LAPORAN TUGAS KECIL II

STRATEGI ALGORITMA

MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA DIVIDE AND CONQUER

Kelas Mahasiswa K03

Dosen: Ir. Rila Mandala, M.Eng., Ph.D.

Diajukan sebagai tugas kecil Mata Kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2022/2023

Disusun Oleh:

Jason Rivalino (13521008)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

DAFTAR ISI

BAB I

DESKRIPSI MASALAH

Proses pencarian sepasang titik terdekat dengan Algoritma Divide and Conquer sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Dalam Tugas Kecil 2 kali ini, tugas yang diberikan adalah untuk mengembangkan algoritma mencari sepasang 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 titk P1 = (x1, y1, z1) dan P2 = (x2, y2, z2) dihitung dengan rumus Euclidean berikut:

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

Program yang dibuat adalah untuk mencari sepasang titik yang jaraknya terdekat datu sama lain dengan menerapkan algoritma divide and conquer untuk penyelesaiannya, dan perbandingannya dengan Algoritma Brute Force.

BAB II

TEORI SINGKAT

I. Teori Singkat Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* merupakan algoritma yang berasal dari dua kata yaitu *Divide* dan *Conquer*. *Divide* pada persoalan algoritma ini sendiri adalah proses membagi persoalan menjadi *sub*-persoalan yang mirip dengan persoalan semula tetapi memiliki ukuran yang lebih kecil. Sedangkan, *conquer* dalam persoalan algoritma ini adalah proses untuk menyelesaikan masing-masing *sub*-persoalan. Penyelesaian yang dilakukan bisa secara langsung (untuk ukuran kecil) ataupun menggunakan proses rekursif (untuk ukuran besar). Setelah permasalahan diselesaikan, kemudian melakukan proses *combine* untuk menggabungkan solusi masing-masing *sub*-persoalan untuk membentuk solusi semula.

Dalam Algoritma *Divide and Conquer*, terdapat beberapa objek persoalan yang dibagi seperti tabel, matriks, eksponen, polinom, dll. Persoalan-persoalan ini pada umumnya memiliki karakteristik yang sama dengan karakteristik persoalan semula sehingga metode ini lebih cocok diselesaikan dengan skema rekursif.

Kompleksitas yang terdapat dalam Algoritma *Divide and Conquer* sendiri adalah sebagai berikut:

$$T(n) = \begin{cases} g(n), & n \leq n_o \\ T(n_1) + T(n_2) + \cdots + T(n_r) + f(n), & n > n_o \end{cases}$$

Penjelasan:

$T(n)$ = kompleksitas waktu penyelesaian persoalan P berukuran n

$g(n)$ = kompleksitas waktu untuk proses *solving* jika *n* sudah berukuran kecil

$T(n_1) + T(n_2) + \cdots + T(n_r)$ = kompleksitas waktu untuk memproses setiap *sub*-persoalan

$f(n)$ = kompleksitas waktu untuk proses *combine* solusi dari setiap masing-masing *sub*-persoalan

Beberapa contoh dari penerapan Algoritma *Brute Force* ini adalah sebagai berikut:

1. Persoalan pencarian nilai maksimum dan minimum (MinMaks)
2. Persoalan menghitung perpangkatan

3. Persoalan mengevaluasi pohon ekspresi
4. Persoalan pengurutan (*sorting*)
5. Teorema Master
6. Mencari pasangan titik terdekat (*Closest Pair*)
7. Perkalian matriks
8. Perkalian bilangan bulat yang besar
9. Perkalian polinom
10. *Convex Hull*

BAB III

PENJELASAN ALGORITMA *DIVIDE AND CONQUER* & SOURCE PROGRAM YANG DIBUAT

I. Penjelasan Algoritma Penyelesaian Masalah

Untuk tahapan dari penyelesaian masalah *Closest Pair* dengan algoritma *Divide and Conquer* adalah sebagai berikut:

1. Mengurutkan tiap titik koordinat menaik berdasarkan nilai absisnya (pra-proses)
2. Menghitung basis untuk kondisi titik koordinat bernilai dua atau tiga secara langsung dengan rumus *Euclidean*
3. Jika titik ada lebih dari tiga, melakukan proses pembagian titik menjadi himpunan dua bagian dengan jumlah titik yang sama, pembagian kedua titik ini dibatasi oleh sebuah garis imaginasi
4. Melakukan proses penyelesaian permasalahan secara rekursif dengan algoritma *Divide and Conquer* untuk mencari sepasang titik terdekat
5. Jika kondisi titik terdekat berada dalam bagian kiri atau kanan garis imaginasi, solusi ditemukan
6. Ada kemungkinan titik terdekat berada pada posisi yang dipisahkan oleh garis imaginer. Jika demikian, membuat daerah strip sebesar dua kali nilai minimal dari solusi langkah 5 dan menentukan koordinat-koordinat yang termasuk didalam daerah strip
7. Untuk koordinat yang ada dalam daerah strip, urutkan berdasarkan nilai ordinatnya
8. Menghitung jarak yang ada dalam strip dan membandingkannya dengan solusi yang didapat dari langkah 5

II. Penjelasan Algoritma Program yang Dibuat

Pada program yang saya buat, program terdiri dari beberapa file yang tersusun atas satu file utama (*main file*) yaitu main.py dan terdapat beberapa file yang terdiri dari berbagai macam fungsi untuk menjalankan program utamanya antara lain splash.py, inputting.py, projection.py, dan processing.py.

a) Splash.py

Program pada file ini berfungsi untuk menampilkan splash screen saat *opening* dan *closing* dari program yang dijalankan. Deskripsi dari fungsi yang terdapat dalam file ini antara lain:

Nama Fungsi/Prosedur	Deskripsi
def openingSplash()	Splash Screen untuk ditampilkan saat membuka aplikasi
def closingSplash()	Splash Screen untuk ditampilkan saat menutup aplikasi

b) Inputting.py

Program pada file ini berfungsi untuk melakukan proses input jumlah titik, ukuran dimensi, dan juga letak koordinat titik-titik yang akan dicari jarak terdekatnya. Pada program input ini, untuk input titik, hanya dibatasi antara 2 hingga 1000 titik. Untuk input dimensi juga hanya terdapat opsi untuk dua dan tiga dimensi saja. Pencarian titik koordinat dilakukan dengan menggunakan pembangkit bilangan acak.

Deskripsi dari fungsi yang terdapat dalam file ini antara lain:

Nama Fungsi/Prosedur	Deskripsi
def inputTitik(n)	Fungsi untuk melakukan input jumlah titik yang akan dibandingkan
def inputDimension(n)	Fungsi untuk melakukan input dimensi yang ingin dipilih
def inputKoordinat(n, dimensi)	Fungsi untuk melakukan input koordinat secara acak (bisa untuk satu hingga tiga dimensi)

```python
import numpy as np
from numpy import random

# Melakukan input jumlah titik yang ingin dihitung
def inputTitik (n):
    n = int(input("Masukkan jumlah titik (2-1000): "))
    while (n < 2 or n > 1000):
        n = int(input("Masukkan jumlah titik (2-1000): "))
    return n

# Melakukan input dimensi
def inputDimension (n):
    print("Opsi Pilihan:")
    print("1) Mencari titik pada 1 dimensi")
    print("2) Mencari titik pada 2 dimensi")
    print("3) Mencari titik pada 3 dimensi")
    n = int(input("Masukkan pilihan dimensi: "))
    while (n != 1 and n != 2 and n != 3):
        n = int(input("Masukkan pilihan dimensi: "))
    return n

```

```python
# Melakukan input koordinat dengan pembangkit bilangan acak
def inputKoordinat(n,dimensi):
    coordinates = []
    for i in range (n):
        if (dimensi == 1):
            x = random.randint(1,1000)
            coordinates.append([x])
        elif (dimensi == 2):
            x = random.randint(1,250)
            y = random.randint(1,250)
            coordinates.append([x,y])
        elif (dimensi == 3):
            x = random.randint(1,250)
            y = random.randint(1,250)
            z = random.randint(1,250)
            coordinates.append([x,y,z])
    return coordinates
```

c) Projection.py

Program pada file ini berfungsi untuk memproses koordinat yang sudah diinput sebelumnya untuk diproyeksikan kedalam bentuk gambar visual.

Deskripsi dari fungsi yang terdapat dalam file ini antara lain:

Nama Fungsi/Prosedur	Deskripsi
def projectionBefore (listKoordinat,dimensi)	Fungsi untuk menentukan fungsi proyeksi mana yang akan dijalankan berdasarkan dimensinya (menentukan fungsi yang sebelum mencari titik terdekat)
def projectionAfter(listKoordinat, listKoordinatTerdekat, dimensi)	Fungsi untuk menentukan fungsi proyeksi mana yang akan dijalankan berdasarkan dimensinya (menentukan fungsi setelah ditemukan titik terdekat)
def projectionBefore1D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat sebelum pencarian jarak titik terdekat (tampilan dalam 1D)
def projectionBefore2D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat sebelum pencarian jarak titik terdekat (tampilan dalam 2D)
def projectionBefore3D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat sebelum pencarian jarak titik terdekat (tampilan dalam 3D)
def projectionAfter1D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat setelah pencarian jarak titik terdekat beserta dengan dua titik terdekatnya (tampilan dalam 1D)
def projectionAfter2D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat setelah pencarian jarak titik terdekat beserta dengan dua titik terdekatnya (tampilan dalam 2D)
def projectionAfter3D (listKoordinat)	Fungsi untuk memproyeksikan tampilan titik koordinat setelah pencarian jarak titik terdekat

	beserta dengan dua titik terdekatnya (tampilan dalam 3D)

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

# Fungsi untuk menampilkan proyeksi awal
def projectionBefore(listKoordinat, dimensi):
    if dimensi == 1:
        projectionBefore1D(listKoordinat)
    elif dimensi == 2:
        projectionBefore2D(listKoordinat)
    elif dimensi == 3:
        projectionBefore3D(listKoordinat)

# Fungsi untuk menampilkan proyeksi jarak titik terdekat
def projectionAfter(listKoordinat, listKoordinatTerdekat, dimensi):
    if dimensi == 1:
        projectionAfter1D(listKoordinat, listKoordinatTerdekat)
    elif dimensi == 2:
        projectionAfter2D(listKoordinat, listKoordinatTerdekat)
    elif dimensi == 3:
        projectionAfter3D(listKoordinat, listKoordinatTerdekat)

```

```python
# Fungsi proyeksi awal 1D
def projectionBefore1D(listKoordinat):
    # Data untuk proyeksi 1D
    for i in range (len(listKoordinat)):
        plt.scatter(listKoordinat[i][0], 0, color='red')
    print("Menampilkan Proyeksi Titik Koordinat Awal...")
    plt.show()

```

```python
# Fungsi proyeksi awal 2D
def projectionBefore2D(listKoordinat):
    # Data untuk proyeksi 2D
    for i in range (len(listKoordinat)):
        plt.scatter(listKoordinat[i][0], listKoordinat[i][1], color='red')
    print("Menampilkan Proyeksi Titik Koordinat Awal...")
    plt.show()

# Fungsi proyeksi awal 3D
def projectionBefore3D(listKoordinat):
    axes = plt.axes(projection='3d')

    # Data untuk proyeksi 3D
    for i in range (len(listKoordinat)):
        np.linspace(listKoordinat[i][0], listKoordinat[i][1], listKoordinat[i][2])

        # Plotting titik-titik
        axes.scatter3D(listKoordinat[i][0], listKoordinat[i][1], listKoordinat[i][2], color='red'

    print("Menampilkan Proyeksi Titik Koordinat Awal...")
    plt.show()

```

```python
# Fungsi proyeksi jarak titik terdekat 1D
def projectionAfter1D(listKoordinat, listKoordinatTerdekat):
    for i in range (len(listKoordinat)):
        plt.scatter(listKoordinat[i][0], 0, color='red')

    for i in range (len(listKoordinatTerdekat)):
        plt.scatter(listKoordinatTerdekat[i][0], 0, color='blue')

    print("Menampilkan Proyeksi Titik Koordinat Terdekat...")
    print("============================================================")
    plt.show()

# Fungsi proyeksi jarak titik terdekat 2D
def projectionAfter2D(listKoordinat, listKoordinatTerdekat):
    for i in range (len(listKoordinat)):
        plt.scatter(listKoordinat[i][0], listKoordinat[i][1], color='red')

    for i in range (len(listKoordinatTerdekat)):
        plt.scatter(listKoordinatTerdekat[i][0], listKoordinatTerdekat[i][1], color='blue')

    print("Menampilkan Proyeksi Titik Koordinat Terdekat...")
    print("============================================================")
    plt.show()

# Fungsi proyeksi jarak titik terdekat 3D
def projectionAfter3D(listKoordinat, listKoordinatTerdekat):
    axes = plt.axes(projection='3d')

    # Data untuk proyeksi 3D
    for i in range (len(listKoordinat)):
        np.linspace(listKoordinat[i][0], listKoordinat[i][1], listKoordinat[i][2])

        # Plotting titik-titik
        axes.scatter3D(listKoordinat[i][0], listKoordinat[i][1], listKoordinat[i][2], color='red')

    # Plotting garis antara titik terdekat
    for i in range (len(listKoordinatTerdekat)):
        axes.scatter3D(listKoordinatTerdekat[i][0], listKoordinatTerdekat[i][1], listKoordinatTerdekat[i][2], color='blue')

    print("Menampilkan Proyeksi Titik Koordinat Terdekat...")
    print("============================================================")
    plt.show()
```

d) Processing.py

Program pada file ini berfungsi untuk melakukan proses pencarian titik terdekat. Beberapa fungsi yang terdapat dalam program ini antara lain fungsi untuk melakukan *sorting* terurut menaik berdasarkan koordinat titik. Lalu ada fungsi untuk melakukan *brute force* dan juga untuk *divide and conquer*. Selain itu, ada juga fungsi untuk menampilkan spesifikasi *device* yang dipergunakan untuk melakukan *running* program.

Deskripsi dari fungsi yang terdapat dalam file ini antara lain:

Nama Fungsi/Prosedur	Deskripsi
def sortingX(listKoordinat)	Fungsi untuk melakukan *sorting* koordinat terurut berdasarkan nilai x (absis)
def sortingY(listKoordinat)	Fungsi untuk melakukan *sorting* koordinat terurut berdasarkan nilai y (ordinat)
def bruteForce (listKoordinat,dimensi)	Fungsi untuk melakukan pencarian titik terdekat dengan algoritma *brute force*

def divideAndConquer (listKoordinat,n,dimensi)	Fungsi untuk melakukan pencarian titik terdekat dengan algoritma *divide and conquer*
def spekLaptop()	Fungsi untuk menampilkan spesifikasi laptop yang akan digunakan untuk *running* program

```python
import math
import psutil
import platform

# Sorting untuk mengurutkan koordinat berdasarkan x menaik
def sortingX(listKoordinat):
    for i in range (len(listKoordinat)):
        for j in range (len(listKoordinat)):
            if listKoordinat[i][0] < listKoordinat[j][0]:
                listKoordinat[i], listKoordinat[j] = listKoordinat[j], listKoordinat[i]
    return listKoordinat

# Sorting untuk mengurutkan koordinat berdasarkan y menaik
def sortingY(listKoordinat):
    for i in range (len(listKoordinat)):
        for j in range (len(listKoordinat)):
            if listKoordinat[i][1] < listKoordinat[j][1]:
                listKoordinat[i], listKoordinat[j] = listKoordinat[j], listKoordinat[i]
    return listKoordinat

```

```python
#inisialisasi variabel global
hitungEuclideanBF = 0

# Melakukan proses brute force untuk mencari jarak terdekat
def bruteForce(listKoordinat,dimensi):
    global hitungEuclideanBF
    jarak = []
    min = float("inf")
    for i in range (len(listKoordinat)):
        for j in range (len(listKoordinat)):
            if i != j:
```

```python
                if dimensi == 1:
                    jarak.append(math.sqrt((pow((listKoordinat[j][0]-listKoordinat[i][0]),2))))
                    hitungEuclideanBF += 1
                elif dimensi == 2:
                    jarak.append(math.sqrt((pow((listKoordinat[j][0]-listKoordinat[i][0]),2))+
                                           (pow((listKoordinat[j][1]-listKoordinat[i][1]),2))))
                    hitungEuclideanBF += 1
                elif dimensi == 3:
                    jarak.append(math.sqrt((pow((listKoordinat[j][0]-listKoordinat[i][0]),2))+
                                           (pow((listKoordinat[j][1]-listKoordinat[i][1]),2))+
                                           (pow((listKoordinat[j][2]-listKoordinat[i][2]),2))))
                    hitungEuclideanBF += 1
    for i in range (len(jarak)):
        if jarak[i] < min:
            min = jarak[i]
    return min

```

```python
#inisialisasi variabel global
hitungEuclideanDC = 0

# Melakukan proses divide and conquer untuk mencari jarak terdekat
def divideAndConquer(listKoordinat, n, dimensi):
    global hitungEuclideanDC
    n = len(listKoordinat)
    jarak = 0
    # Basis 1
    if n == 2:
        if dimensi == 1:
            jarak = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2)))
            hitungEuclideanDC += 1
            tempNearest = [listKoordinat[0], listKoordinat[1]]
            return jarak, tempNearest
        elif dimensi == 2:
            jarak = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2))+
                              (pow((listKoordinat[1][1]-listKoordinat[0][1]),2)))
            hitungEuclideanDC += 1
            tempNearest = [listKoordinat[0], listKoordinat[1]]
            return jarak, tempNearest
        elif dimensi == 3:
            jarak = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2))+
                              (pow((listKoordinat[1][1]-listKoordinat[0][1]),2))+
                              (pow((listKoordinat[1][2]-listKoordinat[0][2]),2)))
            hitungEuclideanDC += 1
            tempNearest = [listKoordinat[0], listKoordinat[1]]
            return jarak, tempNearest
```

```python
    # Basis 2
    elif n == 3:
        if dimensi == 1:
            jarak1 = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2)))
            jarak2 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[0][0]),2)))
            jarak3 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[1][0]),2)))
            if jarak1 < jarak2 and jarak1 < jarak3:
                hitungEuclideanDC += 1
                jarak = jarak1
                tempNearest = [listKoordinat[0], listKoordinat[1]]
            elif jarak2 < jarak1 and jarak2 < jarak3:
                hitungEuclideanDC += 1
                jarak = jarak2
                tempNearest = [listKoordinat[0], listKoordinat[2]]
            else:
                hitungEuclideanDC += 1
                jarak = jarak3
                tempNearest = [listKoordinat[1], listKoordinat[2]]
            return jarak, tempNearest
```

```
        elif dimensi == 2:
            jarak1 = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2))+
                               (pow((listKoordinat[1][1]-listKoordinat[0][1]),2)))
            jarak2 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[0][0]),2))+
                               (pow((listKoordinat[2][1]-listKoordinat[0][1]),2)))
            jarak3 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[1][0]),2))+
                               (pow((listKoordinat[2][1]-listKoordinat[1][1]),2)))
            if jarak1 < jarak2 and jarak1 < jarak3:
                hitungEuclideanDC += 1
                jarak = jarak1
                tempNearest = [listKoordinat[0], listKoordinat[1]]
            elif jarak2 < jarak1 and jarak2 < jarak3:
                hitungEuclideanDC += 1
                jarak = jarak2
                tempNearest = [listKoordinat[0], listKoordinat[2]]
            else:
                hitungEuclideanDC += 1
                jarak = jarak3
                tempNearest = [listKoordinat[1], listKoordinat[2]]
            return jarak, tempNearest
```

```
        elif dimensi == 3:
            jarak1 = math.sqrt((pow((listKoordinat[1][0]-listKoordinat[0][0]),2))+
                               (pow((listKoordinat[1][1]-listKoordinat[0][1]),2))+
                               (pow((listKoordinat[1][2]-listKoordinat[0][2]),2)))
            jarak2 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[0][0]),2))+
                               (pow((listKoordinat[2][1]-listKoordinat[0][1]),2))+
                               (pow((listKoordinat[2][2]-listKoordinat[0][2]),2)))
            jarak3 = math.sqrt((pow((listKoordinat[2][0]-listKoordinat[1][0]),2))+
                               (pow((listKoordinat[2][1]-listKoordinat[1][1]),2))+
                               (pow((listKoordinat[2][2]-listKoordinat[1][2]),2)))
            if jarak1 < jarak2 and jarak1 < jarak3:
                jarak = jarak1
                tempNearest = [listKoordinat[0], listKoordinat[1]]
                # print(tempNearest)
                hitungEuclideanDC += 1
            elif jarak2 < jarak1 and jarak2 < jarak3:
                jarak = jarak2
                tempNearest = [listKoordinat[0], listKoordinat[2]]
                # print(tempNearest)
                hitungEuclideanDC += 1
            else:
                jarak = jarak3
                tempNearest = [listKoordinat[1], listKoordinat[2]]
                # print(tempNearest)
                hitungEuclideanDC += 1
            return jarak, tempNearest
```

```
    # Rekurens
    else:
        # Membagi data menjadi 2 bagian
        middle = n//2
        left_area = []
        right_area = []
        for i in range (middle):
            left_area.append(listKoordinat[i])
        for i in range (middle, n):
            right_area.append(listKoordinat[i])

        # Mencari jarak terdekat di bagian kiri dan kanan secara rekursif
        jarak1, koordinat1 = divideAndConquer(left_area, n, dimensi)
        hitungEuclideanDC += 1
        jarak2, koordinat2 = divideAndConquer(right_area, n, dimensi)
        hitungEuclideanDC += 1

```

```
        # Mencari koordinat dua titik terdekat
        if jarak1 <= jarak2:
            jarak = jarak1
            tempNearest = koordinat1
        elif jarak1 > jarak2:
            jarak = jarak2
            tempNearest = koordinat2

        # Mencari jarak terdekat di antara kiri dan kanan jika jaraknya kurang dari strip (cek kondisi di tengah garis batas)
        strip = []
        for i in range (n):
            if abs(listKoordinat[i][0] - listKoordinat[middle][0]) < jarak:
                strip.append(listKoordinat[i])

        # Mencari jarak terdekat di antara titik-titik yang ada di dalam strip
        if dimensi == 1:
            for i in range (len(strip)):
                for j in range (i+1, len(strip)):
                    if (strip[j][0] - strip[i][0]) >= jarak and (strip[j][1] - strip[i][1]) >= jarak:
                        break
                    newJarak = math.sqrt((pow((strip[j][0]-strip[i][0]),2)))
                    if newJarak < jarak:
                        jarak = newJarak
                        tempNearest = [strip[i], strip[j]]
                        hitungEuclideanDC += 1
            return jarak, tempNearest

        elif dimensi == 2 or dimensi == 3:
            # Sorting strip berdasarkan koordinat y menaik
            strip_sorted = sortingY(strip)

            # Mencari jarak terdekat di antara titik-titik yang ada di dalam strip
            for i in range (len(strip_sorted)):
                for j in range (i+1, len(strip_sorted)):
                    if (strip_sorted[j][0] - strip_sorted[i][0]) >= jarak and (strip_sorted[j][1] - strip_sorted[i][1]) >= jarak:
                        break
                    if dimensi == 2:
                        newJarak = math.sqrt((pow((strip_sorted[j][0]-strip_sorted[i][0]),2))+
                                             (pow((strip_sorted[j][1]-strip_sorted[i][1]),2)))
                    elif dimensi == 3:
                        newJarak = math.sqrt((pow((strip_sorted[j][0]-strip_sorted[i][0]),2))+
                                             (pow((strip_sorted[j][1]-strip_sorted[i][1]),2))+
                                             (pow((strip_sorted[j][2]-strip_sorted[i][2]),2)))
                    if newJarak < jarak:
                        jarak = newJarak
                        tempNearest = [strip_sorted[i], strip_sorted[j]]
                        hitungEuclideanDC += 1
            return jarak, tempNearest

    # Menampilkan spesifikasi laptop untuk running program
    def spekLaptop():
        print("============================================================")
        print("SPESIFIKASI LAPTOP UNTUK RUNNING PROGRAM")
        print("------------------------------------------------------------")
        print("Processor:", platform.processor())
        print("Operating System:", platform.platform())
        print("Machine:", platform.machine())
        total_ram = psutil.virtual_memory().total / (1024.0 **3)
        print("RAM:",total_ram, "GB")
        print("============================================================")
```

e) Main.py

Program ini merupakan program utama yang berfungsi untuk menjalankan program-program yang telah dibuat pada file lainnya. Dalam program main ini, terdapat satu fungsi yaitu def mainSolvingProblem yang menerima dimensi dan koordinat. Fungsi ini bertujuan untuk menjalankan fungsi-fungsi yang telah dibuat pada *file* lain sebelumnya. Fungsi-fungsi lain yang diimplementasikan dalam fungsi di *main* ini adalah sebegaia berikut:

1) Fungsi untuk menampilkan proyeksi titik-titik koordinat, semua titik memiliki warna merah
2) Fungsi untuk menampilkan spesifikasi laptop yang digunakan untuk running program

3) Fungsi untuk melakukan pencarian dua titik koordinat terdekat dengan algoritma *brute force* dan algoritma *divide and conquer* serta menampilkan hasil jarak terdekat, berapa kali perhitungan jarak *Euclidean*, dan waktu yang diperlukan untuk *running* program
4) Fungsi untuk menampilkan proyeksi kedua titik yang memiliki jarak terdekat, untuk titik yang jaraknya terdekat akan berwarna biru, dan titik-titik lainnya akan berwarna merah

Untuk urutan proses main programnya adalah program akan menampilkan *splash screen* pembuka dan menu untuk menanyakan kepada *user* mengenai input jumlah titik dan juga memilih dimensi (pilihan antara 1D, 2D, atau 3D). Setelah itu, program akan menginput koordinat titik secara random dengan pembangkit bilangan acak. Jumlah titik yang diinput adalah sejumlah titik yang dimasukkan *user* sebelumnya. Setelah itu, program akan melakukan *sorting* titik secara terurut berdasarkan nilai absis pada koordinat titik. Setelahnya, program akan menjalankan fungsi def mainSolvingProblem dengan proses-proses yang sudah dijelaskan sebelumnya. Jika fungsi ini sudah selesai dijalankan, program akan selesai dan menampilkan *splash screen* penutup. Untuk programya sendiri adalah sebagai berikut:

```python
import inputting as inp
import processing as process
import projection as project
import splash as splash
import time

# Fungsi utama untuk mencari jarak terdekat antar titik dan menampilkan hasilnya
def mainSolvingProblem(dimensi, koordinatSorting):
    project.projectionBefore(koordinatSorting, dimensi)

    print("")

    process.spekLaptop()

    print("")

    # Melakukan proses brute force
    start_bf_time = time.time()
    jarak = process.bruteForce(koordinatSorting, dimensi)
    stop_bf_time = time.time()

    # Menampilkan hasil dengan Algoritma Brute Force
    print("============================================================")
    print("MENCARI DENGAN ALGORITMA BRUTE FORCE")
    print("------------------------------------------------------------")
    print("Jarak terdekat antar titik:", jarak)
    print("Banyaknya perhitungan operasi Euclidean:", process.hitungEuclideanBF)
    print("Waktu menjalankan program: %.7s detik" % (stop_bf_time - start_bf_time))
    print("============================================================")

    print("")

```

```
    # Melakukan proses divide and conquer
    start_bf_time = time.time()
    jarak, koordinat = process.divideAndConquer(koordinatSorting, len(koordinatSorting), dimensi)
    stop_bf_time = time.time()

    # Menampilkan hasil dengan Algoritma Divide and Conquer
    print("============================================================")
    print("MENCARI DENGAN ALGORITMA DIVIDE AND CONQUER")
    print("------------------------------------------------------------")
    print("Jarak terdekat antar titik:", jarak)
    print("Banyaknya perhitungan operasi Euclidean:", process.hitungEuclideanDC)
    print("Waktu menjalankan program: %.7s detik" % (stop_bf_time - start_bf_time))
    print("============================================================")

    print("")

    # Menampilkan koordinat titik terdekat dan proyeksinya
    print("============================================================")
    print("Koordinat titik terdekat: ", koordinat)
    project.projectionAfter(koordinatSorting, koordinat, dimensi)


# MAIN PROGRAM

# Opening Splash
splash.openingSplash()

# Inisialisasi variabel
process.hitungEuclideanBF = 0
process.hitungEuclideanDC = 0
n = 0

# Input jumlah titik
n = inp.inputTitik(n)

# Input dimensi
print("")
dimensi = inp.inputDimension(n)

# Input koordinat
listKoordinat = inp.inputKoordinat(n, dimensi)

# Melakukan proses sorting
koordinatSorting = process.sortingX(listKoordinat)

# Memanggil fungsi utama untuk mencari jarak terdekat antar titik dan menampilkan hasilnya
mainSolvingProblem(dimensi, koordinatSorting)

print("")

# Closing Splash
splash.closingSplash()
```

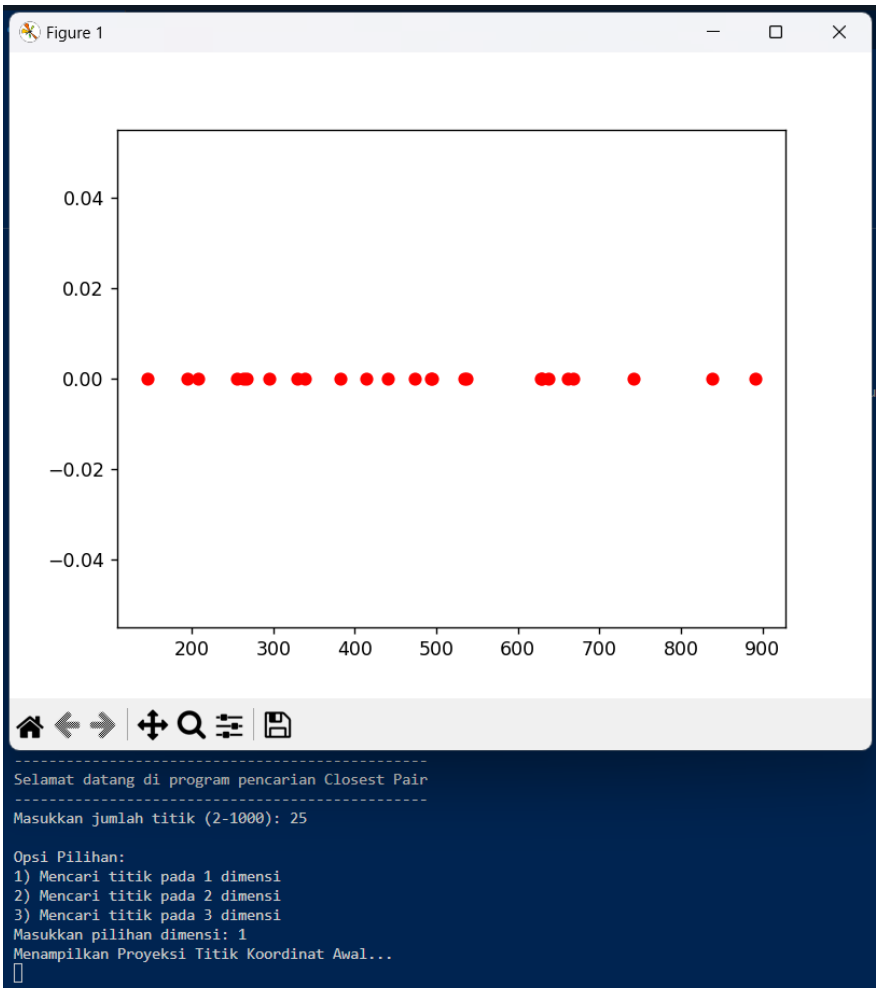
BAB IV

EKSPERIMEN

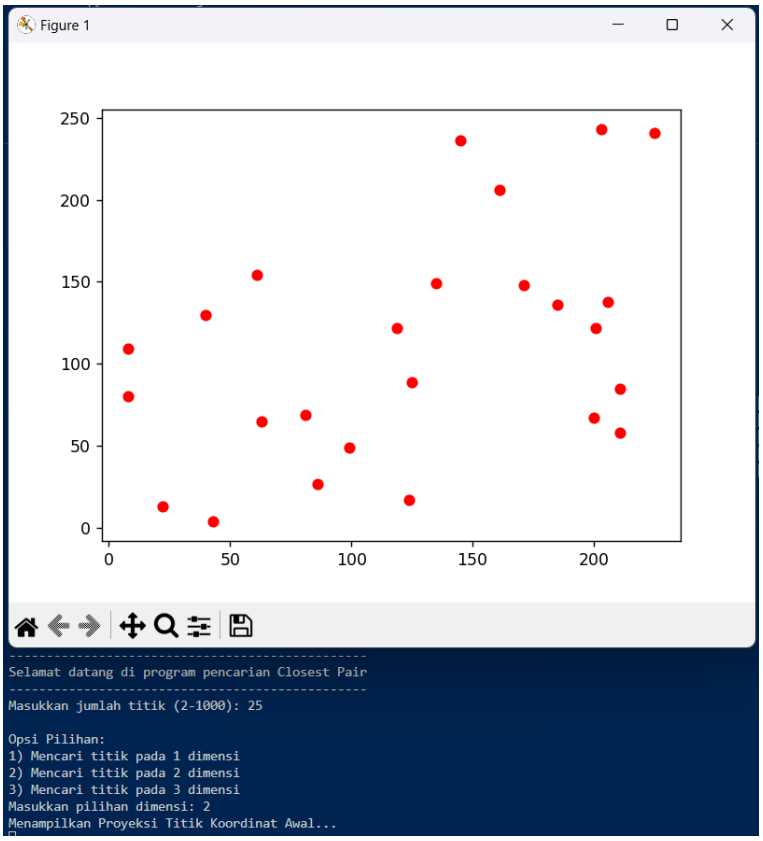
I. Hasil Running Program

a) Splash Screen Pembuka

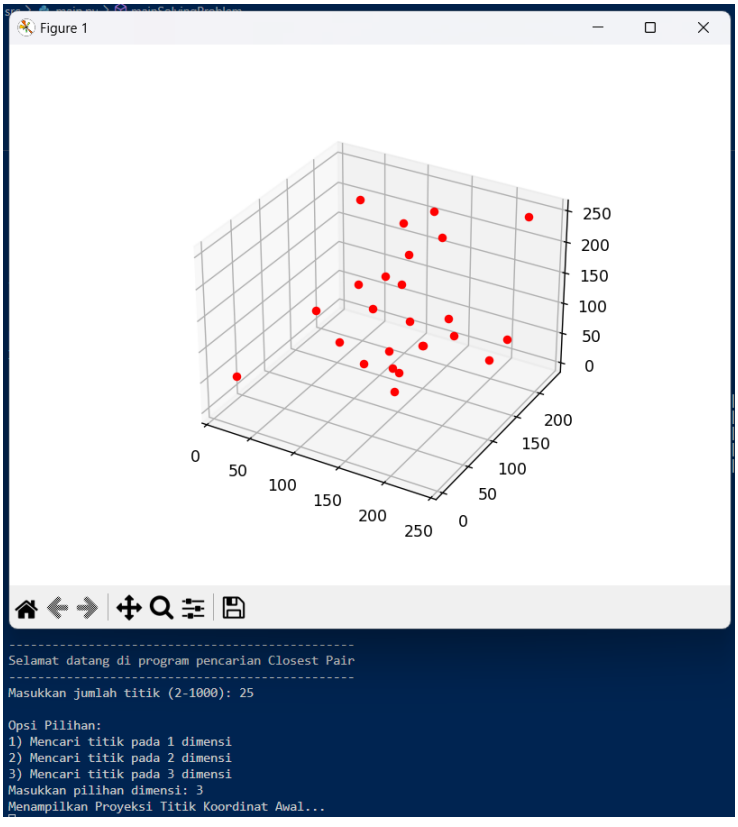
b) Memasukkan jumlah titik dan ukuran dimensi serta memproyeksikan titik-titik koordinat dalam bidang satu dimensi



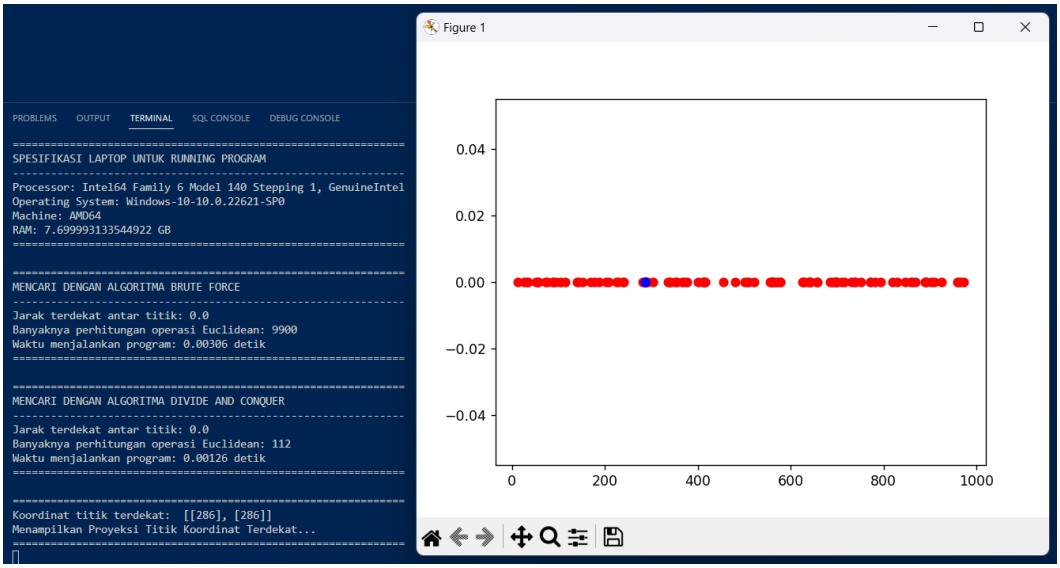
c) Memasukkan jumlah titik dan ukuran dimensi serta memproyeksikan titik-titik koordinat dalam bidang dua dimensi



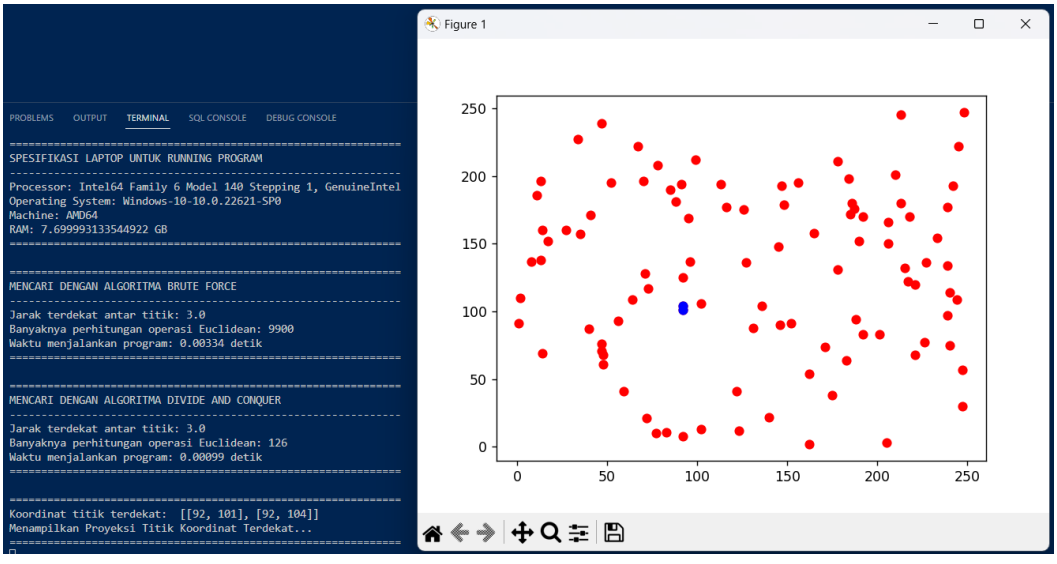
d) Memasukkan jumlah titik dan ukuran dimensi serta memproyeksikan titik-titik koordinat dalam bidang tiga dimensi



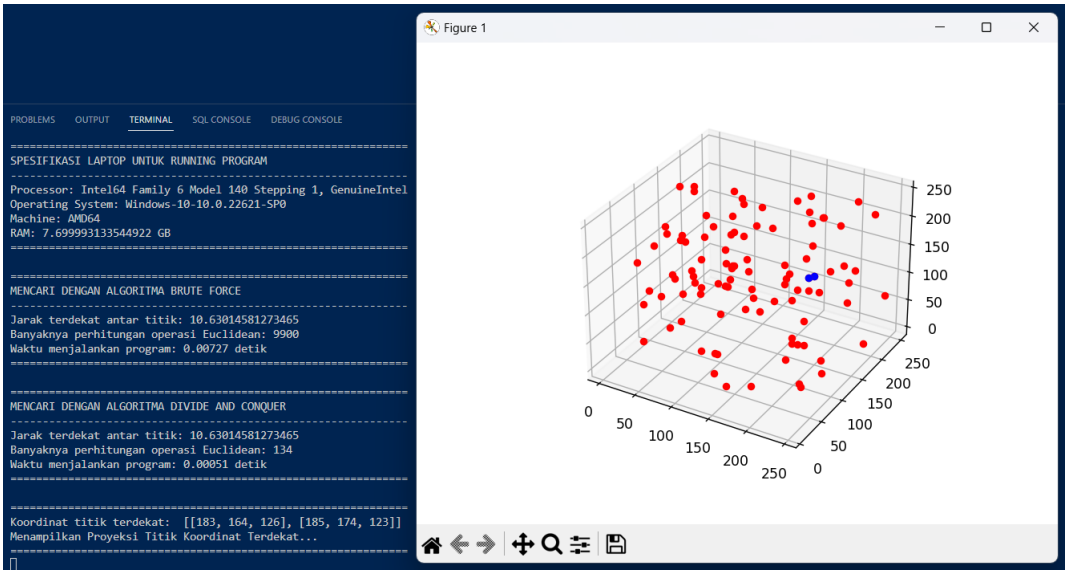
e) Menampilkan spesifikasi laptop untuk *running* program, hasil jarak terdekat dengan algoritma *brute force* dan algoritma *divide and conquer*, banyaknya perhitungan *euclidean*, waktu menjalankan program, koordinat titik terdekat, dan juga proyeksi untuk titik koordinat terdekat dalam bidang satu dimensi



f) Menampilkan spesifikasi laptop untuk *running* program, hasil jarak terdekat dengan algoritma *brute force* dan algoritma *divide and conquer*, banyaknya perhitungan *euclidean*, waktu menjalankan program, koordinat titik terdekat, dan juga proyeksi untuk titik koordinat terdekat dalam bidang dua dimensi



g) Menampilkan spesifikasi laptop untuk *running* program, hasil jarak terdekat dengan algoritma *brute force* dan algoritma *divide and conquer*, banyaknya perhitungan *euclidean*, waktu menjalankan program, koordinat titik terdekat, dan juga proyeksi untuk titik koordinat terdekat dalam bidang tiga dimensi

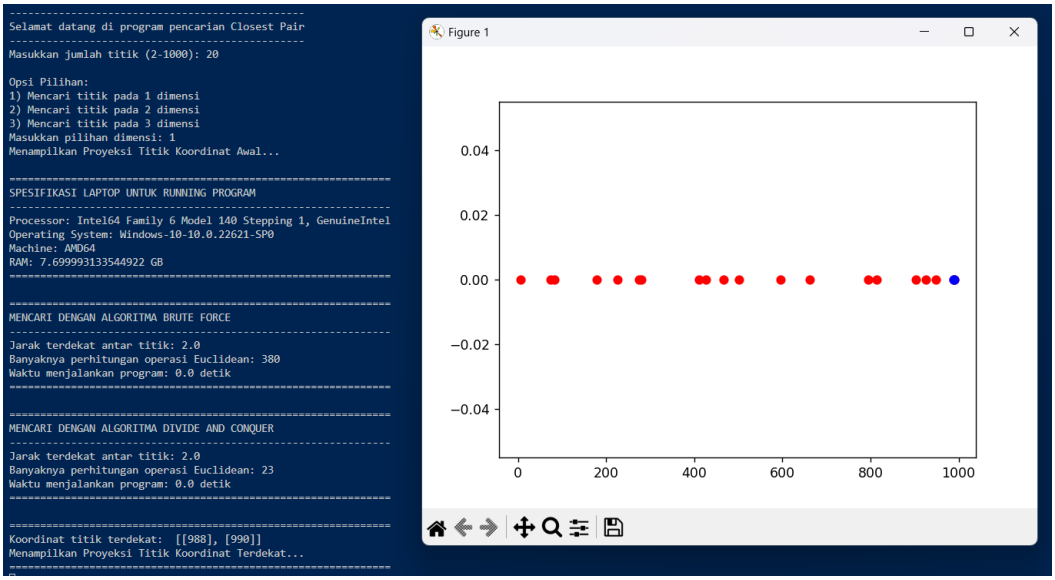


h) Menampilkan splash screen closing jika program sudah selesai

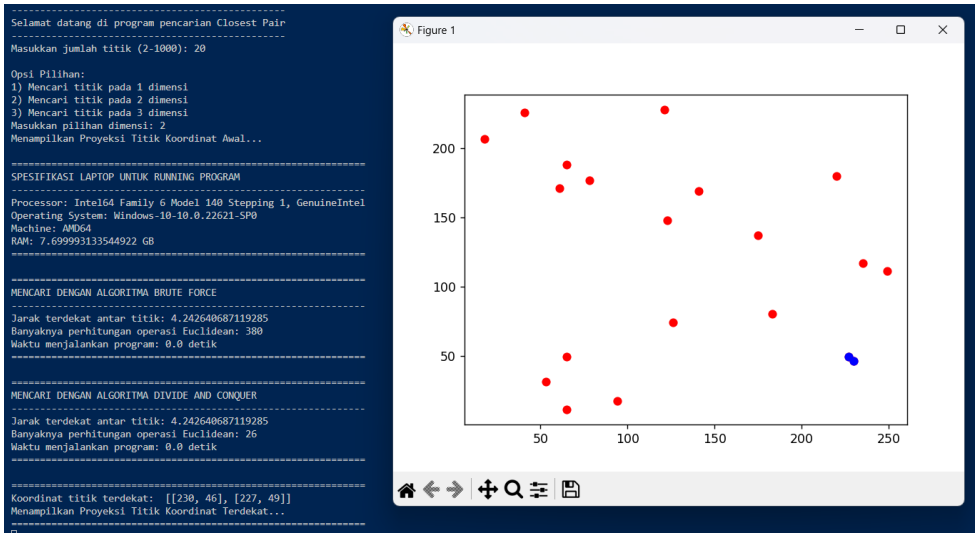
II. Uji Coba Hasil Program

a) Pengujian 20 titik

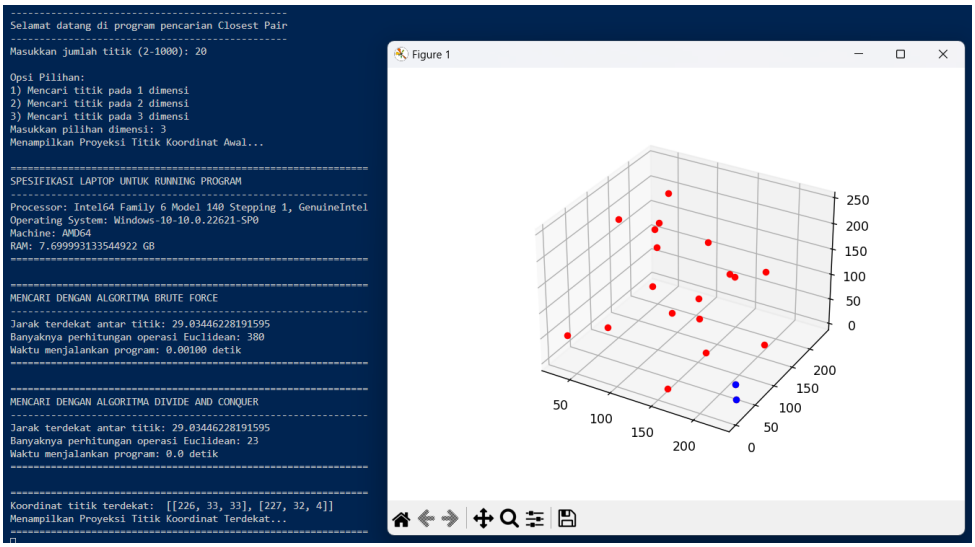
1) Bidang 1 dimensi



2) Bidang 2 dimensi

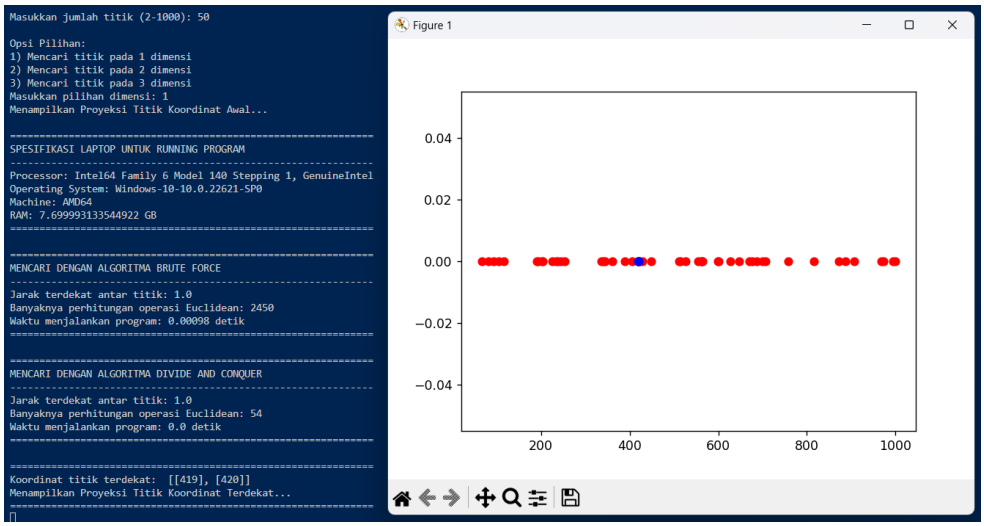


3) Bidang 3 dimensi

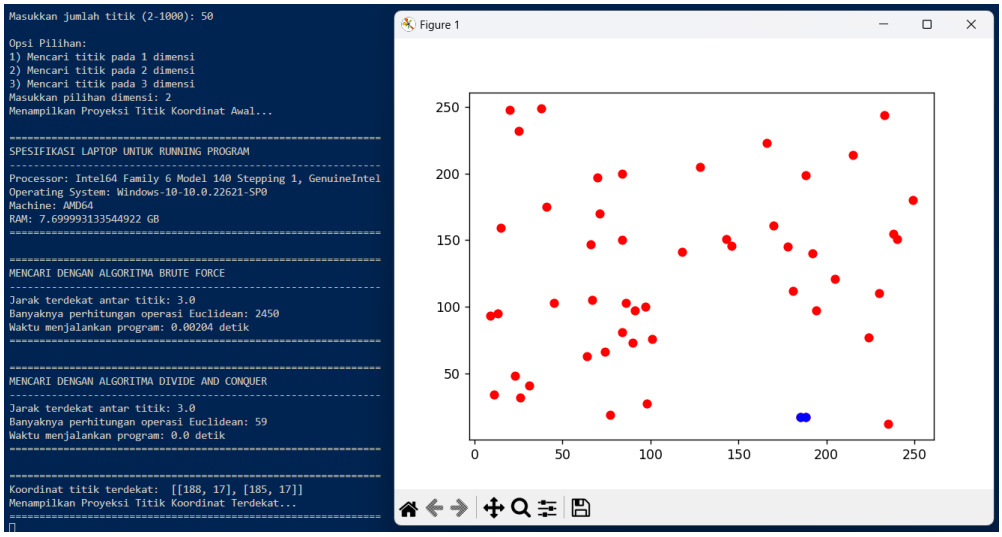


b) Pengujian 50 titik

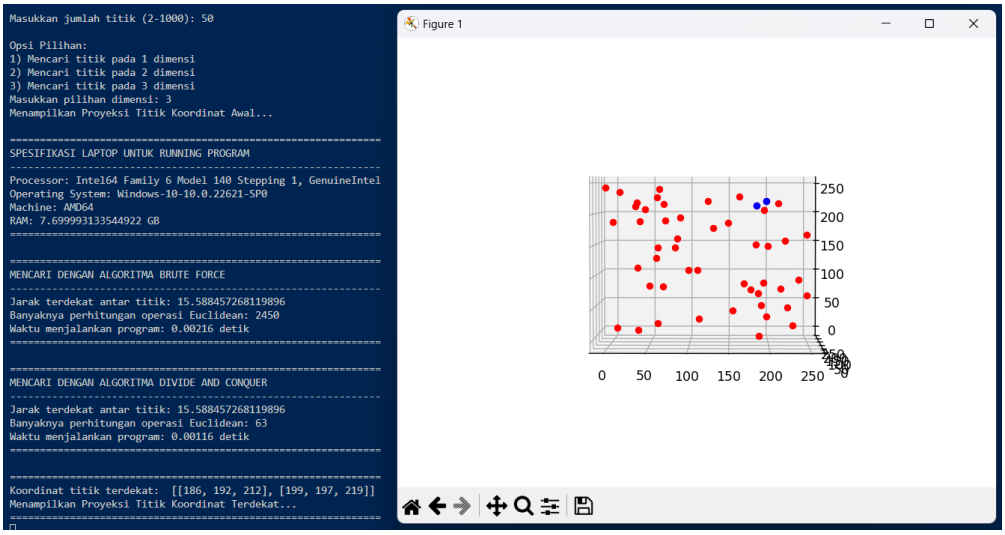
1) Bidang 1 dimensi



2) Bidang 2 dimensi

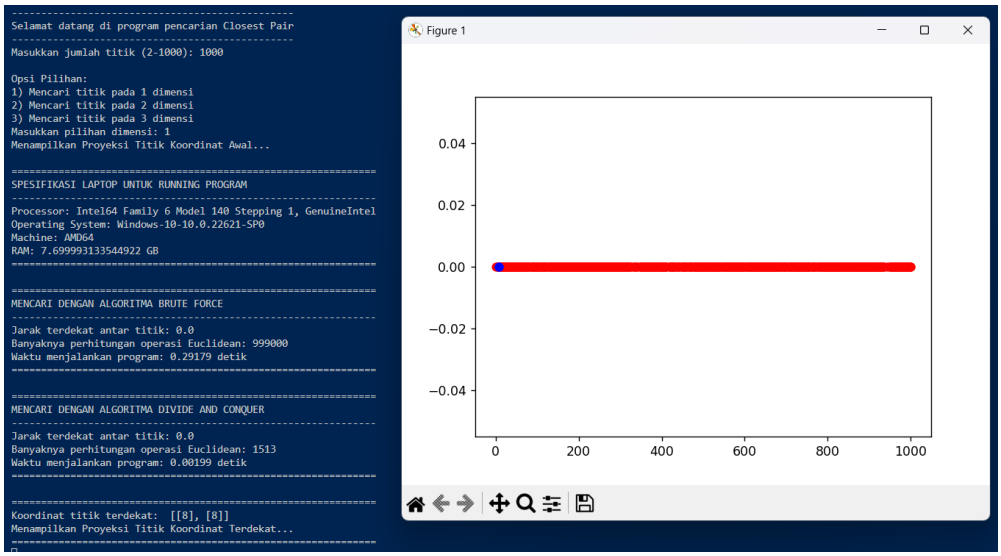


3) Bidang 3 dimensi

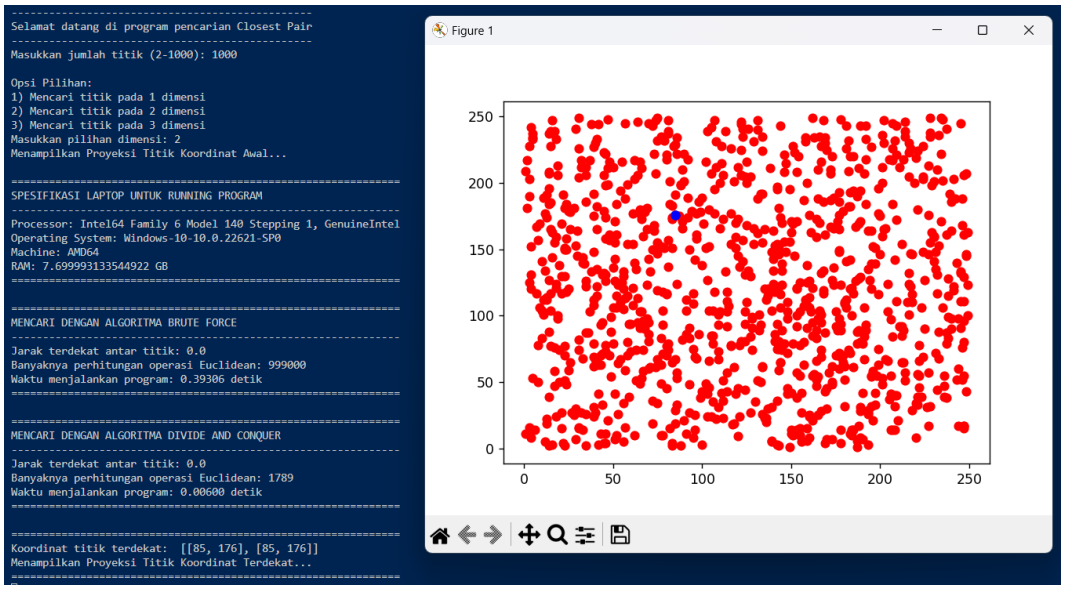


c) Pengujian 1000 titik

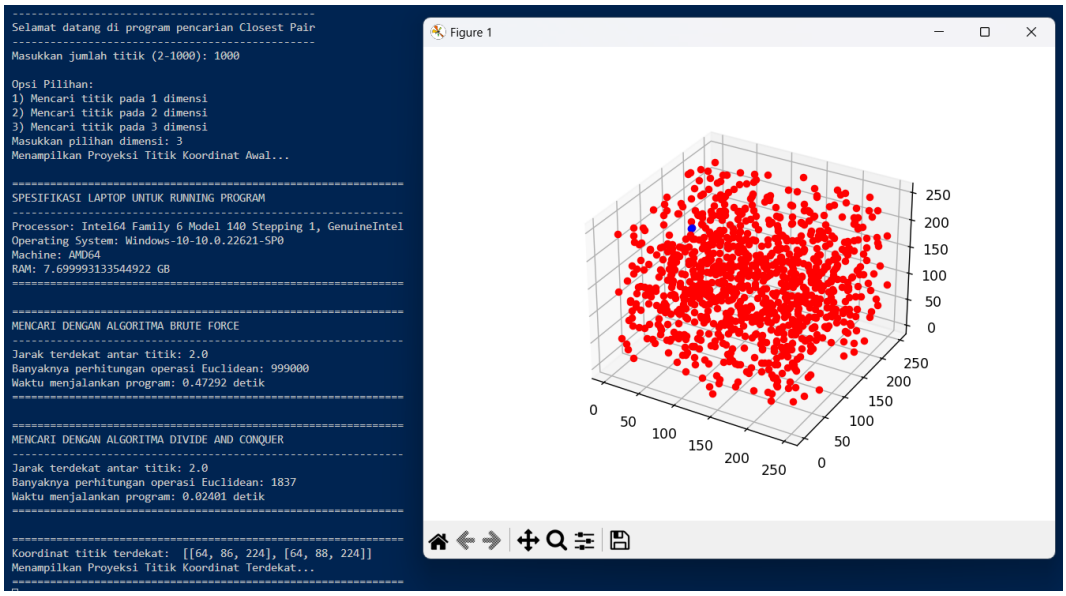
1) Bidang 1 dimensi



2) Bidang 2 dimensi



3) Bidang 3 dimensi



BAB V

KESIMPULAN

Kesimpulan yang didapat dari pengerjaan tugas kecil ini adalah sebagai berikut:

1) Algoritma *Divide and Conquer* dapat dipergunakan untuk menyelesaikan beberapa macam persoalan dan memiliki kompleksitas algoritma yang lebih mangkus jika dibandingkan dengan algoritma *Brute Force*
2) Persoalan pencarian titik terdekat (*closest pair*) dapat menggunakan algoritma *Divide and Conquer* untuk mencari solusi dari permasalahannya

REFERENSI

[1] R. Munir (2022). Algoritma Divide and Conquer Bagian 1 [Powerpoint Slides]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf

LAMPIRAN

Pranala Github: https://github.com/jasonrivalino/Tucil2_13521008

Checklist:

Poin	Judul Fitur	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil *running*	✓	
3	Program dapat membaca input / generate sendiri dan memberikan luaran	✓	
4	Luaran program sudah benar (solusi closest pair benar)	✓	
5	Bonus 1 dikerjakan	✓	
6	Bonus 2 dikerjakan	✓	

Note: untuk bonus 2, implementasi yang dibuat hanya bisa untuk mencari sepasang titik terdekat dalam bidang 1-3 dimensi