LAPORAN TUGAS KECIL
IF2211 STRATEGI ALGORITMA

Mencari Pasangan Titik Terdekat 3D dengan Algoritma *Divide and Conquer*

Disusun oleh

Jeffrey Chow	13521046
Noel Christoffel Simbolon	13521096

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKOLOGI BANDUNG

2023

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

1.1 Spesifikasi Tugas

Dalam tugas ini, kami diminta untuk mengembangkan algoritma untuk mencari sepasang titik terdekat pada bidang tiga dimensi. Misalkan terdapat n buah titik pada ruang tiga dimensi. Setiap titik P di dalam ruang dinyatakan dengan koordinat $P = (x, y, z)$. Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak antara dua titik $P_1 = (x_1, y_1, z_1)$ dan $P_2 = (x_2, y_2, z_2)$ dihitung dengan persamaan *Euclidean distance* berikut.

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

Program dibuat dalam dalam bahasa pemrograman C, C++, Java, Python, Go, Ruby, atau Perl. Algoritma yang dikembangkan harus menerapkan algoritma *divide-and-conquer*. Kami juga perlu membandingkan algoritma *divide-and-conquer* dengan algoritma *brute-force* dalam menyelesaikan masalah.

1.2 Spesifikasi Program

Program yang kami buat untuk menyelesaikan masalah ini diimplementasikan dalam bahasa pemrograman Python. Kami mengimplementasikan algoritma *divide-and-conquer* serta *brute-force* dalam program tersebut. Selain dari spesifikasi utama, kami mengimplementasikan spesifikasi bonus pada tugas ini. Oleh karena itu, program kami digeneralisasi untuk mencari sepasang titik terdekat untuk sekumpulan titik di R^n. Selain itu, terdapat visualisasi data dalam bentuk *scatter plot* tiga dimensi jika masukan jumlah dimensi bernilai tiga. Semua spesifikasi program ini dikemas dalam sebuah *graphical user interface*.

BAB II
ALGORITMA DIVIDE AND CONQUER

2.1 Penjelasan Algoritma

Implementasi algoritma *divide and conquer* menggunakan prinsip rekursif sehingga pada algoritma yang kami gunakan terdapat basis dan rekurens. Terdapat sebuah larik `points` yang berisi titik-titik, *integer* `dimension` yang berupa dimensi dari titik-titik yang terdapat pada larik, dan `divide_by` sebagai sumbu pada titik untuk membagi dan mengurutkan suatu larik. Secara *default*, `divide_by` memiliki nilai 0 yang berarti sumbu pertama yang terdapat pada titik. Misalnya terdapat sebuah titik (x, y, z), maka secara *default* algoritma *divide-and*-conquer akan membagi dan mengurutkan suatu larik berdasarkan nilai x.

Berikut adalah *function signature* dari fungsi yang mengimplementasikan algoritma *divide-and-conquer*.

```
def find_closest_pair_dnc(points: np.ndarray[np.ndarray[float]], dimension: int,
divide_by: int = 0) -> Tuple[Tuple[np.ndarray[float], np.ndarray[float]], float,
int]:
```

Luaran dari fungsi tersebut ada tiga, yaitu sebuah *tuple* yang berisi `closest_points`, `distance`, dan `euclidean_count`

Seperti yang telah dijelaskan sebelumnya, terdapat dua kondisi basis dan satu kondisi rekurens dalam fungsi ini. Rekurens terjadi jika terdapat lebih dari tiga titik dalam larik points yang merupakan salah satu parameter dari fungsi `find_closest_pair_dnc`. Berikut langkah-langkah yang terjadi jika fungsi masuk ke dalam proses rekurens.

1. Urutkan larik `points` berdasarkan `divide_by` secara terurut menaik
2. Bagi dua larik yang sudah diurutkan menjadi dua sama panjang sehingga terbentuk larik sebelah kiri dan sebelah kanan (implementasi *divide*)
3. Lakukan rekursi dengan memanggil fungsi `find_closest_pair_dnc` untuk larik kiri dan larik kanan. Nilai `divide_by` ditambah satu dan dilakukan *modulo operation* dengan `dimension - 1` jika `dimension` lebih besar dari 1 agar selanjutnya larik diproses pada sumbu yang berbeda atau dilakukan *modulo operation* dengan `dimension` jika `dimension` bernilai 1
4. Hasil dari rekursi akan menghasilkan `closest_points`, `distance`, dan `euclidean_count`. Selanjutnya, dibandingkan antara hasil dari rekursi larik kiri dan larik kanan. Lalu diambil `closest_points` yang memiliki *distance* (`min_dist`) paling kecil (implementasi *conquer*).
5. Selanjutnya, perlu dilakukan pengecekan titik-titik di sekitar area yang terbagi menjadi larik kiri dan kanan karena ada kemungkinan pasangan titik yang terdekat terpisah akibat adanya pembagian larik. Pengecekan antara dua titik hanya perlu dilakukan apabila semua jarak antar sumbu lebih kecil atau sama dengan `min_dist`. Apabila kedua titik tersebut memiliki *euclidean distance* yang lebih kecil daripada `min_dist`, maka kedua titik itu yang akan diambil sebagai `closest_points` dan *euclidean distance* antara keduanya akan menjadi `min_dist` (implementasi *conquer*).

6. *Return* `closest_points`, `min_dist`, dan `euclidean_count`. `euclidean_count` adalah jumlah melakukan perhitungan *euclidean distance*, yang didapatkan dari penambahan `euclidean_count` pada rekursi kiri dan kanan, serta jumlah perhitungan *euclidean distance* yang dilakukan pada proses poin nomor 5.

Selain dari kondisi rekurens, terdapat dua kondisi basis yang dapat dicapai dalam fungsi ini. Basis pertama terjadi apabila banyak titik dalam larik sama dengan tiga. Berikut langkah-langkah yang terjadi jika fungsi masuk ke dalam basis pertama.

1. Cari pasangan titik terdekat dari antara tiga titik. Pencarian ini dilakukan dengan menggunakan algoritma *brute force*. (`euclidean_count = 3`)
2. *Return* `closest_points`, `min_dist`, dan `euclidean_count`.

Basis kedua terjadi apabila banyak didik dalam larik sama dengan dua. Berikut langkah-langkah yang terjadi jika fungsi masuk ke dalam basis kedua.

1. Cari *euclidean distance* antara dua titik. (`euclidean_count = 1`)
2. *Return* `closest_points`, `min_dist`, dan `euclidean_count`.

BAB III
IMPLEMENTASI DALAM BAHASA PYTHON

3.1 `main.py`

File main.py merupakan *file* utama yang mengatur masukan, keluaran, dan *control flow* dari program. Fungsi-fungsi yang mengimplementasikan algoritma *divide-and-conquer* serta *brute-force* dipanggil pada *file* ini. Selain itu, *graphical user interface* dan visualisasi data juga diimplementasikan pada *file* ini.

```
import random
from timeit import default_timer as timer

import customtkinter
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import brute_force as bf
import divide_and_conquer as dnc
import util

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

class App(customtkinter.CTk):
    # initialize inputs
    number_of_dimensions = 0
    number_of_points = 0
    points = None

    # Output variables for divide-and-conquer
    # ed_operations means the number of euclidean distance operations performed
    dnc_ed_operations = ""
    dnc_execution_time = ""
    dnc_closest_pair = None
    dnc_distance = None

    # Output variables for brute-force
    bf_ed_operations = ""
    bf_execution_time = ""
    bf_closest_pair = None
    bf_distance = None

    def __init__(self):
        super().__init__()

        # Font variables
        title_font = customtkinter.CTkFont(family="Arial Bold", size=-18)
        input_heading_font = customtkinter.CTkFont(family="Arial", size=-14)
        output_heading_font = customtkinter.CTkFont(family="Arial Bold", size=-
14)
        placeholder_font = customtkinter.CTkFont(family="Arial", size=-12)
        select_theme_font = customtkinter.CTkFont(family="Arial", size=-12)
        randomize_input_font = customtkinter.CTkFont(family="Arial Bold", size=-
12)
```

```
        start_button_font = customtkinter.CTkFont(family="Arial Bold", size=-12)
        status_font = customtkinter.CTkFont(family="Arial", size=-12)
        validation_label_font = customtkinter.CTkFont(family="Arial", size=-12)
        output_font = customtkinter.CTkFont(family="Arial", size=-14)

        # Main window configurations
        self.title("Closest Pair of Points")

        # ============ create two frames ============

        # configure grid layout (1 x 2)
        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=1)

        self.left_frame = customtkinter.CTkFrame(master=self, width=180)
        self.left_frame.grid(row=0, column=0, sticky="nswe", padx=(20, 0),
pady=20)

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

        # ============ left_frame ============

        # Configure grid layout (19 x 1)
        self.left_frame.grid_rowconfigure(0, minsize=10)  # empty row with
minsize as spacing
        self.left_frame.grid_rowconfigure(2, minsize=20)  # space between input
fields and app title
        self.left_frame.grid_rowconfigure(12, weight=1)  # empty row as spacing
        self.left_frame.grid_rowconfigure(15, minsize=20)  # empty row as spacing
        self.left_frame.grid_rowconfigure(18, minsize=20)  # empty row with
minsize as spacing

        # App title
        self.app_title = customtkinter.CTkLabel(master=self.left_frame,
                                                text="Closest Pair of Points",
                                                font=title_font)
        self.app_title.grid(row=1, column=0, pady=10, padx=20)

        # Number of points
        self.number_of_points_label =
customtkinter.CTkLabel(master=self.left_frame,
                                                        text="Number of
Points:",
                                                        font=input_heading_f
ont)
        self.number_of_points_label.grid(row=3, column=0, pady=0, padx=20)

        self.number_of_points_entry =
customtkinter.CTkEntry(master=self.left_frame,
                                                        placeholder_text="In
teger greater than 1",
                                                        font=placeholder_fon
t)
        self.number_of_points_entry.grid(row=4, column=0, pady=5, padx=20,
sticky="ew")
```

```
        self.number_of_points_validation_label =
customtkinter.CTkLabel(master=self.left_frame,
                                                                text="",
                                                                text_colo
r="red",
                                                                font=vali
dation_label_font)
        self.number_of_points_validation_label.grid(row=5, column=0, pady=(0,
10), padx=20)

        # Number of dimensions
        self.number_of_dimensions_label =
customtkinter.CTkLabel(master=self.left_frame,
                                                            text="Number of
Dimensions:",
                                                            font=input_headi
ng_font)
        self.number_of_dimensions_label.grid(row=6, column=0, pady=0, padx=20)

        self.number_of_dimensions_entry =
customtkinter.CTkEntry(master=self.left_frame,
                                                            placeholder_text
="Integer greater than 0",
                                                            font=placeholder
_font)
        self.number_of_dimensions_entry.grid(row=7, column=0, pady=5, padx=20,
sticky="ew")

        self.number_of_dimensions_validation_label =
customtkinter.CTkLabel(master=self.left_frame,
                                                                text=
"",
                                                                text_
color="red",
                                                                font=
validation_label_font)
        self.number_of_dimensions_validation_label.grid(row=8, column=0, pady=(0,
10), padx=20)

        # Randomize input button
        self.randomize_input_button =
customtkinter.CTkButton(master=self.left_frame,
                                                        text="Randomize
Input",
                                                        font=randomize_inpu
t_font,
                                                        command=self.random
ize_input)
        self.randomize_input_button.grid(row=9, column=0, pady=5, padx=20)

        # Start button
        self.start_button = customtkinter.CTkButton(master=self.left_frame,
                                                    text="Start",
                                                    font=start_button_font,
                                                    command=self.start)
        self.start_button.grid(row=10, column=0, pady=5, padx=20)

        self.space_label = customtkinter.CTkLabel(master=self.left_frame,
```

```
                                                text="")
        self.space_label.grid(row=11, column=0, pady=5, padx=20)

        # Result
        self.status_heading_label =
customtkinter.CTkLabel(master=self.left_frame,
                                                        text="Status:",
                                                        font=output_heading_fo
nt)
        self.status_heading_label.grid(row=13, column=0, pady=0, padx=20)

        # Prints the status of the program
        self.status_label = customtkinter.CTkLabel(master=self.left_frame,
                                                   text="",
                                                   font=status_font)
        self.status_label.grid(row=14, column=0, pady=0, padx=20)

        # Select GUI theme
        self.theme_label = customtkinter.CTkLabel(master=self.left_frame,
                                                  text="Select theme:",
                                                  font=select_theme_font)
        self.theme_label.grid(row=16, column=0, pady=0, padx=20, sticky="s")

        self.theme_options = customtkinter.CTkOptionMenu(master=self.left_frame,
                                                         values=["Dark", "Light",
"System"],
                                                         font=select_theme_font,
                                                         command=change_appearanc
e_mode)
        self.theme_options.grid(row=17, column=0, pady=5, padx=20, sticky="")

        # ============ right_frame ============

        # Configure grid layout (3x3) and its weights
        # weight=0 means it will not expand
        self.right_frame.rowconfigure((0, 1, 2, 3), weight=1)
        self.right_frame.columnconfigure((0, 1, 2), weight=1)
        self.right_frame.rowconfigure(1, weight=0)
        self.right_frame.columnconfigure(1, weight=0)
        self.right_frame.rowconfigure(3, weight=0)

        # Frame containing matplotlib 3D scatter plot (only shows up if the
number of dimensions is 3)
        self.visualization_frame =
customtkinter.CTkFrame(master=self.right_frame, fg_color="transparent")
        self.visualization_frame.grid(row=1, column=1, pady=(20, 20), padx=(20,
20), sticky="nswe")

        # Initialize a canvas for 3D scatter plot
        # This initialization is useful to avoid displaying multiple plots
        # by destroying the canvas before creating a new one every time a plot is
want to be drawn
        self.visualization_canvas = FigureCanvasTkAgg(None,
master=self.visualization_frame)

        # Initialize label for output when number of dimensions != 3
        self.points_output_label =
customtkinter.CTkLabel(master=self.visualization_frame)
```

```
        # Output frame: contains output comparison between divide-and-conquer and
brute-force
        self.output_frame = customtkinter.CTkFrame(master=self.right_frame)
        self.output_frame.grid(row=3, column=1, pady=(0, 20), padx=(20, 20),
sticky="nswe")
        self.output_frame.rowconfigure((0, 1), weight=0)
        self.output_frame.columnconfigure((0, 1, 2, 3), weight=0)

        # output labels for divide-and-conquer algorithm
        self.divide_and_conquer_heading_label =
customtkinter.CTkLabel(master=self.output_frame,
                                                                text="Divi
de and Conquer",
                                                                font=outpu
t_heading_font,
                                                                anchor="n"
)
        self.divide_and_conquer_heading_label.grid(row=0, column=0, padx=(20,
10), pady=(20, 0))

        self.divide_and_conquer_label =
customtkinter.CTkLabel(master=self.output_frame,
                                                           text="Euclidean
Distance Operations: \n"
                                                                "Execution
Time: ",
                                                           font=output_font,
                                                           anchor="w",
                                                           justify="left")
        self.divide_and_conquer_label.grid(row=1, column=0, padx=(20, 10),
pady=(0, 20))

        self.divide_and_conquer_output_label =
customtkinter.CTkLabel(master=self.output_frame,
                                                                text=f'{App
.dnc_ed_operations}\n'
                                                                     f'{App
.dnc_execution_time}',
                                                                font=output
_font,
                                                                text_color=
"green",
                                                                anchor="w",
                                                                justify="le
ft")
        self.divide_and_conquer_output_label.grid(row=1, column=1, padx=(0, 10),
pady=(0, 20))

        # output labels for brute-force algorithm
        self.brute_force_heading_label =
customtkinter.CTkLabel(master=self.output_frame,
                                                           text="Brute
Force",
                                                           font=output_headi
ng_font,
                                                           anchor="n")
```

```
        self.brute_force_heading_label.grid(row=0, column=2, padx=(20, 10),
pady=(20, 0))

        self.brute_force_label = customtkinter.CTkLabel(master=self.output_frame,
                                                        text="Euclidean Distance
Operations: \n"
                                                             "Execution Time: ",
                                                        font=output_font,
                                                        anchor="w",
                                                        justify="left")
        self.brute_force_label.grid(row=1, column=2, padx=(20, 10), pady=(0, 20))

        self.brute_force_output_label =
customtkinter.CTkLabel(master=self.output_frame,
                                                                text=f'{App.bf_ed_
operations}\n'
                                                                     f'{App.bf_exe
cution_time}',
                                                                font=output_font,
                                                                text_color="green"
,
                                                                anchor="w",
                                                                justify="left")
        self.brute_force_output_label.grid(row=1, column=3, padx=(0, 10),
pady=(0, 20))

    def randomize_input(self):
        """
        Randomizes input for the number of dimensions and the number of points
        """
        # Generate random integer
        App.number_of_dimensions = random.randint(1, 50)
        App.number_of_points = random.randint(2, 1000)

        # Clear entry first
        self.number_of_dimensions_entry.delete(0, 4)
        self.number_of_points_entry.delete(0, 4)

        # Then, insert the generated random integer
        self.number_of_dimensions_entry.insert(0, str(App.number_of_dimensions))
        self.number_of_points_entry.insert(0, str(App.number_of_points))

    def start(self):
        """
        Generate points and find the closest pair of points
        """
        # if the number of dimensions or the number of points is not valid,
        # display error message until it is valid
        if not
(util.validate_number_of_dimensions(self.number_of_dimensions_entry.get()) and
                util.validate_number_of_points(self.number_of_points_entry.get())
):

            # clear status label if input is invalid
            self.status_label.configure(text="")

            # if the number of dimensions is not valid
```

```
            if not
util.validate_number_of_dimensions(self.number_of_dimensions_entry.get()):
                self.number_of_dimensions_validation_label.configure(text="Invali
d number of dimensions.")
            else:
                self.number_of_dimensions_validation_label.configure(text="")

            # if the number of points is not valid
            if not
util.validate_number_of_points(self.number_of_points_entry.get()):
                self.number_of_points_validation_label.configure(text="Invalid
number of points.")
            else:
                self.number_of_points_validation_label.configure(text="")

            return

        # cool. if the input is valid, then clear any error message
        self.number_of_dimensions_validation_label.configure(text="")
        self.number_of_points_validation_label.configure(text="")

        # Assign inputs
        App.number_of_dimensions = int(self.number_of_dimensions_entry.get())
        App.number_of_points = int(self.number_of_points_entry.get())

        # Generate an array of arrays containing float
        App.points = np.random.uniform(low=-100, high=100,
size=(App.number_of_points, App.number_of_dimensions))

        self.status_label.configure(text="Calculating...",
                                    text_color="yellow")

        dnc_start_time = timer()

        App.dnc_closest_pair, App.dnc_distance, App.dnc_ed_operations = \
            dnc.find_closest_pair_dnc(App.points, App.number_of_dimensions)

        dnc_end_time = timer()

        bf_start_time = timer()

        App.bf_closest_pair, App.bf_distance, App.bf_ed_operations =
bf.find_closest_pair_bf(App.points)

        bf_end_time = timer()

        # If the number of dimensions is 3, display 3D scatter plot
        if App.number_of_dimensions == 3:
            self.visualize()
        else:
            # Destroy visualization canvas (if a plot is present this destroys
it)
            self.visualization_canvas.get_tk_widget().destroy()
            # Destroy text output
            self.points_output_label.destroy()
            self.points_output_label =
customtkinter.CTkLabel(master=self.visualization_frame,
```

```
                                                                    text=f'Closest Pair
of Points: '
                                                                         f'{App.dnc_clo
sest_pair[0]} and '
                                                                         f'{App.dnc_clo
sest_pair[1]}\n\n'
                                                                         f'Euclidean
Distance: '
                                                                         f'{App.dnc_dis
tance}',
                                                                    justify="left")
            self.points_output_label.pack(padx=20, pady=20)

        self.status_label.configure(text="Done!",
                                    text_color="green")

        App.dnc_execution_time = f"{dnc_end_time - dnc_start_time:.5f} s"
        App.bf_execution_time = f"{bf_end_time - bf_start_time:.5f} s"

        self.show_ed_operations_and_execution_time()

    def visualize(self):
        """
        Draw a 3D scatter plot if the number of dimensions is 3
        """

        first_point = App.dnc_closest_pair[0]
        second_point = App.dnc_closest_pair[1]

        # initialize arrays containing coordinates of points for each axis
        x_coordinates = np.array([])
        y_coordinates = np.array([])
        z_coordinates = np.array([])

        for i in range(App.number_of_points):
            # if the current point is one of the points in the closest pair,
don't append it
            if any(np.array_equal(App.points[i], p) for p in
App.dnc_closest_pair):
                continue
            x_coordinates = np.append(x_coordinates, App.points[i][0])
            y_coordinates = np.append(y_coordinates, App.points[i][1])
            z_coordinates = np.append(z_coordinates, App.points[i][2])

        # Create the plot
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')

        # Scatter all points except the closest pair in blue
        ax.scatter(x_coordinates, y_coordinates, z_coordinates, alpha=0.1, c='b')

        # Scatter the closest pair of points in red
        ax.scatter([first_point[0], second_point[0]],
                   [first_point[1], second_point[1]],
                   [first_point[2], second_point[2]], c='r')

        # Draw a line between the closest pair of points
        ax.plot([first_point[0], second_point[0]],
```

```
                [first_point[1], second_point[1]],
                [first_point[2], second_point[2]], c='k')

        # Add labels and title
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        ax.set_title('3D Scatter Plot with Highlighted Closest Pair of Points')

        # Add text annotations for closest pair of points and its distance
        fig.text(0.03, 0.03, f'Closest Pair of Points: '
                             f'({first_point[0]:.3f}, {first_point[1]:.3f},
{first_point[2]:.3f}) and '
                             f'({second_point[0]:.3f}, {second_point[1]:.3f},
{second_point[2]:.3f})\n'
                             f'Euclidean Distance: {App.dnc_distance:.3f}',
fontsize=8, color='k')

        # Destroy visualization canvas (if a plot is present this destroys it)
        self.visualization_canvas.get_tk_widget().destroy()

        # Destroy text output
        if self.points_output_label.cget("text") != "CTkLabel" or "":
            self.points_output_label.pack_forget()

        # Then, draw a new plot to a Tkinter canvas
        self.visualization_canvas = FigureCanvasTkAgg(fig,
master=self.visualization_frame)
        self.visualization_canvas.draw()
        self.visualization_canvas.get_tk_widget().pack()

    def show_ed_operations_and_execution_time(self):
        """
        Display Euclidean distance operations and execution time in the GUI
        """
        self.divide_and_conquer_output_label.configure(text=f'{App.dnc_ed_operati
ons}\n'
                                                            f'{App.dnc_execution_
time}')
        self.brute_force_output_label.configure(text=f'{App.bf_ed_operations}\n'
                                                      f'{App.bf_execution_time}')

def change_appearance_mode(new_appearance_mode: str):
    """
    Changes the GUI theme
    :param new_appearance_mode: string: "dark", "light", or "system"
    """
    customtkinter.set_appearance_mode(new_appearance_mode)

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

3.2 `divide_and_conquer.py`

File ini berisi satu fungsi yang merupakan implementasi algoritma *divide-and-conquer*.

```
from typing import Tuple

import numpy as np

import brute_force as bf
import util

# divide_by means divide by either x-axis or y-axis.
# divide_by = 0 , represent divide by x
# divide_by = 1, represent divide by y
def find_closest_pair_dnc(points: np.ndarray[np.ndarray[float]], dimension: int,
divide_by: int = 0) -> \
        Tuple[Tuple[np.ndarray[float], np.ndarray[float]], float, int]:
    """
    Finds the closest pair of points using divide-and-conquer algorithm
    :param points: a numpy array of points
    :param dimension: the number of dimensions of the points
    :param divide_by: the index of column to divide-and-conquer by
    :return: a tuple: (the closest pair of points, its distance, number of
Euclidean distance operations)
    """

    if len(points) > 3:  # recurrence

        # sort points
        sorted_points = util.quick_sort(points, divide_by)

        # find index to divide
        divide_at = len(sorted_points) // 2

        # split points
        points_left = sorted_points[:divide_at]
        points_right = sorted_points[divide_at:]

        # recurrence
        closest_left, dist_left, count_left = find_closest_pair_dnc(points_left,
dimension,
                                                                    (divide_by +
1) %
                                                                    (dimension -
1 if dimension > 1 else dimension))
        closest_right, dist_right, count_right =
find_closest_pair_dnc(points_right, dimension,
                                                                       (divide_by
+ 1) %
                                                                       (dimension
- 1 if dimension > 1 else dimension))

        # find minimum
        closest_vector, min_dist = (closest_left, dist_left) if dist_left <
dist_right else (closest_right, dist_right)

        # find if there were nearer points separated by the strip
        count_gray = 0
        for i in range(len(sorted_points)):
            for j in range(i + 1, len(sorted_points)):

                check_further = True
```

```
                for k in range(dimension):

                    if abs(sorted_points[i][k] - sorted_points[j][k]) > min_dist:
                        check_further = False
                        break

                if check_further:
                    temp_dist = util.euclidean_distance(sorted_points[i],
sorted_points[j])
                    count_gray += 1

                    if temp_dist < min_dist:
                        closest_vector, min_dist = (sorted_points[i],
sorted_points[j]), temp_dist

        # return
        return closest_vector, min_dist, count_left + count_right + count_gray

    elif len(points) == 3:  # first basis
        return bf.find_closest_pair_bf(points)

    else:  # second basis
        # find distance between 2 points
        dist = util.euclidean_distance(points[0], points[1])

        # return the two points and distance between them
        return (points[0], points[1]), dist, 1
```

3.3 `brute_force.py`

File ini berisi satu fungsi yang merupakan implementasi algoritma *brute-force*.

```
from typing import Tuple

import numpy as np

import util

def find_closest_pair_bf(points: np.ndarray[np.ndarray[float]]) -> \
        Tuple[Tuple[np.ndarray[float], np.ndarray[float]], float, int]:
    """
    Finds the closest pair of points using brute-force algorithm
    :param points: a numpy array of points
    :return: a tuple: (the closest pair of points, its distance, number of
Euclidean distance operations)
    """

    distance_count = 0
    min_dist = float('inf')
    closest_pair = None
    n = len(points)

    for i in range(n):
        for j in range(i + 1, n):

            dist = util.euclidean_distance(points[i], points[j])
```

```
            distance_count += 1

            if dist < min_dist:
                min_dist = dist
                closest_pair = (points[i], points[j])

    return closest_pair, min_dist, distance_count
```

3.4 `util.py`

File ini berisi fungsi-fungsi pembantu yang berguna untuk implementasi algoritma *divide-and conquer*, *brute-force*, dan *main program*.

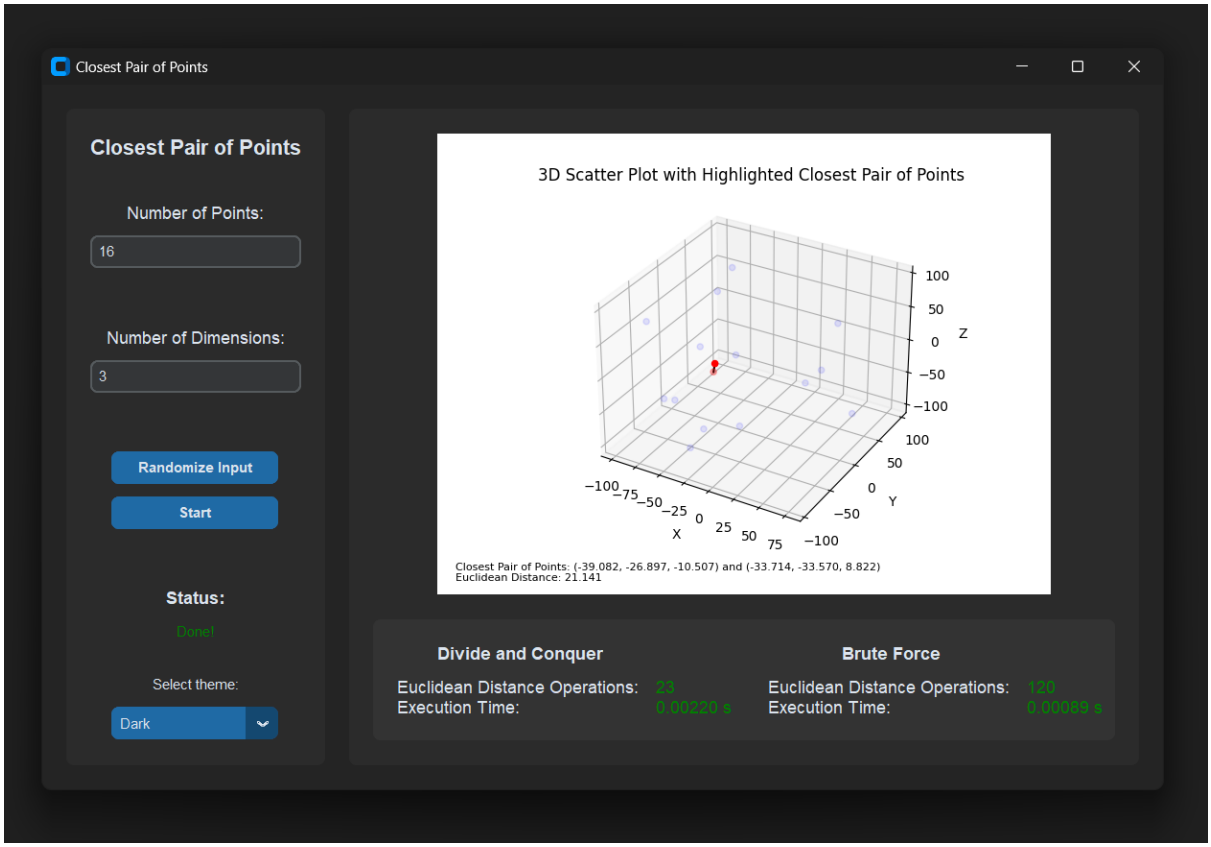
```
import math

import numpy as np

def euclidean_distance(first_point: np.ndarray[float], second_point:
np.ndarray[float]) -> float:
    """
    :param second_point: a numpy array of floats representing a single point
    :param first_point: a numpy array of floats representing a single point
    :return: the Euclidean distance between two points.
    """
    return math.sqrt(sum([(first_point[i] - second_point[i]) ** 2 for i in
range(len(first_point))]))

def quick_sort(points: np.ndarray[np.ndarray[float]], divide_by: int) ->
np.ndarray[np.ndarray[float]]:
    """
    Performs quick sort to an array of points
    :param points: a numpy array of points
    :param divide_by: the index of column to sort by
    :return: points sorted by a certain column (in this case, columns represent
axes)
    """

    if len(points) <= 1:
        return points

    pivot = points[len(points) // 2][divide_by]

    # Create array with all elements smaller than pivot
    left = points[points[:, divide_by] < pivot]

    # Create array consists only pivot
    middle = points[points[:, divide_by] == pivot]

    # Create array with all elements greater than pivot
    right = points[points[:, divide_by] > pivot]

    return np.concatenate((quick_sort(left, divide_by), middle, quick_sort(right,
divide_by)))

def validate_number_of_dimensions(d: str) -> bool:
```

```
    """
    Validates the number of dimensions
    :return: whether the number of points is >= 1
    """
    try:
        if int(d) >= 1:
            return True
    except ValueError:
        return False

    return False

def validate_number_of_points(n: str) -> bool:
    """
    Validates the number of points
    :return: whether the number of points is >= 2
    """
    try:
        if int(n) >= 2:
            return True
    except ValueError:
        return False

    return False
```

BAB IV
PENGUJIAN PROGRAM

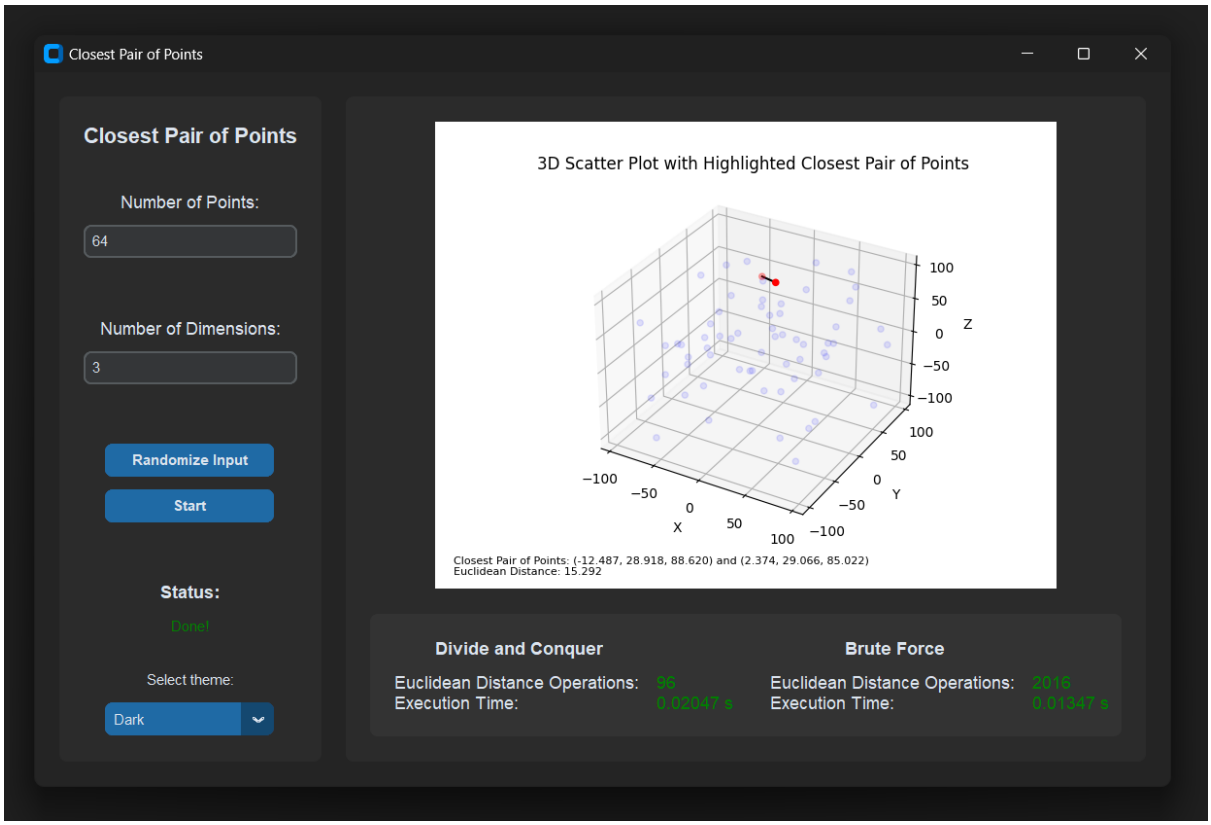
4.1 Jumlah Titik = 16

Berikut adalah *screenshot* program dengan jumlah titik 16 dan jumlah dimensi 3.



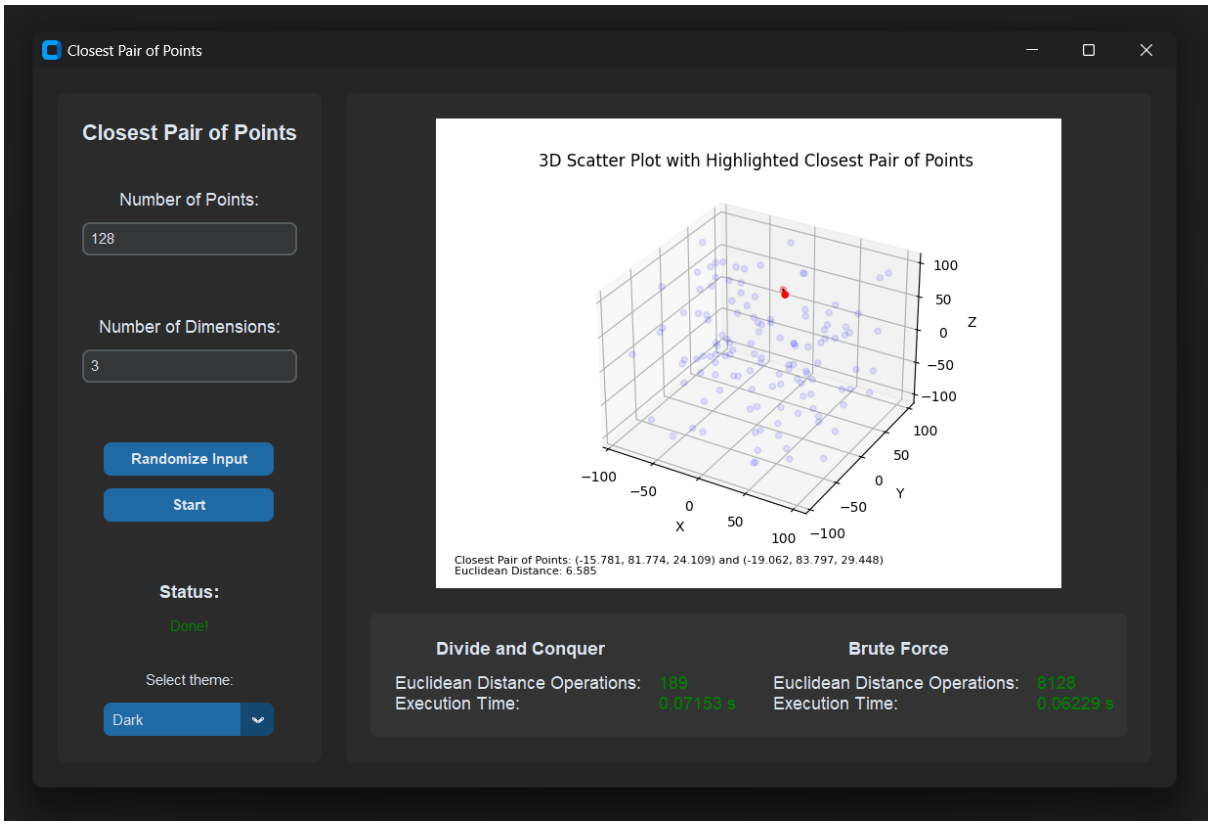
4.2 Jumlah Titik = 64

Berikut adalah *screenshot* program dengan jumlah titik 64 dan jumlah dimensi 3



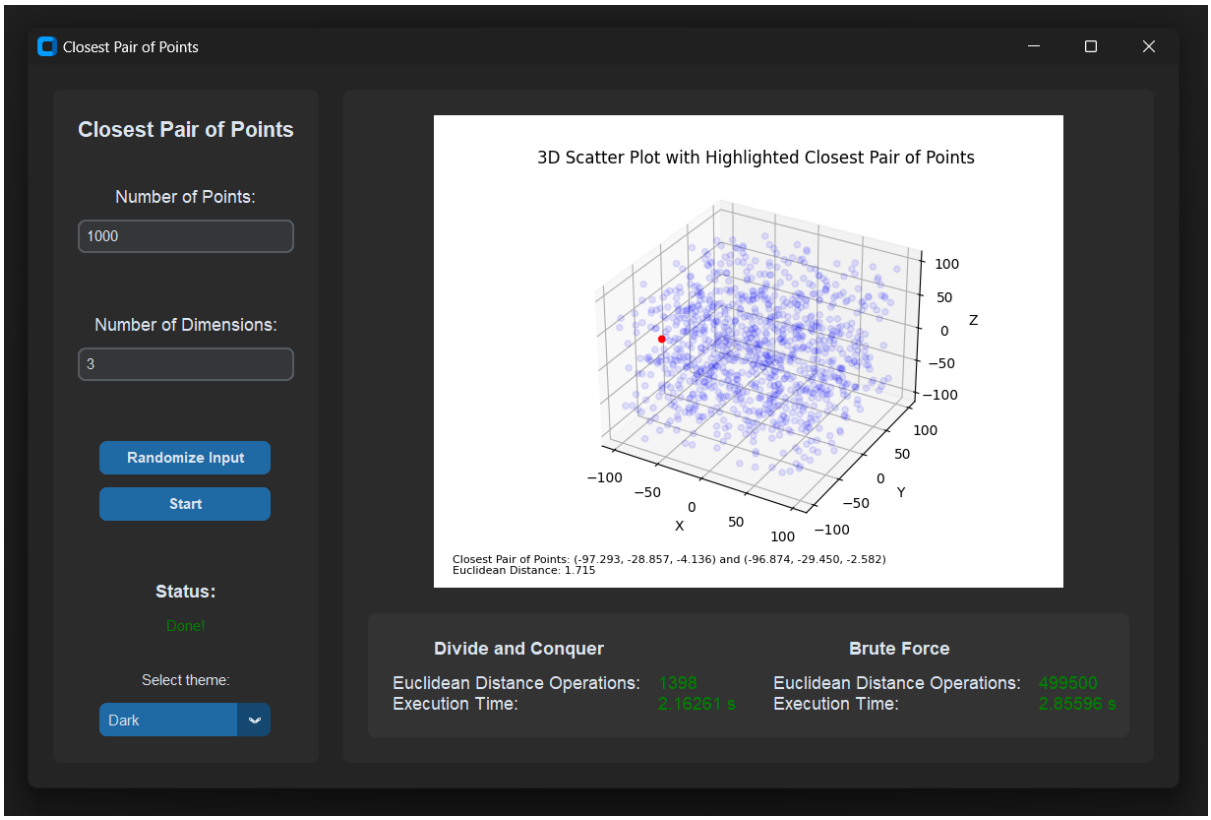
4.3 Jumlah Titik = 128

Berikut adalah *screenshot* program dengan jumlah titik 128 dan jumlah dimensi 3



4.4 Jumlah Titik = 1000

Berikut adalah *screenshot* program dengan jumlah titik 1000 dan jumlah dimensi 3



BAB V
PENUTUP

5.1 Kesimpulan

Kami berhasil berhasil mengembangkan algoritma *divide-and-conquer* untuk menyelesaikan masalah dalam menemukan pasangan titik terdekat pada kumpulan vektor di R^n. Selain itu, kami berhasil mengimplementasikan visualisasi data dalam bentuk *scatter plot* tiga dimensi jika masukan jumlah dimensi adalah tiga.

Dari hasil pengujian pada bab IV, jumlah operasi *euclidean distance* pada algoritma *divide-and-conquer* selalu lebih sedikit dibandingkan dengan menggunakan algoritma *brute force*. Selain itu, dari hasil pengujian kami, untuk jumlah titik yang banyak dan jumlah dimensi yang sedikit (tetapi lebih dari satu), waktu eksekusi algoritma *divide-and-conquer* lebih cepat daripada waktu eksekusi algoritma *brute-force*. Namun, untuk jumlah dimensi yang banyak, waktu eksekusi algoritma *divide-and-conquer* lebih lama daripada waktu eksekusi algoritma *brute-force*.

5.2 Saran

Algoritma *divide-and-conquer* yang kami implementasikan masih memiliki *room for improvement* untuk dioptimasi dari segi performa dan efisiensi. Selain itu, visualisasi data dapat dikembangkan lebih lanjut untuk meng-*handle* jika masukan jumlah dimensi satu dan dua.

LAMPIRAN

Tautan *remote repository*

Berikut adalah tautan *remote repository* yang berisi *source code* untuk tugas ini.

https://github.com/noelsimbolon/Tucil2_13521046_13521096/

***Checklist* program**

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