Tugas Kecil 2 IF2211 Strategi Algoritma

Semester II tahun 2022/2023

Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer

Disusun oleh :

Muhammad Zulfiansyah Bayu Pratama	13521028
Fahrian Afdholi	13521031

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

1. Deskripsi Masalah

Pada Tugas Kecil 2 kali ini, mahasiswa diharuskan 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:

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

Penulis membuat program dalam bahasa Python untuk mencari sepasang titik yang jaraknya terdekat datu sama lain dengan menerapkan algoritma *divide and conquer* untuk penyelesaiannya, dan perbandingannya dengan algoritma *brute force*.

Masukan dalam program adalah

- n, yaitu banyaknya titik yang akan dibentuk
- d, yaitu dimensi ruang
- titik-titik yang dibangkitkan secara acak

Luaran Program

- sepasang titik yang jaraknya terdekat dan nilai jaraknya
- banyaknya operasi perhitungan rumus Euclidian
- waktu riil dalam detik (spesifikasikan komputer yang digunakan)
- penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya.

2. Metode Penyelesaian

2.1. Penjelasan Algoritma *Divide and Conquer*

Divide and conquer adalah sebuah strategi algoritma dalam ilmu komputer yang melibatkan memecah masalah besar menjadi sub-masalah yang lebih kecil, yang kemudian dipecah lagi menjadi sub-masalah yang lebih kecil lagi, dan seterusnya, hingga dapat diselesaikan secara lebih efisien dan mudah.

Dalam strategi *divide and conquer*, masalah asal dibagi menjadi dua atau lebih sub-masalah yang serupa. Setiap sub-masalah kemudian diselesaikan secara independen dan secara rekursif hingga sub-masalah mencapai ukuran yang cukup kecil untuk dipecahkan dengan mudah. Setelah semua sub-masalah selesai, solusi untuk masalah asal dihasilkan dengan menggabungkan solusi dari setiap sub-masalah.

2.2. Penerapan Algoritma *Divide and Conquer*

Pada tugas kecil kali ini, penulis membuat dua ADT *(Abstract Data Type)* pada program. Yang pertama adalah ADT *Point* dengan atributnya adalah list koordinat. Lalu yang kedua adalah ADT *listPoint* yang akan menyimpan seluruh titik pada masukan. Di dalam ADT *listPoint* terdapat fungsi untuk mencari pasangan titik terdekat yang bernama $getClosestPointPairByDivideAndConquer()$. Berikut adalah tahapan dari algoritma yang terdapat pada fungsi tersebut :

- Masukkan seluruh *point* yang telah dibentuk ke dalam $listPoint$
- Cek kasus dasar atau *base case*. Jika ukuran list titik kurang dari 2, maka tidak mungkin ada pasangan titik terdekat, kembalikan None. Jika ukuran list titik sama dengan 2, maka hitung jarak antara kedua titik tersebut dan kembalikan hasilnya dalam bentuk tuple dengan urutan titik pertama, titik kedua, dan jumlah perhitungan jarak Euclidean yang dilakukan.
- Urutkan list titik berdasarkan koordinat x
- Bagi list titik menjadi dua bagian, yaitu $list1$ dan $list2$, dengan jumlah titik yang hampir sama.
- Lakukan rekursi dengan memanggil fungsi $getClosestPointPairByDivideAndConquer()$ untuk masing-masing bagian list, sehingga didapatkan pasangan titik terdekat pada kedua bagian tersebut.

- Bandingkan kedua pasangan titik terdekat tersebut. Jika salah satu pasangan kosong, kembalikan pasangan titik dari bagian yang tidak kosong. Jika keduanya tidak kosong, bandingkan jarak kedua pasangan titik tersebut dan ambil yang terkecil sebagai pasangan titik terdekat. Hitung jumlah perhitungan jarak Euclidean yang dilakukan.
- Dapatkan jarak terkecil $minDistance$ dari pasangan titik terdekat yang ditemukan pada langkah 5.
- Buat $list3$ untuk menampung titik-titik yang berada di antara kedua bagian list dan memiliki jarak terhadap garis tengah yang membagi list menjadi dua kurang dari atau sama dengan $minDistance$. Titik-titik ini akan dicek satu per satu apakah ada pasangan titik terdekat di antara mereka yang memiliki jarak kurang dari minDistance. Jika ada, *update* pasangan titik terdekat dengan pasangan baru tersebut. Hitung jumlah perhitungan jarak Euclidean yang dilakukan.
- Kembalikan hasil akhir dalam bentuk tuple dengan urutan titik pertama, titik kedua, dan jumlah perhitungan jarak Euclidean yang dilakukan.

3. *Souce Code* dengan Bahasa Python

3.1. *src/main.cpp*

```
import numpy as np
import matplotlib.pyplot as plt
import os
import time
import platform
import psutil

from ADT.point import Point
from ADT.listPoint import ListPoint

# Warna ANSI
red = '\033[91m'
green = '\033[92m'
yellow = '\033[93m'
blue = '\033[94m'
magenta = '\033[95m'
cyan = '\033[96m'
white = '\033[97m'


# Style ANSI
bold = '\033[1m'
underline = '\033[4m'
italic = '\033[3m'
reset = '\033[0m'

# Prosedur Membuat splashscreen


def splashScreen():
    os.system('cls')
    print(red)
    print('                                                                                ')
    print('                                                                                ')
    print('                                                                                ',
white)
    print('                                                                                ')
    print('                                                                                ')
    print('                                                                                ')
    print('                                                                                
')
    print('                                                                                
')
    print()
    print('Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and
Conquer')
    print('------------------------------------------------------------------
------')
    print(
        '|      ' + bold + 'Anggota' + reset + ' : • 13521028 - Muhammad
Zulfiansyah Bayu Pratama      |')
    print('|                 • 13521031 - Fahrian
Afdholi                        |')
    print('|_____________________________________________________________________
_____|')
    print()


def checkInputN(warning):
    # Prosedur untuk mengecek inputan N (jumlah titik yang akan dibuat)
    splashScreen()
    if (warning != ""):
```

```
        print(red + "WARNING:", warning, "\n")

    N = input(green + "Jumlah titik  : " + yellow)
    if (N.isdigit() == False):
        warning = "Input harus berupa bilangan bulat positif dan lebih dari
1"
        return checkInputN(warning)
    elif (int(N) <= 1):
        warning = "Input harus berupa bilangan bulat positif dan lebih dari
1"
        return checkInputN(warning)
    else:
        N = int(N)
        return N


def printSystemInfo():
    print(cyan + bold + "\nInformasi sistem" + reset)
    print(blue + "Nama sistem operasi  :" + white, platform.system())
    print(blue + "Versi sistem operasi :" + white, platform.release())
    print(blue + "Processor            :" + white, platform.processor())
    print(blue + "Jumlah core          :" + white, psutil.cpu_count())
    print(blue + "Jumlah thread        :" +
          white, psutil.cpu_count(logical=False))
    print(blue + "RAM                  :" + white,
          psutil.virtual_memory().total / 1024 / 1024, "MB")


def checkInputD(warning, N):
    splashScreen()
    if (warning != ""):
        print(red + "WARNING:", warning, "\n")

    print(green + "Jumlah titik  :" + yellow, N)
    # Prosedur untuk mengecek inputan D (dimensi)
    D = input(green + "Dimensi ruang : " + yellow)
    if (D.isdigit() == False):
        warning = "Dimensi harus berupa bilangan bulat positif"
        return checkInputD(warning, N)
    elif (int(D) <= 0):
        warning = "Dimensi harus berupa bilangan bulat positif"
        return checkInputD(warning, N)
    else:
        D = int(D)
        return D


def visualitation(listPoint, pair, D, text):
    # Beri warna merah pada warna titik terdekat, sedangkan titik lainnya
berwarna biru
    if (D == 2):
        # Plot 2D
        x = []
        y = []
        for i in range(N):
            if (listPoint.get(i) != pair[0] and listPoint.get(i) !=
pair[1]):
                x.append(listPoint.get(i).coordinates[0])
                y.append(listPoint.get(i).coordinates[1])

        plt.scatter(x, y, color='blue')
        plt.scatter([pair[0].coordinates[0], pair[1].coordinates[0]], [
                    pair[0].coordinates[1], pair[1].coordinates[1]],
color='red')
        # Berikan informasi kooridnat titik terdekat
        plt.text(pair[0].coordinates[0], pair[0].coordinates[1], str(
```

```
            pair[0].coordinates[0]) + "," + str(pair[0].coordinates[1]))
        plt.text(pair[1].coordinates[0], pair[1].coordinates[1], str(
            pair[1].coordinates[0]) + "," + str(pair[1].coordinates[1]))

        plt.title(text)
        plt.show()

    elif (D == 3):
        # Plot 3D
        x = []
        y = []
        z = []
        for i in range(N):
            # Jika titik bukan titik terdekat, maka beri warna biru
            if (listPoint.get(i) != pair[0] and listPoint.get(i) !=
pair[1]):
                x.append(listPoint.get(i).coordinates[0])
                y.append(listPoint.get(i).coordinates[1])
                z.append(listPoint.get(i).coordinates[2])

        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(x, y, z, color='blue')
        ax.scatter([pair[0].coordinates[0], pair[1].coordinates[0]], [
            pair[0].coordinates[1], pair[1].coordinates[1]],
[pair[0].coordinates[2], pair[1].coordinates[2]], color='red')

        # Berikan informasi kooridnat titik terdekat
        ax.text(pair[0].coordinates[0], pair[0].coordinates[1],
pair[0].coordinates[2], str(
            pair[0].coordinates[0]) + "," + str(pair[0].coordinates[1]) +
"," + str(pair[0].coordinates[2]))
        ax.text(pair[1].coordinates[0], pair[1].coordinates[1],
pair[1].coordinates[2], str(
            pair[1].coordinates[0]) + "," + str(pair[1].coordinates[1]) +
"," + str(pair[1].coordinates[2]))
        ax.set_title(text)
        plt.show()

    elif (D == 1):
        # Plot 1D
        x = []
        for i in range(N):
            x.append(listPoint.get(i).coordinates[0])

        plt.scatter(x, [0] * N, color='blue')
        plt.scatter([pair[0].coordinates[0], pair[1].coordinates[0]], [
                    0, 0], color='red')

        # Berikan informasi kooridnat titik terdekat
        plt.text(pair[0].coordinates[0], 0, str(
            pair[0].coordinates[0]))
        plt.text(pair[1].coordinates[0], 0, str(
            pair[1].coordinates[0]))

        plt.title(text)
        plt.show()

    print(reset)


# Main Program
if __name__ == "__main__":
    # Input berapa banyak titik yang akan dibuat
    N = checkInputN("")

    # Input dimensi
```

```
    D = checkInputD("", N)

    # Membuat list of point
    listPoint = ListPoint()

    # Membuat titik-titik secara random
    print(green + "\nTitik-titik yang dibuat secara random: ")
    for i in range(N):
        coordinates = []
        for j in range(D):
            coordinates.append(np.random.randint(0, 100))
        listPoint.add(Point(coordinates))
        print(yellow + "Titik", i+1, ":" + white, listPoint.get(i))

    # Mencari pasangan titik terdekat
    timeStart = time.time()
    pair = listPoint.getClosestPointPairByDivideAndConquer()
    timeNow = time.time() - timeStart
    printSystemInfo()
    # Mencetak pasangan titik terdekat

    print(magenta + bold + "\nAlgoritma Divide and Conquer" + reset)
    print(blue + "Pasangan titik terdekat          :", white, end='')
    print("(" + str(pair[0]) + ", " + str(pair[1]) + ")")
    print(blue + "Jarak                            :" +
          white, pair[0].distance(pair[1]))
    print(blue + "Jumlah operasi perhitungan jarak :" + white, pair[2])
    print(blue + "Waktu eksekusi                   :" +
          white, timeNow, "detik\n")

    # Bandingkan dengan algoritma brute force
    print(magenta + bold + "Algoritma Brute Force" + reset)
    timeStart = time.time()
    pair2 = listPoint.getClosestPointPairByBruteForce()
    timeNow = time.time() - timeStart
    print(blue + "Pasangan titik terdekat          :", white, end='')
    print("(" + str(pair2[0]) + ", " + str(pair2[1]) + ")")
    print(blue + "Jarak                            :" +
          white, pair2[0].distance(pair2[1]))
    print(blue + "Jumlah operasi perhitungan jarak :" + white, pair2[2])
    print(blue + "Waktu eksekusi                   :" +
          white, timeNow, "detik\n")

    if (pair[0] == pair2[0] and pair[1] == pair2[1]):
        print(green + "Hasil sama!")
    elif (pair[0].distance(pair[1]) == pair2[0].distance(pair2[1])):
        print(yellow + "Hasil sama, namun urutan pasangan berbeda!")
    else:
        print(red + "Hasil berbeda!")

    # Membuat plot sesuai dengan dimensi
    visualitation(listPoint, pair, D, "Hasil Divide and Conquer")
    if (pair[0] != pair2[0] or pair[1] != pair2[1]):
        visualitation(listPoint, pair2, D, "Hasil Brute Force")
```

3.2. *src/ADT/point.py*

```
from math import sqrt

class Point:
    def __init__(self, coordinates):
        self.coordinates = coordinates

    def dimension(self):
        return len(self.coordinates)

    def distance(self, other):
```

```
        return sqrt(sum((x-y)**2 for x, y in zip(self.coordinates,
other.coordinates)))

    def __str__(self):
        return str(self.coordinates)
```

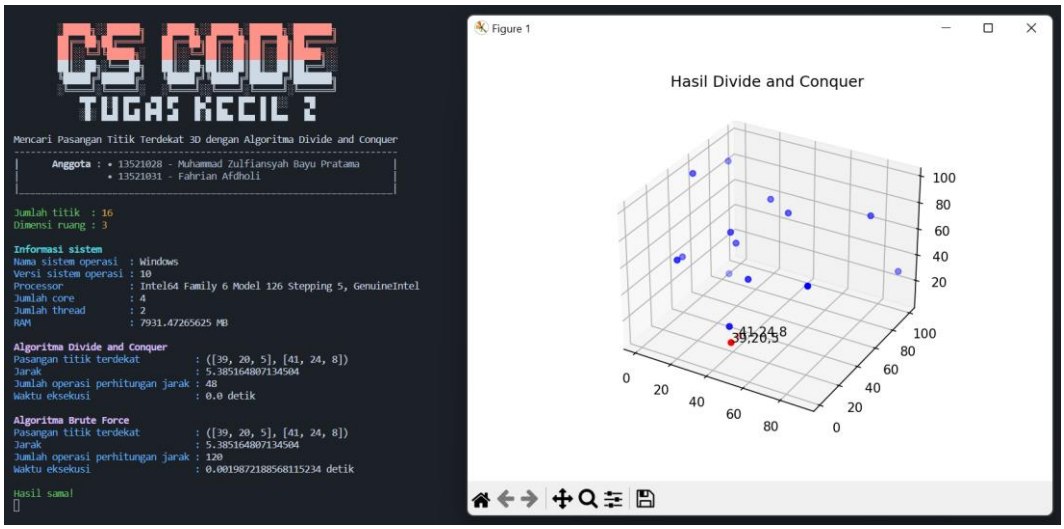
3.3. *src/ADT/listPoint.py*

```
# ADT List of Point

class ListPoint:
    def __init__(self):
        self.listPoint = []

    def dimension(self):
        return self.listPoint[0].dimension()

    def size(self):
        return len(self.listPoint)

    def get(self, index):
        return self.listPoint[index]

    def add(self, point):
        self.listPoint.append(point)

    def isEmpty(self):
        return self.size() == 0

    def getClosestPointPairByBruteForce(self):
        # Mencari pasangan titik terdekat dengan brute force
        countEucledian = 0
        minDistance = float('inf')
        closestPointPair = None
        N = self.size()
        for i in range(N):
            for j in range(i+1, N):
                countEucledian += 1
                distance = self.get(i).distance(self.get(j))
                if distance < minDistance:
                    minDistance = distance
                    closestPointPair = (self.get(i), self.get(j))

        closestPointPair = (
            closestPointPair[0], closestPointPair[1], countEucledian)
        return closestPointPair

    def getClosestPointPairByDivideAndConquer(self):
        countEucledian = 0
        # Base Case
        if self.isEmpty():
            return None

        elif self.size() == 1:
            return None

        elif self.size() == 2:
            return (self.get(0), self.get(1), 1)

        # Relasi Rekurens
        else:
            # Urutkan list berdasarkan koordinat x
            self.listPoint.sort(key=lambda point: point.coordinates[0])

            # Bagi list menjadi 2 bagian
```

```
            list1 = ListPoint()
            list2 = ListPoint()
            for i in range(self.size()):
                if i < self.size() // 2:
                    list1.add(self.get(i))
                else:
                    list2.add(self.get(i))

            # Cari pasangan titik terdekat dari kedua bagian
            pair1 = list1.getClosestPointPairByDivideAndConquer()
            pair2 = list2.getClosestPointPairByDivideAndConquer()

            closestPointPair = None

            # Lalu bandingkan dengan kedua bagian tersebut
            if pair1 == None:
                closestPointPair = pair2
                countEucledian += pair2[2]
            elif pair2 == None:
                closestPointPair = pair1
                countEucledian += pair1[2]
            else:
                countEucledian += pair1[2] + pair2[2]
                if pair1[0].distance(pair1[1]) <
pair2[0].distance(pair2[1]):
                    closestPointPair = pair1
                else:
                    closestPointPair = pair2

            minDistance = closestPointPair[0].distance(closestPointPair[1])

            # Proses Conquer
            x = self.get(self.size()//2).coordinates[0]
            list3 = ListPoint()
            for i in range(self.size()):
                if abs(self.get(i).coordinates[0] - x) < minDistance:
                    list3.add(self.get(i))

            # Mencari pasangan titik terdekat dari titik-titik yang berada
di antara kedua bagian
            if (list3.size() > 1):
                temp = list3.getClosestPointPairByBruteForce()
                countEucledian += temp[2]
                if temp[0].distance(temp[1]) < minDistance:
                    closestPointPair = temp

            closestPointPair = (
                closestPointPair[0], closestPointPair[1], countEucledian)
            return closestPointPair
```

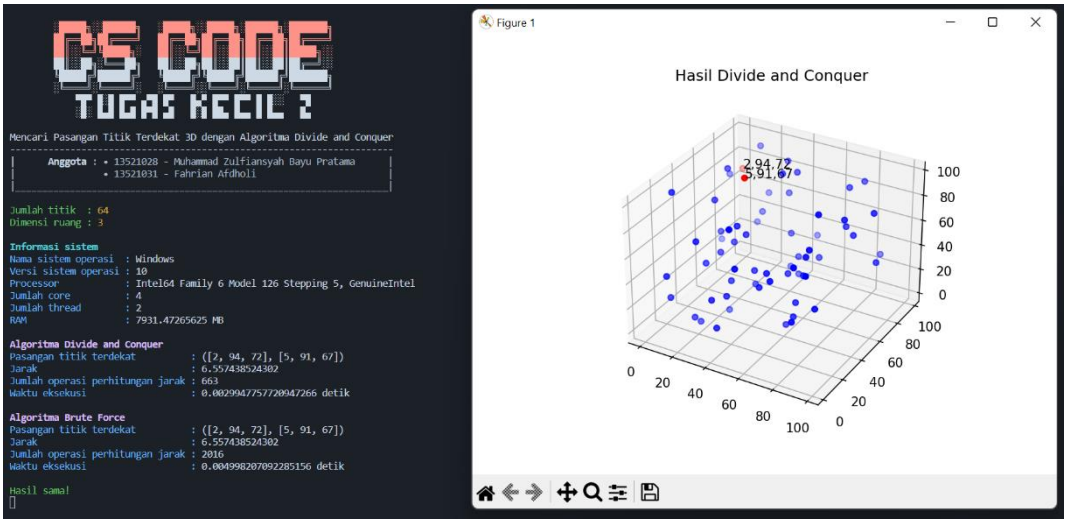
4. Ekperimen

4.1. Testcase #1 ($n = 16, d = 3$)



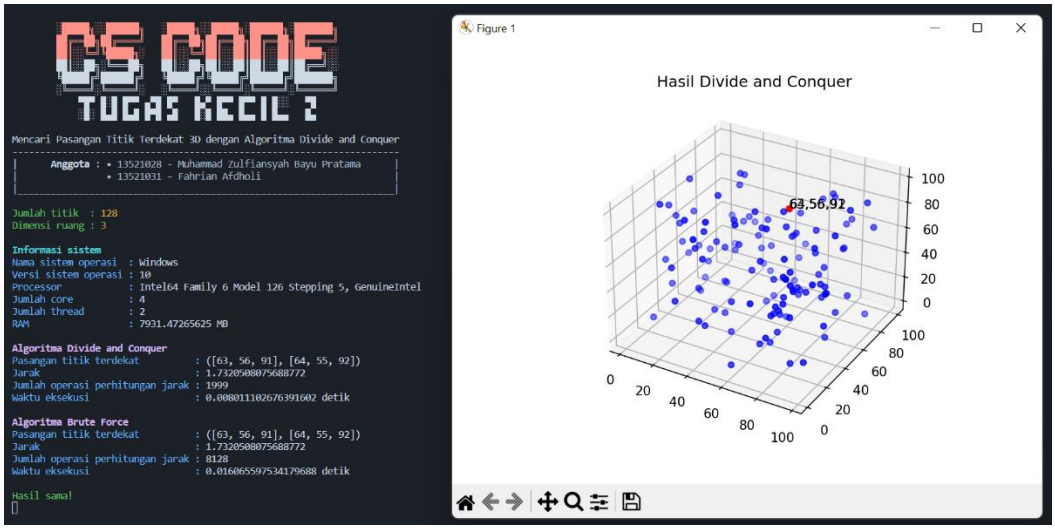
Gambar 4.1.1. *Screenshoot* program ($n = 16, d = 3$) dan *visualizer* dari *closest pair*

4.2. Testcase #2 ($n = 64, d = 3$)



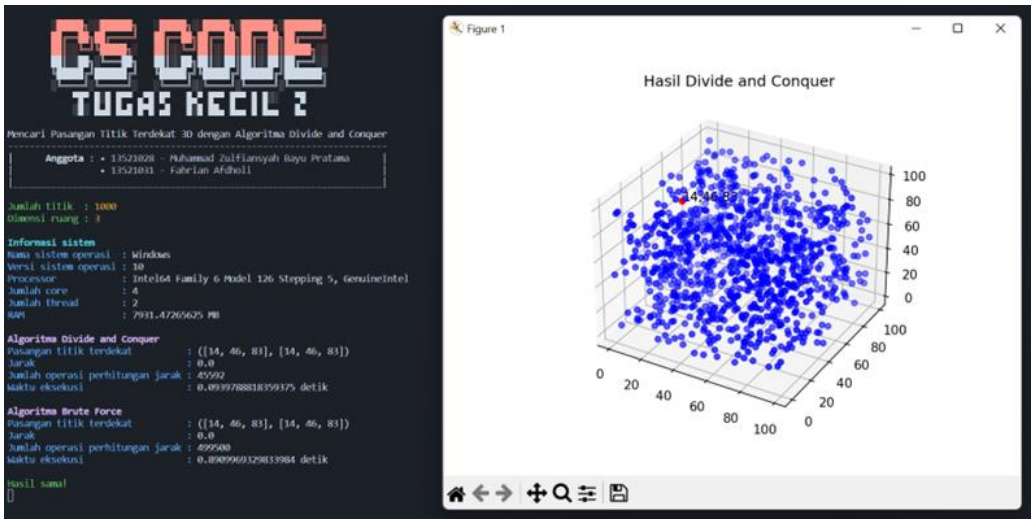
Gambar 4.2.1. *Screenshoot* program ($n = 64, d = 3$) dan *visualizer* dari *closest pair*

4.3. Testcase #3 ($n = 128, d = 3$)



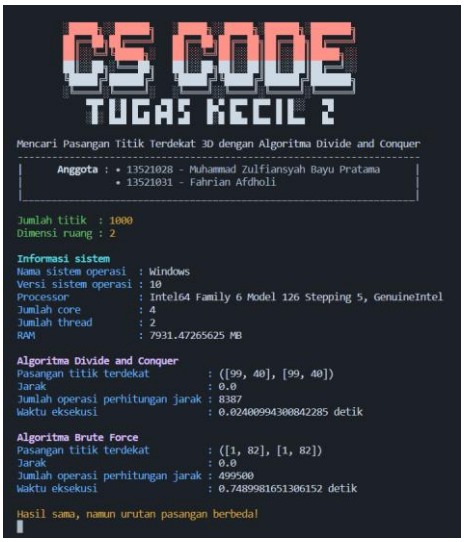
Gambar 4.3.1. *Screenshoot* program ($n = 120, d = 3$) dan *visualizer* dari *closest pair*

4.4. Testcase #4 ($n = 1000, d = 3$)

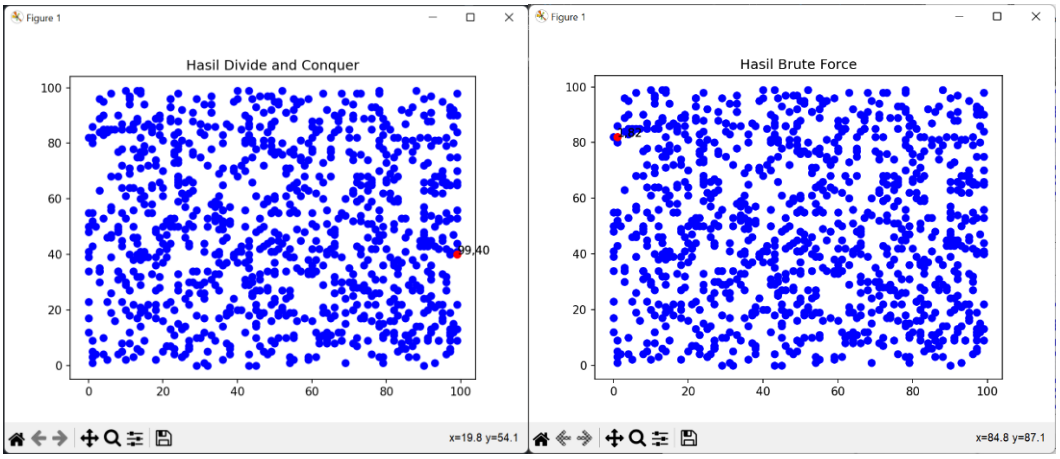


Gambar 4.4.1. *Screenshoot* program ($n = 1000, d = 3$) dan *visualizer* dari *closest pair*

4.5. Testcase #5 ($n = 1000, d = 2$)



Gambar 4.5.1. *Screenshoot* program ($n = 1000, d = 2$)



Gambar 4.5.2. *Screenshoot visualizer* dari *closest path* ($n = 1000, d = 2$)

4.6. Testcase #6 ($n = 1000, d = 4$)

Gambar 4.5.2. *Screenshoot visualizer* dari *closest path* ($n = 1000, d = 2$)

5. Link Github

https://github.com/zulfiansyah404/Tucil2_13521028_13521031

6. *Checklist*

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	√	
2. Program berhasil *running*	√	
3. Program dapat membaca input / generate sendiri dan memberikan luaran	√	
4. Solusi yang diberikan program memenuhi (solusi *closest pair* benar)	√	
5. Bonus 1 dikerjakan	√	
6. Bonus 2 dikerjakan	√	