

LAPORAN TUGAS KECIL 2 IF2211 STRATEGI ALGORITMA

MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA DIVIDE AND CONQUER

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



Disusun oleh :

Mohd Farhan Fahrezy 13521106

Frankie Huang 13521092

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2022/2023

Daftar Isi

Daftar Isi	1
BAB I	2
BAB II	3
A. Algoritma Brute Force	3
B. Algoritma Divide and Conquer	3
BAB III	5
BAB IV	7
A. bruteForce.py	7
B. displayCoordinate.py	7
C. divideAndConquer.py	8
D. generateCoordinate.py	9
E. utilities.py	9
F. main.py	10
G. mainWindow.py	11
BAB V	14
A. Test Case	14
B. Analisis	17
BAB VI	18
Daftar Referensi	19
Lampiran	20

BAB I

DESKRIPSI MASALAH

Persoalan mencari pasangan titik terdekat dengan Algoritma *Divide and Conquer* telah dijelaskan dalam kuliah Strategi Algoritma. Dalam kuliah tersebut, persoalan dirumuskan untuk titik pada bidang datar (2D). Pada Tugas Kecil 2 kali ini penulis 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 $P = (x, y, z)$. Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak dua buah titik $P_1 = (x_1, y_1, z_1)$ dan $P_2 = (x_2, y_2, z_2)$ dihitung dengan rumus Euclidean berikut:

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

Pada tugas kecil ini, penulis diminta untuk membuat program dalam Bahasa Python untuk mencari sepasang titik yang jaraknya terdekat satu sama lain dengan menerapkan *algoritma divide and conquer* untuk penyelesaiannya, dan membandingkan hasilnya dengan hasil Algoritma *Brute Force*. Penulis juga diminta untuk melakukan generalisasi program sehingga dapat mencari pasangan titik terdekat untuk sekumpulan vektor di R^n dan setiap vektor dinyatakan dalam bentuk $x = (x_1, x_2, \dots, x_n)$.

BAB II

TEORI DASAR

A. Algoritma Brute Force

Brute Force merupakan salah satu pendekatan yang lempang (straight-forward) dalam hal pemecahan suatu masalah atau persoalan dengan sangat sederhana (simple), langsung, dan jelas (obvious way). Algoritma Brute Force sering kali disebut juga sebagai algoritma naif (naive algorithm).

Karakteristik algoritma brute force umumnya tidak “pintar” dan tidak mangkus, karena ia membutuhkan jumlah langkah yang besar dalam penyelesaiannya, sehingga terkadang algoritma brute force disebut juga algoritma naif (naïve algorithm). Algoritma brute force seringkali merupakan pilihan yang kurang disukai karena ketidakmangkusannya itu, tetapi dengan mencari pola-pola yang mendasar, keteraturan, atau trik-trik khusus, biasanya akan membantu kita menemukan algoritma yang lebih cerdas dan lebih mangkus. Untuk masalah yang ukurannya kecil, kesederhanaan brute force biasanya lebih diperhitungkan daripada ketidakmangkusannya. Algoritma brute force sering digunakan sebagai basis bila membandingkan beberapa alternatif algoritma yang mangkus. Algoritma brute force seringkali lebih mudah diimplementasikan daripada algoritma yang lebih canggih, dan karena kesederhanaannya, kadang-kadang algoritma brute force dapat lebih mangkus (ditinjau dari segi implementasi)

B. Algoritma Divide and Conquer

Divide and Conquer adalah salah satu algoritma dalam pemrograman yang dilakukan dengan memecah persoalan menjadi persoalan-persoalan yang lebih kecil dengan tipe yang sama hingga menjadi cukup sederhana untuk diselesaikan secara langsung. Solusi dari persoalan kecil tersebut kemudian digabungkan sehingga membentuk solusi persoalan semula.

Objek dari persoalan yang menggunakan algoritma ini umumnya adalah objek multi-value, seperti sebuah array. Tiap sub-persoalan memiliki karakteristik yang serupa dengan karakteristik persoalan semula, sehingga metode divide and conquer lebih natural diungkapkan dalam skema rekursif.

Perancangan algoritma ini umumnya terbagi menjadi dua, yaitu *easy divide hard combine* dan *hard divide easy combine*. Perancangan yang pertama adalah kasus dimana pembagian masalah cukup sederhana (seperti membagi array menjadi dua bagian), namun tahap

penggabungannya cukup kompleks (terdapat pengecekan-pengecekan untuk mengambil solusi yang benar dari upa-persoalan).

BAB III

ALGORITMA

Secara singkat penjelasan algoritma *Closest Pair Problem* dengan *Divide and Conquer* yang digunakan adalah sebagai berikut.

1. Cek kasus terbaik, yaitu:
 1. Jika jumlah titiknya 1, maka kembalikan tak hingga.
 2. Jika jumlah titiknya 2, maka kembalikan jarak antara kedua titik.
2. Urutkan titik berdasarkan nilai salah satu sumbu, kemudian belah kedua titik tersebut menjadi dua bagian yang sama rata.
3. Lalu kita panggil kembali fungsi divide and conquer untuk masing-masing titik yang berada di kiri dan titik yang berada di kanan.
4. Bandingkan jarak antara kedua titik, lalu simpan nilai terkecilnya (*delta*).
5. Tarik garis sepanjang *delta* dari garis tengah, kemudian simpan semua titik yang berada pada daerah tersebut ke dalam *strip*.
6. Cek dua jenis kemungkinan:
 1. Jika dimensi yang dikerjakan ≥ 3 , maka panggil kembali fungsi divide and conquer pada array *strip* dengan 1 dimensi yang lebih rendah
 2. Jika dimensi yang dikerjakan = 2, maka secara iteratif, cari jarak terkecil di antara titik tersebut, kemudian bandingkan dengan nilai *delta*. Kembalikan nilai yang lebih kecil.

Algoritma *Quick Sort* dengan *Divide and Conquer* yang diimplementasikan dibagi ke dalam 2 buah fungsi. Terdapat fungsi utama untuk membagi array dan sebuah fungsi lain untuk mencari indeks partisi dari array.

Algoritma fungsi utama:

1. Cek kasus terbaik, yaitu jika panah kanan berada di indeks yang lebih kecil dari panah kiri
2. Panggil fungsi partisi, kemudian panggil fungsi quickSort untuk elemen di kiri indeks partisi dan elemen di kanan indeks partisi
3. Kembalikan array

Algoritma fungsi mencari indeks partisi:

1. Set sebuah elemen dari array menjadi pivot (dalam program dipilih elemen dengan indeks tertinggi)
2. Cari sebuah elemen dari kiri yang lebih besar atau sama dengan dari pivot
3. Cari sebuah elemen dari kanan yang lebih kecil atau sama dengan pivot (dalam kasus ini nilainya adalah pivot)
4. Pertukarkan kedua nilai, lalu kembalikan indeks dari pivot.

BAB IV

SOURCE CODE

A. bruteForce.py

```
from utilities import calculateDistance

def closestPair(arrayCoordinate):
    closest = 1000000
    tempi = -1
    tempj = -1
    count = 0
    for i in range(len(arrayCoordinate)-1):
        for j in range(i+1, len(arrayCoordinate)):
            count+=1
            temp = calculateDistance(arrayCoordinate[i], arrayCoordinate[j], len(arrayCoordinate[i]))
            if temp < closest:
                closest = temp
                tempi = i
                tempj = j
    return tempi, tempj
```

Fungsi `closestPair()` digunakan untuk menentukan dua titik dalam `arrayCoordinate` yang memiliki jarak terdekat dengan menggunakan algoritma *brute force*.

B. displayCoordinate.py

```
import matplotlib.pyplot as plt

def displayCoordinate(arrayCoordinate, c1, c2):
    # c1 dan c2 adalah 2 koordinat yang akan digaris
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Masukan semua koordinat ke diagram
    for i in range(len(arrayCoordinate)):
        x, y, z = arrayCoordinate[i]
        ax.scatter(x, y, z, c='g', s=1)

    # Buat garis diantara c1 dan c2
    x1, y1, z1 = c1
    x2, y2, z2 = c2
    ax.scatter(x1, y1, z1, c='r', s=10)
    ax.scatter(x2, y2, z2, c='r', s=10)
    ax.plot([x1, x2], [y1, y2], [z1, z2], c='r')

    ax.set_xlabel('Sumbu X')
    ax.set_ylabel('Sumbu Y')
    ax.set_zlabel('Sumbu Z')
    plt.show()
```

Fungsi `displayCoordinate()` digunakan untuk menampilkan grafik persebaran titik dengan 2 titik terdekat memiliki warna merah dan sisanya hijau. Fungsi ini menggunakan *library* `matplotlib` untuk menampilkan grafik.

C. divideAndConquer.py

```
from utilities import quickSort, calculateDistance

def closestStrip(strip: list[list[float]], size: int, delta: float) -> float:
    """Find the smallest distance in the strip array.

    :param strip: Strip array solved in a two dimensional plane
    :param size: Number of elements in the array
    :param delta: Delta value
    """

    point_one = strip[0]
    point_two = strip[0]
    min_dist = delta

    strip = quickSort(strip, 0, len(strip)-1, 1)

    for i in range(size):
        for j in range(i+1, size):
            if (strip[j][1] - strip[i][1]) >= min_dist:
                break
            if (calculateDistance(strip[i], strip[j], len(strip[i])) < min_dist):
                point_one = strip[i]
                point_two = strip[j]
                min_dist = calculateDistance(strip[i], strip[j], len(strip[i]))

    if (min_dist == delta):
        return float("inf")
    else:
        return point_one, point_two, min_dist
```

Fungsi `closestStrip()` digunakan untuk menentukan jarak terkecil di dalam strip.

```
def divideAndConquer(Array: list[list[float]], size: int, dimension: int) -> (list[float], list[float], float):
    """Solve the closest pair problem with divide and conquer algorithm, returns the index of closest points. Array should be sorted by the last dimension beforehand.

    :param Array: Array of points
    :param size: Number of elements in the array
    :dimension: Number of dimension in the points
    :return: (list[float], list[float], float)
    """

    # Divide
    if (size == 1):
        return (Array[0], Array[0], float("inf"))
    if (size == 2):
        return (Array[0], Array[1], calculateDistance(Array[0], Array[1], len(Array[0])))

    mid = size // 2
    mid_point = Array[mid]

    left_first_point, left_second_point, left_delta = divideAndConquer(Array[:mid], mid, dimension)
    right_first_point, right_second_point, right_delta = divideAndConquer(Array[mid:], size-mid, dimension)

    if (left_delta < right_delta):
        first_point = left_first_point
        second_point = left_second_point
        delta = left_delta
    else:
        first_point = right_first_point
        second_point = right_second_point
        delta = right_delta

    strip = []
    for i in range(size):
        if (abs(Array[i][dimension-1] - mid_point[dimension-1]) < delta):
            strip.append(Array[i])

    # Conquer
    # Check if 2nd dimension. If not, then we call dnc for all points in the strip
    if (dimension == 2):
        strip = quickSort(strip, 0, len(strip)-1, dimension-1)

        point_one, point_two, strip_delta = divideAndConquer(strip, len(strip), dimension-1)

        if (strip_delta < delta):
            return point_one, point_two, strip_delta
        else:
            return first_point, second_point, delta
    else:
        if (closestStrip(strip, len(strip), delta) == float("inf")):
            return first_point, second_point, delta
        else:
            return closestStrip(strip, len(strip), delta)
```

Fungsi `divideAndConquer()` digunakan untuk menemukan jawaban persoalan dengan menggunakan algoritma *divide and conquer*.

```
def solveDivideAndConquer(Array: list[list[float]], size: int, dimension: int) -> (int, int, float):
    """Call the divideAndConquer function to retrieve the values of first_point and second_point, then returns the index of first_point and second_point from the unsorted Array

    :param Array: Array of points
    :param size: Number of elements in the array
    :dimension: Number of dimension in the points
    :return: (int, int, float)
    """

    first_point, second_point, distance = divideAndConquer(Array, size, dimension)

    first_index = -1
    second_index = -1

    i = 0
    while (i < size):
        if (Array[i] == first_point):
            first_index = i
        if (Array[i] == second_point):
            second_index = i
        i += 1

    return first_index, second_index, distance
```

Fungsi `solveDivideAndConquer()` digunakan sebagai fungsi antara yang memanggil fungsi `divideAndConquer()` dan memberikan keluaran index dari titik terdekat

D. generateCoordinate.py

```
import random

def generateCoordinate(n,r):
    arrayOfCoordinate = [ [0 for i in range(r)] for j in range(n)]
    for i in range(n):
        for j in range(r):
            arrayOfCoordinate[i][j] = random.uniform(-1000,1000)
    return arrayOfCoordinate

def printCoordinate(coordinate):
    print("",end="")
    for i in range(len(coordinate)):
        if (i != len(coordinate)-1):
            print(str(coordinate[i])+",",end="")
        else:
            print(coordinate[i],end="")
    print("",end="")

def printArrayCoordinate(arrayCoordinate):
    for i in range(len(arrayCoordinate)):
        printCoordinate(arrayCoordinate[i])
    print()
```

Fungsi generateCoordinate() digunakan untuk menggenerate titik random sejumlah n pada sebuah dengan vektor domain R^r . Prosedur printCoordinate() dan printArrayCoordinate() digunakan untuk mencetak array ke layar.

E. utilities.py

```
def partition(Array: list[list[float]], Left: int, right: int, dimension: int) -> (list[list[float]], int):
    """Function to find the partition index

    :param Array: Array of points
    :param left: Index of the left-most value
    :param right: Index of the right-most value
    :param dimension: The dimension to be sorted by
    :return: (list[list[float]], int)
    """

    pivot = Array[right][dimension-1]

    i = Left-1

    for j in range(left, right):
        if (Array[j][dimension-1] <= pivot):
            i += 1
            Array[i], Array[j] = Array[j], Array[i]

    Array[i+1], Array[right] = Array[right], Array[i+1]

    return (Array, i + 1)
```

Fungsi partition() digunakan untuk menentukan index dari partisi array.

```
def quickSort(Array: list[list[float]], Left: int, right: int, dimension: int) -> list[list[float]]:
    """Function to sort the array A by the value of the d-th dimension

    :param Array: Array of points
    :param left: Index of the left-most value
    :param right: Index of the right-most value
    :param dimension: The dimension to be sorted by (1, 2, 3, ...)
    :return: list[list[float]]
    """

    if (Left < right):
        Array, index = partition(Array, Left, right, dimension)

        quickSort(Array, Left, index-1, dimension)

        quickSort(Array, index+1, right, dimension)

    return Array
```

Fungsi quickSort() digunakan untuk melakukan sorting pada array berdasarkan value dari dimensi ke-d.

<pre>def calculateDistance(first_point: list[float], second_point: list[float], size: int) -> float: """Calculate the distance between P1 and P2 :param first_point: Coordinate (list) of the first point. :param second_point: Coordinate (list) of the second point. :param size: Size of first_point and second_point. :return: float """ value = 0 for i in range(size): value += (first_point[i] - second_point[i])**2 return math.sqrt(value)</pre>	<p>Fungsi <code>calculateDistance()</code> digunakan untuk menentukan jarak antara 2 titik.</p>
<pre>def pointToStr(point): strPoint = "(" for i in range(len(point)): strPoint+=str(round(point[i],2)) if(i!=len(point)-1): strPoint+=", " return strPoint+")"</pre>	<p>Fungsi <code>pointToStr()</code> digunakan untuk mengubah suatu titik menjadi string.</p>

F. main.py

<pre>from generateCoordinate import generateCoordinate, printCoordinate from displayCoordinate import displayCoordinate from bruteForce import closestPair from divideAndConquer import solveDivideAndConquer from utilities import calculateDistance from time import time def main(): """Main function""" n = int(input("Enter amount of points (n) : ")) R = int(input("Enter dimension of points (R) : ")) Array = generateCoordinate(n, R) choice = input("1) Brute Force\n2) Divide and Conquer\nChoose Algorithm : ") start = time() * 1000 if (choice == 0): first, second = closestPair(Array) distance = calculateDistance(Array[first], Array[second], R) else: first, second, distance = solveDivideAndConquer(Array, n, R) finish = time() * 1000 time_taken = finish - start print("Closest Pair: ", end = '') printCoordinate(Array[first]) printCoordinate(Array[second]) print() print(f'Distance: {distance}') print("Executed Time: ", time_taken, "ms") if (R == 3): displayCoordinate(Array, Array[first], Array[second]) if __name__ == '__main__': main()</pre>	<p>Fungsi <code>main()</code> digunakan untuk menjalankan program secara utuh. User dapat memasukkan jumlah titik (n) dan dimensi dari titik (R) serta menentukan algoritma yang akan digunakan. Program akan memberikan luaran hasil pasangan terdekat, jarak, waktu eksekusi dan visualisasi titik.</p>
---	---

G. mainWindow.py

```
window = Tk()

window.geometry("1280x720")
window.configure(bg = "#2E2E4E")
window.title("Divide & Conquer")

canvas = Canvas(
    window,
    bg = "#2E2E4E",
    height = 720,
    width = 1280,
    bd = 0,
    highlightthickness = 0,
    relief = "flat"
)

canvas.place(x = 0, y = 0)
image_image_1 = PhotoImage(
    file=relative_to_assets("image_1.png"))
image_1 = canvas.create_image(
    640.0,
    360.0,
    image=image_image_1
)

entry_image_1 = PhotoImage(
    file=relative_to_assets("entry_1.png"))
entry_image_1 = canvas.create_image(
    211.0,
    300.0,
    image=entry_image_1
)

entry_1 = Entry(
    window,
    bd=0,
    bg="#E8E8E8",
    fg="#FFFFFF",
    font=("Arial Black", 20),
    selectbackground="#29293F",
    justify="center",
    insertbackground="#FFFFFF",
    highlightthickness=0,
    validate="key",
    validatecommand=(window.register(validate_int), '%S')
)
```

```
entry_1.place(
    x=50.0,
    y=281.0,
    width=118.0,
    height=50.0
)

entry_image_2 = PhotoImage(
    file=relative_to_assets("entry_2.png"))
entry_image_2 = canvas.create_image(
    211.0,
    480.0,
    image=entry_image_2
)

entry_2 = Entry(
    window,
    bd=0,
    bg="#E8E8E8",
    fg="#FFFFFF",
    font=("Arial Black", 20),
    selectbackground="#29293F",
    justify="center",
    insertbackground="#FFFFFF",
    highlightthickness=0,
    validate="key",
    validatecommand=(window.register(validate_int), '%S')
)

entry_2.place(
    x=50.0,
    y=454.0,
    width=118.0,
    height=50.0
)

button_image_1 = PhotoImage(
    file=relative_to_assets("button_1.png"))
button_1 = Button(
    image=button_image_1,
    borderwidth=0,
    highlightthickness=0,
    command=calculate_dec,
    relief="flat"
)
```

```
button_1.place(
    x=24.0,
    y=611.0,
    width=102.0,
    height=73.0
)

button_image_2 = PhotoImage(
    file=relative_to_assets("button_2.png"))
button_2 = Button(
    image=button_image_2,
    borderwidth=0,
    highlightthickness=0,
    command=calculate_sf,
    fg="#29293F",
    relief="flat"
)

button_2.place(
    x=215.0,
    y=611.0,
    width=102.0,
    height=73.0
)

button_image_3 = PhotoImage(
    file=relative_to_assets("button_3.png"))
button_3 = Button(
    image=button_image_3,
    borderwidth=0,
    highlightthickness=0,
    command=generate_array,
    relief="flat"
)

button_3.place(
    x=30.0,
    y=533.0,
    width=100.0,
    height=60.0
)
```

```
button_image_4 = PhotoImage(
    file=relative_to_assets("button_4.png"))
button_4 = Button(
    image=button_image_4,
    borderwidth=0,
    highlightthickness=0,
    command=open_github,
    relief="flat"
)

button_4.place(
    x=338.0,
    y=72.0,
    width=79.0,
    height=84.0
)

window.resizable(False, False)
window.mainloop()
```

File mainWindow.py digunakan untuk menjalankan algoritma *divide and conquer* melalui GUI menggunakan library tkinter.

<pre>def generate_array(): clear_label(window) input_n = entry_1.get() input_R = entry_2.get() if(input_n=="0" or input_R=="0" or input_n==" " or input_R==" "): show_message("Input cannot be blank\nor zero","#FFC9C1") else: show_message("Point generated\nsucessfully!","#DCFFC1") n = int(input_n) R = int(input_R) global Array Array = generateCoordinate(n, R)</pre>	<p>Prosedur generate_array() digunakan untuk mengisi variable global 'Array' dengan array dari koordinat.</p>
<pre>def calculate_BF(): clear_label(window) if(Array == -999): show_message("Point is not generated\n,yet!","#FFC9C1") else: start = time() * 1000 first, second, eucledianCount = closestPair(Array) distance = calculateDistance(Array[first], Array[second], len(Array[0])) finish = time() * 1000 time_taken = finish - start show_results(Array[first], Array[second],distance,"Brute Force",time_taken,eucledianCount) if (len(Array[0]) == 3): show_plot(Array, Array[first], Array[second])</pre>	<p>Prosedur calculate_BF() digunakan saat button BruteForce ditekan dan memberikan output yang diminta menggunakan algoritma <i>brute force</i>.</p>
<pre>def calculate_DnC(): clear_label(window) if(Array == -999): show_message("Point is not generated\n,yet!","#FFC9C1") else: start = time() * 1000 first, second, distance, eucledianCount = solveDivideAndConquer(Array, len(Array), len(Array[0])) finish = time() * 1000 time_taken = finish - start show_results(Array[first], Array[second],distance,"Divide\n& Conquer",time_taken,eucledianCount) if (len(Array[0]) == 3): show_plot(Array, Array[first], Array[second])</pre>	<p>Prosedur calculate_DnC() digunakan saat button Divide&Conquer ditekan dan memberikan output yang diminta menggunakan algoritma <i>divide and conquer</i>.</p>
<pre>def validate_int(val): if val.isdigit(): return True else: return False</pre>	<p>Fungsi validate_int() digunakan untuk memvalidasi input user pada GUI agar hanya dapat memasukkan integer.</p>

```
def show_result(point1, point2, dist, algorithm, time, execution):
    point_1 = pointToStr(point1)
    point_2 = pointToStr(point2)
    result_p1 = Label(window,
        text = point_1,
        font=("Arial Black",10),
        justify="left",
        bg="#E12E48",
        fg="WDCFFC1")

    result_p1.place(
        x=450,0,
        y=414,0,
    )

    result_p2 = Label(window,
        text = point_2,
        font=("Arial Black",10),
        justify="left",
        bg="#E12E48",
        fg="WDCFFC1")

    result_p2.place(
        x=450,0,
        y=436,0,
    )

    result_dist = Label(window,
        text = str(round(dist,2)),
        font=("Arial Black",10),
        justify="center",
        bg="#E12E48",
        fg="WDCFFC1")

    result_dist.place(
        x=500,0,
        y=485,0,
    )

    result_algorithm = Label(window,
        text = algorithm,
        font=("Arial Black",10),
        justify="center",
        bg="#E12E48",
        fg="WDCFFC1")

    result_algorithm.place(
        x=400,0,
        y=565,0,
    )
```

```
executed = Label(window,
    text = str(round(time,2)) + " ms",
    font=("Arial Black",10),
    anchor="w",
    bg="#E12E48",
    fg="WDCFFC1")

executed.place(
    x=400,0,
    y=640,0,
)

executionCount = Label(window,
    text = execution,
    font=("Arial Black",10),
    justify="center",
    bg="#E12E48",
    fg="WDCFFC1")

executionCount.place(
    x=510,0,
    y=530,0,
)

def show_plot(arrayCoordinate, c1, c2):
    # c1 dan c2 adalah 2 koordinat yang akan digaris
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Masukkan semua koordinat ke diagram
    for i in range(len(arrayCoordinate)):
        x, y, z = arrayCoordinate[i]
        ax.scatter(x, y, z, c='g', s=1)

    # Buat garis diantara c1 dan c2
    x1, y1, z1 = c1
    x2, y2, z2 = c2
    ax.scatter(x1, y1, z1, c='r', s=10)
    ax.scatter(x2, y2, z2, c='r', s=10)
    ax.plot([x1, x2], [y1, y2], [z1, z2], c='r')

    ax.set_xlabel("Sumbu X")
    ax.set_ylabel("Sumbu Y")
    ax.set_zlabel("Sumbu Z")

    canvas.figure = figureCanvasTkAgg(fig, master=canvas)
    canvas.figure.draw()
    canvas.figure.get_tk_widget().place(x=0, y=0, width=600, height=600)

    canvas.graph = canvas.figure
```

```
def show_message(message,color):
    messageShow = Label(window,
        text = message,
        font=("Arial Black",10),
        anchor="w",
        justify="left",
        bg="#939393",
        fg=color)

    messageShow.place(
        x=450,0,
        y=250,0,
    )

def clear_label(window):
    for child in window.winfo_children():
        if isinstance(child, Label):
            child.destroy()
```

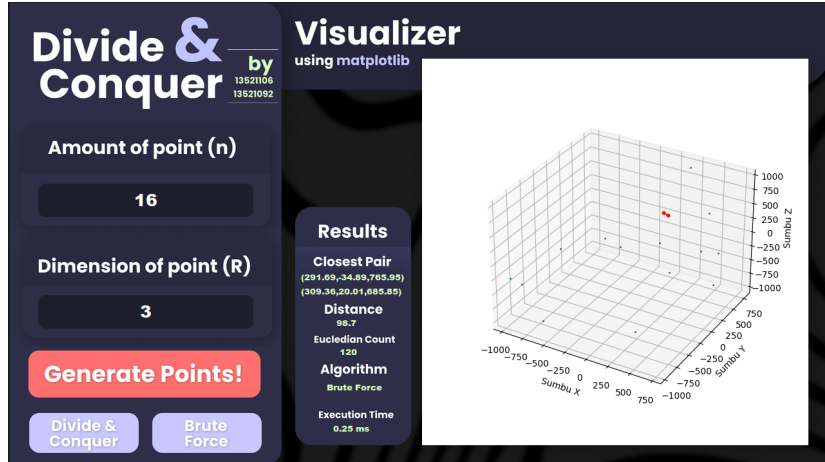
Prosedur
show_result(),
show_plot(),
show_message()
masing-masing digunakan
untuk menampilkan data
hasil (execute time, hasil
perhitungan algoritma),
menampilkan grafik, dan
menampilkan pesan custom
pada GUI.
Prosedur clear_label()
digunakan untuk
menghapus semua output
temporary pada GUI.

BAB V

EKSPERIMEN

A. Test Case

Brute Force:



Test Case 1

Jumlah titik (n) : 16

Dimensi titik (R) : 3

Waktu eksekusi :

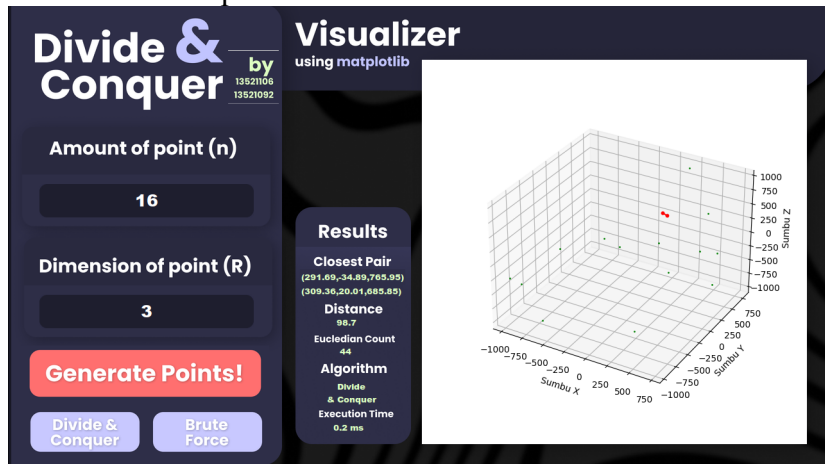
- DnC : 0.2 ms
- BF : 0.25 ms

Jumlah operasi Euclidean:

- DnC : 44
- BF : 120

Apakah hasil pasangan sama?

Divide and Conquer:



Brute Force:

Test Case 2

Jumlah titik (n) : 64

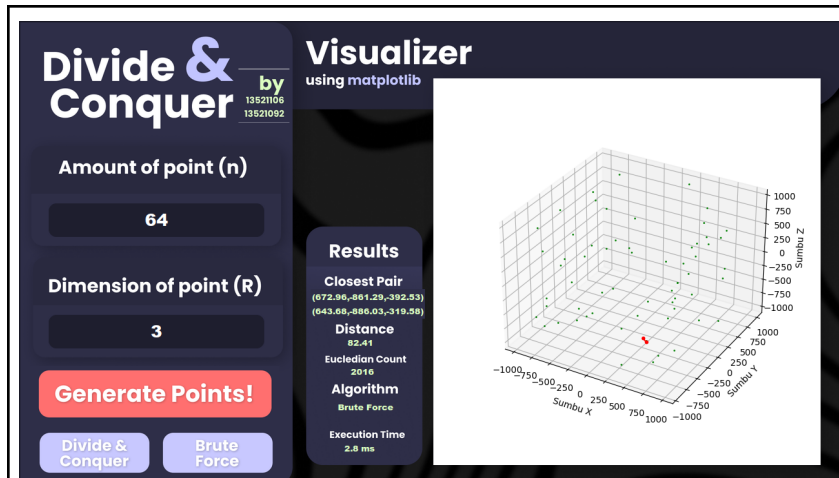
Dimensi titik (R) : 3

Waktu eksekusi :

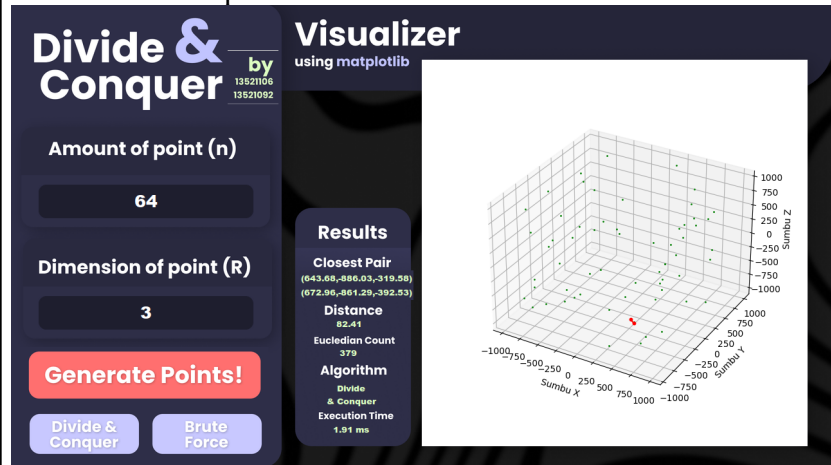
- DnC : 1.91 ms
- BF : 2.8 ms

Jumlah operasi Euclidean:

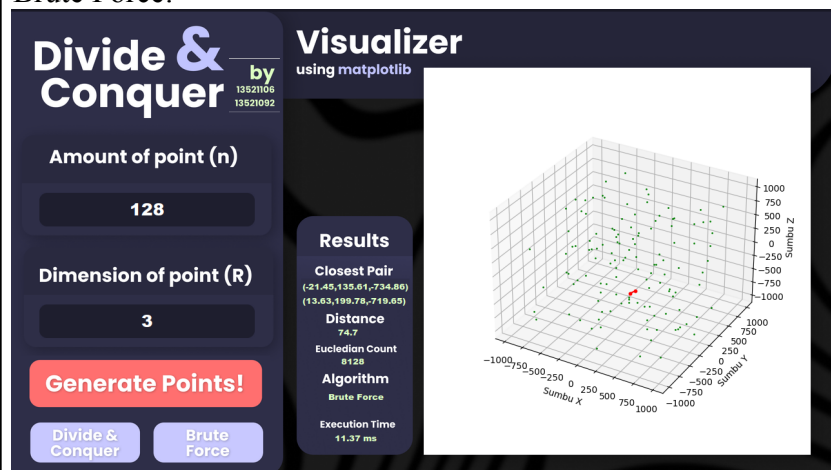
- DnC : 379
- BF : 2016



Divide and Conquer:



Brute Force:



Divide and Conquer:

Test Case 3

Jumlah titik (n) : 128

Dimensi titik (R) : 3

Waktu eksekusi :

- DnC : 4.47 ms
- BF : 11.37 ms

Jumlah operasi Euclidean:

- DnC : 1083
- BF : 8128

B. Analisis

Dari percobaan yang dilakukan, dapat dilihat bahwa untuk titik dengan jumlah dibawah sama dengan 128 titik, algoritma *Divide and Conquer* masih dapat memberikan solusi *closest pair* jika dibandingkan dengan algoritma *Brute Force*. Namun, untuk titik yang berada di atas 128, misalnya 1000, algoritma *Divide and Conquer* mulai memberikan solusi *closest pair* yang berbeda jika dibandingkan dengan algoritma *Brute Force*. Hal ini disebabkan oleh semakin padatnya volume ruangan yang diisi titik yang digenerate secara random. Pada program ini, penulis memasang range angka random dari -1000 sampai 1000. Oleh karena itu, galat perhitungan akan semakin tinggi jika menggunakan algoritma *Divide and Conquer*.

Di lain sisi, jumlah operasi Euclidean pada algoritma *Divide and Conquer* jauh lebih sedikit dibandingkan dengan jumlah operasi Euclidean pada algoritma *Brute Force*. Hal ini disebabkan karena algoritma *Divide and Conquer* memiliki teknik pencarian yang tidak mengharuskan algoritma untuk mengecek seluruh kemungkinan pasangan. Hal tersebut juga membuat waktu eksekusi menjadi lebih cepat dengan perbandingan waktu eksekusi algoritma *Divide and Conquer* dan *Brute Force* sekitar 1:2-6 dengan ukuran titik 100-1000 dan berada pada ruang vektor R^3 .

BAB VI

KESIMPULAN

Pada tugas ini telah dibuat sebuah program penyelesaian persoalan mencari pasangan titik terdekat dengan bahasa pemrograman Python. Dalam pengerjaannya digunakan Algoritma *Divide and Conquer* untuk mencari jawaban dari persoalan tersebut dan membandingkan hasilnya dengan hasil dari Algoritma *Brute Force*. Dari percobaan yang dilakukan, dapat dilihat bahwa untuk titik dengan jumlah dibawah sama dengan 128 titik, Algoritma *Divide and Conquer* masih dapat memberikan solusi *closest pair* jika dibandingkan dengan Algoritma *Brute Force*. Namun, untuk titik yang berada di atas 128, misalnya 1000, Algoritma *Divide and Conquer* mulai memberikan solusi *closest pair* yang berbeda jika dibandingkan dengan Algoritma *Brute Force*.

Selain itu, dalam pengerjaan Tugas Kecil 2 ini, penulis mendapatkan banyak ilmu yang baru dalam pengerjaannya, seperti mendapatkan *insight* mengenai aplikasi dari algoritma *divide and conquer*, pemanfaatan berbagai *library* Python serta pengalaman membangun GUI desktop menggunakan *library* tkinter Python. Harapan penulis untuk tugas kedepannya agar diberikan waktu pengerjaan yang lebih lapang agar dapat memberikan hasil yang lebih maksimal.

Daftar Referensi

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf) [diakses pada 28/02/2023]

Lampiran

Link Repository Github: https://github.com/farhanfahreezy/Tucil2_13521092_13521106

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