# Laporan Tugas Kecil 2
# IF2211 Strategi Algoritma

**Semester II tahun 2022/2023**

# Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer


**Disusun oleh**

Jauza Lathifah Annassalafi 13521030

**Program Studi Teknik Informatika**
**Sekolah Teknik Elektro dan Informatika**
**Institut Teknologi Bandung**
**2022**

# DAFTAR ISI

# BAB 1

# ALGORITMA DIVIDE AND CONQUER

## 1.1 Mencari Pasangan Titik Terdekat 3D

Pada tugas kecil 2 ini, diberi tugas untuk mengembangkan algoritma mencari pasangan titik terdekat pada bidang 3D. Misalkan terdapat n buah titik pada ruang 3D. Setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x, y, z). Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titik P1 = (x1, y2, z3) dan P2 = (x2, y2, z2) dihitung dengan rumus Euclidean berikut:

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

Persamaan Euclidean Distance 3D

Untuk mencari pasangan titik yang jaraknya terdekat dari titik-titik lain, diterapkan sebuah algoritma, yaitu algoritma divide and conquer untuk penyelesaiannya, dan perbandingannya dengan Algoritma Brute Force.

## 1.2 Algoritma Divide And Conquer

Algoritma Divide and Conquer adalah strategi pemecahan masalah yang besar dengan cara melakukan pembagian masalah tersebut menjadi beberapa bagian yang lebih kecil secara rekursif hingga masalah tersebut dapat dipecahkan secara langsung. Solusi yang didapat dari setiap bagian kemudian digabungkan untuk membentuk sebuah solusi yang utuh. Langkah - langkah umum algoritma Divide and Conquer:

1. Divide : Membagi masalah menjadi beberapa upa masalah yang memiliki kemiripan dengan masalah
2. Conquer : Memecahkan (menyelesaikan) masing-masing upa-masalah(secara rekursif)
3. Combine : Menggabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula

Prinsip dasar dari algoritma ini adalah dengan membagi n input menjadi k subset yang berbeda (1 < k <= n). Dari k subset input yang berbeda akan terdapat k submasalah. Setiap submasalah mempunyai solusinya masing-masing sehingga akan diperoleh k subsolusi. Kemudian, dari k subsolusi akan diperoleh solusi yang optimal atau solusi yang diharapkan.
Jika submasalah dianggap terlalu besar, maka metode Divide and Conquer dapat digunakan lagi secara rekursif.

Algoritma divide and conquer mempunyai kelebihan yang dapat mengurangi kompleksitas pencarian solusi suatu masalah karena prinsip kerjanya yang membagi-bagi masalah menjadi upamasalah-upamasalah yang lebih kecil. Kelebihan tersebut banyak menguntungkan dari segi waktu, tenaga, dan sumberdaya. Salah satu penerapan dari algoritma ini adalah pada mekanisme komputer (atau mesin) paralel. Algoritma Divide And Conquer

terbukti menampilkan hasil yang paling baik dan paling sesuai untuk komputer dengan hirarki memori tinggi serta memiliki cache.

Penyelesaian mencari 2 titik terdekat menggunakan algoritma Divide and Conquer didapat dengan cara berikut:

1. SOLVE : jika n = 2, maka jarak kedua titik dihitung langsung dengan rumus Euclidean.

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

Persamaan Euclidean Distance 3D

$$D(a;b) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$$

Persamaan Euclidean Distance n Dimensi

2. DIVIDE : Bagi himpunan titik ke dalam dua bagian, S1 dan S2 , setiap bagian mempunyai jumlah titik yang sama. L adalah garis maya yang membagi dua himpunan titik ke dalam dua sub-himpunan, masing-masing n/2 titik. Titik-titik sudah diurut menaik berdasarkan absis (x).
3. CONQUER : Secara rekursif, terapkan algoritma Divide and Conquer pada masing-masing bagian untuk mencari pasangan titik terdekat.
4. COMBINE : Pasangan titik yang jaraknya terdekat ada tiga kemungkinan letaknya:
   (a) Pasangan titik terdekat terdapat di dalam bagian S1
   (b) Pasangan titik terdekat terdapat di dalam bagian S2
   (c) Pasangan titik terdekat dipisahkan oleh garis batas L, yaitu satu titik di S1 dan satu titik di S2 .

   Jika kasusnya adalah (c), maka lakukan tahap ketiga (akan dijelaskan kemudian) untuk mendapatkan jarak dua titik terdekat sebagai solusi persoalan semula.

Jika terdapat pasangan titik pleft and pright yang jaraknya lebih kecil dari d, maka kasusnya adalah:

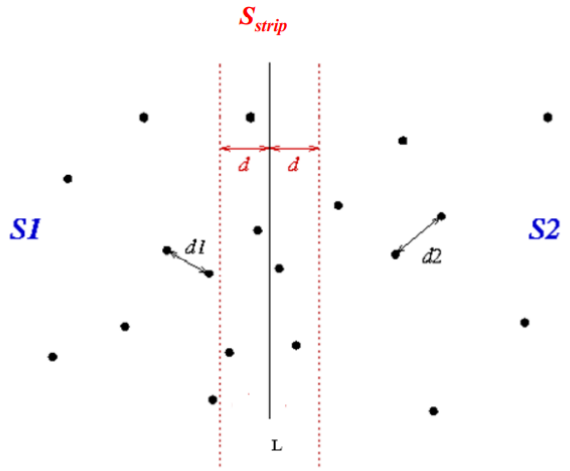
(i) Absis x dari pleft dan pright berbeda paling banyak sebesar d.

(ii) Ordinat y dari pleft dan pright berbeda paling banyak sebesar d.

Ini berarti pleft and pright adalah sepasang titik yang berada di daerah sekitar garis vertikal L (daerah abu-abu). Kita membatasi titik-titik di dalam strip selebar 2d. Oleh karena itu, implementasi tahap ketiga adalah sebagai berikut:

(i) Temukan semua titik di S1 yang memiliki absis x minimal $x_{\frac{n}{2}} - d$ .

(ii ) Temukan semua titik di S2 yang memiliki absis x maksimal $x_{\frac{n}{2}} + d$.




Keterangan: $d = \text{MIN}(d1, d2)$

Sebut semua titik-titik yang ditemukan pada langkah (i) dan (ii) tersebut sebagai himpunan Sstrip yang berisi s buah titik. Urutkan titik-titik di dalam Sstrip dalam urutan ordinat y yang menaik. Misalkan q1 , q2 , ..., qs menyatakan hasil pengurutan. Hitung jarak setiap pasang titik di dalam Sstrip dan bandingkan apakah jaraknya lebih kecil dari d. Kompleksitas algoritma ini adalah

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + cn, & n > 2 \\ a, & n = 2 \end{cases}$$

## 1.3 Algoritma Brute Force sebagai Pembanding

Brute force adalah sebuah pendekatan yang lempang (straight forward) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (problem statement) dan definisi konsep yang dilibatkan. Algoritma brute force memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas. Pada dasarnya, dengan algoritma brute force akan diperiksa semua kemungkinan yang ada untuk memberikan solusi dari permasalahan. Algoritma brute force umumnya tidak cerdas, karena ia membutuhkan jumlah langkah yang besar dalam penyelesaiannya. Algoritma brute force seringkali merupakan pilihan yang kurang disukai karena ketidakmangkusannya itu, tetapi dengan mencari pola-pola yang mendasar, keteraturan, atau trik-trik khusus, biasanya akan membantu kita menemukan algoritma yang lebih cerdas dan lebih mangkus.

Pada algoritma ini, semua titik akan dicari kemungkinan pasangannya dan dihitung jaraknya. Hasil jaraknya akan di assign ke suatu variabel dan akan

dibandingkan dengan jarak pasangan titik lainnya, sehingga akan dihasilkan pasangan titik yang memiliki jarak terdekat. Kompleksitas algoritma brute force adalan $O(n^2)$.

## 1.4 Alur Kerja Program

Program bekerja dengan alur sebagai berikut:

1. Pada saat program dijalankan, program akan meminta 2 buah inputan, yaitu banyaknya titik dan dimensi. Jika memasukan sebuah input yang tidak sesuai, maka program akan terus meminta sebuah inputan hingga benar. Program akan memberikan titik secara random sebanyak banyak titik yang dimasukkan dengan rentang angka -100 sampai dengan 100 yang berbentuk float 2 angka belakang koma. TItik juga langsung diurut menaik berdasarkan absis (x).
2. Setelah mendapatkan titik-titik sebanyak n pada sebuah dimensi, program akan menampilkan dua titik terdekat beserta jaraknya, banyaknya operasi perhitungan rumus Euclidean, dan waktu eksekusi dalam mendapatkan solusi tersebut. Solusi-solusi tersebut didapatkan dengan cara:
    1) Fungsi FindClosestPair(array, countEdDnC) yang dipanggil di program utama akan memberikan solusi berupa dua buah titik terdekat dengan algoritma Divide and Conquer, jaraknya, dan banyaknya operasi perhitungan rumus Euclidean. Fungsi ini memiliki 2 parameter yaitu array berupa array of array yang berisi array-array yang setiap arraynya itu adalah sebuah titik. Fungsi ini akan memanggil fungsi divide(array) yang akan membagi array of array menjadi 2 bagian (A1 dan A2). Akan dipanggil fungsi FindClosestPair(array, countEdDnC) kembali dengan memasukkan bagian array of array yang sudah dibagi, secara rekursif. Pada saat kedua bagian tersebut sudah tersisa 2 titik, maka disetiap bagian akan dihitung jarak euclideannya dan akan dicari yang memiliki jarang paling kecil. Apabila terdapat bagian yang memiliki 3 titik, maka akan dicari dua titik terdekat menggunakan persamaan jarak euclidean secara brute force. Setelah menemukan dua titik yang memiliki jarak terdekat akan dibandingkan kembali dengan titik dan jarak terdekat hasil fungsi Sstrip(array, distance, countEdDnC). Pada fungsi ini, akan ditemukan semua titik di A1 yang memiliki absis x minimal xn/2 – d dan juga semua titik di A2 yang memiliki absis x maksimal xn/2 + d. Titik- titik yang ditemukan akan ditambahkan ke sebuah array sebagai himpunan Sstrip yang akan dihitung tiap pasang titik yang ada dan dicari pasangan titik yang memiliki jarak terdekat. Sepasang titik yang memiliki jarak terdekat hasil fungsi Sstrip akan dibandingkan dengan sepasang titik yang

memiliki jarak terdekat hasil fungsi FindClosestPair dan mengembalikan hasil yang memiliki jarak terkecil/terdekat.

2) Dalam menghitung banyaknya operasi perhitungan rumus Euclidean, dibuat sebuah variabel countEdDnC. Variabel ini akan bertambah setiap fungsi EuclideanDistance(array, countEdDnC) dipanggil.

3. Pada saat menghitung di dimensi 2 atau 3 program dapat menampilkan visualisasi pada dimensi tersebut dengan pasangan titik yang memiliki jarak paling dekat memiliki warna dan ukuran titik yang berbeda.

# BAB 2

# PROGRAM

## 2.1 Library dan Fungsi

Tugas ini diselesaikan menggunakan bahasa pemrograman Python dengan bantuan library yang berbeda-beda di setiap filenya dan membuat beberapa fungsi.

### 2.1.1 File function.py

Library yang digunakan adalah sebagai berikut:

1. import math : digunakan untuk menghitung akar pangkat
2. import random : digunakan untuk memberikan titik-titik random

#### 2.1.1.1 sort(array)

Fungsi ini akan dipanggil pada saat program mengeluarkan titik-titik acak, sehingga array yang terdiri dari banyak titik terurut menaik berdasarkan absis (x).

```
# fungsi untuk mengurutkan array berdasarkan nilai x
def sort(array):
    for i in range(len(array)):
        for j in range(len(array)-1):
            if array[j][0] > array[j+1][0]:
                array[j], array[j+1] = array[j+1], array[j]
    return array
```

### 2.1.1.2 inputan()

Fungsi ini berfungsi untuk menerima inputan banyak titik dan dimensi dan melakukan validasi terhadap masukkannya yang hanya menerima banyak titik yang lebih dari sama dengan 2.

```
# fungsi untuk menginputkan banyak dimensi dan titik secara random
def inputan():
    array = []
    global dimensi
    global banyakTitik
    banyakTitik = int(input("   Banyaknya titik: "))
    while banyakTitik < 2:
        banyakTitik = int(input("   Input tidak valid! Banyaknya titik: "))
    dimensi = int(input("   Banyak dimensi: "))
    while dimensi < 2:
        dimensi = int(input("   Input tidak valid! Banyak dimensi: "))
    for i in range(banyakTitik):
        temp =[]
        for j in range(dimensi):
            titik = round(random.uniform(-100, 100), 2)
            temp.append(titik)
        array.append(temp)
    return (sort(array))
```

### 2.1.1.3 EuclideanDistance(array, countEdDnC)

Fungsi ini berfungsi untuk menghitung jarak euclidean dari dua titik pada n dimensi. Setiap fungsi ini dipanggil variabel countEdDnC akan bertambah 1 yang akan menjadi nilai dari banyaknya operasi jarak euclidean dalam menemukan dua titik terdekat.

```
# fungsi untuk menghitung jarak euclidean
def EuclideanDistance(array, countEdDnC):
    #mencari jarak euclidean dimensi > 2
    dimensi = len(array[0])
    if dimensi > 2:
        jarak = 0
        for i in range(dimensi):
            jarak += (array[0][i]-array[1][i])**2
        jarak = math.sqrt(jarak)
        closest = [array[0], array[1]]
    #mencari jarak euclidean dimensi 2
    else:
        jarak = math.sqrt((array[0][0]-array[1][0])**2 + (array[0][1]-array[1][1])**2)
        closest = [array[0], array[1]]
    countEdDnC += 1
    return closest, jarak, countEdDnC
```

### 2.1.1.4 divide(array)

Fungsi ini digunakan untuk membagi suatu array of array menjadi 2 bagian.

```
# fungsi untuk membagi suatu array menjadi 2 array
def divide(array):
    A1 = []
    A2 = []
    for i in range(len(array)):
        if i < (len(array)//2):
            A1.append(array[i])
        else:
            A2.append(array[i])
    return A1, A2
```

### 2.1.1.5 FindClosestPair(array, countEdDnC)

Fungsi FindClosestPair(array, countEdDnC) yang dipanggil di program utama akan memberikan solusi berupa dua buah titik terdekat dengan algoritma Divide and Conquer, jaraknya, dan banyaknya operasi perhitungan rumus Euclidean.

```
# fungsi untuk mencari pasangan titik terdekat dengan divide and conquer
def FindClosestPair(array, countEdDnC):
    if len(array) == 2:
        closest, distance, countEdDnC = EuclideanDistance(array, countEdDnC)
    elif len(array) == 3:
        twoPoint1, distance1, countEdDnC = EuclideanDistance([array[0], array[1]], countEdDnC)
        twoPoint2, distance2, countEdDnC = EuclideanDistance([array[0], array[2]], countEdDnC)
        twoPoint3, distance3, countEdDnC = EuclideanDistance([array[1], array[2]], countEdDnC)
        distance = min(distance1, distance2, distance3)
        if distance == distance1:
            closest = twoPoint1
        elif distance == distance2:
            closest = twoPoint2
        else:
            closest = twoPoint3
    else:
        A1, A2 = divide(array)
        twoPoint4, distance1, countEdDnC = FindClosestPair(A1, countEdDnC)
        twoPoint5, distance2, countEdDnC = FindClosestPair(A2, countEdDnC)
        distance = min(distance1, distance2)
        twoPoint, distance, countEdDnC = Sstrip(array, distance, countEdDnC)
        if distance == distance1:
            closest = twoPoint4
        elif distance == distance2:
            closest = twoPoint5
        else:
            closest = twoPoint
    return closest, distance, countEdDnC
```

### 2.1.1.6 Sstrip(array, distance, countEdDnC)

Fungsi ini berfungsi untuk menemukan semua titik di array of array yang sudah dibagi bagianya yang memiliki absis x minimal xn/2 – d dan maksimal xn/2 + d lalu tempatkan titik-titik tersebut sebagai himpunan Sstrip yang berisi s buah titik. Hitung jarak setiap pasang titik di dalam Sstrip(ambil dua titik yang memiliki jarak terdekat)..

```
# fungsi untuk mencari titik terdekat pada suatu strip
def Sstrip(array, distance, countEdDnC):
    closest = []
    if (len(array) % 2 == 1):
        middle = array[len(array)//2][0]
    else :
        middle = (array[len(array)//2][0] + array[(len(array)//2)+1][0])/2
    temp = []
    for i in range(len(array)):
        if (array[i][0] <= middle + distance and array[i][0] >= middle - distance):
            temp.append(array[i])
    for i in range (len(temp)):
        for j in range (i+1, len(temp)):
            array, dist, countEdDnC = EuclideanDistance([temp[i], temp[j]], countEdDnC)
            if dist < distance:
                distance = dist
                closest = [temp[i], temp[j]]
            else :
                continue
    return closest, distance, countEdDnC
```

### 2.1.1.7 bruteForce(array)

Fungsi ini digunakan untuk mencari pasangan titik yang memiliki jarak terdekat dengan menghitung semua kemungkinan dua pasang titik yang akan dibandingkan setiap jaraknya dan mengeluarkan dua pasang titik terdekat beserta jaraknya.

```
# fungsi untuk mencari pasangan titik terdekat dengan brute force
def bruteForce(array):
    #mencari jarak euclidean terdekat dari banyak pasang titik
    distance = 999999
    countEdBF = 0
    for i in range(len(array)):
        for j in range(i+1,len(array)):
            if i != j:
                twoPoint, distance1, countEdBF = EuclideanDistance([array[i], array[j]], countEdBF)
                if distance1 < distance:
                    distance = distance1
                    closest = [array[i], array[j]]
                else :
                    continue
    return closest, distance, countEdBF
```

Gambar 2.1.1.7.1 bruteForce

### 2.1.1.8 hasil(array, distance, time, countEd)

Fungsi ini digunakan untuk mengeluarkan hasil dari dua algoritma berupa pasangan titik terdekat beserta jaraknya. Ditampilkan juga lama waktu eksekusi program di setiap algoritma dan banyaknya operasi jarak euclidean.

```
# fungsi untuk menampilkan hasil
def hasil(array, distance, time, countEd):
    print(" ")
    print("   Titik 1          = ", end=" ")
    for i in range (len(array[0])):
        print(array[0][i], end="  ")
    print(" ")
    print("   Titik 2          = ", end=" ")
    for i in range (len(array[0])):
        print(array[1][i], end="  ")
    print(" ")
    print("   Jarak            = ", distance)
    print("   Waktu Eksekusi   = ", time)
    print("   Banyak Operasi Euclidean Distance = ", countEd)
    print(" ")
```

Gambar 2.1.1.8.1 hasil

## 2.1.2 File visualize.py

Library yang digunakan adalah sebagai berikut:

1. import numpy as np : digunakan untuk vektor dan matriks
2. import matplotlib.pyplot as plt : digunakan untuk visualisasi data

### 2.1.2.1 Visualization(array, arrayDnC)

Fungsi ini akan dipanggil di program utama, pada saat user memasukkan ukuran dimensinya adalah 3 dan user ingin memvisualisasikannya maka fungsi ini akan akan menampilkan hasil dalam bentuk grafik 3D dengan pasangan titik yang memiliki jarak paling dekat memiliki warna dan ukuran titik yang berbeda.

Pada saat user memasukkan ukuran dimensinya adalah 2 dan user ingin memvisualisasikannya maka fungsi ini akan menampilkan hasil dalam bentuk grafik 2D dengan pasangan titik yang memiliki jarak paling dekat memiliki warna dan ukuran titik yang berbeda.

```python
import numpy as np  # library untuk operasi vektor dan matriks
import matplotlib.pyplot as plt # library untuk visualisasi data
import function as f # library untuk fungsi-fungsi yang dibutuhkan

# fungsi untuk menampilkan hasil dalam bentuk grafik 2D/3D
def Visualization(array, arrayDnC):
    if f.dimensi == 3:
        visualisasi = str(input("   Tampilkan penggambaran semua titik dalam bidang 3D? (y/n) : "))
        if visualisasi == "y" or visualisasi == "Y":
            titik1 = array

            titik1.remove(arrayDnC[0])
            titik1.remove(arrayDnC[1])

            dots = np.array(titik1)
            DnC = np.array(arrayDnC)

            fig = plt.figure(figsize = (10,5.5))
            ax = plt.axes(projection='3d')
            ax.grid()
            if (len(dots)!= 0):
                ax.scatter(dots[:,0], dots[:,1], dots[:,2], s = 15, alpha = 0.5, edgecolors = 'grey', color = 'yellow')
            ax.scatter(DnC[:,0], DnC[:,1], DnC[:,2], s = 30, color = 'red', marker = 'o', edgecolors = 'black')
            ax.set_title('3D Scatter Plot')

            ax.set_xlabel('x', labelpad=20)
            ax.set_ylabel('y', labelpad=20)
            ax.set_zlabel('z', labelpad=20)

            plt.show()
```

```python
    elif f.dimensi == 2:
        visualisasi = str(input("   Tampilkan penggambaran semua titik dalam bidang 2D? (y/n) : "))
        if visualisasi == "y" or visualisasi == "Y":
            titik1 = array
            titik1.remove(arrayDnC[0])
            titik1 = array
            dots = np.array(titik1)
            DnC = np.array(arrayDnC)

            plt.rcParams["figure.figsize"] = [7.50, 3.50]
            plt.rcParams["figure.autolayout"] = True
            if (len(dots)!= 0):
                plt.scatter(dots[:, 0], dots[:, 1], color = 'green', s = 15)
            plt.scatter(DnC[:, 0], DnC[:, 1], color = 'red', s = 30)

            plt.show()
    else:
        print("   Maaf, Program hanya bisa menampilkan visualisasi pada bidang 2D dan 3D.")
```

Gambar 2.1.2.1.1 Visualization

### 2.1.3 File splashScreen.py

#### 2.1.3.1 output(output)

Fungsi ini akan dipanggil pada program utama yang bertujuan untuk memperindah hasil keluaran program.

```
# fungsi splash screen
def output(output):
    if output == 1:
        print('''
 _______ _               _      _____ _                                 _____      _
(_______|_)             | |    / ____) |                               (____ \    (_)
 _____    _ ____    _  | |   | /     | | ___   ___  ____  ___| |_      ____) )___ _   ____
|  ___) | |  _ \ \ / || |   | |     | |/ _ \ /___)/ _  )/___)  _)    |  ____/ _  | |/ ___)
| |     | | | | | ( (_| |   | \_____| | |_| |___ ( (/ /|___ | |__    | |   ( ( | | | |
|_|     |_|_| |_|\____|    \______)_|\___/(___/ \____|___/ \___)   |_|    \_||_|_|_|

        ''')
    elif output == 2:
        print('''
 ____________________________________________________________________________________________________
|                                                                                                    |
|                                  Algoritma Divide and Conquer                                      |
|____________________________________________________________________________________________________|
        ''')
    elif output == 3:
        print('''
 ____________________________________________________________________________________________________
|                                                                                                    |
|                                    Algoritma Brute Force                                           |
|____________________________________________________________________________________________________|
        ''')
    elif output == 4 :
        print('''
                                    Terima kasih!🙏
 ____________________________________________________________________________
|                                                                            |
|       Rasa ingin menyerah itu wajar, tetapi menyerah bukan keputusan       |
|         yang tepat untuk seorang manusia super tangguh sepertimu.          |
|              Tetaplah melangkah, walau harus merubah arah.                 |
|                          - Martabak Legit Group-                           |
|____________________________________________________________________________|
        ''')
```

Gambar 2.1.3.1.1 output

### 2.1.4 File main.py

Library yang digunakan adalah sebagai berikut:

1. import os : digunakan clear screen
2. import time : digunakan untuk menangani waktu dalam mengeksekusi lamanya program saat dijalankan

Pada program utama akan memanggil beberapa fungsi, yaitu output() untuk menampilkan header sebagai pembuka dan penutup, inputan() untuk melakukan input banyak titik dan dimensi, titik-titik akan ditentukan secara acak dari program sebanyak inputan banyak titik. time.time()) untuk menghitung waktu lama program dalam menemukan solusi-solusi, FindClosestPair(array, countEdDnC) untuk menemukan pasangan titik terdekat beserta jaraknya

menggunakan algoritma Divide and Conquer, bruteForce(array) untuk menemukan pasangan titik terdekat beserta jaraknya menggunakan algoritma brute force, hasil(twoPoint, distance, Time, countEd) untuk menampilkan hasil berupa pasangan titik terdekat beserta jaraknya, waktu eksekusi program, dan banyak operasi euclidean distance. Visualization(array, arrayDnC) untuk menampilkan hasil dalam bentuk grafik 2D/3D dengan pasangan titik yang memiliki jarak paling dekat memiliki warna dan ukuran titik yang berbeda.

```python
import os                   # untuk fungsi clear screen
import time                 # untuk fungsi time
import visualize as v       # untuk memanggil fungsi yang ada di file visualisasi
import splashScreen as s    # untuk memanggil fungsi yang ada di file splashScreen
import function as f        # untuk memanggil fungsi yang ada di file function

def main():
    os.system('cls')

    s.output(1)

    array = f.inputan()
    countEdDnC = 0

    # Find Closest Pair with Divide and Conquer
    s.output(2)
    start = time.time()
    twoPointDnC, distanceDnC, countEdDnC = f.FindClosestPair(array, countEdDnC)
    end = time.time()
    DnCTime = end-start
    f.hasil(twoPointDnC, distanceDnC, DnCTime, countEdDnC)

    # Find Closest Pair with Brute Force
    s.output(3)
    start1 = time.time()
    twoPointBF, distanceBF, countEdBF = f.bruteForce(array)
    end1 = time.time()
    BFTime = end1-start1
    f.hasil(twoPointBF, distanceBF, BFTime, countEdBF)

    # Visualisasi
    v.Visualization(array, twoPointDnC)

    print(" ")
    clear = str(input("   Apakah anda ingin menginput kembali? (y/n) : "))
    if clear == "y" or clear == "Y":
        main()
    else:
        os.system('cls')
        s.output(1)
        s.output(4)

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

Gambar2.1.4.1 main

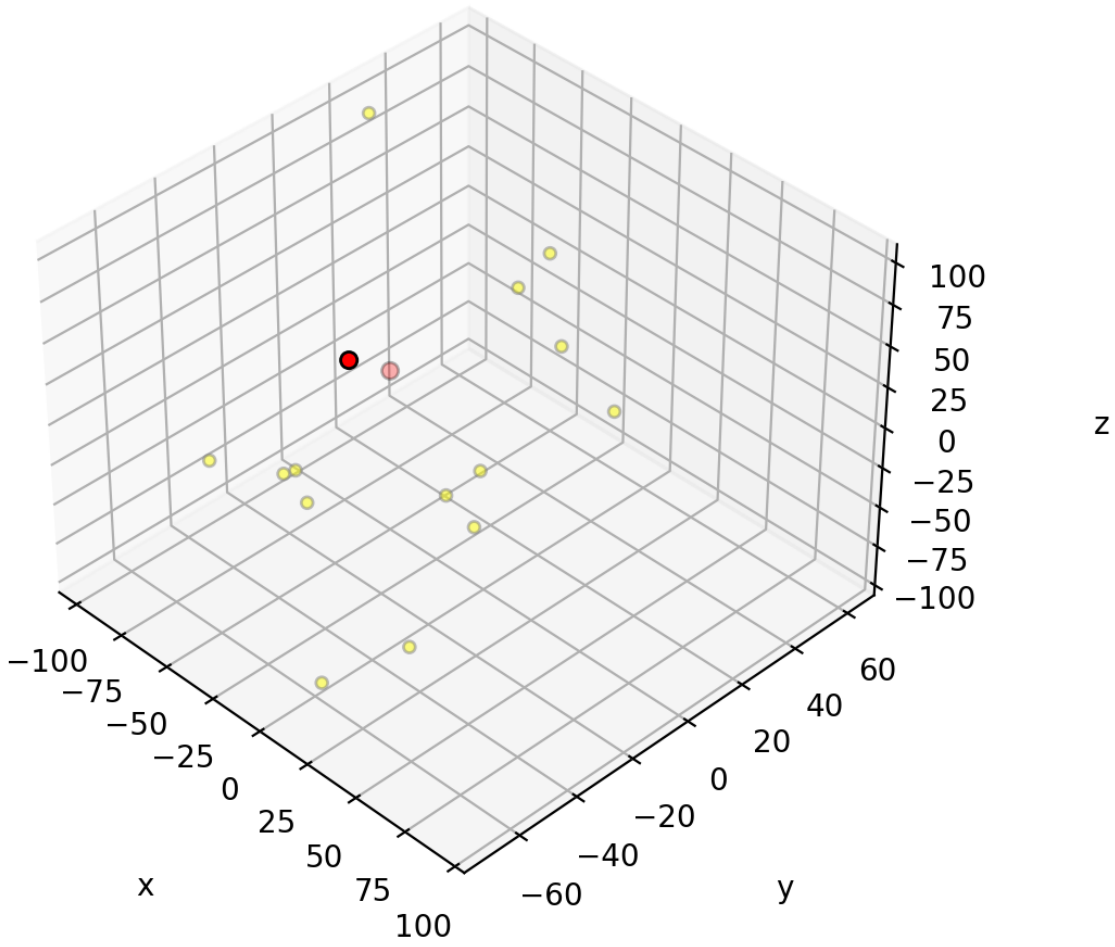
# BAB 3

# TEST CASE

## 3.1 Test Case 1

- Banyak Titik : 16
- Dimensi : 3
- Visualisasi : Ya

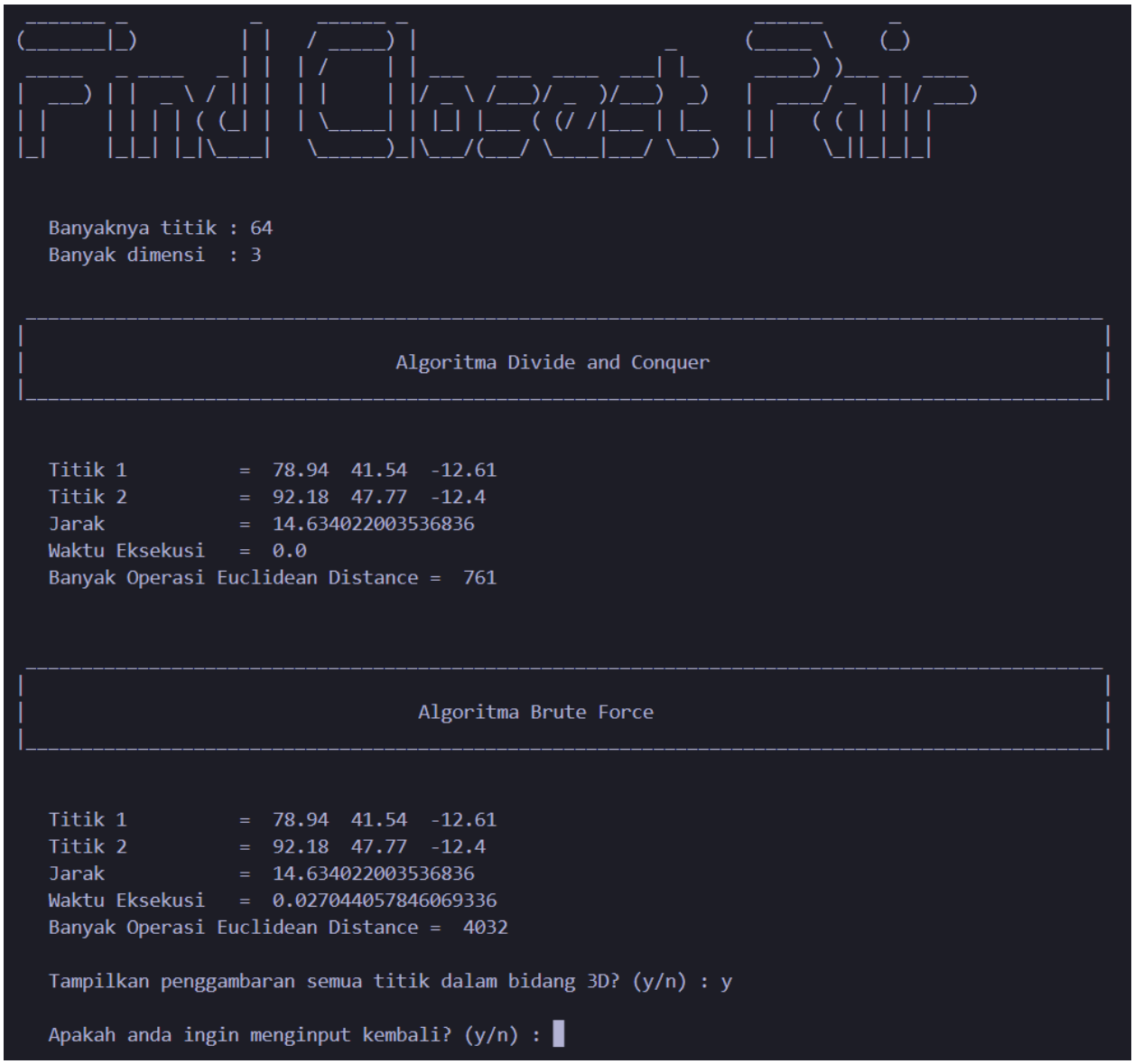

Gambar 3.1.1 Test case 1



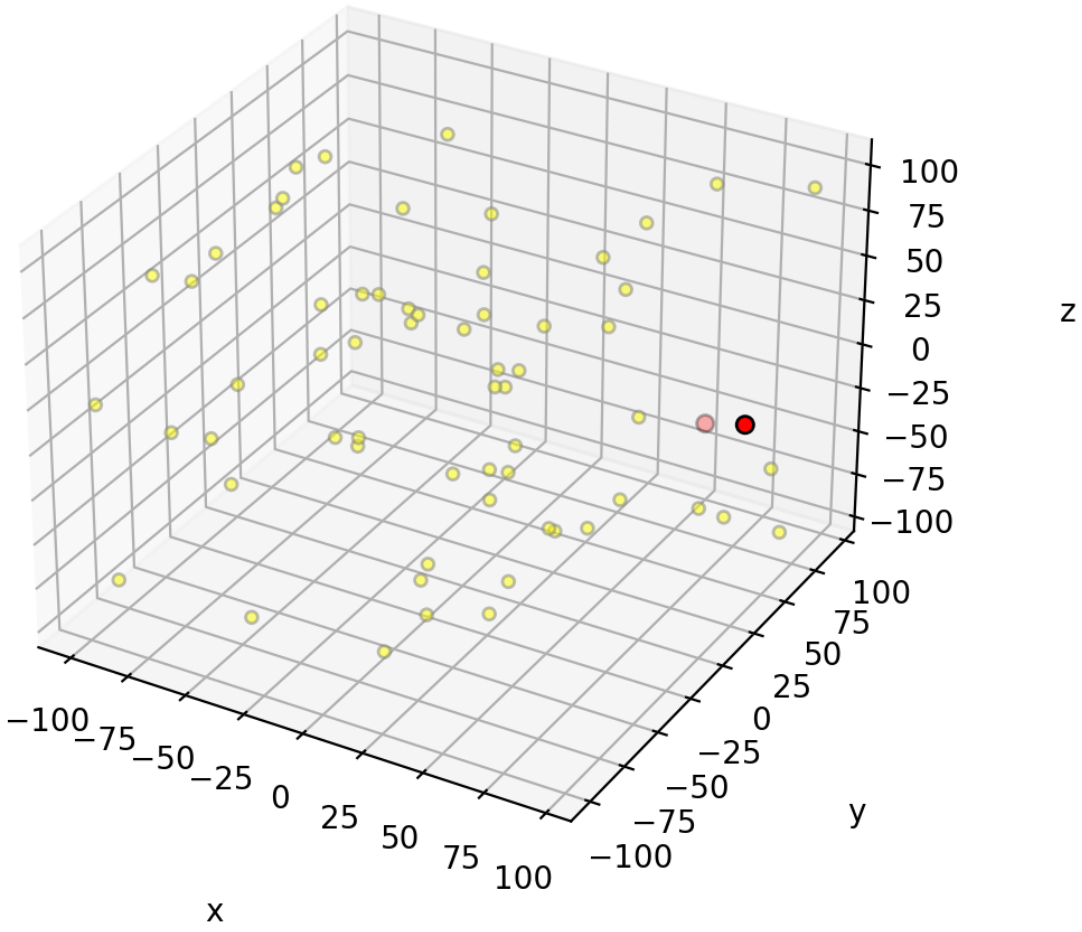

Gambar 3.1.2 Visualisasi 3D Test case 1

## 3.2 Test Case 2

- Banyak Titik : 64
- Dimensi : 3
- Visualisasi : Ya




Gambar 3.2.1 Test case 2

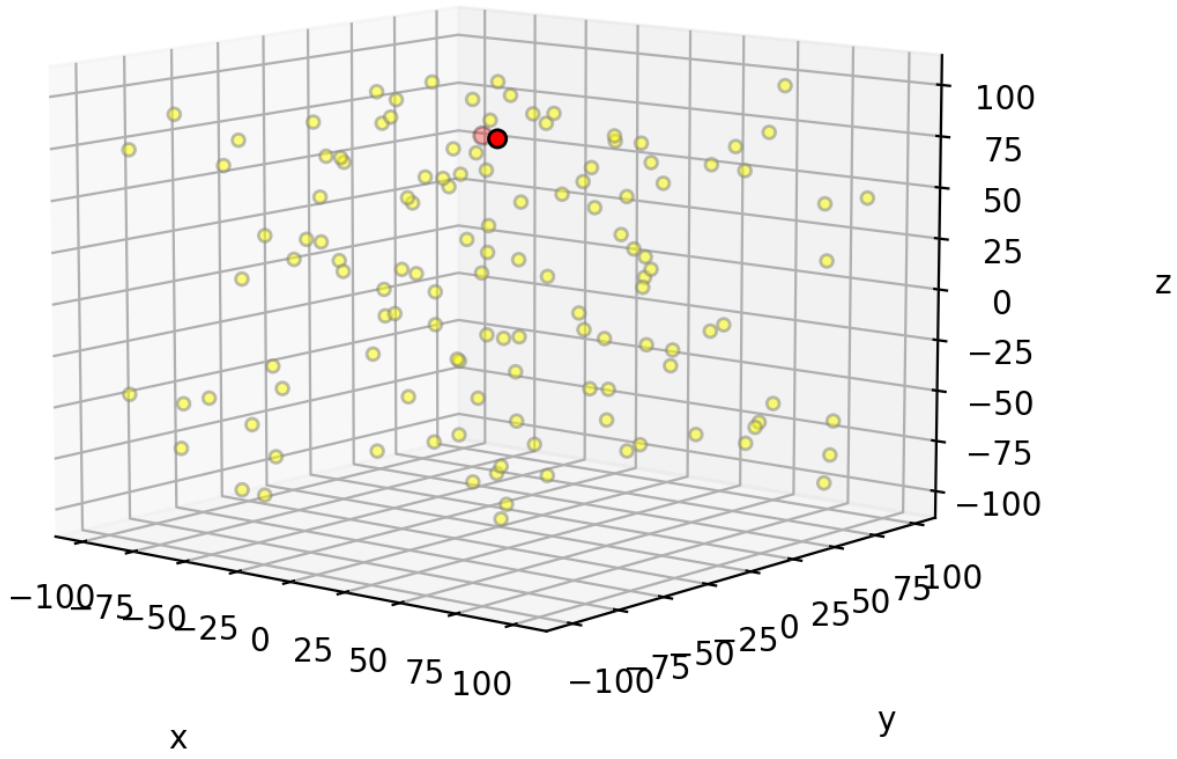



Gambar 3.2.2 Visualisasi 3D test case 2

## 3.3 Test Case 3

- Banyak Titik : 128
- Dimensi : 3
- Visualisasi : Ya

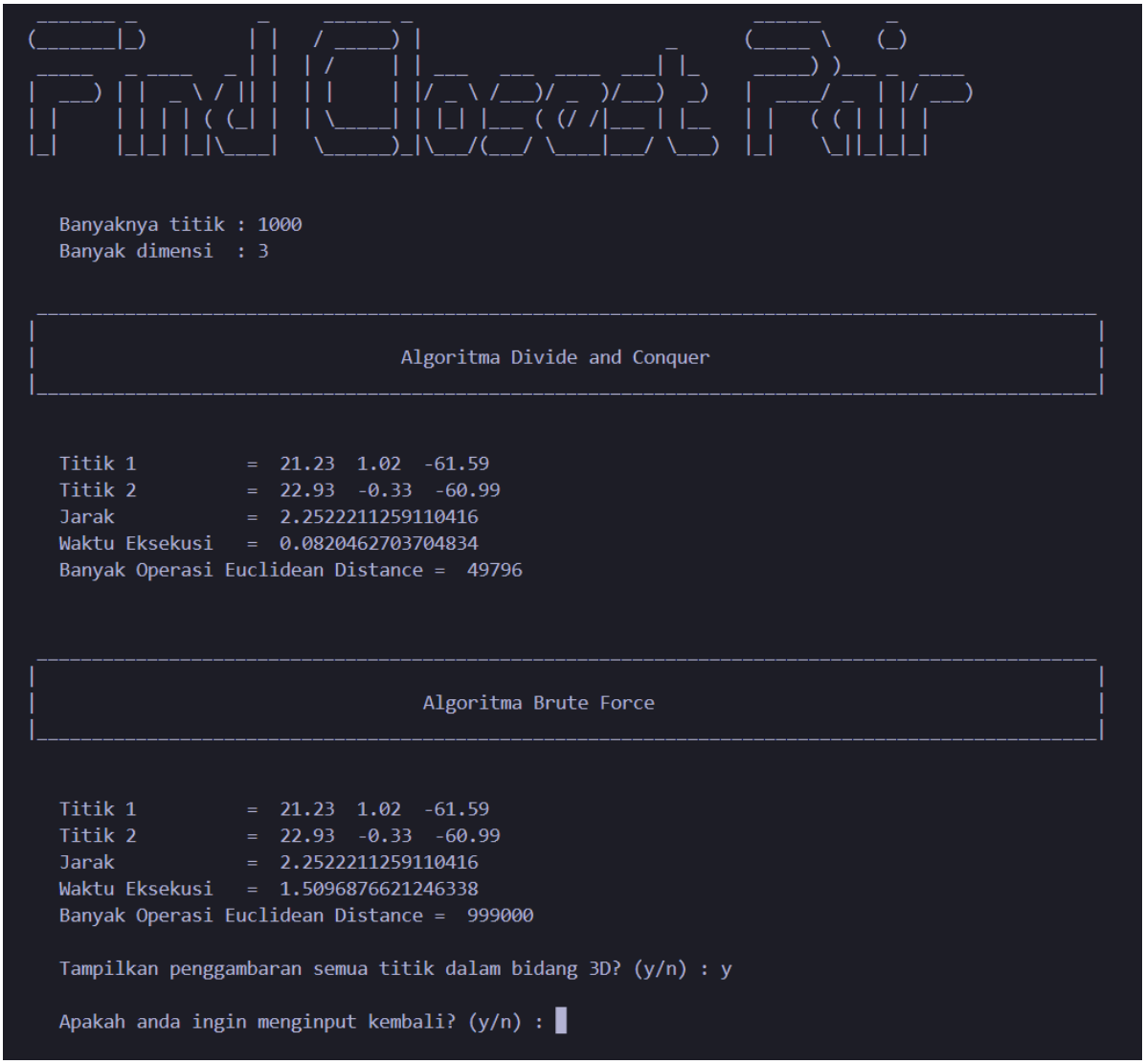

Gambar 3.3.1 Test case 3



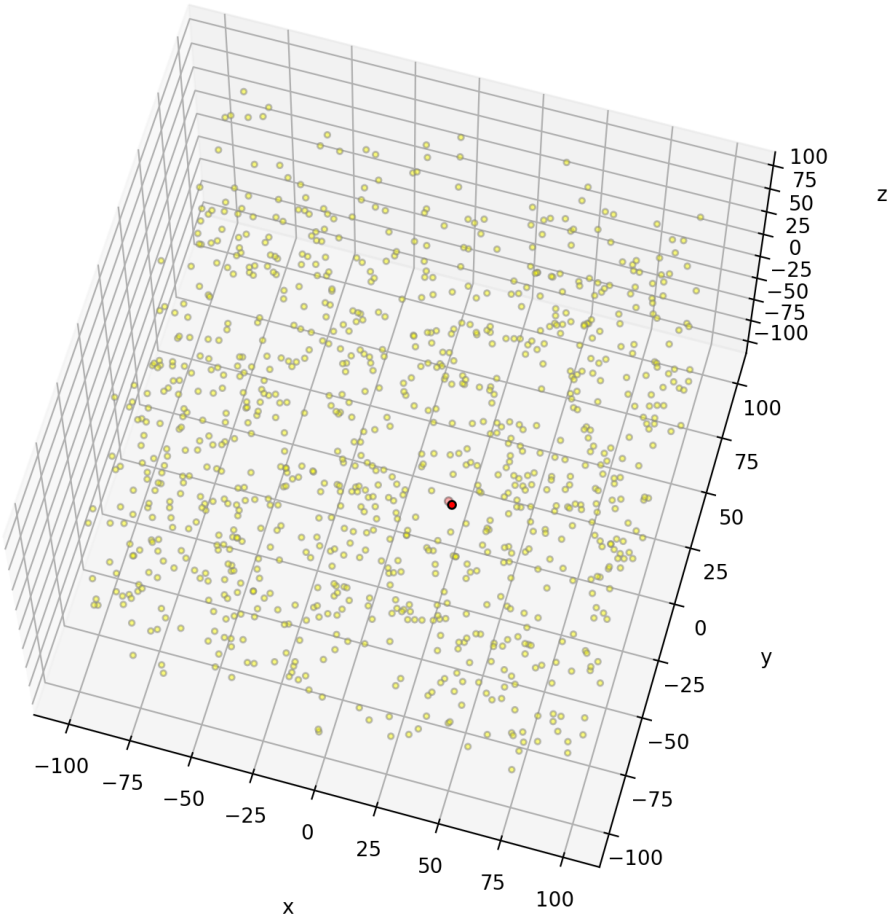

Gambar 3.3.2 Visualisasi 3D test case 3

## 3.4 Test Case 4

- Banyak Titik : 1000
- Dimensi : 3
- Visualisasi : Ya




Gambar 3.4.1 Test case 4

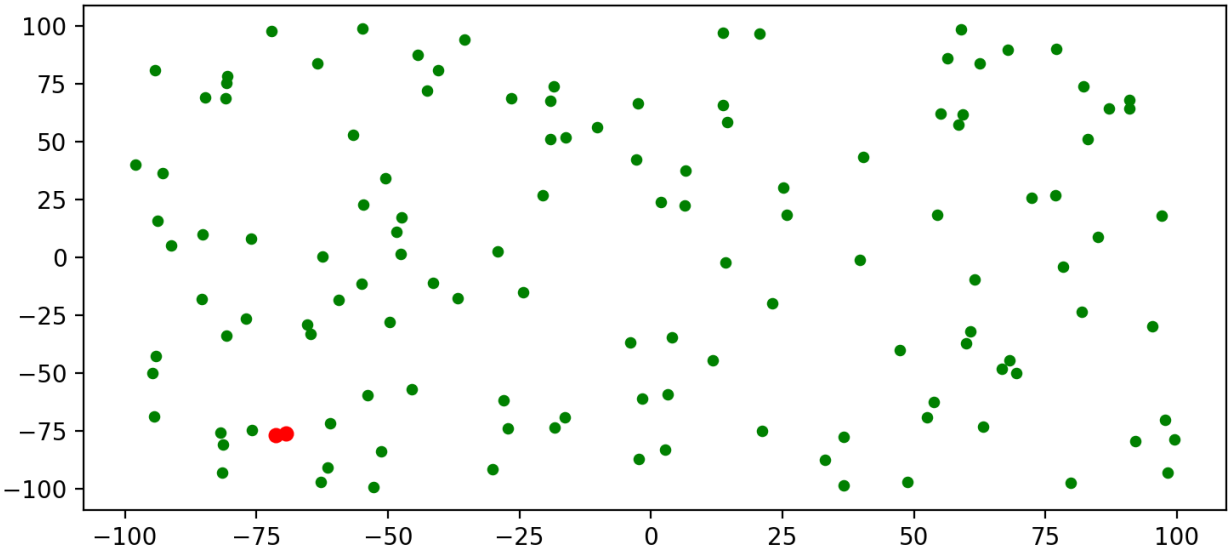



Gambar 3.4.2 Visualisasi 3D test case 4

## 3.5 Test Case 5

- Banyak Titik : 128
- Dimensi : 2
- Visualisasi(2D) : Ya

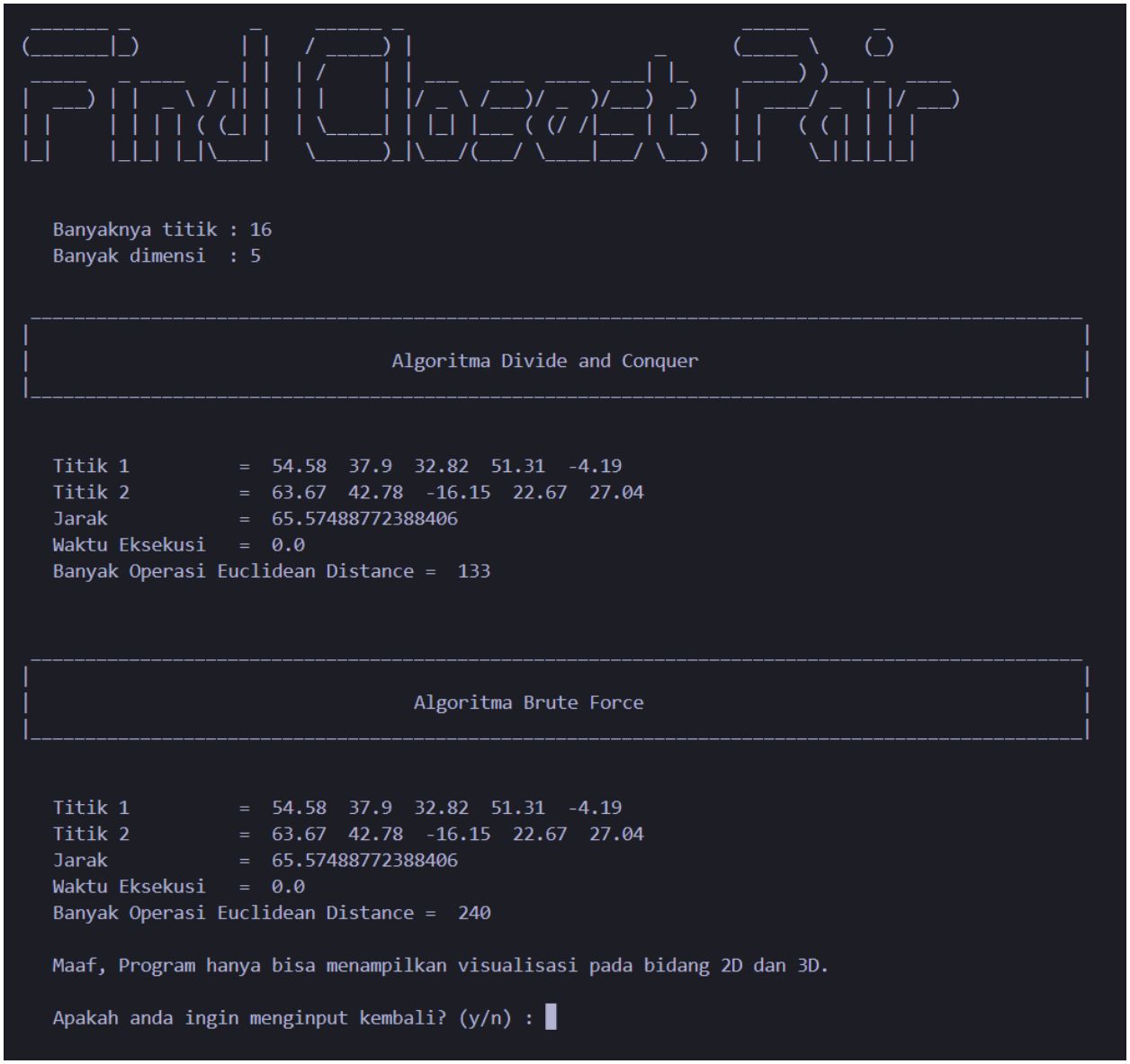

Gambar 3.5.1 Test case 5




Gambar 3.5.2 Visualisasi 2D test case 5

## 3.6 Test Case 6

- Banyak Titik : 16
- Dimensi : 5
- Visualisasi : -




Gambar 3.6.1 Test case 6

## 3.7 Test Case 7

- Banyak Titik : 64
- Dimensi : 6
- Visualisasi : -


Gambar 3.7.1 Test case 7

## 3.8 Test case 8

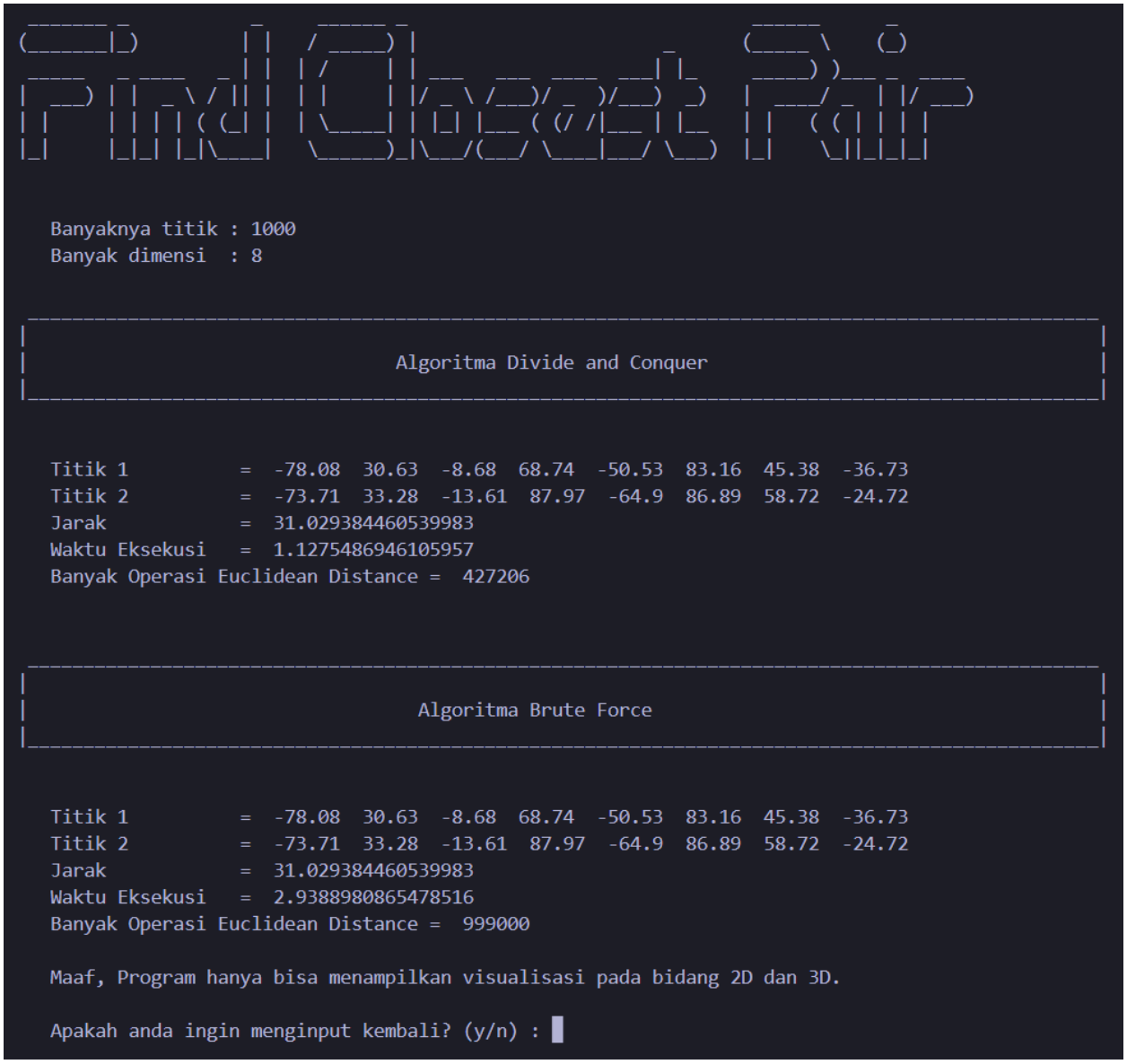
- Banyak Titik : 128
- Dimensi : 7
- Visualisasi : -

```
 _______ _           _     ______ _                          ______      _
(_______|_)         | |   / _____) |                        (_____ \    (_)
 _____    _ ____    _ | |  | /     | | ___   ___  ____  ___| |_    _____) )___ _  ____
|  ___) | |  _ \ / || |  | |     | |/ _ \ /___)/ _  )/___)  _)   |  ____/ _  | |/ ___)
| |     | | | | ( (_| |  | \_____| | |_| |___ ( (/ /|___ | |__  | |   ( ( | | | |
|_|     |_|_| |_|\____|   \______)_|\___/(___/ \____|___/ \___) |_|    \_||_|_|_|


   Banyaknya titik : 128
   Banyak dimensi  : 7

 ____________________________________________________________________________________________
|                                                                                            |
|                                Algoritma Divide and Conquer                                |
|____________________________________________________________________________________________|


   Titik 1          =  -80.38  -30.1  -56.61  10.65  -40.91  84.8  -74.61
   Titik 2          =  -70.89  -36.9  -26.66  -21.69  -82.65  78.77  -63.73
   Jarak            =  63.05664992687132
   Waktu Eksekusi   =  0.023498058319091797
   Banyak Operasi Euclidean Distance =  11335


 ____________________________________________________________________________________________
|                                                                                            |
|                                  Algoritma Brute Force                                     |
|____________________________________________________________________________________________|


   Titik 1          =  -80.38  -30.1  -56.61  10.65  -40.91  84.8  -74.61
   Titik 2          =  -70.89  -36.9  -26.66  -21.69  -82.65  78.77  -63.73
   Jarak            =  63.05664992687132
   Waktu Eksekusi   =  0.04692578315734863
   Banyak Operasi Euclidean Distance =  16256

   Maaf, Program hanya bisa menampilkan visualisasi pada bidang 2D dan 3D.

   Apakah anda ingin menginput kembali? (y/n) :
```

Gambar 3.8.1 Test case 8

## 3.9 Test Case 9

- Banyak Titik : 1000
- Dimensi : 8
- Visualisasi : -




Gambar 3.9.1 Test case 9

# BAB 4

# LAMPIRAN

## 4.1 Link Repository

Link : https://github.com/lostgirrlll/Tucil2_13521030

## 4.2 Cek List

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