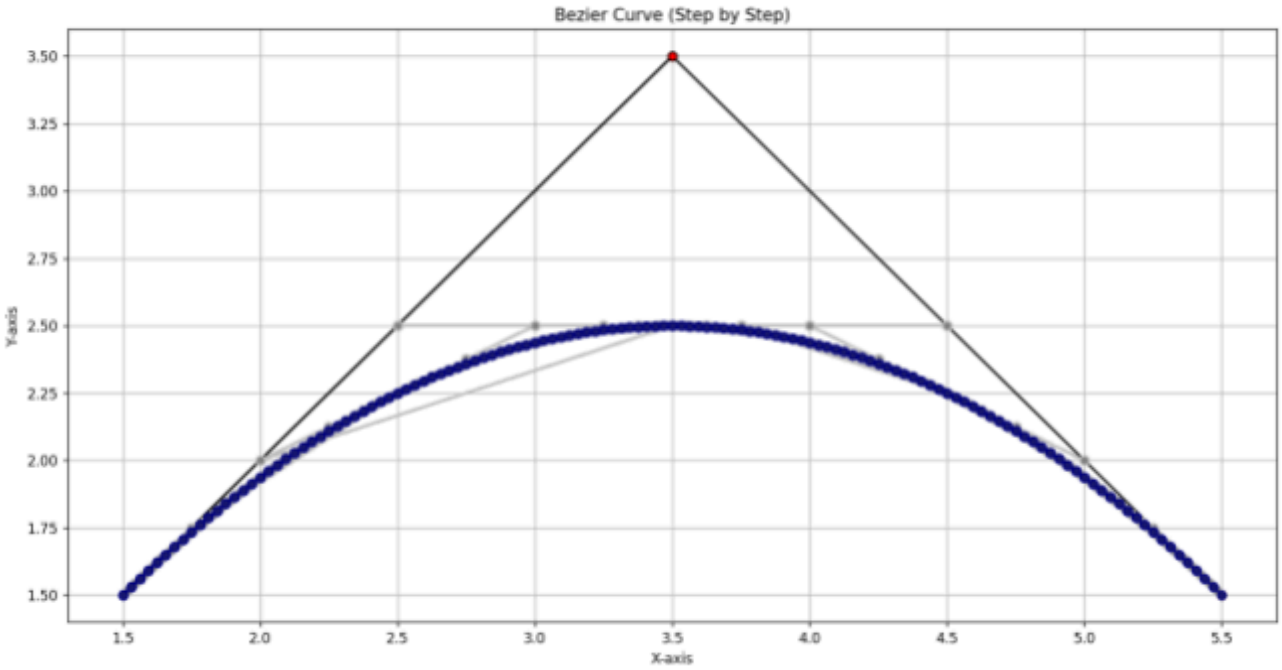
```
Masukkan jumlah titik kontrol:
3
Masukkan titik ke-1: 1.5 1.5
Masukkan titik ke-2: 3.5 3.5
Masukkan titik ke-3: 5.5 1.5
Masukkan jumlah iterasi : 7
Bezier Curve Points:
(1.5, 1.5)
[1.53125, 1.53100585937 5]
[1.5625, 1.561523437 5]
[1.59375, 1.591552734375]
[1.625, 1.62109375]
[1.65625, 1.650146484375]
[1.6875, 1.678710937 5]
[1.71875, 1.706787109375]
[1.75, 1.734375]
[1.78125, 1.761474609375]
[1.8125, 1.7880859375]
[1.84375, 1.814208984375]
[1.875, 1.83984375]
[1.90625, 1.864990234375]
[1.9375, 1.8896484375]
[1.96875, 1.913818359375]
[2.0, 1.9375]
[2.03125, 1.960693359375]
[2.0625, 1.983398437 5]
[2.09375, 2.005615234375]
[2.125, 2.02734375]
[2.15625, 2.048583984375]
[2.1875, 2.069335937 5]
[2.21875, 2.089599609375]
[2.25, 2.109375]
[2.28125, 2.128662109375]
[2.3125, 2.147460937 5]
[2.34375, 2.165771484375]
[2.375, 2.18359375]
[2.40625, 2.200927734375]
[2.4375, 2.217773437 5]
[2.46875, 2.234130859375]
[2.5, 2.25]
```

```
[2.5, 2.25]
[2.53125, 2.265380859375]
[2.5625, 2.280273437 5]
[2.59375, 2.294677734375]
[2.625, 2.30859375]
[2.65625, 2.322021484375]
[2.6875, 2.334960937 5]
[2.71875, 2.347412109375]
[2.75, 2.359375]
[2.78125, 2.370849609375]
[2.8125, 2.381835937 5]
[2.84375, 2.392333984375]
[2.875, 2.40234375]
[2.90625, 2.411865234375]
[2.9375, 2.420898437 5]
[2.96875, 2.429443359375]
[3.0, 2.4375]
[3.03125, 2.445068359375]
[3.0625, 2.452148437 5]
[3.09375, 2.458740234375]
[3.125, 2.46484375]
[3.15625, 2.470458984375]
[3.1875, 2.475585937 5]
[3.21875, 2.480224609375]
[3.25, 2.484375]
[3.28125, 2.488037109375]
[3.3125, 2.491210937 5]
[3.34375, 2.493896484375]
[3.375, 2.49609375]
[3.40625, 2.497802734375]
[3.4375, 2.499023437 5]
[3.46875, 2.499755859375]
[3.5, 2.5]
[3.53125, 2.499755859375]
[3.5625, 2.499023437 5]
[3.59375, 2.497802734375]
[3.625, 2.49609375]
[3.65625, 2.493896484375]
[3.6875, 2.491210937 5]
```

```
[3.6875, 2.491210937 5]
[3.71875, 2.488037109375]
[3.75, 2.484375]
[3.78125, 2.480224609375]
[3.8125, 2.475585937 5]
[3.84375, 2.470458984375]
[3.875, 2.46484375]
[3.90625, 2.458740234375]
[3.9375, 2.452148437 5]
[3.96875, 2.445068359375]
[4.0, 2.4375]
[4.03125, 2.429443359375]
[4.0625, 2.420898437 5]
[4.09375, 2.411865234375]
[4.125, 2.40234375]
[4.15625, 2.392333984375]
[4.1875, 2.381835937 5]
[4.21875, 2.370849609375]
[4.25, 2.359375]
[4.28125, 2.347412109375]
[4.3125, 2.334960937 5]
[4.34375, 2.322021484375]
[4.375, 2.30859375]
[4.40625, 2.294677734375]
[4.4375, 2.280273437 5]
[4.46875, 2.265380859375]
[4.5, 2.25]
[4.53125, 2.234130859375]
[4.5625, 2.217773437 5]
[4.59375, 2.200927734375]
[4.625, 2.1835937 5]
[4.65625, 2.165771484375]
[4.6875, 2.147460937 5]
[4.71875, 2.128662109375]
[4.75, 2.109375]
[4.78125, 2.089599609375]
[4.8125, 2.069335937 5]
[4.84375, 2.048583984375]
[4.875, 2.02734375]
```

```
[4.875, 2.02734375]
[4.90625, 2.005615234375]
[4.9375, 1.983398437 5]
[4.96875, 1.960693359375]
[5.0, 1.9375]
[5.03125, 1.913818359375]
[5.0625, 1.8896484375]
[5.09375, 1.864990234375]
[5.125, 1.83984375]
[5.15625, 1.814208984375]
[5.1875, 1.7880859375]
[5.21875, 1.761474609375]
[5.25, 1.734375]
[5.28125, 1.706787109375]
[5.3125, 1.678710937 5]
[5.34375, 1.650146484375]
[5.375, 1.62109375]
[5.40625, 1.591552734375]
[5.4375, 1.561523437 5]
[5.46875, 1.531005859375]
(5.5, 1.5)

Execution Time: 0.0009870529174804688 seconds
```

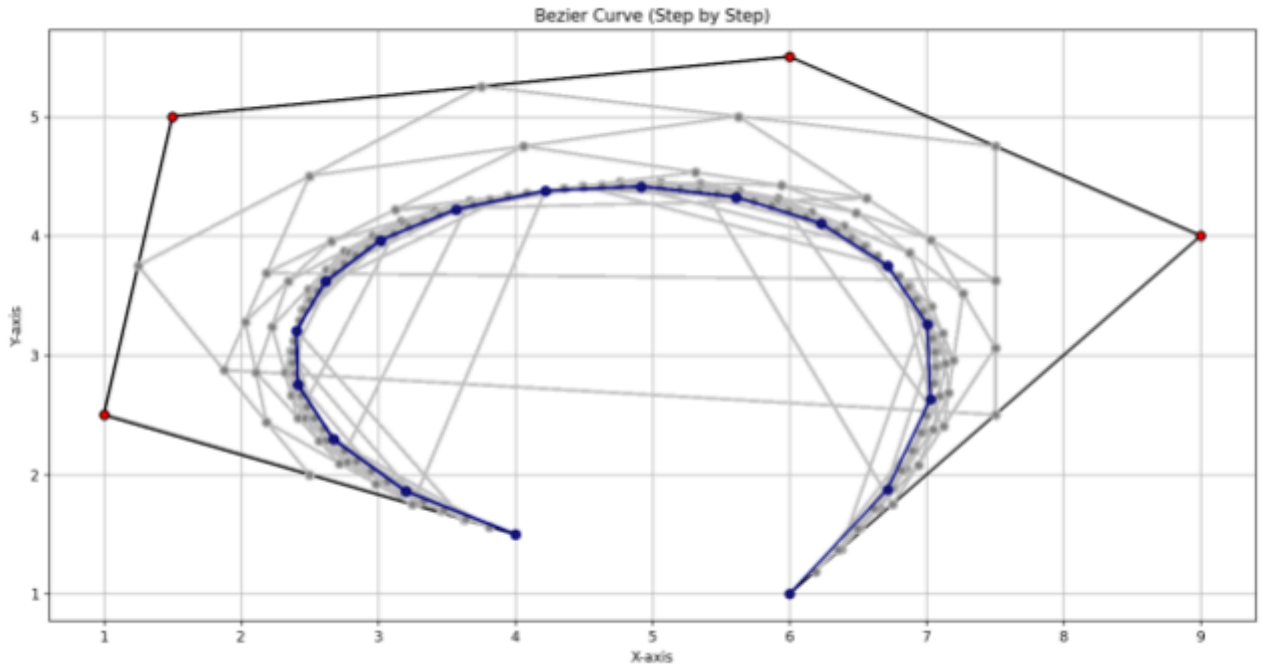


4.5. Test Case 5

test > tes5.txt

```
6
4 1.5
1 2.5
1.5 5
6 5.5
9 4
6 1
4
```

```
Masukkan jumlah titik kontrol:
6
Masukkan titik ke-1: 4 1.5
Masukkan titik ke-2: 1 2.5
Masukkan titik ke-3: 1.5 5
Masukkan titik ke-4: 6 5.5
Masukkan titik ke-5: 9 4
Masukkan titik ke-6: 6 1
Masukkan jumlah iterasi : 4
Bezier Curve Points:
(4.0, 1.5)
[3.19988836517334, 1.8628129959106445]
[2.674530029296875, 2.295196533203125]
[2.415471076965332, 2.7550649642944336]
[2.4052734375, 3.2060546875]
[2.6178159713745117, 3.617180824279785]
[3.019195556640625, 3.962493896484375]
[3.568528175354004, 4.220736503601074]
[4.21875, 4.375]
[4.917418479919434, 4.412381172180176]
[5.607513427734375, 4.323638916015625]
[6.228238105773926, 4.102850914001465]
[6.7158203125, 3.7470703125]
[7.0043134689331055, 3.2559823989868164]
[7.026397705078125, 2.631561279296875]
[6.714180946350098, 1.8777265548706055]
(6.0, 1.0)
```


Bezier Curve (Step by Step)
Y-axis
X-axis
1
2
3
4
5
6
7
8
9

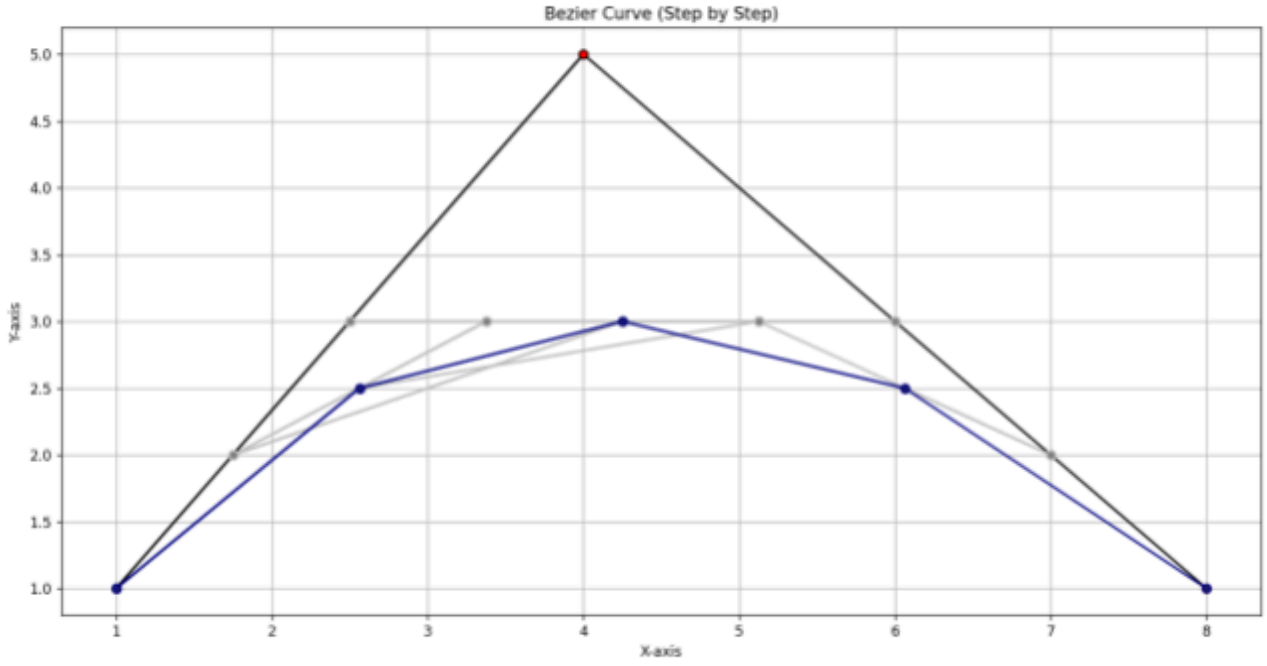
4.6. Test Case 6

test > tes6.txt

```
3
1 1
4 5
8 1
2
```

```
Masukkan jumlah titik kontrol:
3
Masukkan titik ke-1: 1 1
Masukkan titik ke-2: 4 5
Masukkan titik ke-3: 8 1
Masukkan jumlah iterasi : 2
Bezier Curve Points:
(1.0, 1.0)
[2.5625, 2.5]
[4.25, 3.0]
[6.0625, 2.5]
(8.0, 1.0)

Execution Time: 0.0 seconds
```



BAB 5
PENUTUP

5.1. Kesimpulan

Dalam menggambar kurva bezier dapat dilakukan dengan bruteforce dan divide and conquer. namun pada implementasinya bruteforce didapati lebih cepat dari dnc karena bruteforce memerlukan lebih sedikit iterasi.

5.2. Saran

masih banyak soal yang lebih efektif menggunakan dnc daripada bruteforce. untuk tugas kali ini bruteforce lebih cepat dari dnc wkwk.

5.3. Komentar dan Refleksi

Kami berterima kasih kepada tuhan yang maha esa sudah memberikan kami kekuatan dan ketabahan untuk mengerjakan tugas ini

5.4. Tabel *Checkpoint*

Poin	***Ya***	***Tidak***
1.Program berhasil dijalankan	✓	
2.Program dapat memvisualisasikan kurva bezier	✓	
3.Solusi yang diberikan program optimal	✓	
4. program dapat membuat kurva untuk n titik kontrol	✓	
5. program dapat melakukan visualisasi pembuatan kurva	✓	

DAFTAR PUSTAKA

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Tucil2-Stima-2024.pdf
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf

LAMPIRAN

LINK REPOSITORY

Link repository GitHub : https://github.com/mroihn/Tucil2_13522079_13522152

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13522079	Emery Fathan Zwageri	Bruteforce,laporan
13522152	Muhammad Roihan	bonus, DNC,laporan