

Laporan Tugas Kecil 2 IF2211 Strategi Algoritma
Mencari Pasangan Titik Terdekat 3D
Menggunakan Algoritma Divide and Conquer



Oleh:

Althaaf Khasyi Atisomya 13521130 K-02
Cetta Reswara Parahita 13521133 K-01

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021

A. Spesifikasi Tugas

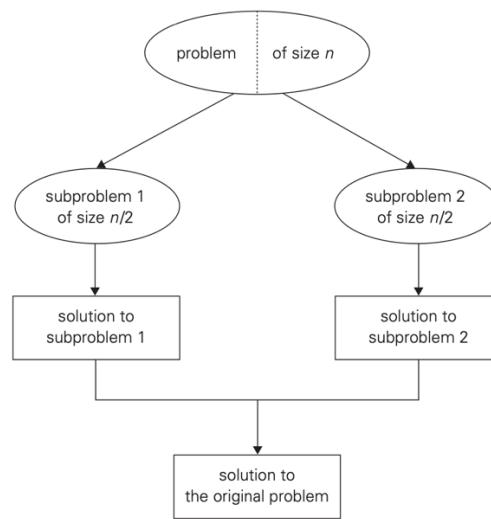
Pada tugas ini, diharapkan dapat dihasilkan luaran program sederhana yang dapat mencari sepasang titik terdekat dari sekumpulan n titik pada bidang 3 dimensi. Penyelesaian masalah ini dapat dilakukan dengan melakukan eksplorasi algoritma menggunakan algoritma *divide and conquer*. Untuk memastikan efektivitas penggunaan algoritma ini, hasil penyelesaian selanjutnya dibandingkan dengan penyelesaian yang dilakukan menggunakan algoritma *brute force*. Perbandingan ini dilakukan dengan memperhatikan parameter waktu, ketepatan penemuan titik terdekat, dan jumlah penggunaan perhitungan *Euclidean distance* menggunakan rumus:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Selain pada bidang 3 dimensi, penyelesaian masalah juga diharapkan dapat mencakup titik-titik pada bidang multidimensi. Hasil dari titik-titik itu juga perlu diproyeksikan dalam bentuk grafik sehingga memudahkan penentuan kebenaran dari algoritma perhitungan yang dilakukan.

B. Dasar Teori

Berbeda dengan penyelesaian permasalahan menggunakan algoritma *brute force* yang melakukan pendekatan secara *straightforward* sehingga memerlukan sumberdaya yang lebih berat apabila masukan unit permasalahan lebih besar, algoritma *divide and conquer* menjadi salah satu teknik penyelesaian masalah yang paling dikenal lebih efisien dalam mengimplementasikan kalkulasi.



Gambar 2.1
Diagram Penyelesaian *Divide and Conquer*

Secara umum penyelesaian *divide and conquer* melewati tiga tahap sebagai berikut:

- 1) Sebuah persoalan dibagi menjadi beberapa subpersoalan, idealnya tiap subpersoalan memiliki ukuran yang seimbang.
- 2) Subpersoalan tersebut diselesaikan (biasanya secara rekursif, kadang algoritma yang berbeda juga diterapkan pada subpersoalan yang sangat kecil dan spesifik)
- 3) Bila diperlukan, solusi dari tiap subpersoalan itu digabungkan untuk mendapatkan solusi dari persoalan utama

Secara umum penyelesaian menggunakan *divide and conquer* dengan ukuran n secara rekursif akan mengikuti pola kebutuhan waktu sebagai berikut:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Dengan $f(n)$ merupakan fungsi yang menghitung keperluan waktu dalam membagi permasalahan menjadi b bagian. Secara general, kebutuhan waktu menggunakan *divide and conquer* dapat disederhanakan menggunakan *master theorem* untuk menghasilkan notasi *big-O* dan *big-Omega* sebagai berikut:

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d, \\ \Theta(n^d \log n) & \text{if } a = b^d, \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

C. Penyelesaian Persoalan dengan Algoritma Divide and Conquer

Algoritma Divide and Conquer dilakukan dengan membagi persoalan menjadi beberapa sub-persoalan yang lebih kecil yang kemudian tiap-tiap persoalan akan diselesaikan secara langsung jika telah berukuran sangat kecil atau secara rekursif jika masih berukuran besar. Solusi dari masing-masing sub-persoalan kemudian akan digabungkan untuk membentuk solusi persoalan semula.

Pada program mencari pasangan titik terdekat ini, Langkah-langkah yang dilakukan adalah:

1. Program akan meminta user untuk menginput banyak titik dan dimensi dari titik tersebut. Program kemudian akan mengenerate secara random titik dengan jumlah dan dimensi sesuai input user. Titik-titik tersebut kemudian diurutkan berdasarkan axis x yang terurut membesar.
2. Program akan mencari titik terdekat dengan pendekatan brute force sebagai pembanding dengan algoritma divide and conquer.
3. Algoritma divide and conquer dilakukan dengan memeriksa apakah jumlah kurang dari sama dengan tiga. Apabila jumlah titik ≤ 3 maka akan digunakan algoritma brute force untuk mencari titik terdekat.
4. Apabila jumlah titik lebih dari tiga program akan membagi himpunan titik menjadi dua sisi. Kemudian secara rekursif akan memanggil algoritma divide and conquer untuk mencari pasangan titik terdekat tiap sisi.

5. Pada proses penggabungan solusi terdapat tiga kemungkinan solusi sepasang titik terdekat yaitu sepasang titik terdekat terdapat pada bagian satu, sepasang titik terdekat terdapat pada bagian dua, atau sepasang titik terdekat tepisah pada bagian satu dan dua. Pertama-tama sepasang titik terdekat pada bagian satu dan sepasang titik terdekat pada bagian dua dibandingkan mana yang paling dekat dan disimpan sebagai titik terdekat sementara.
6. Untuk menangani kasus apabila sepasang titik terpisah pada bagian satu dan dua akan dilakukan pencarian titik yang memiliki jarak tiap axis lebih kecil dari jarak terdekat sementara.
7. Titik-titik tersebut kemudian akan dibandingkan jaraknya apakah lebih kecil dari jarak terdekat sementara dan di dapatkanlah sepasang titik terdekat.

D. Source Code Program

Persoalan ini diselesaikan dengan menggunakan bahasa pemrograman *python* menggunakan *library-library* yang tersedia dalam manajemen paket *anaconda* yang antara lain mencakup *library random, math, time, numpy, pandas, dan plotly*. *Source code* lengkap program dapat dilihat pada laman berikut: https://github.com/althaafka/Tucil2_13521130_13521133.git

Tiap *library* yang digunakan dalam penyelesaian persoalan ini memiliki deskripsi penggunaan sebagai berikut:

Nama Library	Deskripsi Penggunaan
random	Library random digunakan dalam pembuatan n buah titik riil secara acak dengan memanfaatkan metode <i>random()</i> yang akan menghasilkan sebuah bilangan riil sembaang dari range $(0,1)$ kemudian akan dikalikan dengan faktor <i>maxNumber</i> yang telah ditetapkan sebagai konstanta sebelumnya.
math	Library math digunakan untuk melakukan perhitungan <i>Euclidean</i> yang memerlukan fungsi-fungsi dasar pada math.
time	Library time digunakan untuk melakukan tracking waktu penyelesaian pada masing-masing algoritma sehingga dapat dibandingkan algoritma yang memiliki waktu penyelesaian relatif lebih cepat.
numpy	Library numpy digunakan untuk pengolahan data dalam bentuk array sehingga mudah dialokasikan menjadi tipe dataframe sebelum divisualisasi.
pandas	Library pandas digunakan untuk menyimpan data menjadi berbentuk dataframe sehingga dapat dimanupulasi dan divisualisasikan dengan optimal.
plotly	Libarary plotly digunakan untuk melakukan visualisasi dengan metode <i>scatter plot</i> baik berdimensi 2, 3, atau lebih dari itu.

Selanjutnya pada *source code*, algoritma dibagi menjadi tiga file python sehingga penggunaan library dapat disesuaikan berdasarkan kebutuhan proses yang dilakukan pada tiap file. File-file tersebut terdiri dari file *closestPair.py*, file *visualisasi.py*, dan file *main.py*

Pada *closestPair.py* dilakukan komputasi dasar memanfaatkan *library* dengan header sebagai berikut:

```
import random
import math
import time

maxNum = 1000
euclidianCounter = 0
```

Selain itu, untuk mencari pasangan terdekat dibuat beberapa fungsi, antara lain:

- a) `createPoints(n, dimention)`

Fungsi untuk membuat *n* point riil random dengan dimensi *dimention* dan mengembalikan list *points*

```
def createPoints(n, dimention):
    points = [[random.random()*maxNum for i in range(dimention)] for i in range(n)]
    sortPointsbyX(points)
    print(points)
    return points
```

- b) `displayPoints(points)`

Fungsi untuk mencetak point sebagai bagian dari testing

```
def displayPoints(points):
    print(points)
```

- c) `euclidean(point1, point2)`

Fungsi untuk menghitung jarak *Euclidean* dari dua titik

```
def euclidean(point1,point2):
    dimention = len(point1)
    sum=0;
    for i in range(dimention):
        sum+= (point1[i]-point2[i])**2
    return math.sqrt(sum)
```

- d) `closestPointBrute(points)`

Fungsi untuk mengembalikan titik-titik terdekat dengan algoritma *bruteforce*

```
def closestPointBrute(points):
    if len(points)==2 :
        return points;
    else:
        global euclidianCounter
        min = 999999999999;
        minPair = [points[0],points[0]];
        for i in range(len(points)):
            for j in range(i+1,len(points)):
                euclidianCounter+=1
                if (euclidean(points[i],points[j])<min):
                    min = euclidean(points[i],points[j])
                    minPair = [points[i],points[j]]
        return minPair
```

- e) `sortPointsbyX(points)`

Fungsi untuk mengurutkan titik-titik berdasarkan posisi X

```
def sortPointsbyX(points):
    points.sort(key=lambda x: x[0])
```

f) getPointsinStrip(points, closestDistance)

Fungsi untuk mendapatkan titik-titik pada lokasi strip

```
def getPointsinStrip(points, closestDistance):
    strip = []
    mid = points[len(points)//2]
    for i in range(len(points)):
        if (abs(points[i][0]-mid[0])<closestDistance):
            strip.append(points[i])
    return strip
```

g) isClosestPairCandidate(point1, point2, closestDistance)

Fungsi untuk melakukan pengecekan apakah dua titik merupakan *closest pair*

```
def isClosestPairCandidate(points1,points2, closestDistance):
    for i in range(len(points1)):
        if (abs(points1[i]-points2[i])>=closestDistance):
            return False
    return True
```

h) closestPointsDivideenConquer(points)

Fungsi untuk mengembalikan titik-titik terdekat dengan algoritma *divide and conquer*

```
def closestPointsDividenConquer(points):
    if (len(points)<=3):
        return closestPointBrute(points)
    else:
        mid = len(points)//2
        points1 = points[:mid]
        points2 = points[mid:]
        closestPair1 = closestPointsDividenConquer(points1)
        closestPair2 = closestPointsDividenConquer(points2)

        if (euclidian(closestPair1[0],closestPair1[1])<=euclidian(closestPair2[0],closestPair2[1])):
            closestPair = closestPair1
        else:
            closestPair = closestPair2

        closestDistance = euclidian(closestPair[0],closestPair[1])

        strip = getPointsinStrip(points,closestDistance)

        global euclidianCounter
        for i in range(len(strip)):
            for j in range(i+1, len(strip)):
                if (isClosestPairCandidate(strip[i],strip[j],closestDistance)):
                    euclidianCounter+=1
                    if (euclidian(strip[i],strip[j])<closestDistance):
                        closestPair = [strip[i],strip[j]]
                        closestDistance = euclidian(strip[i],strip[j])

    return closestPair
```

Pada *visualisasi.py* dilakukan manipulasi tipe data dan penggambaran dataframe menjadi sebuah grafik *scatterplot* sesuai dengan dimensi yang ada. Hanya terdapat satu fungsi utama yakni *makegraph(points, closestPair)* yang digunakan untuk mengubah tipe data list points menjadi dataframe, kemudian melakukan pengecekan dan memberikan attribute *True* pada points yang menjadi kandidat *closestPair*, dan terakhir membentuk sebuah grafik 2, 3, atau multidimensi sesuai dengan dimensi titik yang sedang divisualisasi.

```

from closestPair import *
import numpy as np
import pandas as pd
import plotly
import plotly.express as px

def makegraph(points, closestPair) :
    # Mengubah list point dan closestPair menjadi array
    npoints = np.array(points)
    nclosest = np.array(closestPair)

    # Untuk kasus 3D
    if len(nclosest[0]) == 3 :
        # Mengubah array menjadi dataframe
        df = pd.DataFrame({'x':npoints[:,0],
                            'y':npoints[:,1],
                            'z':npoints[:,2]})

        newdf = pd.DataFrame({'x':nclosest[:,0],
                            'y':nclosest[:,1],
                            'z':nclosest[:,2]})

        # Memberi attribut Closest TRUE pada point yang menjadi closestPair
        d = pd.merge(df, newdf, on=['x','y','z'], how='left', indicator='Closest')
        d['Closest'] = np.where(d.Closest == 'both', True, False)

        # Visualisasi data
        fig3d = px.scatter_3d(d, x='x', y='y', z='z', color='Closest')
        fig3d.show()

    # Untuk kasus 2D
    elif len(nclosest[0]) == 2 :
        # Mengubah array menjadi dataframe
        df = pd.DataFrame({'x':npoints[:,0],
                            'y':npoints[:,1]})

        newdf = pd.DataFrame({'x':nclosest[:,0],
                            'y':nclosest[:,1]})

        # Memberi attribut Closest TRUE pada point yang menjadi closestPair
        d = pd.merge(df, newdf, on=['x','y'], how='left', indicator='Closest')
        d['Closest'] = np.where(d.Closest == 'both', True, False)

        # Visualisasi data
        fig2d = px.scatter(d, x='x', y='y', color='Closest')
        fig2d.show()

    # Untuk kasus N-Dimensi
    else :
        # Mengubah array menjadi dataframe
        trydf = pd.DataFrame()
        for i in range (len(nclosest[0])) :
            trydf[i] = npoints[:,i]

        trynewdf = pd.DataFrame()
        for i in range (len(nclosest[0])) :
            trynewdf[i] = nclosest[:,i]

        # Memberi attribut Closest TRUE pada point yang menjadi closestPair
        tryattribut = list(range(len(nclosest[0])))

        tryd = pd.merge(trydf, trynewdf, on=tryattribut, how='left', indicator='Closest')
        tryd['Closest'] = np.where(tryd.Closest == 'both', True, False)

        # Visualisasi data
        figmatrix = px.scatter_matrix(tryd, color='Closest')
        figmatrix.show()

if __name__ == "__main__":
    points = createPoints(800,2)
    closestPair = closestPointsDividenConquer(points)
    makegraph(points,closestPair)

```

E. Testing

a) Dimensi 3

- $n = 16$

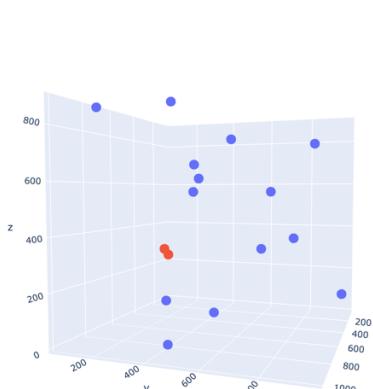
CLOSEST POINT GENERATOR

BY ALTHA CETTA

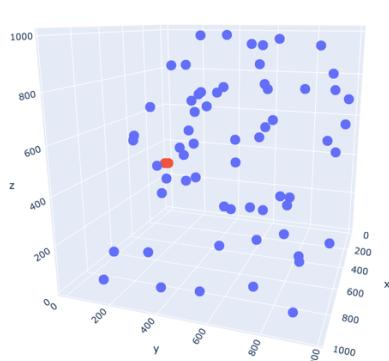
```
Masukkan banyak titik = 16
Masukkan dimensi yang diinginkan = 3
```

```
Closest pair by Brute Force = [[140.63, 57.42, 267.65], [193.85, 104.88, 245.97]]
Shortest distance          = 74.53081510355297
Brute time                  = 0.34499168395996094
Euclidean used             = 120
```

```
Closest pair by Divide and Conquer = [[140.63, 57.42, 267.65], [193.85, 104.88, 245.97]]
Shortest distance                 = 74.53081510355297
Divide and Conquer time          = 0.3230571746826172
Euclidian used                  = 12
```



$$- \quad n = 64$$



- n = 128

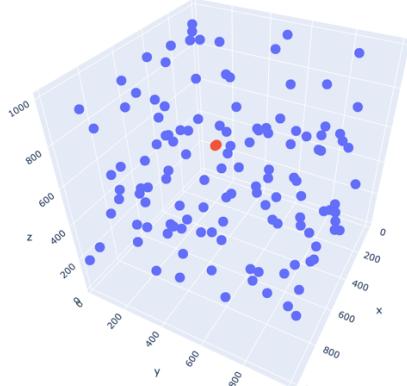
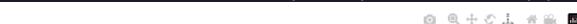
CLOSEST POINT GENERATOR

BY ALTHA CETTA

Masukkan banyak titik = 128
Masukkan dimensi yang diinginkan = 3

```
Closest pair by Brute Force = [[492.13, 386.01, 606.66], [494.46, 395.33, 617.2]]  
Shortest distance          = 14.261237674199299  
Brute time                 = 18.466949462899625  
Euclidean used             = 8128
```

```
Closest pair by Divide and Conquer = [[492.13, 386.01, 606.66], [494.46, 395.33, 617.2]]
Shortest distance                = 14.261237674199299
Divide and Conquer time          = 3.9348602294921875
Euclidian used                  = 109
```



- **n = 1000**

CLOSEST POINT GENERATOR

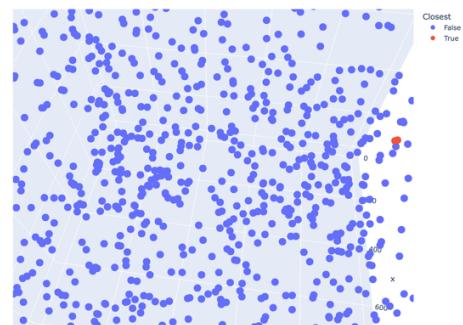
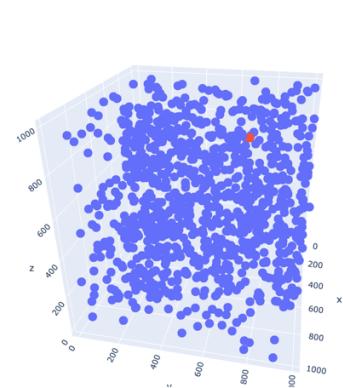
BY ALTHA CETTA

Masukkan banyak titik = 1000
Masukkan dimensi yang diinginkan = 3

[[714.83, 732.5, 909.29], [715.52, 734.24, 911.71]]
3.059428051123331
649.399995803833
499500

[[714.83, 732.5, 909.29], [715.52, 734.24, 911.71]]
3.059428051123331
35.63189506530762
907

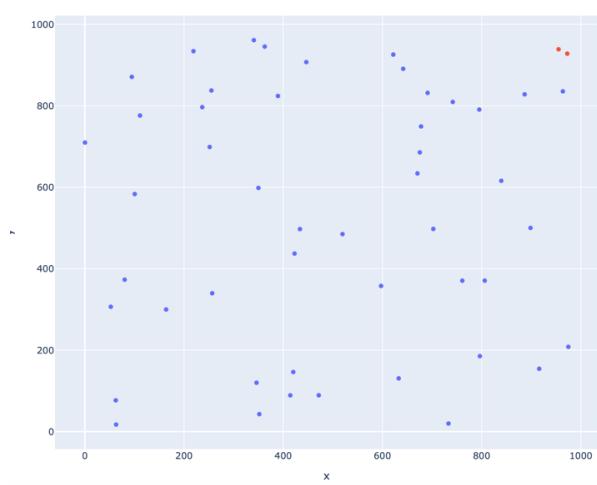
[[714.83, 732.5, 909.29], [715.52, 734.24, 911.71]]
3.059428051123331
35.63189506530762
907



b) Dimensi Banyak

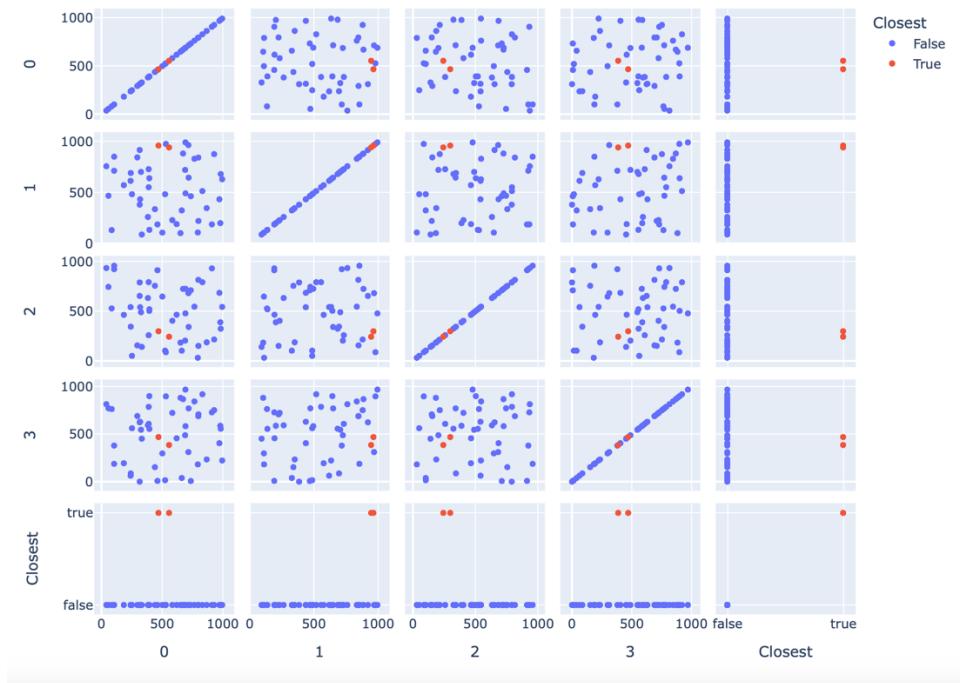
Bagian bonus, akan diuji dengan n=50

a. Dimensi 2



b. Dimensi 4





c. Dimensi 6

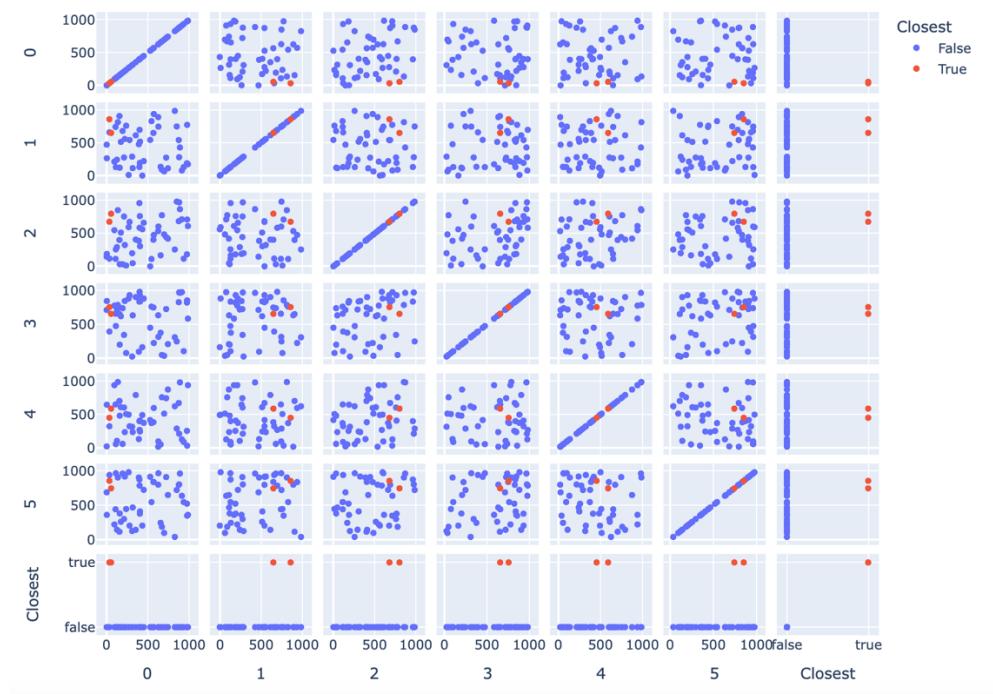
CLOSEST POINT GENERATOR

BY ALTHA CETTA

```
Masukkan banyak titik = 50
Masukkan dimensi yang diinginkan = 6
```

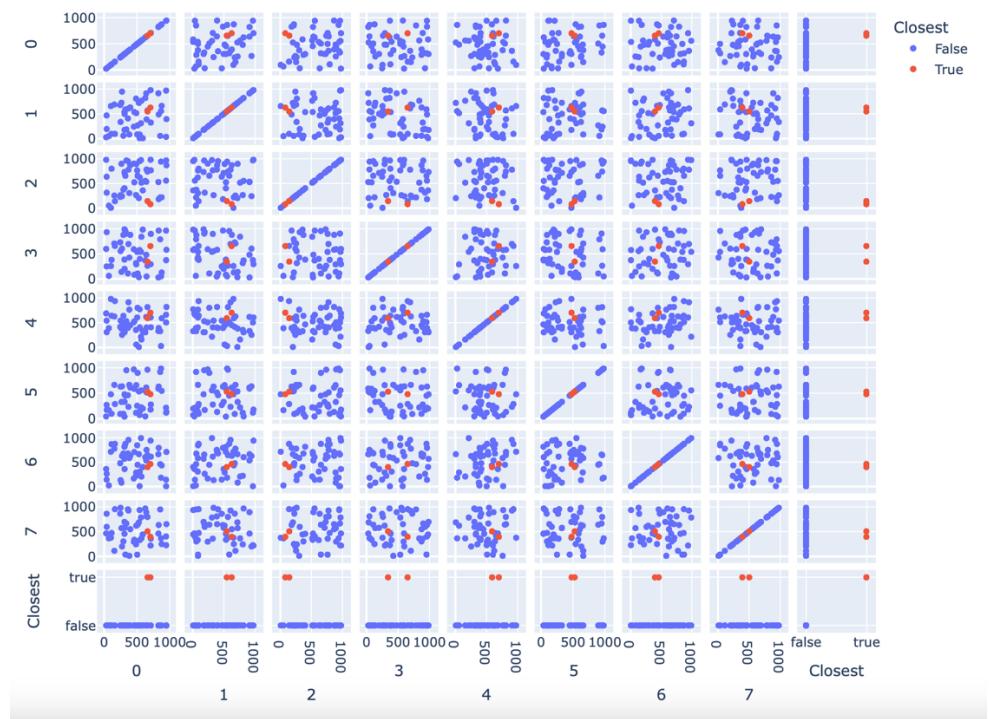
```
[[35.05, 864.11, 683.67, 763.55, 457.0, 859.06], [56.14, 654.51, 806.33, 664.03, 594.48, 749.92]]
Shortest distance = 316.4473796699856
Brute time = 4.155158996582031
Euclidean used = 1225
```

```
[[35.05, 864.11, 683.67, 763.55, 457.0, 859.06], [56.14, 654.51, 806.33, 664.03, 594.48, 749.92]]
Shortest distance = 316.4473796699856
Divide and Conquer time = 3.3452510833740234
Euclidian used = 136
```



d. Dimensi 8





F. Lampiran Milestone

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa ada kesalahan.	✓	
2. Program berhasil running	✓	
3. Program dapat menerima masukan dan menuliskan luaran.	✓	
4. Luaran program sudah benar (solusi <i>closest pair</i> benar)	✓	
5. Bonus 1 dikerjakan	✓	
6. Bonus 2 dikerjakan	✓	