LAPORAN TUGAS KECIL IF2211
MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA DIVIDE AND CONQUER

Ditujukan untuk memenuhi salah satu
tugas kecil mata kuliah IF2122 Strategi Algoritma
pada Semester II Tahun Akademik 2022/2023

Disusun oleh:

Rizky Abdillah Rasyid	13521109
Muhammad Rizky Sya'ban	13521119

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

DAFTAR ISI 1
BAB I 2
BAB II 4
BAB III 7
BAB IV 23
A. Titik dalam 3 dimensi 23
B. Titik dalam dimensi lebih dari 3 25
C. Kasus-kasus ekstrim 26
BAB V 29
BAB VI 30

BAB I
DESKRIPSI MASALAH

Mencari sepasang titik terdekat dengan Algoritma Divide and Conquer sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tucil 2 kali ini Anda diminta 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 $P_1 = (x_1, y_1, z_1)$ dan $P_2 = (x_2, y_2, z_2)$ dihitung dengan rumus Euclidean berikut:

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

Buatlah program dalam dalam Bahasa C/C++/Java/Python/Golang/Ruby/Perl (pilih salah satu) 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 program:

- N
- titik-titik (dibangkitkan secara acak) dalam koordinat (x, y, z)

Luaran program:

- sepasang titik yang jaraknya terdekat dan nilai jaraknya
- banyaknya operasi perhitungan rumus Euclidian
- waktu riil dalam detik (spesifikasikan komputer yang digunakan)

Luaran Tambahan :

- penggambaran semua titik dalam bidang 3D, sepasang titik yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya.

- Generalisasi program anda sehingga dapat mencari sepasang titik terdekat untuk sekumpulan vektor di R_n, setiap vektor dinyatakan dalam bentuk x = (x1, x2,…, xn)

BAB II
ALGORITMA PROGRAM

A. Data Type

Terdapat 2 tipe data dalam program ini, yang pertama adalah tipe data point dan couple.

1. Tipe data point

   Tipe data point dibuat sebagai sebuah class ‘point’ yang merepresentasikan titik dalam ruang dimensi tertentu. Class point memiliki dua atribut yaitu dimensi yang menunjukkan jumlah dimensi titik tersebut dan value yang menunjukkan koordinat titik tersebut. Tipe data ini dibandingkan berdasarkan koordinat pertamanya pada value.

2. Tipe data couple

   Tipe data couple dibuat sebagai sebuah class ‘couple’ yang merepresentasikan sepasang titik dalam ruang dimensi tertentu. Class couple memiliki 3 atribut yaitu ‘point1’ yang merepresentasikan titik pertama, point2 yang merepresentasikan titik kedua, dan distance yang merepresentasikan jarak euclidean antara kedua titik tersebut. Tipe data couple dibandingkan berdasarkan atribut distance.

B. Brute Force

Algoritma brute force adalah metode penyelesaian masalah dengan menguji semua kemungkinan solusi secara sistematis. Meskipun akurat, algoritma ini cenderung lambat dan memerlukan waktu yang lama terutama pada masalah dengan jumlah kemungkinan solusi yang banyak.

Pada program ini akan melakukan pengecekan semua jarak tiap titik dengan melakukan pengecekan sebanyak $C(n, 2) = n(n - 1)/2$ pasangan titik dengan n adalah jumlah titik dan C adalah kombinasi. Setiap pasangan titik akan dihitung jaraknya dengan rumus euclidean dan akan dipilih pasangan titik yang mempunyai jarak euclidean terkecil. Sehingga pengoperasian rumus euclidean hanya bergantung pada jumlah titik tidak terpengaruh dengan jumlah dimensi titik.

Kompleksitas algoritma brute force pada pencarian pasangan titik terdekat dalam multidimensi adalah $O(n^2)$ pada semua kasus (*best case* dan *worst case*) dimana n adalah jumlah titik.

C. Divide and Conquer

Algoritma divide and conquer adalah metode penyelesaian masalah dengan memecah masalah menjadi submasalah yang lebih kecil, menyelesaikan masing-masing submasalah secara terpisah, dan menggabungkan solusi submasalah tersebut untuk mendapatkan solusi akhir masalah asli.

Pada pencarian titik terdekat dengan algoritma divide and conquer permasalahan didekomposisi dengan konfigurasi sebagai berikut:

1. Preproses: Sebelum dilakukan pencarian titik terdekat, kumpulan titik-titik diurutkan berdasarkan nilai indeks pertama ($e1$) pada vektor titik.
2. Solve: Jika n = 2, maka jarak kedua titik dihitung dengan rumus euclidean. Jika n = 3, maka jarak minimum diantara tiga titik dihitung secara brute force.
3. Divide: dilakukan pembagian himpunan menjadi dua bagian, left_part dan right_part, setiap bagian memiliki jumlah titik yang sama (right_part = left_part) atau jumlah right_part = left_part + 1 (kasus n ganjil).
4. Conquer: Algoritma dilakukan secara rekursif dengan basis n ≤ 3, dan diterapkan bagian Solve untuk mendapatkan pasangan titik terdekat
5. Combine: Melakukan pemilihan titik terdekat dari hasil pencarian pada dua partisi dengan kemungkinan:
    a. Pasangan titik terdekat terdapat pada *left_part*
    b. Pasangan titik terdekat terdapat pada *right_part*
    c. Pasangan titik terdekat dipisahkan garis yang membagi dua partisi (satu titik terdapat di *left_part* dan pasangan titik lainnya berada di *right_part*)
6. Kasus 5.c: Untuk kasus pasangan titik terdekat yang dipisahkan garis terhadap $e1$ yang membagi dua partisi dilakukan ketika sudah mendapat jarak minimum *d* dari jarak minimum dari partisi left_right dan partisi right_part dengan langkah-langkah:
    a. Menentukan posisi garis pemisah mid yaitu titik tengah antara titik paling kanan *left_part* dan titik paling kiri *right_part* .
    ($mid = (left_part[-1].value[0] + right_part[0].value[0]) / 2$)
    b. Lakukan pencarian pasangan titik yang memungkinkan untuk memiliki jarak kurang dari *d*. Pada *left_part* dilakukan pencarian titik dari garis pemisah (mid)

hingga sejauh *d* ke kiri ($|mid - left_part[i]| \leq d$). Pada *right_part* dilakukan pencarian titik dari garis pemisah (mid) hingga sejauh *d* ke kanan ($|mid - right_part[i]| \leq d$).

c. Jika telah didapat titik dari *left_part* dan *right_part*, dilakukan pengecekan jarak di tiap dimensi pada setiap titik di *left_part* dengan setiap titik di *right_part* tersebut. jika
d. Dilakukan perhitungan jarak euclidean untuk setiap pasangan titik yang memiliki jarak kurang dari *d* di setiap dimensinya.
e. Jarak terdekat merupakan nilai terkecil antara semua nilai jarak pada perhitungan tersebut (bagian d.) dengan nilai *d*

Adapun kompleksitas algoritma ini adalah $T(n) = 2\,T(n/2) + c^2$ dimana suku pertama berasal dari pembagian himpunan menjadi dua bagian dan suku kedua berasal dari tahap 6 yang mana kasus terbaiknya adalah $c = 0$ dan kasus terburuknya adalah $c = n/2$ sehingga dari teorema master didapatkan kompleksitas untuk *worst case* adalah $O(n^2)$ dan untuk *best case* nya adalah $O(n \log n)$

BAB III

KODE PROGRAM

A. dataType.py

```
import math
class point:
    def __init__(self, dim : int, val : list[int]):
        self.dimensi = dim # dimensi titik
        self.value = val # koordinat titik

    def getDimensi(self) -> int:
        return self.dimensi

    def __lt__(self, other):
        # less than
        return self.value[0] < other.value[0]

    def __gt__(self, other):
        # greater than
        return self.value[0] > other.value[0]

    def __str__(self):
        # print point
        res:str = ""
        for i in range(len(self.value)):
            if (i == 0):
                res += ("(" + str(self.value[i]) + ", ")
            elif (i == len(self.value) - 1) :
                res += (str(self.value[i]) + ")")
            else :
                res += (str(self.value[i]) + ", ")
        return res


class couple:
    def __init__(self, p1: point, p2: point):
        self.point1 = p1 # titik pertama
        self.point2 = p2 # titik kedua
        self.distance = getDistance(p1,p2,p1.dimensi)
        # jarak antara titik pertama dan kedua
```

```
    def __gt__(self, other):
        # greater than
        return self.distance > other.distance

    def __lt__(self, other):
        # less than
        return self.distance < other.distance

    def __str__(self):
        # print couple
        res:str = ""
        for i in range(len(self.point1.value)):
            if (i == 0):
                res += "({:.3f}, ".format( self.point1.value[i])
            elif (i == len(self.point1.value) - 1) :
                res += "{:.3f})".format(self.point1.value[i])
            else :
                res += "{:.3f}, ".format(self.point1.value[i])
        res += " - "
        for i in range(len(self.point2.value)):
            if (i == 0):
                res += "({:.3f}, ".format(self.point2.value[i])
            elif (i == len(self.point2.value) - 1) :
                res += "{:.3f})".format(self.point2.value[i])
            else :
                res += "{:.3f}, ".format(self.point2.value[i])
        return res

def getDistance(p : point, q : point, dim : int) -> float:
    """
    Return Euclidean distance between point p and point q
    """
    temp : float = 0
    for i in range(dim):
        temp += (p.value[i] - q.value[i])**2
    return math.sqrt(temp)
```

B. bruteForce.py

```
import numpy as np
import random as rand
from dataType import point, couple

def bruteForce(points : list[point]) -> couple:
    """
    return 2 closest point as a couple in points
    and number of euclidean distance calculation
    using brute force algorithm
    """
    closestPair:couple = couple(points[0], points[1])
    numEuclidean:int = 1
    for i in range(len(points)):
        for j in range(i+1, len(points)):
            tempCouple = couple(points[i], points[j])
            numEuclidean += 1
            if (closestPair > tempCouple):
                closestPair = tempCouple

    return closestPair, numEuclidean

def driver():
    # 100 titik
    # 3 dimensi
    points = np.empty((0), dtype=point)

    for i in range(100):
        val = np.empty((3), dtype=float)
        for j in range(3):
            val[j] = rand.uniform(-1000, 1000)
            print(val[j], end=" ")
        points = np.append(points, point(3, val))

    for titik in points:
        print(titik)

    nearestCouple, _ = bruteForce(points)
    print("nearest by BruteForce : ", nearestCouple)
```

C. divideConquer.py

```
import numpy as np
from dataType import point, couple

def isNeed(p1: point, p2: point, d:float):
    """
    return False if there is distance between p1 and p2
    in the same axis that greater than d
    """
    for i in range(p1.dimensi):
        if (abs(p1.value[i] - p2.value[i]) >= d):
            return False
    return True

def quicksort(points: list[point]):
    if len(points) <= 1:
        return points
    else:
        pivot = points[0]
        left_part = np.empty((0), dtype=point)
        right_part = np.empty((0), dtype=point)
        for i in range(1, len(points)):
            if points[i] < pivot:
                left_part = np.append(left_part, points[i])
            else:
                right_part = np.append(right_part, points[i])
        res = np.concatenate((quicksort(left_part), np.array([pivot])
, quicksort(right_part)))
        return res

def divideConquer(points : list[point]) -> couple:
    """
    Initial State : points was sorted by x1
    return 2 closest point as a couple in points
    and number of euclidean distance calculation
    using divide and conquer algorithm
    """
    n : int  = len(points)
    numEuclidean:int = 0
    if (n <= 2):
        numEuclidean += 1
        return couple(points[0], points[1]), numEuclidean

    elif (n == 3):
```

```
        numEuclidean += 3
        Couple1 = couple(points[0], points[1])
        Couple2 = couple(points[1], points[2])
        Couple3 = couple(points[0], points[2])
        return min(Couple1, Couple2, Couple3), numEuclidean

    else:
        left_part = points[:n//2]
        right_part = points[n//2:]

        closest_left, tempNumLeft = divideConquer(left_part)
        closest_right, tempNumRight = divideConquer(right_part)
        numEuclidean += (tempNumLeft + tempNumRight)

        closestTemp:couple = min(closest_left, closest_right)
        mid:float = (left_part[-1].value[0] + right_part[0].value[0])
/ 2
        dist:float = abs(left_part[-1].value[0] -
right_part[0].value[0])
        if (dist < closestTemp.distance):
            for left_point in left_part:
                if (abs(mid-left_point.value[0]) <=
closestTemp.distance-(dist/2)) :
                    for right_point in right_part :
                        if (abs(mid-right_point.value[0]) <=
closestTemp.distance-(dist/2)) :
                            if isNeed(left_point, right_point,
closestTemp.distance):
                                numEuclidean += 1
                                newClosestTemp: couple =
couple(left_point, right_point)
                                if (newClosestTemp < closestTemp) :
                                    closestTemp = newClosestTemp

        return closestTemp, numEuclidean

def isNeed(p1: point, p2: point, d:float):
    """
    return False if there is distance between p1 and p2
    in the same axis that greater than d
    """
    for i in range(p1.dimensi):
        if (abs(p1.value[i] - p2.value[i]) > d):
            return False
    return True
```

```
def driver() :
    A1 = point(3,[1,3,6])
    A2 = point(3,[8,1,2])
    A3 = point(3,[8,1,1])

    List = [A1, A2, A3]

    for titik in List:
        print(titik)

    sorted(List)
    nearestCouple = divideConquer(List)
    print("nearest by Divide n Conquer : ", nearestCouple)
```

D. visual.py

```
import numpy as np
import matplotlib.pyplot as plt
from dataType import point, couple
from bruteForce import bruteForce

def display1D(arrayOfPoint : list[point], pair : couple):
    """
    display scatter plot 1D of arrayOfPoint with pair highlighted as
red
    """
    fig = plt.figure()

    plot = fig.add_subplot()

    for point in arrayOfPoint:
        xp = point.value[0]
        if (point == pair.point1 or point == pair.point2):
            plot.scatter(xp, marker='o', c='red')
        else:
            plot.scatter(xp, marker='o', c='blue')

    plot.set_xlabel('SUMBU-X')
    plt.show()

def display2D(arrayOfPoint : list[point], pair : couple):
```

```
    """
    display scatter plot 2D of arrayOfPoint with pair highlighted as
red
    """
    fig = plt.figure()

    plot = fig.add_subplot()

    for point in arrayOfPoint:
        xp = point.value[0]
        yp = point.value[1]
        if (point == pair.point1 or point == pair.point2):
            plot.scatter(xp,yp, marker='o', c='red')
        else:
            plot.scatter(xp,yp, marker='o', c='blue')

    plot.set_xlabel('SUMBU-X')
    plot.set_ylabel('SUMBU-Y')
    plt.show()


def display3D(arrayOfPoint : list[point], pair : couple):
    """
    display scatter plot 3D of arrayOfPoint with pair highlighted as
red
    """
    fig = plt.figure()

    plot = fig.add_subplot(projection='3d')

    for point in arrayOfPoint:
        xp = point.value[0]
        yp = point.value[1]
        zp = point.value[2]
        if (point == pair.point1 or point == pair.point2):
            plot.scatter(xp,yp,zp, marker='o', c='red')
        else:
            plot.scatter(xp,yp,zp, marker='o', c='blue')

    plot.set_xlabel('SUMBU-X')
    plot.set_ylabel('SUMBU-Y')
    plot.set_zlabel('SUMBU-Z')
    plt.show()

def driver3D():
```

```
    A1 : point = point(3, [4,5,6])
    A2 : point = point(3, [4,6,6])
    A3 : point = point(3, [7,5,6])

    listP = [A1, A2, A3]

    display3D(listP, bruteForce(listP))
```

E. IO.py

```
import customtkinter
import random as rand
import numpy as np
import time as t

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from divideConquer import divideConquer, quicksort
from matplotlib.figure import Figure
from dataType import couple, point
from bruteForce import bruteForce

customtkinter.set_appearance_mode("Light")  # Modes: "System"
(standard), "Dark", "Light"
customtkinter.set_default_color_theme("blue")  # Themes: "blue"
(standard), "green", "dark-blue"

class App(customtkinter.CTk):
    def __init__(self):
        super().__init__()

        # configure window
        self.title("Closest Pair Finder")
        self.geometry(f"{1295}x{720}")

        # configure grid layout (4x4)
        self.grid_columnconfigure((1,2,3,4,5,6,7), weight=1)
        # self.grid_columnconfigure((2, 3), weight=0)
        self.grid_rowconfigure((0,1,2), weight=1)
```

```
        # create sidebar frame with widgets
        self.sidebar_frame = customtkinter.CTkFrame(self, width=480,
corner_radius=0)
        self.sidebar_frame.grid(row=0, column=0, rowspan=4,
sticky="nsew")
        self.sidebar_frame.grid_rowconfigure((4,7), weight=1)

        self.logo_label = customtkinter.CTkLabel(self.sidebar_frame,
text="DIVIDE AND CONQUER", font=customtkinter.CTkFont(size=20,
weight="bold"))
        self.logo_label.grid(row=0, column=0, columnspan=2, padx=0,
pady=(20, 0))
        self.sub_label = customtkinter.CTkLabel(self.sidebar_frame,
text="Rizky & Rizky 's", font=customtkinter.CTkFont(size=14,
weight="normal"))
        self.sub_label.grid(row=1, column=0, columnspan=2, padx=0,
pady=(0, 20))

        self.label_number =
customtkinter.CTkLabel(self.sidebar_frame, text="Jumlah Titik :",
font=customtkinter.CTkFont(size=14, weight="bold"))
        self.label_number.grid(row=2, column=0, padx=(20, 15),
pady=(0, 5), sticky='e')

        self.label_dimensi =
customtkinter.CTkLabel(self.sidebar_frame, text="Jumlah Dimensi :",
font=customtkinter.CTkFont(size=14, weight="bold"))
        self.label_dimensi.grid(row=3, column=0, padx=(20, 15),
pady=(0, 5), sticky='e')

        self.entry_number =
customtkinter.CTkEntry(self.sidebar_frame, placeholder_text="0")
        self.entry_number.grid(row=2, column=1, padx=(0, 40),
pady=(0, 5), sticky="nsew")

        self.entry_dimensi =
customtkinter.CTkEntry(self.sidebar_frame, placeholder_text="3")
        self.entry_dimensi.grid(row=3, column=1, padx=(0, 40),
pady=(0, 5), sticky="nsew")

        self.time_bf_label =
customtkinter.CTkLabel(self.sidebar_frame, text="BF Time :",
font=customtkinter.CTkFont(size=14, weight="bold"))
        self.time_bf_label.grid(row=5, column=0, padx=(20, 15),
pady=(0, 5), sticky='e')
```

```
        self.time_dnc_label =
customtkinter.CTkLabel(self.sidebar_frame, text="DnC Time :",
font=customtkinter.CTkFont(size=14, weight="bold"))
        self.time_dnc_label.grid(row=6, column=0, padx=(20, 15),
pady=(0, 5), sticky='e')

        self.timeRes_bf_label =
customtkinter.CTkLabel(self.sidebar_frame, text="0 second",
font=customtkinter.CTkFont(size=14, weight="normal"))
        self.timeRes_bf_label.grid(row=5, column=1, padx=(5, 10),
pady=(0, 5), sticky='w')
        self.timeRes_dnc_label =
customtkinter.CTkLabel(self.sidebar_frame, text="0 second",
font=customtkinter.CTkFont(size=14, weight="normal"))
        self.timeRes_dnc_label.grid(row=6, column=1, padx=(5, 10),
pady=(0, 5), sticky='w')

        self.run_button = customtkinter.CTkButton(self.sidebar_frame,
text="Generate & Run", command=self.run)
        self.run_button.grid(row=8, column=0, columnspan=2, padx=20,
pady=30, sticky="s")

        self.vis_frame = customtkinter.CTkFrame(self,
corner_radius=0)
        self.vis_frame.grid(row=0, column=1, columnspan=6, pady=0)

        self.fig = Figure(figsize=(12,5), dpi=100)
        self.ax = self.fig.add_subplot(111, projection='3d')

        # Create a canvas to display plot
        self.canvas = FigureCanvasTkAgg(self.fig,
master=self.vis_frame)
        self.canvas.get_tk_widget().pack()

        self.terminal_frame = customtkinter.CTkFrame(self,
corner_radius=10)
        self.terminal_frame.grid(row=1, column=1, columnspan=6,
pady=0, sticky='nswe')
        self.terminal_frame.grid_columnconfigure((0,1,2,3,4,5,6,7),
weight=1)
        self.terminal_frame.grid_rowconfigure((0,1,2), weight=1)

        self.bf_label = customtkinter.CTkLabel(self.terminal_frame,
text="Brute Force", font=customtkinter.CTkFont(size=20,
weight="bold"))
```

```
        self.bf_label.grid(row=0, column=0, columnspan=3,
sticky='we', pady=5, padx=5)
        self.comp_label = customtkinter.CTkLabel(self.terminal_frame,
text="Efficiency", font=customtkinter.CTkFont(size=20,
weight="bold"))
        self.comp_label.grid(row=0, column=3, columnspan=2,
sticky='we', pady=5, padx=5)
        self.dnc_label = customtkinter.CTkLabel(self.terminal_frame,
text="Divide & Conquer", font=customtkinter.CTkFont(size=20,
weight="bold"))
        self.dnc_label.grid(row=0, column=5, columnspan=3,
sticky='we', pady=5, padx=5)

        self.bf_frame = customtkinter.CTkFrame(self.terminal_frame,
corner_radius=10, fg_color='#ebebeb')
        self.bf_frame.grid(row=1, column=0, columnspan=3,
sticky='nswe', pady=5, padx=5)
        self.comp_frame = customtkinter.CTkFrame(self.terminal_frame,
corner_radius=10, fg_color='#ebebeb')
        self.comp_frame.grid(row=1, column=3, columnspan=2,
sticky='nswe', pady=5, padx=5)
        self.dnc_frame = customtkinter.CTkFrame(self.terminal_frame,
corner_radius=10, fg_color='#ebebeb')
        self.dnc_frame.grid(row=1, column=5, columnspan=3,
sticky='nswe', pady=5, padx=5)

        self.bf_frame.grid_rowconfigure((0), weight=1)
        self.bf_frame.grid_columnconfigure((1), weight=1)

        self.bf_result1 = customtkinter.CTkLabel(self.bf_frame,
wraplength=300, text="", font=customtkinter.CTkFont(size=14,
weight="normal"))
        self.bf_result1.grid(row=2, column=0, sticky='ne',
pady=(0,10), padx=(15,0))
        self.bf_result1_value = customtkinter.CTkLabel(self.bf_frame,
wraplength=300, text="",font=customtkinter.CTkFont(size=14,
weight="normal"))
        self.bf_result1_value.grid(row=2, column=1, sticky='w',
pady=(0,10), padx=(10,0))

        self.bf_result2 = customtkinter.CTkLabel(self.bf_frame,
wraplength=300,text="", font=customtkinter.CTkFont(size=14,
weight="normal"))
        self.bf_result2.grid(row=1, column=0, sticky='se', pady=0,
padx=(15,0))
```

```
        self.bf_result2_value = customtkinter.CTkLabel(self.bf_frame,
wraplength=300, text="",font=customtkinter.CTkFont(size=14,
weight="normal"))
        self.bf_result2_value.grid(row=1, column=1, sticky='w',
pady=0, padx=(10,0))

        self.bf_result3 = customtkinter.CTkLabel(self.bf_frame,
wraplength=300, text="",font=customtkinter.CTkFont(size=12,
weight="normal"))
        self.bf_result3.grid(row=0, columnspan = 2, sticky='s',
pady=(0,10), padx=0)

        self.dnc_frame.grid_rowconfigure((0), weight=1)
        self.dnc_frame.grid_columnconfigure((1), weight=1)

        self.dnc_result1 = customtkinter.CTkLabel(self.dnc_frame,
wraplength=300, text="", font=customtkinter.CTkFont(size=14,
weight="normal"))
        self.dnc_result1.grid(row=2, column=0, sticky='ne',
pady=(0,10), padx=(15, 0))
        self.dnc_result1_value =
customtkinter.CTkLabel(self.dnc_frame, wraplength=300,
text="",font=customtkinter.CTkFont(size=14, weight="normal"))
        self.dnc_result1_value.grid(row=2, column=1, sticky='w',
pady=(0,10), padx=(10,0))

        self.dnc_result2 = customtkinter.CTkLabel(self.dnc_frame,
wraplength=300,text="", font=customtkinter.CTkFont(size=14,
weight="normal"))
        self.dnc_result2.grid(row=1, column=0, sticky='se', pady=0,
padx=(15,0))
        self.dnc_result2_value =
customtkinter.CTkLabel(self.dnc_frame, wraplength=300,
text="",font=customtkinter.CTkFont(size=14, weight="normal"))
        self.dnc_result2_value.grid(row=1, column=1, sticky='w',
pady=0, padx=(10,0))

        self.dnc_result3 = customtkinter.CTkLabel(self.dnc_frame,
wraplength=300, text="",font=customtkinter.CTkFont(size=12,
weight="normal"))
        self.dnc_result3.grid(row=0, columnspan = 2, sticky='s',
pady=(0,10), padx=0)

        self.comp_frame.grid_rowconfigure((0), weight=1)
        self.comp_frame.grid_columnconfigure((1), weight=1)
```

```
        self.comp_result1 = customtkinter.CTkLabel(self.comp_frame,
wraplength=200, text="", font=customtkinter.CTkFont(size=13,
weight="normal"))
        self.comp_result1.grid(row=0, column=0, sticky='se',
pady=(10,0), padx=(10,0))
        self.comp_result1_value =
customtkinter.CTkLabel(self.comp_frame, wraplength=200, text="",
font=customtkinter.CTkFont(size=13, weight="normal"))
        self.comp_result1_value.grid(row=0, column=1, sticky='sw',
pady=(10,0), padx=(10,0))

        self.comp_result2 = customtkinter.CTkLabel(self.comp_frame,
wraplength=200, text="", font=customtkinter.CTkFont(size=13,
weight="normal"))
        self.comp_result2.grid(row=1, column=0, sticky='ne',
pady=(0,30), padx=(10,0))
        self.comp_result2_value =
customtkinter.CTkLabel(self.comp_frame, wraplength=200, text="",
font=customtkinter.CTkFont(size=13, weight="normal"))
        self.comp_result2_value.grid(row=1, column=1, sticky='nw',
pady=(0,30), padx=(10,0))


    def run(self):
        points = np.empty((0), dtype=point)

        for i in range(int(self.entry_number.get())):
            val = np.empty(int(self.entry_dimensi.get()),
dtype=float)
            for j in range(int(self.entry_dimensi.get())):
                val[j] = rand.uniform(-1000, 1000)
            points = np.append(points,
point(int(self.entry_dimensi.get()), val))

        startBF = t.perf_counter()
        closestCoupleBF, numBF = bruteForce(points)
        stopBF = t.perf_counter()

        startDnC = t.perf_counter()
        closestCoupleDnC, numDnC = divideConquer(quicksort(points))
        stopDnC = t.perf_counter()

        timeDnC = stopDnC-startDnC
        self.timeRes_dnc_label.configure(text="{:.6f}
```

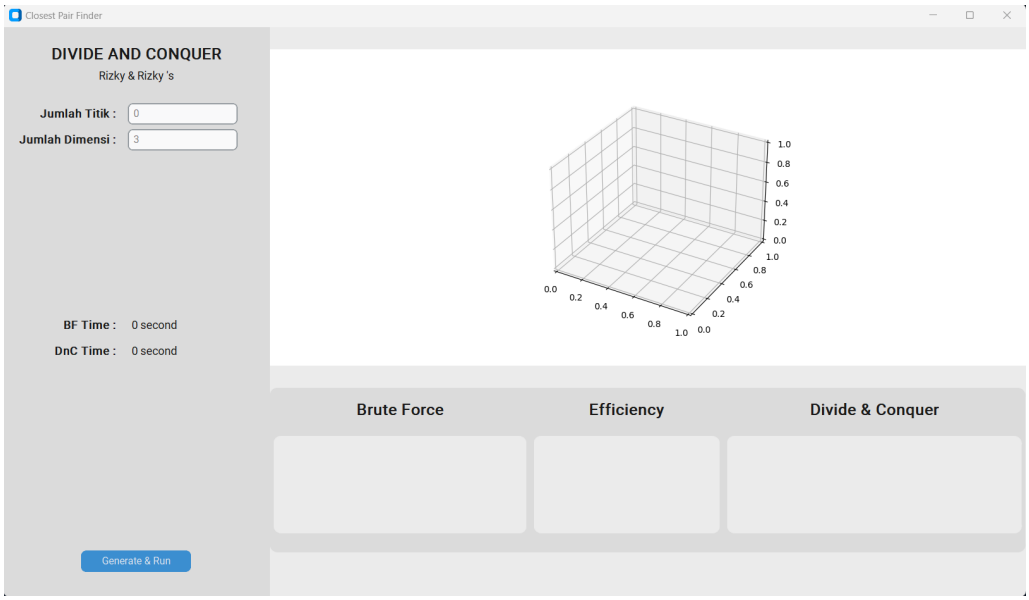
```
detik".format(timeDnC))
        timeBF = stopBF-startBF
        self.timeRes_bf_label.configure(text="{:.6f}
detik".format(timeBF))

        self.display(points, closestCoupleDnC)
        self.detailResult(closestCoupleDnC, closestCoupleBF, numDnC,
numBF, timeDnC, timeBF)

    def display(self, arrayOfPoint : list[point], pair : couple):
        if (self.entry_dimensi.get() == "3"):
            self.ax.clear()
            self.ax = self.fig.add_subplot(111, projection='3d')
            for point in arrayOfPoint:
                xp = point.value[0]
                yp = point.value[1]
                zp = point.value[2]
                if (point == pair.point1 or point == pair.point2):
                    self.ax.scatter(xp,yp,zp, marker='o', c='red')
                else:
                    self.ax.scatter(xp,yp,zp, marker='o', c='blue')

            self.ax.set_xlabel('SUMBU-X')
            self.ax.set_ylabel('SUMBU-Y')
            self.ax.set_zlabel('SUMBU-Z')

            self.canvas.draw()

        elif (self.entry_dimensi.get() == "2"):
            self.ax.clear()
            self.ax = self.fig.add_subplot()
            for point in arrayOfPoint:
                xp = point.value[0]
                yp = point.value[1]
                if (point == pair.point1 or point == pair.point2):
                    self.ax.scatter(xp,yp, marker='o', c='red')
                else:
                    self.ax.scatter(xp,yp, marker='o', c='blue')
            self.ax.set_xlabel('SUMBU-X')
            self.ax.set_ylabel('SUMBU-Y')

            self.canvas.draw()

        elif (self.entry_dimensi.get() == "1"):
            self.ax.clear()
```

```
            self.ax = self.fig.add_subplot()
            static_y = 0
            for point in arrayOfPoint:
                xp = point.value[0]
                if (point == pair.point1 or point == pair.point2):
                    self.ax.scatter(xp, static_y, marker='o',
c='red')
                else:
                    self.ax.scatter(xp, static_y, marker='o',
c='blue')

            self.ax.set_xlabel('SUMBU-X')

            self.canvas.draw()

    def detailResult(self, dncPair: couple, bfPair: couple, numDnc:
int, numBf: int, timeDnc: float, timeBf: float):
        self.bf_result3.configure(text="{0}".format(bfPair))
        self.bf_result2.configure(text="Euclidean Distance
:")
        self.bf_result1.configure(text="Number of Euclidean       :")

self.bf_result2_value.configure(text="{0}".format(bfPair.distance))
        self.bf_result1_value.configure(text="{0}".format(numBf))

        self.dnc_result3.configure(text="{0}".format(dncPair))
        self.dnc_result2.configure(text="Euclidean Distance
:")
        self.dnc_result1.configure(text="Number of Euclidean
:")

self.dnc_result2_value.configure(text="{0}".format(dncPair.distance))
        self.dnc_result1_value.configure(text="{0}".format(numDnc))

        self.comp_result1.configure(text="Operation Eff     :")
        self.comp_result1_value.configure(text="{0:.3f}
%".format(((numBf-numDnc)/numBf)*100))

        self.comp_result2.configure(text="Time Eff             :")
        self.comp_result2_value.configure(text="{0:.3f}
%".format(((timeBf-timeDnc)/timeBf)*100))
```

F. main.py

```
from IO import *

if __name__ == "__main__":
    app = App()
    app.mainloop()
```

BAB IV
EKSPERIMEN

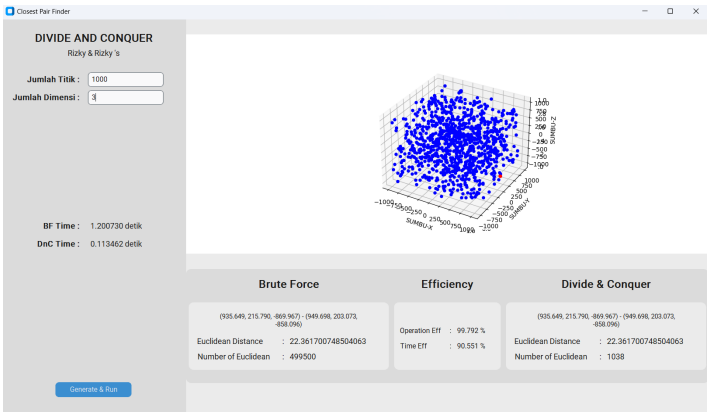


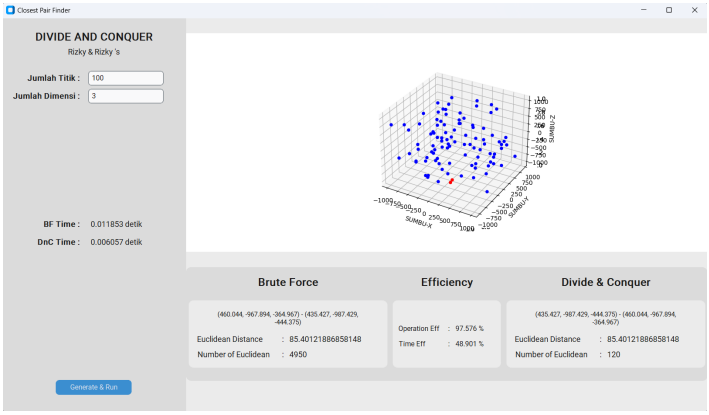
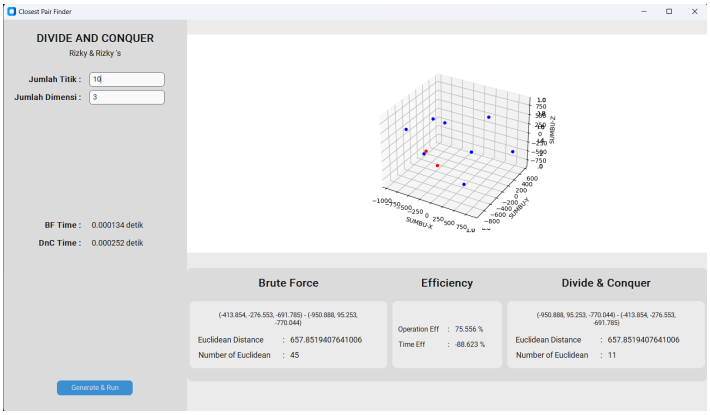
Eksperimen dilakukan pada laptop dengan spesifikasi :

Processor : 8 core 16 thread

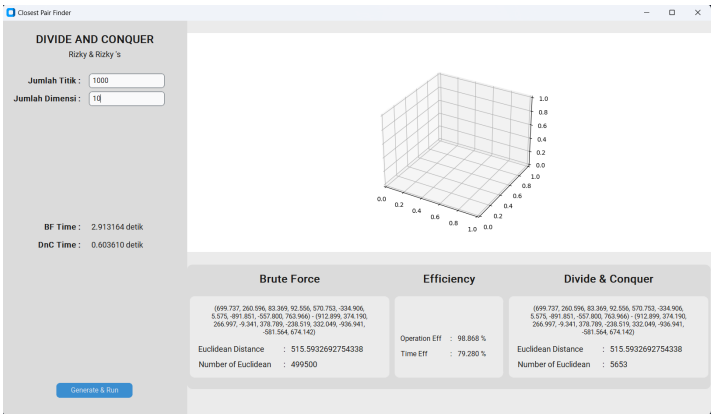
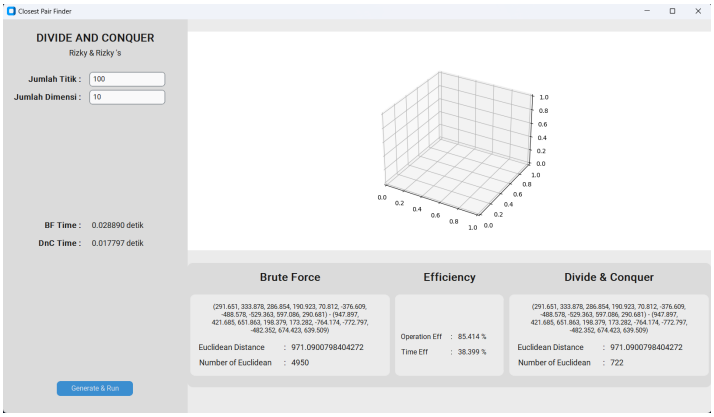
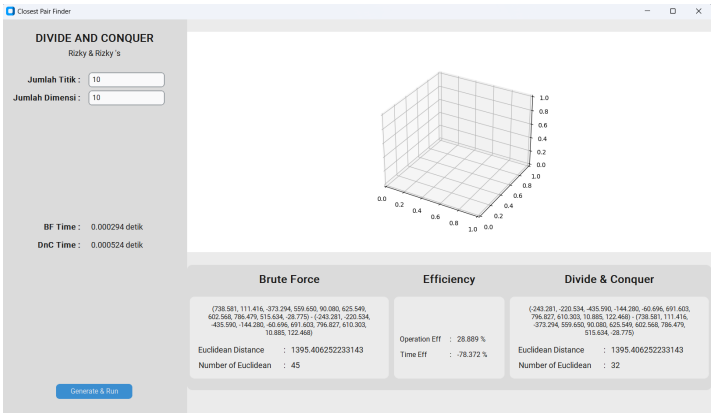
RAM : 24 GB

A. Titik dalam 3 dimensi

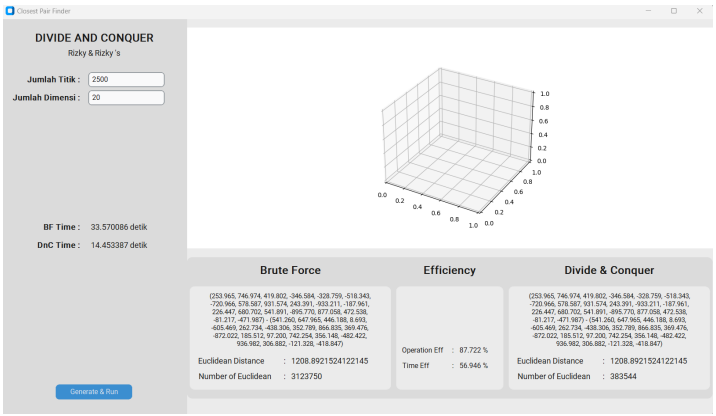
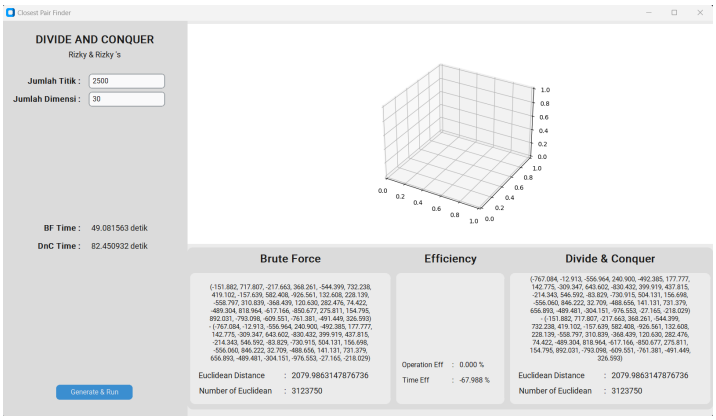
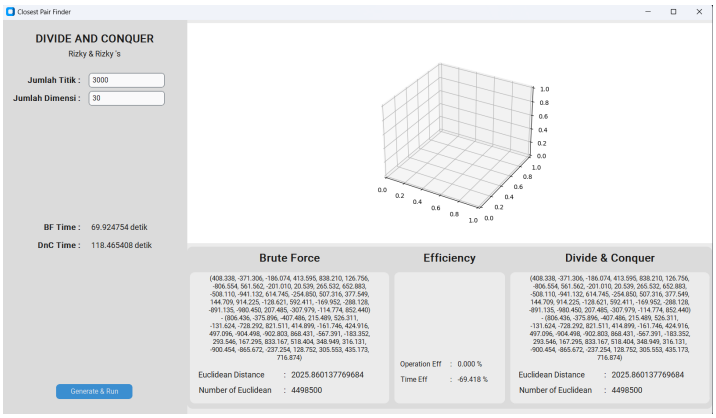
Tampilan	Deskripsi
	Algoritma divide and conquer optimal

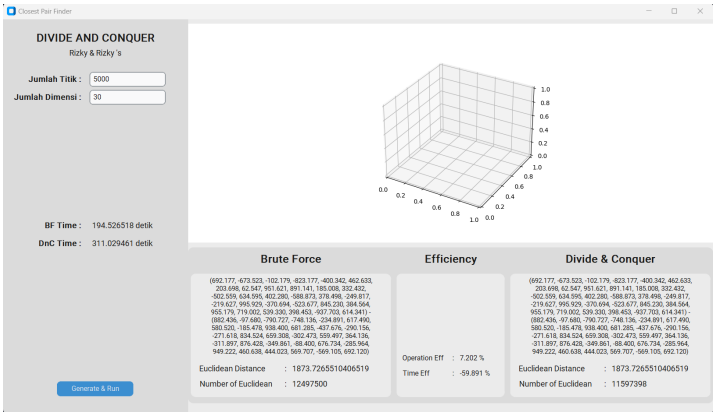
	Algoritma divide and conquer optimal
	Algoritma divide and conquer optimal, namun dalam prosesnya algoritma brute force membutuhkan waktu yang lebih sedikit untuk jumlah titik yang kecil

B. Titik dalam dimensi lebih dari 3

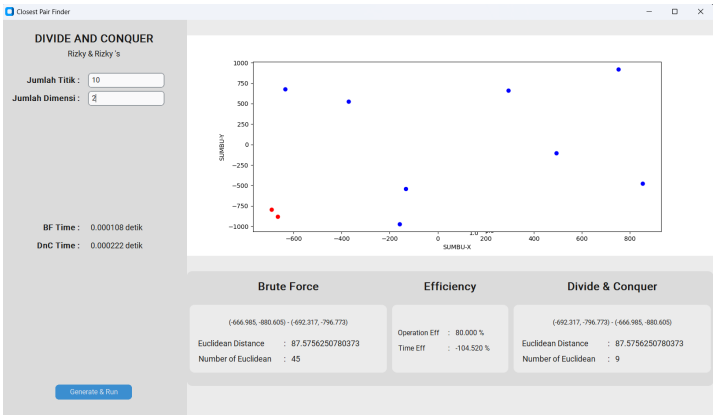
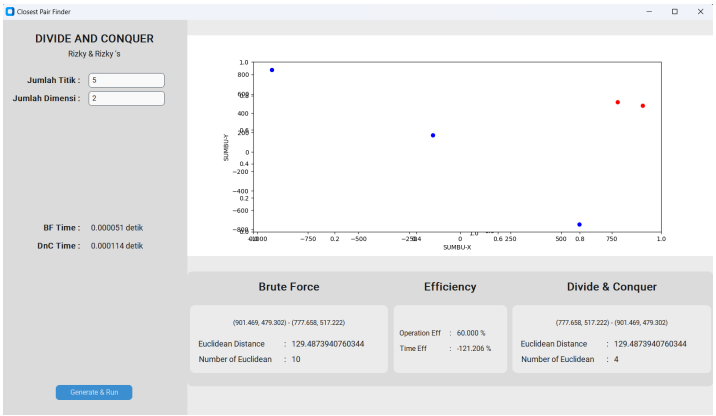
Tampilan	Deskripsi
	Algoritma divide and conquer optimal
	Algoritma divide and conquer optimal
	Algoritma divide and conquer optimal, namun dalam prosesnya algoritma brute force membutuhkan waktu yang lebih sedikit untuk jumlah titik yang kecil

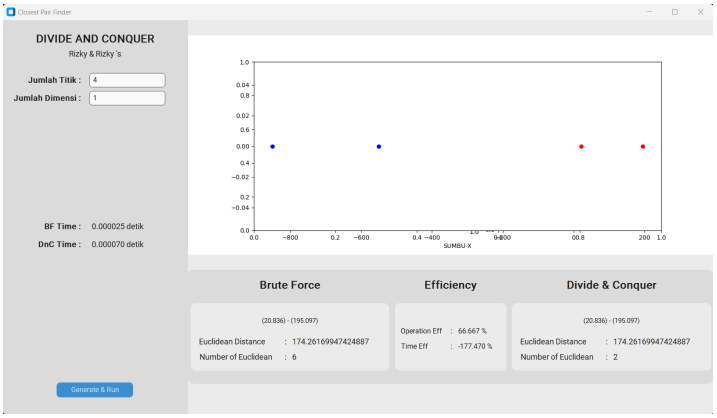
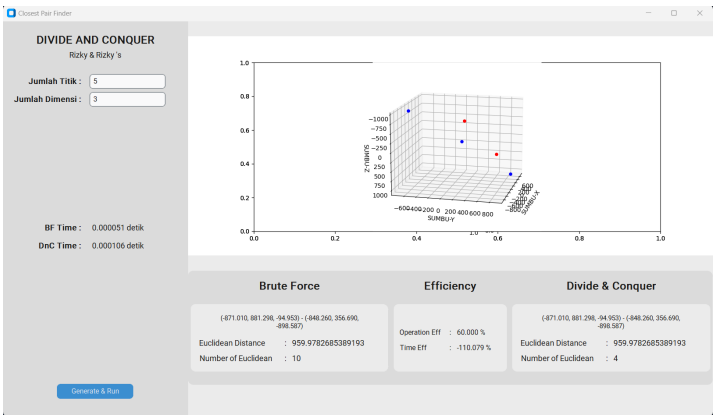
C. Kasus-kasus ekstrim

Tampilan	Deskripsi
	Algoritma divide and conquer optimal
	Algoritma divide and conquer tidak lagi optimal karena dimensi titik yang besar sedangkan persebaran titik tidak cukup padat
	Algoritma divide and conquer tidak lagi optimal karena dimensi titik yang besar sedangkan persebaran titik tidak cukup padat

	Algoritma divide and conquer optimal, namun dalam prosesnya algoritma brute force membutuhkan waktu yang lebih sedikit karena dikarenakan proses divide dan conquer yang cukup banyak

D. Kasus Ekstrim kecil

Tampilan	Deskripsi
	Algoritma divide and conquer optimal, namun dalam prosesnya algoritma brute force membutuhkan waktu yang lebih sedikit untuk jumlah titik yang kecil
	Algoritma divide and conquer optimal, namun dalam prosesnya algoritma brute force membutuhkan waktu yang lebih sedikit untuk jumlah titik yang kecil

	Algoritma divide and conquer optimal, namun dalam prosesnya algoritma brute force membutuhkan waktu yang lebih sedikit untuk jumlah titik yang kecil
	Algoritma divide and conquer optimal, namun dalam prosesnya algoritma brute force membutuhkan waktu yang lebih sedikit untuk jumlah titik yang kecil

BAB V
KESIMPULAN

Algoritma Divide and Conquer cukup optimal untuk seluruh kasus untuk mencari pasangan titik terdekat pada multidimensi. Hanya saja untuk kasus-kasus tertentu *execution time* dari algoritma ini masih terlalu besar dibandingkan dengan algoritma brute-force dikarenakan beberapa hal yaitu tingginya jumlah dimensi sehingga besar kemungkinan strip pencarian mencakup seluruhnya dan juga proses pengolahan data yang cukup banyak oleh algoritma divide and conquer yang cukup memakan waktu.

BAB VI

LAMPIRAN

A. Pranala Github

https://github.com/mrsyaban/Tucil2_13521109-13521119

B. Referensi

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf

http://euro.ecom.cmu.edu/people/faculty/mshamos/1976ShamosBentley.pdf

C. Tabel 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		