PENCARIAN PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA *DIVIDE AND CONQUER*

Diajukan sebagai pemenuhan tugas kecil 2.

Oleh:

Antonio Natthan Krishna

13521162

Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S.T, M.Sc.

IF2211 - Strategi Algoritma

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

BAGIAN 1
DESKRIPSI MASALAH

Salah satu pemanfaatan algoritma Divide and Conquer adalah pencarian titik terdekat pada bidang R^n. Penjelasan mengenai hal ini telah dijelaskan pada kuliah, namun penerapannya hanya dijelaskan pada bidang 2D. Pada tugas ini, saya diminta 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). Terdapat pasangan titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titik P1 = (x1, y1, z1) dan P2 = (x2, y2, z2) dihitung dengan rumus Euclidean berikut:

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

BAGIAN 2
PENERAPAN ALGORITMA PADA PERSOALAN

2.1. Algoritma *Divide and Conquer*

Algoritma Divide and Conquer adalah strategi pemecahan masalah yang besar dengan cara melakukan pembagian masalah tersebut menjadi beberapa bagian yang lebih kecil secara rekursif sehingga pada suatu titik tertentu masalah tersebut dapat secara langsung dipecahkan. Solusi dari bagian bagian kecil tersebut digabungkan untuk menjadi sebuah solusi yang utuh

Langkah-langkah umum algoritma Divide and Conquer :

1. Divide : Membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama).
2. Conquer : Memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif).
3. Combine : Menggabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.

2.2. Metode *Divide and Conquer* yang Dipakai

Metode *Divide and Conquer* yang dipakai didasarkan pada jumlah *axis* (dimensi titik) dengan rincian sebagai berikut,

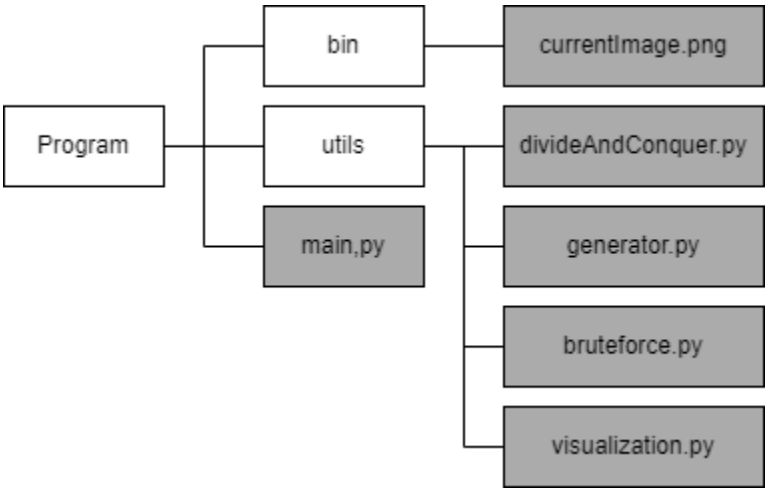
1. Untuk setiap titik, cari nilai minimum dan maksimum dari properti sumbu paling akhir pada tuple lokasi titik tersebut.
2. Kemudian bagi titik tersebut menjadi dua bagian berdasarkan area yang dibatasi oleh garis tengah yang didefinisikan berdasarkan nilai maksimum dan minimum dari properti sumbu paling akhir.
3. Lakukan pembagian secara rekursif dengan decrement dimensi titik - 1 (dimensi melambangkan banyaknya properti *tuple* titik yang belum diperiksa).
4. Setelah dimensi titik = 0 (semua properti tuple sudah diperiksa), lakukan pencarian titik terdekat oleh titik titik yang berada di area tersebut.
5. Dapatkan hasil titik terdekat solusi, banyak pemanggilan *fungsi euclidean*, dan solusi dari setiap area.
6. Gabungkan hasil dari area kanan dan area kiri.

7. Lakukan pencarian di sekitar sumbu pembagi yang memiliki distance lebih kecil daripada jarak terdekat
8. Lakukan update pada solusi apabila memenuhi
9. Gabungkan seluruh solusi kembali hingga kembali ke dimensi semula.
10. Seluruh solusi telah didapatkan.

BAGIAN 3
IMPLEMENTASI ALGORITMA PADA PROGRAM

3.1. Struktur Program

Program ditulis dengan menggunakan bahasa Python dengan menggunakan Tkinter untuk pemrosesan antarmuka dengan struktur sebagai berikut,



Gambar 1. Struktur Program

Deskripsi File ditunjukkan oleh tabel berikut,

File	Fungsi
main.py	Pemegang kendali GUI dari program
divideAndConquer.py	Pengatur kendali algoritma divide and conquer
visualization.py	Pemrosesan grafik
generator.py	Pemrosesan input random dan output
bruteforce.py	Melakukan perhitungan jarak terdekat dengan algoritma brute force
currentImage.png	Grafik yang sedang ditampilkan pada aplikasi (atau yang terakhir kali)

3.2. Implementasi Program

a. main.py

Fungsi: pemegang kendali GUI dari program

```
from utils.divideAndConquer import *
```

```
from utils.visualization import *
from utils.generator import *
from utils.bruteforce import *

# import GUI
import tkinter
import tkinter.messagebox
import customtkinter
import time as t
import numpy as np
from PIL import ImageTk, Image

# setting customtkinter
customtkinter.set_appearance_mode("Dark")
customtkinter.set_default_color_theme("blue")

class App(customtkinter.CTk):
    WIDTH = 1500
    HEIGHT = 750

    def __init__(self):
        super().__init__()
        #DEKLARASI VARIABEL
        self.title("Closest Pair of Point Using Divide and Conquer")
        self.geometry(f"{App.WIDTH}x{App.HEIGHT}")
        self.protocol("WM_DELETE_WINDOW", self.on_closing)

        self.grid_columnconfigure(1, weight=1)
        self.grid_rowconfigure(0, weight=1)
        self.dimension = None
        self.numOfPoints = None
        self.listPoint = []
        self.graph = None
        self.solution = None
        self.countEuclidean = None
        self.shortestDistance = None
        self.time = 0
        self.maxAxisVal = 10

        self.frame_right = customtkinter.CTkFrame(master=self)
        self.frame_right.grid(row=0, column=1, sticky="nswe", padx=20, pady=20)

        # Konfigurasi
        self.optionmenu_1 =
                    customtkinter.CTkOptionMenu(master=self.frame_right,
                                        values=["Light", "Dark"],
                                        command=self.change_appearance_mode)
        self.optionmenu_1.grid(row=3, column=0, pady=10, padx=20, sticky="w")

        # Frame
        self.frame_right.columnconfigure(1, weight=6)
        self.frame_right.columnconfigure(0, weight=1)
        self.frame_right.rowconfigure(0, weight=1)
        self.frame_right.rowconfigure(1, weight=2)
        self.frame_right.rowconfigure(2, weight=1)


        self.frame_input = customtkinter.CTkFrame(master=self.frame_right)
        self.frame_input.grid(row=0, column=0, rowspan= 3, pady=20, padx=20,
                                                        sticky="nsew")

        self.frame_info = customtkinter.CTkFrame(master=self.frame_right)
        self.frame_info.grid(row=0, column=1, rowspan= 3, pady=20, padx=20,
                                                      sticky="nsew")

        # Frame Input
        self.frame_input.rowconfigure(0, weight=1)
        self.frame_input.rowconfigure(7, weight=7)
        self.frame_input.columnconfigure(0, weight=1)
        self.frame_input.columnconfigure(1, weight=1)
```

```
        self.headFrameInput = customtkinter.CTkLabel(master=self.frame_input,
                                            text="Customize Your Data",
                                            text_font=("Roboto Medium", -20))
        self.headFrameInput.grid(row=0, column=0, columnspan=2, pady=20,
                                                padx=10, sticky="")

        self.dimension = customtkinter.CTkLabel(master=self.frame_input,
                                            text="Dimension of Point",
                                            text_font=("Roboto Medium", -16))
        self.dimension.grid(row=1, column=0, pady=0, padx=10)

        self.dimensionBuffer = customtkinter.CTkEntry(master=self.frame_input,
                                            placeholder_text="Integer (> 0)",
                                            width=250)
        self.dimensionBuffer.grid(row=1, column=1, pady=5, sticky="w")


        self.numPointsLabel = customtkinter.CTkLabel(master=self.frame_input,
                                            text="Number of points",
                                            text_font=("Roboto Medium", -16))
        self.numPointsLabel.grid(row=2, column=0, pady=0, padx=10)

        self.numPointsBuffer = customtkinter.CTkEntry(master=self.frame_input,
                                            placeholder_text="Integer (> 1)",
                                            width=250)
        self.numPointsBuffer.grid(row=2, column=1, pady=5, sticky="w")

        self.maxAxis = customtkinter.CTkLabel(master=self.frame_input,
                                            text="Axis Width",
                                            text_font=("Roboto Medium", -16))
        self.maxAxis.grid(row=3, column=0, pady=0, padx=10)

        self.maxAxis = customtkinter.CTkOptionMenu(master=self.frame_input,
                                            values=["10", "100", "1000"],
                                            command=self.getMaxAxis)
        self.maxAxis.grid(row=3, column=1, pady=5, sticky="w")

        self.generateButton = customtkinter.CTkButton(master=self.frame_input,
                                              text="Generate",
                                              border_width=2,
                                              width=120,
                                              text_color="white",
                                              fg_color="#1f6aa5",
                                              state="normal",
                                              command=self.checkvalid)
        self.generateButton.grid(row=4, column=0, columnspan=2, padx=20,
                                              pady=(20, 0))

        self.bufferValidityLabel =
                customtkinter.CTkLabel(master=self.frame_input,
                            text="All points will be unique set of integers.",
                            text_font=("Roboto Medium", -10))
        self.bufferValidityLabel.grid(row=5, column=0, columnspan=3, pady=0,
                                                    padx=10, sticky="we")

        self.generatedPointLabel =
                              customtkinter.CTkLabel(master=self.frame_input,
                                            text="Generated Points",
                                            text_font=("Roboto Medium", -16))
        self.generatedPointLabel.grid(row=6, column=0, columnspan=3, pady=0,
                                                    padx=10)

        self.generatedPointFrame = customtkinter.CTkFrame(self.frame_input,
                                                       border_width=2,
                                                       fg_color=("white",
                                                       "gray38"))
        self.generatedPointFrame.grid(row=7, column=0, columnspan=3, pady=20,
                                                    padx=20, sticky="nsew")

        self.generatedPointCanvas =
```

```
                        customtkinter.CTkCanvas(self.generatedPointFrame)

        self.generatedPointScrollbar =
                        customtkinter.CTkScrollbar(self.generatedPointFrame,
                        command=self.generatedPointCanvas.yview)
        self.generatedPointTextFrame =
                        customtkinter.CTkFrame(self.generatedPointCanvas,
                                               corner_radius=0,
                                               fg_color="white")

        self.generatedPointCanvas.configure(
                         yscrollcommand=self.generatedPointScrollbar.set,
                         bg="white", highlightthickness=0)

        self.generatedPointTextLabel = customtkinter.CTkLabel(
                                       master=self.generatedPointTextFrame,
                                       text="",
                                       text_font=("Roboto Medium", -12),
                                       fg_color="white",
                                       justify="left",
                                       anchor="w",
                                       text_color="gray38")

        self.generatedPointTextLabel.pack(fill="both", expand=True)

        self.generatedPointCanvas.bind(
            "<Configure>",
            lambda e: self.generatedPointCanvas.configure(
                scrollregion=self.generatedPointCanvas.bbox("all")
            )
        )
        self.generatedPointCanvas.configure(
                        height=self.generatedPointTextLabel.winfo_height())

        self.generatedPointCanvas.create_window((0, 0),
                           window=self.generatedPointTextFrame, anchor="nw")
        self.generatedPointCanvas.pack(side="left", fill="both", expand=True)
        self.generatedPointScrollbar.pack(side="right", fill="y")

        # Layout Gambar
        self.frame_info.rowconfigure(0, weight=0)
        self.frame_info.rowconfigure(1, weight=1)
        self.frame_info.columnconfigure(0, weight=1)

        self.frame_info.configure(height=self.frame_input.winfo_height())

        self.graphLabel = customtkinter.CTkLabel(master=self.frame_info,
                                                 text="Graph Plot",
                                                 text_font=("Roboto Medium", -28))
        self.graphLabel.grid(row=0, column=0, padx=20, pady=20)


        self.solutionLabel = customtkinter.CTkLabel(master=self.frame_info,
                                               corner_radius=6,
                                               text = "",
                                               text_font=("Roboto Medium", -28),
                                               fg_color=("white", "gray38"),
                                               justify=tkinter.LEFT)
        self.solutionLabel.grid(column=0, row=1, sticky="nsew", pady=15)

        self.label_infot3 = customtkinter.CTkLabel(master=self.frame_right,
                                                 text="Execution time: ",
                                                 text_font=("Roboto Medium", -15),
                                                 justify = "left")
        self.label_infot3.grid(row=3, column=1, columnspan=2, sticky="nw")

        # set default values
        self.optionmenu_1.set("Dark")

    def getMaxAxis(self, newMaxAxis):
```

```
        self.maxAxisVal = int(newMaxAxis)

    def getVisualization(self):
        if self.dimension <= 3:
            getGraph(self.dimension, self.numOfPoints, self.listPoint,
self.solution, self.maxAxisVal)
            self.graph = ImageTk.PhotoImage(Image
                       .open("bin/currentPlot.png")
                       .resize((int(self.solutionLabel.winfo_height()*1.3),
                         self.solutionLabel.winfo_height()), Image.ANTIALIAS),
                         master=self.solutionLabel)
            self.solutionLabel.configure(image=self.graph)
            self.solutionLabel.configure(text="")
        else:
          self.solutionLabel.configure(image="")
          self.solutionLabel.configure(text="Not Available For Dimension > 3")

    def displayPoints(self, textToDisplay):
        self.generatedPointCanvas.destroy()
        self.generatedPointTextFrame.destroy()
        self.generatedPointTextLabel.destroy()
        self.generatedPointScrollbar.destroy()

        self.generatedPointCanvas =
                            customtkinter.CTkCanvas(self.generatedPointFrame)
        self.generatedPointScrollbar =
                            customtkinter.CTkScrollbar(self.generatedPointFrame,
                            command=self.generatedPointCanvas.yview)
        self.generatedPointTextFrame =
                           customtkinter.CTkFrame(self.generatedPointCanvas,
                                                  corner_radius=0,
                                                  fg_color="white")

        self.generatedPointCanvas.configure(
                yscrollcommand=self.generatedPointScrollbar.set, bg="white",
                highlightthickness=0)

        self.generatedPointTextLabel =
               customtkinter.CTkLabel(master=self.generatedPointTextFrame,
                                      text=textToDisplay,
                                      text_font=("Roboto Medium", -12),
                                      fg_color="white",
                                      justify="left",
                                      anchor="w",
                                      text_color="gray38",
                                height=self.generatedPointCanvas.winfo_height())

        self.generatedPointTextLabel.grid(row=0, column=0, padx=20, pady=20)
        self.generatedPointCanvas.configure(
                             height=self.generatedPointTextLabel.winfo_height())
        self.generatedPointCanvas.bind(
            "<Configure>",
            lambda e: self.generatedPointCanvas.configure(
                scrollregion=self.generatedPointCanvas.bbox("all")
            )
        )
        self.generatedPointCanvas.create_window((0, 0),
                              window=self.generatedPointTextFrame, anchor="nw")
        self.generatedPointCanvas.pack(side="left", fill="both", expand=True)
        self.generatedPointScrollbar.pack(side="right", fill="y")

    def process(self):
        begin = t.perf_counter()
        self.listPoint = pointGenerator(self.dimension, self.numOfPoints,
                                                        self.maxAxisVal)
        listOfAllPoints = stringDisplayGenerator(self.listPoint,
                                             self.numOfPoints, self.dimension)
        self.displayPoints(listOfAllPoints)
        self.shortestDistance, self.solution, self.countEuclidean =
             shortestDistance(self.dimension, self.numOfPoints, self.listPoint)
```

```
        self.getVisualization()
        end = t.perf_counter()
        self.time = end-begin
        self.label_infot3.configure(text=stringOutputGenerator(self.time,
         self.solution, self.dimension, self.shortestDistance,
          self.countEuclidean))

    def checkvalid(self):
        self.dimension = self.dimensionBuffer.get()
        self.numOfPoints = self.numPointsBuffer.get()
        self.listPoint = []
        if (self.dimension.isnumeric() and self.numOfPoints.isnumeric()):
            self.dimension = int(self.dimension)
            self.numOfPoints = int(self.numOfPoints)
            if (not self.dimension > 0 or not self.numOfPoints > 1):
                self.bufferValidityLabel.configure(text="Dimension should be
                integer > 0 and Number of points should be integer > 1")
            else:
                if (self.numOfPoints > ((2*self.maxAxisVal + 1) **
                    self.dimension)):
                    self.bufferValidityLabel.configure(text="Number of points
                    should be lower than (2 *Axis Width)^dimension + 1")
                else:
                    self.process()
        else:
            if self.dimension.isnumeric() and not self.numOfPoints.isnumeric():
                self.bufferValidityLabel.configure(text="Number of points
                        should be integer > 1")
            elif not self.dimension.isnumeric() and
                     self.numOfPoints.isnumeric():
                if (int(self.numOfPoints) > 0):
                    self.bufferValidityLabel.configure(text="Dimension should
                    be integer > 0")
                else:
                    self.bufferValidityLabel.configure(text="Dimension should
                    be integer > 0 and Number of points should be integer > 1")
            else:
                self.bufferValidityLabel.configure(text="Dimension should be
                integer > 0 and Number of points should be integer > 1")

    def change_appearance_mode(self, new_appearance_mode):
        customtkinter.set_appearance_mode(new_appearance_mode)

    def on_closing(self, event=0):
        self.quit()
        self.destroy()

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

b. divideAndConquer.py

```
def shortestDistance(dimension, numOfPoints, listOfAllPoints):
    countEuclidean = 0
    max = -9999
    min = 9999
    for i in range (numOfPoints):
        if listOfAllPoints[i][dimension-1] > max:
            max = listOfAllPoints[i][dimension-1]
        if listOfAllPoints[i][dimension-1] < min:
            min = listOfAllPoints[i][dimension-1]
    divideAt = (max + min)/2
    left = []
    right = []
    for i in range (len(listOfAllPoints)):
        if listOfAllPoints[i][dimension-1] >= divideAt:
```

```
            right.append(listOfAllPoints[i])
        else:
            left.append(listOfAllPoints[i])
    shortest = 9999
    print(max, min, divideAt, left, right)
    solution = []

    if (dimension == 1):
        for i in range (len(right)):
            for j in range (i+1, len(right)):
                countEuclidean = countEuclidean + 1
                pointDistance = euclideanDistance(right[i], right[j])
                if (pointDistance < shortest):
                    solution.clear()
                    solution.append([right[i], right[j]])
                    shortest = pointDistance
                elif pointDistance == shortest:
                    solution.append([right[i], right[j]])

        for i in range (len(left)):
            for j in range (i+1, len(left)):
                countEuclidean = countEuclidean + 1
                pointDistance = euclideanDistance(left[i], left[j])
                if (pointDistance < shortest):
                    solution.clear()
                    solution.append([left[i], left[j]])
                    shortest = pointDistance
                elif pointDistance == shortest:
                    solution.append([left[i], left[j]])

    else:
        shortestRight = 9999
        shortestLeft = 9999
        countEuclideanRight = 0
        countEuclideanLeft = 0
        solutionRight = []
        solutionLeft = []
        if (len(right) > 1):
            shortestRight, solutionRight, countEuclideanRight =
shortestDistance(dimension-1, len(right), right)
        if (len(left) > 1):
            shortestLeft, solutionLeft, countEuclideanLeft =
shortestDistance(dimension-1, len(left), left)
        if (shortestRight < shortestLeft):
            shortest = shortestRight
            solution = solutionRight
        elif (shortestRight == shortestLeft):
            solution = solutionRight + solutionLeft
            shortest = shortestLeft
        else:
            shortest = shortestLeft
            solution = solutionLeft

        countEuclidean = countEuclideanLeft + countEuclideanRight

    nearLine = []
    for i in range (len(listOfAllPoints)):
        if (listOfAllPoints[i][dimension-1] >= divideAt - shortest and
listOfAllPoints[i][dimension-1] <= divideAt + shortest):
            nearLine.append(listOfAllPoints[i])
    for i in range (len(nearLine)):
        for j in range (i+1, len(nearLine)):
            if ((nearLine[i] in right and nearLine[j] in left) or (nearLine[j]
in right and nearLine[i] in left)):
                countEuclidean = countEuclidean + 1
                pointDistance = euclideanDistance(nearLine[i], nearLine[j])
                if (pointDistance < shortest):
                    print("cleared")
                    solution.clear()
                    solution.append([nearLine[i], nearLine[j]])
```

```
                    shortest = pointDistance
                elif pointDistance == shortest:
                    if (not [nearLine[i], nearLine[j]] in solution):
                        solution.append([nearLine[i], nearLine[j]])

    print(shortest, solution, countEuclidean)
    return (shortest, solution, countEuclidean)

def euclideanDistance(pointA, pointB):
    sumSquare = 0
    for i in range (len(pointA)):
        sumSquare += (pointA[i]-pointB[i])**2
    return sumSquare ** 0.5
```

c. visualization.py

```
from PIL import Image
from matplotlib import pyplot as mplot

def getGraph(dimension, numOfPoints, listOfAllPoints, solution, maxAxisVal):
    mplot.close()

    if (dimension == 1):
        for i in range (numOfPoints):
            found = False
            for j in range (len(solution)):
                if listOfAllPoints[i] in solution[j]:
                    found = True
            if not found:
                mplot.plot(listOfAllPoints[i][0], 0, marker = "o", markersize =
4, markerfacecolor="green", markeredgecolor="green")
            else :
                mplot.plot(listOfAllPoints[i][0], 0, marker = "o", markersize =
4, markerfacecolor="red", markeredgecolor="red")

        for i in range (len(solution)):
            mplot.plot([solution[i][0][0], solution[i][1][0]], [0,0],
color="red")
    elif dimension == 2:
        mplot.xlim(maxAxisVal*-1,maxAxisVal)
        mplot.ylim(maxAxisVal*-1,maxAxisVal)

        mplot.grid()
        for i in range (numOfPoints):
            found = False
            for j in range (len(solution)):
                if listOfAllPoints[i] in solution[j]:
                    found = True
            if not found:
                mplot.plot(listOfAllPoints[i][0], listOfAllPoints[i][1], marker
= "o", markersize = 4, markerfacecolor="green", markeredgecolor="green")
            else :
                mplot.plot(listOfAllPoints[i][0], listOfAllPoints[i][1], marker
= "o", markersize = 4, markerfacecolor="red", markeredgecolor="red")

        for i in range (len(solution)):
            x = [solution[i][0][0], solution[i][1][0]]
            y = [solution[i][0][1], solution[i][1][1]]
            mplot.plot(x, y, color="red")
    elif dimension == 3:
        axis = mplot.axes(projection='3d')
        axis.set_xlim3d(maxAxisVal*-1,maxAxisVal)
        axis.set_ylim3d(maxAxisVal*-1,maxAxisVal)
        axis.set_zlim3d(maxAxisVal*-1,maxAxisVal)
        for i in range (numOfPoints):
            found = False
```

```
            for j in range (len(solution)):
                if listOfAllPoints[i] in solution[j]:
                    found = True
            if not found:
                axis.scatter3D(listOfAllPoints[i][0], listOfAllPoints[i][1],
listOfAllPoints[i][2], color="green")
            else :
                axis.scatter3D(listOfAllPoints[i][0], listOfAllPoints[i][1],
listOfAllPoints[i][2], color="red")

        for i in range (len(solution)):
            x = [solution[i][0][0], solution[i][1][0]]
            y = [solution[i][0][1], solution[i][1][1]]
            z = [solution[i][0][2], solution[i][1][2]]
            axis.plot3D(x, y, z, color="red")


    mplot.savefig("bin/currentPlot.png")
```

d. generator.py

```
from PIL import Image
from matplotlib import pyplot as mplot

def getGraph(dimension, numOfPoints, listOfAllPoints, solution, maxAxisVal):
    mplot.close()

    if (dimension == 1):
        for i in range (numOfPoints):
            found = False
            for j in range (len(solution)):
                if listOfAllPoints[i] in solution[j]:
                    found = True
            if not found:
                mplot.plot(listOfAllPoints[i][0], 0, marker = "o", markersize =
4, markerfacecolor="green", markeredgecolor="green")
            else :
                mplot.plot(listOfAllPoints[i][0], 0, marker = "o", markersize =
4, markerfacecolor="red", markeredgecolor="red")

        for i in range (len(solution)):
            mplot.plot([solution[i][0][0], solution[i][1][0]], [0,0],
color="red")
    elif dimension == 2:
        mplot.xlim(maxAxisVal*-1,maxAxisVal)
        mplot.ylim(maxAxisVal*-1,maxAxisVal)

        mplot.grid()
        for i in range (numOfPoints):
            found = False
            for j in range (len(solution)):
                if listOfAllPoints[i] in solution[j]:
                    found = True
            if not found:
                mplot.plot(listOfAllPoints[i][0], listOfAllPoints[i][1], marker
= "o", markersize = 4, markerfacecolor="green", markeredgecolor="green")
            else :
                mplot.plot(listOfAllPoints[i][0], listOfAllPoints[i][1], marker
= "o", markersize = 4, markerfacecolor="red", markeredgecolor="red")

        for i in range (len(solution)):
            x = [solution[i][0][0], solution[i][1][0]]
            y = [solution[i][0][1], solution[i][1][1]]
            mplot.plot(x, y, color="red")
    elif dimension == 3:
        axis = mplot.axes(projection='3d')
        axis.set_xlim3d(maxAxisVal*-1,maxAxisVal)
```

```
        axis.set_ylim3d(maxAxisVal*-1,maxAxisVal)
        axis.set_zlim3d(maxAxisVal*-1,maxAxisVal)
        for i in range (numOfPoints):
            found = False
            for j in range (len(solution)):
                if listOfAllPoints[i] in solution[j]:
                    found = True
            if not found:
                axis.scatter3D(listOfAllPoints[i][0], listOfAllPoints[i][1],
listOfAllPoints[i][2], color="green")
            else :
                axis.scatter3D(listOfAllPoints[i][0], listOfAllPoints[i][1],
listOfAllPoints[i][2], color="red")

        for i in range (len(solution)):
            x = [solution[i][0][0], solution[i][1][0]]
            y = [solution[i][0][1], solution[i][1][1]]
            z = [solution[i][0][2], solution[i][1][2]]
            axis.plot3D(x, y, z, color="red")


    mplot.savefig("bin/currentPlot.png")

def generateTerminal(distanceBrute, countBrute, solutionBrute, solution,
countEuclidean, shortestDistance, listOfAllPoints):
    print("-------------------------------------------------")
    print("LIST OF ALL POINTS")
    print("-------------------------------------------------")
    for i in range(len(listOfAllPoints)):
        print(str(i+1) + ". " + str(listOfAllPoints[i]))
    print("-------------------------------------------------")
    print("SEARCHING RESULTS")
    print("-------------------------------------------------")
    print("1. Brute Force")
    print("Shortest Distance:", distanceBrute)
    print("Count Euclidean:", countBrute)
    print("Number Solution:", len(solutionBrute))
    print()
    print("2. Divide and Conquer")
    print("Shortest Distance:", shortestDistance)
    print("Count Euclidean:", countEuclidean)
    print("Number Solution:", len(solution))
    print()
    print("SOLUTION")
    for i in range (len(solution)):
        print(solution[i])
```

e. bruteForce.py

```
def bruteForce(numOfPoints, listOfAllPoints):
    shortest = 9999
    solution = []
    euclideanCount = 0
    for i in range (numOfPoints - 1):
        for j in range (i+1, numOfPoints):
            distance = euclideanDistanceBrute(listOfAllPoints[i],
listOfAllPoints[j])
            euclideanCount = euclideanCount + 1
            if (distance < shortest):
                shortest = distance
                solution = []
                solution.append([listOfAllPoints[i], listOfAllPoints[j]])
            elif (distance == shortest):
                solution.append([listOfAllPoints[i], listOfAllPoints[j]])
    return shortest, solution, euclideanCount
```

```
def euclideanDistanceBrute(pointA, pointB):
    sumSquare = 0
    for i in range (len(pointA)):
        sumSquare += (pointA[i]-pointB[i])**2
    return sumSquare ** 0.5
```

3.3. Catatan Implementasi

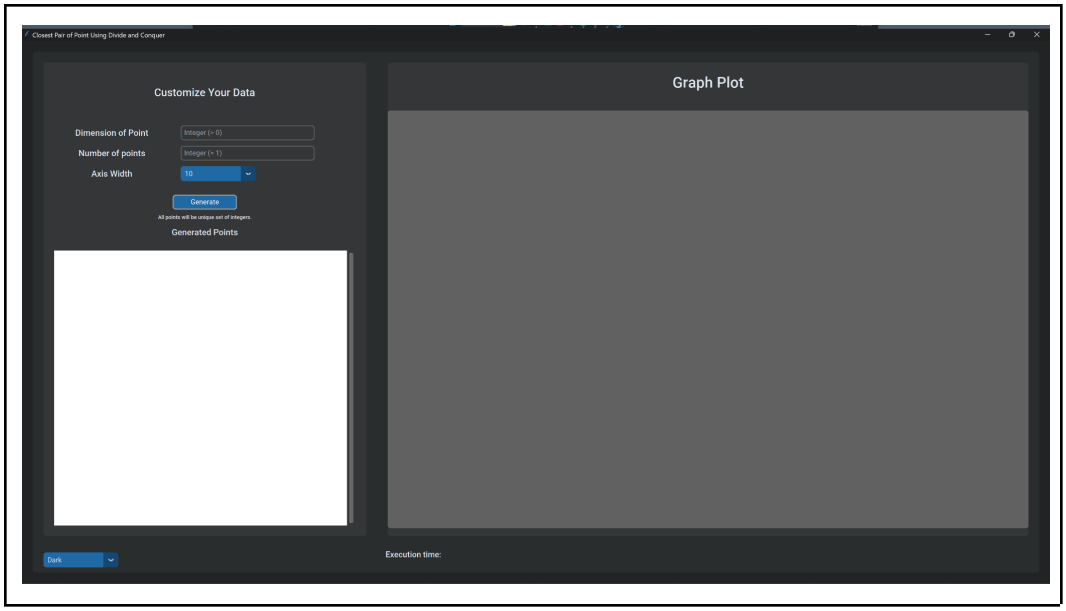
1. Karena waktu yang terbatas, GUI dikembangkan tidak secara komprehensif. Terdapat beberapa bug yang masih ada, seperti tidak tampilnya daftar titik secara keseluruhan untuk jumlah titik yang banyak (> 1000) dan tidak konstannya lebar frame pada aplikasi.
2. Bug pada (1) tidak memengaruhi jalannya program, melainkan hanya pada tampilannya saja. Sebagai alternatif, disediakan pula laporan yang dapat dilihat melalui terminal yang dikeluarkan secara realtime saat penggunaan aplikasi.
3. Generator titik HANYA menghasilkan output titik titik INTEGER yang UNIK sehingga terdapat batasan jumlah titik untuk batas axis tertentu.
4. Pengembangan aplikasi dilakukan pada sistem operasi Windows pada laptop DELL XPS 9560

BAB 4
HASIL PENGUJIAN

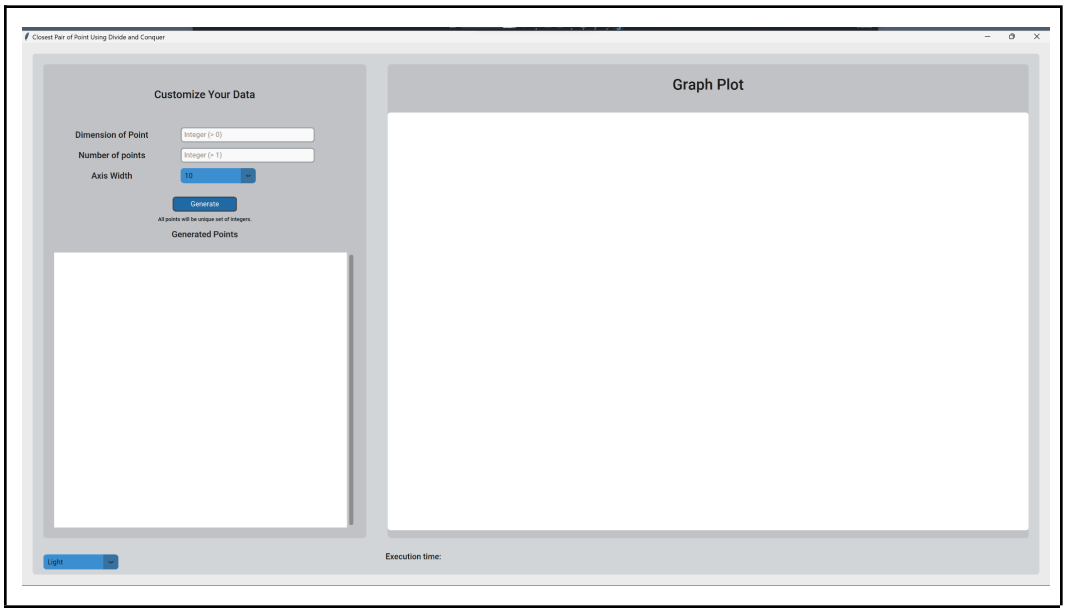
4.1. Antarmuka Aplikasi

Diakses melalui dropdown pada bagian kiri bawah aplikasi

a. Dark Mode

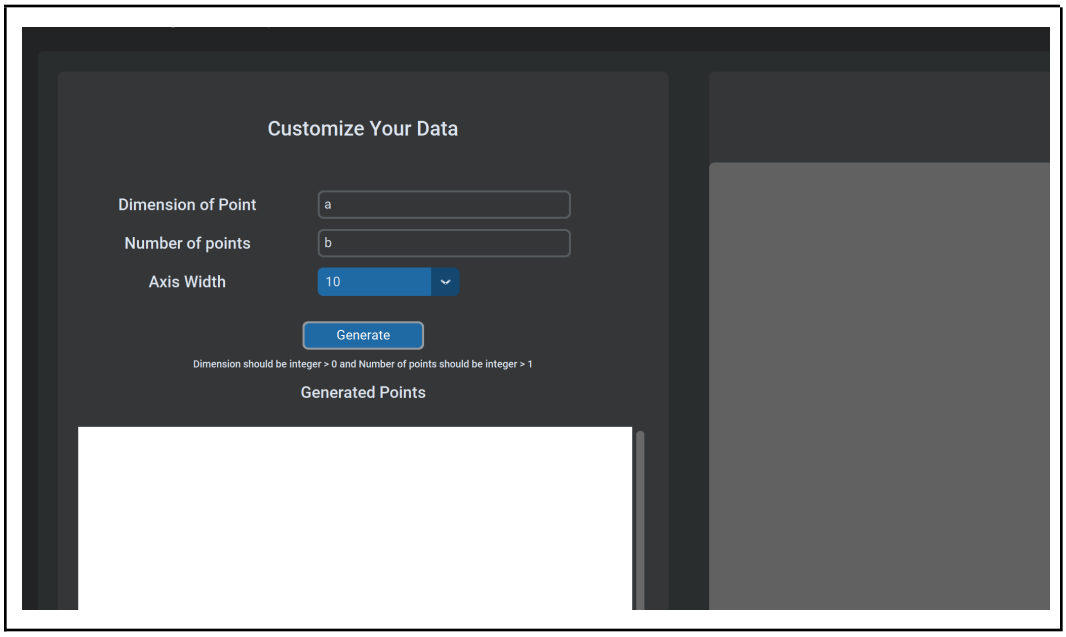


b. Light Mode

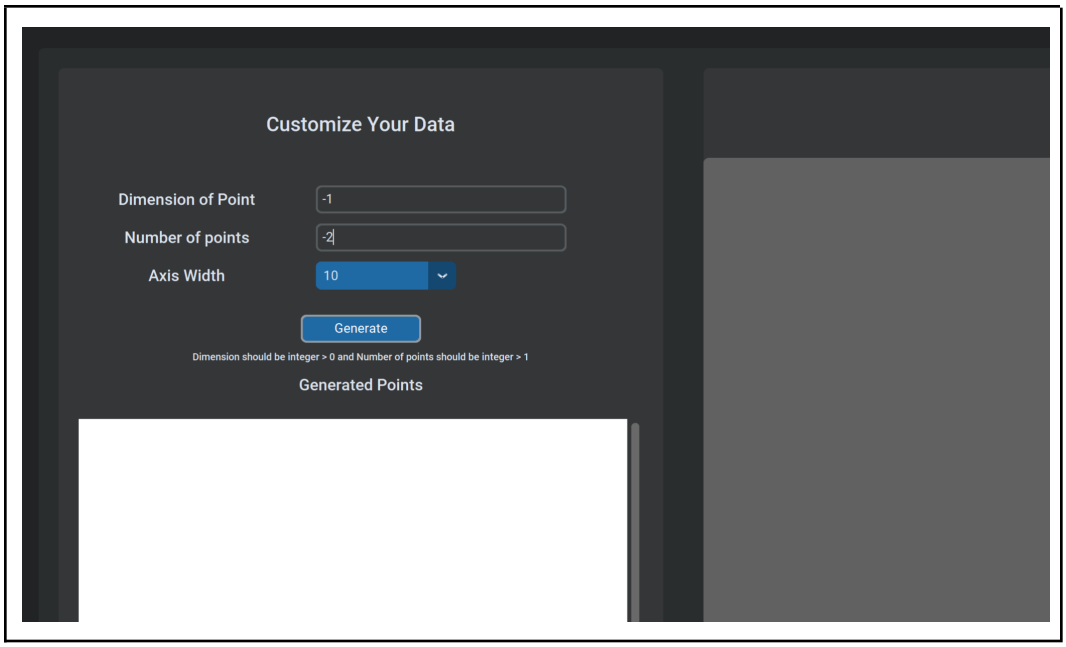


4.2. Error Handling Input

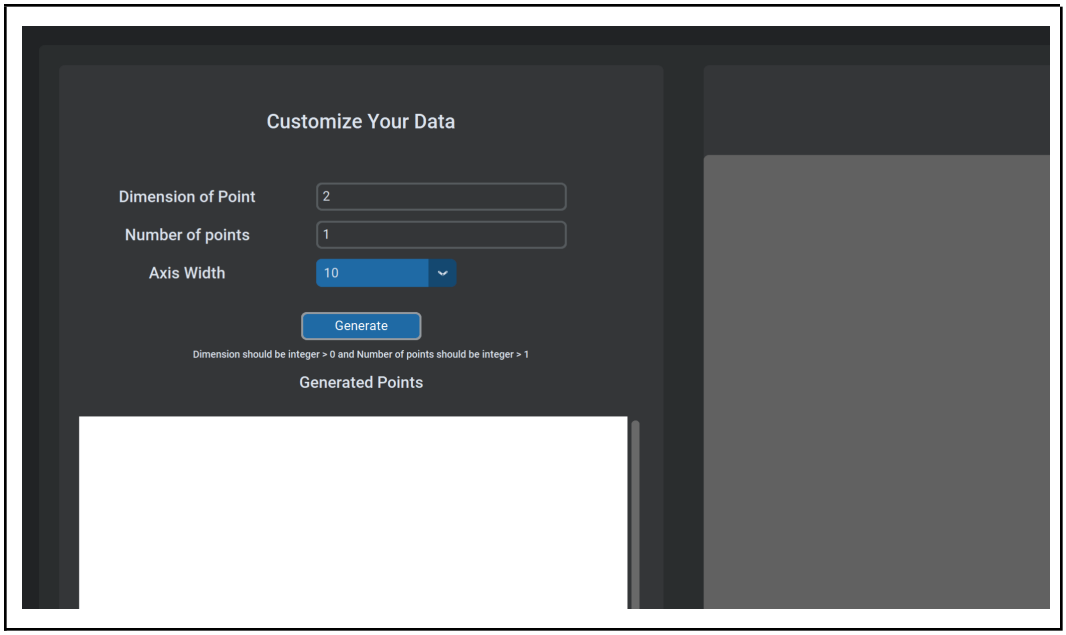
a. Input bukan angka



b. Input angka negatif



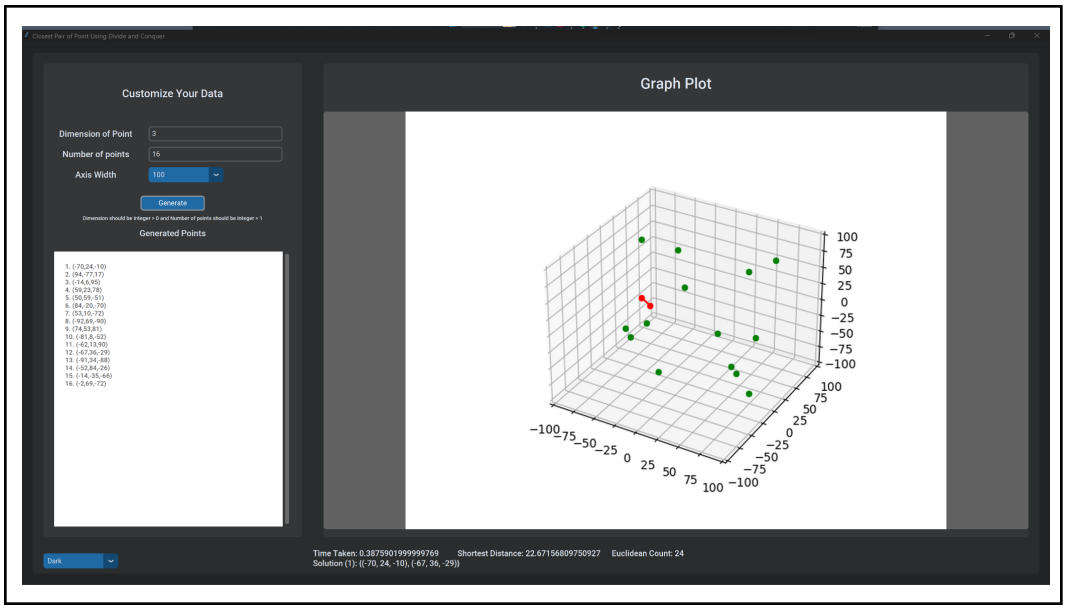
c. Number of points = 1 (tidak bisa mencari titik terdekat)



4.3. Jarak Terdekat Titik 3D

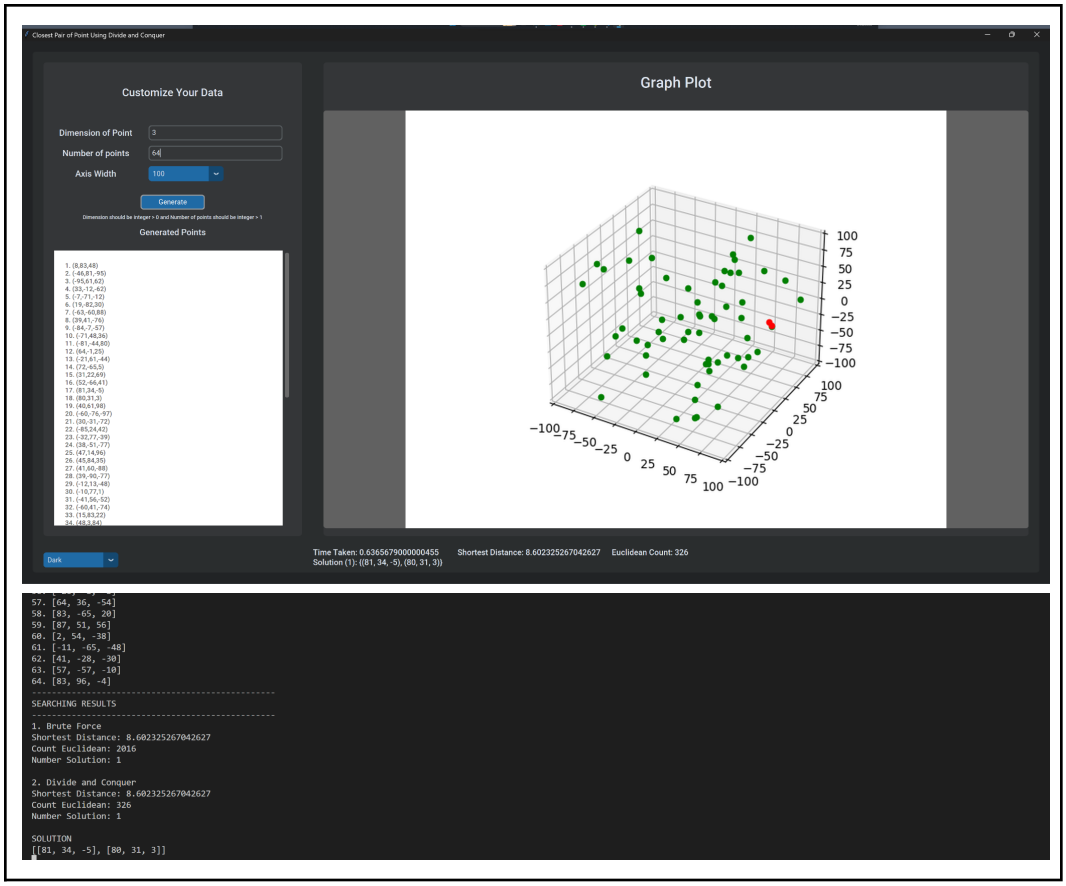
Dijalankan pada laptop DELL XPS 9560

a. N = 16

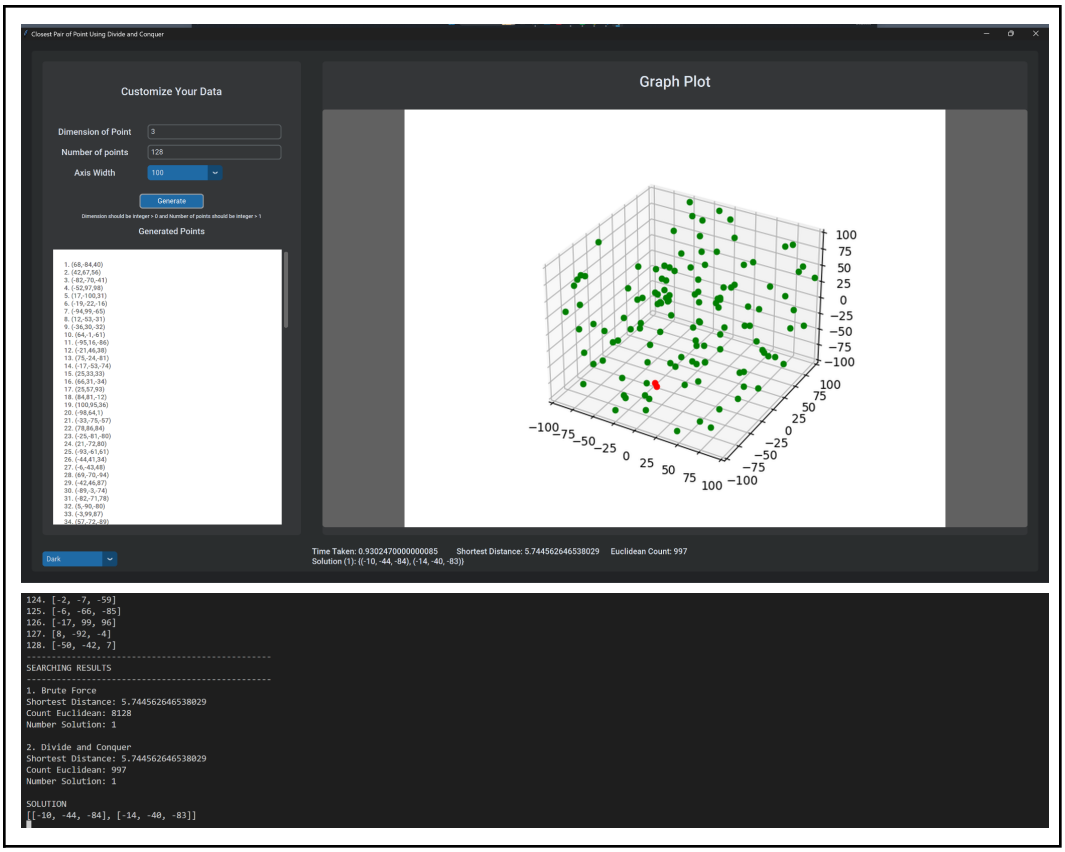



LIST OF ALL POINTS
1. [-70, 24, -10]
2. [94, -77, 17]
3. [-14, 6, 95]
4. [59, 23, 78]
5. [50, 59, -51]
6. [84, -20, -70]
7. [53, 10, -72]
8. [-92, 69, -90]
9. [74, 53, 81]
10. [-81, 8, -52]
11. [-62, 13, 90]
12. [-67, 36, -29]
13. [-91, 34, -88]
14. [-52, 84, -26]
15. [-14, -35, -66]
16. [-2, 69, -72]
SEARCHING RESULTS
1. Brute Force
Shortest Distance: 22.67156809750927
Count Euclidean: 120
Number Solution: 1
2. Divide and Conquer
Shortest Distance: 22.67156809750927
Count Euclidean: 24
Number Solution: 1
SOLUTION
[[-70, 24, -10], [-67, 36, -29]]

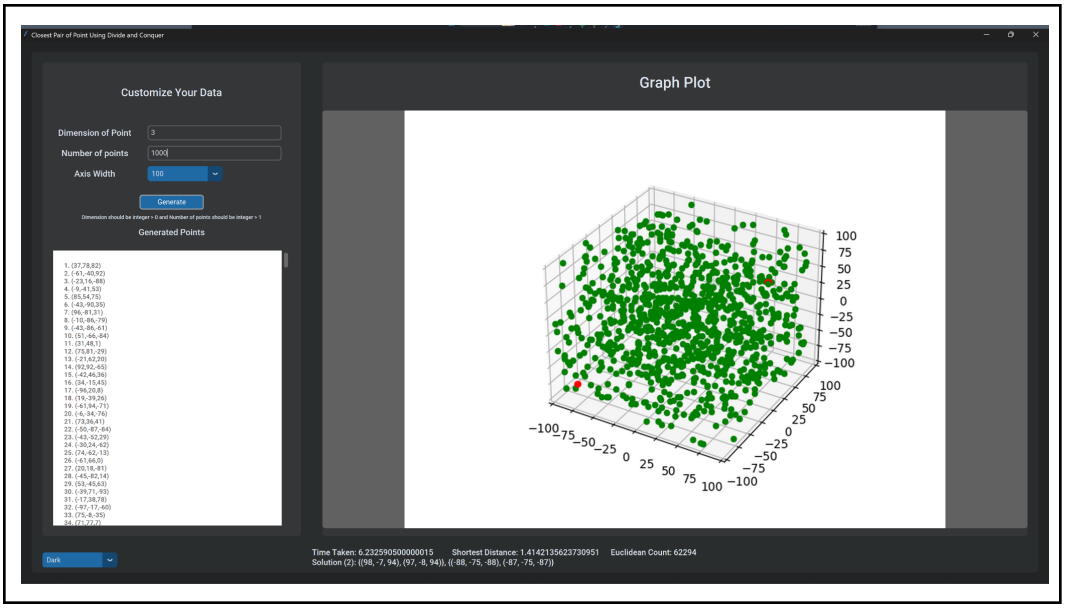
b. N = 64


Closest Pair of Point Using Divide and Conquer
Customize Your Data
Dimension of Point
3
Number of points
64
Axis Width
100
Generate
Dimension should be integer > 0 and Number of points should be integer > 1
Generated Points
Graph Plot
Dark
Time Taken: 0.6365679000000455
Shortest Distance: 8.602325267042627
Euclidean Count: 326
Solution (1): ((81, 34, -5), (80, 31, 3))
57. [64, 36, -54]
58. [83, -65, 20]
59. [87, 51, 56]
60. [2, 54, -38]
61. [-11, -65, -48]
62. [41, -28, -30]
63. [57, -57, -10]
64. [83, 96, -4]
SEARCHING RESULTS
1. Brute Force
Shortest Distance: 8.602325267042627
Count Euclidean: 2016
Number Solution: 1
2. Divide and Conquer
Shortest Distance: 8.602325267042627
Count Euclidean: 326
Number Solution: 1
SOLUTION
[[81, 34, -5], [80, 31, 3]]

c. N = 128



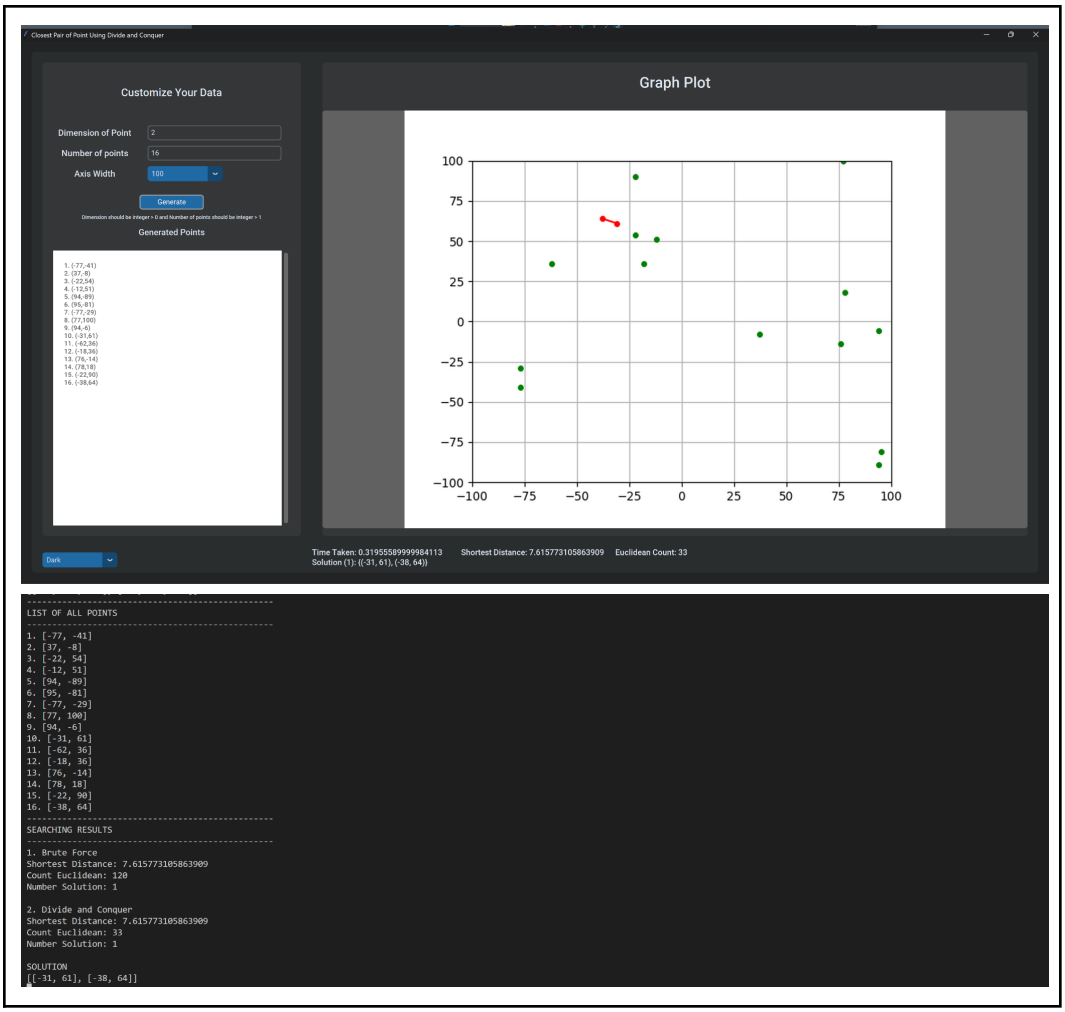
d. N = 1000 (beserta kasus lebih dari satu solusi jarak terdekat)



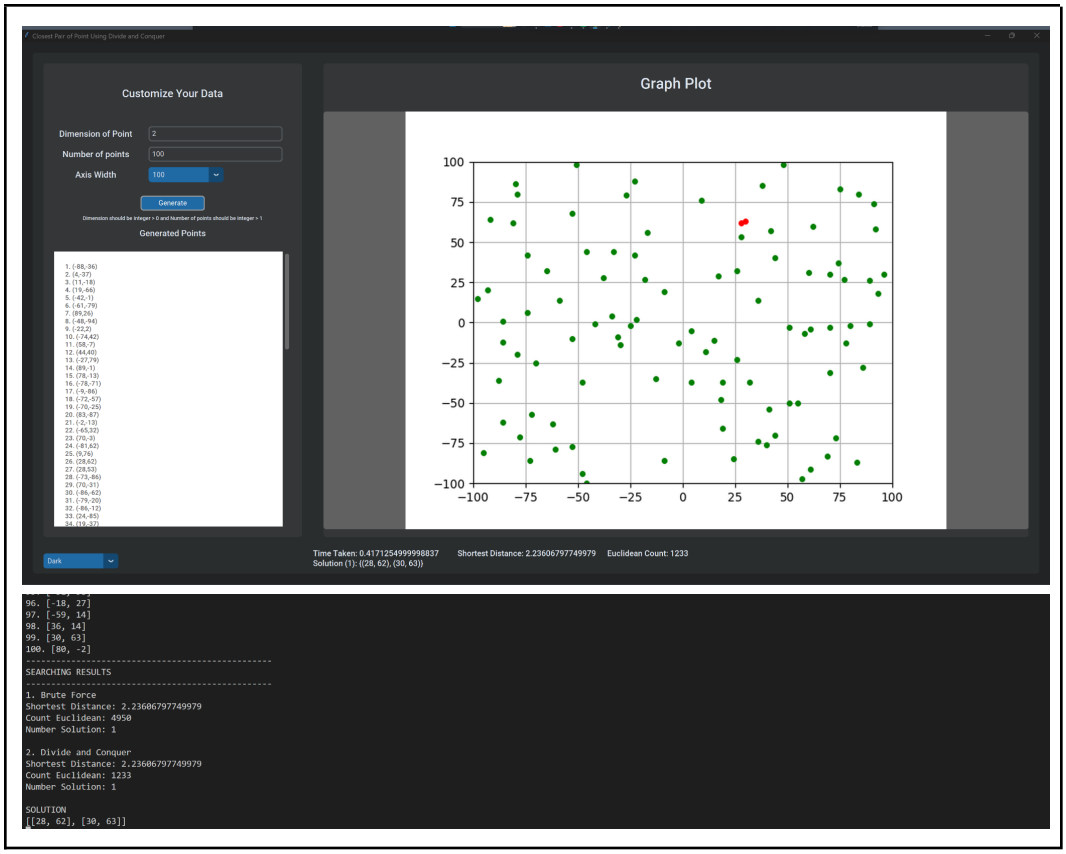
4.4. Jarak Terdekat Titik 2D

Dijalankan pada laptop DELL XPS 9560

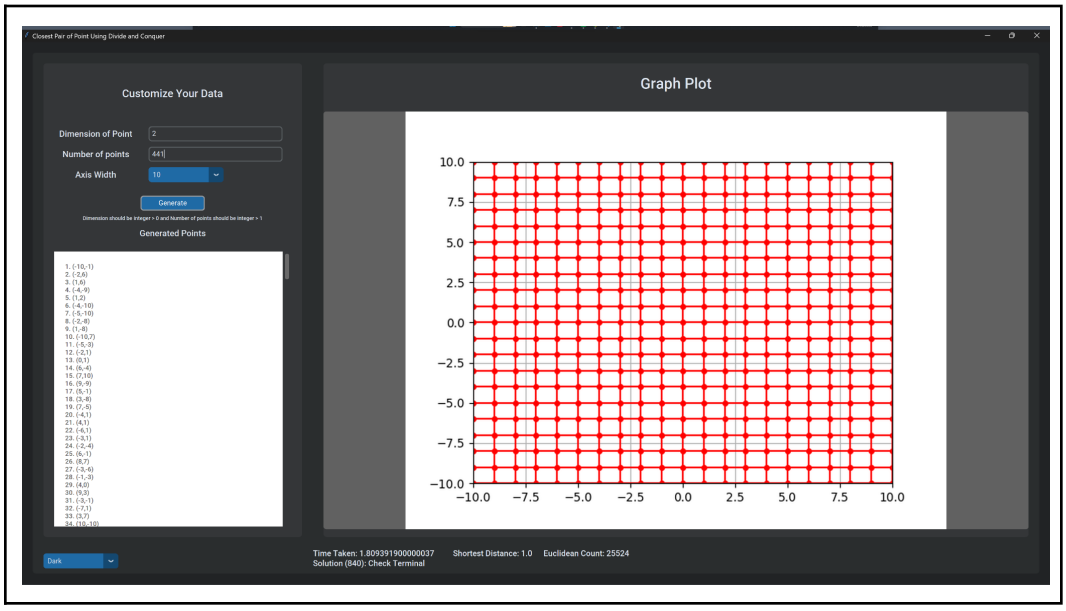
a. N = 16



b. N = 100 (dengan kasus lebih dari satu



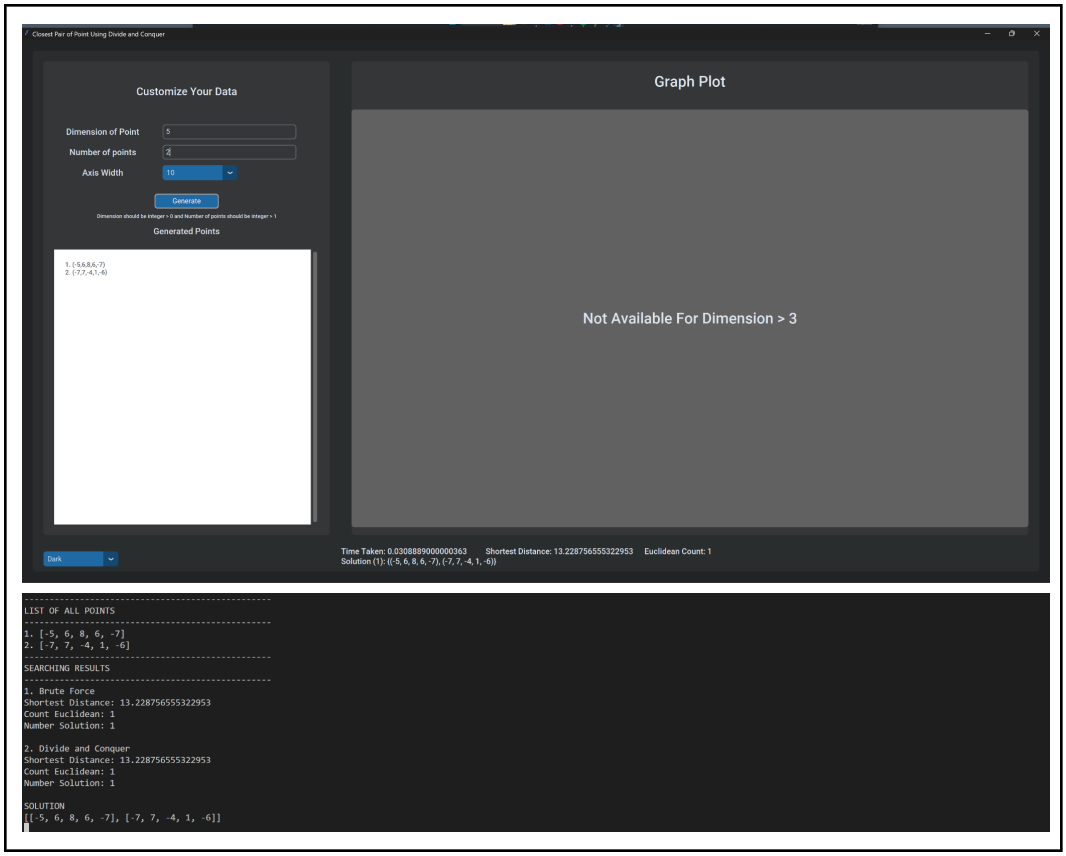
c. N = 441 dengan Axis Width 10 (jumlah titik maksimum 2D pada Axis Width 10)



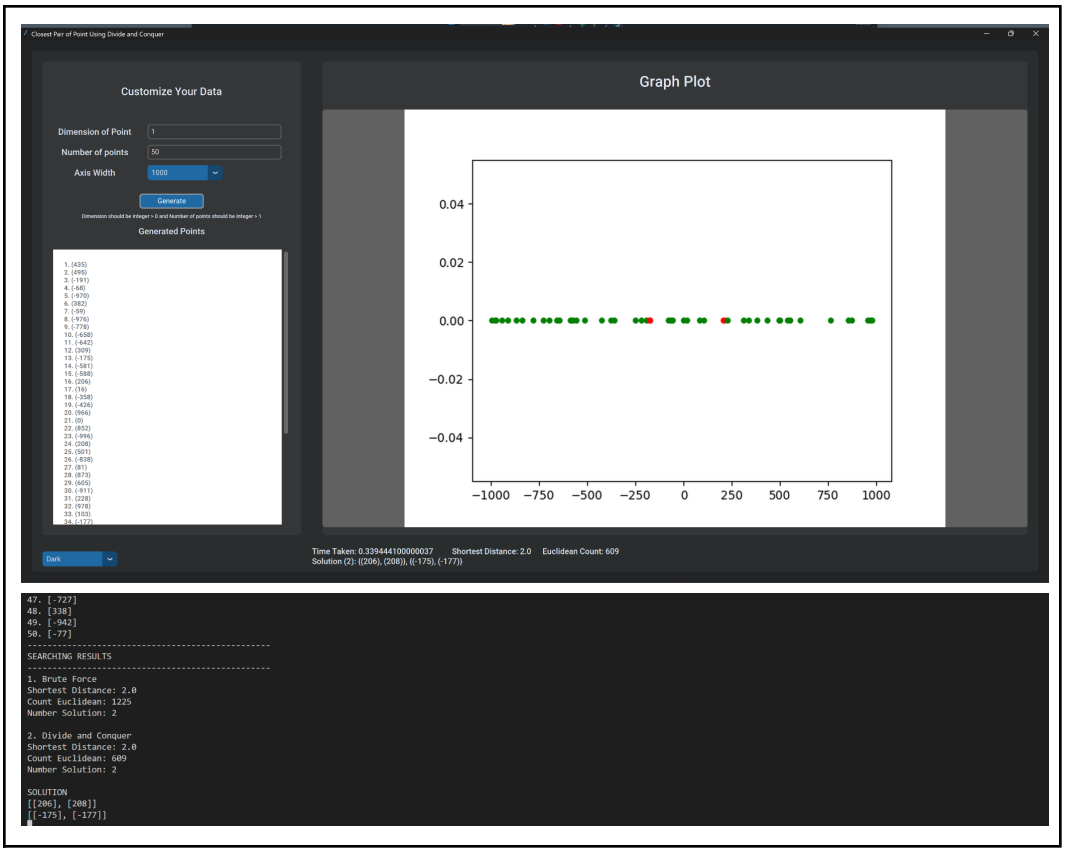
4.5. Jarak Terdekat Dimensi Acak

Dijalankan pada laptop DELL XPS 9560

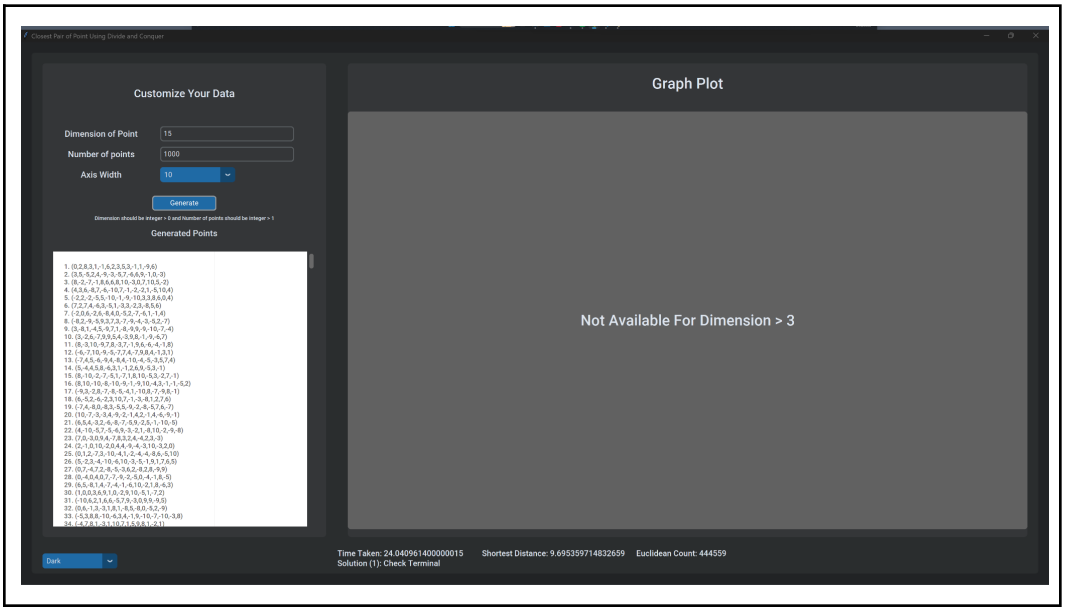
a. N = 2, Axis Width 10 pada R5

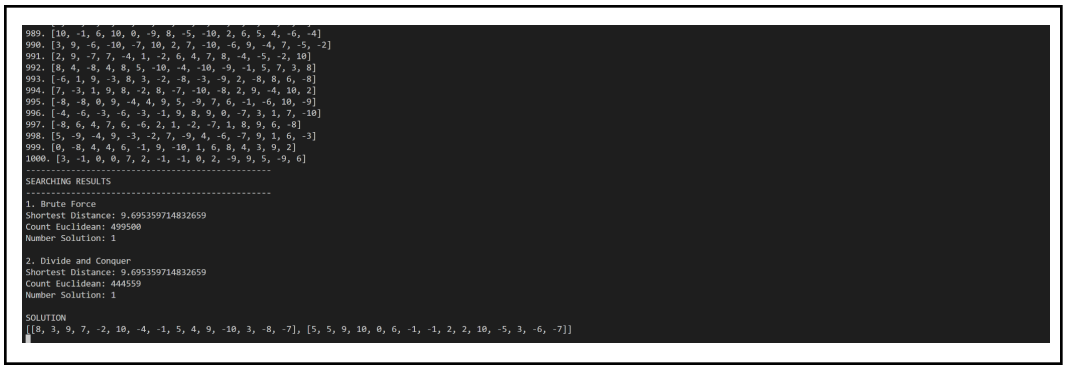


b. N = 50, Axis Width 1000 pada R1



c. N = 1000 dengan Axis Width 10 pada R15





4.6. Analisis Pengujian

Algoritma Divide And Conquer selalu menghasilkan solusi terbaik melalui metode ini. Meskipun tidak ada cara matematis yang dapat membuktikannya, setidaknya seluruh percobaan (termasuk yang dilakukan di luar laporan ini) selalu memberikan solusi yang sama dengan yang didapatkan menggunakan algoritma brute force (yang mana hasilnya dijamin optimal). Selain itu, algoritma divide and conquer selalu memberikan perhitungan komputasi euclidean yang tidak lebih besar dibandingkan algoritma brute force sehingga dapat dikatakan komputasi yang dilakukan menjadi lebih efektif.

BAB 5
PENUTUP

5.1. Komentar, Saran, dan Refleksi

Setelah melakukan berbagai eksplorasi terkait algoritma divide and conquer serta pengimplementasinya secara langsung dalam persoalan pencarian titik terdekat. Saya sangat bangga terhadap diri saya telah berusaha semaksimal mungkin dalam membangun program ini hingga akhir batas waktu dengan sangat baik. Saya ingin mengucapkan terima kasih kepada Bu Ulfa telah mengajarkan saya dasar-dasar algoritma divide and conquer sehingga saya dapat mengimplementasikannya dalam kehidupan nyata secara langsung melalui pembuatan program ini.

Saya juga belajar sangat banyak mengenai algoritma ini dan library GUI tkinter yang tentunya sangat membantu saya dalam menyelesaikan persoalan algoritma divide and conquer. Saya berharap ilmu yang kami dapatkan saat ini dapat saya gunakan dan bermanfaat bagi saya dan membantu saya dalam pengerjaan tugas-tugas yang akan datang.

LAMPIRAN

LINK REPOSITORY

Link repository GitHub : https://github.com/natthankrish/Tucil2_135211612

CHECKLIST

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