Tugas Kecil 2 IF2211 Strategi Algoritma
Mencari Pasangan Titik Terdekat dengan Algoritma *Divide and Conquer*

Disusun oleh:
Eunice Sarah Siregar
13521013

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2022/2023

BAB I
PENDAHULUAN

A. Algoritma *Divide and Conquer*

Algoritma *divide and conquer* adalah proses membagi permasalahan ke dalam bagian-bagian yang memiliki kemiripan dengan permasalahan awal, tetapi dibentuk menjadi ukuran lebih kecil. Hal tersebut bertujuan agar permasalahan dapat lebih mudah diselesaikan. Dalam algoritma *divide and conquer* terdapat tiga proses, yaitu *divide*, *combine*, dan *conquer*. Langkah pertama yaitu melakukan pembagian masalah ke dalam upa-permasalahan dengan ukuran yang kecil. Selanjutnya, setiap upa-permasalahan tersebut akan diselesaikan secara rekursif. Terakhir, dari masing-masing solusi dari upa-permasalahan tersebut di-*combine* sehingga dapat menyelesaikan permasalahan awal.

B. Pencarian Pasangan Titik Terdekat dengan Algoritma *Divide and Conquer*

Untuk mendapatkan pasangan titik terdekat, penulis menggunakan algoritma *divide and conquer* dengan deskripsi langkah- langkah sebagai berikut.

1. Dalam matrix, dilakukan pengurutan terlebih dahulu berdasarkan nilai absis,
2. Setelah mengurutkan, matrix tersebut dibagi menjadi dua bagian kiri dan kanan,
3. Ketika berhasil melakukan *divide*, selanjutnya terdapat tiga kondisi, yaitu
    a. Saat panjang matrix adalah 2, maka titik terdekat langsung ditemukan
    b. Namun, ketika panjang matrix adalah 3 maka titik terdekat dicari menggunakan algoritma brute force
    c. Ketika tidak termasuk keduanya, matrix akan dibagi menjadi dua bagian (kiri dan kanan) sampai panjang matrix berukuran 2 atau 3 yang dilakukan dengan cara rekursif. Terdapat kondisi lain ketika membagi matrix, yaitu kondisi dimana titik terdekat berada diantara pembagi wilayah.
4. Untuk mencari jarak terdekat, dapat dicari dengan rumus Euclidian:

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

BAB II
SOURCE PROGRAM DALAM PYTHON

A. function.cpp

```
import math as m
import random
import os

os.system('cls')
print("
")
print("
")
print("
")
print("
")
print("
")
print("
")
print("
")
print("
")
print("==========================================================================
====================")
dot = int(input("Masukkan jumlah titik: "))
dim = int(input("Masukkan jumlah dimensi: "))

def random_matrix(dot, dim):
    matrix = [[0 for i in range(dim)] for j in range(dot)]
    for i in range(dim):
        for j in range(dot):
            matrix[j][i] = random.randint(0, 100)

    return matrix


def distance_matrix(matrix):
    dist_matrix = [[0 for i in range(dim)] for j in range(dim)]
    sum = 0
    for i in range(dim):
        for j in range(dim):
            dist_matrix[j][i] = (matrix[0][i] - matrix[1][i])**2
        sum += dist_matrix[j][i]
    distance = m.sqrt(sum)
    return distance, matrix[0], matrix[1]
a = random_matrix(dot, dim)

def sortX(matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if matrix[i][0] < matrix[j][0]:
                matrix[i], matrix[j] = matrix[j], matrix[i]
    return matrix
```

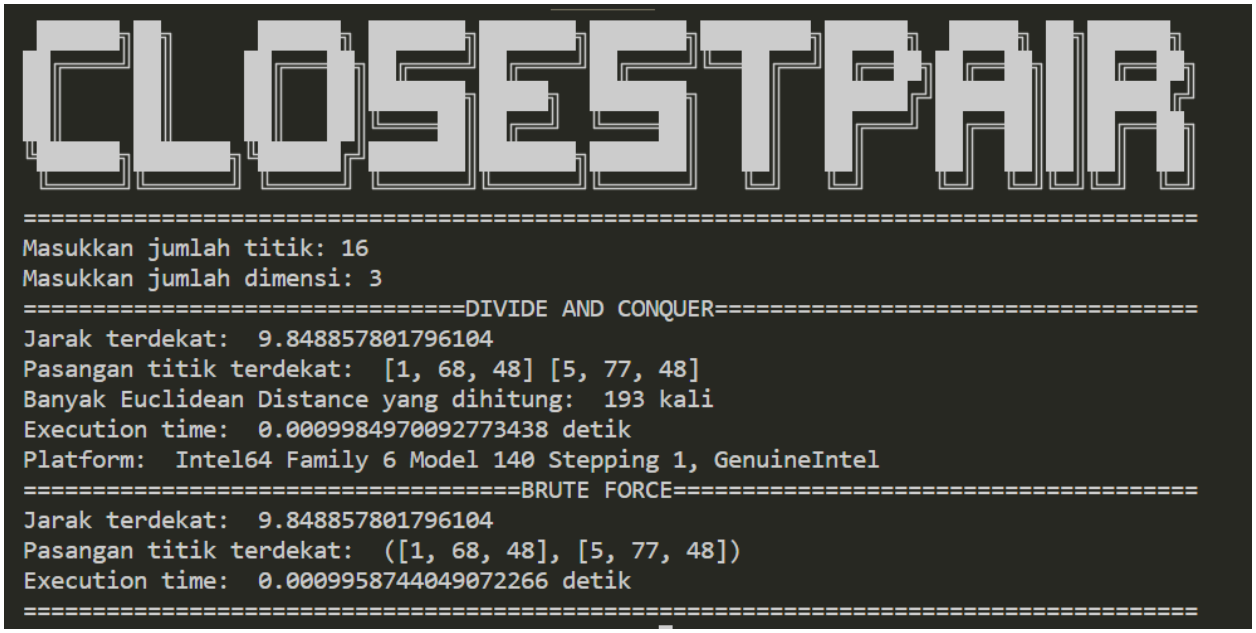
B. algorithm.cpp

```
from function import *
import time as t
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import platform

count = 0

def divide(matrix):
    # left = []
    # right = []
    if len(matrix) % 2 == 0:
        mid = len(matrix) // 2
        left = matrix[:mid]
        right = matrix[mid:]
    else:
        mid = len(matrix) // 2
        left = matrix[:mid]
        right = matrix[mid+1:]
        # left.append(l)
        # right.append(r)
        # return min(divide(left), divide(right))
    # print("Left: ", left)
    # print("Right: ", right)
    return left, right

def bruteForce(matrix):
    global count
    dmin = 9999
    A = []
    for i in range(len(matrix)):
        for j in range(i+1, len(matrix)):
            d = distance_matrix([matrix[i], matrix[j]])
            # count += 1
            if d[0] < dmin:
                dmin = d[0]
                A = d[1], d[2]
    # print("terkecil : ", dmin)
    return dmin, A


def sStrip(matrix, distance):
    global count
    terjauh = matrix[len(matrix)-1][0] - matrix[0][0]/2
    for i in range(len(matrix)):
        for j in range(i+1, len(matrix)):
            # if matrix[j][0] - matrix[i][0] < terjauh:
                d = distance_matrix([matrix[i], matrix[j]])
                count += 1
                if d[0] < distance:
                    return d[0], d[1], d[2]
```

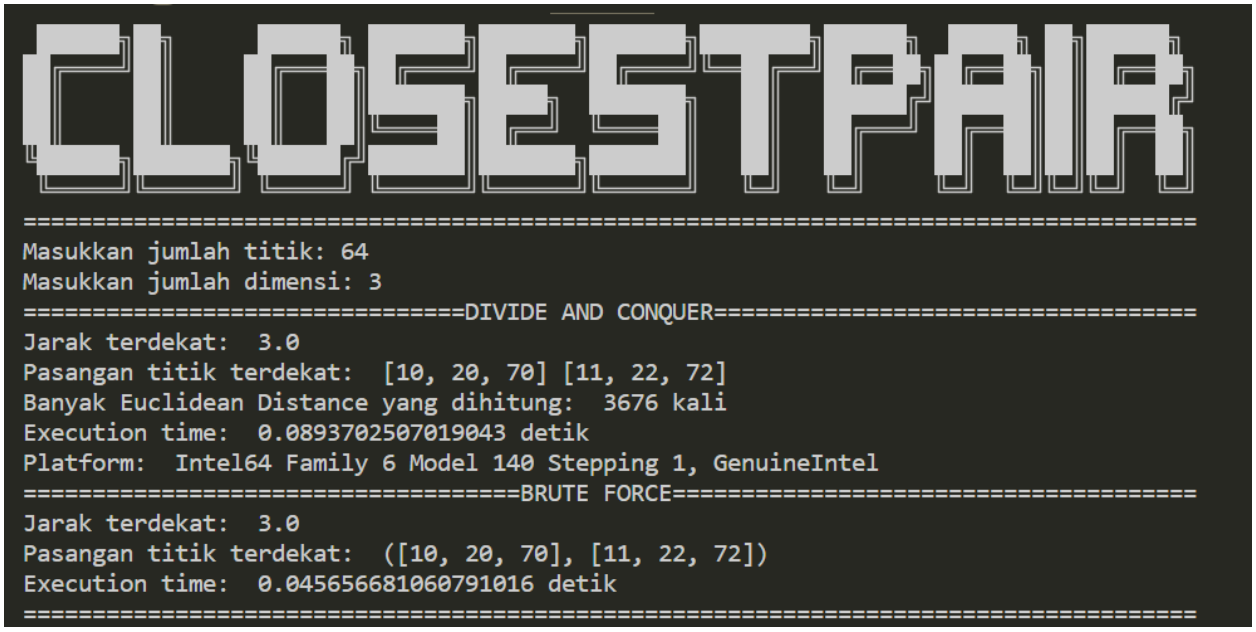
```
    return distance, matrix[0], matrix[1]



def closestPair(matrix,dot):
    # t.sleep(1)
    global count
    if len(matrix) == 2:
        distance, point1, point2 = distance_matrix(matrix)
        count += 1
        # print("matrix terdekat adalah dist", point1, "dan", point2, "dengan
jarak", distance)
        return distance, point1, point2

    elif len(matrix) == 3:
        distance, A = bruteForce(matrix)
        point1 = A[0]
        point2 = A[1]
        # print("matrix terdekat adalah brute", point1, "dan", point2, "dengan
jarak", distance)
        return distance, point1, point2
    else:
        left, right = divide(sortX(matrix))
        # print(left)
        # print(right)
        dl = closestPair(left, len(matrix)//2)
        # print(dl)
        point11, point21 = left[0], left[1]
        dr = closestPair(right, len(matrix)//2)
        # print(dr)
        point12, point22 = right[0], right[1]
        if dl < dr :
            point1 = point11
            point2 = point21
        else:
            point1 = point12
            point2 = point22
        distance = min(dl, dr)[0]
        # print("distance: ", distance)
        minimum, point13, point23 = sStrip(matrix, distance)
        # print("sStrip: ", minimum)
        if minimum < distance:
            distance = minimum
            point1 = point13
            point2 = point23
            # print("matrix terdekat adalah minimum", point1, "dan", point2,
"dengan jarak", distance)
            return distance, point1, point2
        else:
            point1 = min(dl, dr)[1]
            point2 = min(dl, dr)[2]
            # print("matrix terdekat adalah distance", point1, "dan", point2,
"dengan jarak", distance)

            return distance, point1, point2
    # return distance, point1, point2
```

```
# distance_matrix(a)

# print("halo: ", sStrip(a, distance_matrix(a)[0]))
def visual(matrix, point1, point2):
    fig = plt.figure(figsize = (10, 7))
    ax = plt.axes(projection ="3d")
    for i in range(len(matrix)):
        x = matrix[i][0]
        y = matrix[i][1]
        z = matrix[i][2]
        ax.scatter(x, y, z, c='black', marker='o')
    ax.scatter(point1[0], point1[1], point1[2], c='red', marker='o')
    ax.scatter(point2[0], point2[1], point2[2], c='red', marker='o')
    plt.show()

def main():
    startTime = t.time()
    z, b, c = closestPair(a, dot)
    endTime = t.time()
    print("================================DIVIDE AND
CONQUER================================")
    print("Jarak terdekat: ", z)
    print("Pasangan titik terdekat: ", b, c,)
    print("Banyak Euclidean Distance yang dihitung: ", count, "kali")
    print("Execution time: ", endTime - startTime, "detik")
    print("Platform: ", platform.processor())
    print("====================================BRUTE
FORCE====================================")
    startTimeBrute = t.time()
    d, e = bruteForce(a)
    endTimeBrute = t.time()
    print("Jarak terdekat: ", d)
    print("Pasangan titik terdekat: ", e)
    print("Execution time: ", endTimeBrute - startTimeBrute, "detik")

print("================================================================================
==============")
    yay = input("Menampilkan visualisasi titik terdekat? (y/n) ")
    if yay == 'y' and dim == 3:
        visual(a, b, c )
    elif yay == 'y' and dim != 3:
        print("Maaf dimensi tidak tersedia untuk visualisasi")
    print("Terima kasih telah bermain!")

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

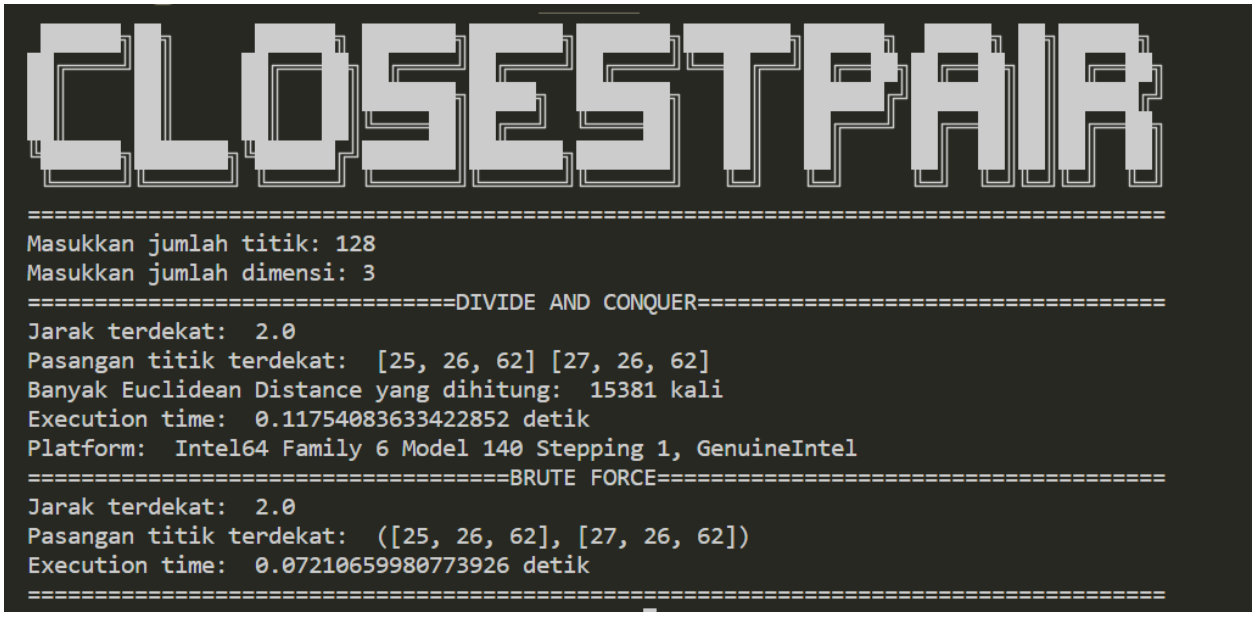
BAB III
TEST CASE



Gambar 1. Hasil Test Case I

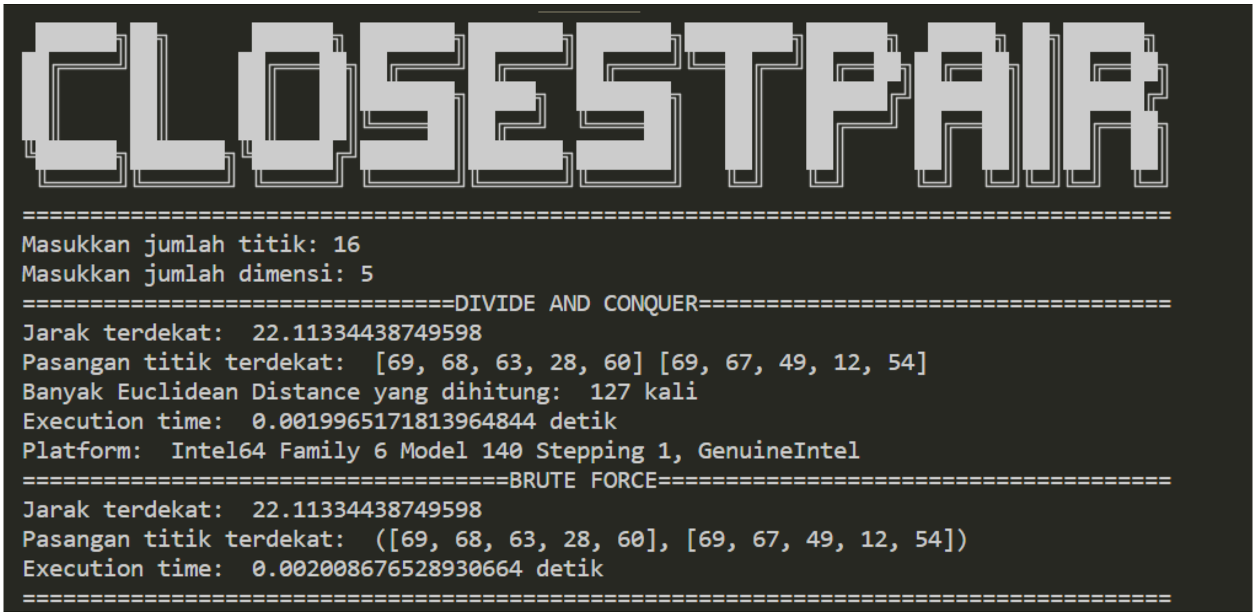


Gambar 2. Hasil Test Case II

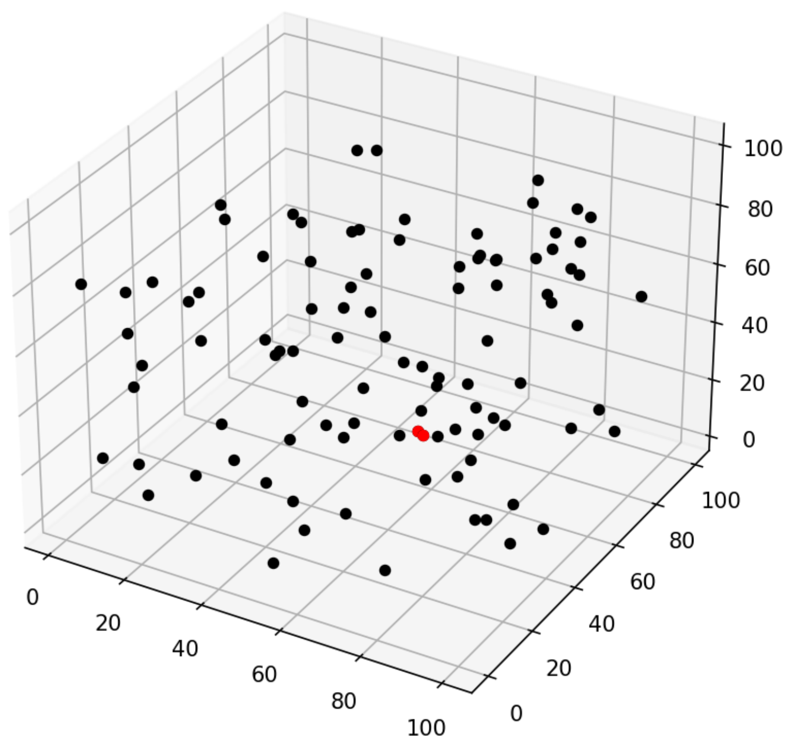


Gambar 3. Hasil Test Case III

Gambar 4. Hasil Test Case IV



Gambar 5. Hasil Test Case V



Gambar 6. Hasil Test Case VI

LAMPIRAN

Link Github
https://github.com/eunicesarah/Tucil2_13521013.git

Checklist

Poin	Ya	Tidak
Program berhasil dikompilasi tanpa ada kesalahan	✅	
Program berhasil running	✅	
Program dapat menerima masukan dan dan menuliskan luaran.	✅	
Luaran program sudah benar (solusi closest pair benar)	✅	
Bonus 1 dikerjakan	✅	
Bonus 2 dikerjakan	✅	