Tugas Kecil II IF2211 Strategi Algoritma

Semester II Tahun 2022/2023

Penyelesaian *Closest Pair of Points Problem* dengan Algoritma *Divide and Conquer*

Disusun oleh :

Syarifa Dwi Purnamasari K03 / 13521018

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

1. Algoritma *Divide and Conquer*

Dalam algoritma *divide and conquer*, definisi *divide* ialah membagi persoalan menjadi beberapa subpersoalan yang memiliki kemiripan dengan persoalan awal dengan ukuran yang lebih kecil (idealnya berukuran hampir sama), sedangkan *conquer* berarti menyelesaikan (*solve*) masing-masing subpersoalan dengan rekursif jika masih berukuran besar dan secara langsung jika sudah berukuran kecil. Secara umum, algoritma *divide and conquer* merupakan suatu metode pemecahan masalah dengan cara memecah persoalan yang besar menjadi upapersoalan yang lebih kecil dan sederhana, kemudian menyelesaikan masing-masing upapersoalan tersebut, lalu menggabungkannya (*combine*) kembali menjadi algoritma yang utuh sehingga mampu membentuk solusi persoalan semula. Pada awalnya, algoritma ini merupakan strategi militer yang dikenal dengan nama *divide at impera* atau yang berarti pecah dan berkuasa.

Algoritma *divide and conquer* sering mengimplementasikan algoritma rekursif untuk menyelesaikan permasalahannya seperti yang telah banyak dijelaskan sebelumnya. Cara kerja algoritma rekursif ialah dengan memanggil dirinya sendiri sebagai solusi dari suatu *input* yang diberikan dan berhenti ketika *input* tersebut memenuhi suatu kondisi yang dikehendaki. Dikarenakan rekursivitas algoritmanya, kompleksitas waktu dari algoritma ini dapat dituliskan sebagai berikut.

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \ldots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

Gambar 1.1 Kompleksitas waktu algoritma *divide and conquer*

- $T(n)$: kompleksitas waktu penyelesaian persoalan *P* yang berukuran *n*
- $g(n)$: kompleksitas waktu untuk *solve* jika *n* sudah berukuran kecil
- $T(n_1) + T(n_2) \ldots + T(n_r)$: kompleksitas waktu untuk memproses setiap subpersoalan
- $f(n)$: kompleksitas waktu untuk *combine* solusi dari masing-masing subpersoalan

Algoritma *divide and conquer* biasanya digunakan dalam persoalan yang dapat dipecah menjadi subpersoalan dengan karakteristik yang sama dengan karakteristik persoalan semula seperti pada permasalahan penggunaan struktur data *collection* yang memiliki properti rekursif.

Selain itu, algoritma ini dapat digunakan pada implementasi *Convex Hull* dan permasalahan yang diangkat pada tugas kecil kali ini yaitu pencarian *Closest Pair of Points*.

Pada permasalahan *Closest Pair of Points*, algoritma ini diminta mencari sepasang titik terdekat dari sekumpulan titik dalam ruang 3D atau di R^n dengan algoritma *divide and conquer*. Beberapa langkah yang dapat digunakan dalam mendapatkan solusi dari permasalahan ini dengan menggunakan algoritma *divide and conquer* antara lain.

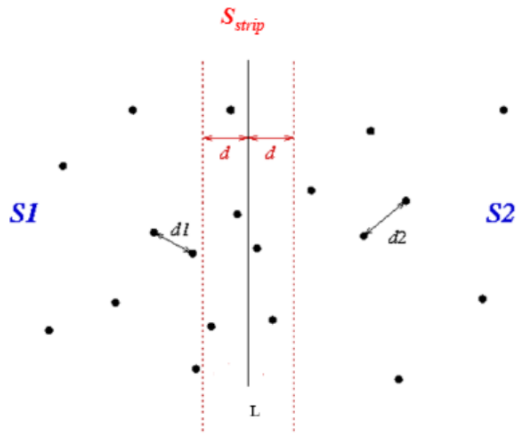
1. Asumsikan jumlah titik dalam ruang tersebut sebagai perpangkatan dua ($n = 2^k$).
2. Sebelum masuk pada algoritma, lakukan *sorting* pada sekumpulan vektor tersebut berdasarkan nilai absisnya (*x*).
3. SOLVE : Jika *input* jumlah titik dalam ruang hanya dua (*n = 2*), langsung masuk pada bagian *solve* dengan cara menghitung jarak antara kedua titik tersebut dengan rumus *euclidean*.

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

Gambar 1.2 *Euclidean distance* ruang 3D

4. DIVIDE : Jika terdapat himpunan titik yang dapat dibagi menjadi bagian yang lebih kecil, bagi sekumpulan titik menjadi 2 bagian (*divide*) misal S_1 dan S_2 dengan jumlah titik yang sama dengan catatan sekumpulan titik tadi sudah dilakukan *sorting* berdasarkan absisnya. Bagian S_1 berisi himpunan titik yang berada di sebelah kiri garis maya L ($y = x_{n/2}$) dan bagian S_2 berisi himpunan titik yang berada di sebelah kanan garis maya L.
5. CONQUER : Cari jarak titik terdekat tiap himpunan dengan rekursivitas algoritma *divide and conquer*.
6. COMBINE : Terdapat tiga kemungkinan letak pasangan titik yang memiliki jarak terdekat yaitu di himpunan S1, himpunan S2, atau pasangan titik yang dipisahkan oleh garis batas L sehingga satu titik berada di bagian S1 dan titik lainnya berada di bagian S2 (S_{strip}). Jika pasangan titik terdekat terdapat pada S_{strip}, maka perlu dilakukan perhitungan lanjutan untuk mendapatkan solusi persoalan semula.

7. Jika terdapat pasangan titik yang berada di p_{left} dan p_{right} yang jaraknya lebih kecil dari *d*, maka selisih absis x dan ordinat y dari p_{left} dan p_{right} paling banyak sebesar *d*. Hal ini dapat diartikan bahwa lebar daerah S_{strip} sebesar *2d*. Oleh karena itu, implementasi dari pencarian pasangan titik terdekat di daerah ini dapat dilakukan dengan menemukan semua titik yang berada di daerah S_{strip} dan menghitung jarak tiap pasangan titik lalu membandingkannya dengan *d*.



Gambar 1.3 Daerah kemungkinan letak *closest pair*

2. Algoritma *Brute Force* sebagai Pembanding

Algoritma brute force merupakan salah satu metode untuk menemukan solusi yang dapat dibilang cukup praktis dan sederhana. Cara kerja dari algoritma brute force adalah dengan mencoba semua kemungkinan yang mungkin terjadi secara berurutan untuk mengidentifikasi dari seluruh kemungkinan tersebut, kemungkinan mana saja yang memenuhi untuk menjadi solusi permasalahan. Hal tersebut menjadikan algoritma brute force membutuhkan waktu eksekusi yang cukup lama.

Pada penyelesaian permasalahan *Closest Pair of Points*, algoritma ini bekerja dengan langkah-langkah sebagai berikut.

1. Hitung jarak setiap pasang titik. Jumlah perhitungan yang digunakan untuk setiap n titik ialah $C(n, 2) = n(n - 1)/2$.
2. Pilih pasangan titik yang mempunyai jarak paling kecil sebagai solusi persoalan.

Kompleksitas dari algoritma *brute force* adalah $O(n^2)$, sedangkan dengan algoritma *divide and conquer*, waktu yang diperlukan sebesar $O(n \log n)$. Maka dari itu, algoritma *divide and conquer* lebih disarankan dalam penyelesaian persoalan ini dengan catatan data yang menjadi persoalan memiliki jumlah yang relatif besar.

3. *Source Code*

a. algorithm.py

```
import random
import math

# Input Titik dengan Pembangkit Acak
def inputArray() :
    dimension = int(input("Masukkan jumlah dimensi ruang
dari titik     : "))
    while (dimension <= 1) :
        print("Masukkan salah. Silakan input lagi!")
        dimension = int(input("Masukkan jumlah dimensi
ruang dari titik     : "))

    vertex = int(input("Masukkan jumlah titik yang ingin
dibentuk    : "))
    while (vertex <= 1) :
        print("Masukkan salah. Silakan input lagi!")
        vertex = int(input("Masukkan jumlah titik yang
ingin dibentuk    : "))

    array = [[0 for x in range(dimension)] for y in
range(vertex)]
    for i in range(vertex) :
        for j in range(dimension) :
            array[i][j] =
round(random.uniform(-100,100),2)
    return array

# Print Titik
def printArray(array) :
    for i in range(len(array)) :
        for j in range(len(array[0])) :
```

```
            print(str(array[i][j]), end = " ")
        print("")

# Sorting dengan Bubble Sort
def sortingArray(array) :
    for i in range(len(array)) :
        for j in range(0, len(array)-i-1) :
            if array[j] > array[j+1] :
                array[j], array[j+1] = array[j+1],
array[j]
    return array

# Membagi Titik menjadi 2 Himpunan (Kiri(S1) dan
Kanan(S2))
def divide(array) :
    S1 = []
    S2 = []
    for i in range(len(array)) :
        if (i < len(array)//2) :
            S1.append(array[i])
        else :
            S2.append(array[i])
    return S1, S2

# Menghitung Euclidean Distance
def euclideanDistance(array1, array2, n) :
    sum = 0
    for i in range(len(array1)) :
        sum += (array1[i] - array2[i])**2
    distance = math.sqrt(sum)
    array = [array1,array2]
    n += 1
    return distance, array, n

# Mencari Minimum
def min(d1, d2) :
    if (d1 < d2) :
        return d1
    else :
        return d2
```

```
# Mencari Pasangan Titik Terdekat
def findClosestPair(array, n) :
    if (len(array) == 2) :
        dist, closestArray, n =
euclideanDistance(array[0], array[1], n)
    elif (len(array) == 3) :
        dist1, closestArray1, n =
euclideanDistance(array[0], array[1], n)
        dist2, closestArray2, n =
euclideanDistance(array[0], array[2], n)
        dist3, closestArray3, n =
euclideanDistance(array[1], array[2], n)
        dist = min(min(dist1,dist2),dist3)
        if (dist == dist1) :
            closestArray = closestArray1
        else :
            if (dist == dist2) :
                closestArray = closestArray2
            else :
                closestArray = closestArray3
    else :
        S1, S2 = divide(array)
        d1, closestA1, n = findClosestPair(S1, n)
        d2, closestA2, n = findClosestPair(S2, n)
        dist = min(d1, d2)
        if (dist == d1) :
            closestArray = closestA1
        else :
            if (dist == d2) :
                closestArray = closestA2
        dist, closestArray, n = sStrip(dist,
closestArray, array, n)
    return dist, closestArray, n

# Mencari Titik Terdekat dalam Sstrip
def sStrip(d, closestA, array, n) :
    if (len(array) % 2 == 1) :
        x = array[len(array)//2+1][0]
    else :
```

```
        x = (array[len(array)//2][0] +
array[len(array)//2+1][0])/2
    S = []
    for i in range(len(array)) :
        if (array[i][0] >= x-d and array[i][0] <= x+d) :
            S.append(array[i])
    for i in range(len(S)) :
        for j in range(i+1,len(S)) :
            dist, a, n = euclideanDistance(S[i],S[j], n)
            if (dist < d) :
                d = dist
                closestA = a
    return d, closestA, n

# Algoritma Brute Force
def bruteForce(array):
    d, A, n = euclideanDistance(array[0],array[1], -1)
    for i in range(len(array)):
        for j in range(i+1,len(array)):
            d1, a, n =
euclideanDistance(array[i],array[j], n)
            if d1 <= d:
                d = d1
                A = a
    return d, A, n
```

Gambar 3.1 *Source code algorithm.py*

b. visualize.py

```
import numpy as np
import matplotlib.pyplot as plt

def visualize(A, B) :
    A.remove(B[0])
    A.remove(B[1])
    A = np.array(A)
    B = np.array(B)
    fig = plt.figure()
    if (len(B[0]) == 3) :
        ax = plt.axes(projection = '3d')
        if (len(A) != 0) :
            ax.scatter3D(A[:,0], A[:,1], A[:,2], c='y')
```

```
        ax.scatter3D(B[:,0], B[:,1], B[:,2], c='r')
        ax.set_xlabel('x')
        ax.set_ylabel('y')
        ax.set_zlabel('z')
        plt.show()
    elif (len(B[0]) == 2) :
        if (len(A) != 0) :
            plt.scatter(A[:,0], A[:,1], c='y')
        plt.scatter(B[:,0], B[:,1], c='r')
        plt.show()
```

Gambar 3.2 *Source code visualize.py*

c. splashScreen.py

```
def splashScreen() :
    print("")
    print("
                                                                  ")
    print("
                                                                  ")
    print("
                                                                  ")
    print("
                                                                  ")
    print("
                                                                  ")
    print("
                                                                  ")
    print("
                                                                  ")
    print("
                                                                  ")
    print("
                                                                  ")
    print("
                                                                  ")
    print("
                                                                  ")
    print("
")
    print("")
```

Gambar 3.3 *Source code splashScreen.py*

d. main.py

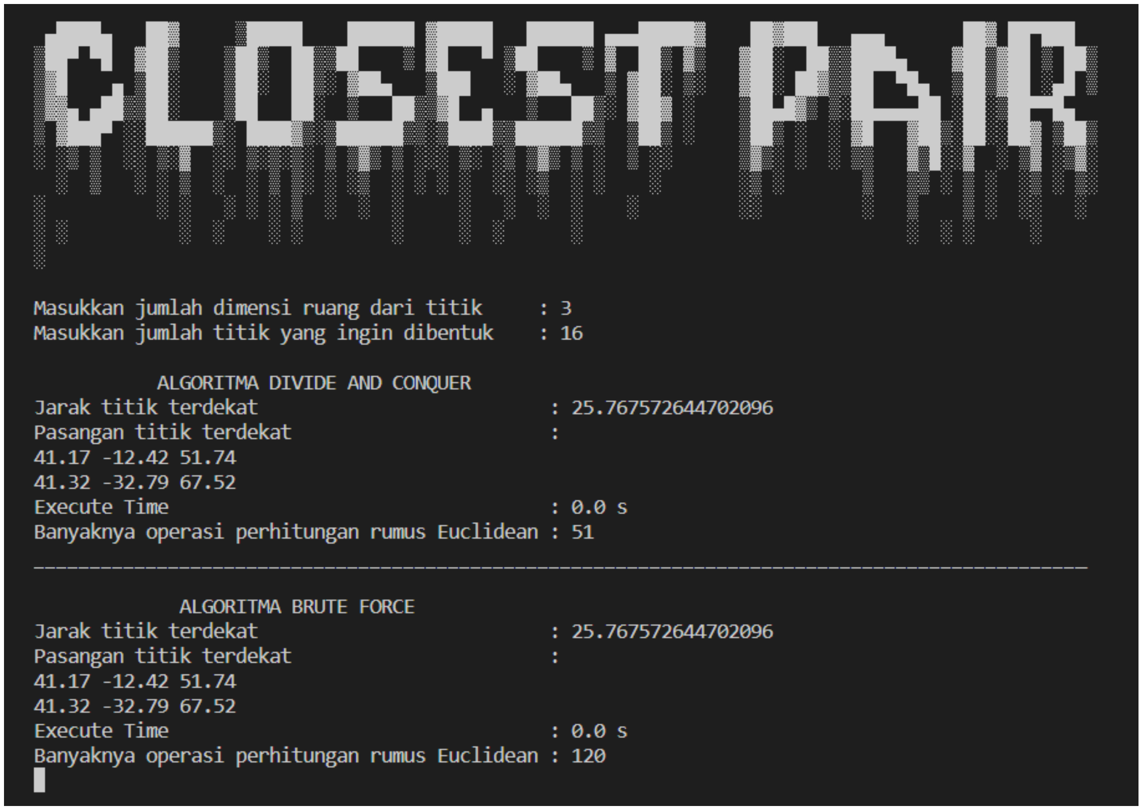
```
import time
import os
from splashScreen import *
from algorithm import *
from visualize import *

def main() :
    flag = False
    while (flag == False) :
        splashScreen()
        arrayV = inputArray()
        arrayV = sortingArray(arrayV)
        # print("\nS : ")
        # printArray(arrayV)
        n1 = 0
        startTimeDnC = time.time()
        distDnC, closestArrayDnC, n1 =
findClosestPair(arrayV, n1)
        finishTimeDnC = time.time()
        startTimeBF = time.time()
        distBF, closestArrayBF, n2 = bruteForce(arrayV)
        finishTimeBF = time.time()
        print("\n           ALGORITMA DIVIDE AND CONQUER
")
        print("Jarak titik terdekat
:", (distDnC))
        print("Pasangan titik terdekat
:")
        printArray(closestArrayDnC)
        print("Execute Time
:", (finishTimeDnC-startTimeDnC), "s")
        print("Banyaknya operasi perhitungan rumus
Euclidean :", n1)

print("_____________________________________________________
_____________________________________________")
        print("\n             ALGORITMA BRUTE FORCE
")
```

```
        print("Jarak titik terdekat
:", (distBF))
        print("Pasangan titik terdekat
:")
        printArray(closestArrayBF)
        print("Execute Time
:", (finishTimeBF-startTimeBF), "s")
        print("Banyaknya operasi perhitungan rumus
Euclidean :", n2)
        if ((len(arrayV[0]) == 3) or (len(arrayV[0]) ==
2)) :
            visualize(arrayV, closestArrayDnC)
        choose = str(input("\nApakah anda ingin mencari
closest pair lagi? (yay/nay) "))
        while ((choose != "yay") and (choose != "nay")) :
            print("Masukkan salah. Silakan input lagi!")
            choose = str(input("Apakah anda ingin mencari
closest pair lagi? (yay/nay) "))
        if (choose == "yay") :
            os.system('cls')
        else :
            flag = True

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

Gambar 3.4 *Source code main.py*

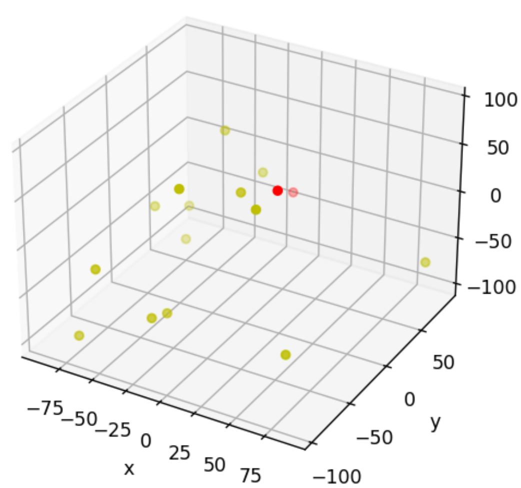
4. *Screenshot Input* dan *Output* dari Program

a. *n* = 16 pada ruang 3D



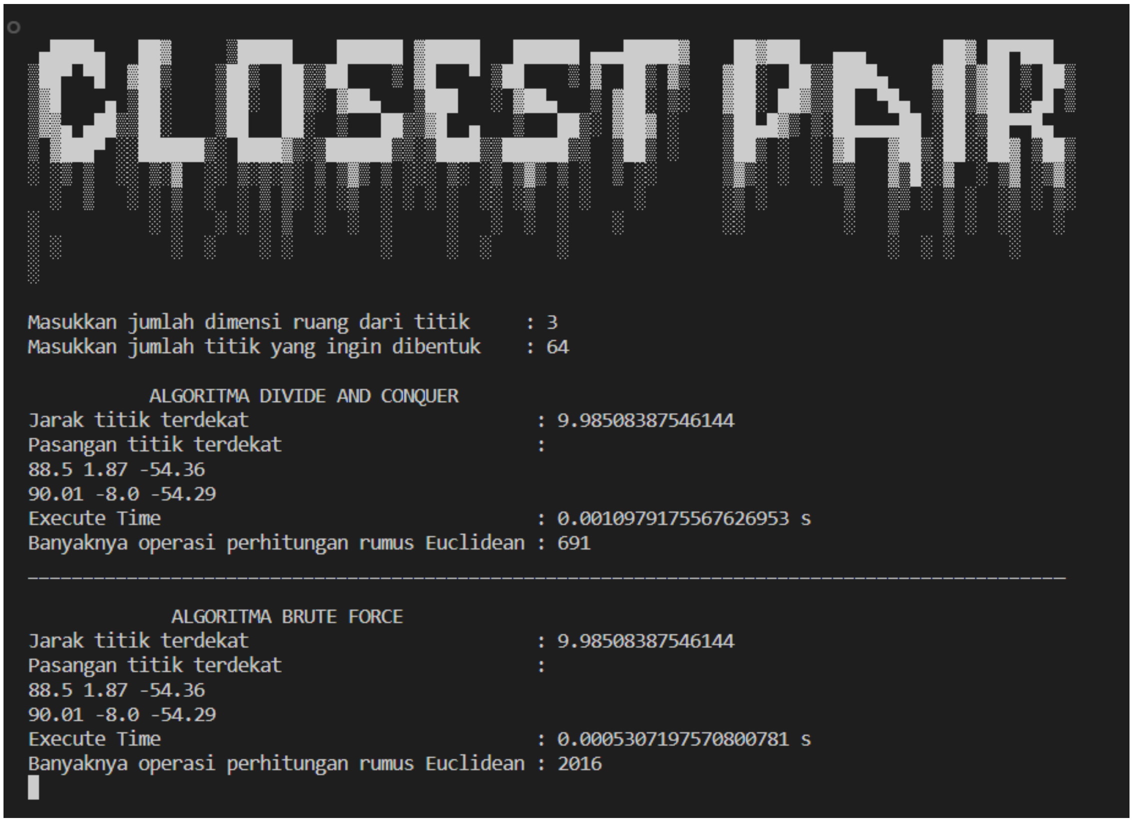
```
CLOSEST PAIR

Masukkan jumlah dimensi ruang dari titik     : 3
Masukkan jumlah titik yang ingin dibentuk    : 16

           ALGORITMA DIVIDE AND CONQUER
Jarak titik terdekat                          : 25.767572644702096
Pasangan titik terdekat                       :
41.17 -12.42 51.74
41.32 -32.79 67.52
Execute Time                                  : 0.0 s
Banyaknya operasi perhitungan rumus Euclidean : 51
---------------------------------------------------------------------------------------------

             ALGORITMA BRUTE FORCE
Jarak titik terdekat                          : 25.767572644702096
Pasangan titik terdekat                       :
41.17 -12.42 51.74
41.32 -32.79 67.52
Execute Time                                  : 0.0 s
Banyaknya operasi perhitungan rumus Euclidean : 120
```

Gambar 4.1.1 *Input* dan *Output* n = 16 pada ruang 3D



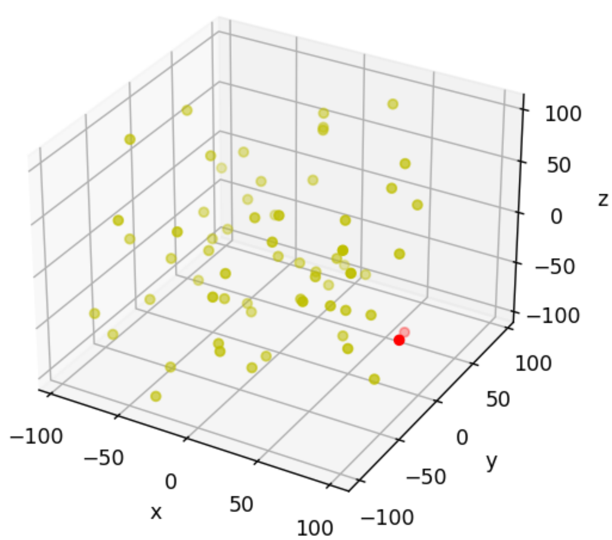
Gambar 4.1.2 Visualisasi n = 16 pada ruang 3D

b. *n* = 64 pada ruang 3D



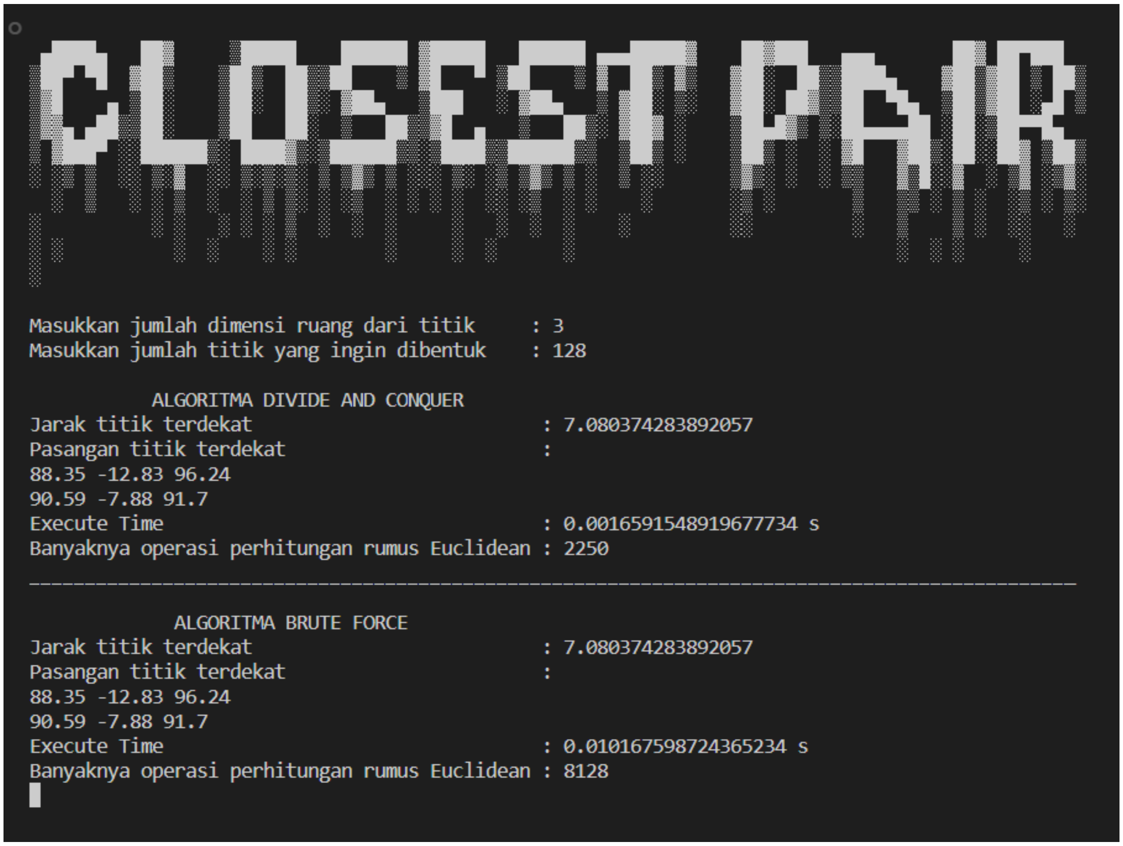
```
Masukkan jumlah dimensi ruang dari titik     : 3
Masukkan jumlah titik yang ingin dibentuk    : 64

           ALGORITMA DIVIDE AND CONQUER
Jarak titik terdekat                          : 9.98508387546144
Pasangan titik terdekat                       :
88.5 1.87 -54.36
90.01 -8.0 -54.29
Execute Time                                  : 0.0010979175567626953 s
Banyaknya operasi perhitungan rumus Euclidean : 691
-----------------------------------------------------------------------------------------------

             ALGORITMA BRUTE FORCE
Jarak titik terdekat                          : 9.98508387546144
Pasangan titik terdekat                       :
88.5 1.87 -54.36
90.01 -8.0 -54.29
Execute Time                                  : 0.0005307197570800781 s
Banyaknya operasi perhitungan rumus Euclidean : 2016
```

Gambar 4.2.1 *Input* dan *Output* n = 64 pada ruang 3D

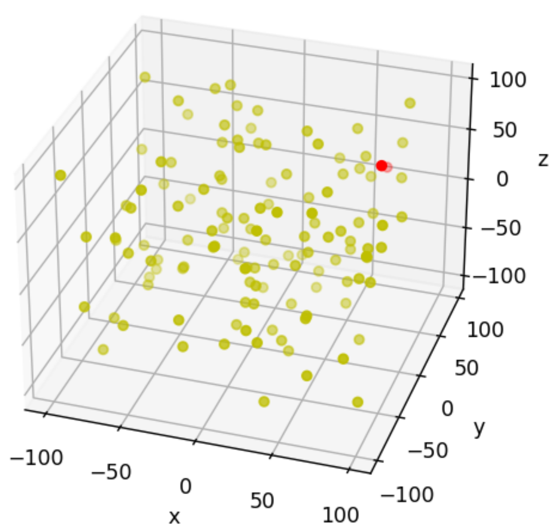


Gambar 4.2.2 Visualisasi n = 64 pada ruang 3D

c. *n = 128* pada ruang 3D

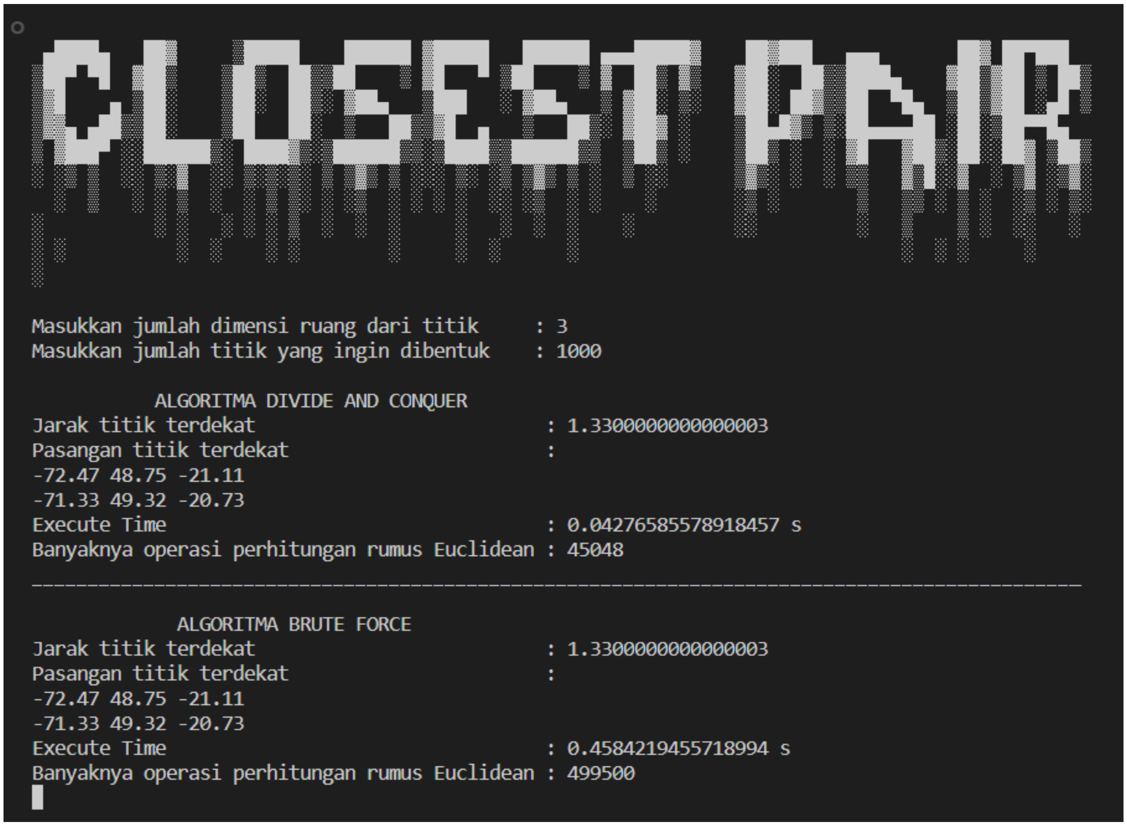


Gambar 4.3.1 *Input* dan *Output* n = 128 pada ruang 3D

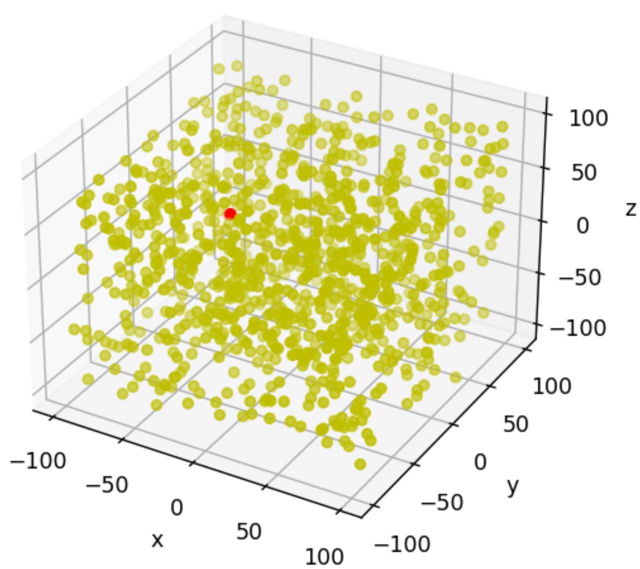


Gambar 4.3.2 Visualisasi n = 128 pada ruang 3D

d. *n = 1000* pada ruang 3D

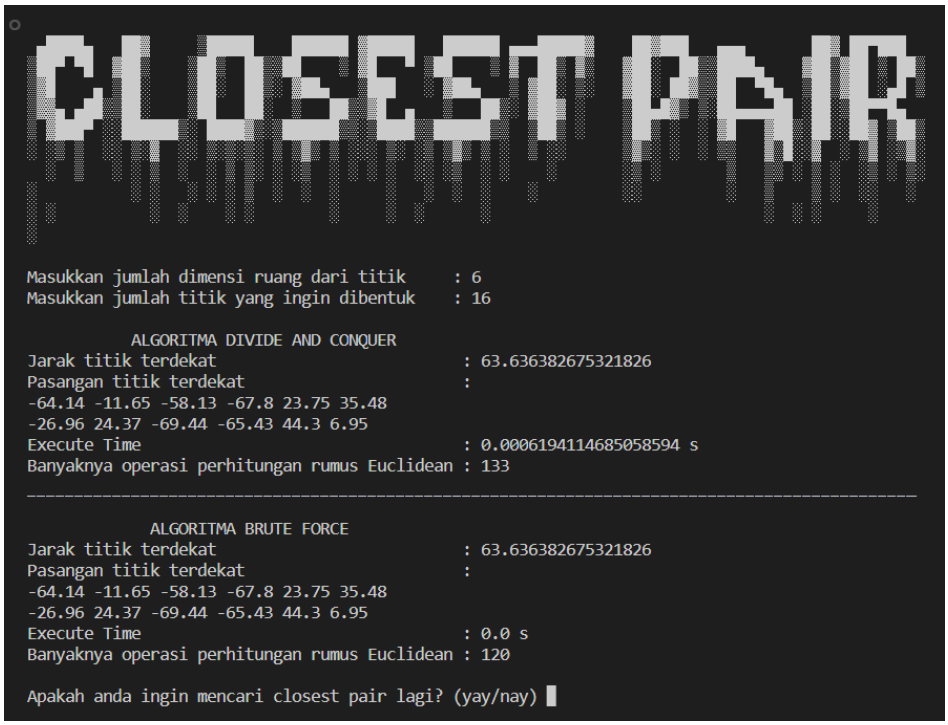


Gambar 4.4.1 *Input* dan *Output* n = 1000 pada ruang 3D



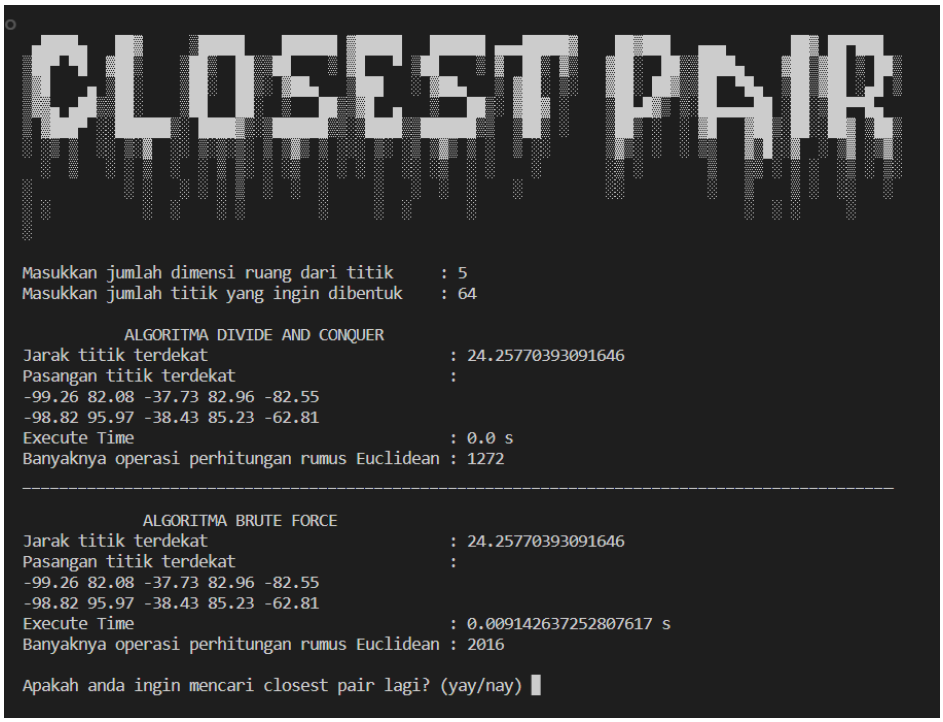
Gambar 4.4.2 Visualisasi n = 1000 pada ruang 3D

e. n = 16 pada ruang 6D



Gambar 4.5.1 *Input* dan *Output* n = 16 pada ruang 6D

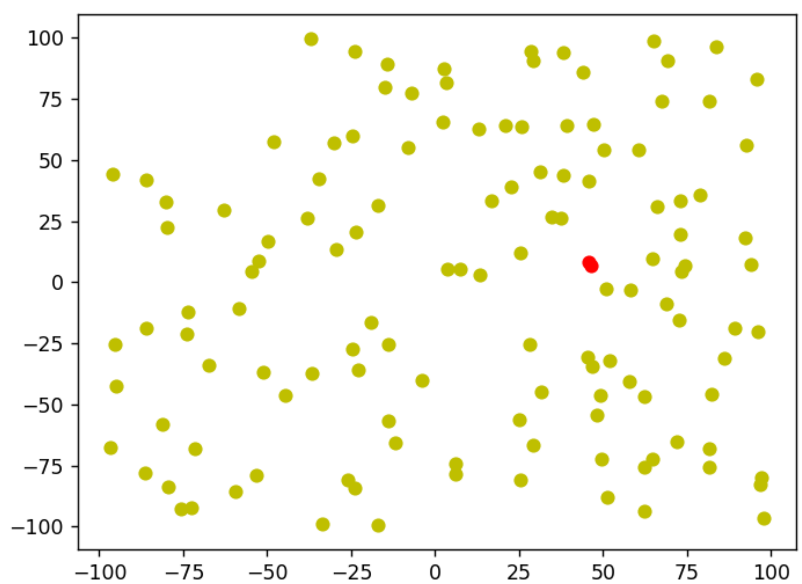
f. n = 64 pada ruang 5D



Gambar 4.6.1 *Input* dan *Output* n = 64 pada ruang 5D

g. *n = 128* pada ruang 2D

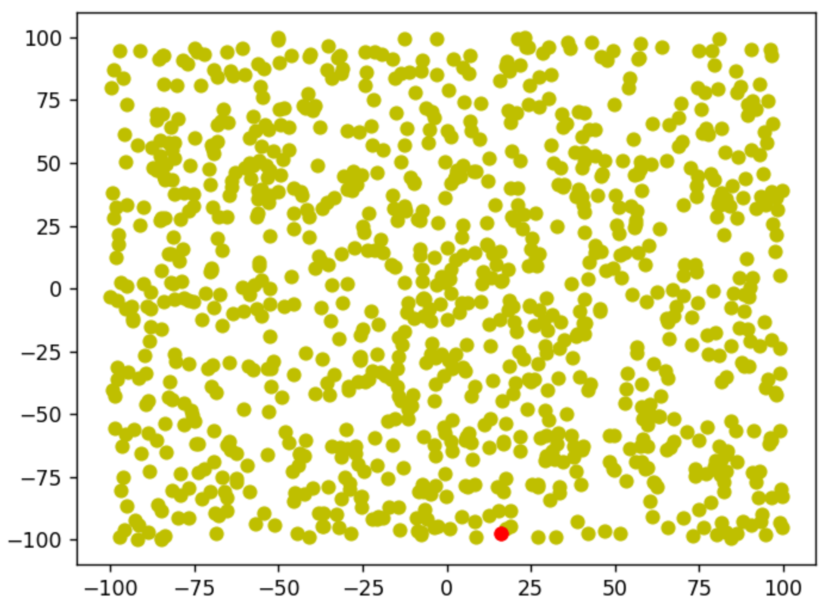
Gambar 4.7.1 *Input* dan *Output* n = 128 pada ruang 2D



Gambar 4.7.2 Visualisasi n = 128 pada ruang 2D

h. *n = 1000* pada ruang 2D

Gambar 4.8.1 *Input* dan *Output* n = 1000 pada ruang 2D



Gambar 4.7.2 Visualisasi n = 1000 pada ruang 2D

5. **Link *Repository GitHub***

   https://github.com/syrifaa/Tucil2_13521018

6. ***Checklist***

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

Tabel 6 *Checklist* Laporan